MELBOURNE
SCHOOL OF
ENGINEERING

MCEN90028
Robotic Systems

Assignment 3:
Virtual Robot simulation

Jiawei Liao 756560

Department of Mechanical Engineering

June 3, 2020

Contents

1 Description of Assignment

The purpose of this assignment is to implement joint space and task space controllers on a virtual planar two-link revolute revolute robot (Figure 1) to track a planned trajectory while avoiding a circular obstacle.
The assignment is comprised of 5 main tasks:

1. **Task 1**: Derive and compute the equations of motion of the robot and construct the simulation environment. Evaluate the response using zero driving torque and gravity compensation torque.

2. **Task 2**: Design and implement joint space PID controllers using joint velocities as inputs and joint torque as output. Evaluate performance against a set joint space trajectory.

3. **Task 3**: Test the joint space PID controllers designed in Task 2 on 2 defined task space trajectories A and B where trajectory B was deliberately designed to reach beyond the robot workspace. The trajectories are converted to joint space using inverse kinematics for this task.

4. **Task 4**: Design and test task space PI controllers on the robot tracking the Trajectory A defined in Task 3. In this task the task space controller outputs are converted back to joint velocity commands to be sent to the velocity commanded robot.

5. **Task 5**: Design and implement a general purpose obstacle avoidance protocol to be used alongside the task space controllers. Evaluate the performance using Trajectory A against different settings of obstacles.

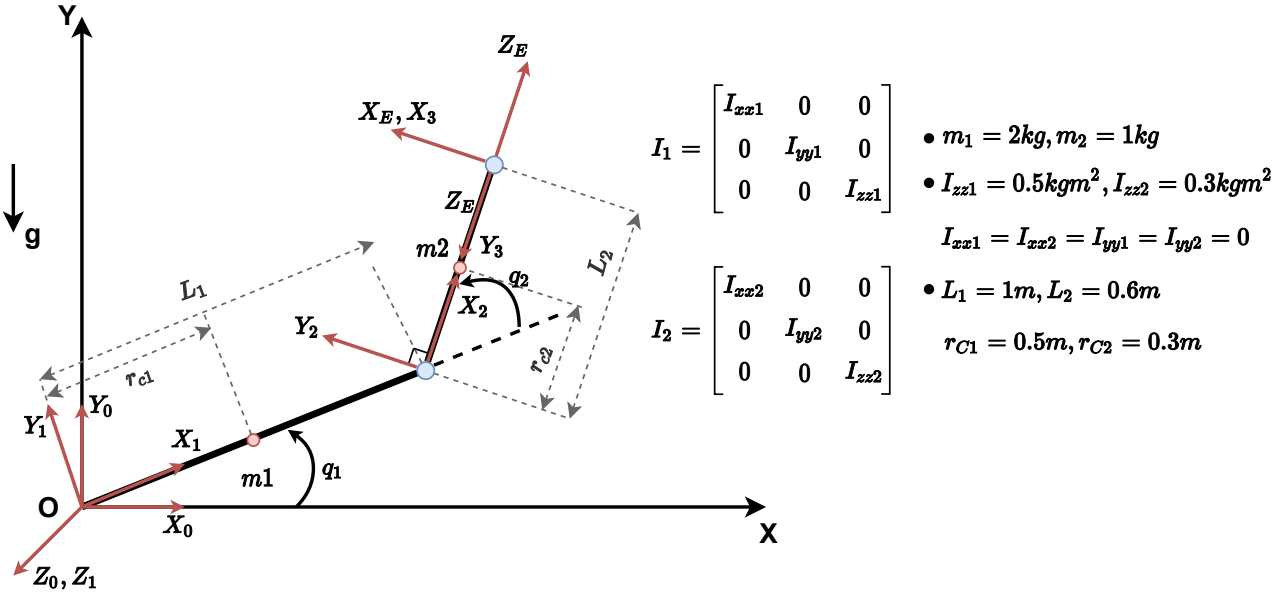


Figure 1: Planar 2-Link Revolute-Revolute Robot Diagram

2 Task 1. Construction of a Robot Dynamic Simulator

2.1 Derivation of the Equation of Motion

2.1.1 Deriving the Jacobian at COM_1 and COM_2

Using the frames defined in Figure 1, we can produce the Denavit–Hartenberg parameters for this robot configuration. Note that only frame 1 and frame 2 are associated with joint angular displacement q_1, q_2. Frame 3 and Frame E were created for end-effector frame alignment.

DH Parameter Table				
i	$\mathbf{a_{i-1}}$	α_{i-1}	$\mathbf{d_i}$	θ_i
1	0	0	0	q_1
2	L_1	0	0	q_2
3	L_2	0	0	$\pi/2$
E	0	$\pi/2$	0	0

Table 1: DH Parameter Table

The fist section in the MATLAB script "Task1_main.m" computes the transformation matrices given the DH table, where functions are recycled from Assignment 1 submissions.

$$ {}^{0}_{1}T = \begin{pmatrix} \cos(q_1) & -\sin(q_1) & 0 & 0 \\ \sin(q_1) & \cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{1} $$

$$ {}^{0}_{2}T = \begin{pmatrix} \cos(q_1+q_2) & -\sin(q_1+q_2) & 0 & L_1\cos(q_1) \\ \sin(q_1+q_2) & \cos(q_1+q_2) & 0 & L_1\sin(q_1) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}, \tag{2} $$

$$ {}^{0}_{E}T = \begin{pmatrix} -\sin(q_1+q_2) & 0 & \cos(q_1+q_2) & L_2\cos(q_1+q_2)+L_1\cos(q_1) \\ \cos(q_1+q_2) & 0 & \sin(q_1+q_2) & L_2\sin(q_1+q_2)+L_1\sin(q_1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{3} $$

Using the transformation matrices we could now compute the coordinate of the two centres of masses COM_1 and COM_2 in Frame $\{0\}$.

$$ \begin{pmatrix} {}^{0}r_{COM1} \\ 1 \end{pmatrix} = {}^{0}_{1}T([{}^{1}r_{COM1};0] - [{}^{1}r_{O1};0]) = {}^{0}_{1}T \begin{pmatrix} \mathrm{r}_{c1} \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathrm{r}_{c1}\cos(q_1) \\ \mathrm{r}_{c1}\sin(q_1) \\ 0 \\ 1 \end{pmatrix} \tag{4} $$

$$ \begin{pmatrix} {}^{0}r_{COM2} \\ 1 \end{pmatrix} = {}^{0}_{2}T([{}^{2}r_{COM2};0] - [{}^{2}r_{O2};0]) = {}^{0}_{2}T \begin{pmatrix} \mathrm{r}_{c2} \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathrm{r}_{c2}\cos(q_1+q_2)+L_1\cos(q_1) \\ \mathrm{r}_{c2}\sin(q_1+q_2)+L_1\sin(q_1) \\ 0 \\ 1 \end{pmatrix} \tag{5} $$

The point of COM_1 is attached to Frame $\{1\}$ where its absolute angular velocity is:

$$ {}^0\omega_1 = {}^0_1R \cdot \omega_{1,0} = {}^0_1R \cdot \begin{pmatrix} 0 \\ 0 \\ \dot{q_1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dot{q_1} \end{pmatrix} $$

The point of COM_2 is attached to Frame $\{2\}$ where its absolute angular velocity is:

$$ {}^0\omega_2 = {}^0_2R \cdot \omega_{2,1} + {}^0\,\omega_1 = {}^0_2R \cdot \begin{pmatrix} 0 \\ 0 \\ \dot{q_2} \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ \dot{q_1} \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ \dot{q_1} + \dot{q_2} \end{pmatrix} $$

Now we could derive the Jacobian at COM_1 and COM_2 using functional derivatives:

$$ \begin{pmatrix} JV_{m1} \\ J\omega_{m1} \end{pmatrix} = \begin{pmatrix} \frac{\partial({}^0\vec{r}_{COM1})}{\partial(q_1)} & \frac{\partial({}^0\vec{r}_{COM1})}{\partial(q_2)} \\ \frac{\partial({}^0\vec{\omega}_1)}{\partial(\dot{q_1})} & \frac{\partial({}^0\vec{\omega}_1)}{\partial(\dot{q_2})} \end{pmatrix} = \begin{pmatrix} -\mathrm{r}_{c1}\sin(q_1) & 0 \\ \mathrm{r}_{c1}\cos(q_1) & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \end{pmatrix} \tag{6} $$

$$ \begin{pmatrix} JV_{m2} \\ J\omega_{m2} \end{pmatrix} = \begin{pmatrix} \frac{\partial({}^0\vec{r}_{COM2})}{\partial(q_1)} & \frac{\partial({}^0\vec{r}_{COM2})}{\partial(q_2)} \\ \frac{\partial({}^0\vec{\omega}_2)}{\partial(\dot{q_1})} & \frac{\partial({}^0\vec{\omega}_2)}{\partial(\dot{q_2})} \end{pmatrix} = \begin{pmatrix} -\mathrm{r}_{c2}\sin(q_1+q_2) - L_1\sin(q_1) & -\mathrm{r}_{c2}\sin(q_1+q_2) \\ \mathrm{r}_{c2}\cos(q_1+q_2) + L_1\cos(q_1) & \mathrm{r}_{c2}\cos(q_1+q_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \tag{7} $$

2.1.2 Deriving components of the Equations of Motion

The dynamics of the robot could be described by the following equation of motion:

$$ \tau = A\ddot{q} + B(q)[\dot{q}\dot{q}] + C(q)[\dot{q}^2] + G(q) \tag{8} $$

where

- $\tau \in R^N$ is the joint torques,
- $A \in R^{N\times N}$ is the inertia matrix,
- $\ddot{q}$ is the joint acceleration,
- $B(q)[\dot{q}\dot{q}] \in R^{N\times N(N-1)/2}$ is the Coriolis terms,
- $C(q)[\dot{q}^2] \in R^{N\times N}$ is the centrifugal terms,
- $G(q)$ is the gravitational terms.

$A(q)$ could be directly computed using the Jacobian calculated in Equations 6,7:

$$ \begin{aligned} A(q) &= \sum_{i=1}^{N}(m_i J_{Vi}^T J_{Vi} + J_{\omega i}^T I_{ci} J_{\omega i}) \\ &= m_1 JV_{m1}^T JV_{m1} + J\omega_{m1}^T I_{m1} J\omega_{m1} + m_2 JV_{m2}^T JV_{m2} + J\omega_{m2}^T I_{m2} J\omega_{m2} \\ &= \begin{pmatrix} m_2\,{L_1}^2 + 2\,m_2\cos(q_2)\;L_1\,\mathrm{r}_{c2} + m_1\,{\mathrm{r}_{c1}}^2 + m_2\,{\mathrm{r}_{c2}}^2 + \mathrm{Izz}_1 + \mathrm{Izz}_2 & m_2\,{\mathrm{r}_{c2}}^2 + L_1\,m_2\cos(q_2)\;\mathrm{r}_{c2} + \mathrm{Izz}_2 \\ m_2\,{\mathrm{r}_{c2}}^2 + L_1\,m_2\cos(q_2)\;\mathrm{r}_{c2} + \mathrm{Izz}_2 & m_2\,{\mathrm{r}_{c2}}^2 + \mathrm{Izz}_2 \end{pmatrix} \end{aligned} \tag{9} $$

We can now define a notation a_{ij} to simplify the derivation of $B(q)$ and $C(q)$, where a_{ij} represents the ij element of the inertia matrix A.
For this particular assignment, A is of size 2×2 and could be represented as :

$$A(q) = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \tag{10}$$

Next, we define the notation a_{ijk} which is the derivative of a_{ij} with respect to q_k.

$$a_{ijk} = \frac{\partial a_{ij}}{\partial q_k} \tag{11}$$

The Christoffel symbols bi, jk could then be constructed:

$$b_{i,jk} = \frac{1}{2}(a_{ijk} + a_{ikj} - a_{jki}) \tag{12}$$

$B(q)$ and $C(q)$ can now be derived using the Christoffel symbols:

$$B(q) = \begin{pmatrix} 2b1,12 \\ 2b2,12 \end{pmatrix} = \begin{pmatrix} -2\,L_1\,m_2\,\mathrm{r}_{c2}\sin(q_2) \\ 0 \end{pmatrix} \tag{13}$$

$$C(q) = \begin{pmatrix} b1,11 & b1,22 \\ b2,11 & b2,22 \end{pmatrix} = \begin{pmatrix} 0 & -L_1\,m_2\,\mathrm{r}_{c2}\sin(q_2) \\ L_1\,m_2\,\mathrm{r}_{c2}\sin(q_2) & 0 \end{pmatrix} \tag{14}$$

The expression for the gravitational component of the joint torques could be derived. Note in this assignment setting the gravity points in the negative Y_0 direction.

$$\begin{aligned} G(q) &= \sum_{i=1}^{N} J_{vi}^T(-m_i g\hat{Y}_0) \\ &= JV_{m1}(-m_1 g\hat{Y}_0) + JV_{m2}(-m_2 g\hat{Y}_0) \\ &= \begin{pmatrix} g\,m_2\left(\mathrm{r}_{c2}\cos(q_1+q_2) + L_1\cos(q_1)\right) + g\,m_1\,\mathrm{r}_{c1}\cos(q_1) \\ g\,m_2\,\mathrm{r}_{c2}\cos(q_1+q_2) \end{pmatrix} \end{aligned} \tag{15}$$

Now we have derived all the components in the equation of motion 8.

$$\begin{aligned} \tau_1 =& -L_1\,m_2\,\mathrm{r}_{c2}\sin(q_2)\;\dot{\mathrm{q}}_2^2 - 2\,L_1\,m_2\,\dot{\mathrm{q}}_1\,\mathrm{r}_{c2}\sin(q_2)\;\dot{\mathrm{q}}_2 + \ddot{\mathrm{q}}_2\left(m_2\,{\mathrm{r}_{c2}}^2 + L_1\,m_2\cos(q_2)\;\mathrm{r}_{c2} + \mathrm{Izz}_2\right) \\ &+ \ddot{\mathrm{q}}_1\left(m_2\,{L_1}^2 + 2\,m_2\cos(q_2)\;L_1\,\mathrm{r}_{c2} + m_1\,{\mathrm{r}_{c1}}^2 + m_2\,{\mathrm{r}_{c2}}^2 + \mathrm{Izz}_1 + \mathrm{Izz}_2\right) \\ &+ g\,m_2\left(\mathrm{r}_{c2}\cos(q_1+q_2) + L_1\cos(q_1)\right) + g\,m_1\,\mathrm{r}_{c1}\cos(q_1) \end{aligned} \tag{16}$$

$$\begin{aligned} \tau_2 =& +L_1\,m_2\,\mathrm{r}_{c2}\sin(q_2)\;\dot{\mathrm{q}}_1^2 + \ddot{\mathrm{q}}_2\left(m_2\,{\mathrm{r}_{c2}}^2 + \mathrm{Izz}_2\right) + \ddot{\mathrm{q}}_1\left(m_2\,{\mathrm{r}_{c2}}^2 + L_1\,m_2\cos(q_2)\;\mathrm{r}_{c2} + \mathrm{Izz}_2\right) \\ &+ g\,m_2\,\mathrm{r}_{c2}\cos(q_1+q_2) \end{aligned} \tag{17}$$

Note that in the simulation we have also introduced a friction term $Fr(q) = C_{Fr} \cdot \vec{q}$ on the right side of the EOM.
Thus the $\ddot{q}$ could be solved in the MATLAB dynamics function "Task1_dynamics.m" during its call by:

$$\ddot{q} = inv(A(q)) \cdot (\tau - Fr(q) - B(q) \cdot (\dot{q}_1 \cdot \dot{q}_2) - C(q) \cdot [\dot{q}_1^2; \dot{q}_1^2] - G(q)) \tag{18}$$

2.2 Simulation using zero driving torque

A simulation of the robot is produced for 5 seconds from the initial displacement of $q_1 = q_2 = 0$ at rest, with sampling period of $dt = 0.01s$ and joint torque $\tau_1 = \tau_2 = 0$ throughout. The friction coefficient C_{Fr} is set as 0.1.
The second section of the MATLAB script "Task1_main.m" could be run by setting the torque to 0 for this section to produce the plots q_1 vs t and q_2 vs t. The animation of the simulation could be found as "Animation/task1_2.gif".

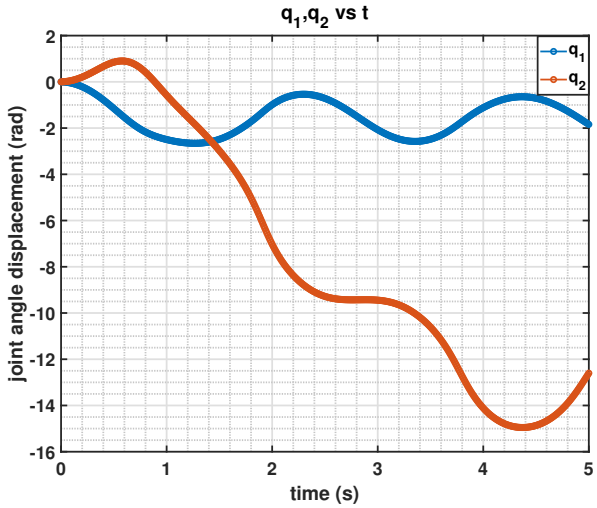


Figure 2: q_1, q_2 vs t with $\tau_1 = \tau_2 = 0$

Without a joint angle limit, we could observe in the animation and the plot 2 that joint 2 rotates clockwise for several revolutions during the simulation. Joint 1's angle displacement is an oscillation between 0 and $-\pi$ with a decaying amplitude due to the introduced friction term.

2.3 Simulation using gravity compensation torque

A simulation of the robot is produced for 5 seconds from the initial displacement of $q_1 = q_2 = 0$ at rest, with sampling period of $dt = 0.01s$ and joint torque $\tau = G(q)$ to compensate for the gravity forces. The friction coefficient C_{Fr} is set as 0.1.
By uncommenting the $\tau = G(q)$ line in the main loop of the script, we are able to produce the plots q_1 vs t and q_2 vs t (Figure 3) and the associated animation can be found as "Animation/-task1_3.gif".
Since the gravitational forces are exactly balanced by the driving torque, the robot now remains stationary in its initial position throughout the simulation. However, this is unrealistic to achieve in a practical setting as neither the modelling of the robot nor the torque command could be perfect. If we change the L_1 length in the robot dynamics function to $1.005m$, which is just a 0.5% offset, the robot will cease to stay still. (Figure 4)To compensate for gravitational forces in a practical setting, it is necessary to use a feedback controller to monitor the discrete real-time deviation of the robot from the expected configuration and correct its path.

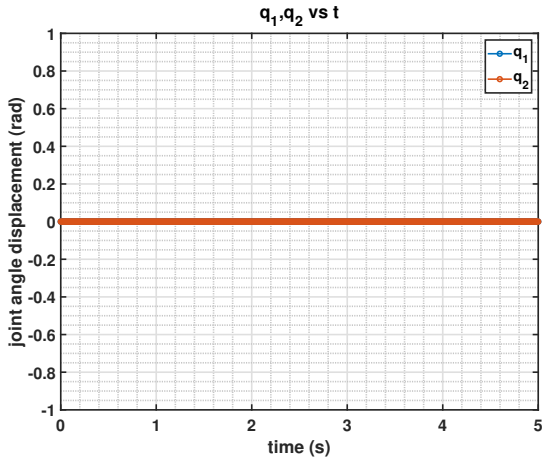


Figure 3: $q_{1,2}$ vs t, $\tau = G(q)$

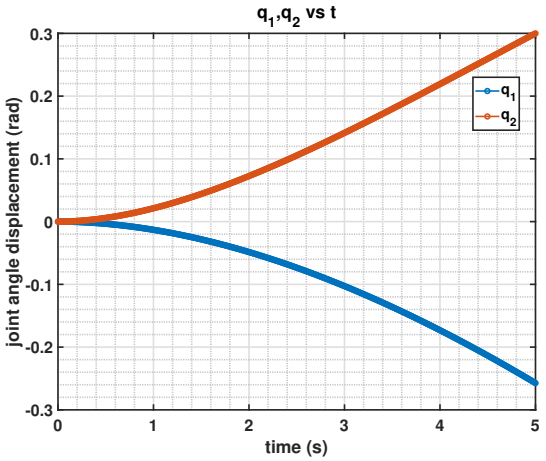


Figure 4: $q_{1,2}$ vs t, $\tau = G(q)$, $L_1 + 0.5\%$ offset

3 Task 2. PID Controller on robot in joint space

After Task 1, the robot, represented by the ode45 function, takes the input of joint torque $[\tau_1, \tau_2]^T$. The robot is therefore in "torque-controlled mode".
In Task 2, we would implement joint-space PID controllers taking velocity commands and outputs the driving torque to the robot dynamics function "Task2_dynamics.m".

- The PID controller operates at a sampling period $dt_{PIDloop} = 0.001s$. At each sampling instance a new joint velocity command is issued and a new controller output is generated.
- The main robot control loop has a sampling period $dt = 0.01s$. This means that the PID controller output only reaches the robot every $0.01s$. In simpler words, the robot only receives every 5^{th} command from the PID controller.

3.1 Design and tuning of joint-space PID controllers

The joint-space PID control loop could be described by the following flowchart.

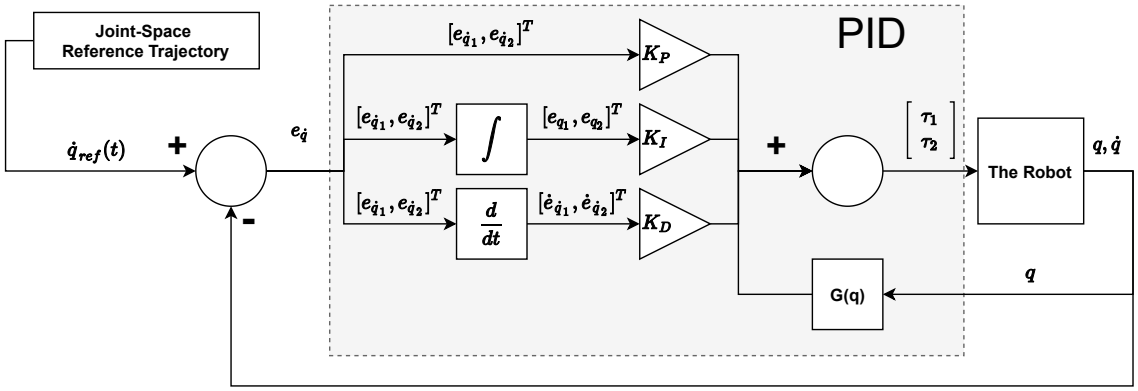


Figure 5: Joint-space PID control loop flowchart

A few crucial elements of this flowchart is explained below:

- The $q(t)$ reference trajectory is constructed by cubic polynomials using the "TrajGen01.m" and "TrajVal01.m" function in Assignment 3. $\dot{q}(t)$ reference is obtained by differentiating $q(t)$.
- The $G(q)$ term is added to the controller dynamics to make the system more linearised. However, in practical world we might not have the sensors to measure the joint angles so we would also showcase the PID without the $G(q)$ compensation.
- The input of the controller is the velocity error $e_{\dot{q}}$. the integral of velocity error is simply the position error e_q can be approximated using the rectangular method.

$$e_{qk} \approx e_{qk-1} + \dot{e}_{qk-1} \cdot dt_{PIDloop} \tag{19}$$

- The time derivative of the velocity error could be well approximated by the first order Euler difference given $dt_{PIDloop} = 0.001s$ is sufficiently small.

$$\dot{e}_{\dot{q}k} \approx \frac{e_{\dot{q}k} - e_{\dot{q}k-1}}{dt_{PIDloop}} \tag{20}$$

3.1.1 PID controller tuning

There is no strict performance criteria for the PID controller, but a settling time of $\leq 0.01s$ for $\delta q = 1^\circ$ commanded step displacement for individual joint is desired.
For the tuning of the PID controllers, the classical Zielger-Nichols' method is applied for each individual joint. (Ziegler, Nichols, et al. 1942) The general process is:

1. The robot is initialised in the position $q_1 = q_2 = 0$ at rest. Set controller for the other joint to be off.
2. The reference signal of $q(t)$ is a $\delta q = 1^\circ/s$ displacement.
3. Set $K_I = K_D = 0$, and tune K_P until the critical gain K_u.
4. Lower the K_u but make sure that that the settling time roughly $\leq 0.01s$ (5% to 10% tolerance).
5. Tune K_I and K_D accordingly to improve the transient behavior on a non-step cubic trajectory to improve the transient behavior.
6. Repeat the process for the second joint.
    - The detailed process of tuning K_I and K_D is not included in this report.
    - The values of $K_{P1} = 200$ and $K_{P2} = 37.5$ are chosen to satisfy the $0.01s$ settling time criteria and are shown in Figure 6 and Figure 7.

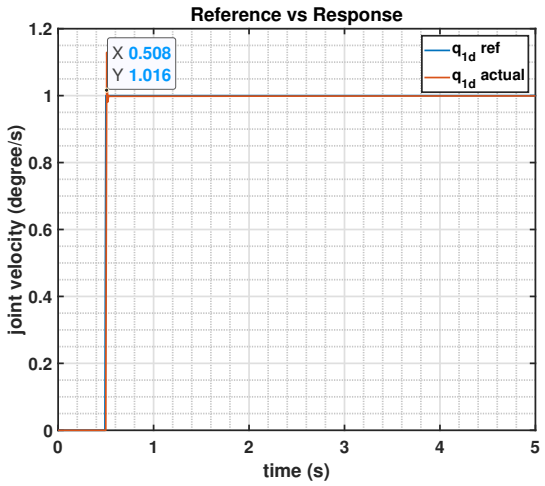


Figure 6: $q_1$ref vs q_1 response,$K_{P1} = 200$

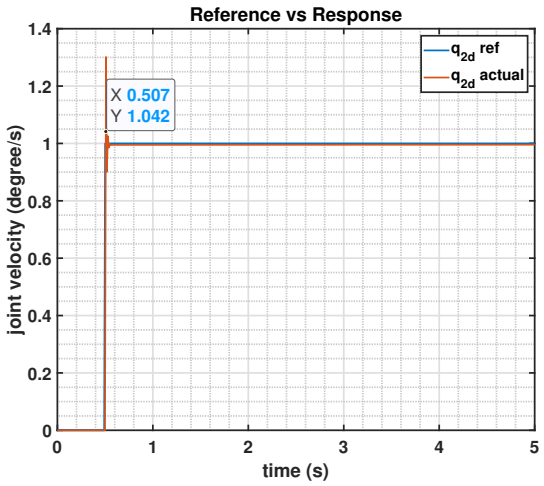


Figure 7: $q_2$ref vs q_2 response,$K_{P2} = 37.5$

- The PID controllers with $G(q)$ compensation is defined by:

$$K_{\tau_1} = \begin{bmatrix} K_{P1} \\ K_{I1} \\ K_{D1} \end{bmatrix} = \begin{bmatrix} 200 \\ 0.5 \\ 0.125 \end{bmatrix}, \quad K_{\tau_2} = \begin{bmatrix} K_{P2} \\ K_{I2} \\ K_{D2} \end{bmatrix} = \begin{bmatrix} 37.5 \\ 0.1 \\ 0.005 \end{bmatrix} \tag{21}$$

- The PID controllers without $G(q)$ compensation (model free) is defined by:

$$K_{\tau_1} = \begin{bmatrix} K_{P1} \\ K_{I1} \\ K_{D1} \end{bmatrix} = \begin{bmatrix} 200 \\ 25 \\ 0.01 \end{bmatrix}, \quad K_{\tau_2} = \begin{bmatrix} K_{P2} \\ K_{I2} \\ K_{D2} \end{bmatrix} = \begin{bmatrix} 37.5 \\ 5 \\ 0.0025 \end{bmatrix} \tag{22}$$

3.2 Simulation using a joint-space trajectory

A joint-space trajectory is designed to test the PID controller's field performance.

- At $t_i = 0s$, $q_1 = 30°, q_2 = 75°, \dot{q}_1 = \dot{q}_2 = 0$.
- At $t_f = 5s$, $q_1 = -15°, q_2 = 45°, \dot{q}_1 = \dot{q}_2 = 0$.
- $q_1(t)$ and $q_2(t)$ references are constructed by cubic polynomial using "TrajGen01.m" and "TrajVal01.m" written for Assignment 3. $\dot{q}_1(t)$ and $\dot{q}_2(t)$ references are computed as the time derivative of $q(t)$ using the gradient function.
- Friction coefficient $C_{Fr} = 0.1$ and the robot main sampling period $dt = 0.01s$.

1. If **the gravitational force is compensated in the control loop** the tracking performance is close to perfect. It can be seen in Figure 8 that the joint displacement change over time is very smooth during the action. The joint space trajectory are constructed using cubic polynomials and naturally the graph of $q_1(t)$ and $q_2(t)$ faithfully resemble the

shape of a cubic inflection curve. Since the velocity constraint at both ends are 0, the inflection point occurs at the half time of the action.
The joint velocities $\dot{q}_1(t)$ and $\dot{q}_2(t)$, which are the derivatives of their respective cubic polynomial displacements, are quadratic parabolas with the maximum angular speed occurring at the mid point of the action. (Figure 9)

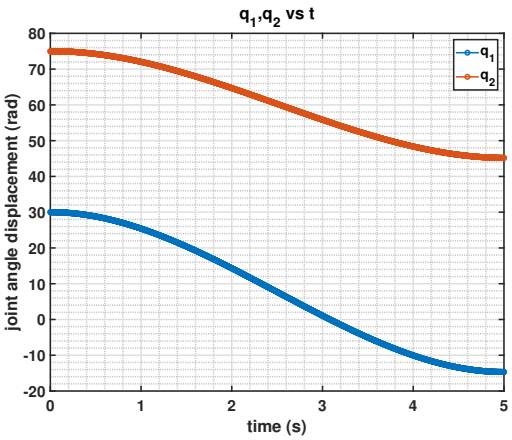


Figure 8: q_1, q_2 vs t with $(G(q))$

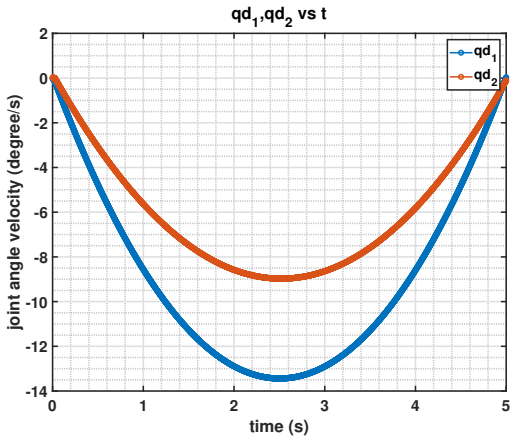


Figure 9: $\dot{q}_1, \dot{q}_2$ vs t with $(G(q))$

2. However, if the controller uses the model free approach. Even with good efforts, the controller could not successfully track the velocity reference and has a large error in the joint angle displacement response against the displacement reference. The initial 'fall' from the starting position due to gravitational forces could not be well corrected alone by the model-free PID controller.

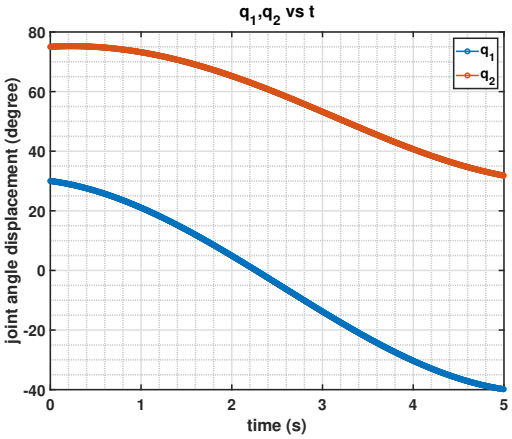


Figure 10: q_1, q_2 vs t (model free)

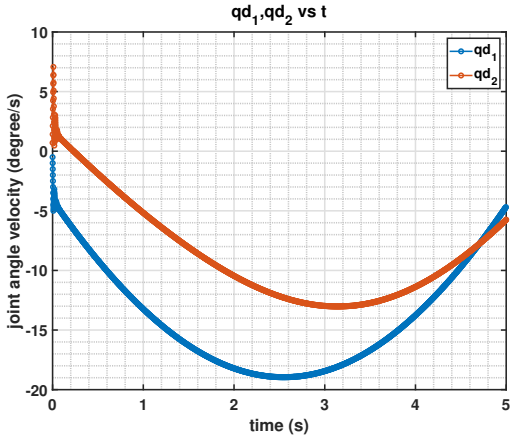


Figure 11: $\dot{q}_1, \dot{q}_2$ vs t with (model free)

The torques generated throughout the trajectory using the two settings could provided us insights on why model-free PID performed so poorly as compared to PID with gravity compensation. showcase the difference between these two approaches are smooth as seen in Figure 12 so the controller behavior during the action could be considered stable.

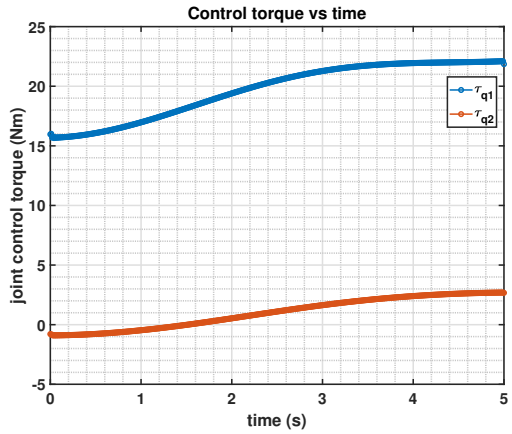


Figure 12: τ vs t (G(q) compensated)

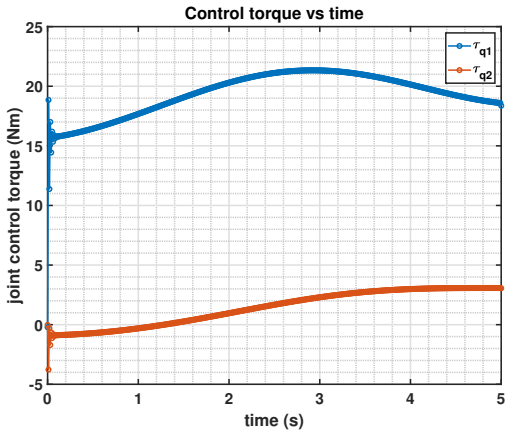


Figure 13: τ vs t (model free)

The largest proportion of the torque is required to compensate the gravitational terms. The torque required to compensate the gravitational forces depends on the angles the robot makes with the horizon. The expression of $G(q)$ (15) includes only the cosine terms. The more the robot is aligning with either side of the X-axis, the more torque is required as the cosine terms are the largest when the angle is 0.
For the model-free case, there is no initial force to balance the robot in its starting position, and the initial torque is extra turbulent. The non-linear force remains difficult to balance throughout the trajectory. (13) The XY plot of the robot is given for every 0.5s in Figure 14 and 15.

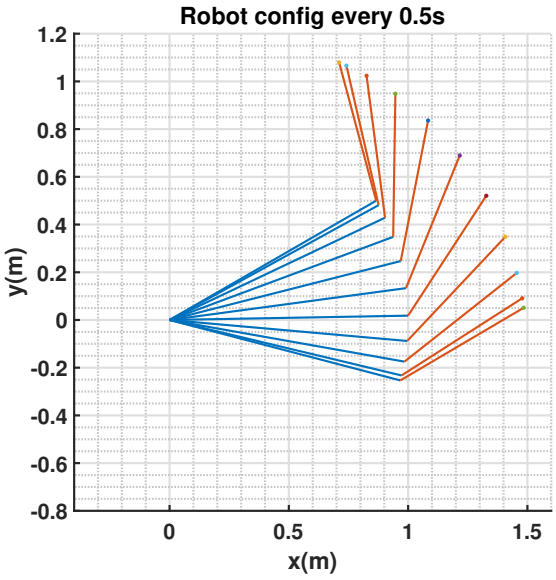


Figure 14: Robot animation (models G)

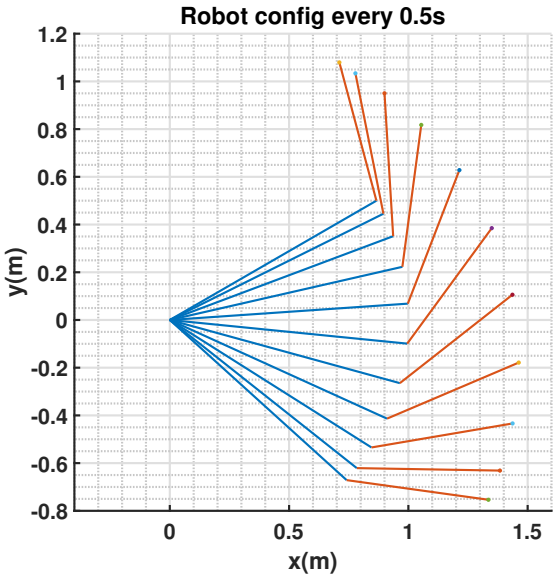


Figure 15: Robot animation (model free)

It is difficult to do velocity commands without sensors to model the the gravitational forces can be compensated. We will use apply gravitational compensation to Task 3.

4 Task 3. Joint space control of the velocity commanded robot

In Task 3, the reference trajectory is generated in the task-space (end-effector displacement). The task space reference trajectory $(x_{ref}(t), y_{ref}(t))$ is constructed using cubic polynomials from an initial coordinate (x_i, y_i) to the final coordinate (x_f, y_f) in a straight line. Then the displacement reference can be differentiated to obtain the task space velocity reference $(\dot{x}_{ref}, \dot{y}_{ref})$.
In Task 3, the robot dynamics function "Task3_dynamics.m" has included a joint limit.

- $-60° \leq q_1 \leq 60°$
- $-90° \leq q_2 \leq 90°$

4.1 Converting task-space commands to joint-space commands using inverse kinematics

The task space displacement reference can be converted directly using the inverse kinematics function "Ik2d2link.m" from Assignment 3. The inverse kinematics for the robot could be derived explicitly (as explained in Lectures 7-8). The diagram and two solution manifolds are given below:

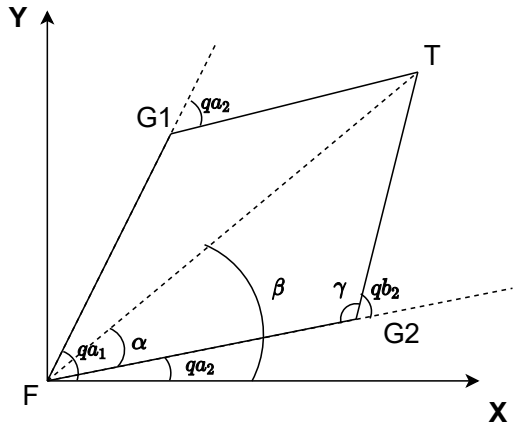


Figure 16: IK solution of planar revolute-revolute robot

$$
\begin{aligned}
q_2 &= \pm \arccos\left(\frac{x^2 + y^2 - (L_1^2 + L_2^2)}{2L_1L_2}\right) = \pm D, D \geq 0 \\
\gamma &= abs(\pi - q_2) \\
\alpha &= \arcsin\left(\frac{\arcsin\left(L_2 \sin\left(\gamma\right)\right)}{\sqrt{(x^2 + y^2)}}\right) \\
\beta &= \arctan\left(\frac{y}{x}\right) \\
q_1 &= \beta - \alpha, \text{if } q_2 = D \\
q_1 &= \beta + \alpha, \text{if } q_2 = -D
\end{aligned}
\tag{23}
$$

This solution in 23 is used to initialise the robot for the starting position in the elbow-down configuration. We will assume that the robot will operate in the elbow-down configuration for the rest of the assignment.
The task space velocity reference $[\dot{x}(t), \dot{y}(t)]$ can be converted to the joint space velocity commands $[\dot{q}_1(t), \dot{q}_2(t)]$ by using the inverse of the Jacobian at the end-effector.

$$[\dot{q}_1(t); \dot{q}_2(t)]^T = J^{-1} \cdot [\dot{x}(t); \dot{y}(t)]^T \tag{24}$$

To obtain the term J in Equation 24, we need to revisit the DH table 1.
The transformation matrix from the end-effector frame to the inertia frame is given by:

$${}^0_E T = \begin{pmatrix} -\sin(q_1+q_2) & 0 & \cos(q_1+q_2) & L_2\cos(q_1+q_2)+L_1\cos(q_1) \\ \cos(q_1+q_2) & 0 & \sin(q_1+q_2) & L_2\sin(q_1+q_2)+L_1\sin(q_1) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{25}$$

The absolute displacement of the end-effector ${}^0r_{OE}$ could be obtained from the right column of ${}^0_E T$:

$$\begin{pmatrix} {}^0r_{OE} \\ 1 \end{pmatrix} = {}^0_E T([{}^E r_{OE}; 0] - [{}^E r_{OE}; 0]) = {}^0_E T \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} \mathrm{r}_{c2}\cos(q_1+q_2)+L_1\cos(q_1) \\ \mathrm{r}_{c2}\sin(q_1+q_2)+L_1\sin(q_1) \\ 0 \\ 1 \end{pmatrix} \tag{26}$$

Again we could obtain the Jacobian using the functional derivative method in the inertial frame $\{0\}$.

$$J_E = \begin{pmatrix} JV_E \\ J\omega_E \end{pmatrix} = \begin{pmatrix} \frac{\partial({}^0\vec{r}_{OE})}{\partial(q_1)} & \frac{\partial({}^0\vec{r}_{OE})}{\partial(q_2)} \\ \frac{\partial({}^0\vec{\omega}_2)}{\partial(\dot{q}_1)} & \frac{\partial({}^0\vec{\omega}_2)}{\partial(\dot{q}_2)} \end{pmatrix} = \begin{pmatrix} -\mathrm{L}_2\sin(q_1+q_2)-L_1\sin(q_1) & -\mathrm{L}_2\sin(q_1+q_2) \\ \mathrm{L}_2\cos(q_1+q_2)+L_1\cos(q_1) & \mathrm{L}_2\cos(q_1+q_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{pmatrix} \tag{27}$$

Although the Jacobian J_E has 6 rows, only the first two rows are needed for the conversion in Equation 24 since the robot has only two joints to achieve a 2-DOF task. It is not possible to independently change the orientation of the end-effector.
Thus the J term in Equation 24 is just:

$$J = \begin{pmatrix} -\mathrm{L}_2\sin(q_1+q_2)-L_1\sin(q_1) & -\mathrm{L}_2\sin(q_1+q_2) \\ \mathrm{L}_2\cos(q_1+q_2)+L_1\cos(q_1) & \mathrm{L}_2\cos(q_1+q_2) \end{pmatrix} \tag{28}$$

The control sequence for Task 3 can be viewed in Figure 17

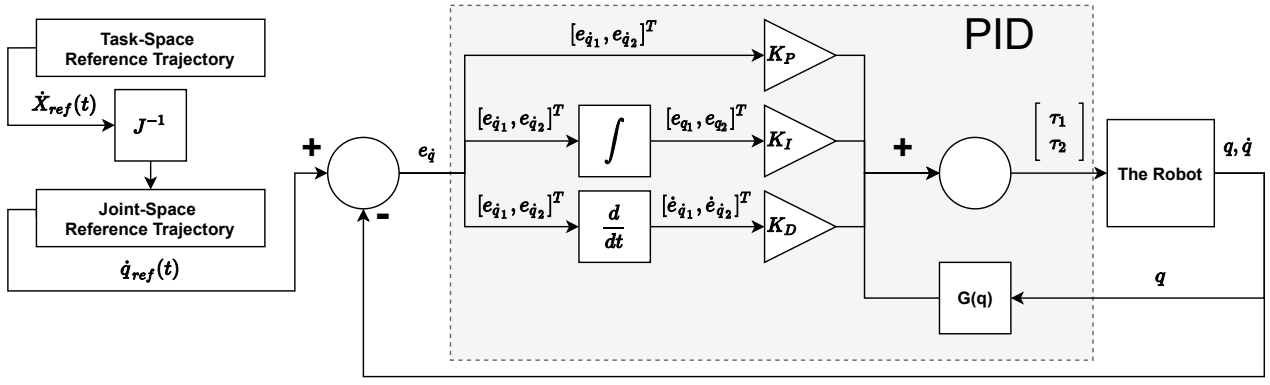


Figure 17: Task-space reference for joint velocity commanded robot flowchart

4.2 Simulation using task-space Trajectory A

Trajectory A is defined by the following parameters.

- $(x_i, y_i) = (0.71, 1.08), \quad (\dot{x}_i, \dot{y}_i) = (0, 0)$ at $t_i = 0s$
- $(x_f, y_f) = (1.485, 0.041), \quad (\dot{x}_f, \dot{y}_f) = (0, 0)$ at $t_f = 5s$

The simulation can be started by running the MATLAB script "Task3_main.m" with reference A set as the main loop reference.
The plots of the end-effector position x vs t and y vs t for Trajectory A are produced in Figure 18 and Figure 19 respectively. Reference trajectories x_{ref} and y_{ref} are superimposed on the graphs for assessing the controller performance.

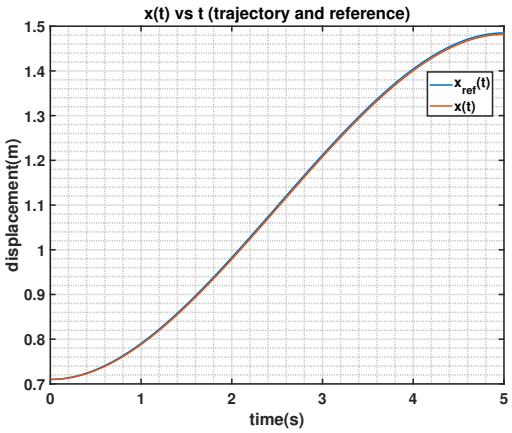


Figure 18: $x(t), x_{ref}(t)$ vs t

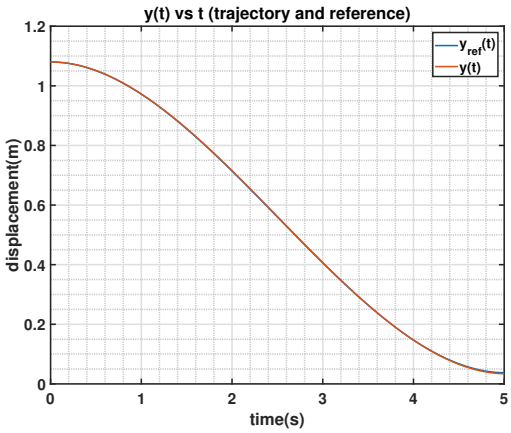
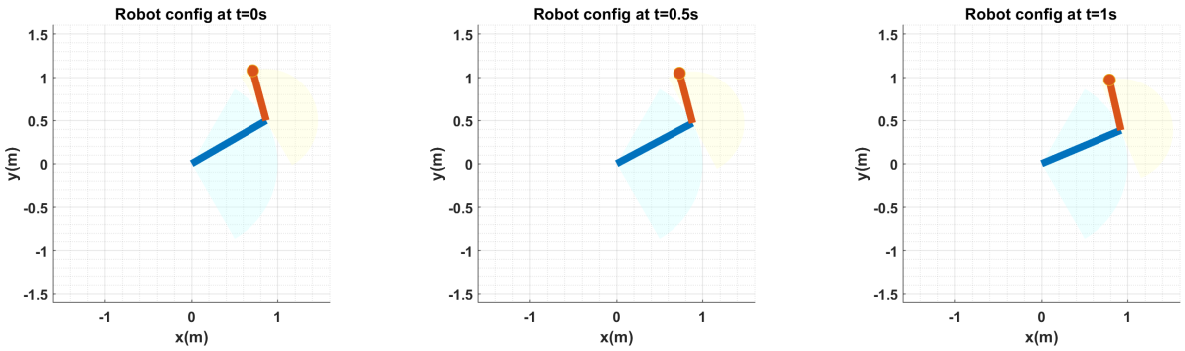


Figure 19: $y(t), y_{ref}(t)$ vs t

It can be seen in Figure 18 and Figure 19 that the robot can track the $y_{ref}(t)$ very well but the tracking performance of $x(t)$ is not satisfying. The error $x_{ref} - x_r$ is small but noticeable.
The animation of the simulation can be found as "animation/task3a.gif".For the ease of documentation purposes, the XY plot of the robot is given for every 0.5s in Figure 20. The workspace of the robot is included in the background for reference.



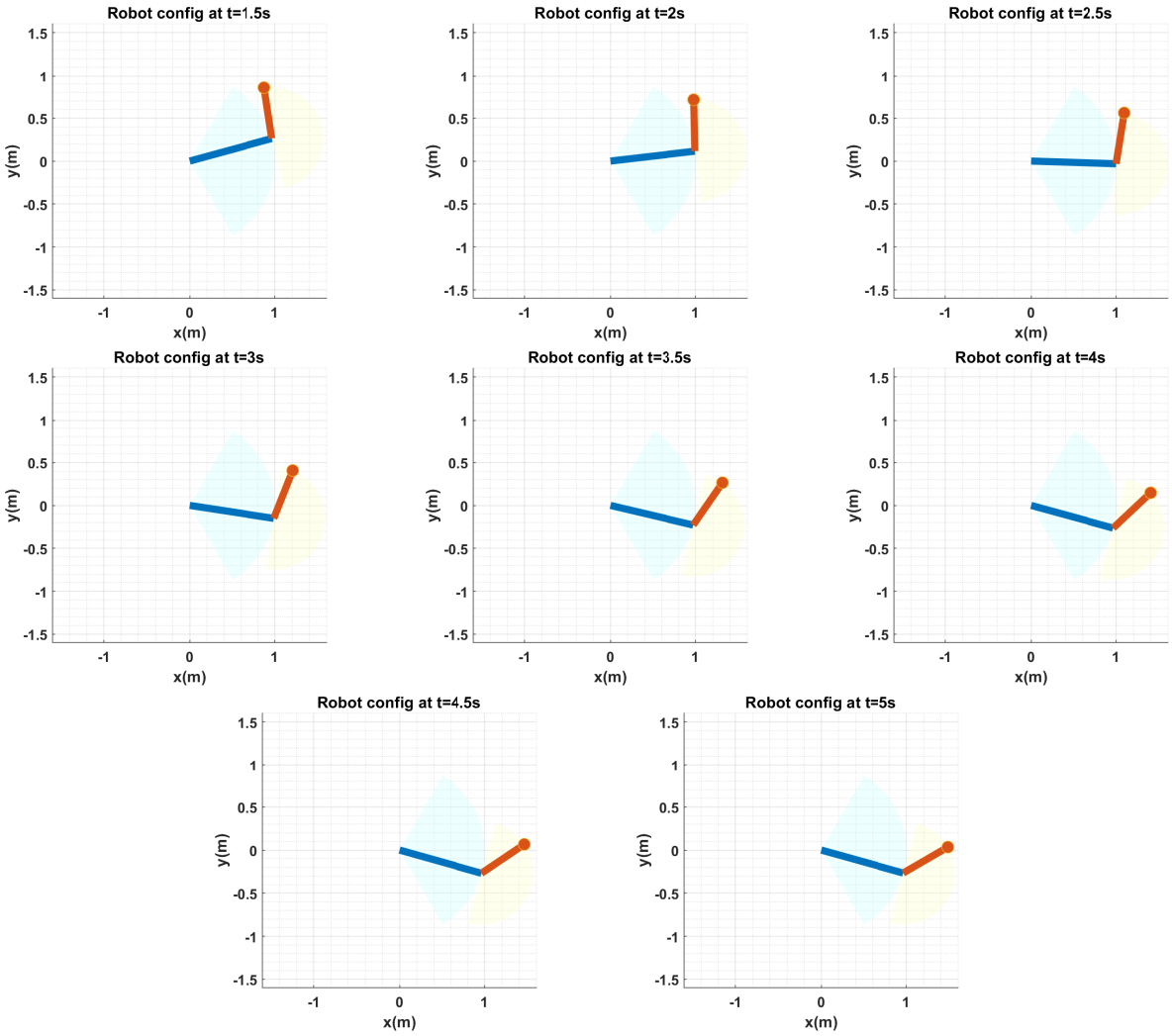


Figure 20: Trajectory A: XY plot of the robot for every $0.5s$

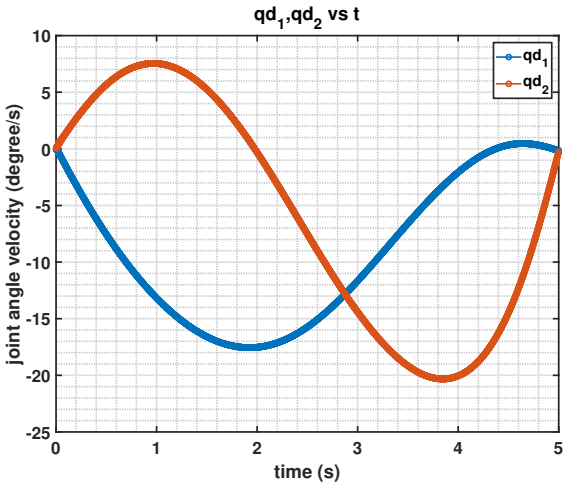


Figure 21: joint velocity vs time

The motion differs from that in Task 2 (Figure 14).

- In Task 2 the cubic polynomials were constructed for the joint space, so the joint displacements in Task 2 are a cubic functions of time (Figure 8). The joint velocities are thus quadratic (Figure 9).
- In Task 3 the cubic polynomials were constructed for the task space displacement. The joint displacement is no longer cubic as the joint velocity is no longer quadratic (21). A more direct comparison can be made by watching the animation. In Task 3, the link 2 only starts turning significantly when it is near the destination, but in task 2 the turning of Link2 is more seamless.
- The lacking of the x_{ref} tracking performance could be explained by the newly introduced joint limits. From Figure 20 we could observe that Link 2 has reached the $+90°$ limit during the course of the action. The robot's dynamics function "Task3_dynamics.m" would keep q_2 within the joint limits and reset the joint velocity to 0 if it hits the boundary limit.
- The robot's PID control only considers the velocity command it is issued, where the velocity command is directly converted from the task space velocity reference using the Jacobian. Without feedback on the robot's task space displacement, the robot's tracking error could accumulate over time.

4.3 Simulation using task-space Trajectory B

Trajectory B is defined by the following parameters. Note that Trajectory B is deliberately designed to pass outside the reachable workspace dictated by the joint limits of the robot.

- $(x_i, y_i) = (0.71, 1.48), (\dot{x}_i, \dot{y}_i) = (0, 0)$ at $t_i = 0s$
- $(x_f, y_f) = (1.39, -0.59), (\dot{x}_f, \dot{y}_f) = (0, 0)$ at $t_f = 5s$

The simulation can be started by running the MATLAB script "Task3_main.m" with reference B set as the main loop reference. The plots of the end-effector position x vs t and y vs t for Trajectory A are produced in Figure 22 and Figure 23 respectively. The reference trajectory x_{ref} and y_{ref} are superimposed on the plot.

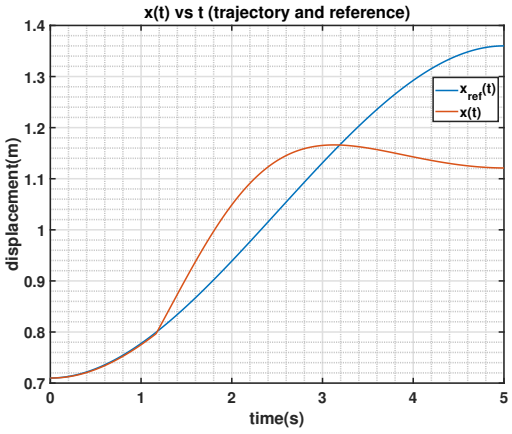


Figure 22: $x(t), x_{ref}(t)$ vs t

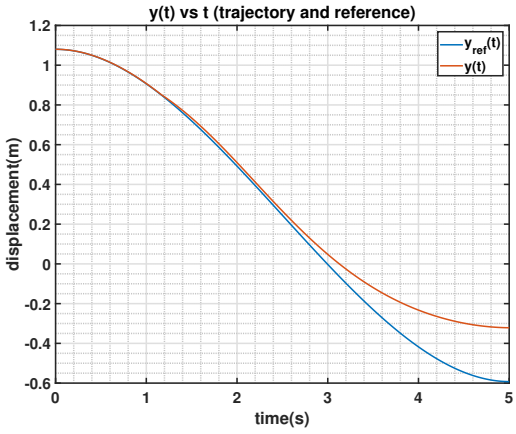
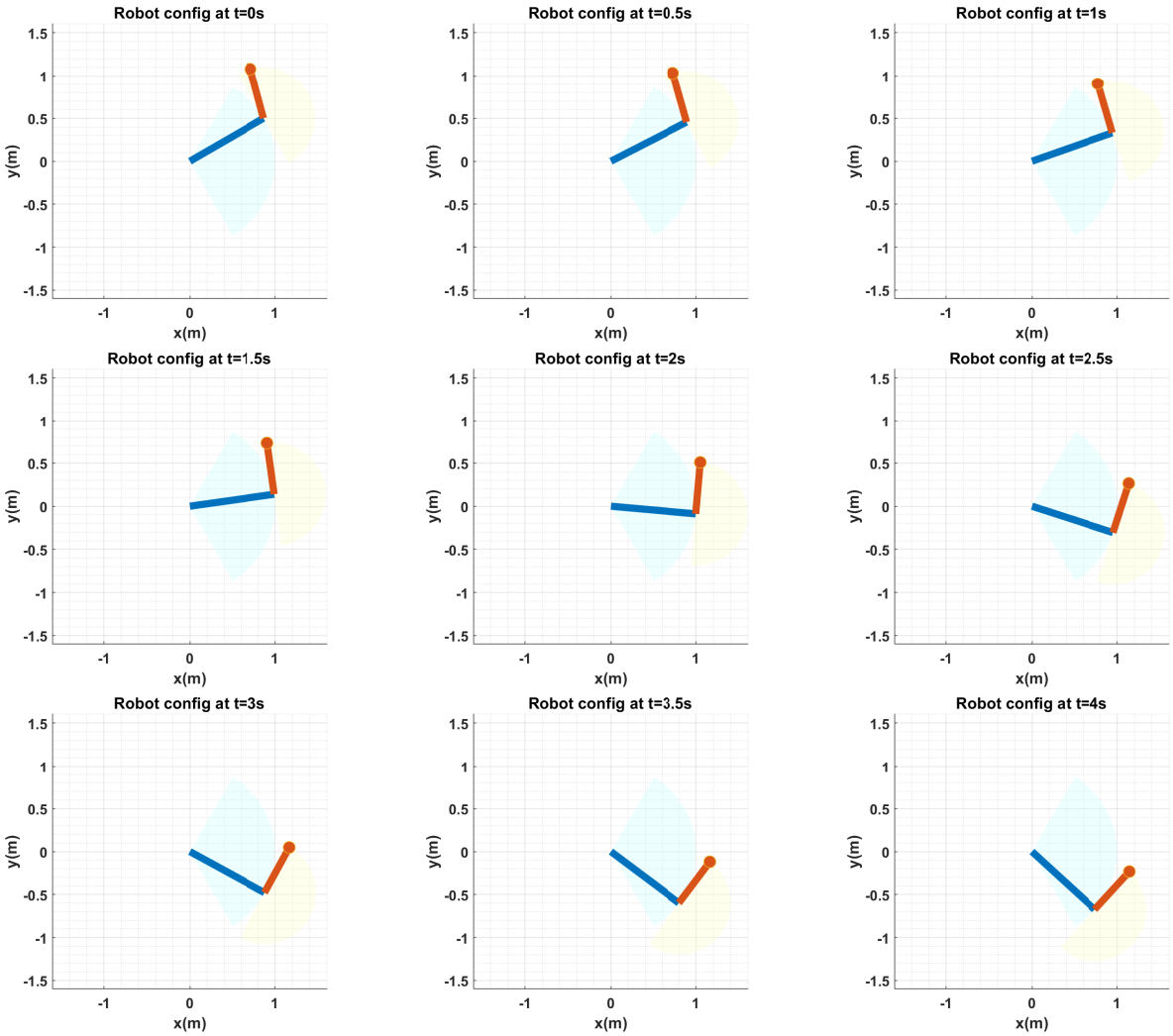


Figure 23: $y(t), y_{ref}(t)$ vs t

It can be seen in 22 and Figure 23 that the robot could not track the reference during the simulation, and that the task space tracking error is indeed accumulating as previously speculated. Since Trajectory B is designed to pass outside the reachable workspace dictated by the joint limits of the robot, the robot will "hit" the joint boundary limit multiple times during the trajectory, where the velocity of the joint would be reset to 0. In the animation it is clear that q_2 has hit the cap of $+90°$ for a large portion of the trajectory. Since the robot's feedback control is only in joint space, there is no protocol to compensate for the error generated in the task space.
The animation of the simulation can be found as "animation/task3b.gif". For the ease of documentation purposes, the XY plot of the robot is given for every 0.5s in Figure 24. Robot workspace is superimposed on the plots.



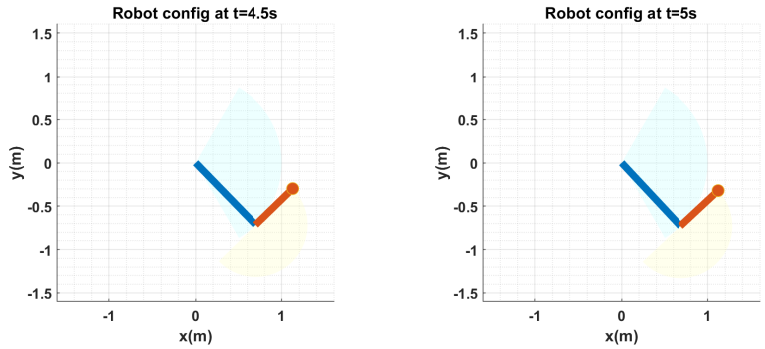


Figure 24: Trajectory B: XY plot of the robot for every $0.5s$

5 Task 4. Task space control of the velocity commanded robot

In Task 4, we would implement task space PI controllers on top of the existing velocity commanded PID control loop we established in Task 3. The end-effector Trajectory A defined in Task 3 will be used to test the control performance.

- The $x_{ref}(t)$ and $y_{ref}(t)$ signals are constructed using cubic polynomials and differentiated using the gradient function to obtain the $\dot{x}_{ref}(t)$ and $\dot{y}_{ref}(t)$ signals.
- The error signal $e_x = x_{ref} - x_r$ and $e_y = y_{ref} - y_r$ are computed at every sampling instance $dt = 0.01s$, where x_r and y_r are computed from forward kinematics.
- The task space PI controllers updates the $[\dot{x}_{ref}; \dot{y}_{ref}]$ command at every sampling instance using the following control law:

$$\begin{bmatrix} \dot{x}_C \\ \dot{y}_C \end{bmatrix} = \begin{bmatrix} \dot{x}_{ref} \\ \dot{y}_{ref} \end{bmatrix} + \begin{bmatrix} K_{Px} & 0 \\ 0 & K_{Py} \end{bmatrix} \begin{bmatrix} e_x \\ e_y \end{bmatrix} + \begin{bmatrix} K_{Ix} & 0 \\ 0 & K_{Iy} \end{bmatrix} \begin{bmatrix} \int_{t=0}^{t_k} e_x dt \\ \int_{t=0}^{t_k} e_y dt \end{bmatrix} \tag{29}$$

- The task space velocity command is converted to joint space velocity command using Equation 24

$$[\dot{q}_1(t); \dot{q}_2(t)]^T = J^{-1} \cdot [\dot{x}(t); \dot{y}(t)]^T$$

- The following parameters for the task-space PI controller were found to produce optimal tracking performance in the simulation for PID with gravitational compensation.

$$K_x = [K_{Px}; K_{Ix}] = [1; 0.02], \quad K_y = [K_{Py}; K_{Iy}] = [0.5; 0.01] \tag{30}$$

- With task space feedback, it is possible to achieve acceptable performance even without gravitational force modelling of $G(q)$, but requires much higher integral gain to decrease the tracking error.

$$K_x = [K_{Px}; K_{Ix}] = [50; 1], \quad K_y = [K_{Py}; K_{Iy}] = [25; 0.5] \tag{31}$$

- The control loop of Task 4 could be described by the following flowchart.

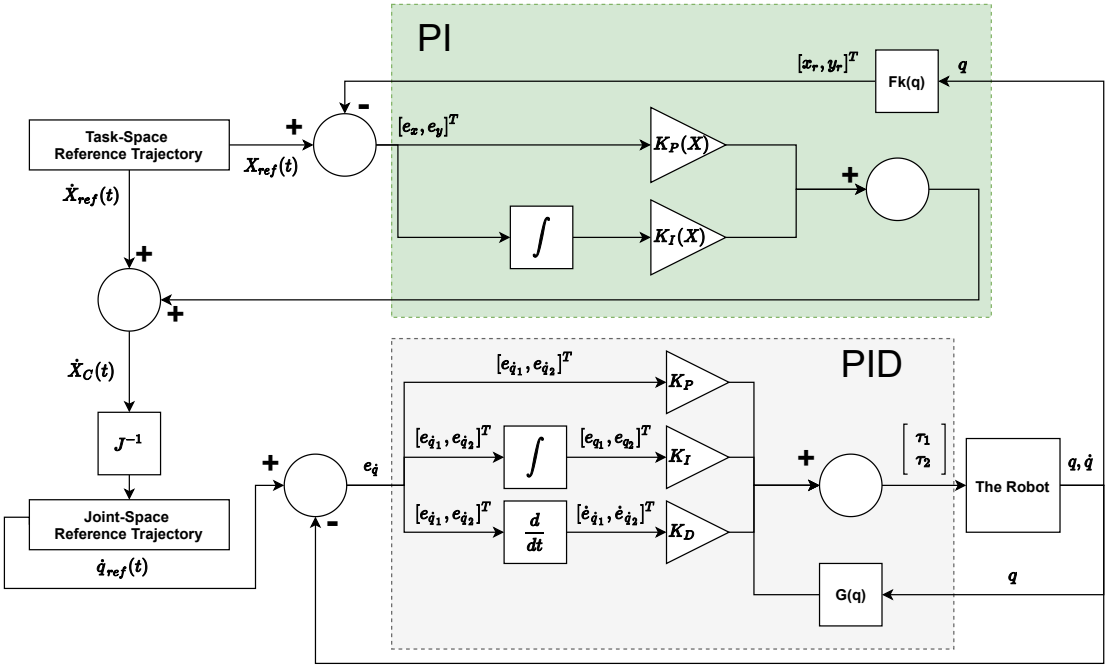


Figure 25: Task Space PI control loop flowchart

5.1 Simulation using Trajectory A

5.1.1 using PID with gravitational force compensation

The simulation can be started by running the MATLAB script "Task4_main.m".
The plots of the end-effector position x vs t and y vs t for Trajectory A are produced in Figure 26 and Figure 27 respectively.(Again using the PID with gravitational compensation.) The reference trajectory x_{ref} and y_{ref} are superimposed on the plot.

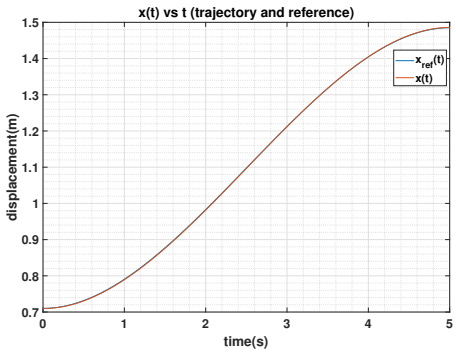


Figure 26: $x(t), x_{ref}(t)$ vs t

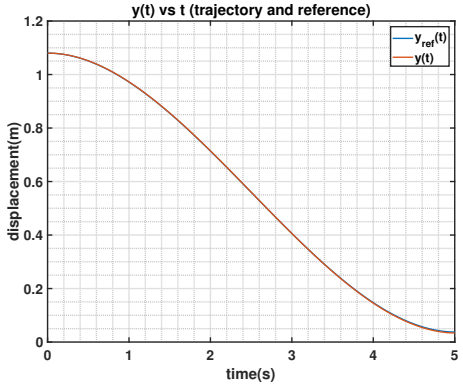


Figure 27: $y(t), y_{ref}(t)$ vs t

It can be seen in Figure 26 and Figure 27 that the robot can now track both the $x_{ref}(t)$ and $y_{ref}(t)$ very precisely as compared to Figure 18 and Figure 19 in Task 3. This is because the

task space PI controller could now compensate for the task space displacement error and adjust the velocity commands accordingly.

5.1.2 using PID without gravitational force compensation

With the task space feedback, even PID without gravitational compensation could track the trajectory reasonably well using much larger task space PI gain (31), as seen in Figure 28 and 29

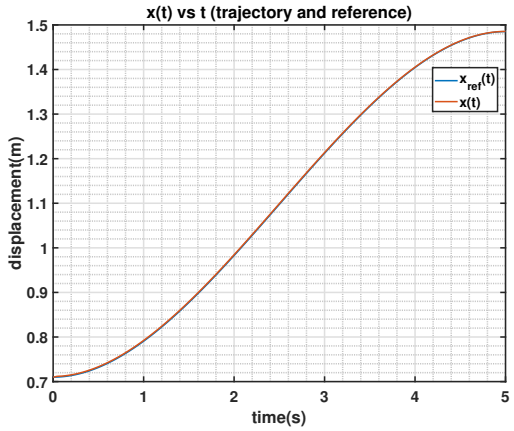


Figure 28: $x(t), x_{ref}(t)$ vs t

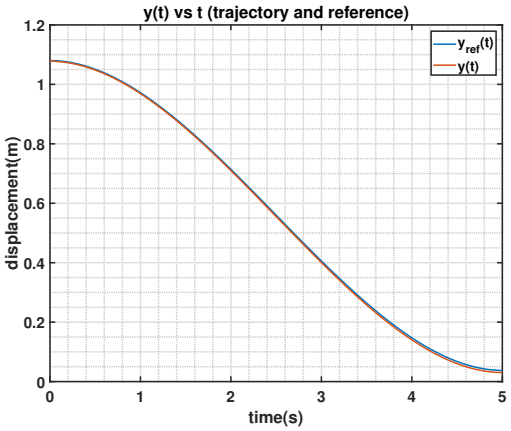


Figure 29: $y(t), y_{ref}(t)$ vs t

5.1.3 cross comparison between two PID settings

The animation of the simulation can be found as "animation/task4.gif". For the ease of documentation purposes, the XY plot of the robot is given for every 0.5s in Figure 30 and 31.

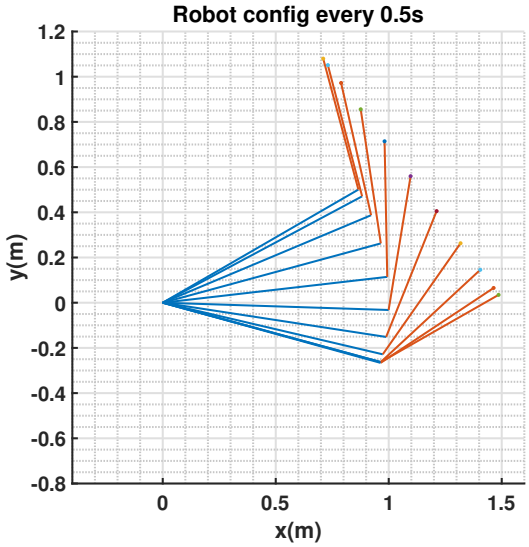


Figure 30: Robot animation (models G)

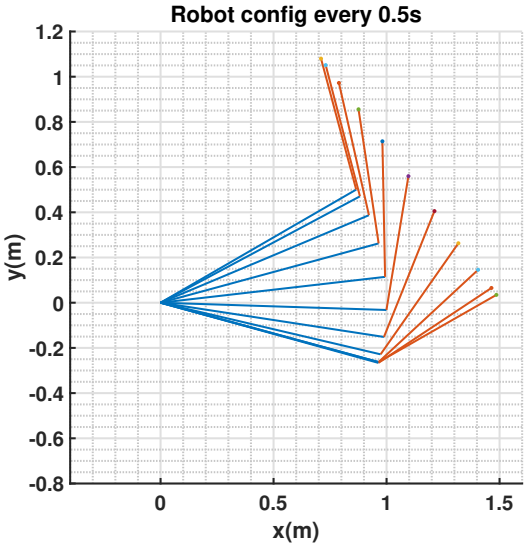


Figure 31: Robot animation (models free)

6 Task 5. Task space control of the velocity commanded robot with reactive obstacle avoidance

Task 5 continues from Task 4, where the robot is to execute Trajectory A with Task Space control of the velocity commanded robot. A circular obstacle of random size is generated somewhere near the path and the robot does not know where it is beforehand.
However, a senor function *sensor_t5.p* is provided to measure the distance of the obstacle to the end-effector and the angle of approach at each time instance.

```
[d,theta] = sensor_t5(obs_init,xr,yr)
```

Where:

- I: (**obs_init**), A binary variable. When set to 1, a new obstacle will be generated, should be set to 1 on first call and changed to 0 for one simulation.
- I: (**xr**, **yr**), x, y coordinates of the end-effector in frame $\{0\}$.
- O: (**d**), Cartesian distance from nearest point of obstacle to end-effector, effective range is within 0.35m, returns 1000 if outside of range.
- O: (**theta**), The angle to the same point from the end-effector, measured from X_0 axis of the inertial frame.

A diagram is provided below to illustrate the context of this task. The obstacle is assumed to be always a circle.

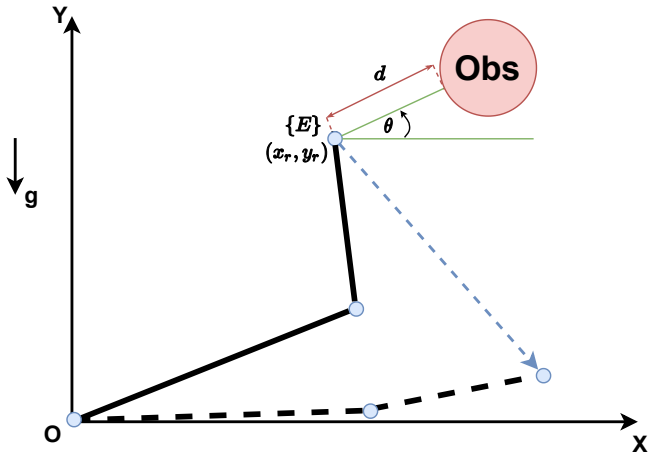


Figure 32: Obstacle detection during trajectory traversal diagram.

6.1 Reactive obstacle avoidance strategies

The generated obstacle could create three scenarios for the robot during its traversal of the original planned trajectory, depending on the location of spawn and the size of the radius.

1. The obstacle is completely obstructive that no possible trajectory exists between the initial starting point to the end point given the robot's defined joint limits.

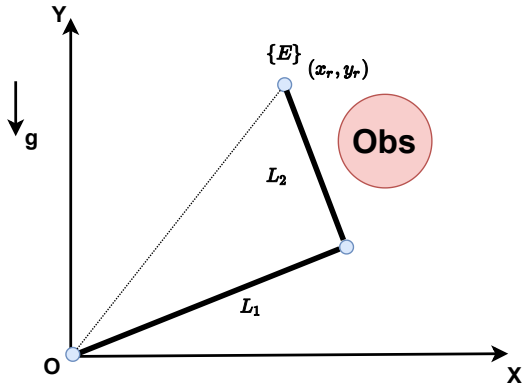


Figure 33: Minimum distance of end-effector from origin

Since the joint limit of $-90° \leq q_2 \leq 90°$, the minimum distance the end-effector could keep from the origin is the length of hypotenuse of the right-side triangle formed by Link 2 and Link 1 of the robot. (Figure 33)

$$dist_{min} = \sqrt{L_1^2 + L_2^2} = \sqrt{1^2 + 0.6^2} \approx 1.166m \tag{32}$$

If any point on the obstacle is detected to be less than $dist_{min}$ during the motion, then it is necessary to stop the robot motion by setting $\dot{q}_{ref}$ to 0.

2. The obstacle is obstructive to the planned trajectory but the robot could an alternative path exists within the robot's reachable workspace to evade a potential collision. This occurs when obstacle is obstructive to the straight line segment between the start and destination of Trajectory A. (Figure 34)

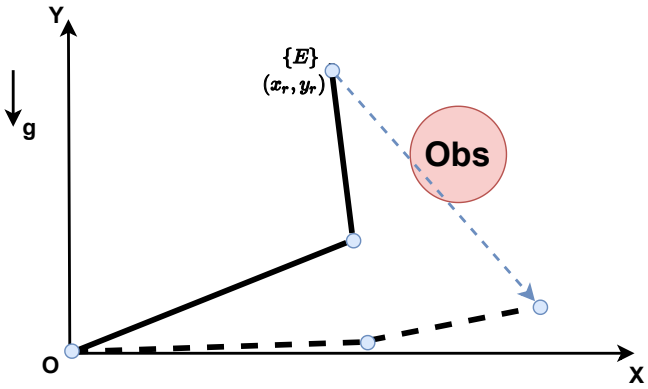


Figure 34: Avoidable obstructive obstacle

We could use a modified Assignment 3 Task 3 approach to first map out the obstacle circle and then find a intermediate via point D to construct an updated trajectory to the destination while adhering to the 5 seconds time frame.

3. The obstacle is clear of the planned trajectory and no evasive manoeuvre is needed. This case is trivial as no action is required.

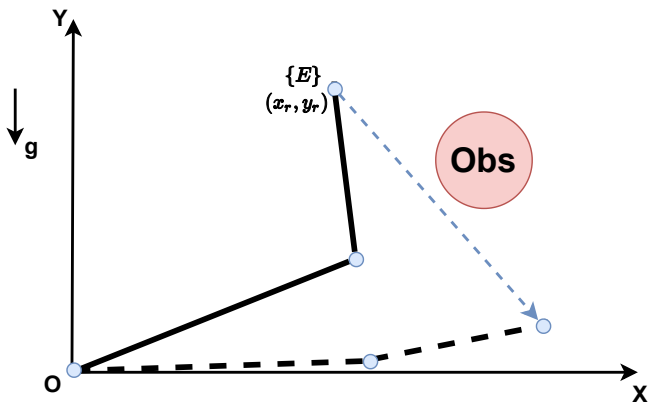


Figure 35: Non-obstructive obstacle

6.1.1 Obstacle mapping and safe distance threshold

At every time sampling instant we could map a point of the obstacle in task space using the $sensor\_t5$ function.

$$x_{Obs} = x_r + d\cos\theta, \quad y_{Obs} = y_r + d\sin\theta \tag{33}$$

Using this function over multiple sampling instances could provide the robot with a collection of the obstacle boundaries. If we detect that the distance $d < 0.005m$, the robot's motor's joint velocity commands will be set to 0 to avoid a total collision.
Since we know that the obstacle is circular, we could use at least 3 collected obstacle points to reconstruct the equation of the circle of the obstacle. For multiple points, it is possible to use a least square fitting method to fit the collected data points to a circle. (Figure 36)
A function written by Bucher izhak in 1991 was adapted to fulfill this task conveniently. (Bucher 2004 (accessed May 16 2020))

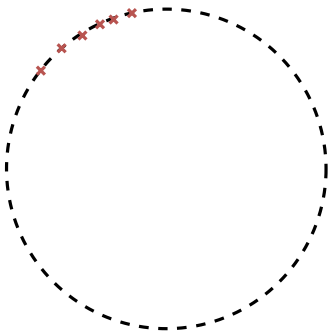


Figure 36: Least Square Fitting of obstacle circle

A total of 50 obstacle points $0.5s$ are collected during every simulation to map the obstacle circle. Then we could use the information to decide how to plan and execute the evasive trajectory.

6.1.2 Path re-planning using a trajectory via point

Once the obstacle circle is mapped out, we will apply the same method used in Assignment 3 Task 3 to find a task-space viapoint then reconstruct the task-space reference trajectory using cubic polynomials to first arrive at the viapoint and then leave from the viapoint to the destination. We will briefly go over the method of finding the viapoint explained in Assignment 3 Task 2.

A commonly used algorithm to determine the shortest trajectory for mobile robots and circular modelled obstacles is to search for the shortest path over the collision-free tangent graph as explained by (Alexopoulos and Griffin 1992).
An example of such a collision free path is presented in Figure 37.

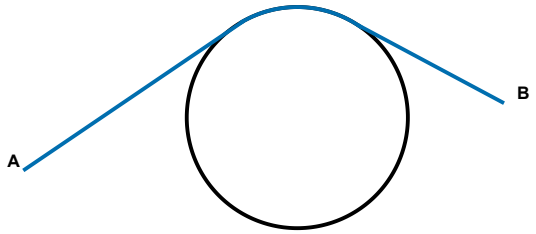


Figure 37: Example of shortest path between two points around a circular obstacle

For straight line segment trajectory using one viapoint, the minimum distance trajectory requires the viapoint to be at the intersection between the two tangent lines formed respectively by the starting point with the circle and the finishing point with the circle.

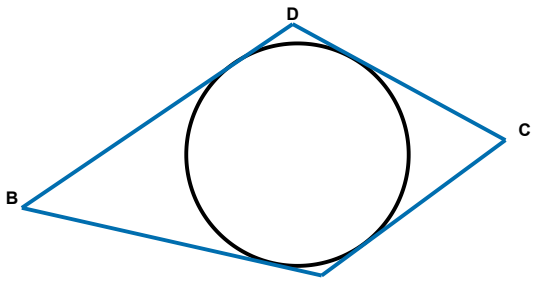


Figure 38: Example of optimal point D

i.e, Let point B be the starting point and point C to be the finishing point, the viapoint D then needs to be found at the intersection of tangential lines BD and CD (Figure 38). Point B can form two tangent lines to the Obs1, and so does point D. In total there are up to 4 intersection points formed if all point B tangents and all point C tangents are not parallel. But only the two closest to the circle are relevant and only the point under the obstacle is relevant for this task. The function $[pD1, pD2] = findviapoint(pB, pC, pOBS, ro)$ from Assignment 3 was recycled to implement this task. It will first generate the equation of the line connecting the the starting point pB to pC. Then the equation of the line and the equation of the obstacle circle would be solved to see if there is any collision points. In the case of a collision or if the obstacle circle is below the trajectory path, the bottom via-point candidates are computed.
For safety precautions, the viapoint is computed by adding a safety offset of $0.025m$ to the

mapped obstacle circle radius. The via point will be shifted to the left-most x-coordinate of the obstacle circle $x_{Obs} - r_{Obs} - 0.05$ if its x-coordinate is greater than $x_{Obs} - r_{Obs}$ to minimise the risk of linkage collision with the obstacle.

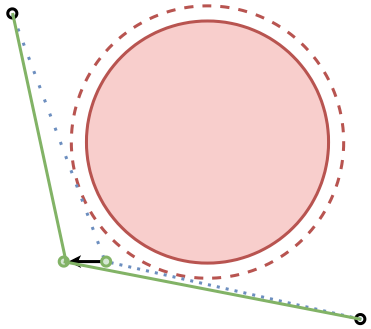


Figure 39: The viapoint shift

This approach is simpler than Assignment 3 implementation and does not check the Ik validity of the new trajectory, thus the minimum distance fail-safe serves to stop the robot if necessary.

6.2 Testing of algorithm using Trajectory A and arbitrarily generated obstacles

6.2.1 An successful obstacle avoidance trajectory:

This is an example of a successful obstacle avoidance trajectory, where the obstacle circle is very well mapped and outside of the $1.167m$ dead zone.It can be seen in Figure 41 that the distance from the obstacle to the end-effector remained above 0.

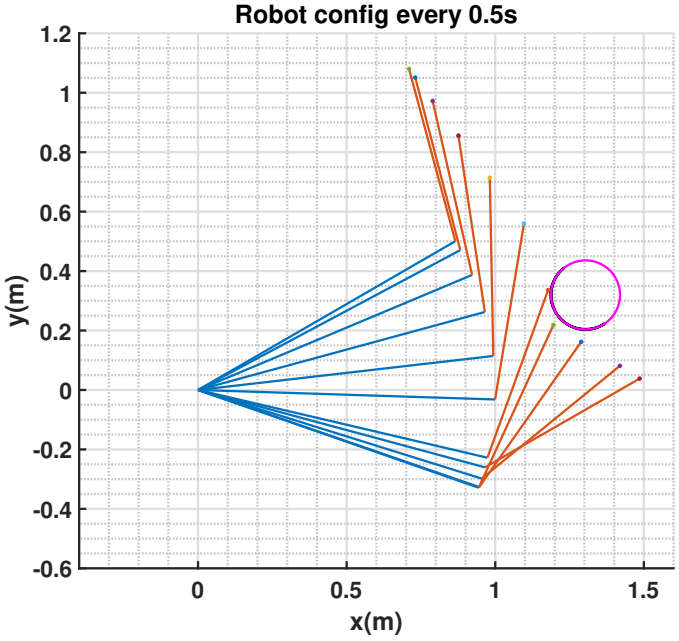


Figure 40: Example of a successful obstacle avoidance trajectory

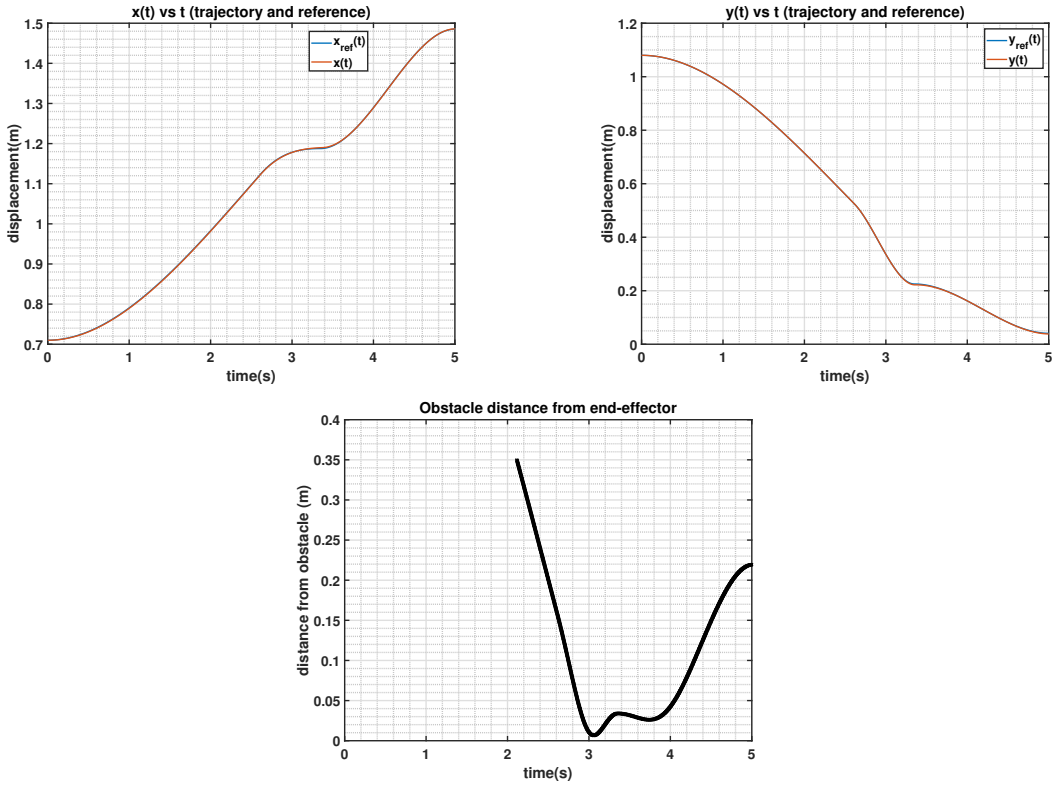


Figure 41: Trajectory feature plots collections

6.2.2 An unsuccessful but not catastrophic obstacle avoidance trajectory:

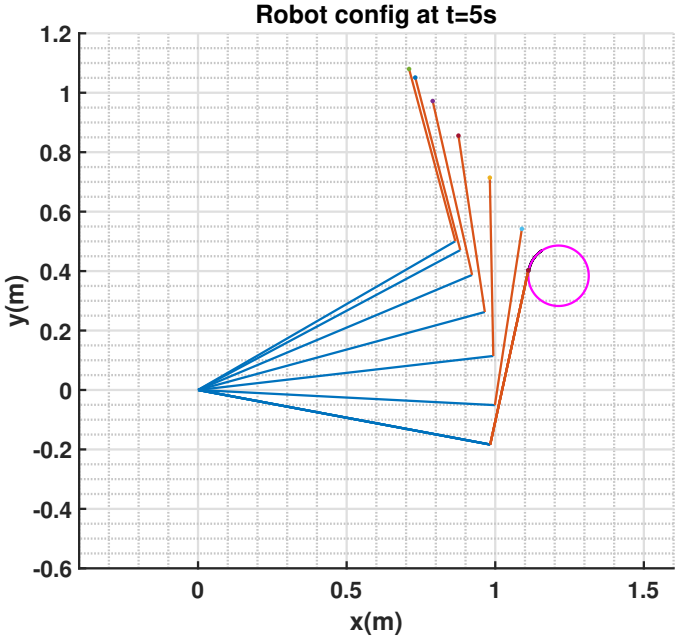


Figure 42: Example of a successful obstacle avoidance trajectory

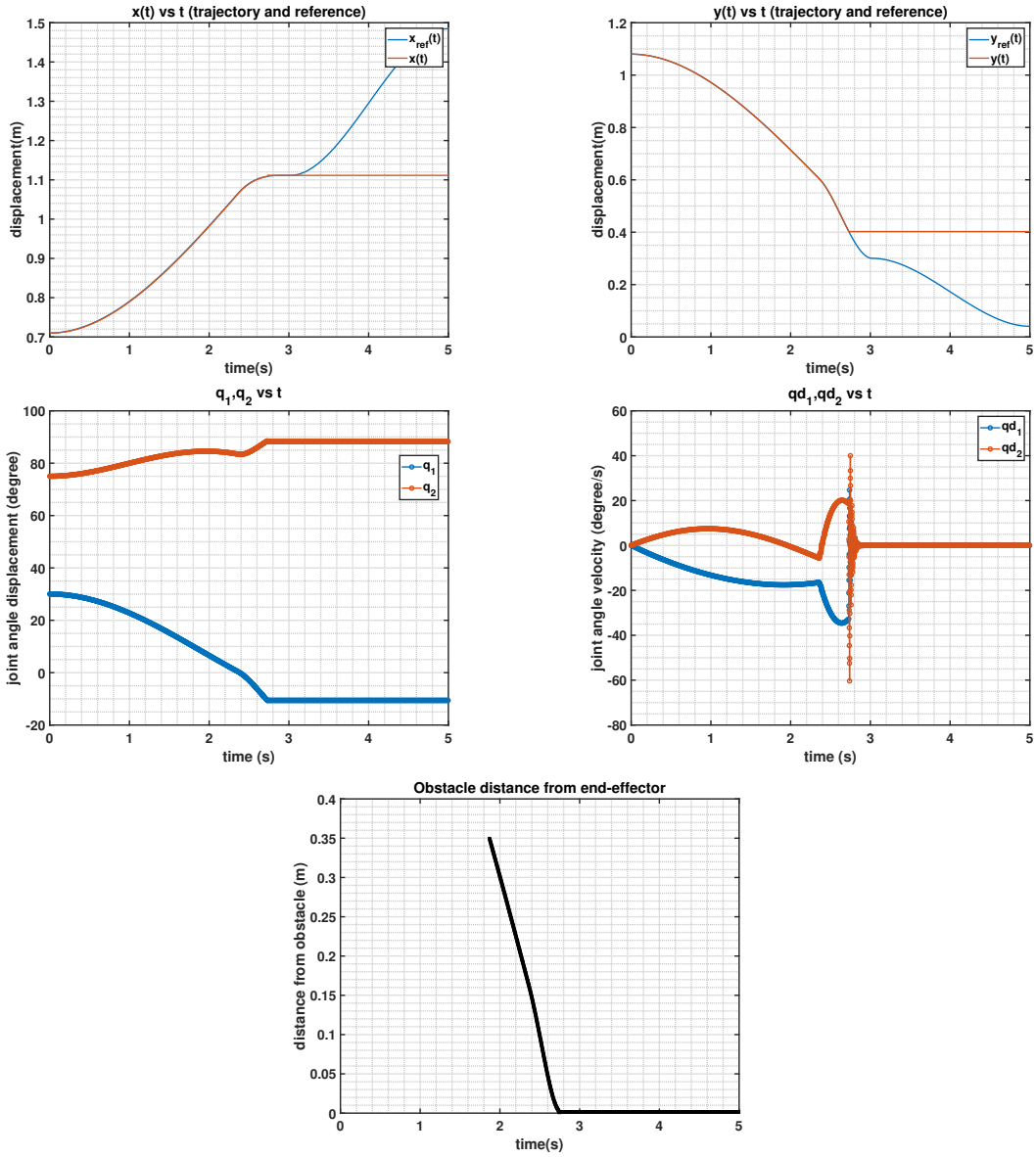


Figure 43: Trajectory feature plots collections

In this example the robot fails to track the obstacle avoidance trajectory as it has hit the joint limit of $q_2 \leq 90°$ during the manoeuvre. The $0.005m$ fail-safe stopped the robot right before the collision, zoom into the obstacle distance subplot in Figure 43 to see that the distance from the obstacle to the end-effector remained above 0.

6.3 Algorithm drawback and possible future improvements

This trivial algorithm relies heavily on the correct mapping of the obstacle circle in the 0.5 second time frame before triggering the fail-safe. It also does not actively check the potential collision of links against the obstacle. It is possible to map the obstacle incorrectly when the collected data points are too close, such as the following example. (Figure 44). In practical

applications obstacles are less likely to be perfectly circular, and it is best practice to use more reactive methodology to generate a feedback task space velocity reference when obstacle to one of the joints. This will make the original timing difficult to achieve but is a very good safety measure if foreign objects entered the robot work-space.

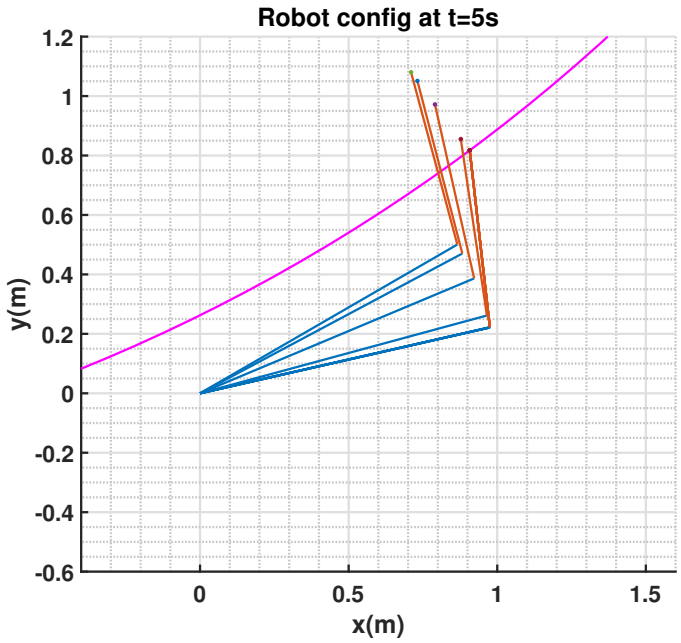


Figure 44: Example of a complete failure

7 Conclusion

We have produced a simulation environment using task space controllers to control a velocity commanded 2-D planar revolute-revolute robot. We compared the behavior of joint space trajectory and task space trajectory and implemented a trivial obstacle avoidance algorithm to avoid circular obstacle along the planned trajectory using via-point generation. Since the algorithm is still based from an offline methodology, the timing is guaranteed but is not good for practical applications.

References

[1] Christos Alexopoulos and Paul M Griffin. "Path planning for a mobile robot". In: *IEEE Transactions on systems, man, and cybernetics* 22.2 (1992), pp. 318–322.

[2] Izhak Bucher. *Least Square Circle fits*. 2004 (accessed May 16 2020). URL: https://www.mathworks.com/matlabcentral/fileexchange/5557-circle-fit.

[3] John G Ziegler, Nathaniel B Nichols, et al. "Optimum settings for automatic controllers". In: *trans. ASME* 64.11 (1942).