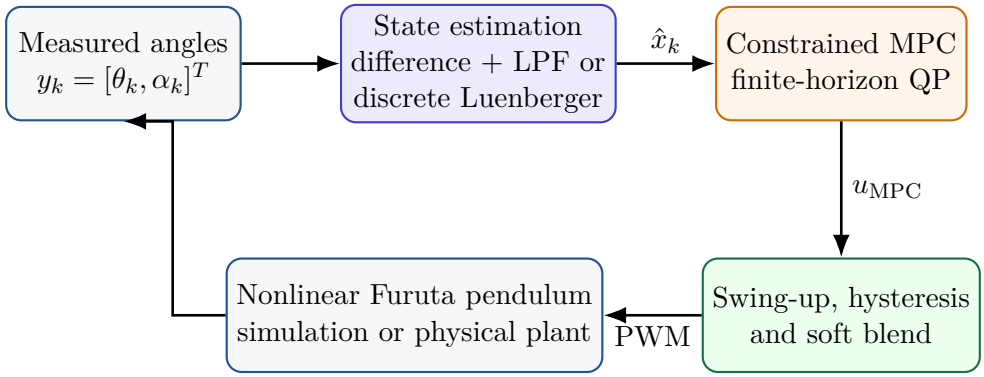
Model Predictive Control of a Rotary Inverted Pendulum

Class 10 Laboratory Technical Manual

AI Enabled Control Engineering



Class-10 design objective. Keep the nonlinear Furuta plant, 200 Hz sampling, phase-pumping swing-up, switching thresholds, soft blend, visual interface and CSV/PNG logging style used in Class 9. Replace the upright LQR law by a constrained linear MPC controller. Compare two ways of constructing the four-state input required by MPC: filtered numerical differentiation and a discrete Luenberger observer.

Course staff: Professor Xun Huang; TAs: Zhixiang Ju and Haozhe Wang
Version: July 2026

Contents

1 Learning Objectives and Class-10 Architecture

1.1 What changes from Class 9

Class 9 used a fixed linear feedback law

$$u_k = -Kx_k.$$

Once the LQR gain was calculated, the online computation was only one matrix–vector multiplication. Class 10 retains the same hybrid swing-up and local-stabilisation structure, but the upright command is now obtained from an online finite-horizon optimisation problem:

$$u_k = \left[\arg\min_U J_k(U)\right]_0.$$

The notation $[\cdot]_0$ means that only the first element of the optimised future input sequence is applied.

The state order remains

$$\boldsymbol{x}_k = \begin{bmatrix}\theta_k & \dot{\theta}_k & \alpha_k & \dot{\alpha}_k\end{bmatrix}^T,$$

where $\alpha = 0$ is the upright pendulum position.

1.2 The two main topics

By the end of the laboratory, students should be able to:

1. explain why angle measurements alone are insufficient for full-state MPC;
2. derive and implement a discrete Luenberger observer;
3. distinguish observer prediction, innovation and correction;
4. derive the stacked finite-horizon prediction model;
5. convert the quadratic cost into a condensed box-constrained QP;
6. explain receding-horizon control and warm starting;
7. describe how the STM32 constructs and solves the MPC problem at 200 Hz;
8. tune prediction horizon, state weights, input weight and solver iterations systematically rather than by changing all parameters at once.

IMPORTANT

MPC in this class is still a **local upright stabiliser**. The prediction model is linearised around $\alpha = 0$. It does not replace the nonlinear swing-up controller. The firmware switches to MPC only inside the configured upright neighbourhood.

2 Why State Estimation Is Required

2.1 Measured and unmeasured states

The physical apparatus directly measures the rotary-arm angle θ and the pendulum angle α. The output vector is

$$\boldsymbol{y}_k = C\boldsymbol{x}_k, \qquad C = \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\end{bmatrix}.$$

The two angular velocities are not measured directly. However, both LQR and MPC require a four-state vector because velocity information indicates the direction and rate at which the system is moving.

The upright linearised model is observable from these two angle outputs. Thus, in principle, the unmeasured velocities can be reconstructed from the input, the angle history and the model.

2.2 Baseline method: numerical difference with a low-pass filter

The simplest velocity estimate is

$$\dot{\theta}_{\mathrm{raw},k} = \frac{\theta_k - \theta_{k-1}}{T_s}, \qquad \dot{\alpha}_{\mathrm{raw},k} = \frac{\mathrm{wrap}(\alpha_k - \alpha_{k-1})}{T_s}.$$

The angle difference for α must be wrapped to $(-\pi, \pi]$ so that a crossing of the angular branch cut is not interpreted as a very large velocity. The Class-10 code filters the raw estimates using

$$\hat{\dot{\theta}}_k = \hat{\dot{\theta}}_{k-1} + \beta \left(\dot{\theta}_{\mathrm{raw},k} - \hat{\dot{\theta}}_{k-1} \right),$$

$$\hat{\dot{\alpha}}_k = \hat{\dot{\alpha}}_{k-1} + \beta \left(\dot{\alpha}_{\mathrm{raw},k} - \hat{\dot{\alpha}}_{k-1} \right),$$

where $0 < \beta \leq 1$ is the panel parameter **Velocity LPF beta**.

- Small β: stronger smoothing, lower noise, more delay.
- Large β: faster response, but noisier velocity and PWM.
- $\beta = 1$: no smoothing; the estimate becomes the raw difference.

Numerical differentiation amplifies high-frequency measurement noise because the angle difference is divided by the small sampling period $T_s = 0.005$ s. The LPF reduces noise but introduces phase lag. This noise–delay trade-off motivates use of a model-based observer.

3 Discrete Luenberger Observer

3.1 Continuous-time idea

For the continuous linear model

$$\dot{\boldsymbol{x}} = A\boldsymbol{x} + Bu, \qquad \boldsymbol{y} = C\boldsymbol{x},$$

a Luenberger observer has the form

$$\dot{\hat{\boldsymbol{x}}} = A\hat{\boldsymbol{x}} + Bu + L\left(\boldsymbol{y} - C\hat{\boldsymbol{x}}\right).$$

The two components are:

1. **model prediction**: $A\hat{x} + Bu$;
2. **measurement correction**: $L(y - C\hat{x})$.

The quantity

$$\boldsymbol{\nu} = \boldsymbol{y} - C\hat{\boldsymbol{x}}$$

is called the innovation or output residual.

Define the estimation error

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

Subtracting the observer equation from the plant equation gives

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

Hence the continuous observer is asymptotically stable when every eigenvalue of $A - LC$ lies in the open left half-plane.

3.2 Zero-order-hold discrete model

The controller runs at

$$f_s = 200 \text{ Hz}, \qquad T_s = 0.005 \text{ s}.$$

With a zero-order-held input, the discrete model is

$$\boldsymbol{x}_{k+1} = A_d \boldsymbol{x}_k + B_d u_k,$$

where

$$A_d = e^{AT_s}, \qquad B_d = \int_0^{T_s} e^{A\tau} B \, d\tau.$$

The model used by the Class-10 MPC implementation is

$$A_d = \begin{bmatrix} 1.00000000 & 0.00486127 & -0.00061123 & 0.00000487 \\ 0 & 0.94505029 & -0.24196123 & 0.00172377 \\ 0 & 0.00020708 & 1.00211617 & 0.00498311 \\ 0 & 0.08197316 & 0.84220198 & 0.99398867 \end{bmatrix},$$

$$B_d = \begin{bmatrix} 0.00001002 \\ 0.00397029 \\ -0.00001496 \\ -0.00592283 \end{bmatrix}.$$

3.3 Discrete observer equation and error dynamics

A discrete Luenberger observer can be written as

$$\boxed{\hat{\boldsymbol{x}}_{k+1} = A_d \hat{\boldsymbol{x}}_k + B_d u_k + L_d \left(\boldsymbol{y}_k - C \hat{\boldsymbol{x}}_k \right)}.$$

Equivalently,

$$\hat{\boldsymbol{x}}_{k+1} = (A_d - L_d C)\hat{\boldsymbol{x}}_k + B_d u_k + L_d \boldsymbol{y}_k.$$

The estimation-error dynamics are

$$\boldsymbol{e}_{k+1} = (A_d - L_d C)\boldsymbol{e}_k.$$

The discrete stability condition is therefore

$$\boxed{\rho(A_d - L_d C) < 1},$$

where $\rho(\cdot)$ is the spectral radius. All observer-error poles must lie inside the unit circle.

IMPORTANT

Do not apply the continuous-time rule "negative real parts" directly to a discrete observer. For a discrete system, the eigenvalue magnitudes must be less than one.

3.4 Observer speed versus noise sensitivity

Placing the observer poles closer to the origin usually makes the estimation error converge more rapidly. However, aggressive correction gains also inject more sensor noise into the estimated velocities. Pole selection therefore requires a compromise:

- poles too close to the unit circle: slow convergence and large lag;
- poles too close to the origin: fast convergence but noisy estimates;
- badly chosen poles outside the unit circle: divergent observer.

The observer used in this class was validated previously using physical CSV logs. The gains are intentionally fixed in the student panel so that the Class-10 experiment focuses on comparing estimation methods and tuning MPC, not on changing the observer matrix and the controller simultaneously.

3.5 Fixed observer used by the Class-10 firmware

For efficient execution, the firmware stores the observer in the combined form

$$\hat{\boldsymbol{x}}_k = A_o\hat{\boldsymbol{x}}_{k-1} + B_o u_{k-1} + L_o\boldsymbol{y}_k.$$

The fixed matrices are

$$A_o = \begin{bmatrix} 0.82469568 & 0.00438721 & -0.31915638 & -0.00085553 \\ -1.75095359 & 0.94056709 & -0.04181246 & 0.00193564 \\ -0.05215683 & 0.00005656 & 0.76549687 & 0.00439733 \\ -1.38405407 & 0.07685667 & -15.83946960 & 0.95027468 \end{bmatrix},$$

$$B_o = \begin{bmatrix} 0.00001119 \\ 0.00396389 \\ -0.00001398 \\ -0.00583846 \end{bmatrix}, \qquad L_o = \begin{bmatrix} 0.17530432 & 0.31833008 \\ 1.75095359 & -0.19974721 \\ 0.05215683 & 0.23646539 \\ 1.38405407 & 16.66931771 \end{bmatrix}.$$

These matrices come from the validated earlier MPC firmware and are not panel parameters.

3.6 Implementation safeguards

The linear observer is valid only near the upright equilibrium. The Class-10 code therefore uses the following safeguards:

1. filtered difference velocities remain active throughout swing-up;
2. when the pendulum first enters the MPC region, the observer is initialised as
$$\hat{x} = [\theta, 0, \alpha, 0]^T;$$
3. the previous controller PWM is used as u_{k-1} in the observer update;
4. if the observer becomes non-finite, or if the observer pendulum angle differs from the measured angle by more than 35°, it is reinitialised;
5. estimated angular velocities are clipped to a wide numerical safety bound;
6. when the pendulum leaves the upright region, the observer is reset.

The selected **Velocity LPF beta** affects the difference estimator only. It is disabled as a tuning variable for the Luenberger recursion. Nevertheless, the difference estimate continues to be used by the swing-up energy logic, even when Luenberger mode is selected for upright MPC.

STUDENT TASK

With all MPC parameters fixed at their defaults, compare:

1. differential + LPF with at least three β values;
2. the fixed Luenberger observer;
3. the resulting $\dot{\alpha}$, $\dot{\theta}$, PWM smoothness and upright angle performance.

Do not tune MPC weights while carrying out the estimator comparison.

4 Linear Model Predictive Control Theory

4.1 Finite-horizon prediction

At time k, MPC predicts the system over N future samples:

$$\boldsymbol{x}_{k+i+1|k} = A_d\boldsymbol{x}_{k+i|k} + B_d u_{k+i|k}, \qquad i = 0, \ldots, N-1,$$

with

$$\boldsymbol{x}_{k|k} = \hat{\boldsymbol{x}}_k.$$

The physical prediction time is

$$T_p = NT_s.$$

For example, the default value $N = 8$ corresponds to

$$T_p = 8(0.005) = 0.04 \text{ s}.$$

The first few predicted states are

$$\begin{aligned}
\boldsymbol{x}_{k+1|k} &= A_d\boldsymbol{x}_k + B_d u_{k|k},\\
\boldsymbol{x}_{k+2|k} &= A_d^2\boldsymbol{x}_k + A_dB_d u_{k|k} + B_d u_{k+1|k},\\
\boldsymbol{x}_{k+3|k} &= A_d^3\boldsymbol{x}_k + A_d^2B_d u_{k|k} + A_dB_d u_{k+1|k} + B_d u_{k+2|k}.
\end{aligned}$$

4.2 Stacked prediction model

Define the future input sequence

$$U = \begin{bmatrix} u_{k|k} & u_{k+1|k} & \cdots & u_{k+N-1|k} \end{bmatrix}^T \in \mathbb{R}^N,$$

and the stacked future state vector

$$X = \begin{bmatrix} \boldsymbol{x}_{k+1|k}^T & \boldsymbol{x}_{k+2|k}^T & \cdots & \boldsymbol{x}_{k+N|k}^T \end{bmatrix}^T \in \mathbb{R}^{4N}.$$

Then

$$\boxed{X = S_x\boldsymbol{x}_k + S_uU},$$

where

$$S_x = \begin{bmatrix} A_d \\ A_d^2 \\ \vdots \\ A_d^N \end{bmatrix},$$

and

$$S_u = \begin{bmatrix} B_d & 0 & \cdots & 0 \\ A_dB_d & B_d & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A_d^{N-1}B_d & A_d^{N-2}B_d & \cdots & B_d \end{bmatrix}.$$

This condensed representation eliminates the future states as independent optimisation variables. Only U remains to be optimised.

4.3 Finite-horizon cost

The Class-10 cost has stage, input and terminal terms:

$$J = \sum_{i=0}^{N-1} \left(\boldsymbol{x}_{k+i|k}^T Q \boldsymbol{x}_{k+i|k} + u_{k+i|k}^T R u_{k+i|k} \right) + \boldsymbol{x}_{k+N|k}^T P \boldsymbol{x}_{k+N|k}.$$

The panel uses a diagonal state-weight matrix

$$Q = \operatorname{diag}(q_\theta, q_{\dot\theta}, q_\alpha, q_{\dot\alpha}), \qquad R = r > 0.$$

The default values are

$$Q = \operatorname{diag}(1, 0.05, 80, 2), \qquad R = 0.001.$$

The large q_α places the highest priority on the upright pendulum angle.

The terminal weight P is the stabilising solution of the discrete algebraic Riccati equation

$$P = A_d^T P A_d - A_d^T P B_d (R + B_d^T P B_d)^{-1} B_d^T P A_d + Q.$$

Using the infinite-horizon LQR value matrix as the terminal weight reduces the short-sightedness caused by a short prediction horizon.

4.4 Condensed quadratic program

Define

$$\bar{Q} = \mathrm{diag}(Q, Q, \ldots, P), \qquad \bar{R} = \mathrm{diag}(R, R, \ldots, R).$$

Substituting $X = S_x x_k + S_u U$ into the cost and omitting terms independent of U gives

$$J_U(U) = \frac{1}{2} U^T H U + (G x_k)^T U,$$

where

$$\boxed{H = 2(S_u^T \bar{Q} S_u + \bar{R})}, \qquad \boxed{G = 2 S_u^T \bar{Q} S_x}.$$

For fixed A_d, B_d, N, Q, R, P, matrices H and G do not change during a run. Only the linear term

$$f_k = G \hat{x}_k$$

changes at each control sample.

4.5 Input constraint

The motor command is bounded by

$$-u_{\max} \le U_i \le u_{\max}, \qquad i = 0, \ldots, N-1.$$

The default is $u_{\max} = 150$ PWM. The optimisation problem is therefore

$$\boxed{\begin{aligned} \min_{U} \quad & \frac{1}{2} U^T H U + (G \hat{x}_k)^T U, \\ \text{s.t.} \quad & -u_{\max} \le U_i \le u_{\max}. \end{aligned}}$$

This is a convex box-constrained quadratic program when H is positive definite.

> This is the central practical difference from the Class-9 LQR experiment. LQR first calculates an unconstrained command and then clips it. MPC includes the future PWM bounds inside the optimisation problem.

4.6 Receding horizon

Solving the QP gives

$$U_k^* = \begin{bmatrix} u_{k|k}^* & u_{k+1|k}^* & \cdots & u_{k+N-1|k}^* \end{bmatrix}^T.$$

Only the first entry is applied:

$$\boxed{u_k = u_{k|k}^*}.$$

At the next sample, the new state estimate is used and the optimisation is solved again. This repeated prediction–optimisation–application process is called receding-horizon control.

5 Projected-Gradient Solution on the STM32

5.1 Gradient and projection

For

$$J(U) = \frac{1}{2} U^T H U + f^T U,$$

the gradient is

$$\nabla J(U) = H U + f.$$

A projected-gradient iteration is

$$U^{(j+1)} = \Pi_{[-u_{\max}, u_{\max}]} \left[U^{(j)} - \eta (H U^{(j)} + f) \right],$$

where projection means element-wise clipping:

$$U_i \leftarrow \max(-u_{\max}, \min(u_{\max}, U_i)).$$

5.2 Automatically selected PGD step

The older fixed-parameter implementation could use a manually selected step because H never changed. In the Class-10 package, students can change N, Q and R, so the Hessian changes whenever settings are saved. The code therefore computes the conservative step

$$\boxed{\eta = \frac{0.95}{\max_i \sum_j |H_{ij}|}}.$$

The denominator is the maximum absolute row sum, which bounds the spectral radius of H. The factor 0.95 provides margin. Students do not enter η manually.

IMPORTANT

Do not compare the automatically generated Class-10 PGD step directly with a fixed step used by an older precomputed-H firmware. The numerical value of the safe step depends on the Hessian scaling and therefore on N, Q, R and the objective convention.

5.3 Warm start

The previous optimal sequence is shifted forward:

$$U_k^{(0)} = \begin{bmatrix} u^*_{k|k-1} & u^*_{k+1|k-1} & \cdots & u^*_{k+N-2|k-1} & u^*_{k+N-2|k-1} \end{bmatrix}^T.$$

Because the state changes only slightly over one 5 ms sample, this is normally a much better initial guess than an all-zero sequence. Warm starting allows a small fixed number of iterations to produce a useful approximate solution.

5.4 What is rebuilt when SAVE is pressed

The Python panel sends N, Q, R and other parameters to the STM32. Before the firmware acknowledges the configuration, it:

1. validates all parameter ranges;
2. solves the DARE iteratively to obtain terminal P;
3. generates powers of A_d and the prediction blocks;
4. builds the condensed matrices H and G;
5. symmetrises/checks the Hessian;
6. computes the safe PGD step;
7. clears the warm-start sequence;
8. sends `ACK,CONFIG` only after the matrices are ready.

The maximum static prediction horizon is

$$N_{\max} = 20.$$

Arrays are allocated statically for this maximum size; runtime loops use only the selected horizon. This avoids dynamic memory allocation inside the control loop.

5.5 Online 200 Hz control sequence

At each 5 ms control update, the firmware performs the following operations:

1. read the potentiometer and encoder;
2. convert raw sensor values to α and θ;
3. update filtered difference velocities;

4. update the upright-region hysteresis state;
5. initialise, update or reset the Luenberger observer if selected;
6. construct the four-state controller input;
7. calculate the swing-up command;
8. compute $f = G\hat{x}$;
9. warm start and execute the selected number of PGD iterations;
10. take the first MPC input;
11. apply soft blending, saturation and motor sign;
12. store the applied controller PWM for the next observer update;
13. transmit telemetry to the Python panel.

The Python program configures, visualises and logs the experiment. The real-time state estimation and control calculation run on the STM32.

6 Parameters Students Must Tune

6.1 Primary Class-10 parameters

Panel parameter	Default	Allowed range	Main effect
Prediction horizon N	8	4–20	Longer look-ahead and larger QP; may improve anticipation but increases computation and sensitivity to model mismatch.
q_θ	1	≥ 0	Penalises rotary-arm displacement from the reference.
$q_{\dot{\theta}}$	0.05	≥ 0	Penalises rotary-arm angular speed and rapid arm motion.
q_α	80	≥ 0	Penalises pendulum-angle error; usually the largest state weight.
$q_{\dot{\alpha}}$	2	≥ 0	Penalises pendulum angular speed and upright oscillation.
Input weight R	0.001	> 0	Larger R reduces PWM effort; smaller R makes control more aggressive.
PGD iterations	16	1–100	More iterations improve QP convergence but increase control computation.
State estimator	Difference + LPF	two choices	Selects filtered numerical differentiation or fixed Luenberger estimation for upright MPC.
Velocity LPF β	0.25	0.001–1	Used only in difference mode; sets the smoothing/delay trade-off.

The software checks that at least one state weight is positive and that $R > 0$. Extremely large or poorly scaled weights can still produce an impractical controller even when they pass the numerical range checks.

6.2 Secondary parameters that should normally remain fixed

Parameter	Default	Guidance
PWM max	150	A safety and actuator constraint. Do not increase during initial tuning.
Swing pump PWM	120	Belongs to the retained swing-up stage, not the MPC design.
MPC enter angle	15°	Determines when local MPC is captured.
MPC exit angle	25°	Must exceed the enter angle to provide hysteresis.
Soft blend λ	0.18	Controls the transition rate between swing-up and MPC.
Noise settings	Class-9 defaults	Keep fixed during controlled comparisons.
Observer matrices	fixed	Not edited by students in Class 10.
PGD step η	automatic	Recomputed from H at SAVE; not a panel parameter.

6.3 How each MPC weight changes behaviour

The weights must be interpreted relative to each other.

- Increase q_α when the pendulum angle is too loose near upright. Excessive values may cause aggressive PWM, noise amplification or arm motion.
- Increase $q_{\dot\alpha}$ when the pendulum repeatedly crosses upright with excessive speed or oscillates.
- Increase q_θ when the rotary arm drifts too far from its reference. This can compete with the immediate pendulum-balancing objective.
- Increase $q_{\dot\theta}$ when arm motion is too fast or mechanically harsh.
- Increase R when PWM is noisy, saturated frequently or unnecessarily large. Too large an R can make the controller too weak to stabilise.

6.4 Recommended tuning sequence

STUDENT TASK

Use the following sequence. Change only one parameter group at a time.

1. Start from the defaults:
$$N = 8, \quad Q = \mathrm{diag}(1, 0.05, 80, 2), \quad R = 0.001, \quad n_{\text{iter}} = 16.$$
2. In simulation, disable noise briefly and verify that switching, observer initialisation and MPC operation are correct.
3. Restore the standard noise settings.
4. Compare difference + LPF and Luenberger using identical MPC settings.
5. Select the estimator to use for the remaining MPC study.
6. Sweep N through a small set such as $4, 8, 12, 16$ while keeping Q, R and PGD iterations fixed.
7. Fix N and tune q_α and $q_{\dot\alpha}$ first.
8. Tune q_θ and $q_{\dot\theta}$ to control arm excursion.
9. Tune R to balance tracking and PWM effort.
10. Reduce or increase PGD iterations and check whether performance changes. If results are unchanged, the extra iterations are unnecessary.

11. Transfer only a simulation-validated parameter set to the real system.

6.5 Do not use random simultaneous tuning

Changing N, Q, R, estimator and solver iterations at the same time makes it impossible to identify why performance changed. Every comparison must keep all non-investigated parameters constant. Use informative run names and record the full parameter set for every result.

7 Performance Metrics and CSV Analysis

The simulation and physical panels retain the Class-9 result style. The CSV columns are

`time_s, theta_rad, theta_dot_rad_s, alpha_rad, alpha_dot_rad_s, pwm, mode, blend`.

The result figure contains pendulum angle, rotary-arm angle, PWM history and the stable-stage PWM distribution. Important comparison quantities include:

- time at which the final upright phase begins;
- mean and standard deviation of $|\alpha|$ in the final stable phase;
- mean and standard deviation of |PWM|;
- maximum rotary-arm excursion;
- loss of capture or repeated MPC exit events;
- visible velocity-estimate noise and PWM chattering;
- fraction of samples at the PWM limit.

A smaller angle error is not automatically a better design if it requires continuous saturation, excessive arm travel or a much larger computational budget. Evaluate stability, actuator effort and implementation cost together.

8 Running the Nonlinear Simulation Environment

8.1 Files and dependencies

File	Purpose
`rip_mpc_sim.py`	Student nonlinear digital twin with editable MPC parameters and estimator selection.

Install the required packages:

```
python3 -m pip install numpy scipy matplotlib PyQt5
```

Run the student simulation:

```
python3 rip_mpc_sim.py
```

8.2 Student-panel workflow

1. Open `rip_mpc_sim.py`.
2. Select the estimator.
3. Enter N, Q, R and PGD iterations.
4. Keep initial condition, duration, swing-up, blend and noise settings fixed during a controlled comparison.

5. Click **SAVE / Apply Settings**. SAVE rebuilds terminal P, H, G and the safe PGD step. Unsaved values are ignored by GO.
6. Click **GO**. Observe the nonlinear 3D motion and the runtime mode.
7. Inspect the automatically generated result figure and CSV.
8. Repeat using a clear identifier such as `SIM_LUEN_N08_QA80_R001_I16_R1`.

8.3 Suggested simulation experiment matrix

Stage	Variable	Minimum comparison
A	estimator	Difference + LPF versus Luenberger, all MPC settings identical.
B	LPF β	Three values in difference mode, for example 0.10, 0.25, 0.50.
C	horizon N	At least four values, for example 4, 8, 12, 16.
D	pendulum weights	Change q_α and $q_{\dot\alpha}$ systematically.
E	arm weights	Change q_θ and $q_{\dot\theta}$ systematically.
F	input weight	At least three values around the baseline $R = 0.001$.
G	PGD iterations	Compare a low, baseline and higher iteration count.

8.4 Optional headless execution

A repeatable headless run can be used for parameter sweeps. Example:

```
python3 rip_mpc_sim.py --headless --duration 10 --csv \
  --horizon 8 \
  --q-theta 1 --q-theta-dot 0.05 \
  --q-alpha 80 --q-alpha-dot 2 \
  --r-input 0.001 --pgd-iterations 16 \
  --estimator luenberger
```

Use `-estimator differential` together with `-velocity-lpf` to evaluate the filtered-difference method.

9 Running the Physical System

9.1 Files

File	Purpose
`rip_mpc_hardware.ino`	STM32 firmware: 200 Hz sensing, two estimator modes, swing-up, dynamic MPC matrix construction, PGD, blend, motor drive and telemetry.
`rip_mpc_hardware.py`	Student physical-system panel with editable MPC parameters and estimator selection.

The physical Python panel requires:

```
python3 -m pip install numpy matplotlib PyQt5 pyserial
```

DANGER

The pendulum and rotary arm move quickly during swing-up. Keep the complete rotation area clear. Do not touch the apparatus while the motor supply is connected. Press **STOP** immediately if the arm repeatedly hits a limit, a cable moves, the observer estimate is visibly unreasonable, or the measured angles are incorrect.

9.2 Calibration and experiment sequence

1. Flash `rip_mpc_hardware.ino` using the same STM32 board and pin configuration used in the previous classes.
2. Start the student panel:

```
python3 rip_mpc_hardware.py
```

3. Select the serial port and click **Connect**. Wait for `READY,RIP_MPC_HW,4`.
4. Hold the pendulum upright and click **Hold upright** → **Calibrate** $\alpha = 0$.
5. Let the pendulum hang down and click **Hang down** → **Calibrate** $\alpha = \pi$.
6. Verify the encoder direction and motor sign.
7. Start from the default validated MPC parameters and select the desired estimator.
8. Click **SAVE / Send Parameters to STM32**. Wait for the configuration acknowledgement; the firmware is rebuilding the MPC matrices.
9. Clear the complete motion area and click **GO**.
10. Save the automatically generated PNG and CSV files.
11. Repeat trials without changing settings before drawing conclusions.

9.3 Safe transfer from simulation to hardware

Use the following order:

1. verify capture and stability in noiseless simulation;
2. verify performance with the standard simulation noise;
3. confirm that PWM is not continuously saturated;
4. use the default PWM limit on hardware;
5. use a short experiment duration for the first physical trial;
6. run at least two repetitions before changing parameters;
7. stop immediately if arm excursion or vibration is unsafe.

IMPORTANT

A simulation result does not guarantee physical stability. Model mismatch, friction, motor dead zone, sensor calibration error and cable forces can all change the effective dynamics. Increase aggressiveness only after a stable baseline physical run has been obtained.

10 Required Student Analysis

For the Class-10 study, record and explain the following.

STUDENT TASK

1. Write the discrete Luenberger observer equation and derive $e_{k+1} = (A_d - L_d C)e_k$.
2. Explain why the discrete stability condition is a unit-circle condition.
3. Compare the noise–delay behaviour of difference + LPF with the model-based observer.
4. Derive $X = S_x x_k + S_u U$.

5. Derive H and G for the condensed QP.
6. Explain why only the first optimised input is applied.
7. Explain warm start and the automatically selected PGD step.
8. Present a controlled parameter study of N, Q, R and PGD iterations.
9. Compare at least one simulation result with a physical-system result.
10. Discuss angle error, PWM effort, arm excursion, estimator quality and computational cost together.

A parameter table should accompany every result. At minimum include:

$$N,\ q_\theta,\ q_{\dot\theta},\ q_\alpha,\ q_{\dot\alpha},\ R,\ n_{\text{iter}},\ \text{estimator},\ \beta,\ u_{\max}.$$

11 Quick Troubleshooting

Symptom	Check
GO is disabled	Connect, calibrate, click SAVE, and wait for the STM32 acknowledgement.
Firmware mismatch	Flash `rip_mpc_hardware.ino`; the panel expects protocol version 4.
`ERR,MPC_BUILD`	Return to moderate positive weights and R, use $N = 8$, then SAVE again.
Simulation uses old values	Click SAVE after every parameter or estimator change.
Immediate divergence near upright	Check angle calibration, encoder/motor sign, state convention and estimator selection.
Pendulum never reaches MPC	Check swing PWM, calibration, mechanical freedom and MPC enter angle.
Repeated MPC enter/exit	Check calibration, increase hysteresis gap, inspect estimator noise, and avoid overly weak MPC settings.
Very noisy PWM in difference mode	Reduce β, increase R, or compare with the Luenberger observer.
Luenberger estimate jumps	Check calibration and input sign. The observer is reset if its angle disagrees by more than 35°.
Slow or weak stabilisation	Check that R is not too large and that q_α is not too small.
Continuous PWM saturation	Increase R, reduce aggressive state weights, or reduce the capture region.
Large arm drift	Increase q_θ gradually, while checking that pendulum stability is retained.
Changing PGD iterations has no effect	The lower iteration count may already be sufficient because warm start is effective.
Physical result differs from simulation	Check calibration, friction, motor dead zone, cable force and model mismatch; do not compensate by changing every weight simultaneously.

12 Summary

Class 10 links state estimation, optimal control and embedded implementation. The discrete Luenberger observer reconstructs the four-state vector from two angle measurements by combining model prediction with innovation correction. The MPC controller then predicts future motion, penalises state error and

PWM, enforces the input bound and repeatedly solves a small box-constrained QP.

The main student decisions are the estimator selection, LPF coefficient in difference mode, prediction horizon, four state weights, input weight and PGD iteration count. The observer matrices and PGD step are deliberately fixed or automatically generated. A valid experiment changes one parameter group at a time and evaluates angle regulation, arm motion, control effort, estimator quality and real-time feasibility together.