MAE 204 Lab 3 Project

Dihan Liu, Kaiyu Liu, Lingyi Wei, Meng-Chin Chiang

Overview

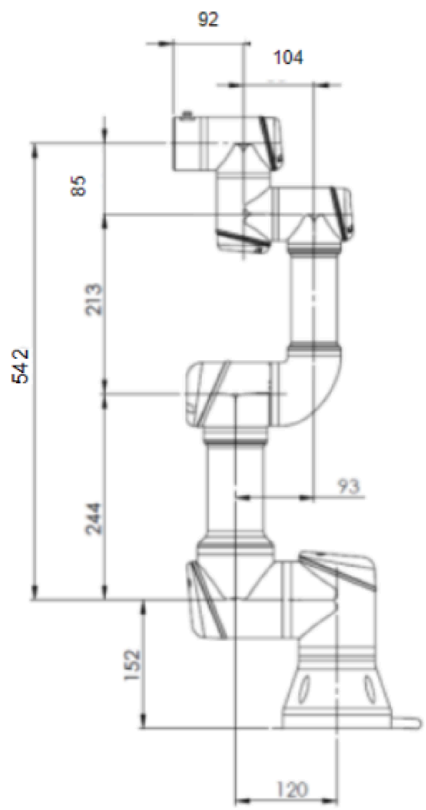
The purpose of the third lab is to write a script that utilizes inverse kinematics using Newton-Raphson iterative algorithm to find joint angles of each waypoint for UR3e robot assembling three parts of the Effigy.

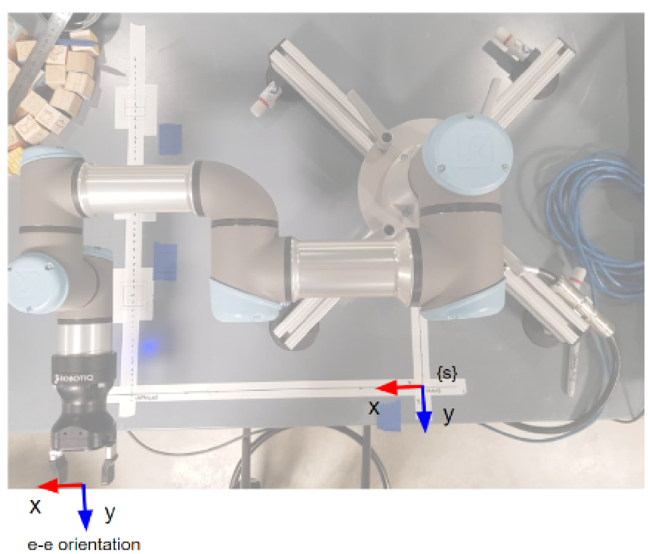
(a) **Initialization**: Given $x_d \in \mathbb{R}^m$ and an initial guess $\theta^0 \in \mathbb{R}^n$, set $i = 0$.

(b) Set $e = x_d - f(\theta^i)$. While $\|e\| > \epsilon$ for some small ϵ:

- Set $\theta^{i+1} = \theta^i + J^{\dagger}(\theta^i)e$.
- Increment i.

Results





Notice: The length of the gripper is 155 mm

Screw Axes

S1 = [0, 0, 1, -300, 0, 0]';

S2 = [0, 1, 0, -152-88, 0, 0]';

S3 = [0, 1, 0, -152-88, 0, 244]';

```
S4 = [0, 1, 0, -152-88, 0, 244+213]';
S5 = [0, 0, -1, 300-120+93-104, 244+213, 0]';
S6 = [0, 1, 0, -152-88+85, 0, 244+213]';
Slist = [S1, S2, S3, S4, S5, S6];
```

Transformation matrix

```
M = [1 0 0 244+213;
    0 1 0 -(300-120+93-104-92-78);
    0 0 1 152+88-85;
    0 0 0 1];
```

Waypoints for operation sequence

```
standby = [0, 0, 1, 323.6; -1, 0, 0, -335.6; 0, -1, 0, 237; 0, 0, 0, 1];
standby1 = [1, 0, 0, 400; 0, 0, 1, -200; 0, -1, 0, 150; 0, 0, 0, 1];
legs_grab = [1, 0, 0, 400; 0, 0, 1, -200; 0, -1, 0, 50; 0, 0, 0, 1];
legs_release = [1, 0, 0, 450; 0, 0, 1, -150; 0, -1, 0, 55; 0, 0, 0, 1];
body_grab = [0, 0, 1, 450; -1, 0, 0, -450; 0, -1, 0, 70; 0, 0, 0, 1];
body_release = [1, 0, 0, 450; 0, 0, 1, -150; 0, -1, 0, 85; 0, 0, 0, 1];
head_grab = [0, 0, 1, 450; -1, 0, 0, -300; 0, -1, 0, 130; 0, 0, 0, 1];
head_release =[1, 0, 0, 450; 0, 0, 1, -150; 0, -1, 0, 110; 0, 0, 0, 1];
% {standby1} is the safe point between grabbing legs and releasing legs
```

```
Tlist =
[{standby};{legs_grab};{standby1};{legs_release};{standby};{body_grab};{standby};{body
_release};{standby};{head_grab};{standby};{head_release};{standby}];
```

Gripper state

```
gripperstate = [0,1,1,0,0,1,1,0,0,1,1,0,0]';
% 0 for open and 1 for close
```

Joint angles

```
  -30.0059  -89.9924   89.9924  -90.0000  -90.0000  149.9941
   -4.4890  -53.5825   94.4433 -130.8609  -90.0000   85.5110
   -4.4890  -65.0646   82.6225 -107.5579  -90.0000   85.5110
    2.4031  -41.5198   69.4421 -117.9223  -90.0000   92.4031
  -30.0059  -89.9924   89.9924  -90.0000  -90.0000  149.9941
  -34.4668  -43.2030   68.2093 -115.0062  -90.0000  145.5332
  -30.0059  -89.9924   89.9924  -90.0000  -90.0000  149.9941
    2.4031  -44.7231   66.6961 -111.9730  -90.0000   92.4031
  -30.0059  -89.9924   89.9924  -90.0000  -90.0000  149.9941
```

-16.9245	-54.5157	71.3492	-106.8335	-90.0000	163.0755
-30.0059	-89.9924	89.9924	-90.0000	-90.0000	149.9941
2.4031	-46.8638	63.5206	-106.6569	-90.0000	92.4031
-30.0059	-89.9924	89.9924	-90.0000	-90.0000	149.9941

Video Link
https://youtu.be/Leb1e4IefcQ?si=SxmRG1MQV1oW0UNv

Summary
1.Our method of planning robot trajectories involves adding a safety point between each target location to avoid collisions along the path. Initially, we used stand by point as the safe location between all target positions because it is the starting position and theoretically should not collide with any path. However, during experiments, we discovered that this stand by point still resulted in collisions when moving from grabbing legs to the leg release position. This was because we did not account for the increased length at the end-effector after picking up an item. Therefore, we established a new position, "stand by1," to circumvent this specific collision scenario.

2.The initial guess theta0 should be close to the desired theta. A closer initial guess to the desired theta can significantly reduce the computational time and increase the likelihood of the algorithm converging to a correct solution. We can use the position from the previous movement as the initial position for the next iteration. We found that using either the previous movement position or the standby point as theta0 can lead to accurate results through iteration, but the number of iterations required differs. When the position from the last movement is closer to the next target than the standby position, the number of iterations needed is reduced.

3.The input angle and output angle should be in degrees. We need to be aware that MATLAB uses radians, whereas the robot operates in degrees. Using radians during experiments resulted in the robot barely moving, a phenomenon that occurred during our first experiment.

🎉

```
%% Lab 3 (Inverse Kinematics) template
% MAE204
% Harry
%
% Computes inverse kinematics for each waypoints in the sequence, then
% outputs the joint angle sets as well as gripper state as waypoint_array.csv file
% waypoint_array.csv will be saved in Matlab's current directory

clc
clear
close all

%% Part 1: Establishing screw axes S and end-effector zero config M
% First, define the screw axes, in (mm)

S1 = [0, 0, 1, -300, 0, 0]';

% Your code should begin here

S2 = [0, 1, 0, -152-88, 0, 0]';
S3 = [0, 1, 0, -152-88, 0, 244]';
S4 = [0, 1, 0, -152-88, 0, 244+213]';
S5 = [0, 0, -1, 300-120+93-104, 244+213, 0]';
S6 = [0, 1, 0, -152-88+85, 0, 244+213]';
Slist = [S1, S2, S3, S4, S5, S6];

% Next, define the M matrix (the zero-position e-e transformation matrix),
% in (mm)

M = [1 0 0 244+213;
     0 1 0 -(300-120+93-104-92)+155;
     0 0 1 152+88-85;
     0 0 0 1];


%% Part 2: UR3e sequence planning
% You may use this space to define the waypoints for your sequence (I
% recommend using SE(3) matrices to define gripper configurations)

standby = [0, 0, 1, 323.6; -1, 0, 0, -335.6; 0, -1, 0, 237; 0, 0, 0, 1];
standby1 = [1, 0, 0, 400; 0, 0, 1, -200; 0, -1, 0, 150; 0, 0, 0, 1];
legs_grab = [1, 0, 0, 400; 0, 0, 1, -200; 0, -1, 0, 50; 0, 0, 0, 1];
legs_release = [1, 0, 0, 450; 0, 0, 1, -150; 0, -1, 0, 55; 0, 0, 0, 1];
body_grab = [0, 0, 1, 450; -1, 0, 0, -450; 0, -1, 0, 70; 0, 0, 0, 1];
body_release = [1, 0, 0, 450; 0, 0, 1, -150; 0, -1, 0, 85; 0, 0, 0, 1];
head_grab = [0, 0, 1, 450; -1, 0, 0, -300; 0, -1, 0, 130; 0, 0, 0, 1];
head_release =[1, 0, 0, 450; 0, 0, 1, -150; 0, -1, 0, 110; 0, 0, 0, 1];

Tlist = [{standby};{legs_grab};{standby1};{legs_release};{standby};{body_grab};{standby}; ↙
{body_release};{standby};{head_grab};{standby};{head_release};{standby}];
```

```
%% Part 3: Inverse kinematics for each waypoint
% Compute inverse kinematics to obtain 6 joint angles for each waypoint,
% then save them in waypoint_array
%
% waypoint_array = n x 7 array where:
% n = number of waypoints
% First 6 columns in each row = joint angles 1...6, in degrees
% Last column in each row = gripper state (0 for open, 1 for close)
eomg = 0.0001;
ev = 0.001;
gripperstate = [0,1,1,0,0,1,1,0,0,1,1,0,0]';
% waypoint_array = zeros(7);

n =  1;
thetalist0 = deg2rad([-30,-90,90,-90,-90,150]')
thetalist = thetalist0;

eomg=0.01;
ev=0.001;
for i=1:13
[thetalist,success]= IKinSpace(Slist,M,cell2mat(Tlist(i)),thetalist0,eomg,ev);
theta(i,:)=transpose(thetalist);
end
thetalist = theta;
waypoint_array =[thetalist,gripperstate];

% waypoint_array
waypoint_array(:,1:6) = rad2deg(waypoint_array(:,1:6));
waypoint_array
% Your code should end here

%% Some basic sanity checks (DO NOT EDIT THIS PART)
% size of waypoint_array check
if length(waypoint_array(1,:)) ~= 7
    error('waypoint_array should have 7 columns')
end

for i = 1:length(waypoint_array(:,1))
    for j = 1:5
        % Joint limit check (error if out of joint limit bounds)
        if waypoint_array(i,j) > 360 || waypoint_array(i,j) < -360
            error(['Error: joint ',num2str(j),' in waypoint number ',num2str(i),' is out of joint limit ↙
bounds']);
        end
        % Gripper state check (error if not 0 or 1)
        if waypoint_array(i,7) ~= 0 && waypoint_array(i,7) ~= 1
            error(['Error: gripper state in waypoint number ',num2str(i),' is invalid. It should be 0↙
or 1']);
        end
    end
```

```
end

%% Output array to waypoint_array.csv
% waypoint_array.csv will be located in Matlab's current directory
writematrix(waypoint_array,'waypoint_array.csv')
```

```
function [thetalist, success] ...
         = IKinSpace(Slist, M, T, thetalist0, eomg, ev)
thetalist = thetalist0;
i = 0;
maxiterations = 20;
Tsb = FKinSpace(M, Slist, thetalist);
Vs = Adjoint(Tsb) * se3ToVec(MatrixLog6(TransInv(Tsb) * T));
err = norm(Vs(1: 3)) > eomg || norm(Vs(4: 6)) > ev;
while err && i < maxiterations
    thetalist = thetalist + pinv(JacobianSpace(Slist, thetalist)) * Vs;
    i = i + 1;
    Tsb = FKinSpace(M, Slist, thetalist);
    Vs = Adjoint(Tsb) * se3ToVec(MatrixLog6(TransInv(Tsb) * T));
    err = norm(Vs(1: 3)) > eomg || norm(Vs(4: 6)) > ev;
end
success = ~ err;
end
```