AI Enabled Control Engineering

Lecture 7: Rotary Inverted Pendulum Modeling, Hardware System Introduction, and STM32 Env Setup

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

- Reinforcement learning overview.
- Q-learning and why Q-tables fail for CartPole.
- Deep Q-Network (DQN): replay buffer and target network.
- Practical workflow of CartPole training and testing.

Lecture 7

- Why move from CartPole to the rotary inverted pendulum.
- Hardware system introduction: motor, sensors, motor driver and STM32F446RE microcontroller.
- Nonlinear dynamics modelling by the Lagrange method.
- Arduino IDE setup, flashing and debugging for STM32F446RE.

Why move from CartPole to the rotary inverted pendulum?

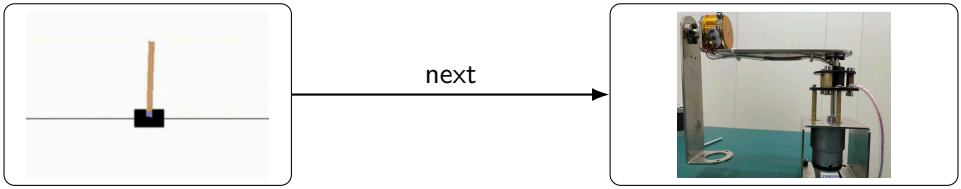
CartPole is a very useful first benchmark, but it is still a software environment.

The rotary inverted pendulum is more suitable for the next stage because

- it is a portable experimental platform,
- it is nonlinear, coupled, and underactuated,

So the course now moves from a toy benchmark to a real hardware system.

From simulation to real hardware



CartPole

Rotary Inverted Pendulum

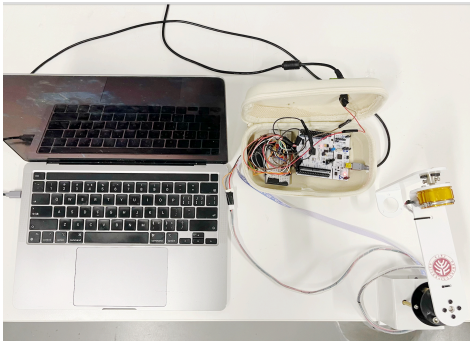
The real system introduces several new issues:

- sensors ,
- motor driver and PWM,
- embedded code,
- communication delay.

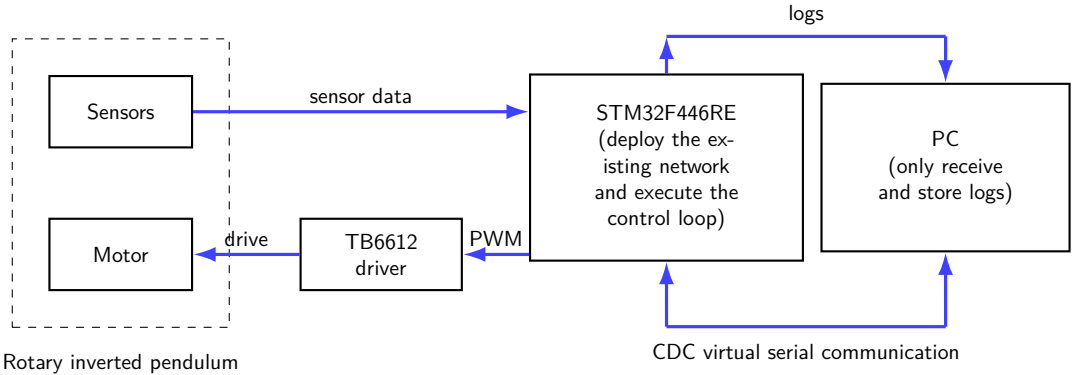
Overall hardware system

Typical hardware modules:

- rotary arm and vertical pendulum,
- angle sensors,
- DC motor and TB6612 driver,
- STM32F446RE control board,
- host computer for data logging.



Hardware architecture



Motor

The horizontal arm of the rotary inverted pendulum is driven by a DC motor with a gear reduction mechanism.

Important motor parameters:

- **Rated voltage Rated torque**
- **Stall torque**: maximum torque when the motor shaft is blocked.
- **Gear ratio**: ratio between motor-shaft rotation and output-shaft rotation.

$$\omega_{\text{out}} \approx \frac{\omega_{\text{motor}}}{N}, \qquad \tau_{\text{out}} \approx \eta N \tau_{\text{motor}}.$$

Motor used in this system:

Rated voltage	12 V
Rated current	0.36 A
Rated power	4.32 W
Rated torque	0.06468 N · m
Stall current	2.8 A
Stall torque	0.6468 N · m
No-load speed	11000 rpm
Output speed	549 ± 15 rpm

Sensors

The rotary inverted pendulum needs at least two angle-related measurements:

- **arm angle** θ,
- **pendulum angle** α.

Typical implementations include

- incremental encoder for the rotary arm,
- potentiometer or angle sensor for the pendulum.

From these measurements, we can further estimate the angular velocities

$$\dot{\theta}, \qquad \dot{\alpha}$$

using an observer.

Sensor: Magnetic encoder

A magnetic encoder measures rotation by detecting the change of magnetic field.

In this system, the motor tail is equipped with an incremental magnetic encoder.

When the motor shaft rotates, the magnetic encoder generates pulse signals.

- The number of pulses reflects angular displacement.
- The pulse density reflects rotational speed.
- Two phase-shifted signals are used to determine direction.

Therefore, the encoder provides the basic measurement for the rotary arm angle:

$$\text{encoder pulses} \quad \Longrightarrow \quad \theta.$$

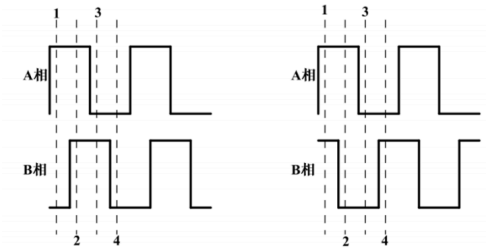
A/B phase signals and rotation direction

An incremental encoder usually outputs two square-wave signals: A phase and B phase.

- A phase and B phase have the same frequency.
- Their phase difference is approximately 90°.
- The leading phase determines the rotation direction.

For example:

- if A phase leads B phase, the shaft rotates in one direction;
- if B phase leads A phase, the shaft rotates in the opposite direction.



From encoder pulses to rotation angle

For this motor encoder, one motor-shaft revolution generates 13 pulses.

The gear ratio is approximately $20:1$, meaning that the motor shaft rotates 20 turns while the output shaft rotates 1 turn.

Therefore, one output-shaft revolution generates $13 \times 20 = 260$ encoder pulse periods.

Because both A and B phases are counted on rising and falling edges, each pulse period provides four countable edges:

$$260 \times 4 = 1040.$$

Thus,

$$360^\circ \Longleftrightarrow 1040 \text{ counts}, \qquad 1 \text{ count} \Longleftrightarrow \frac{360^\circ}{1040}.$$

Firmware: counting encoder pulses

In the STM32 firmware, the encoder A/B signals are connected to two external interrupt pins. The variable g_enc stores the accumulated encoder count.

```
static const int PIN_ENC_A = 2;
static const int PIN_ENC_B = 3;
static volatile int32_t g_enc = 10000;
void setup() {
  pinMode(PIN_ENC_A, INPUT);
  pinMode(PIN_ENC_B, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN_ENC_A),
                  isr_enc_a, CHANGE);
  attachInterrupt(digitalPinToInterrupt(PIN_ENC_B),
                  isr_enc_b, CHANGE);
}
```

Each rising or falling edge of A/B triggers an interrupt, so the firmware can count pulses without continuously polling the encoder in the main loop.

Firmware: direction judgment by A/B phase

Whenever A or B changes its logic level, the corresponding interrupt service routine is called.

```
void isr_enc_a() {
  bool A = digitalRead(PIN_ENC_A);
  bool B = digitalRead(PIN_ENC_B);
  if (A) g_enc += (B ? 1 : -1);
  else g_enc += (B ? -1 : 1);
}
void isr_enc_b() {
  bool A = digitalRead(PIN_ENC_A);
  bool B = digitalRead(PIN_ENC_B);
  if (B) g_enc += (A ? -1 : 1);
  else g_enc += (A ? 1 : -1);
}
```

This logic uses the current A/B levels to decide whether the count should increase or decrease.

Firmware: from encoder count to arm angle

When the command `GO` is received, the current encoder count is saved as the zero position:

```
if (t == "GO") {
  noInterrupts();
  ENC_ZERO = g_enc; ...
  interrupts();
}
```

During each control tick, the firmware reads the current encoder count and converts it to the arm angle:

```
int32_t enc = g_enc;
int32_t enc0 = ENC_ZERO;
g_theta_meas = (float)(enc - enc0) * THETA_SCALE;
```

Sensor: potentiometer

The pendulum angle α is measured by a potentiometer connected to the STM32 analog input pin.

A potentiometer is a variable resistor. As the pendulum rotates, its resistance changes and produces a different analog voltage.

The STM32 ADC converts this analog voltage into a digital value.

```
static const int PIN_POT = A0;
static const int32_t POT_MOD = 4096;

pinMode(PIN_POT, INPUT);
analogReadResolution(12);
```

Because 12-bit ADC is used, the raw potentiometer value is in the range

$$0 \leq pot\_raw \leq 4095.$$

From potentiometer value to pendulum angle

Two calibration values are used in the firmware:

```
static volatile int32_t POT_ZERO = 3090; // upright position
static volatile int32_t POT_DOWN_RAW = 980; // hanging-down position
```

The firmware reads the potentiometer in each control tick:

```
int pot = analogRead(PIN_POT);
g_pot_raw = (int16_t)pot;

float alpha_branch = alphaFromPotRawBranchCutAtDown(pot);
g_alpha_meas = alpha_branch;
```

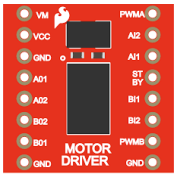
$$\text{pendulum angle} \rightarrow \text{resistance} \rightarrow \text{voltage} \rightarrow \textit{pot\_raw} \rightarrow \alpha.$$

Here, `POT_ZERO` corresponds to the upright position, and `POT_DOWN_RAW` is used to define the branch cut near the hanging-down position.

Motor driver: TB6612

The STM32 GPIO pins can only output low-power logic signals, so they cannot drive the DC motor directly.

The TB6612 motor driver is used as a power amplifier between the STM32 and the motor.



- **VM**: motor power supply.
- **VCC**: logic power supply.
- **GND**: common ground.
- **AO1, AO2**: motor output terminals.

The TB6612 receives control signals from STM32 and provides higher current to the DC motor.

TB6612 control pins in firmware

For one DC motor, the STM32 mainly controls three types of signals:

- **PWMA**: controls motor speed / torque level by PWM.
- **AI1, AI2**: control motor rotation direction.
- **STBY**: enables or disables the motor driver.

In the firmware, the corresponding pins are:

```
static const int PIN_IN1 = 4; // AI1
static const int PIN_IN2 = 5; // AI2
static const int PIN_STBY = 7; // STBY
static const int PIN_PWM = 10; // PWMA
```

STM32 PWM + direction $\rightarrow$ TB6612 $\rightarrow$ DC motor $\rightarrow$ rotary arm.

STM32F446RE development board

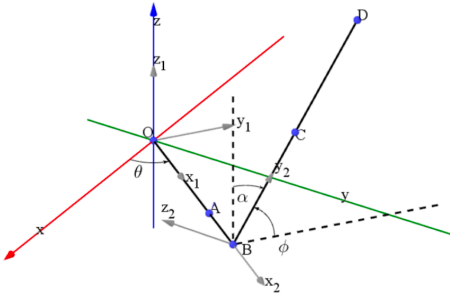
The STM32F446RE is the real-time controller of the rotary inverted pendulum system.

Main tasks:

- reading potentiometer by ADC;
- counting encoder pulses by external interrupts;
- generating PWM for the TB6612 driver;
- executing the control loop at a fixed frequency;
- communicating with the host computer through USB serial.

It connects sensing, computation, and actuation in the embedded control loop.

Physical model of the rotary inverted pendulum



The rotary inverted pendulum consists of two rigid bodies:

- horizontal rotary arm OB,
- vertical pendulum BD.

We define two generalized coordinates:

$$q_1 = \theta, \qquad q_2 = \alpha.$$

- θ: rotary arm angle;
- α: pendulum angle relative to the upright position.

State variables and modelling objective

The system state is defined as

$$x = \begin{bmatrix} \theta & \dot{\theta} & \alpha & \dot{\alpha} \end{bmatrix}^\top.$$

The goal of physical modelling is to derive the dynamic relationship between the motor input and the system motion:

$$u \quad \Longrightarrow \quad \theta,\ \dot{\theta},\ \alpha,\ \dot{\alpha}.$$

The rotary inverted pendulum is

- nonlinear,
- coupled,
- unstable around the upright equilibrium.

Therefore, we first build a nonlinear model and then linearize it around the upright position.

Lagrange equation

The dynamics are derived using the Lagrange equation:

$$\frac{d}{dt}\left(\frac{\partial T}{\partial \dot{q}_i}\right) - \frac{\partial T}{\partial q_i} + \frac{\partial U}{\partial q_i} = Q_i, \qquad i = 1, 2.$$

Here,

- T is the total kinetic energy,
- U is the potential energy,
- Q_i is the generalized force corresponding to q_i.

$$q_1 = \theta, \qquad q_2 = \alpha.$$

The two Lagrange equations give the coupled dynamics of the rotary arm and the pendulum.

Physical parameters and generalized forces

Symbol	Meaning
m_1	mass of the rotary arm OB
m_2	mass of the pendulum BD
l_1	distance from O to the center of mass of arm OB
l_{21}	distance from joint B to the center of mass of pendulum BD
I_{1z}	moment of inertia of arm OB about its z axis
I_{2x}, I_{2y}, I_{2z}	moments of inertia of pendulum BD about its body axes
c_θ	viscous damping coefficient of the rotary arm
c_α	viscous damping coefficient of the pendulum
τ	motor output torque
$Q_\theta = \tau - c_\theta \dot{\theta}$	generalized force corresponding to θ
$Q_\alpha = -c_\alpha \dot{\alpha}$	generalized force corresponding to α

Kinetic and potential energy

The total kinetic energy is composed of the arm part and the pendulum part:

$$T = T_{OB} + T_{BD}.$$

The rotary arm contributes rotational kinetic energy:

$$T_{OB} = \frac{1}{2} m_1 l_1^2 \dot{\theta}^2 + \frac{1}{2} I_{1z} \dot{\theta}^2.$$

The pendulum contributes both translational and rotational kinetic energy:

$$T_{BD} = \frac{1}{2} m_2 v_C^2 + \frac{1}{2} \left(I_{2x} \dot{\alpha}^2 + I_{2y} \dot{\theta}^2 \sin^2 \alpha + I_{2z} \dot{\theta}^2 \cos^2 \alpha \right).$$

The potential energy comes from gravity:

$$U = -m_2 g l_{21} (1 - \sin \phi).$$

Nonlinear dynamic equations

After substituting T and U into the Lagrange equations, the nonlinear coupled dynamics can be obtained.

Using $\alpha = \frac{\pi}{2} - \phi$, the equations can be written as

$$\begin{aligned}
&\left(m_1 l_1^2 + I_{1z} + m_2 l_1^2 + m_2 l_{21}^2 \sin^2\alpha + I_{2y}\cos^2\alpha + I_{2z}\sin^2\alpha\right)\ddot{\theta} \\
&\quad + (m_2 l_1 l_{21}\cos\alpha)\ddot{\alpha} - (m_2 l_1 l_{21}\sin\alpha)\dot{\alpha}^2 \\
&\quad - 2\sin\alpha\cos\alpha\left(I_{2y} - I_{2z} - m_2 l_{21}^2\right)\dot{\alpha}\dot{\theta} = Q_\theta, \\
&\quad - (m_2 l_1 l_{21}\cos\alpha)\ddot{\theta} + \sin\alpha\cos\alpha\left(I_{2z} - I_{2y} + m_2 l_{21}^2\right)\dot{\theta}^2 \\
&\quad\quad - (I_{2x} + m_2 l_{21}^2)\ddot{\alpha} + m_2 g l_{21}\sin\alpha = Q_\alpha.
\end{aligned}$$

Linearization around the upright equilibrium

We linearize the nonlinear dynamics around the upright equilibrium:

$$\theta = 0, \qquad \alpha = 0.$$

Near the upright position, the small-angle approximations are

$$\sin\alpha \approx 0, \qquad \cos\alpha \approx 1, \qquad \dot{\alpha}^2 \approx 0, \qquad \dot{\theta}^2 \approx 0.$$

After neglecting higher-order nonlinear terms, the nonlinear equations become

$$\left(m_1 l_1^2 + I_{1z} + m_2 l_1^2 + I_{2y}\right)\ddot{\theta} + m_2 l_1 l_{21}\ddot{\alpha} = Q_\theta,$$

$$-m_2 l_1 l_{21}\ddot{\theta} - \left(I_{2x} + m_2 l_{21}^2\right)\ddot{\alpha} + m_2 g l_{21}\alpha = Q_\alpha.$$

Linearized model

Define four constants:

$$C_1 = m_1 l_1^2 + I_{1z} + m_2 l_1^2 + I_{2y}, \qquad C_2 = m_2 l_1 l_{21},$$

$$C_3 = I_{2x} + m_2 l_{21}^2, \qquad C_4 = m_2 g l_{21}.$$

Then the linearized dynamics can be written compactly as

$$C_1 \ddot{\theta} + C_2 \ddot{\alpha} = Q_\theta,$$

$$-C_2 \ddot{\theta} - C_3 \ddot{\alpha} + C_4 \alpha = Q_\alpha.$$

Where

$$Q_\theta = \tau - c_\theta \dot{\theta}, \qquad Q_\alpha = -c_\alpha \dot{\alpha}.$$

State-space form

After solving the two linearized second-order equations for $\ddot{\theta}$ and $\ddot{\alpha}$, the system can be written as

$$\dot{x} = Ax + B\tau.$$

The symbolic form is

$$\frac{d}{dt}x = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & d_1 & a_1 & d_2 \\ 0 & 0 & 0 & 1 \\ 0 & d_3 & a_2 & d_4 \end{bmatrix} x + \begin{bmatrix} 0 \\ b_1 \\ 0 \\ b_2 \end{bmatrix} \tau.$$

State-space form

For the state vector

$$x = \begin{bmatrix} \theta & \dot{\theta} & \alpha & \dot{\alpha} \end{bmatrix}^\top,$$

the linearized model can be written as

$$\dot{x} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & d_1 & a_1 & d_2 \\ 0 & 0 & 0 & 1 \\ 0 & d_3 & a_2 & d_4 \end{bmatrix} x + \begin{bmatrix} 0 \\ b_1 \\ 0 \\ b_2 \end{bmatrix} \tau.$$

where

$$a_1 = -\frac{C_2 C_4}{C_1 C_3 - C_2^2}, \quad a_2 = \frac{C_1 C_4}{C_1 C_3 - C_2^2}, \quad b_1 = \frac{C_3}{C_1 C_3 - C_2^2}, \quad b_2 = -\frac{C_2}{C_1 C_3 - C_2^2},$$

$$d_1 = -\frac{C_3 c_\theta}{C_1 C_3 - C_2^2}, \quad d_2 = \frac{C_2 c_\alpha}{C_1 C_3 - C_2^2}, \quad d_3 = \frac{C_2 c_\theta}{C_1 C_3 - C_2^2}, \quad d_4 = -\frac{C_1 c_\alpha}{C_1 C_3 - C_2^2}.$$

From motor torque to PWM command

The mechanical model uses motor torque τ as the input, but the real system sends a PWM command u through STM32 and TB6612.

Therefore, we need a DC motor model to connect the implemented command to the physical torque:

$$u \quad \Longrightarrow \quad \tau.$$

For a DC motor,

$$V_{\text{emf}} = k_i \omega, \qquad I = \frac{V_s - V_{\text{emf}}}{R}, \qquad \tau = k_t I.$$

where V_{emf} is the back electromotive force, ω is motor angular velocity, V_s is the applied voltage, R is armature resistance, I is motor current, and k_t, k_i are motor constants.

PWM-to-torque relation

The motor voltage is approximately proportional to the PWM command: $V_s = k_u u$.
Substituting this into the DC motor model gives

$$\tau = k_t \frac{V_s - k_i \omega}{R} = \frac{k_t k_u}{R} u - \frac{k_t k_i}{R} \omega.$$

Therefore,

$$\tau = K_u u - K_\omega \dot{\theta},$$

where

$$K_u = \frac{k_t k_u}{R}, \qquad K_\omega = \frac{k_t k_i}{R}.$$

Substituting PWM input into the state-space model

Before considering the motor electrical model, the linearized mechanical model is

$$\dot{x} = A_\tau x + B_\tau \tau,$$

where

$$A_\tau = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & d_1 & a_1 & d_2 \\ 0 & 0 & 0 & 1 \\ 0 & d_3 & a_2 & d_4 \end{bmatrix}, \qquad B_\tau = \begin{bmatrix} 0 \\ b_1 \\ 0 \\ b_2 \end{bmatrix}.$$

From the previous slide,

$$\tau = K_u u - K_\omega \dot{\theta}.$$

Therefore,

$$\dot{x} = A_\tau x + B_\tau \left(K_u u - K_\omega \dot{\theta} \right).$$

PWM-input state-space model

Since the state vector is

$$x = \begin{bmatrix} \theta & \dot{\theta} & \alpha & \dot{\alpha} \end{bmatrix}^\top,$$

the term $\dot{\theta}$ is the second state variable.
Thus, the back-EMF term $-K_\omega B_\tau \dot{\theta}$ is absorbed into the second column of the state matrix:

$$\dot{x} = A_1 x + B_1 u,$$

where

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & d_1 - K_\omega b_1 & a_1 & d_2 \\ 0 & 0 & 0 & 1 \\ 0 & d_3 - K_\omega b_2 & a_2 & d_4 \end{bmatrix}, \qquad B_1 = \begin{bmatrix} 0 \\ K_u b_1 \\ 0 \\ K_u b_2 \end{bmatrix}.$$

State-space form

After substituting the physical parameters, the final linearized model is

$$\dot{x}(t) = A_1 x(t) + B_1 u(t),$$

where $u(t)$ is the motor PWM command, and

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & -11.3205 & -49.9955 & 0.4820 \\ 0 & 0 & 0 & 1 \\ 0 & 16.9206 & 171.2602 & -1.6510 \end{bmatrix}, \qquad B_1 = \begin{bmatrix} 0 \\ 0.8171 \\ 0 \\ -1.2213 \end{bmatrix}.$$

This model describes the local dynamics of the rotary inverted pendulum near the upright equilibrium.

Arduino IDE setup for STM32F446RE

Arduino IDE can be used as a lightweight development environment for STM32F446RE.

The basic workflow is:

1. install Arduino IDE 2,
2. add the STM32 board package URL,
3. install `STM32 MCU based boards`,
4. select the correct STM32F446RE board,
5. choose the upload method,
6. compile and upload the firmware.

This allows us to write Arduino-style sketches while still running them on the STM32 microcontroller.

Install STM32 board package

Open Arduino IDE:

`File` → `Preferences` → `Additional Boards Manager URLs`

Add the STM32duino package index URL:

```
https://github.com/stm32duino/BoardManagerFiles/raw/main/
    package_stmicroelectronics_index.json
```

Then open:

`Tools` → `Board` → `Boards Manager`

Search for

```
STM32 MCU based boards
```

and install the package.

Select the STM32F446RE board

After installing the STM32 board package, select the board in Arduino IDE.

For the STM32F446RE Nucleo board, the typical selection path is:

$$\texttt{Tools} \rightarrow \texttt{Board} \rightarrow \texttt{STM32 MCU based boards} \rightarrow \texttt{Nucleo-64}$$

Then select the board part number:

$$\texttt{Board part number} \rightarrow \texttt{Nucleo F446RE}$$

Other useful settings are usually:

- **Upload method**: `OpenOCD STLink` .
- **Port**: select the USB serial port of the board.

Upload firmware to STM32F446RE

Connect the STM32F446RE board to the computer through USB.

1. select the correct board , board part number and upload method,
2. select the correct port,
3. click **Verify** to compile and click **Upload** to flash the firmware.

A minimal test program is:

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
}
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);
  delay(500);
  digitalWrite(LED_BUILTIN, LOW);
  delay(500);
}
```

Serial monitor test

After uploading the firmware, we can use Serial Monitor to check communication. Open:

`Tools` → `Serial Monitor`

Set the baud rate to the same value used in the firmware, for example:

```
void setup() {
  Serial.begin(115200);
}
void loop() {
  Serial.println("STM32 serial OK");
  delay(500);
}
```

If the setup is correct, the Serial Monitor should repeatedly print:

```
STM32 serial OK
```

Flashing and debugging checklist

If upload or serial communication fails, check the following items:

- Is the STM32 board package installed?
- Is `Nucleo F446RE` selected as the board part number?
- Is the correct upload method selected?
- Is the correct USB port selected?
- Is the USB cable a data cable rather than a charging-only cable?
- Is the Serial Monitor baud rate the same as `Serial.begin()`?
- Is another program already occupying the serial port?

Summary

In this lecture, we moved from a software benchmark to a real rotary inverted pendulum platform.

- **Hardware system.**
- **Sensing and actuation.**
- **Physical modelling.**
- **PWM-input state-space model.** the motor torque model was combined with the mechanical model to obtain

$$\dot{x}(t) = A_1 x(t) + B_1 u(t),$$

  where $u(t)$ is the PWM command.
- **STM32 development workflow.**