

---

# **Robotic Manipulator Documentation**

***Release 1.0***

**Thiago Souto**

**May 14, 2020**

**CONTENTS:**

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

## FORWARD\_KINEMATICS MODULE

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

Bases: `object`

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

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

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

Returns: It returns Rotation and translation matrices.

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

**rot\_x** (alpha)

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

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

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

**rot\_z** (theta)

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

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

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

**trans\_x** (a)

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

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

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

**trans\_z** (d)

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

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

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

Forward\_Kinematics.**main**()

Assessment 02 Robotic manipulator design - Forward Kinematics.

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

## 1.1 Python Program

```

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

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

104     :type theta: string
105     :param theta: Rotation Angle around the Z axis
106
107     Returns: The Rotational Matrix at the Z axis by an *given* angle
108     """
109     self._z_angle = theta
110
111     r_z = sympy.Matrix([[sympy.cos(self._z_angle), -sympy.sin(self._z_angle), 0,
112 ↪ 0],
113                        [sympy.sin(self._z_angle), sympy.cos(self._z_angle), 0,
114 ↪ 0],
115                        [0, 0, 1, 0],
116                        [0, 0, 0, 1]])
117
118     r_z = r_z.evalf()
119
120     return r_z
121
122 # def printM(expr, num_digits):
123 #     return expr.xreplace({n.evalf(): n if type(n) == int else Float(n, num_digits)
124 ↪ for n in expr.atoms(Number)})
125
126 def printM(expr, num_digits):
127     return expr.xreplace({n.evalf(): round(n, num_digits) for n in expr.atoms(Number)
128 ↪ })
129
130 def main():
131     """
132     Assessment 02 Robotic manipulator design - Forward Kinematics.
133     """
134     a1 = ForwardKinematics()      # Rx(ai-1)
135     a2 = ForwardKinematics()      # Dx(ai-1)
136     a3 = ForwardKinematics()      # Dz(di)
137     a4 = ForwardKinematics()      # Rz(thetai)
138
139     print('Matrix t0_1:')
140     t_0_1 = (a1.rot_x('0')) * (a2.trans_x('0')) * (a3.trans_z('l1')) * (a4.rot_z(
141 ↪ 'theta_1'))
142     print(sympy.pretty(t_0_1))
143
144     print('\nMatrix t1_2:')
145     t_1_2 = (a1.rot_x('alpha_1')) * (a2.trans_x('0')) * (a3.trans_z('0')) * (a4.rot_z(
146 ↪ 'theta_2'))
147     t_1_2_subs = t_1_2.subs('alpha_1', np.deg2rad(90.00))
148     print(sympy.pretty(printM(t_1_2_subs, 3)))
149
150     print('\nMatrix t2_3:')
151     t_2_3 = (a1.rot_x('0')) * (a2.trans_x('l2')) * (a3.trans_z('0')) * (a4.rot_z(
152 ↪ 'theta_3'))
153     print(sympy.pretty(t_2_3))
154
155     print('\nMatrix t3_4:')
156     t_3_4 = (a1.rot_x('0')) * (a2.trans_x('l3')) * (a3.trans_z('0')) * (a4.rot_z(
157 ↪ 'theta_4'))
158     print(sympy.pretty(t_3_4))

```

(continues on next page)

(continued from previous page)

```

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

```

## 1.2 Output

```

Matrix t_0_1:
1.0cos(θ1)  -1.0sin(θ1)   0   0
1.0sin(θ1)   1.0cos(θ1)   0   0
      0           0       1.0  1.0l1
      0           0       0    1.0

Matrix t_1_2:
1.0cos(θ2)  -1.0sin(θ2)   0   0
      0           0       -1.0  0
1.0sin(θ2)   1.0cos(θ2)   0.0  0

```

(continues on next page)

(continued from previous page)

```

0          0          0      1.0

```

Matrix t\_2\_3:

```

1.0cos(θ3)  -1.0sin(θ3)  0      1.0l2
1.0sin(θ3)  1.0cos(θ3)  0      0
0          0          1.0      0
0          0          0      1.0

```

Matrix t\_3\_4:

```

1.0cos(θ4)  -1.0sin(θ4)  0      1.0l3
1.0sin(θ4)  1.0cos(θ4)  0      0
0          0          1.0      0
0          0          0      1.0

```

Matrix t\_4\_5:

```

1.0      0      0      1.0l4
0      1.0      0      0
0      0      1.0      0
0      0      0      1.0

```

Matrix T\_0\_5: with substitutions Round

```

1.0cos(θ1)cos(θ2 + θ3 + θ4)  -1.0sin(θ2 + θ3 + θ4)cos(θ1)  1.0sin(θ1)  1.0(l2cos(θ2) +
↪ l3cos(θ2 + θ3) + l4cos(θ2 + θ3 + θ4))cos(θ1)
↪
1.0sin(θ1)cos(θ2 + θ3 + θ4)  -1.0sin(θ1)sin(θ2 + θ3 + θ4)  -1.0cos(θ1)  1.0(l2cos(θ2) +
↪ l3cos(θ2 + θ3) + l4cos(θ2 + θ3 + θ4))sin(θ1)
↪
1.0sin(θ2 + θ3 + θ4)  1.0cos(θ2 + θ3 + θ4)  0  1.0l1 + 1.
↪ 0l2sin(θ2) + 1.0l3sin(θ2 + θ3) + 1.0l4sin(θ2 + θ3 + θ4)
↪
↪          0          0          0
↪          1.0          0          0

```

Matrix T\_0\_5: with substitutions Round

(continues on next page)



(continued from previous page)

```

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

↪

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

↪

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

↪

0 0 0 0
↪

1.0

```

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

```

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

↪

0 0 -1.0 0
↪

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

↪

0 0 0 1.
↪ 0

```

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

```

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

↪

0 0 -1.0 0
↪

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

↪

0 0 0 1.
↪ 0

```

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

```

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

```

(continues on next page)

(continued from previous page)

```

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

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

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

```

## INVERSE\_KINEMATICS MODULE

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

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

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

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

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

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

### 2.1 Python Program

```

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

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)

(continued from previous page)

```

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

```

## 2.2 Output

```

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

```

(continues on next page)

(continued from previous page)

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

(continues on next page)

(continued from previous page)

```

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

```

(continues on next page)



(continued from previous page)

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

## INDICES AND TABLES

At the website you can navigate through the menus below:

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

### 3.1 Running the documentation with Sphinx

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

Install Spinx:

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

Install the “Read the Docs” theme:

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

**`make clean`**

**`make html`**

### 3.2 GitHub Repository

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

## PYTHON MODULE INDEX

### f

Forward\_Kinematics, 1

### i

Inverse\_Kinematics, 9