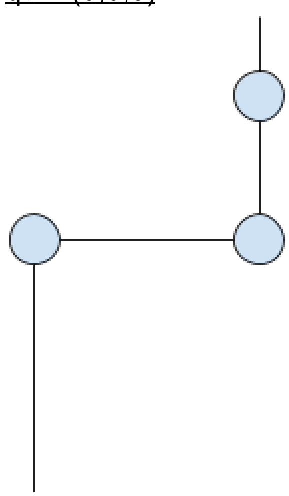
TU Berlin Robotics WS2018/19

Lab Assignment #2

Group: 2_Fri_K - Abhiraj Bishnoi(0406221), Anqi Chen (0399302) ,Puriwat Khantiviriya (0397363)

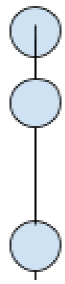
A. Forward Kinematics

1. Transformation between frames
2. End effector position in Operational space
3. Compute the End Effector Jacobian
4. Understanding the Jacobian Matrix and Pose Singularities
    a. Sketch end effector position and the column vectors of the Jacobian for the following configurations:
        - q1 = (0,0,0)



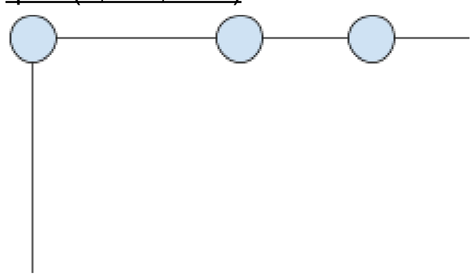
Jacobian = {
0.488 0.488 0.055
0.412 0.000 0.000
1.000 1.000 1.000}

- <u>q2 = (π/2, -π/2, 0)</u>



Jacobian = {
0.077 0.488 0.055
-0.000 0.000 0.000
1.000 1.000 1.000}

- <u>q3 = (0, π/2, 0.01)</u>



Jacobian = {
-0.001 -0.001 -0.001
0.900 0.488 0.055
1.000 1.000 1.000}

b. Using your illustrations, determine for each given configurations whether it is close to a singularity, is in a singularity, or is fully controllable in all directions (and orientations). Explain how this can be concluded from your illustrations of the column vectors.

With q1 = (0,0,0), the robot becomes fully controllable, as the robot has no sign of falling into a singularity, as the Jacobian matrix.
With q2 = (π/2, -π/2, 0), the robot falls into a singularity, and unable to control in translation at Y-axis, as the Jacobian matrix becomes 0 in the second row.
With q3 = (0, π/2, 0.01), the robot is close to the singularity, as the value in the first row of the Jacobian matrix is almost 0.

The column vector of the Jacobian matrix indicates the velocity of the translation and rotation in operational space. Which can be used to check for a singularity. A row that tends toward 0 means more chance to fall into a singularity.

B Trajectory Generation in Joint Space

1. [10 Points] Theoretical: generation of smooth trajectories with polynomial splines We want to move the robot from zero configuration qa = (0, 0, 0)T to the joint configuration qc =(−π/2 , π/4, 0)T within 5 seconds. After half the time, the manipulator should pass this intermediate point:

qb = (−π/4, π/2, 0)T

The end effector velocity at this via point is not defined. We apply a simple heuristic to choose joint velocities (which can also be found in the Craig textbook): If direction of velocity changes for a joint, its desired velocity at the via point is set to 0. Otherwise we set it to the average velocity of the two adjacent segments.

(a) Compute the cubic spline parameters that describe the complete trajectory in joint space. [6 Points] Assume that your robot controller takes four scalar values and executes the spline as: q(t) = a1+a2(t− tstart) + a3 (t − tstart)2 + a4 (t − tstart)3 with tstart being the starting time The parameters should be written with 2 decimal precision into the pdf-file!

a0 = q_0
a1 = dq_0
a2 = $3(q_f - q_0)/(t_f - t_0)^2 \; - \; 2(dq_{\,0})/(t_f - t_0) \; - \; (dq_f)/(t_f - t_0)$
a3 = $-2/(t_f - t_0)^3 + (dq_0 + dq_f)/(t_f - t_0)^2$
We split it into two splines, one from the initial position to the via point and the second from the via point to the final position

Considering the first spline from the initial position to the via point,

a0 = q0 =0
a1 = Vi = 0
a2 = (3/t_via^2) * (q_via - q0) - (2/t_via) * Vi - (1/t_via) * V_via = pi/(12.5 * s^2)
a3 = -(2/t_via^3) * (q_via - qo) + (1/t_via**2) * (V + V_via) = (pi/2)/(31.25 * s**2) = 0.05/s^3

And the second spline from the via point to the final position,
B0 = q_via = -pi/4
B1 = V_via = (-pi/2)/5*s
B2 = 3/(t_final - t_via) ^ 2 * (q_final - q_via) - (2/(t_final - t_via)) * V_via - (1/t_final - t_via) * V_final = (pi/4)/6.25^s = -0.13/s^2
B3 = -2/(t_final - t_via)^3 * (q_final - q_via) + 1/(t_final -t_via)^2 * (V_via + V_final) = (pi/2)/31.25 * s^3 = 0.05/s^3

(b) Create a diagram of the joint angles with respect to time. [4 Points]

C Operational Space Control

1. x = radius * cos(angularVelocity * time + startingPoint) + offset

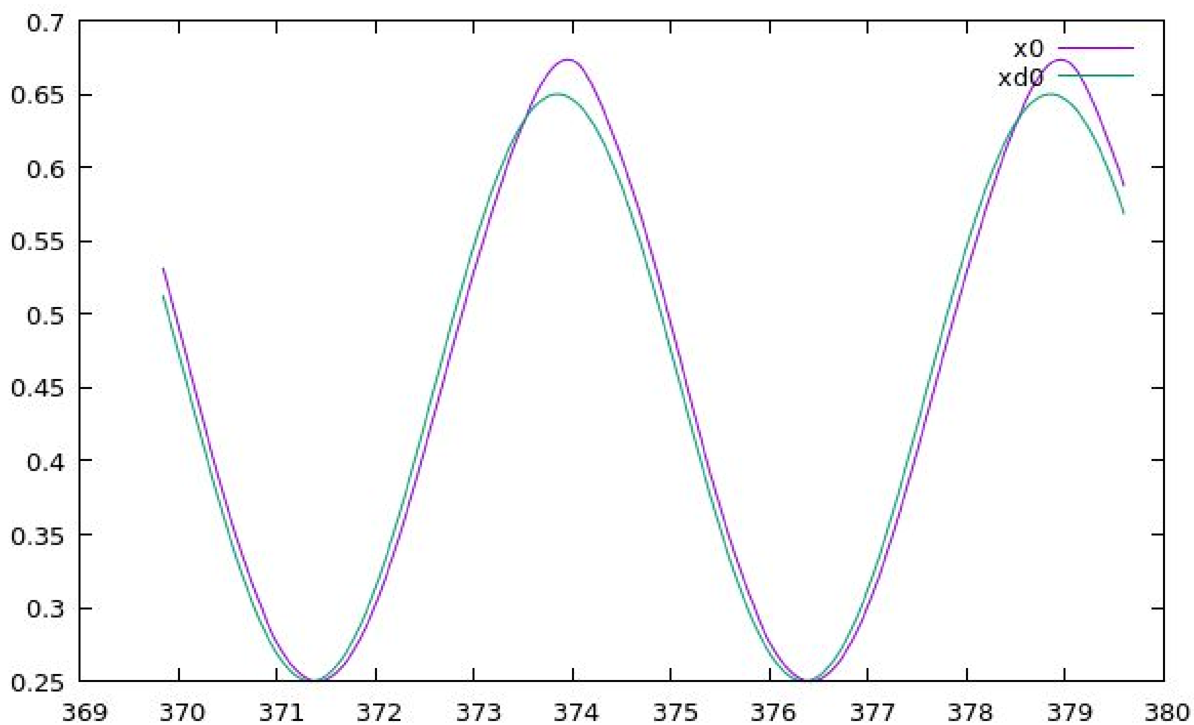
= 0.2 * cos(-2pi / 5 * time + pi / 2) + 0.45

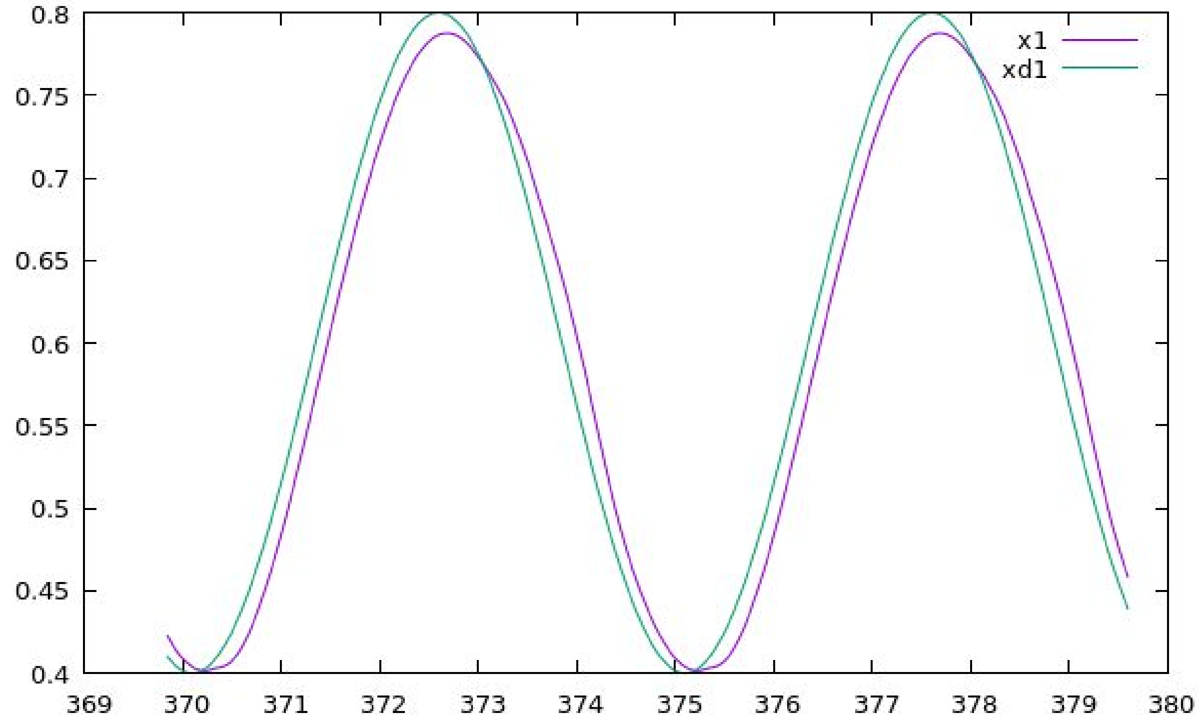
y = radius * sin(angularVelocity * time + startingPoint) + offset

= 0.2 * sin(-2pi / 5 * time + pi / 2) + 0.6

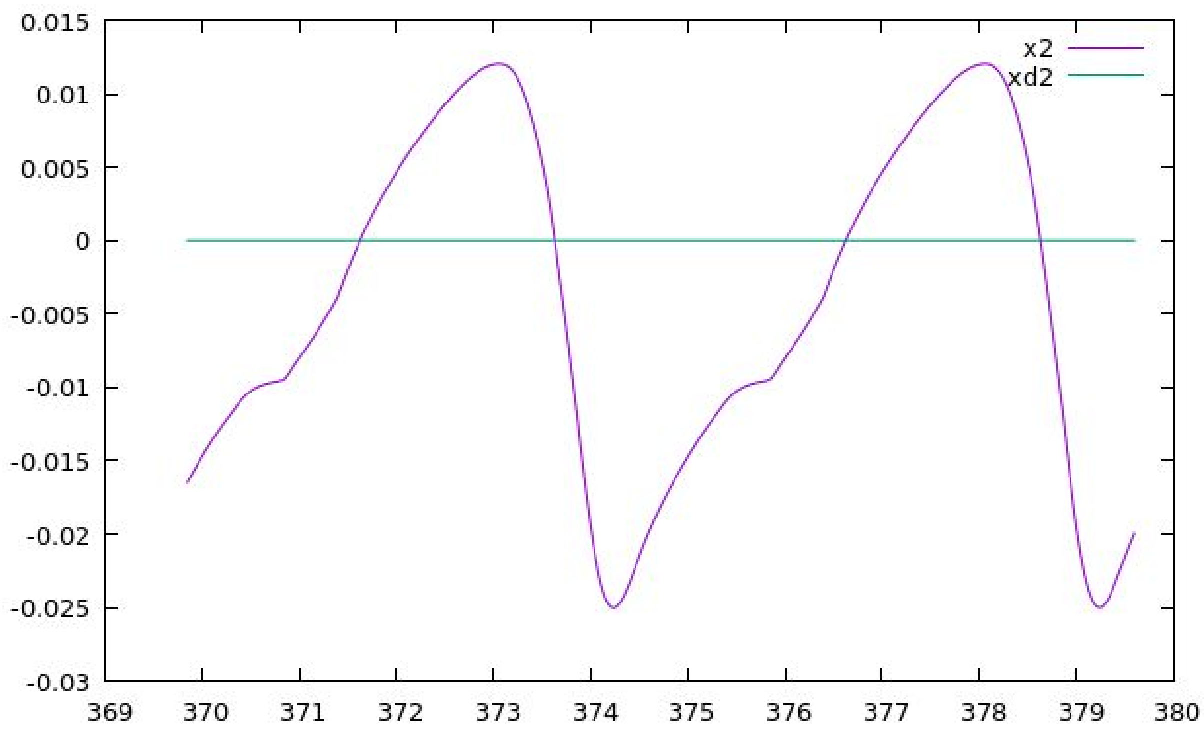
3. Kp = (4400, 4000, 200) , Kv = (200, 300, 60)

Keeping kv constant, higher kp values resulted in bigger errors.

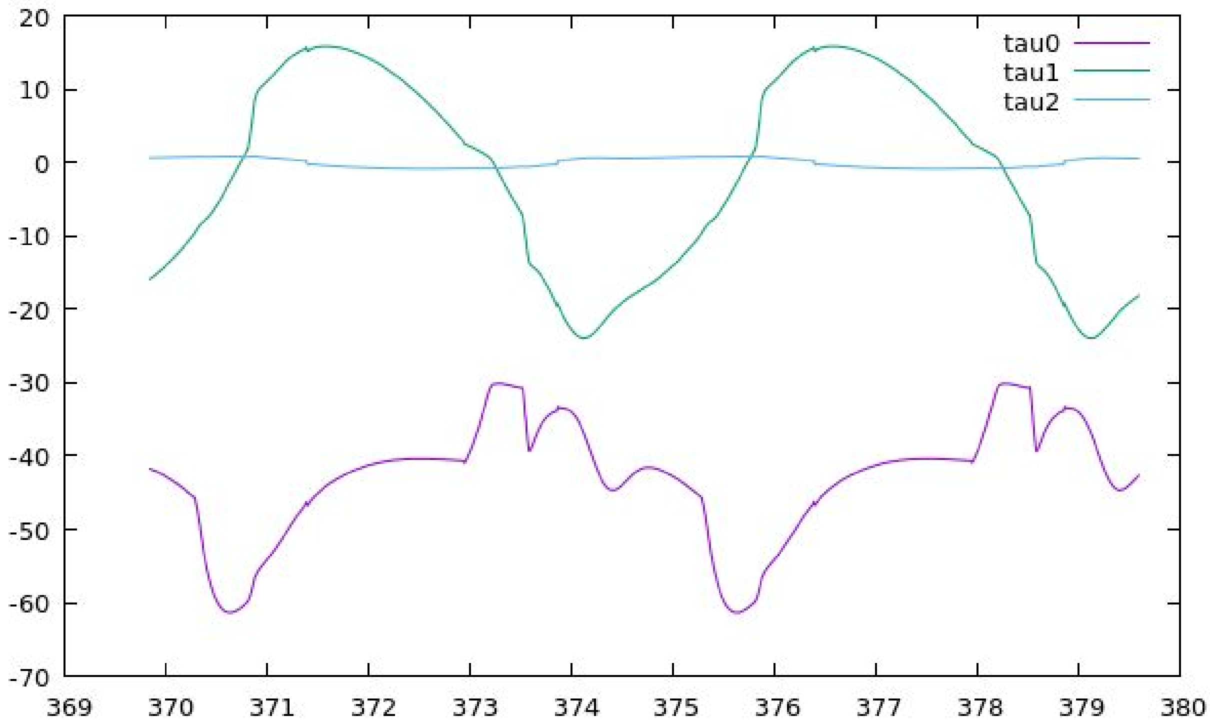




x1
xd1
0.8
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
369
370
371
372
373
374
375
376
377
378
379
380



x2
xd2
0.015
0.01
0.005
0
-0.005
-0.01
-0.015
-0.02
-0.025
-0.03
369
370
371
372
373
374
375
376
377
378
379
380



4. tb = beta’ / beta’’
   = 5s

   beta(tb) = 0.5 * beta’’ * t ^ 2
   = pi

   total_time = 2tb + (6pi - 2beta(tb)) / beta’
   = 20s

   total_betal = 6pi

Now we need to get different beta’ and beta’’ for the three sections (accelerating, constant velocity, decelerating) of the trajectory.

Accelerating (t < tb)

beta(t) = 0.5 * beta’’ * t ^ 2
beta’(t) = beta’’ * t

Constant Velocity (tb < t < t_total - tb)

beta(t) = beta’ * (t - tb) + beta(tb)

beta’(t) = beta’

Decelerating (t < t_total - tb)

beta(t) = beta’’ - 0.5 * beta’’ * (t - t_total) ^ 2

beta’(t) = -beta’’ * (t - t_total)

x’ = - angularVelocity * radius * sin(angularVelocity * time + startingPoint)

y’ =   angularVelocity * radius * cos(angularVelocity * time + startingPoint)

Student Name	A1	A2	A3	(A4)	(B1)	B2	C1	C2	C3	C4	C5
Abhiraj Bishnoi	x	x	x			x					
Anqi Chen							x	x	x	x	x
Puriwat Khantiviriya	x	x	x			x					