Contents

1. Introduction

In this project, I was supposed to model and control the UR5 robot, a product of the Universal Robotics Incorporation. This robot has 6 Degrees of Freedom (DoF) and is very popular for research purposes. The UR5 has 6 different links and 6 revolute joints that are modelled in Matlab software. Figure 1 shows the robot and its links and joints.

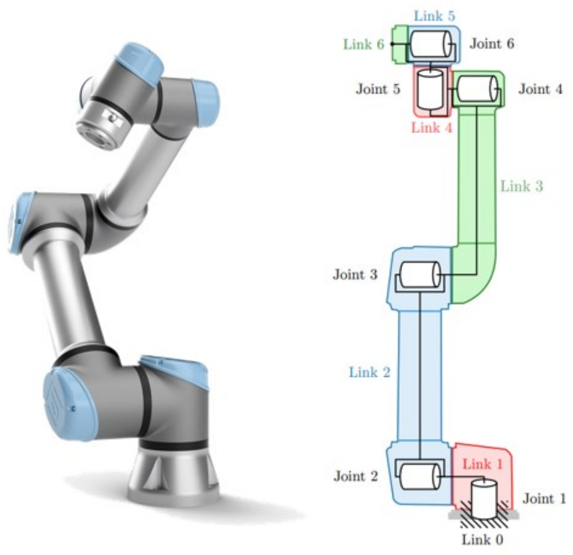


Figure 1: UR5 robot and its links and joints

In the following sections, I will discuss the steps I have taken throughout this project, including kinematic modelling, dynamic modelling, trajectory planning and controlling of the robot. The results of different controlling schemes are depicted and discussed.

2. Kinematic Analysis

In this section, I will first discuss the derivation of the DH table and transformation matrices based on that table. Second, I will talk about the process of inverse kinematics by using geometry-based methods.

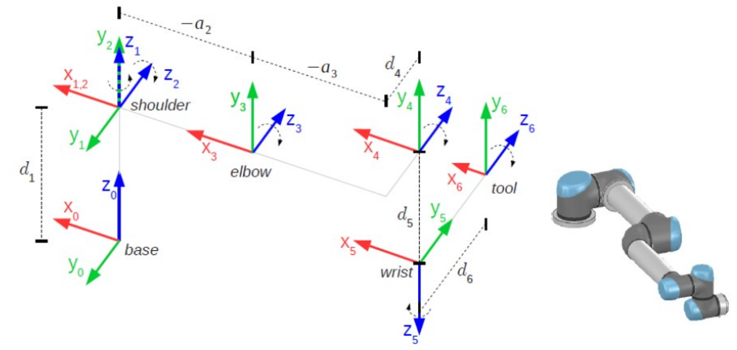
2.1 Forward Kinematics

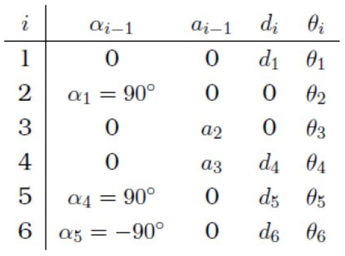
The first step of modelling a robot is finding out the physical properties of the robot. Fortunately, the manufacturer has reported these values in the following table:

Table 1: physical properties of the UR5

UR5							
Kinematics	theta [rad]	a [m]	d [m]	alpha [rad]	**Dynamics**	Mass [kg]	Center of Mass [m]
Joint 1	0	0	0.089159	π/2	Link 1	3.7	[0, -0.02561, 0.00193]
Joint 2	0	-0.425	0	0	Link 2	8.393	[0.2125, 0, 0.11336]
Joint 3	0	-0.39225	0	0	Link 3	2.33	[0.15, 0.0, 0.0265]
Joint 4	0	0	0.10915	π/2	Link 4	1.219	[0, -0.0018, 0.01634]
Joint 5	0	0	0.09465	-π/2	Link 5	1.219	[0, 0.0018,0.01634]
Joint 6	0	0	0.0823	0	Link 6	0.1879	[0, 0, -0.001159]

The frame attachment is based on the modified Denavit-Hartenberg (DH) method and is shown in Figure 3. You can also see the DH table in this figure. The notations are the same as the course slides.





i	α_{i-1}	a_{i-1}	d_i	θ_i
1	0	0	d_1	θ_1
2	$\alpha_1 = 90^\circ$	0	0	θ_2
3	0	a_2	0	θ_3
4	0	a_3	d_4	θ_4
5	$\alpha_4 = 90^\circ$	0	d_5	θ_5
6	$\alpha_5 = -90^\circ$	0	d_6	θ_6

Figure 2: Frame assignments and DH-table of UR5

To derive the forward kinematic transformation matrix, I used the chain-law. The resultant matrix is reported in Figure 4. Each transformation matrix can be calculated as follows:

$$
{}^{i-1}_{\ i}T = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & 0 & a_{i-1} \\ \sin\theta_i\cos(\alpha_{i-1}) & \cos\theta_i\cos(\alpha_{i-1}) & -\sin(\alpha_{i-1}) & -\sin(\alpha_{i-1})d_i \\ \sin\theta_i\sin(\alpha_{i-1}) & \cos\theta_i\sin(\alpha_{i-1}) & \cos(\alpha_{i-1}) & \cos(\alpha_{i-1})d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

And the final transformation matrix from the base to the end-effector can be derived by cross production of all these matrices.

$$
{}^{0}_{N}T = {}^{0}_{1}T\,{}^{1}_{2}T\,{}^{2}_{3}T\,{}^{3}_{4}T\ldots\ldots{}^{N-1}_{\ \ N}T
$$

This process has been done in Matlab and the result is as follows:

$$
{}^{1}_{6}T = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}
$$

$$
\begin{aligned}
n_x &= c_6(s_1s_5 + ((c_1c_{234} - s_1s_{234})c_5)/2.0 + ((c_1c_{234} + s_1s_{234})c_5)/2.0) - (s_6((s_1c_{234} + c_1s_{234}) - (s_1c_{234} - c_1s_{234})))/2.0 \\
n_y &= c_6(((s_1c_{234} + c_1s_{234})c_5)/2.0 - c_1s_5 + ((s_1c_{234} - c_1s_{234})c_5)/2.0) + s_6((c_1c_{234} - s_1s_{234})/2.0 - (c_1c_{234} + s_1s_{234})/2.0) \\
n_z &= (s_{234}c_6 + c_{234}s_6)/2.0 + s_{234}c_5c_6 - (s_{234}c_6 - c_{234}s_6)/2.0
\end{aligned}
$$

$$
\begin{aligned}
o_x &= -(c_6((s_1c_{234} + c_1s_{234}) - (s_1c_{234} - c_1s_{234})))/2.0 - s_6(s_1s_5 + ((c_1c_{234} - s_1s_{234})c_5)/2.0 + ((c_1c_{234} + s_1s_{234})c_5)/2.0) \\
o_y &= c_6((c_1c_{234} - s_1s_{234})/2.0 - (c_1c_{234} + s_1s_{234})/2.0) - s_6(((s_1c_{234} + c_1s_{234})c_5)/2.0 - c_1s_5 + ((s_1c_{234} - c_1s_{234})c_5)/2.0) \\
o_z &= (c_{234}c_6 + s_{234}s_6)/2.0 + (c_{234}c_6 - s_{234}s_6)/2.0 - s_{234}c_5s_6
\end{aligned}
$$

$$
\begin{aligned}
a_x &= c_5s_1 - ((c_1c_{234} - s_1s_{234})s_5)/2.0 - ((c_1c_{234} + s_1s_{234})s_5)/2.0 \\
a_y &= -c_1c_5 - ((s_1c_{234} + c_1s_{234})s_5)/2.0 + ((c_1s_{234} - s_1c_{234})s_5)/2.0 \\
a_z &= (c_{234}c_5 - s_{234}s_5)/2.0 - (c_{234}c_5 + s_{234}s_5)/2.0
\end{aligned}
$$

$$
\begin{aligned}
p_x &= -(d_5(s_1c_{234} - c_1s_{234}))/2.0 + (d_5(s_1c_{234} + c_1s_{234}))/2.0 + d_4s_1 - (d_6(c_1c_{234} - s_1s_{234})s_5)/2.0 - (d_6(c_1c_{234} + s_1s_{234})s_5)/2.0 + a_2c_1c_2 + d_6c_5s_1 + a_3c_1c_2c_3 - a_3c_1s_2s_3) \\
p_y &= -(d_5(c_1c_{234} - s_1s_{234}))/2.0 + (d_5(c_1c_{234} + s_1s_{234}))/2.0 - d_4c_1 - (d_6(s_1c_{234} + c_1s_{234})s_5)/2.0 - (d_6(s_1c_{234} - c_1s_{234})s_5)/2.0 - d_6c_1c_5 + a_2c_2s_1 + a_3c_2c_3s_1 - a_3s_1s_2s_3) \\
p_z &= d_1 + (d_6(c_{234}c_5 - s_{234}s_5))/2.0 + a_3(s_2c_3 + c_2s_3) + a_2s_2 - (d_6(c_{234}c_5 + s_{234}s_5))/2.0 - d_5c_{234}
\end{aligned}
$$

2.2 Inverse Kinematics

The UR5 robot is a 6R robot and can be divided into an arm and a wrist. To find out the rotation angle of each joint based on a given position, I have used the geometric method.

we first determine the location of frame 5 (the wrist frame) in relation to the base frame:

$$\theta_1 = \operatorname{atan2}\left({}^0P_{5y}, {}^0P_{5x}\right) \pm \operatorname{acos}\left(\frac{d_4}{\sqrt{{}^0P_{5x}{}^2 + {}^0P_{5y}{}^2}}\right) + \frac{\pi}{2}$$

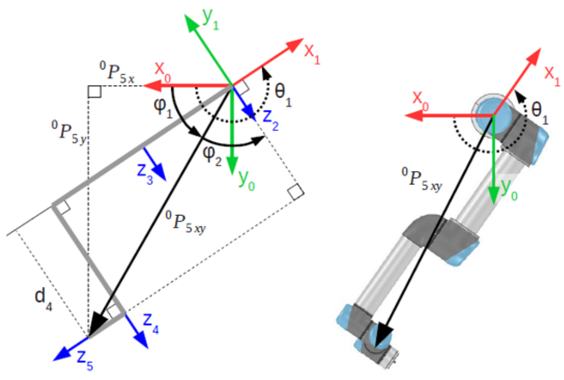


Figure 3: Finding θ_1

The two solutions correspond to the shoulder being “left” or “right”. Second, we can find the angle θ_5 as shown here:

$$\theta_5 = \pm\operatorname{acos}\left(\frac{{}^0P_{6x}\sin\theta_1 - {}^0P_{6y}\cos\theta_1 - d_4}{d_6}\right)$$

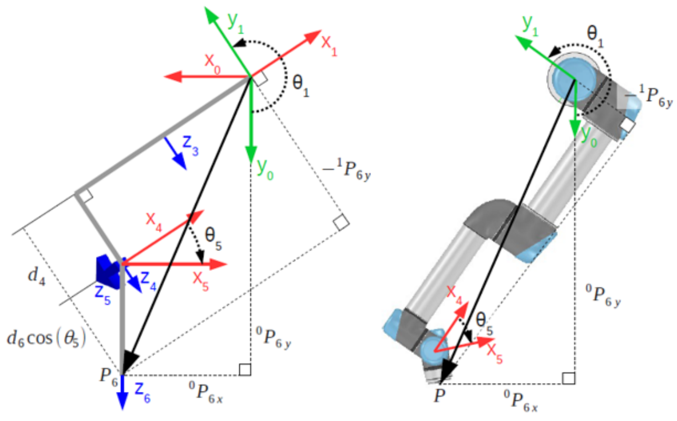
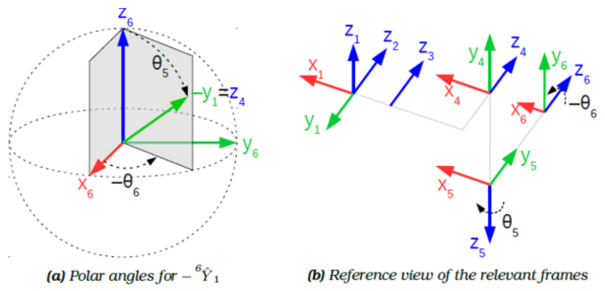


Figure 4: Finding θ_5

These correspond to the wrist being “up” or “down”. Third, I will find θ_6 based on the previous angles that were found:

$$\theta_6 = \operatorname{atan2}\left(\frac{-{}^6\hat{X}_{0y}\cdot\sin\theta_1 + {}^6\hat{Y}_{0y}\cdot\cos\theta_1}{\sin\theta_5}, \frac{{}^6\hat{X}_{0x}\cdot\sin\theta_1 - {}^6\hat{Y}_{0x}\cdot\cos\theta_1}{\sin\theta_5}\right)$$



z6
θ5
−y1=z4
y6
x6
−θ6
(a) Polar angles for −⁶Ŷ1
z1
z2
z3
x1
y1
y4
z4
x4
y6
z6
x6
−θ6
y5
x5
θ5
z5
(b) Reference view of the relevant frames

Figure 5: Finding θ_6

Now, the remaining parts are just a 3R robot in a planar surface. We cand find θ_3 as shown:

$$\theta_3 = \pm \mathrm{acos}\left(\frac{|{}^1P_{4xz}|^2 - {a_2}^2 - {a_3}^2}{2a_2a_3}\right)$$

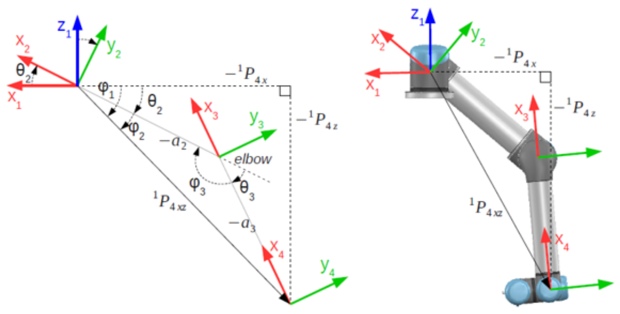


Figure 6: Finding θ_3

Two values could be found for this angle. And then we will find θ_2 as shown:

$$\theta_2 = \phi_1 - \phi_2 = \mathrm{atan2}(-{}^1P_{4z}, -{}^1P_{4x}) - \mathrm{asin}\left(\frac{-a_3 \sin\theta_3}{|{}^1P_{4xz}|}\right)$$

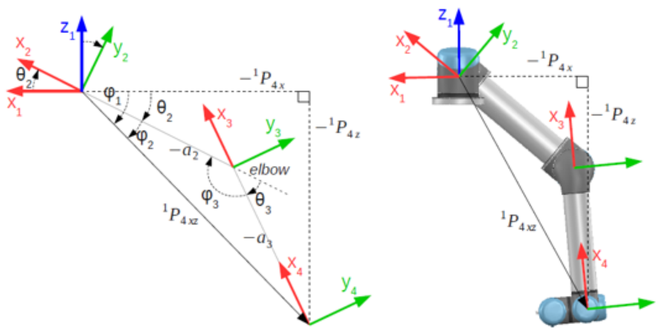


Figure 7: Finding θ_2

θ_4 is defined as the angle from X3 to X4 measured about Z4, and can easily be derived from the last remaining transformation matrix, ${}^3_4T$:

$$\theta_4 = \mathrm{atan2}({}^3\hat{X}_{4y}, {}^3\hat{X}_{4x})$$

To sum up, a total of 8 solutions exists in general for the general inverse kinematic problem of the UR5:

$$2_{\theta_1} \times 2_{\theta_5} \times 1_{\theta_6} \times 2_{\theta_3} \times 1_{\theta_2} \times 1_{\theta_4}$$

2.3 Jacobian Matrix

The Jacobian relates the joint velocities to the linear and angular velocities of the end effector. For rotational joints, we have:

$$J = \begin{bmatrix} J_{L_i} \\ J_{A_i} \end{bmatrix} = \begin{bmatrix} b_{i-1} \times r_{i-1,e} \\ b_{i-1} \end{bmatrix}$$

Where b_{i-1} is the unit vector representing the z-axis of joint $i-1$ with respect to the base (frame 0), $r_{i-1,e}$ is the end-effector position with respect to frame $i-1$, and $J_{L_i}$and J_{A_i} represent the parts of the Jacobian that relate the joint velocities to the linear and angular ones, respectively.

By finding the positions in Matlab, we can calculate each J matrix as:

$$J_A = \begin{bmatrix} 0 & s_1 & s_1 & s_1 & c_1 s_{234} & r_{13} \\ 0 & -c_1 & -c_1 & -c_1 & s_1 s_{234} & r_{23} \\ 1 & 0 & 0 & 0 & -c_{234} & r_{33} \end{bmatrix}$$

$$J_{L_1} = \begin{bmatrix} -p_y \\ p_x \\ 0 \end{bmatrix} \qquad J_{L_2} = \begin{bmatrix} -c_1(p_z - d_1) \\ -s_1(p_z - d_1) \\ s_1 p_y + c_1 p_x \end{bmatrix} \qquad J_{L_3} = \begin{bmatrix} c_1(s_{234}s_5 d_6 + c_{234} d_5 - s_{23} a_3) \\ s_1(s_{234}s_5 d_6 + c_{234} d_5 - s_{23} a_3) \\ -c_{234}s_5 d_6 + s_{234} d_5 + c_{23} a_3 \end{bmatrix}$$

$$J_{L_4} = \begin{bmatrix} c_1(s_{234}s_5 d_6 + c_{234} d_5) \\ s_1(s_{234}s_5 d_6 + c_{234} d_5) \\ -c_{234}s_5 d_6 + s_{234} d_5 \end{bmatrix} \qquad J_{L_5} = \begin{bmatrix} -d_6(s_1 s_5 + c_1 c_{234} c_5) \\ d_6(c_1 s_5 - c_{234} c_5 s_1) \\ -c_5 s_{234} d_6 \end{bmatrix} \qquad J_{L_6} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

2.4 Singularities

To find the configurations where the robot faces a singularity, the determinant of the Jacobian matrix should be computed and be set to zero. By doing so in Matlab, we will get:

$$\det(J) = s_3 s_5 a_2 a_3 (c_2 a_2 + c_{23} a_3 + s_{234} d_5)$$

There are three configurations where singularities occur; The first on happens when the last term becomes zero (Shoulder singularity):

$$c_2 a_2 + c_{23} a_3 + s_{234} d_5 = 0$$

The second configuration happens if $s_5 = 0$, that we call it a wrist singularity. The last one happens when $s_3 = 0$. It is called an elbow singularity. Depicted below is an example of the robot in a shoulder singularity, where the joints can not move the end-effector towards the z6 direction.

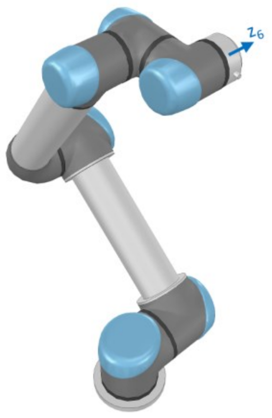


Figure 8: UR5 robot in shoulder singularity

2.5 Trajectory in Cartesian space

In this part, we define a 5 DoF task (e.g., welding) where 3 values for pose vector and 2 values for the angles are needed.

2.5.1 Jacobian-based method (pseudo-inverse)

Here, we define desired $\dot{r}$ with Simulink blocks and cubic interpolation and find the equivalent $\dot{q}$ with $\dot{q} = J_{DLS}\dot{r}$, where $J_{DLS} = J^T(JJ^T + \mu^2 I)^{-1}$ for full row rank matrix. By using the damped Jacobian, we can avoid cross singularities. The block diagram and plots are as follows:

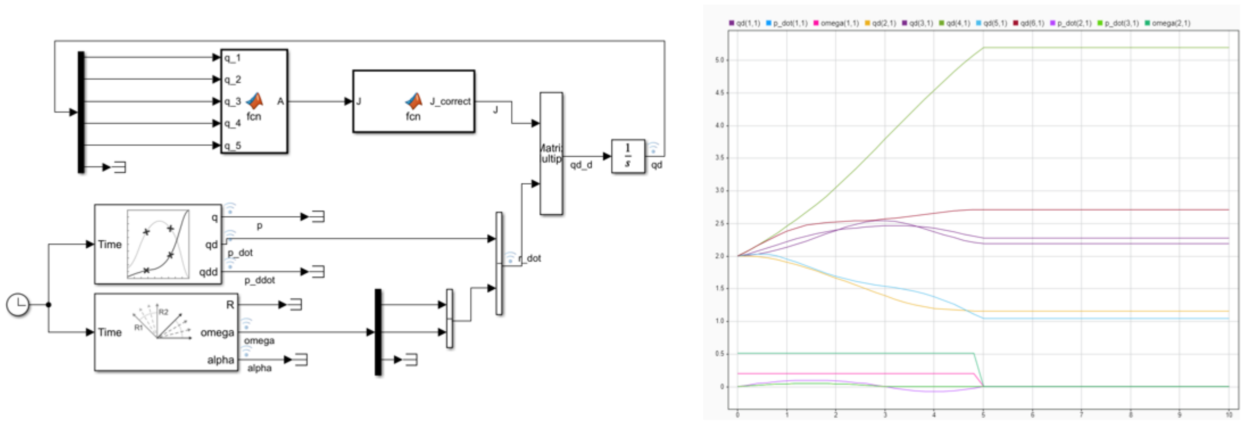


Figure 9: Block diagram and results of pseudo-inverse method trajectory planning

3. Dynamic Analysis

In this section, I have used the Euler-Lagrange method in Matlab to derive the matrices representing Mass inertia, Coriolis/Centrifugal matrix and Gravity. All of these have been done using symbolic expressions, and the matrices are saved to the directory. Unfortunately, the expressions are so long that cannot be shown in the Matlab command window or in this report.

3.1 Mass Matrix (M)

This matrix can be calculated using the following equation and the Jacobians we had calculated before:

$$\sum_{i=1}^{N} m_i J_{Li}^T(q) J_{Li}(q) + J_{Ai}^T(q) I_{ci} J_{Ai}(q)$$

The resulting matrix is saved as "UR5M.txt" in the attached documents.

3.2 Coriolis / Centrifugal Matrix (C)

This matrix can be calculated using the following equation and the Mass matrix we just calculated before:

$$C_k(q) = \frac{1}{2}\left(\frac{\partial \overset{\text{K-th column of } M}{M_k}}{\partial q} + \left(\frac{\partial M_k}{\partial q}\right)^T - \frac{\partial M}{\partial q_k}\right)$$

The resulting matrix is saved as "UR5C.txt" in the attached documents.

3.3 Gravity Vector ($\boldsymbol{G}$)

This matrix can be calculated using the following equation and the mechanical properties of the robot's links:

$$\boldsymbol{G(q) = \frac{\partial V(q)}{\partial q}}$$

The resulting matrix is saved as "UR5G.txt" in the attached documents.

3.4 Linear Parametrization

The equation of the robot can be written as follows:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau = Y(q,\dot{q},\ddot{q})p$$

To find these matrices, we should solve this equation:

$$\left[\frac{d}{dt}\frac{\partial L}{\partial \dot{q}} - \frac{\partial L}{\partial q}\right]^T = \sum_{i=1}^{n}\left[\frac{d}{dt}\frac{\partial L^{(i)}}{\partial \dot{q}} - \frac{\partial L^{(i)}}{\partial q}\right]^T = \sum_{i=1}^{n} Y^{(i)}p^{(i)}$$

Based on a work done by Gabriele Porcelli, and using a simplifying assumption that the robot is consisted of two parts (Base, Shoulder, and Elbow (BSE) and the Wrist group(W)), we can find the regressor matrices as:

$$Y_{BSE} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} \\ \mathbf{0} & Y_{22} & Y_{23} \\ \mathbf{0} & \mathbf{0} & Y_{33} \end{bmatrix} \quad Y_W = \begin{bmatrix} Y_{44} & Y_{45} & Y_{46} \\ \mathbf{0} & Y_{55} & Y_{56} \\ \mathbf{0} & \mathbf{0} & Y_{66} \end{bmatrix} \quad Y'_{total} = \begin{bmatrix} Y_{11} & Y_{12} & Y_{13} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & Y_{22} & Y_{23} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & Y_{33} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & Y_{44} & Y_{45} & Y_{46} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & Y_{55} & Y_{56} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} & Y_{66} \end{bmatrix}$$

And by using QR decomposition technique, the resultant matrices will be:

$$p_{B_{BSE}} = \left\{\begin{pmatrix} J_{1zz} + J_{3yy} - J_{3zz} \\ m_2 \\ m_2 p_{2G_{2_x}} \\ m_2 p_{2G_{2_y}} \\ J_{2xy} \\ J_{2yy} - J_{2zz} \\ J_{2yz} \\ m_3 \\ m_3 p_{3G_{3_x}} \\ m_3 p_{3G_{3_y}} \\ m_3 p_{3G_{3_z}} \\ J_{3xx} - J_{3yy} + J_{3zz} \\ J_{3xy} \\ J_{3xz} \\ J_{3yz} \end{pmatrix}\right\} \quad p_{B_W} = \left\{\begin{pmatrix} 0.2m_6 p_{6G_{6_z}} - 0.2m_5 p_{5G_{5_y}} \\ m_5 p_{5G_{5_z}} \\ m_5 p_{5G_{5_z}} + m_6 p_{6G_{6_z}} \\ J_{5xy} \\ J_{5xz} \\ 0.2m_6 p_{5G_{5_z}} \\ J_{5yz} \\ J_{5zz} \\ m_6 p_{6G_{6_x}} \\ m_6 p_{6G_{6_y}} \\ J_{6xx} - J_{6yy} \\ J_{6xy} \\ J_{6xz} \\ J_{6yz} \\ J_{6zz} \end{pmatrix}\right\}$$

4. Controllers

In this part, several controllers have been designed and simulated in Simulink. For each controller, the block diagram and plots are shown and discussed.

4.1 Regulation

In a regulation problem, we want the robot to reach a specific configuration and stay there.

4.1.1 PD with no gravity compensation

The control law here is:

$$u = K_P(q_d - q) - K_D\dot{q}$$

We choose Kp and Kd to be positive-definite and symmetric. The desired joint configuration is chosen to be:

$$q_d = [\frac{\pi}{4}, \frac{\pi}{2}, \frac{\pi}{8}, \frac{\pi}{3}, \frac{\pi}{4}, \frac{\pi}{5}]'$$

For simplicity, a PID block is used and in this case, K_I is chosen to be zero. The simulation has been done with $K_P = 300$ and $K_d = 20$.

The block diagram is illustrated below:

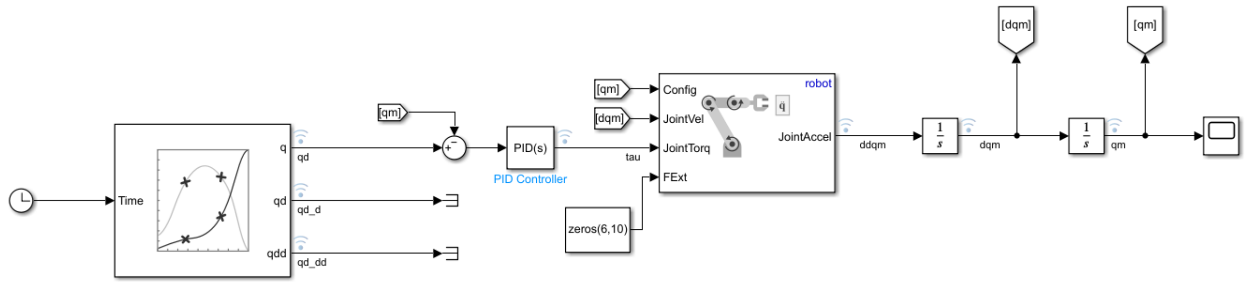


Figure 10: Block diagram of PD with no gravity compensation

And the resulting plot is as follows:

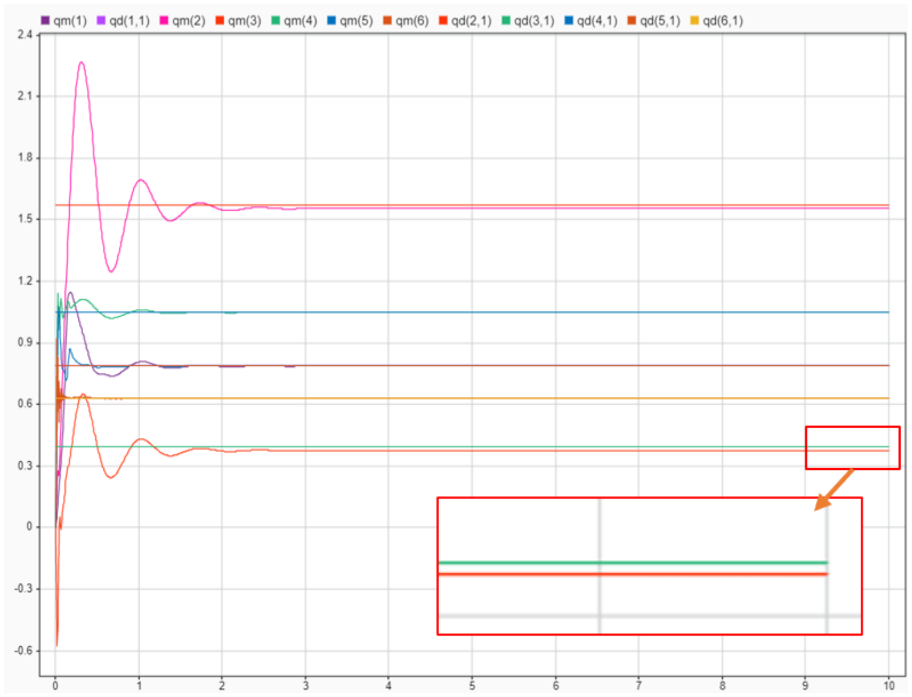


Figure 11: result of PD with no gravity compensation

Since we did not compensate for the gravity of the robot, we can see that there exists a steady-state error for some of the joint positions.

4.1.2 PD with exact gravity compensation

Here, we assume we know the exact characteristics of the robot and we add the exact gravity term to the control law, to compensate for the gravity.

$$u = K_P(q_d - q) - K_D\dot{q} + g(q)$$

The block diagram is shown here:

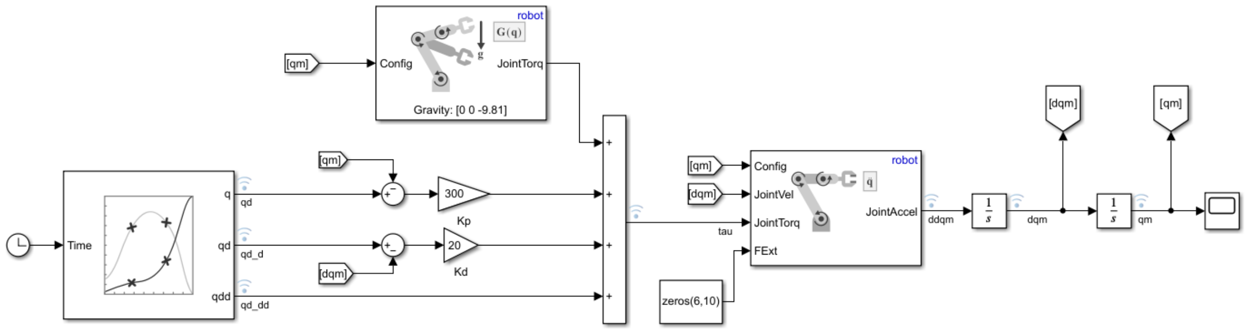


Figure 12: Block diagram of PD with exact gravity compensation

And the results are plotted as:

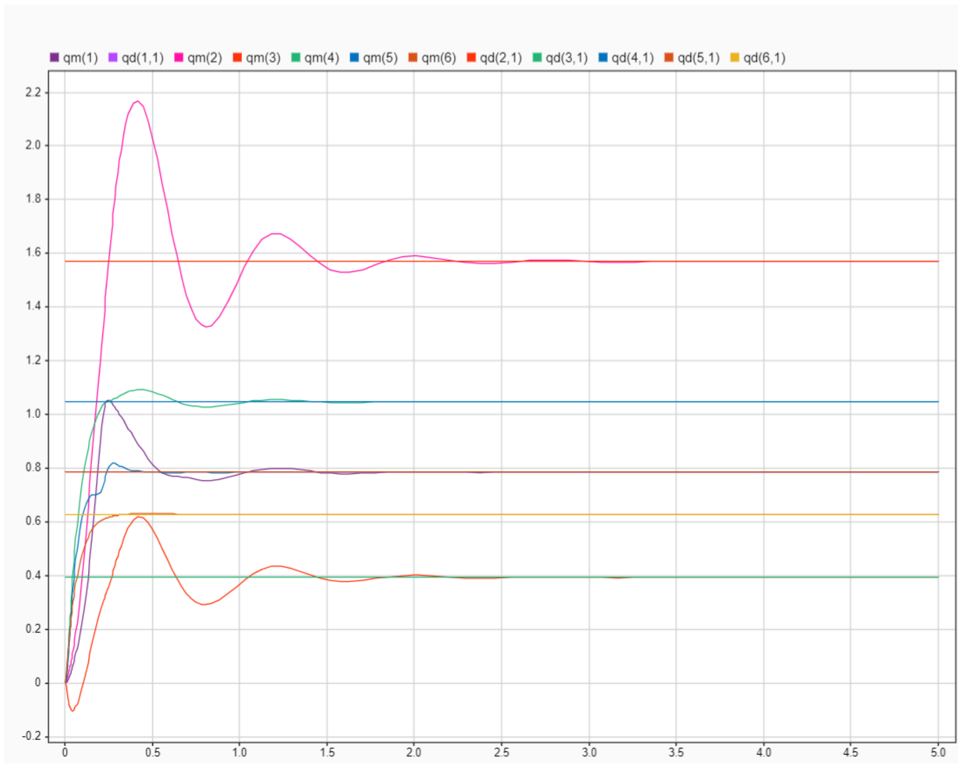


Figure 13: Results of PD with exact gravity compensation

We can see that all the joints could reach the desired configurations.

4.1.3 PD with inexact gravity compensation

In this part, the block diagram remains the same, but we multiply the output of G block by 10%.

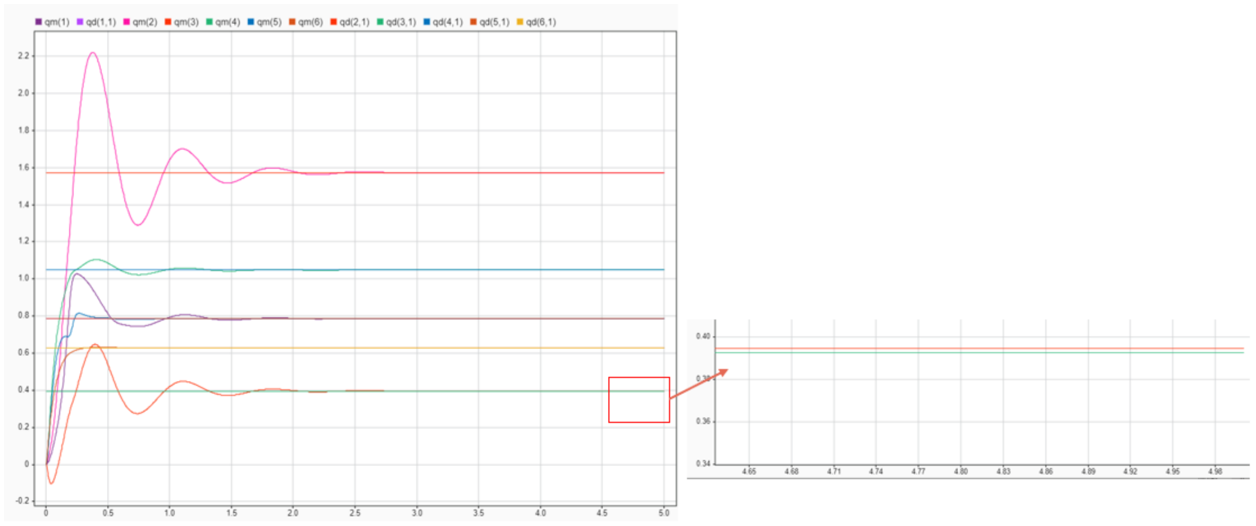


Figure 14: Results of PD with inexact gravity compensation

By zooming into the steady-state answers, we can see the offset. It means that if we cannot have the exact parameters of the robot (the real-world case), we won't be able to reach the exact position with a PD controller.

4.1.4 PD with gravity compensation at the desired configuration

Here, we assume we know the exact gravity vector at the desired configuration. So, the control law will be:

$$u = K_P(q_d - q) - K_D\dot{q} + g(q_d)$$

The block diagram is shown here:

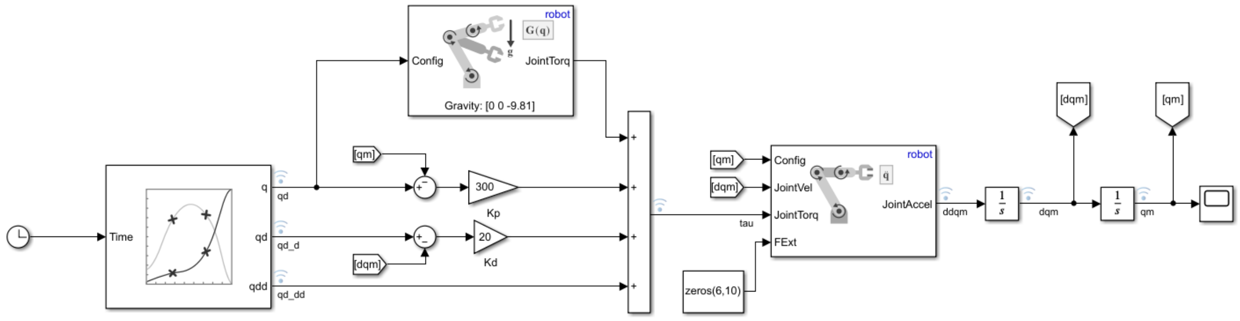


Figure 15: Block diagram of PD with gravity compensation at the desired configuration

And the results are plotted as:

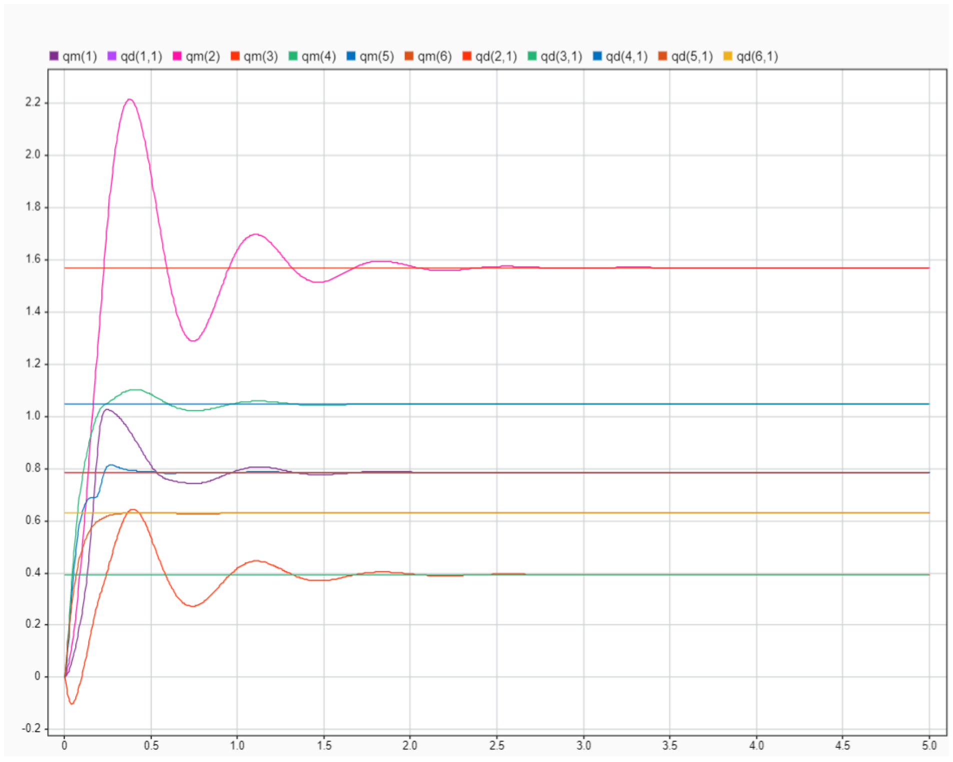


Figure 16: Results of PD with gravity compensation at the desired configuration

We can see that all the joints could reach the desired configurations.

4.1.5 PID without gravity compensation

Here, we assume we don't have the information of the gravity term, but instead, we use an integral term to compensate for this term and make the steady-state error converge to zero. The control law will be:

$$u = K_P\big(q_d - q(t)\big) + K_I\int_0^t \big(q_d - q(\tau)\big)d\tau - K_D\dot{q}(t)$$

The block diagram is the same as the one in section 4.1.1, but we set the K_I term to 50. The resulting plot is as follows.

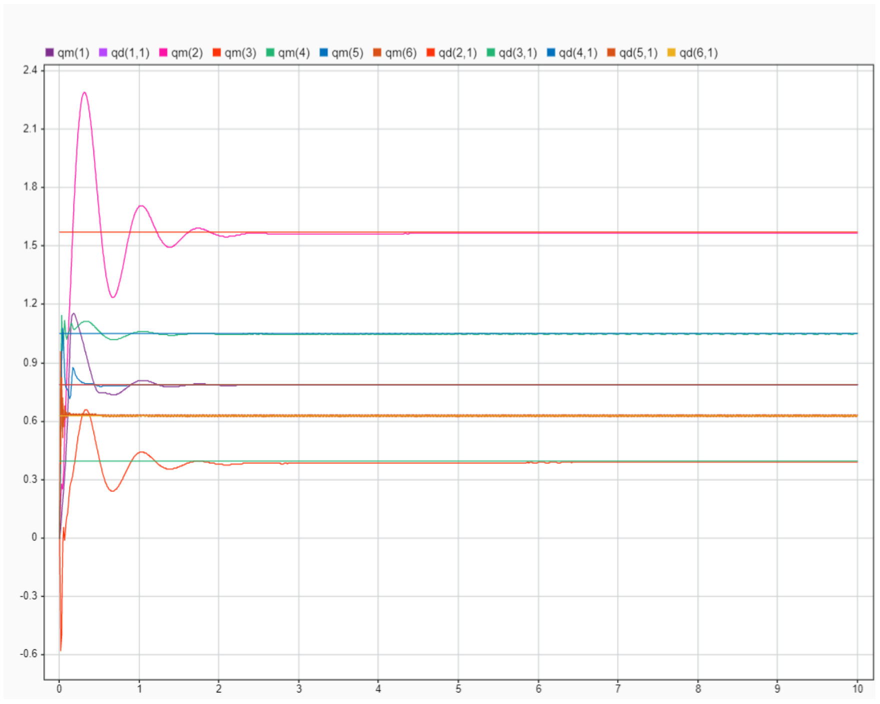


Figure 17: Results of PID without gravity compensation

It could be seen that there are some noises and offset in the steady-state signals. This could be improved by tuning the PID controller, although there is a trade-off, and these noise and errors cannot be fully removed.

4.1.6 ILC for gravity compensation

In this part, we use iterative learning scheme for compensating the unknown gravity vector. The control law and update law are as follows:

$$u = \gamma K_P(q_d - q) - K_d\dot{q} + g(q_{i-1}) \qquad \gamma > 0$$

$$g(q_i) = \gamma K_p(q_d - q_i) + g(q_{i-1})$$

We assume $g_0 = 0$ and $\gamma = 3$ for this simulation. The update law is applied after every 2.5 seconds. The block diagram is shown here:

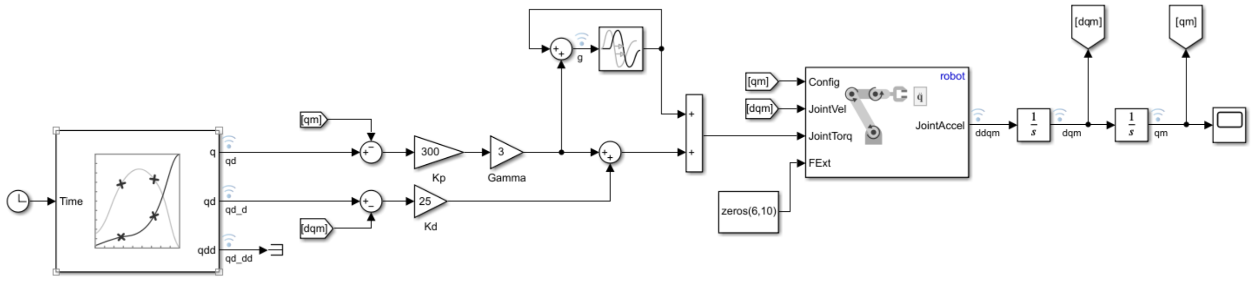


Figure 18: Block diagram of ILC for gravity compensation

And the results are plotted as:

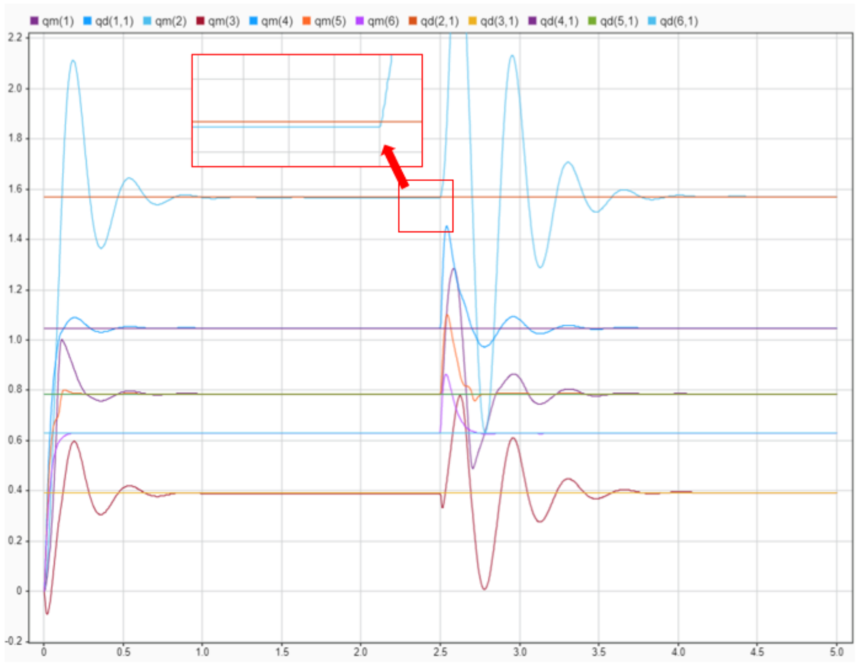


Figure 19: Results of ILC for gravity compensation

We can see that while there remains an offset after the first steady-state configuration, updating the g term can decrease this error to almost zero.

The minimum theoretical Kp with $\gamma = 3$ is about 280. By using 250 and the same update time, we will get:

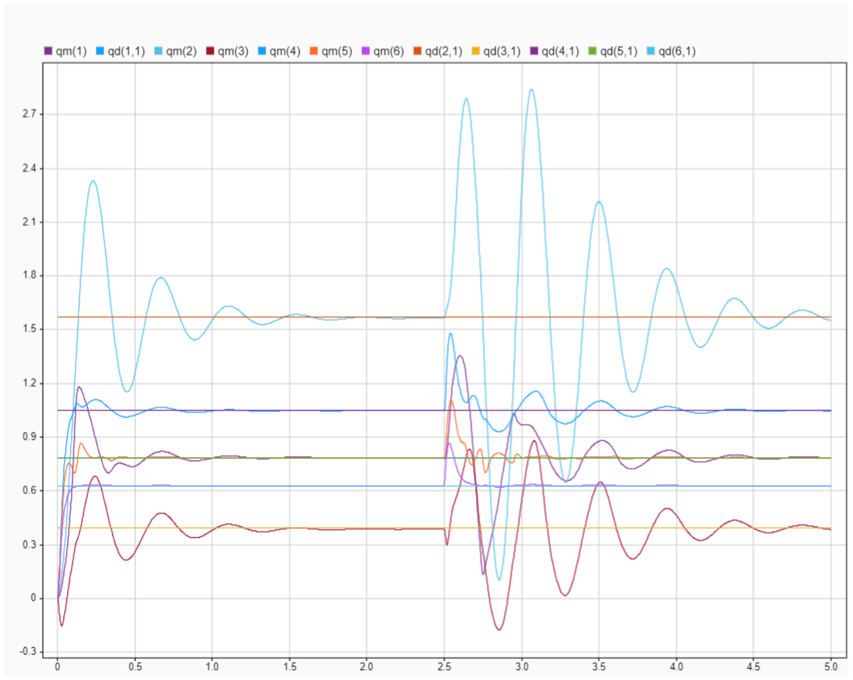


Figure 20: Results of ILC for gravity compensation, Kp lower than needed

We can see that it does not converge, and from the second iteration, oscillation increases, and it won't settle down after 2.5 seconds (it will be even worse).

4.2 Trajectory tracking

The following controllers are able to follow a trajectory. So, we will use the same final values, except that it takes about 5 seconds to reach there.

4.2.1 Inverse dynamics compensation (FFW)+PD

The control law here is: $u = \hat{u}_d + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q})$ and the block diagram is shown below. The Integral term is set to zero.

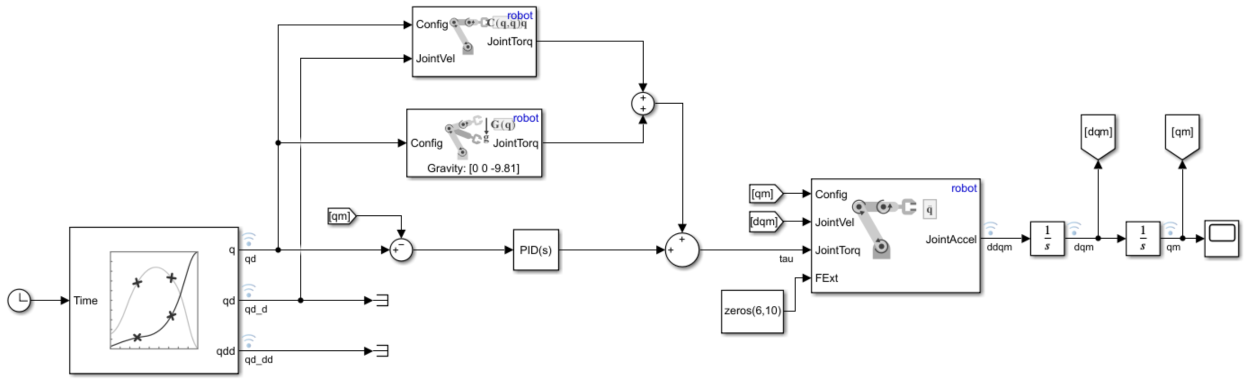


Figure 21: Block diagram of Inverse dynamics compensation (FFW)+PD

The results are plotted as:

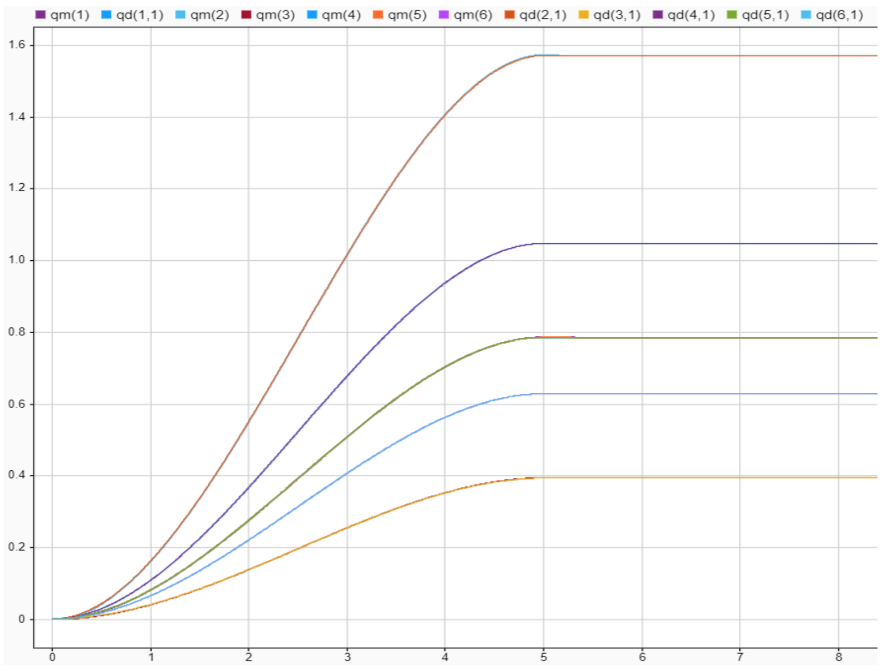


Figure 22: Results of Inverse dynamics compensation (FFW)+PD

We can see the controlled system can follow the trajectory almost instantly.

4.2.2 Inverse dynamics compensation (FFW) + variable PD

The control law here is: $u = \hat{u}_d + \widehat{M}(q_d)[K_P(q_d - q) + K_D(\dot{q}_d - \dot{q})]$ and the block diagram is shown below. The Integral term is set to zero.

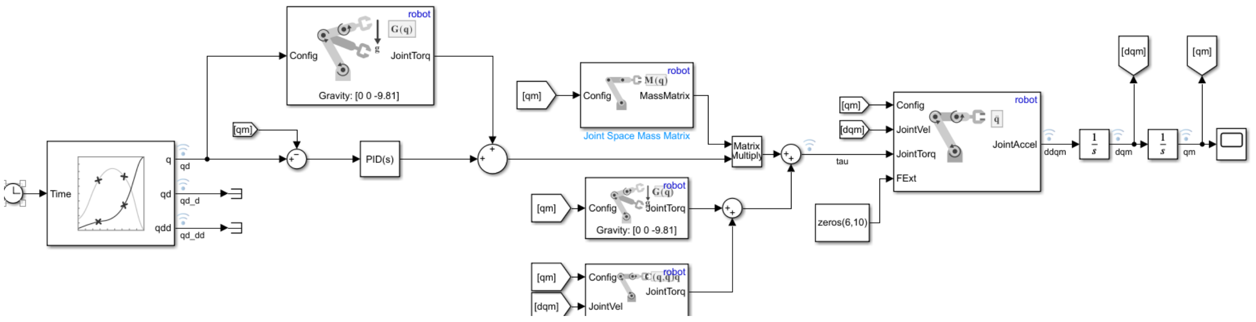


Figure 23: Block diagram of Inverse dynamics compensation (FFW) + variable PD

The results are plotted as:

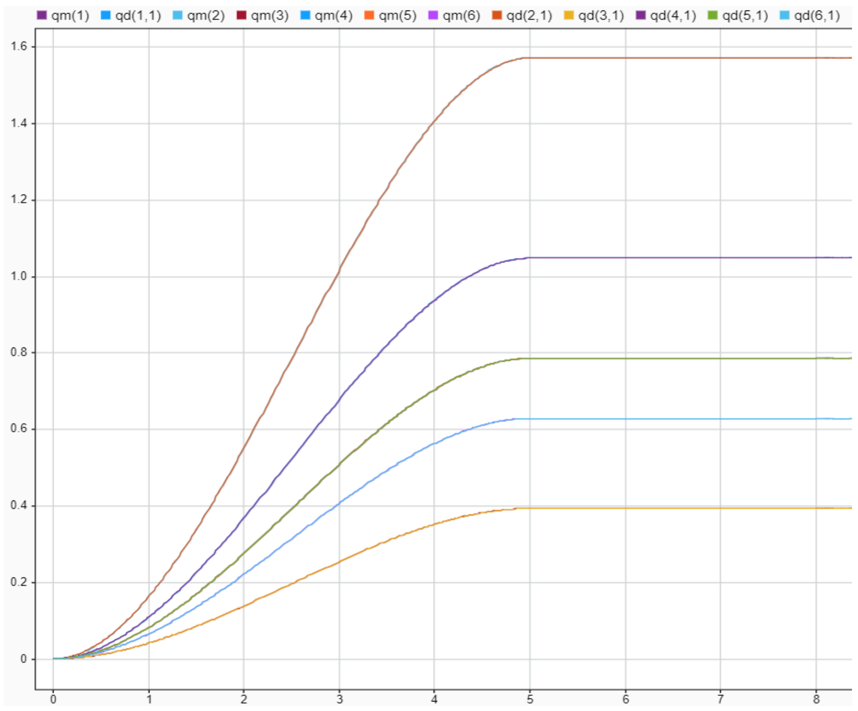


Figure 24: Results of Inverse dynamics compensation (FFW) + variable PD

We can see the controlled system can follow the trajectory even better than the previous one in transitions (from second 4 to 5).

4.2.3 Feedback linearization (FBL) + [PD + FFW]

The control law here is: $u = \widehat{M}(q)[\ddot{q}_d + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q})] + \hat{n}(q, \dot{q})$ and the block diagram is shown below. The Integral term is set to zero.

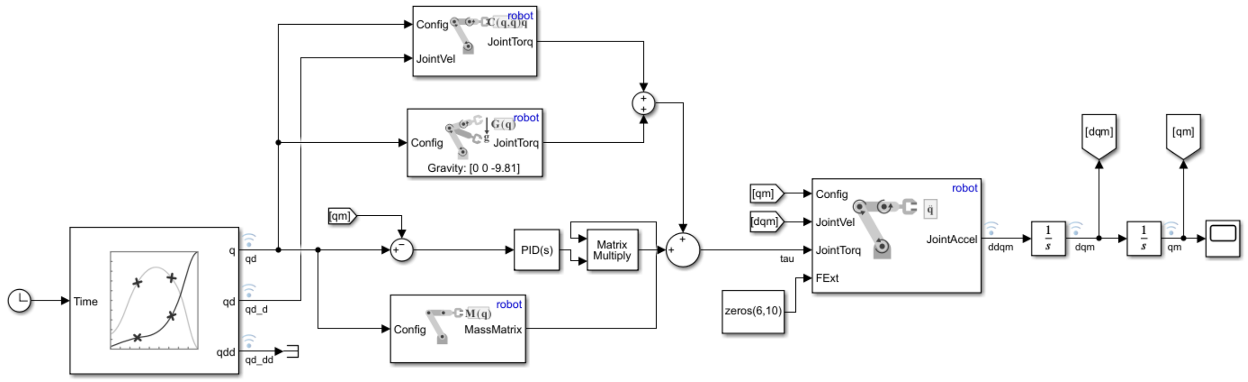


Figure 25: Block diagram of Feedback linearization (FBL) + [PD + FFW]

The results are plotted as:

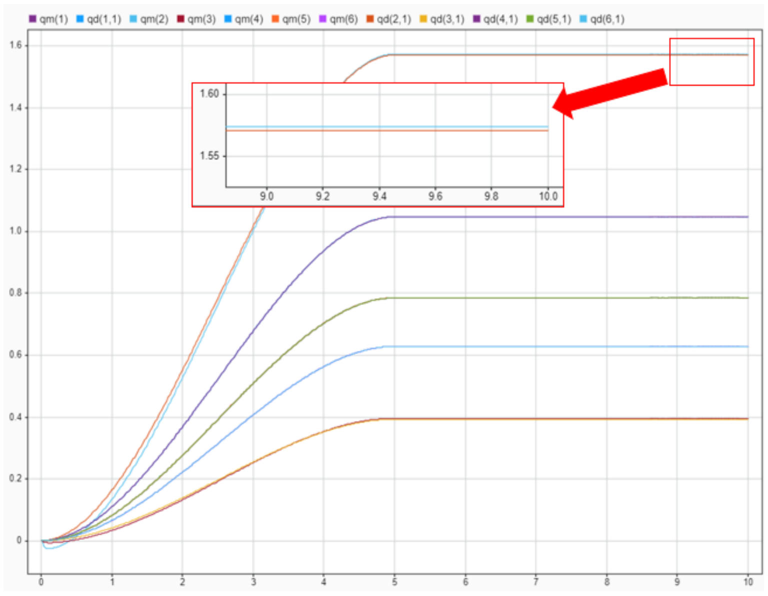


Figure 26: Results of Feedback linearization (FBL) + [PD + FFW]

We can see that there remains steady-state error.

4.2.4 Feedback linearization (FBL) + [PID+FFW]

The control law here is: $uu = \widehat{M}(q)[\ddot{q}_d + K_P(q_d - q) + K_D(\dot{q}_d - \dot{q}) + K_I \int (q_d - q)dt] + \hat{n}(q, \dot{q})$ and the block diagram is same as section 4.2.3. The Integral term is set to 1000. The results are plotted as:

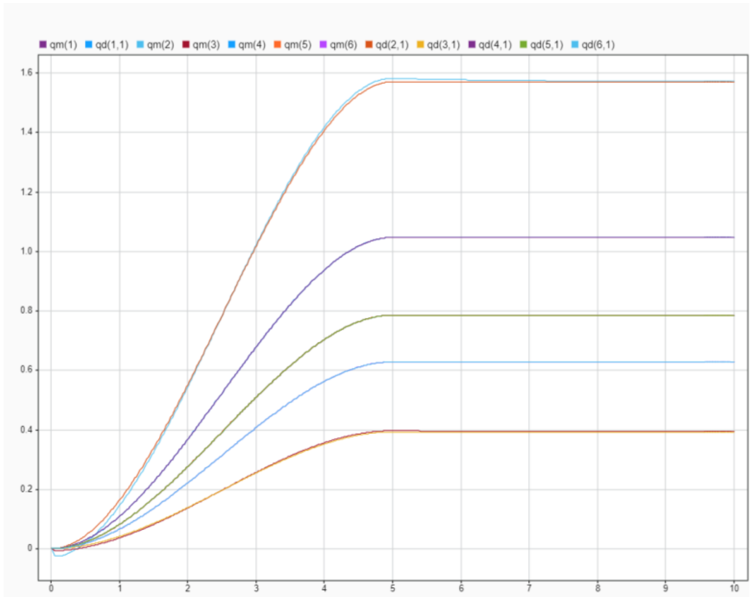


Figure 27: Results of Feedback linearization (FBL) + [PID+FFW]

We can see that the transient following is weaker, but the steady-state error converges to zero.

4.3 Cartesian space control

In this part, our desired configuration is a point in space $p = [0.1, 0.2, 0.3]'$

4.3.1 *P* in task space, *D* and *g* in joint space

The control law here is: $u = J^T(q)K_P(p_d - p) - K_D\dot{q} + g(q)$. The block diagram is shown below.

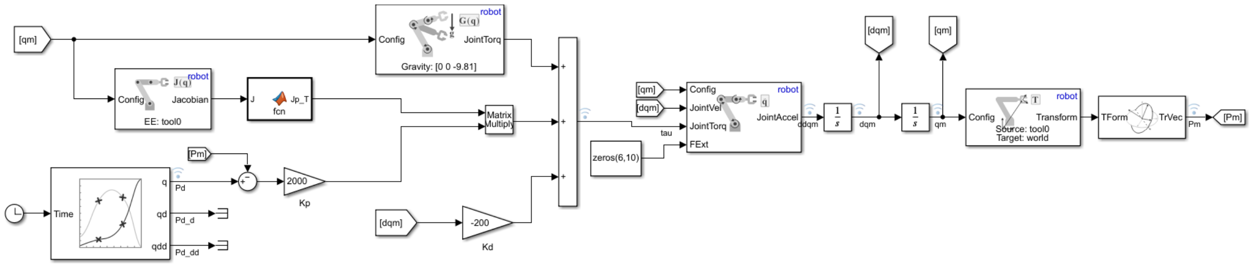


Figure 28: Block diagram of P in task space, D and g in joint space

The results are plotted as:

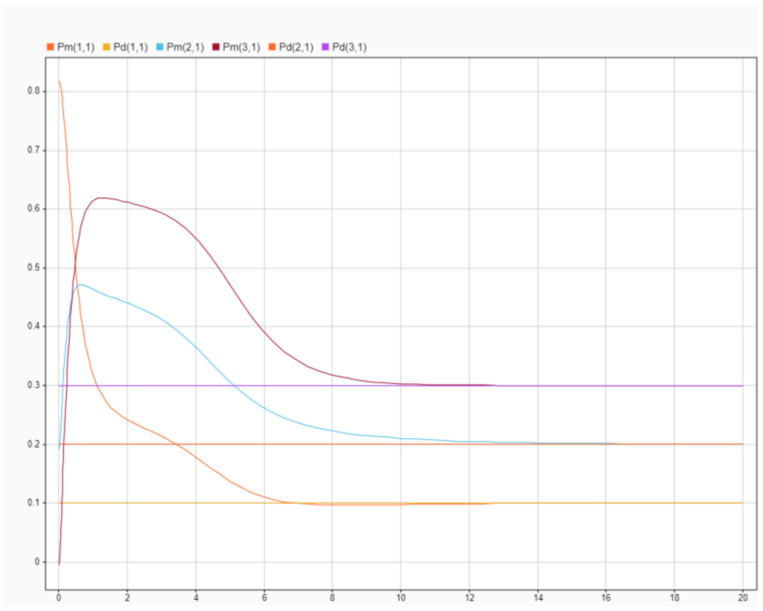


Figure 29: Results of P in task space, D and g in joint space

We can see that steady-state error of the end-effector position converges to zero.

4.3.2 *P* and D in task space, *g* in joint space

The control law here is: $u = J^T(q)K_P(p_d - p) - K_D\dot{q} + g(q)$. The block diagram is shown below.

-++/

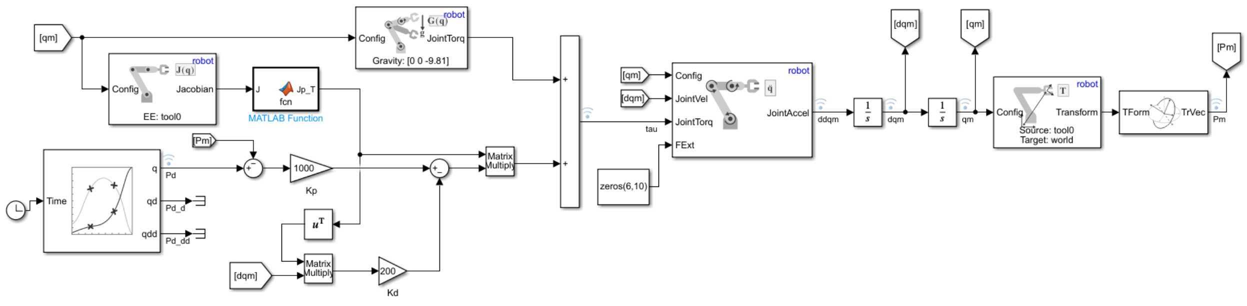


Figure 30: Block diagram of P and D in task space, g in joint space

The results are plotted as:

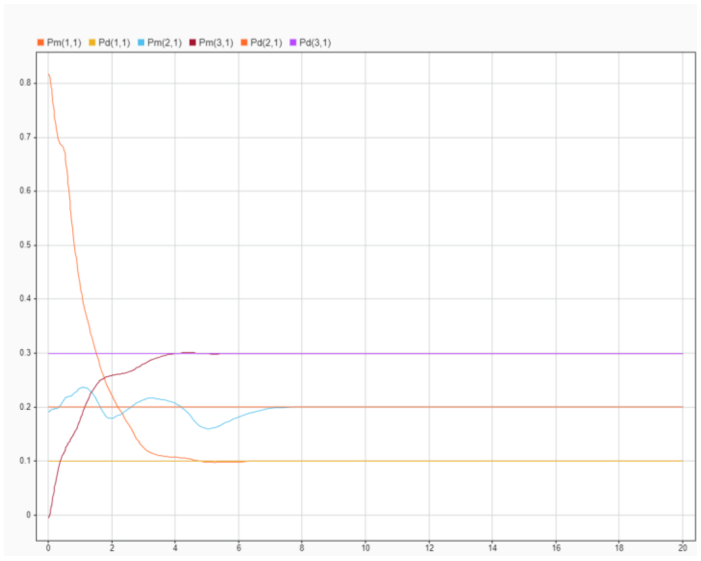
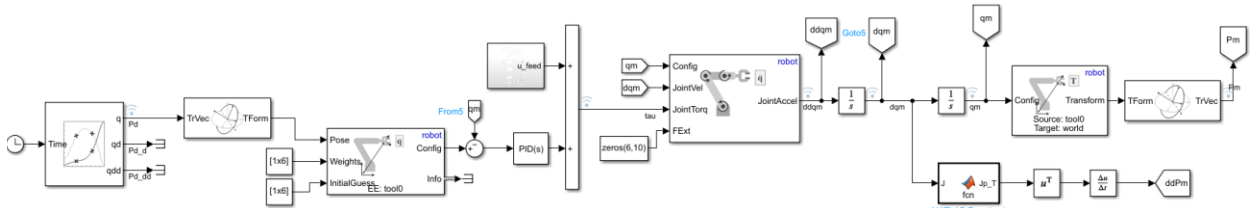


Figure 31: Results of P and D in task space, g in joint space

We can see that with $K_p = 200$ (one tenth of the previous section), the convergence happens much sooner. This is because we pay attention to the error of the end-effector, instead of the joints.

4.3.3 input/output feedback linearization

The control law is of form: $u = M(q)J^{-1}(q)a + c(q,\dot{q}) + g(q) - M(q)J^{-1}(q)\dot{J}(q)\dot{q}$ and block diagram is:



Unfortunately, the computation was very slow, and I could not run it for more than a few milliseconds.