Task 1. Construction of a Robot Dynamic Simulator

Task 1.1 equation of motion

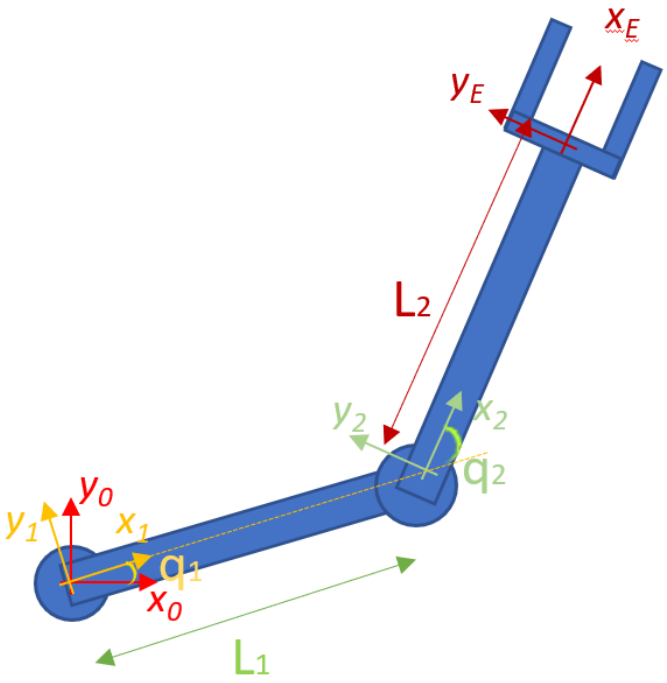

Figure 1Diagram of the robotic arm

The DH table for the manipulator is :

	D_x	R_x	Dz	Rz
i	a_{i-1}	α_{i-1}	d_{i-1}	θ_{i-1}
1	0	0	0	q_1
2	L_1	0	0	q_2
E	L_2	0	0	0

Figure 2 DH table for the 2-arm robot

With $L_1 = 1.0$ and $L_2 = 0.6$, q_1 and q_2 being a variable for our robot

With the notation ${}^0z_1$ meaning normalized z-axis of frame 1 in frame 0 and

${}^0p_{1\_mass1}$ meaning distance of frame 1 to mass of link 1 in frame 0

Jacobian matrices for the centres of masses are

$$\mathrm{J_{mass1}} = \begin{bmatrix} \{{}^0z_1 \times {}^0p_{1\_mass1}\}^T & 0 \\ {}^0z_1 & 0 \end{bmatrix} = \begin{bmatrix} \{(0,0,1) \times (r_{C1}\cos(q_1), r_{C1}\sin(q_1), 0)\}^T & \vec{0} \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\mathrm{J_{mass1}} = \begin{bmatrix} -r_{C1}\sin(q_1) & 0 \\ r_{C1}\cos(q_1) & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \quad (1)$$

$$\mathrm{J}_{\mathrm{mass2}} = \begin{bmatrix} \{{}^0z_1 \times {}^0p_{1\_mass2}\}^T & \{{}^0z_2 \times {}^0p_{2\_mass2}\}^T \\ {}^0z_1 & {}^0z_2 \end{bmatrix}$$

$$= \begin{bmatrix} \{(0,0,1) \times (r_{C2}\cos(q_1+q_2) + L_1\cos(q_1), r_{C2}\sin(q_1+q_2) + L_1\sin(q_1), 0)\}^T & \{(0,0,1) \times (r_{C2}\cos(q_1+q_2), r_{C2}\sin(q_1+q_2), 0)\}^T \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix}$$

$$\mathrm{J}_{\mathrm{mass2}} = \begin{bmatrix} -r_{C2}\sin(q_1+q_2) - L_1\sin(q_1) & -r_{C2}\sin(q_1+q_2) \\ r_{C2}\cos(q_1+q_2) + L_1\cos(q_1) & r_{C2}\cos(q_1+q_2) \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 1 \end{bmatrix} \quad (2)$$

With

$$\mathrm{J}_{\mathrm{v\_mass1}} = \text{first three row of } \mathrm{J}_{\mathrm{mass1}}, \mathrm{J}_{\omega\_\mathrm{mass1}} = \text{bottom three row of } \mathrm{J}_{\mathrm{mass1}}$$

$$\mathrm{J}_{\mathrm{v\_mass2}} = \text{first three row of } \mathrm{J}_{\mathrm{mass2}}, \mathrm{J}_{\omega\_\mathrm{mass2}} = \text{bottom three row of } \mathrm{J}_{\mathrm{mass2}}$$

Considering the equation of motion of end effector be

$$\tau = A\ddot{q} + B(q,\dot{q}) + G(q) \quad (3)$$

Matrix A in the motion of equation is found by :

$$A = m_1 * \mathrm{J}_{\mathrm{v\_mass1}}^T * \mathrm{J}_{\mathrm{v\_mass1}} + \mathrm{J}_{\omega\_\mathrm{mass1}}^T * I_1 * \mathrm{J}_{\omega\_\mathrm{mass1}} + m_2 * \mathrm{J}_{\mathrm{v\_mass2}}^T * \mathrm{J}_{\mathrm{v\_mass2}} + \mathrm{J}_{\omega\_\mathrm{mass2}}^T * I_2 * \mathrm{J}_{\omega\_\mathrm{mass2}} \quad (4)$$

$m * \mathrm{J}_{\mathrm{v\_mass}}^T * \mathrm{J}_{\mathrm{v\_mass}}$ is the contribution from linear component, while $\mathrm{J}_{\omega\_\mathrm{mass}}^T * I * \mathrm{J}_{\omega\_\mathrm{mass}}$ is the contribution from the angular component

With I_1 and I_2 representing

$$I_1 = \begin{bmatrix} I_{xx1} & 0 & 0 \\ 0 & I_{yy1} & 0 \\ 0 & 0 & I_{zz1} \end{bmatrix} \quad (5)$$

$$I_2 = \begin{bmatrix} I_{xx2} & 0 & 0 \\ 0 & I_{yy2} & 0 \\ 0 & 0 & I_{zz2} \end{bmatrix} \quad (6)$$

Substituting equation (1) and (2) into (4) yields

$$A = \begin{bmatrix} \mathrm{m}_2L_1^2 + 2\mathrm{m}_2\mathrm{L}_1r_{C2}\cos(\mathrm{q}_2) + \mathrm{m}_1{r_{C1}}^2 + \mathrm{m}_2{r_{C2}}^2 + \mathrm{I}_{zz1} + \mathrm{I}_{zz2} & \mathrm{m}_2{r_{C2}}^2 + \mathrm{L}_1\mathrm{m}_2r_{C2}\cos(\mathrm{q}_2) + \mathrm{I}_{zz2} \\ \mathrm{m}_2{r_{C2}}^2 + \mathrm{L}_1 * \mathrm{m}_2r_{C2}\cos(\mathrm{q}_2) + \mathrm{I}_{zz2} & \mathrm{m}_2{r_{C2}}^2 + \mathrm{I}_{zz2} \end{bmatrix} \quad (7)$$

Consider the notation of $A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}$ and $a_{ijk} = \frac{\partial a_{ij}}{\partial q_k}$

With that we can construct the Christoffel symbols $b_{i,jk} = \frac{1}{2}(a_{ijk} + a_{ikj} - a_{jki})$,

With the hope of Christoffel symbol, the matrix $b(q, \dot{q})$ in the EoM can be broke down into matrix B and C in the relationship:

$$b(q, \dot{q}) = B(q)[\dot{q}\dot{q}] + C(q)[\dot{q}^2] \tag{8}$$

$$B(q) = \begin{bmatrix} 2b_{1,12} \\ 2b_{2,12} \end{bmatrix} = \begin{bmatrix} -2 * \mathrm{L_1 m_2 r_{C2} sin(q_2)} \\ 0 \end{bmatrix} \tag{9}$$

$$C(q) = \begin{bmatrix} \mathrm{b_{1,11}} & \mathrm{b_{1,22}} \\ \mathrm{b_{2,11}} & \mathrm{b_{2,22}} \end{bmatrix} = \begin{bmatrix} 0 & -\mathrm{L_1 m_2 r_{C2} sin(q_2)} \\ \mathrm{L_1 m_2 r_{C2} sin(q_2)} & 0 \end{bmatrix} \tag{10}$$

$$[\dot{q}\dot{q}] = [\dot{q_1}\dot{q_2}] \tag{11}$$

$$[\dot{q}^2] = \begin{bmatrix} \dot{q_1^2} \\ \dot{q_2^2} \end{bmatrix} \tag{12}$$

Therefore we can solve equation (8) by substituting in with equation (9-12) is :

$$b(q, \dot{q}) = B(q)[\dot{q}\dot{q}] + C(q)[\dot{q}^2]$$

$$= \begin{bmatrix} -2\mathrm{L_1 m_2 r_{C2} sin(q_2)} \\ 0 \end{bmatrix} [\dot{q_1}\dot{q_2}] + \begin{bmatrix} 0 & -\mathrm{L_1 m_2 r_{C2} sin(q_2)} \\ \mathrm{L_1 m_2 r_{C2} sin(q_2)} & 0 \end{bmatrix} \begin{bmatrix} \dot{q_1^2} \\ \dot{q_2^2} \end{bmatrix}$$

$$b(q, \dot{q}) = \begin{bmatrix} -L_1 m_2 \dot{q_2} r_{c2} \sin(q_2) * (2\dot{q_1} + \dot{q_2}) \\ L_1 m_2 \dot{q_1^2} r_{C2} \sin(q_2) \end{bmatrix} \tag{14}$$

Matrix G is with the consideration of gravity is downwards on y axis:

$$G = J_{v\_mass1}^T * \begin{bmatrix} 0 \\ m_1 g \\ 0 \end{bmatrix} + J_{v\_mass2}^T * \begin{bmatrix} 0 \\ m_2 g \\ 0 \end{bmatrix}$$

$$G = \begin{bmatrix} g m_2 (r_{C2} \cos(q_1 + q_2) + L_1 \cos(q_1)) + g m_1 r_{C1} \cos(q_1) \\ g m_2 r_{C2} \cos(q_1 + q_2) \end{bmatrix} \tag{15}$$

With matrix A, b and G known to us, substituting equation (7),(14) and (15) in the equation (3) can give us an equation of motion that can be numerically calculated.

$$\begin{bmatrix} \tau_1 \\ \tau_2 \end{bmatrix} = \begin{bmatrix} \mathrm{m_2 L1^2 + 2L_1 r_{C2} m_2 cos(q_2) + m_1 r_{C1}{}^2 + m_2 r_{C2}{}^2 + I_{zz1} + I_{zz2}} & \mathrm{m_2 r_{C2}{}^2 + L_1 m_2 r_{C2} cos(q_2) + I_{zz2}} \\ \mathrm{m_2 r_{C2}{}^2 + L_1 * m_2 r_{C2} cos(q_2) + I_{zz2}} & \mathrm{m_2 r_{C2}{}^2 + I_{zz2}} \end{bmatrix} \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \end{bmatrix} \tag{16}$$
$$+ \begin{bmatrix} -L_1 m_2 \dot{q_2} r_{c2} \sin(q_2) * (2\dot{q_1} + \dot{q_2}) \\ L_1 m_2 \dot{q_1^2} r_{C2} \sin(q_2) \end{bmatrix}$$
$$+ \begin{bmatrix} g m_2 (r_{C2} \cos(q_1 + q_2) + L_1 \cos(q_1)) + g m_1 r_{C1} \cos(q_1) \\ g m_2 r_{C2} \cos(q_1 + q_2) \end{bmatrix}$$

Task 1.2 Free swing condition

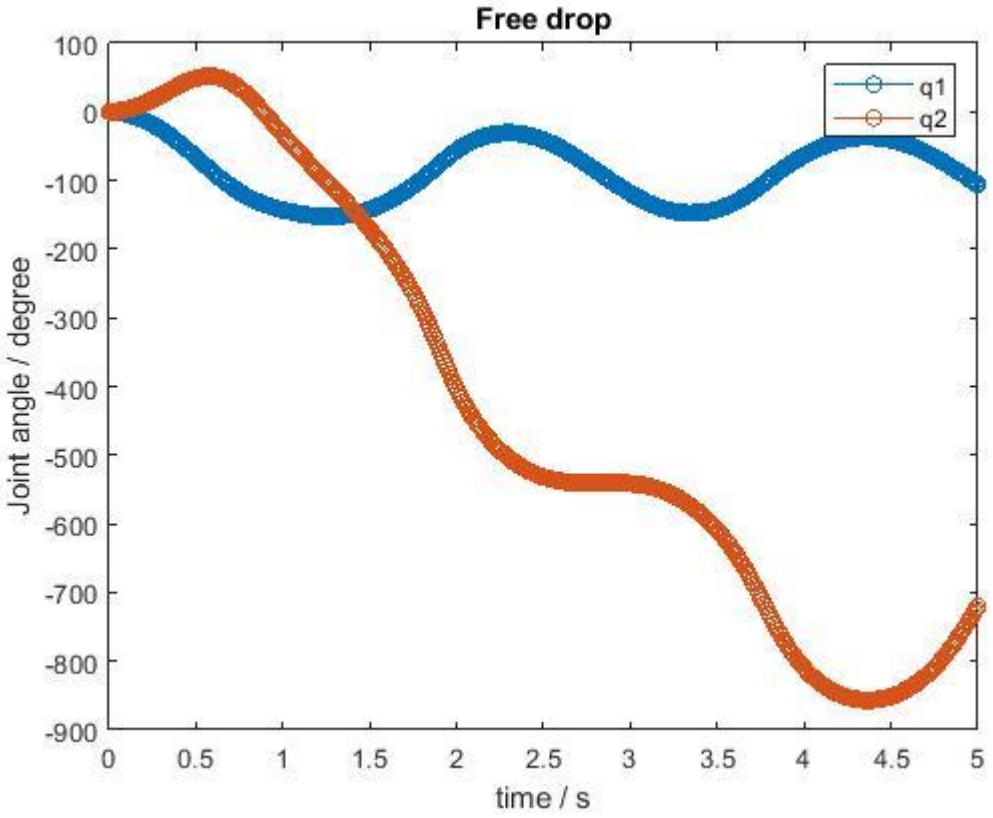

Figure 3 Free swing

When the robot can swing freely, it drops downward by the effect of gravity. Two things to take note of in this graph. First the oscillation on joint 1 is getting slightly smaller as time goes on. This is because the joints are set with a friction coefficient of 0.1 and not 0.0. Second, the angles of the joint 1 is approaching to the resting position, which is $q_1 = -90^{\circ}$. To reach that angle quicker, one could increase the friction to increase the damping of the motion. For q_2, the swing is more chaotic because it the swing is only damped by the friction, and the direction of the gravity keeps on changes as joint 1 is changing.

Task 1.3 Gravity compensated swing condition

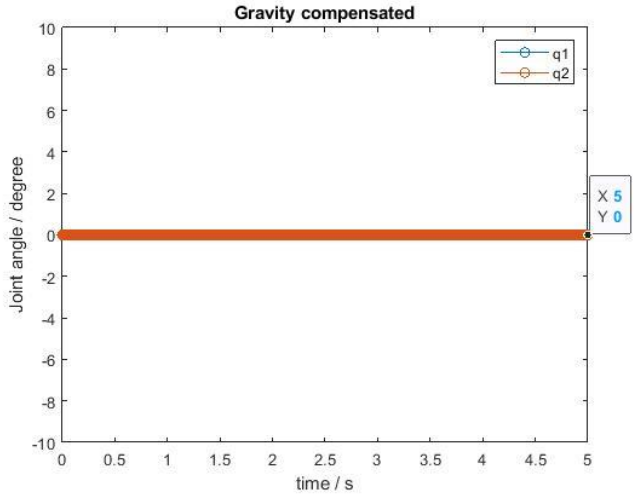

Figure 4 Gravity compensated swing condition

The method I attempted is by using the equation of motion to calculate the proper torque to put onto the system. With the equation of motion (equation (16)):

From the task 1.1 calculation we already know the coefficient of A,B,C and G, from the given information we know the $q_1, q_2, \dot{q}_1, \dot{q}_2$ are set with initial value at 0. With all these, the only unknowns are $\ddot{q}_1$ and $\ddot{q}_2$. Assuming the gravity compensator can do a great job, we can set $q_1, q_2, \dot{q}_1, \dot{q}_2$ as 0 for equation of motion. With that we can calculate the torque on the two joints with equation:

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = A\begin{bmatrix}0\\0\end{bmatrix} + B(0,0)[0] + C(0,0)\begin{bmatrix}0\\0\end{bmatrix} + G(0,0)$$

$$\begin{bmatrix}\tau_1\\ \tau_2\end{bmatrix} = \begin{bmatrix}gm_2(r_{C2} + L_1) + gm_1 r_{C1}\\ gm_2 r_{C2}\end{bmatrix} \quad (17)$$

By applying this force with the correct value of masses and length, we get the gravity compensated graph. From the figure 4, we see a very reasonable balanced robotic arm that does not have any movement for 5 seconds.

When the torque calculated is offset by an under-estimated mass m_2 = 0.995 instead of 1, we can see from figure 5 that the gravity compensation is flawed. The under-estimated mass meant the torque provided by it will be too low to compensate the gravity, thus the arm is slowly swinging downwards.

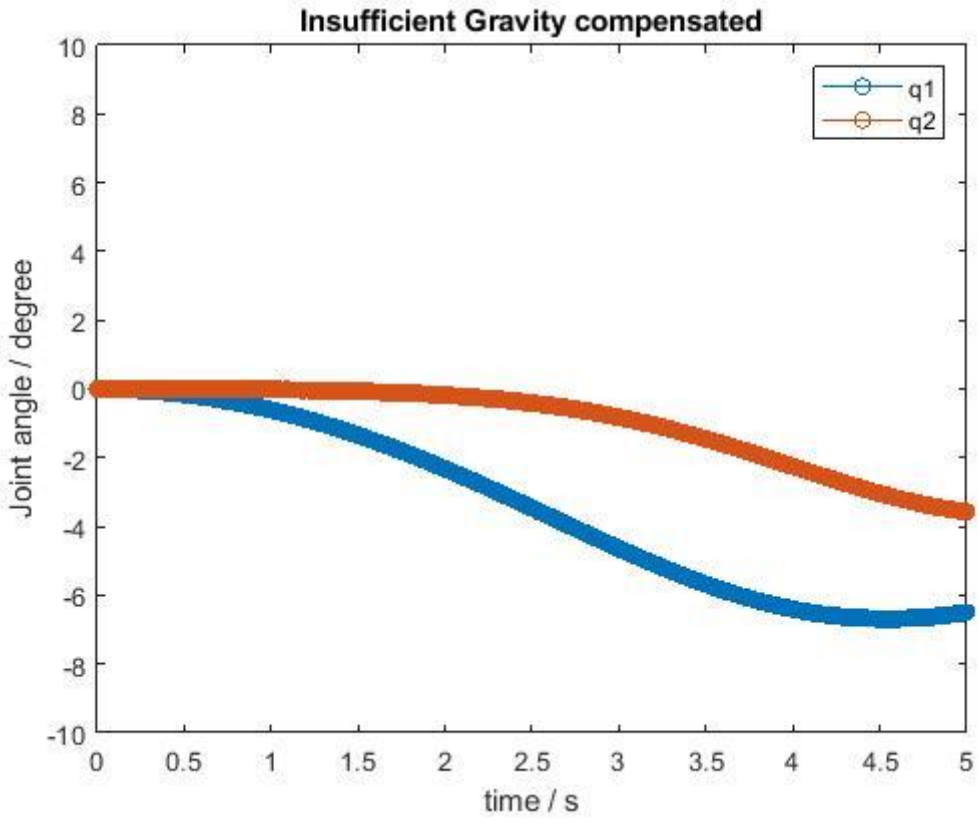

Figure 5 Incorrect gravity compensated swing condition

Task 2. PID Controller on robot in joint space

Task 2.1 PID controller robot angular position and velocity plots

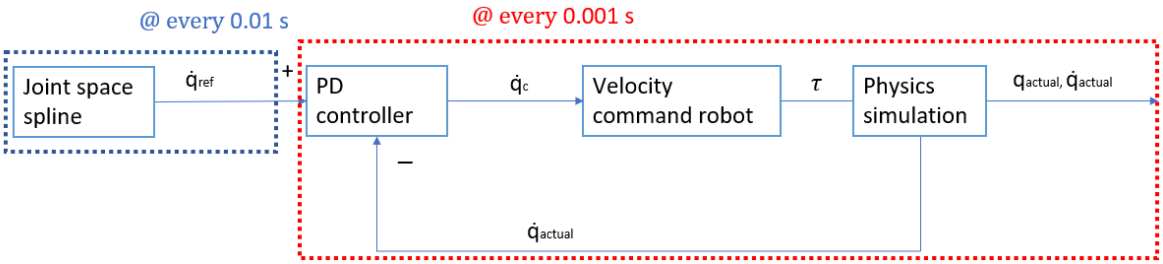

Figure 6 Block diagram of the PD controller system in task 2

The PD controller is picked because it is reasonably good enough without the integral error term. The structure of it is depicted by figure 6. The reference signal is slower than the PD control loop because sampling is slower. The control signal output by the controller is calculated by the equations:

$$\dot{q}_{1c} = k_{p1} * \left(\dot{q}_{1ref} - \dot{q}_{1actual}\right) + k_{d1} * \left(\frac{d\dot{q}_{1ref}}{dt} - \frac{d\dot{q}_{1actual}}{dt}\right) \quad (18)$$

$$\dot{q}_{2c} = k_{p2} * \left(\dot{q}_{2ref} - \dot{q}_{2actual}\right) + k_{d2} * \left(\frac{d\dot{q}_{2ref}}{dt} - \frac{d\dot{q}_{2actual}}{dt}\right) \quad (19)$$

The differentiated terms are defined as:

$$\frac{d\dot{q}_{1ref}(i)}{dt} = \frac{\dot{q}_{1ref}(i) - \dot{q}_{1ref}(i-1)}{dt} \quad (20)$$

$$\frac{d\dot{q}_{1actual}(i)}{dt} = \frac{\dot{q}_{1actual}(i) - \dot{q}_{1actual}(i-1)}{dt_{PID}} \quad (21)$$

$\dot{q}_{1ref}(i)$ is the i[th] term of the respective angular velocity in time.

dt and dt_{PID} are constants set by the assignment that dictates the refresh rate of the reference and controller.

In the special case of $i = 1$, the differential is set as 0.

Then these control signals are passed directly as the torque as we are doing a model-free control, so the conversion between the two is unknown. The torque will be regulated by the output from PD controller ($\dot{q}_c$) and the gravity compensation from equation (17).

$$\tau_1 = \dot{q}_{1c} + gm_2(r_{C2} + L_1) + gm_1 r_{C1} \quad (22)$$

$$\tau_2 = \dot{q}_{2c} + gm_2 r_{C2} \quad (23)$$

With that, the velocity command robot calculates the trajectory in next time step with the MATLAB function ode45. The equation inside the ode45 is from Task 1 equation of motion (equation (16))

For robustness of the controller, it depends on how well tuned the gains of the PD controller are. The tuning process is tuned by trial and error with the assignment recommended step response. The target of the step response is getting to 1°/s in 0.01 second.

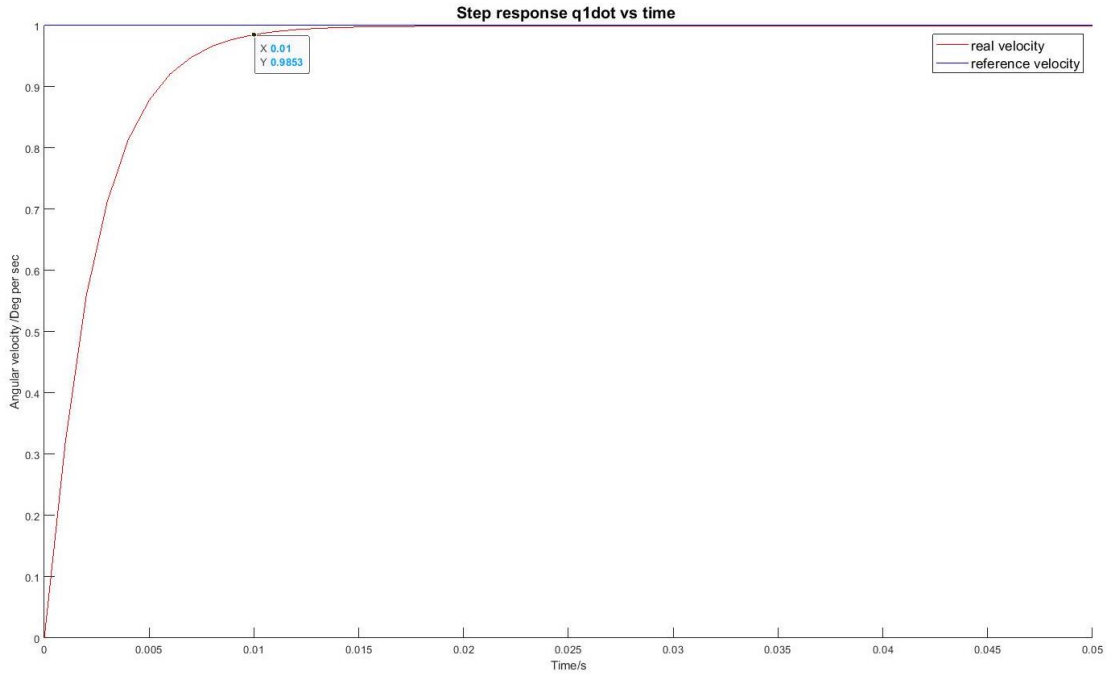

Figure 7 Step response for q1 velocity command

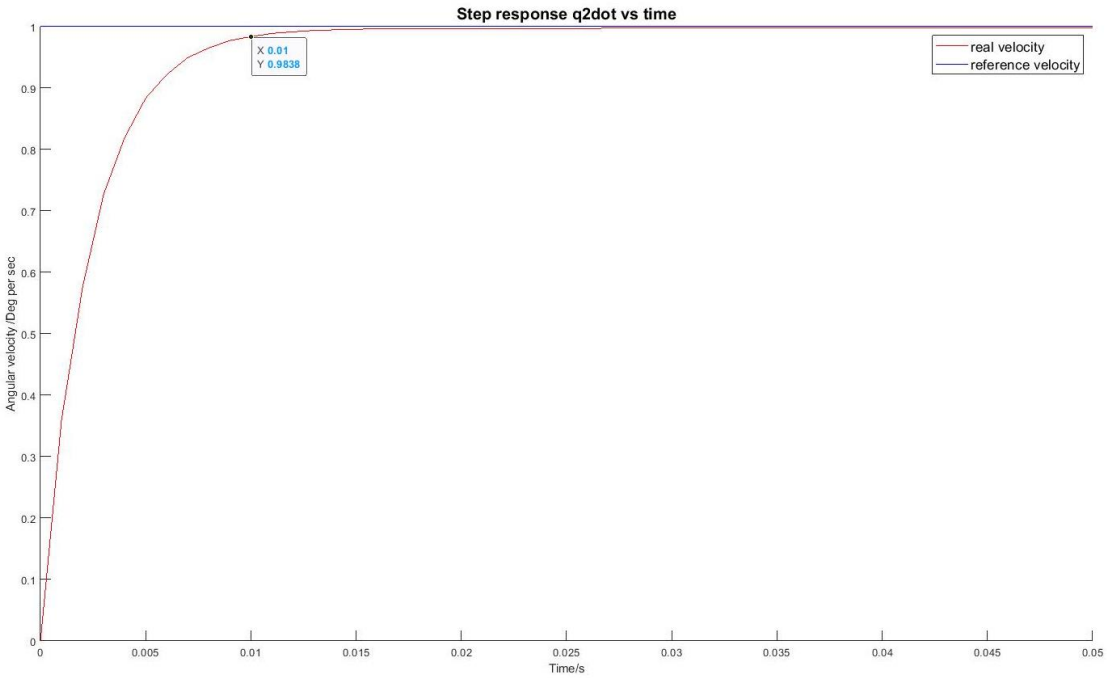

Figure 8 Step response for q2 velocity command

Aiming at arriving within 2% of the target velocity, which is between 0.98 – 1.02, this set of gain is can reach the requirement.

The final values for the tuned gains are:

$$k_{p1} = 1250$$
$$k_{p2} = 380$$
$$k_{d1} = 0.05$$
$$k_{d2} = 0.02 \quad (24)$$

After tuning the PD controller according to the rough guide, the robot can follow the command velocity to reach the commanded angles. We can see that from the overlapping between the reference signal and the actual angles. Even though the robot do not know the actual angular position to track, by keeping the joint velocity accurate, the trajectory in angular position is also followed.

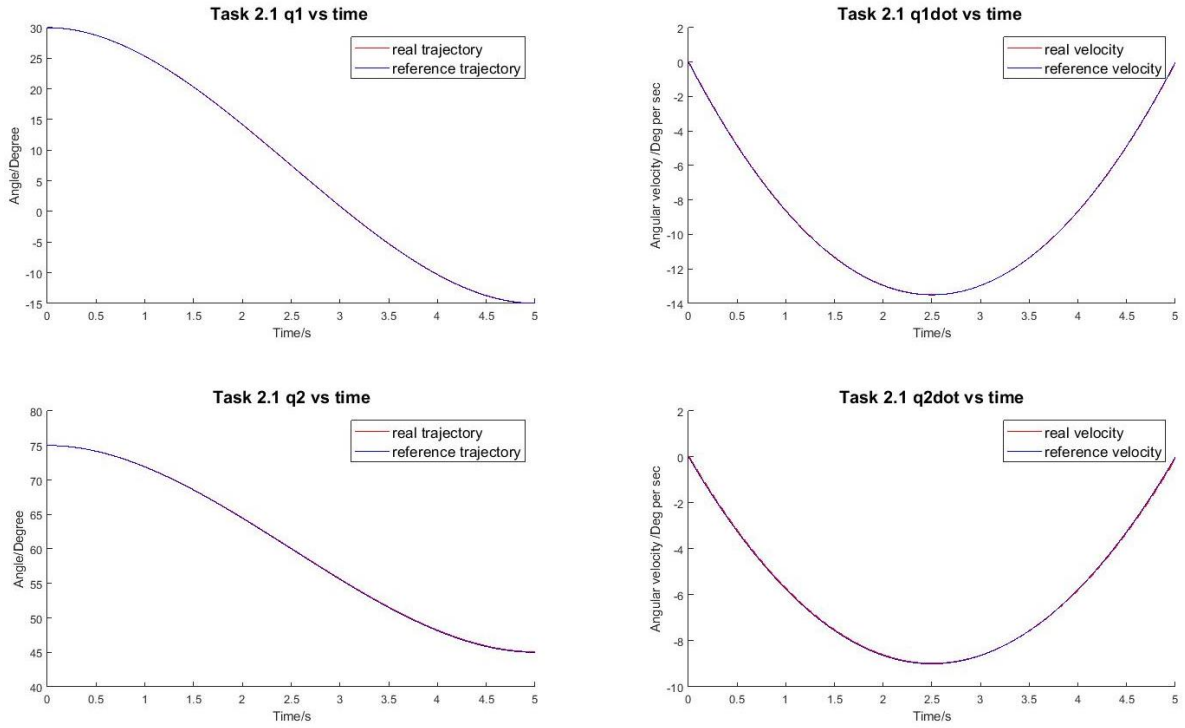

Figure 9 Comparison between real and reference angles and angular velocities

Task 2.2 PID controller robot torques plots

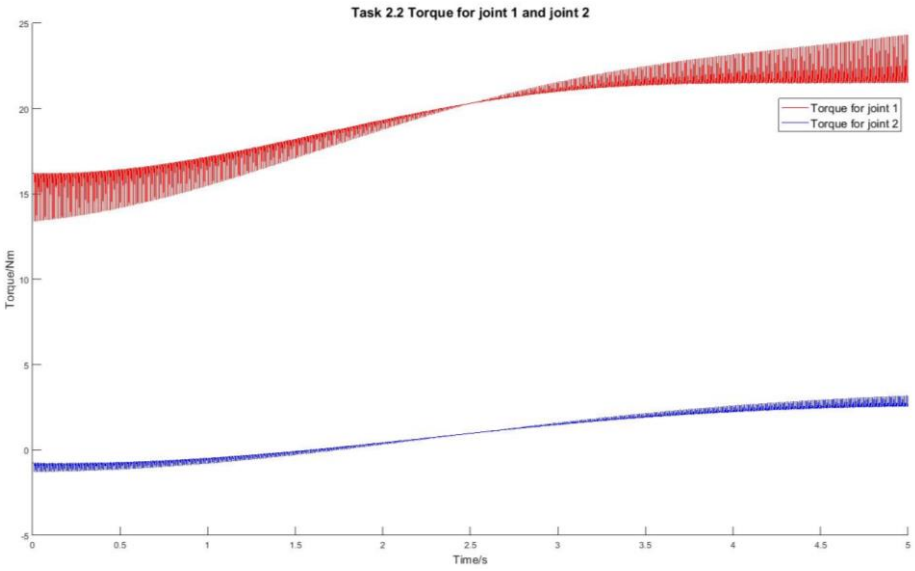

Figure 10 Torque on joint 1 and 2

This is the torque for joint 1 and joint 2 during the 5 second trajectory. As the controller is not perfect, there is considerable fluctuation in the torque. Yet the torque has never destabilize the plant, therefore the control is still acceptable. The presence of fluctuation also make sense because we are not doing a torque control. We never know what the correct torque is to apply to the robot to achieve the target velocity, so we would be floating around the proper value. Also, this torque includes the gravity compensation, which is the reason why the fluctuation is not about 0 torque.

Task 2.3 PID controller robot task space time lapse plots

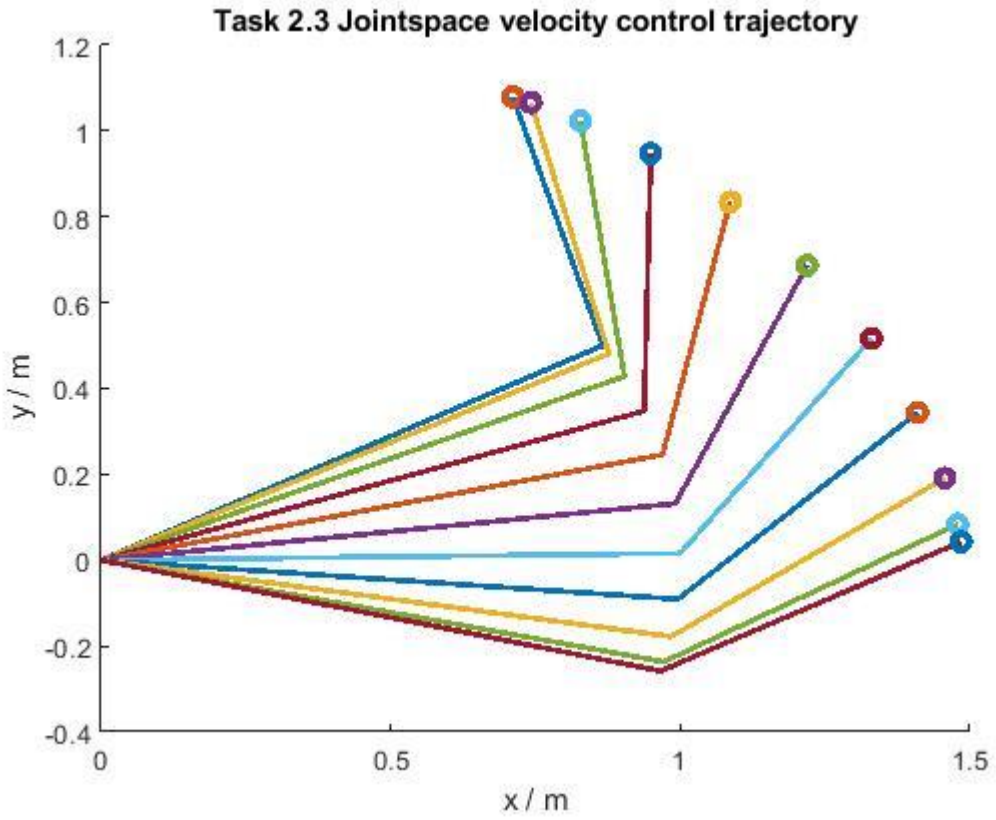

Figure 11 XY plot of the robot under joint space velocity control

The resultant trajectory is shown here. Since we are doing a joint space spline, the trajectory is not going to be a straight line. More discussion of that will be provided in Task 3.2 when we have the straight lien trajectory. The time lapse also show us the cubic spline effect on the velocity. As it is a cubic spline with zero start and final velocity, the trajectory is fastest around the middle of the time span, which is indicated by the further apart of the dot that represents the end effector.

Task 3. Joint space control of the velocity commanded robot

Task 3.1 end-effector x vs t, y vs t plot

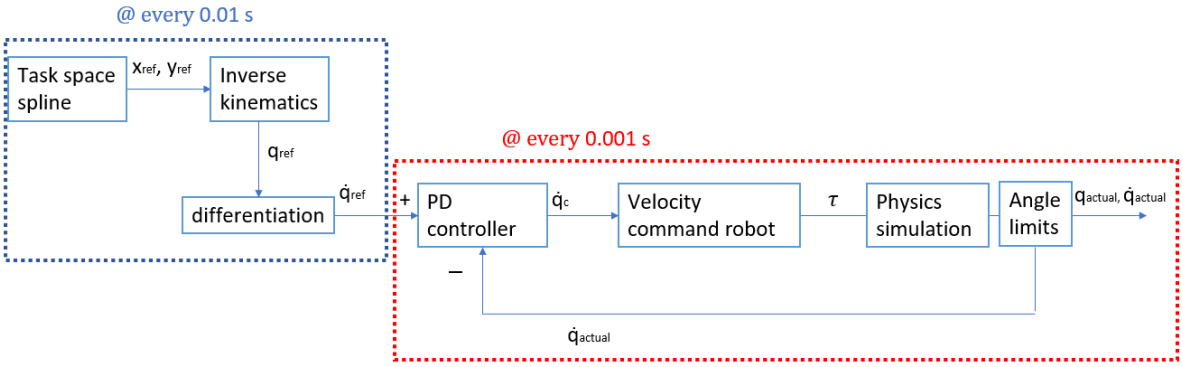

Figure 12 lock diagram of the task space splining system in task 3

There are three differences in task 3.1 compared to task 2.1. First, the joints cannot reach all 360 degrees, they are limited by the set boundary:

$$-60° \leq q_1 \leq 60° \qquad (25)$$
$$-90° \leq q_2 \leq 90°$$

Second, the splining occurs in the task space instead of the joint space. By doing so, the trajectory will end up in a straight line instead of the curved path in task 2. From figure 13 we can see that the robot is able to follow the trajectory considerably well.

Third, to obtain the reference joint velocities, differentiation on the joint angles is required to get the angular velocities that are sent to the PD conrtoller. The joint velocities could be found by using inverse jacobian as well. However, since the assignment did not specify which method is preferred, the differentiation method providing sufficiently good joint velocities without the need of computing the jacboian, which is a more computation heavy than finding the differential.

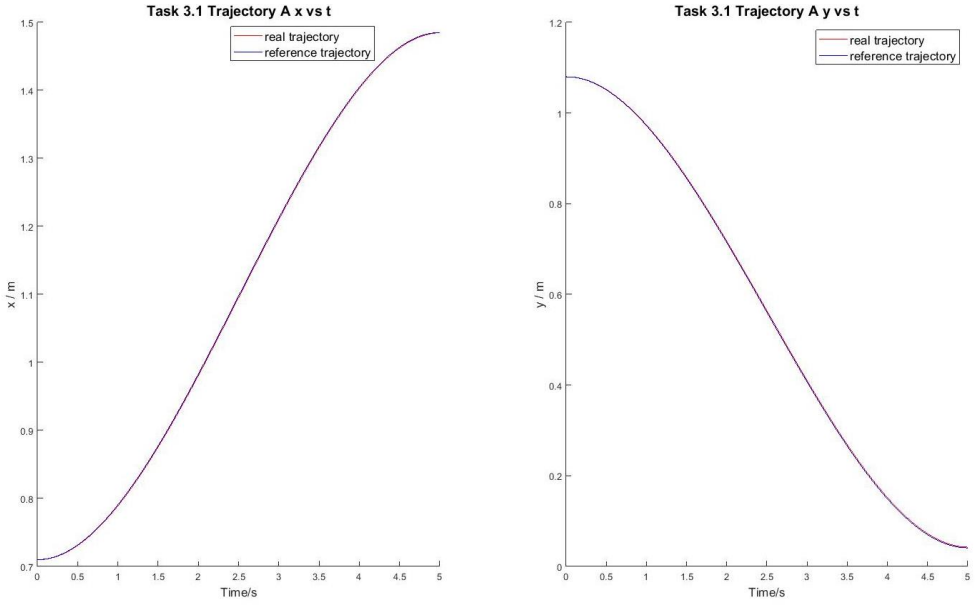

Figure 13 End effector x vs t and y vs t plot of trajectory A

Task 3.2 Trajectory A plot

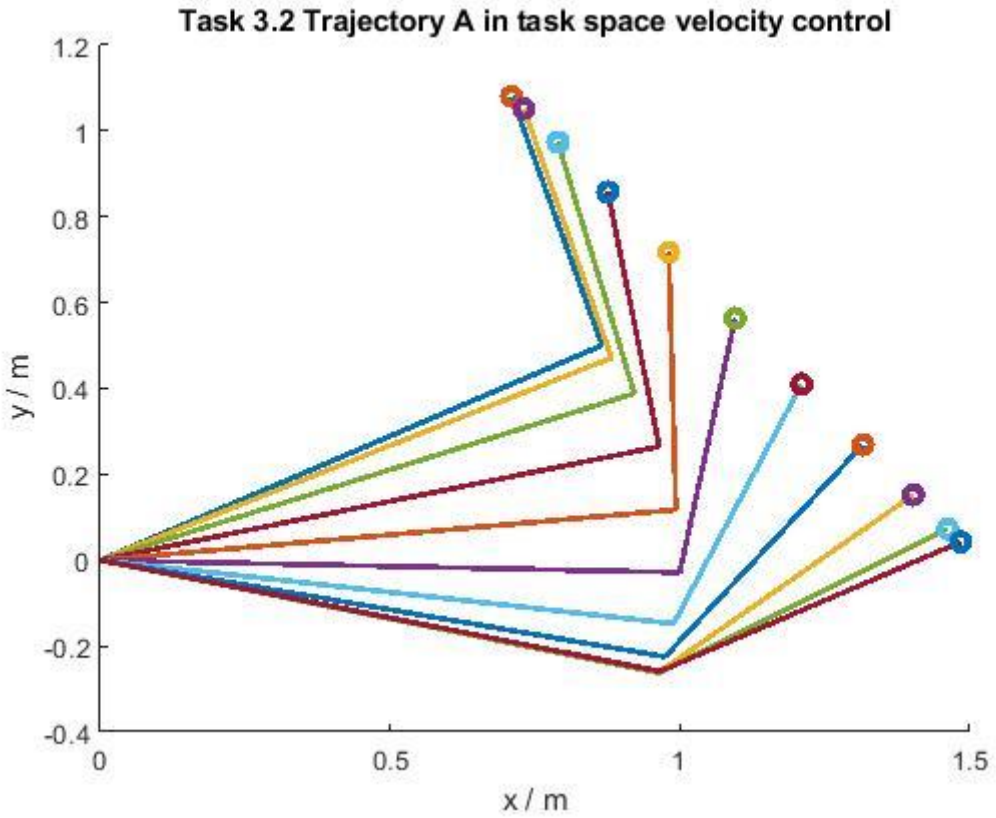

Figure 14 XY plot of task space splining trajectory A

Comparing this plot with the joint space control plot (figure 11), the task space control trajectory is straight. This is because when the joint space control is splining, the splined joint angles do not create the straight-line path in task space. When we spline the points in task space, the velocity of x and y are in a nearly fixed ratio, which meant that the robot is moving in a straight line in task space.

Task 3.3 Trajectory B plot

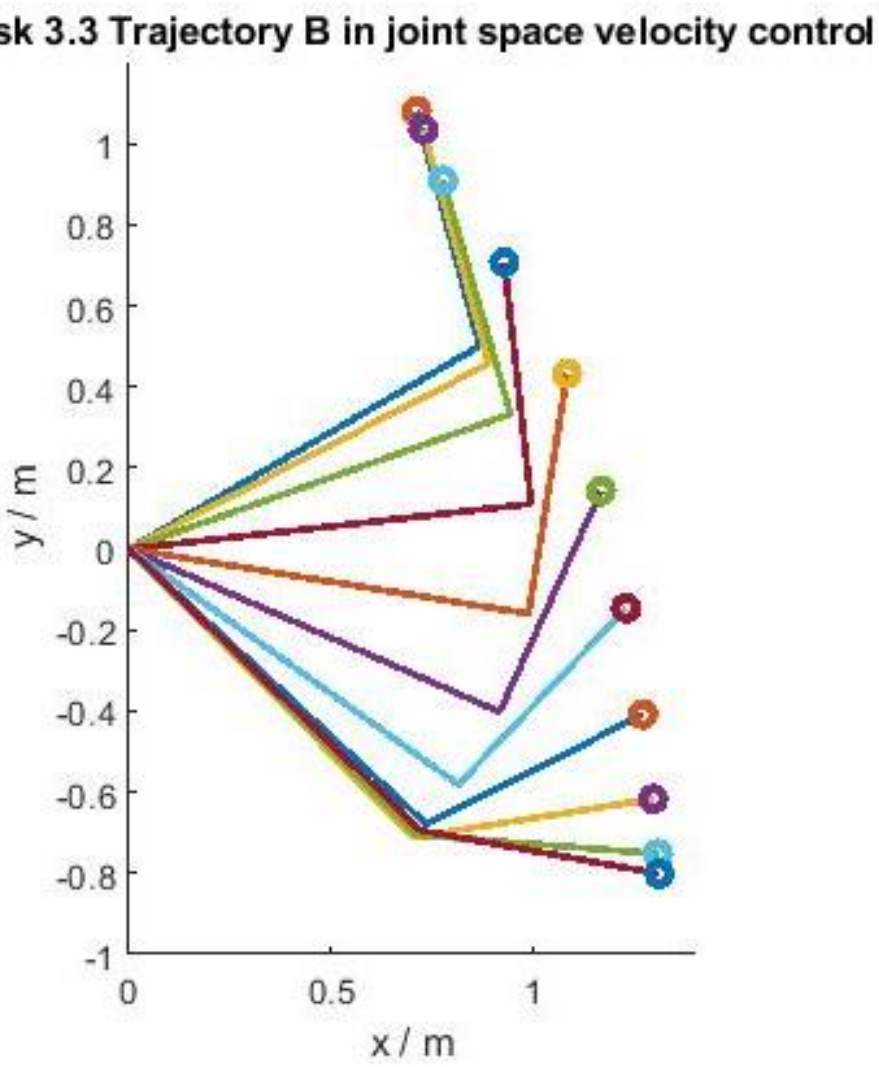

Figure 15 XY plot of task space splining trajectory B

As the joint limit forbids the joint 2 to go pass 90°, the end-effector could not reach the planned straight path. As the place I put the joint limit is after the simulation, it has similar effect of having a physical stopping mechanism, which we allow the motor to drive into the physical stop and stop it there. We can see that in the middle of the trajectory, the joint 2 is at a right angle, which is the 90° limit. To form a direct straight line trajectory, a few options can be considered including relaxing the joint limit, changing the length of the robot to allow the reachable workspace to cover the straight path or put a via point so the trajectory is in two straight segment.

The problem also propagates down in the trajectory. The final point stopped at the trajectory is not the destination planned. This is because the robot was not equipped with the ability to correct positional error, it was only correcting and tracking the velocity alone. Therefore as the robot was pushed away from the trajectory in the middle section, the trajectory would never be put back on track.

Task 4. Task space control of the velocity commanded robot

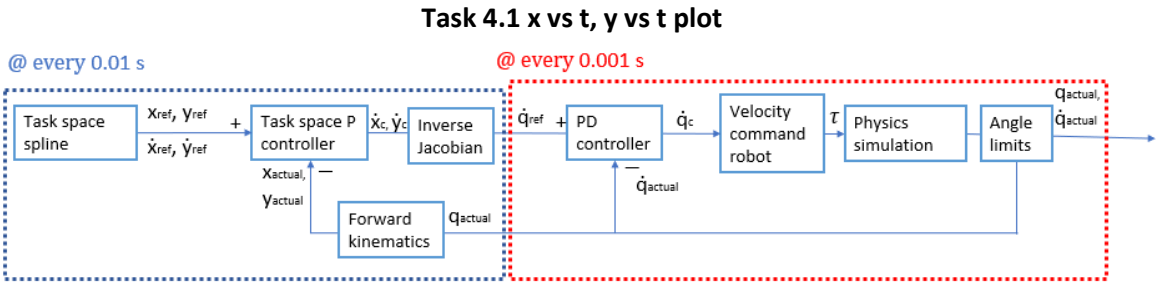

Figure 16 Block diagram for the Task space control system for task 4

In task 4, the change from previous tasks is implement a position control in task space. The position control is done with a proportion controller. With assuming the update loop is quick enough and no noise to affect the trajectory, which are valid assumptions in our simulation, the trajectory will be closely matched with the reference. As there is not much error, a simple P controller will do the job. When the trajectory is somehow end up far away from the trajectory, a suitably tuned P gain will not destabilize the plant, so P controller is a suitable choice for our task. The structure of the controller is depicted by figure 16. The reference in this case is similar to task 3, which is splined trajectory in task space. The difference is that we now have access to error term in task space. The control signal output by the controller is calculated by the equations:

$$\begin{aligned} \dot{x}_c &= \dot{x}_{ref} + k_{px} * (x_{ref} - x_{actual}) \\ \dot{y}_c &= \dot{y}_{ref} + k_{py} * (y_{ref} - y_{actual}) \end{aligned} \tag{26}$$

Where k_{px} and k_{py} are found by trial and error. The values for a robust performance are:

$$\begin{aligned} k_{px} &= 50 \\ k_{py} &= 50 \end{aligned} \tag{27}$$

From a arbitrary step response, we can see the trajectory is put back to the reference without fluctuation with in 0.1 seconds.

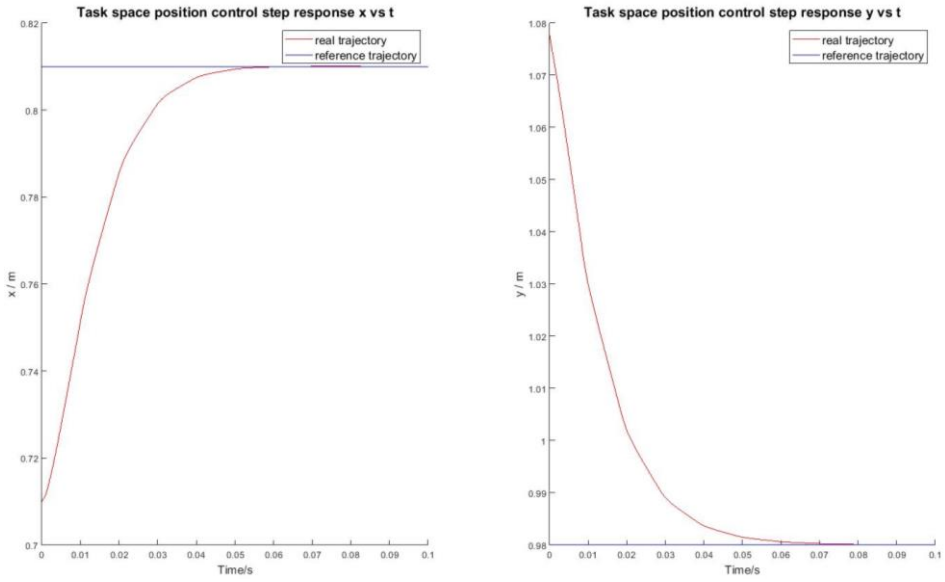

Figure 17 step response in task space position error

To get the actual angular position of the robot at every sampling instance, forward kinematics is performed:

$$\begin{bmatrix} x_{actual} \\ y_{actual} \end{bmatrix} = \begin{bmatrix} L_1 \cos(q_1) + L_2 \cos(q_1 + q_2) \\ L_1 \sin(q_1) + L_2 sin(q_1 + q_2) \end{bmatrix} \quad (28)$$

As we are dealing with task space control, some conversion between task space and joint space must be done. The task space velocity control signal is converted by inverse Jacobian back to joint space velocity control signal. Then the control signal in task space is converted into joint space reference signal by ${J_{v,xy}}^{-1}$, inverse Jacobian of end effector x and y velocity component:

$$\begin{bmatrix} \dot{q}_{1ref} \\ \dot{q}_{2ref} \end{bmatrix} = {J_{v,xy}}^{-1} \begin{bmatrix} \dot{x}_c \\ \dot{y}_c \end{bmatrix} \quad (29)$$

Which the Jacobian matrix $J_{v,xy}$ is defined by joint angles:

$$J_{v,xy} = \begin{bmatrix} -L_2 \sin(q_1 + q_2) - L_1 \sin(q_1) & -L_2 \sin(q_1 + q_2) \\ L_2 \cos(q_1 + q_2) + L_1 \cos(q_1) & L_2 \cos(q_1 + q_2) \end{bmatrix} \quad (30)$$

Then this joint space (equation (29)) reference is passed into the joint space velocity commanded robot that is same as the one in Task 2 and Task 3. The PD gain and such have not been changed.

The result of the trajectory is depicted by the figure:

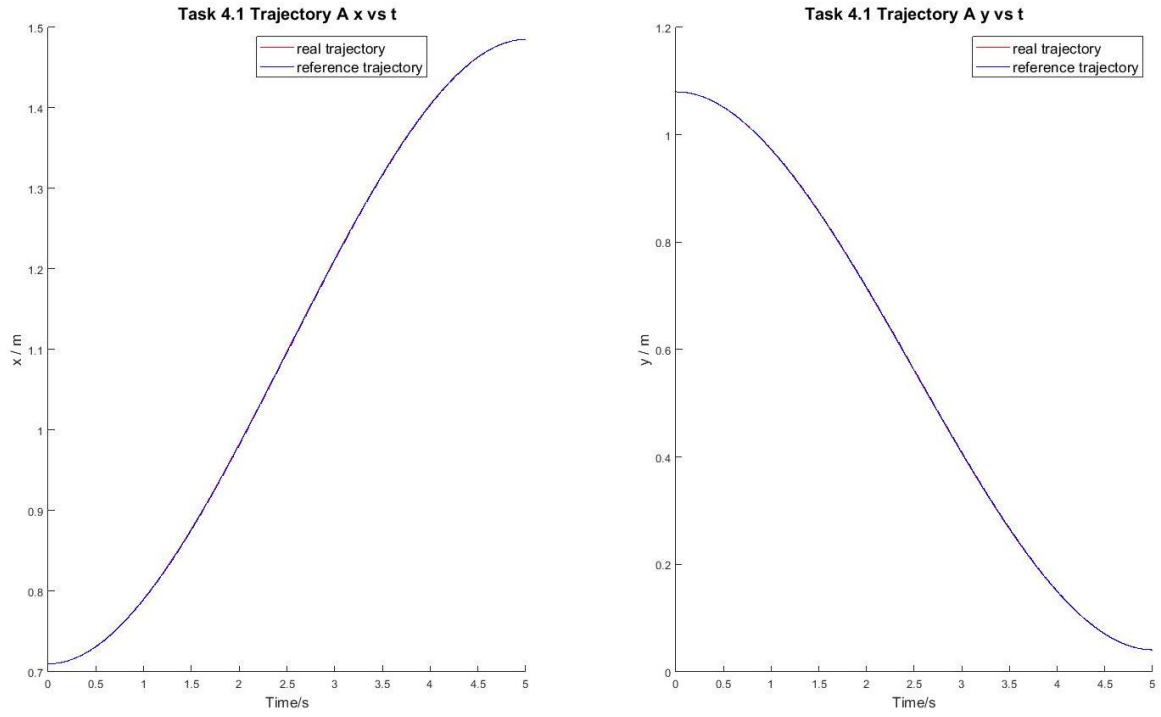

Figure 18 End effector x vs t and y vs t plot from task space control

We can see that the trajectory is followed by the robot precisely.

Task 4.2 Trajectory A XY plot

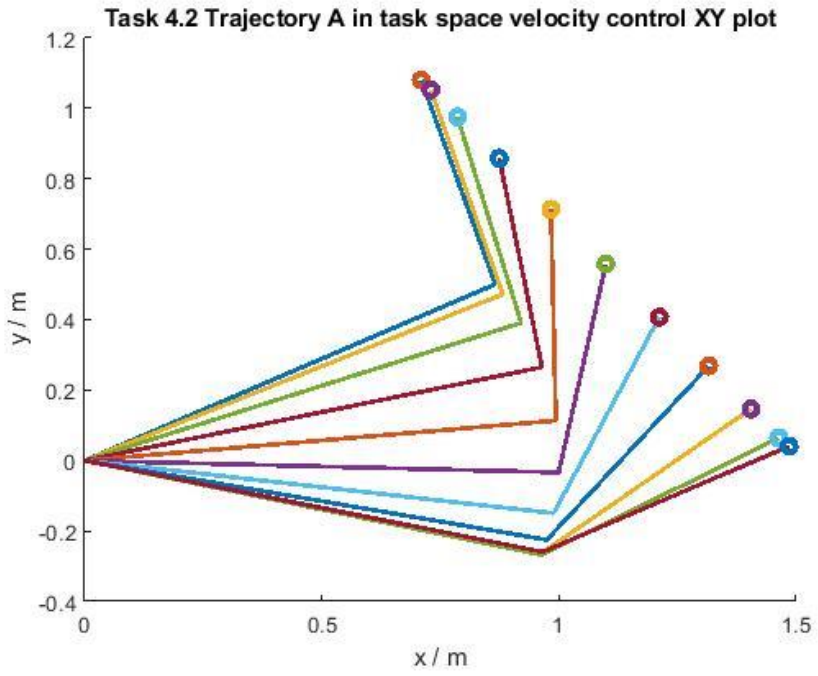

Figure 19 XY plot of task space control

This following discussion is applicable to both task 4.1 and 4.2. When comparing the resultant trajectory from task space velocity control and joint space velocity control, they are virtually the same. Even though the control scheme for task 3 and 4 are different, but they are both following the same splined reference in task space. So with a sufficiently good control on both sides, the resulting trajectory would not be any visible difference.

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

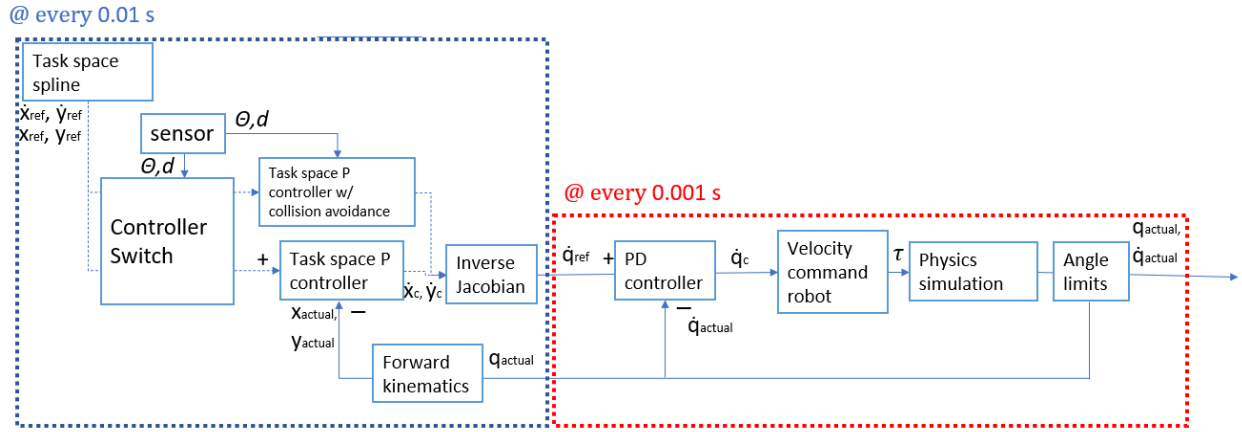

Figure 20 Block diagram of the collision avoidance system for task 5

Strategy for the collision avoidance:

The obstacle avoidance strategy is a reactive one, that the end effector would only be avoiding the obstacle if there is a need for it. Before getting into the mathematical part, the idea is presented in a logical manner first. When it is approaching an obstacle, the end effector would be pushed away from the obstacle. The direction of motion will still partially the original path with a component of velocity with respect to where the obstacle is. During the collision avoidance stage, the original control signal will be switched to the collision avoidance branch. After passing the obstacle, the original controller will take back the control.

As the end effector is, the avoidance controller only starts reacting when the sensors distance value *d* is smaller than the threshold $interact\_limit$ depicted in equation (30)

$$d < interact\_limit, interact\_limit = 0.15 \quad (31)$$

Reason behind this is to prevent reacting too early and unnecessarily.

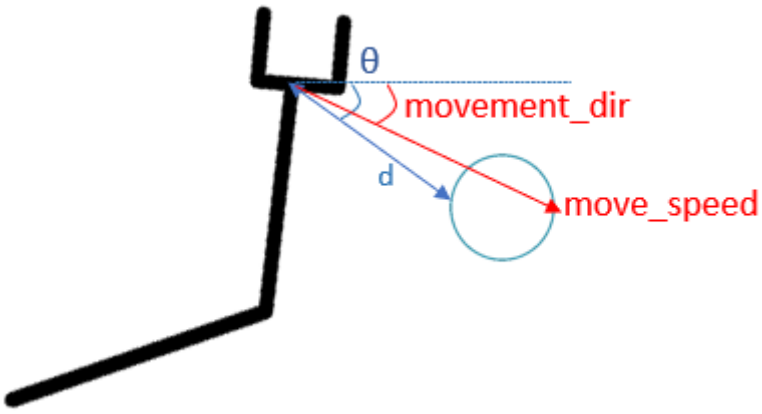

Figure 21 Diagram for the angles and notation used in collision avoidance

The movement direction and speed of the end effector is calculated using workspace control reference signal. Based on the knowledge from sensor and our velocity reference, which includes distance (*d*), angle between the obstacle and end effector (ϑ), movement direction ($movement\_dir$) and movement speed ($move\_speed$), the end effector will try to go above or below the obstacle. The equation (32) and (33) shows how $move\_speed$ and $movement\_dir$ are calculated:

$$move\_speed = \sqrt{\dot{x}_{ref}^2 + \dot{y}_{ref}^2} \tag{32}$$

$$movement\_dir = \tan^{-1}(\frac{\dot{y}_{ref}}{\dot{x}_{ref}}) \tag{33}$$

From the trial done with the control scheme, the result slightly favours the obstacle avoidance strategy with moving above it. So when $movement\_dir + 30^0 < \theta$, the end effector moves below, if it is not then the end effector moves above the obstacle. With that in mind, the avoidance has two possibilities: either going above it or below it.

The avoidance velocity signal based on this equation, where the first sign in plus–minus sign refers to going above trajectory, while the second sign refers to going below trajectory.

$$\begin{aligned} \dot{x}_c &= \dot{x}_{ref} + move\_speed * \frac{force\_field\_constant * force\_field\_limit}{d + force\_field\_limit} * \pm \sin(\theta) \\ \dot{y}_c &= \dot{y}_{ref} + move\_speed * \frac{force\_field\_constant * force\_field\_limit}{d + force\_field\_limit} * \mp \cos(\theta) \end{aligned} \tag{34}$$

The avoidance speed is scaled with the $move\_speed$ to ensure the avoidance is able to override original movement. The d in denominator ensures a higher velocity signal when the end effector is closer to the obstacle. $force\_field\_limit$ ensures the velocity is bounded as *d* approaches zero.

The $force\_field\_constant$ *and* $force\_field\_limit$ are tuned by experiment are:

$$\begin{aligned} force\_field\_limit &= 0.01 \\ force\_field\_constant &= 8 \end{aligned} \tag{35}$$

After avoiding the obstacle, determined by $\|movement\_dir - \theta\| > 90^0$, is important to put the trajectory back in track, else the control signal will observe a large error in, causing the robot to have large fluctuation. The path taken before will determine whether we are merging from above or below. The equation for the merge back section is identical to equation (34)

After dodging the obstacle, the original task space control scheme with velocity reference will guide the trajectory to the original target. Some examples of it can be seen here:

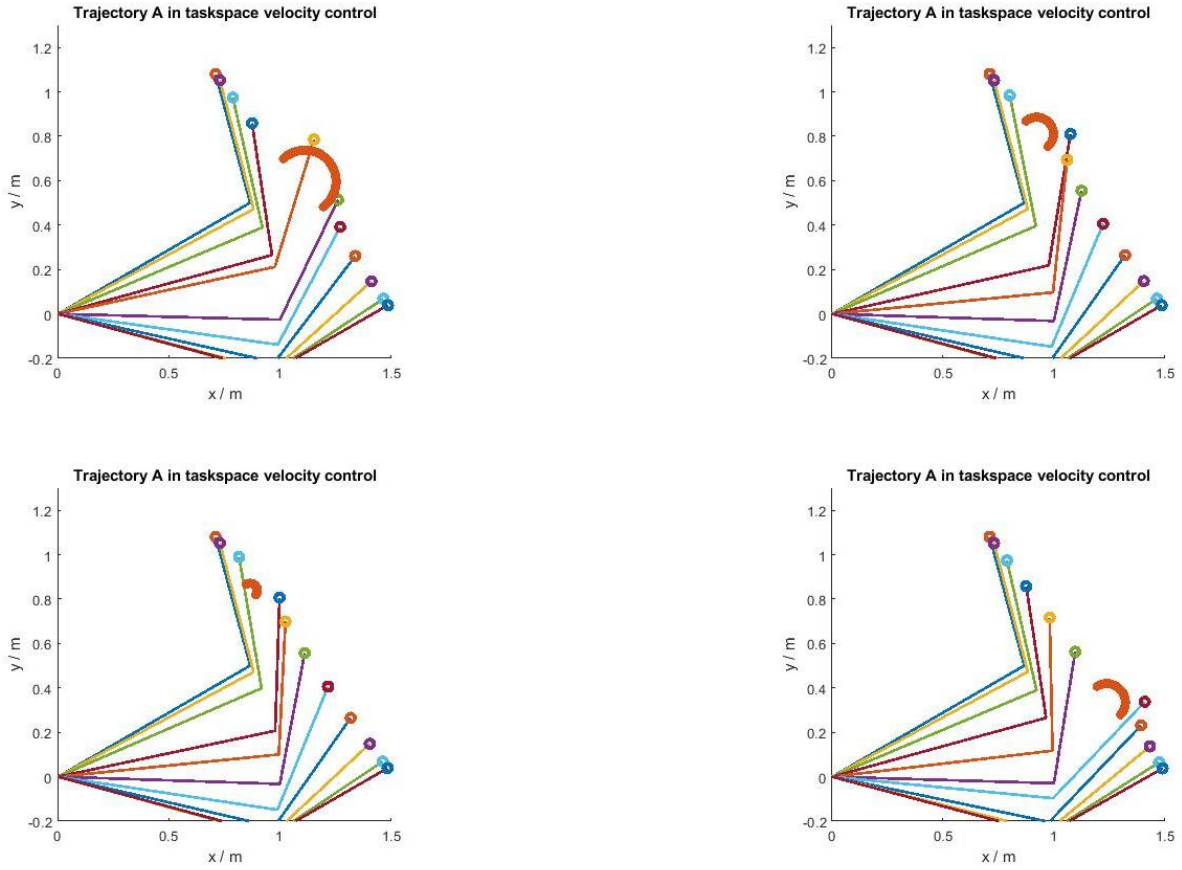

Figure 22 XY plots of collision avoidance task