Contents

SOLVE USING INVERSE KINEMATICS

```
l_1 = 4;
l_2 = 3;
l_3 = 2;

sols = {};

disp('Solve for joint angles using inverse kinematics:');

T_0_H1 = [1 0 0 9; 0 1 0 0; 0 0 1 0; 0 0 0 1];
sol1 = inverse_kinematics(T_0_H1, l_1, l_2, l_3);
disp('Part i joint angles:');
disp(sol1);

T_0_H2 = [0.5 -.866 0 7.5373; 0.866 0.5 0 3.9266; 0 0 1 0; 0 0 0 1];
sol2 = inverse_kinematics(T_0_H2, l_1, l_2, l_3);
disp('Part ii joint angles:');
disp(sol2);

T_0_H3 = [0 1 0 -3; -1 0 0 2; 0 0 1 0; 0 0 0 1];
sol3 = inverse_kinematics(T_0_H3, l_1, l_2, l_3);
disp('Part iii joint angles:');
disp(sol3);

disp('The fourth transformation matrix throws an error. This makes sense, given that we are requesting a y position longer than the arm can reach (9m)')
T_0_H4 = [0.866 0.5 0 -3.1245; -0.5 0.866 0 9.1674; 0 0 1 0; 0 0 0 1];
% sol4 = inverse_kinematics(T_0_H, l_1, l_2, l_3);
% disp(sol4);
```

```
Solve for joint angles using inverse kinematics:
Part i joint angles:
    {[0 0 0]}    {[0 0 0]}

Part ii joint angles:
    {[0.1745 0.3491 0.5236]}    {[0.4732 -0.3491 0.9230]}

Part iii joint angles:
    {[1.5708 1.5708 -4.7124]}    {[2.8578 -1.5708 -2.8578]}

The fourth transformation matrix throws an error. This makes sense, given that we are requesting a y position longer than the arm can reach (9m)
```

CHECK WITH FORWARD KINEMATICS

```
sols=[sol1, sol2, sol3];
T_3_H = [1 0 0 l_3; 0 1 0 0; 0 0 1 0; 0 0 0 1];

[r, c] = size(sols);

for i=1:c
    t_1_deg = sols{i}(1) * 180/pi;
    t_2_deg = sols{i}(2) * 180/pi;
    t_3_deg = sols{i}(3) * 180/pi;

    % Plug into forward kinematic equation
    dh_table = [0 0 0 t_1_deg; l_1 0 0 t_2_deg; l_2 0 0 t_3_deg];
    result = find_T_total(dh_table, false);

    T_0_H_fkine = result * T_3_H;

    disp('Forward kinematics check to match problem statement')
    fprintf('*Using t1 = %3.4f, t2 = %3.4f, t3 = %3.4f, T is:\n', sols{i}(1), sols{i}(2), sols{i}(3));
    disp(T_0_H_fkine)
end
```

```
Forward kinematics check to match problem statement
*Using t1 = 0.0000, t2 = 0.0000, t3 = 0.0000, T is:
     1     0     0     9
     0     1     0     0
     0     0     1     0
     0     0     0     1
```

```
Forward kinematics check to match problem statement
*Using t1 = 0.0000, t2 = -0.0000, t3 = 0.0000, T is:
     1     0     0     9
     0     1     0     0
     0     0     1     0
     0     0     0     1

Forward kinematics check to match problem statement
*Using t1 = 0.1745, t2 = 0.3491, t3 = 0.5236, T is:
    0.5000   -0.8660         0    7.5373
    0.8660    0.5000         0    3.9266
         0         0    1.0000         0
         0         0         0    1.0000

Forward kinematics check to match problem statement
*Using t1 = 0.4732, t2 = -0.3491, t3 = 0.9230, T is:
    0.5000   -0.8660         0    7.5373
    0.8660    0.5000         0    3.9266
         0         0    1.0000         0
         0         0         0    1.0000

Forward kinematics check to match problem statement
*Using t1 = 1.5708, t2 = 1.5708, t3 = -4.7124, T is:
     0     1     0    -3
    -1     0     0     2
     0     0     1     0
     0     0     0     1

Forward kinematics check to match problem statement
*Using t1 = 2.8578, t2 = -1.5708, t3 = -2.8578, T is:
         0    1.0000         0   -3.0000
   -1.0000         0         0    2.0000
         0         0    1.0000         0
         0         0         0    1.0000
```

CHECK WITH CORKE ROBOTICS TOOLBOX

```
% 0 is revolute joint, 1 is prismatic
% I used the standard definition of DH parameters in my solution, but this
% is just semantics - the inverse kinematic solution still works
% THETA D A ALPHA SIGMA OFFSET
Link1 = Link([0, 0, l_1, 0], 'standard');
Link2 = Link([0, 0, l_2, 0], 'standard');
Link3 = Link([0, 0, l_3, 0], 'standard');

robot = SerialLink([Link1 Link2 Link3], 'name', 'Matlab2 Robot');

disp('Check with Corke Toolbox:');
disp('Part i joint angles:');
disp(robot.ikine(T_0_H1, [0 0 0], 'mask', [1 1 0 0 0 1]));
disp('Part ii joint angles');
disp(robot.ikine(T_0_H2, [0 0 0], 'mask', [1 1 0 0 0 1]));
disp('Part iii joint angles');
disp(robot.ikine(T_0_H3, [0 0 0], 'mask', [1 1 0 0 0 1]));

disp('These match the solution above');
```

```
Check with Corke Toolbox:
Part i joint angles:
     0     0     0

Part ii joint angles
    0.1745    0.3491    0.5236

Part iii joint angles
    2.8578   -1.5708   -2.8578

These match the solution above
```

Published with MATLAB® R2021a