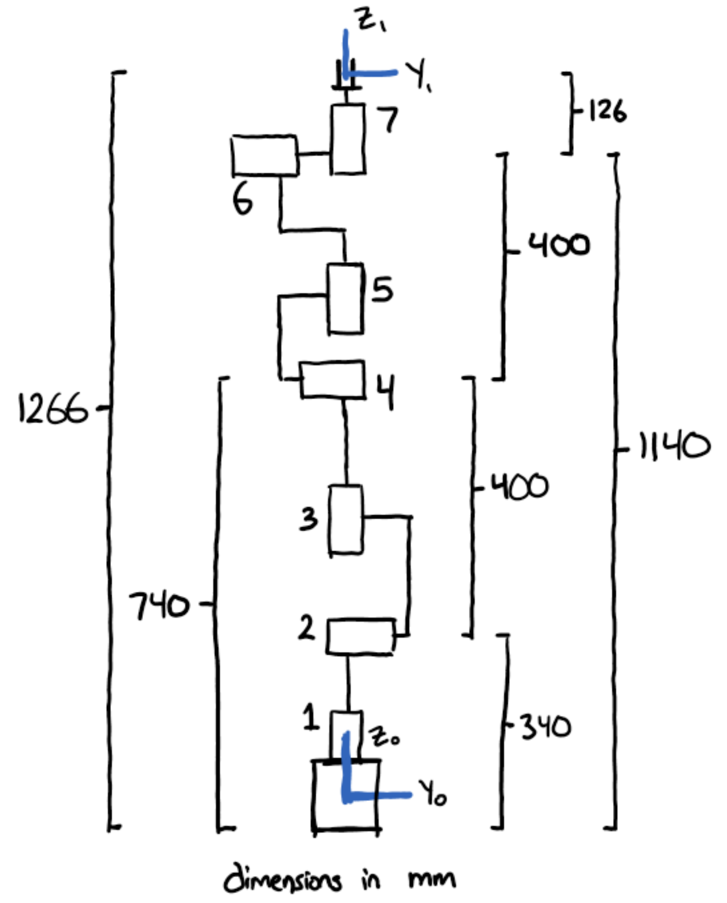
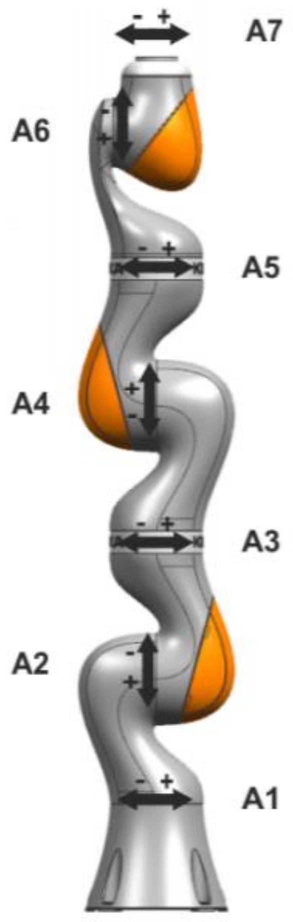
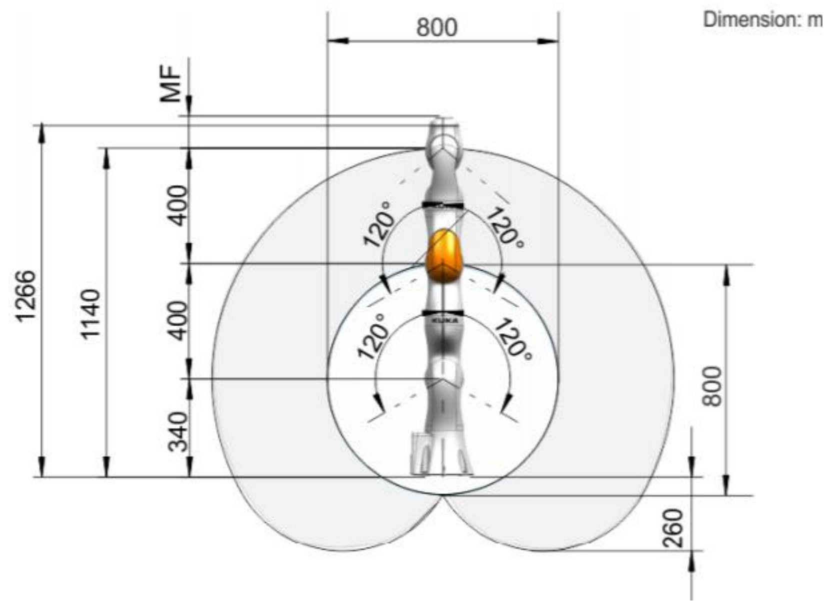
Schematic and Derivation of Forward Kinematics

Drawn Schematic + References







Code used to derive and implement forward kinematics:

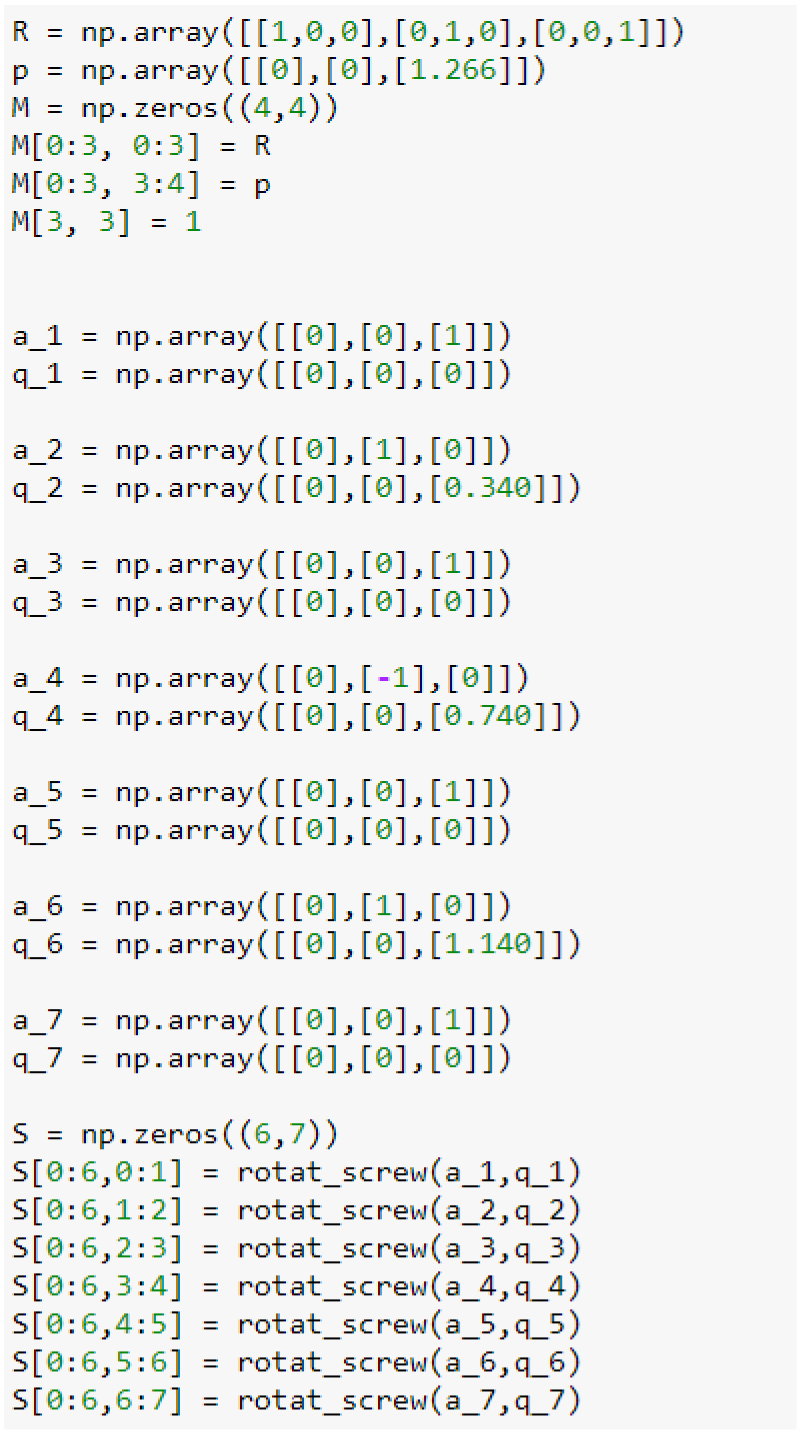
Schematic and Derivation of Forward Kinematics

Code header with helper functions

```
import numpy as np
import scipy.linalg as syl

def skew_sym(arr):
    return np.array([[0,-arr[2],arr[1]],[arr[2],0,-arr[0]],[-arr[1],arr[0],0]])

def matrix_rep(arr):
    matrix = np.zeros((4,4))
    matrix[0:3,0:3] = skew_sym(arr[0:3,0:1])
    matrix[0:3,3:4] = arr[3:6,0:1]
    return matrix

def prism_screw(a,q):
    S = np.zeros((6,1))
    S[3:6,0:1] = a
    return S

def rotat_screw(a,q):
    S = np.zeros((6,1))
    S[0:3,0:1] = a
    S[3:6,0:1] = -skew_sym(a) @ q
    return S
```

Schematic and Derivation of Forward Kinematics

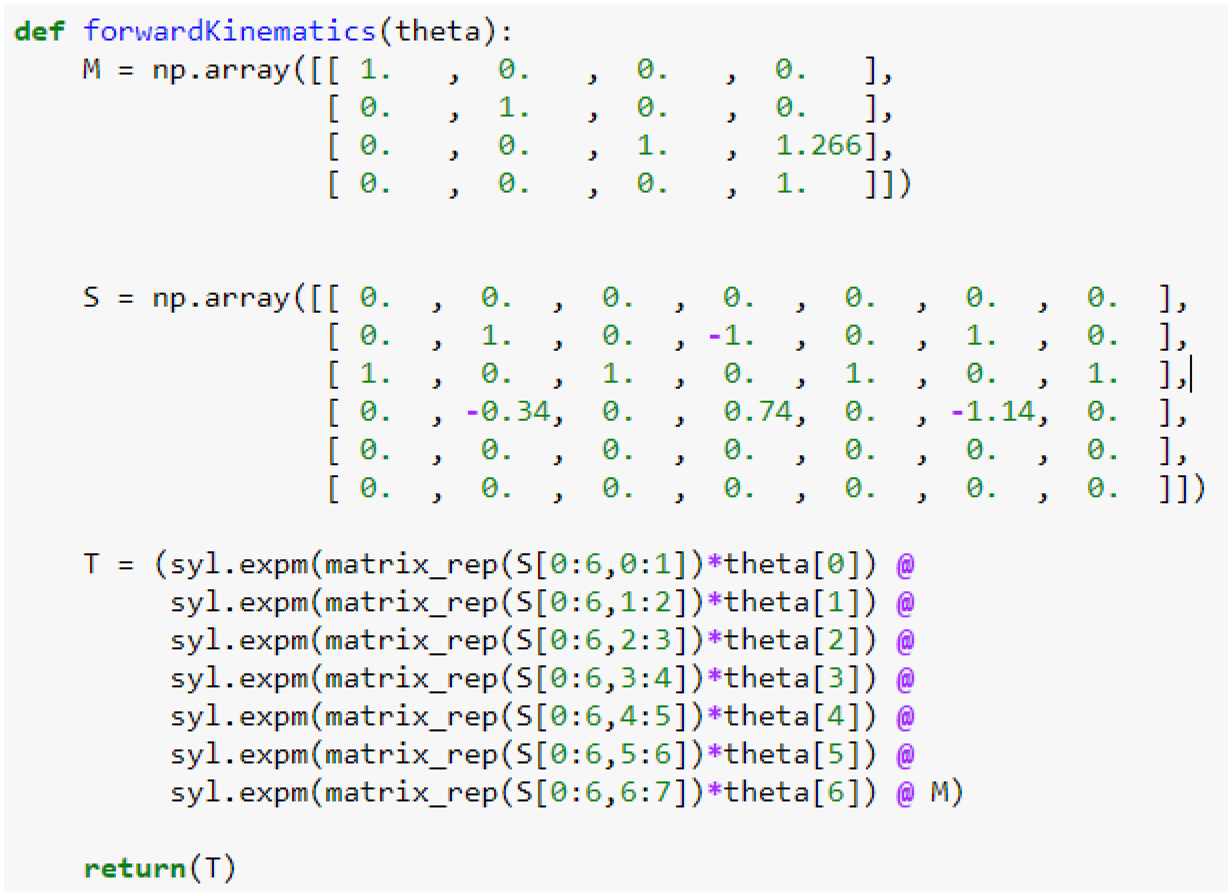
Code used to derive screw and initial pose



```
R = np.array([[1,0,0],[0,1,0],[0,0,1]])
p = np.array([[0],[0],[1.266]])
M = np.zeros((4,4))
M[0:3, 0:3] = R
M[0:3, 3:4] = p
M[3, 3] = 1


a_1 = np.array([[0],[0],[1]])
q_1 = np.array([[0],[0],[0]])

a_2 = np.array([[0],[1],[0]])
q_2 = np.array([[0],[0],[0.340]])

a_3 = np.array([[0],[0],[1]])
q_3 = np.array([[0],[0],[0]])

a_4 = np.array([[0],[-1],[0]])
q_4 = np.array([[0],[0],[0.740]])

a_5 = np.array([[0],[0],[1]])
q_5 = np.array([[0],[0],[0]])

a_6 = np.array([[0],[1],[0]])
q_6 = np.array([[0],[0],[1.140]])

a_7 = np.array([[0],[0],[1]])
q_7 = np.array([[0],[0],[0]])

S = np.zeros((6,7))
S[0:6,0:1] = rotat_screw(a_1,q_1)
S[0:6,1:2] = rotat_screw(a_2,q_2)
S[0:6,2:3] = rotat_screw(a_3,q_3)
S[0:6,3:4] = rotat_screw(a_4,q_4)
S[0:6,4:5] = rotat_screw(a_5,q_5)
S[0:6,5:6] = rotat_screw(a_6,q_6)
S[0:6,6:7] = rotat_screw(a_7,q_7)
```

Schematic and Derivation of Forward Kinematics

Code used to predict end pose based off of calculated screw and initial pose, as well as given theta joint angles



```
def forwardKinematics(theta):
    M = np.array([[ 1.    ,  0.    ,  0.    ,  0.    ],
                  [ 0.    ,  1.    ,  0.    ,  0.    ],
                  [ 0.    ,  0.    ,  1.    ,  1.266],
                  [ 0.    ,  0.    ,  0.    ,  1.    ]])


    S = np.array([[ 0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ],
                  [ 0.   ,  1.   ,  0.   , -1.   ,  0.   ,  1.   ,  0.   ],
                  [ 1.   ,  0.   ,  1.   ,  0.   ,  1.   ,  0.   ,  1.   ],
                  [ 0.   , -0.34,  0.   ,  0.74,  0.   , -1.14,  0.   ],
                  [ 0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ],
                  [ 0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ,  0.   ]])

    T = (syl.expm(matrix_rep(S[0:6,0:1])*theta[0]) @
         syl.expm(matrix_rep(S[0:6,1:2])*theta[1]) @
         syl.expm(matrix_rep(S[0:6,2:3])*theta[2]) @
         syl.expm(matrix_rep(S[0:6,3:4])*theta[3]) @
         syl.expm(matrix_rep(S[0:6,4:5])*theta[4]) @
         syl.expm(matrix_rep(S[0:6,5:6])*theta[5]) @
         syl.expm(matrix_rep(S[0:6,6:7])*theta[6]) @ M)

    return(T)
```