ENPM662

INTRODUCTION TO ROBOT MODELING

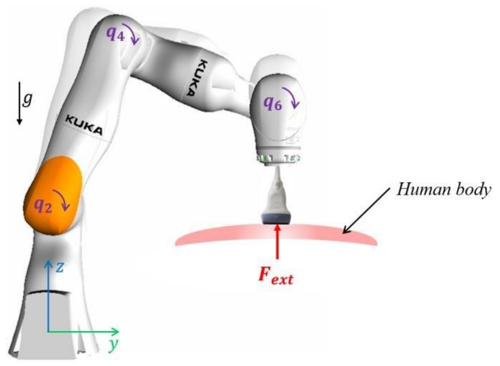
Project 2 Final Report

Modelling and Simulation of a Cyberknife System

&

Hardware Implementation of Turtlebot3

UNIVERSITY OF MARYLAND



Team Members

JOSEPH PRANADEER REDDY KATAKAM | 117517958

BHARGAV KUMAR SOOTHRAM | 117041088

BHARADWAJ CHUKKALA | 118341705

Contents

Abstract

Treating tumours is often a very tough task that involves very Precise radiotherapy. Even things like breathing, filling of the bladder or gas in the bowel make the tumour move around and this makes for overdosing or underdosing. Overdosing results in side effects and underdosing results in ineffective treatment. Cyberknife, which was invented by Dr John Adler, a neurosurgeon at Stanford, USA, came into practice in the 1990s. A lightweight linear accelerator fitted onto an industrial 6 DOF robot makes treatment possible precisely in the desired way. So, we set out to implement similarly Precise robotic movement with motion synchronization akin to Cyberknife (Trademarked) using a KUKA robot. The Goal of this project here is to model/make and implement Precise movements that track the target's movements, this involves modelling forward/reverse kinematics that we learnt through the course for the robot.

1. Introduction:

Robotics is believed to have the same impact on healthcare in the next few decades as it had manufacturing in the past 30 years. The field of robot-integrated surgery is rapidly developing, providing innovative and minimally invasive solutions to heal complex injuries and diseases. Well over 200,000 successful operations have been accomplished so far primarily in neurosurgery, orthopaedics, ENT surgery, pediatrics, and interventional radiology. In the near future, newly developed robotic systems may conquer even the most challenging fields to support better patient care and medical outcome. Surgical robots inherited many features and concepts from industrial manipulators, while greatly evolving them.

Industrial robotics have already conquered the field of assembly, automobile industry, appliance, undersea exploration, space missions and hazardous material handling. In the past two decades, a radically new field, Robot Integrated Surgery (RIS) has also been gaining significance. Most commonly, mechatronic devices have been used for prostatectomy, neurosurgery, nephrectomy, cholecystectomy, orthopedics, and radiosurgery. Robot-aided procedures offer remarkable advantages both for the patient and the surgeon, making microsurgery and Minimally Invasive Surgery (MIS) a reality, and even complete teleoperation accessible. RIS can increase the stability and robustness of the system, navigate accurately based on medical images and help positioning the surgical tool to the target point (image guided surgery). Furthermore, there is the option to introduce advanced digital signal processing and control or to record the spatial points-of-interest and motions. This can be useful for surgical simulation and risk-free training. Finally, robotized equipment can greatly add to the ergonomics of the procedure, especially in the case of laparoscopic MIS. Due to the extensive research and application background behind industrial robotics, initial surgical robot setups heavily relied on those systems, and even today, some of the most successful products effectively integrate industrial solutions. Cyberknife manipulator being one such exemplary of the greatest technological advancements in integration of industrial robots into healthcare.

The Cyberknife is a precision and accuracy-based radiosurgery technique which drew motivation from the concept of avoiding radiation accidents due to human error and is devised to provide the best possible treatment to patients. It is a stereotactic radiosurgery (SRS) system which combines the principles of stereotaxy or three-dimensional target localization, and radiation beams from multiple directions cross firing the tumour precisely. Due to the high degree of precision, it is possible to deliver a very high dose of radiation to the target with minimal damage to the normal tissues and structures surrounding the tumour. The ideal aim is to ablate the tumour with a high radiation dose noninvasively. It has been proved to be an effective alternative to surgery for small tumours and selected medical conditions.

The core of the Cyberknife in reality is a robot manufactured by KUKA, the Cyberknife was conceptualized from an industrial manipulator named Kr Quantec, its primary purpose was to serve the industry and had the functionality which was well suitable for the requirements to create the Cyberknife. KUKA has innovated many industrial manipulators and one of them is a soft robot and a cobot named KUKA LBR iiwa which has time and again proven to be one of the most researched robots in terms of human robot interaction. So. instead of using Kr Quantec, the model we made use of the LBR iiwa model to perform the operations of a Cyberknife.

Radiosurgery is surgery using radiation, that is, the destruction of precisely selected areas of tissue using ionizing radiation rather than excision with a blade. Like other forms of radiation therapy (also called radiotherapy), it is usually used to treat cancer. Radiosurgery was originally defined by the Swedish neurosurgeon Lars Leksell as "a single high dose fraction of radiation, stereotactically directed to an intracranial region of interest". In stereotactic radiosurgery (SRS), the word "stereotactic" refers to a three-dimensional coordinate system that enables accurate correlation of a virtual target seen in the patient's diagnostic images with the actual target position in the patient. Stereotactic radiosurgery may also be called stereotactic body radiation therapy (SBRT) or stereotactic ablative radiotherapy (SABR) when used outside the central nervous system (CNS).

Coing to why we chose the cobot KUKA LBR iiwa R800, the lbr iiwa 7 r800 eliminates the need for barriers and makes it a reality for humans and robots to work alongside each other. It gives operators a third arm by being able to assist with many processes. It is built entirely from aluminium to keep its robot mass down to just 24 kg. Due to its lightweight it is extremely agile, following contours quickly under force control. Its kinematic system is designed to replicate the human arm. This design proves to be extremely flexible when it comes to achieving different positions and can operate in the tightest of spaces. This makes it ideal for those with limited space. And, it is also being used to perform the toughest Precision tasks, hence it is an ideal alternative to make a Cyberknife considering the factors like budget and treatment cost.

There are many other factors that need to be taken into consideration while performing the radiosurgery. The real time motion synchronization is very important and why is that? Why does motion management matter? The patient breathes, the patient could change their position or move their head, even slightly. The patient could cough. Muscles tense and relax. The Cyberknife System is the only device designed to accommodate all forms of patient and tumour motion, even while the treatment is being delivered. With its motion adaptive delivery technology, the Cyberknife System enables smaller treatment margins around the tumour, minimizing the amount of healthy tissue exposed to high-dose radiation.

2. Motivation and Application:

Since the advent of radiation, accidents involving radiation damage have been rare and have had a very low reproducibility rate. The causes and severity of these accidents vary greatly. In some cases, the radiation source is identified, and the exposed individuals do not experience adverse effects, while in other cases the accident goes unnoticed until injuries appear. In medical practice, radiotherapy can lead to "accidental over irradiation syndrome," which is the result of very high doses of irradiation to the organs. In this situation, symptoms do not instantly appear even in the case of much higher-than-normal doses of irradiation.

In December 1990, a malfunction of the electron linear accelerator in the Clinic of Zaragoza led to irradiation of 36 MeV, which was the highest beam energy. In this accident, 27 patients, who were supposed to receive much lower energies, received doses up to seven times higher in their targeted regions. These treatments targeted the cervical region, chest wall, inguinal region, trapezius muscle area, dorsal region, frontal lobe, and mammary nodes. Due to the high dose, many of the patient experienced negative symptoms, with the earliest symptoms becoming present six days after their final radiotherapy session.

The first visual symptom presented by the over irradiated patients was radio dermatitis, which occurred in all but one patient. Within the first month after the beginning of treatment, other symptoms became apparent, such as dysphagia and changes to intestinal transit. However, as time continued to pass, it became apparent that patients had damage to many organs, including the oesophagus, oropharynx, lungs, cervical cord, liver, colon, stomach, and skin. Although treatment was used in an attempt to cure these patients, 20 of the 27 died within three years, three of which dying from causes unrelated to the over irradiation.

Fig 1: Clinic of Zaragoza radiotherapy accident

Now if we take a look at this scenario from a closer perspective, we can notice that it is evident, the incident could have been prevented by proper calibration and maintenance of the equipment. This is the case with the reported accidents with the most severe consequences because a mistake in calibration may not be promptly discovered. Historically, these types of mistakes have been due to insufficient training of the medical staff, lack of quality assurance, and misunderstanding the implications of imprecise adjustments of the equipment. Therefore, it is important that written procedures and protocols are put in place for new radiotherapy equipment, as well as a quality assurance program. Although the accidents of the past are unfortunate, we can learn many lessons from them. Moving forward, it is critical that we attend to the deficits within radiotherapy in order to improve patient safety and confidence.

To improve the possibility of safety and reducing the damage to the patient, technology at our disposal poses many solutions which significantly help in the achievement of the intended. Fatal radiotherapy accidents posing to human lives can be prevented significantly with the right technology. Hence, drawing motivation from this incident we decided to go with the Cyberknife cobot for our project and chose KUKA LBR iiwa R800 as the core of the system. This robot has the primary application in the field of healthcare, like it has been iterated many times across the report, radiosurgery will be the most important application.

3. Robot and Workspace Description:

It is a frameless, image-guided robotic technology used to deliver stereotactic radiosurgery and radiotherapy anywhere in the body where it is clinically indicated. The robotics aspect of this system plays as much as a key role as the surgical aspect. The target pose is localized into a common reference frame using a combination of image registration algorithms and precisely calibrated coordinate transformations. A robotic couch, on which the patient is positioned, and a robotic treatment manipulator on which a medical linear accelerator is mounted, are aligned using the method of localization and mapping.

3.1. Robot Type:

The Robot system we propose to work on consists of a 6DOF Robot manipulator with a linac (linear accelerator) module mounted as the end effector, the system also consists of a robotic treatment couch which is based on a SCARA manipulator. The treatment manipulator for this system is the KUKA LBR iiwa R800 robot. This is a 6DOF robot that has been first developed for industrial purposes but in 1999 John Adler at Stanford university coupled it with a 4MeV Linear accelerator to create what was called the Cyberknife system.

There are three primary coordinate frames when describing the robot workspace within the treatment room. The robot world frame has its origin at the base of the robot, centered on the first axis of rotation where the robot is mounted on top of the pedestal. The robot tool frame is defined by a laser mounted inside the linac, such that the laser is coincident with the radiation beam, and the origin is the treatment X-ray beam source (center of the linac target). The robot user frame is defined with its origin at the machine center, with rotations aligned to the robot world frame. Again the robot itself has 6 coordinate frames for each of its joints.

There is a patient couch (RoboCouch) in the workspace, it is also a serial link manipulator which brings additional functionality to the Cyberknife System. RoboCouch is a custom-designed robot, based on the selective compliance articulated robotic arm (SCARA) configuration, which utilizes the same KUKA controller and KUKA wrist as the treatment robot. The first axis is a vertical axis to provide Z motion, with the two subsequent axes providing planar X/Y motion, followed by a three-axis intersecting wrist to enable rotation positioning

General Specifications	
Weight of robot	23.9kg
Degrees of freedom of Cyberknife	6
Degrees of freedom of the RoboCouch	6
Rated Payload – Maximum Payload (Cyberknife)	7kg
Pose repeatability (ISO 9283)	± 0.1mm
Maximum Reach	800mm
Number of axes	6
Mounting position	On an elevated surface
Structure	Articulated
Version Environment	Standard or a Surgical Setup
End effector module	Linac (Linear Particle accelerator)
Custom pedestal (d = offset)	412mm
Detectors and Imaging technology	X-ray and optical imaging systems, CT scan
Applications	Stereotactic Radiosurgery, Hair transplantation
Working Ambient temperatures	0 °C to 55 °C
Controllers	KR C5, KR C4
Protection rating	IP 65, IP 67

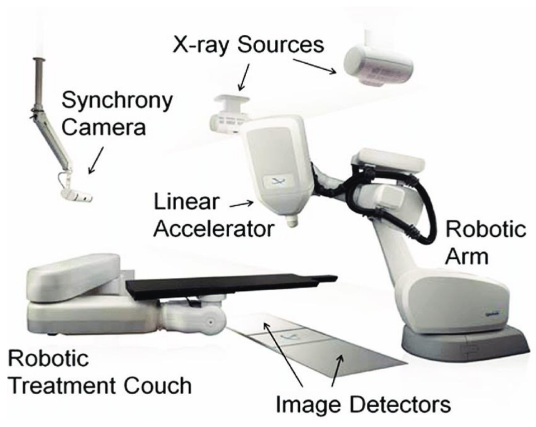


Fig 2: Obstacles in the workspace where Cyberknife is used.

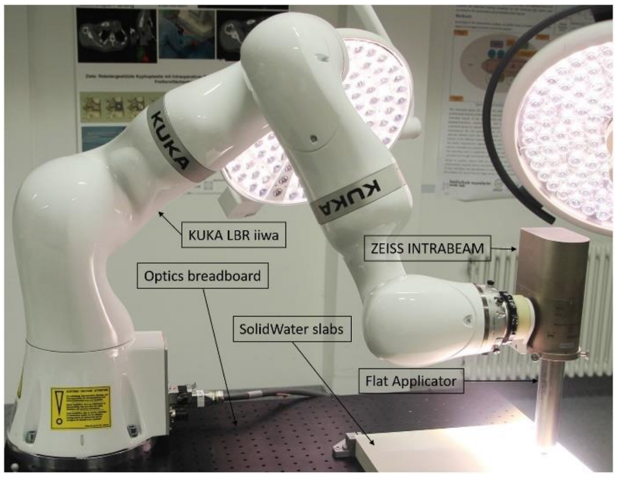


Fig 3: The KUKA LBR iiwa R800 cobot with Linac end effector, surface irradiation testing on metals before medical integration as a Cyberknife.

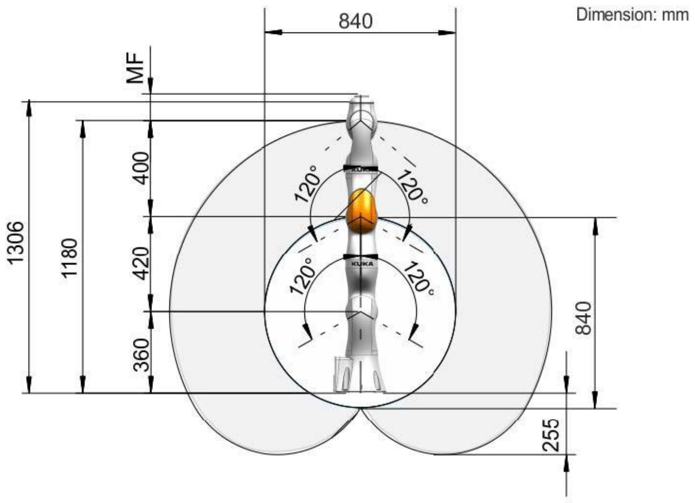


Fig 4: Cyberknife workspace graphic

3.2. DH Parameters

A commonly used convention for selecting frames of reference in robotic applications is the Denavit Hartenberg (DH) convention.

The manipulator arm can be thought of as a chain of rigid bodies with a coordinate axis or frame attached to each one. The DH notation defines four parameters that specify the relative location of frame i with respect to the previous one.

- $\boldsymbol{a_i}$: The length of the common normal, called the **link length**; that is, the distance from the intersection of the z_{i-1} axis with the x_i axis to the origin of the ith frame along the x_i axis
- $\boldsymbol{d_i}$: The joint variable for a prismatic joint; the distance between the origin O_{i-1} and the point H_i, the intersection of the z_{i-1} axis with the x_i axis along the z_{i-1} axis. This is called the **link offset**.
- $\boldsymbol{\alpha_i}$: The angle between the joint axis i and the zi axis about the xi axis in the right-hand sense, called the **twist of the link**.
- $\boldsymbol{q_i}$: The joint variable for a revolute joint; the angle from the x_{i-1} axis to the x_i axis about the z_{i-1} axis, using the right-hand rule. This is called **joint angle**.

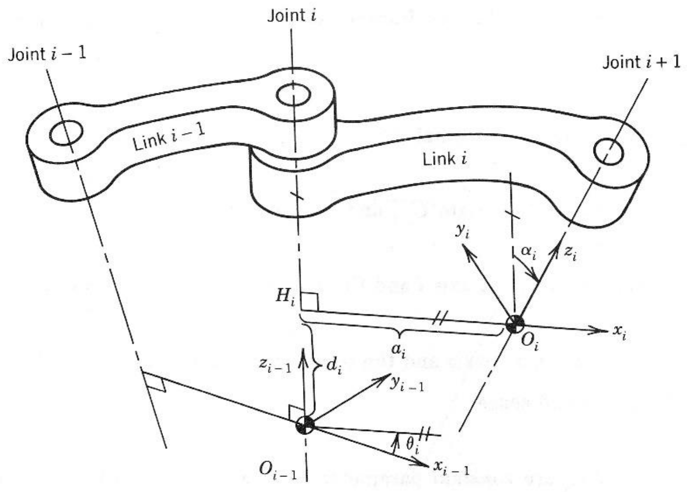


Fig 3: The Denavit Hartenberg parameters

The DH parameters for the KUKA LBR iiwa R800 cobot that we have chosen for our project is given in the table below.

Table 1: DH Parameters

Link	Twist(α)	Length(a)	Angle(θ)	Offset(d)
1	-90	0	$\theta 1$	360
2	90	0	$\theta 2$	0
3	90	0	$\theta 3$	420
4	-90	0	$\theta 4$	0
5	-90	0	$\theta 5$	399.5
6	90	0	$\theta 6$	0
7	0	0	$\theta 7$	205.5

3.3. DOFs, Coordinate Frames and joint specifications

The Degrees of freedom of the chosen robot is 6. The robot manipulator is made up of 7 links and 6 joints. Each joint has one degree of freedom. All the 6 joints are revolute and have a constrained motion to avoid singularities and self-collision. The robot is constrained based on the joint speeds and joint motion. Below are the specifications of the constraints applied to KUKA LBR iiwa R800 collaborative robot.

Table 2: Robot motion speed

Joint 1	1.71 rad/s (98˚/s)
Joint 2	1.71 rad/s (98˚/s)
Joint 3	1.75 rad/s (100˚/s)
Joint 4	2.27 rad/s (30˚/s)
Joint 5	2.44 rad/s (140˚/s)
Joint 6	3.14 rad/s (180˚/s)

Table 3: Robot motion range

Joint 1	±170˚
Joint 2	±120˚
Joint 3	±170˚
Joint 4	±120˚
Joint 5	±170˚
Joint 6	±120˚

The Coordinate frames play a key role in determining the DH parameters of a robot. Denavit-Hartenberg (D-H) frames help us to derive the equations that enable us to control a robotic arm.
The D-H frames of a particular robotic arm can be classified as follows:

- Global coordinate frame: This coordinate frame can have many names like world frame, base frame, etc.
- Joint frames: We need a coordinate frame for each joint.
- End-effector frame: We need a coordinate frame for the end effector of the robot (i.e., the gripper, hand, etc. that piece of the robot that has a direct effect on the world).

To draw the frames (i.e. x, y, and z axes), we follow four rules that will enable us to take a shortcut when deriving the mathematics for the robot. These rules collectively are known as the Denavit-Hartenberg Convention. Here are the four rules that guide the drawing of the D-H coordinate frames:

1. The z-axis is the axis of rotation for a revolute joint.
2. The x-axis must be perpendicular to both the current z-axis and the previous z-axis.
3. The y-axis is determined from the x-axis and z-axis by using the right-hand coordinate system.
4. The x-axis must intersect the previous z-axis (rule does not apply to frame 0).

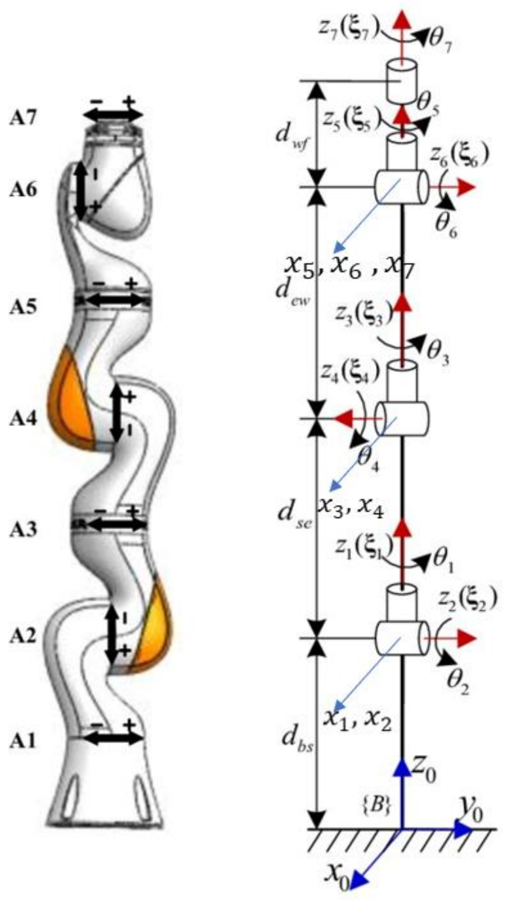


Fig 4: The DH Coordinate Frame assignment for the KUKA LBR iiwa R800 robot

4. Robot Appropriateness:

The Cyberknife system is the state of the art currently. The model we are determined to make is a 6DOF manipulator which has a primary functionality in the field of radio therapy and surgery. There is a high voltage Linear particle accelerator as it’s end effector which is necessary and relevant towards the non-invasiveness of the surgical functionality.

The manipulator appropriateness comes into the bigger picture when the radio beam, created by the collimator in the Linac module, has to precisely and accurately target the tumour or any other cancerous cell build-up present in the patient's organs. The Precision and accuracy of the surgery can be guaranteed only by the robot or a skilled radiologist. The Manipulator with its 6 degrees of freedom can traverse easily in its widespread workspace without creating a singularity by running into itself.

The robot will have a base put on the pedestal which offsets the world coordinate system by a length. The pedestal is sheathed with a protective industrial cover which also has enough volume to consist of some electronic components required by the linac module. There are other subsystems to the robot manipulator like the cabling, treatment head, tensioning mechanism. The linac end effector is a part of the spherical wrist of the robot manipulator giving it access to 3 degrees of freedom just for the Precise collimation of the radio beam. The available workspace that the robot can reach for clinical use is discretized into specific points, referred to as nodes, which are grouped into larger sets, referred to as paths. The robot can be oriented such that the treatment beam source (center of the linac X-ray target) is coincident with each node, and therefore nodes represent the source positions of treatment beams.

The robot is not only accurate in collimation of the beam but also in the collision avoidance and proximity detection. Although some paths are default designs to avoid collisions with other components of the system, as well as a large bound on the patient volume, the system continuously monitors the position of each robot (treatment robot and patient couch) and calculates distances between these moving objects versus other moving components and static obstacles in the room.

We intend to mimic the movements required to implement a Precise task such as radiotherapy using a KUKA robot. Given the set of positions of the target, the robot should be able to make changes so as to adjust its position to match the movements.

5. Scope Description and Future study:

- *Fallback:*
    - Designing a Robo Couch Scara Robot, a simple robot which is part of the Cyberknife System used to carry the patient.
    - Creating an arm manipulator with 6-DOF.
    - Performing a simple simulation of moving the end effector around in the workspace.

- *Ambitious goals:*
    - Building a medical surgery room world in a gazebo to exactly demonstrate the process.
    - Optimizing the robot to move along the desired trajectory with an enhanced speed.
    - Simulation of the tumor cells getting destroyed by using the KUKA Quantec built.
    - Automating the simulation process.

- *Learning outcomes*:
    - To understand and calculate Forward and Inverse kinematics for any real-world robot.
    - To learn building CAD models for any given real-time robot.
    - To learn simulated environments and model the real world in ROS - Gazebo & Rviz.
    - To understand ROS and its functionalities like Teleop, Publisher and Subscriber
    - To do and learn trajectory planning for the manipulator arm with Precision.
    - To validate and verify calculated Forward and Inverse kinematics solutions using a developed simulated environment.
    - To understand the integration of sensors, controllers in a simulated environment.
    - To understand the difference between working in a simulated environment and real-world robot.
    - To overcome the issues when working with the hardware implementation of robots.

- *Scope Appropriateness:*

    The Cyberknife System is an image-guided robotic technology used to deliver stereotactic radiosurgery and radiotherapy anywhere in the body where it is clinically indicated. The KUKA Quantec robot manipulator used in this system is used to treat the tumor cells present in the body. A primary reason for selecting this robot was the higher payload required for the addition of the multileaf collimator (MLC).

Working on such a robot would give us the necessary knowledge on how everyday medical robots are built and used. Also, the kinematic equations, simulations and working on hardware implementation on such a robot would make us industry prepared. This project will lay a solid foundation of essentials required to work on a robot in the real world by learning to model, integrate sensors & controllers, performing gazebo simulations and finally implementing it in real-time.

6. Model Assumptions:

The model we intend to make will have 6 DOF, each joint has one degree of freedom. There will be one base with a revolute joint with z axis pointing upwards, two main links which will also be revolute in nature. There is a spherical wrist with 3 DOF and to it is attached an end effector (our linac module).

In the model the cabling of the linac subsystem will be eliminated to avoid unnecessary part additions and for ease of modelling. All the parts will be considered as rigid bodies and frictionless. The environmental conditions in a hospital are highly uncertain so modelling it is a tricky task, which we choose to avoid, considering an ideal scenario with no extra external entities in the surroundings. The Linac sensor functionality cannot be modelled or simulated so we will assume a sensor that seems apt for the purpose while we integrate the robot model.

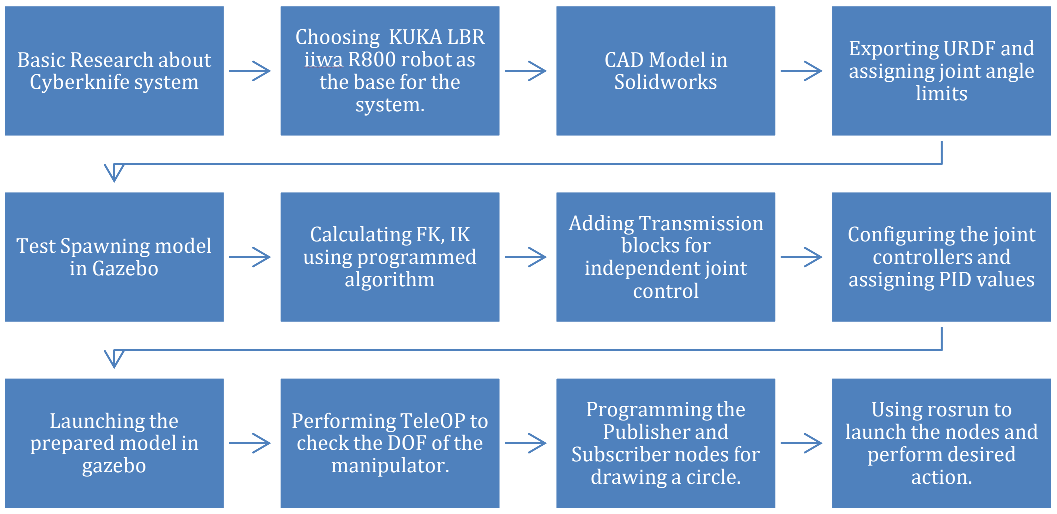
The singularities in the workspace of the robot created by self-collision cannot be integrated so we will try and limit the angular motion of each joint while we model the robot in solidworks, so that the singularities are limited. There are many custom configurations that are possible for the robot so some link offsets may vary based on the test case scenario. In some cases, some joint angles will be locked, and length of the links will be assume to be zero, the link twist will also be negated to zero for brevity and ease of calculation of DH parameters and solutions.

7. Approach to performing the work:

As we are working on a robot which has practical applications in everyday life, we took time to understand how the entire system works and how this project can lay the foundation required to work in the medical industry. We started by designing the KUKA LBR robot using Solidworks and then exported it as a URDF file. Then, the testing of the model in Gazebo empty world environment (the robot model and it’s coordinate frames are given a check) was done. Next, the necessary sensors, controllers, and transmission blocks are integrated onto the robot.

We calculate the Forward kinematics using the DH parameters method followed by Inverse Kinematics by which we will understand the robot workspace and constraints. Using this knowledge, we simulated the robot in Gazebo and wrote a publisher and subscriber node to make the robot perform intended action. Our ambitious goal is to design a surgery world in gazebo and program the robot to perform a specific task on a dummy human. The manually calculated kinematic solutions will be validated by passing values in the gazebo environment followed by Jacobian analysis of the robot. Even though the simulation in gazebo was performed for a KUKA LBR robot, the real-time application will differ in accordance with the KUKA robot available in the Lab, where the entire knowledge of our simulation will be used in real-time.

The Pipeline we have followed to go about the Project:


Basic Research about Cyberknife system
Choosing KUKA LBR iiwa R800 robot as the base for the system.
CAD Model in Solidworks
Exporting URDF and assigning joint angle limits
Test Spawning model in Gazebo
Calculating FK, IK using programmed algorithm
Adding Transmission blocks for independent joint control
Configuring the joint controllers and assigning PID values
Launching the prepared model in gazebo
Performing TeleOP to check the DOF of the manipulator.
Programming the Publisher and Subscriber nodes for drawing a circle.
Using rosrun to launch the nodes and perform desired action.

8. Forward and Inverse Kinematics:

In order to perform tasks on different manipulator platforms, a kinematic model has to be developed for each platform. Because the equations can become too cumbersome to deal with manually when the robots have more than just several joints, a way is needed to automate the formation of the kinematic model. Hence, we have used python3 to develop the code for our algorithm (Ref. to appendix for code).

Industrial robots are usually developed for specific predetermined tasks. The manipulator arm configuration, along with equations needed for the manipulator arm motion, are determined, and solved during robot development. This limits the robot to the prescribed tasks and to no others. However, for robots that must adapt to their environment or perform a wide range of tasks, a way is needed for the manipulator arm to adapt to changes. Changes to the equations (Jacobian and forward kinematic expressions), are required when something changes the geometry of the manipulator arm, such as when a tool is added to the end-effector.
The Steps and Formulas we have used to code the algorithm are as follows:

- ❖ Firstly, the individual transformation matrices are calculated using the parameters given in the DH Table:
- ❖ The Formula to calculate individual transformation matrices is derived as follows:

$$\begin{aligned} A_i &= Rot_{z,\theta_i}\mathrm{Trans}_{z,d_i}\mathrm{Trans}_{x,a_i}Rot_{x,\alpha_i} \\ &= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &\quad\times \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

- ❖ The Transformation matrix from joint i-1 to i is:

$$A_i^{i-1}(q_i) = \begin{bmatrix} \cos\theta_i & -sin\theta_i cos\alpha_i & sin\theta_i cos\alpha_i & a_i cos\theta_i \\ sin\theta_i & cos\theta_i cos\alpha_i & -cos\theta_i sin\alpha_i & a_i sin\theta_i \\ 0 & sin\alpha_i & cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- ❖ Now we have to find the individual transformation matrices by substituting the values of appropriate joint angle in the A matrix.
- ❖ The final transformation matrix, which contains the mathematical key to move the robot in such a way that the end effector moves to the intended pose, can be calculated by the product of all the individual joint transformation matrices.

- It is given as: $T_0^n = A_1^0(q_1) * A_2^1(q_2) * A_3^2(q_3) * \dots * A_n^{n-1}(q_n)$
- We have chosen an initial pose for the manipulator which doesn't give any singularities and will place the end effector on the intended plane of action.
- The joint angles that we have chosen for the initial pose are as follows:

$$q_1 = [0 \quad 30^\circ \quad -45^\circ \quad 0 \quad 75^\circ \quad 0]^T$$

- After choosing this joint angle matrix, these joint angles are then substituted in the formula that was derived earlier, thus giving the individual transformation matrices.
- The individual transformation matrices are then multiplied to give the final transformation matrix which gives us the position of the end effector.
- The end effector was then positioned on the XZ plane at a certain height.
- The intended action desired by the user is to make the robot draw a circle on the XZ plane.
- The end effector is positioned on the XZ plane on a point which is a point on the circumference of the circle that needs to be drawn. This point is where the robot starts its motion in the simulation.
- Now Inverse kinematics come into the picture.
- Till now end effector velocity was calculated using the joint velocities as

$$\dot{X} = J\dot{q}$$

- Now to draw the circle we can calculate the trajectory of the circumference by finding the planar velocity of the robot at each point on the circumference of the circle.
- Now to calculate the planar velocity of the end effector, we have used the cylindrical coordinates of each point on the circumference of the circle which can be denoted by

$$x = rcos\theta$$
$$z = rsin\theta$$

- The velocity vector can be now written as

$$v_1 = [\dot{x} \quad 0 \quad \dot{z} \quad 0 \quad 0 \quad 0]^T$$
$$\dot{x} = -r\dot{\theta}sin\theta$$
$$\dot{z} = r\dot{\theta}cos\theta$$
$$v_1 = [-r\dot{\theta}sin\theta \quad 0 \quad r\dot{\theta}cos\theta \quad 0 \quad 0 \quad 0]^T$$

- After we get the velocity vector, we use that planar velocity vector to derive the joint velocity vector.

- The joint velocity vector is derived because it is then used to find the joint angle matrix which will give input to the robot to perform the motion of drawing a circle.
- X_dot is what we earlier called as v. All the other values except x_dot and z_dot will be zero as previously shown in the velocity vector.

$$\dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \dot{q} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \\ \dot{\theta}_7 \end{bmatrix}$$

$$\dot{q} = J^{-1}\dot{X}$$

The Jacobian and forward kinematic expressions for a robot manipulator can be derived analytically from a Denavit-Hartenberg (D-H) table. These expressions become complicated when the arm consists of more than just two or three joints. For example, some individual terms of the forward kinematic expressions for a six degree-of-freedom PUMA3 robot contain more than 30 trigonometric functions each. With the advent of tools such as MATLAB, it was easier to produce these expressions, but they were still produced for a particular robot through a manual process and then incorporated into robot control or simulation software through another manual process.

The approach is to perform symbolic computations to construct, differentiate, and simplify mathematical expressions used for forward and inverse kinematic computations. The expressions involve variables that can be evaluated numerically for given values of the dependent variables (the joint variables)

- We have previously not defined the Jacobian so here is how we define a jacobian using the second method or the differentiation method.

$$J = \begin{bmatrix} J_v \\ J_\omega \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_i} \\ \frac{\partial y}{\partial \theta_i} \\ \frac{\partial z}{\partial \theta_i} \\ Z_i \end{bmatrix}$$

$$J = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} & \frac{\partial x}{\partial \theta_6} & \frac{\partial x}{\partial \theta_7} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} & \frac{\partial y}{\partial \theta_6} & \frac{\partial y}{\partial \theta_7} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} & \frac{\partial z}{\partial \theta_6} & \frac{\partial z}{\partial \theta_7} \\ z_1 & z_2 & z_4 & z_5 & z_6 & z_7 \end{bmatrix}$$

- The joint velocities are found as follows:

$$\begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_4 \\ \dot{\theta}_5 \\ \dot{\theta}_6 \\ \dot{\theta}_7 \end{bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_4} & \frac{\partial x}{\partial \theta_5} & \frac{\partial x}{\partial \theta_6} & \frac{\partial x}{\partial \theta_7} \\ \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_4} & \frac{\partial y}{\partial \theta_5} & \frac{\partial y}{\partial \theta_6} & \frac{\partial y}{\partial \theta_7} \\ \frac{\partial z}{\partial \theta_1} & \frac{\partial z}{\partial \theta_2} & \frac{\partial z}{\partial \theta_4} & \frac{\partial z}{\partial \theta_5} & \frac{\partial z}{\partial \theta_6} & \frac{\partial z}{\partial \theta_7} \\ z_1 & z_2 & z_4 & z_5 & z_6 & z_7 \end{bmatrix}^{-1} * \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

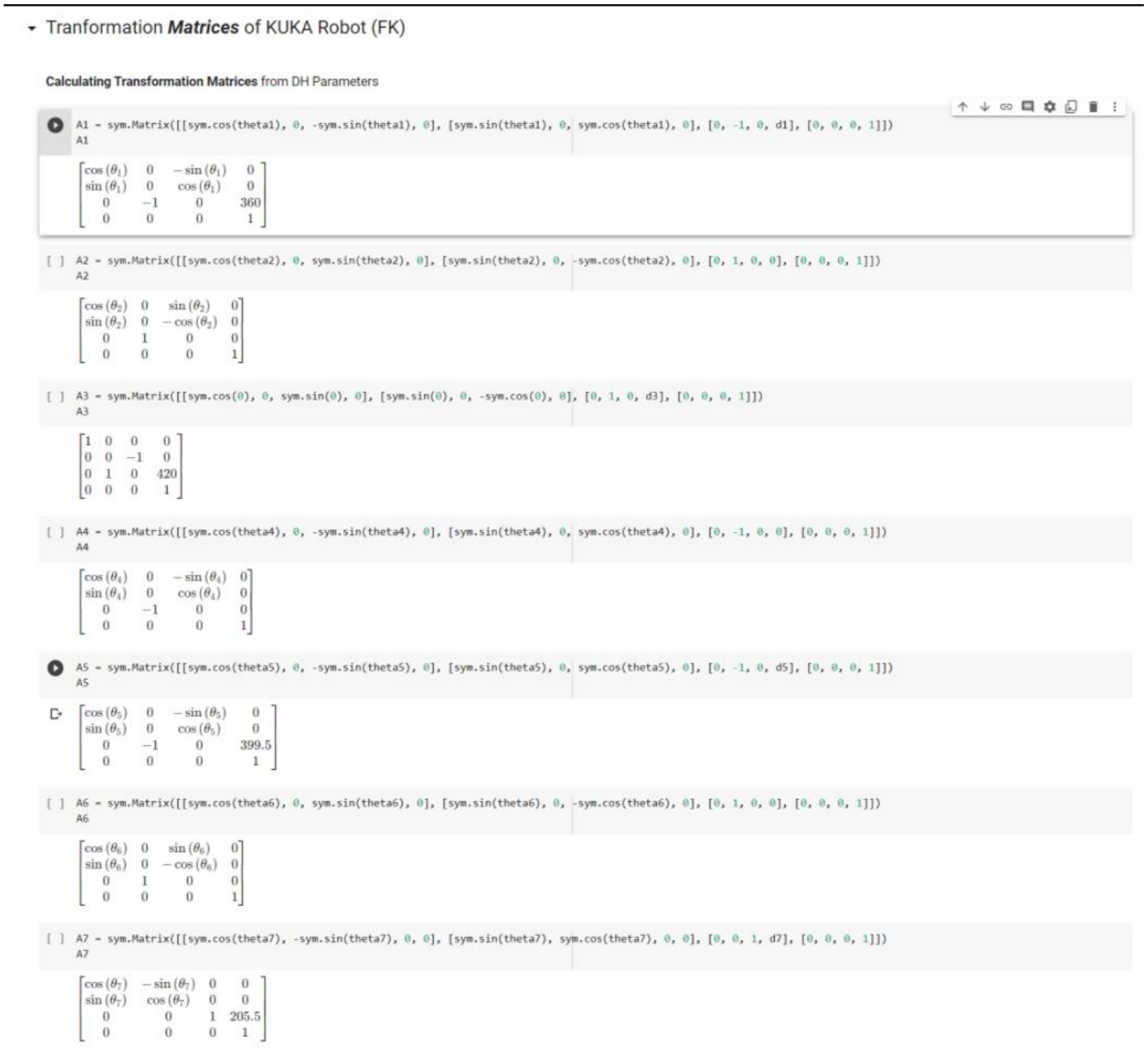
- The joint velocities that we get are used to find the joint angles of the manipulator, which in turn will act as the input parameters to draw the circle.

9. Validation Plan

Validation for the Forward and Inverse Kinematic equations obtained above are described in this section. Python code has been developed to simulate the end effector position coordinates and joint torques. In this section, code snippets have been provided for each step mentioned above and goes on to prove how following the above procedure gives the required output..

The forward kinematics is verified by comparing the expected joint motion with the actual robot's motion obtained in the Gazebo Environment. To validate the algorithm that we have developed upon the Kuka manipulator which represents Cyberknife, the simulation will be of significant understanding.

- Initially in order to simulate, using the information provided by the DH table, the Transformation Matrix at each joint is calculated which yields the rotation and translation matrix at each joint. Multiplying these Transformation Matrices, the Final Transformation Matrix is obtained.



Tranformation ***Matrices*** of KUKA Robot (FK)

Calculating Transformation Matrices from DH Parameters

```
A1 = sym.Matrix([[sym.cos(theta1), 0, -sym.sin(theta1), 0], [sym.sin(theta1), 0, sym.cos(theta1), 0], [0, -1, 0, d1], [0, 0, 0, 1]])
A1
```

$$\begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & -1 & 0 & 360 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
A2 = sym.Matrix([[sym.cos(theta2), 0, sym.sin(theta2), 0], [sym.sin(theta2), 0, -sym.cos(theta2), 0], [0, 1, 0, 0], [0, 0, 0, 1]])
A2
```

$$\begin{bmatrix} \cos(\theta_2) & 0 & \sin(\theta_2) & 0 \\ \sin(\theta_2) & 0 & -\cos(\theta_2) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
A3 = sym.Matrix([[sym.cos(0), 0, sym.sin(0), 0], [sym.sin(0), 0, -sym.cos(0), 0], [0, 1, 0, d3], [0, 0, 0, 1]])
A3
```

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 420 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
A4 = sym.Matrix([[sym.cos(theta4), 0, -sym.sin(theta4), 0], [sym.sin(theta4), 0, sym.cos(theta4), 0], [0, -1, 0, 0], [0, 0, 0, 1]])
A4
```

$$\begin{bmatrix} \cos(\theta_4) & 0 & -\sin(\theta_4) & 0 \\ \sin(\theta_4) & 0 & \cos(\theta_4) & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
A5 = sym.Matrix([[sym.cos(theta5), 0, -sym.sin(theta5), 0], [sym.sin(theta5), 0, sym.cos(theta5), 0], [0, -1, 0, d5], [0, 0, 0, 1]])
A5
```

$$\begin{bmatrix} \cos(\theta_5) & 0 & -\sin(\theta_5) & 0 \\ \sin(\theta_5) & 0 & \cos(\theta_5) & 0 \\ 0 & -1 & 0 & 399.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
A6 = sym.Matrix([[sym.cos(theta6), 0, sym.sin(theta6), 0], [sym.sin(theta6), 0, -sym.cos(theta6), 0], [0, 1, 0, 0], [0, 0, 0, 1]])
A6
```

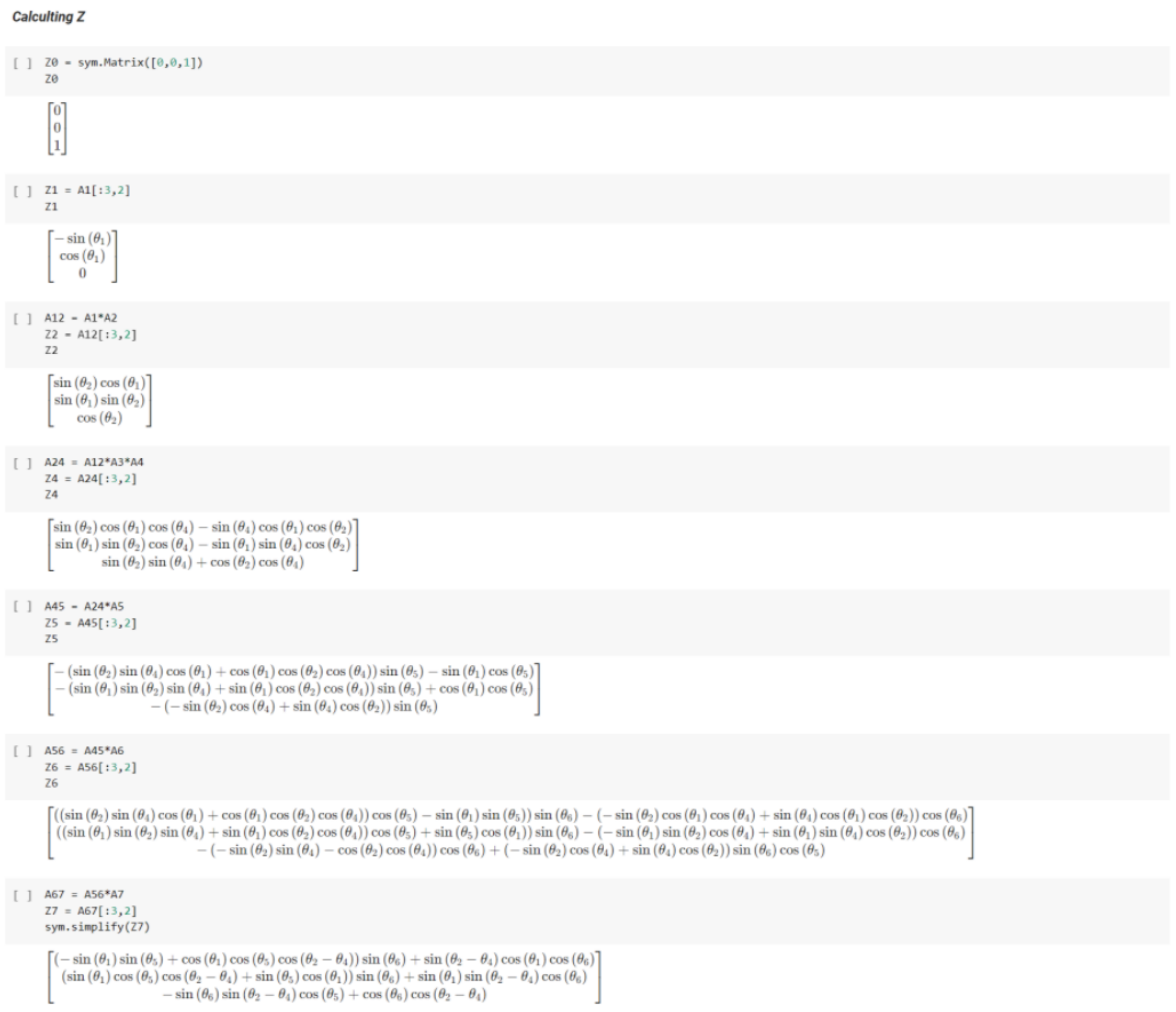
$$\begin{bmatrix} \cos(\theta_6) & 0 & \sin(\theta_6) & 0 \\ \sin(\theta_6) & 0 & -\cos(\theta_6) & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

```
A7 = sym.Matrix([[sym.cos(theta7), -sym.sin(theta7), 0, 0], [sym.sin(theta7), sym.cos(theta7), 0, 0], [0, 0, 1, d7], [0, 0, 0, 1]])
A7
```

$$\begin{bmatrix} \cos(\theta_7) & -\sin(\theta_7) & 0 & 0 \\ \sin(\theta_7) & \cos(\theta_7) & 0 & 0 \\ 0 & 0 & 1 & 205.5 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Figure: Snippet of Transformation Matrices. Refer to Appendix C for code.

- Next, we store the required values from the above obtained transformation matrices to calculate the Jacobian matrix. These values are stored in the variables called ‘Z’ and ‘O’.



Calculting Z

```
[ ] Z0 = sym.Matrix([0,0,1])
    Z0
```

$$\begin{bmatrix}0\\0\\1\end{bmatrix}$$

```
[ ] Z1 = A1[:3,2]
    Z1
```

$$\begin{bmatrix}-\sin(\theta_1)\\ \cos(\theta_1)\\ 0\end{bmatrix}$$

```
[ ] A12 = A1*A2
    Z2 = A12[:3,2]
    Z2
```

$$\begin{bmatrix}\sin(\theta_2)\cos(\theta_1)\\ \sin(\theta_1)\sin(\theta_2)\\ \cos(\theta_2)\end{bmatrix}$$

```
[ ] A24 = A12*A3*A4
    Z4 = A24[:3,2]
    Z4
```

$$\begin{bmatrix}\sin(\theta_2)\cos(\theta_1)\cos(\theta_4)-\sin(\theta_4)\cos(\theta_1)\cos(\theta_2)\\ \sin(\theta_1)\sin(\theta_2)\cos(\theta_4)-\sin(\theta_1)\sin(\theta_4)\cos(\theta_2)\\ \sin(\theta_2)\sin(\theta_4)+\cos(\theta_2)\cos(\theta_4)\end{bmatrix}$$

```
[ ] A45 = A24*A5
    Z5 = A45[:3,2]
    Z5
```

$$\begin{bmatrix}-(\sin(\theta_2)\sin(\theta_4)\cos(\theta_1)+\cos(\theta_1)\cos(\theta_2)\cos(\theta_4))\sin(\theta_5)-\sin(\theta_1)\cos(\theta_5)\\ -(\sin(\theta_1)\sin(\theta_2)\sin(\theta_4)+\sin(\theta_1)\cos(\theta_2)\cos(\theta_4))\sin(\theta_5)+\cos(\theta_1)\cos(\theta_5)\\ -(-\sin(\theta_2)\cos(\theta_4)+\sin(\theta_4)\cos(\theta_2))\sin(\theta_5)\end{bmatrix}$$

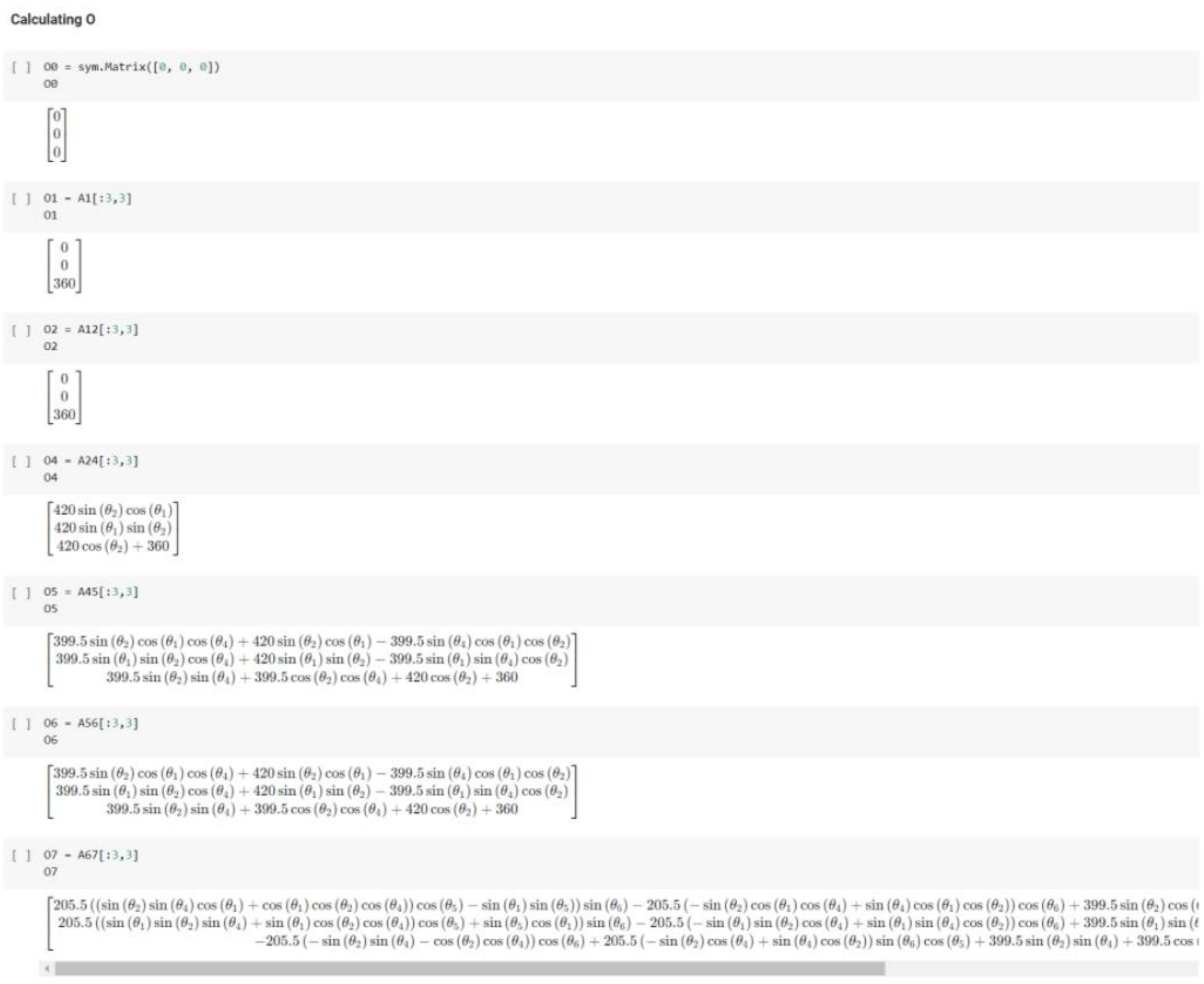
```
[ ] A56 = A45*A6
    Z6 = A56[:3,2]
    Z6
```

$$\begin{bmatrix}((\sin(\theta_2)\sin(\theta_4)\cos(\theta_1)+\cos(\theta_1)\cos(\theta_2)\cos(\theta_4))\cos(\theta_5)-\sin(\theta_1)\sin(\theta_5))\sin(\theta_6)-(-\sin(\theta_2)\cos(\theta_1)\cos(\theta_4)+\sin(\theta_4)\cos(\theta_1)\cos(\theta_2))\cos(\theta_6)\\ ((\sin(\theta_1)\sin(\theta_2)\sin(\theta_4)+\sin(\theta_1)\cos(\theta_2)\cos(\theta_4))\cos(\theta_5)+\sin(\theta_5)\cos(\theta_1))\sin(\theta_6)-(-\sin(\theta_1)\sin(\theta_2)\cos(\theta_4)+\sin(\theta_1)\sin(\theta_4)\cos(\theta_2))\cos(\theta_6)\\ -(-\sin(\theta_2)\sin(\theta_4)-\cos(\theta_2)\cos(\theta_4))\cos(\theta_6)+(-\sin(\theta_2)\cos(\theta_4)+\sin(\theta_4)\cos(\theta_2))\sin(\theta_6)\cos(\theta_5)\end{bmatrix}$$

```
[ ] A67 = A56*A7
    Z7 = A67[:3,2]
    sym.simplify(Z7)
```

$$\begin{bmatrix}(-\sin(\theta_1)\sin(\theta_5)+\cos(\theta_1)\cos(\theta_5)\cos(\theta_2-\theta_4))\sin(\theta_6)+\sin(\theta_2-\theta_4)\cos(\theta_1)\cos(\theta_6)\\ (\sin(\theta_1)\cos(\theta_5)\cos(\theta_2-\theta_4)+\sin(\theta_5)\cos(\theta_1))\sin(\theta_6)+\sin(\theta_1)\sin(\theta_2-\theta_4)\cos(\theta_6)\\ -\sin(\theta_6)\sin(\theta_2-\theta_4)\cos(\theta_5)+\cos(\theta_6)\cos(\theta_2-\theta_4)\end{bmatrix}$$

Figure : Snippet of Z variables. Refer to Appendix C for code.



Calculating O

```
[ ] O0 = sym.Matrix([0, 0, 0])
    O0
```

$$\begin{bmatrix}0\\0\\0\end{bmatrix}$$

```
[ ] O1 = A1[:3,3]
    O1
```

$$\begin{bmatrix}0\\0\\360\end{bmatrix}$$

```
[ ] O2 = A12[:3,3]
    O2
```

$$\begin{bmatrix}0\\0\\360\end{bmatrix}$$

```
[ ] O4 = A24[:3,3]
    O4
```

$$\begin{bmatrix}420\sin(\theta_2)\cos(\theta_1)\\420\sin(\theta_1)\sin(\theta_2)\\420\cos(\theta_2)+360\end{bmatrix}$$

```
[ ] O5 = A45[:3,3]
    O5
```

$$\begin{bmatrix}399.5\sin(\theta_2)\cos(\theta_1)\cos(\theta_4)+420\sin(\theta_2)\cos(\theta_1)-399.5\sin(\theta_4)\cos(\theta_1)\cos(\theta_2)\\399.5\sin(\theta_1)\sin(\theta_2)\cos(\theta_4)+420\sin(\theta_1)\sin(\theta_2)-399.5\sin(\theta_1)\sin(\theta_4)\cos(\theta_2)\\399.5\sin(\theta_2)\sin(\theta_4)+399.5\cos(\theta_2)\cos(\theta_4)+420\cos(\theta_2)+360\end{bmatrix}$$

```
[ ] O6 = A56[:3,3]
    O6
```

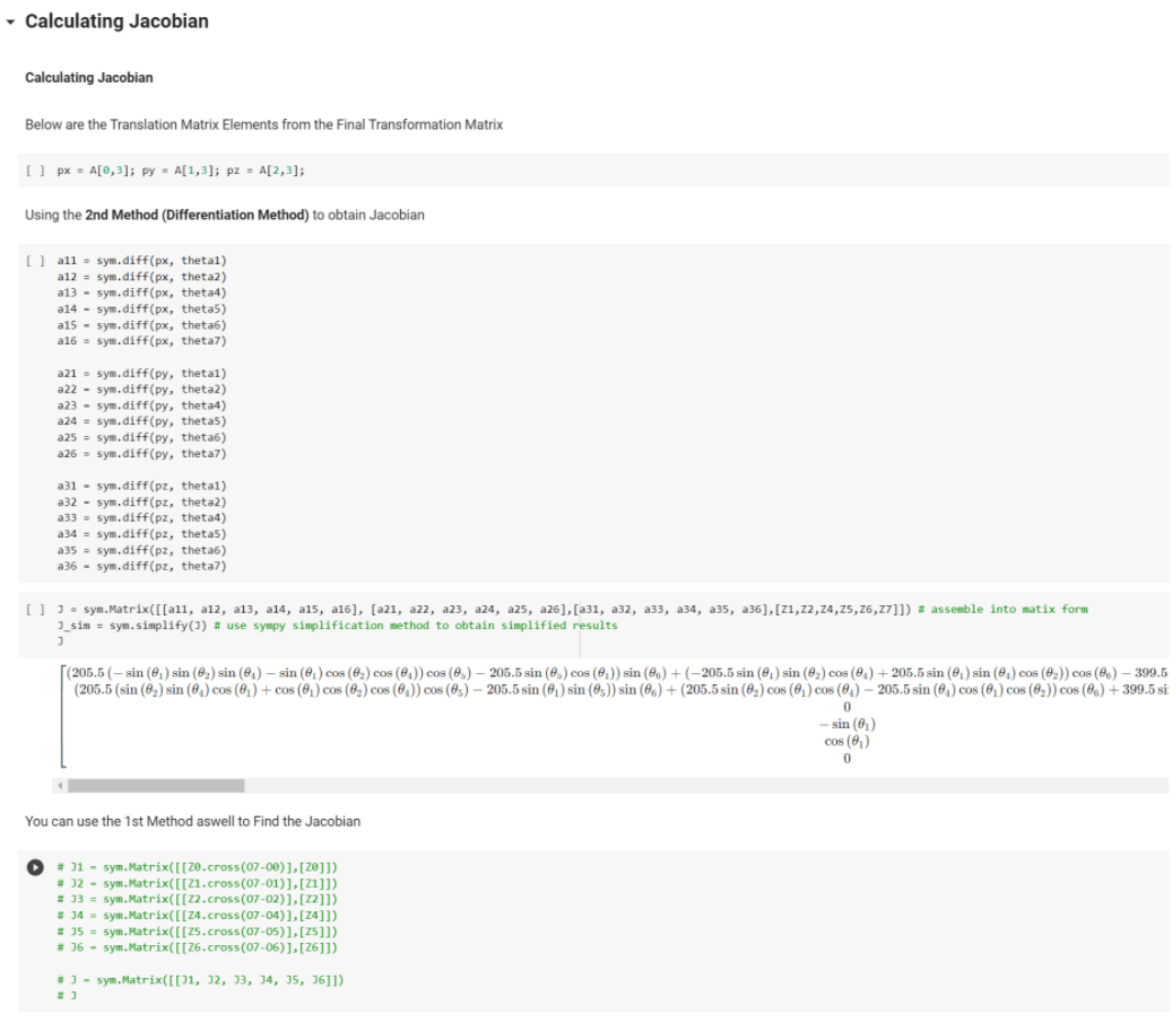
$$\begin{bmatrix}399.5\sin(\theta_2)\cos(\theta_1)\cos(\theta_4)+420\sin(\theta_2)\cos(\theta_1)-399.5\sin(\theta_4)\cos(\theta_1)\cos(\theta_2)\\399.5\sin(\theta_1)\sin(\theta_2)\cos(\theta_4)+420\sin(\theta_1)\sin(\theta_2)-399.5\sin(\theta_1)\sin(\theta_4)\cos(\theta_2)\\399.5\sin(\theta_2)\sin(\theta_4)+399.5\cos(\theta_2)\cos(\theta_4)+420\cos(\theta_2)+360\end{bmatrix}$$

```
[ ] O7 = A67[:3,3]
    O7
```

$$\begin{bmatrix}205.5\left((\sin(\theta_2)\sin(\theta_4)\cos(\theta_1)+\cos(\theta_1)\cos(\theta_2)\cos(\theta_4))\cos(\theta_5)-\sin(\theta_1)\sin(\theta_5)\right)\sin(\theta_6)-205.5\left(-\sin(\theta_2)\cos(\theta_1)\cos(\theta_4)+\sin(\theta_4)\cos(\theta_1)\cos(\theta_2)\right)\cos(\theta_6)+399.5\sin(\theta_2)\cos\\205.5\left((\sin(\theta_1)\sin(\theta_2)\sin(\theta_4)+\sin(\theta_1)\cos(\theta_2)\cos(\theta_4))\cos(\theta_5)+\sin(\theta_5)\cos(\theta_1)\right)\sin(\theta_6)-205.5\left(-\sin(\theta_1)\sin(\theta_2)\cos(\theta_4)+\sin(\theta_1)\sin(\theta_4)\cos(\theta_2)\right)\cos(\theta_6)+399.5\sin(\theta_1)\sin\\-205.5\left(-\sin(\theta_2)\sin(\theta_4)-\cos(\theta_2)\cos(\theta_4)\right)\cos(\theta_6)+205.5\left(-\sin(\theta_2)\cos(\theta_4)+\sin(\theta_4)\cos(\theta_2)\right)\sin(\theta_6)\cos(\theta_5)+399.5\sin(\theta_2)\sin(\theta_4)+399.5\cos\end{bmatrix}$$

Figure : Snippet of O variables having the Translational data. Refer to Appendix C for code.

- The jacobian matrix is calculated using the differential method as delineated in the previous section.



Calculating Jacobian

Calculating Jacobian

Below are the Translation Matrix Elements from the Final Transformation Matrix

```
[ ]  px = A[0,3]; py = A[1,3]; pz = A[2,3];
```

Using the **2nd Method (Differentiation Method)** to obtain Jacobian

```
[ ]  a11 = sym.diff(px, theta1)
     a12 = sym.diff(px, theta2)
     a13 = sym.diff(px, theta4)
     a14 = sym.diff(px, theta5)
     a15 = sym.diff(px, theta6)
     a16 = sym.diff(px, theta7)

     a21 = sym.diff(py, theta1)
     a22 = sym.diff(py, theta2)
     a23 = sym.diff(py, theta4)
     a24 = sym.diff(py, theta5)
     a25 = sym.diff(py, theta6)
     a26 = sym.diff(py, theta7)

     a31 = sym.diff(pz, theta1)
     a32 = sym.diff(pz, theta2)
     a33 = sym.diff(pz, theta4)
     a34 = sym.diff(pz, theta5)
     a35 = sym.diff(pz, theta6)
     a36 = sym.diff(pz, theta7)
```

```
[ ]  J = sym.Matrix([[a11, a12, a13, a14, a15, a16], [a21, a22, a23, a24, a25, a26],[a31, a32, a33, a34, a35, a36],[Z1,Z2,Z4,Z5,Z6,Z7]]) # assemble into matix form
     J_sim = sym.simplify(J) # use sympy simplification method to obtain simplified results
     J
```

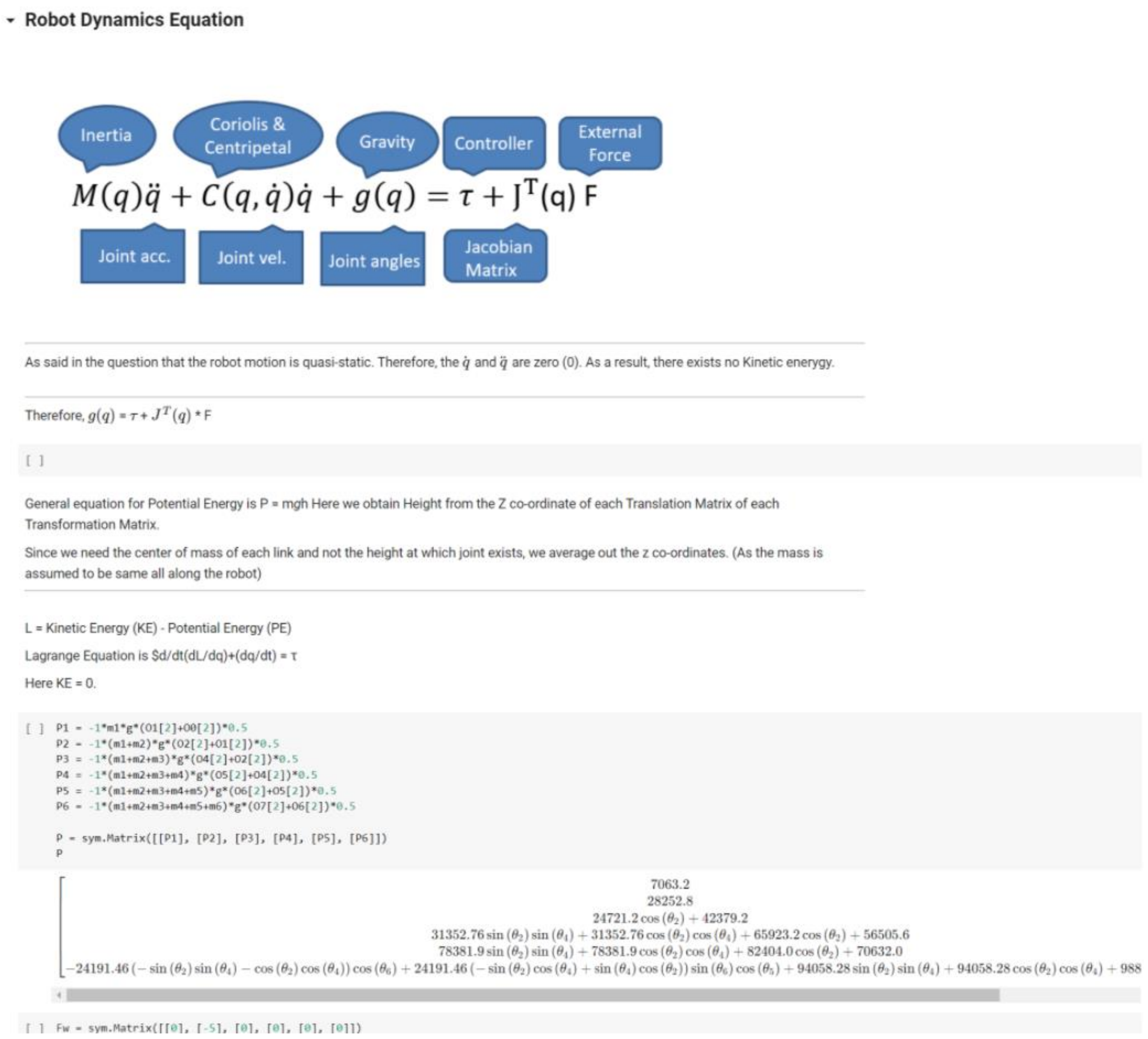
```
[(205.5 (− sin (θ1) sin (θ2) sin (θ4) − sin (θ1) cos (θ2) cos (θ4)) cos (θ5) − 205.5 sin (θ5) cos (θ1)) sin (θ6) + (−205.5 sin (θ1) sin (θ2) cos (θ4) + 205.5 sin (θ1) sin (θ4) cos (θ2)) cos (θ6) − 399.5
 (205.5 (sin (θ2) sin (θ4) cos (θ1) + cos (θ1) cos (θ2) cos (θ4)) cos (θ5) − 205.5 sin (θ1) sin (θ5)) sin (θ6) + (205.5 sin (θ2) cos (θ1) cos (θ4) − 205.5 sin (θ4) cos (θ1) cos (θ2)) cos (θ6) + 399.5 si
                                                                                                   0
                                                                                              − sin (θ1)
                                                                                               cos (θ1)
                                                                                                   0
```

You can use the 1st Method aswell to Find the Jacobian

```
# J1 = sym.Matrix([[Z0.cross(O7-O0)],[Z0]])
# J2 = sym.Matrix([[Z1.cross(O7-O1)],[Z1]])
# J3 = sym.Matrix([[Z2.cross(O7-O2)],[Z2]])
# J4 = sym.Matrix([[Z4.cross(O7-O4)],[Z4]])
# J5 = sym.Matrix([[Z5.cross(O7-O5)],[Z5]])
# J6 = sym.Matrix([[Z6.cross(O7-O6)],[Z6]])

# J = sym.Matrix([[J1, J2, J3, J4, J5, J6]])
# J
```

Figure : Snippet of Jacobian Matrix. Refer to Appendix C for code.

- In order to compute the joint torque values, the Robot Dynamics equation and the lagrangian equation are used. Then a wrench vector is assigned to tell the robot how much force has been exerted by the end effector and in which direction the force has to be exerted.



Robot Dynamics Equation

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + g(q) = \tau + J^T(q)\,F$$

As said in the question that the robot motion is quasi-static. Therefore, the $\dot{q}$ and $\ddot{q}$ are zero (0). As a result, there exists no Kinetic enerygy.

Therefore, $g(q)$ = τ + $J^T(q)$ * F

```
[ ]
```

General equation for Potential Energy is P = mgh Here we obtain Height from the Z co-ordinate of each Translation Matrix of each Transformation Matrix.

Since we need the center of mass of each link and not the height at which joint exists, we average out the z co-ordinates. (As the mass is assumed to be same all along the robot)

L = Kinetic Energy (KE) - Potential Energy (PE)

Lagrange Equation is $d/dt(dL/dq)+(dq/dt) = τ

Here KE = 0.

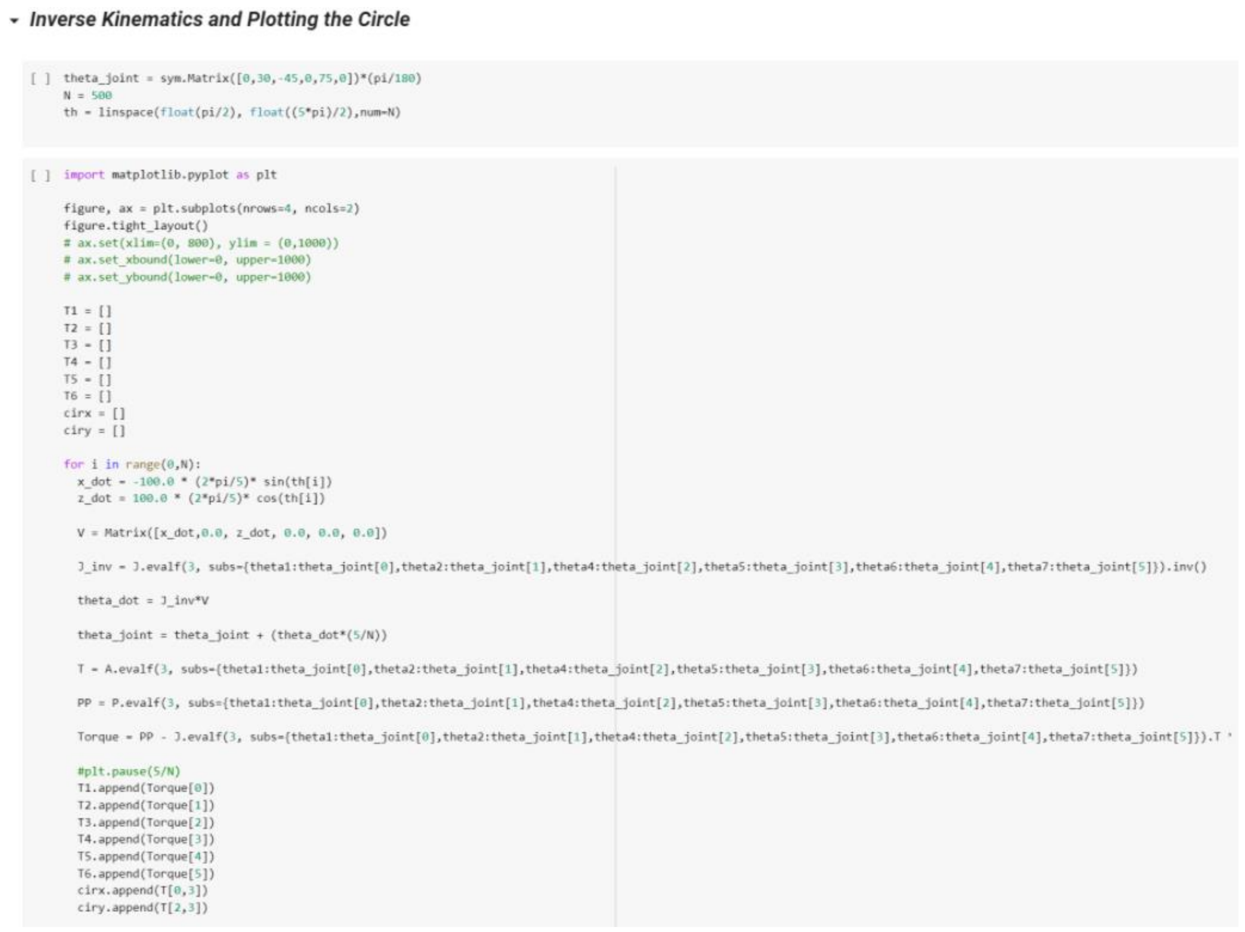
```
[ ] P1 = -1*m1*g*(O1[2]+O0[2])*0.5
    P2 = -1*(m1+m2)*g*(O2[2]+O1[2])*0.5
    P3 = -1*(m1+m2+m3)*g*(O4[2]+O2[2])*0.5
    P4 = -1*(m1+m2+m3+m4)*g*(O5[2]+O4[2])*0.5
    P5 = -1*(m1+m2+m3+m4+m5)*g*(O6[2]+O5[2])*0.5
    P6 = -1*(m1+m2+m3+m4+m5+m6)*g*(O7[2]+O6[2])*0.5

    P = sym.Matrix([[P1], [P2], [P3], [P4], [P5], [P6]])
    P
```

$$\left[\begin{matrix}7063.2\\ 28252.8\\ 24721.2\cos(\theta_2) + 42379.2\\ 31352.76\sin(\theta_2)\sin(\theta_4) + 31352.76\cos(\theta_2)\cos(\theta_4) + 65923.2\cos(\theta_2) + 56505.6\\ 78381.9\sin(\theta_2)\sin(\theta_4) + 78381.9\cos(\theta_2)\cos(\theta_4) + 82404.0\cos(\theta_2) + 70632.0\\ -24191.46(-\sin(\theta_2)\sin(\theta_4) - \cos(\theta_2)\cos(\theta_4))\cos(\theta_6) + 24191.46(-\sin(\theta_2)\cos(\theta_4) + \sin(\theta_4)\cos(\theta_2))\sin(\theta_6)\cos(\theta_5) + 94058.28\sin(\theta_2)\sin(\theta_4) + 94058.28\cos(\theta_2)\cos(\theta_4) + 988\end{matrix}\right.$$

```
[ ] Fw = sym.Matrix([[0], [-5], [0], [0], [0], [0]])
```

Figure : Snippet of Torque values. Refer to Appendix C for code.

- Finally, the inverse kinematics is performed and the theta values are obtained to move the robot's end effector in the desired direction.



▾ *Inverse Kinematics and Plotting the Circle*

```
[ ] theta_joint = sym.Matrix([0,30,-45,0,75,0])*(pi/180)
    N = 500
    th = linspace(float(pi/2), float((5*pi)/2),num=N)
```

```
[ ] import matplotlib.pyplot as plt

    figure, ax = plt.subplots(nrows=4, ncols=2)
    figure.tight_layout()
    # ax.set(xlim=(0, 800), ylim = (0,1000))
    # ax.set_xbound(lower=0, upper=1000)
    # ax.set_ybound(lower=0, upper=1000)

    T1 = []
    T2 = []
    T3 = []
    T4 = []
    T5 = []
    T6 = []
    cirx = []
    ciry = []

    for i in range(0,N):
      x_dot = -100.0 * (2*pi/5)* sin(th[i])
      z_dot = 100.0 * (2*pi/5)* cos(th[i])

      V = Matrix([x_dot,0.0, z_dot, 0.0, 0.0, 0.0])

      J_inv = J.evalf(3, subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]}).inv()

      theta_dot = J_inv*V

      theta_joint = theta_joint + (theta_dot*(5/N))

      T = A.evalf(3, subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]})

      PP = P.evalf(3, subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]})

      Torque = PP - J.evalf(3, subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]}).T *

      #plt.pause(5/N)
      T1.append(Torque[0])
      T2.append(Torque[1])
      T3.append(Torque[2])
      T4.append(Torque[3])
      T5.append(Torque[4])
      T6.append(Torque[5])
      cirx.append(T[0,3])
      ciry.append(T[2,3])
```

Figure : Snippet of Inverse Kinematics. Refer to Appendix C for code.

- The below figure shows the end-effector's motion along the x-z plane. This is the desired motion to be obtained when simulated in the gazebo environment.

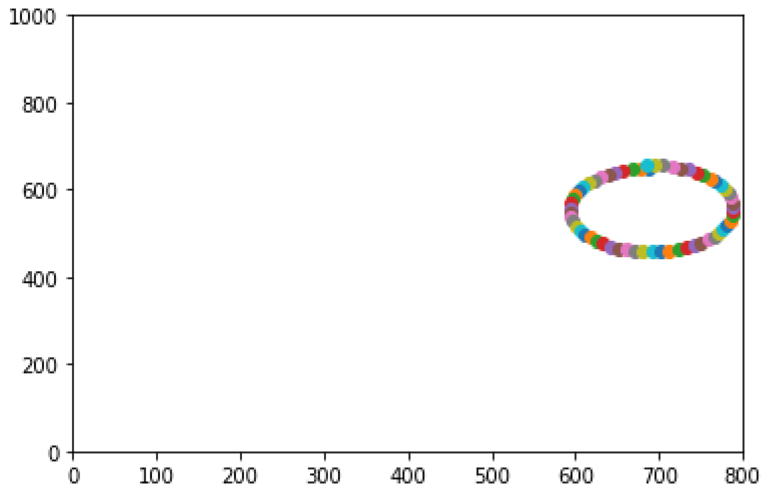


Figure : Snippet of End-effector's motion. Refer to Appendix C for code.

Graphs have been plotted to showcase how the torque values at each joint vary over time.

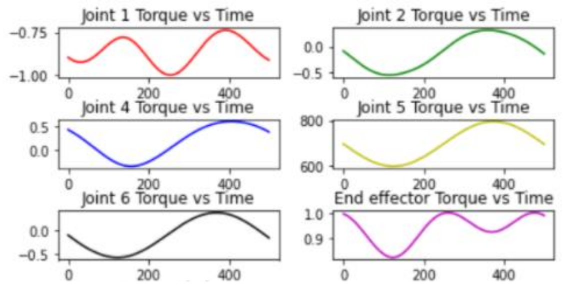


Figure : Snippet of Torque Graphs. Refer to Appendix C for code.

- To automate the process, publisher and subscriber nodes have been developed. The Publisher computes the entire process explained above and using the topic name ‘turn’ sends the messages containing the information of joint angles. The subscriber receives these values from the same topic (‘turn’) and sends the information from the message to each controller of the robot’s joint. These effort controllers respond in accordance with the obtained data, performing the desired motion.

The code for Publisher and Subscriber can be referred from Appendix A and B respectively.

In the gazebo environment, when the publisher and subscriber are launched, the end effector of the Cyber knife manipulator will first move to an intended x,y,z coordinates. This is considered as the initial position and to get to this initial position the cyber knife manipulator will move its joints to a certain pose (position and orientation). This pose is indicated by the user priorly and this pose is decided upon by choosing the plane of action on which the end effector has to be positioned.

Upon the launching of the publisher and subscriber, the robot will first move to its initial position by creating an intended pose like discussed before. Thereafter the cyber knife manipulator will move according to the user input. Here, the user input being a circle, the user also inputs the starting position of the circle from where the robot starts drawing the circle. This starting position is the initial position that has been discussed above. Now if we look at the simulation, the robot is performing the desired action of drawing a circle by moving all of its joints. This desired action is the validation that the code is properly functional.
As a part of using a ROS GUI plug-in, the exact coordinates of each point on the circumference of the circle can be found out, hence tracing the trajectory of the desirable action intended by the user as a given input to the cyberknife manipulator.

We have tried using this ROS GUI plug-in to successfully trace the trajectory of the desired action i.e. drawing the circle but, due to time constraint, our efforts could not be showcased. To achieve this goal was not a part of our proposal but still we gave it a shot and unfortunately it proved to be an ambitious goal which we intend to achieve very soon.

10. Gazebo Visualization

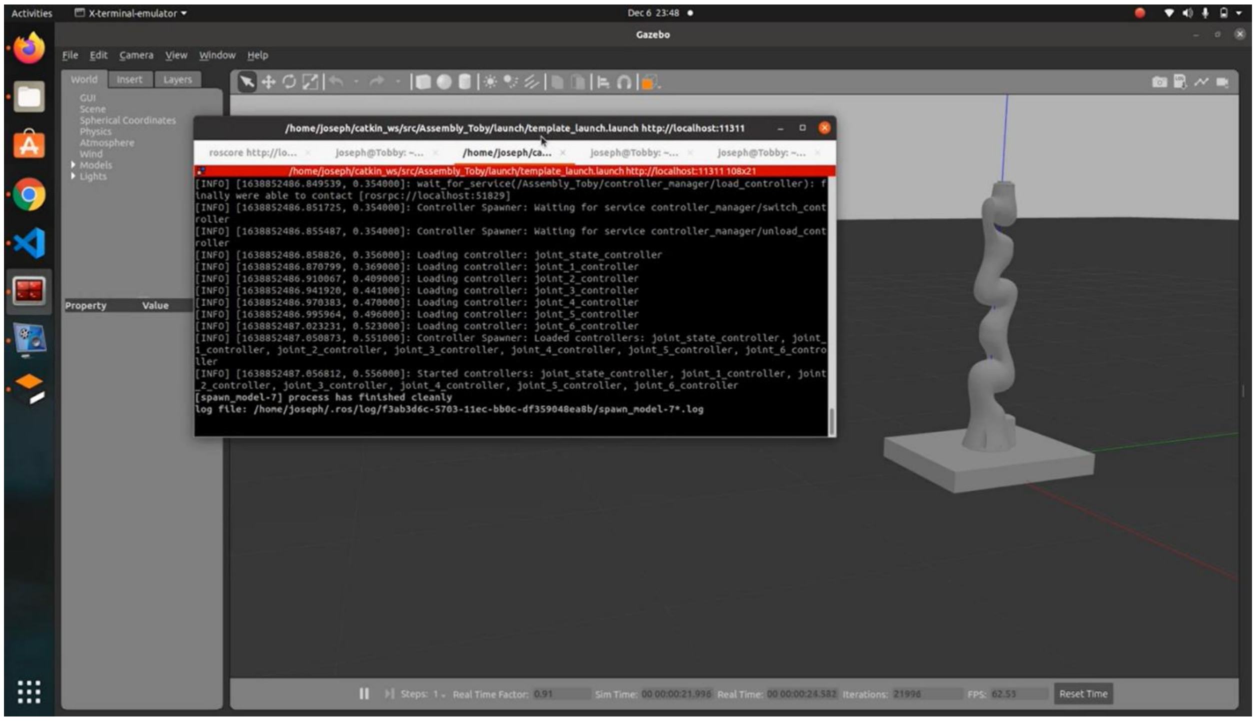
When the robot is spawned in the gazebo world, the KUKA robot stands erect. Once the publisher and subscriber are executed, the robot immediately aligns itself in accordance with the specified initial coordinates in the subscriber module and then goes on to simulate the desired motion from the joint angles received from the publisher. As no sensors were used in the project, rviz hasn’t been used for visualization.

Roadmap to execute the files in the package to Simulate the output.

1. Open a new terminal and build the package: $ catkin_make
2. Run the Gazebo launch file: $ roslaunch Assembly_Toby template_launch.launch
3. In a new terminal, make the python files executable by: $ chmod +x publisher.py and $ chmod +x subscriber.py
4. Run the Publisher Script: $ rosrun Assembly_Toby publisher.py
5. In a new terminal, run the Subscriber Script: $ rosrun Assembly_Toby subscriber.py

Here are the output results from the gazebo simulation:

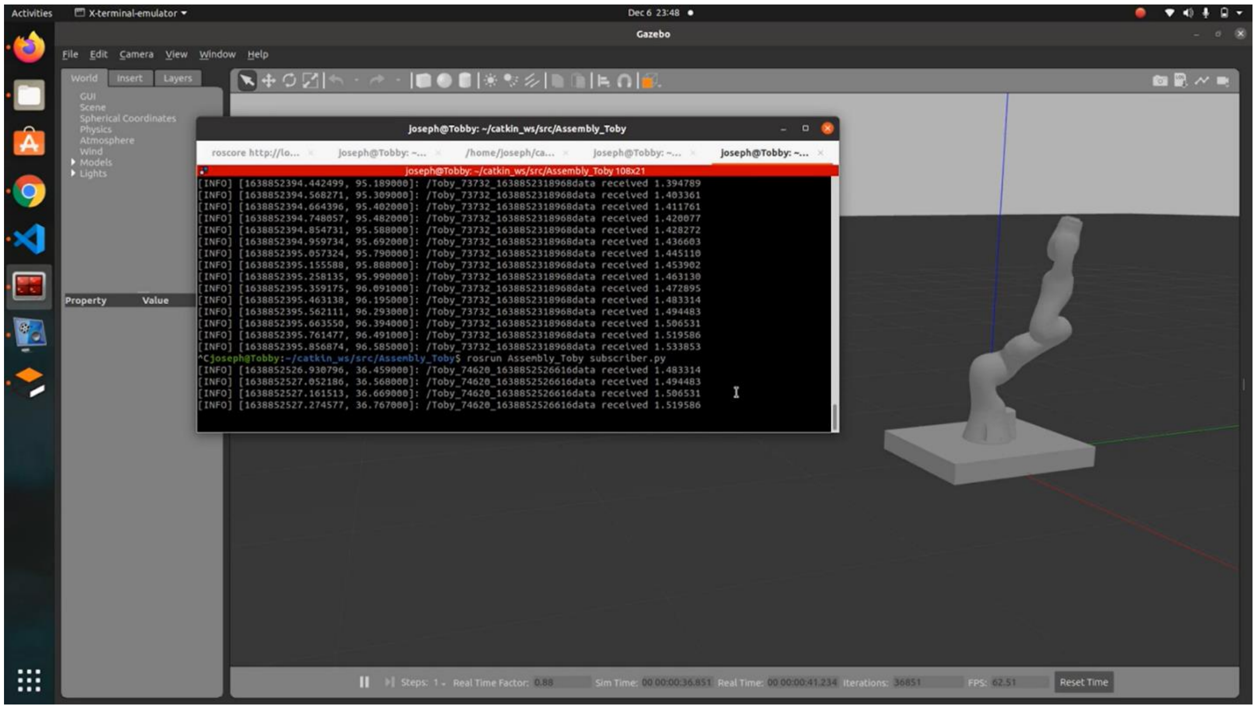
- Robot is spawned in the Gazebo World:



```
/home/joseph/catkin_ws/src/Assembly_Toby/launch/template_launch.launch http://localhost:11311 108x21
[INFO] [1638852486.849539, 0.354000]: wait_for_service(/Assembly_Toby/controller_manager/load_controller): f
inally were able to contact [rosrpc://localhost:51829]
[INFO] [1638852486.851725, 0.354000]: Controller Spawner: Waiting for service controller_manager/switch_cont
roller
[INFO] [1638852486.855487, 0.354000]: Controller Spawner: Waiting for service controller_manager/unload_cont
roller
[INFO] [1638852486.858826, 0.356000]: Loading controller: joint_state_controller
[INFO] [1638852486.870799, 0.369000]: Loading controller: joint_1_controller
[INFO] [1638852486.910067, 0.409000]: Loading controller: joint_2_controller
[INFO] [1638852486.941920, 0.441000]: Loading controller: joint_3_controller
[INFO] [1638852486.970383, 0.470000]: Loading controller: joint_4_controller
[INFO] [1638852486.995964, 0.496000]: Loading controller: joint_5_controller
[INFO] [1638852487.023231, 0.523000]: Loading controller: joint_6_controller
[INFO] [1638852487.050873, 0.551000]: Controller Spawner: Loaded controllers: joint_state_controller, joint_
1_controller, joint_2_controller, joint_3_controller, joint_4_controller, joint_5_controller, joint_6_contro
ller
[INFO] [1638852487.056812, 0.556000]: Started controllers: joint_state_controller, joint_1_controller, joint
_2_controller, joint_3_controller, joint_4_controller, joint_5_controller, joint_6_controller
[spawn_model-7] process has finished cleanly
log file: /home/joseph/.ros/log/f3ab3d6c-5703-11ec-bb0c-df359048ea8b/spawn_model-7*.log
```

Figure : Robot in Gazebo World

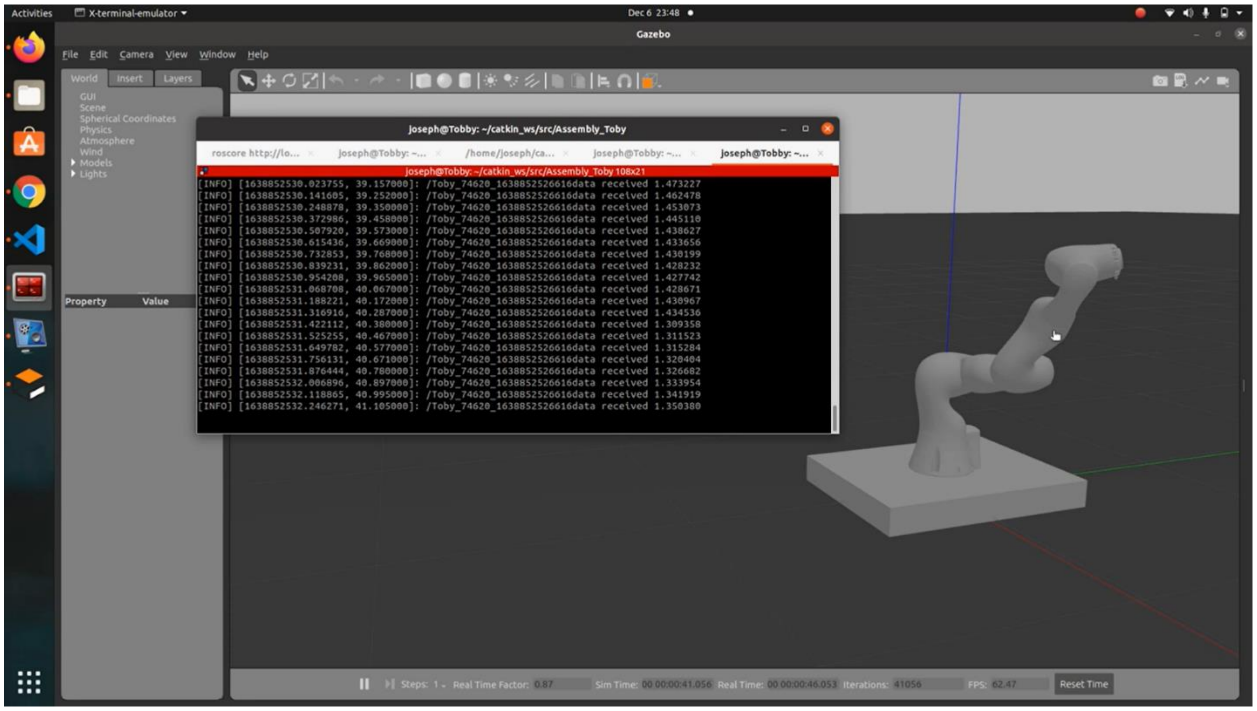
- Robot in the Initial Position:



```
[INFO] [1638852394.442499, 95.189000]: /Toby_73732_1638852318968data received 1.394789
[INFO] [1638852394.568271, 95.309000]: /Toby_73732_1638852318968data received 1.403361
[INFO] [1638852394.664396, 95.402000]: /Toby_73732_1638852318968data received 1.411761
[INFO] [1638852394.748057, 95.482000]: /Toby_73732_1638852318968data received 1.420077
[INFO] [1638852394.854731, 95.588000]: /Toby_73732_1638852318968data received 1.428272
[INFO] [1638852394.959734, 95.692000]: /Toby_73732_1638852318968data received 1.436603
[INFO] [1638852395.057324, 95.790000]: /Toby_73732_1638852318968data received 1.445110
[INFO] [1638852395.155588, 95.888000]: /Toby_73732_1638852318968data received 1.453902
[INFO] [1638852395.258135, 95.990000]: /Toby_73732_1638852318968data received 1.463130
[INFO] [1638852395.359175, 96.091000]: /Toby_73732_1638852318968data received 1.472895
[INFO] [1638852395.463138, 96.195000]: /Toby_73732_1638852318968data received 1.483314
[INFO] [1638852395.562111, 96.293000]: /Toby_73732_1638852318968data received 1.494483
[INFO] [1638852395.663550, 96.394000]: /Toby_73732_1638852318968data received 1.506531
[INFO] [1638852395.761477, 96.491000]: /Toby_73732_1638852318968data received 1.519586
[INFO] [1638852395.856874, 96.585000]: /Toby_73732_1638852318968data received 1.533853
^Cjoseph@Tobby:~/catkin_ws/src/Assembly_Toby$ rosrun Assembly_Toby subscriber.py
[INFO] [1638852526.930796, 36.459000]: /Toby_74620_1638852526616data received 1.483314
[INFO] [1638852527.052186, 36.568000]: /Toby_74620_1638852526616data received 1.494483
[INFO] [1638852527.161513, 36.669000]: /Toby_74620_1638852526616data received 1.506531
[INFO] [1638852527.274577, 36.767000]: /Toby_74620_1638852526616data received 1.519586
```

Figure : Initial Position of Robot

- Robot performing the Desired Motion.



```
[INFO] [1638852530.023755, 39.157000]: /Toby_74620_1638852526616data received 1.473227
[INFO] [1638852530.141605, 39.252000]: /Toby_74620_1638852526616data received 1.462478
[INFO] [1638852530.248878, 39.350000]: /Toby_74620_1638852526616data received 1.453073
[INFO] [1638852530.372986, 39.458000]: /Toby_74620_1638852526616data received 1.445110
[INFO] [1638852530.507920, 39.573000]: /Toby_74620_1638852526616data received 1.438627
[INFO] [1638852530.615436, 39.669000]: /Toby_74620_1638852526616data received 1.433656
[INFO] [1638852530.732853, 39.768000]: /Toby_74620_1638852526616data received 1.430199
[INFO] [1638852530.839231, 39.862000]: /Toby_74620_1638852526616data received 1.428232
[INFO] [1638852530.954208, 39.965000]: /Toby_74620_1638852526616data received 1.427742
[INFO] [1638852531.068708, 40.067000]: /Toby_74620_1638852526616data received 1.428671
[INFO] [1638852531.188221, 40.172000]: /Toby_74620_1638852526616data received 1.430967
[INFO] [1638852531.316916, 40.287000]: /Toby_74620_1638852526616data received 1.434536
[INFO] [1638852531.422112, 40.380000]: /Toby_74620_1638852526616data received 1.309358
[INFO] [1638852531.525255, 40.467000]: /Toby_74620_1638852526616data received 1.311523
[INFO] [1638852531.649782, 40.577000]: /Toby_74620_1638852526616data received 1.315284
[INFO] [1638852531.756131, 40.671000]: /Toby_74620_1638852526616data received 1.320404
[INFO] [1638852531.876444, 40.780000]: /Toby_74620_1638852526616data received 1.326682
[INFO] [1638852532.006896, 40.897000]: /Toby_74620_1638852526616data received 1.333954
[INFO] [1638852532.118865, 40.995000]: /Toby_74620_1638852526616data received 1.341919
[INFO] [1638852532.246271, 41.105000]: /Toby_74620_1638852526616data received 1.350380
```

Figure : Robot performing the desired motion in Gazebo World.

11. Control method

PID controllers are the most common choice of controllers used in most industries. PID stands for Proportional, Integral and Derivative and this controller is a feedback controller that helps attain a set point irrespective of disturbances or any variation in characteristics of the plant of any form. It calculates its output based on the measured error and the three controller gains: proportional gain Kp, integral gain Ki, and derivative gain Kd.

We spent a bit of time studying how to tune the PID gain values for the controller and made the following inferences:

- Proportional term: The motor current is set in proportion to the existing error. The proportional gain simply multiplies the error by a factor K_p. If the arm is too low, forward power is applied. If the arm is too high, reverse power is applied. This method will fail if the arm must lift different weights.
- Integral term: An integral term increases action in relation not only to the error but also the time for which it has persisted. The integral term is a multiplication of the integral gain and the sum of the recent errors. So, if the applied force is not enough to bring the error to zero, this force will be increased as time passes.
- Derivative term: A derivative term does not consider the error (meaning it cannot bring it to zero: a pure D controller cannot bring the system to its setpoint), but the rate of change of error, trying to bring this rate to zero. It aims at flattening the error trajectory into a horizontal line, damping the force applied, and so reduces overshoot.

Finally, we found that, for the purposes of our model, high proportional gain values combined with decently small integral gain values were able to properly stabilize the system.

12. Hardware Implementation

Hardware Implementation was done on the Turtlebot 3- burger model. From the knowledge of project 0, time was dedicated to refresh the concepts of turtlebot. Next we downloaded the existing github repository on turtlebot3 and played around simulating the various packages present in the repository to get familiar with the existing work and understanding how different packages worked and performed different tasks. In the process the potential of the turtlebot 3 was understood. We learnt that turtlebot can be used to perform SLAM, Navigation, Autonomous Driving when camera calibration is done, etc. For this project, SLAM was selected as the task to achieve.

TurtleBot

It is a robot inspired from Roomba, designed with the purpose of teaching the ROS. It is the world's most popular open source robot for teaching and education. It is the most affordable platform with compact size which can be used to extend ideas beyond imaginations with various SBC, sensors, actuators and flexible structure.

Hardware Components:

Core components of Turtlebot3 are the following:

Chassis: Flexible PCB Plate support,

Motors: 2 DYNAMIXEL (XL430-W250-T) which has a cored motor with PID controlled algorithm

Wheels: 1 Ball caster, 2 wheels and tires,

Boards: OpenCR 1.0v and SBC- Raspberry Pi 3B+

Sensors: LIDAR sensor (LDS-01 (HLS-LFCD2))

Battery: SMPS 12V5A and LIPO Battery 11.1V 1,800mAh

CAD Model:

Online source to design Turtlebot CAD model,

ROBOTIS company has provided a platform to develop the CAD model of Turtle Bot online: https://cad.onshape.com/documents/2586c4659ef3e7078e91168b/w/14abf4cb615429a14a2732cc/e/9ae9841864e78c02c4966c5e. The CAD data is released to the Onshape, which is a full-cloud 3D CAD editor. Get access through a web browser from your PC or from portable devices. Onshape allows drawing and assembling parts with co-workers.

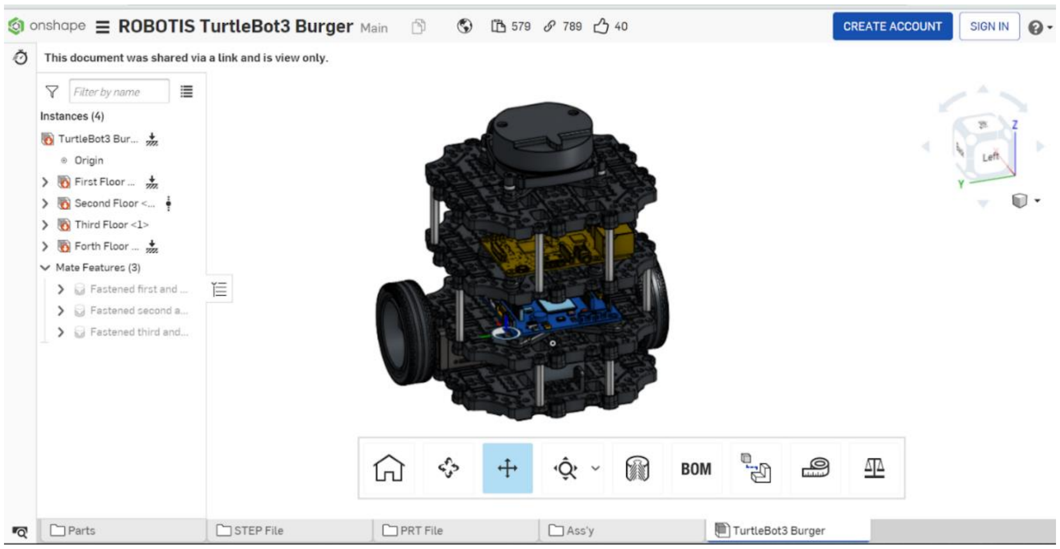


Figure : Editing the CAD model online.

Procedure:

1. Setting up the Remote Connection:

Firstly, in order to obtain the result achieved in Gazebo simulation, the turtle has to be controlled remotely

Setup of Remote desktop:

All the required dependencies for the ros and turtlebot3 packages have been installed.

On the remote desktop:

1. The ip address of the remote desktop is found by $ ifconfig
2. Next, this ip address is made static from the settings, so that the address dosen't change every time the desktop.
3. The bash file is edited using $ nano ~/.bashrc and the ip address of ROS_MASTER_URI and ROS_HOSTNAME is changed to the ip address of the remote desktop.
4. This bashrc is sourced. Therefore, any roslaunch done from here on would open the gazebo world, but rather implement the code on the Hardware.

2. Setup on SBC (Raspberry Pi):

1. The Raspberry Pi is connected to a Monitor and keyboard.
2. The ip address of the SBC is found using $ ifconfig
3. Edit the bashrc file and input the ROS_MASTER_URI with the ip address of the remote desktop and ROS_HOSTNAME with the ip address of the SBC.

2. OpenCR setup:

OpenCR is used for power and sensor control. The OpenCR board comes with the required firmware already uploaded for turtlebot3. OpenCR uses Arduino IDE which makes the board easy to code, to control the sensors and actuators.

3. Running the Turtlebot

The turtlebot is controlled remotely using the ssh connection by executing ssh ubuntu@{IP_ADDRESS_OF_RASPBERRY_PI}

4. Programs

Time was dedicated to understanding the teleoperation program. The program was manipulated to achieve the teleoperation using a different set of keys at different default control speeds. Later, I went on to check the topics of TurtleBot3 using the rqt command provided by ROS. The rqt is a Qt-based framework for GUI development for ROS. The rqt is a tool that allows users to easily see the topic status by displaying all the topics in the topic list.

Various information on odometry, battery stats, diagnostics, sensor statistics and scanned data can be learnt using rqt.

5. SLAM

The **SLAM (Simultaneous Localization and Mapping)** is a technique to draw a map by estimating current location in an arbitrary space. The SLAM is a well-known feature of TurtleBot from its predecessors.

To launch this file,

1. Run roscore from Remote PC
2. Open a new terminal from Remote PC and run the 'bring up' launch file on turtlebot.

```
$ ssh pi@{IP_ADDRESS_OF_RASPBERRY_PI}
$ roslaunch turtlebot3_bringup turtlebot3_robot.launch
```

3. On the remote PC, run the SLAM launch file to start GMapping. The Gmapping is used as a default SLAM method.

```
$ export TURTLEBOT3_MODEL=burger
$ roslaunch turtlebot3_slam turtlebot3_slam.launch
```

The map is drawn based on the robot's odometry, tf and scan information. These map data is drawn in the RViz window as the TurtleBot3 was traveling. After creating a complete map of desired area, the map data can be saved for later use.

The map uses two-dimensional **Occupancy Grid Map (OGM)**, which is commonly used in ROS. **white** area is collision free area while **black** area is occupied and inaccessible area, and **gray** area represents the unknown area. This map is used for the Navigation.

The more time given to generate the data over a distance, more accurate map is obtained.

13.Problems Faced

- ❖ While we have faced quite a few issues during the project, we highlight a few here that we invested a bit of time in. Firstly, the conversion of the STL files into SolidWorks part files was creating issues with improper import of the files - meshes were not getting generated in the way that was intended.

- ❖ Assigning the right coordinate frames to the model in SolidWorks was a big issue, and this had to get sorted because the model orientation was improper when spawned in the Gazebo world. Due to our relatively low exposure to SolidWorks and with the parts being designed by external sources, we could not get the coordinate frames assigned very easily. This was causing many issues with the model – model's Z-axis not being in line with the z-axis in the Gazebo world and the model breaking upon getting spawned. After a lot of time was put in, we were able to assign the frames correctly in the URDF exporter. This finally worked and our model was spawning properly in the environment.

- But there was one pending issue that needed addressing: because of the presence of gravity in the Gazebo world, the model was falling flat after getting spawned. To counter this, we decided to use a payload at the bottom of the assembly that would act as a pedestal/mount for the robot. This finally solved our modelling issues.
- The next set of problems showed up when we tried to spawn the controllers: a bit of time was spent to get the independent joint controllers working. Firstly, due to some errors in the launch file, the controller was not getting spawned. We decided to recreate the launch and the controller yaml files to solve the issue. This worked and the controllers got spawned properly for each of the joints. Now, we wanted to test if we could control the joints in the way we intended to: for this, we made a teleop file for each of the joints and tried running them and we were able to verify that the model was moving the way we intended it to.
- We then went ahead with writing the publisher and subscriber files for the model. After we ran the pub/sub, we observed that the model's PID gain values were not tuned properly and so, we set about correcting them. After several iterations of tuning the gain values, we landed upon the set of values that we finally managed to deploy.

14. Takeaways from the Project (Lessons Learnt)

- Working on this project was, in a way, putting all the knowledge and learnings we attained in the course to practical use. We have learned the design/basics of modelling using SolidWorks - sizing up and tailoring the model to our requirements. From being able to build the model properly to assigning proper coordinate frames to the model and then deploying, we have gained good introductory knowledge of the kinematics and dynamics associated when working in the world of robotics.
- We were able to apply most of the concepts learned: Transformation Matrices, assigning coordinate frames and getting the DH parameters and applying forward and inverse kinematics on the model of our choice (which is a popular industrial robot). We were able to validate our model using scripts written in Python, so we had to learn a bit of programming in Python.
- We have learned the nuances of ROS, now being able to understand how to simulate a robot working in a virtual, controlled environment. But working on the Turtlebot has also made us understand that modelling in a virtual environment and real-world implementation are two very different/distinct things. While modelling and simulation has its own set of challenges, putting that simulation to practice requires a degree of knowledge and consideration of a lot of other different factors. We learned how to connect and control the robot virtually through a remote device and this required learning the basics and working knowledge of the robot involved.

15. Discussion

During the course of working on this project we have learned a lot and achieved many of the tasks we wanted to achieve. 1) We learnt about the Cyberknife Model. 2) Creating a CAD model and assigning the axis to spawn properly in the gazebo environment. 3) Assigning the controllers to the joints. 4) Tweaking PID values to get a stable output. 5) Manually controlling the robot using TELEOP operation. 6) Automating the process using Publisher and

Subscriber.7) Hardware Implementation on Turtlebot3 by performing gmapping and navigation.

16.Conclusion

Robotic surgery is entering its adulthood due to the continuous development made by research groups all over the world. From the close co-operation of engineers and physicians' great medical robotic innovations were born to increase treatment delivery precision and augment human dexterity.

In the past two decades, most of the developed systems integrated industrial robots and concepts, however, new manipulators tend to be custom designed to better serve the targete medical field's purposes. Just like industrial robotics before, surgical robotics must find its most profitable application area and reach the critical market size that can support the significant R&D spendings required. In the short term future, computer-integrated surgical technology will already reshape human healthcare.

17.References:

1. Warren Kilby, Michael Naylor, John R. Dooley, Calvin R. Maurer, Jr. and Sohail Sayeh, "A Technical Overview of the CyberKnife System" Handbook of Robotic and Image-Guided Surgery, 2020, Pages 15-38.
2. W. Kilby M.Sc.*, J. R. Dooley, Ph.D., G. Kuduvalli, Ph.D., S. Sayeh, M.S., C. R. Maurer Jr., Ph.D. ,"The CyberKnife Robotic Radiosurgery System in 2010", Technology in Cancer Research and Treatment, ISSN 1533-0346, Volume 9, Number 5, October 2010.
3. Gopalakrishna Kurup, "CyberKnife: A new paradigm in radiotherapy", Journal of Medical Physics, Vol. 35, No. 2, 63-4.
4. Steven D. Chang John R. Adler, "Robotics and Radiosurgery – The Cyberknife" 13th Meet World Soc Stereotact Funct Neurosurg, Adelaide 2001 Stereotact Funct Neurosurg 2001;76:204–208

Appendix

The authentic code used to get the simulated outputs and validation have been given below.

APPENDIX A

1. Publisher CODE:

```
#!/usr/bin/env python3

# Importing necessary Libraries
import rospy
from std_msgs.msg import Float64
from std_msgs.msg import Float64MultiArray
from std_msgs.msg import String
from geometry_msgs.msg import Twist

##############
import sympy as sym
sym.init_printing()
from sympy import *

import numpy as np
from numpy import *
import math
################

#Initializing Variables

theta1, theta2, theta3, theta4, theta5, theta6, theta7 =
sym.symbols("\\theta_1, theta_2, theta_3, theta_4, theta_5, theta_6,
theta_7")
d1, d3, d5, d7 = sym.symbols("d_1,d_3,d_5,d_7")

d1 = 360
d3 = 420
d5 = 399.5
d7 = 205.5


def plotcircle():
    #Initializing Node
    rospy.init_node('publish_node', anonymous=True) # defining the ros
node - publish node
    turning = rospy.Publisher('turn', Float64MultiArray, queue_size=10)
    rate = rospy.Rate(10) # 10hz # fequency at which the publishing occurs
    rospy.loginfo("Analysing the Robot!!!")  # to print on the terminal

    # Obtaining Transformation Matrices
    A1 = sym.Matrix([[sym.cos(theta1), 0, -sym.sin(theta1), 0],
[sym.sin(theta1), 0, sym.cos(theta1), 0], [0, -1, 0, d1], [0, 0, 0, 1]])
    A2 = sym.Matrix([[sym.cos(theta2), 0, sym.sin(theta2), 0],
[sym.sin(theta2), 0, -sym.cos(theta2), 0], [0, 1, 0, 0], [0, 0, 0, 1]])
    A3 = sym.Matrix([[sym.cos(0), 0, sym.sin(0), 0], [sym.sin(0), 0, -
sym.cos(0), 0], [0, 1, 0, d3], [0, 0, 0, 1]])
    A4 = sym.Matrix([[sym.cos(theta4), 0, -sym.sin(theta4), 0],
[sym.sin(theta4), 0, sym.cos(theta4), 0], [0, -1, 0, 0], [0, 0, 0, 1]])
    A5 = sym.Matrix([[sym.cos(theta5), 0, -sym.sin(theta5), 0],
[sym.sin(theta5), 0, sym.cos(theta5), 0], [0, -1, 0, d5], [0, 0, 0, 1]])
    A6 = sym.Matrix([[sym.cos(theta6), 0, sym.sin(theta6), 0],
[sym.sin(theta6), 0, -sym.cos(theta6), 0], [0, 1, 0, 0], [0, 0, 0, 1]])
    A7 = sym.Matrix([[sym.cos(theta7), -sym.sin(theta7), 0, 0],
[sym.sin(theta7), sym.cos(theta7), 0, 0], [0, 0, 1, d7], [0, 0, 0, 1]])

    A = A1*A2*A3*A4*A5*A6*A7
    A12 = A1*A2
    A24 = A12*A3*A4
    A45 = A24*A5
    A56 = A45*A6
    A67 = A56*A7

    # Obtaining Z Matrices from Transformation Matrix
    Z0 = sym.Matrix([0,0,1])
    Z1 = A1[:3,2]
    Z2 = A12[:3,2]
    Z4 = A24[:3,2]
    Z5 = A45[:3,2]
    Z6 = A56[:3,2]
    Z7 = A67[:3,2]

    #Obtaining O Matrix (Translation) from Transformation Matrix
    O0 = sym.Matrix([0, 0, 0])
    O1 = A1[:3,3]
    O2 = A12[:3,3]
    O4 = A24[:3,3]
    O5 = A45[:3,3]
    O6 = A56[:3,3]
    O7 = A67[:3,3]

    # Calculating Jacobian
    px = A[0,3]; py = A[1,3]; pz = A[2,3];
    a11 = sym.diff(px, theta1)
    a12 = sym.diff(px, theta2)
    a13 = sym.diff(px, theta4)
    a14 = sym.diff(px, theta5)
    a15 = sym.diff(px, theta6)
    a16 = sym.diff(px, theta7)

    a21 = sym.diff(py, theta1)
    a22 = sym.diff(py, theta2)
    a23 = sym.diff(py, theta4)
    a24 = sym.diff(py, theta5)
    a25 = sym.diff(py, theta6)
    a26 = sym.diff(py, theta7)

    a31 = sym.diff(pz, theta1)
    a32 = sym.diff(pz, theta2)
    a33 = sym.diff(pz, theta4)
    a34 = sym.diff(pz, theta5)
    a35 = sym.diff(pz, theta6)
    a36 = sym.diff(pz, theta7)

    J = sym.Matrix([[a11, a12, a13, a14, a15, a16], [a21, a22, a23, a24,
a25, a26],[a31, a32, a33, a34, a35, a36],[Z1,Z2,Z4,Z5,Z6,Z7]])

    # Calculate Potential Energy
    P1 = -1*m1*g*(O1[2]+O0[2])*0.5
    P2 = -1*(m1+m2)*g*(O2[2]+O1[2])*0.5
    P3 = -1*(m1+m2+m3)*g*(O4[2]+O2[2])*0.5
    P4 = -1*(m1+m2+m3+m4)*g*(O5[2]+O4[2])*0.5
    P5 = -1*(m1+m2+m3+m4+m5)*g*(O6[2]+O5[2])*0.5
    P6 = -1*(m1+m2+m3+m4+m5+m6)*g*(O7[2]+O6[2])*0.5

    P = sym.Matrix([[P1], [P2], [P3], [P4], [P5], [P6]])

    # Wrench Vector for FORCE
    Fw = sym.Matrix([[0], [-5], [0], [0], [0], [0]])

    # # Inverse Kinematics
    # theta_joint = sym.Matrix([0,30,-45,0,75,0])*(pi/180)
    # N = 60
    # th = linspace(float(pi/2), float((5*pi)/2),num=N)

    while not rospy.is_shutdown():

        # Inverse Kinematics
        theta_joint = sym.Matrix([0,75,-60,60,45,0])*(pi/180)
        N = 60
        th = linspace(float(pi/2), float((5*pi)/2),num=N)

        T1 = []
        T2 = []
        T3 = []
        T4 = []
        T5 = []
        T6 = []
        cirx = []
        ciry = []

        old_min = -0.88
        old_max = 2.00
        new_min = -0.5
        new_max = 0.5

        for i in range(0,N):
            twist = Float64MultiArray()

            x_dot = -100.0 * (2*pi/5)* sin(th[i])
            z_dot = 100.0 * (2*pi/5)* cos(th[i])

            V = Matrix([x_dot,0.0, z_dot, 0.0, 0.0, 0.0])

            J_inv = J.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],th
eta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]}).inv()

            theta_dot = J_inv*V

            theta_joint = theta_joint + (theta_dot*(5/N))

            #T = A.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],th
eta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]})
            #PP = P.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],th
eta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]})

            #Torque = PP - J.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],th
eta5:theta_joint[3],theta6:theta_joint[4],theta7:theta_joint[5]}).T * Fw

            #new_value = ( (theta_joint - old_min) / (old_max - old_min) )
* (new_max - new_min) + new_min


            print(theta_joint)
            twist.data = theta_joint
            #print(test)
            turning.publish(twist)
            rate.sleep()

    #plt.scatter(T[0,3],T[2,3])

if __name__ == '__main__':
    try:
        plotcircle()
    except rospy.ROSInterruptException:
        pass
```

2. Subscriber CODE:

```
#! /usr/bin/env python3

from numpy.lib.financial import rate
import rospy
```

```
from std_msgs.msg import Float64
from std_msgs.msg import Float64MultiArray
from geometry_msgs.msg import Twist


pub1_right = rospy.Publisher('Assembly_Toby/joint_1_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
pub1_left = rospy.Publisher('Assembly_Toby/joint_1_controller/command',
Float64, queue_size=10)

pub2_right = rospy.Publisher('Assembly_Toby/joint_2_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
pub2_left = rospy.Publisher('Assembly_Toby/joint_2_controller/command',
Float64, queue_size=10)

# pub3_right = rospy.Publisher('Assembly_Toby/joint_3_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
# pub3_left = rospy.Publisher('Assembly_Toby/joint_3_controller/command',
Float64, queue_size=10)

pub4_right = rospy.Publisher('Assembly_Toby/joint_4_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
pub4_left = rospy.Publisher('Assembly_Toby/joint_4_controller/command',
Float64, queue_size=10)

pub5_right = rospy.Publisher('Assembly_Toby/joint_5_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
pub5_left = rospy.Publisher('Assembly_Toby/joint_5_controller/command',
Float64, queue_size=10)

pub6_right = rospy.Publisher('Assembly_Toby/joint_6_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
pub6_left = rospy.Publisher('Assembly_Toby/joint_6_controller/command',
Float64, queue_size=10)

pub7_right = rospy.Publisher('Assembly_Toby/joint_7_controller/command',
Float64, queue_size=10) # Add your topic here between ''. Eg
'/my_robot/steering_controller/command'
pub7_left = rospy.Publisher('Assembly_Toby/joint_7_controller/command',
Float64, queue_size=10)

twist = Float64()


def turn_push(data):
    rospy.loginfo(rospy.get_caller_id() + "data received %f",
data.data[1])
```

```
    #### Initial Values to set the pose of robot
    twist.data = 0 # 0 Degrees
    pub1_right.publish(twist)
    pub1_left.publish(twist)

    twist.data = 1.309 # 75 Degrees
    pub2_right.publish(twist)
    pub2_left.publish(twist)

    # twist.data = 0 # 0 Degrees
    # pub3_right.publish(twist)
    # pub3_left.publish(twist)

    twist.data = -1.0472 # -60 Degrees
    pub4_right.publish(twist)
    pub4_left.publish(twist)

    twist.data = 1.0472 # 60 Degrees
    pub5_right.publish(twist)
    pub5_left.publish(twist)

    twist.data = 0.785398 # 45 Degrees
    pub6_right.publish(twist)
    pub6_left.publish(twist)

    twist.data = 0 # 0 Degrees
    pub7_right.publish(twist)
    pub7_left.publish(twist)



    #### Angels from the publisher
    pub1_right.publish(data.data[0])
    pub1_left.publish(data.data[0])

    pub2_right.publish(data.data[1])
    pub2_left.publish(data.data[1])

    pub4_right.publish(data.data[2])
    pub4_left.publish(data.data[2])

    pub5_right.publish(data.data[3])
    pub5_left.publish(data.data[3])

    pub6_right.publish(data.data[4])
    pub6_left.publish(data.data[4])

    pub7_right.publish(data.data[5])
    pub7_left.publish(data.data[5])
```

```
def listener():

    rospy.init_node('Toby', anonymous=True)

    rospy.Subscriber("turn", Float64MultiArray, turn_push)

    rospy.spin()

if __name__ == '__main__':
    listener()
```

3. Validation CODE

```
## **Mass Information from KUKA WIIA datasheet**

The total weight of the KUKA LWR is 23.9 kg
Assumption: Considering the weight of each link of KUKA to be equal due to the lack of
sufficient data.
"""

m1 = 4
m2 = 4
m3 = 4
m4 = 4
m5 = 4
m6 = 4

# Taking Gravitational Constant NEGATIVE to obtain positive Potential Energy
g = -9.81

"""##**Importing Libraries**

**SimPy Library** will allow us to develop and manipulate symbolic expressions.

In SymPy, we initialize printing so that all of the mathematical equations are rendered in
standard mathematical notation.
"""

import sympy as sym
sym.init_printing()

from sympy import *

from sympy.physics.vector import Vector
Vector.simp = True

import numpy as np
from numpy import *
```

```
import math


Initialization of Variables
"""

theta1, theta2, theta3, theta4, theta5, theta6, theta7 = sym.symbols("\\theta_1, theta_2, theta_3,
theta_4, theta_5, theta_6, theta_7")
d1, d3, d5, d7 = sym.symbols("d_1,d_3,d_5,d_7")

d1 = 360
d3 = 420
d5 = 399.5
d7 = 205.5

"""## Tranformation ***Matrices*** of KUKA Robot (FK)

**Calculating Transformation Matrices** from DH Parameters
"""

A1 = sym.Matrix([[sym.cos(theta1), 0, -sym.sin(theta1), 0], [sym.sin(theta1), 0, sym.cos(theta1),
0], [0, -1, 0, d1], [0, 0, 0, 1]])
A1

A2 = sym.Matrix([[sym.cos(theta2), 0, sym.sin(theta2), 0], [sym.sin(theta2), 0, -sym.cos(theta2),
0], [0, 1, 0, 0], [0, 0, 0, 1]])
A2

A3 = sym.Matrix([[sym.cos(0), 0, sym.sin(0), 0], [sym.sin(0), 0, -sym.cos(0), 0], [0, 1, 0, d3], [0,
0, 0, 1]])
A3

A4 = sym.Matrix([[sym.cos(theta4), 0, -sym.sin(theta4), 0], [sym.sin(theta4), 0, sym.cos(theta4),
0], [0, -1, 0, 0], [0, 0, 0, 1]])
A4

A5 = sym.Matrix([[sym.cos(theta5), 0, -sym.sin(theta5), 0], [sym.sin(theta5), 0, sym.cos(theta5),
0], [0, -1, 0, d5], [0, 0, 0, 1]])
A5

A6 = sym.Matrix([[sym.cos(theta6), 0, sym.sin(theta6), 0], [sym.sin(theta6), 0, -sym.cos(theta6),
0], [0, 1, 0, 0], [0, 0, 0, 1]])
A6

A7 = sym.Matrix([[sym.cos(theta7), -sym.sin(theta7), 0, 0], [sym.sin(theta7), sym.cos(theta7), 0,
0], [0, 0, 1, d7], [0, 0, 0, 1]])
A7

"""**Final Transformation Matrix**"""
```

```
A = A1*A2*A3*A4*A5*A6*A7
A



"""##**Calculating Z and O**

***Calculting Z***
"""

Z0 = sym.Matrix([0,0,1])
Z0

Z1 = A1[:3,2]
Z1

A12 = A1*A2
Z2 = A12[:3,2]
Z2

A24 = A12*A3*A4
Z4 = A24[:3,2]
Z4

A45 = A24*A5
Z5 = A45[:3,2]
Z5

A56 = A45*A6
Z6 = A56[:3,2]
Z6

A67 = A56*A7
Z7 = A67[:3,2]
sym.simplify(Z7)

"""**Calculating O**"""

O0 = sym.Matrix([0, 0, 0])
O0

O1 = A1[:3,3]
O1

O2 = A12[:3,3]
O2

O4 = A24[:3,3]
O4
```

```
O5 = A45[:3,3]
O5

O6 = A56[:3,3]
O6

O7 = A67[:3,3]
O7

"""## **Calculating Jacobian**

**Calculating Jacobian**

Below are the Translation Matrix Elements from the Final Transformation Matrix
"""

px = A[0,3]; py = A[1,3]; pz = A[2,3];

"""Using the **2nd Method (Differentiation Method)** to obtain Jacobian"""

a11 = sym.diff(px, theta1)
a12 = sym.diff(px, theta2)
a13 = sym.diff(px, theta4)
a14 = sym.diff(px, theta5)
a15 = sym.diff(px, theta6)
a16 = sym.diff(px, theta7)

a21 = sym.diff(py, theta1)
a22 = sym.diff(py, theta2)
a23 = sym.diff(py, theta4)
a24 = sym.diff(py, theta5)
a25 = sym.diff(py, theta6)
a26 = sym.diff(py, theta7)

a31 = sym.diff(pz, theta1)
a32 = sym.diff(pz, theta2)
a33 = sym.diff(pz, theta4)
a34 = sym.diff(pz, theta5)
a35 = sym.diff(pz, theta6)
a36 = sym.diff(pz, theta7)

J = sym.Matrix([[a11, a12, a13, a14, a15, a16], [a21, a22, a23, a24, a25, a26],[a31, a32, a33,
a34, a35, a36],[Z1,Z2,Z4,Z5,Z6,Z7]]) # assemble into matix form
J_sim = sym.simplify(J) # use sympy simplification method to obtain simplified results
J

"""You can use the 1st Method aswell to Find the Jacobian"""
```

```
# J1 = sym.Matrix([[Z0.cross(O7-O0)],[Z0]])
# J2 = sym.Matrix([[Z1.cross(O7-O1)],[Z1]])
# J3 = sym.Matrix([[Z2.cross(O7-O2)],[Z2]])
# J4 = sym.Matrix([[Z4.cross(O7-O4)],[Z4]])
# J5 = sym.Matrix([[Z5.cross(O7-O5)],[Z5]])
# J6 = sym.Matrix([[Z6.cross(O7-O6)],[Z6]])

# J = sym.Matrix([[J1, J2, J3, J4, J5, J6]])
# J



P1 = -1*m1*g*(O1[2]+O0[2])*0.5
P2 = -1*(m1+m2)*g*(O2[2]+O1[2])*0.5
P3 = -1*(m1+m2+m3)*g*(O4[2]+O2[2])*0.5
P4 = -1*(m1+m2+m3+m4)*g*(O5[2]+O4[2])*0.5
P5 = -1*(m1+m2+m3+m4+m5)*g*(O6[2]+O5[2])*0.5
P6 = -1*(m1+m2+m3+m4+m5+m6)*g*(O7[2]+O6[2])*0.5

P = sym.Matrix([[P1], [P2], [P3], [P4], [P5], [P6]])
P

Fw = sym.Matrix([[0], [-5], [0], [0], [0], [0]])
P-J.T * Fw

"""##***Inverse Kinematics and Plotting the Circle***


"""

theta_joint = sym.Matrix([0,30,-45,0,75,0])*(pi/180)
N = 500
th = linspace(float(pi/2), float((5*pi)/2),num=N)

import matplotlib.pyplot as plt

figure, ax = plt.subplots(nrows=4, ncols=2)
figure.tight_layout()

T1 = []
T2 = []
T3 = []
T4 = []
T5 = []
T6 = []
cirx = []
ciry = []

for i in range(0,N):
  x_dot = -100.0 * (2*pi/5)* sin(th[i])
  z_dot = 100.0 * (2*pi/5)* cos(th[i])

  V = Matrix([x_dot,0.0, z_dot, 0.0, 0.0, 0.0])

  J_inv = J.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:
theta_joint[4],theta7:theta_joint[5]}).inv()

  theta_dot = J_inv*V

  theta_joint = theta_joint + (theta_dot*(5/N))

  T = A.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:
theta_joint[4],theta7:theta_joint[5]})

  PP = P.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:
theta_joint[4],theta7:theta_joint[5]})

  Torque = PP - J.evalf(3,
subs={theta1:theta_joint[0],theta2:theta_joint[1],theta4:theta_joint[2],theta5:theta_joint[3],theta6:
theta_joint[4],theta7:theta_joint[5]}).T * Fw

  #plt.pause(5/N)
  T1.append(Torque[0])
  T2.append(Torque[1])
  T3.append(Torque[2])
  T4.append(Torque[3])
  T5.append(Torque[4])
  T6.append(Torque[5])
  cirx.append(T[0,3])
  ciry.append(T[2,3])


plt.subplot(4, 2, 1)
plt.title('Joint 1 Torque vs Time')
plt.plot(range(0,500),T1,'r')

plt.subplot(4, 2, 2)
plt.title('Joint 2 Torque vs Time')
plt.plot(range(0,500),T2,'g')

plt.subplot(4, 2, 3)
plt.title('Joint 4 Torque vs Time')
plt.plot(range(0,500),T3,'b')

plt.subplot(4, 2, 4)
plt.title('Joint 5 Torque vs Time')
plt.plot(range(0,500),T4,'y')
plt.subplot(4, 2, 5)
plt.title('Joint 6 Torque vs Time')
plt.plot(range(0,500),T5,'k')

plt.subplot(4, 2, 6)
plt.title('End effector Torque vs Time')
plt.plot(range(0,500),T6,'m')

plt.subplot(4, 2, 7)
plt.title('Drawn Circle')
plt.plot(cirx,ciry,'c')

plt.show()
```
18.