

---

# Lab 1: Kinematic Characterization of the Lynx (MATLAB)

---

UNIVERSITY OF PENNSYLVANIA

WESLEY YEE, SHAUN FEDRICK  
MEAM 520  
SEPTEMBER 23, 2020

# 1 Methods

For our experimental setup, we used MATLAB to code our forward kinematics calculations, which communicated to ROS via Gazebo on a local VM running Ubuntu. A 6-DOF robot arm manipulator was programmed to run in Gazebo. Once the code was run in MATLAB on the host computer, the commands for operating the robot were then sent to Gazebo which then actuated the robot to the desired joint variable positions.

The ROS robot was modeled off of an actual physical robot located at Penn, but was not accessible due to COVID-19.

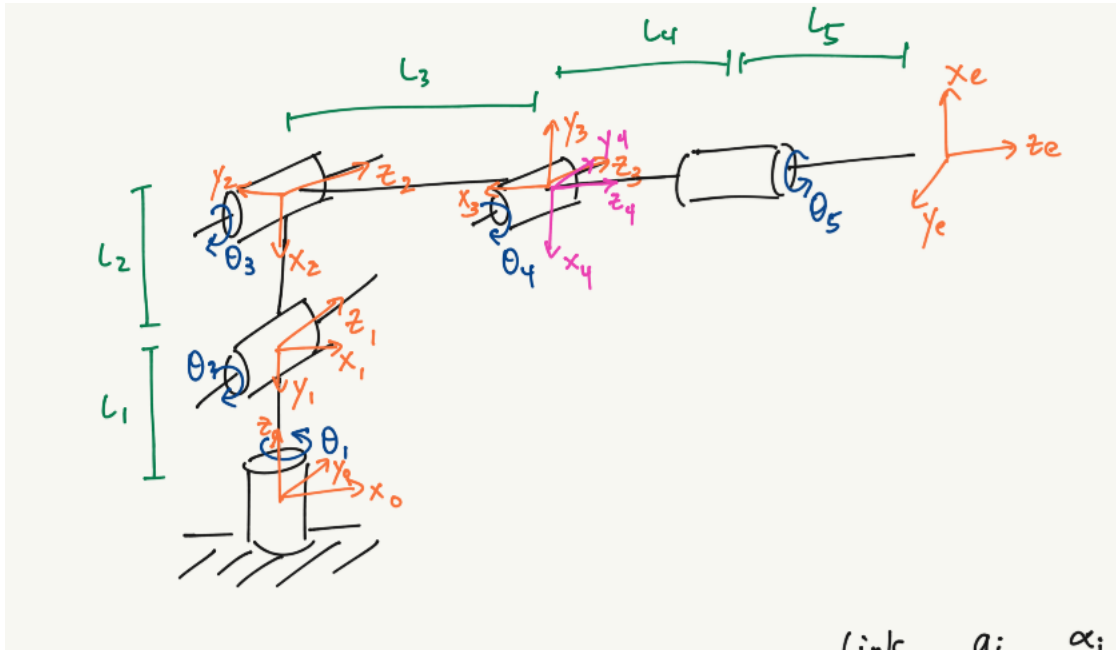


Figure 1: Revised symbolic representation

1. As can be seen from our schematic we chose to use DH4 convention to describe the forward kinematics of our robot. To do this, we needed to follow the DH4 parameters for each link. This can be seen in the

matrix below.

$$DH_4 = \begin{bmatrix} Link & a_i & \alpha_i & d_i & \theta_i \\ 1 & 0 & -\frac{\pi}{2} & L_1 & \theta_1 \\ 2 & -L_2 & 0 & 0 & \theta_2 + \frac{\pi}{2} \\ 3 & -L_3 & 0 & 0 & \theta_3 + \frac{\pi}{2} \\ 4 & 0 & \frac{\pi}{2} & 0 & \theta_4 - \frac{\pi}{2} \\ 5(e) & 0 & 0 & L_4 + L_5 & \theta_5 + \pi \end{bmatrix} \quad (1)$$

It is worth noting that DH parameters work by specifying a Homogeneous transform consisting of a rotation about  $\theta_i$  along with a translation along  $z_{i-1}$ . Then from this intermediate frame you perform translation in x by  $a_i$  along with a final rotation about x by  $\alpha_i$ . Multiplying these transformations gives the homogeneous transform  $A_i^{i-1}$ , we use c to represent cosine and s to represent sine A is given as:

$$A_i^{i-1} = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_ic_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_is_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\theta_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2)$$

2. Using  $A_i^{i-1}$  we get the transformation from consecutive links to be:

$$T_1^0 = \begin{bmatrix} c_{\theta_1} & 0 & -s_{\theta_1} & 0 \\ s_{\theta_1} & 0 & c_{\theta_1} & 0 \\ 0 & -1 & c_{\theta_1} & L_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3)$$

$$T_2^1 = \begin{bmatrix} -s_{\theta_2} & -c_{\theta_2} & 0 & L_2s_{\theta_2} \\ c_{\theta_2} & -s_{\theta_2} & 0 & -L_2c_{\theta_2} \\ 0 & 0 & -s_{\theta_1} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4)$$

$$T_3^2 = \begin{bmatrix} -s_{\theta_3} & -c_{\theta_3} & 0 & L_3s_{\theta_3} \\ c_{\theta_3} & -s_{\theta_3} & 0 & -L_3c_{\theta_3} \\ 0 & 0 & -s_{\theta_3} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (5)$$

$$T_4^3 = \begin{bmatrix} s_{\theta_4} & 0 & -c_{\theta_4} & 0 \\ -c_{\theta_4} & 0 & -s_{\theta_4} & 0 \\ 0 & 1 & s_{\theta_4} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6)$$

$$T_{5(e)}^4 = \begin{bmatrix} -c_{\theta_5} & s_{\theta_5} & 0 & 0 \\ -s_{\theta_5} & -c_{\theta_5} & 0 & 0 \\ 0 & 0 & -c_{\theta_5} & L_4 + L_5 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7)$$

3. To get the position of the end effector in terms of the end effector frames all we need to do is post multiply all the matrices from the previous part. Writing out the entirety of this matrix is not feasible, but we can write it in terms of the matrices we are multiplying. This gives:  
 $T_{5(e)}^0 = T_1^0 * T_2^1 * T_3^2 * T_4^3 * T_{5(e)}^4$

## 2 Evaluation

1. The following is the zero joint orientation and location of the end effector. This matrix matches the expected transformation matrix from the pre-lab.

$$T_e^0 = \begin{bmatrix} 0 & 0 & 1 & 255.325mm \\ 0 & -1 & 0 & 0mm \\ 1 & 0 & 0 & 222.25mm \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (8)$$

2. (a) The following is the matrix for  $q = [pi/4 \ 0 \ 0 \ 0 \ 0 \ 0]$ :

$$T_e^0 = \begin{bmatrix} 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 180.542mm \\ 0 & -\frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} & 180.542mm \\ 1 & 0 & 0 & 222.25mm \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (9)$$

- (b) The following is the matrix for  $q = [-pi/2 \ 0 \ pi/4 \ 0 \ 0 \ pi/2]$ :

$$T_e^0 = \begin{bmatrix} -1 & 0 & 0 & 0mm \\ 0 & \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & -180.542mm \\ 0 & -\frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} & 41.708mm \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (10)$$

3. The first interesting configuration we found was done by attempting to make the end effector collide with the base of the robot. This was very possible despite it being within the joint limits, and this was done by imputing  $q=[0,0,1.5882,1.700,0,0]$ .

```

[INFO]: Finish calibration ...
[INFO]: Now the position is: ...
    0.0000   -0.0003   1.5882   1.7000   -0.0000   0.0200

Now the current state value is : ...
    0.0000   -0.0003   1.5882   1.7000   -0.0000   0.0200

Finish the command!
Simulation Joint Positions =
    0         0         0
    0         0   76.2000
    0.0001   0.0000  222.2500
   -0.0012   0.0000  34.9250
  -33.7165   0.0000  39.3057
  -67.4330   0.0001  43.6864

Predicted Joint Positions =
    0         0         0
    0         0   76.2000
   -0.0000  -0.0000  222.2500
    0.0000   0.0000  34.9250
  -33.7166   0.0000  39.3057
  -67.4332   0.0000  43.6864

Simulation Toe =
   -0.1288   0.0000  -0.9917  -67.4330
   -0.0000   1.0000   0.0000   0.0001
    0.9917   0.0000  -0.1288   43.6864
    0         0         0   1.0000

Predicted Toe =
   -0.1288  -0.0000  -0.9917  -67.4332
    0.0000  -1.0000   0.0000   0.0000
  -0.9917  -0.0000   0.1288   43.6864
    0         0         0   1.0000

```

Figure 2: First interesting configuration

Another interesting point was entering the joint limit of every joint. We were hoping to see if gazebo misbehaved right at its dexterous workspace, but saw no such behavior. This was reassuring because it allows us to trust gazebo to handle inputs that approach or even go pass the joint limits.

```

[INFO]: Start send message ...
Warning: [WARN]: State 6 is above the limit 30
> In ArmController/add_command (line 124)
  In TestFK_Sim (line 12)
[INFO]: The command is ...
      1.4000  -1.2000  -1.8000  -1.9000   1.5000  30.0000

[INFO]: Finish calibration ...
[INFO]: Now the position is: ...
      1.3849  -1.2001  -1.8002  -1.8957   1.4997  -0.0000

Now the current state value is : ...
      1.3849  -1.2001  -1.8002  -1.8957   1.4997  -0.0000

Finish the command!
Simulation Joint Positions =
      0      0      0
      0      0  76.2000
 -23.1410 -134.1630 129.0715
 -54.6594 -316.9025 155.5154
 -53.5798 -310.6468 122.1027
 -52.5012 -304.3891  88.6915

Predicted Joint Positions =
      0      0      0
      0      0  76.2000
 -23.1367 -134.1437 129.1224
 -54.6571 -316.8957 155.5577
 -53.5793 -310.6465 122.1543
 -52.5015 -304.3974  88.7509

Simulation T0e =
      0.9824   0.1841   0.0317  -52.5012
     -0.1863   0.9777   0.0969  -304.3891
     -0.0132  -0.1011   0.9948   88.6915
           0           0           0   1.0000

Predicted T0e =
      0.9948  -0.0969   0.0317  -52.5015
     -0.1011  -0.9778   0.1838  -304.3974
      0.0132  -0.1860  -0.9825   88.7509
           0           0           0   1.0000

```

Figure 3: Entering the joint limits of every joint

We also noticed that going over the joint limits causes our predicted values to deviate greatly. This is because Ros and gazebo prevent the joint from traveling further along each joint than is possible. However, our code does not account for going over the joint limits. As a result, entering -5 into  $\theta_2$  as we did in our example causes our predicted end effector position to be significantly off.

```

9 % add command
10 %q = [0 0 pi/2 1.7 0 0];
11 q = [0 -3 -5 0 0 100];
12 con = add_command(con,q);
13 disp("Now the current state value is : ...");
14 disp(con.cur_state);
15 disp("Finish the command!");

```

Command Window

The value of the ROS\_IP environment variable, 192.168.1.159, will be used to set the

[INFO]: Current state value is ...  
-0.0000 -0.0000 -1.8000 -0.0000 0.0000 -0.0000

[INFO]: Start send message ...  
Warning: [WARN]: State 2 is below the limit -1.2  
> In ArmController/add\_command (line 120)  
In TestFK\_Sim (line 12)  
Warning: [WARN]: State 3 is below the limit -1.8  
> In ArmController/add\_command (line 120)  
In TestFK\_Sim (line 12)  
Warning: [WARN]: State 6 is above the limit 30  
> In ArmController/add\_command (line 124)  
In TestFK\_Sim (line 12)

[INFO]: The command is ...  
0 -1.2000 -1.8000 0 0 30.0000

[INFO]: Finish calibration ...  
[INFO]: Now the position is: ...  
-0.0000 -1.1853 -1.8001 0.0000 0.0000 0.0000

Now the current state value is : ...  
-0.0000 -1.1853 -1.8001 0.0000 0.0000 0.0000

Finish the command!  
Simulation Joint Positions =  
0 0 0  
0 0 76.2000  
-136.1264 -0.0004 129.1170  
-321.5806 0.0007 155.5317  
-355.2402 0.0011 160.3352  
-388.9078 0.0015 165.1538

Predicted Joint Positions =  
0 0 0  
0 0 76.2000  
140.0509 -0.0000 117.6289  
-17.1282 0.0000 15.7201  
-45.6566 0.0000 -2.7766  
-74.1850 0.0000 -21.2733

Simulation Toe =  
-0.1411 0.0000 -0.9900 -388.9078  
-0.0000 1.0000 0.0000 0.0015  
0.9900 0.0000 -0.1411 165.1538  
0 0 0 1.0000

Predicted Toe =  
0.5440 0.0000 -0.8391 -74.1850  
0.0000 -1.0000 0.0000 0.0000  
-0.8391 -0.0000 -0.5440 -21.2733  
0 0 0 1.0000

Figure 4: Interesting q case

This is a simple starting position, but we consider it an interesting point because if you look at the orientation that gazebo outputs you can clearly see that the z axis is incorrectly oriented. This is very important to know when evaluating the orientation of the end effector.



```

[INFO]: Start send message ...
Warning: [WARN]: State 6 is above the limit 30
> In ArmController/add_command (line 124)
  In TestFK Sim (line 12)
[INFO]: The command is ...
    0    0    0    0    0    30

[INFO]: Finish calibration ...
[INFO]: Now the position is: ...
    0    0    0    0    0    0.0133

Now the current state value is : ...
    0    0    0    0    0    0.0133

Finish the command!
Simulation Joint Positions =
    0    0    0
    0    0    76.2000
    0.0002 -0.0000 222.2500
    187.3252 -0.0002 222.2500
    221.3252 -0.0003 222.2500
    255.3252 -0.0003 222.2500

Predicted Joint Positions =
    0    0    0
    0    0    76.2000
   -0.0000 -0.0000 222.2500
    187.3250 -0.0000 222.2500
    221.3250 -0.0000 222.2500
    255.3250 -0.0000 222.2500

Simulation Toe =
    0.0000 -0.0000 1.0000 255.3252
    0.0000 1.0000 0.0000 -0.0003
   -1.0000 0.0000 0.0000 222.2500
    0    0    0    1.0000

Predicted Toe =
   -0.0000 0.0000 1.0000 255.3250
    0.0000 -1.0000 0.0000 -0.0000
    1.0000 0.0000 0.0000 222.2500
    0    0    0    1.0000

```

Figure 5: Inverted z axis

We chose to orient the arm straight up along the z axis. In gazebo we noticed that maintaining this orientation, seemed to be unstable and the arm continually swayed in the positive negative x direction. We found this very interesting, and became curious whether this is a result of simulated physics in ros.

```

[INFO]: Finish calibration ...
[INFO]: Now the position is: ...
-0.0000 -0.0011 -1.5711 -0.0002 -0.0000 0.0000

Now the current state value is : ...
-0.0000 -0.0011 -1.5711 -0.0002 -0.0000 0.0000

Finish the command!
Simulation Joint Positions =
      0      0      0
      0      0  76.2000
  0.1102 -0.0000 222.2500
 -1.6386 0.0000 409.5664
  2.3644 -0.0000 443.5635
 -2.4525 0.0000 477.5616

Predicted Joint Positions =
      0      0      0
      0      0  76.2000
-0.0000 -0.0000 222.2500
-0.0000 -0.0000 409.5750
-0.0000 -0.0000 443.5750
-0.0000 -0.0000 477.5750

Simulation T0e =
-1.0000 0.0000 -0.0088 -2.4525
-0.0000 1.0000 0.0000 0.0000
 0.0088 0.0000 -1.0000 477.5616
      0      0      0  1.0000

Predicted T0e =
-1.0000 -0.0000      0 -0.0000
 0.0000 -1.0000      0 -0.0000
      0      0  1.0000 477.5750
      0      0      0  1.0000

```

Figure 6: Pointing straight up

4. Calculating the workspace was a matter of walking through the joint space. That is to say, that the permutation of all possible joint coordinates will give the entirety of the workspace. To implement this, we used 5 for loops, giving a time complexity of  $O(n^5)$ . Instead of reducing the dimensionality of joint space to reduce the time complexity, we chose to use a larger step sizes as we walked through each dimension in joint space. This was feasible by using MATLAB's trisurf command that allowed us to plot a 3d surface with relatively few 3d points. To get the corresponding point in 3 dimensional space from a coordinate in joint space, we used the  $T_{5(e)}^0$  that we calculated previously. By inputting the joint parameters into  $T_{5(e)}^0$  and pulling the final column of this homogeneous matrix, we were able to get a point in 3D that represented the location of the end effector given some joint parameters.

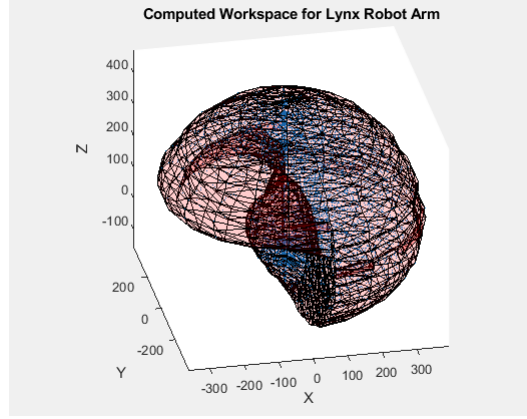


Figure 7: Computed Workspace of Lynx Robot

5. As seen in the code below, the predicted joint positions are almost exactly aligned with the simulated joint positions, aside from a 0.002mm difference in the x position of Joint 3. This is negligible and can be explained due to possible compounded rounding errors in the ROS robot.

However, when examining the  $T_e^0$  matrices, the Simulation  $T_e^0$  records different values of  $x_0$  and  $y_0$  in the end effector frame. We believe that this is due to a mistake in the provided ROS robot code used to calculate Simulation  $T_e^0$ . Justification for this conjecture is provided in Problem 1 of the Analysis section.

Listing 1: Outputs of TestFK\_Sim.m for  $q = [0 \ 0 \ 0 \ 0 \ 0 \ 0]$

Simulation Joint Positions =		
0	0	0
0	0	76.2000
0.0002	0.0000	222.2500
187.3252	-0.0002	222.2500
221.3252	-0.0003	222.2500
255.3252	-0.0003	222.2500
Predicted Joint Positions =		
0	0	0
0	0	76.2000

-0.0000	-0.0000	222.2500	
187.3250	-0.0000	222.2500	
221.3250	-0.0000	222.2500	
255.3250	-0.0000	222.2500	
Simulation T0e =			
0.0000	-0.0000	1.0000	255.3252
0.0000	1.0000	0.0000	-0.0003
-1.0000	0.0000	0.0000	222.2500
0	0	0	1.0000
Predicted T0e =			
-0.0000	0.0000	1.0000	255.3250
0.0000	-1.0000	0.0000	-0.0000
1.0000	0.0000	0.0000	222.2500
0	0	0	1.0000

### 3 Analysis

1. As expected, the results of our evaluation were correct. From the figure below, we use Right-Hand Rule convention to establish that the base frame's red axis is  $x_0$ , the green axis is  $y_0$  and the blue axis is  $z_0$ . The same axes are applied to the end effector frame, respectively.

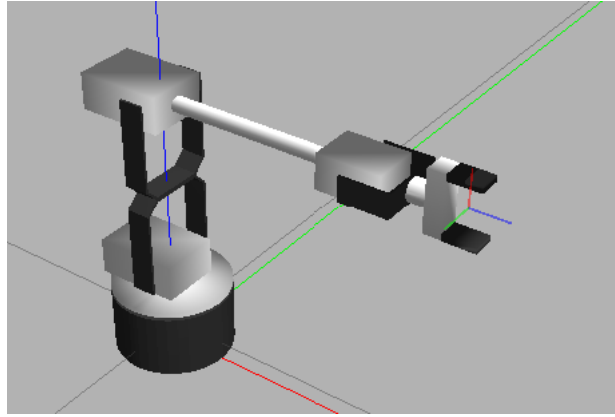


Figure 8: ROS Robot in Zero Configuration

If we use these axes to construct a rotation matrix  $R$  by observing each end effector's axis' projection on the base frame's axes, we can compute the following rotation matrix:

$$R = \begin{bmatrix} 0 & 0 & 1 \\ 0 & -1 & 0 \\ 1 & 0 & 0 \end{bmatrix} \quad (11)$$

For example,  $R$  states that the projection of the end effector's y axis  $y_e$  is in the opposite direction of the base frame's y axis  $y_0$ . This value validates the Predicted  $T_e^0$  in Analysis Problem 2.4 and shows that the Simulated  $T_e^0$  is incorrect.

Fortunately, as seen in the same problem, the joint positions were not affected by this error.

2. Yes, many points that would've collided with the frame of the robot are reachable in our predicted workspace. This occurred because neither

Ros, Gazebo, or ourselves took collision into account. Withstanding those points, there were still points predicted as reachable, but were not actually reachable in the simulation. This is because our calculateFk did not take joint limits into account it only utilized the  $T_e^0$ . As a result, it was possible to input joint parameters that resulted in dramatically different results than the simulation, as long as those joint inputs were outside the designated joint limits defined by ros. A way to account for this would be to include joint limits in calculateFK. The inverse is not true; we did not find any points that were reachable in the simulation that were predicted as unreachable. The main sources of deviation from simulation and prediction was caused by not having sufficient checks of joint limits in calculateFk.

## 4 Code Makeup

The code essentially implements DH4 convention to find the coordinate frames of each joint and the end effector. We use a helper function called createA.m to create an  $A_i^{i-1}$ . This function takes in DH4 parameters and outputs the corresponding  $A_i^{i-1}$  matrix. We then create another matrix that contains all the relevant DH4 parameters, and by walking through each row in the DH4 parameter matrix, inputting it into create A, and post multiplying we were able to get the intermediate frames and  $T_e^0$ . Using joint coordinates and  $T_e^0$  gave us the location of the end effector and its orientation. Using this in combination with walking through each joint to create the workspace, and getting the joint location of each joint in the base frame was simple matter of post multiplying until you got to the frame containing the desired joint location. This final matrix gave both the orientation and location of the joint. The only exception was joint 4 because the coordinate frame did not exist on the joint; as a result, we had to multiply  $T_4^0$  with the column vector  $q = [0, 0, L_4, 1]$  to translate the frame  $T_4^0$  to the location of joint 4.

## 5 Conclusion

This lab was an exercise in implementing forward kinematics, in the simulated ROS environment. Despite the the sophistication of Ros, there were noticeable deviations from reality. The main deviation was a result of colli-

sion. The Robot was able to collide with itself with absolutely no resistance. There were also cases where the robot did not directly collide with itself, but instead entered positions that most certainly would've torn wires present in the real robot. We also didn't control the speed of the robot and focused primarily on the kinematics. These considerations are important because when we eventually use a real robot we will need to consider these differences to prevent damaging physical systems.

## 6 Appendix

```
In [4]: import numpy as np
import sympy as sym
from sympy import *
from IPython.display import display
init_printing(use_latex='mathjax')
```

Function used to systematically calculate  $A_i^{i-1}$  for verifying the joint coordinates in our code

```
In [5]: def createA(DH4):
a=DH4[0]
alpha=DH4[1]
d=DH4[2]
theta=DH4[3]
row1=[cos(theta),-sin(theta)*cos(alpha),sin(theta)*sin(alpha),a*cos(theta)
)]
row2=[sin(theta),cos(theta)*cos(alpha),-cos(theta)*sin(alpha),a*sin(theta)
)]
row3=[0,sin(alpha),cos(theta),d]
row4=[0,0,0,1]
Ai_1i=Matrix([row1,row2,row3,row4])
return Ai_1i
```

Defines the link parameters for each link to check symbolically against matlab code.

```
In [6]: link1=[0,-pi/2,symbols("L{1}"),symbols("theta1")]
link2=[symbols("-L_{2}"),0,0,symbols("theta2")+pi/2]
link3=[symbols("-L_{3}"),0,0,symbols("theta3")+pi/2]
link4=[0,pi/2,0,symbols("theta4")-pi/2]
linke=[0,0,symbols("L_{4+5}"),symbols("theta5")+pi]
testlink=[symbols("a_{1}"),symbols("alpha1"),symbols("d_{1}"),symbols("theta1"
)]
```

```
In [7]: T1=createA(link1)
T2=createA(link2)
T3=createA(link3)
T4=createA(link4)
T5=createA(linke)
test=createA(testlink)
```

Example of displaying  $T_1^0$  we could enter T1,T2,etc to display the corresponding matrix

```
In [10]: display(T1)
```

$$\begin{bmatrix} \cos(\theta_1) & 0 & -\sin(\theta_1) & 0 \\ \sin(\theta_1) & 0 & \cos(\theta_1) & 0 \\ 0 & -1 & \cos(\theta_1) & L1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$



This is why analytically writing  $T_e^0$  is unfeasible. This monstrous section is just part of the first column.

In [13]: `display(T1*T2*T3*T4*T5)`

$$\left[ \begin{array}{l} -((\sin(\theta_2)\sin(\theta_3)\cos(\theta_1) - \cos(\theta_1)\cos(\theta_2)\cos(\theta_3))\sin(\theta_4) - (\sin(\theta_2)\cos(\theta_3) + \sin(\theta_1)\sin(\theta_2)\sin(\theta_3))\sin(\theta_4) \\ -((\sin(\theta_1)\sin(\theta_2)\sin(\theta_3) - \sin(\theta_1)\cos(\theta_2)\cos(\theta_3))\sin(\theta_4) - (\sin(\theta_1)\sin(\theta_2)\cos(\theta_3) + \sin(\theta_1)\sin(\theta_2)\sin(\theta_3)\cos(\theta_4)) \\ -(-(-\sin(\theta_2)\sin(\theta_3) + \cos(\theta_2)\cos(\theta_3))\cos(\theta_4) + (\sin(\theta_2)\cos(\theta_3) + \sin(\theta_2)\sin(\theta_3)\cos(\theta_4))\sin(\theta_1) \end{array} \right]$$

In [ ]: