Robotic Manipulator Documentation

Release 1.0

Thiago Souto

May 14, 2020

CONTENTS:

1 Forward_Kinematics module 1
- 1.1 Python Program 2
- 1.2 Output 5

2 Inverse_Kinematics module 9
- 2.1 Python Program 9
- 2.2 Output 12

3 Indices and tables 16
- 3.1 Running the documentation with Sphinx 16
- 3.2 GitHub Repository 16

Python Module Index 17

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

```
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)

```python

    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:

$$\begin{matrix} 1.0\cos(\theta_1) & -1.0\sin(\theta_1) & 0 & 0 \\ 1.0\sin(\theta_1) & 1.0\cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1.0 & 1.0l_1 \\ 0 & 0 & 0 & 1.0 \end{matrix}$$

Matrix t_1_2:

$$\begin{matrix} 1.0\cos(\theta_2) & -1.0\sin(\theta_2) & 0 & 0 \\ 0 & 0 & -1.0 & 0 \\ 1.0\sin(\theta_2) & 1.0\cos(\theta_2) & 0.0 & 0 \end{matrix}$$

(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

`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

f

Forward_Kinematics, 1

i

Inverse_Kinematics, 9