THA Programming Assignment #4

ME 397 ASBR, Sp 23, Dr. Farshid Alambeigi

Jared Rosenbaum and Steven Swanbeck

**PA 1)**

*Problem Description:*

For this assignment, the robotic manipulator previously considered in THA2 will be used, along with an attached tool, to implement a constrained optimization-based control algorithm that keeps the tool-tip within a certain distance of the goal tool-tip position and obeys the manipulator’s joint limits.



*Fig. 1: Franka Panda manipulator with attached tool.*

*Method for Solution:*

This algorithm is formulated as a constrained optimization problem of the form:

Subject to

For this problem, the manipulator joint angle update, , is formulated as:

subject to the equality constraints

,

the inequality constraint

where the tolerance is a radial distance within which the solution must place the tool tip, and the upper and lower bounds defined by

where and are the lower and upper joint limits of the manipulator, respectively.

The objective function can be manipulated into the form by making the substitutions of and and isolating the terms with from those without. Doing so, and applying the skew-symmetric matrix and reversible properties of cross products to isolate yields:

The matrix *A* that partially describes the inequality constraints can be solved in a similar manner, where

The vector *b* is more challenging, though, as the addition of the tolerance outside of the two-norm operation complicates its solution. As an approximation of this constraint, we solved *b* as

where

such that this additional component is a uniform vector with magnitude equal to the provided radial tolerance. Note that, because we have a spherical tolerance about the goal position, this vector should be represented by all possible vectors with magnitude equal to the tolerance. However, we struggled to find a way to represent this, so settled on the above formulation. The joint limits of our robot, the Franka Emika Panda, were found using the same online resource previously used to generate our robot model [1]. It is worth noting that the presented formulations of the *C* matrix and *d* vector are redundant with the radial tolerance constraint. As such, alternative formulations of *C* and *d* as

were also considered to strictly minimize the required changes in joint positions, . The results obtained using this formulation were less consistent than the previously presented formulation, so most of our results were produced using the original formulation.

These components were used with MATLAB’s *lsqlin( )* least-squares linear constrained optimization function to find the that minimized the objective function subject to the above constraints and limits, as shown in the following section.

*Explanation of Program:*

The primary code snippet used for this problem is shown below.

clc; clear; format compact; clf; close all;  
  
 [M, theta0, S\_mat, B\_mat, M\_intermediates, joint\_limits] = instantiate\_robot("franka");  
 % p\_goal = [0.45 0.15 0.6]';  
 % p\_goal = [0.4 0 0.5]';  
 % p\_goal = [0.3 -0.2 0.5]';  
 % p\_goal = [0.7 -0.2 0.5]';  
 % p\_goal = [0.3 0.4 0.9]';  
 p\_goal = [0.3 0.2 0]';  
 p\_tip = [0 0 0.1]';  
 tol = 3e-3; % m  
  
 [thetas, iteration] = IK\_constrained(M, S\_mat, p\_tip, theta0, joint\_limits, p\_goal, tol);  
  
 [~] = FK\_space(M, S\_mat, theta0, false, true, M\_intermediates);  
 [F\_space2, ~] = FK\_space(M, S\_mat, thetas, false, true, M\_intermediates, false);  
  
 [x1, y1, z1] = sphere();  
 r = 0.003;  
 x2 = x1 \* r;  
 y2 = y1 \* r;  
 z2 = z1 \* r;  
 surf(x2 + p\_goal(1), y2 + p\_goal(2), z2 + p\_goal(3), 'EdgeColor','none', 'FaceColor','m', 'FaceAlpha', 0.3)  
  
 animate\_joint\_goals("franka", [theta0' thetas'], true, "THA4", false, 30);

This code uses several of the functions created for THA2. It begins by instantiating our robot with all its variables using a slightly modified version of the *instantiate\_robot( )* function that now returns the joint limits necessary for the constrained optimization. A new function, *IK\_constrained( )*, is then called, which is shown below.

function [thetas, i] = IK\_constrained(M, S\_mat, p\_tip, thetas, joint\_limits, p\_goal, tol, iterations)  
 %Accepts a goal position, tool tip position expressed in end-effector  
 %frame, list of joint angles, M matrix of robot, matrix of screw axis  
 %column vectors corresponding to the robot, spherical radius tolerance from  
 %goal, and number of iterations and returns the set of joint angles that  
 %satisfy the constraints and the iteration of convergence.  
 % Detailed explanation goes here  
 if nargin < 8  
 iterations = 100;  
 end  
 p\_t = ones(3, 1);  
 coords = [];  
 i = 1;  
 while norm(p\_t - p\_goal) > tol  
 J = SpaceJacobian(S\_mat, thetas);  
 [F\_space, ~] = FK\_space(M, S\_mat, thetas);  
  
 t4 = F\_space \* [p\_tip; 1];  
 t = t4(1:3);  
  
 A = -Axis2SkewSymmetricMatrix(t) \* J(1:3, :) + J(4:6, :);  
 tol\_vec = sqrt(tol^2 / 3) \* ones(3,1);  
 b = p\_goal - t + sqrt(tol\_vec);  
  
 lb = joint\_limits(:,1) - thetas';  
 ub = joint\_limits(:,2) - thetas';  
  
 % C = eye(7);  
 % d = zeros(7, 1);  
 C = -Axis2SkewSymmetricMatrix(t) \* J(1:3, :) + J(4:6, :);  
 d = p\_goal - t;  
  
 x = lsqlin(C, d, A, b, [], [], lb, ub);  
 thetas = thetas + x';  
  
 [F\_space2, ~] = FK\_space(M, S\_mat, thetas + x');  
 new\_p\_tip = F\_space2 \* [p\_tip; 1];  
 p\_t = new\_p\_tip(1:3);  
  
 coords = [coords p\_t];  
 if i > iterations  
 break  
 end  
 i = i + 1;  
 end  
 end

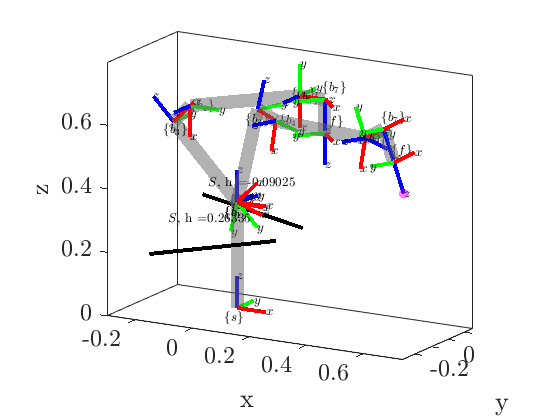
This function iteratively performs the constrained control algorithm to generate a set of joint values that place the tool tip at the desired location within a given tolerance while obeying the joint limits of the manipulator. We found that allowing the algorithm to iterate a few times produced more desirable results that better aligned with the constraints than we achieved on a single iteration. This is most likely because the objective function and inequality constraint as they are formulated are dependent on the small transformation approximation, which, depending on the distance between the current and goal states, may not be a good approximation. Therefore, we must iterate until the approximation is close to reality, at which point the algorithm can produce an optimal set of joint updates.

The above function makes use of the THA2 function *SpaceJacobian( )* and calculates all the inputs to *lsqlin( )* as described previously. Note that in combination with the definition of the inequality constraint vector, there is an additional check within the function that verifies that the two-norm of the deviance vector of the tool tip at the current iteration is within the tolerance before exiting. There is also a maximum number of iterations before which the function will return if it has not converged to the constraints.

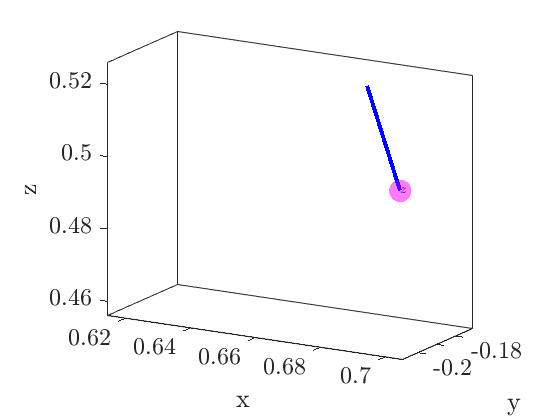
Returning to the main script, the THA2 *FK\_space( )* function is called with both the original joint positions and the returned set of joint positions to visualize the results. A sphere with radius equal to the tolerance and centered about the goal position is also plotted to help verify the results. Finally, the *animate\_joint\_goals( )* function, also from THA2, is called to visualize the motion of the joint updates.

*Answer / Test Cases:*

To verify the functionality of this approach, several test cases were created. The first is for the goal position pgoal = [0.7 -0.2 0.5] for which the joint positions [-0.1573 0.2563 -0.1289 -1.4729 -0.0238 2.2256 -0.0000] were calculated. Note that for the following examples, in the first image for each case, the visual radius of the tolerance sphere about the goal position has been inflated by a factor of 5 for visibility, then shown to true scale in the zoomed second image.

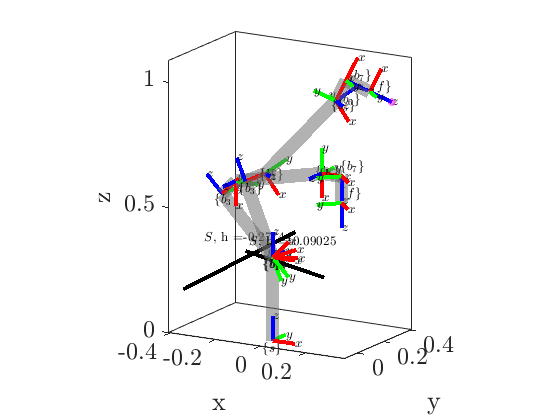


*Fig. 2: Comparison of the robot in its starting ‘ready’ position and its calculated joint state to achieve the desired tool tip position, shown in the the magenta sphere here scaled visually by a factor of 5 to improve visibility.*

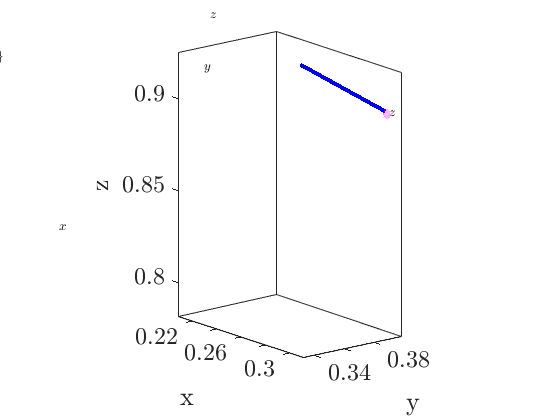


*Fig. 3: Zoomed-in view of tool tip within true scale tolerance sphere with 3mm radius, showing this requirement has been achieved.*

The second test case used the goal position for which the joint positions [0.3248 -0.3478 0.4287 -1.3508 0.1255 2.1145 -0.0000] were calculated.

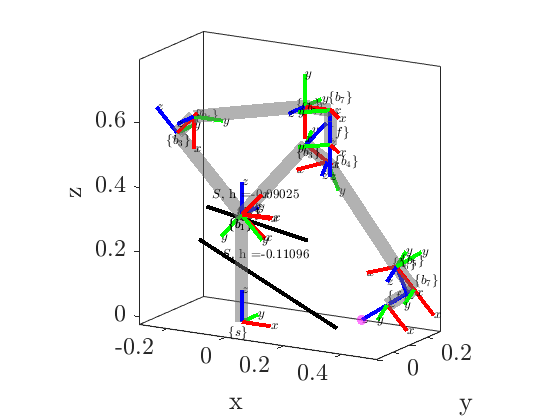


*Fig. 4: Comparison of the robot in its starting ‘ready’ position and its calculated joint state to achieve the desired tool tip position, shown in the the magenta sphere here scaled visually by a factor of 5 to improve visibility.*

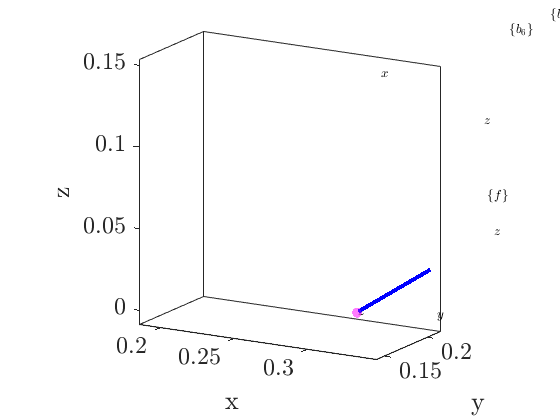


*Fig. 5: Zoomed-in view of tool tip within true scale tolerance sphere with 3mm radius, showing this requirement has been achieved.*

The second test case used the goal position pgoal = [0.3 0.2 0], for which the joint positions [0.1336 0.7208 0.2842 -2.1212 0.1303 1.8770 0] were calculated.



*Fig. 6: Comparison of the robot in its starting ‘ready’ position and its calculated joint state to achieve the desired tool tip position, shown in the the magenta sphere here scaled visually by a factor of 5 to improve visibility.*



*Fig. 7: Zoomed-in view of tool tip within true scale tolerance sphere with 3mm radius, showing this requirement has been achieved.*

In each of these test cases, plus all the others noted within the code shown, the algorithm successfully achieved the desired tool tip position within the set tolerance of 3mm. The joint limits of the robot were also all respected, and the algorithm produced an update to the joint positions that minimized the kinetic energy of the robot required to achieve the desired tool tip position.

Other Relevant Code

The only code not previously shown that has been updated since THA2 to work for this problem is *instantiate\_robot()* featuring the joint limits taken from [1], which is partially shown below.

function [M, thetas, S\_mat, B\_mat, M\_intermediates, joint\_limits] = instantiate\_robot(robot\_name, is\_symbolic)  
 if nargin < 1  
 robot\_name = "franka";  
 end  
 if nargin < 2  
 is\_symbolic = false;  
 end  
  
 if robot\_name == "franka"  
 if is\_symbolic == false  
 L1 = 0.333; % m  
 L2 = 0.316; % m  
 L3 = 0.384; % m  
 L4 = 0.088; % m  
 L5 = 0.107; % m  
 a = 0.0825; % m  
  
 % Base configuration  
 M\_rot = [1 0 0;  
 0 -1 0;  
 0 0 -1];  
 M\_trans = [L4;  
 0;  
 (L1 + L2 + L3 - L5)];  
 M = [M\_rot M\_trans;  
 zeros(1,3) 1];  
  
 % Fixed frame twists  
 S1 = [0 0 1 0 0 0]';  
 S2 = [0 1 0 -L1 0 0]';  
 S3 = [0 0 1 0 0 0]';  
 S4 = [0 -1 0 (L1 + L2) 0 -a]';  
 S5 = [0 0 1 0 0 0]';  
 S6 = [0 -1 0 (L1 + L2 + L3) 0 0]';  
 S7 = [0 0 -1 0 L4 0]';  
  
 S\_mat = [S1 S2 S3 S4 S5 S6 S7];  
  
 % Body frame twists  
 B1 = [0 0 -1 0 -L4 0]';  
 B2 = [0 -1 0 L2 + L3 - L5 0 L4]';  
 B3 = [0 0 -1 0 -L4 0]';  
 B4 = [0 1 0 L5 - L3 0 -L4 + a]';  
 B5 = [0 0 -1 0 -L4 0]';  
 B6 = [0 1 0 L5 0 -L4]';  
 B7 = [0 0 1 0 0 0]';  
  
 B\_mat = [B1 B2 B3 B4 B5 B6 B7];  
  
 % Adding M-type configuration of each joint to enable plotting  
 M1 = [eye(3) [0 0 L1]';  
 zeros(1,3), 1];  
  
 M2\_rot = [1 0 0;  
 0 0 1;  
 0 -1 0];  
 M2\_trans = [0 0 L1]';  
 M2 = [M2\_rot M2\_trans;  
 zeros(1,3), 1];  
  
 M3 = [eye(3) [0 0 L1 + L2]';  
 zeros(1,3), 1];  
  
 M4\_rot = [1 0 0;  
 0 0 -1;  
 0 1 0];  
 M4\_trans = [a 0 L1 + L2]';  
 M4 = [M4\_rot M4\_trans;  
 zeros(1,3), 1];  
  
 M5 = [eye(3) [0 0 L1 + L2 + L3]';  
 zeros(1,3), 1];  
  
 M6\_rot = [1 0 0;  
 0 0 -1;  
 0 1 0];  
 M6\_trans = [0 0 L1 + L2 + L3]';  
 M6 = [M6\_rot M6\_trans;  
 zeros(1,3), 1];  
  
 M7\_rot = [1 0 0;  
 0 -1 0;  
 0 0 -1];  
 M7\_trans = [L4 0 L1 + L2 + L3]';  
 M7 = [M7\_rot M7\_trans;  
 zeros(1,3), 1];  
  
 M\_intermediates = {M1, M2, M3, M4, M5, M6, M7};  
  
 % Joint Angles  
 thetas = [0 -pi/4 0 -3\*pi/4 0 pi/2 pi/4]; % PANDA NORMAL CONFIG  
  
 % Joint Limits  
 joint\_limits\_lower = [-2.8973 -1.7628 -2.8973 -3.0718 -2.8973 -0.0175 -2.8973]';  
 joint\_limits\_upper = [2.8973 1.7628 2.8973 -0.0698 2.8973 3.7525 2.8973]';  
 joint\_limits = [joint\_limits\_lower joint\_limits\_upper];  
  
 .

.

.

end  
 end  
 end

Contributions of Group Members

Responsibility was shared on the formulation and implementation of this problem.

References

1. “Robot and Interface Specifications.” *Robot and Interface Specifications - Franka Control Interface (FCI) Documentation*, https://frankaemika.github.io/docs/control\_parameters.html.
2. Murray, R.M., Li, Z., & Sastry, S.S. (1994). A Mathematical Introduction to Robotic Manipulation (1st ed.). CRC Press. <https://doi.org/10.1201/9781315136370>
3. Lynch, K. M., & Park, F. C. (2017). *Modern Robotics: Mechanics, Planning, and Control*. Cambridge University Press.