Mini-Project 2: Predictive Control Design and Robustness Analysis

1 Introduction

This project requires the design and comparison of several predictive control strategies. Using the previously given continuous-time nonlinear plant model (which should be MIMO and at least input-constrained) and its discretized linear approximation, you will implement:

- **Dynamic Matrix Control (DMC)** — one of the first-generation MPC algorithms.
- **Generalized Predictive Control (GPC)** — an influential predictive control method introduced in 1987.
- **Receding-Horizon Linear MPC** — standard optimal control minimizing a quadratic cost over a finite horizon.
- **Linear Quadratic Regulator (LQR)** — the infinite-horizon optimal controller for linear systems.

Each controller will be first designed using the discretized linear model, then applied to the original nonlinear plant for tracking a user-defined reference trajectory. The controllers must enforce actuator (input) constraints, as predictive controllers can naturally handle bounds via quadratic programs. The performance and robustness of all four controllers will be evaluated and compared under identical conditions.

2 Model and Trajectory Design

Students should use the provided plant model and discretize it at a suitable sampling time (e.g. one-tenth of the dominant time constant). The prediction model (step-response or state-space) obtained from this discretized model will be the basis for all controller designs.

Next, define a *reference trajectory* $r(t)$ that the plant output should follow. It must be at least a quadratic function of time (e.g. $r(t) = at^2 + bt + c$). This nontrivial trajectory will test the controllers' tracking performance and is known to challenge control systems more than simple steps or ramps.

3 Controller Designs

3.1 Discrete-time DMC

1. **Model:** Extract the step (or impulse) response of the discretized linear model. Form the free response and the dynamic matrix mapping future input increments to future outputs.

2. **Tuning:** Choose a prediction horizon N_p, control horizon N_u, and a weight (penalty) on input moves (often denoted λ). The DMC cost is typically a least-squares term on output error plus a penalty on input changes.

3. **Optimization:** At each step solve for the input increments that minimize the cost. For unconstrained DMC this has a closed-form solution, but with constraints it becomes a quadratic program (QDMC).

4. **Implementation:** Implement the DMC law in code (Matlab or Python). Iterate: predict outputs, solve for optimal input sequence, apply the first increment, then update.

3.2 Discrete-time GPC

1. **Model:** Use the discretized linear model in a predictor form (e.g. ARX), and derive the prediction model by utilizing the Diophantine equation.

2. **Tuning:** Define prediction (N_p) and control (N_u) horizons and control weight (λ) for the GPC cost.

3. **Controller:** Derive the GPC control law: minimize the sum of future squared output deviations plus weighted input changes. This yields a predictive feedback formula.

4. **Implementation:** Code the GPC algorithm (e.g. solve the normal equations or use a QP to include constraints). GPC is essentially an MPC variant introduced as an adaptive scheme, here applied with the fixed model.

Remark: GPC was one of the first adaptive MPC algorithms; in this project it will be implemented with the given linear model.

3.3 Receding-Horizon Linear MPC

1. **Model:** Use the same discrete-time linear model (in state-space format) and derive the preditcition model using the recursion the state-space model.

2. **Cost & Constraints:** Formulate a finite-horizon optimization: minimize a quadratic cost (tracking error + control effort) subject to linear dynamics and input (and optionally output) constraints. In the simplest (linear) MPC, both plant and constraints are linear and the cost is quadratic.

3. **Horizon & Weights:** Select prediction horizon N_p, control horizon N_u, and weighting matrices Q, R for state/output error and input moves.

4. **QP Solver:** At each sampling instant, solve the MPC quadratic program: predict future outputs, enforce constraints, and minimize the cost. Then apply only the first control move and repeat (receding horizon).

5. **Implementation:** Implement in Matlab or Python (using, e.g., quadprog or cvxopt). Update the state estimate each step using plant feedback.

3.4 Linear Quadratic Regulator (LQR) Controller

1. **Design:** Using the discretized linear state-space model, solve the infinite-horizon LQR problem.

2. **Tracking:** Use reference-state error formulation (and the tracking error instead of the system states) so that the LQR can follow the time-varying reference.

3. **Implementation:** Apply the state-feedback control $u = -K(x - x_{ref})$ (where K is the LQR gain). Note that LQR does not inherently handle constraints, so ensure inputs respect bounds (e.g., using saturation).

4 Simulation and Performance Evaluation

- **Nominal Tracking:** Simulate the nonlinear plant for each controller tracking the reference trajectory. Plot output vs. time and input vs. time for all methods. Verify stability and good tracking for each design.

- **Performance Metrics:** Quantify tracking performance using standard indices. For example, compute the Integral of Squared Error (ISE), Integral of Absolute Error (IAE) for each controller. Also record maximum overshoot, settling time (e.g. within 2% band), and total control effort (e.g. $\sum |u|$ or $\sum u^2$). These metrics allow fair comparison of control quality.

- **Summary Table:** Present the results in a table (controllers as columns, metrics as rows). Compare which controller has lower tracking error, overshoot, or control usage.

5 Robustness Analysis

Test each controller under perturbations:

1. **Measurement Noise:** Add zero-mean noise to the plant output before feedback. Evaluate how tracking degrades (recompute error metrics).

2. **Model Uncertainty:** Introduce mismatch between plant and model (e.g. $\pm$10–20% parametric errors or unmodeled dynamics). Re-run simulations and compare performance; note that MPC has some built-in robustness due to re-optimizing at each step.

3. **External Disturbance:** Inject a disturbance into the plant (for instance, add a step disturbance to the plant input or output). Observe which controller rejects the disturbance best (e.g. fastest recovery).

4. **Actuator Fault:** Simulate an actuator fault, such as a multiplicative gain or an additive bias applied to inputs. Analyze how each controller handles it, especially noting that predictive controllers enforce constraints explicitly.

For each scenario, plot the outputs and compute the same performance metrics. Compare the degraded performance to the nominal case and discuss which controllers are more robust under each condition.

6 Deliverables

Students must submit:

- A **7-page report** (PDF) containing: the brief model description, controller derivations, tuning choices, and all results. Include plots of outputs and inputs for each scenario, tables of performance metrics, and a discussion comparing the controllers. Explain why certain controllers perform better or worse under different conditions.

- A **7-minute video** demonstrating the project. The video should show the *code execution* (screen capture of simulations) and explain the results. It must include running the simulations for each controller, displaying key plots, and narrating the theory behind the code (e.g. how each controller algorithm works and why it was tuned that way).

- The complete **code** (Matlab or Python) used for the simulations. The code should be well-commented and organized, with separate functions/scripts as appropriate.

Students should make sure that the video shows the code running in real time (generating the results) and that they clearly explain the control theory behind their implementations. Use the performance metrics (ISE, overshoot, settling time, etc.) to substantiate claims in the report. Both Matlab and Python are acceptable. Note that enforcing input constraints in the predictive controllers is mandatory. Use identical initial conditions and reference trajectory in all comparisons for fairness.