DD2425 Lab

Trajectory control for a robot manipulator

Lab designed by Christian Smith - ccs@kth.se

General Instructions

This assignment contains two parts. Part one is theoretic with some applied calculations. In order to do this part, you may need access to material on manipulator mechanics, which is available for download from BILDA. For this part, there are explicit questions with space to write in the answer. All questions and tasks are signified with a bullet.

The second part of the assignment is an applied problem to be solved in matlab.

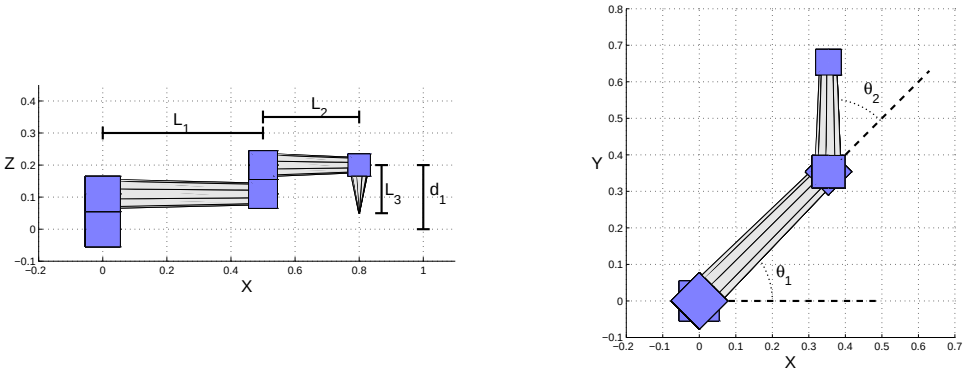
This assignment will be orally presented. You will **not** use a computer during the presentation, so make sure that you have all papers that you need ready beforehand. For part 1, you can write all answers in the space provided after each question. For part 2, you should prepare plot printouts and whatever else you need to explain your solution.

1 Robot specifications and parameters

In this exercise we will be working with a SCARA type robot. SCARAs are used to solve some simpler industrial tasks, like picking objects off a conveyor belt and placing in a box (pick-and-place). Thanks to their simplicity, they are fast and robust. The robot is shown in Figures 1(a) and 1(b). This robot has 3 degrees of freedom (DoF), the two angles θ_1 and θ_2, and the variable length of the last link, L_3. The value of d_1 is 0.2 m. The limits on motion are given in Table 1.

Table 1: Motion limits for the SCARA robot.

	min	max
θ_1	-180^o	180^o
θ_2	-45^o	180^o
L_3	0.035 m	0.2 m
$\dot{\theta}_1$	-50^o/s	50^o/s
$\dot{\theta}_2$	-50^o/s	50^o/s
$\dot{L}_3$	-0.1 m/s	0.1 m/s



(a) Side view

(b) Top view

Figure 1: Schematic drawing of the SCARA robot.

1.1 Workspace

Assume the following robots, with structure as in Figures 1(a) and 1(b):

Robot A: $L_1 = 0.4$m, $L_2 = 0.4$m
Robot B: $L_1 = 0.5$m, $L_2 = 0.3$m
Robot C: $L_1 = 0.3$m, $L_2 = 0.5$m

For each robot A-C, answer the following questions:

- What is the workspace for the robot?

- Are there any points in space which the arm can reach but where it is not able to move in all directions (not because of joint limits). These points are so called kinematic **singularities**? If so, where?

1.2 Kinematics

The position of the robot can be given in two different coordinate spaces, *cartesian space* or *joint space*. In cartesian space we will here call the position X_{cart}, which is the vector $[x\ y\ z]^T$. In joint space we will call the position Θ, which is the vector $[\theta_1\ \theta_2\ L_3]^T$. The forward kinematics, K_f, is the function that relates cartesian and joint space:

$$X_{cart} = K_f(\Theta) \tag{1}$$

- Write down the function K_f explicitly.

1.3 Inverese Kinematics

The inverese kinematics K_i is the inverse function of K_f, that is,

$$\Theta = K_i(X_{cart}), \tag{2}$$

- How many different joint space solutions can be found for each set of cartesian coordinates?

- Write down the function K_i, making sure you get all possible solutions.

- What happens when the cartesian coordinates are a singularity point (see earlier)?

1.4 Velocity Jacobian

The velocity Jacobian J is the function that relates the velocities in cartesian and joint space:

$$J_{ij} = \frac{\delta K_{f,i}}{\delta \Theta_j} \tag{3}$$

Note that since K_f is non-linear, J will be a function of Θ, that can be written $J(\Theta)$, and we will have that

$$\dot{X}_{cart} = J(\Theta)\dot{\Theta} \tag{4}$$

- Calculate the Jacobian J.

- What happens to J when the arm is in a singularity?

J also has an inverse function, J_{inv}, such that

$$\dot{\Theta} = J_{inv}(X_{cart})\dot{X}_{cart} \tag{5}$$

- How can we calculate J_{inv} using only the expressions previously derived in (1.3) and (1.4)?

- What happens to J_{inv} when the arm is in a singularity?

- Which DoFs are coupled, and how does this affect complexity and the structure of J?

2 Robot simulation

The robot simulator `rob_sim` is a function that **simulates the robot A described in part 1**. As input, it takes a matrix, where each row corresponds to time t, and the first column is θ_1, the second column is θ_2, and the third column is L3. Angles are given in degrees. The time increment from one row to the next is 0.1s. i.e, in order to move the last joint from 0.1 m to 0.2 m elongation in 1 second at constant velocity, while the two revolute joints are kept at their initial angles, we would write:

```
> trajectory = [60 -40 0.1;...
                60 -40 0.11;...
                60 -40 0.12;...
                60 -40 0.13;...
                60 -40 0.14;...
                60 -40 0.15;...
                60 -40 0.16;...
                60 -40 0.17;...
                60 -40 0.18;...
                60 -40 0.19;...
                60 -40 0.2];
> X_cart = rob_sim(trajectory)

X_cart =
    0.5759    0.4832    0.1000
    0.5759    0.4832    0.0900
    0.5759    0.4832    0.0800
    0.5759    0.4832    0.0700
    0.5759    0.4832    0.0600
    0.5759    0.4832    0.0500
    0.5759    0.4832    0.0400
    0.5759    0.4832    0.0300
    0.5759    0.4832    0.0200
    0.5759    0.4832    0.0100
    0.5759    0.4832    0.0000
```

If we call the simulator with an additional parameter, we can get it to draw the output:

```
> draw_robot = 1;  % setting to 0 means no drawing
> X_cart = rob_sim(trajectory,draw_robot)
```

NOTE:

- Dimensions given by robot **A** described in part 1
- The robot cannot exceed its maximum velocities, but will do its best to reach a desired setpoint as fast as it can.
- This robot is simplified, and does not have any limits on acceleration.
- If a setpoint is given that is outside the robot's range, the robot will clip the setpoint to fit the range. For example, if you try to set θ_2 to -180 deg, the robot will treat this as a setpoint to -45.
- The robot always starts in the same position. If may be a good idea to start your trajectories in this position:

  $\theta_1 = 60$, θ_2=-40, $L_3 = 0.1$.

2.1 Line welding

We'll assume that the end effector of the robot is equipped with a welding tool, and we want to weld a straight line from cartesian coordinates [0.1 0.3 0] to [-0.5 0.1 0]. Using the simulator above, set the second parameter (`draw_robot` in the example) to 2, and the simulator program will draw a line when the end effector is all the way down. This line represents the weld. A dotted line will show the desired path to weld.

The function `check_weld(X_cart)` will tell you if your weld is good or not. These are the definitions of a good weld:

i We should only weld where we are supposed to (allowance: ok to weld 0.01 m outside path).

ii We should weld the entire path we are supposed to, once (allowance: miss 1% of desired weld).

iii We should have a constant cartesian velocity throughout the weld (allowance: velocity std of 1% of average velocity).

The direction of the weld is arbitrary, and as you can see, it doesn't have to be perfect, but only small deviations are allowed.

- Your task is to generate a joint space trajectory that can be used in `robot_sim` that fullfills these requirements. Run the functions without parameters to get definitions of inputs and outputs. Plot your trajectory as a function of time, both for cartesian space and joint space. Show these plots when you present your work.

2.2 Performance

In the task above, you are required to weld at constant velocity. What is the shortest time in which the task can be accomplished? Time is measured as the time from the start of the simulation to the end of the weld. Do not find this by trial and error, but give a reasoning answer, supported by calculations (typically numerical rather than analytical). Feel free to verify this experimentally.