AI Enabled Control Engineering

Lecture 8: Hardware Assembly, Sensor Test, and the Bellman-to-Control Bridge

Professor Xun Huang

TA: Zhixiang Ju Haozhe Wang

Aeronautics and Astronautics
College of Engineering
Peking University

huangxun@pku.edu.cn
https://xunger99.github.io/xunger/

Recap: from Lecture 7 to Lecture 8

- Lecture 7 introduced the rotary inverted pendulum hardware modules.
- We discussed the DC motor, TB6612 driver, sensors, and STM32F446RE board.
- We derived the modelling objective: input u determines $\theta, \dot{\theta}, \alpha, \dot{\alpha}$.
- Today we connect the physical system to real sensor readings and optimal control ideas.

Lecture 8

- Build the real rotary inverted pendulum circuit .
- Flash firmware and test potentiometer and encoder readings.
- Start from Bellman's optimality principle and reach HJB, LQR, MPC, and RL.

Learning objectives

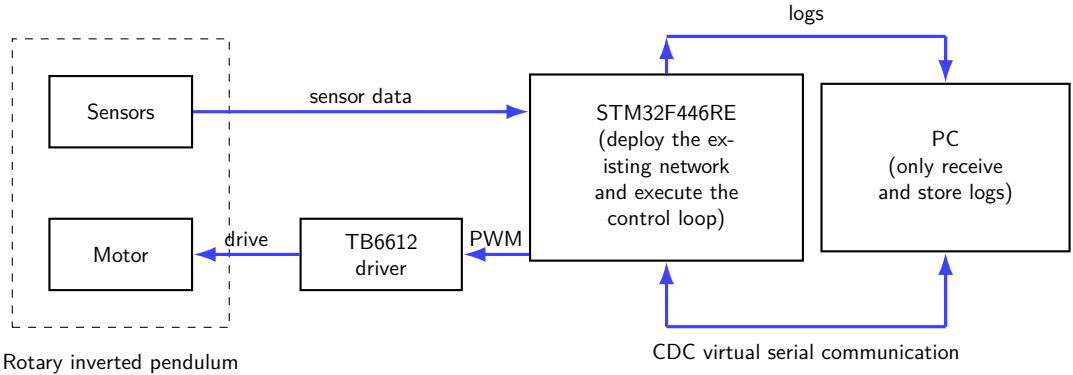
By the end of this lab, students should be able to:

- explain the signal flow from sensor to STM32 ;
- safely wire and test the STM32–TB6612–sensor system;
- convert raw potentiometer and encoder signals into angle-related measurements;
- describe why LQR, MPC, and RL can be viewed under a common optimal-control framework.

Lab safety

- Do not connect the motor power before the wiring is checked.
- Always connect a common ground between STM32, TB6612, and power supply.
- Keep the pendulum free from obstacles before any motor test.
- In this lecture, the motor is not used for closed-loop balancing yet.
- When uncertain, disconnect motor power first, then debug the code.

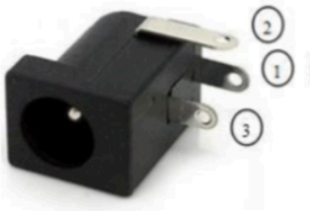
Hardware architecture



DC Power Jack: Interface and Pins

Pin definition

- Pin 1: positive terminal, connected to $V_{\text{in}}+$.
- Pin 2: negative terminal, connected to GND.
- Pin 3: switched negative terminal, not used in our circuit.

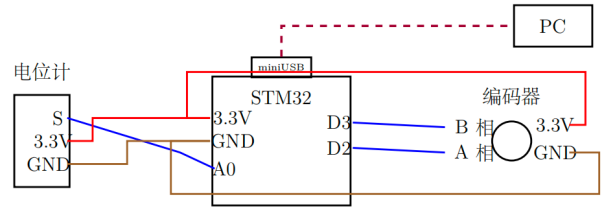


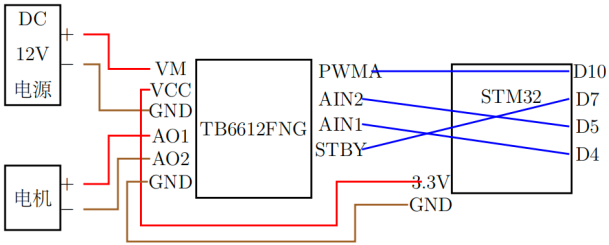
DC-002 / DC-005 power jack used for external DC input.

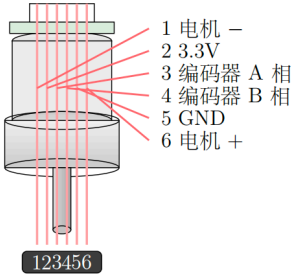
Pins used in our experiment

1: $V_{\text{in}} + (VM)$ 3: GND

Wiring Diagram of the PC–STM32 RIP System







TA Check Before Power-on

Before power-on, each group should ask a TA to check the circuit.

1. Is the STM32 logic supply connected to $3.3\,\mathrm{V}$ instead of $5\,\mathrm{V}$?
2. Are all Dupont wires firmly connected?
3. Are the breadboard vertical rails correctly assigned?
4. Are there four wires on the GND rail and four wires on the $3.3\,\mathrm{V}$ rail?
5. Is the external $12\,\mathrm{V}$ supply connected only to the TB6612 motor-power side?

Next step: flash sensor-test firmware

- The circuit has been connected and checked.
- Now we flash a simple test program to STM32.
- This program does not control the motor.
- It only reads:
  - potentiometer raw value;
  - encoder count value.

Safety reminder

During this test, the external motor power can remain disconnected.

Step 1: create a new Arduino sketch

```
static const int PIN_POT = A0;
static const int PIN_ENC_A = 2;
static const int PIN_ENC_B = 3;

static volatile long g_enc = 0;

void setup() {
   Serial.begin(115200);

   pinMode(PIN_POT, INPUT);

   pinMode(PIN_ENC_A, INPUT_PULLUP);
   pinMode(PIN_ENC_B, INPUT_PULLUP);

   attachInterrupt(
      digitalPinToInterrupt(PIN_ENC_A),
      isr_enc_a,
      CHANGE
   );

   attachInterrupt(
      digitalPinToInterrupt(PIN_ENC_B),
      isr_enc_b,
      CHANGE
   );
}
```

Step 1: serial output code

- Then add the loop function below.
- Save the sketch before uploading it to STM32.

```
void loop() {
   int pot_raw = analogRead(PIN_POT);

   noInterrupts();
   long enc_now = g_enc;
   interrupts();

   Serial.print("pot_raw = ");
   Serial.print(pot_raw);

   Serial.print(", enc = ");
   Serial.println(enc_now);

   delay(100);
}
```

Purpose

The STM32 will send potentiometer and encoder values to the PC every 100 ms.

Step 2: select board and port

- Connect STM32 to the PC using the onboard miniUSB cable.
- In Arduino IDE, select the correct board:
  - `Nucleo-64`
  - or the corresponding STM32F446RE board option.
- Select the correct serial port.

Common mistake

If the wrong board or port is selected, the code cannot be uploaded successfully.

Step 3: upload the firmware

- Click the **Upload** button in Arduino IDE.
- Wait until the IDE shows that uploading is completed.
- If uploading fails:
  - check the USB cable;
  - check the board selection;
  - check the serial port;
  - reconnect STM32 and try again.

Expected result

After successful upload, STM32 starts reading the sensors automatically.

Step 4: open the serial monitor

- After uploading, open:

$$\texttt{Tools} \rightarrow \texttt{Serial Monitor}$$

- Set the baud rate to:

$$115200$$

- The serial monitor should display sensor data continuously.

Important

The baud rate in the serial monitor must match the baud rate in the firmware.

Expected serial output

- If the firmware is running correctly, the serial monitor shows values like:

```
pot_raw = 3092, enc = 10000
pot_raw = 3091, enc = 10002
pot_raw = 3093, enc = 10005
```

- `pot_raw`: raw ADC value of the potentiometer.
- `enc`: accumulated encoder count.

What students should do

Rotate the pendulum and rotary arm by hand, then observe how the two values change.

From raw sensor values to angles

- In the previous firmware, we printed raw values:

```
pot_raw      enc
```

- Now we make a minimum modification:
  - convert encoder count to arm angle θ;
  - convert potentiometer raw value to pendulum angle α.
- This conversion method will be used as the common firmware convention in later experiments.

Sensor logging files provided

- We provide two files for this experiment:

  `rip_sensor_logger.ino` `rip_record_10s.py`

- The STM32 firmware reads the potentiometer and encoder at 200 Hz.
- The Python script sends the `GO` command, records data for 10 s, saves a CSV file, and plots the angle curves.

Important

The motor is not controlled in this test. This step is only for sensor calibration and data logging.

Step 1: calibrate the potentiometer zero

1. Hold the vertical pendulum arm at the upright position.
2. Keep the pendulum still.
3. Open the Arduino Serial Monitor.
4. Read the current potentiometer raw value.
5. Modify `POT_ZERO` in `rip_sensor_logger.ino`.

Example

If the serial monitor shows `RAW,pot=3132,...`, then set: `POT_ZERO = 3132`.

Step 2: modify the calibration constants

```
static volatile int32_t POT_ZERO = 3132;
static volatile int32_t POT_DOWN_RAW = 1036;
```

- `POT_ZERO`: raw value when the pendulum is upright.
- `POT_DOWN_RAW`: raw value when the pendulum is naturally hanging down.
- After modifying these values, upload the firmware again.

Very important

A wrong `POT_ZERO` will cause a wrong pendulum angle α. This will directly affect all later control experiments.

Step 3: upload the firmware

- Open `rip_sensor_logger.ino` in Arduino IDE.
- Select the correct STM32 board and serial port.
- Upload the firmware to STM32.
- Open the Serial Monitor and set the baud rate to:

921600

Expected before `GO`

The STM32 prints raw sensor readings slowly, so students can check the potentiometer and encoder values.

Step 4: run the Python logging script

- Close the Arduino Serial Monitor first.
- Open a terminal in the folder containing `rip_record_10s.py`.
- Run:

```
python rip_record_10s.py
```

- The script automatically finds the serial port when possible.
- It sends `GO` to STM32 and records data for 10 s.

Step 5: what happens after GO

- STM32 records the current encoder count as the zero position:

$$\text{current arm position} \Rightarrow \theta = 0$$

- STM32 sends one data line every $5\,\mathrm{ms}$.
- The serial output format is:

```
DATA,t_ms,pot,enc,theta,alpha,theta_dot,alpha_dot
```

After 10 seconds

Python saves the latest CSV file and automatically plots angle and angular-velocity curves.

Expected output

- A CSV file will be generated:

  `rip_sensor_log_YYYYMMDD_HHMMSS.csv`

- A figure will also be generated:

  `rip_sensor_log_YYYYMMDD_HHMMSS.png`

- The figure contains:
  - $\theta(t)$ and $\alpha(t)$;
  - $\dot{\theta}(t)$ and $\dot{\alpha}(t)$.

Motor test files provided

- We now test whether STM32 can drive the motor through TB6612FNG.
- We provide two files:

  `rip_motor_test.ino` `rip_motor_test_10s.py`

- The new firmware is based on the previous sensor logger.
- It keeps 200 Hz sensor logging and adds a safe PWM motor test.

Before motor test

Make sure the pendulum is free to move, all wires are firm, and the external 12 V motor power is connected only to the TB6612 motor-power side.

Run the motor test

1. Upload `rip_motor_test.ino` to STM32.
2. Close the Arduino Serial Monitor.
3. Run the Python script:

```
python rip_motor_test_10s.py
```

4. The Python script sends `GO` and then sends a small PWM test sequence.
5. After 10 s, it saves a CSV file and plots the sensor and PWM curves.

Expected result

The motor should rotate gently according to the PWM command, while the PC records θ, α, $\dot{\theta}$, $\dot{\alpha}$, and PWM.

Bellman, optimal control, and reinforcement learning

- We have just built and tested the real rotary inverted pendulum system.
- Now we return to the central question:

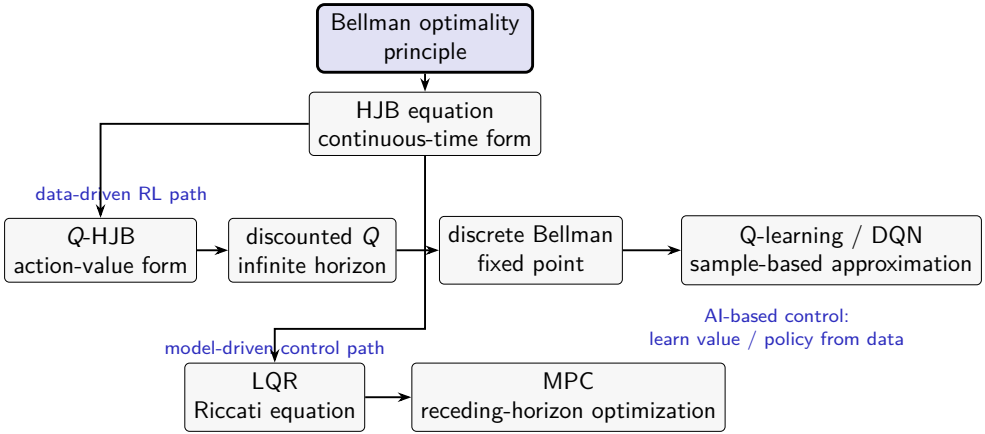
  How should a controller choose the best action?

- This question leads to Bellman's optimality principle.
- From this principle, we can understand LQR, MPC, and DQN under one framework.

Key message

AI-based control is not isolated from classical control. It can be viewed as approximate optimal control when the model, value function, or action space becomes difficult to handle explicitly.

Unified view under Bellman's optimality principle



Bellman's optimality principle

- A globally optimal policy has a recursive structure.
- After the first action is chosen, the remaining policy must still be optimal for the new state.
- Therefore, an optimal control problem can be decomposed into:

$$\text{current cost} \quad + \quad \text{optimal future cost}.$$

Intuition

The best current action should not only reduce the immediate cost, but also put the system into a state with small future cost.

Continuous-time optimal control problem

Consider a deterministic continuous-time system:

$$\dot{x}(t) = f(x(t), u(t), t), \qquad x(t) \in \mathbb{R}^n, \quad u(t) \in \mathcal{U}.$$

The finite-horizon cost is:

$$J(u(\cdot); x_0) = \int_0^T g(x(t), u(t), t)\, dt + h(x(T)).$$

The optimal cost-to-go is:

$$J^*(t, x) = \inf_{u(\cdot)} \left\{ \int_t^T g(x(s), u(s), s)\, ds + h(x(T)) \right\}.$$

$J^*(t, x)$ measures the minimum remaining cost from state x at time t.

Discrete cost and Bellman recursion

Using a small time step δ, define

$$t_k = k\delta, \qquad x_k = x(t_k), \qquad u_k = u(t_k).$$

The continuous dynamics can be approximated by

$$x_{k+1} \approx x_k + f(x_k, u_k, t_k)\delta.$$

The finite-horizon cost becomes

$$\tilde{J}(\{u_k\}; x_0) = h(x_N) + \sum_{k=0}^{N-1} g(x_k, u_k, t_k)\delta.$$

Therefore, the optimal cost-to-go satisfies

$$\tilde{J}^*(t_k, x) = \min_{u \in \mathcal{U}} \left\{ g(x, u, t_k)\delta + \tilde{J}^*\left(t_{k+1}, x + f(x, u, t_k)\delta\right) \right\}.$$

From Bellman recursion to HJB

From the previous slide, Bellman recursion gives

$$\tilde{J}^*(k\delta, x) = \min_{u \in \mathcal{U}} \left\{ g(x, u, k\delta)\delta + \tilde{J}^*\big((k+1)\delta,\ x + f(x, u, k\delta)\delta\big) \right\}.$$

Assume $\tilde{J}^*(t, x)$ is sufficiently smooth. Then

$$\begin{aligned} &\tilde{J}^*\big((k+1)\delta,\ x + f(x, u, k\delta)\delta\big) \\ &= \tilde{J}^*(k\delta, x) + \frac{\partial \tilde{J}^*}{\partial t}(k\delta, x)\delta + \nabla_x \tilde{J}^*(k\delta, x)^\top f(x, u, k\delta)\delta + o(\delta). \end{aligned}$$

Substitute the Taylor expansion into Bellman recursion and cancel $\tilde{J}^*(k\delta, x)$.

From Bellman recursion to HJB

After substitution and cancellation,

$$0 = \min_{u \in \mathcal{U}} \left\{ \left[g(x, u, k\delta) + \frac{\partial \tilde{J}^*}{\partial t}(k\delta, x) + \nabla_x \tilde{J}^*(k\delta, x)^\top f(x, u, k\delta) \right] \delta + o(\delta) \right\}.$$

Dividing by δ, and letting

$$\delta \to 0, \qquad k\delta \to t,$$

we obtain

$$0 = \min_{u \in \mathcal{U}} \left\{ g(x, u, t) + \frac{\partial J^*}{\partial t}(t, x) + \nabla_x J^*(t, x)^\top f(x, u, t) \right\}.$$

From Bellman recursion to HJB

$$-\frac{\partial J^*}{\partial t} = \min_{u \in \mathcal{U}} \left\{ g(x, u, t) + \nabla_x J^*(t, x)^\top f(x, u, t) \right\}.$$

- $g(x, u, t)$: immediate cost.
- $\nabla_x J^{*\top} f$: effect of the current action on future cost.
- The optimal action minimizes the sum of these two terms.

Difficulty

For nonlinear high-dimensional systems, HJB is usually a nonlinear partial differential equation and is hard to solve directly.

LQR: a special solvable case of HJB

For a linear system:

$$\dot{x} = Ax + Bu,$$

with quadratic cost:

$$J = \frac{1}{2}x(T)^\top Hx(T) + \frac{1}{2}\int_t^T \left(x^\top R_{xx}x + u^\top R_{uu}u\right) d\tau.$$

Because the system is linear and the cost is quadratic, assume:

$$J^*(t,x) = \frac{1}{2}x^\top P(t)x.$$

Key idea

LQR is obtained by substituting a quadratic value function into the HJB equation.

From HJB to the LQR feedback law

For

$$J^*(t,x) = \frac{1}{2}x^\top P(t)x,$$

we have:

$$\nabla_x J^*(t,x) = P(t)x.$$

Substituting into HJB:

$$-\frac{1}{2}x^\top \dot{P}x = \min_u \left\{ \frac{1}{2}x^\top R_{xx}x + \frac{1}{2}u^\top R_{uu}u + x^\top P(Ax+Bu) \right\}.$$

Taking derivative with respect to u:

$$R_{uu}u + B^\top Px = 0.$$

Therefore:

$$u^*(t) = -R_{uu}^{-1}B^\top P(t)x(t).$$

Riccati equation and LQR controller

After substituting the optimal u^* back into HJB:

$$-\dot{P} = PA + A^\top P + R_{xx} - PBR_{uu}^{-1}B^\top P.$$

For infinite-horizon time-invariant LQR:

$$A^\top P + PA + R_{xx} - PBR_{uu}^{-1}B^\top P = 0.$$

The feedback controller is:

$$u = -Kx, \qquad K = R_{uu}^{-1}B^\top P.$$

Conclusion

LQR is Bellman's optimality principle under linear dynamics, quadratic cost, and a quadratic value-function assumption.

MPC: another model-driven realization

HJB gives the optimal feedback law if J^* is known:

$$u^*(t,x) \in \arg\min_u \left\{ g(x,u,t) + \nabla_x J^*(t,x)^\top f(x,u,t) \right\}.$$

But for general systems:

- the HJB equation is hard to solve;
- constraints are important in real hardware;
- the controller must be computable online.

MPC idea

Instead of solving the full HJB equation, solve a finite-horizon optimization problem repeatedly at each sampling time.

MPC finite-horizon optimization

At time step k, given the current state x_k, solve:

$$U^*(x_k) = \arg\min_U \left\{ \sum_{i=0}^{N-1} \ell(x_{k+i|k}, u_{k+i|k}) + V_f(x_{k+N|k}) \right\}.$$

Subject to:

$$x_{k+i+1|k} = F(x_{k+i|k}, u_{k+i|k}), \qquad x_{k|k} = x_k.$$

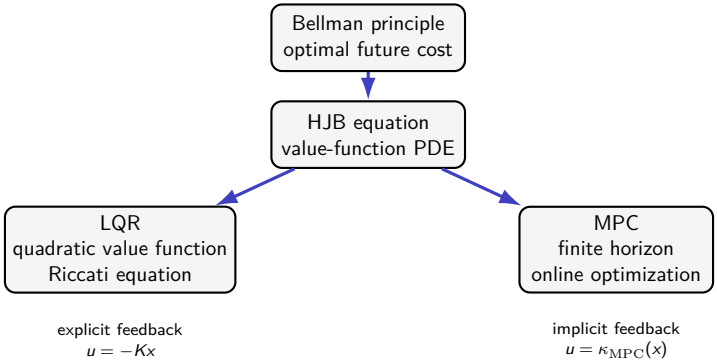
The optimal control sequence is:

$$U^* = \begin{bmatrix} u^*_{k|k} & u^*_{k+1|k} & \cdots & u^*_{k+N-1|k} \end{bmatrix}^\top.$$

Receding horizon

Only the first control input $u^*_{k|k}$ is applied. Then the state is measured again and the optimization is solved again.

LQR and MPC under the Bellman viewpoint



For the rotary inverted pendulum

LQR is fast and simple near the upright equilibrium. MPC can include input constraints and prediction, but requires more online computation.

Why introduce the Q function?

HJB selects the optimal action by:

$$u^*(t,x) \in \arg\min_u \left\{ g(x,u,t) + \nabla_x J^*(t,x)^\top f(x,u,t) \right\}.$$

This expression depends on:

- the system model $f(x,u,t)$;
- the value-function gradient $\nabla_x J^*$;
- solving or approximating the HJB equation.

Action-value idea

Instead of evaluating only states, define a value for each state-action pair:

$$Q(x,u).$$

Then action selection can be written as

$$u^*(x) \in \arg\min_u Q(x,u).$$

Continuous-time action-value function

To define an action-value function in continuous time, we treat the current control value u as part of the state.

Introduce an augmented system:

$$\begin{cases} \dot{x}(t) = f(x(t), u(t), t), \\ \dot{u}(t) = a(t), \end{cases} \qquad u(t) \in \mathcal{U}, \quad \|a(t)\|_2 \leq M.$$

Here $a(t)$ is the rate of change of the control input.

Why introduce $a(t)$?

In continuous time, the current action u is not just selected once and then forgotten. To define $Q(x, u, t)$, we fix the current value of u, and optimize how u changes afterwards.

Definition of continuous-time Q function

Given current state x, current control value u, and time t, define

$$Q(x,u,t) \triangleq \inf_{a(\cdot)} \left\{ \int_t^T g(x(s),u(s),s)\,ds + h(x(T)) \;\middle|\; \begin{array}{l} \dot{x}(s) = f(x(s),u(s),s), \\ \dot{u}(s) = a(s), \\ x(t) = x, \quad u(t) = u, \\ u(s) \in \mathcal{U}, \quad \|a(s)\|_2 \le M \end{array} \right\}.$$

Relation between J^* and Q

$$J^*(t,x) = \min_{u \in \mathcal{U}} Q(x,u,t).$$

The optimal state value is obtained by choosing the best initial control value u.

Dynamic programming equation for Q

For any small time interval $h > 0$, Bellman's principle gives

$$Q(x, u, t) = \inf_{\|a(\cdot)\|_2 \le M} \left\{ \int_t^{t+h} g(x(s), u(s), s)\, ds + Q(x(t+h), u(t+h), t+h) \right\}.$$

The terminal state $(x(t+h), u(t+h))$ is generated by

$$\dot{x} = f(x, u, t), \qquad \dot{u} = a.$$

One-step structure

Current Q value equals short-time cost plus the optimal future Q value from the augmented next state.

First-order expansion of the augmented system

For small h, the stage cost satisfies

$$\int_t^{t+h} g(x(s), u(s), s)\, ds = g(x, u, t)h + o(h).$$

The augmented state evolves as

$$x(t+h) = x + f(x, u, t)h + o(h),$$

$$u(t+h) = u + ah + o(h).$$

Blackboard derivation

The next step is to expand $Q(x(t+h), u(t+h), t+h)$ at (x, u, t).

Taylor expansion of Q

Assume

$$Q \in C^1(\mathbb{R}^n \times \mathbb{R}^m \times [0, T]).$$

Then

$$\begin{aligned} &Q(x(t+h), u(t+h), t+h) \\ &= Q(x, u, t) + \partial_t Q(x, u, t)h \\ &\quad + \nabla_x Q(x, u, t)^\top f(x, u, t)h + \nabla_u Q(x, u, t)^\top ah + o(h). \end{aligned}$$

New term

Compared with the ordinary HJB derivation, an extra term

$$\nabla_u Q(x, u, t)^\top a$$

appears because u is also treated as a state variable.

Substitution into the dynamic programming equation

Substitute the expansions into

$$Q(x,u,t) = \inf_{\|a(\cdot)\|_2 \le M} \left\{ \int_t^{t+h} g\,ds + Q(x(t+h), u(t+h), t+h) \right\}.$$

After cancelling $Q(x,u,t)$, dividing by h, and letting $h \to 0$, we obtain

$$\partial_t Q(x,u,t) + \nabla_x Q(x,u,t)^\top f(x,u,t) + g(x,u,t) + \inf_{\|a\|_2 \le M} \nabla_u Q(x,u,t)^\top a = 0.$$

This is the Q-form HJB equation before simplifying the a-minimization.

Solving the minimization over a

Let

$$p = \nabla_u Q(x, u, t).$$

We need to compute

$$\inf_{\|a\|_2 \le M} p^\top a.$$

By Cauchy–Schwarz,

$$p^\top a \ge -\|p\|_2 \|a\|_2 \ge -M\|p\|_2.$$

The lower bound is attained by

$$a^* = -M\frac{p}{\|p\|_2}, \qquad p \neq 0.$$

Therefore,

$$\inf_{\|a\|_2 \le M} p^\top a = -M\|p\|_2.$$

The Q-HJB equation

Using

$$p = \nabla_u Q(x, u, t), \qquad \inf_{\|a\|_2 \le M} p^\top a = -M\|p\|_2,$$

we obtain

$$\boxed{\partial_t Q(x, u, t) + \nabla_x Q(x, u, t)^\top f(x, u, t) - M\|\nabla_u Q(x, u, t)\|_2 + g(x, u, t) = 0.}$$

Interpretation

The term $\nabla_x Q^\top f$ describes how the state dynamics affect future cost. The term $-M\|\nabla_u Q\|_2$ comes from optimally changing the current control value u.

Meaning of the Q-HJB equation

- Ordinary HJB uses a state value function:

$$J^*(t, x).$$

- Q-HJB uses an action-value function:

$$Q(x, u, t).$$

- Once Q is known, the optimal action can be selected by

$$u^*(t, x) \in \arg\min_{u \in \mathcal{U}} Q(x, u, t).$$

Connection to reinforcement learning

The next step is to discretize time and action space. Then Q-based optimal control naturally leads to the discrete Bellman equation, Q-learning, and finally DQN.

From Q-HJB to discounted Q-HJB

$$\partial_t Q(x,u,t) + \nabla_x Q(x,u,t)^\top f(x,u,t) - M\|\nabla_u Q(x,u,t)\|_2 + g(x,u,t) = 0.$$

This equation is still a finite-horizon equation.

- The value function depends on absolute time t.
- The terminal cost $h(x(T))$ appears at the final time.
- For reinforcement learning, we often use an infinite-horizon discounted formulation.

Next step

Introduce an infinite-horizon discounted Q function and derive its steady-state Q-HJB equation.

Why introduce discounting?

If we directly use the infinite-horizon cost

$$\int_0^\infty g(x(t), u(t))\, dt,$$

it may not converge.

Therefore, introduce a continuous-time discount rate

$$\gamma > 0.$$

The discounted cost is

$$\int_0^\infty e^{-\gamma t} g(x(t), u(t))\, dt.$$

Discounted infinite-horizon Q function

Consider the time-invariant augmented system

$$\dot{x}(t) = f(x(t), u(t)), \qquad \dot{u}(t) = a(t), \qquad u(t) \in \mathcal{U}, \quad \|a(t)\|_2 \leq M.$$

Define the discounted action-value function:

$$Q(x, u) \triangleq \inf_{a(\cdot)} \left\{ \int_0^\infty e^{-\gamma t} g(x(t), u(t))\, dt \;\middle|\; \begin{array}{l} \dot{x}(t) = f(x(t), u(t)), \\ \dot{u}(t) = a(t), \\ x(0) = x, \quad u(0) = u, \\ u(t) \in \mathcal{U}, \quad \|a(t)\|_2 \leq M \end{array} \right\}.$$

Relation between discounted J^* and Q

The corresponding discounted state-value function is

$$J^*(x) = \min_{u \in \mathcal{U}} Q(x, u).$$

That is, for a given state x, we choose the best initial control value u.

Action selection

If $Q(x, u)$ is known, the optimal control input can be selected by

$$u^*(x) \in \arg\min_{u \in \mathcal{U}} Q(x, u).$$

Connection to RL

This is why reinforcement learning often focuses on learning the action-value function Q, rather than directly solving for J^*.

Steady discounted Q-HJB equation

From the finite-horizon Q-HJB form, the infinite-horizon discounted setting gives

Discounted Q-HJB equation

$$\boxed{\nabla_x Q(x,u)^\top f(x,u) - M\|\nabla_u Q(x,u)\|_2 + g(x,u) - \gamma Q(x,u) = 0.}$$

- No $\partial_t Q$ term appears because the problem is time-invariant.
- The new term $-\gamma Q(x,u)$ comes from the exponential discount factor $e^{-\gamma t}$.

Interpretation of discounted Q-HJB

$$\nabla_x Q^\top f - M\|\nabla_u Q\|_2 + g - \gamma Q = 0.$$

- $\nabla_x Q^\top f$: effect of state evolution on future cost.
- $-M\|\nabla_u Q\|_2$: best possible local change of the control value u.
- $g(x,u)$: instantaneous cost.
- $-\gamma Q(x,u)$: discounting effect in infinite horizon.

Next step

The discounted Q-HJB equation is still a continuous-time PDE. To obtain Q-learning, we next derive a Bellman fixed-point relation after sampling and action discretization.

Evolution along the optimal augmented trajectory

Let $(x^*(t), u^*(t))$ be the optimal augmented trajectory:

$$\dot{x}^*(t) = f(x^*(t), u^*(t)), \qquad \dot{u}^*(t) = a^*(t), \qquad \|a^*(t)\|_2 \le M.$$

View $Q(x, u)$ as a composite function along this trajectory. By the chain rule,

$$\frac{d}{dt} Q(x^*(t), u^*(t)) = \nabla_x Q(x^*(t), u^*(t))^\top f(x^*(t), u^*(t)) + \nabla_u Q(x^*(t), u^*(t))^\top a^*(t).$$

Along the optimal trajectory,

$$\nabla_u Q(x^*, u^*)^\top a^* = \inf_{\|a\|_2 \le M} \nabla_u Q(x^*, u^*)^\top a = -M \|\nabla_u Q(x^*, u^*)\|_2.$$

A differential equation along the optimal trajectory

Therefore,

$$\frac{d}{dt}Q(x^*(t),u^*(t)) = \nabla_x Q(x^*,u^*)^\top f(x^*,u^*) - M\|\nabla_u Q(x^*,u^*)\|_2.$$

From the discounted Q-HJB equation,

$$\nabla_x Q^\top f - M\|\nabla_u Q\|_2 + g - \gamma Q = 0.$$

Thus the right-hand side can be replaced by

$$\gamma Q(x^*(t),u^*(t)) - g(x^*(t),u^*(t)).$$

So we get

$$\frac{d}{dt}Q(x^*(t),u^*(t)) = \gamma Q(x^*(t),u^*(t)) - g(x^*(t),u^*(t)).$$

From trajectory ODE to continuous-time Bellman equation

Rewrite the trajectory equation as

$$\frac{d}{dt}Q(x^*(t),u^*(t)) - \gamma Q(x^*(t),u^*(t)) = -g(x^*(t),u^*(t)).$$

Multiplying both sides by the integrating factor $e^{-\gamma t}$, we obtain

$$\frac{d}{dt}\left[e^{-\gamma t}Q(x^*(t),u^*(t))\right] = -e^{-\gamma t}g(x^*(t),u^*(t)).$$

Integrating from 0 to h,

$$e^{-\gamma h}Q(x^*(h),u^*(h)) - Q(x,u) = -\int_0^h e^{-\gamma t}g(x^*(t),u^*(t))\,dt.$$

Continuous-time Bellman equation

Rearranging the previous equation gives

$$Q(x,u) = \int_0^h e^{-\gamma t} g(x^*(t), u^*(t))\, dt + e^{-\gamma h} Q(x^*(h), u^*(h)), \qquad \forall h > 0.$$

Meaning

The current action-value equals the discounted cost accumulated over a short interval plus the discounted optimal action-value at the future augmented state.

Important

This is still a continuous-time relation along the optimal augmented trajectory. To obtain Q-learning, we next introduce sampling and action discretization.

Sampling and zero-order hold

Choose a sampling period $h > 0$.
On each interval $[kh, (k+1)h)$, use zero-order hold:

$$u(t) \equiv u_k, \qquad t \in [kh, (k+1)h).$$

Then the continuous-time system induces a discrete-time transition:

$$x_{k+1} = F_h(x_k, u_k),$$

where F_h is the sampled state transition map.
Further discretize the action space:

$$\mathcal{U}_d = \{u^{(1)}, u^{(2)}, \ldots, u^{(L)}\}.$$

After sampling and action discretization, we obtain a discrete optimal control problem.
Q-learning will solve this discrete approximation, not the continuous PDE directly.

Discrete one-step cost and discount factor

Define the one-step discounted stage cost:

$$c_h(x_k, u_k) \triangleq \int_0^h e^{-\gamma t} g(x(t), u_k)\, dt,$$

where $x(t)$ starts from x_k and evolves under the constant input u_k.

Define the discrete discount factor:

$$\Gamma \triangleq e^{-\gamma h}, \qquad 0 < \Gamma < 1.$$

If h is small, then

$$c_h(x_k, u_k) \approx h\, g(x_k, u_k).$$

The continuous cost accumulated in one sampling interval becomes the one-step cost used by the discrete Bellman equation.

Discrete Bellman fixed-point equation

The discrete optimal action-value function satisfies

$$Q_d^*(x_k, u_k) = c_h(x_k, u_k) + \Gamma \min_{u' \in \mathcal{U}_d} Q_d^*(x_{k+1}, u'),$$

where

$$x_{k+1} = F_h(x_k, u_k).$$

Equivalently, define the Bellman optimality operator $\mathcal{T}$:

$$(\mathcal{T}Q)(x_k, u_k) \triangleq c_h(x_k, u_k) + \Gamma \min_{u' \in \mathcal{U}_d} Q(x_{k+1}, u').$$

Then the optimal Q function is the fixed point:

$$Q_d^* = \mathcal{T}Q_d^*.$$

Fixed-point iteration

If the current estimate is Q_k, it may not satisfy

$$Q_k = \mathcal{T} Q_k.$$

The Bellman residual at (x_k, u_k) is

$$(\mathcal{T} Q_k)(x_k, u_k) - Q_k(x_k, u_k).$$

A natural fixed-point iteration is

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \left[(\mathcal{T} Q_k)(x_k, u_k) - Q_k(x_k, u_k) \right].$$

Interpretation

The estimate Q_k is moved toward its Bellman target. This is the basic idea behind Q-learning.

Model-free difficulty

If the model is known, we can compute

$$x_{k+1} = F_h(x_k, u_k), \qquad c_h(x_k, u_k),$$

and therefore compute $(\mathcal{T}Q_k)(x_k, u_k)$.

But in reinforcement learning, the model is usually unknown.

Instead, we interact with the system and observe a sample transition:

$$(x_k, u_k, c_k, x_{k+1}).$$

Here c_k is the observed one-step cost.

Key idea

Replace the exact Bellman target by a sample Bellman target computed from observed data.

Sample Bellman target

Given a sample transition

$$(x_k, u_k, c_k, x_{k+1}),$$

define the sample Bellman target:

$$y_k \triangleq c_k + \Gamma \min_{u' \in \mathcal{U}_d} Q_k(x_{k+1}, u').$$

This is a data-driven approximation of

$$(\mathcal{T} Q_k)(x_k, u_k).$$

Substituting y_k into the fixed-point iteration gives

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \left[y_k - Q_k(x_k, u_k) \right].$$

Q-learning update rule

Using the definition of y_k, we obtain

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \left[c_k + \Gamma \min_{u' \in \mathcal{U}_d} Q_k(x_{k+1}, u') - Q_k(x_k, u_k) \right].$$

Define the temporal-difference error:

$$\delta_k \triangleq c_k + \Gamma \min_{u' \in \mathcal{U}_d} Q_k(x_{k+1}, u') - Q_k(x_k, u_k).$$

Then

$$Q_{k+1}(x_k, u_k) = Q_k(x_k, u_k) + \alpha_k \delta_k.$$

What Q-learning is doing

- The continuous-time Q-HJB equation is a PDE.
- After sampling, zero-order hold, and action discretization, we obtain a discrete Bellman fixed-point equation:

$$Q_d^* = \mathcal{T} Q_d^*.$$

- Q-learning does not directly solve the Q-HJB PDE.
- Instead, it uses data samples to approximate the Bellman target and reduce the TD error.

Q-learning is a data-driven stochastic approximation to the Bellman fixed-point equation of the sampled and action-discretized optimal control problem.

Where DQN fits

- In tabular Q-learning, each state-action pair has its own Q value.
- For real control systems, the state is continuous:

$$x = [\theta, \dot{\theta}, \alpha, \dot{\alpha}]^\top.$$

- A table is no longer practical.
- DQN replaces the table by a neural network:

$$Q_\theta(x, u) \approx Q^*(x, u).$$

Summary

- We assembled the real rotary inverted pendulum circuit, including STM32, TB6612FNG, motor, potentiometer, encoder, and external DC power input.
- We uploaded STM32 firmware to observe raw potentiometer and encoder values through the Arduino IDE Serial Monitor.
- We performed a safe motor test through STM32 and TB6612FNG while continuing to record sensor and PWM data.
- Finally, we connected Bellman's optimality principle to HJB, LQR, MPC, Q-HJB, and Q-learning.

Take-home message

Lecture 8 connects three layers of AI-enabled control engineering: real hardware implementation, reliable data acquisition, and the optimal-control foundation behind learning-based control.