COMP0211 Final Project

November 16, 2024 (Last Update: 20241116–1730)

1 Introduction

In this coursework, you will have to solve different tasks. In the first task, you will have to control a DC motor for which we provide the full electromechanical dynamics.

2 Prerequisites

To run Task 1 you will have to use the Mamba environment called roboenv2. If you do not have first ensure to have Mamba installed and after this download The latest version of RoboEnv from the GitHub page. Inside the repository, you will find a file called *environment_ ros2.yaml*. To install the environment in the terminal you will have to type:

```
mamba env create -f environment_ros2.yaml
```

for more details regarding potential issues with the installation process on different OS please refer to the README.md located in the RoboEnv repository.

3 Downloading the Coursework Code

The coursework code can be found **at this link**

4 Tasks

Task 1: Observer Design and Control Implementation (45%)

In this project, you will work with a DC motor model and implement state estimation and control strategies. The system dynamics are provided below:

Model Description

The DC motor is modeled using the following state-space representation:

State Variables:

$$x_1 = \omega \quad \text{(Angular velocity in rad/s)}$$
$$x_2 = I_a \quad \text{(Armature current in A)}$$

Input:

$$u = V_a \quad \text{(Armature voltage in V)}$$

Output:

$$y = V_{\text{terminal}} = K_e x_1 + R_a x_2 \quad \text{(Terminal voltage in V)}$$

System Equations:

$$\dot{x}_1 = \frac{1}{J}(K_t x_2 - b x_1)$$
$$\dot{x}_2 = \frac{1}{L_a}(u - R_a x_2 - K_e x_1)$$

Parameters:

$$\begin{aligned} J &= 0.01\,\text{kg m}^2 \quad \text{(Rotor inertia)} \\ b &= 0.1\,\text{N m s} \quad \text{(Viscous friction coefficient)} \\ K_t &= 0.01\,\text{N m/A} \quad \text{(Motor torque constant)} \\ K_e &= 0.01\,\text{V s/rad} \quad \text{(Back EMF constant)} \\ R_a &= 1\,\Omega \quad \text{(Armature resistance)} \\ L_a &= 0.001\,\text{H} \quad \text{(Armature inductance)} \end{aligned}$$

State-Space Matrices:

$$A = \begin{bmatrix} -\frac{b}{J} & \frac{K_t}{J} \\ -\frac{K_e}{L_a} & -\frac{R_a}{L_a} \end{bmatrix} = \begin{bmatrix} -10 & 1 \\ -10{,}000 & -1{,}000 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{L_a} \end{bmatrix} = \begin{bmatrix} 0 \\ 1{,}000 \end{bmatrix}, \quad C = \begin{bmatrix} K_e & R_a \end{bmatrix} = \begin{bmatrix} 0.01 & 1 \end{bmatrix}$$

1. **Task 1.1**

   Design a state observer for the DC motor system where only the terminal voltage V_{terminal} is measured. The observer should estimate the full state vector of the system. The function to update is located inside the *observer.py* script and is called

   ```
   update(self, u, y)
   ```

   . After that, you can change the behavior of the observer by updating the value of the desired eigenvalues by updating these 2 lines of code:

   ```
   lambda_1 =
   lambda_2 =
   ```

   located in the file called *task1_1.py*

2. **Task 1.2** In this task first ensure that the Implement a linear Model Predictive Control (MPC) regulator for the DC motor system using the estimated states from the observer as inputs to MPC. The MPC should operate without constraints and terminal cost. First, define the appropriate cost matrices Q and R located in the script called *task1_2.py* to ensure that we are going only to control the *angular velocity* of the system (x_1). Second, to test the controller pass to it the current state of the system

```
x_cur
```

and see if the regulation works. You can change the setpoint (if you want to) by updating the variables called

```
x_ref
```

. Both variables are located in the file called *task1_2.py*. Moreover, check if the state estimator is performing correctly (estimated state close to the real one). If you observe a big discrepancy between estimated state x_hat_cur and x_cur adjust the value of lambda_1, lambda_2 to improve the observer performances. Once the observer is looking good replace x_hat_cur with x_cur in the function called

```
u_mpc = regulator.compute_solution(x_hat_cur)
```

3. **Task 1.3**

   Replace the MPC controller with a Linear Quadratic Regulator (LQR) controller. To this aim, solve the discrete algebraic Riccati equation at line 72 and provide the definition of the gain K at line 75. Both lines are located in the script called *task1_3.py*. Compare the performance of the MPC and LQR controllers using appropriate metrics such as settling time, overshoot, and steady-state error and establish which is the best. Show some plots to show it.

4. **Task 1.4**

   Analyze the robustness of both controllers in the presence of modeling uncertainties or external disturbances. iTo perturb the model you need to alter the matrices that are used both in the constructor of the regulator object and in the computation of the matrices P and K for the linear quadratic regulator. To test for the robustness alter only these 3 parameters K_e, R_a, and L_a in the range of ±20% around the nominal value. For this task is recommended to create a new script called *task1_4.py* to study the effect of the parameter changes on the controller robustness

Hints

- **Task 1.1**: Consider designing a Luenberger observer by choosing observer gains to place the poles of $A - LC$ at desired locations for convergence. Ensure the system is observable with the given C matrix. For this, we provide a function defined in the script *observer.py* called
- **Task 1.2**: Formulate the MPC problem by defining the prediction model using the state-space discrete-time representation. Develop a cost function that penalizes deviations from the desired state and control effort. Since constraints are not included, focus on the weighting matrices to achieve the desired performance.
- **Task 1.3**: Design the LQR controller by selecting an appropriate state and control weighting matrices Q and R. Use the solution of the Algebraic Riccati Equation to compute the optimal feedback gain matrix.
- **Task 1.4**: Introduce modeling uncertainties or disturbances in your simulations, or external torque disturbances) to evaluate the robustness of both controllers.

Task 2: Manipulator Pick and Place (45%)

You are tasked with controlling a simulated manipulator that performs a pick-and-place operation. The manipulator is controlled at the torque level using feedback linearization, and a linear constrained Model Predictive Control (MPC) is utilized for regulation and tracking tasks in joint space.

Activities

1. **Task 2.1** Use the provided code to use a regulation controller using Model Predictive Control (MPC) that drives the manipulator to a specified joint position. First, define the matrix A and B for the continuous case assuming that the system is a chain of integrators (due to the feedback linearization). Drive the robot to the desired position (by setting it in the **q_ref**). Simulate the manipulator's response and plot the joint positions over time.

2. **Task 2.2**

   Your task is to impose individual constraints on each joint of the manipulator. These constraints represent forbidden regions or limitations that the manipulator must avoid. For example, you may set specific minimum and maximum joint angles for certain joints to simulate restricted areas. the robot has to perform a regulation task where the manipulator moves from an initial joint configuration **init_joint_position** to a desired goal configuration **goal_position**. Run the simulation 3 times and for each execution impose very strict position constraints around the joints 1,2,3. After running the simulation, you will report the results by providing plots of the joint trajectories over time, highlighting how each joint moves within its allowed range. Additionally, you should report if the robot end-effector reaches the desired position despite the joint constraints.

Hints

- **task 2.1** refer to the appendix B to see the structure of the Continuos A and B matrices. Then execute the code and briefly comment the results

- **task 2.2** to change the constraint you need to update the variables **B_in** and **B_out** according to the specified requests

Task 3: Using LTV Formulation to Manage Non-Linearities (10%)

In this task, you will explore an alternative control strategy for the manipulator by assuming that it is controlled using **gravity cancellation** instead of feedback linearization. Gravity cancellation involves compensating for the gravitational forces acting on the manipulator by applying torques that counteract gravity. Since the gravitational torque depends only on the link masses and joint positions, and does not involve velocities or accelerations, it is simpler to implement gravity cancellation compared to full feedback linearization, which requires precise modeling of the manipulator's entire dynamics.

Your objective is to implement a tracking MPC without constraints to control the manipulator under gravity cancellation. Specifically, you will:

Activities

1. **Task 3.1**

Assume the manipulator is now controlled using gravity cancellation instead of feedback linearization. Use the linear MPC tracker provided in the code that does not account for the nonlinear dynamics. Update the provided code to reflect this control strategy. Simulate the manipulator following the desired trajectory and record the tracking performance.

2. **Task 3.2**

   Enhance your controller by using a linear time-varying (LTV) formulation for the MPC. Linearize the robot's dynamics around the desired joint trajectory to obtain a time-varying linear model. Update the `class ModelTracker` code to implement this LTV MPC controller. Ensure that the controller updates the linearized model at each time step based on the desired trajectory.

3. **Task 3.3**

   Compare the results from the standard MPC tracker (from Task 3.1) and the LTV MPC tracker (from Task 3.2). Analyze the performance differences between the two approaches in terms of tracking accuracy, control effort, and any observed deviations from the desired trajectory. Provide plots and discuss your findings.

Hints

- **Task 3.1:**

  Utilize the provided code for the standard MPC controller. Since you are using gravity cancellation, the control input should compensate for gravitational torques. The remaining dynamics can be approximated as linear for small deviations. Assume that the system is fully feedback linearized and used as an internal model for the MPC the one that has been defined in task 2

- **Task 3.2:**

  Linearize the manipulator's nonlinear dynamics around the desired trajectory at each time step. This involves calculating the Jacobian matrices $A(t)$ and $B(t)$ that represent the system's state-space model. Update these matrices in your MPC formulation to create an LTV MPC. Modify the `class ModelTracker` to include this time-varying model.

- **Task 3.3:**

  Compare the tracking errors by plotting the difference between the desired and actual joint positions over time for both controllers. Analyze the control inputs (torques) applied in each case. Discuss how accounting for the nonlinear dynamics in the LTV MPC improves performance compared to the standard MPC.

Appendix

A Motor Dynamics Derivation

State Variables

Let's define the state variables:

$$x_1 = \omega \quad \text{(Angular velocity)}$$
$$x_2 = I_a \quad \text{(Armature current)}$$

Input

$$u = V_a \quad \text{(Armature voltage, known input)}$$

Output

$$y = V_{\text{terminal}} \quad \text{(Measured terminal voltage)}$$

Motor Equations

Electrical Dynamics

The voltage equation for the armature circuit is:

$$V_a = R_a I_a + L_a \frac{dI_a}{dt} + K_e \omega$$

Rewriting:

$$L_a \frac{dI_a}{dt} = V_a - R_a I_a - K_e \omega$$

Mechanical Dynamics

The motion equation for the motor is:

$$J \frac{d\omega}{dt} = K_t I_a - b\omega - T_L$$

Assuming the load torque T_L is known or negligible:

$$J \frac{d\omega}{dt} = K_t I_a - b\omega$$

State-Space Representation

State Equations

Express the dynamics in state-space form:

$$\dot{x}_1 = \frac{1}{J}(K_t x_2 - b x_1)$$
$$\dot{x}_2 = \frac{1}{L_a}(u - R_a x_2 - K_e x_1)$$

Output Equation

The terminal voltage V_{terminal} is the voltage across the armature resistance and back EMF:

$$y = V_{\text{terminal}} = R_a x_2 + K_e x_1$$

State-Space Matrices

The system can be represented in matrix form:

$$\begin{aligned}\dot{x} &= Ax + Bu \\ y &= Cx\end{aligned}$$

Where:

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}, \quad u = V_a$$
$$A = \begin{bmatrix} -\frac{b}{J} & \frac{K_t}{J} \\ -\frac{K_e}{L_a} & -\frac{R_a}{L_a} \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ \frac{1}{L_a} \end{bmatrix}, \quad C = \begin{bmatrix} K_e & R_a \end{bmatrix}$$

Observer Design

Objective

Design an observer to estimate the state vector x using the measured output $y = V_{\text{terminal}}$.

Observer Structure

The observer (Luenberger observer) has the form:

$$\begin{aligned}\dot{\hat{x}} &= A\hat{x} + Bu + L(y - \hat{y}) \\ \hat{y} &= C\hat{x}\end{aligned}$$

Where:

- $\hat{x}$ is the estimated state vector.
- $\hat{y}$ is the estimated output.
- L is the observer gain matrix to be designed.

Error Dynamics

Define the estimation error:

$$e = x - \hat{x}$$

The error dynamics are given by:

$$\dot{e} = (A - LC)e$$

To ensure e converges to zero, we need to design L such that the eigenvalues of $A - LC$ are placed in the left half of the complex plane (for continuous-time systems), ensuring stability and convergence.

Designing the Observer Gain L

Step 1: Compute $A - LC$

Let $L = \begin{bmatrix} L_1 \\ L_2 \end{bmatrix}$. Then:

$$A - LC = \begin{bmatrix} -\frac{b}{J} & \frac{K_t}{J} \\ -\frac{K_e}{L_a} & -\frac{R_a}{L_a} \end{bmatrix} - \begin{bmatrix} L_1 \\ L_2 \end{bmatrix} \begin{bmatrix} K_e & R_a \end{bmatrix}$$

Compute the matrix:

$$A - LC = \begin{bmatrix} -\frac{b}{J} - L_1 K_e & \frac{K_t}{J} - L_1 R_a \\ -\frac{K_e}{L_a} - L_2 K_e & -\frac{R_a}{L_a} - L_2 R_a \end{bmatrix}$$

Step 2: Set Up the Characteristic Equation

Let $\Phi = A - LC$, then the characteristic equation is:

$$\det(sI - \Phi) = 0$$

Compute the determinant:

$$|sI - (A - LC)| = 0$$

Step 3: Equate to Desired Characteristic Polynomial

Choose desired eigenvalues λ_1 and λ_2 for $A - LC$ (e.g., negative real numbers for stability). The desired characteristic equation is:

$$(s - \lambda_1)(s - \lambda_2) = s^2 - (\lambda_1 + \lambda_2)s + \lambda_1 \lambda_2$$

Step 4: Solve for L

Set up the equations by equating the coefficients of the characteristic equations:

$$\det(sI - (A - LC)) = s^2 + a_1 s + a_0$$

Equate:

$$s^2 + a_1 s + a_0 = s^2 - (\lambda_1 + \lambda_2)s + \lambda_1 \lambda_2$$

Solve for L_1 and L_2 such that:

$$a_1 = -(\lambda_1 + \lambda_2), \quad a_0 = \lambda_1 \lambda_2$$

Detailed Computation

Let's compute a_1 and a_0 in terms of L_1 and L_2.

First, compute $\Phi = A - LC$:

$$\Phi = \begin{bmatrix} \Phi_{11} & \Phi_{12} \\ \Phi_{21} & \Phi_{22} \end{bmatrix}$$

Where:

$$\Phi_{11} = -\frac{b}{J} - L_1 K_e$$
$$\Phi_{12} = \frac{K_t}{J} - L_1 R_a$$
$$\Phi_{21} = -\frac{K_e}{L_a} - L_2 K_e$$
$$\Phi_{22} = -\frac{R_a}{L_a} - L_2 R_a$$

Compute the characteristic equation:

$$\det(sI - \Phi) = \begin{vmatrix} s - \Phi_{11} & -\Phi_{12} \\ -\Phi_{21} & s - \Phi_{22} \end{vmatrix} = 0$$

Compute the determinant:

$$(s - \Phi_{11})(s - \Phi_{22}) - \Phi_{12}\Phi_{21} = 0$$

Expand:

$$s^2 - (\Phi_{11} + \Phi_{22})s + (\Phi_{11}\Phi_{22} - \Phi_{12}\Phi_{21}) = 0$$

Therefore:

$$a_1 = -(\Phi_{11} + \Phi_{22})$$
$$a_0 = \Phi_{11}\Phi_{22} - \Phi_{12}\Phi_{21}$$

Equate to Desired Characteristic Polynomial Set:

$$a_1 = -(\lambda_1 + \lambda_2)$$
$$a_0 = \lambda_1\lambda_2$$

Solving for L_1 and L_2

Now, express a_1 and a_0 in terms of L_1 and L_2, then solve the equations.

Expression for a_1 and a_0

Substituting a_1 and a_0 with the expressions in terms of the observer gain components:

$$-\Phi_{11} - \Phi_{22} = -(\lambda_1 + \lambda_2),$$
$$\Phi_{11}\Phi_{22} - \Phi_{12}\Phi_{21} = \lambda_1\lambda_2.$$

B Feedback Linearization Control and Resulting Dynamics

The general equation of motion for a manipulator is given by:

$$M(q)\ddot{q} + n(q, \dot{q}) = u$$

In this equation: - $M(q)$ is the mass (inertia) matrix, which is symmetric and positive-definite. - $n(q, \dot{q})$ represents the nonlinear terms, including Coriolis, centrifugal, and gravitational forces. - u is the control input vector (joint torques).

To simplify the system dynamics, we design the control input u to cancel out the nonlinear terms. The control law is defined as:

$$u = M(q)a + n(q, \dot{q})$$

Here, a is the desired acceleration vector for the manipulator joints.

By substituting the control law into the manipulator dynamics, we get:

$$M(q)\ddot{q} + n(q, \dot{q}) = M(q)a + n(q, \dot{q})$$

Subtracting $n(q, \dot{q})$ from both sides simplifies the equation:

$$M(q)\ddot{q} = M(q)a$$

Assuming $M(q)$ is invertible (which it is, since it's positive-definite), we multiply both sides by $M^{-1}(q)$:

$$\ddot{q} = a$$

This shows that the manipulator's actual acceleration $\ddot{q}$ directly follows the desired acceleration a.

We define the state vector x by combining the position and velocity vectors:

$$x = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$$

This state vector x is a 14×1 column vector for a 7-DOF manipulator.

The state-space representation of the system dynamics is then expressed as:

$$\dot{x} = Ax + Ba$$

Where: - $\dot{x}$ is the derivative of the state vector. - A is the system matrix. - B is the input matrix.

The matrices A and B are given by:

$$A = \begin{bmatrix} 0_{7\times7} & I_{7\times7} \\ 0_{7\times7} & 0_{7\times7} \end{bmatrix}, \quad B = \begin{bmatrix} 0_{7\times7} \\ I_{7\times7} \end{bmatrix}$$

In these matrices: - $0_{7\times7}$ is a 7×7 zero matrix. - $I_{7\times7}$ is a 7×7 identity matrix.

The state derivative $\dot{x}$ consists of the velocities and accelerations:

$$\dot{x} = \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ a \end{bmatrix}$$

This shows that the rate of change of the state vector depends on the current velocities and the desired accelerations.

The system matrix A represents the relationships between the state variables:

$$A = \begin{bmatrix} \frac{\partial \dot{q}}{\partial q} & \frac{\partial \dot{q}}{\partial \dot{q}} \\ \frac{\partial \ddot{q}}{\partial q} & \frac{\partial \ddot{q}}{\partial \dot{q}} \end{bmatrix} = \begin{bmatrix} 0_{7\times7} & I_{7\times7} \\ 0_{7\times7} & 0_{7\times7} \end{bmatrix}$$

- The upper-right block $I_{7\times7}$ indicates that the derivative of position q with respect to velocity $\dot{q}$ is the identity matrix. - The other blocks are zero because, after feedback linearization, acceleration $\ddot{q}$ does not directly depend on q or $\dot{q}$.

The input matrix B shows how the control input a affects the state derivative:

$$B = \begin{bmatrix} 0_{7\times7} \\ I_{7\times7} \end{bmatrix}$$

- The lower block $I_{7\times7}$ indicates that the control input a directly influences the acceleration $\ddot{q}$.
- The upper block is zero because the control input does not directly affect the position q.

By employing the control law $u = M(q)a + n(q, \dot{q})$, the nonlinear dynamics are effectively canceled through feedback linearization. The manipulator is now represented by a linear state-space model, which simplifies the controller design process. The dynamics reduce to seven decoupled double integrators, allowing for independent control of each joint.

C Linear Time-Varying (LTV) MPC Formulation

Cost Function

The cost function $J(z, x_0)$ for the LTV MPC is defined as:

$$\begin{aligned} J(z, x_0) = x_0^\top Q x_0 + &\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}^\top \begin{bmatrix} Q & 0 & \cdots & 0 \\ 0 & Q & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & P \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} \\ + &\begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix}^\top \begin{bmatrix} R & 0 & \cdots & 0 \\ 0 & R & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & R \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \end{aligned} \tag{1}$$

State Evolution Equations

Subject to the state evolution equations:

$$\begin{aligned} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = &\underbrace{\begin{bmatrix} B_0 & 0 & \cdots & 0 \\ A_1 B_0 & B_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_{N-1}\cdots A_1 B_0 & A_{N-1}\cdots A_2 B_1 & \cdots & B_{N-1} \end{bmatrix}}_{\text{Matrix } S} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} \\ + &\underbrace{\begin{bmatrix} A_0 \\ A_1 A_0 \\ \vdots \\ A_{N-1}\cdots A_0 \end{bmatrix}}_{\text{Matrix } T} x_0 \end{aligned} \tag{2}$$

Simplified Cost Function

The cost function can be rewritten as:

$$\begin{aligned} J(z, x_0) &= (Sz + Tx_0)^\top Q (Sz + Tx_0) + z^\top R z + x_0^\top Q x_0 \\ &= \frac{1}{2} z^\top (2R + 2S^\top Q S) z + x_0^\top (2T^\top Q S) z + \frac{1}{2} x_0^\top (2Q + 2T^\top Q T) x_0 \end{aligned} \tag{3}$$

Definitions of A_k and B_k

The matrices A_k and B_k are the linearized system matrices at each time step k, obtained by linearizing the manipulator's nonlinear dynamics around the desired trajectory $(\mathbf{q}_d(k), \dot{\mathbf{q}}_d(k))$.

$$x_{k+1} = A_k x_k + B_k u_k \tag{4}$$

where:

- $x_k = \delta \mathbf{q}_k = \mathbf{q}_k - \mathbf{q}_d(k)$ is the deviation of the state from the desired trajectory at time k.
- $u_k = \delta \mathbf{u}_k = \mathbf{u}_k - \mathbf{u}_d(k)$ is the deviation of the control input from the nominal control input at time k.
- $A_k = \left.\frac{\partial f}{\partial x}\right|_{x=x_d(k),\, u=u_d(k)}$ is the Jacobian of the system dynamics with respect to the state, evaluated at the desired trajectory.
- $B_k = \left.\frac{\partial f}{\partial u}\right|_{x=x_d(k),\, u=u_d(k)}$ is the Jacobian of the system dynamics with respect to the control input, evaluated at the desired trajectory.

System Dynamics Linearization

For the manipulator under gravity cancellation, the nonlinear dynamics are:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{u} - \mathbf{g}(\mathbf{q}) \tag{5}$$

After gravity cancellation, the dynamics simplify to:

$$\mathbf{M}(\mathbf{q})\ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\dot{\mathbf{q}} = \mathbf{u} \tag{6}$$

Linearizing around the desired trajectory, we obtain:

$$\delta \dot{x}_k = A_k \delta x_k + B_k \delta u_k \tag{7}$$

where $\delta x_k = \begin{bmatrix} \delta \mathbf{q}_k \\ \delta \dot{\mathbf{q}}_k \end{bmatrix}$.

Matrices A_k and B_k Expressions

The expressions for A_k and B_k are:

$$A_k = \begin{bmatrix} 0 & I \\ -\mathbf{M}^{-1}(\mathbf{q}_d(k)) \left(\frac{\partial \mathbf{C}(\mathbf{q}_d(k), \dot{\mathbf{q}}_d(k))\dot{\mathbf{q}}_d(k)}{\partial \mathbf{q}} \right) & -\mathbf{M}^{-1}(\mathbf{q}_d(k))\mathbf{C}(\mathbf{q}_d(k), \dot{\mathbf{q}}_d(k)) \end{bmatrix} \tag{8}$$

$$B_k = \begin{bmatrix} 0 \\ \mathbf{M}^{-1}(\mathbf{q}_d(k)) \end{bmatrix} \tag{9}$$

Explanation of Terms

- $\mathbf{M}(\mathbf{q})$ is the mass/inertia matrix of the manipulator.
- $\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})$ is the Coriolis and centrifugal matrix.
- $\mathbf{g}(\mathbf{q})$ is the gravitational torque vector.
- I is the identity matrix.
- δx_k and δu_k represent small deviations from the desired state and control input.