EE571 Final Exam Bonus Problem
Steering a Vehicle Along a Reference Track
Discrete LQR and Pole Placement on a Linearized Error Model

1 Goal and Objective

1.1 Goal

Design and compare **two state-feedback regulators** for a vehicle trajectory-tracking task, using a **linearized error-state model** while simulating the **nonlinear bicycle-model plant**.

1.2 Objective

Track a **time-parameterized reference path** using a feedback controller designed on a **linearized tracking-error model**, while the closed-loop system is evaluated on the **nonlinear bicycle-model plant**.

2 Nonlinear Plant Model Used for Simulation (Bicycle Dynamics)

2.1 Plant State and Inputs

The nonlinear plant state is:

$$\boldsymbol{x} = \begin{bmatrix} X & Y & \psi & v_x & v_y & r \end{bmatrix}^T \tag{1}$$

where:

- X, Y: global position
- ψ: yaw angle (heading)
- v_x, v_y: body-frame longitudinal and lateral velocities
- r: yaw rate

The plant input is:

$$\boldsymbol{u} = \begin{bmatrix} \delta \\ a_x \end{bmatrix} \tag{2}$$

where:

- δ: steering angle input
- a_x: longitudinal acceleration command (named `throttle` in the MATLAB file)

2.2 Kinematics

The handout and code use:

$$\dot{X} = v_x \cos\psi - v_y \sin\psi \tag{3}$$

$$\dot{Y} = v_x \sin\psi + v_y \cos\psi \tag{4}$$

$$\dot{\psi} = r \tag{5}$$

2.3 Tire Slip and Lateral Forces (Small-Angle Approximation, $v_x > 0$)

Slip angles:

$$\alpha_f \approx \delta - \frac{v_y + l_f r}{v_x} \tag{6}$$

$$\alpha_r \approx -\frac{v_y - l_r r}{v_x} \tag{7}$$

Linear tire model:

$$F_{yf} = C_f \alpha_f \tag{8}$$

$$F_{yr} = C_r \alpha_r \tag{9}$$

2.4 Dynamics

$$\dot{v}_x = a_x + r v_y \tag{10}$$

$$\dot{v}_y = \frac{F_{yf} + F_{yr}}{m} - r v_x \tag{11}$$

$$\dot{r} = \frac{l_f F_{yf} - l_r F_{yr}}{I_z} \tag{12}$$

This completes the nonlinear plant $\dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u})$. In the code, the plant is integrated with RK4 using a smaller internal step `dt_int` $= T_s/10$, and the input is held constant over each sampling interval (ZOH).

2.5 Parameters Used in `vehicle_dlqr.m`

Near the top of the script:

- $m = 1500\,\mathrm{kg}$
- $I_z = 2500\,\mathrm{kg}\cdot\mathrm{m}^2$
- $l_f = 1.2\,\mathrm{m}$, $l_r = 1.6\,\mathrm{m}$
- $C_f = 80000\,\mathrm{N/rad}$, $C_r = 80000\,\mathrm{N/rad}$

where C_f, C_r are cornering stiffnesses, and l_f, l_r are distances from CG to front and rear axles.

3 Reference Trajectory and Tracking-Error Definitions

3.1 Reference Signals

The reference is time-parameterized:

$$\big(X_{\mathrm{ref}}(t), Y_{\mathrm{ref}}(t), \psi_{\mathrm{ref}}(t), v_{\mathrm{ref}}(t), \kappa_{\mathrm{ref}}(t)\big) \tag{13}$$

Reference yaw-rate relation:

$$\dot{\psi}_{\mathrm{ref}}(t) = v_{\mathrm{ref}}(t)\,\kappa_{\mathrm{ref}}(t) \tag{14}$$

3.2 How the MATLAB File Generates the Reference

In `vehicle_dlqr.m`:

- Sample time: $T_s = 0.02\,\mathrm{s}$ (50 Hz)
- Duration: $T_{\mathrm{end}} = 25\,\mathrm{s}$
- Nominal speed used for linearization: $V_{x0} = 15\,\mathrm{m/s}$

Curvature and speed profiles:

$$\kappa_{\mathrm{ref}}(t) = 0.01\sin(0.35t) + 0.005\sin(0.10t) \quad [1/\mathrm{m}] \tag{15}$$
$$v_{\mathrm{ref}}(t) = V_{x0} + 1.0\sin(0.15t) \quad [\mathrm{m/s}] \tag{16}$$

Reference acceleration is approximated by finite difference:

$$a_{\mathrm{ref}}(t) \approx \frac{v_{\mathrm{ref}}(t+T_s) - v_{\mathrm{ref}}(t)}{T_s} \tag{17}$$

The function `integrate_reference(t, v_ref, kappa_ref)` computes:

$$\dot{\psi}_{\mathrm{ref}} = v_{\mathrm{ref}}\kappa_{\mathrm{ref}} \tag{18}$$
$$\dot{X}_{\mathrm{ref}} = v_{\mathrm{ref}}\cos\psi_{\mathrm{ref}} \tag{19}$$
$$\dot{Y}_{\mathrm{ref}} = v_{\mathrm{ref}}\sin\psi_{\mathrm{ref}} \tag{20}$$

using simple forward-Euler integration.

3.3 Tracking Errors Used for Regulation

Define position error:

$$\boldsymbol{e}_p = \begin{bmatrix} X - X_{\mathrm{ref}} \\ Y - Y_{\mathrm{ref}} \end{bmatrix} \tag{21}$$

Define the reference normal vector:

$$\hat{\boldsymbol{n}} = \begin{bmatrix} -\sin\psi_{\mathrm{ref}} \\ \cos\psi_{\mathrm{ref}} \end{bmatrix} \tag{22}$$

Cross-track error:

$$e_y = \hat{\boldsymbol{n}}^T \boldsymbol{e}_p \tag{23}$$

Heading error (wrapped to $[-\pi, \pi]$):

$$e_\psi = \mathrm{wrapToPi}(\psi - \psi_{\mathrm{ref}}) \tag{24}$$

Speed error (the code uses longitudinal speed):

$$e_v = v_x - v_{\mathrm{ref}} \tag{25}$$

4 Linearized Tracking-Error Model Used for Controller Design

The handout emphasizes that the plant is nonlinear, so control is designed on a **linear error model** obtained by linearizing around a nominal operating point.

4.1 Nominal Operating Point

The linear model assumes:

$$v_x \approx V_{x0} > 0, \quad v_y \approx 0, \quad r \approx 0, \quad \delta \approx 0 \tag{26}$$

In the MATLAB file: $V_{x0} = 15$.

4.2 Lateral-Yaw Linear Approximation

The standard linear model uses:

$$\begin{bmatrix} \dot{v}_y \\ \dot{r} \end{bmatrix} = A_{\text{lat}} \begin{bmatrix} v_y \\ r \end{bmatrix} + B_\delta \, \delta \tag{27}$$

with:

$$A_{\text{lat}} = \begin{bmatrix} -\frac{C_f + C_r}{m V_{x0}} & -\left(V_{x0} + \frac{l_f C_f - l_r C_r}{m V_{x0}}\right) \\ -\frac{l_f C_f - l_r C_r}{I_z V_{x0}} & -\frac{l_f^2 C_f + l_r^2 C_r}{I_z V_{x0}} \end{bmatrix} \tag{28}$$

$$B_\delta = \begin{bmatrix} \frac{C_f}{m} \\ \frac{l_f C_f}{I_z} \end{bmatrix} \tag{29}$$

This matches how the script defines `A11, A12, A21, A22` and the steering-input terms `Bvydelta, Brdelta`.

4.3 Error-State Vector for Control Design

The handout and code define a 5-state error vector:

$$\boldsymbol{x}_e = \begin{bmatrix} v_y \\ r \\ e_y \\ e_\psi \\ e_v \end{bmatrix} \tag{30}$$

The regulation input is:

$$\boldsymbol{u}_{\text{reg}} = \begin{bmatrix} \delta_{\text{reg}} \\ a_{x,\text{reg}} \end{bmatrix} \tag{31}$$

4.4 Error Propagation Equations Used in the Script

The original handout indicates the error derivatives include reference terms:

$$\dot{e}_y \approx v_y + V_{x0} e_\psi \tag{32}$$

$$\dot{e}_\psi \approx r - V_{x0} \kappa_{\text{ref}}(t) \tag{33}$$

$$\dot{e}_v \approx a_x - a_{\text{ref}}(t) \tag{34}$$

However, the MATLAB script handles the reference curvature and acceleration terms via **feedforward**, so the **regulation model** used to build (A_c, B_c) uses the simpler forms:

$$\dot{e}_y \approx v_y + V_{x0} e_\psi \tag{35}$$

$$\dot{e}_\psi \approx r \tag{36}$$

$$\dot{e}_v \approx a_x \tag{37}$$

This separation allows the feedforward to handle the reference tracking while the regulator focuses on error correction.

4.5 Continuous-Time State Space Form in `vehicle_dlqr.m`

The script constructs:

$$\dot{\boldsymbol{x}}_e = A_c \boldsymbol{x}_e + B_c \boldsymbol{u}_{\text{reg}} \tag{38}$$

with $A_c \in \mathbb{R}^{5\times5}$, $B_c \in \mathbb{R}^{5\times2}$. The matrix structure in the code is:

- Rows 1 to 2: lateral-yaw dynamics for $[v_y, r]$ driven by δ
- Row 3: $\dot{e}_y = v_y + V_{x0} e_\psi$
- Row 4: $\dot{e}_\psi = r$
- Row 5: $\dot{e}_v = a_x$

5 Control Structure: Feedforward Plus Feedback Regulation

The required structure is:

$$\boldsymbol{u} = \boldsymbol{u}_{\text{ff}}(t) + \boldsymbol{u}_{\text{reg}}(\boldsymbol{x}_e) \tag{39}$$

5.1 Feedforward Terms Provided in the MATLAB Script

Inside the simulation loop, the script defines starter feedforward:

- Steering feedforward:

$$\delta_{\text{ff}} \approx (l_f + l_r)\kappa_{\text{ref}}(t) \tag{40}$$

  This is the geometric relation $\delta \approx L\kappa$ for small angles, where $L = l_f + l_r$ is the wheelbase.

- Longitudinal acceleration feedforward:

$$a_{x,\text{ff}} = a_{\text{ref}}(t) = \dot{v}_{\text{ref}}(t) \tag{41}$$

The feedforward terms make the vehicle follow the reference in an ideal, error-free case, while the regulator stabilizes and drives the error states toward zero.

5.2 What You Must Compute

You must compute `steering_reg` and `ax_reg` using one of the two regulators, then combine:

- `steering_input = steering_feed_forward + steering_reg`
- `throttle = throttle_feed_forward + ax_reg`

The script already applies saturations after this combination:

- steering limited to ±25 degrees
- acceleration limited to $[-6, 3]\,\text{m/s}^2$

6 Discretization and Simulation Timing

6.1 Discretize the Continuous Error Model (Required)

You must discretize (A_c, B_c) using **zero-order hold** at T_s:

$$\boldsymbol{x}_{e,k+1} = A_d \boldsymbol{x}_{e,k} + B_d \boldsymbol{u}_{\text{reg},k} \tag{42}$$

The script includes an exact ZOH helper: `c2d_zoh_exact(A, B, Ts)`.
It uses the augmented matrix exponential identity:

$$\exp\left(\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} T_s\right) = \begin{bmatrix} A_d & B_d \\ 0 & I \end{bmatrix} \tag{43}$$

You can use this function directly and avoid relying on MATLAB's `c2d` function.

6.2 Simulation of the Nonlinear Plant

The plant is integrated with RK4 using an internal step:

- `dt_int` $= T_s/10$

Inputs are held constant within each T_s interval, which matches the ZOH assumption used for discretizing the controller model.

7 Regulator 1: Discrete-Time Infinite-Horizon LQR (DLQR)

After discretization, DLQR solves the discrete algebraic Riccati equation for an infinite-horizon quadratic cost:

$$J = \sum_{k=0}^{\infty} \left(\boldsymbol{x}_{e,k}^T Q \boldsymbol{x}_{e,k} + \boldsymbol{u}_{\text{reg},k}^T R \boldsymbol{u}_{\text{reg},k}\right) \tag{44}$$

and yields:

$$\boldsymbol{u}_{\text{reg},k} = -K_{\text{LQR}} \boldsymbol{x}_{e,k} \tag{45}$$

7.1 Implementation Steps

1. Compute `[Ad, Bd] = c2d_zoh_exact(Ac, Bc, Ts)`.
2. Choose $Q \succeq 0$ (5×5) and $R \succ 0$ (2×2), symmetric and positive semi-definite / definite respectively.
3. In MATLAB, compute `[K_LQR, , ] = dlqr(Ad, Bd, Q, R)`.
4. In the simulation loop form:

$$\boldsymbol{x}_e = [v_y,\ r,\ e_y,\ e_\psi,\ e_v]^T \tag{46}$$

then compute `u_reg = -K_LQR * x_e`.

8 Regulator 2: Discrete-Time Pole Placement (Real Poles Only)

Pole placement chooses a gain K_{PP} so that the closed-loop discrete matrix:

$$A_{\text{cl}} = A_d - B_d K_{\text{PP}} \tag{47}$$

has desired eigenvalues (poles).

8.1 Constraints from the Problem Statement

- Choose **real** poles only (no complex pole locations).
- For discrete time, stable poles must lie inside the unit circle.

8.2 Practical Notes

- If (A_d, B_d) is not fully controllable, you can only assign poles of the controllable modes. The uncontrollable modes remain fixed.
- Because the input $\boldsymbol{u}$ is 2D, you will select 5 poles total only if the discrete system is controllable. Otherwise, you select poles for the controllable subspace.
- In MATLAB you typically use: `K_PP = place(Ad, Bd, desired_poles)` (or `acker` for SISO systems).

9 Initial-Condition Scaling Experiments (Default, $\times 2$, $\times 3$)

The MATLAB file's default initial condition (referenced as "line 87" in the problem statement) offsets the plant from the reference:

$$[X(0), Y(0), \psi(0), v_x(0)] = [X_{\text{ref}}(0) - 2.0\,\text{m},\ Y_{\text{ref}}(0) + 1.0\,\text{m},\ \psi_{\text{ref}}(0) + 8^\circ,\ V_{x0} - 5\,\text{m/s}] \quad (48)$$

and:

$$v_y(0) = 0, \quad r(0) = 0 \quad (49)$$

To form the $\times 2$ and $\times 3$ cases, scale these offsets from the reference by factors 2 and 3, while keeping the reference itself unchanged.

9.1 Implementation Pattern

1. Build the baseline offset $\Delta \boldsymbol{x}_0$ relative to $(X_{\text{ref}}(0), Y_{\text{ref}}(0), \psi_{\text{ref}}(0), V_{x0})$.
2. For scale $s \in \{1, 2, 3\}$, set $\boldsymbol{x}_0 = \boldsymbol{x}_{\text{ref},0} + s\Delta \boldsymbol{x}_0$.
3. Run the same simulation for each s and for each regulator.

10 Additional Notes

10.1 Integration Method

The nonlinear plant is simulated with RK4 using an internal integration step `dt_int` $= T_s/10$. The controller updates every T_s and holds inputs constant within each sample interval, which matches the ZOH assumption used for discretizing the controller model.

10.2 Saturations

The script applies saturations after combining feedforward and regulation inputs:

- Steering: $\delta \in [-25^\circ, +25^\circ]$
- Acceleration: $a_x \in [-6, +3]\,\text{m/s}^2$

These limits are applied each control update.

10.3 Reference Integration

The function `integrate_reference(t, v_ref, kappa_ref)` computes the reference pose using simple forward-Euler integration:

- ψ_{ref} via $\psi_{\text{ref},k+1} = \psi_{\text{ref},k} + T_s\, v_{\text{ref},k}\kappa_{\text{ref},k}$
- X_{ref} and Y_{ref} via Euler integration using $v_{\text{ref}}\cos\psi_{\text{ref}}$ and $v_{\text{ref}}\sin\psi_{\text{ref}}$