Digital version

ENGS261 Control Systems - Homework 3

Mher Saribekyan A09210183

October 31, 2026

Part 1: Identify and validate the plant

Training data was collected by accelerating and decelerating the car in small steps. The following two functions were implemented, data was fitted on a second order model.

```
def predict_second_order(
    trial: TrialData,
    K_dc: float,
    tau1: float,
    tau2: float,
) -> np.ndarray:
    """Predict delta_v for P(s)=K_dc/((tau1*s+1)(tau2*s+1))."""

    x_a = np.zeros_like(trial.delta_v)
    delta_v = np.zeros_like(trial.delta_v)

    n = len(trial.time_s)
    for i in range(1, n):
        delta_t = trial.time_s[i] - trial.time_s[i - 1]
        x_a[i] = x_a[i - 1] + delta_t * (trial.delta_q[i] - x_a[i - 1]) / tau1
        delta_v[i] = delta_v[i - 1] + delta_t * ((K_dc * x_a[i - 1]) - delta_v[i - 1]) /
                                                                    tau2

    return delta_v
```

```
def fit_second_order(training: Sequence[TrialData]) -> tuple[float, float, float]:
    """Estimate and return positive (K_dc, tau1, tau2) using training data only."""
    def calculate_error(
        trials: Sequence[TrialData],
        K_dc: float,
        tau1: float,
        tau2: float,
    ):
        measured_delta_v = np.concatenate([trial.delta_v for trial in trials])
        predicted_delta_v = np.concatenate([predict_second_order(trial, K_dc, tau1, tau2)
                                                    for trial in trials])

        return nrmse(measured_delta_v, predicted_delta_v)

    def is_valid(K_dc, tau1, tau2):
        return (K_dc > 0) and (tau1 > 0) and (tau2 > 0) and (tau1 <= tau2)
```

```
    # K_dc tau1 tau2
    params = [20, 0.1, 2] # initial guess
    steps = [0.1, 0.1, 0.1] # initial step sizes

    least_error: float = calculate_error(training, params[0], params[1], params[2])

    iterations = 2500

    for i in range(iterations):
        improved = False
        for j in range(3):
            for direction in (-1, 1):
                test = [params[0], params[1], params[2]]
                test[j] += direction * steps[j]

                # Check positivity and tau1 <= tau2
                if not is_valid(test[0], test[1], test[2]):
                    continue

                error = calculate_error(training, test[0], test[1], test[2])

                if error < least_error:
                    params = test
                    least_error = error
                    improved = True
        if not improved:
            steps = [step / 2 for step in steps]

    return (params[0], params[1], params[2])
```

The first function predicts Δv values depending on t and Δq, given K_{dc}, τ_1 and τ_2. The following two differential equations are modified and approximated for discrete time data:

$$\tau_a \dot{x}_a + x_a = \Delta q \implies x_{k+1} = x_k + \Delta t \frac{\Delta q_{k+1} - x_k}{\tau_a}$$

$$\tau_v \Delta \dot{v} + \Delta v = K_{\text{dc}} x_a \implies \Delta v_{k+1} = \Delta v_k + \Delta t \frac{K_{\text{dc}} x_k - \Delta v_k}{\tau_v}$$

The second function uses coordinate descent algorithm to search for the best values of K_{dc}, τ_1 and τ_2 to give the smalles error, using the following error function:

$$\text{NRMSE} = \frac{\sqrt{N^{-1} \sum_{k=1}^{N} (\Delta v_k - \Delta \hat{v}_k)^2}}{\max(\Delta v_k) - \min(\Delta v_k)}.$$

The training data consisted of accelerating and decelerating trials. The model was estimated in two ways: first trained on the accelerating set and validated on the decelerating set, then the other way around. With the training data used, the best NMRSE error occured at $K_{\text{dc}} \approx 58.32$, $\tau_1 \approx 0.4219$ and $\tau_2 \approx 3.1124$, giving validation error of NRMSE ≈ 0.02786. Therefore, the car (plant) can be approximated with:

$$P(s) = \frac{\Delta V(s)}{\Delta Q(s)} \approx \frac{58.32}{(0.4219s + 1)(3.1124s + 1)} \approx \frac{44.41}{(s + 0.3213)(s + 2.3704)}$$

Part 2: Proportional root-locus analysis

For the P controller, we select $C(s) = K_p$:

$$C(s)P(s) = \frac{44.41K_p}{(s + 0.3213)(s + 2.3704)}$$

The poles of the open loop system are $s_1 \approx -0.3213$ and $s_2 \approx -2.3704$. We have two poles and no zeros. The asymptote angles and the centroid are:

$$\theta_1 = \frac{180^\circ}{2} = 90^\circ \text{ and } \theta_2 = \frac{3 \cdot 180^\circ}{2} = 270^\circ \text{ and } \sigma = \frac{-0.3213 - 2.3704}{2} = -1.3459$$

As we increase K_p, the poles s_1 and s_2 approach $s = -1.3459$, when $K_p \approx 0.02363$. After that, one of the poles goes to the $+j\omega$ direciton towards infinity, while the other goes to the $-j\omega$ direction towards negative infinity.
The closed loop transfer function is:

$$\frac{\Delta V(s)}{\Delta V_r(s)} = \frac{C(s)P(s)}{1 + C(s)P(s)} = \frac{44.41K_p}{(s + 0.3213)(s + 2.3704) + 44.41K_p}$$

Therefore the characteristic equation is:

$$P_{\text{cl}}(s) = (s + 0.3213)(s + 2.3704) + 44.41K_p = s^2 + 2.6917s + 0.7616 + 44.41K_p$$

The design requirements are %OS $\leq 10\%$, $T_p \leq 1.0$s, $T_s < 0.6$s and absolute steady-state tracking error ≤ 0.20m/s.

$$\zeta = \frac{-\ln(\text{OS})}{\sqrt{\pi^2 + \ln^2(\text{OS})}} \geq \frac{-\ln(0.1)}{\sqrt{\pi^2 + \ln^2(0.1)}} \approx 0.5912 \implies \theta \cos^{-1}(\zeta) \leq \cos^{-1}(0.5912) \approx 53.76^\circ$$

The intersection of the overshoot line and the root locus is at the point $s = -1.3459 \pm j1.3459 \tan(53.76^\circ) \approx -1.3459 \pm j1.8362$. To find the K_p value, we can use the magnitude condition:

$$K_p = \frac{\sqrt{(-1.3459 + 0.3213)^2 + (1.8362)^2} \cdot \sqrt{(-1.3459 + 2.3704)^2 + (1.8362)^2}}{44.41} \approx 0.09956$$

Therefore, for $K_p = 0.09956$ we should get 10% overshoot.