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 a fixed platform in 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 Mathworks Simulink and OpenModelica were chosen.

***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 6DOF (degrees of freedom) robot manipulator, consisting of a fixed base, a rotating base, two solid links and a toll, as shown in the Figure 1.

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 object and place it onto a horizontal surface.

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 14.

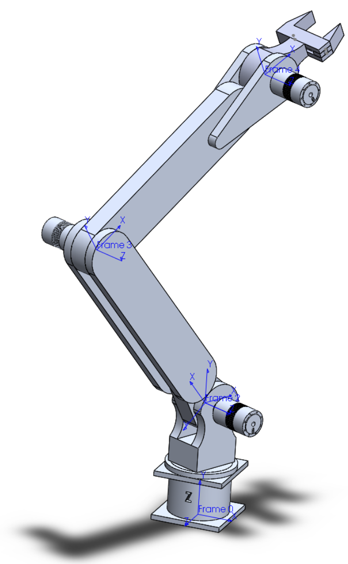

Fig. 1. Proposed robot manipulator system.

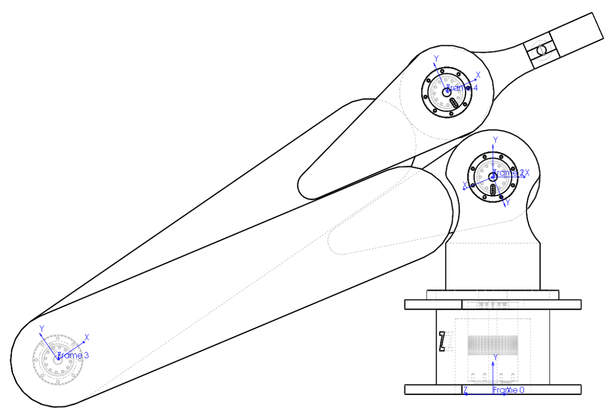

Fig. 2. Arms flexibility on Wireframe view.

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

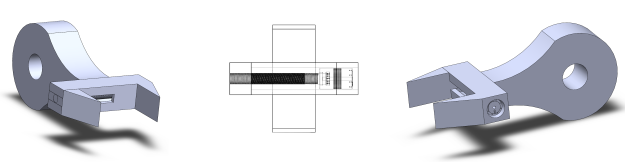

Fig. 3. Gripper details.

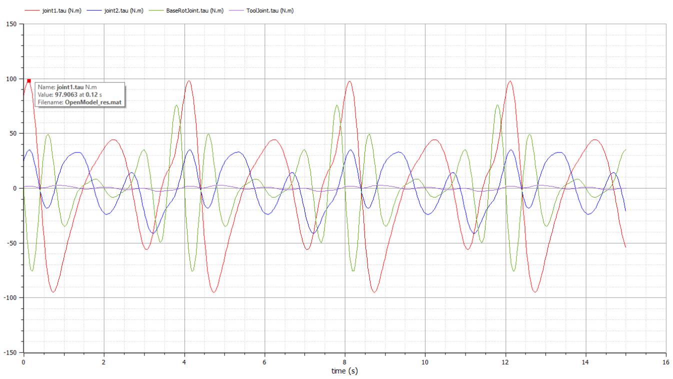

Fig. 4. Torque plot for various angles of the 4 joints, Max. 90.7566 N.m.

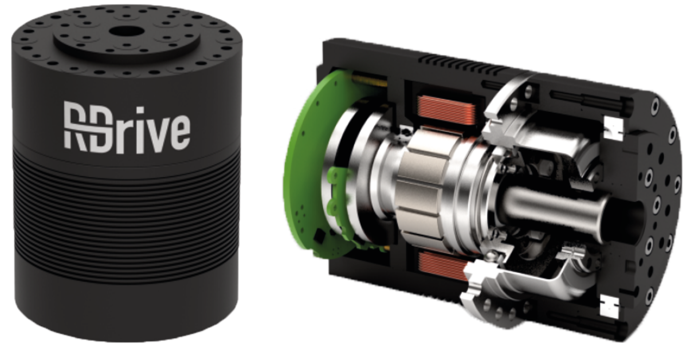

Fig. 5. RDrive motor.

C. Entire Model

Create a model of the robot manipulator with the correct dimensions. Using any method, software and hardware you are familiar with to simulate the movement (pick and place) of the robot manipulator. (3 marks)

IV. Other Components

Also, send a sheet of paper or PDF with complete contact information for all authors. Include full mailing addresses, telephone numbers, fax numbers, and e-mail addresses.

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 were made using Python (Apenddix A) and confirmed 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].

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.

Also a loading cell can be used on the toll to sense the amount of pressure to set a trigger to avoid damage on the object.

C. Controllers

The final printed size of an author photograph is exactly 1 inch wide by 1 1/4 inches long (6 picas $\times$ 7 1/2 picas). Please ensure that the author photographs you submit are proportioned similarly. If the author's photograph does not appear at the end of the paper, then please size it so that it is proportional to the standard size of 1 9/16 inches wide by 2 inches long (9 1/2 picas $\times$ 12 picas). JPEG files are only accepted for author photos.

D. Control Methodology

IEEE accepts color graphics in the following formats: EPS, PS, TIFF, Word, PowerPoint, Excel, and PDF. The resolution of a RGB color TIFF file should be 400 dpi.

When sending color graphics, please supply a high quality hard copy or PDF proof of each image. If we cannot achieve a satisfactory color match using the electronic version of your files, we will have your hard copy scanned. Any of the files types you provide will be converted to RGB color EPS files.

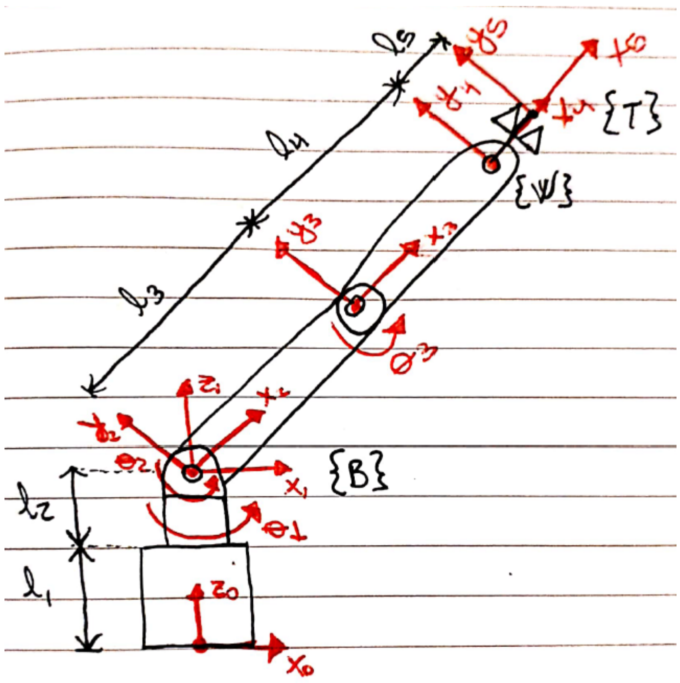

Fig. 6. $\hat{Z}, \hat{Y}, \hat{X}axis, \theta_1, \theta_2, \theta_3, l1, l2, l3$ and $l4$.

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 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

Finally, the forward kinematics equation is presented on Equation 8.

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

(1)

Denavit–Hartenberg parameters

$$ {}^0_1T = \begin{bmatrix} cos\theta_1 & -sin\theta_1 & 0 & 0 \\ sin\theta_1 & cos\theta_1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2} $$

$$ {}^1_2T = \begin{bmatrix} cos\theta_2 & -sin\theta_2 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ sin\theta_2 & cos\theta_2 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3} $$

$$ {}^2_3T = \begin{bmatrix} cos\theta_3 & -sin\theta_3 & 0 & l_2 \\ sin\theta_3 & cos\theta_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4} $$

$$ {}^3_4T = \begin{bmatrix} cos\theta_4 & -sin\theta_4 & 0 & l_3 \\ sin\theta_4 & cos\theta_4 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5} $$

$$ {}^4_5T = \begin{bmatrix} 1 & 0 & 0 & l_4 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6} $$

$$ {}^0_5T = {}^0_1T\,{}^1_2T\,{}^2_3T\,{}^3_4T\,{}^4_5T => {}^0\cancel{{}_1T^1_2T^2_3T^3_4T^4}{}_5T = {}^0_5T \tag{7} $$

$$ {}^0_5T = \begin{bmatrix} C_1C_{234} & -S_{234}C_1 & S_1 & C_1(l_2C_2 + l_3C_{23} + l_4C_{234}) \\ S_1C_{234} & -S_1S_{234} & -C_1 & S_1(l_2C_2 + l_3C_{23} + l_4C_{234}) \\ S_{234} & C_{234} & 0 & l_1 + l_2S_2 + l_3S_{23} + l_4S_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{8} $$

VI. Inverse Kinematics

There are many methods to find the equations for the inverse kinematic here two methodologies are presented. The position for θ_1 will be considered 0° and 180° and shown on Figure 7

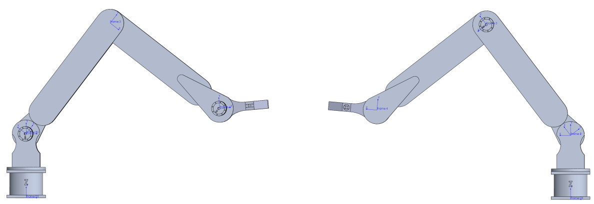

Fig. 7. θ_1 at positions 0° and 180°.

Considering these two options we have:

${}^1_5T$, for $\theta = 0$,

$$\begin{bmatrix} C_{234} & -S_{234} & 0 & l_2C_2 + l_3C_{23} + l_4C_{234} \\ 0 & 0 & -1 & 0 \\ S_{234} & C_{234} & 0 & l_1 + l_2S_2 + l_3S_{23} + l_4S_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

${}^1_5T$, for $\theta = 180$,

$$\begin{bmatrix} -C_{234} & S_{234} & 0 & -l_2C_2 - l_3C_{23} - l_4C_{234} \\ 0 & 0 & 1 & 0 \\ S_{234} & C_{234} & 0 & l_1 + l_2S_2 + l_3S_{23} + l_4S_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

And,

$${}^1_5T =^1_2 T^2_3T^3_4T^4_5T => ^1 \cancel{{}_2T^2_3T^3_4T^4} {}_5T =^1_5 T \tag{11}$$

A. *Resolving for $\theta = 0$*

Another approach to resolve the inverse kinematics is by systematically find the equations for θ_n by first isolating the dependent transpose on the left side by multiplying both sides of Equation 9 by the dependent transpose inverse ${}^1_2T(\theta_1)^{-1}$, as in Equation 46.

$${}^1_5T^2_3T(\theta_1)^{-1} =^1_2 T(\theta_3)^3_4T(\theta_4)^4_5T \tag{12}$$

Where,

$${}^1_2T(\theta_2)^2_3T(\theta_3)^3_4T(\theta_4)^4_5T =^1_5 T$$

Therefore,

$$\begin{bmatrix} C_3 & S_3 & 0 & 0 \\ -S_3 & C_3 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ l_2 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} =^1_5 T \tag{13}$$

We can equate:

$$\begin{bmatrix} -S_3 & C_3 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = 0 \tag{14}$$

Then we have:

$$-S_3P_x + C_3P_y = 0 \tag{15}$$

We can make the trigonometric substitutions:

$$\begin{aligned} P_x &= \rho sin\theta_1, \\ P_y &= -\rho cos\theta_1, \end{aligned} \tag{16}$$

$$\rho = \sqrt{P_x^2 - P_y^2} \tag{17}$$

Therefore,

$$\theta_3 = Atan2(P_x, P_y) \tag{18}$$

VII. 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 8 as if they were on the same plane.

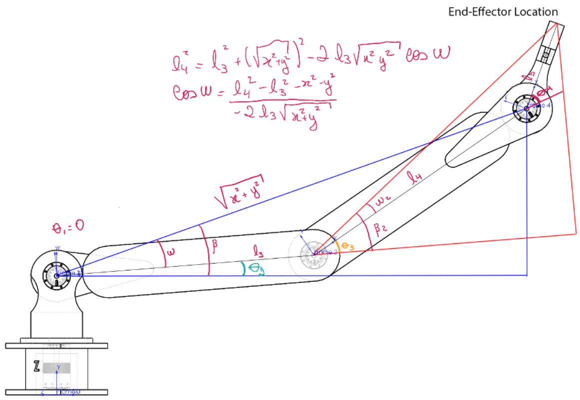

Fig. 8. Triangles from frame $\{1\}$ to $\{4\}$ and $\{3\}$ to $\{5\}$.

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 9 by the dependent transpose inverse T^{-1}, as shown below.

If we consider Equation 19:

$$ {}^0_5T = {}^0_1T\, {}^1_2T\, {}^2_3T\, {}^3_4T\, {}^4_5T \tag{19} $$

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

$$ {}^0_5T\, {}^0_1T^{-1} = {}^0_1T^{-1}\, {}^0_1T\, {}^1_2T\, {}^2_3T\, {}^3_4T\, {}^4_5T \tag{20} $$

Which is the same as,

$$ {}^1_5T = {}^1_2T\, {}^2_3T\, {}^3_4T\, {}^4_5T \tag{21} $$

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,

$$ {}^1_4T = {}^1_5T\, {}^4_5T^{-1} \tag{22} $$

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

$$ {}^3_5T = {}^1_5T\, {}^1_2T^{-1}\, {}^2_3T^{-1} \tag{23} $$

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

$$ {}^1_4T = \begin{bmatrix} C_{234} & -S_{234} & 0 & l_2C_2 + l_3C_{23} \\ 0 & 0 & -1 & 0 \\ S_{234} & C_{234} & 0 & l_2S_2 + l_3S_{23} \\ 0 & 0 & -1 & 1 \end{bmatrix} \tag{24} $$

And ${}^3_5T$,

$$ {}^3_5T = $$
$$ \begin{bmatrix} C_3C_{34} & S_3C_{34} & -S_{34} & l_2C_2 - l_2C_3C_{34} + L_3C_{23} + L_4C_{234} \\ -S_3 & C_3 & 0 & l_2S_3 \\ S_{34}C_3 & S_3S_{34} & cos_{34} & l_2S_2 - l_2C_3S_{34} + L_3S_{23} + L_4S_{234} \\ 0 & 0 & -1 & 1 \end{bmatrix} \tag{25} $$

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

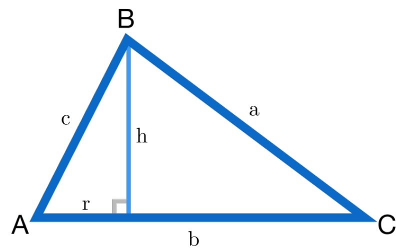

Fig. 9. Law of cosines.

$$ \begin{gathered} c^2 = \vec{c}.\vec{c} = \\ (\vec{b} - \vec{a}).(\vec{b} - \vec{a}) = \\ b^2 + a^2 - 2\vec{a}.\vec{b} = \\ b^2 + a^2 - 2abcosC. \end{gathered} \tag{26} $$

Applying to the blue triangle on Figure 8,

$$ \begin{gathered} cos\omega = \frac{l_4^2 - l_3^2 - x^2 - y^2}{-2l_3\sqrt{x^2+y^2}} \\ sin\omega = \sqrt{1 - cos\omega^2} \end{gathered} \tag{27} $$

Therefore,

$$ \begin{gathered} \beta = Atan2(x_{15}, y_{15}) \\ \theta_2 = \beta - \omega \end{gathered} \tag{28} $$

Applying the same to the red triangle and considering (x, y) as from ${}^3_5T$, we have,

$$ \begin{gathered} l_5^2 = l_4^2 + \sqrt{x^2+y^2}^2 - 2l_4\sqrt{x^2+y^2}cos\omega_2 = \\ cos\omega_2 = \frac{l_5^2 - l_4^2 - x^2 - y^2}{-2l_4\sqrt{x^2+y^2}} \\ sin\omega_2 = \sqrt{1 - cos\omega_2{}^2} \end{gathered} \tag{29} $$

Therefore,

$$ \begin{gathered} \beta_2 = Atan2(x_{35}, y_{35}) \\ \theta_3 = \beta_2 - \omega_2 \end{gathered} \tag{30} $$

For θ_4 we have,

$$ \theta_4 = Atang(x_{14}, y_{14}) + \frac{{l_3}^2 - {l_2}^2 - {x_{14}}^2 - {y_{14}}^2}{-2l_2\sqrt{{x_{14}}^2 + {y_{14}}^2}} \tag{31} $$

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

A. First Method

To find the joint displacements leading the centre of the end-effector from a vertical position to a horizontal position with correct orientation is necessary to obtain the inverse kinematics equations. So let Equation 32 be:

$$ {}^{0}_{5}T = \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{32} $$

for θ_1, θ_2, θ_3 and θ_4, we have:

$$ {}^{0}_{5}T = {}^{0}_{1}T(\theta_1){}^{1}_{2}T(\theta_2){}^{2}_{3}T(\theta_3){}^{3}_{4}T(\theta_4){}^{4}_{5}T \tag{33} $$

Now to resolve for θ_1, θ_2, θ_3, and θ_4, we have to find $Atan2(S_1, C_1)$, $Atan2(S_2, C_2)$, $Atan2(S_3, C_3)$, and $Atan2(S_4, C_4)$. To achieve that we have to isolate the sin and cos by equating the elements of the matrices calculated. For θ_1 we can equate element (1, 3) and element (2, 3) of Equation 32 in relation to Equation 8:

For r_{13}:

$$ S_1 = r_{13} \tag{34} $$

For r_{23}:

$$ -C_1 = r_{23} \tag{35} $$

We can make the trigonometric substitutions:

$$ \begin{gathered} r_{13} = \rho sin\theta_1, \\ r_{23} = -\rho cos\theta_1, \end{gathered} \tag{36} $$

$$ \rho = \sqrt{r_{13}^2 - r_{23}^2} \tag{37} $$

Therefore,

$$ \begin{gathered} \theta_1 = Atan2(r_{13}, -r_{23}), or \\ \theta_1 = Atan2(S_1, C_1) \end{gathered} \tag{38} $$

If both $r_{13} = 0$ and $r_{23} = 0$ the goal is unattainable.

For θ_2 we can equate element (1, 4), element (2, 4), and and element (3, 4) of Equation 32 in relation to Equation 8:

For r_{14}:

$$ P_x = C_1(l_2C_2 + l_3C_{23} + l_4C_{234}) \tag{39} $$

For r_{24}:

$$ P_y = S_1(l_2C_2 + l_3C_{23} + l_4C_{234}) \tag{40} $$

For r_{34}:

$$ P_z = l_1 + l_2S_2 + l_3S_{23} + l_4S_{234} \tag{41} $$

Using the trigonometric substitutions:

$$ cos(\theta_1 + \theta_2) = C_1C_2 - S_1S_2 \tag{42} $$

$$ sin(\theta_1 + \theta_2) = C_1S_2 - S_1C_2 \tag{43} $$

We can find:

$$ \begin{gathered} C_{234} = \\ C((\theta_2 + \theta_3) + \theta_4) = \\ C_{23}C_4 - S_{23}S_4 = \\ C_4(C_2C_3 - S_2S_3) - S_4(C_2S_3 - S_2C_3) = \\ \\ C_2C_3C_4 - S_2S_3C_4 - C_2S_3S_4 + S_2C_3S_4 \end{gathered} \tag{44} $$

and,

$$ \begin{gathered} S_{234} = \\ S((\theta_2 + \theta_3) + \theta_4) = \\ C_{23}S_4 - S_{23}C_4 = \\ S_4(C_2C_3 - S_2S_3) - C_4(C_2S_3 - S_2C_3) \\ \\ C_2C_3S_4 - S_2S_3S_4 - C_2S_3C_4 + S_2C_3C_4 \end{gathered} \tag{45} $$

B. Second Method

Another approach to resolve the inverse kinematics is by systematically find the equations for θ_n by first isolating the dependent transpose on the left side by multiplying both sides of Equation 33 by the dependent transpose inverse ${}^0_1T(\theta_1)^{-1}$, as in Equation 46.

$$ {}^0_5T {}^0_1T(\theta_1)^{-1} = {}^1_2T(\theta_2) {}^2_3T(\theta_3) {}^3_4T(\theta_4) {}^4_5T \tag{46} $$

Where,

$$ {}^1_2T(\theta_2) {}^2_3T(\theta_3) {}^3_4T(\theta_4) {}^4_5T = {}^1_5T $$

Therefore,

$$ \begin{bmatrix} C_1 & S_1 & 0 & 0 \\ -S_1 & C_1 & 0 & 0 \\ 0 & 0 & 1 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & P_x \\ r_{21} & r_{22} & r_{23} & P_y \\ r_{31} & r_{32} & r_{33} & P_z \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^1_5T \tag{47} $$

$$ {}^1_5T = \begin{bmatrix} C_{234} & -S_{234} & 0 & l_2C_2 + l_3C_{23} + l_4C_{234} \\ 0 & 0 & -1 & 0 \\ S_{234} & C_{234} & 0 & l_2S_2 + l_3S_{23} + l_4S_{234} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{48} $$

From this we can take that:

$$ \begin{bmatrix} C_1 & S_1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = l_2C_2 + l_3C_{23} + l_4C_{234} \tag{49} $$

$$ C_1P_x + S_1P_y = l_2C_2 + l_3C_{23} + l_4C_{234} $$

$$ \begin{bmatrix} -S_1 & C_1 & 0 & 0 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = 0 \tag{50} $$

$$ -S_1P_x + C_1P_y = 0 $$

$$ \begin{bmatrix} 0 & 0 & 1 & l_1 \end{bmatrix} \times \begin{bmatrix} P_x \\ P_y \\ P_z \\ 1 \end{bmatrix} = l_2S_2 + l_3S_{23} + l_4S_{234} \tag{51} $$

$$ P_z + l_1 = l_2S_2 + l_3S_{23} + l_4S_{234} $$

$$ \begin{bmatrix} 0 & 0 & 1 & l_1 \end{bmatrix} \times \begin{bmatrix} r_{11} \\ r_{21} \\ r_{31} \\ 1 \end{bmatrix} = S_{234} \tag{52} $$

$$ r_{31} + l_1 = S_{234} $$

$$ \begin{bmatrix} 0 & 0 & 1 & l_1 \end{bmatrix} \times \begin{bmatrix} r_{12} \\ r_{22} \\ r_{32} \\ 1 \end{bmatrix} = C_{234} \tag{53} $$

$$ r_{32} + l_1 = C_{234} $$

So,

$$ \begin{cases} C_1P_x + S_1P_y = l_2C_2 + l_3C_{23} + l_4C_{234} \\ -S_1P_x + C_1P_y = 0 \\ P_z + l_1 = l_2S_2 + l_3S_{23} + l_4S_{234} \end{cases} \tag{54} $$

Solving for θ_1 we have:

VIII. 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]

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 10 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 11.

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 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 12. 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 13.

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 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

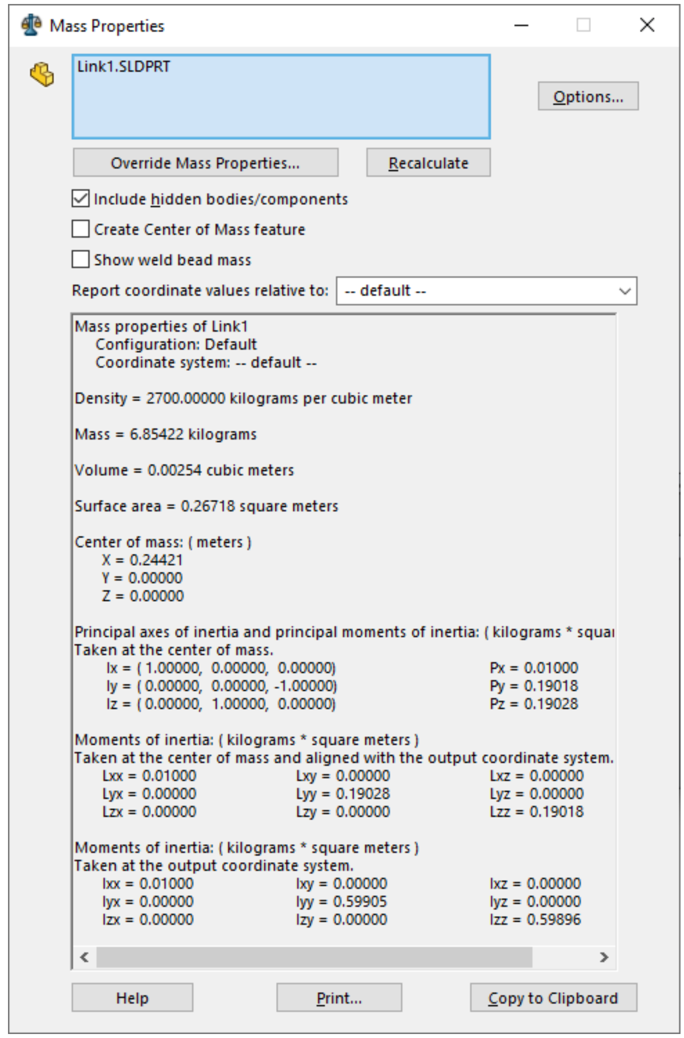

Fig. 10. Mass, center of mass and moments of inertia used on the simulation from link1 - SolidWorks.

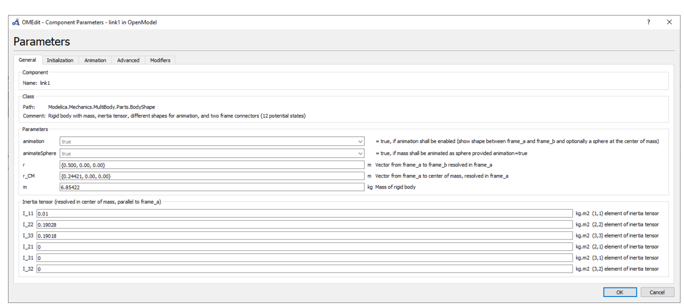

Fig. 11. OpenModelica, link1 parameters.

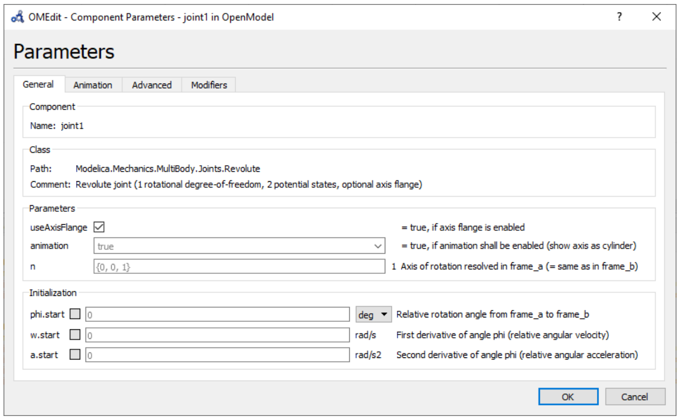

Fig. 12. OpenModelica, Joint1 parameters.

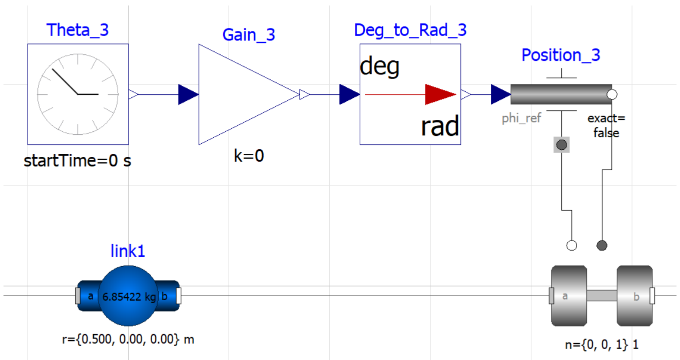

Fig. 13. Joint1, Link1 and controller.

import parameters for link 1 on Figure 15, by exporting the coordinate system placed on the frame position, Figure 14, 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 11 and Figure 16.

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 17 the values for the torque for joint 1 and 2 using $\theta_{1-4} = 0°$.

At Figure 18 the simulation modelling at OpenModelica is shown, the parameters used at the $Gain$ block results in an animation, Figure 19, that goes from all the angles been 0 to the values set.

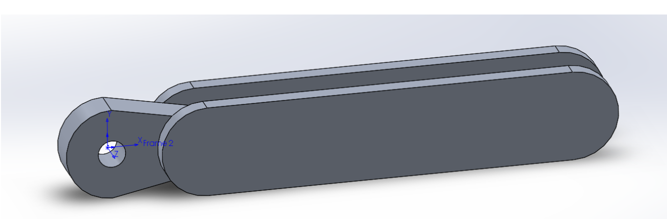

Fig. 14. Coordinate system on the frame $\{4\}$ position for link 1.

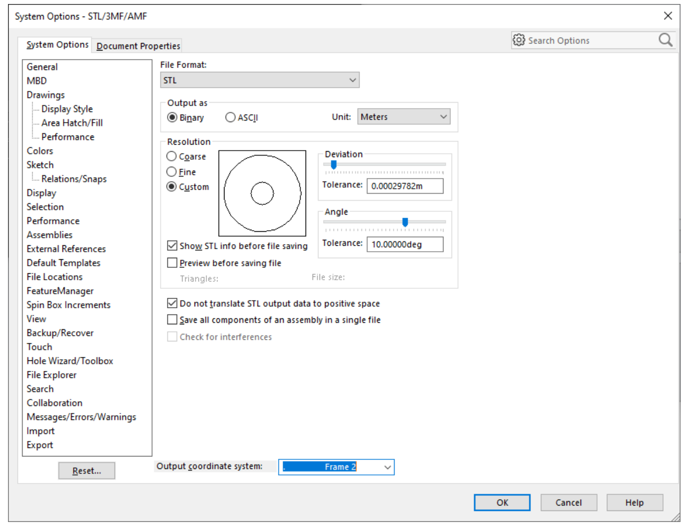

Fig. 15. Exporting parameters.

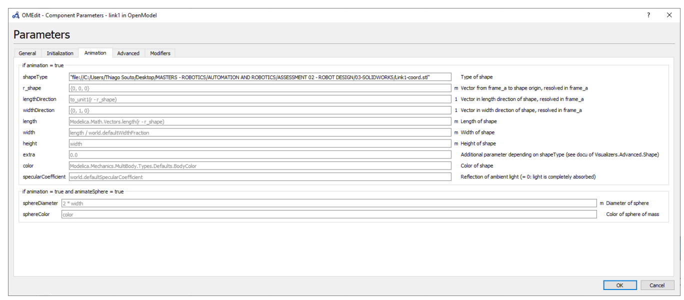

Fig. 16. Importing parameters.

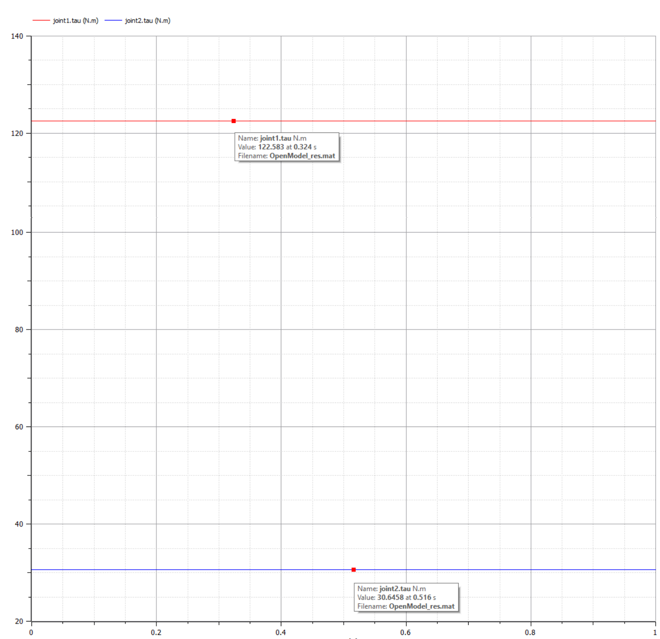

Fig. 17. Torques required for $\theta_{1-4} = 0°$.

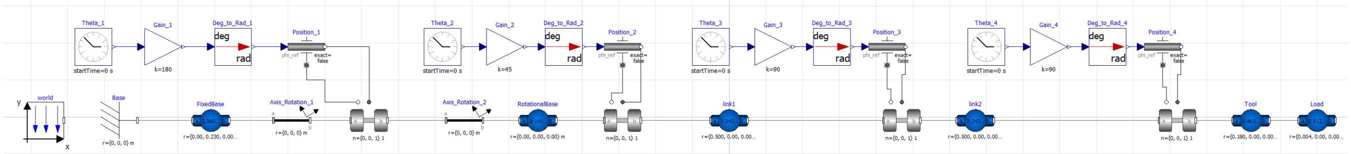

Fig. 18. Simulation model, Base, rotational link and 2 more links.

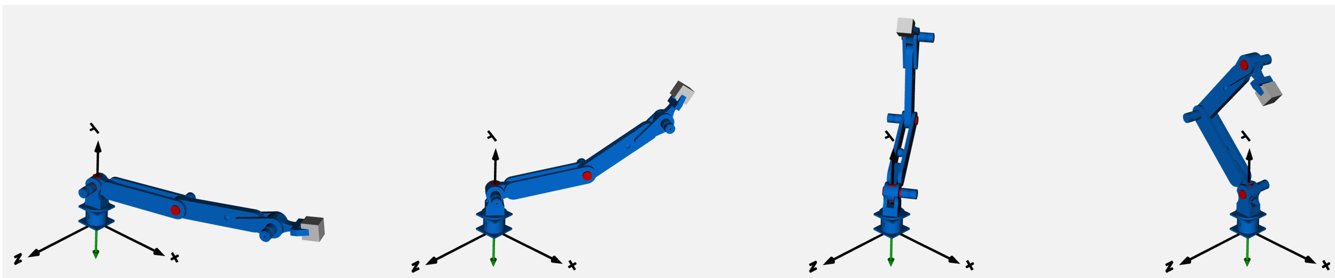

Fig. 19. Simulation animation.

Another simulation was made in SolidWorks to show that the proposed design is capable of picking an object from a vertical wall/shelf and placing it onto a horizontal surface.

IX. Discussion and Conclusions

The IEEE Graphics Checker Tool enables users to check graphic files. The tool will check journal article graphic files against a set of rules for compliance with IEEE requirements. These requirements are designed to ensure sufficient image quality so they will look acceptable in print. After receiving a graphic or a set of graphics, the tool will check the files against a set of rules. A report will then be e-mailed listing each graphic and whether it met or failed to meet the requirements. If the file fails, a description of why and instructions on how to correct the problem will be sent. The IEEE Graphics Checker Tool is available at http://graphicsqc.ieee.org/

For more Information, contact the IEEE Graphics H-E-L-P Desk by e-mail at mailto:graphics@ieee.org. You will then receive an e-mail response and sometimes a request for a sample graphic for us to check.

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://blogs.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

Robot manipulator Documentation

Release 1.0

Thiago Souto

May 12, 2020

CONTENTS:

CHAPTER ONE

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

```python
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)

```python
        :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(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


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_1_5:

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_1_5: for theta_1 = 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_1_5: for theta_1 = 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)

(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 T_1_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 t_3_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
```

CHAPTER

TWO

INVERSE_KINEMATICS MODULE

2.1 Python Program

```
import numpy as np
import sympy

np.set_printoptions(precision=3,
                    suppress=True)
sympy.init_printing(num_columns=240)

l2 = 500.0
l3 = 500.0
l4 = 230.0

theta_2 = np.deg2rad(0.0)
theta_3 = np.deg2rad(45.0)
theta_4 = np.deg2rad(45.0)

# joint 1

x05 = 500 + 500 * np.cos(theta_2)
y05 = 0.0
z05 = 500 + 500 * np.sin(theta_3)

t1 = np.arctan2(y05, x05)

print("theta_2 = {}".format(t1))

# joint 2

x14 = 500 + 500 * np.cos(theta_3)
y14 = 500 * np.sin(theta_3)
z14 = 0.0

B = np.arctan2(y14, x14)
c2 = (l3**2 - l2**2 - x14**2 - y14**2)/(-2*l2*np.sqrt(x14**2+y14**2))
s2 = np.sqrt(1 - c2**2)
w = np.arctan2(s2, c2)
t2 = B - w

print("theta_3 = {:.3f}".format(np.rad2deg(t2)))

t2 = np.arctan2(y14, x14) + np.arccos((l3**2 - l2**2 - x14**2 - y14**2) / (-2*l2*np.sqrt(x14**2 + y14**2)))
```

(continues on next page)

(continued from previous page)

```

print("theta_4 = {:.3f}".format(np.rad2deg(t2)))
```

2.2 Output

```
theta_2 = 0.0
theta_3 = -0.000
theta_4 = 45.000
```

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