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

Please Check the links.

Contents

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.

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

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.

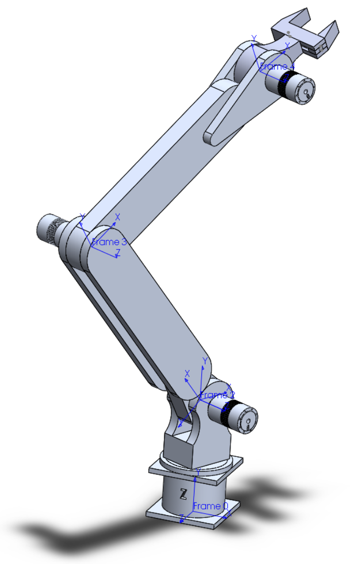


Fig. 1. Proposed robot manipulator system.

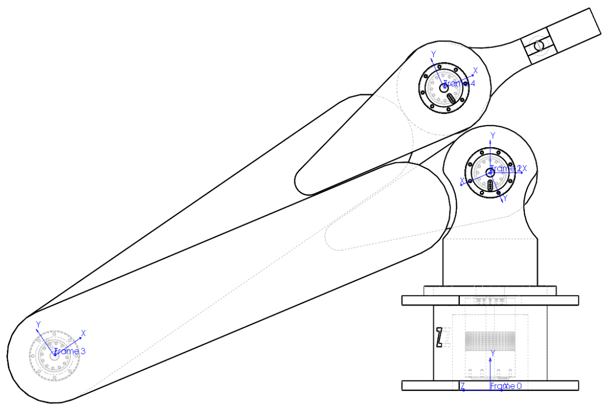


Fig. 2. Arms flexibility on Wireframe view.

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

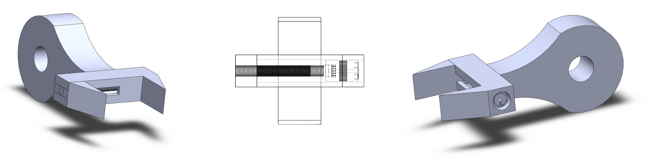


Fig. 3. Gripper details.

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 τ 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].

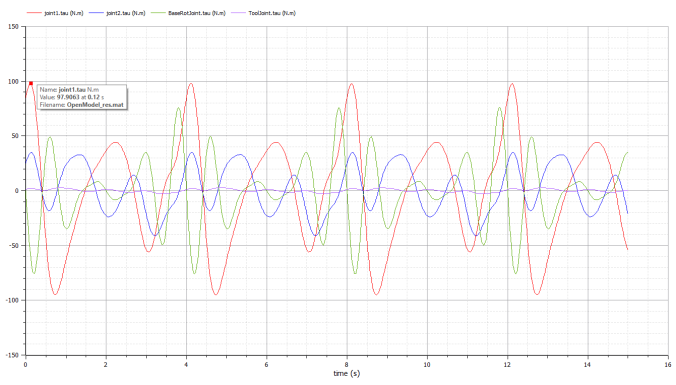


Fig. 4. Torque plot for various angles of the 4 joints, Max. 90.7566 N.m.

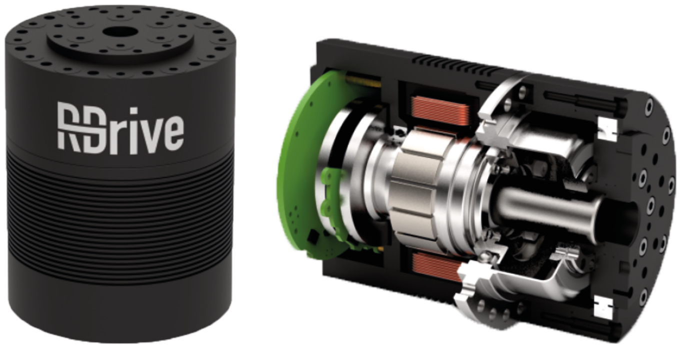


Fig. 5. RDrive motor.

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 θ_1, θ_2, θ_3 and θ_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

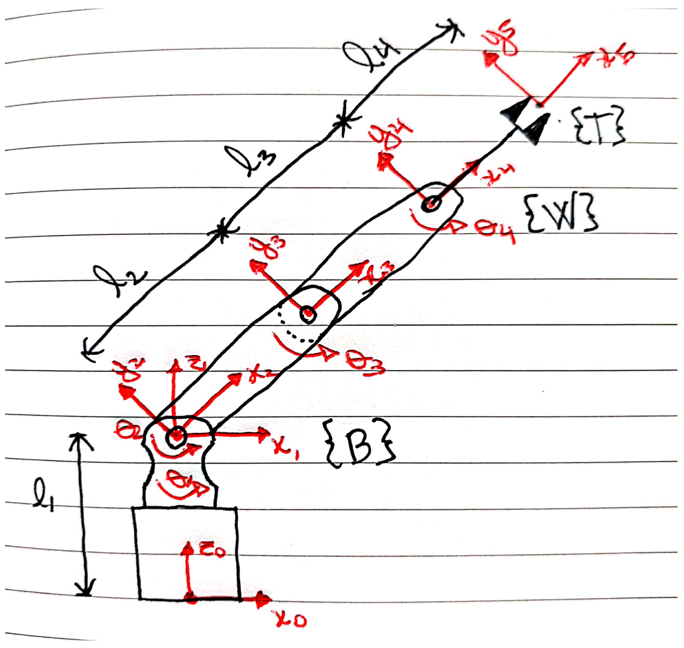


Fig. 6. $\hat{Z}, \hat{Y}, \hat{X} axis, \theta_1, \theta_2, \theta_3, l1, l2, l3$ and $l4$.

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 θ_i for the rotation on the $\hat{Z}$ axis and α_{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 solution we can draw 2 triangles, on 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:

$$^0_5T = ^0_1 T^1_2T^2_3T^3_4T^4_5T \tag{10}$$

i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	l_1	θ_1
2	90°	0	0	θ_2
3	0	l_2	0	θ_3
4	0	l_3	0	θ_4
5	0	l_4	0	0

(2)

Denavit–Hartenberg parameters

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

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

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

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

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

$$^0_5T = ^0_1 T^1_2T^2_3T^3_4T^4_5T => ^0 \cancel{_1T^1_2T^2_3T^3_4T^4}{}_5T = ^0_5 T \tag{8}$$

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

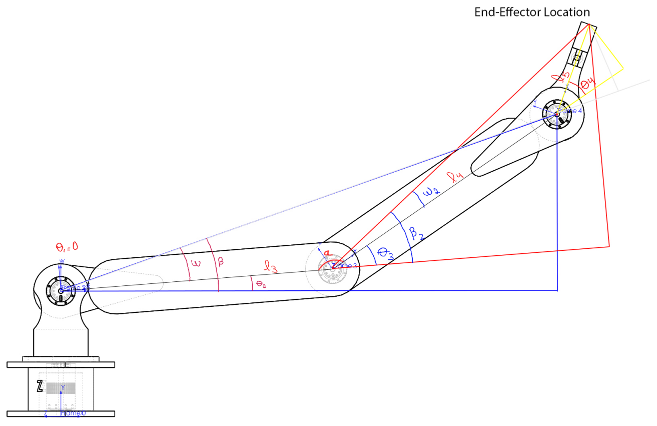


Fig. 7. Triangles from frame {1} to {4} and {3} to {5}.

and we multiply both sides by ${}^0_1T^{-1}$ we have,

$$ {}^0_5T\,{}^0_1T^{-1} = {}^0_1T^{-1}\,{}^0_1T\,{}^1_2T\,{}^2_3T\,{}^3_4T\,{}^4_5T \quad (11)$$

Which is the same as,

$$ {}^1_5T = {}^1_2T\,{}^2_3T\,{}^3_4T\,{}^4_5T \quad (12)$$

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,

$$ {}^1_4T = {}^1_5T\,{}^4_5T^{-1} \quad (13)$$

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

$$ {}^3_5T = {}^1_5T\,{}^1_2T^{-1}\,{}^2_3T^{-1} \quad (14)$$

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,

$$ \begin{gathered} c^2 = \vec{c}.\vec{c} = \\ (\vec{b} - \vec{a}).(\vec{b} - \vec{a}) = \\ b^2 + a^2 - 2\vec{a}.\vec{b} = \\ b^2 + a^2 - 2abcosC. \end{gathered} \quad (17)$$

Applying to the blue triangle on Figure 7,

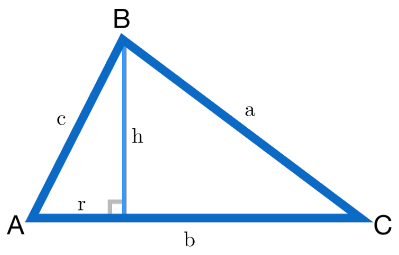


Fig. 8. Law of cosines.

$$ \begin{gathered} cos\omega = \frac{l_4^2 - l_3^2 - x^2 - y^2}{-2l_3\sqrt{x^2+y^2}} \\ sin\omega = \sqrt{1 - cos\omega^2} \end{gathered} \quad (18)$$

Therefore,

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

Solving for the red triangle and considering (x, y) as from ${}^3_5T$, we have, a direct relation to θ_3 by using the relation with the angle a where,

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

then we can have,

$$ \begin{gathered} (\sqrt{x^2 + y^2})^2 = {l_3}^2 + {l_4}^2 + 2l_3l_4cos\theta_3 \\ cos\theta_3 = \frac{x^2 + z^2 - {l_3}^2 - {l_4}^2}{2l_3l_4} \\ sin\theta_3 = \sqrt{1 - cos(\omega_2)^2} \\ \theta_3 = Atan2(sin\omega_2, cos\omega_2) \end{gathered} \quad (21)$$

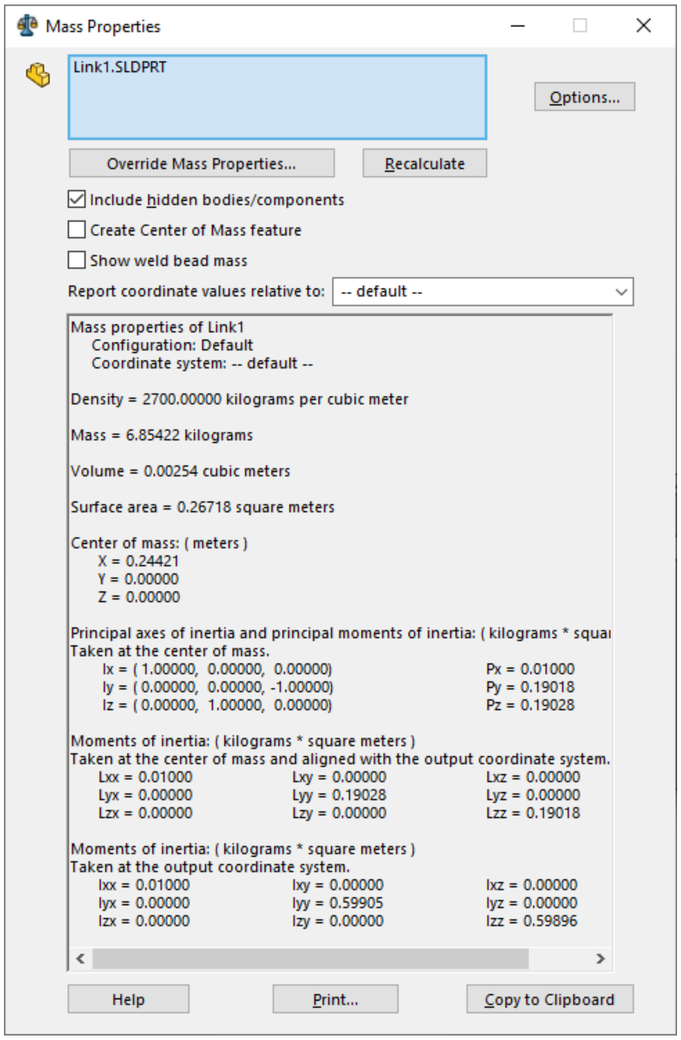
For θ_4 we cannot use the cosine law, instead we have to consider the yellow triangle with l_5 as hypotenuses and $x - l_4$ and we have,

$$ \begin{gathered} cos\theta_4 = \frac{x - l_4}{l_5} \\ sin\theta_4 = \sqrt{1 - {cos\theta_4}^2} \\ \theta_4 = Atan2(sin\theta_4, cos\theta_4) \end{gathered} \quad (22)$$

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]



Link1.SLDPRT

☑ Include hidden bodies/components
☐ Create Center of Mass feature
☐ Show weld bead mass

Report coordinate values relative to: -- default --

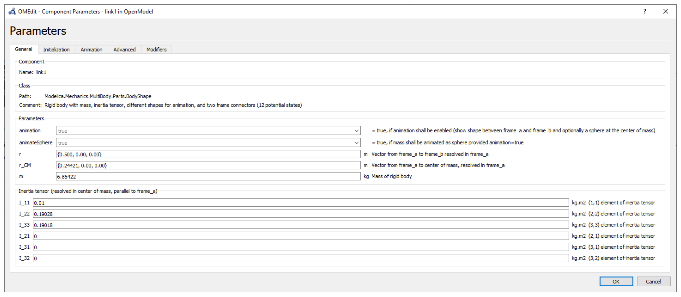
```
Mass properties of Link1
   Configuration: Default
   Coordinate system: -- default --

Density = 2700.00000 kilograms per cubic meter

Mass = 6.85422 kilograms

Volume = 0.00254 cubic meters

Surface area = 0.26718 square meters

Center of mass: ( meters )
    X = 0.24421
    Y = 0.00000
    Z = 0.00000

Principal axes of inertia and principal moments of inertia: ( kilograms * squa
Taken at the center of mass.
     Ix = ( 1.00000,  0.00000,  0.00000)          Px = 0.01000
     Iy = ( 0.00000,  0.00000, -1.00000)          Py = 0.19018
     Iz = ( 0.00000,  1.00000,  0.00000)          Pz = 0.19028

Moments of inertia: ( kilograms * square meters )
Taken at the center of mass and aligned with the output coordinate system.
    Lxx = 0.01000          Lxy = 0.00000          Lxz = 0.00000
    Lyx = 0.00000          Lyy = 0.19028          Lyz = 0.00000
    Lzx = 0.00000          Lzy = 0.00000          Lzz = 0.19018

Moments of inertia: ( kilograms * square meters )
Taken at the output coordinate system.
    Ixx = 0.01000          Ixy = 0.00000          Ixz = 0.00000
    Iyx = 0.00000          Iyy = 0.59905          Iyz = 0.00000
    Izx = 0.00000          Izy = 0.00000          Izz = 0.59896
```

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



Parameters

Component
Name: link1

Class
Path: Modelica.Mechanics.MultiBody.Parts.BodyShape
Comment: Rigid body with mass, inertia tensor, different shapes for animation, and two frame connectors (12 potential states)

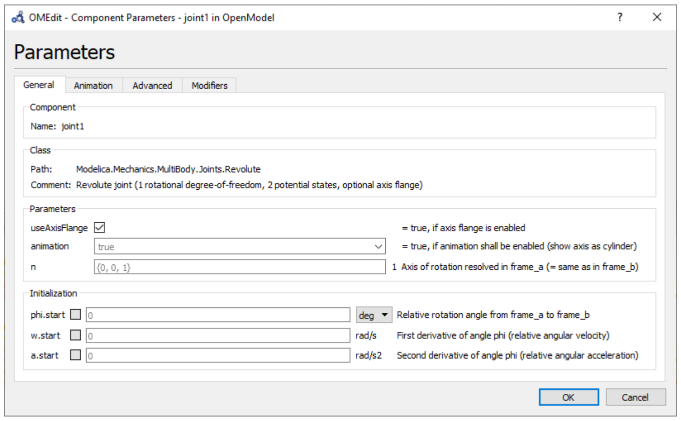
Parameters

animation	true		= true, if animation shall be enabled (show shape between frame_a and frame_b and optionally a sphere at the center of mass)
animateSphere	true		= true, if mass shall be animated as sphere provided animation=true
r	{0.500, 0.00, 0.00}	m	Vector from frame_a to frame_b resolved in frame_a
r_CM	{0.24421, 0.00, 0.00}	m	Vector from frame_a to center of mass, resolved in frame_a
m	6.85422	kg	Mass of rigid body

Inertia tensor (resolved in center of mass, parallel to frame_a)

I_11	0.01	kg.m2	(1,1) element of inertia tensor
I_22	0.19028	kg.m2	(2,2) element of inertia tensor
I_33	0.19018	kg.m2	(3,3) element of inertia tensor
I_21	0	kg.m2	(2,1) element of inertia tensor
I_31	0	kg.m2	(3,1) element of inertia tensor
I_32	0	kg.m2	(3,2) element of inertia tensor

Fig. 10. OpenModelica, link1 parameters.



Parameters

Component
Name: joint1

Class
Path: Modelica.Mechanics.MultiBody.Joints.Revolute
Comment: Revolute joint (1 rotational degree-of-freedom, 2 potential states, optional axis flange)

Parameters

useAxisFlange	☑		= true, if axis flange is enabled
animation	true		= true, if animation shall be enabled (show axis as cylinder)
n	{0, 0, 1}	1	Axis of rotation resolved in frame_a (= same as in frame_b)

Initialization

phi.start	☐	0	deg	Relative rotation angle from frame_a to frame_b
w.start	☐	0	rad/s	First derivative of angle phi (relative angular velocity)
a.start	☐	0	rad/s2	Second derivative of angle phi (relative angular acceleration)

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

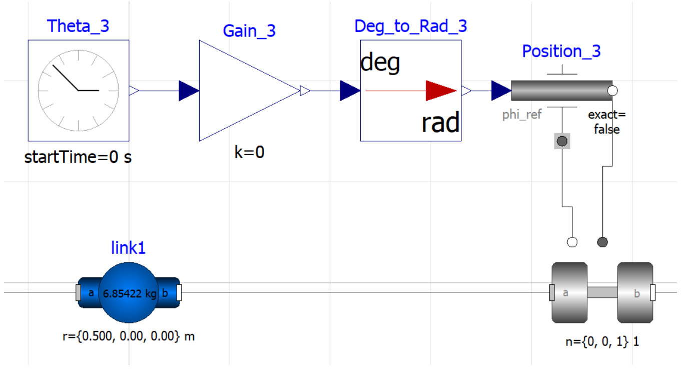


Fig. 12. Joint1, Link1 and controller.

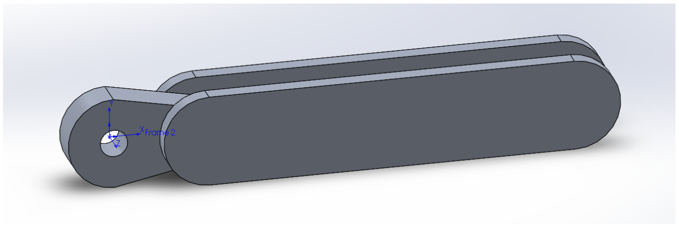
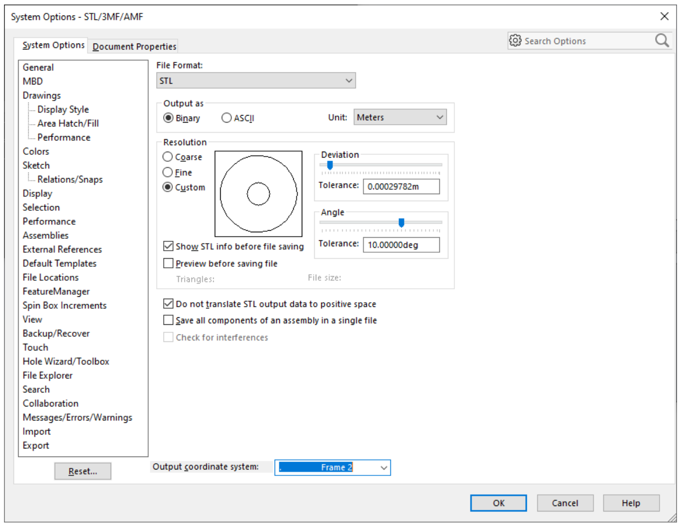


Fig. 13. Coordinate system on the frame $\{4\}$ position for link 1.



System Options - STL/3MF/AMF

System Options | Document Properties

General
MBD
Drawings
- Display Style
- Area Hatch/Fill
- Performance
Colors
Sketch
- Relations/Snaps
Display
Selection
Performance
Assemblies
External References
Default Templates
File Locations
FeatureManager
Spin Box Increments
View
Backup/Recover
Touch
Hole Wizard/Toolbox
File Explorer
Search
Collaboration
Messages/Errors/Warnings
Import
Export

File Format: STL

Output as: [✓ Binary] ASCII   Unit: Meters

Resolution: Coarse, Fine, [✓ Custom]

Deviation — Tolerance: 0.00029782m

Angle — Tolerance: 10.00000deg

[✓] Show STL info before file saving

[ ] Preview before saving file

Triangles: File size:

[✓] Do not translate STL output data to positive space

[ ] Save all components of an assembly in a single file

[ ] Check for interferences

Output coordinate system: Frame 2

Fig. 14. Exporting parameters.

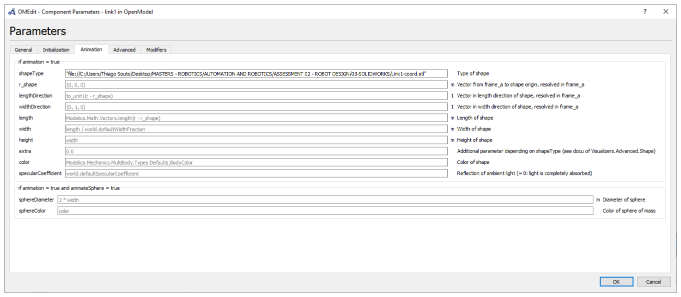
companies in 80 countries. [5] SolidWorks goal is building 3D CAD software that was easy-to-use, affordable, and available on Windows. [6] The main reasons to use SolidWorks in this project were the easy of use, the calculations that can be used as OpenModelica parameters and the exporting features that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



OMEdit - Component Parameters - link1 in OpenModel

Parameters

General | Initialization | Animation | Advanced | Modifiers

if animation = true

shapeType	"file://C:/Users/Thiago Souto/Desktop/MASTERS - ROBOTICS/AUTOMATION AND ROBOTICS/ASSESSMENT 02 - ROBOT DESIGN/03-SOLIDWORKS/Link1-coord.stl"		Type of shape
r_shape	{0, 0, 0}	m	Vector from frame_a to shape origin, resolved in frame_a
lengthDirection	to_unit1(r - r_shape)	1	Vector in length direction of shape, resolved in frame_a
widthDirection	{0, 1, 0}	1	Vector in width direction of shape, resolved in frame_a
length	Modelica.Math.Vectors.length(r - r_shape)	m	Length of shape
width	length / world.defaultWidthFraction	m	Width of shape
height	width	m	Height of shape
extra	0.0		Additional parameter depending on shapeType (see docu of Visualizers.Advanced.Shape)
color	Modelica.Mechanics.MultiBody.Types.Defaults.BodyColor		Color of shape
specularCoefficient	world.defaultSpecularCoefficient		Reflection of ambient light (= 0: light is completely absorbed)

if animation = true and animateSphere = true

sphereDiameter	2 * width	m	Diameter of sphere
sphereColor	color		Color of sphere of mass

Fig. 15. Importing parameters.

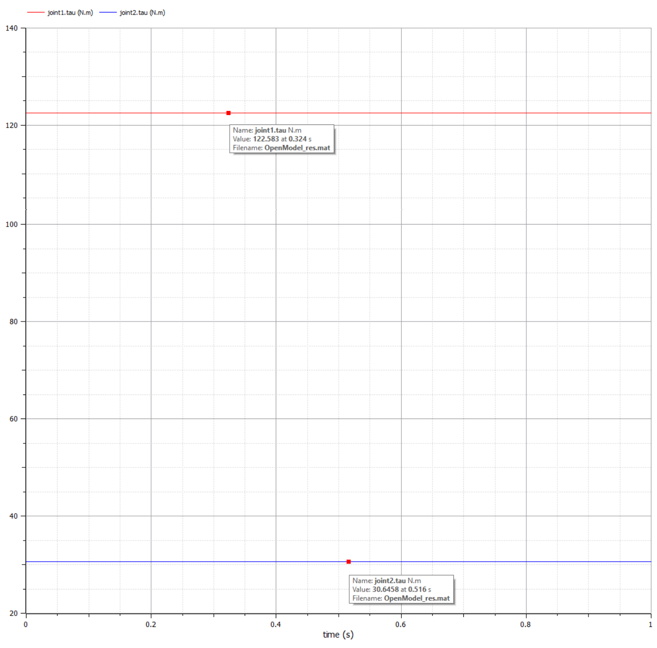


Fig. 16. Torques required for $\theta_{1-4} = 0°$.

2 using $\theta_{1-4} = 0°$.

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.

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

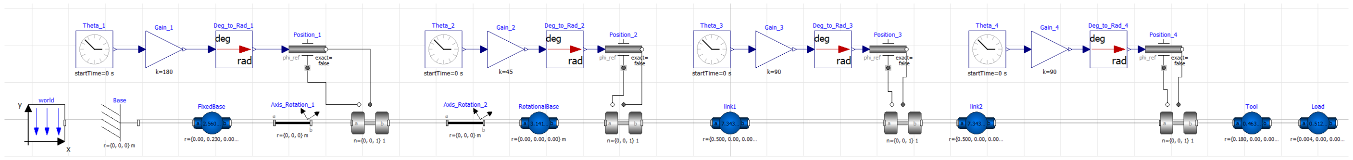


Fig. 17. Simulation model, Base, rotational link and 2 more links.

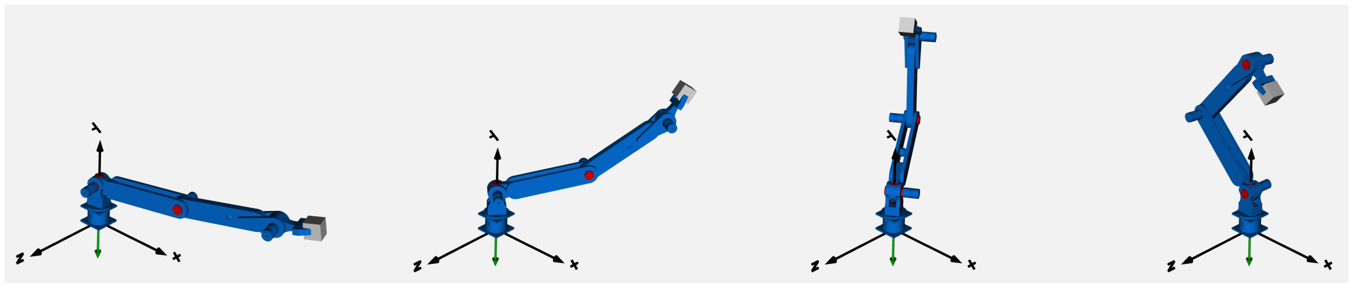


Fig. 18. Simulation animation.

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.

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 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: Massechusetts, 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] (SolidWoks - 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:

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

```
import numpy as np
import sympy as sympy
from sympy import *


class ForwardKinematics:

    """
    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
    """
    np.set_printoptions(precision=3, suppress=True)

    sympy.init_printing(use_unicode=True, num_columns=400)

    def __init__(self, **kwargs):
        """
        Initializes the Object.
        """
        self._x_angle = kwargs['x_angle'] if 'x_angle' in kwargs else 'alpha_i-1'
        self._x_dist = kwargs['x_dist'] if 'x_dist' in kwargs else 'a_i-1'
        self._y_angle = kwargs['y_angle'] if 'y_angle' in kwargs else '0'
        self._y_dist = kwargs['y_dist'] if 'y_dist' in kwargs else '0'
        self._z_angle = kwargs['z_angle'] if 'z_angle' in kwargs else 'theta_i'
        self._z_dist = kwargs['z_dist'] if 'z_dist' in kwargs else 'd_i'

    def trans_x(self, 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.

        :type a: string
        :param a: Distance translated on the X-axis

        Returns: The Translation Matrix on the *X* axis by a given distance
        """
        self._x_dist = a

        t_x = sympy.Matrix([[1, 0, 0, self._x_dist],
                            [0, 1, 0, 0],
```

(continues on next page)

(continued from previous page)

```
                            [0, 0, 1, 0],
                            [0, 0, 0, 1]])

        t_x = t_x.evalf()

        return t_x

    def trans_z(self, 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.

        :type d: string
        :param d: Distance translated on the Z-axis

        Returns: The Translation Matrix on the *Z* axis by a given distance
        """
        self._z_dist = d

        t_z = sympy.Matrix([[1, 0, 0, 0],
                            [0, 1, 0, 0],
                            [0, 0, 1, self._z_dist],
                            [0, 0, 0, 1]])

        t_z = t_z.evalf()

        return t_z

    def rot_x(self, alpha):
        """
        Definition: Receives an alpha angle and returns the rotation matrix for the␣
↪given angle at the *X* axis.

        :type alpha: string
        :param alpha: Rotation Angle around the X axis

        Returns: The Rotational Matrix at the X axis by an *given* angle
        """
        self._x_angle = alpha

        r_x = sympy.Matrix([[1, 0, 0, 0],
                            [0, sympy.cos(self._x_angle), -sympy.sin(self._x_angle),␣
↪0],
                            [0, sympy.sin(self._x_angle), sympy.cos(self._x_angle),␣
↪0],
                            [0, 0, 0, 1]])

        r_x = r_x.evalf()

        return r_x

    def rot_z(self, theta):
        """
        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)

```
        :type theta: string
        :param theta: Rotation Angle around the Z axis

        Returns: The Rotational Matrix at the Z axis by an *given* angle
        """
        self._z_angle = theta

        r_z = sympy.Matrix([[sympy.cos(self._z_angle), -sympy.sin(self._z_angle), 0,␣
↪0],
                            [sympy.sin(self._z_angle), sympy.cos(self._z_angle), 0,␣
↪0],
                            [0, 0, 1, 0],
                            [0, 0, 0, 1]])

        r_z = r_z.evalf()

        return r_z


# def printM(expr, num_digits):
#     return expr.xreplace({n.evalf(): n if type(n) == int else Float(n, num_digits)␣
↪for n in expr.atoms(Number)})

def printM(expr, num_digits):
    return expr.xreplace({n.evalf(): round(n, num_digits) for n in expr.atoms(Number)}
↪)


def main():
    """
    Assessment 02 Robotic manipulator design - Forward Kinematics.
    """
    a1 = ForwardKinematics()       # Rx(a_i-1)
    a2 = ForwardKinematics()       # Dx(a_i-1)
    a3 = ForwardKinematics()       # Dz(d_i)
    a4 = ForwardKinematics()       # Rz(theta_i)

    print('Matrix t_0_1:')
    t_0_1 = (a1.rot_x('0')) * (a2.trans_x('0')) * (a3.trans_z('l1')) * (a4.rot_z(
↪'theta_1'))
    print(sympy.pretty(t_0_1))

    print('\nMatrix t_1_2:')
    t_1_2 = (a1.rot_x('alpha_1')) * (a2.trans_x('0')) * (a3.trans_z('0')) * (a4.rot_z(
↪'theta_2'))
    t_1_2_subs = t_1_2.subs('alpha_1', np.deg2rad(90.00))
    print(sympy.pretty(printM(t_1_2_subs, 3)))

    print('\nMatrix t_2_3:')
    t_2_3 = (a1.rot_x('0')) * (a2.trans_x('l2')) * (a3.trans_z('0')) * (a4.rot_z(
↪'theta_3'))
    print(sympy.pretty(t_2_3))

    print('\nMatrix t_3_4:')
    t_3_4 = (a1.rot_x('0')) * (a2.trans_x('l3')) * (a3.trans_z('0')) * (a4.rot_z(
↪'theta_4'))
    print(sympy.pretty(t_3_4))
```

(continues on next page)

(continued from previous page)

```

    print('\nMatrix t_4_5:')
    t_4_5 = (a1.rot_x('0')) * (a2.trans_x('l4')) * (a3.trans_z('0')) * (a4.rot_z('0'))
    print(sympy.pretty(t_4_5))

    t_0_5 = sympy.simplify(t_0_1 * t_1_2 * t_2_3 * t_3_4 * t_4_5)
    print('\nMatrix T_0_5: with substitutions Round')
    print(sympy.pretty(sympy.simplify(printM(t_0_5.subs('alpha_1', np.deg2rad(90.00)),
↪ 3))))
    t_0_5_subs = t_0_5.subs([('alpha_1', np.deg2rad(90.00)), ('l1', 230), ('l2', 500),
↪ ('l3', 500), ('l4', 180)])
    print('\nMatrix T_0_5: with substitutions Round')
    print(sympy.pretty(sympy.simplify(printM(t_0_5_subs, 3))))

    t_1_5 = sympy.simplify(t_1_2 * t_2_3 * t_3_4 * t_4_5)
    print('\nMatrix T_1_5:')
    print(sympy.pretty(printM(t_1_5.subs('alpha_1', np.deg2rad(90.00)), 3)))

    print('\nMatrix T_1_5: for theta_1 = 0 ')
    print(sympy.pretty(printM(t_1_5.subs([('alpha_1', np.deg2rad(90.00)), ('theta_1',␣
↪np.deg2rad(0.00))]), 3)))

    # Calculations for the Inverse kinematics problem

    t_1_4 = sympy.simplify(t_1_5 * t_4_5.inv())
    print('\nMatrix T_1_4:')
    print(sympy.pretty(printM(t_1_4.subs('alpha_1', np.deg2rad(90.00)), 3)))

    t_3_5 = sympy.simplify(t_1_5 * t_1_2.inv() * t_2_3.inv())
    print('\nMatrix t_3_5:')
    print(sympy.pretty(printM(t_3_5.subs('alpha_1', np.deg2rad(90.00)), 3)))


if __name__ == '__main__':
    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(θ_3)  -1.0sin(θ_3)   0   1.0l_2

1.0sin(θ_3)  1.0cos(θ_3)    0     0

     0            0        1.0    0

     0            0         0    1.0


Matrix t_3_4:

1.0cos(θ_4)  -1.0sin(θ_4)   0   1.0l_3

1.0sin(θ_4)  1.0cos(θ_4)    0     0

     0            0        1.0    0

     0            0         0    1.0


Matrix t_4_5:

1.0   0    0   1.0l_4

 0   1.0   0     0

 0    0   1.0    0

 0    0    0    1.0


Matrix T_0_5: with substitutions Round

1.0cos(θ_1)cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_2 + θ_3 + θ_4)cos(θ_1)  1.0sin(θ_1)      1.0(l_2cos(θ_2) +␣
↪l_3cos(θ_2 + θ_3) + l_4cos(θ_2 + θ_3 + θ_4))cos(θ_1)
                                                                                                  ␣
↪
1.0sin(θ_1)cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_1)sin(θ_2 + θ_3 + θ_4)  -1.0cos(θ_1)     1.0(l_2cos(θ_2) +␣
↪l_3cos(θ_2 + θ_3) + l_4cos(θ_2 + θ_3 + θ_4))sin(θ_1)
                                                                                                  ␣
↪
    1.0sin(θ_2 + θ_3 + θ_4)          1.0cos(θ_2 + θ_3 + θ_4)            0         1.0l_1 + 1.
↪0l_2sin(θ_2) + 1.0l_3sin(θ_2 + θ_3) + 1.0l_4sin(θ_2 + θ_3 + θ_4)
                                                                                                  ␣
↪
              0                              0                      0                             ␣
↪                         1.0


Matrix T_0_5: with substitutions Round
```

(continues on next page)

(continued from previous page)

```
1.0cos(θ_1)cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_2 + θ_3 + θ_4)cos(θ_1)  1.0sin(θ_1)   (500.0cos(θ_2) +␣
↪500.0cos(θ_2 + θ_3) + 180.0cos(θ_2 + θ_3 + θ_4))cos(θ_1)
                                                                                                  ␣
↪
1.0sin(θ_1)cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_1)sin(θ_2 + θ_3 + θ_4)  -1.0cos(θ_1)  (500.0cos(θ_2) +␣
↪500.0cos(θ_2 + θ_3) + 180.0cos(θ_2 + θ_3 + θ_4))sin(θ_1)
                                                                                                  ␣
↪
    1.0sin(θ_2 + θ_3 + θ_4)            1.0cos(θ_2 + θ_3 + θ_4)            0          500.0sin(θ_2)␣
↪+ 500.0sin(θ_2 + θ_3) + 180.0sin(θ_2 + θ_3 + θ_4) + 230.0
                                                                                                  ␣
↪
              0                               0                       0                           ␣
↪                       1.0


Matrix T_1_5:

1.0cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_2 + θ_3 + θ_4)    0    1.0l_2cos(θ_2) + 1.0l_3cos(θ_2 + θ_3) + 1.
↪0l_4cos(θ_2 + θ_3 + θ_4)
                                                                                                  ␣
↪
          0                       0              -1.0                                            0␣
↪
                                                                                                  ␣
↪
1.0sin(θ_2 + θ_3 + θ_4)  1.0cos(θ_2 + θ_3 + θ_4)    0.0   1.0l_2sin(θ_2) + 1.0l_3sin(θ_2 + θ_3) + 1.
↪0l_4sin(θ_2 + θ_3 + θ_4)
                                                                                                  ␣
↪
          0                       0              0                                               1.
↪0


Matrix T_1_5: for theta_1 = 0

1.0cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_2 + θ_3 + θ_4)    0    1.0l_2cos(θ_2) + 1.0l_3cos(θ_2 + θ_3) + 1.
↪0l_4cos(θ_2 + θ_3 + θ_4)
                                                                                                  ␣
↪
          0                       0              -1.0                                            0␣
↪
                                                                                                  ␣
↪
1.0sin(θ_2 + θ_3 + θ_4)  1.0cos(θ_2 + θ_3 + θ_4)    0.0   1.0l_2sin(θ_2) + 1.0l_3sin(θ_2 + θ_3) + 1.
↪0l_4sin(θ_2 + θ_3 + θ_4)
                                                                                                  ␣
↪
          0                       0              0                                               1.
↪0


Matrix T_1_5: for theta_1 = 0

1.0cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_2 + θ_3 + θ_4)    0    1.0l_2cos(θ_2) + 1.0l_3cos(θ_2 + θ_3) + 1.
↪0l_4cos(θ_2 + θ_3 + θ_4)
                                                                                                  ␣
↪
```

(continues on next page)

(continued from previous page)

```
          0                      0             -1.0                                0␣
↪
                                                                                    ␣
↪
1.0sin(θ_2 + θ_3 + θ_4)  1.0cos(θ_2 + θ_3 + θ_4)    0.0   1.0l_2sin(θ_2) + 1.0l_3sin(θ_2 + θ_3) + 1.
↪0l_4sin(θ_2 + θ_3 + θ_4)
                                                                                    ␣
↪
          0                      0              0                                 1.
↪0


Matrix T_1_4:

1.0cos(θ_2 + θ_3 + θ_4)  -1.0sin(θ_2 + θ_3 + θ_4)    0    1.0l_2cos(θ_2) + 1.0l_3cos(θ_2 + θ_3)
                                                                                    ␣
↪
          0                      0             -1.0                  0              ␣
↪
                                                                                    ␣
↪
1.0sin(θ_2 + θ_3 + θ_4)  1.0cos(θ_2 + θ_3 + θ_4)    0.0   1.0l_2sin(θ_2) + 1.0l_3sin(θ_2 + θ_3)
                                                                                    ␣
↪
          0                      0              0                   1.0             ␣
↪


Matrix t_3_5:

1.0cos(θ_3)cos(θ_3 + θ_4)  1.0sin(θ_3)cos(θ_3 + θ_4)   -sin(θ_3 + θ_4)    1.0l_2cos(θ_2) -␣
↪l_2cos(θ_3)cos(θ_3 + θ_4) + 1.0l_3cos(θ_2 + θ_3) + 1.0l_4cos(θ_2 + θ_3 + θ_4)
                                                                                    ␣
↪
        -sin(θ_3)                1.0cos(θ_3)                0.0                     ␣
↪                    1.0l_2sin(θ_3)
                                                                                    ␣
↪
1.0sin(θ_3 + θ_4)cos(θ_3)  1.0sin(θ_3)sin(θ_3 + θ_4)  1.0cos(θ_3 + θ_4)  1.0l_2sin(θ_2) - l_2sin(θ_3 +␣
↪θ_4)cos(θ_3) + 1.0l_3sin(θ_2 + θ_3) + 1.0l_4sin(θ_2 + θ_3 + θ_4)
                                                                                    ␣
↪
           0                        0                    0                          ␣
↪                          1.0
```

CHAPTER

TWO

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

```
import numpy as np
import sympy

np.set_printoptions(precision=3, suppress=True)

sympy.init_printing(use_unicode=True, num_columns=400)


def disp(expr):
    """Displays a simplified Sympy expression."""

    e = sympy.simplify(expr)
    e = sympy.expand(e)
    e = e.evalf()

    print(sympy.pretty(e))

    return


def rot_x(alpha):
    """Returns a homogeneous transform for just a rotation about the X axis by alpha."
↪""

```

(continues on next page)

(continued from previous page)

```
    T = sympy.Matrix([[1, 0, 0, 0],
                      [0, sympy.cos(alpha), -sympy.sin(alpha), 0],
                      [0, sympy.sin(alpha), sympy.cos(alpha), 0],
                      [0, 0, 0, 1]])

    return T


def trans_x(a):
    """Returns a homogeneous transform for just a translation along the X axis by a.""
↪"

    T = sympy.Matrix([[1, 0, 0, a],
                      [0, 1, 0, 0],
                      [0, 0, 1, 0],
                      [0, 0, 0, 1]])

    return T


def rot_z(theta):
    """Returns a homogeneous transform for just a rotation about the Z axis by theta."
↪""

    T = sympy.Matrix([[sympy.cos(theta), -sympy.sin(theta), 0, 0],
                      [sympy.sin(theta), sympy.cos(theta), 0, 0],
                      [0, 0, 1, 0],
                      [0, 0, 0, 1]])

    return T


def trans_z(d):
    """Returns a homogeneous transform for just a rotation along the Z axis by d."""

    T = sympy.Matrix([[1, 0, 0, 0],
                      [0, 1, 0, 0],
                      [0, 0, 1, d],
                      [0, 0, 0, 1]])

    return T


def h_T(alpha, a, theta, d):
    """Returns a general homogeneous transform."""

    T = rot_x(alpha) @ trans_x(a) @ rot_z(theta) @ trans_z(d)

    return T


# Forward kinematics for the given problem.

theta_1 = np.deg2rad(0.0)
T01 = h_T(0, 0, theta_1, 0)
print("Matrix T01:")
disp(T01)
```

(continues on next page)

(continued from previous page)

```python

alpha_1 = np.deg2rad(90)
theta_2 = np.deg2rad(0.0)
T12 = h_T(alpha_1, 0, theta_2, 0)
print("\nMatrix T12:")
disp(T12)

l1 = 500
theta_3 = np.deg2rad(45.0)
T23 = h_T(0, l1, theta_3, 0)
print("\nMatrix T23:")
disp(T23)

l2 = 500
theta_4 = np.deg2rad(45.0)
T34 = h_T(0, l2, theta_4, 0)
print("\nMatrix T34:")
disp(T34)

l3 = 230
theta_5 = np.deg2rad(0)
T45 = h_T(0, l3, theta_5, 0)
print("\nMatrix T45:")
disp(T45)

T05 = T01 @ T12 @ T23 @ T34 @ T45
print("\nMatrix T05:")
disp(T05)

print("\nProblem: location x, y, z")
x05 = float(T05[0, 3])
y05 = float(T05[1, 3])
z05 = float(T05[2, 3])

print("x05: {}".format(x05))
print("y05: {}".format(y05))
print("z05: {}".format(z05))

# Inverse kinematics for the given problem.

x = 853.553
y = 0.0
z = 583.553

theta_1 = np.arctan2(y, z)
T01 = h_T(0, 0, theta_1, 0)
print("\nMatrix T01 for given problem:")
disp(T01)

T45 = h_T(0, 230, 0, 0)
print("\nMatrix T45 for given problem:")
disp(T45)

T15 = T05 @ T01.inv()
print("\nMatrix T15 for given problem:")
disp(T15)

```

(continues on next page)

(continued from previous page)

```
T14 = T15 @ T45.inv()
print("\nMatrix T14 for given problem:")
disp(T14)

x14 = float(T14[0, 3])
y14 = float(T14[1, 3])
z14 = float(T14[2, 3])

print("x14: {}".format(x14))
print("y14: {}".format(y14))
print("z14: {}".format(z14))

B = np.arctan2(z14, x14)
c2 = (l2**2 - l1**2 - x14**2 - z14**2) / (-2*l1*np.sqrt(x14**2 + z14**2))
s2 = np.sqrt(1 - c2**2)
w = np.arctan2(s2, c2)
theta_2 = B - w

c3 = (x14**2 + z14**2 - l1**2 - l2**2)/(2*l1*l2)
s3 = np.sqrt(1 - c3**2)
theta_3 = np.arctan2(s3, c3)

T35 = T34 @ T45
disp(T35)

x35 = float(T35[0, 3])
y35 = float(T35[1, 3])
z35 = float(T35[2, 3])

print("x35: {}".format(x35))
print("y35: {}".format(y35))
print("z35: {}".format(z35))

c4 = (x35 - l2) / l3
s4 = np.sqrt(1 - c4**2)
theta_4 = np.arctan2(s4, c4)

print("\ntheta_2: {}".format(np.rad2deg(theta_2)))
print("\ntheta_3: {}".format(np.rad2deg(theta_3)))
print("\ntheta_4: {}".format(np.rad2deg(theta_4)))
```

2.2 Output

```
  Matrix T01:
  1.0    0     0     0

   0    1.0    0     0

   0     0    1.0    0

   0     0     0    1.0

  Matrix T12:
  1.0           0                    0             0

```

(continues on next page)

(continued from previous page)

```
 0   6.12323399573677e-17          -1.0            0

 0           1.0            6.12323399573677e-17   0

 0            0                      0            1.0

Matrix T23:
0.707106781186548  -0.707106781186547   0   500.0

0.707106781186547  0.707106781186548    0     0

        0                  0           1.0    0

        0                  0            0    1.0

Matrix T34:
0.707106781186548  -0.707106781186547   0   500.0

0.707106781186547  0.707106781186548    0     0

        0                  0           1.0    0

        0                  0            0    1.0

Matrix T45:
1.0   0    0   230.0

 0   1.0   0     0

 0    0   1.0    0

 0    0    0    1.0

Matrix T05:
2.22044604925031e-16          -1.0                   0               853.
↪553390593274
                                                                          ␣
↪
6.12323399573677e-17  1.23259516440783e-32          -1.0          3.57323395960819e-
↪14
                                                                          ␣
↪
        1.0           2.22044604925031e-16  6.12323399573677e-17    583.
↪553390593274
                                                                          ␣
↪
         0                     0                     0                    1.0      ␣
↪


Problem: location x, y, z

x05: 853.5533905932738
y05: 3.573233959608189e-14
z05: 583.5533905932737

Matrix T01 for given problem:
```

(continues on next page)

(continued from previous page)

```
1.0   0    0    0

 0   1.0   0    0

 0    0   1.0   0

 0    0    0   1.0

Matrix T45 for given problem:
1.0   0    0   230.0

 0   1.0   0     0

 0    0   1.0    0

 0    0    0    1.0

Matrix T15 for given problem:
2.22044604925031e-16          -1.0                   0              853.
↪553390593274
                                                                                     ␣
↪
6.12323399573677e-17  1.23259516440783e-32          -1.0          3.57323395960819e-
↪14
                                                                                     ␣
↪
        1.0           2.22044604925031e-16  6.12323399573677e-17    583.
↪553390593274
                                                                                     ␣
↪
         0                     0                     0                    1.0       ␣
↪

Matrix T14 for given problem:
2.22044604925031e-16          -1.0                   0              853.
↪553390593274
                                                                                     ␣
↪
6.12323399573677e-17  1.23259516440783e-32          -1.0          2.16489014058873e-
↪14
                                                                                     ␣
↪
        1.0           2.22044604925031e-16  6.12323399573677e-17    353.
↪553390593274
                                                                                     ␣
↪
         0                     0                     0                    1.0       ␣
↪

x14: 853.5533905932738
y14: 2.164890140588733e-14
z14: 353.55339059327366

0.707106781186548  -0.707106781186547   0   662.634559672906

0.707106781186547  0.707106781186548    0   162.634559672906

```

(continues on next page)

(continued from previous page)

```
          0                  0            1.0          0

          0                  0             0          1.0

  x35: 662.6345596729059
  y35: 162.6345596729059
  z35: 0.0


  theta_2: -9.54166404439055e-15
  theta_3: 45.000000000000014
  theta_4: 45.000000000000014
```

CHAPTER THREE

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 Spinxs:

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