

# ROBOT MANIPULATOR DESIGN ASSIGNMENT

Souto T.L.



**MASSEY UNIVERSITY**  
**TE KUNENGA KI PŪREHUROA**

In this paper is reported the design of a robotic manipulator with four degrees of freedom on a fixed platform placed on a flat surface. It's able to be integrated with a robotic gripper also designed. The objective of this robotic system is to pick an object from a shelf or from the wall and place it onto a horizontal surface. Several tools were used to accomplish the objective of this project. For the calculations of forward and inverse kinematics Python programming language and the Pycharm IDE(Integrated Development Environment) were used, for modelling the robotic system SolidWorks, to simulate OpenModelica was chosen. The description of the design and how this tools were applied as well as why they were chosen is explained here. As part of this report, a documentation website and a electronic report were built as well as a GitHub repository with all the files.

***Keywords***—robotic systems, forward kinematics, inverse kinematics

This paper is a Individual design Project and It is part of the second assignment of the course Master of engineering - Mechatronics at Massey university, Auckland.

An electronic version of this report can be found by following this [link](#), and the programming documentation can be found by following this [link](#), also a GitHub repository with all the files used in this project is available [here](#).

Thiago Lima Souto is a student register under the number 19044686 at Massey university. Questions, comments or communications can be addressed via email [thiago.souto@yahoo.com.br](mailto:thiago.souto@yahoo.com.br)

Please Check the links.

## CONTENTS

<b>I</b>	<b>INTRODUCTION</b>	<b>3</b>
<b>II</b>	<b>Robot System Initial Proposal</b>	<b>3</b>
<b>III</b>	<b>Manipulator Design</b>	<b>3</b>
III-A	Robot Arms . . . . .	3
III-B	Robot Gripper . . . . .	3
III-C	Entire Model . . . . .	4
<b>IV</b>	<b>Other Components</b>	<b>4</b>
IV-A	Motors . . . . .	4
IV-A1	Inertia matching . . . . .	4
IV-A2	Torque requirements . . . . .	4
IV-A3	Power requirements . . . . .	4
IV-B	Sensors . . . . .	4
IV-C	Controllers . . . . .	4
<b>V</b>	<b>Forward Kinematic</b>	<b>4</b>
<b>VI</b>	<b>Inverse Kinematics</b>	<b>5</b>
<b>VII</b>	<b>System Simulation</b>	<b>6</b>
VII-A	Solidworks Simulation . . . . .	8
<b>VIII</b>	<b>Discussion and Conclusions</b>	<b>8</b>
	<b>References</b>	<b>10</b>
	<b>Appendix</b>	<b>29</b>

## I. INTRODUCTION

Industrial robot systems as well as computer-aided design and manufacturing (CAD and CAM) are leading the industrial automation. [1]

The mechanical manipulator is the most important form of the industrial robot and the localization of objects in the three-dimensional space is one of the most important aspect of the mechanical manipulator. Links, parts, tools, other objects on the manipulator environment and the motion of these objects are the subjects of study of Kinematics, as well as all the geometrical and time-based properties of the motion, with no regards to the forces applied that causes it.

The two basic problems in the study of mechanical manipulation are forward and inverse kinematics, the first computes the position and orientation of the end-effector on the manipulator and the second calculates all possible sets of joint angles that could be used a given position and orientation.

Nowadays, CAD and CAM advanced software's are of easy access and used to design, simulate and calculate all that is necessary for modern robot design.

The main objective of this assessment is to design a robotic manipulator with a fixed platform and flat surface, that is able to be integrated with a robot gripper for picking an object vertical wall/shelf and placing it onto a horizontal surface.

To accomplish this objective a robot system is proposed after this introduction followed by the manipulator and other components design. The forward and inverse kinematics of the robot system are studied with manual calculations as well as computed calculations. A Model with correct dimensions and a simulation of the proposed robot are made using Solidworks and OpenModelica. Finally, the results are discussed and the report is concluded.

## II. ROBOT SYSTEM INITIAL PROPOSAL

Aiming on the objective of designing a robot system, basically, capable of picking an object from a shelf and placing it onto a horizontal surface, the system proposed is a  $4dof$  (degrees of freedom, calculated using Grubler's formula) robot manipulator, consisting of a fixed base, a rotating base, two solid links and a toll, as shown in the Figure 1.

$$\begin{aligned} dof &= m(N - 1 - J) + \sum_{i=1}^J f_i \\ dof &= 6(5 - 1 - 4) + 4 \\ dof &= 4 \end{aligned} \quad (1)$$

Gluber's Formula

Where,

$N$  = number of bodies, counting ground as one body.

$J$  = Number of joints.

$m = 6$  for spatial bodies.

$f_i$  = degrees of freedom of each joint.

To control the joints four brushless AC motors are used. The motors have attached a magnetic absolute encoder and a integrated controller. At the end of the system there is a simple gripper making the robotic manipulator able to pickup an place the object.

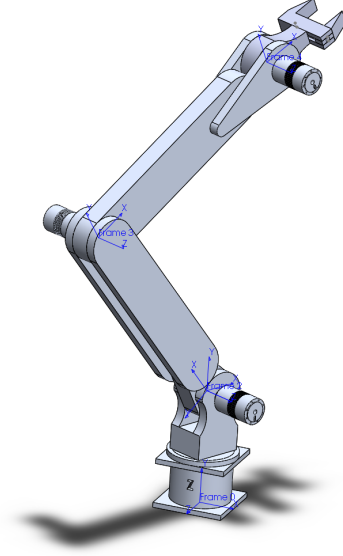


Fig. 1. Proposed robot manipulator system.

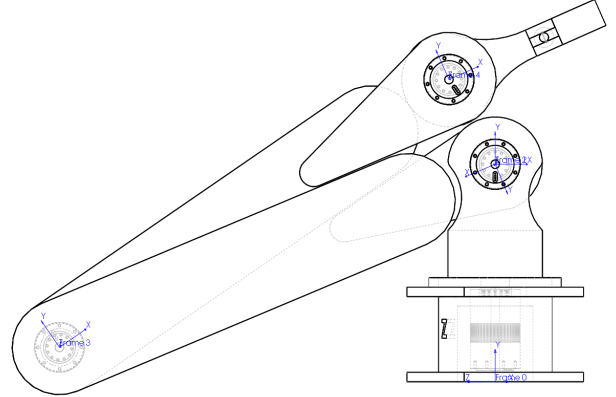


Fig. 2. Arms flexibility on Wireframe view.

## III. MANIPULATOR DESIGN

### A. Robot Arms

Link 1 and Link 2 constitute the "arms" of the manipulator, they are designed to maximize the joint angle reach for more flexibility. Also in the rotation base there are two cuts two maximize even further the arms reach as can be seen on Figure 2. In this picture, with the Wireframe view with hidden lines visible, the fourth motor located inside the base can be seen, It is hidden on Figure 13.

### B. Robot Gripper

A gripper capable to hold a 80 mm square object is connected to frame  $\{W\}$ . In one of the gripper's claws there is a micro-motor to enable the gripper to hold. One of the claws of the gripper is attached to a threaded cylinder and then attached to a micro-motor, which will rotate the threaded

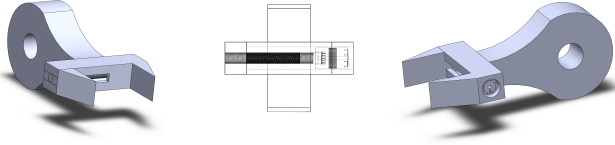


Fig. 3. Gripper details.

cylinder and make the claw move. Figure 3 shows the details of the gripper.

### C. Entire Model

Together with the Arms and the Gripper there is a fixed base and a rotational base. The Soliworks model shown on Figure 1 the electronics are discussed on the next topic.

## IV. OTHER COMPONENTS

The actuator chosen for the project is a complete system with sensors, controller and a motor. It can be used on several applications such as robots and drones, model aircrafts, conveyor belts, sorting and assembly lines, and packing equipment to name some.

### A. Motors

There are various types of motors and key factors need to be taken into account when selecting one for a particular application [3], in this case to control the joints of a robotic manipulator. The main factors are:

#### 1) Inertia matching

The robotic system have to be capable to achieve a required torque to give a load a moment of particular inertia and to achieve a desired angular acceleration. The moment of inertia was calculated using the Iterative Newton-Euler Dynamics Algorithm [7], and this is solved in two parts, first the links velocities and accelerations are iteratively computed across the links applying the Newton-Euler equations to each link, then the forces and torques of interaction and joint actuator torques are computed recursively from the last link to the first.

This calculation was made by simulating the system using a simulation software, OpenModelica. The values for  $\tau$  of each joint during the time of the simulation, 15s, are shown on Figure 4, and, as can be noted, the maximum torque required was 97.9063N.m. The simulation was made with the load attached to the system.

#### 2) Torque requirements

High torque means a mechanism is able to handle heavier loads. The motor used for the modelling is capable of 157N.m and should be able to handle the 97.9063N.m with a 59.0937N.m margin.

#### 3) Power requirements

As well as the torque requirements this project don't require that the motors run at maximum velocity, therefore overheating will not be a problem, and this is one of the main aspects of power requirements for a motor. The total power required is the sum of the power needed to overcome friction and that needed to accelerate the load [4].

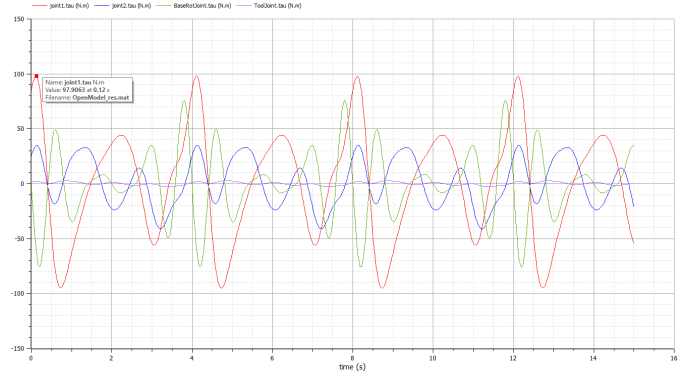


Fig. 4. Torque plot for various angles of the 4 joints, Max. 90.7566 N.m.

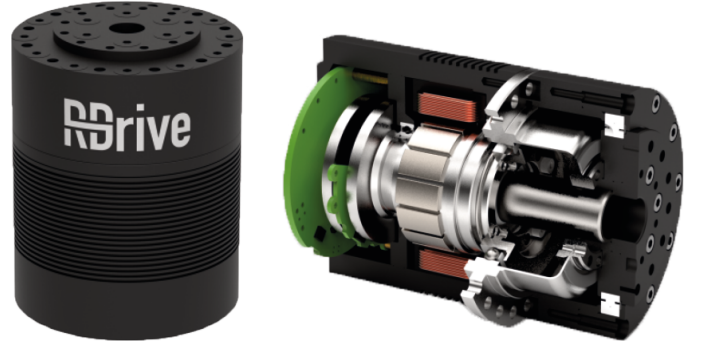


Fig. 5. RDrive motor.

After analysing and taking into consideration the aspects discussed above the RDrive 85 motor with rated torque of 108N.m and peak torque of 157N.m and 450W of Power was chosen to the task. [8] For the gripper a 20 mm diameter motor was used.

### B. Sensors

To know the angular position of the joints absolute encoders shall be the choice because they give the actual angular position, a unique identification of an angle. The incremental encoders would detect the changes but in relation to some Datum. [3]

So with the absolute encoders we can track  $\theta_1, \theta_2, \theta_3$  and  $\theta_4$  and rearrange the links accordingly with the joints angles.

Each motor comes with a 19-bit absolute encoder built-in.

### C. Controllers

Rozum robotics RDrive have an integrated controller supporting CANopen communication encoders. This motors come with a Python/java API and can be programmed from a control unit like an arduino, a PIC or a STM32 ARM microcontrollers.

## V. FORWARD KINEMATIC

To calculate the forward kinematics equation, a Python Class called "FowardKinematics" was created, this Class has two main methods involved on the calculations, the rotation and the translation for the  $\hat{Z}$  and  $\hat{X}$  axis. The parameters for these methods are extracted from the Denavit-Hartenberg

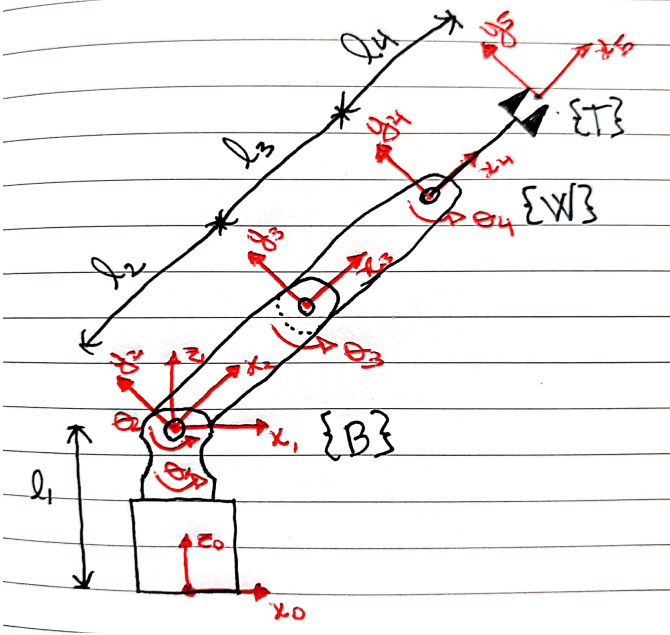


Fig. 6.  $\hat{Z}, \hat{Y}, \hat{X}$  axis,  $\theta_1, \theta_2, \theta_3, l_1, l_2, l_3$  and  $l_4$ .

parameters at (1), the coordinate systems and also the basic frames  $\{B\}$ ,  $\{W\}$  and  $\{T\}$ , Base, Wrist and Tools respectively are identified on the Figure 6. The size of the links are  $l_1 = 230$ ,  $l_2 = 500$ ,  $l_3 = 500$ ,  $l_4 = 180$ .

The forward kinematics calculations were confirmed by a python programming code that can be found in the Appendix A.

The rotation method receives an argument *self* and  $\theta_i$  for the rotation on the  $\hat{Z}$  axis and  $\alpha_{i-1}$  for  $\hat{X}$ . The *self* argument is what makes this a method and not just a plain function, this is filled in automatically, when we call this method on the object. So we'll just provide one argument, and the fact that it's being called on the method will provide the first argument, self. It will then build a *sympy* symbolic matrix and passes the *self* argument to the method to be put in place, if no arguments are passed default values will be put in place as specified in the key word arguments (\* *kwargs*) on the `__init__` function. A Matrix is then returned after calling the `.evalf()` function to evaluate.

Like in the rotation method the translation method receives a argument  $d_i$  and  $a_{i-1}$  to return a matrix that translates in  $\hat{Z}$  and  $\hat{X}$  axis respectively. This class is also detailed in the Appendix A.

The objective of the forward kinematics is to provide a kinematics equation relating the end-effector orientation and position. This is done by finding the Denavit–Hartenberg parameters and the homogeneous transforms for each step from 0 to 5. These matrix are shown from (2) to (09). Finally, the forward kinematics equation is presented on Equation 9.

## VI. INVERSE KINEMATICS

There are many methods to find the equations for the inverse kinematic here two methodologies are presented. First let's look at the trigonometric solution. For the trigonometric

$i$	$\alpha_{i-1}$	$a_{i-1}$	$d_i$	$\theta_i$
1	0	0	$l_1$	$\theta_1$
2	$90^\circ$	0	0	$\theta_2$
3	0	$l_2$	0	$\theta_3$
4	0	$l_3$	0	$\theta_4$
5	0	$l_4$	0	0

Denavit–Hartenberg parameters

$${}^0_1T = \begin{bmatrix} \cos\theta_1 & -\sin\theta_1 & 0 & 0 \\ \sin\theta_1 & \cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$${}^1_2T = \begin{bmatrix} \cos\theta_2 & -\sin\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ \sin\theta_2 & \cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

$${}^2_3T = \begin{bmatrix} \cos\theta_3 & -\sin\theta_3 & 0 & l_2 \\ \sin\theta_3 & \cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$${}^3_4T = \begin{bmatrix} \cos\theta_4 & -\sin\theta_4 & 0 & l_3 \\ \sin\theta_4 & \cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$${}^4_5T = \begin{bmatrix} 1 & 0 & 0 & l_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

$${}^0_5T = {}^0_1T {}^1_2T {}^2_3T {}^3_4T {}^4_5T = {}^0_1T {}^1_2T {}^2_3T {}^3_4T {}^4_5T = {}^0_5T \quad (8)$$

$${}^0_5T = \begin{bmatrix} C_1C_{234} & -S_{234}C_1 & S_1 & C_1(l_2C_2 + l_3C_{23} + l_4C_{234}) \\ S_1C_{234} & -S_1S_{234} & -C_1 & S_1(l_2C_2 + l_3C_{23} + l_4C_{234}) \\ S_{234} & C_{234} & 0 & l_1 + l_2S_2 + l_3S_{23} + l_4S_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9)$$

solution we can draw 2 triangles, one from frame  $\{1\}$  to  $\{4\}$  and the second one from frame  $\{3\}$  to frame  $\{5\}$ . This way we can study the relation between the angles.

If we assume  $\theta_1 = 0$  we can calculate the angle relation between the triangles on Figure 7 as if they were on the same plane.

To find the hypotenuses of the triangles we have to find  ${}^1_4T$ ,  ${}^3_5T$  and  ${}^1_5T$ . This can be achieved by multiplying both sides of Equation 8 by the dependent transpose inverse  $T^{-1}$ , as shown below.

If we consider Equation 10:

$${}^0_5T = {}^0_1T {}^1_2T {}^2_3T {}^3_4T {}^4_5T \quad (10)$$

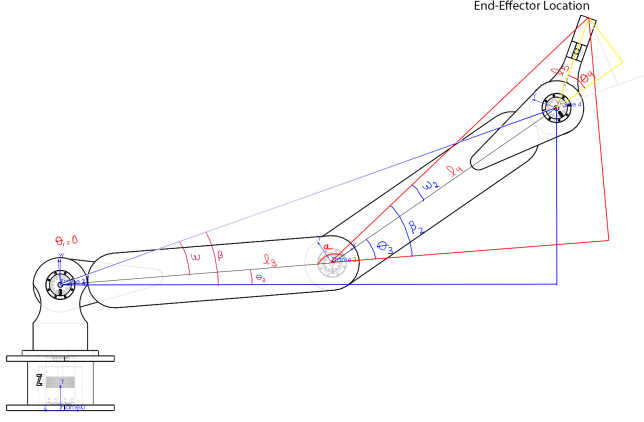


Fig. 7. Triangles from frame {1} to {4} and {3} to {5}.

and we multiply both sides by  ${}^0T^{-1}$  we have,

$${}^0T_1^0T^{-1} = {}^0T^{-1}{}_1T_2^1T_3^2T_4^3T_5^4T \quad (11)$$

Which is the same as,

$${}^1T = {}^1_2T_3^2T_4^3T_5^4T \quad (12)$$

To find  ${}^1_4T$  we can "go" to  ${}^1_5T$  and "coming back" one step by computing the inverse  ${}^4_5T^{-1}$ . Then we have,

$${}^1_4T = {}^1_5T_5^4T^{-1} \quad (13)$$

And for  ${}^3_5T$  from the red triangle we can multiply  ${}^1_2T^{-1}$  and  ${}^2_3T^{-1}$  by  ${}^1_5T$ . Then we have,

$${}^3_5T = {}^1_5T_2^1T_3^{-1}T^{-1} \quad (14)$$

Computing  ${}^1_4T$ , calculation made on Appendix A,

$${}^1_4T = \begin{bmatrix} C_{234} & -S_{234} & 0 & l_2C_2 + l_3C_{23} \\ 0 & 0 & -1 & 0 \\ S_{234} & C_{234} & 0 & l_2S_2 + l_3S_{23} \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad (15)$$

And  ${}^3_5T$ ,

$${}^3_5T =$$

$$\begin{bmatrix} C_3C_{34} & S_3C_{34} & -S_{34} & l_2C_2 - l_2C_3C_{34} + L_3C_{23} + L_4C_{234} \\ -S_3 & C_3 & 0 & l_2S_3 \\ S_{34}C_3 & S_3S_{34} & \cos 34 & l_2S_2 - l_2C_3S_{34} + L_3S_{23} + L_4S_{234} \\ 0 & 0 & -1 & 1 \end{bmatrix} \quad (16)$$

By using the law of cosines which states that, on a triangle  $A, B, C$ , we have,

$$\begin{aligned} c^2 &= \vec{c} \cdot \vec{c} = \\ (\vec{b} - \vec{a}) \cdot (\vec{b} - \vec{a}) &= \\ b^2 + a^2 - 2\vec{a} \cdot \vec{b} &= \\ b^2 + a^2 - 2ab\cos C. \end{aligned} \quad (17)$$

Applying to the blue triangle on Figure 7,

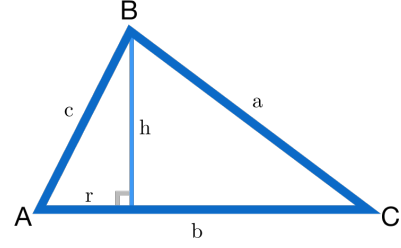


Fig. 8. Law of cosines.

$$\cos \omega = \frac{l_4^2 - l_3^2 - x^2 - y^2}{-2l_3\sqrt{x^2 + y^2}} \quad (18)$$

$$\sin \omega = \sqrt{1 - \cos^2 \omega}$$

Therefore,

$$\begin{aligned} \omega &= \text{Atan2}(\sin \omega, \cos \omega) \\ \theta_2 &= \beta - \omega \end{aligned} \quad (19)$$

Solving for the red triangle and considering  $(x, y)$  as from  ${}^3_5T$ , we have, a direct relation to  $\theta_3$  by using the relation with the angle  $a$  where,

$$\begin{aligned} a &= 180 - \theta_3 \\ \cos(a) &= \cos(180 - \theta_3) = \\ &= -\cos \theta_3 \end{aligned} \quad (20)$$

then we can have,

$$(\sqrt{x^2 + y^2})^2 = l_3^2 + l_4^2 + 2l_3l_4\cos \theta_3$$

$$\cos \theta_3 = \frac{x^2 + y^2 - l_3^2 - l_4^2}{2l_3l_4} \quad (21)$$

$$\sin \theta_3 = \sqrt{1 - \cos^2(\omega_2)}$$

$$\theta_3 = \text{Atan2}(\sin \omega_2, \cos \omega_2)$$

For  $\theta_4$  we cannot use the cosine law, instead we have to consider the yellow triangle with  $l_5$  as hypotenuses and  $x - l_4$  and we have,

$$\cos \theta_4 = \frac{x - l_4}{l_5}$$

$$\sin \theta_4 = \sqrt{1 - \cos^2 \theta_4} \quad (22)$$

$$\theta_4 = \text{Atan2}(\sin \theta_4, \cos \theta_4)$$

As a proof of the equations for the inverse kinematics a program in python was written using the a simple case where  $\theta_2 = 0$ ,  $\theta_3 = 45$  and  $\theta_4 = 45$ . The results were consistent and the program is exposed on Appendix A.

## VII. SYSTEM SIMULATION

OpenModelica is currently the most complete open-source Modelica and FMI based modelling, simulation, optimization, and model-based development environment. Its long-term development is supported by a non-profit organization – the Open Source Modelica Consortium (OSMC). [2]



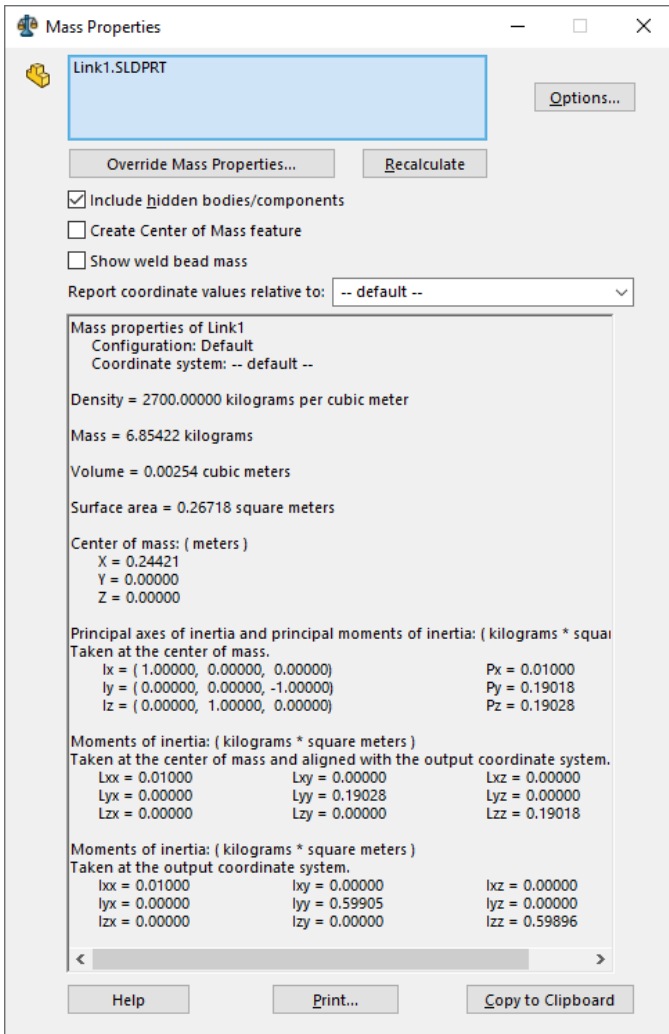


Fig. 9. Mass, center of mass and moments of inertia used on the simulation from link1 - SolidWorks.

This system was chosen, mainly, because of the open-source aspect, since Mathworks Simulink requires a paid plugin to connect the Solidworks model. Also due to the fact that Its a complete system for simulation modelling, versatile and capable of very complex tasks, much more complex than the current project.

For the modelling simulation parameters, information from the CAD simulator (SolidWorks) regarding to mass, center of mass and moments of inertia, were confronted with the Python simulation and was consistent as shown on Figure 9 and at the Python programming documentation that can be found on Appendix A. More information about the model is also provided but not included necessary to the link properties at OpenModelica, like density, volume, surface area, among others. The parameters necessary for the model were the length, mass and center of mass, as well as the inertia tensors, as shown on Figure 10.

To simulate the manipulator the component *Joints.Revolute* was used for the joints, and *Parts.BodyShapes* was used for the links, base and tool, and also some auxiliary component blocks to simulate

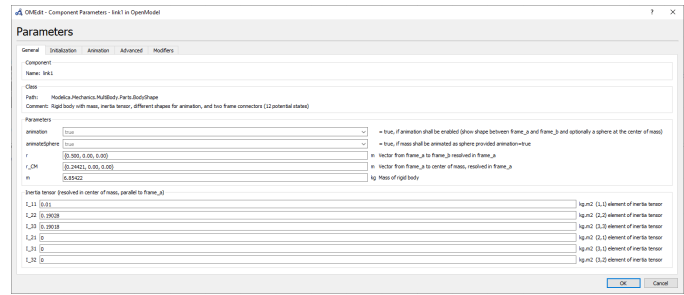


Fig. 10. OpenModelica, link1 parameters.

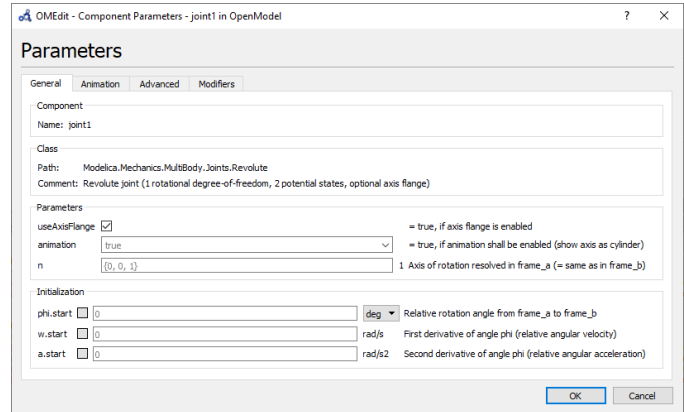


Fig. 11. OpenModelica, Joint1 parameters.

controllers, world conditions and a fixed base (ground). On the *Joints.Revolute* component for the joints the option *useAxisFlange*, allow the control of the rotation and this option was used as shown on the *joint1* parameters on Figure 11. A unit conversion block has to be used to convert from degrees to radians, there is a math block for that. With this control system the position can provide for *joint1* by setting a value to the *Gain* block. This will take the gain input and will provide it as a signal that the joint can use. This 6 blocks represent the first joint link of the system as represented in Figure 12.

The CAD modeling software chosen was SolidWorks from Dassault Systems, It is a solid modeling computer-aided design software with 2.3 million active users at over 234,800

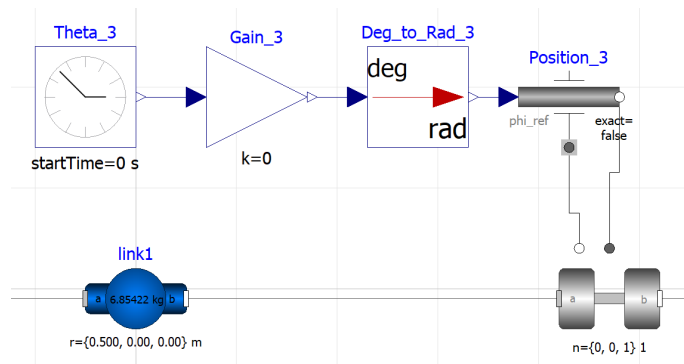


Fig. 12. Joint1, Link1 and controller.

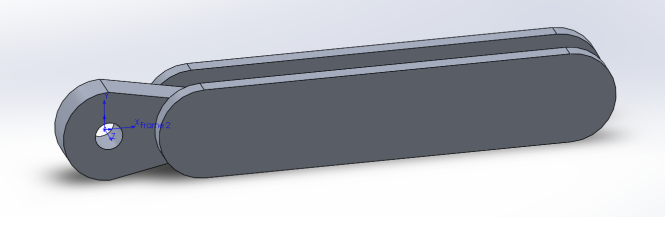


Fig. 13. Coordinate system on the frame {4} position for link 1.

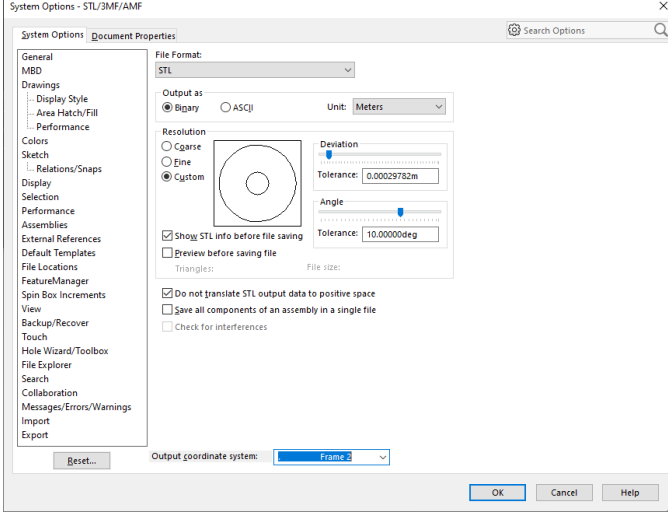


Fig. 14. Exporting parameters.

companies in 80 countries. [5] SolidWorks goal is building 3D CAD software that was easy-to-use, affordable, and available on Windows. [6] The main reasons to use SolidWorks in this project were the easy of use, the calculations that can be used as OpenModelica parameters and the exporting features that allows easy integration between the two software.

An import aspect of exporting from Solidworks to OpenModelica is the compatibility, the exported file can be a .STL file. Although the measurements have to be in meters. In the STL exporting window there is the option "Do not translate STL output data to positive space", this option makes exported parts maintain their original position in global space, relative to the origin [4].

There are many ways to export from one software to the other, on this project approach, at the export STL window, the coordinate system is been output, as can be seen in the import parameters for link 1 on Figure 14, by exporting the coordinate system placed on the frame position, Figure 13, the vector from frame {A} to the shape origin, resolved in {A} is equal to 0, because the frame coordinate system exported is located at the origin.

We can use the center of mass directly from SolidWorks as well as use the exact distance between the frames at the  $r$  parameter as shown in the Figure 10 and Figure 15.

The calculations for the moments of inertia were made in Python, refer to Appendix A, and confirmed in the mass evaluation at SolidWorks, also, in OpenModelica, as can be seen see on Figure 16 the values for the torque for joint 1 and

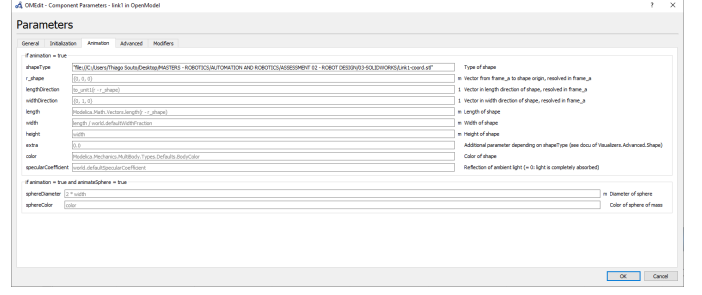


Fig. 15. Importing parameters.

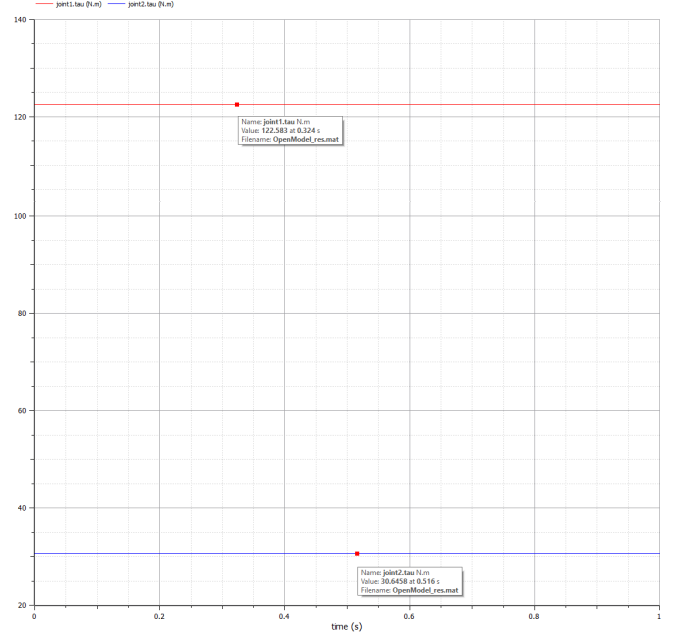


Fig. 16. Torques required for  $\theta_{1-4} = 0^\circ$ .

2 using  $\theta_{1-4} = 0^\circ$ .

At Figure 17 the simulation modelling at OpenModelica is shown, the parameters used at the *Gain* block results in an animation, Figure 18, that goes from all the angles been 0 to the values set.

#### A. Solidworks Simulation

A Solidworks simulation of the manipulator picking a box on a conveyor and placing it in another conveyor is shown on the electronic report, to see the report follow this [link](#).

### VIII. DISCUSSION AND CONCLUSIONS

Robot manipulators are huge contributor for the robot revolution and will continue to push the industry ahead. The study of it's components and the interactions between them are very important to predict if the designed manipulator will be able to accomplish the task for what it is purposed. New tools for simulation of these robots are improving every day and with such tools, in this project, we were able to demonstrate the design of a robotic manipulator capable of picking objects as well as its forward and inverse kinematics equations. Further developments will include an interface to



The figure displays four 3D renderings of a robotic arm, illustrating its range of motion. Each rendering shows the arm in a different pose, with its joints and end effector (a gripper) clearly visible. The arm is blue, and the joints are marked with red dots. A 3D coordinate system (x, y, z) is shown in the bottom left of each rendering, with the x-axis pointing right, the y-axis pointing up, and the z-axis pointing out of the page.

control the manipulator with real time calculations displayed with the aim of better understand the variables of such an interesting and important tool.

## REFERENCES

- [1] J. J. Craig, *Introduction To Robotics: Mechanics And Control*, 3rd ed., Ed. New York: Pearson Education, 2009.
- [2] P. Fritzson, et al. *The OpenModelica Integrated Modeling, Simulation and Optimization Environment* (Conference paper), PROCEEDINGS OF THE 1ST AMERICAN MODELICA CONFERENCE. Cambridge, USA: Massachusetts, 2018.
- [3] W. Bolton, *Mechatronics - Electronic control systems in mechanical and electrical engineering*. United Kingdom: Pearson Education Limited, 2019.
- [4] (SOLIDWORKS Online Help) Dassault Systems. (2020, May). Available: <https://help.solidworks.com/>.
- [5] (The SOLIDWORKS blog) Dassault Systems. (2020, May). Available: <https://bloHello Rhys, Thanks gs.solidworks.com/solidworksblog/2016/10/growing-solidworks-nation.html>.
- [6] (SolidWorks - Wikipedia)(2020, May). Available: <https://en.wikipedia.org/wiki/SolidWorks>.
- [7] K.N. Frazer, School of Food and Advanced Technology, Massey University, Slides from Lecture 12, March 2019.
- [8] *RDrive Datasheet* ROZUM Robotics LLC., Mountain House, CA - USA, 2020.
- [9] (Atan2 - Wikipedia)(2020, May). Available: <https://en.wikipedia.org/wiki/Atan2>

---

# **Robotic Manipulator Documentation**

***Release 1.0***

**Thiago Souto**

**May 14, 2020**

CONTENTS:

<b>1</b>	<b>Forward_Kinematics module</b>	<b>1</b>
1.1	Python Program . . . . .	2
1.2	Output . . . . .	5
<b>2</b>	<b>Inverse_Kinematics module</b>	<b>9</b>
2.1	Python Program . . . . .	9
2.2	Output . . . . .	12
<b>3</b>	<b>Indices and tables</b>	<b>16</b>
3.1	Running the documentation with Sphinx . . . . .	16
3.2	GitHub Repository . . . . .	16
	<b>Python Module Index</b>	<b>17</b>

## FORWARD\_KINEMATICS MODULE

**class** Forward\_Kinematics.**ForwardKinematics** (\*\*kwargs)

Bases: `object`

Definition: This class generates Homogeneous transform matrices, although it uses a symbolic approach that can be used to multiply any matrix and obtain the translation or rotation.

sympy.cos and sympy.sin: cos and sin for sympy

sympy.simplify: SymPy has dozens of functions to perform various kinds of simplification. simplify() attempts to apply all of these functions in an intelligent way to arrive at the simplest form of an expression.

Returns: It returns Rotation and translation matrices.

Obs: **\*\*kwargs** (keyword arguments) are used to facilitate the identification of the parameters, so initiate the object

**rot\_x** (*alpha*)

Definition: Receives an alpha angle and returns the rotation matrix for the given angle at the X axis.

**Parameters** **alpha** (*string*) – Rotation Angle around the X axis

Returns: The Rotational Matrix at the X axis by an *given* angle

**rot\_z** (*theta*)

Definition: Receives an theta angle and returns the rotation matrix for the given angle at the Z axis.

**Parameters** **theta** (*string*) – Rotation Angle around the Z axis

Returns: The Rotational Matrix at the Z axis by an *given* angle

**trans\_x** (*a*)

Definition: Translates the matrix a given amount *a* on the X axis by Defining a 4x4 identity matrix with *a* as the (1,4) element.

**Parameters** **a** (*string*) – Distance translated on the X-axis

Returns: The Translation Matrix on the X axis by a given distance

**trans\_z** (*d*)

Definition: Translate the matrix a given amount *d* on the Z axis. by Defining a matrix T 4x4 identity matrix with *d* (3,4) element position.

**Parameters** **d** (*string*) – Distance translated on the Z-axis

Returns: The Translation Matrix on the Z axis by a given distance

Forward\_Kinematics.**main**()

Assessment 02 Robotic manipulator design - Forward Kinematics.

Forward\_Kinematics.**printM**(*expr*, *num\_digits*)

## 1.1 Python Program

```

1 import numpy as np
2 import sympy as sympy
3 from sympy import *
4
5
6 class ForwardKinematics:
7
8     """
9     Definition: This class generates Homogeneous transform matrices, although it uses
10     ↪ a symbolic approach
11         that can be used to multiply any matrix and obtain the translation or rotation.
12
13     sympy.cos and sympy.sin: cos and sin for sympy
14
15     sympy.simplify: SymPy has dozens of functions to perform various kinds of
16     ↪ simplification.
17     simplify() attempts to apply all of these functions
18     in an intelligent way to arrive at the simplest form of an expression.
19
20     Returns: It returns Rotation and translation matrices.
21
22     Obs: **kwargs (keyword arguments) are used to facilitate the identification of
23     ↪ the parameters, so initiate the
24     object
25     """
26     np.set_printoptions(precision=3, suppress=True)
27
28     sympy.init_printing(use_unicode=True, num_columns=400)
29
30     def __init__(self, **kwargs):
31         """
32         Initializes the Object.
33         """
34         self._x_angle = kwargs['x_angle'] if 'x_angle' in kwargs else 'alpha_i-1'
35         self._x_dist = kwargs['x_dist'] if 'x_dist' in kwargs else 'a_i-1'
36         self._y_angle = kwargs['y_angle'] if 'y_angle' in kwargs else '0'
37         self._y_dist = kwargs['y_dist'] if 'y_dist' in kwargs else '0'
38         self._z_angle = kwargs['z_angle'] if 'z_angle' in kwargs else 'theta_i'
39         self._z_dist = kwargs['z_dist'] if 'z_dist' in kwargs else 'd_i'
40
41     def trans_x(self, a):
42         """
43         Definition: Translates the matrix a given amount 'a' on the *X* axis by
44         ↪ Defining a 4x4 identity
45         matrix with 'a' as the (1,4) element.
46
47         :type a: string
48         :param a: Distance translated on the X-axis
49
50         Returns: The Translation Matrix on the *X* axis by a given distance
51         """
52         self._x_dist = a
53
54         t_x = sympy.Matrix([[1, 0, 0, self._x_dist],
55                             [0, 1, 0, 0],

```

(continues on next page)



(continued from previous page)

```

52         [0, 0, 1, 0],
53         [0, 0, 0, 1]])
54
55     t_x = t_x.evalf()
56
57     return t_x
58
59     def trans_z(self, d):
60         """
61         Definition: Translate the matrix a given amount `d` on the *Z* axis. by_
        ↪Defining a matrix T 4x4 identity
62         matrix with *d* (3,4) element position.
63
64         :type d: string
65         :param d: Distance translated on the Z-axis
66
67         Returns: The Translation Matrix on the *Z* axis by a given distance
68         """
69         self._z_dist = d
70
71         t_z = sympy.Matrix([[1, 0, 0, 0],
72                             [0, 1, 0, 0],
73                             [0, 0, 1, self._z_dist],
74                             [0, 0, 0, 1]])
75
76         t_z = t_z.evalf()
77
78         return t_z
79
80     def rot_x(self, alpha):
81         """
82         Definition: Receives an alpha angle and returns the rotation matrix for the_
        ↪given angle at the *X* axis.
83
84         :type alpha: string
85         :param alpha: Rotation Angle around the X axis
86
87         Returns: The Rotational Matrix at the X axis by an *given* angle
88         """
89         self._x_angle = alpha
90
91         r_x = sympy.Matrix([[1, 0, 0, 0],
92                             [0, sympy.cos(self._x_angle), -sympy.sin(self._x_angle),
93 ↪0],
94                             [0, sympy.sin(self._x_angle), sympy.cos(self._x_angle),
95 ↪0],
96                             [0, 0, 0, 1]])
97
98         r_x = r_x.evalf()
99
100        return r_x
101
102    def rot_z(self, theta):
103        """
104        Definition: Receives an theta angle and returns the rotation matrix for the_
        ↪given angle at the *Z* axis.

```

(continues on next page)

(continued from previous page)

```

104     :type theta: string
105     :param theta: Rotation Angle around the Z axis
106
107     Returns: The Rotational Matrix at the Z axis by an *given* angle
108     """
109     self._z_angle = theta
110
111     r_z = sympy.Matrix([[sympy.cos(self._z_angle), -sympy.sin(self._z_angle), 0,
112 ↪0],
113
114                             [sympy.sin(self._z_angle), sympy.cos(self._z_angle), 0,
115 ↪0],
116
117                             [0, 0, 1, 0],
118                             [0, 0, 0, 1]])
119
120     r_z = r_z.evalf()
121
122     return r_z
123
124 # def printM(expr, num_digits):
125 #     return expr.xreplace({n.evalf(): n if type(n) == int else Float(n, num_digits)
126 ↪for n in expr.atoms(Number)})
127
128 def printM(expr, num_digits):
129     return expr.xreplace({n.evalf(): round(n, num_digits) for n in expr.atoms(Number)
130 ↪})
131
132 def main():
133     """
134     Assessment 02 Robotic manipulator design - Forward Kinematics.
135     """
136     a1 = ForwardKinematics()      # Rx(ai-1)
137     a2 = ForwardKinematics()      # Dx(ai-1)
138     a3 = ForwardKinematics()      # Dz(di)
139     a4 = ForwardKinematics()      # Rz(thetai)
140
141     print('Matrix t0_1:')
142     t_0_1 = (a1.rot_x('0')) * (a2.trans_x('0')) * (a3.trans_z('11')) * (a4.rot_z(
143 ↪'theta_1'))
144     print(sympy.pretty(t_0_1))
145
146     print('\nMatrix t1_2:')
147     t_1_2 = (a1.rot_x('alpha_1')) * (a2.trans_x('0')) * (a3.trans_z('0')) * (a4.rot_z(
148 ↪'theta_2'))
149     t_1_2_subs = t_1_2.subs('alpha_1', np.deg2rad(90.00))
150     print(sympy.pretty(printM(t_1_2_subs, 3)))
151
152     print('\nMatrix t2_3:')
153     t_2_3 = (a1.rot_x('0')) * (a2.trans_x('12')) * (a3.trans_z('0')) * (a4.rot_z(
154 ↪'theta_3'))
155     print(sympy.pretty(t_2_3))
156
157     print('\nMatrix t3_4:')
158     t_3_4 = (a1.rot_x('0')) * (a2.trans_x('13')) * (a3.trans_z('0')) * (a4.rot_z(
159 ↪'theta_4'))
160     print(sympy.pretty(t_3_4))

```

(continues on next page)

(continued from previous page)

```

153
154 print('\nMatrix t_4_5:')
155 t_4_5 = (a1.rot_x('0')) * (a2.trans_x('14')) * (a3.trans_z('0')) * (a4.rot_z('0'))
156 print(sympy.pretty(t_4_5))
157
158 t_0_5 = sympy.simplify(t_0_1 * t_1_2 * t_2_3 * t_3_4 * t_4_5)
159 print('\nMatrix T_0_5: with substitutions Round')
160 print(sympy.pretty(sympy.simplify(printM(t_0_5.subs('alpha_1', np.deg2rad(90.00)),
→ 3))))
161 t_0_5_subs = t_0_5.subs([('alpha_1', np.deg2rad(90.00)), ('l1', 230), ('l2', 500),
→ ('l3', 500), ('l4', 180)])
162 print('\nMatrix T_0_5: with substitutions Round')
163 print(sympy.pretty(sympy.simplify(printM(t_0_5_subs, 3))))
164
165 t_1_5 = sympy.simplify(t_1_2 * t_2_3 * t_3_4 * t_4_5)
166 print('\nMatrix T_1_5:')
167 print(sympy.pretty(printM(t_1_5.subs('alpha_1', np.deg2rad(90.00)), 3)))
168
169 print('\nMatrix T_1_5: for theta_1 = 0 ')
170 print(sympy.pretty(printM(t_1_5.subs([('alpha_1', np.deg2rad(90.00)), ('theta_1',
→ np.deg2rad(0.00))]), 3)))
171
172 # Calculations for the Inverse kinematics problem
173
174 t_1_4 = sympy.simplify(t_1_5 * t_4_5.inv())
175 print('\nMatrix T_1_4:')
176 print(sympy.pretty(printM(t_1_4.subs('alpha_1', np.deg2rad(90.00)), 3)))
177
178 t_3_5 = sympy.simplify(t_1_5 * t_1_2.inv() * t_2_3.inv())
179 print('\nMatrix t_3_5:')
180 print(sympy.pretty(printM(t_3_5.subs('alpha_1', np.deg2rad(90.00)), 3)))
181
182
183 if __name__ == '__main__':
184     main()

```

## 1.2 Output

```

Matrix t_0_1:

1.0cos(θ1)  -1.0sin(θ1)  0  0
1.0sin(θ1)  1.0cos(θ1)  0  0
0  0  1.0  1.0l1
0  0  0  1.0

Matrix t_1_2:

1.0cos(θ2)  -1.0sin(θ2)  0  0
0  0  -1.0  0
1.0sin(θ2)  1.0cos(θ2)  0.0  0

```

(continues on next page)

(continued from previous page)

```

0      0      0      1.0

Matrix t_2_3:
1.0cos(θ3)  -1.0sin(θ3)  0      1.0l2
1.0sin(θ3)  1.0cos(θ3)  0      0
0      0      1.0      0
0      0      0      1.0

Matrix t_3_4:
1.0cos(θ4)  -1.0sin(θ4)  0      1.0l3
1.0sin(θ4)  1.0cos(θ4)  0      0
0      0      1.0      0
0      0      0      1.0

Matrix t_4_5:
1.0      0      0      1.0l4
0      1.0      0      0
0      0      1.0      0
0      0      0      1.0

Matrix T_0_5: with substitutions Round
1.0cos(θ1)cos(θ2 + θ3 + θ4)  -1.0sin(θ2 + θ3 + θ4)cos(θ1)  1.0sin(θ1)  1.0(l2cos(θ2) +
↪ l3cos(θ2 + θ3) + l4cos(θ2 + θ3 + θ4))cos(θ1)
↪
1.0sin(θ1)cos(θ2 + θ3 + θ4)  -1.0sin(θ1)sin(θ2 + θ3 + θ4)  -1.0cos(θ1)  1.0(l2cos(θ2) +
↪ l3cos(θ2 + θ3) + l4cos(θ2 + θ3 + θ4))sin(θ1)
↪
1.0sin(θ2 + θ3 + θ4)  1.0cos(θ2 + θ3 + θ4)  0  1.0l1 + 1.
↪ 0l2sin(θ2) + 1.0l3sin(θ2 + θ3) + 1.0l4sin(θ2 + θ3 + θ4)
↪
↪      0      0      0
↪      1.0      0      0

Matrix T_0_5: with substitutions Round

```

(continues on next page)

(continued from previous page)

```

1.0cos(θ1)cos(θ2 + θ3 + θ4)  -1.0sin(θ2 + θ3 + θ4)cos(θ1)  1.0sin(θ1)  (500.0cos(θ2) +
↪ 500.0cos(θ2 + θ3) + 180.0cos(θ2 + θ3 + θ4))cos(θ1)

↪
1.0sin(θ1)cos(θ2 + θ3 + θ4)  -1.0sin(θ1)sin(θ2 + θ3 + θ4)  -1.0cos(θ1)  (500.0cos(θ2) +
↪ 500.0cos(θ2 + θ3) + 180.0cos(θ2 + θ3 + θ4))sin(θ1)

↪
1.0sin(θ2 + θ3 + θ4)  1.0cos(θ2 + θ3 + θ4)  0  500.0sin(θ2)
↪ + 500.0sin(θ2 + θ3) + 180.0sin(θ2 + θ3 + θ4) + 230.0

↪
0  0  0
↪ 1.0

```

Matrix T<sub>1\_5</sub>:

```

1.0cos(θ2 + θ3 + θ4)  -1.0sin(θ2 + θ3 + θ4)  0  1.0l2cos(θ2) + 1.0l3cos(θ2 + θ3) + 1.
↪ 0l4cos(θ2 + θ3 + θ4)

↪
0  0  -1.0  0
↪
1.0sin(θ2 + θ3 + θ4)  1.0cos(θ2 + θ3 + θ4)  0.0  1.0l2sin(θ2) + 1.0l3sin(θ2 + θ3) + 1.
↪ 0l4sin(θ2 + θ3 + θ4)

↪
0  0  0  1.
↪ 0

```

Matrix T<sub>1\_5</sub>: **for** theta<sub>1</sub> = 0

```

1.0cos(θ2 + θ3 + θ4)  -1.0sin(θ2 + θ3 + θ4)  0  1.0l2cos(θ2) + 1.0l3cos(θ2 + θ3) + 1.
↪ 0l4cos(θ2 + θ3 + θ4)

↪
0  0  -1.0  0
↪
1.0sin(θ2 + θ3 + θ4)  1.0cos(θ2 + θ3 + θ4)  0.0  1.0l2sin(θ2) + 1.0l3sin(θ2 + θ3) + 1.
↪ 0l4sin(θ2 + θ3 + θ4)

↪
0  0  0  1.
↪ 0

```

Matrix T<sub>1\_5</sub>: **for** theta<sub>1</sub> = 0

```

1.0cos(θ2 + θ3 + θ4)  -1.0sin(θ2 + θ3 + θ4)  0  1.0l2cos(θ2) + 1.0l3cos(θ2 + θ3) + 1.
↪ 0l4cos(θ2 + θ3 + θ4)

```

(continues on next page)

## 1.2. Output

(continued from previous page)

```

0 0 -1.0 0
→
→
1.0sin(θ2 + θ3 + θ4) 1.0cos(θ2 + θ3 + θ4) 0.0 1.0l2sin(θ2) + 1.0l3sin(θ2 + θ3) + 1.
→ 0l4sin(θ2 + θ3 + θ4)
→
0 0 0 1.
→ 0

Matrix T1_4:
1.0cos(θ2 + θ3 + θ4) -1.0sin(θ2 + θ3 + θ4) 0 1.0l2cos(θ2) + 1.0l3cos(θ2 + θ3)
→
0 0 -1.0 0
→
→
1.0sin(θ2 + θ3 + θ4) 1.0cos(θ2 + θ3 + θ4) 0.0 1.0l2sin(θ2) + 1.0l3sin(θ2 + θ3)
→
0 0 0 1.0
→

Matrix t3_5:
1.0cos(θ3)cos(θ3 + θ4) 1.0sin(θ3)cos(θ3 + θ4) -sin(θ3 + θ4) 1.0l2cos(θ2) -
→ l2cos(θ3)cos(θ3 + θ4) + 1.0l3cos(θ2 + θ3) + 1.0l4cos(θ2 + θ3 + θ4)
→
-sin(θ3) 1.0cos(θ3) 0.0
→ 1.0l2sin(θ3)
→
1.0sin(θ3 + θ4)cos(θ3) 1.0sin(θ3)sin(θ3 + θ4) 1.0cos(θ3 + θ4) 1.0l2sin(θ2) - l2sin(θ3 +
→ θ4)cos(θ3) + 1.0l3sin(θ2 + θ3) + 1.0l4sin(θ2 + θ3 + θ4)
→
0 0 0
→ 1.0

```



## INVERSE\_KINEMATICS MODULE

`Inverse_Kinematics.disp(expr)`  
Displays a simplified Sympy expression.

`Inverse_Kinematics.h_T(alpha, a, theta, d)`  
Returns a general homogeneous transform.

`Inverse_Kinematics.rot_x(alpha)`  
Returns a homogeneous transform for just a rotation about the X axis by alpha.

`Inverse_Kinematics.rot_z(theta)`  
Returns a homogeneous transform for just a rotation about the Z axis by theta.

`Inverse_Kinematics.trans_x(a)`  
Returns a homogeneous transform for just a translation along the X axis by a.

`Inverse_Kinematics.trans_z(d)`  
Returns a homogeneous transform for just a rotation along the Z axis by d.

### 2.1 Python Program

```
1  import numpy as np
2  import sympy
3
4  np.set_printoptions(precision=3, suppress=True)
5
6  sympy.init_printing(use_unicode=True, num_columns=400)
7
8
9  def disp(expr):
10     """Displays a simplified Sympy expression."""
11
12     e = sympy.simplify(expr)
13     e = sympy.expand(e)
14     e = e.evalf()
15
16     print(sympy.pretty(e))
17
18     return
19
20
21 def rot_x(alpha):
22     """Returns a homogeneous transform for just a rotation about the X axis by alpha."
23     ↪ """
```

(continues on next page)

(continued from previous page)

```

24     T = sympy.Matrix([[1, 0, 0, 0],
25                       [0, sympy.cos(alpha), -sympy.sin(alpha), 0],
26                       [0, sympy.sin(alpha), sympy.cos(alpha), 0],
27                       [0, 0, 0, 1]])
28
29     return T
30
31
32 def trans_x(a):
33     """Returns a homogeneous transform for just a translation along the X axis by a."""
34     ↪
35
36     T = sympy.Matrix([[1, 0, 0, a],
37                       [0, 1, 0, 0],
38                       [0, 0, 1, 0],
39                       [0, 0, 0, 1]])
40
41     return T
42
43 def rot_z(theta):
44     """Returns a homogeneous transform for just a rotation about the Z axis by theta."""
45     ↪
46
47     T = sympy.Matrix([[sympy.cos(theta), -sympy.sin(theta), 0, 0],
48                       [sympy.sin(theta), sympy.cos(theta), 0, 0],
49                       [0, 0, 1, 0],
50                       [0, 0, 0, 1]])
51
52     return T
53
54 def trans_z(d):
55     """Returns a homogeneous transform for just a rotation along the Z axis by d."""
56
57     T = sympy.Matrix([[1, 0, 0, 0],
58                       [0, 1, 0, 0],
59                       [0, 0, 1, d],
60                       [0, 0, 0, 1]])
61
62     return T
63
64
65 def h_T(alpha, a, theta, d):
66     """Returns a general homogeneous transform."""
67
68     T = rot_x(alpha) @ trans_x(a) @ rot_z(theta) @ trans_z(d)
69
70     return T
71
72
73 # Forward kinematics for the given problem.
74
75 theta_1 = np.deg2rad(0.0)
76 T01 = h_T(0, 0, theta_1, 0)
77 print("Matrix T01:")
78 disp(T01)

```

(continues on next page)

(continued from previous page)

```

79
80 alpha_1 = np.deg2rad(90)
81 theta_2 = np.deg2rad(0.0)
82 T12 = h_T(alpha_1, 0, theta_2, 0)
83 print("\nMatrix T12:")
84 disp(T12)
85
86 l1 = 500
87 theta_3 = np.deg2rad(45.0)
88 T23 = h_T(0, l1, theta_3, 0)
89 print("\nMatrix T23:")
90 disp(T23)
91
92 l2 = 500
93 theta_4 = np.deg2rad(45.0)
94 T34 = h_T(0, l2, theta_4, 0)
95 print("\nMatrix T34:")
96 disp(T34)
97
98 l3 = 230
99 theta_5 = np.deg2rad(0)
100 T45 = h_T(0, l3, theta_5, 0)
101 print("\nMatrix T45:")
102 disp(T45)
103
104 T05 = T01 @ T12 @ T23 @ T34 @ T45
105 print("\nMatrix T05:")
106 disp(T05)
107
108 print("\nProblem: location x, y, z")
109 x05 = float(T05[0, 3])
110 y05 = float(T05[1, 3])
111 z05 = float(T05[2, 3])
112
113 print("x05: {}".format(x05))
114 print("y05: {}".format(y05))
115 print("z05: {}".format(z05))
116
117 # Inverse kinematics for the given problem.
118
119 x = 853.553
120 y = 0.0
121 z = 583.553
122
123 theta_1 = np.arctan2(y, z)
124 T01 = h_T(0, 0, theta_1, 0)
125 print("\nMatrix T01 for given problem:")
126 disp(T01)
127
128 T45 = h_T(0, 230, 0, 0)
129 print("\nMatrix T45 for given problem:")
130 disp(T45)
131
132 T15 = T05 @ T01.inv()
133 print("\nMatrix T15 for given problem:")
134 disp(T15)
135

```

(continues on next page)

(continued from previous page)

```

136 T14 = T15 @ T45.inv()
137 print("\nMatrix T14 for given problem:")
138 disp(T14)
139
140 x14 = float(T14[0, 3])
141 y14 = float(T14[1, 3])
142 z14 = float(T14[2, 3])
143
144 print("x14: {}".format(x14))
145 print("y14: {}".format(y14))
146 print("z14: {}".format(z14))
147
148 B = np.arctan2(z14, x14)
149 c2 = (l2**2 - l1**2 - x14**2 - z14**2) / (-2*l1*np.sqrt(x14**2 + z14**2))
150 s2 = np.sqrt(1 - c2**2)
151 w = np.arctan2(s2, c2)
152 theta_2 = B - w
153
154 c3 = (x14**2 + z14**2 - l1**2 - l2**2) / (2*l1*l2)
155 s3 = np.sqrt(1 - c3**2)
156 theta_3 = np.arctan2(s3, c3)
157
158 T35 = T34 @ T45
159 disp(T35)
160
161 x35 = float(T35[0, 3])
162 y35 = float(T35[1, 3])
163 z35 = float(T35[2, 3])
164
165 print("x35: {}".format(x35))
166 print("y35: {}".format(y35))
167 print("z35: {}".format(z35))
168
169 c4 = (x35 - l2) / l3
170 s4 = np.sqrt(1 - c4**2)
171 theta_4 = np.arctan2(s4, c4)
172
173 print("\ntheta_2: {}".format(np.rad2deg(theta_2)))
174 print("\ntheta_3: {}".format(np.rad2deg(theta_3)))
175 print("\ntheta_4: {}".format(np.rad2deg(theta_4)))

```

## 2.2 Output

```

1  Matrix T01:
2  1.0    0    0    0
3
4    0    1.0    0    0
5
6    0    0    1.0    0
7
8    0    0    0    1.0
9
10 Matrix T12:
11 1.0          0          0          0
12

```

(continues on next page)

(continued from previous page)

```

13  0  6.12323399573677e-17      -1.0      0
14
15  0      1.0      6.12323399573677e-17  0
16
17  0      0      0      1.0
18
19  Matrix T23:
20  0.707106781186548  -0.707106781186547  0  500.0
21
22  0.707106781186547  0.707106781186548  0  0
23
24      0      0      1.0  0
25
26      0      0      0  1.0
27
28  Matrix T34:
29  0.707106781186548  -0.707106781186547  0  500.0
30
31  0.707106781186547  0.707106781186548  0  0
32
33      0      0      1.0  0
34
35      0      0      0  1.0
36
37  Matrix T45:
38  1.0  0  0  230.0
39
40  0  1.0  0  0
41
42  0  0  1.0  0
43
44  0  0  0  1.0
45
46  Matrix T05:
47  2.22044604925031e-16      -1.0      0      853.
↪ 553390593274
48
↪
49  6.12323399573677e-17  1.23259516440783e-32      -1.0      3.57323395960819e-
↪ 14
50
↪
51      1.0      2.22044604925031e-16  6.12323399573677e-17  583.
↪ 553390593274
52
↪
53      0      0      0      1.0
↪
54
55
56  Problem: location x, y, z
57
58  x05: 853.5533905932738
59  y05: 3.573233959608189e-14
60  z05: 583.5533905932737
61
62  Matrix T01 for given problem:

```

(continues on next page)

(continued from previous page)

```

63  1.0  0  0  0
64
65  0  1.0  0  0
66
67  0  0  1.0  0
68
69  0  0  0  1.0
70
71  Matrix T45 for given problem:
72  1.0  0  0  230.0
73
74  0  1.0  0  0
75
76  0  0  1.0  0
77
78  0  0  0  1.0
79
80  Matrix T15 for given problem:
81  2.22044604925031e-16  -1.0  0  853.
82  ↪553390593274
83
84  ↪
85  6.12323399573677e-17  1.23259516440783e-32  -1.0  3.57323395960819e-
86  ↪14
87
88  ↪
89  1.0  2.22044604925031e-16  6.12323399573677e-17  583.
90  ↪553390593274
91
92  ↪
93  0  0  0  1.0
94  ↪
95  Matrix T14 for given problem:
96  2.22044604925031e-16  -1.0  0  853.
97  ↪553390593274
98
99  ↪
100  6.12323399573677e-17  1.23259516440783e-32  -1.0  2.16489014058873e-
101  ↪14
102
103  ↪
104  1.0  2.22044604925031e-16  6.12323399573677e-17  353.
105  ↪553390593274
106
107  ↪
108  0  0  0  1.0
109  ↪
110  x14: 853.5533905932738
111  y14: 2.164890140588733e-14
112  z14: 353.55339059327366
113
114  0.707106781186548  -0.707106781186547  0  662.634559672906
115
116  0.707106781186547  0.707106781186548  0  162.634559672906

```

(continues on next page)



(continued from previous page)

106	0	0	1.0	0
107				
108	0	0	0	1.0
109				
110	x35: 662.6345596729059			
111	y35: 162.6345596729059			
112	z35: 0.0			
113				
114				
115	theta_2: -9.54166404439055e-15			
116	theta_3: 45.000000000000014			
117	theta_4: 45.000000000000014			

## INDICES AND TABLES

At the website you can navigate through the menus below:

- [genindex](#)
- [modindex](#)
- [search](#)

### 3.1 Running the documentation with Sphinx

To run the documentation for this project run the following commands, at the project folder:

Install Spinx:

**`python -m pip install sphinx`**

Install the “Read the Docs” theme:

**`pip install sphinx-rtd-theme`**

**`make clean`**

**`make html`**

### 3.2 GitHub Repository

Find all the files at the GitHub repository [here](#).

## PYTHON MODULE INDEX

### f

Forward\_Kinematics, [1](#)

### i

Inverse\_Kinematics, [9](#)