ELECROW

All-in-one Starter Kit for Pico 2 User Manual

Arduino Version

20+

Lessons

17

Modules in one

www.elecrow.com

Table of contents

Introduction

Welcome to the User Manual for the All-in-one Starter Kit for PICO 2. Let's begin our journey into the world of the PICO 2 development board and its sensors.

Rest assured, this development board is equipped with 21 courses that are designed to be progressively challenging, engaging, and thought-provoking. These courses will guide you step-by-step through the essential knowledge. Here, you will become familiar with electronic modules, hone your logical thinking skills, enhance your creative design capabilities, and implement the functionality of these modules through programming.

The learning process starts with understanding how to install the programming software, followed by an introduction to the PICO 2 development board and its various sensors. You will then delve into the programming functions of these sensors and the programming language they utilize, ultimately learning how to apply these sensors in practical applications. Each step is meticulously explained, making it easy for beginners to quickly grasp C/C++ programming.

The All-in-one Starter Kit for PICO 2 includes 17 electronic modules, each with its unique characteristics and functions, specifically designed for beginners and an ideal choice for getting started. For example, the light sensor allows beginners to control real-world lighting devices through programming.

In summary, by working with this development board, you will learn the fundamental knowledge and principles of sensors, understand important concepts such as digital and analog signals, analog-to-digital conversion, and programming logic, and master the use of some complex electronic modules. Most importantly, through PICO 2 programming, you will further enhance your logical thinking skills.

For the programming software, we will utilize the Arduino IDE. Arduino IDE is an easy-to-use open-source platform and one of the best choices for learning programming.

Getting Started

Installing Arduino IDE

Download Arduino in Windows system

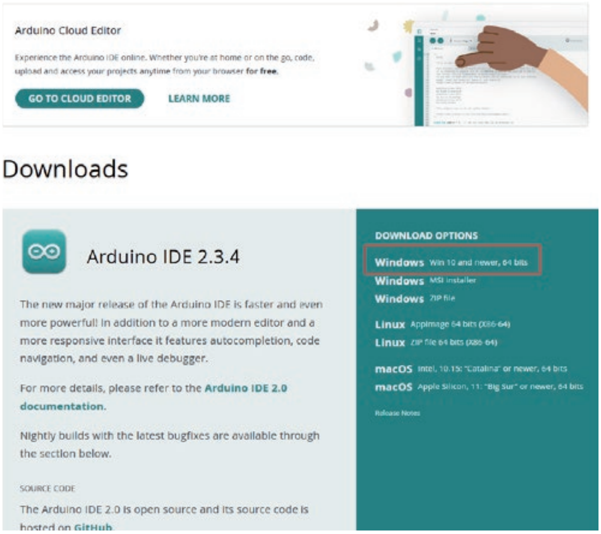
• STEP 1:

Login to Arduino official website, download Arduino**.**

Arduino official website: https://www.arduino.cc/en/software/

• STEP 2:

Select your computer's corresponding system to download, such as Window system.



Arduino Cloud Editor

Experience the Arduino IDE online. Whether you're at home or on the go, code, upload and access your projects anytime from your browser **for free**.

GO TO CLOUD EDITOR LEARN MORE

Downloads

Arduino IDE 2.3.4

The new major release of the Arduino IDE is faster and even more powerful! In addition to a more modern editor and a more responsive interface it features autocompletion, code navigation, and even a live debugger.

For more details, please refer to the **Arduino IDE 2.0 documentation**.

Nightly builds with the latest bugfixes are available through the section below.

SOURCE CODE

The Arduino IDE 2.0 is open source and its source code is hosted on **GitHub**.

DOWNLOAD OPTIONS

Windows Win 10 and newer, 64 bits
Windows MSI installer
Windows ZIP file

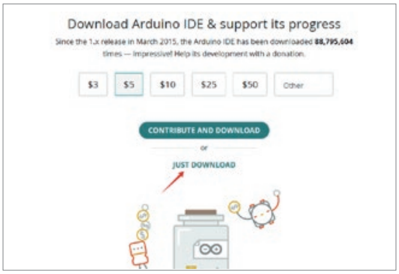
Linux AppImage 64 bits (X86-64)
Linux ZIP file 64 bits (X86-64)

macOS Intel, 10.15: "Catalina" or newer, 64 bits
macOS Apple Silicon, 11: "Big Sur" or newer, 64 bits

Release Notes

• STEP 3:

Click ***JUST DOWNLOAD*** and select the save location to start the download.



Download Arduino IDE & support its progress

Since the 1.x release in March 2015, the Arduino IDE has been downloaded **88,795,604** times — impressive! Help its development with a donation.

$3 $5 $10 $25 $50 Other

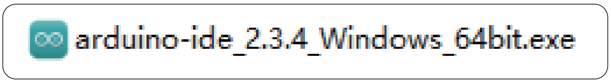
CONTRIBUTE AND DOWNLOAD

or

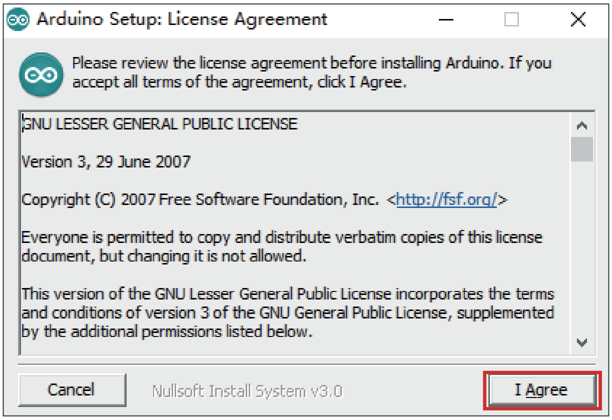
JUST DOWNLOAD

• **STEP 4:**

1. When installing Arduino, please locate the executable file with the .exe extension within the folder where you previously saved, which is the Arduino installation package.


arduino-ide_2.3.4_Windows_64bit.exe

2. After double-clicking the installation package, this page will appear. Click on '***I Agree*** '.



Please review the license agreement before installing Arduino. If you accept all terms of the agreement, click I Agree.

GNU LESSER GENERAL PUBLIC LICENSE

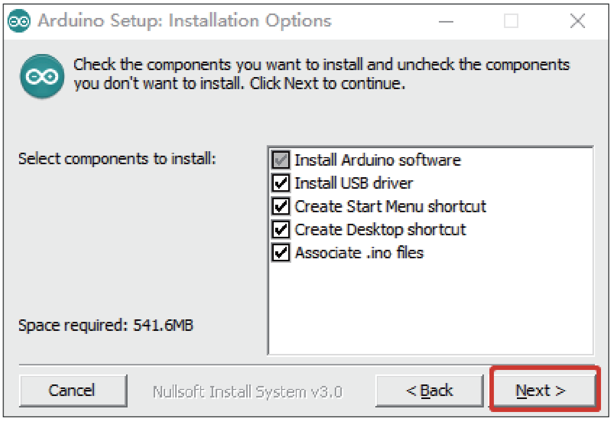
Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

3. Check all options by default and click ***Next***.


Arduino Setup: Installation Options

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

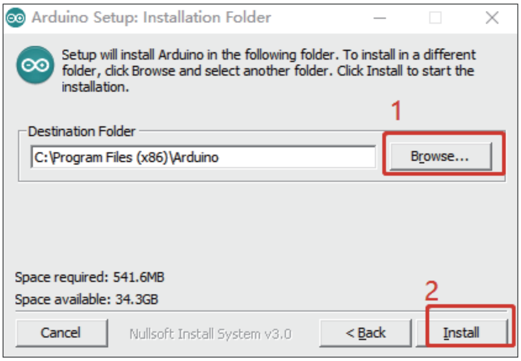
Select components to install:

- ✓ Install Arduino software
- ✓ Install USB driver
- ✓ Create Start Menu shortcut
- ✓ Create Desktop shortcut
- ✓ Associate .ino files

Space required: 541.6MB

Cancel
Nullsoft Install System v3.0
< Back
Next >

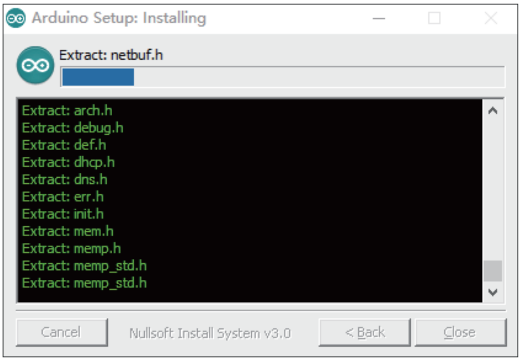
4. Click on '***Browse***' to select the installation location, it is recommended to install it on any drive other than the C: drive. Then click '***Install*** '.


Arduino Setup: Installation Folder

Setup will install Arduino in the following folder. To install in a different folder, click Browse and select another folder. Click Install to start the installation.

Destination Folder: C:\Program Files (x86)\Arduino

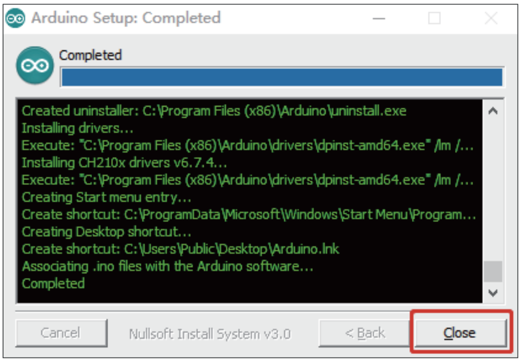
Space required: 541.6MB
Space available: 34.3GB


Arduino Setup: Installing

Extract: netbuf.h

```
Extract: arch.h
Extract: debug.h
Extract: def.h
Extract: dhcp.h
Extract: dns.h
Extract: err.h
Extract: init.h
Extract: mem.h
Extract: memp.h
Extract: memp_std.h
Extract: memp_std.h
```

5. Installation Complete,click '***Close***'.


Arduino Setup: Completed

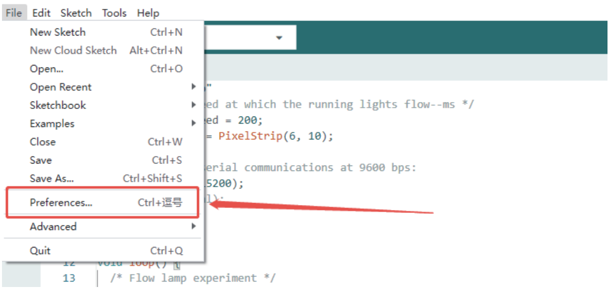
Completed

```
Created uninstaller: C:\Program Files (x86)\Arduino\uninstall.exe
Installing drivers...
Execute: "C:\Program Files (x86)\Arduino\drivers\dpinst-amd64.exe" /lm /...
Installing CH210x drivers v6.7.4...
Execute: "C:\Program Files (x86)\Arduino\drivers\dpinst-amd64.exe" /lm /...
Creating Start menu entry...
Create shortcut: C:\ProgramData\Microsoft\Windows\Start Menu\Program...
Creating Desktop shortcut...
Create shortcut: C:\Users\Public\Desktop\Arduino.lnk
Associating .ino files with the Arduino software...
Completed
```

Download PICO development board options

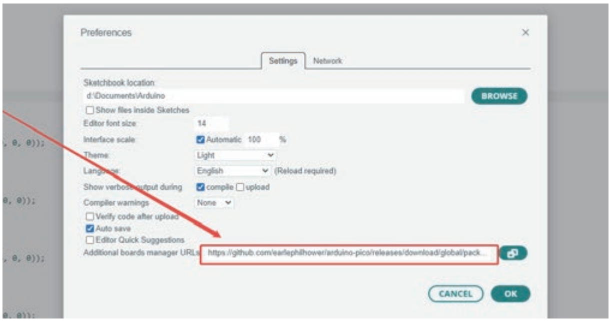
After waiting for the Arduino IDE installation to complete, open the Arduino IDE.

1. Find the settings in Arduino IDE



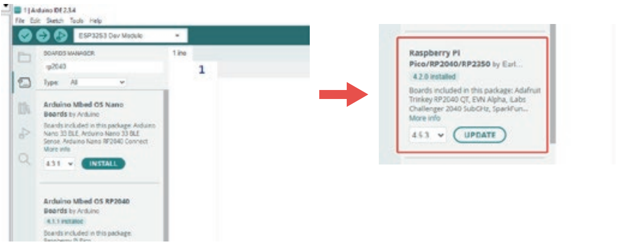
```
ed at which the running lights flow--ms */
ed = 200;
= PixelStrip(6, 10);

erial communications at 9600 bps:
5200);

void loop() {
/* Flow lamp experiment */
```

2. Put this link up：
https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json



Sketchbook location: d:\Documents\Arduino
Show files inside Sketches
Editor font size: 14
Interface scale: ✓ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during: ✓ compile, upload
Compiler warnings: None
Verify code after upload
✓ Auto save
Editor Quick Suggestions
Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

3. Search for rp2040(If you have previously downloaded and installed other versions of RP2040, please uninstall the previous version thoroughly before reinstalling version 4.2.0.)



rp2040
Type: All

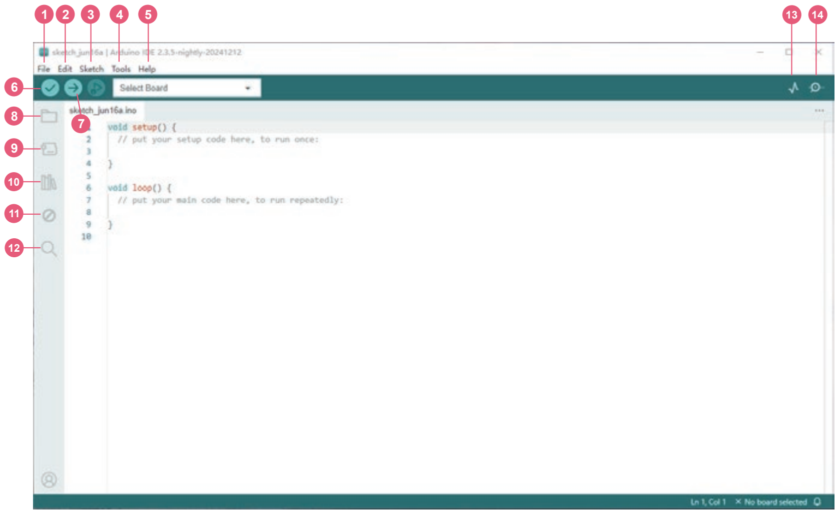
Arduino Mbed OS Nano Boards by Arduino
Boards included in this package: Arduino Nano 33 BLE, Arduino Nano 33 BLE Sense, Arduino Nano RP2040 Connect
More info
4.3.1

Arduino Mbed OS RP2040 Boards by Arduino
4.1.1 installed
Boards included in this package: Raspberry Pi Pico

Raspberry Pi Pico/RP2040/RP2350 by Earl...
4.2.0 installed
Boards included in this package: Adafruit Trinkey RP2040 QT, EVN Alpha, iLabs Challenger 2040 SubGHz, SparkFun...
More info
4.5.3

This way, we can run the code we will write later on the All-in-one Starter Kit for Pico2.

Arduino IDE Introduction



sketch_jun16a | Arduino IDE 2.3.5-nightly-20241212

sketch_jun16a.ino

```
void setup() {
  // put your setup code here, to run once:

}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Ln 1, Col 1 × No board selected

1 Flie: This lets you create, open, save, and manage sketch files, access sample code, adjust editor preferences, export compiled files, and exit the Arduino IDE—making it easy to handle project files and configure your workspace.

2 Edit: This is for editing your code, including undo, redo, cut, copy, paste, find and replace, select all, as well as commenting and uncommenting code—helping you modify your program quickly and efficiently.

3 Sketch: This handles compiling and uploading your code, allowing you to verify syntax, upload programs to your board, manage libraries, and export build results—streamlining your development and debugging process.

4 Tools: Choose your board model, serial port, and programmer, open the serial monitor and plotter, manage libraries, and check board details—essential for hardware setup and debugging.

5 **Help:** Provide official Arduino reference materials, FAQs, troubleshooting guides, and software version information to help users learn, use, and resolve issues they may encounter during development.

6 **Verify:** Compile the Arduino code to check for syntax errors and issues without uploading it, ensuring the code is correct and executable.

7 **Upload:** Upload the compiled code to the Arduino board to run and test the program on the actual hardware.

8 **Sketchbook:** Used for managing and quickly accessing all saved Arduino sketches, making it easy for users to open, edit, and organize their project code.

9 **Boards manager:** Used to install, update, and manage various Arduino board support packages, extending the IDE’s compatibility with different hardware.

10 **Library manager:** Used to search for, install, and manage Arduino libraries, helping users easily integrate various functional modules and streamline the development process.

11 **Debug:** Used to assist with code debugging by printing messages and errors through the serial port, helping identify issues in the program and improving development efficiency.

12 **Search:** The Search feature allows quick find-and-replace within the code, making it easier to locate and edit specific content and boosting editing efficiency.

13 **Serial Plotter:** Used to plot numerical data sent from the Arduino board via the serial port in real time, helping users visually analyze sensor readings and variable changes.

14 **Serial Monitor:** Used for serial communication with the Arduino board, allowing real-time sending and displaying of text to facilitate debugging and monitoring program status.

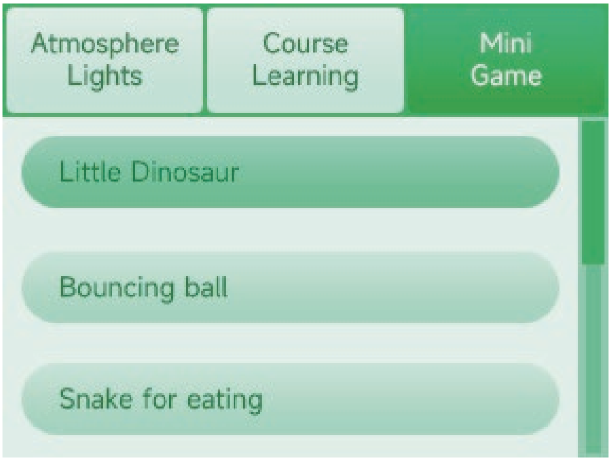
Course Usage Instructions

The course firmware is provided for use only. The function description and download firmware address in the firmware are as follows:

Source code address:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/factory_sourcecode/All_in_one_Starter_Kit_for_Pico2_SPI_Curriculum_V02_20250114

Game Usage Tutorial



Game Tutorials

- **Game 1: Little Dinosaur Game**

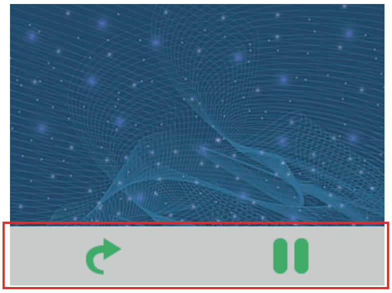
How to Play: Control the little dinosaur to jump and avoid obstacles by tapping the screen.

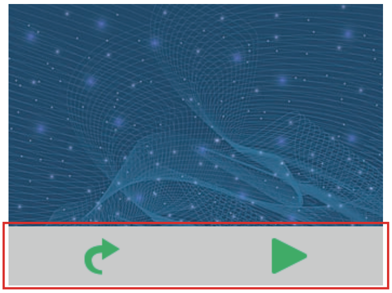
- **Game 2: Bouncing Ball Game**

How to Play: Use the left and right buttons to move the paddle left or right and bounce the ball.

- **Game 3: Snake Game**

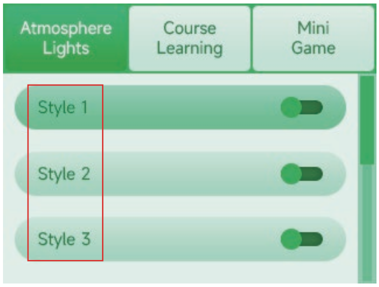
How to Play: Use the up, down, left, and right buttons to control the movement of the snake and eat the fruits.





For the above games, swiping up on the screen will bring up the menu. Clicking the "**Return**" button will exit to the factory UI interface. Clicking the "**Start/Stop**" button can control the game status. Once the "**Start**" button is clicked, the menu interface will automatically exit.

The functions are the implementation functions for the four types of lighting effects in the UI interface.



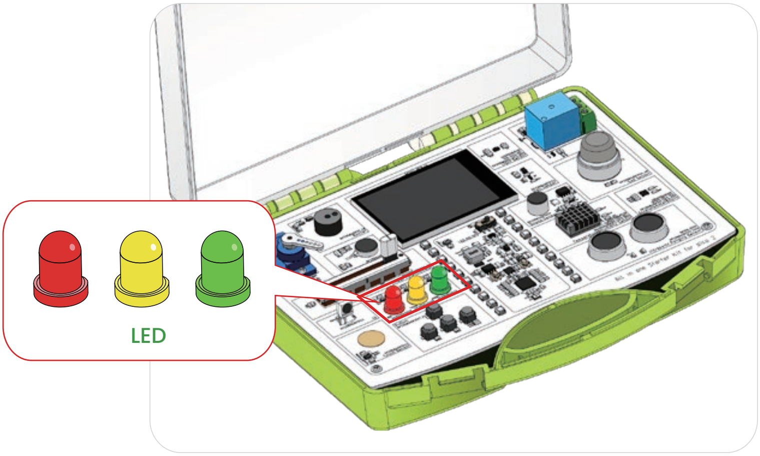
By modifying these four functions, you can achieve the goal of customizing the factory-default lighting effects.

Lesson 1 - LED Control

Introduction

In this lesson, we'll focus on hands-on programming with LED control. By writing code to configure GPIO pins and implement control logic, students will learn how to make LEDs blink alternately at fixed intervals. Along the way, they'll gain practical experience with basic hardware control while developing a deeper understanding of time management and loop logic—key concepts in embedded programming. This foundational knowledge will lay the groundwork for tackling more complex projects in the future.

Hardware Used in This Lesson:



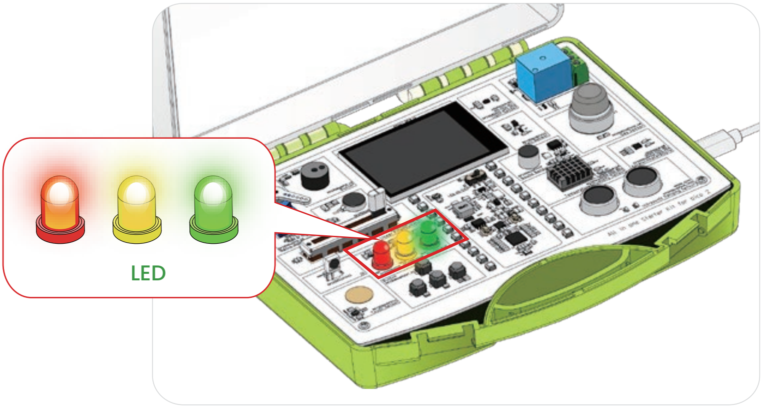
Working Principle of LED

At the heart of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During this recombination, electrons drop from a higher energy level to a lower one, releasing the excess energy in the form of photons—producing light. The color (or wavelength) of the emitted light is determined by the energy band gap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

After Running:

You can see: The blinking effect of the red, green, and yellow lights on the All in One Starter Kit for Pico2.



Key Explanations

1. Hardware Connection and Initialization

```
#define Red_LED 18
#define Yellow_LED 20
#define Green_LED 19
```

- This part of the code defines the pins connected to the three LEDs, making it easier to reference and use them in the rest of the program.

2. Time Control Mechanism

```
unsigned long previousMillis = 0;
const long interval = 1000;
```

- **previousMillis** is used to keep track of the last time the LED state changed.
- **interval** defines how often the LED should toggle; in this case, it's set to 1000 milliseconds, or 1 second.

3. LED State Management

```
int redState = LOW;
int yellowState = LOW;
int greenState = LOW;
```

- These three variables keep track of the current state of the red, yellow, and green LEDs. In Arduino, **HIGH** means the LED is on, and **LOW** means it's off.

4. Initialization Function

```
void setup() {
  pinMode(Red_LED, OUTPUT);
  pinMode(Yellow_LED, OUTPUT);
  pinMode(Green_LED, OUTPUT);
  digitalWrite(Red_LED, redState);
  digitalWrite(Yellow_LED, yellowState);
  digitalWrite(Green_LED, greenState);
}
```

- The **pinMode()** function sets the LED pins as output, allowing the Arduino to control their voltage levels.
- The **digitalWrite()** function sets the voltage level of each LED pin based on the values of **redState**, **yellowState**, and **greenState**. At the beginning, all three LEDs are off.

5. Main Loop and Time Judgment

```
void loop() {
  unsigned long currentMillis = millis();
  if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;
    redState = (redState == LOW) ? HIGH : LOW;
    yellowState = (yellowState == LOW) ? HIGH : LOW;
    greenState = (greenState == LOW) ? HIGH : LOW;
    digitalWrite(Red_LED, redState);
    digitalWrite(Yellow_LED, yellowState);
    digitalWrite(Green_LED, greenState);
  }
}
```

- The **millis()** function returns the number of milliseconds that have passed since the Arduino was powered on.
- By checking the condition **currentMillis - previousMillis >= interval**, the code can determine whether the set 1-second interval has passed.
- Once the interval is reached, the following actions are performed:

• **previousMillis** is updated to record the time of the current state change.
The states of the three LEDs are toggled using the ternary operator **(redState == LOW) ? HIGH : LOW,** which flips the state from off to on or vice versa.

"(redState == LOW) ? HIGH : LOW" is a ternary operator expression whose core logic is to invert the state of "redState". Specifically:
When "redState" equals "LOW", the condition is true, and the expression returns "HIGH".
When "redState" does not equal "LOW" (i.e., it is "HIGH"), the condition is false, and the expression returns "LOW".

• Finally, the new states are applied to the LED pins using the **digitalWrite()** function so that the LEDs reflect the updated status.

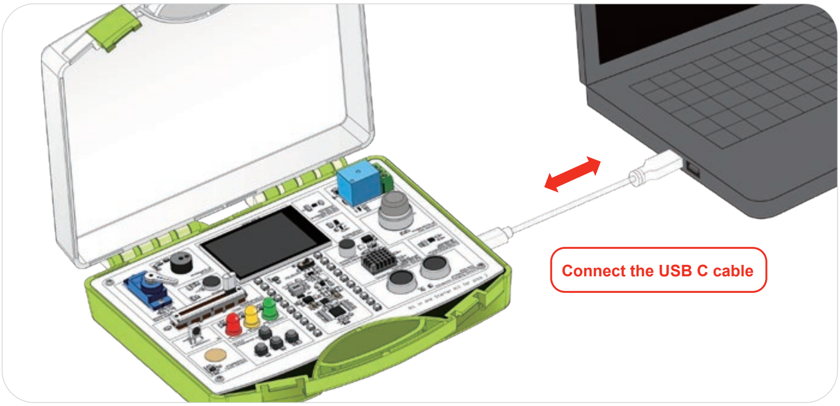
Complete Code

Kindly click the link below to view the full code implementation.

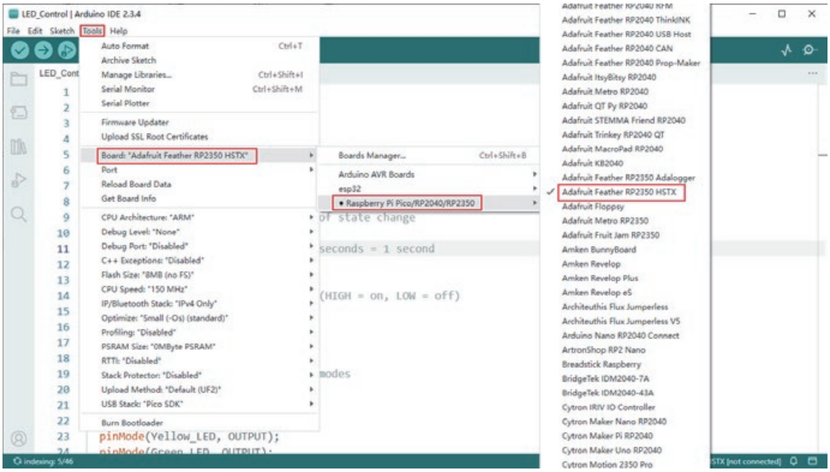
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

Programming Steps

To upload the code to the All-in-One Starter Kit for Pico2, start by connecting the board to your computer.



Next, you can copy the complete code from the folder into the Arduino IDE.
Make sure to select the correct development board you're using from the **Tools > Board** menu before uploading.


LED_Control | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter
Firmware Updater
Upload SSL Root Certificates
Port
Reload Board Data
Get Board Info
Burn Bootloader
Boards Manager... Ctrl+Shift+B
Arduino AVR Boards
esp32
• Raspberry Pi Pico/RP2040/RP2350
Adafruit Feather RP2040 RFM
Adafruit Feather RP2040 ThinkINK
Adafruit Feather RP2040 USB Host
Adafruit Feather RP2040 CAN
Adafruit Feather RP2040 Prop-Maker
Adafruit ItsyBitsy RP2040
Adafruit Metro RP2040
Adafruit QT Py RP2040
Adafruit STEMMA Friend RP2040
Adafruit Trinkey RP2040 QT
Adafruit MacroPad RP2040
Adafruit KB2040
Adafruit Feather RP2350 Adalogger
✓ Adafruit Feather RP2350 HSTX
Adafruit Floppsy
Adafruit Metro RP2350
Adafruit Fruit Jam RP2350
Amken BunnyBoard
Amken Revelop
Amken Revelop Plus
Amken Revelop eS
Architeuthis Flux Jumperless
Architeuthis Flux Jumperless V5
Arduino Nano RP2040 Connect
ArtronShop RP2 Nano
Breadstick Raspberry
BridgeTek IDM2040-7A
BridgeTek IDM2040-43A
Cytron IRIV IO Controller
Cytron Maker Nano RP2040
Cytron Maker Pi RP2040
Cytron Maker Uno RP2040
Cytron Motion 2350 Pro
indexing: 5/46
STX [not connected]

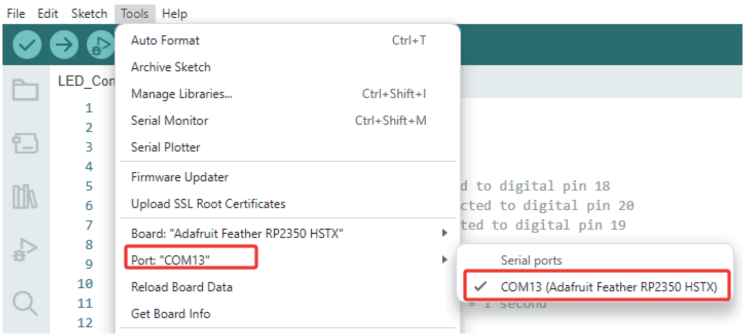
Board: "Adafruit Feather RP2350 HSTX"
CPU Architecture: "ARM"
Debug Level: "None"
Debug Port: "Disabled"
C++ Exceptions: "Disabled"
Flash Size: "8MB (no FS)"
CPU Speed: "150 MHz"
IP/Bluetooth Stack: "IPv4 Only"
Optimize: "Small (-Os) (standard)"
Profiling: "Disabled"
PSRAM Size: "0MByte PSRAM"
RTTI: "Disabled"
Stack Protector: "Disabled"
Upload Method: "Default (UF2)"
USB Stack: "Pico SDK"

```
of state change
seconds = 1 second
(HIGH = on, LOW = off)
modes
pinMode(Yellow_LED, OUTPUT);
pinMode(Green_LED, OUTPUT);
```

Before uploading the code, make sure to put the All-in-One Starter Kit for Pico2 into bootloader mode:

- **Press and hold the BOOT** button on the board.
- While holding BOOT, **press and release the RESET** button.
- Then, **release the BOOT** button.

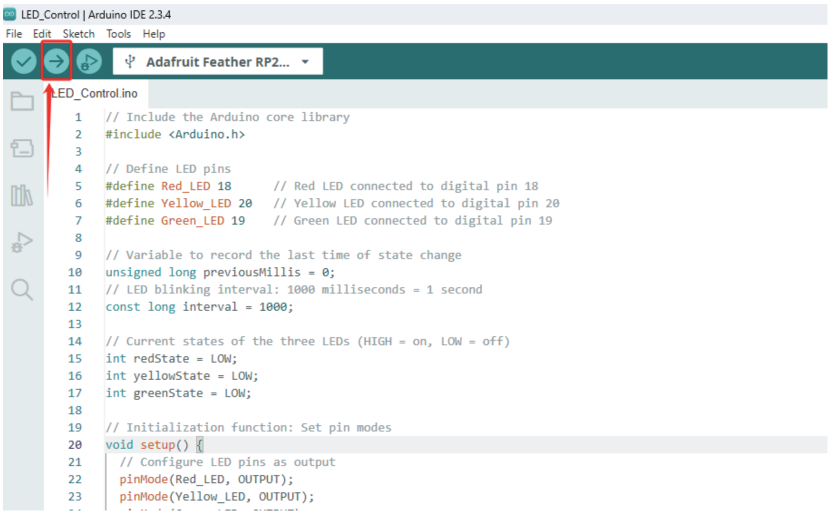
The board will now enter bootloader mode. Your computer should detect it as a new serial device. In the Arduino IDE, select this new port from the **Tools > Port** menu, and you'll be ready to upload the code.


File Edit Sketch Tools Help
Auto Format Ctrl+T
Archive Sketch
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter
Firmware Updater
Upload SSL Root Certificates
Reload Board Data
Get Board Info
Serial ports
✓ COM13 (Adafruit Feather RP2350 HSTX)

Board: "Adafruit Feather RP2350 HSTX"
Port: "COM13"

```
d to digital pin 18
cted to digital pin 20
ted to digital pin 19
= 1 second
```

Click “**Run**” to start the process.



LED_Control | Arduino IDE 2.3.4

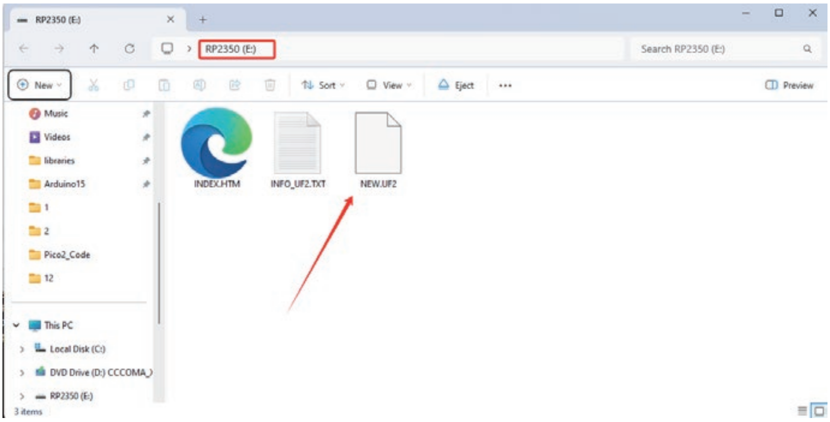
File Edit Sketch Tools Help

Adafruit Feather RP2...

LED_Control.ino

```
// Include the Arduino core library
#include <Arduino.h>

// Define LED pins
#define Red_LED 18      // Red LED connected to digital pin 18
#define Yellow_LED 20   // Yellow LED connected to digital pin 20
#define Green_LED 19    // Green LED connected to digital pin 19

// Variable to record the last time of state change
unsigned long previousMillis = 0;
// LED blinking interval: 1000 milliseconds = 1 second
const long interval = 1000;

// Current states of the three LEDs (HIGH = on, LOW = off)
int redState = LOW;
int yellowState = LOW;
int greenState = LOW;

// Initialization function: Set pin modes
void setup() {
  // Configure LED pins as output
  pinMode(Red_LED, OUTPUT);
  pinMode(Yellow_LED, OUTPUT);
```

Once everything runs successfully, a window will pop up.
In that window, you’ll see a file with the **.uf2** extension—this represents the executable file generated from your code upload.
Seeing the **.uf2** file confirms that the code has been successfully flashed to the board!



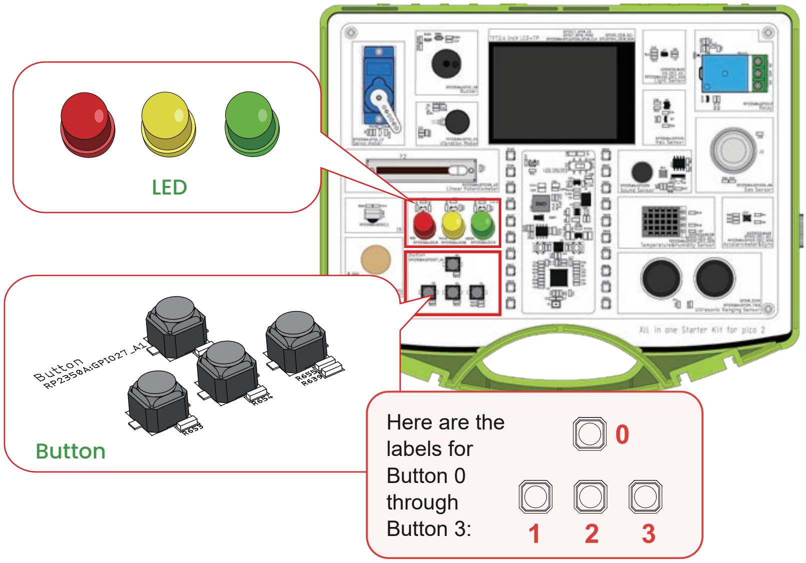
Finally, you will be able to see the display of relevant experimental results.

Lesson 2 – Button Control LED

Introduction

In this lesson, we'll use the All-in-One Starter Kit for Pico2 to learn how to control red, green, and yellow LEDs using button input. The course covers how to connect buttons to the development board, initialize the corresponding input pins, write logic to detect button states, and control the LEDs—turning them on or off—based on button presses.

Hardware Used in This Lesson:


LED
Button
Button RP2350A:GP1027_A1
Here are the labels for Button 0 through Button 3:
0
1
2
3
All in one Starter Kit for pico 2

Working Principle of an MCU-Based Button-Controlled LED System

This system uses analog voltage detection to handle multiple button inputs. Its core functionality is divided into three key components:

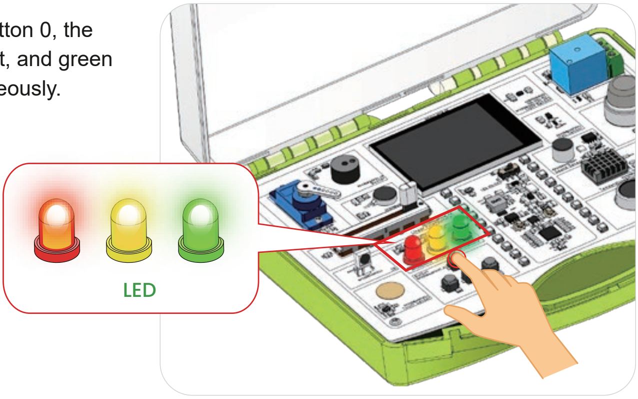
1. Button Detection: Multiple buttons are wired through a resistor voltage divider network to the MCU's ADC (Analog-to-Digital Converter) pin. When a button is pressed, it produces a unique voltage level. The MCU samples these voltages using median filtering to obtain stable readings, and then compares the result against predefined voltage ranges to identify which specific button was pressed.

2. Logic Processing: Once a button is identified, the MCU updates internal state flags (such as **redOn**, **yellowOn**, etc.) accordingly.
A debouncing mechanism—typically using a flag like **keyIsPressed**—ensures that each press is registered only once, preventing multiple toggles due to signal noise or rapid mechanical bouncing.

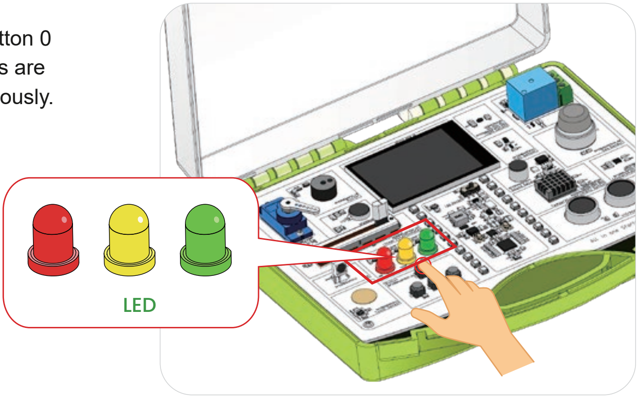
3. LED Driving: The MCU's GPIO pins control the LEDs based on the state flags. When a flag is set to **true**, the corresponding GPIO outputs a **HIGH** signal. This current, limited by a resistor (eg: 220Ω), powers the LED, turning it on. For example, when **redOn** is true, the red LED receives power and lights up.

Operation Effect Diagram

• When pressing button 0, the red light, yellow light, and green light are lit simultaneously.



• When pressing button 0 again, all three lights are turned off simultaneously.

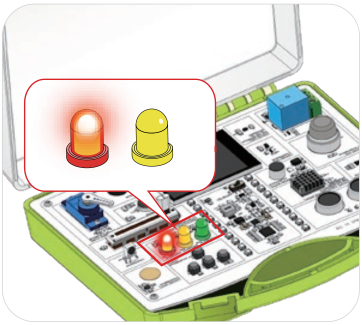


• Pressing button 1 lights the red light;

• Pressing button 1 again turns off the red light;

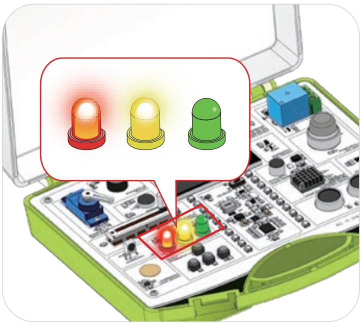
• Pressing button 2 lights the yellow light;

• Pressing button 2 again turns off the yellow light;



• Pressing button 3 lights the green light;

• Pressing button 3 again turns off the green light.



Key Explanations

1. Hardware Connection and Constant Definition

```
#define Red_LED    18
#define Yellow_LED  20
#define Green_LED  19
#define Button_pin  27
#define B0_L       740
#define B0_H       750
```

- The code defines the GPIO pins for the three LEDs as well as the analog pin connected to the button input.
- It also sets specific ADC value ranges for each button, which will later be used to identify which button has been pressed.

2. LED State Management

```
bool redOn = false;
bool yellowOn = false;
bool greenOn = false;
bool allOn = false;
```

- A value of **false** means the LED is **off**, while true indicates the LED is **on**.

3. Button Debounce Processing

```
bool keyIsPressed = false;
```

- This flag is used to prevent repeated triggers from a single button press, ensuring that each press is recognized only once.

4. Median Filter Function

```
int readMedianADC(int pin) {
  int readings[5];
  for (int i = 0; i < 5; i++) {
    readings[i] = analogRead(pin);
    delay(3);
  }
  for (int i = 0; i < 4; i++) {
    for (int j = i + 1; j < 5; j++) {
      if (readings[i] > readings[j]) {
        int temp = readings[i];
        readings[i] = readings[j];
        readings[j] = temp;
      }
    }
  }
  return readings[2];
}
```

- The purpose of this approach is to filter out occasional noise or interference, improving the accuracy and stability of the ADC readings.

5. Initialization Settings

```
void setup() {
  Serial.begin(115200);
  pinMode(Red_LED, OUTPUT);
  pinMode(Yellow_LED, OUTPUT);
  pinMode(Green_LED, OUTPUT);
  pinMode(Button_pin, INPUT);
  digitalWrite(Red_LED, LOW);
  digitalWrite(Yellow_LED, LOW);
  digitalWrite(Green_LED, LOW);
  Serial.println("System Ready.");
}
```

- Initializes serial communication,making it easier to output debug information.
- Sets the LED pins as outputs and the button pin as an input.
- Turns all LEDs off as the initial state.

6. Main Loop Logic

```
void loop() {
  int adcValue = readMedianADC(Button_pin);
  Serial.print("ADC: ");
  Serial.println(adcValue);
  bool inRange =
    (adcValue >= B0_L && adcValue <= B0_H) ||
    (adcValue >= B1_L && adcValue <= B1_H) ||
    (adcValue >= B2_L && adcValue <= B2_H) ||
    (adcValue >= B3_L && adcValue <= B3_H);
  if (inRange && !keyIsPressed) {
    keyIsPressed = true;
    if (adcValue >= B0_L && adcValue <= B0_H) {
      allOn = !allOn;
      redOn = yellowOn = greenOn = allOn;
      digitalWrite(Red_LED, redOn ? HIGH : LOW);
      digitalWrite(Yellow_LED, yellowOn ? HIGH : LOW);
      digitalWrite(Green_LED, greenOn ? HIGH : LOW);
      Serial.println("Button 0: Toggle ALL LEDs");
    }
    else if (adcValue >= B1_L && adcValue <= B1_H) {
      redOn = !redOn;
      digitalWrite(Red_LED, redOn ? HIGH : LOW);
      Serial.println("Button 1: Toggle RED");
    }
  }
  if (!inRange && keyIsPressed) {
    keyIsPressed = false;
  }
  delay(20);
}
```

▶ Calls the readMedianADC function to read and filter the ADC value from the button input.

▶ Checks whether the ADC value falls within a predefined range to determine if a specific button has been pressed.

▶ **When a new button press is detected:**

• Uses the keyIsPressed flag to prevent repeated triggers from the same press.
• Executes the corresponding LED control logic based on the detected ADC range.
• Applies the ternary operator (eg: redOn ? HIGH : LOW) to set each LED's state.
• Once the button is released, the debounce flag is reset, preparing the system to detect the next valid button press.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you can start modifying the functionality—such as using button presses to control the LEDs in different ways. In upcoming lessons, we'll also explore how to work with even more hardware features available on the All-in-One Starter Kit for Pico2.

Programming Steps

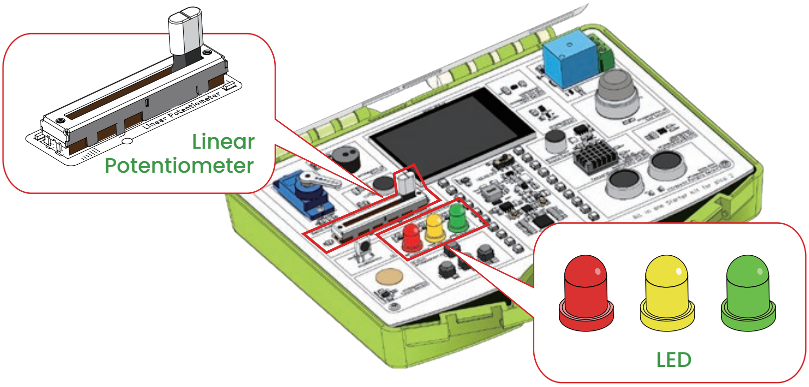
You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.

Lesson 3 - Breathing Led

Introduction

The slider module is a linear potentiometer with a maximum resistance of 10kΩ. Moving the wiper from left to right linearly increases the output voltage from 0V up to VCC.In this lesson, we'll use a potentiometer as the control input. By sampling its analog signal with the microcontroller's ADC and applying PWM modulation, we'll create a smooth, breathing-like LED brightness effect that responds to the slider's position. This provides a clear demonstration of how analog signal control and digital signal processing work together.

Hardware Used in This Lesson:

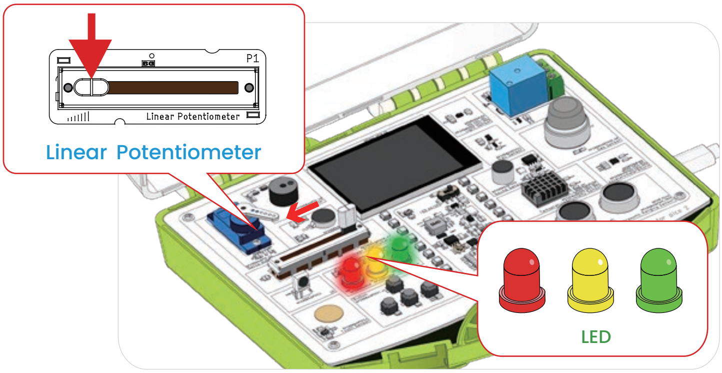


Principle of Sliding Rheostat Controlling LED Brightness

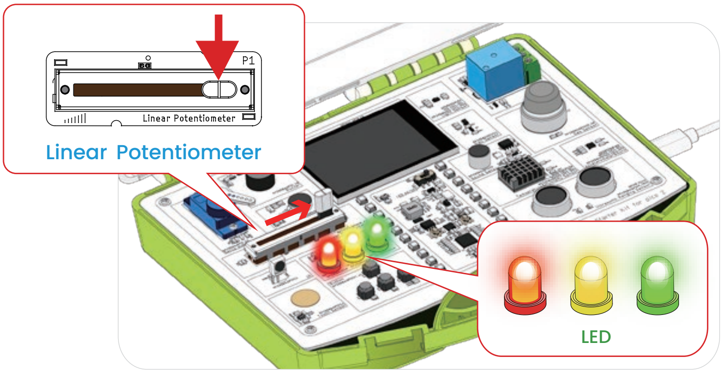
This system uses a sliding rheostat (potentiometer) to adjust the output voltage of a voltage divider circuit. The microcontroller's ADC (12-bit resolution, range 0–4095) reads this analog voltage and linearly maps it to an 8-bit PWM value (0–255). The PWM signal is then output through the microcontroller's GPIO pin. By adjusting the duty cycle, the system controls the average current flowing through the LED: a higher duty cycle means the LED stays on longer during each cycle, making it appear brighter; a lower duty cycle shortens the on-time, dimming the LED.

Operation Effect Diagram

• When I slide the potentiometer all the way to the left, the LED is at its dimmest.



• When I slide it all the way to the right, the LED reaches its maximum brightness.



Key Explanations

1. Hardware Connections and Pin Definitions

```
#define SLIDER_PIN     28
#define RED_LED_PIN    18
#define YELLOW_LED_PIN 20
#define GREEN_LED_PIN  19
```

- The pins connected to the potentiometer and the three LEDs are defined in the code.
- It's important to make sure the LED pins are assigned to PWM-capable GPIOs; otherwise, brightness control via PWM won't work.

2. Initialization Setup

```
void setup() {
  Serial.begin(115200);
  while (!Serial);
  pinMode(RED_LED_PIN, OUTPUT);
  pinMode(YELLOW_LED_PIN, OUTPUT);
  pinMode(GREEN_LED_PIN, OUTPUT);
  Serial.println("LED Brightness Control with Slider Ready.");
}
```

- The serial communication is initialized to allow relevant information to be printed during debugging.
- The LED pins are set as output so that the Arduino can control them properly.

3. Main Loop Logic

```
void loop() {
  int analogValue = analogRead(SLIDER_PIN);
  int pwmValue = map(analogValue, 0, 4095, 0, 255);
  analogWrite(RED_LED_PIN, pwmValue);
  analogWrite(YELLOW_LED_PIN, pwmValue);
  analogWrite(GREEN_LED_PIN, pwmValue);
  Serial.print("ADC: ");
  Serial.print(analogValue);
  Serial.print(" | PWM: ");
  Serial.println(pwmValue);
  delay(100);
}
```

▸ Reading the Analog Value:

• analogRead(SLIDER_PIN) reads the voltage from the potentiometer and converts it to a digital value.

• On the RP2040, the ADC has 12-bit resolution, meaning it maps the 0–3.3V input range to values between 0 and 4095.

▸ Mapping the Value Range:

• The map() function is used to convert the 0–4095 ADC range to the 0–255 range required for PWM.

• For example, if the ADC reads 2048, the mapped PWM value would be 127 — roughly 50% brightness.

▸ PWM Brightness Control:

• The analogWrite() function uses PWM to control the LED's brightness.

• A PWM value of 0 turns the LED off completely, while 255 sets it to full brightness.

▸ Debug Output:

• The serial monitor prints both the raw ADC value and the mapped PWM value, making it easier to observe how the system is behaving.

▸ Delay Control:

• delay(100) ensures the loop runs every 100 milliseconds, which reduces serial output clutter and prevents overly frequent readings from the potentiometer.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you can start modifying the functionality—such as using button presses to control the LEDs in different ways. In upcoming lessons, we'll also explore how to work with even more hardware features available on the All-in-One Starter Kit for Pico2.

Programming Steps

You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.

Lesson 4 - 2.4 inch TFT Display

Introduction

In this chapter, we'll use the TFT display module on the All-in-One Starter Kit for Pico2 to display text. The module communicates with the board via an SPI serial interface, making both the hardware connection and software integration relatively straightforward—ideal for beginners and quick prototyping.

Hardware Used in This Lesson:

TFT Display

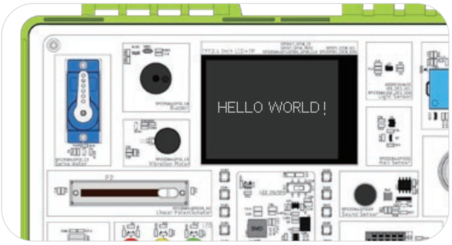
Working Principle of Dynamic Text Display on the Display Screen

The TFT display screen receives pixel data and control instructions sent by the MCU through the SPI bus. The ST7789 driver chip converts the received RGB data into a progressive scan signal, and precisely controls the voltage of each liquid crystal cell through the Source Driver and Gate Driver, thereby changing the light transmittance to realize image display. The touch function reports coordinate data through the I2C protocol by the FT5x06 chip to realize human-computer interaction. The backlight uses an independent GPIO to control the brightness of the LED backlight source.

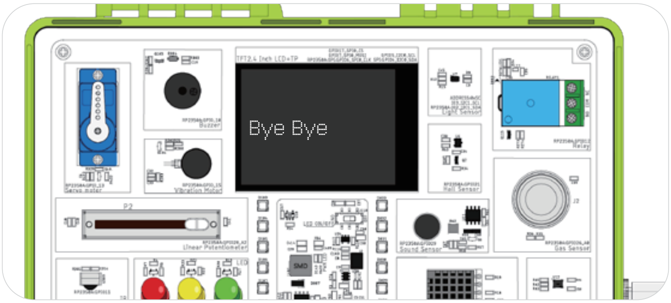
Operation Effect Diagram

Once the program runs, the TFT screen will follow this display sequence:

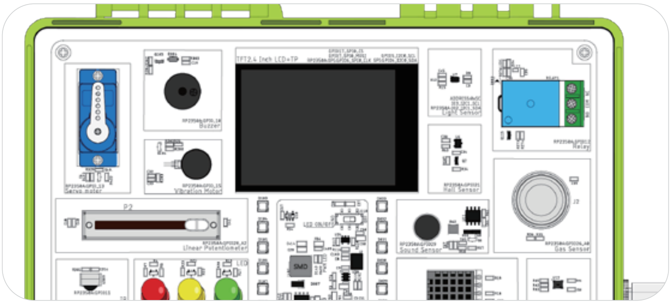
- 1.First, it shows "HELLO WORLD!";



- 2. After a 1-second delay, the text changes to "Bye Bye";



- 3. One second later, the screen is cleared and the display turns off.



Key Explanations

1. Display Driver and Library Imports

```
#include <LovyanGFX.hpp>
#include <lvgl.h>
```

- **LovyanGFX:** This library handles the low-level drivers for the TFT display, such as SPI communication and pixel operations. It simplifies hardware control and makes screen management much easier.
- **LVGL:** A lightweight graphics library used for building user interfaces. In this code, it's only maintained by calling lv_timer_handler() in the loop() function to keep the library running properly.

2. Initialization and Display Logic

```
void setup() {
  Serial.begin(115200);
  gfx.init();
  pinMode(0, OUTPUT);
  digitalWrite(0, HIGH);
  gfx.setTextSize(2);
  gfx.setTextColor(TFT_WHITE);
  gfx.setCursor(60, 100);
  gfx.print("HELLO WORLD!");
  delay(1000);
  gfx.fillScreen(TFT_BLACK);
  gfx.setCursor(100, 100);
  gfx.print("Bye Bye");
  delay(1000);
  gfx.fillScreen(TFT_BLACK);
  digitalWrite(0, LOW);
}
```

▶ **Key Display Control Functions:**

• **fillScreen(color)**: Fills the entire screen with the specified color (using black effectively clears the screen).

• **setCursor(x, y):** Sets the position of the top-left corner of the text; the origin (0, 0) is at the top-left of the screen.

• **setTextSize(n):** Sets the text size to n times the default font size (with n = 1 being the smallest).

3. Main Loop (loop Function)

```
void loop() {
  lv_timer_handler();
  delay(10);
}
```

▶ **LVGL Timer Handling:**

• **lv_timer_handler()** is a required function call in the LVGL library. It manages internal timers that handle animations, events, and other time-based operations. Even if your interface doesn't have dynamic effects, this function still needs to be called regularly to keep the library functioning properly.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the above code, you can control a 2.4-inch TFT display based on this development board to achieve dynamic text display: After the program starts, it first displays "HELLO WORLD!" at a specified position on the screen and pauses for 1 second, then clears the screen to display "Bye Bye" for another second, and finally clears the screen and turns off the backlight. The code configures hardware information such as SPI communication parameters and screen resolution through a custom LGFX class, uses the display interface of the LovyanGFX library to control text output, and integrates the LVGL library to maintain system operation. Once mastered, you can adjust the text content, display position, font size or color, add more display phases, or combine sensor data to achieve dynamic interactive display.

Programming Steps

Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**LovyanGFX-develop**),so it's important to include it before running the code to avoid compilation errors.

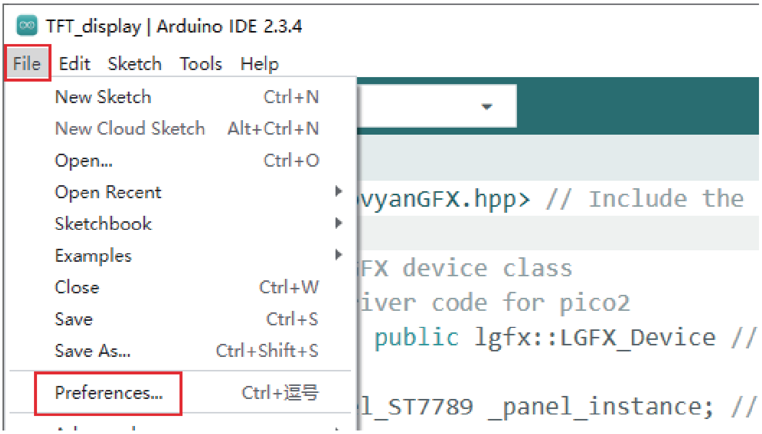
1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop** folder.

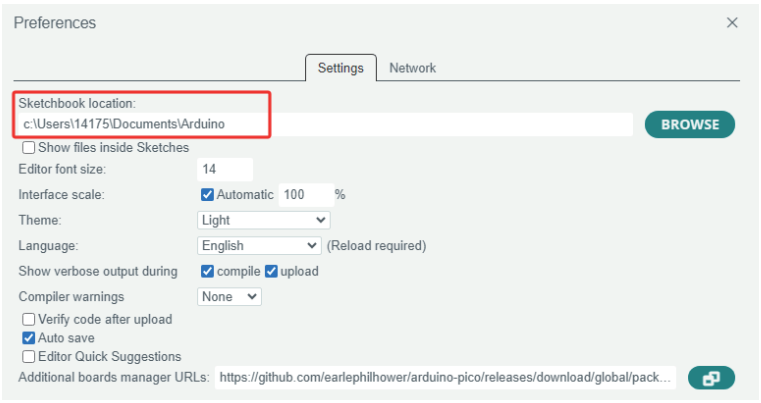
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



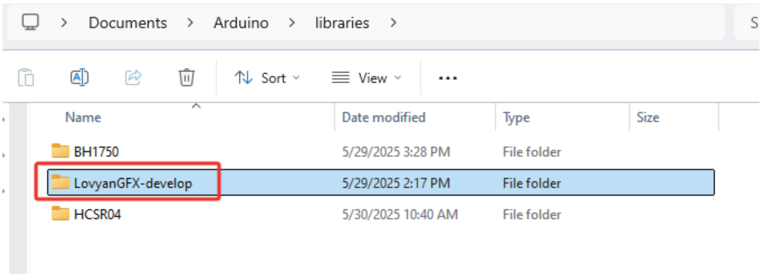
```
vyanGFX.hpp> // Include the
FX device class
iver code for pico2
 public lgfx::LGFX_Device //
l_ST7789 _panel_instance; //
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ✓ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ✓ compile ✓ upload
Compiler warnings None
☐ Verify code after upload
✓ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

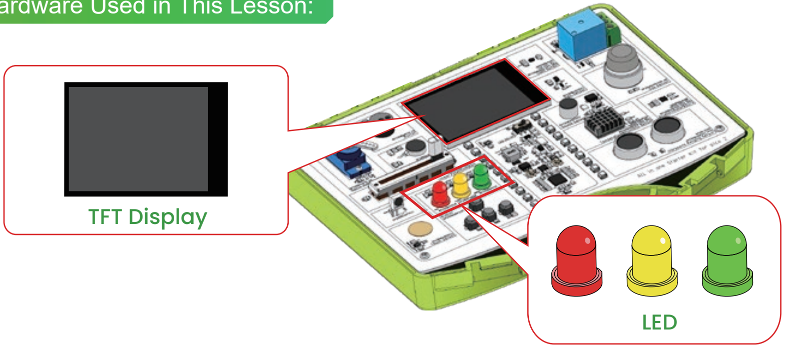
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 5 - Traffic Light

Introduction

In this lesson, we'll use an LED module to simulate the operation of real-world traffic lights, combined with a TFT display to show a countdown timer. This hands-on project is designed to help you understand the fundamentals of timing control and hardware interaction in embedded systems.

Hardware Used in This Lesson:



Traffic Light System Working Principle

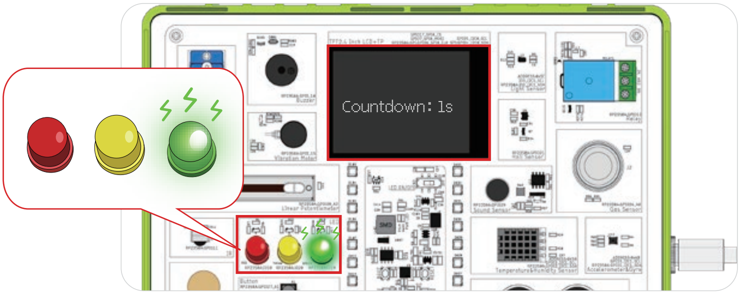
This traffic light system is based on a microcontroller (MCU) that directly drives the red, yellow, and green LEDs through GPIO pins. Each LED is connected in series with a current-limiting resistor to ensure the operating current stays within a safe range. The system uses a finite state machine (FSM) to implement a three-phase cycle: the green light stays on for 30 seconds, the yellow light stays on for 3 seconds, and the red light stays on for 20 seconds, with precise timing managed by a hardware timer. The countdown timer is displayed in real-time on an ST7789 TFT screen (driven by an SPI interface with an 80 MHz clock). During the last 5 seconds, the currently active LED flashes every 500ms as a warning. The entire system is designed with non-blocking programming to ensure that the coordination of peripherals (LEDs, display) does not affect real-time performance. Additionally, hardware debounce technology ensures the reliability of touch detection.

Operation Effect Diagram

Green Light Phase

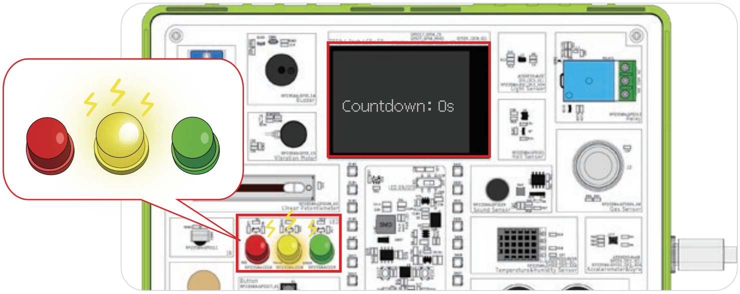
• The green light stays on, and the TFT displays "Countdown: 30 seconds," with the number decreasing by 1 each second.

• When the countdown reaches 5 seconds, the green light begins to flash.



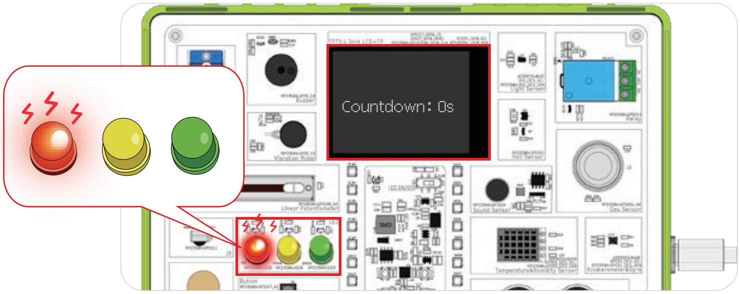
▸ **Yellow Light Phase**

• The green light turns off, the yellow light stays on, and the TFT displays "Countdown: 3 seconds."
• After 3 seconds, the yellow light turns off and switches to the red light.



▸ **Red Light Phase**

• The red light stays on, and the TFT displays "Countdown: 20 seconds."
• When the countdown reaches 5 seconds, the red light begins to flash, and after 20 seconds, the cycle returns to the green light phase.



Key Explanations

1. Traffic Light Phase Switching Logic

```
if (remainingTime == 0) {
  digitalWrite(Green_LED, LOW);
  digitalWrite(Yellow_LED, LOW);
  digitalWrite(Red_LED, LOW);
  switch(currentStage) {
    case GREEN_STAGE:
      currentStage = YELLOW_STAGE;
      remainingTime = 3;
      digitalWrite(Yellow_LED, HIGH);
      break;
    case YELLOW_STAGE:
      currentStage = RED_STAGE;
      remainingTime = 20;
      digitalWrite(Red_LED, HIGH);
      break;
    case RED_STAGE:
      currentStage = GREEN_STAGE;
      remainingTime = 30;
      digitalWrite(Green_LED, HIGH);
      break;
  }
}
```

▶ **Phase Switching**: This part of the code implements the functionality of switching the traffic light from one phase to another.

• When **remainingTime** (the remaining time of the current phase) reaches 0, it indicates the end of the current phase.

• First, all LEDs are turned off, and then, based on the value of currentStage, the system switches to the next phase.

• Each phase has a fixed duration:

• The green light phase (GREEN_STAGE) lasts for 30 seconds.

• The yellow light phase (YELLOW_STAGE) lasts for 3 seconds.

• The red light phase (RED_STAGE) lasts for 20 seconds.

▶ **LED Control**: When switching to a new phase, the corresponding LED for that phase is turned on, and the other LEDs are turned off.

• During the green light phase, the green LED is turned on.

• During the yellow light phase, the yellow LED is turned on.

2. Countdown Display Logic

```
if (now - lastUpdate >= 1000) {
  remainingTime--;
  lastUpdate = now;
  gfx.fillRect(TEXT_AREA_X, TEXT_AREA_Y, TEXT_AREA_WIDTH,
TEXT_AREA_HEIGHT, TFT_BLACK);
  gfx.setTextColor(TFT_WHITE);
  gfx.setCursor(TEXT_AREA_X, TEXT_AREA_Y);
  gfx.printf("%2ds", remainingTime);
}
```

▶ **Time Update:**

This part of the code implements the functionality of updating the countdown every second.

• The millis() function is used to get the current time in milliseconds.

• If 1000 milliseconds (i.e., 1 second) have passed since the last countdown update, remaining-Time is decreased by 1, and lastUpdate is updated to the current time.

▶ **Screen Display Update:**

• The gfx.fillRect() function is used to clear the countdown number area on the screen (to avoid clearing the "Countdown:" text).

• The text color is set to white, and the cursor is moved to the starting position of the countdown number area.

• The gfx.printf() function is used to print the updated countdown value in a formatted manner (eg: "29s").

3. Last 5 Seconds Flashing Logic

```
if (remainingTime > 0 && remainingTime <= 5) {
  if (now - lastBlink >= 500) {
    blinkState = !blinkState;
    lastBlink = now;
    switch(currentStage) {
      case GREEN_STAGE:
        digitalWrite(Green_LED, blinkState);
        break;
      case RED_STAGE:
        digitalWrite(Red_LED, blinkState);
        break;
    }
  }
}
```

- **Flashing Condition:** The flashing logic is triggered when the remaining time is greater than 0 and less than or equal to 5 seconds.

- **Flashing Frequency:** The LED state (on or off) switches every 500 milliseconds (0.5 seconds).

  • The **millis()** function is used to get the current time and compare it with lastBlink (the last blink time). If the time difference is greater than or equal to 500 milliseconds, the **blinkState** (flashing state) is toggled.
  • lastBlink is updated to the current time.

- **LED Control:**
  • During the green light phase (GREEN_STAGE), the green LED will flash.
  • During the red light phase (RED_STAGE), the red LED will flash.

4. Screen Initialization and Display Configuration

```
LGFX gfx;
static const uint16_t screenWidth = 320;
static const uint16_t screenHeight = 240;
```

- **Screen Object:** LGFX **gfx**; A screen object gfx is created to control the screen display.
- **Screen Size:** The screen width (**screenWidth**) and height (**screenHeight**) are defined as 320 pixels and 240 pixels, respectively

```
gfx.init();
pinMode(0, OUTPUT);
digitalWrite(0, HIGH);
```

- **Screen Initialization:**
  • The gfx.init() function is called to initialize the screen.
  • Pin 0 is set to output mode, and the screen backlight is turned on by calling digitalWrite(0, HIGH);.

5. Initialization Settings

```
pinMode(Red_LED, OUTPUT);
pinMode(Yellow_LED, OUTPUT);
pinMode(Green_LED, OUTPUT);
```

- **Pin Mode Setup:** The pinMode() function is used to set the pins for the red, yellow, and green LEDs to output mode.
  • The Red_LED (red LED) pin is set to 18.
  • The Yellow_LED (yellow LED) pin is set to 20.
  • The Green_LED (green LED) pin is set to 19.

6. Countdown Text Display Area Configuration

```
const int TEXT_AREA_X = 210;
const int TEXT_AREA_Y = 100;
const int TEXT_AREA_WIDTH = 60;
const int TEXT_AREA_HEIGHT = 30;
```

- **Text Area Position and Size**: The position and size of the countdown number display area are defined.
  • **TEXT_AREA_X** and **TEXT_AREA_Y** represent the X and Y coordinates of the text area, respectively.
  • **TEXT_AREA_WIDTH** and **TEXT_AREA_HEIGHT** represent the width and height of the text area, respectively.

- When displaying the countdown number, only the contents of this area are cleared to avoid clearing other parts (such as the **"Countdown:" text**).

7. Time Update and Logic Processing in the Main Loop

```
uint32_t now = millis();
delay(10);
```

- **Time Retrieval:** In the **loop()** function, the **millis()** function is used to get the current time in milliseconds, which is used to determine whether the countdown needs to be updated or the phase needs to be switched.

- **Minimum Delay:** A **delay(10)**; is added at the end of the loop to ensure the program does not run too fast and to prevent interference with screen refresh and other operations.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you can adjust the functionality as needed. The overall code implements a traffic light control system based on the Arduino platform, which simulates traffic light changes by controlling different colored LEDs (red, yellow, green) and displays the current countdown time of the traffic light on an LCD screen. When the green light is on, the countdown is displayed. After the yellow light stays on for 3 seconds, it switches to the red light. When the red light is on, the countdown is displayed. In the last 5 seconds, the green or red light will blink to alert.

Programming Steps

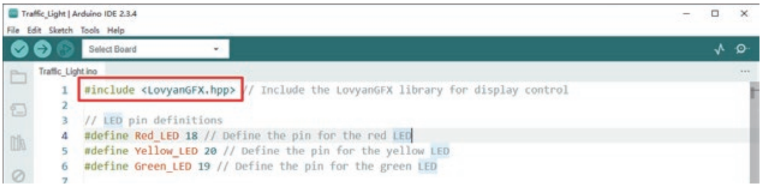
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**LovyanGFX-develop**),so it’s important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

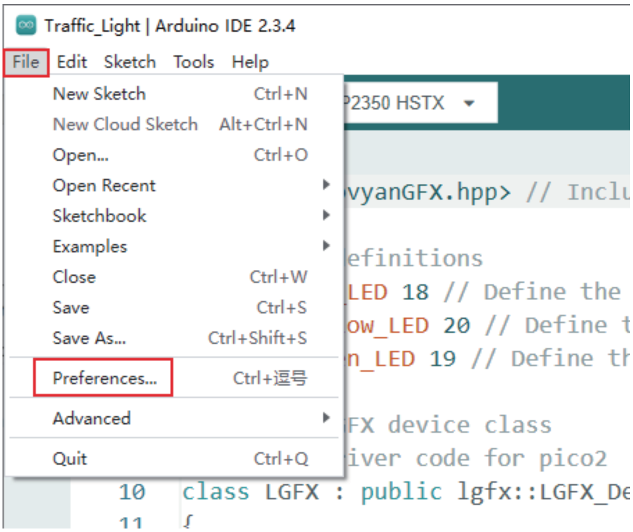
• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop** folder.


Traffic_Light | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Select Board
Traffic_Light.ino

```
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control

// LED pin definitions
#define Red_LED 18 // Define the pin for the red LED
#define Yellow_LED 20 // Define the pin for the yellow LED
#define Green_LED 19 // Define the pin for the green LED
```

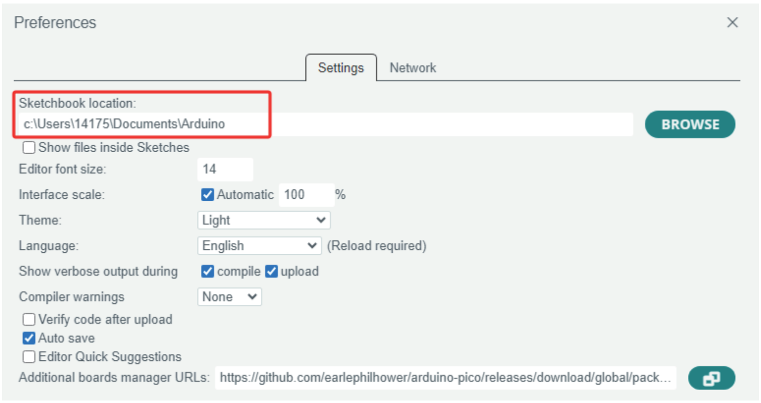
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
vyanGFX.hpp> // Inclu

efinitions
LED 18 // Define the
ow_LED 20 // Define t
n_LED 19 // Define th

FX device class
iver code for pico2
10   class LGFX : public lgfx::LGFX_De
11   {
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.


Preferences

Settings | Network

Sketchbook location:
c:\Users\14175\Documents\Arduino

☐ Show files inside Sketches

Editor font size: 14

Interface scale: ✓ Automatic 100 %

Theme: Light

Language: English (Reload required)

Show verbose output during ✓ compile ✓ upload

Compiler warnings: None

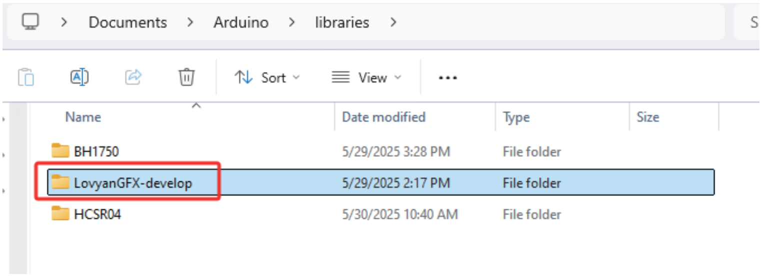
☐ Verify code after upload

✓ Auto save

☐ Editor Quick Suggestions

Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.


Documents > Arduino > libraries >

Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

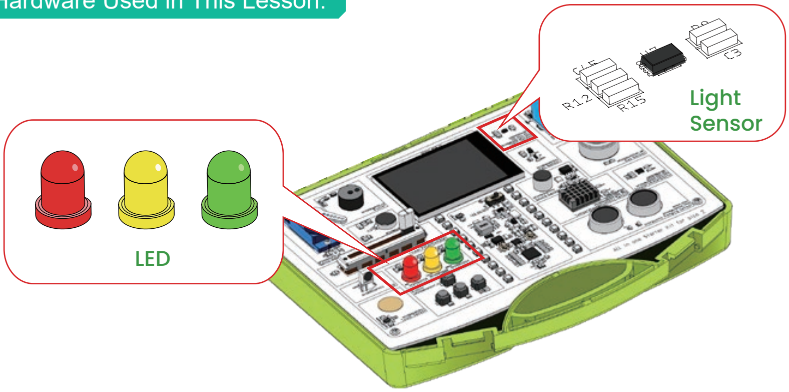
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 6 - Intelligent Street Light

Introduction

This chapter will provide a detailed explanation of how to collect brightness data from the light module, and how to implement intelligent on/off control of the LED based on the collected data. By setting a reasonable brightness threshold mechanism, the system can automatically control the on/off state of the LED based on ambient light intensity, avoiding unnecessary lighting, effectively reducing energy consumption, and achieving the goal of energy conservation and efficiency improvement.

Hardware Used in This Lesson:



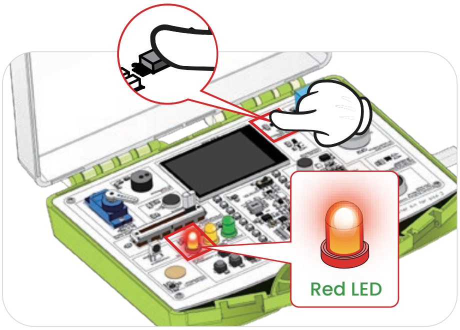
Working Principle of Light-Controlled LED System

This system uses a digital light intensity sensor (such as the BH1750) to collect ambient light intensity data and automatically control the on/off state of the LED based on the collected light intensity. The system uses the I2C communication protocol to interact with the light sensor through the microcontroller, obtaining real-time ambient light illuminance. In terms of hardware, the system connects the sensor to the microcontroller via the I2C bus, periodically reads the sensor data, and determines whether the ambient light intensity meets the criteria for activating the LED based on a set threshold. If the ambient light intensity falls below the set threshold (eg: 100lx), the system will turn on the LED, otherwise, it will turn off. The system uses low-power light sensors (such as the BH1750), with a working current of only 0.12mA, enabling long-term stable operation. Additionally, the system implements a non-blocking delay mechanism to ensure sampling real-time accuracy and avoid interference with other functions due to overly fast execution. This design enables the automatic adjustment of the LED's on/off state based on ambient light intensity, effectively saving energy and ensuring lighting is only on when needed.

Operation Effect Diagram

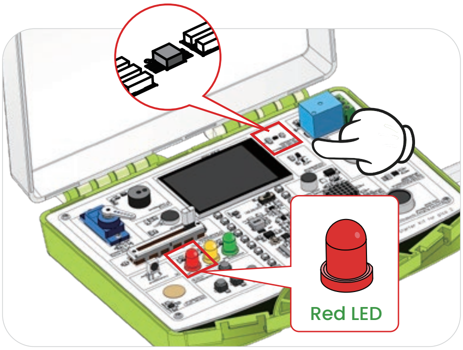
• Simulating a Dark Environment:

By covering the top of the light sensor with your hand, the ambient light intensity decreases, and the red LED will automatically turn on.



• Simulating a Light Environment:

When you remove your hand, the ambient light intensity increases, and the red LED will automatically turn off.



Key Explanations

1. Hardware Connections and Library Inclusion

```
#define I2C_SDA 2
#define I2C_SCL 3
#define Red_LED 18
BH1750 lightMeter(0x5c);
```

▸ **I2C Communication:**
Communication between the sensor and Arduino is achieved through two wires (SDA for data, SCL for clock), which is a commonly used low-speed serial communication protocol.

▸ **Sensor Address:**
Each I2C device has a unique address (0x5C is the default address for the BH1750), which is used by the Arduino to identify the device.

2. Initialization Settings

```
void setup() {
  Serial.begin(9600);
  delay(100);
  Wire1.setSDA(I2C_SDA);
  Wire1.setSCL(I2C_SCL);
  Wire1.begin();
  if(lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE, 0x5c,
&Wire1)) {
    Serial.println("BH1750 sensor initialized.");
  } else {
    Serial.println("Failed to initialize BH1750 sensor.");
  }
  pinMode(Red_LED, OUTPUT);
}
```

- **Sensor Initialization Mode:** CONTINUOUS_HIGH_RES_MODE means the sensor operates in continuous high-resolution mode, with a measurement range of 1-65535 lux.
- **Error Detection:** The initialization of the sensor is checked using an if statement to ensure it is successful, which is helpful for debugging.

3. Last 5 Seconds Flashing Logic

```
void loop() {
  if (lightMeter.measurementReady(true)) {
    float lux = lightMeter.readLightLevel();
    Serial.print("Current light level: ");
    Serial.print(lux);
    Serial.println(" lx");
    if (lux < 100) {
      digitalWrite(Red_LED, HIGH);
    } else {
      digitalWrite(Red_LED, LOW);
    }
  }
  delay(200);
}
```

- **Measurement Readiness Check:** measurementReady(true) will block the program until the sensor completes a measurement, ensuring the validity of the data.
- **Light Intensity Reading**: readLightLevel() returns a floating-point value of light intensity, for example, 50.5 lx represents 50.5 lux.
- **LED Control Logic:** The LED on/off state is controlled through a simple condition check (lux < 100), using digitalWrite to output high/low voltage.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After learning the above code, you can modify the functionality to create an intelligent lighting control system based on a light sensor. By real-time detection of ambient light intensity and comparing it with a set threshold, the system can automatically control the on/off state of the LED, achieving energy-saving effects.

Programming Steps

Note: For detailed programming steps, you can refer to the programming process in the first lesson.

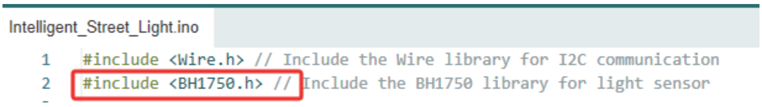
In this lesson, we use an additional library (BH1750),so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

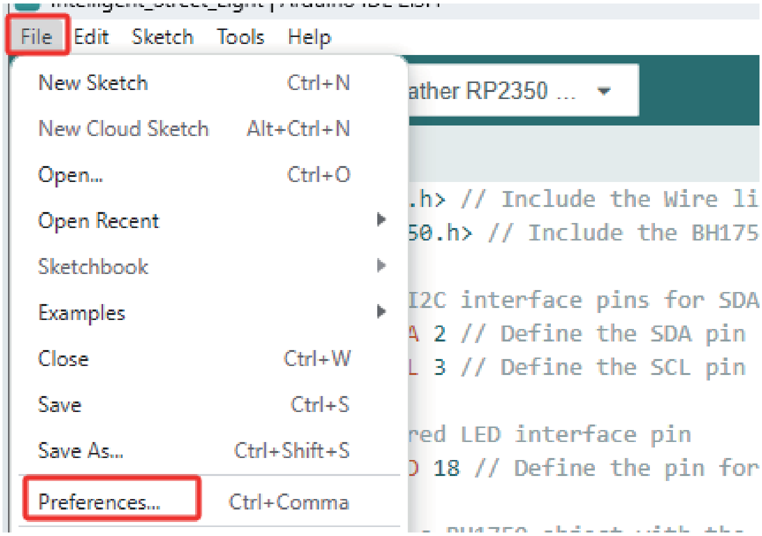
• Navigate to the **/example/libraries** directory and download the **BH1750** folder.


Intelligent_Street_Light.ino

```
#include <Wire.h> // Include the Wire library for I2C communication
#include <BH1750.h> // Include the BH1750 library for light sensor
```

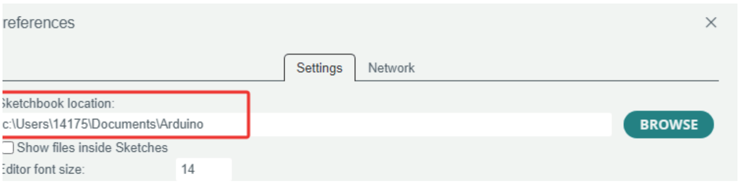
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



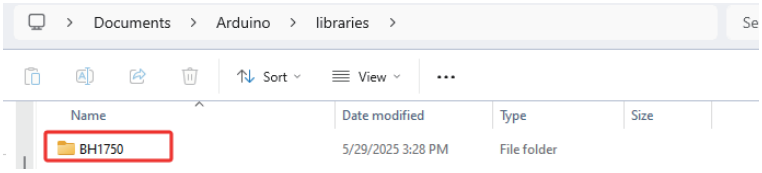
```
.h> // Include the Wire li
50.h> // Include the BH175

I2C interface pins for SDA
A 2 // Define the SDA pin
L 3 // Define the SCL pin

red LED interface pin
D 18 // Define the pin for
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
Show files inside Sketches
Editor font size: 14

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

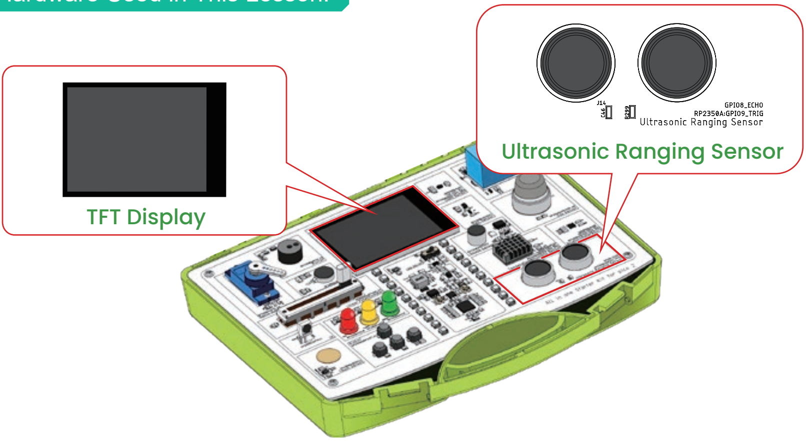
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 7 - Ultrasonic Ranging Display

Introduction

In this section, you'll learn how to use an ultrasonic sensor module to measure the distance between the sensor and an object in front of it. You'll build a simple ultrasonic distance meter that displays the measured values in real time on an LCD screen. This lesson covers the basic principle of ultrasonic distance measurement, hardware connections between the sensor and the display, and how to write code to collect data and visualize the results.

Hardware Used in This Lesson:

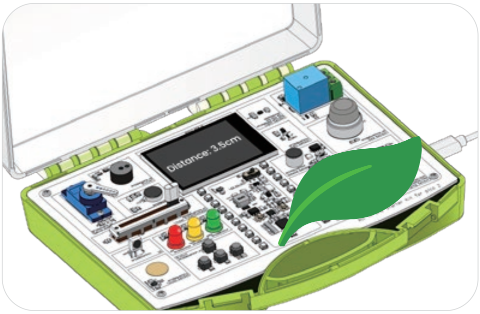


Working Principle of the Ultrasonic Distance Display System

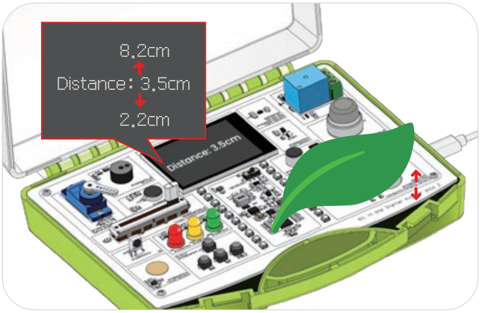
This ultrasonic distance display system uses the HC-SR04 module for non-contact distance measurement. A 40kHz ultrasonic pulse is triggered via GP9, and the echo is received through GP8. The microcontroller calculates the time difference to determine the distance (with an effective range of 2.8–400 cm). Every 500 ms, the system collects three measurements and applies median filtering to ensure stable readings. The filtered distance data is transmitted to an ST7789 LCD screen over an 80 MHz SPI bus (SCLK = GP6, MOSI = GP7) and displayed in real time at a resolution of 240×320.
Hardware-wise, the ultrasonic module operates at 5V, and its Echo signal is level-shifted to be compatible with the 3.3V system. The display backlight is directly controlled by GPIO0 and configured with a proper refresh rate to prevent flicker.A digital filtering algorithm is integrated to suppress environmental noise, and if invalid data is detected, the screen displays an "Out of range" warning. The entire measurement process uses a non-blocking design to maintain real-time responsiveness.

Operation Effect Diagram

Once the program starts running, the LCD screen will continuously update to show the real-time distance measured by the ultrasonic sensor.



As you move a flat object in front of the ultrasonic module, the distance value displayed on the screen will change dynamically.



Key Explanations

1. Ultrasonic Sensor Initialization and Configuration

```
const byte triggerPin = 9;
const byte echoPin = 8;
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);
```

- The trigger and echo pins for the ultrasonic sensor are defined, and an instance of the sensor is created.

- The trigger and echo pins for the ultrasonic sensor are defined, and an instance of the sensor is created.The UltraSonicDistanceSensor class, provided by a library, simplifies working with ultrasonic sensors.By specifying the trigger and echo pins, you can instantiate a sensor object to begin distance measurements.

2. Obtaining Stable Distance Readings

```
float getStableDistance() {
  float total = 0;
  int validCount = 0;
  for (int i = 0; i < 3; i++) {
    float d = distanceSensor.measureDistanceCm();
    if (d > 0 && d < 500) {
      total += d;
      validCount++;
    }
    delay(20);
  }
  if (validCount == 0) return -1;
  return total / validCount;
}
```

- To improve measurement stability, the code takes multiple readings and averages them to produce a more accurate result.
- The measureDistanceCm function is used to measure distance in centimeters.To reduce errors, the code performs 3 measurements and checks if each value falls within a valid range (greater than 0 and less than 500 cm) using if (d > 0 && d < 500).。 If a reading is valid, it is added to the total, and the count of valid readings (validCount) is incremented.Finally, the average is calculated using total / validCount. If no valid readings were obtained, -1 is returned to indicate a failed measurement.

3. Displaying the Distance Value on the Screen

```
gfx.fillRect(TEXT_AREA_X, TEXT_AREA_Y, TEXT_AREA_WIDTH + 50,
TEXT_AREA_HEIGHT, TFT_BLACK);
gfx.setTextColor(TFT_WHITE);
gfx.setCursor(TEXT_AREA_X, TEXT_AREA_Y);
if (distance > 0) {
  gfx.printf("%.2f cm", distance);
} else {
  gfx.print("Out of range");
}
```

- This section of code handles displaying the measured distance on the screen and clearing the old value with each update.

- The gfx.fillRect function clears a rectangular area of the screen to prevent overlapping digits., gfx.setTextColor sets the text color, gfx.setCursor defines the text position, and gfx.printf formats and prints the distance value.The format %.2f ensures the number is displayed as a float with two decimal places.If the measured distance is invalid (less than or equal to 0), the message “Out of range” is displayed instead.

4. Timed Distance Display Update

```
if (currentTime - lastUpdateTime >= UPDATE_INTERVAL) {
    lastUpdateTime = currentTime;
    float distance = getStableDistance();
    gfx.fillRect(TEXT_AREA_X, TEXT_AREA_Y, TEXT_AREA_WIDTH + 50,
TEXT_AREA_HEIGHT, TFT_BLACK);
    gfx.setTextColor(TFT_WHITE);
    gfx.setCursor(TEXT_AREA_X, TEXT_AREA_Y);
    if (distance > 0) {
      gfx.printf("%.2f cm", distance);
    } else {
      gfx.print("Out of range");
    }
  }
```

- The update frequency of the distance value is controlled by a timer to prevent screen flickering or high CPU usage caused by overly frequent updates.

- UPDATE_INTERVAL defines the update interval time (500 milliseconds). The condition current-Time - lastUpdateTime >= UPDATE_INTERVAL checks whether it's time to update. If it's time to update, the getStableDistance function is called to obtain the current stable distance and update the screen display.lastUpdateTime stores the last update timestamp and is used to determine when the next update should occur.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the above code, you can modify its functions to implement an ultrasonic distance measurement system that displays the measured distance on the screen in real time. It periodically (every 500 milliseconds) obtains stable distance data and dynamical-ly updates the display, ensuring both real-time responsiveness and display stability.

Programming Steps

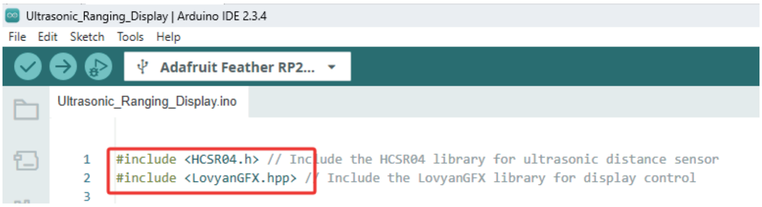
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use two additional libraries: **LovyanGFX-develop** and **HCSR04**.,so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

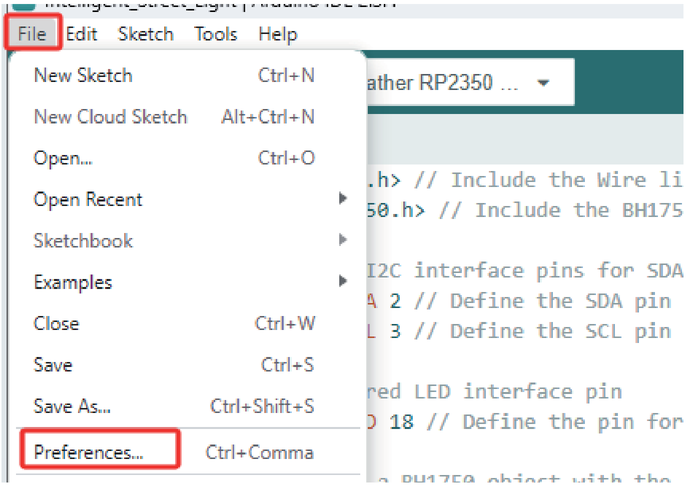
• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop** and **HCSR04.** folder.


Ultrasonic_Ranging_Display | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2...
Ultrasonic_Ranging_Display.ino

```
#include <HCSR04.h> // Include the HCSR04 library for ultrasonic distance sensor
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control
```

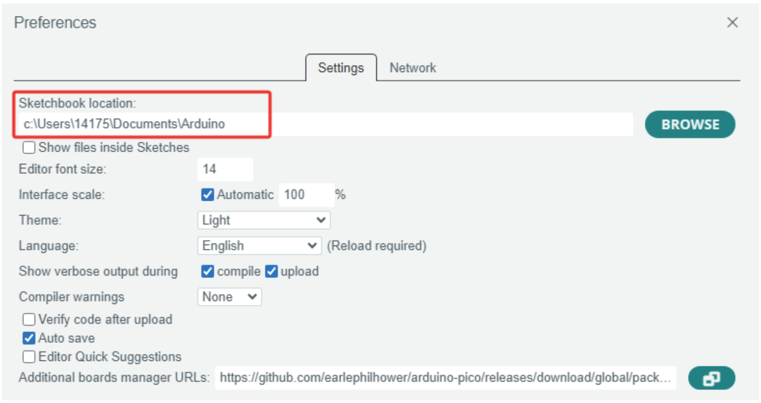
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
.h> // Include the Wire li
50.h> // Include the BH175

I2C interface pins for SDA
A 2 // Define the SDA pin
L 3 // Define the SCL pin

red LED interface pin
D 18 // Define the pin for
```

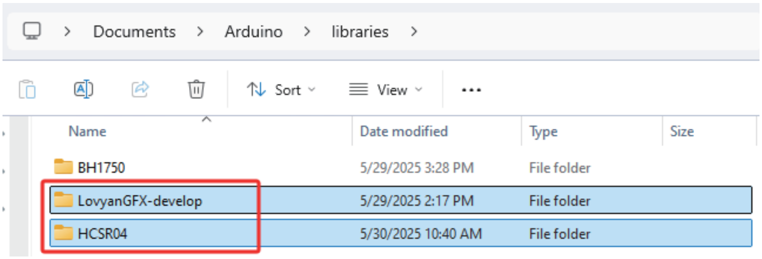
• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.


Preferences

Sketchbook location:
c:\Users\14175\Documents\Arduino

☐ Show files inside Sketches
Editor font size: 14
Interface scale: ✓ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ✓ compile ✓ upload
Compiler warnings: None
☐ Verify code after upload
✓ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.


Documents > Arduino > libraries >

Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

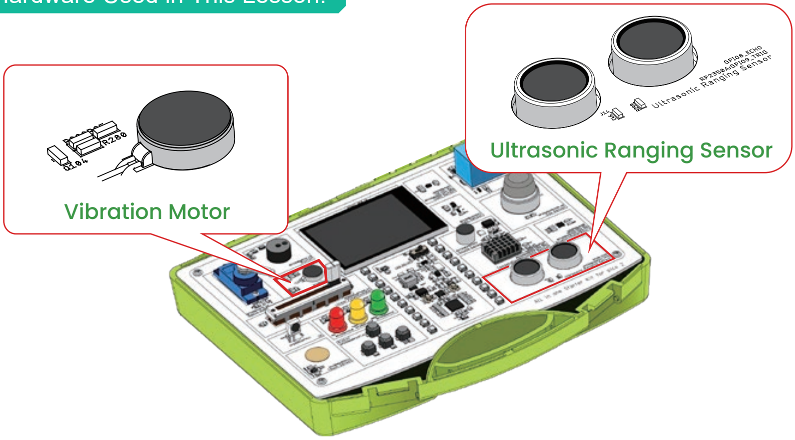
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 8 – Obstacle Close Range Alarm

Introduction

In this chapter, we will explore the application of the ultrasonic module and its coordinated control with other modules. By reading distance data from the ultrasonic sensor, we will implement a threshold-based control logic for a vibration motor: when an obstacle is detected within 30 cm, the vibration motor is activated as an alert; when the distance is 30 cm or more, the motor is turned off, indicating a safe state. This mechanism can be used to implement a basic ultrasonic obstacle avoidance function.

Hardware Used in This Lesson:

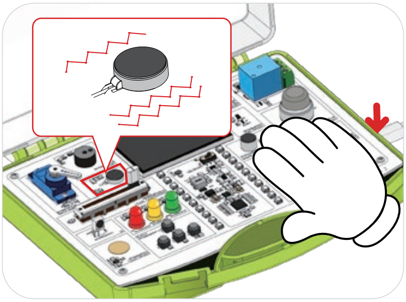


Working Principle of Ultrasonic Ranging Anti-Collision System

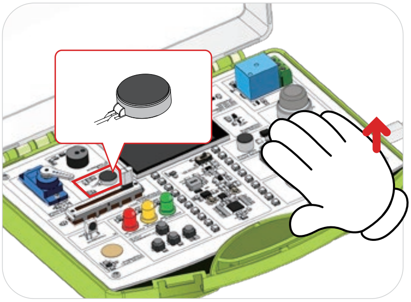
The ultrasonic ranging anti-collision system realizes obstacle detection through the HC-SR04 module. It triggers 40kHz ultrasonic pulses via GP9, receives echo signals through GP8, and the MCU calculates the time difference to convert it into a distance value. The system performs measurements every 100ms. When an obstacle within a 30cm range is detected, GP15 outputs a PWM signal to drive the vibration motor for tactile alarm.In hardware design, the 5V-powered ultrasonic module is paired with a 3.3V level conversion circuit. The vibration motor drive uses a MOSFET transistor to ensure sufficient current driving capability (typical operating current: 60mA). The system integrates a digital filtering algorithm to eliminate environmental interference, automatically turns off the motor output in case of anomalies, and outputs distance data and system status in real time via a serial port. It adopts non-blocking programming design to ensure real-time response performance, while reducing system power consumption through appropriate delay control.

Operation Effect Diagram

• **Simulating Obstacle Approaching:** Slowly move a hand or object closer to the ultrasonic sensor (distance < 30 cm). The vibration motor should activate, indicating the presence of an obstacle.



• **Simulating Obstacle Moving Away:** Remove the object (distance ≥ 30 cm). The vibration motor should stop, indicating a safe zone.



Key Explanations

1. Ultrasonic Sensor Initialization and Configuration

```
#include <HCSR04.h>
const byte triggerPin = 9;
const byte echoPin = 8;
UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin);
```

- **Library Inclusion:** #include <HCSR04.h> includes the ultrasonic sensor library, which provides convenient functions for distance measurement.
- **Pin Definitions:**
  - triggerPin is connected to the sensor's trigger pin and used to initiate the measurement.
  - echoPin is connected to the sensor's echo pin and used to receive the reflected ultrasonic signal.
- **Sensor Object Creation:** An ultrasonic sensor object named distanceSensor is created using UltraSonicDistanceSensor distanceSensor(triggerPin, echoPin); for subsequent distance measurements.

2. Distance Measurement and Update Logic

```
const unsigned long UPDATE_INTERVAL = 100;
unsigned long lastUpdateTime = 0;
if (currentTime - lastUpdateTime >= UPDATE_INTERVAL) {
  lastUpdateTime = currentTime;
  float distance = distanceSensor.measureDistanceCm();
}
```

- **Update Interval:** UPDATE_INTERVAL defines the time interval between two distance measurements, set to 100 milliseconds here.
- **Time Variable:** lastUpdateTime is used to record the time of the last distance update.
- **Time Check:** In the loop() function, millis() is used to get the current time (currentTime), and it checks whether the time since the last update has reached UPDATE_INTERVAL.
- **Distance Measurement:** If the interval condition is met, the function distanceSensor.measureDistanceCm() is called to measure the distance in centimeters.

3. Distance Validation and Logic Handling

```
if (distance > 0) {
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  if (distance < 30.0) {
    digitalWrite(vibratePin, HIGH);
    Serial.println("Obstacle detected! Vibration motor ON.");
  } else {
    digitalWrite(vibratePin, LOW);
    Serial.println("Safe distance. Vibration motor OFF.");
  }
} else {
  digitalWrite(vibratePin, LOW);
  Serial.println("Out of range or measurement error.");
}
```

- **Validation Check:** If the measured distance is greater than 0 cm, the measurement is considered valid.
- **Distance Display:** The measured distance is printed via the serial monitor.

- **Logic Processing:**

  • If the distance is less than 30 cm, it indicates an obstacle is detected. The vibration motor is turned on (digitalWrite(vibratePin, HIGH)), and the message “Obstacle detected! Vibration motor ON.” is printed via serial.

  • If the distance is greater than or equal to 30 cm, it indicates a safe distance. The motor is turned off (digitalWrite(vibratePin, LOW)), and the message “Safe distance. Vibration motor OFF.” is printed via serial.

- **Error Handling:** If the measured distance is less than or equal to 0 cm, it indicates an invalid or out-of-range measurement. The motor is turned off and an error message is printed via serial.

4. Vibration Motor Control

```
const int vibratePin = 15;
pinMode(vibratePin, OUTPUT);
digitalWrite(vibratePin, LOW);
```

- **Pin Definition:** vibratePin defines the pin connected to the vibration motor.

- **Pin Mode Configuration:** In the setup() function, pinMode(vibratePin, OUTPUT) sets the pin mode to output.

- **Initial State:** The motor is turned off by default using digitalWrite(vibratePin, LOW).

5. Serial Communication Initialization

```
Serial.begin(9600);
Serial.println("System ready. Starting distance monitoring...");
```

- **Serial Initialization:**

  • Serial.begin(9600) initializes serial communication with a baud rate of 9600.

- **Waiting for Connection:** while (!Serial) waits for the serial port to connect (required on some boards).

- **Startup Message:** Serial.println("System ready. Starting distance monitoring...") prints a startup message to indicate the system is ready.

6. Time Update and Logic Handling in the Main Loop

```
unsigned long currentTime = millis();
delay(10);
```

- **Time Acquisition:** In the loop() function, millis() is used to get the current time (in milliseconds) to determine whether it's time to perform a new distance measurement and control logic.
- **Minimum Delay:** Adding delay(10); at the end of the loop() helps prevent excessively high loop frequency, reducing CPU load and avoiding unnecessary processing.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After completing the code implementation above, you've successfully built a proximity alert system using an ultrasonic sensor. This system continuously monitors the distance to obstacles ahead—triggering a vibration motor alarm when objects come within 30 cm, while maintaining a "safe" status at greater distances. It also incorporates error handling and timed detection mechanisms for reliable operation.

Programming Steps

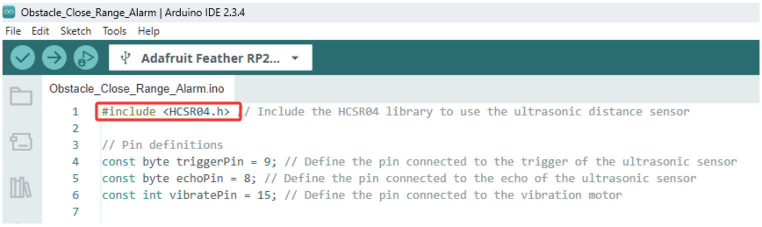
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**HCSR04**),so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

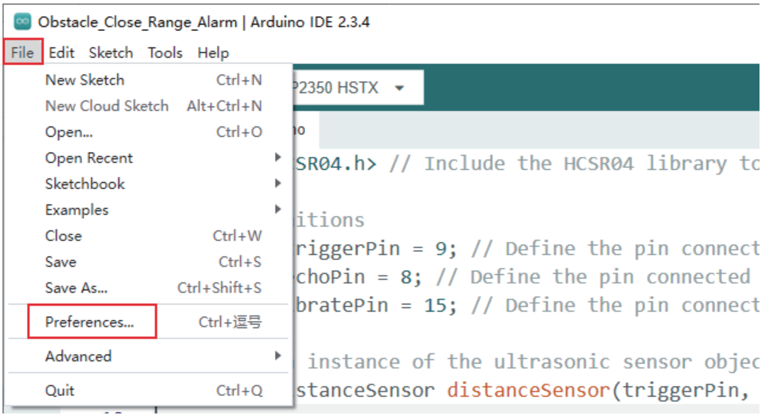
• Navigate to the /example/libraries directory and download the **HCSR04** folder.


Obstacle_Close_Range_Alarm | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2...
Obstacle_Close_Range_Alarm.ino

```
#include <HCSR04.h> // Include the HCSR04 library to use the ultrasonic distance sensor

// Pin definitions
const byte triggerPin = 9; // Define the pin connected to the trigger of the ultrasonic sensor
const byte echoPin = 8; // Define the pin connected to the echo of the ultrasonic sensor
const int vibratePin = 15; // Define the pin connected to the vibration motor

```

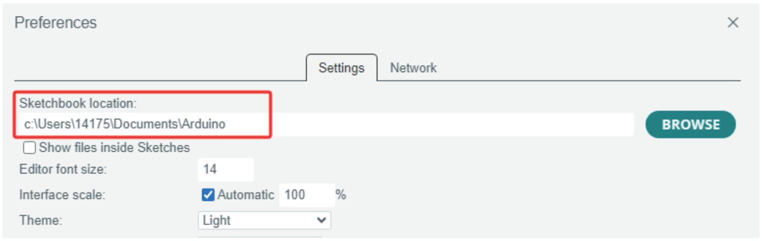
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



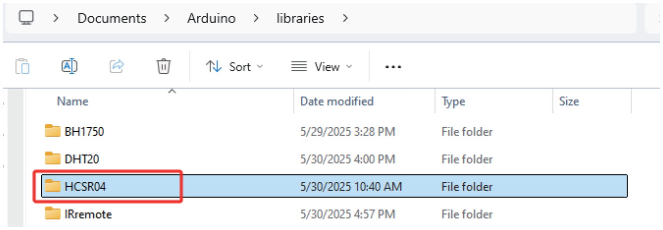
```
SR04.h> // Include the HCSR04 library tc

itions
riggerPin = 9; // Define the pin connect
choPin = 8; // Define the pin connected
bratePin = 15; // Define the pin connect

 instance of the ultrasonic sensor objec
stanceSensor distanceSensor(triggerPin,
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
Show files inside Sketches
Editor font size: 14
Interface scale: Automatic 100 %
Theme: Light

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 4:57 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

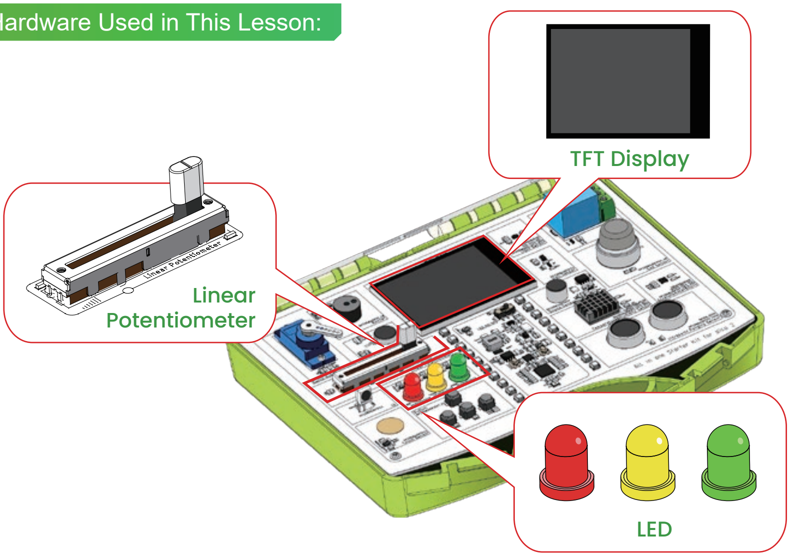
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 9 - Brightness Display

Introduction

In this chapter, you'll learn how to adjust the brightness of an LED using a slide potentiometer, dividing the brightness into 10 levels. By reading changes in the potentiometer's resistance, you'll control the output of a PWM (Pulse Width Modulation) signal, allowing for smooth, linear brightness adjustment and level-based display.

Hardware Used in This Lesson:



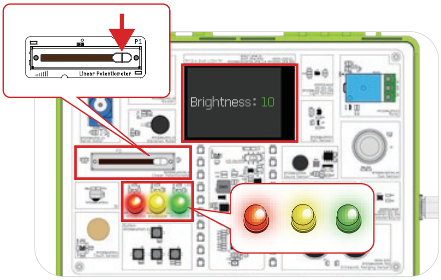
Working Principle of the PWM Dimming System

This intelligent dimming system uses a sliding potentiometer as the input signal. The MCU’s 12-bit ADC module converts the 0–900 mV analog voltage into a digital signal, which is then mapped to a 0–255 range PWM duty cycle. This signal drives RGB LEDs connected to three GPIO pins to achieve smooth brightness adjustment. An 80 MHz SPI bus is used to control a 240×320 ST7789 display in real time to show the brightness level.The hardware includes a voltage divider input circuit, PWM driving circuits (with 220 Ω current-limiting resistors for each LED), SPI display interface, and GPIO-controlled backlight. The system uses a non-blocking architecture to sample and update every 100 ms. With the help of the LVGL graphics library, efficient partial screen refreshing is achieved. Digital filtering ensures signal stability and enhances the system's responsiveness and reliability.

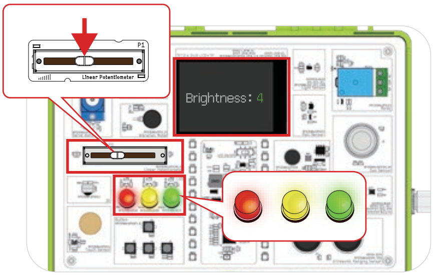
Operation Effect Diagram

Slide the potentiometer and observe both the LED brightness change and the on-screen brightness level display.

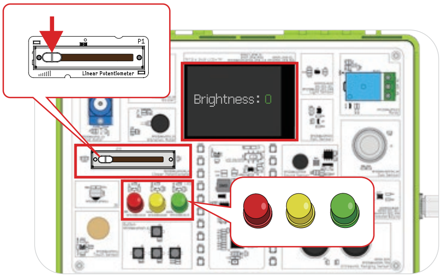
• **1. Set to 10 (Maximum Brightness):**
Slide the potentiometer and observe both the LED brightness change and the on-screen brightness level display.



• **2. Set to 4 (Medium Brightness):**
Move the potentiometer to the middle position.The LED will glow at a moderate level, and the screen will show: "Brightness: 4".



• **3. Set to 0 (Off):**
Slide the potentiometer completely to the left.The LED will turn off, and the screen will indicate: "Brightness: 0".



Key Explanations

1. Display Initialization and Configuration

```
class LGFX : public lgfx::LGFX_Device { ... }
LGFX gfx;
```

- **Custom Display Class:** A custom LGFX class is defined, inheriting from lgfx::LGFX_Device, to configure display hardware parameters (eg: SPI bus, touchscreen, etc.).
- **Display Instance:** A gfx object is instantiated to control the display.
- **Display Initialization:** In the setup() function, gfx.init() is called to initialize the display. The backlight pin (Pin 0) is set to output mode, and the backlight is turned on.
- **Screen Clear & Text Setup:** The screen is cleared using gfx.fillScreen(TFT_BLACK). Text size and color are configured, and "Brightness:" is printed on the screen.

2. Reading the Potentiometer (Analog Slider)

```
int analogValue = analogRead(SLIDER_PIN);
```

- **Pin Definition:** **SLIDER_PIN** is defined as the analog input pin connected to the potentiometer (**Pin 28**).
- **Reading Analog Values:** The potentiometer's analog value is read using **analogRead(SLIDER_PIN)**, typically returning a range of **0 to 4095** (for 12-bit ADC resolution).

3. Mapping Analog Values to PWM & Brightness Levels

```
int pwmValue = map(analogValue, 0, 900, 0, 255);
int level = map(analogValue, 0, 900, 0, 10);
```

- **PWM Value Mapping:** The potentiometer’s analog input range (0–900) is mapped to a PWM output range (0–255) to control LED brightness.
- **Brightness Level Mapping:** The analog value is also converted to an integer brightness level (0–10) for on-screen display.
- **Value Clamping:** The constrain() function ensures mapped values stay within valid ranges: PWM output (0–255) and brightness level (0–10).

4. LED PWM Control

```
analogWrite(RED_LED_PIN, pwmValue);
analogWrite(YELLOW_LED_PIN, pwmValue);
analogWrite(GREEN_LED_PIN, pwmValue);
```

- **Pin Definitions:** Pins connected to the red, yellow, and green LEDs are defined for PWM output.
- **PWM Output:** The mapped PWM value (0–255) is sent to the LED pins using analogWrite(), controlling LED brightness.

5. Displaying Brightness Level on Screen

```
if (level != lastLevel) {
  lastLevel = level;
  gfx.fillRect(240, 100, 60, 30, TFT_BLACK);
  gfx.setCursor(240, 100);
  gfx.setTextColor(TFT_GREEN);
  gfx.print(level);
}
```

- **Brightness Level Change Detection:** The display updates only if the current brightness level differs from the previously recorded lastLevel.
- **Clearing Screen & Setting Cursor:** gfx.fillRect() clears the previous brightness level display area.The cursor position is reset for new text output.
- **Displaying Brightness Level:** The text color is set to green, and the new brightness level is printed.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After learning the above code, a brightness adjustment system based on a slide potentiometer is implemented. The system reads the analog value of the slide potentiometer, maps it to a brightness level from 0 to 10, controls the brightness of red, yellow, and green tri-color LEDs, and real-time displays the brightness level on an ST7789 display, while supporting serial port debugging information output.The code implements an LED brightness adjustment system based on a slide potentiometer. By integrating analog signal acquisition and PWM control technology, the brightness is divided into 10 levels, which are real-timely displayed on the TFT display. It also supports touch interaction functions.

Programming Steps

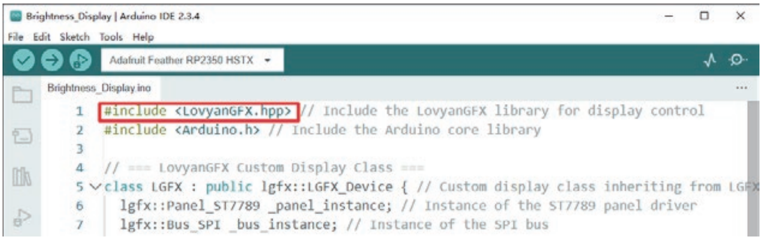
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**LovyanGFX-develop**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

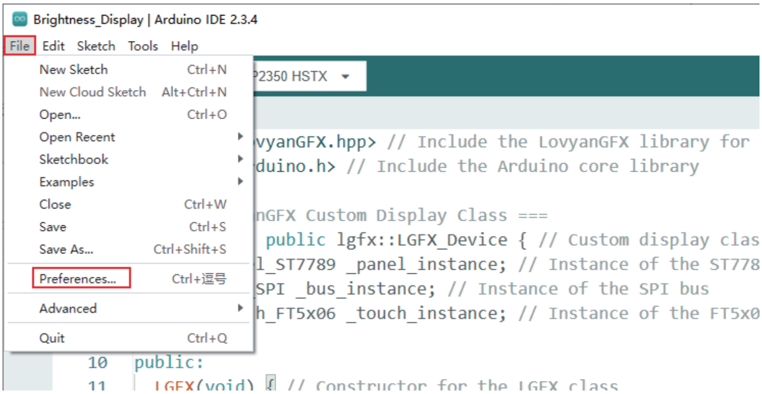
• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop** and **LovyanGFX-develop** folder.


Brightness_Display | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
Brightness_Display.ino

```
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control
#include <Arduino.h> // Include the Arduino core library

// === LovyanGFX Custom Display Class ===
class LGFX : public lgfx::LGFX_Device { // Custom display class inheriting from LGFX
  lgfx::Panel_ST7789 _panel_instance; // Instance of the ST7789 panel driver
  lgfx::Bus_SPI _bus_instance; // Instance of the SPI bus
```

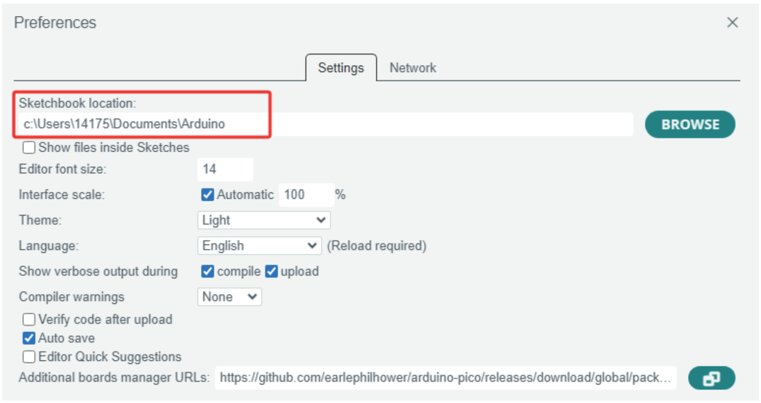
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
vyanGFX.hpp> // Include the LovyanGFX library for
duino.h> // Include the Arduino core library

nGFX Custom Display Class ===
 public lgfx::LGFX_Device { // Custom display clas
l_ST7789 _panel_instance; // Instance of the ST778
SPI _bus_instance; // Instance of the SPI bus
h_FT5x06 _touch_instance; // Instance of the FT5x0

public:
  LGFX(void) { // Constructor for the LGFX class
```

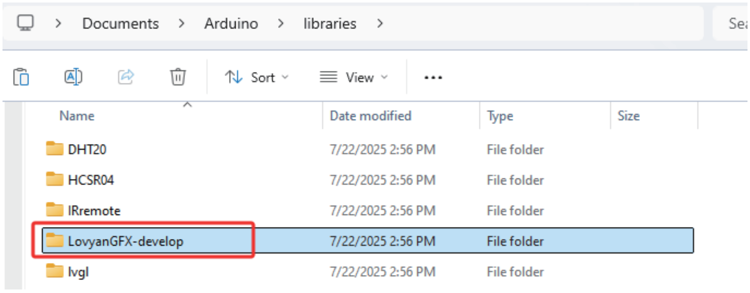
• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.


Preferences

Settings | Network

Sketchbook location:
c:\Users\14175\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ☑ compile ☑ upload
Compiler warnings: None
☐ Verify code after upload
☑ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.


Documents > Arduino > libraries >

Name	Date modified	Type	Size
DHT20	7/22/2025 2:56 PM	File folder	
HCSR04	7/22/2025 2:56 PM	File folder	
IRremote	7/22/2025 2:56 PM	File folder	
LovyanGFX-develop	7/22/2025 2:56 PM	File folder	
lvgl	7/22/2025 2:56 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

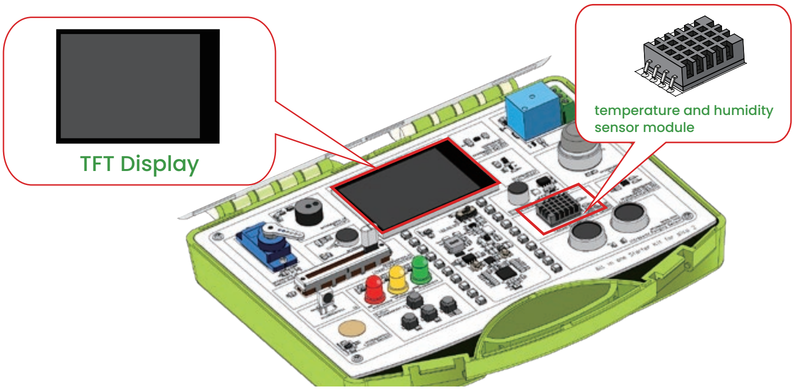
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 10 - Temperature & Humidity Detecting System

Introduction

This chapter will focus on the usage of temperature and humidity sensor modules, including reading sensor data and real-time displaying temperature and humidity values on the screen. Through practice, you will master the I2C or single-wire communication protocol, as well as the interactive logic of data acquisition and display.

Hardware Used in This Lesson:



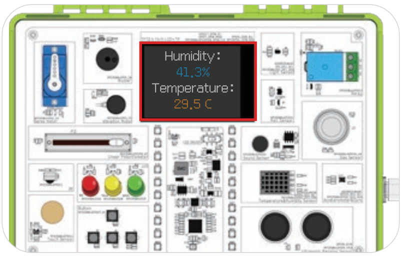
Working Principle of Temperature and Humidity Monitoring Display System

The system is based on an SPI-driven 240×320 ST7789 LCD with an FT5x06 touchscreen, and connects to a DHT20 temperature-humidity sensor via I2C (Wire1). The main controller (eg: ESP32) reads temperature and humidity data once every 1 second. After processing by the LovyanGFX library, the data is centrally displayed on the screen (humidity in blue, temperature in red). The SPI is configured for 80MHz high-speed communication, while I2C operates at 400kHz. The touchscreen is controlled via INT/RST pins. GPIO0 drives the backlight, and non-blocking timing (millis()) ensures real-time refresh to avoid screen afterimages.

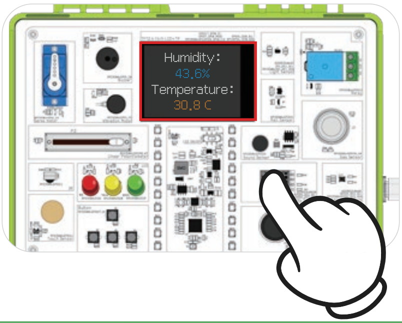
Operation Effect Diagram

After Starting the Program,The system will continuously collect data from the temperature-humidity sensor and display it on the screen in real time.

• **1. Normal Environment Observation**: Place the sensor in a normal room-temperature environment and observe the temperature and humidity data displayed on the screen.



• 2. Gently press your finger against the surface of the DHT20 temperature-humidity sensor and observe the data changes on the display.



Key Explanations

1. LGFX Class Constructor

```
LGFX(void) {
  auto cfg = _bus_instance.config();
  cfg.spi_host = 0;
  cfg.spi_mode = 0;
  cfg.freq_write = 80000000;
  cfg.pin_sclk = 6;
  cfg.pin_mosi = 7;
  cfg.pin_miso = -1;
  cfg.pin_dc = 16;
  _bus_instance.config(cfg);
  _panel_instance.setBus(&_bus_instance);
  ...
  _panel_instance.setTouch(&_touch_instance);
  setPanel(&_panel_instance);
}
```

- **Custom Display Class: This is the constructor of the custom LGFX class, serving as the core for initializing the entire LovyanGFX screen.**
- **cfg.spi_host = 0;:** Selects VSPI (SPI0 of ESP32)
- **cfg.freq_write = 80000000;:** Sets high-frequency write speed (80MHz) to enhance screen refresh rate.
- **cfg.pin_*:** Binds specific screen pins (SCLK, MOSI, DC, etc.).

- **_panel_instance.setBus(...):** Binds the SPI bus to the screen panel.
- **_panel_instance.setTouch(...):** Mounts the touchscreen driver to the panel.
- **setPanel(...):** Finally applies the configuration to LGFX_Device.
- **Function:** Constructs and initializes the hardware abstraction layer for SPI display and touchscreen, serving as the foundation for using the LovyanGFX library.

2. Sensor and I2C Initialization

```
Wire1.setSDA(2);
Wire1.setSCL(3);
Wire1.begin();
DHT.begin();
```

- This section is responsible for initializing the second I2C bus Wire1 for connecting the DHT20 sensor.
- **Wire1.setSDA(2); and setSCL(3);:** Assigns SDA and SCL pins to GPIO2 and GPIO3 to avoid conflicts with the main I2C bus (eg: the I2C used by the touchscreen).
- **DHT.begin();:** Invokes the initialization function of the DHT20 library.

Function: Enables the I2C communication channel with DHT20 to facilitate reading temperature and humidity data.

3. Timed Reading and Screen Update

```
if (currentMillis - preMillis >= interval) {
  if (millis() - DHT.lastRead() >= 1000) {
    int status = DHT.read();
    if (status == DHT20_OK) {
      gfx.fillScreen(TFT_BLACK);
      ...
      gfx.printf("%.1f %%", humidity);
      gfx.printf("%.1f C", temperature);
    }
  }
  preMillis = currentMillis;
}
```

- **This is the core part of the loop():**

  • Execute DHT.read() every 1000ms (1 second).

  • If the reading is successful (DHT20_OK), clear the screen and display the latest temperature and humidity data.

  • Use gfx.printf() combined with setCursor() to center the data display.

- **Function:** Implement the real-time reading of temperature/humidity and the refresh logic for graphical display.

4. Centered Display Technique

```
int centerX = screenWidth / 2;
gfx.setCursor(centerX - 6 * 3 * 5, 60);
gfx.print("Humidity:");
gfx.setCursor(centerX - 6 * 3 * 2, 100);
gfx.printf("%.1f %%", humidity);
```

- centerX is the horizontal midpoint coordinate of the screen.
- 6 * 3 * N is the method to estimate text width:
  - Each character is approximately 6 pixels wide (default font).
  - 3 is the text magnification factor (set via setTextSize(3)).
- • N is the number of characters (estimated based on actual content).

  Function: Achieve horizontally centered text alignment by manually calculating the starting coordinates.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After learning the above code, an Arduino-based environmental monitoring system is implemented. The system uses a DHT20 sensor to collect real-time temperature and humidity data, which is visually displayed on an ST7789 screen. The system updates data once per second, presenting the current ambient humidity (in percentage) and temperature (in Celsius) in a clear and readable manner, with different colors used to distinguish various types of data.

Programming Steps

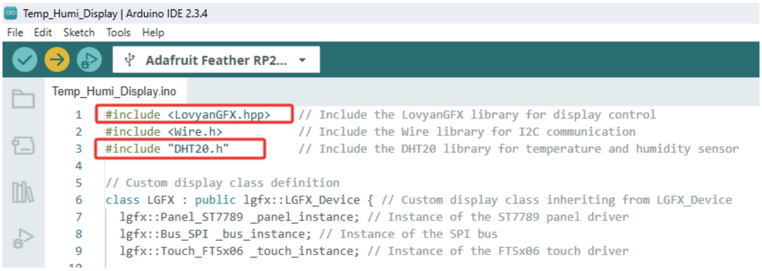
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we used two additional library files: **DHT20 and LovyanGFX-develop**.,so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

• Navigate to the **/example/libraries** directory and download the **DHT20 and LovyanGFX-develop** folder.

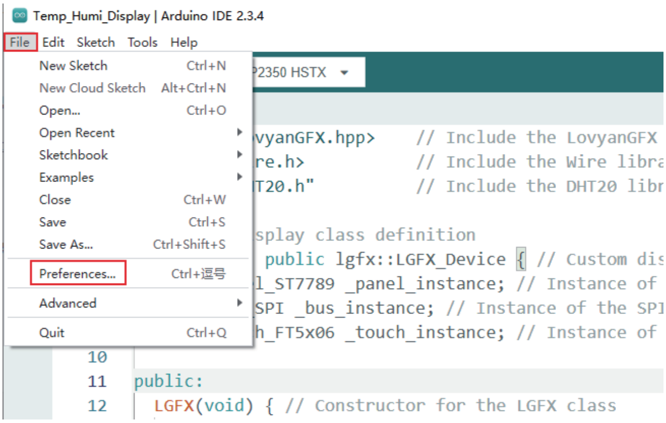


Temp_Humi_Display | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2...
Temp_Humi_Display.ino

```
#include <LovyanGFX.hpp>    // Include the LovyanGFX library for display control
#include <Wire.h>           // Include the Wire library for I2C communication
#include "DHT20.h"          // Include the DHT20 library for temperature and humidity sensor

// Custom display class definition
class LGFX : public lgfx::LGFX_Device { // Custom display class inheriting from LGFX_Device
  lgfx::Panel_ST7789 _panel_instance; // Instance of the ST7789 panel driver
  lgfx::Bus_SPI _bus_instance; // Instance of the SPI bus
  lgfx::Touch_FT5x06 _touch_instance; // Instance of the FT5x06 touch driver
```

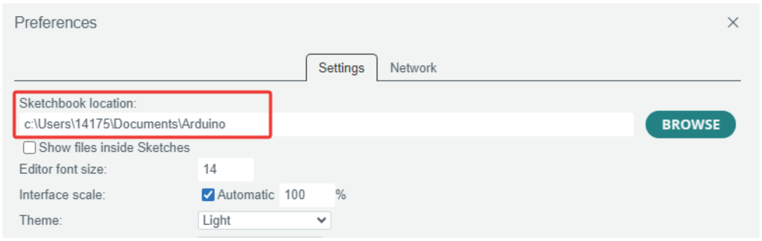
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
vyanGFX.hpp>      // Include the LovyanGFX
re.h>             // Include the Wire libra
T20.h"            // Include the DHT20 libr

splay class definition
 public lgfx::LGFX_Device { // Custom dis
l_ST7789 _panel_instance; // Instance of
SPI _bus_instance; // Instance of the SPI
h_FT5x06 _touch_instance; // Instance of

public:
  LGFX(void) { // Constructor for the LGFX class
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.


Preferences

Sketchbook location:
c:\Users\14175\Documents\Arduino

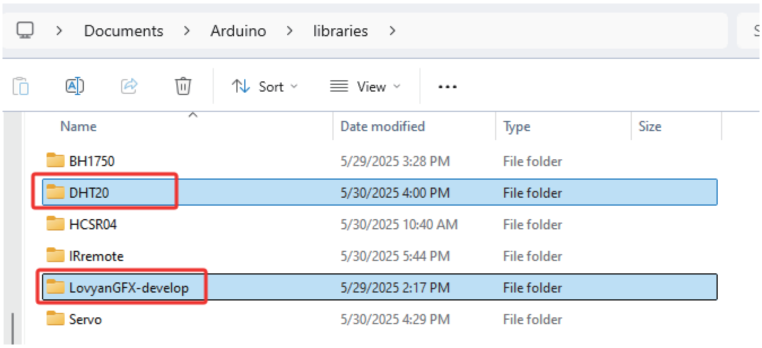
☐ Show files inside Sketches

Editor font size: 14

Interface scale: ✓ Automatic 100 %

Theme: Light

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.


> Documents > Arduino > libraries >

Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
Servo	5/30/2025 4:29 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

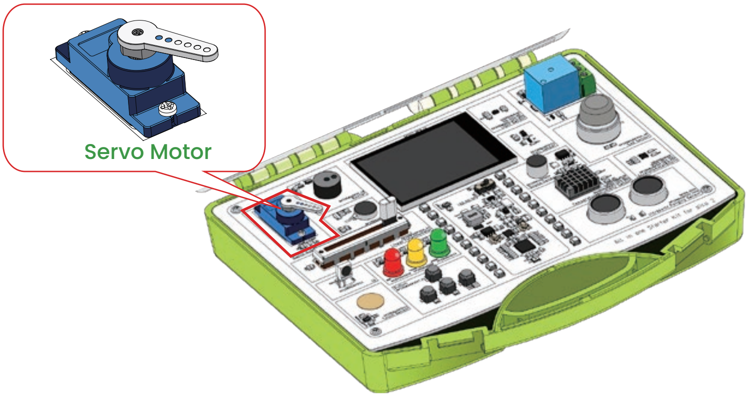
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 11 - Servo Control

Introduction

This section will elaborate on the control method of the servo motor module, realizing the reciprocating swing of the servo within the range of 0 to 180 degrees by configuring PWM control signals. During the course operation, the servo will complete a periodic motion of rotating from 0 degrees to 180 degrees and then reversing back to 0 degrees according to the preset logic.

Hardware Used in This Lesson:



Working Principle of Servo Motor Control

The system uses the Arduino Servo library to drive a servo motor connected to pin GP13, employing PWM signals to control the rotation angle. The pulse width is set to range from 450 to 2520 microseconds, corresponding to a mechanical rotation of 0-180 degrees. The main loop implements a progressive angle adjustment, incrementing or decrementing by 1 degree every 15 milliseconds to achieve smooth reciprocating motion.

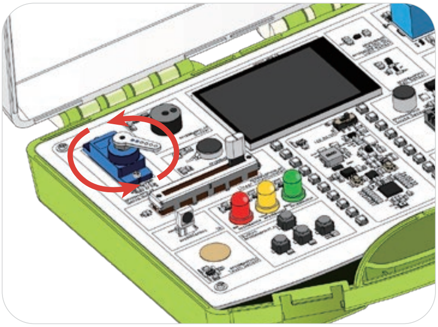
Operation Effect Diagram

• **Forward Rotation:**

After the program starts, the servo motor will slowly rotate from the 0-degree position to the 180-degree position.

• **Reverse Rotation:**

Upon reaching the 180-degree position, the servo motor will automatically reverse its rotation and return to the 0-degree position, forming a continuous reciprocating swing cycle.



Key Explanations

1. LGFX Class Constructor

```
#include <Servo.h>
Servo myservo;
```

▸ **#include <Servo.h>:** Loads the official Arduino Servo library, which encapsulates the PWM generation and timing control functions required for communicating with servo motors.

▸ **Servo myservo;:** Instantiates a Servo object named myservo in the global scope. Subsequent operations on myservo (such as attach() and write()) will be mapped to specific hardware pins and signals of the servo motor.

2. Binding the Servo to a Pin and Setting Pulse Width Range in setup()

```
void setup() {
  myservo.attach(13, 450, 2520);
}
```

▸ **myservo.attach(pin, minPulse, maxPulse);**

• pin=13: Connect the servo signal wire to Arduino digital pin 13.

• 450 (minimum pulse width): Unit is microseconds (μs). The pulse width for 0° of a standard servo is ~500μs; adjusting to 450μs ensures the servo can reach or slightly exceed the minimum angle.

• 2520 (maximum pulse width): Corresponds to the servo's maximum angle (180°). The standard value is ~2500μs; setting to 2520μs ensures the servo can reach or slightly exceed the maximum angle.

▸ **Function:** By specifying minPulse/maxPulse, calibrate the 0° and 180° output positions to the servo's actual physical limits. If these parameters are omitted, the library uses default values (544 μs-2400μs), which may need adjustment for servos with different travel ranges.

3. Forward and Reverse Sweep For Loops in loop()

```
void loop() {
  for (int pos = 0; pos <= 180; pos++) {
    myservo.write(pos);
    delay(15);
  }
  for (int pos = 180; pos >= 0; pos--) {
    myservo.write(pos);
    delay(15);
  }
}
```

▸ **1. for (int pos = 0; pos <= 180; pos++)**

- This loop increments pos from 0 to 180 in steps of 1°. Combined with myservo.write(pos), it enables the servo to "smoothly rotate from the minimum angle to the maximum angle".
- myservo.write(pos): Passes the current angle value (in degrees) to the Servo library, which converts it to the corresponding pulse width signal (450μs→0°, 2520μs→180°) and outputs it to the servo within a 20ms cycle.
- delay(15): Pauses for 15ms after each angle update to allow the servo sufficient time to reach the target position. A 15ms delay with a 1° step produces a smooth sweeping motion. Larger delays result in slower rotation, while smaller delays increase speed but may cause jitter or stuttering.

▸ **2.for (int pos = 180; pos >= 0; pos--)**

- Reverse logic of the previous loop, decrementing pos from 180 to 0 to rotate the servo "from the maximum angle back to the minimum angle".
- Similarly uses myservo.write(pos) to update the angle and delay(15) to control speed.

▸ **3. Function:** These two opposing for loops create a complete "sweep-and-return" operation, causing the servo to oscillate back and forth.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you have implemented a servo control system that enables the servo to perform slow reciprocating swings between 0 and 180 degrees. Starting from 0 degrees, the servo gradually rotates to 180 degrees at a rate of 15 milliseconds per step, then returns to 0 degrees at the same speed, repeating this cycle continuously.

Programming Steps

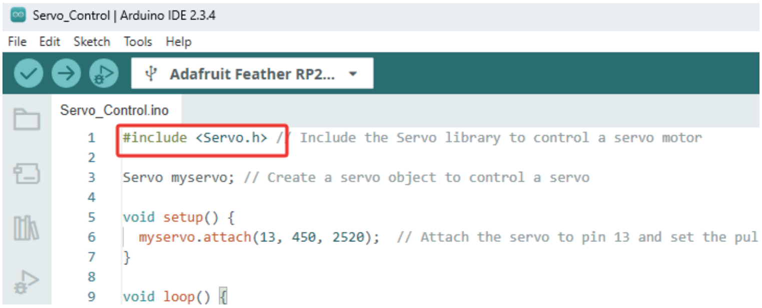
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**Servo**),so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

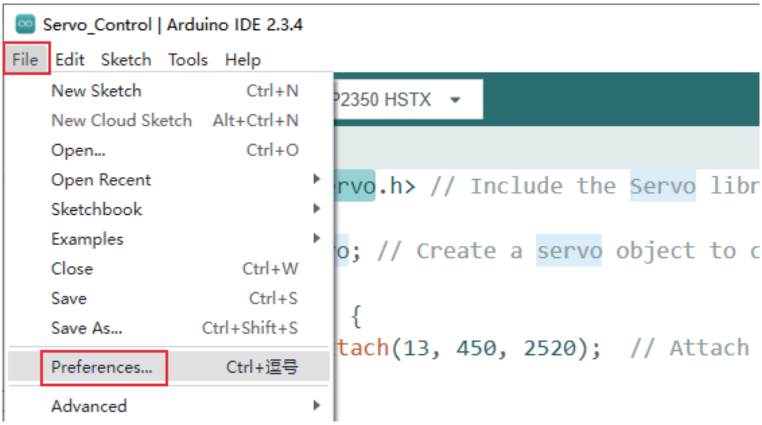
• Navigate to the **/example/libraries** directory and download the **Servo** folder.


Servo_Control | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2...
Servo_Control.ino

```
#include <Servo.h> // Include the Servo library to control a servo motor

Servo myservo; // Create a servo object to control a servo

void setup() {
  myservo.attach(13, 450, 2520);  // Attach the servo to pin 13 and set the pul
}

void loop() {
```

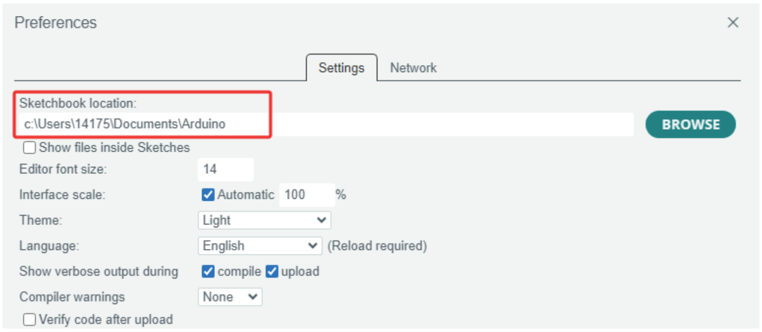
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
rvo.h> // Include the Servo libr
o; // Create a servo object to c
{
tach(13, 450, 2520);  // Attach
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.


Preferences

Settings | Network

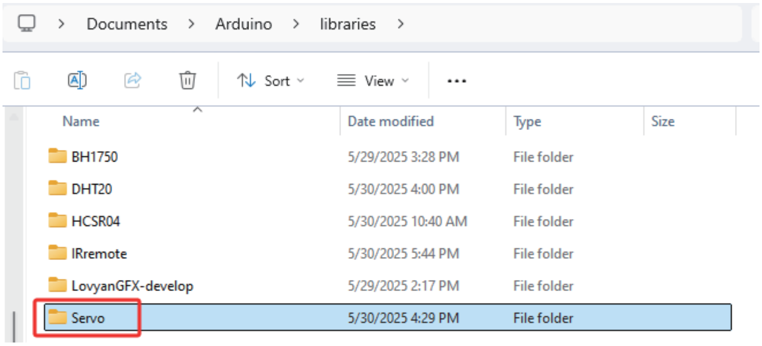
Sketchbook location:
c:\Users\14175\Documents\Arduino
BROWSE

☐ Show files inside Sketches

Editor font size:	14
Interface scale:	✓ Automatic 100 %
Theme:	Light
Language:	English (Reload required)
Show verbose output during	✓ compile ✓ upload
Compiler warnings	None

☐ Verify code after upload

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.


Documents > Arduino > libraries >

Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
Servo	5/30/2025 4:29 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

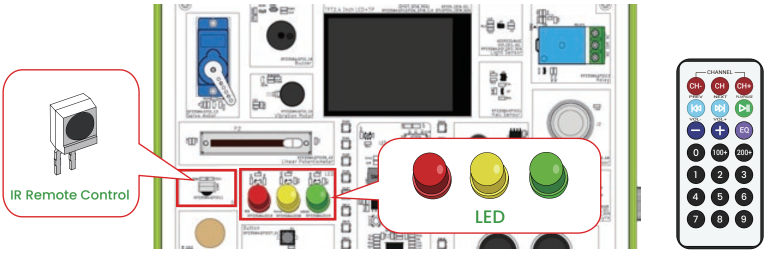
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 12 – IR control LED

Introduction

This section will focus on learning how to implement diversified control of LEDs using an infrared remote control, including single-color LED lighting, multi-color LED flashing, and running light effects. You will master the interactive logic between infrared signal reception and LED driving.

Hardware Used in This Lesson:



Principle of Infrared Remote Sensor Operation

The infrared remote sensor works by receiving infrared signals to achieve control functions. The transmitter modulates control instructions into infrared light signals of a specific frequency. The sensor receives these signals, demodulates them back into control instructions, and then the microcontroller performs the corresponding operations.

Working Principle of LED

At the heart of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During this recombination, electrons drop from a higher energy level to a lower one, releasing the excess energy in the form of photons—producing light. The color (or wavelength) of the emitted light is determined by the energy band gap of the semiconductor material. This process is a direct application of electroluminescence.

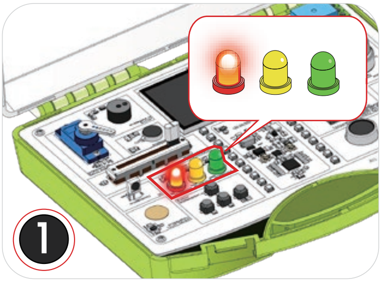
Principle of Infrared Remote Control Operation

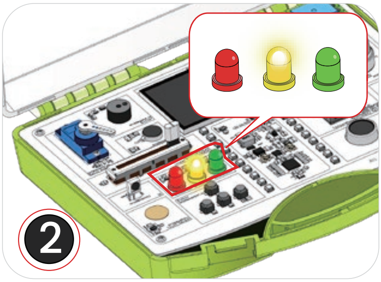
The infrared remote control works by inputting control commands through buttons. The microcontroller encodes and modulates these commands into infrared signals, which are then transmitted by an infrared emitter. The receiving end's infrared sensor captures the signals, demodulates them back into commands, and drives the device to perform the corresponding operations.

Operation Effect Diagram

• Press "1": Red LED lights up;

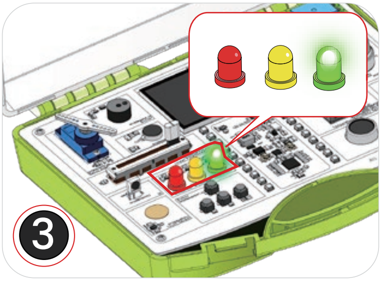
• Press "2": Yellow LED lights up;





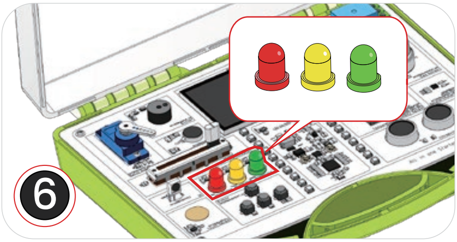
• Press "3": Green LED lights up;

• Press "4": Red, yellow, and green LEDs flash synchronously;



• Press "5": The three-color LEDs light up sequentially to create a chasing light effect;

• Press "6": All LEDs turn off.



Key Explanations

1. Hardware Connections and Library Inclusion

```
if (IrReceiver.decode()) {
  unsigned long code = IrReceiver.decodedIRData.command;
  handleIR(code);
  IrReceiver.resume();
}
```

▸ **Function:** Receive infrared signals from TV/air conditioner remotes, etc.

▸ **Key Points:**

• decode() acts like "ears" to monitor signals.
• command serves as a unique identifier for each button (eg: 0x0C = Button 1).
• resume() functions as a "continue listening" instruction.

2. Button Command Processing (Control Hub)

```
void handleIR(unsigned long code) {
  turnOffAll();
  effectMode = 0;
  switch(code) {
    case 0x0C: digitalWrite(RED_LED_PIN, HIGH); break;
    case 0x18: digitalWrite(YELLOW_LED_PIN, HIGH); break;
    case 0x5E: digitalWrite(GREEN_LED_PIN, HIGH); break;
    case 0x08: effectMode = 1; break;
    case 0x1C: effectMode = 2; break;
    case 0x5A: turnOffAll(); break;
  }
}
```

▸ **Core Logic:**

• First three buttons: Control red/yellow/green LEDs individually.
• Buttons 4/5: Trigger two lighting effects.
• Button 6: Emergency shutdown for all LEDs.

▸ **Special Design:**

• Call turnOffAll() before each button press to avoid conflicts.
• Use effectMode variable to track the current lighting effect.

3. Lighting Effect Processing (Dynamic Effects)

```
void handleEffects() {
  if(effectMode == 1) {
    if(millis() - lastEffectTime > 300) {
      bool on = !on;
      digitalWrite(RED_LED_PIN, on);
      digitalWrite(YELLOW_LED_PIN, on);
      digitalWrite(GREEN_LED_PIN, on);
      lastEffectTime = millis();
    }
  }
  else if(effectMode == 2) {
    if (millis() - lastEffectTime > 300) {
      switch (effectStep % 3) {
        case 0:
          digitalWrite(RED_LED_PIN, HIGH);
          digitalWrite(YELLOW_LED_PIN, LOW);
          digitalWrite(GREEN_LED_PIN, LOW);
          break;
        case 1:
          digitalWrite(RED_LED_PIN, LOW);
          digitalWrite(YELLOW_LED_PIN, HIGH);
          digitalWrite(GREEN_LED_PIN, LOW);
          break;
        case 2:
          digitalWrite(RED_LED_PIN, LOW);
          digitalWrite(YELLOW_LED_PIN, LOW);
          digitalWrite(GREEN_LED_PIN, HIGH);
          break;
      }
      effectStep++;
      lastEffectTime = millis();
    }
  }
}
```

▶ **Flashing Mode:**

• Toggle all LEDs on/off simultaneously every 300ms.

• Use on = !on to invert the state (on → off → on).

▶ **Chasing Light Mode:**

• Light up LEDs sequentially in the order: red → yellow → green.

• Use effectStep % 3 to cycle between 0-2.

• Increment effectStep++ to advance the lighting sequence.

4. Auxiliary Function

```
void turnOffAll() {
  digitalWrite(RED_LED_PIN, LOW);
  digitalWrite(YELLOW_LED_PIN, LOW);
  digitalWrite(GREEN_LED_PIN, LOW);
}
```

▶ **Key Role:** Ensure all lights can be immediately stopped at any time.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the provided code, you have successfully implemented an infrared remote-controlled LED system. This system receives specific infrared signals (corresponding to buttons 1-6 on the remote) to control the on/off states and display modes of three LEDs (red, yellow, and green). The detailed functions are as follows:

• Buttons 1-3: Turn on the corresponding LED (red, yellow, or green, respectively).
• Button 4: Activates synchronized blinking of all three LEDs.
• Button 5: Triggers a running-light effect (sequential cycling through the three LEDs).
• Button 6: Turns off all LEDs.

Programming Steps

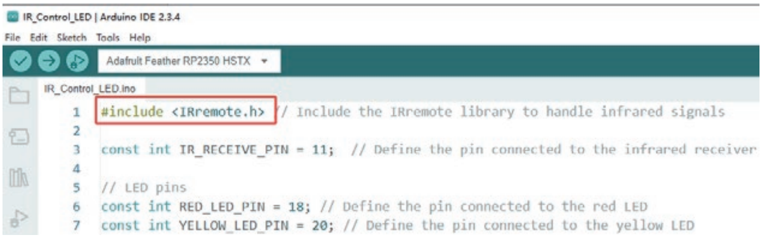
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**IRremote**),so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

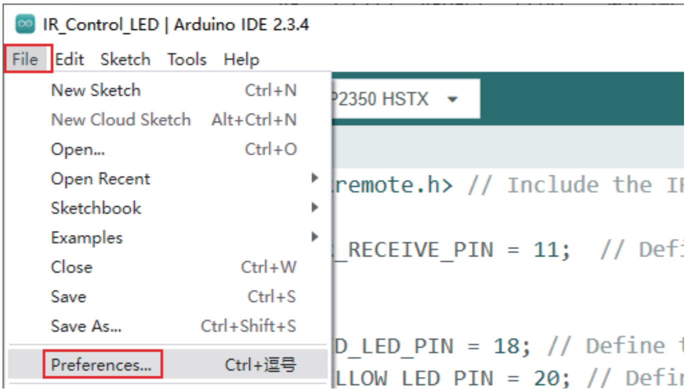
• Navigate to the **/example/libraries** directory and download the **IRremote** folder.


IR_Control_LED | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
IR_Control_LED.ino

```
#include <IRremote.h> // Include the IRremote library to handle infrared signals

const int IR_RECEIVE_PIN = 11;  // Define the pin connected to the infrared receiver

// LED pins
const int RED_LED_PIN = 18; // Define the pin connected to the red LED
const int YELLOW_LED_PIN = 20; // Define the pin connected to the yellow LED
```

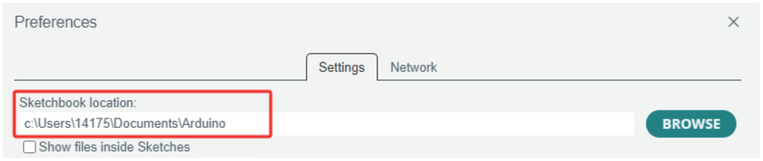
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



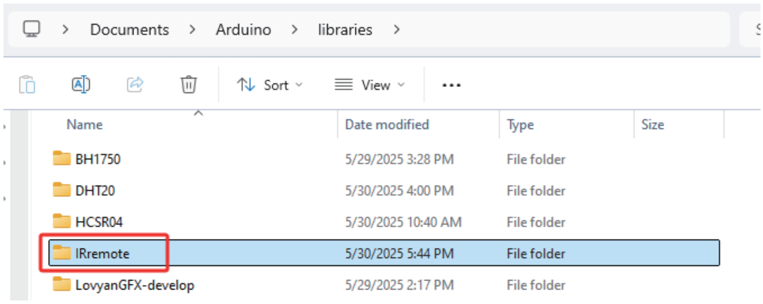
```
remote.h> // Include the IF
_RECEIVE_PIN = 11;   // Def
D_LED_PIN = 18; // Define 1
LLOW LED PIN = 20; // Defin
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
Show files inside Sketches

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

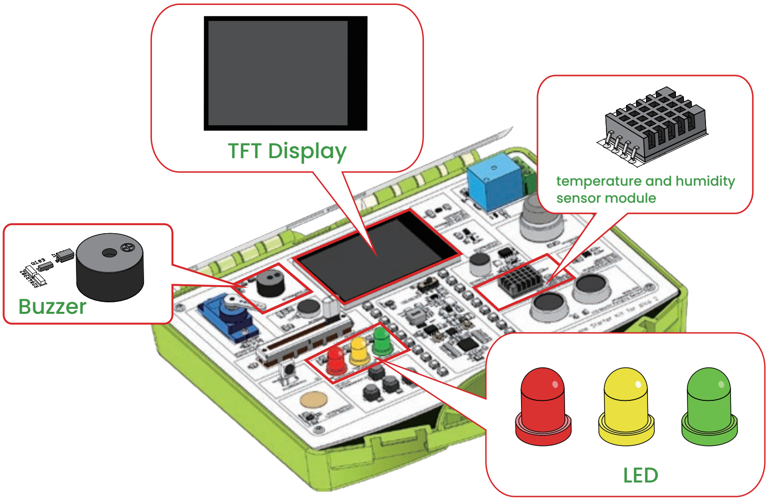
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 13 - Weather Reminder

Introduction

This project is based on the All_in_one_Starter_Kit_for_Pico2 development board. It uses a DHT20 sensor to collect real-time environmental temperature and humidity data, and controls a TFT display, LED lights, and a buzzer based on set thresholds to achieve graphical display of environmental conditions and abnormal warning functions. It’s ideal for basic learning and practical applications in intelligent environmental monitoring scenarios. The project uses SquareLine Studio to design the screen interface and employs the LVGL library for display and interaction control, enabling dynamic visualization of temperature and humidity information and interface prompts for abnormal conditions.

Hardware Used in This Lesson:



Working Principle of Temperature and Humidity Sensor

The sensor measures environmental parameters via sensing elements. Temperature detection relies on thermistors or semiconductor materials whose resistance/voltage varies with temperature. Humidity sensing typically uses capacitive or resistive elements with dielectric constant/resistance changes with moisture. Built-in signal conditioning converts analog signals to digital output for MCU processing.

Principle of Buzzer Operation

The buzzer generates sound by vibrating a diaphragm driven by an electrical signal. When an alternating current is applied, the diaphragm vibrates rapidly due to magnetic or piezoelectric effects, producing sound. The pitch and frequency are determined by the current frequency, and it is commonly used for alerts or alarms.

Working Principle of LED

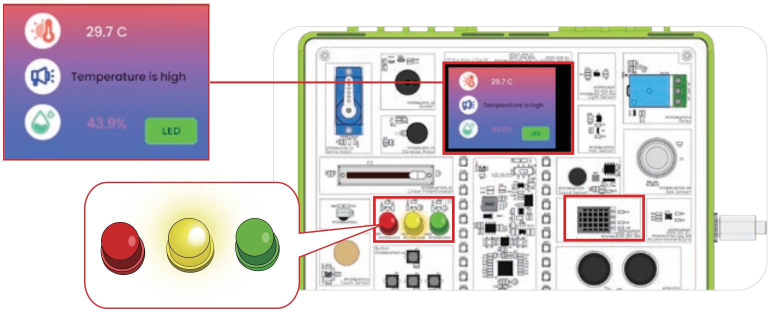
At the heart of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During this recombination, electrons drop from a higher energy level to a lower one, releasing the excess energy in the form of photons—producing light. The color (or wavelength) of the emitted light is determined by the energy band gap of the semiconductor material. This process is a direct application of electroluminescence.

Working Principle of Display Screen

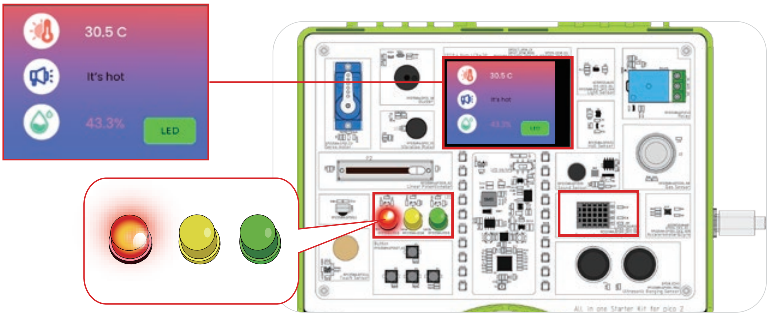
The display controls each pixel's brightness/color via driver circuits. The main IC converts image data to electrical signals, transmitted to row/column driver ICs for line-by-line refreshing. LCDs adjust liquid crystal alignment with voltage to modulate light, while OLEDs emit light directly. A timing controller ensures signal synchronization to prevent artifacts.

Operation Effect Diagram

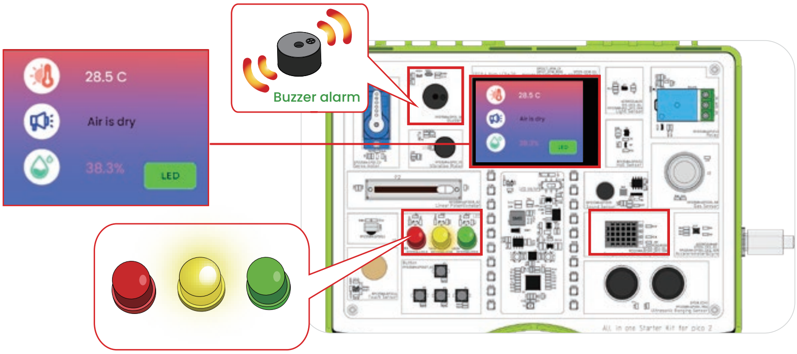
• 1. When the temperature from the DHT20 sensor exceeds 25°C, the yellow LED should turn on, and the TFT display shows "Temperature is high".



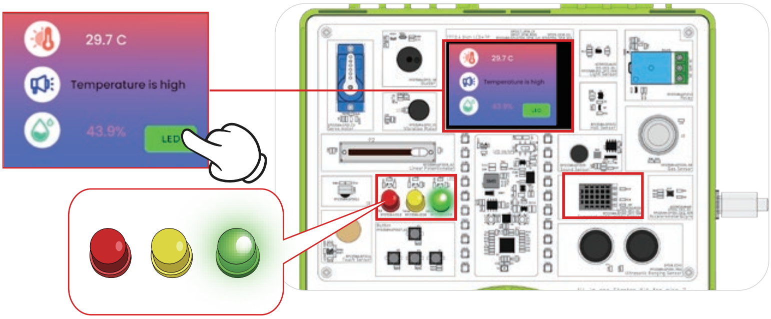
• 2. When the temperature rises above 30°C, the red LED turns on, the TFT display shows "It's hot", and the yellow LED turns off.



• 3. When humidity drops below 40%, the buzzer sounds, the TFT display shows "Air is dry", and the yellow LED turns on if the temperature exceeds 25°C.



• 4. Press the LED button once, the green LED turns on. Press the LED button again, the green LED turns off.



Before we begin designing the user interface we need, it's important to understand a key concept—LVGL. LVGL is a lightweight graphics library that gives us powerful tools to build intuitive and visually appealing interfaces. It forms the core of SquareLine Studio and serves as the foundation for the design work ahead.
Once we grasp the basics of how LVGL works, we'll move on to hands-on practice: installing SquareLine Studio, designing a graphical interface, and exporting it into runnable code.
So, let's start by exploring the concept of LVGL—this will be the first building block of our UI design journey.

What is LVGL?

LVGL (Light and Versatile Graphics Library) is a free and open-source graphics library designed specifically for embedded systems with limited resources. It provides all the essential features needed to build smooth and visually appealing user interfaces, including a wide range of widgets, graphic effects, animations, and event handling. LVGL is lightweight, highly portable, and supports hardware acceleration—making it a powerful choice for microcontroller-based projects.

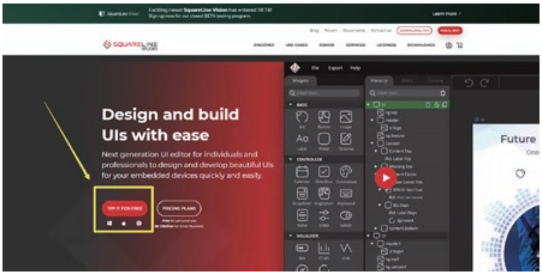
SquareLine Studio

SquareLine Studio is a visual UI design tool (IDE) built on top of LVGL. It features an intuitive drag-and-drop interface designer, real-time previews, and powerful property and event editors—all of which allow developers to create embedded interfaces without writing a lot of low-level code. The tool automatically generates C/C++ code based on LVGL, greatly speeding up development and delivering a true "what-you-see-is-what-you-get" design experience.

Installing SquareLine Studio

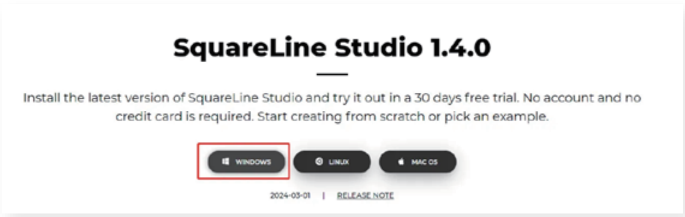
1. Visit the official website

Open your web browser and go to the official SquareLine Studio download page:
SquareLine Studio: https://squareline.io/


Design and build
UIs with ease
Next generation UI editor for individuals and professionals to design and develop beautiful UIs for your embedded devices quickly and easily.
Future

2. Choose the right version

Select the installation package that matches your operating system (Windows, macOS, or Linux).
For this guide, we'll use the Windows version as an example. It's recommended to download version 1.3.4, as it includes a simulation feature that's missing in version 1.4.

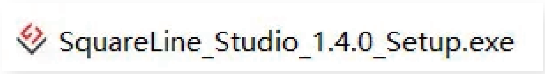


SquareLine Studio 1.4.0

Install the latest version of SquareLine Studio and try it out in a 30 days free trial. No account and no credit card is required. Start creating from scratch or pick an example.

3. Download the installer

Click the download button and wait for the download to finish. Once it's complete, double-click the setup.exe file.

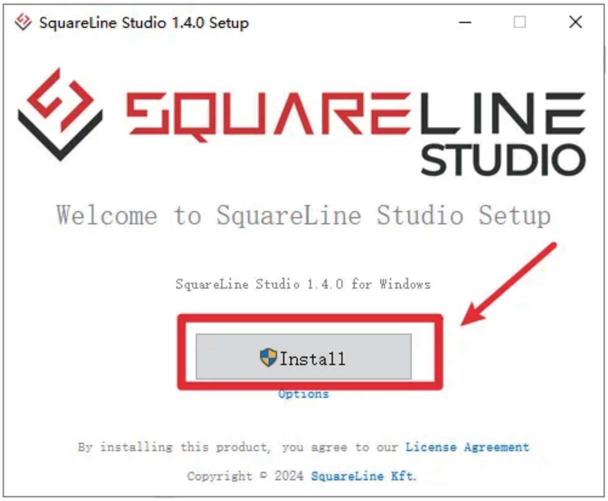


SquareLine_Studio_1.4.0_Setup.exe

4. Run the installer

Follow the prompts in the installation wizard.

Click "Install" .



SquareLine Studio 1.4.0 Setup

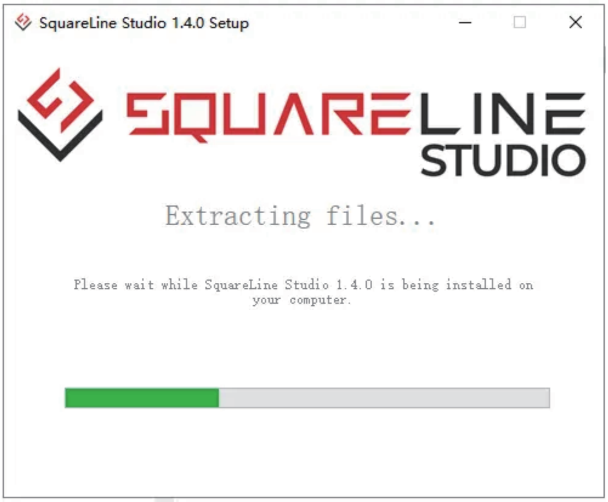
Welcome to SquareLine Studio Setup

SquareLine Studio 1.4.0 for Windows

By installing this product, you agree to our License Agreement

Copyright © 2024 SquareLine Kft.

wait for the installation process to complete.

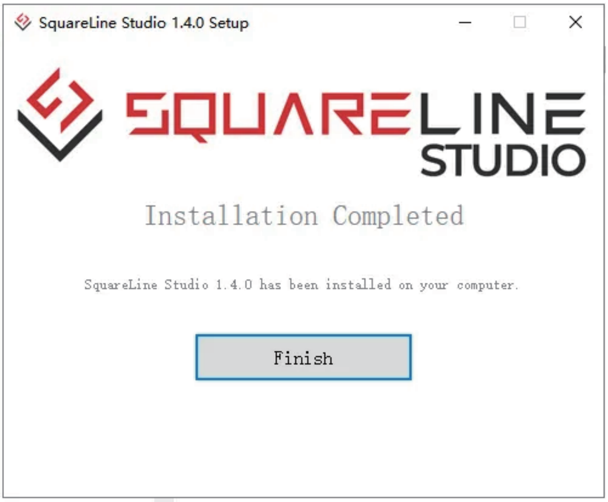

SquareLine Studio 1.4.0 Setup

Extracting files...

Please wait while SquareLine Studio 1.4.0 is being installed on your computer.

• 5. Finish installation

Installation Complete.


SquareLine Studio 1.4.0 Setup

Installation Completed

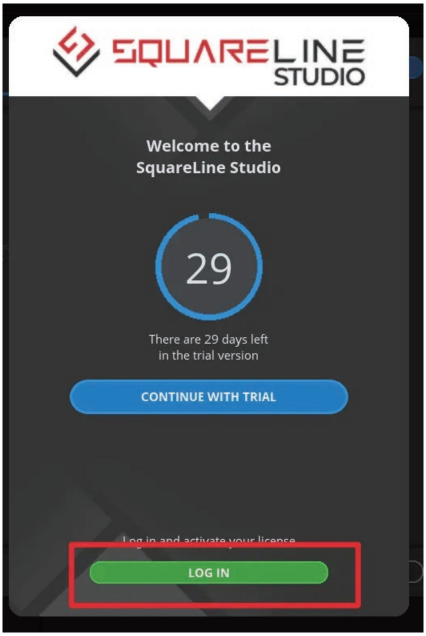
SquareLine Studio 1.4.0 has been installed on your computer.

Project Creation and UI Design with Squareline Studio

A detailed operational guide for designing UI with Squareline Studio and integrating it with Arduino code, covering the complete process of project creation, material addition, label setup, code export, and main program integration:

1. Launch Squareline Studio

Register or log in to an account as prompted (a 30-day trial period is available for first-time use. Please register an account as instructed. When you log in to your account next time, you will continue to use it.)



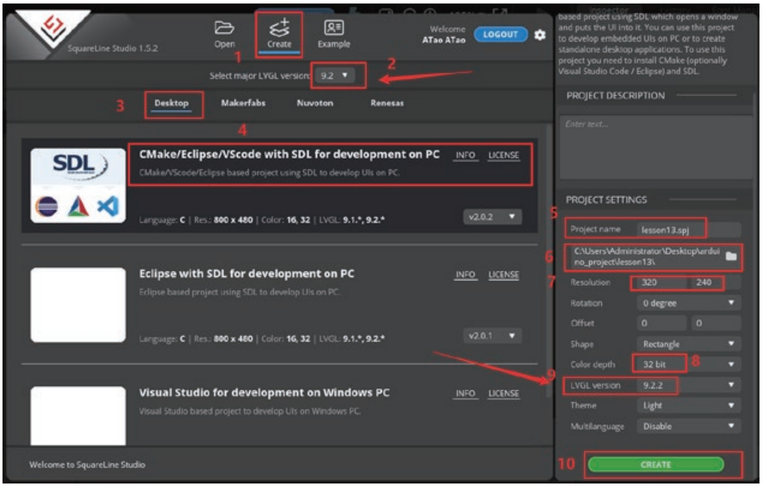
Welcome to the SquareLine Studio

29

There are 29 days left in the trial version

Log in and activate your license

Create a new project by following these steps:



Select major LVGL version: 9.2

CMake/Eclipse/VScode with SDL for development on PC
CMake/VScode/Eclipse based project using SDL to develop UIs on PC.
Language: C | Res.: 800 x 480 | Color: 16, 32 | LVGL: 9.1.*, 9.2.* v2.0.2

Eclipse with SDL for development on PC
Eclipse based project using SDL to develop UIs on PC.
Language: C | Res.: 800 x 480 | Color: 16, 32 | LVGL: 9.1.*, 9.2.* v2.0.1

Visual Studio for development on Windows PC
Visual Studio based project to develop UIs on Windows PC.

based project using SDL which opens a window and puts the UI into it. You can use this project to develop embedded UIs on PC or to create standalone desktop applications. To use this project you need to install CMake (optionally Visual Studio Code / Eclipse) and SDL.

Enter text...

Project name	lesson13.spj
	C:\Users\Administrator\Desktop\arduino_project\lesson13\
Resolution	320 240
Rotation	0 degree
Offset	0 0
Shape	Rectangle
Color depth	32 bit
LVGL version	9.2.2
Theme	Light
Multilanguage	Disable

• **1. Select the creation entry:** Click the "Create" button at the top of the interface to start the project creation process.

• **2. Determine the LVGL major version:** At "Select major LVGL version," select version 9.2.

• **3. Select the hardware/platform type:** Choose "Desktop."

• **4. Select the project template:** Choose the corresponding template (eg: "CMake/E-clipse/VScode with SDL for development on PC," a template for developing UI on PC based on SDL).

• **5. Project name:** lesson13.spj

• **6.Storage path**

• **7 . Screen resolution:** 320*240 (designed according to the screen specifications)

• **8. Color depth:** 32bit

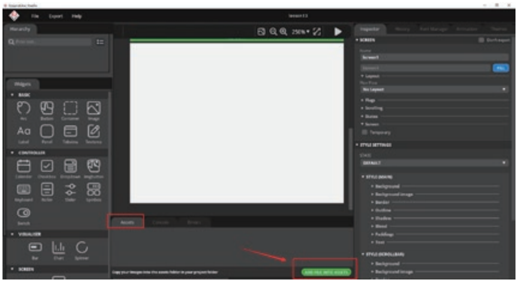
• **9. LVGL version**: 9.2.2

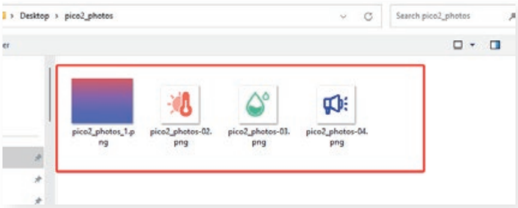
• **10. Complete creation:** After confirming the parameters are correct, click the green "CREATE" button in the lower-right corner to generate the project.

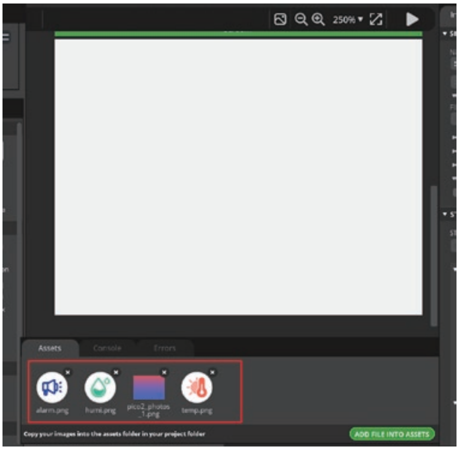
2. Add Design Materials

The tasks we need to complete are: real-time display of temperature values, reminder messages, and humidity values through the LVGL graphic interface. When the temperature is too high, turn on the red or yellow light as a prompt; when the humidity is too low, activate the buzzer alarm; and turn the green light on and off by clicking the button in the lower-right corner of the graphic interface (to verify the touch function of the display).

Open the provided images and add them in.

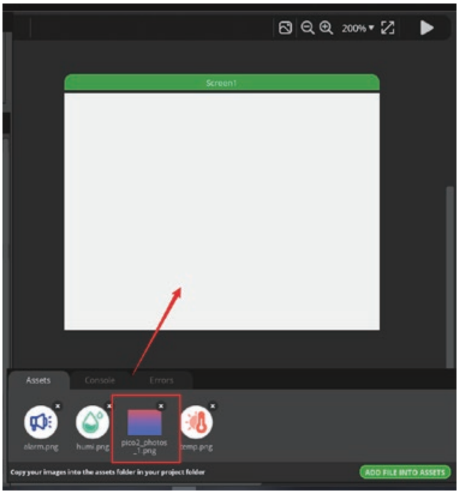




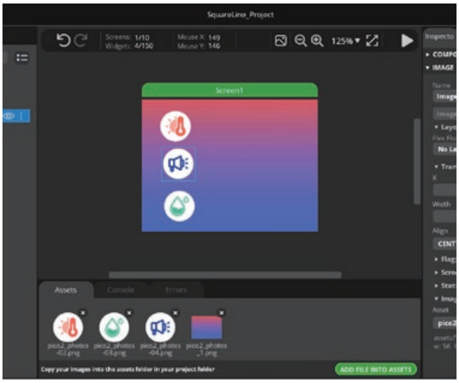


Select the designed background images, temperature-humidity icons, and other materials to add.
Select the designed background images, temperature-humidity icons, and other materials to add.

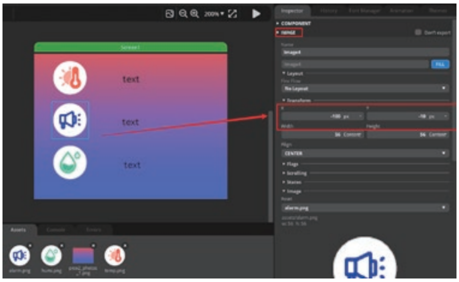
Next, drag the background image to the center of the canvas to cover the entire canvas area as the basic background of the UI interface. In the right-side Properties panel (IMAGE->Transform), precisely set the coordinates (eg: X:0, Y:0) and size to ensure the background image fits the canvas perfectly.



Drag the temperature icon, reminder icon, and humidity icon to appropriate positions on the canvas in sequence. You can arrange them freely according to design requirements, or refine the layout of each icon in the right-side Properties panel by modifying coordinate values, adjusting the scaling ratio, and other parameters.



You can set the specific coordinates for each icon in detail in the right-side Properties panel (IMAGE->Trans-form).



3. Add Text Labels and Value Display

• 1. Add text labels:

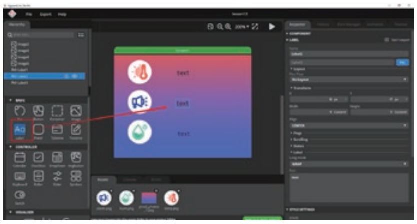
Find the Text Label tool in the interface elements toolbar and add text labels for temperature, reminder information, and humidity in sequence.

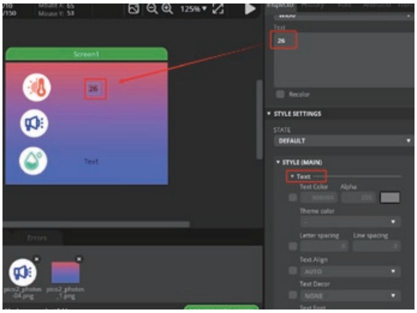
• 2. Set label properties:

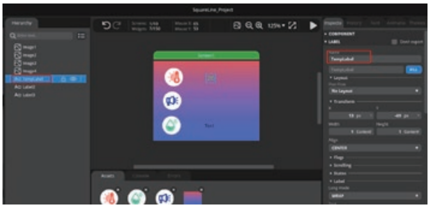
Click each text label and make detailed settings in the right-side Properties panel, including the label name, size, style, etc., to enhance the interface aesthetics and readability.

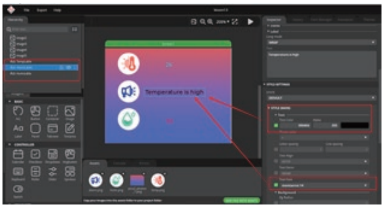
(The temperature label here is TempLable.)
(The reminder information label here is AlarmLable.)
(The humidity label here is HumiLable.)

This allows you to easily call them in the code (TempLable, AlarmLable, HumiLable).



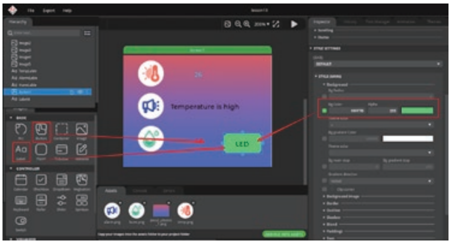




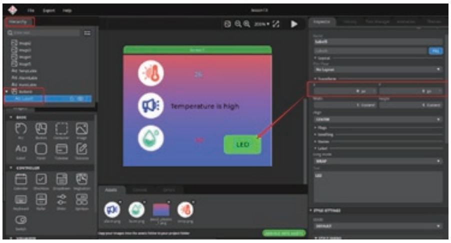


Next, create the button model:

Create a button: Select the BUTTON label from the left sidebar and drag it to the lower-right corner of the interface. In the right-side Properties panel, modify the button's background color to green, the text content to "LED," and the text color to black to intuitively display the button function.

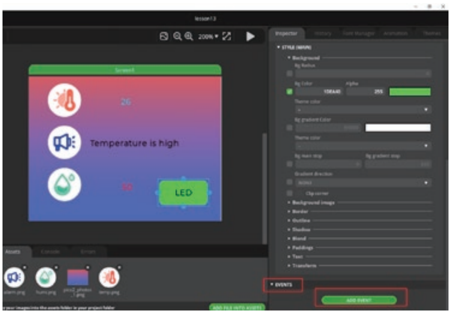


Merge the button and label: Click "Hierarchy" in the left column, drag the LED text label to the Button row to merge them into a whole. Then, in the Properties panel, set the LED text label position to X:0, Y:0 to center the text in the Button label.



4. Add EVENTS Function to the Button to Turn the Green Light On and Off

• 1. Select the button and add an event: Select the "LED" button in the lower-right corner, click "EVENTS->ADD EVENT" in the right-side Properties panel, and create a new interactive event.



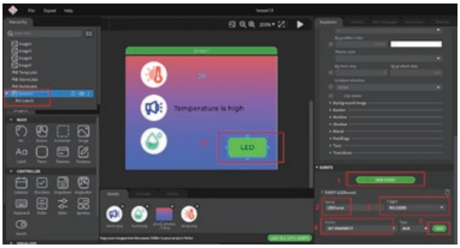
Configure event parameters:

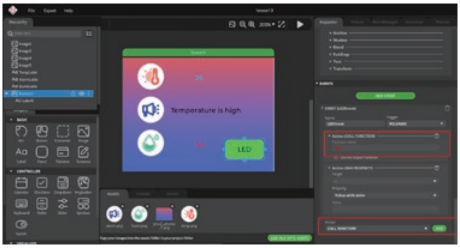
• 2. Name: Enter "LEDEvent" to name the event.

• 3. Select 'RELEASED' as the trigger condition, which triggers the event when the button is released.

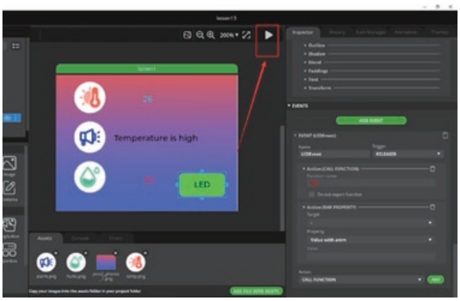
• 4. Select 'CALL FUNCTION' as the action to execute the corresponding operation through a callback function.

• 5. After selection, click "ADD."

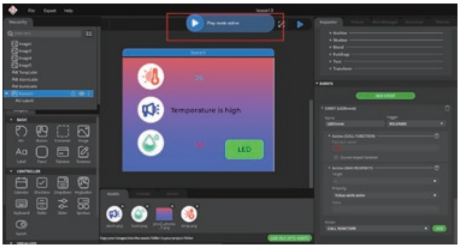




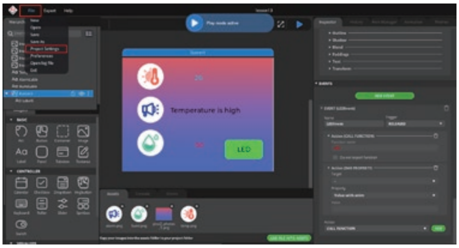
• Add a custom function name to the CALL function, and later write the specific logic of this function in the Arduino code to achieve on/off control of the green light.



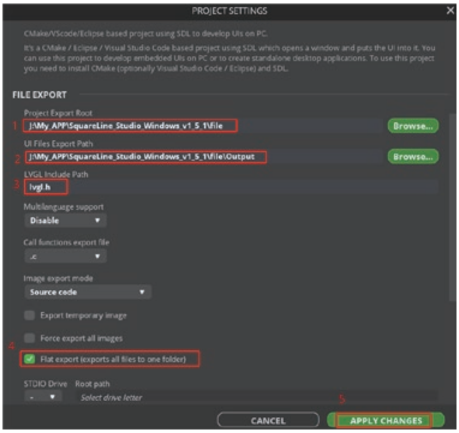
After the setup is complete, perform a simple simulation run first to check if the button event is triggered normally.



After the design is complete, we are ready to export the project.



Follow this order to configure your own project export path.



PROJECT SETTINGS

CMake/VScode/Eclipse based project using SDL to develop UIs on PC.

It's a CMake / Eclipse / Visual Studio Code based project using SDL which opens a window and puts the UI into it. You can use this project to develop embedded UIs on PC or to create standalone desktop applications. To use this project you need to install CMake (optionally Visual Studio Code / Eclipse) and SDL.

FILE EXPORT

Project Export Root
1 J:\My_APP\SquareLine_Studio_Windows_v1_5_1\file Browse...

UI Files Export Path
2 J:\My_APP\SquareLine_Studio_Windows_v1_5_1\file\Output Browse...

LVGL Include Path
3 lvgl.h

Multilanguage support: Disable

Call functions export file: .c

Image export mode: Source code

☐ Export temporary image

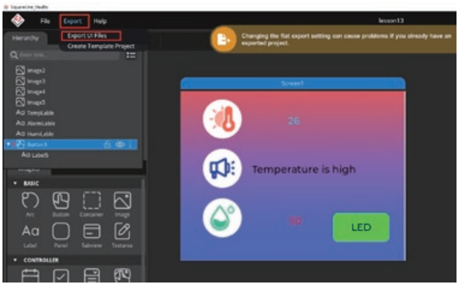
☐ Force export all images

4 ✓ Flat export (exports all files to one folder)

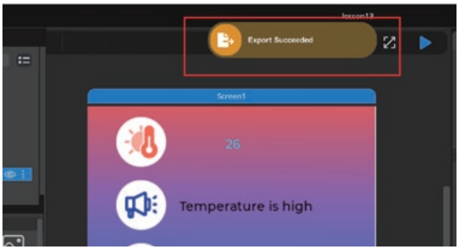
STDIO Drive: - Root path: Select drive letter

5 CANCEL APPLY CHANGES

After the setup is complete, you can export the relevant UI code of the project you designed.

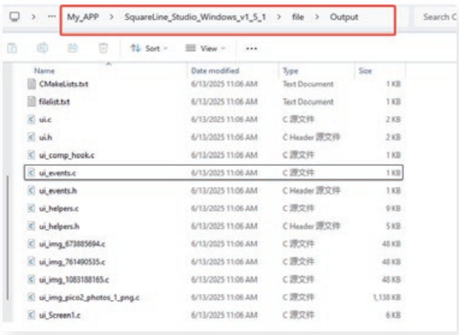


File
Export
Help
Export UI Files
Create Template Project
Changing the flat export setting can cause problems if you already have an exported project.
Screen1
26
Temperature is high
LED



Export UI Code

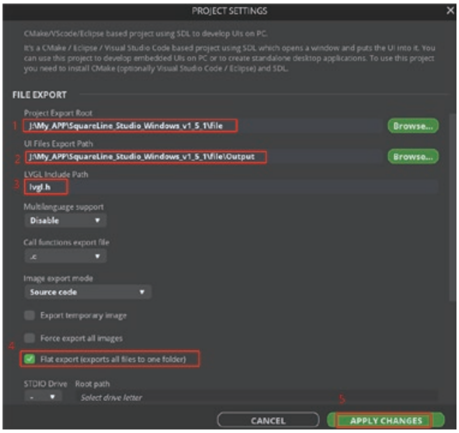
The UI code you just designed is as follows.



My_APP > SquareLine_Studio_Windows_v1_5_1 > file > Output

Name	Date modified	Type	Size
CMakeLists.txt	6/13/2025 11:06 AM	Text Document	1 KB
filelist.txt	6/13/2025 11:06 AM	Text Document	1 KB
ui.c	6/13/2025 11:06 AM	C 源文件	2 KB
ui.h	6/13/2025 11:06 AM	C Header 源文件	2 KB
ui_comp_hook.c	6/13/2025 11:06 AM	C 源文件	1 KB
ui_events.c	6/13/2025 11:06 AM	C 源文件	1 KB
ui_events.h	6/13/2025 11:06 AM	C Header 源文件	1 KB
ui_helpers.c	6/13/2025 11:06 AM	C 源文件	9 KB
ui_helpers.h	6/13/2025 11:06 AM	C Header 源文件	5 KB
ui_img_673885694.c	6/13/2025 11:06 AM	C 源文件	48 KB
ui_img_761490535.c	6/13/2025 11:06 AM	C 源文件	48 KB
ui_img_1083188165.c	6/13/2025 11:06 AM	C 源文件	48 KB
ui_img_pico2_photos_1_png.c	6/13/2025 11:06 AM	C 源文件	1,138 KB
ui_Screen1.c	6/13/2025 11:06 AM	C 源文件	6 KB

This export path is the same as the previous export path. As corresponding to No.2 in the figure, you can find the UI file you made here.



PROJECT SETTINGS

CMake/VScode/Eclipse based project using SDL to develop UIs on PC.

It's a CMake / Eclipse / Visual Studio Code based project using SDL which opens a window and puts the UI into it. You can use this project to develop embedded UIs on PC or to create standalone desktop applications. To use this project you need to install CMake (optionally Visual Studio Code / Eclipse) and SDL.

FILE EXPORT

1 Project Export Root: J:\My_APP\SquareLine_Studio_Windows_v1_5_1\file

2 UI Files Export Path: J:\My_APP\SquareLine_Studio_Windows_v1_5_1\file\Output

3 LVGL Include Path: lvgl.h

Multilanguage support: Disable

Call functions export file: .c

Image export mode: Source code

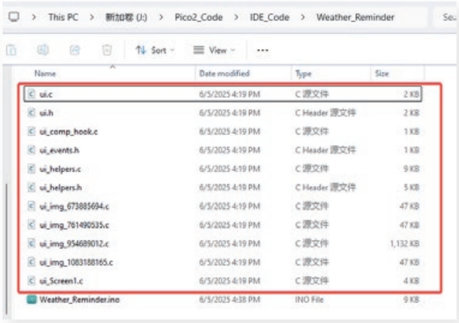
☐ Export temporary image

☐ Force export all images

4 ✓ Flat export (exports all files to one folder)

STDIO Drive: - Root path: Select drive letter

Copy these codes and place them together with your ino main code.



This PC > 新加卷 (J:) > Pico2_Code > IDE_Code > Weather_Reminder

Name	Date modified	Type	Size
ui.c	6/5/2025 4:19 PM	C 源文件	2 KB
ui.h	6/5/2025 4:19 PM	C Header 源文件	2 KB
ui_comp_hook.c	6/5/2025 4:19 PM	C 源文件	1 KB
ui_events.h	6/5/2025 4:19 PM	C Header 源文件	1 KB
ui_helpers.c	6/5/2025 4:19 PM	C 源文件	9 KB
ui_helpers.h	6/5/2025 4:19 PM	C Header 源文件	5 KB
ui_img_673885694.c	6/5/2025 4:19 PM	C 源文件	47 KB
ui_img_761490535.c	6/5/2025 4:19 PM	C 源文件	47 KB
ui_img_954689012.c	6/5/2025 4:19 PM	C 源文件	1,132 KB
ui_img_1083188165.c	6/5/2025 4:19 PM	C 源文件	47 KB
ui_Screen1.c	6/5/2025 4:19 PM	C 源文件	4 KB
Weather_Reminder.ino	6/5/2025 4:38 PM	INO File	9 KB

In this way, the overall design of the UI interface you need to display is completed. Next, you only need to call it in the code!

At the current stage of project development, the code design work of the UI interface has been completed, and the next key task is to write the Arduino main code file to achieve the expected functions. Specifically, the main code needs to implement the collection and processing of temperature-humidity sensor data and send these data to the UI interface through an appropriate communication method to accurately display the temperature and humidity values on the interface.

Key Explanations

1. LVGL Display Engine Integration

```
uint32_t draw_buf[DRAW_BUF_SIZE / 4];
lv_display_set_buffers(disp, draw_buf, NULL, sizeof(draw_buf),
LV_DISPLAY_RENDER_MODE_PARTIAL);
void my_disp_flush(...) {
    gfx.pushPixels((lgfx::rgb565_t *)px_map, w * h);
}
```

▸ **uint32_t draw_buf[DRAW_BUF_SIZE / 4]; // 320x240/10 * 2bytes = 15KB:** Defines a display buffer draw_buf of type uint32_t with a size of DRAW_BUF_SIZE / 4.Since each uint32_t occupies 4 bytes, this allocation results in DRAW_BUF_SIZE bytes of memory.

▸ **lv_display_set_buffers(disp,draw_buf,NULL,sizeof(draw_buf), LV_DISPLAY_RENDER_-MODE_PARTIAL);:** Core LVGL function for setting the display buffer.

• **disp:** Display device object (created using lv_display_create())
• **draw_buf:** Address of the drawing buffer (used by the CPU to write pixel data)
• **NULL:** Indicates single buffering (no double buffer is used)
• **sizeof(draw_buf):** Size of the buffer in bytes
• **LV_DISPLAY_RENDER_MODE_PARTIAL:** 分Partial rendering mode — only updates

▸ **void my_disp_flush(...) :** This function is called by LVGL when a portion of the screen needs to be refreshed. The function name is user-defined and must be registered via lv_display_set_flush_cb().

▸ **gfx.pushPixels((lgfx::rgb565_t *)px_map, w * h); Pushes a block of pixel data to the display.**

• gfx is the display object from LovyanGFX.
• pushPixels(...) is a high-speed pixel transfer function from LovyanGFX, optimized using low-level SPI acceleration.
• (lgfx::rgb565_t *)px_map： Casts the pixel data pointer px_map from LVGL to rgb565_t* (compatible with the screen's RGB565 format)
• w * h： Total number of pixels in the region to be refreshed

2. Sensor Data Handling

```
int status = DHT.read();
if (status == DHT20_OK) {
    float humidity = DHT.getHumidity();
    float temperature = DHT.getTemperature();
    snprintf(tempStr, sizeof(tempStr), "%.1f C", temperature);
}
```

- int status = DHT.read();Calls DHT.read() to initiate a measurement request from the DHT20 sensor.The function returns an integer status code (eg: DHT20_OK) indicating whether the read was successful.
- if (status == DHT20_OK) : Checks whether the sensor read was successful.Data is processed only if the status equals DHT20_OK (typically defined as macro value 0).
- float humidity = DHT.getHumidity(); Retrieves humidity data from the sensor, in %RH (relative humidity), typically with a resolution of 0.1%.
- float temperature = DHT.getTemperature(); Gets the current ambient temperature in degrees Celsius (°C),with a typical precision of 0.1°C.
- snprintf(tempStr, sizeof(tempStr), "%.1f C", temperature);
  - Uses snprintf() to format the floating-point temperature into a string, eg: "24.5 C"
  - tempStr: Destination string buffer (predefined as char tempStr[...])
  - sizeof(tempStr): imits the maximum write length to prevent buffer overflow
  - "%.1f C": Format specifier
  - %.1f formats the float to one decimal place
  - C is the unit suffix (Celsius)

3. Alarm Control Logic

```
if (temperature > 30) {
    digitalWrite(18, HIGH);
    lv_label_set_text(ui_AlarmLabel, "It's hot");
} else if (temperature > 25) {
    digitalWrite(20, HIGH);
}
if (humidity < 40.0) {
    tone(10, 1300);
} else {
    noTone(10);
}
```

- When the temperature exceeds 30°C, turn on the red LED (GPIO18) and display the warning message "It's hot" on the interface.
- If the temperature is between 25°C and 30°C, turn on the yellow LED (GPIO20) to indicate a mildly warm condition.
- When humidity drops below 40%, drive the buzzer by outputting a 1.3 kHz PWM square wave on GPIO10 to sound the alarm.Otherwise, call noTone(10) to stop the buzzer and mute the alarm.

4. Auxiliary Function

```
unsigned long currentMillis = millis();
if (currentMillis - lastUpdate >= interval) {
    lastUpdate = currentMillis;
}
lv_timer_handler();
```

- millis() millis() obtains the number of milliseconds elapsed since the system started (non-blocking and continuously increasing).
- Check whether currentMillis - lastUpdate has reached the predefined interval.
- When the condition is met, execute the critical task and update lastUpdate to record the current time. DHT.begin() initializes the sensor and prepares it for data reading.

5. Core Functions for Touch Functionality

• 1. Touch Configuration (Touch Settings in LGFX Class)

```
auto touch_cfg = _touch.config();
touch_cfg.x_min = 0;
touch_cfg.x_max = 239;
touch_cfg.y_min = 0;
touch_cfg.y_max = 319;
touch_cfg.pin_int = 25;
touch_cfg.pin_rst = 24;
touch_cfg.i2c_port = 0;
touch_cfg.i2c_addr = 0x38;
touch_cfg.pin_sda = 4;
touch_cfg.pin_scl = 5;
touch_cfg.freq = 400000;
_touch.config(touch_cfg);
_panel.setTouch(&_touch);
```

- The code configures parameters for the FT5x06 touch controller:
- Defines the X/Y coordinate range of the touchscreen.
- Sets the interrupt pin and reset pin.
- Configures I2C communication parameters (port, address, SDA/SCL pins, frequency).
- Finally, associates the touch controller with the display panel.

• 2. Touch Read Callback Function (my_touchpad_read)

```
void my_touchpad_read(lv_indev_t *indev, lv_indev_data_t *data)
{
    uint16_t touchX, touchY;
    bool touched = gfx.getTouch(&touchX, &touchY);
    if (touched) {
        data->state = LV_INDEV_STATE_PRESSED;
        data->point.x = touchX;
        data->point.y = touchY;
    } else {
        data->state = LV_INDEV_STATE_RELEASED;
    }
}
```

▶ This function serves as the LVGL input device read callback.

1. Obtain the current touch status and coordinates via gfx.getTouch().
2. If a touch is detected (touched is true):
   - Set the state to PRESSED.
   - Store the touch coordinates into data->point.
3. If no touch is detected:
   - Set the state to RELEASED.

• 3. Touch Device Initialization (in setup() function)

```
lv_indev_t *indev = lv_indev_create();
lv_indev_set_type(indev, LV_INDEV_TYPE_POINTER);
lv_indev_set_read_cb(indev, my_touchpad_read);
```

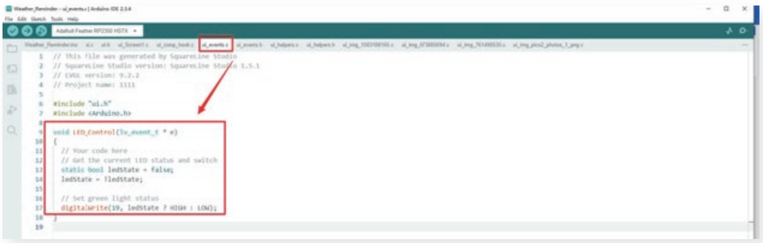
▶ lv_indev_t *indev = lv_indev_create();Creates a new input device instance. lv_indev_create() is an LVGL API that allocates and initializes a new input device structure.It returns a pointer to an lv_indev_t object representing this input device, which will be used for subsequent touch input handling.

▶ lv_indev_set_type(indev, LV_INDEV_TYPE_POINTER); Sets the input device type to LV_INDEV_-TYPE_POINTER, indicating a pointer-type device such as a touchscreen.

▶ lv_indev_set_read_cb(indev, my_touchpad_read); Assigns the input device's read callback function, used to fetch touch coordinates and status.

• 4. Controlling the Green LED On/Off

Add the following code to event.c to implement button-controlled green LED toggling:



```
// This file was generated by SquareLine Studio
// SquareLine Studio version: SquareLine Studio 1.5.1
// LVGL version: 9.2.2
// Project name: 1111

#include "ui.h"
#include <Arduino.h>

void LED_Control(lv_event_t * e)
{
  // Your code here
  // Get the current LED status and switch
  static bool ledState = false;
  ledState = !ledState;

  // Set green light status
  digitalWrite(19, ledState ? HIGH : LOW);
}
```

```
void LED_Control(lv_event_t * e)
{
  static bool ledState = false;
  ledState = !ledState;
  digitalWrite(19, ledState ? HIGH : LOW);
```

- **Function Definition and Trigger Logic:** LED_Control(lv_event_t * e) is an event callback function of LVGL (embedded GUI library). When the associated "LED button" on the display is clicked, LVGL will automatically call this function to execute the logic.
- **State Switching:** static bool ledState = false; : A static boolean variable ledState is defined to record the current state of the green light.
- ledState = !ledState; : Each time the button is clicked, ledState is flipped (switching between on → off / off → on).
- **Controlling Hardware Pins:** digitalWrite(19, ledState ? HIGH : LOW); : According to the value of ledState, control Arduino pin 19 to output a high level (HIGH, turning on the green light) or a low level (LOW, turning off the green light), implementing the function of "clicking the button to switch the state of the green light".

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After learning the above code, a touchscreen-based temperature and humidity monitoring alarm system is implemented. The system uses a DHT20 sensor to collect temperature and humidity data, and displays it in real time through an LVGL graphical interface. When the temperature is too high, red or yellow LEDs are lit for prompt; when the humidity is too low, a buzzer is activated for alarm, achieving a complete function of sensor data collection, screen display, and hardware linkage.

Programming Steps

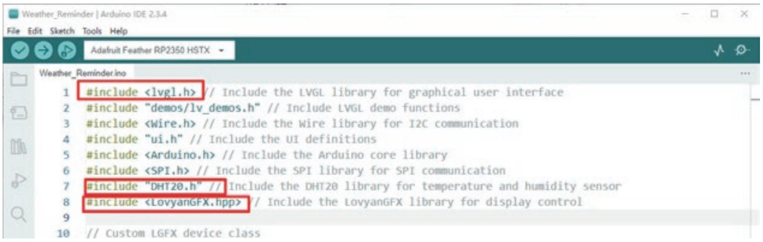
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**DHT20, LovyanGFX-develop, lvgl and lv_conf.h**),so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

• Navigate to the **/example/libraries** directory and download the **DHT20, LovyanGFX-develop, lvgl and lv_conf.h** folder.

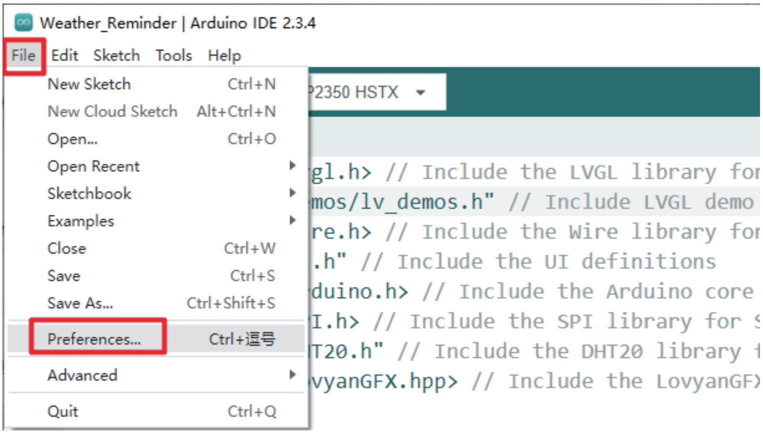


Weather_Reminder | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
Weather_Reminder.ino

```
#include <lvgl.h> // Include the LVGL library for graphical user interface
#include "demos/lv_demos.h" // Include LVGL demo functions
#include <Wire.h> // Include the Wire library for I2C communication
#include "ui.h" // Include the UI definitions
#include <Arduino.h> // Include the Arduino core library
#include <SPI.h> // Include the SPI library for SPI communication
#include "DHT20.h" // Include the DHT20 library for temperature and humidity sensor
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control

// Custom LGFX device class
```

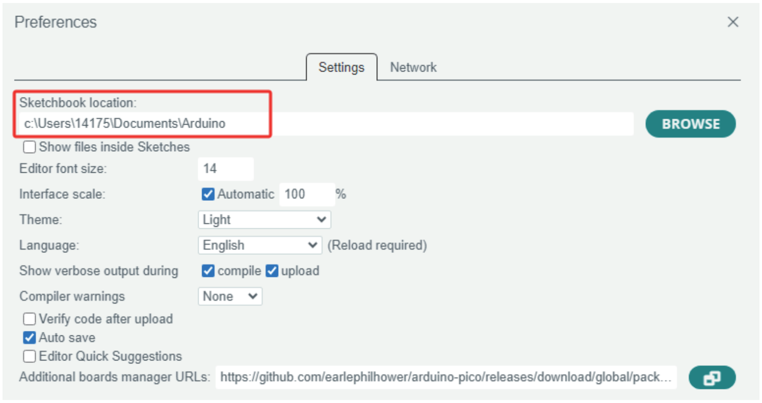
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



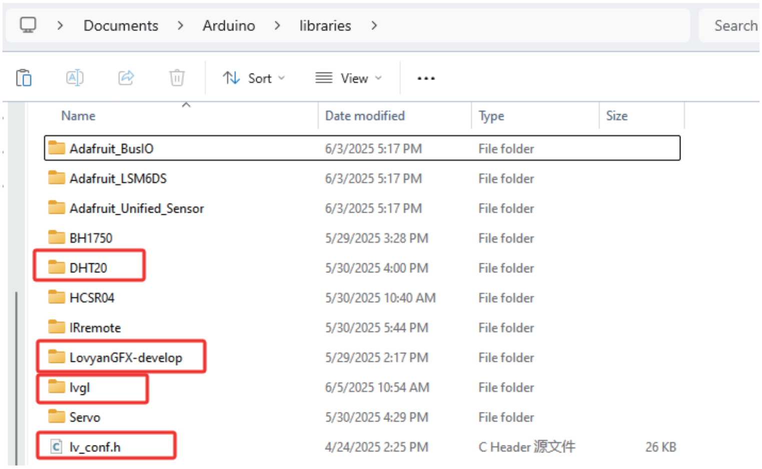
```
gl.h> // Include the LVGL library for
mos/lv_demos.h" // Include LVGL demo
re.h> // Include the Wire library for
.h" // Include the UI definitions
duino.h> // Include the Arduino core
I.h> // Include the SPI library for S
T20.h" // Include the DHT20 library f
vyanGFX.hpp> // Include the LovyanGF
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ☑ compile ☑ upload
Compiler warnings None
☐ Verify code after upload
☑ Auto save
☐ Editor Quick Suggestions
Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
Adafruit_BusIO	6/3/2025 5:17 PM	File folder	
Adafruit_LSM6DS	6/3/2025 5:17 PM	File folder	
Adafruit_Unified_Sensor	6/3/2025 5:17 PM	File folder	
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
lvgl	6/5/2025 10:54 AM	File folder	
Servo	5/30/2025 4:29 PM	File folder	
lv_conf.h	4/24/2025 2:25 PM	C Header 源文件	26 KB

3. Upload and Run the Code

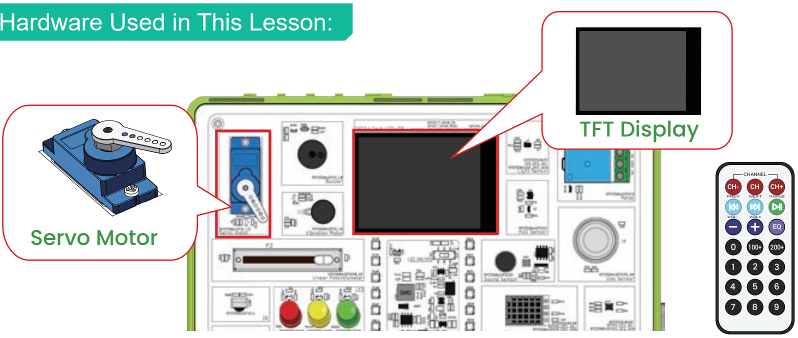
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 14 - Servo Angle Control

Introduction

In this section, we will learn how to control the rotation angle of a servo motor using an infrared remote and display the angle on a TFT screen. Different buttons on the infrared remote can be used to: input angle values, confirm the angle setting, start/stop the servo motor, and display the current angle in real-time on the TFT screen.

Hardware Used in This Lesson:



Working Principle of Servo Motor

A servo motor achieves precise angular positioning via closed-loop control. The control circuit interprets PWM signals to determine target position. A motor drives gears while a potentiometer provides real-time positional feedback to a comparator. Any deviation triggers corrective rotation until error is eliminated. Standard servos offer 0°~180° rotation with high torque and rapid response.

Principle of Infrared Remote Control Operation

The infrared remote control works by inputting control commands through buttons. The microcontroller encodes and modulates these commands into infrared signals, which are then transmitted by an infrared emitter. The receiving end's infrared sensor captures the signals, demodulates them back into commands, and drives the device to perform the corresponding operations.

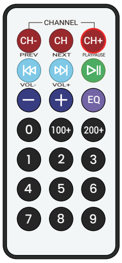
Working Principle of Display Screen

The display controls each pixel's brightness/color via driver circuits. The main IC converts image data to electrical signals, transmitted to row/column driver ICs for line-by-line refreshing. LCDs adjust liquid crystal alignment with voltage to modulate light, while OLEDs emit light directly. A timing controller ensures signal synchronization to prevent artifacts.

Operation Effect Diagram

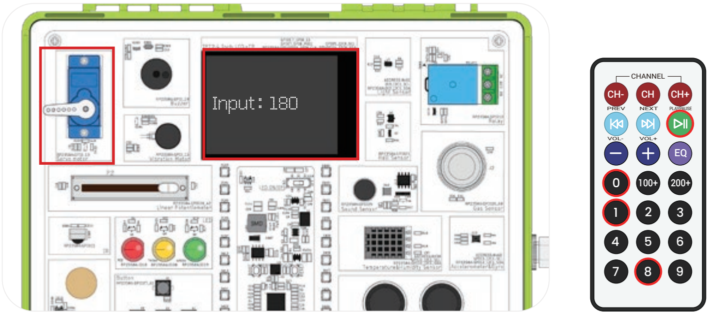
1. Enable IR Remote Control:

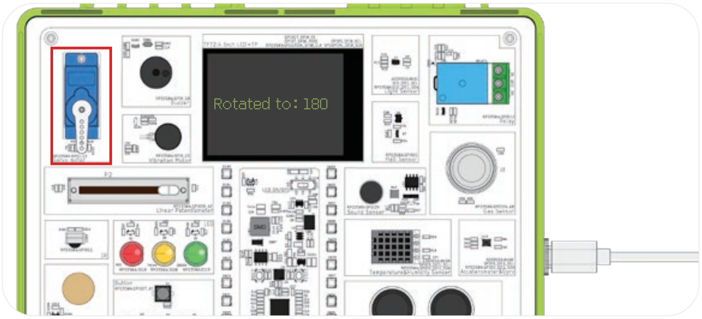
Press the button labeled "CH+" on the device. This operation will activate the infrared remote control function, preparing it for subsequent command reception.



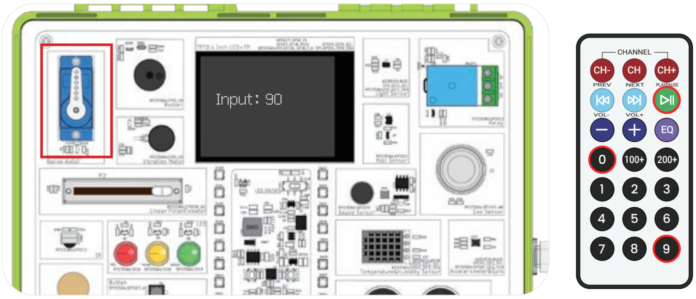
2. Angle Setting and Motor Control (Numeric Keys: 0-9)

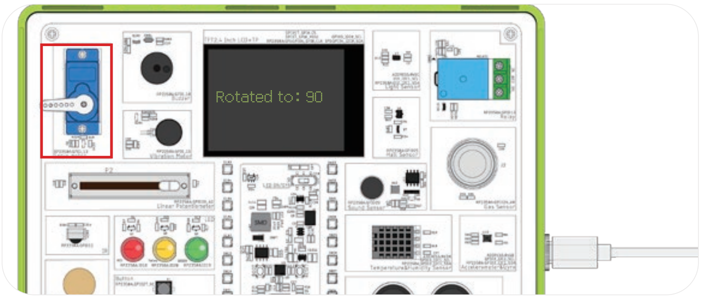
• **Set 180 degrees:** Press the numeric keys and sequentially enter "1", "8", "0" on the input interface. The display will real-time show "Input:180". After confirming the input is correct, press the "Confirm" button ⏯ on the remote control. The system will drive the servo motor to rotate to the target angle of 180 degrees based on this command. Meanwhile, the TFT screen will simultaneously update to display "Rotated to :180", providing an intuitive feedback on the current angle of the motor.



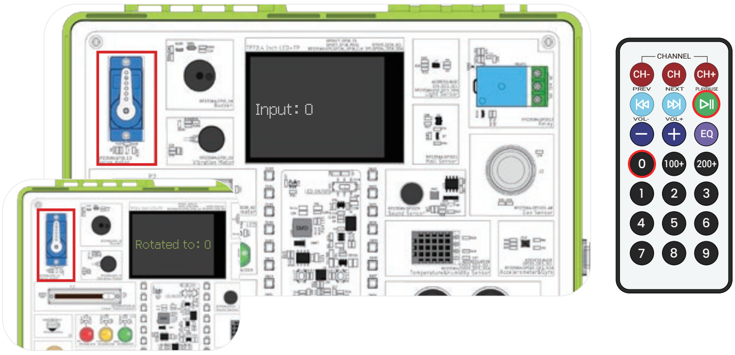


• **Set 90 degrees:** Repeat the above operation process. Press the numeric keys to enter "9" and "0". The display will show "Input:90". Press the "Confirm" button. The servo motor will then rotate to the 90-degree position, and the TFT screen will display "Rotated to :90".





• **Set 0 degrees:** Press the numeric key to enter "0". The display will show "Input:0". Press the "Confirm" button. The servo motor will precisely rotate to 0 degrees, and the TFT screen will show "Rotated to :0".



3. Disable IR Remote Control:

When you need to stop using the IR remote control to operate the servo motor, press the "CH-" button on the device. This action will disable the IR remote control function. Thereafter, no matter which numeric keys are pressed, the system will not respond, and the servo motor will not start, ensuring the device remains safely stationary under unexpected operations.

CHANNEL
CH-
CH
CH+
PREV
NEXT
PLAYPAUSE
VOL-
VOL+
EQ
0
100+
200+
1
2
3
4
5
6
7
8
9

Key Explanations

1. IR Signal Reception & Debounce Mechanism

```
if (IrReceiver.decode()) {
  unsigned long code = IrReceiver.decodedIRData.command;
  static uint32_t lastKeyCode = 0;
  static unsigned long lastKeyTime = 0;
  const unsigned long debounceDelay = 300;
  if (code == lastKeyCode && (millis() - lastKeyTime) < debounceDelay) {
    IrReceiver.resume();
    return;
  }
  lastKeyCode = code;
  lastKeyTime = millis();
```

- IrReceiver.decode() checks if an IR signal from the remote control is received.
- IrReceiver.decodedIRData.command extracts the key code of the pressed button, which is used later to determine which button was pressed.
- Debounce Mechanism: When a button is held down or the remote is jiggled, the same key code is repeatedly sent. This code uses two static variables to ensure that the same key press is processed only once within 300ms, preventing multiple responses.
- This is a standard practice in remote control input programming, resulting in a smoother user experience.

2. IR Button Function Mapping

```
switch (code) {
  case 0x45:
    servoEnabled = false;
    Serial.println("Servo disabled");
    break;
  case 0x47:
    servoEnabled = true;
    Serial.println("Servo enabled");
    break;
  case 0x43:
    angleInput = constrain(angleInput, 0, 180);
    if (servoEnabled) {
      myservo.write(angleInput);
      showRotatedAngle(angleInput);
    } else {
      Serial.println("Servo is disabled. Cannot rotate.");
    }
    angleInput = 0;
    delay(1000);
    break;
  case 0x16: angleInput = angleInput * 10 + 0; break;
  case 0x0C: angleInput = angleInput * 10 + 1; break;
  case 0x18: angleInput = angleInput * 10 + 2; break;
  case 0x5E: angleInput = angleInput * 10 + 3; break;
  case 0x08: angleInput = angleInput * 10 + 4; break;
  case 0x1C: angleInput = angleInput * 10 + 5; break;
  case 0x5A: angleInput = angleInput * 10 + 6; break;
  case 0x42: angleInput = angleInput * 10 + 7; break;
  case 0x52: angleInput = angleInput * 10 + 8; break;
  case 0x4A: angleInput = angleInput * 10 + 9; break;
  default:
    Serial.print("Unknown code: ");
    Serial.println(code, HEX);
    break;
}
```

- **[CH-] Button:** Set the control variable servoEnabled to false, preventing the servo from acting when the confirmation ▷II button is pressed later.
- **[CH+] Button:** Resume servo control by setting servoEnabled to true.
- **[PLAY/PAUSE] Button:**

  • Use constrain(angleInput, 0, 180) to ensure the angle is within the servo's safe range.

  • If the servo is enabled, execute myservo.write(angleInput) to rotate the servo to that angle, and use showRotatedAngle(angleInput) to display "Rotated to xxx°" on the screen.

  • If disabled, the serial port will prompt "Servo not enabled".

  • Clear the input angle and delay for 1000ms to allow the user 1 second to observe the result.
- **Numeric Keys 0-9:**

  • Use the "cumulative input method" for angle entry. For example, pressing "1" then "2" gives angleInput = 1*10 + 2 = 12.

  • Support continuous multi-digit input (up to three digits) for convenient entry of large angles like 100 or 120.

- **Default:** Unrecognized codes are printed for subsequent debugging.

3. Alarm Control Logic

```
if (angleInput > 999) angleInput = 0;
showAngleInput();
IrReceiver.resume();
```

- Limit input to a maximum of three digits (prevents overflow from misoperations).
- For each new input, call showAngleInput() to display the current input angle on the screen in real time, keeping the user informed.
- Resume IR reception and wait for the next button press.

4. Auxiliary Function

```
myservo.attach(13, 450, 2520);
myservo.write(angleInput);
```

- Servo.attach(pin, min, max) binds the servo to a pin and specifies the minimum/maximum pulse width, ensuring compatibility with different servo brands.
- myservo.write(angle) directly inputs the angle, and the library automatically converts the angle to pulse width output to drive the servo.

5. Screen Display Functions

```
void showAngleInput() {
  gfx.fillScreen(TFT_BLACK);
  gfx.setTextSize(3);
  gfx.setCursor(30, 100);
  gfx.setTextColor(TFT_WHITE);
  gfx.printf("Input: %d", angleInput);
}
void showRotatedAngle(int angle) {
  gfx.fillScreen(TFT_BLACK);
  gfx.setCursor(30, 100);
  gfx.setTextSize(3);
  gfx.setTextColor(TFT_GREEN);
  gfx.printf("Rotated to: %d", angle);
}
```

- During input, showAngleInput() displays "Input: [current input angle]" to inform the user of the entered value.
- After confirmation, showRotatedAngle() displays "Rotated to: [angle]" to confirm the servo has executed the command.
- Clear the screen each time to avoid content overlay.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the above code, you have implemented a servo angle adjustment system controlled by an infrared remote control. Combined with a TFT display and touch configuration, the system can achieve the following functions:
Input an angle value via the infrared remote control to rotate the servo to the specified angle, while the TFT screen simultaneously displays the input angle and execution results in real time.

Programming Steps

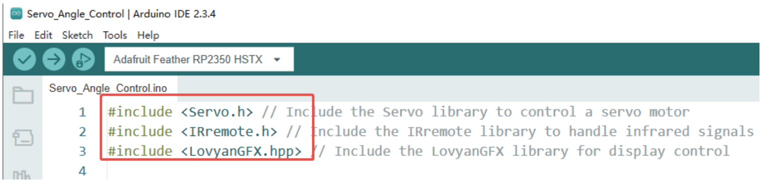
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use three additional library (**IRremote, LovyanGFX-develop, Servo**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

• Navigate to the **/example/libraries** directory and download the **IRremote, LovyanG-FX-develop and Servo** folder.

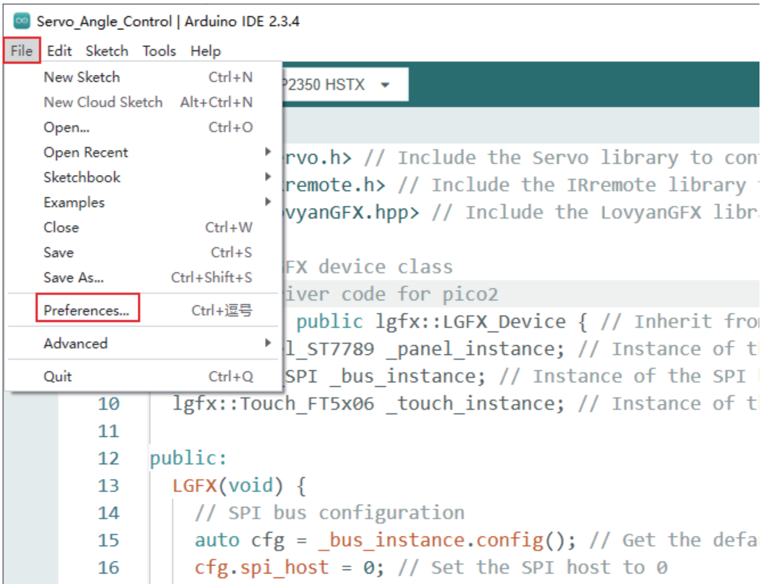


Servo_Angle_Control | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
Servo_Angle_Control.ino

```
#include <Servo.h> // Include the Servo library to control a servo motor
#include <IRremote.h> // Include the IRremote library to handle infrared signals
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control
```

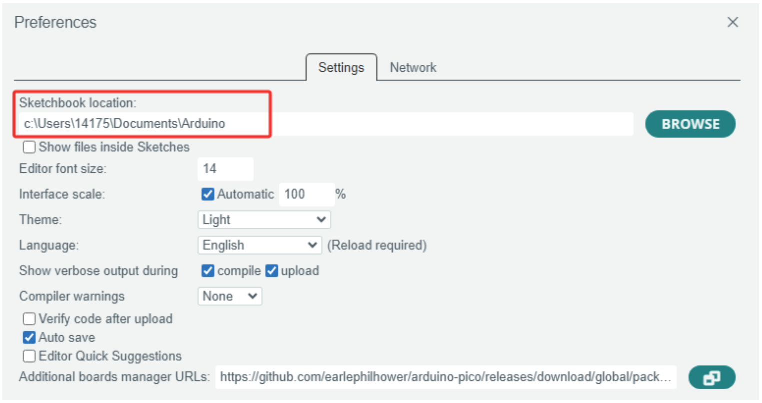
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
rvo.h> // Include the Servo library to con
remote.h> // Include the IRremote library
vyanGFX.hpp> // Include the LovyanGFX libr

FX device class
iver code for pico2
 public lgfx::LGFX_Device { // Inherit fro
l_ST7789 _panel_instance; // Instance of t
SPI _bus_instance; // Instance of the SPI
  lgfx::Touch_FT5x06 _touch_instance; // Instance of t

public:
  LGFX(void) {
    // SPI bus configuration
    auto cfg = _bus_instance.config(); // Get the defa
    cfg.spi_host = 0; // Set the SPI host to 0
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.

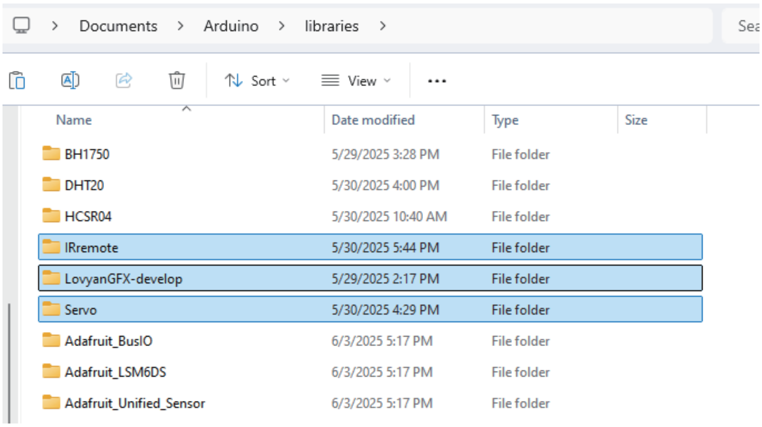

Preferences

Settings | Network

Sketchbook location:
c:\Users\14175\Documents\Arduino

Show files inside Sketches [ ]
Editor font size: 14
Interface scale: [✓] Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during [✓] compile [✓] upload
Compiler warnings None
[ ] Verify code after upload
[✓] Auto save
[ ] Editor Quick Suggestions
Additional boards manager URLs: https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.


Documents > Arduino > libraries >

Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
Servo	5/30/2025 4:29 PM	File folder	
Adafruit_BusIO	6/3/2025 5:17 PM	File folder	
Adafruit_LSM6DS	6/3/2025 5:17 PM	File folder	
Adafruit_Unified_Sensor	6/3/2025 5:17 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

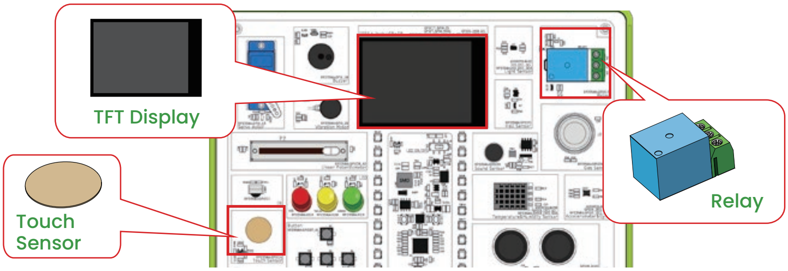
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 15 - Polite Automatic Door

Introduction

In this section, a touch module is used to control a relay, enabling advanced operations of a servo motor to simulate the opening and closing of an automatic door. When the touch button is activated, the relay switches on to open the door, then automatically turns off after 10 seconds to close it. A TFT screen provides real-time status updates during the process.

Hardware Used in This Lesson:



Working Principle of Relay Operation

A relay is a switch device that uses electromagnetic or mechanical principles to achieve control. When the coil is energized, it generates a magnetic field that attracts the armature, causing the normally closed contacts to open and the normally open contacts to close. When de-energized, the magnetic field disappears and the contacts return to their original state. It enables low-current/voltage control of high-current/voltage circuits, widely used for circuit control and signal conversion, providing isolation and amplification.

Working Principle of Touch Sensor

A touch sensor detects touches by monitoring capacitance or resistance changes. Capacitive types measure field variation when body capacitance alters electrode charge; resistive types register pressure-induced contact between conductive layers. Signal processing converts subtle changes to digital outputs with noise immunity. Supports multi-touch and gesture recognition, widely used in smart devices.

Working Principle of Display Screen

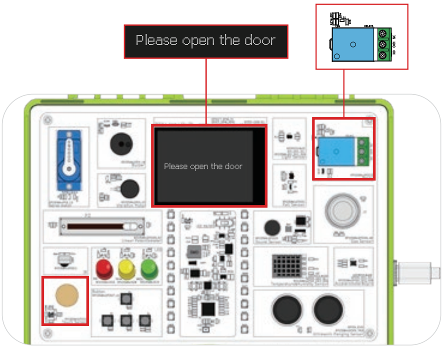
The display controls each pixel's brightness/color via driver circuits. The main IC converts image data to electrical signals, transmitted to row/column driver ICs for line-by-line refreshing. LCDs adjust liquid crystal alignment with voltage to modulate light, while OLEDs emit light directly. A timing controller ensures signal synchronization to prevent artifacts.

Operation Effect Diagram

1. System Startup and Initialization

Once the power is turned on, the program starts automatically:

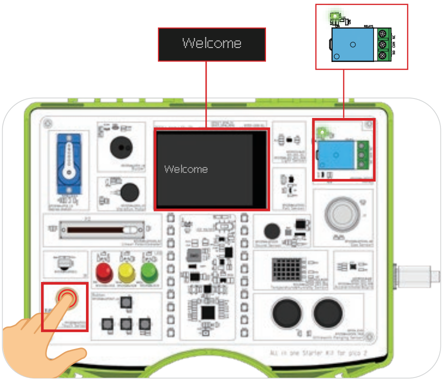
- The TFT screen lights up and displays the initial message: “Please open the door.”
- The relay remains off by default (no power), waiting for a touch input.



2. Touch-Triggered Door Opening

When the user touches the sensor with a finger, the system detects the input:

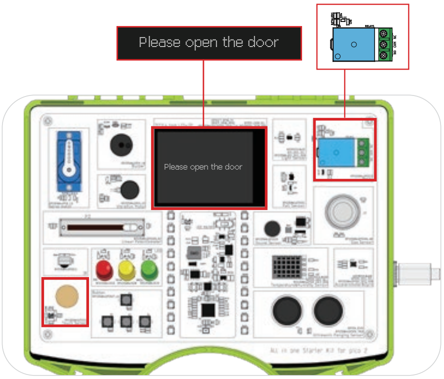
- The relay is energized, simulating the door opening; the relay’s indicator light turns on.
- The TFT screen clears and displays “Welcome” to indicate the door is now open.



3. Auto-Closing After Delay

The door remains open for 10 seconds:

- After the countdown finishes, the relay is turned off, simulating the door closing; the indicator light turns off.
- The TFT screen returns to the “Please open the door” message, and the system goes back to standby mode, ready for the next touch activation.



Key Explanations

1. Display Initialization and Welcome Message

```
gfx.init();
gfx.fillScreen(TFT_BLACK);
gfx.setTextColor(TFT_WHITE);
gfx.setTextSize(2);
gfx.setCursor(20, 100);
gfx.print("Please open the door");
```

- Initialize the display module.
- Set text color, size, and position.
- Show the startup message: “Please open the door.”

2. Touch Button Detection and Door Lock Control Logic

```
if (digitalRead(touchPin)) {
  digitalWrite(relayPin, HIGH);
  gfx.fillScreen(TFT_BLACK);
  gfx.setCursor(60, 100);
  gfx.print("Welcome");
  delay(10000);
  digitalWrite(relayPin, LOW);
  ...
}
```

- Monitor for touch input from the sensor.
- When a touch is detected: activate the relay (to simulate door opening) and display a welcome message.
- After 10 seconds, turn off the relay and return to the initial “Please open the door” prompt.

3. Pin Definitions

```
const int relayPin = 12;
const int touchPin = 14;
const int backlightPin = 0;
```

- relayPin: Controls the door lock via the relay.
- touchPin: Receives input from the external capacitive touch sensor.
- backlightPin: Controls the backlight of the display.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you will have built a smart automatic door control system. The system uses an SPI bus to drive the ST7789 display and an I2C interface to connect the FT5x06 touch module. While in standby mode, the screen displays the message “Please open the door.” When the touch sensor is activated, the relay powers on to simulate the door opening, and the display switches to a “Welcome” screen. After 10 seconds, the relay automatically powers off to simulate door closing, and the screen returns to the original prompt. The entire operation is managed through a state machine, enabling intelligent door control and smooth user interaction.

Programming Steps

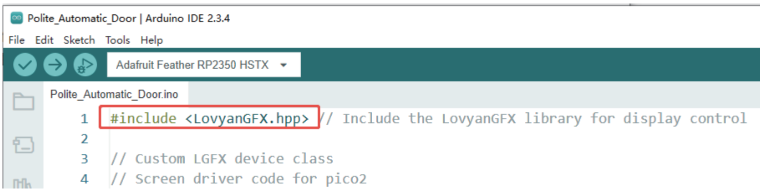
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**LovyanGFX-develop**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

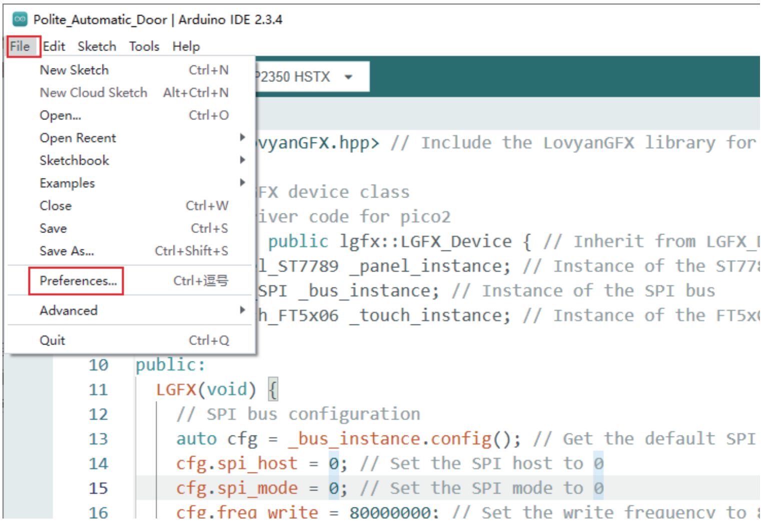
• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop** folder.


Polite_Automatic_Door | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
Polite_Automatic_Door.ino

```
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control

// Custom LGFX device class
// Screen driver code for pico2
```

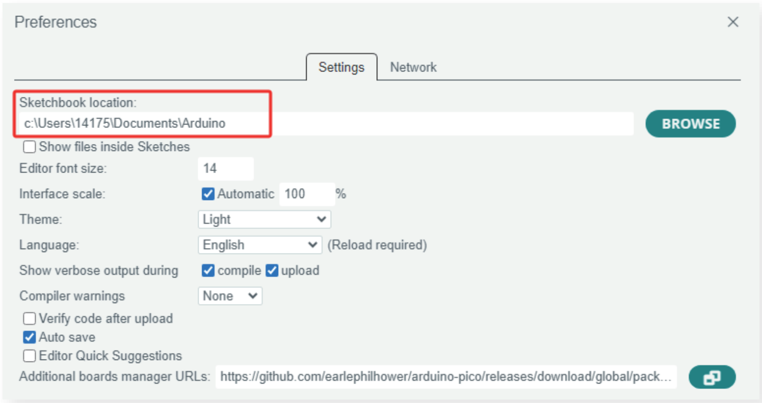
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
vyanGFX.hpp> // Include the LovyanGFX library for

FX device class
iver code for pico2
 public lgfx::LGFX_Device { // Inherit from LGFX_I
l_ST7789 _panel_instance; // Instance of the ST77
SPI _bus_instance; // Instance of the SPI bus
h_FT5x06 _touch_instance; // Instance of the FT5x
public:
  LGFX(void) {
    // SPI bus configuration
    auto cfg = _bus_instance.config(); // Get the default SPI
    cfg.spi_host = 0; // Set the SPI host to 0
    cfg.spi_mode = 0; // Set the SPI mode to 0
    cfg.freq_write = 80000000; // Set the write frequency to
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



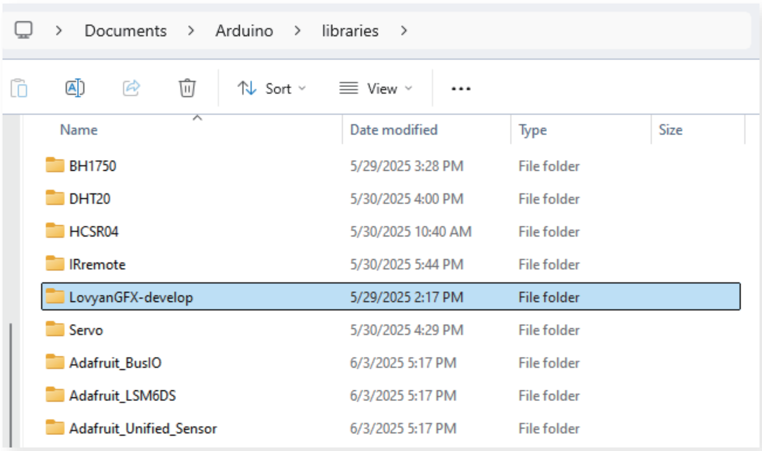
Preferences

Settings | Network

Sketchbook location:
c:\Users\14175\Documents\Arduino

Show files inside Sketches	☐
Editor font size:	14
Interface scale:	✓ Automatic 100 %
Theme:	Light
Language:	English (Reload required)
Show verbose output during	✓ compile ✓ upload
Compiler warnings	None
Verify code after upload	☐
Auto save	✓
Editor Quick Suggestions	☐
Additional boards manager URLs:	https://github.com/earlephilhower/arduino-pico/releases/download/global/pack...

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



> Documents > Arduino > libraries >

Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
Servo	5/30/2025 4:29 PM	File folder	
Adafruit_BusIO	6/3/2025 5:17 PM	File folder	
Adafruit_LSM6DS	6/3/2025 5:17 PM	File folder	
Adafruit_Unified_Sensor	6/3/2025 5:17 PM	File folder	

3. Upload and Run the Code

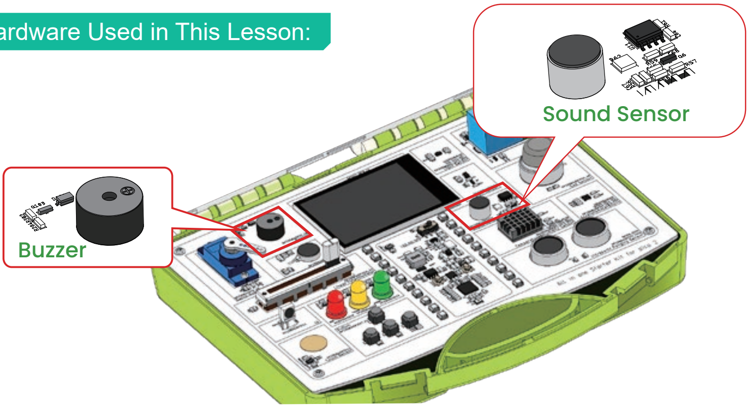
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 16 - Sound Reminder

Introduction

In this section, a sound sensor is used to monitor ambient noise levels. When the noise exceeds a preset threshold, a buzzer is triggered to emit a 1-second alert tone, creating a “noise detection–buzzer alert” linkage. The system continuously monitors sound intensity; if the noise persists, the buzzer repeats the alert. Once the noise stops, the reminders automatically cease.

Hardware Used in This Lesson:



Working Principle of Sound Sensor

A sound sensor transduces acoustic vibrations into electrical signals via transducer elements. Electret microphones generate signals through capacitance variation between diaphragm and backplate, while piezoelectric types rely on deformation potentials. Signals are pre-amplified, filtered, and digitized by ADC. Sensitivity and frequency response depend on diaphragm material and structural design, enabling specific sound pressure detection.

Working Principle of Buzzer Operation

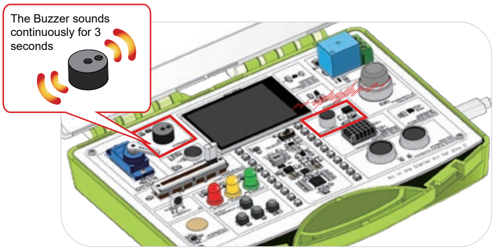
The buzzer generates sound by vibrating a diaphragm driven by an electrical signal. When an alternating current is applied, the diaphragm vibrates rapidly due to magnetic or piezoelectric effects, producing sound. The pitch and frequency are determined by the current frequency, and it is commonly used for alerts or alarms.

Operation Effect Diagram

Once the program starts, the system continuously monitors ambient sound in real time:

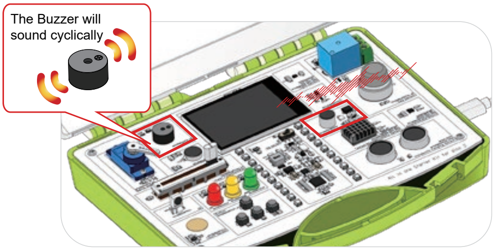
1. Sound Trigger:

When the surrounding noise exceeds the defined threshold, the buzzer emits a “beep” sound that lasts for 3 seconds.



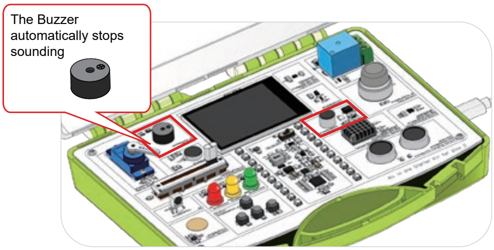
2. Continuous Alerts:

If the noise level remains high, the buzzer will repeatedly sound at regular intervals.



3. Auto-Silence:

When the environment becomes quiet again, the buzzer stops automatically.



Key Explanations

1. Pin Definitions for Sound Sensor and Buzzer

```
#define SOUND_PIN 29
int buzzerPin = 10;
```

- SOUND_PIN is defined as pin 29, which connects to the sound sensor.
- buzzerPin is a variable representing the buzzer connected to pin 10.
- Subsequent code will use these two pins for reading input and controlling output.

2. Reading Analog Values from the Sound Sensor

```
int soundValue = analogRead(SOUND_PIN);
```

- The function analogRead(...) is used to read the analog signal, which corresponds to the voltage output by the sound sensor (range: 0–1023).
- The louder the sound, the higher the voltage, and thus the larger the soundValue.
- This value determines whether the buzzer alarm is triggered.

3. Determining if the Sound Exceeds the Set Threshold

```
if (soundValue >= 300)
```

- This is the key condition: if the analog value from the sound sensor is greater than or equal to 300, it indicates a significant sound in the environment (such as clapping or shouting).
- The value 300 is an empirical threshold and can be adjusted to suit the actual environment sensitivity.

4. Controlling the Buzzer On or Off

• 1. When the sound is loud:

```
tone(buzzerPin, 1300);
delay(1000);
```

- tone(buzzerPin, 1300); plays a 1300Hz tone on the buzzer pin.
- delay(1000); keeps the buzzer sounding for 1 second to enhance the alert

• 2. When the sound is quiet:

```
noTone(buzzerPin);
```

- noTone(...) stops the buzzer, keeping it silent.
- This prevents continuous buzzing that might annoy users.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you will have developed an intelligent environmental monitoring system based on a sound sensor and buzzer. The system continuously reads the ambient sound intensity in real time and compares it to a preset threshold. When the detected sound exceeds the threshold, the buzzer automatically triggers a continuous alert. The design supports customizable sound sensitivity, alert duration, and sampling frequency, and includes a safety feature to turn off the buzzer if the program is interrupted.

Programming Steps

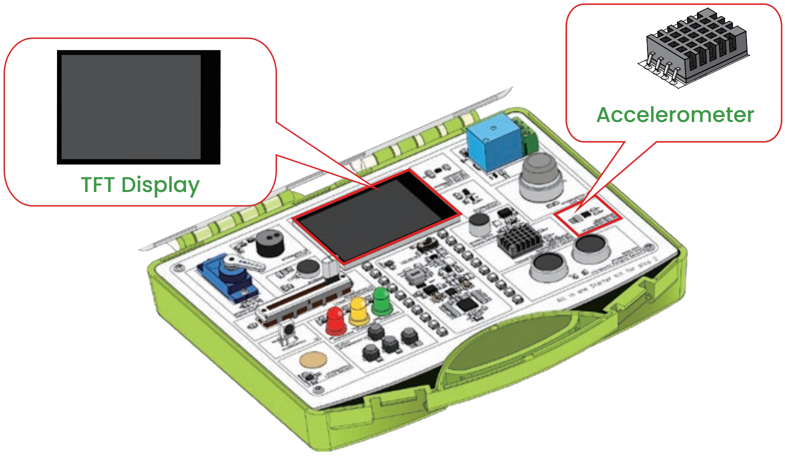
You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.

Lesson 17 - Calculation Of Acceleration

Introduction

This section utilizes the LSM6DS3TRC accelerometer sensor to continuously capture acceleration data along the X, Y, and Z axes of the development board. The measurements are displayed in real time on a TFT screen. When the board is moved rapidly, the acceleration values on the corresponding axes change dynamically, enabling a visual correlation between motion and acceleration.

Hardware Used in This Lesson:



Working Principle of Accelerometer

An accelerometer measures acceleration by detecting inertial force. MEMS types use silicon microstructures where proof mass displacement alters capacitance/piezoresistance, converted to electrical signals. Piezoelectric variants generate charges when stressed. Integrated signal conditioning enables static (gravity) and dynamic acceleration

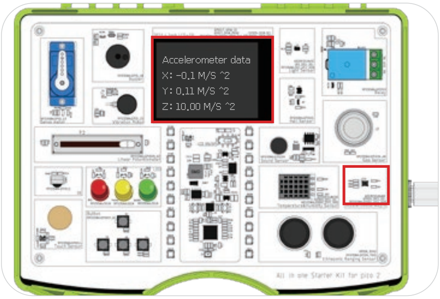
Working Principle of Display Screen

The display controls each pixel's brightness/color via driver circuits. The main IC converts image data to electrical signals, transmitted to row/column driver ICs for line-by-line refreshing. LCDs adjust liquid crystal alignment with voltage to modulate light, while OLEDs emit light directly. A timing controller ensures signal synchronization to prevent artifacts.

Operation Effect Diagram

1. System Auto-Initialization

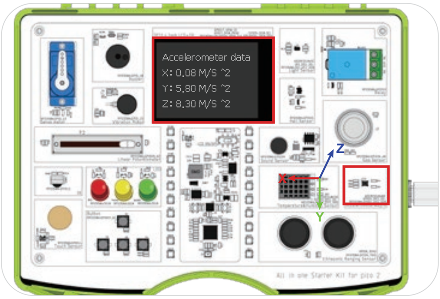
The TFT screen displays the title "Accelerometer data" with labels for the X, Y, and Z axes and their initial values shown below.



```
Accelerometer data
X: -0.1 M/S ^2
Y: 0.11 M/S ^2
Z: 10.00 M/S ^2
```

2. Observing Movement Along Axes

Shake the board rapidly in any direction: the X, Y, and Z axis values update dynamically in real time, reflecting the direction and magnitude of acceleration in 3D space



```
Accelerometer data
X: 0.08 M/S ^2
Y: 5.80 M/S ^2
Z: 8.30 M/S ^2
```

Key Explanations

1. Display Initialization and Configuration

```
gfx.init();
pinMode(0, OUTPUT);
digitalWrite(0, HIGH);
gfx.fillScreen(BG_COLOR);
```

- gfx.init(): Calls the LovyanGFX library method to initialize the TFT display.
- pinMode(0, OUTPUT) and digitalWrite(0, HIGH): Turns on the backlight to ensure the screen is lit.
- gfx.fillScreen(BG_COLOR): Fills the screen with the background color (black) to prevent ghosting.

2. Display Initialization and Configuration

```
Wire1.setSDA(2);
Wire1.setSCL(3);
Wire1.begin();
```

- Specifies using Wire1 (the second I2C bus) for communication.
- Sets SDA to GPIO2 and SCL to GPIO3.
- Starts the I2C bus to prepare for communication with the LSM6DS3TR-C sensor.

```
if (!lsm6ds3trc.begin_I2C(0x6B, &Wire1)) {
  gfx.setTextColor(TFT_RED);
  gfx.setCursor(20, SCREEN_HEIGHT / 2 - 20);
  gfx.print("Sensor initialization failed!");
  while (1) delay(10);
}
```

- begin_I2C(0x6B, &Wire1): Initializes the LSM6DS3TR-C sensor at address 0x6B.
- If initialization fails, an error message is displayed on the screen, and the system enters an infinite loop to prevent ignoring the error.

3. Data Refresh Control Logic

```
static unsigned long lastUpdate = 0;
if (millis() - lastUpdate < UPDATE_INTERVAL) return;
lastUpdate = millis();
```

- static unsigned long lastUpdate = 0; defines a static variable lastUpdate to store the timestamp of the last accelerometer data refresh. Using static ensures that the variable retains its value between function calls instead of being reinitialized every time.
- if (millis() - lastUpdate < UPDATE_INTERVAL) return; is the core logic for controlling data refresh frequency. It prevents excessive updates that could cause screen flickering or waste system resources.

  • Does the current time minus the last update time fall short of the preset refresh interval (UPDATE_INTERVAL = 150 ms)?

  • If yes, the refresh time has not yet arrived, so return exits the loop() early, skipping data acquisition and screen update during this cycle.
- millis() is a built-in Arduino function returning the number of milliseconds since the device powered on, useful for timing intervals.
- lastUpdate = millis(); updates lastUpdate to the current timestamp once the interval requirement is met, preparing for the next refresh cycle.

4. Retrieving Three-Axis Acceleration Data

```
sensors_event_t accel;
lsm6ds3trc.getEvent(&accel, nullptr, nullptr);
```

- sensors_event_t is a data structure from the Adafruit Sensor library.
- getEvent() fetches the latest acceleration values from the sensor in meters per second squared (m/s²).
- Only acceleration data is requested here; the second and third parameters are set to nullptr, indicating that gyroscope and temperature data are not retrieved

5. Displaying Data on the Screen (Clearing Old Values + Writing New Values)

```
for (int i = 0; i < 3; i++) {
  float value = 0;
  switch (i) {
    case 0: value = accel.acceleration.x; break;
    case 1: value = accel.acceleration.y; break;
    case 2: value = accel.acceleration.z; break;
  }
  gfx.fillRect(
    AXIS_AREA[i].x_value,
    AXIS_AREA[i].y_value,
    AXIS_AREA[i].w,
    AXIS_AREA[i].h,
    BG_COLOR
  );
  gfx.setTextColor(TEXT_COLOR);
  gfx.setCursor(AXIS_AREA[i].x_value, AXIS_AREA[i].y_value);
  gfx.printf("%.2f m/s^2", value);
}
```

- gfx.fillRect(...) — clears the screen area first to prevent ghosting.

  • Before displaying new values, the previous value area is erased using the background color.

  • This avoids leftover digits caused by changes in number length (eg: from 9.88 to 10.12),
- gfx.printf(...) — formats and prints the new value.

  • **Uses gfx.printf("%.2f m/s^2", value);** to output acceleration values rounded to two decimal places.

  • Ensures the display remains neat and the precision is appropriate.。
- The loop updates the X, Y, and Z axis data separately:

  • Uses predefined position and size info from AXIS_AREA[i] to locate where each axis is shown on the screen.

  • Each axis's data is refreshed individually to maintain an organized display.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you've built a real-time 3-axis acceleration monitoring system using the LSM6DS3TRC sensor. The system reads acceleration data via the I2C interface, processes it, and dynamically displays the X, Y, and Z values (in m/s²) on a 240×320 resolution TFT screen. Data updates every 150 milliseconds, and when the development board moves, the corresponding axis values change in real time, creating a visual link between motion and acceleration.

Programming Steps

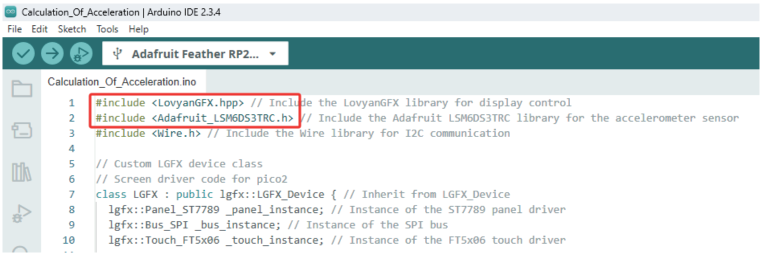
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use four additional library (**LovyanGFX-develop, Adafruit_BusIO, Adafruit_LSM6DS and Adafruit_Unified_Sensor**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

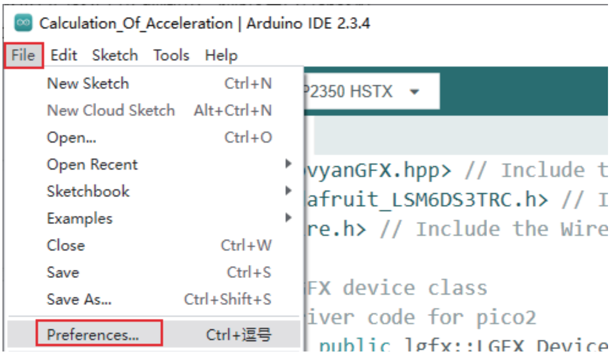
• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop, Adafruit_BusIO, Adafruit_LSM6DS and Adafruit_Unified_Sensor** folder.


Calculation_Of_Acceleration | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2...
Calculation_Of_Acceleration.ino

```
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control
#include <Adafruit_LSM6DS3TRC.h> // Include the Adafruit LSM6DS3TRC library for the accelerometer sensor
#include <Wire.h> // Include the Wire library for I2C communication

// Custom LGFX device class
// Screen driver code for pico2
class LGFX : public lgfx::LGFX_Device { // Inherit from LGFX_Device
  lgfx::Panel_ST7789 _panel_instance; // Instance of the ST7789 panel driver
  lgfx::Bus_SPI _bus_instance; // Instance of the SPI bus
  lgfx::Touch_FT5x06 _touch_instance; // Instance of the FT5x06 touch driver
```

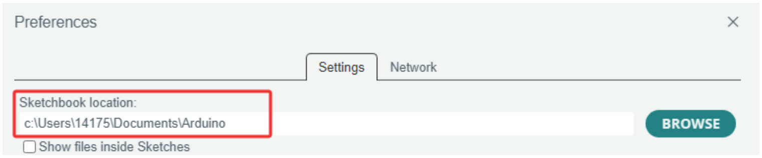
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



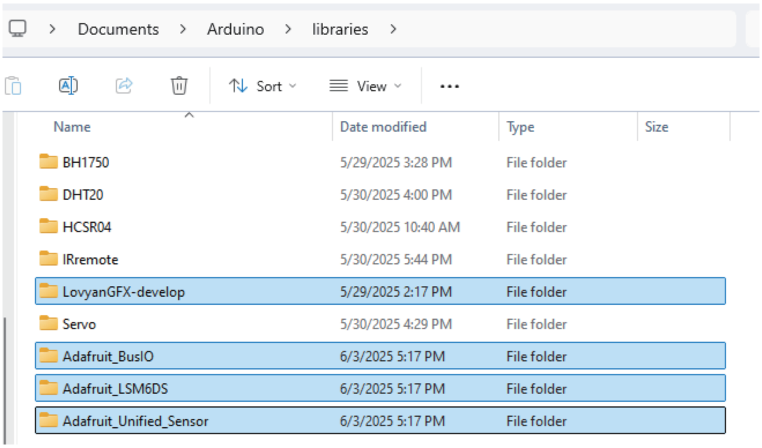
```
vyanGFX.hpp> // Include t
afruit_LSM6DS3TRC.h> // I
re.h> // Include the Wire

FX device class
iver code for pico2
 public lgfx::LGFX_Device
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
Show files inside Sketches

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
Servo	5/30/2025 4:29 PM	File folder	
Adafruit_BusIO	6/3/2025 5:17 PM	File folder	
Adafruit_LSM6DS	6/3/2025 5:17 PM	File folder	
Adafruit_Unified_Sensor	6/3/2025 5:17 PM	File folder	

3. Upload and Run the Code

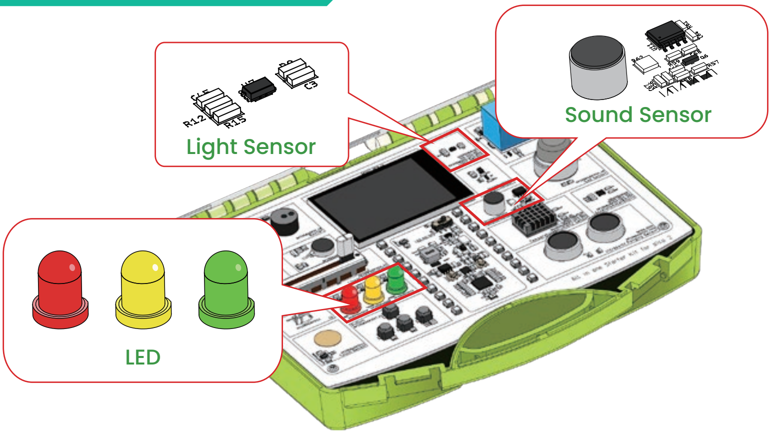
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 18 - Smart Corridor Light

Introduction

In this chapter, you'll learn how to integrate a sound sensor, a light sensor, and an LED to build a smart hallway lighting system. By coordinating multiple sensors, the system can automatically control the LED based on ambient light and sound input—mimicking real-world intelligent lighting behavior.

Hardware Used in This Lesson:



Working Principle of Sound Sensor

A sound sensor transduces acoustic vibrations into electrical signals via transducer elements. Electret microphones generate signals through capacitance variation between diaphragm and backplate, while piezoelectric types rely on deformation potentials. Signals are pre-amplified, filtered, and digitized by ADC. Sensitivity and frequency response depend on diaphragm material and structural design, enabling specific sound pressure detection.

Working Principle of a Light Sensor

A light sensor converts light signals to electrical signals via the photoelectric effect. Ambient light strikes a photosensitive component (eg: photoresistor, photodiode, or phototransistor), where photons excite charge carriers, altering resistance or generating photocurrent. Signal conditioning (eg: ADC/amplifier) processes the output, and an MCU calculates light intensity from voltage/current changes. Different sensors detect visible, IR, or UV wavelengths.

Working Principle of LED

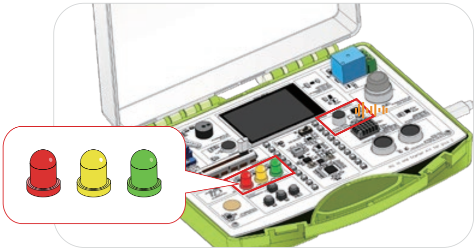
At the heart of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During this recombination, electrons drop from a higher energy level to a lower one, releasing the excess energy in the form of photons—producing light. The color (or wavelength) of the emitted light is determined by the energy band gap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

After the system is powered on, the smart corridor light follows this logic:

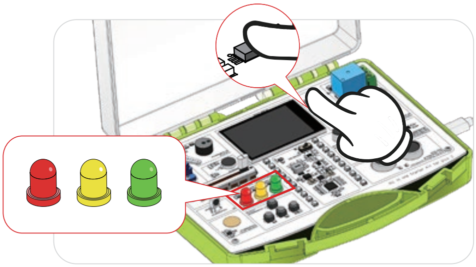
1. Bright Environment:

When there is sufficient natural light, the LED remains off regardless of any sound detected.

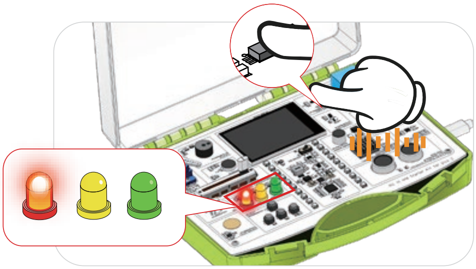


2. Low-Light Environment:

When the lighting is dim or the light sensor is covered, the LED stays off by default.



Once a sound is detected, the LED turns on and stays lit for 10 seconds before automatically turning off.



If the sound continues, the LED remains on continuously. Once the sound stops, it stays on for another 10 seconds before shutting off automatically.This demonstrates that the system operates as expected in all scenarios.

Key Explanations

1. I2C Initialization and Light Sensor Setup

```
Wire1.setSDA(I2C_SDA);
Wire1.setSCL(I2C_SCL);
Wire1.begin();
if (lightMeter.begin(BH1750::CONTINUOUS_HIGH_RES_MODE,
0x5c, &Wire1)) {
    Serial.println(F("BH1750 begin success"));
} else {
    Serial.println(F("BH1750 init failed"));
}
```

This section of code initializes the I2C communication and sets up the BH1750 light sensor:

- Wire1.setSDA() and Wire1.setSCL() define the pins used for I2C communication;
- Wire1.begin() starts the I2C bus;
- lightMeter.begin() initializes the light sensor with the following parameters:
  - CONTINUOUS_HIGH_RES_MODE: sets the sensor to continuous high-resolution measurement mode;
  - 0x5c: the I2C address of the sensor;
  - &Wire1: specifies the I2C interface to use.

2. Main Control Logic Entry

```
static bool isSoundDetected = false;
static bool isLedOn = false;
static unsigned long ledOnTimestamp = 0;
```

The core logic consists of a three-stage process:

Light intensity judgment → Sound trigger detection → Timed LED control, This is implemented through state flags (isSoundDetected, isLedOn) and a timestamp (ledOnTimestamp) to manage smart linkage behavior.

The main logic is handled inside the if (lightMeter.measurementReady(true)) function, with the following process:

• 1. Light Sensor Reading

```
int lux = lightMeter.readLightLevel();
```

- measurementReady(true) checks if the sensor has new data available
- The returned lux value determines the subsequent logic branch (bright/dim light handling).

• 2. Bright Light Handling (lux ≥ 100)

```
digitalWrite(LedPin, LOW);
isLedOn = false;
isSoundDetected = false;
ledOnTimestamp = 0;
Serial.println("[ACTION] Light strong - turning OFF LED");
```

- Immediately turn off the LED (digitalWrite(LedPin, LOW))
- Reset all state flags:
  - **isLedOn = false (LED status)**
  - **isSoundDetected = false (sound detection flag)**
  - **ledOnTimestamp = 0 (timestamp)**

• 3. Dim Light Handling (lux < 100)

```
if (digitalRead(SOUND_PIN)) {
    digitalWrite(LedPin, HIGH);
    isLedOn = true;
    isSoundDetected = true;
    ledOnTimestamp = millis();
} else {
    if (isSoundDetected) {
        if (millis() - ledOnTimestamp < 10000) {
            digitalWrite(LedPin, HIGH);
            isLedOn = true;
        } else {
            digitalWrite(LedPin, LOW);
            isLedOn = false;
            isSoundDetected = false;
        }
    } else {
        digitalWrite(LedPin, LOW);
        isLedOn = false;
    }
}
```

- **When a sound signal is detected (digitalRead(SOUND_PIN) returns HIGH):**
  - Immediately turn on the LED (digitalWrite(LedPin, HIGH)):
  - Set the relevant state flags:
    - isLedOn = true
    - isSoundDetected = true
  - Record the current time (millis()) as the LED activation timestamp
- **Sustained Sound:**
- **If no new sound is detected but one was detected previously:**
  - Keep the LED on for up to 10 seconds (millis() - ledOnTimestamp < 10000)
  - After 10 seconds, turn off the LED and reset flags
- **If no sound is detected:**
  - Turn off the LED directly (digitalWrite(LedPin, LOW))
  - Update the isLedOn flag to false

3. Pin Mode Configuration

```
pinMode(LedPin, OUTPUT);
pinMode(SOUND_PIN, INPUT);
```

- LedPin is a defined constant (value 18), representing the physical pin number connected to the LED
- OUTPUT is a predefined constant that sets the pin to output mode.
- **Purpose:**
  - Configures the pin as a digital output, capable of outputting HIGH (3.3V/5V) or LOW (0V)
  - In this mode, the pin can drive an LED, relay, or other load
- SOUND_PIN is a defined constant (value 29), representing the physical pin number connected to the sound sensor
- INPUT is a predefined constant that sets the pin to input mode.
- **Purpose:**
  - Configures the pin as a digital input, capable of reading HIGH/LOW levels from external signals
  - In this mode, the pin can detect switch states or digital sensor outputs

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the above code, you have implemented an intelligent corridor lighting control system based on a light sensor and a sound sensor. The system controls the LED light through the interaction of ambient light intensity and sound signals:

- When the light intensity exceeds 100 lux, the LED remains off;
- In dim lighting, if a sound is detected, the LED turns on and stays lit for 10 seconds;
- If continuous sound is detected during this period, the timer resets;
- Once the sound stops and the countdown ends, the LED automatically turns off.

Programming Steps

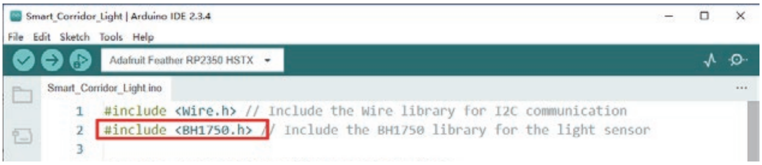
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**BH1750**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

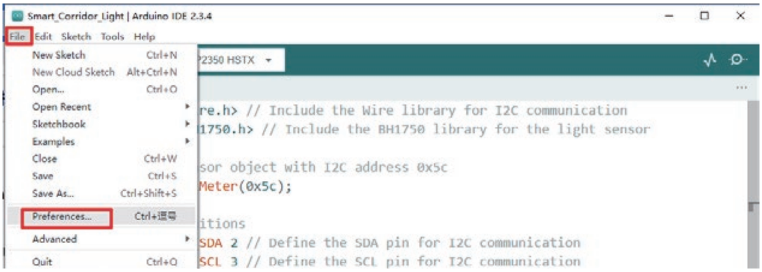
• Navigate to the **/example/libraries** directory and download the **BH1750** folder.


Smart_Corridor_Light | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
Smart_Corridor_Light.ino

```
#include <Wire.h> // Include the Wire library for I2C communication
#include <BH1750.h> // Include the BH1750 library for the light sensor
```

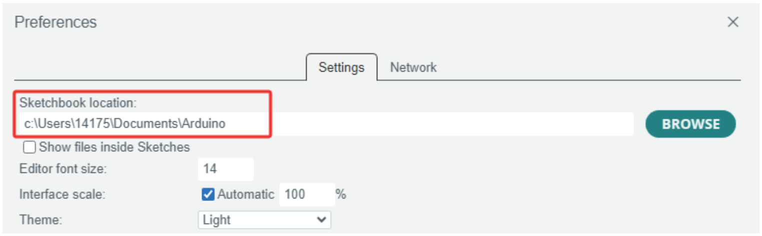
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



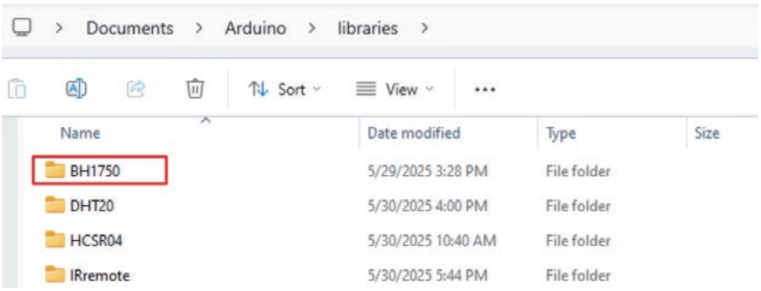
```
re.h> // Include the Wire library for I2C communication
1750.h> // Include the BH1750 library for the light sensor

sor object with I2C address 0x5c
Meter(0x5c);

itions
SDA 2 // Define the SDA pin for I2C communication
SCL 3 // Define the SCL pin for I2C communication
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Light

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	

3. Upload and Run the Code (**You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide**.)

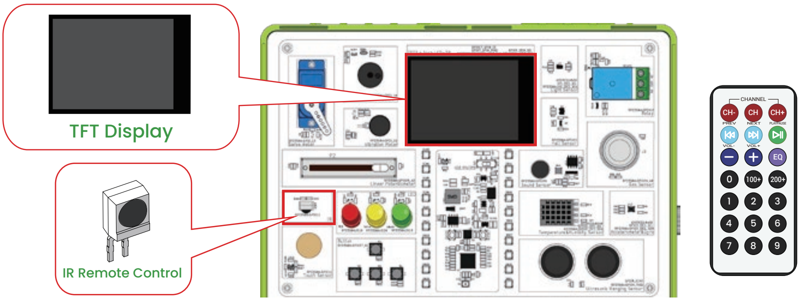
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 19 - Simple Calculator

Introduction

In this chapter, a basic calculator function will be implemented using an IR remote control. Users can input numbers and operators via the remote, and the expressions will be displayed and evaluated on a TFT screen.The main objective is to master input handling from the IR remote, number parsing, and the programming of simple arithmetic logic.

Hardware Used in This Lesson:



Principle of Infrared Remote Sensor Operation

The infrared remote sensor works by receiving infrared signals to achieve control functions. The transmitter modulates control instructions into infrared light signals of a specific frequency. The sensor receives these signals, demodulates them back into control instructions, and then the microcontroller performs the corresponding operations.

Principle of Infrared Remote Control Operation

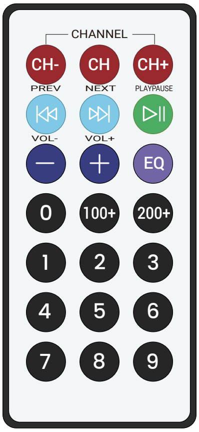
The infrared remote control works by inputting control commands through buttons. The microcontroller encodes and modulates these commands into infrared signals, which are then transmitted by an infrared emitter. The receiving end's infrared sensor captures the signals, demodulates them back into commands, and drives the device to perform the corresponding operations.

Working Principle of Display Screen

The display controls each pixel's brightness/color via driver circuits. The main IC converts image data to electrical signals, transmitted to row/column driver ICs for line-by-line refreshing. LCDs adjust liquid crystal alignment with voltage to modulate light, while OLEDs emit light directly. A timing controller ensures signal synchronization to prevent artifacts.

Operation Effect Diagram

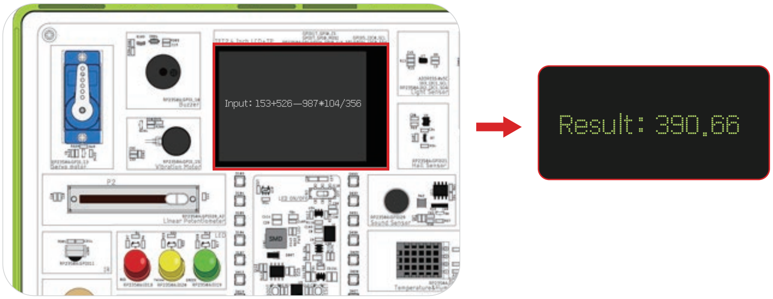
Operator and Control Key Functions



Key Type	Key Label	Function Description
Numeric Keys	0–9	Inputs numeric values; LCD displays “Input: [value]”.
Operator Keys	+	Performs addition; LCD shows current input (eg: “Input: 3+”).
	-	Performs subtraction; LCD shows current input (eg: “Input: 3–”).
	NEXT	Performs multiplication (equivalent to “×”); LCD shows current input (eg: “Input: 3*”).
	PREV	Performs division (equivalent to “/”); LCD shows current input (eg: “Input: 3/”).
Special Keys	-(negative)	Used to input negative numbers; press this key before the number (eg: “-5”).
	100+	Acts as a decimal point; used to enter decimals (eg: press “3 → 100+ → 5” for “3.5”).
	Equals (=)	Confirms calculation and outputs result, rounded to two decimal places (eg: “Result: 8.00”).
	EQ (Clear)	Clears current input and result; returns to initial state (LCD shows “Input:”).

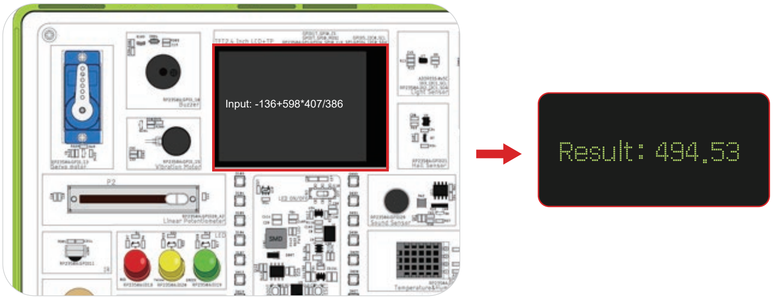
1. Basic Calculation Process

- Press numeric keys to enter values.The LCD displays: “Input: [entered value]”.For example, after entering “153”, the LCD shows: “Input: 153”.。
- Press an operator key (+, –, VOL+ (×), or VOL– (/)) to continue the input.For instance, after entering “153” and then pressing “+”, the LCD shows: “Input: 153+”.
- Press more numeric keys to input another value.For example, entering “526” shows: “Input: 153+526”.You can continue entering a full expression like: “Input: 153+526–987*104/356”.
- Press the equals key (=) to evaluate the expression.The result will be displayed on the LCD with two decimal places.For example, the LCD will show: “Result: 390.66”.



2. Operations Starting with a Negative Sign

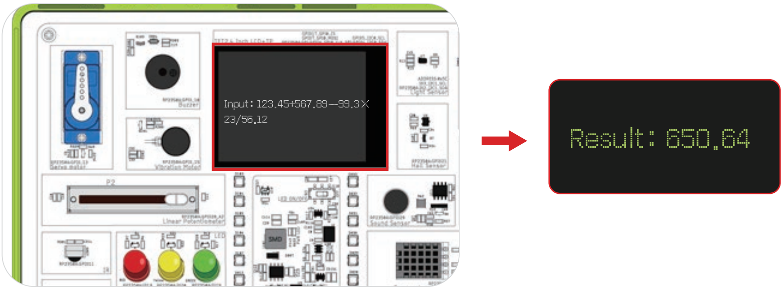
- First, press the minus key “-”, then input the number. For example, inputting “-136” will display on the LCD as “Input: -136”.
- Press an operator key such as “+”, and the LCD will display “Input: -136+”.
- Enter another number, for example “598”, and the LCD will update to “Input: -136+598”. Continue entering the full expression, such as “Input: -136+598*407/386”.
- Press the confirm key to calculate and display the result. In this example, the result is shown as “Result: 494.53”.



3. Decimal Calculation Process

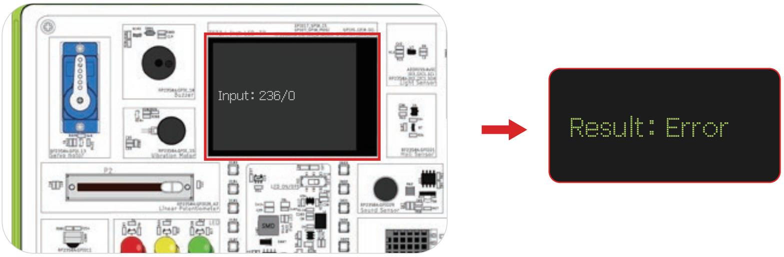
- To enter a decimal number, use the “100+” key as the decimal point.For example, to input “123.45”: first press “123”, then “100+”, and finally “45”.The LCD will show: “Input: 123.45”.
- Press an operator key, such as “+”.The LCD updates to show: “Input: 123.45+”.
- Enter another decimal or integer.For example, entering “567.89” will result in: “Input: 123.45+567.89”.You may continue inputting a more complex expression, such as:“Input: 123.45+567.89–99.3×23/56.12”.
- Press the equals key (=) to evaluate the expression.

  The result will be displayed with two decimal places, eg: “Result: 650.64”.



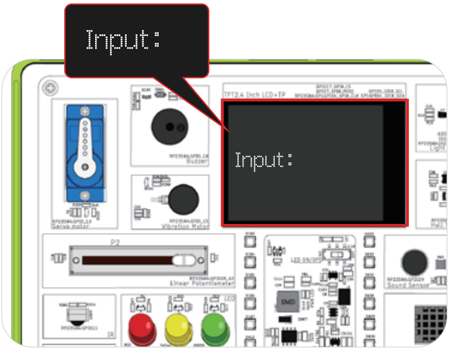
4. Division by Zero Handling Process

- After entering a number and the “/” operator, then inputting “0” (eg: “236/0”),the LCD will display: “Input: 236/0”.
- When the equals key is pressed, the LCD shows: “Result: Error” to indicate an error.
- When a division operation is detected, the system checks if the divisor is zero:if (b == 0) return NAN;If the divisor is 0, the function immediately returns a NAN (Not-a-Number) value to prevent invalid calculation.This NAN value is passed up to the main loop, where isnan(result) is evaluated.If true, the system displays “Error” on the screen to inform the user.This mechanism provides immediate interception and consistent handling of division-by-zero errors,ensuring calculator stability and a user-friendly experience.



5. Using the EQ Clear Key

- During the input process or after completing an expression, if an input error is detected or the user wants to start a new calculation, they can press the EQ Clear Key.
- The LCD will clear the current “Input: [entered content]” along with any previous result,returning to the initial state and waiting for new input.



Key Explanations

1. Infrared Receiver Initialization

```
IrReceiver.begin(11, DISABLE_LED_FEEDBACK);
```

- IrReceiver.begin(pin, option) is a method from the IRremote library used to initialize the infrared receiver.
- The parameter 11 indicates that the IR receiver’s data output pin (OUT) is connected to GPIO11.
- DISABLE_LED_FEEDBACK disables the default onboard LED blinking feedback used by the library, which would otherwise toggle the built-in LED (eg: pin 13) to indicate signal reception.
- This line ensures that the IR receiver module works properly and can respond to remote control key presses.Without correctly setting this pin, the system will not be able to receive IR commands.

2. Backlight Control

```
pinMode(0, OUTPUT);
digitalWrite(0, HIGH);
```

- Set GPIO0 as an output pin to control the LCD backlight power.
- digitalWrite(0, HIGH) sets the backlight control pin to high, which turns on the screen backlight.
- If this line is missing, the display may remain black (no visible content) due to the backlight being off, even though the screen is properly rendering the content.

3. Expression Input Processing (IR Handling in loop())

```
switch (code) {
  case 0x16: input += "0"; break;
  case 0x0C: input += "1"; break;
   ....
  case 0x15: input += "+"; break;
  case 0x07: input += "-"; break;
  case 0x44: input += "/"; break;
  case 0x40: input += "*"; break;
  case 0x09: input = ""; break;
  case 0x43: {
    float result = evaluateExpression(input);
  }
  case 0x19: {
    if (!(input.length() == 0 || input.endsWith(".") || isOperator(input.back()))) {
      input += ".";
    }
    break;
  }
}
```

This part uses a switch-case structure to handle IR remote key codes (code) and update the calculator's input string input accordingly.

- **case 0x16: input += "0"; break;:** Each case corresponds to a remote control button. The ellipsis in the code represents other numeric keys (0-9) corresponding to the cases. When a numeric key is pressed (for example, 0x16 corresponds to 0, 0x0C corresponds to 1, and so on), the number is directly appended to the end of the input string. For the complete code, please check the link to the full code below..
- **case 0x15: input += "+"; break;:** When an operator key (such as +, -, *, /, etc.) is pressed, the corresponding symbol is appended to the input string.
- **case 0x09: input = ""; break;:** When the clear key (eg: 0x09) is pressed, the input string is directly cleared.
- **OK key (0x43):** When the user presses the "=" or "OK" key (corresponding to the infrared code 0x43), a calculation operation is triggered.
- **float result = evaluateExpression(input);:** The evaluateExpression(input) function is called to compute the current input expression string (eg: "3+5*2"), and the result is stored in the result variable.
- **case 0x19:** This corresponds to the decimal point key on the infrared remote control (assuming 0x19 is the infrared code for ".").

- **if (!(input.length() == 0 || input.endsWith(".") || isOperator(input.back()))):** Checks if the decimal point can be entered at this time. If allowed, "." is appended to the input string.
- **input.length() == 0:** The input is empty, preventing a direct leading "." (eg: ".5").
- **input.endsWith("."):** The last character is ".", preventing consecutive decimal points (eg: "5..2").
- **isOperator(input.back()):** The last character is an operator (+, -, *, /), preventing a direct "." after an operator (eg: "5+.3").

4. Computation Logic

• 4.1 Infix to Postfix Conversion： infixToPostfix()

1. **infixToPostfix():** This function converts an infix expression (eg: 3 + 4 * 2) into a postfix expression (eg: 3 4 2 * +).It handles operator precedence and parentheses to ensure correct order of operations.

- The **infixToPostfix** function is responsible for converting infix expressions into postfix notation.

👉 Input (infix) → infixToPostfix() → Output (postfix)

2. **Infix Expression:**： This is the standard format we normally use, where operators are placed between operands.

- **Examples:**

  • 3 + 4 (addition)

  • 10 * (5 - 2) (multiplication with parentheses)

  • (2 + 3) / 5 (division with parentheses)

- In the code, the input variable holds the user-entered infix expression via IR remote (eg: "3+4*2").
- The evaluateExpression() function first converts this infix input to postfix using infixToPostfix() before evaluating it.

2. **Postfix Expression (also known as Reverse Polish Notation):**

- • In this notation, the operator comes after the operands. This format eliminates the need for parentheses and is easier to process using a stack.
- **Examples:**

  • Infix: 3 + 4 → Postfix: 3 4 +

  • Infix: 10 * (5 - 2) → Postfix: 10 5 2 - *

  • Infix: (2 + 3) / 5 → Postfix: 2 3 + 5 /

1. Initialization Section

```
bool infixToPostfix(const String &infix, String &postfix) {
  std::stack<char> stack;
  String num = "";
}
```

- The Infix To Postfix function converts infix expressions into postfix notation (also known as Reverse Polish Notation).
- std::stack<char> stack: Creates a stack to temporarily hold operators.
- Initialize the string 'num' to temporarily store numbers, including multi-digit and decimal numbers.

2. Handling the Negative Sign (Unary Operator)

```
if (c == '-' && (num.isEmpty() || isOperator(infix[i-1]))) {
    num += c;
    continue;
}
```

- **Key conditions:**

  • The current character is a "-"

  • And there is no preceding number (i.e., 'num' is empty) or the previous character is an operator
- **Example:**

  • In the expression "-5 + 3", the "-" is recognized as a negative sign

  • In the expression "3 - 5", the "-" is treated as a subtraction operator (so it doesn't enter this case)

3. Core Logic

```
if (isdigit(c) || c == '.') {
  num += c;
} else if (isOperator(c)) {
  if (num.length() > 0) {
    postfix += num + " ";
    num = "";
  }
  while (!stack.empty() && precedence(stack.top()) >= precedence(c)) {
    postfix += stack.top();
    postfix += " ";
    stack.pop();
  }
  stack.push(c);
} else {
  return false;
}
```

Part 1: Handling Numbers and Decimal Points

```
if (isdigit(c) || c == '.') {
num += c;
}
```

- if (isdigit(c) || c == '.') checks whether the current character c is a digit (using isdigit(c)) or a decimal point ('.'). If yes, it gets added to the temporary number string 'num'.
- num += c; appends the character to 'num', which temporarily holds continuous digits until an operator or the end of the expression is encountered.

▪ 4. Operator Handling

```
else if (isOperator(c)) {
    if (!num.isEmpty()) {
        postfix += num + " ";
        num = "";
    }
    while (!stack.empty() && precedence(stack.top()) >= precedence(c)) {
        postfix += stack.top() + " ";
        stack.pop();
    }
    stack.push(c);
}
```

Suppose the infix expression being processed is 3+4*2, and the current operator scanned is “+”:

- **1. else if (isOperator(c)) { ... }**
  - **Purpose:** This branch runs when an operator (such as +, -, *, /) is encountered.
  - **Example:** When the "+" is scanned, this branch is triggered.
- **2. if (num.length() > 0) { ... }**
  - **Purpose:** Checks if a complete number has been accumulated (eg: 3 or 3.14).
  - **Logic:**
    - If 'num' is not empty (meaning a number was scanned before), add it to the postfix expression and clear 'num' to prepare for the next number.
    - If 'num' is empty (such as when operators appear consecutively like the second "-" in 3+-4), this step is skipped.
  - **Example:**
    - When "+" is scanned and 'num' is "3" (length > 0), execute postfix += "3 ", so the postfix expression becomes "3 ", and 'num' is cleared.
- **3. postfix += num + " "**
  - **Purpose**:
    - Adds the accumulated number (like 3 or 3.14) to the postfix expression, followed by a space for separation.

- **Example:**
  - After adding 3, the postfix expression becomes "3 ".
  - As 4 and 2 are scanned later, the postfix expression updates to "3 4 " and then "3 4 2 ".

4. num = ""

- **Purpose:** Clears the temporary variable 'num' to get ready for the next number.
- **Example:**
  - After adding 3, 'num' resets from "3" to an empty string to prepare for collecting the next number, 4.

5. Operator Precedence Handling (details omitted)

- **Purpose:** Manage stack operations based on operator precedence.
- **Example:**
  - When scanning "+", the stack is empty, so "+" is pushed onto the stack directly.
  - When scanning "", since "" has higher precedence than the "+" on top of the stack, "*" is pushed onto the stack.
  - The final postfix expression becomes 3 4 2 * +.

Handling Operator Precedence:

```
while (!stack.empty() && precedence(stack.top()) >= precedence(c)) {
    postfix += stack.top() + " ";
stack.pop();
}
```

Loop conditions:

- **!stack.empty():** Process the top operator only when the stack is not empty.
- **precedence(stack.top()) >= precedence(c):** The precedence of the operator at the top of the stack is greater than or equal to that of the current operator c.
- **stack.pop():** Pop the top operator from the stack and append it to the postfix expression (eg: +, *).

5. Finalizing the Expression: Add the Last Number and Remaining Operators from the Stack to the Postfix Expression.

```
if (!num.isEmpty()) postfix += num + " ";
while (!stack.empty()) {
    postfix += stack.top() + " ";
    stack.pop();
}
```

- **if (!num.isEmpty()) postfix += num + " ":** Handles the last number in the expression. When reaching the end of the infix expression, the temporary variable 'num' may still hold the last number (eg: 3.14), which needs to be added to the postfix expression. If 'num' is not empty, add it to the postfix expression followed by a space (numbers and operators in postfix are separated by spaces).
- **while (!stack.empty()) { ... }:** Pops all remaining operators from the stack and appends them to the postfix expression. After scanning the infix expression, the stack may still contain operators (such as +, *), which should be popped in order and added to the postfix expression. Due to the stack's Last-In-First-Out nature, the popping order naturally respects operator precedence rules.

• 4.2 Evaluating Postfix Expressions: evaluatePostfix()

- **evaluatePostfix() is a function that calculates the value of a postfix expression (Reverse Polish Notation, RPN). It takes a space-separated postfix expression string (eg: "3 4 2 * +") and returns its computed result (eg: 11).**
- **Postfix Expression → evaluatePostfix() → Final Result.**

```
float evaluatePostfix(const String &postfix) {
    std::stack<float> stack;
    String token = "";
    for (int i = 0; i <= postfix.length(); ++i) {
        char c = postfix[i];
        if (c == ' ' || c == '\0') {
            if (token.length() > 0) {
                if (isOperator(token[0]) && token.length() == 1) {
                    if (stack.size() < 2) return NAN;
                    float b = stack.top(); stack.pop();
                    float a = stack.top(); stack.pop();
                    switch (token[0]) {
                        case '+': stack.push(a + b); break;
                        case '-': stack.push(a - b); break;
                        case '*': stack.push(a * b); break;
                        case '/':
                            if (b == 0) return NAN;
                            stack.push(a / b);
                            break;
                    }
                } else {
                    stack.push(token.toFloat());
                }
                token = "";
            }
        } else {
            token += c;
        }
    }
    return (stack.size() == 1) ? stack.top() : NAN;
}
```

1. Variable Initialization

```
std::stack<float> stack;
String token = "";
```

- **std::stack<float> stack;** This line defines a stack container named 'stack' specifically for storing operands of type float. Its main role is to temporarily hold numbers during the evaluation of the postfix expression.

  • **std::** indicates that the stack belongs to the C++ Standard Library.

  • **Stack:** The stack data structure is chosen because its Last-In-First-Out (LIFO) behavior perfectly matches the evaluation logic of postfix expressions (when an operator is encountered, the most recently pushed numbers are processed first).

  • **Float:** Specifies that the elements in the stack are floating-point numbers, allowing correct handling of integers, decimals, and negative numbers.

  • **stack**(variable name)**:** The name clearly reflects its purpose—to store operands for calculation, which are dynamically managed using push() and pop() operations.

- **String token = "":** This line initializes an empty string variable named 'token', which is used to build up numbers or operators character by character during postfix expression parsing, until a space or string end triggers processing.

  • **String:** Indicates this variable is a string type (likely Arduino's String class or similar), capable of flexibly storing multi-character numbers (eg: "3.14") or single-character operators (eg: "+").

  • **token** (variable name)**:** Named after the compiler concept "token," representing an independent unit extracted from the expression (a complete number or operator).

  • **= "":** Initialized as an empty string to ensure no leftover data before processing each new token.

2. Main Loop

```
for (int i = 0; i <= postfix.length(); ++i) {
    char c = postfix[i];
    if (c == ' ' || c == '\0') {
    } else {
        token += c;
    }
}
```

- **for (int i = 0; i <= postfix.length(); ++i):** The loop condition includes the equal sign (<=) to ensure the iteration reaches the string termination character \0, allowing correct termination even if there's no trailing space.
- **char c = postfix[i];** Retrieves the character at position i, which could be a digit, operator, space, or the string terminator \0.
- **if (c == ' ' || c == '\0'):** When the current character is a space or the string terminator \0, it indicates a complete token has been formed (eg: "5" or "*").

- **Separator branch (if):** Triggered when the token holds a complete unit (like "3.14" or "+"), then enters processing logic.
- **Non-separator branch (else):** Concatenates digits, decimal points, or operator characters to build a complete token.

  • For example, input "3.14" appends '3', '.', '1', '4' sequentially, resulting in token = "3.14".

  • Input "" directly sets token = "" (a single-character operator).

3. Processing the Token

- **1. If it is an operator:**

```
if (token.length() == 1 && isOperator(token[0])) {
    if (stack.size() < 2) return NAN;
    float b = stack.top(); stack.pop();
    float a = stack.top(); stack.pop();
    switch (token[0]) {
        case '+': stack.push(a + b); break;
        case '-': stack.push(a - b); break;
        case '*': stack.push(a * b); break;
        case '/': if (b == 0) return NAN; stack.push(a / b); break;
    }
}
```

- **Operator check: If the token is one of +, -, *, or /:**

  • 1. if (stack.size() < 2) return NAN; — Verify operand count: At least two numbers must be on the stack to perform the operation; otherwise, return NAN indicating an invalid expression.

  • 2. float b = stack.top(); stack.pop(); float a = stack.top(); stack.pop(); — Pop operands: The first popped value is the right operand b, and the second is the left operand a, ensuring the correct order for evaluation.

  • 3. Perform operation: Compute a op b according to the operator and push the result back onto the stack.

  • 4. case '/': if (b == 0) return NAN; stack.push(a / b); break; — Division by zero check: If the divisor b is zero, return NAN to prevent division errors.

- **2. If it is a number:**

```
else {
   stack.push(token.toFloat());}
```

- **stack.push(token.toFloat());: Converts the token to a float and pushes it onto the stack, preparing it for future operations.**

- **3. Reset token:**

```
token = "";
```

- **token = "";:** Resets the string variable token to an empty string, clearing the processed token (whether a number or operator) and preparing to parse the next expression unit.

▶ **4. Return the final result**

```
return (stack.size() == 1) ? stack.top() : NAN;
```

▶ **Using a conditional (ternary) operator with the following logic:**

- **stack.size() == 1:** Checks if there is exactly one element left in the stack.
- **stack.top():** If there is exactly one element, returns it as the final result.
- **NAN:** If the stack size is not one (empty or multiple leftover elements), returns NAN indicating invalid or erroneous calculation.

• 4.3 Helper Function

```
bool isOperator(char c) {
  return c == '+' || c == '-' || c == '*' || c == '/';
}
```

▶ **bool isOperator(char c)**

Parameter: c — the character to check

Return value: boolean; returns true if c is an operator, otherwise false

▶ **return c == '+' || c == '-' || c == '*' || c == '/';**

▶ Uses the logical OR operator (||) to check if the character c matches any of the four basic arithmetic operators:

- **+(addition)**
- **-(subtraction)**
- ***(multiplication)**
- **/ (division)**

5. Decimal Point Handling

```
case 0x19: {
    if (input.length() == 0 || input.endsWith(".") || isOperator(input[input.length() - 1])) {
        break;
    }
    int i = input.length() - 1;
    while (i >= 0 && (isdigit(input[i]) || input[i] == '.')) {
        if (input[i] == '.') break;
        --i;
    }
    if (i >= 0 && input[i] == '.') break;
    input += ".";
    break;
}
```

- **1. Invalid Input Validation:**

▸ **if (input.length() == 0 || input.endsWith(".") || isOperator(input[input.length() - 1]))**

• **input.length() == 0:** Prevents entering a decimal point when there's no prior input.

• **input.endsWith("."):** Prevents entering multiple decimal points in a row.

• **isOperator(input[input.length() - 1]):** Ensures a decimal point doesn't follow directly after an operator.

- **2. Current Number Check:**

```
int i = input.length() - 1;
while (i >= 0 && (isdigit(input[i]) || input[i] == '.')) {
    if (input[i] == '.') break;
    --i;
}
```

▸ **int i = input.length() - 1;:** A variable i is defined and initialized to point to the last character in the input string, so the scan can go from right to left.

▸ **while (i >= 0 && (isdigit(input[i]) || input[i] == '.')):** The loop continues scanning left as long as the index is valid (i >= 0) and the character is either a digit or a decimal point. This helps isolate the current number being typed (like "12.34" in a longer expression).

▸ **if (input[i] == '.') break;:** If a decimal point is found, the loop stops right away, indicating that the current number already includes a decimal point—so no more can be added.

▸ **--i;:** The index i is decremented to continue checking the characters to the left.

- **3. Final Decision:**

```
if (i >= 0 && input[i] == '.') break;
input += ".";
```

▸ **1. if (i >= 0 && input[i] == '.') break;**

• **i >= 0:** Ensures that the index is within valid bounds and prevents out-of-range errors.

• **input[i] == '.':** Checks if the current character is a decimal point.

• **break;**: If a decimal has already been found in the number, exit the loop.

▸ **2. input += ".";**

If no decimal point was found during the scan (meaning the loop wasn't broken), then a decimal point is safely appended to the input string.

Complete Code

Kindly click the link below to view the full code implementation.
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the above code, you will have built a simple calculator controlled by an infrared remote. It allows users to input numbers and operators using the remote, displays the expression in real time on the screen, and performs calculations including decimal support and basic arithmetic operations. Additional features include backlight control, input debounce handling, and validation for decimal point usage. The entire calculation process and result are clearly presented through a graphical user interface.

Programming Steps

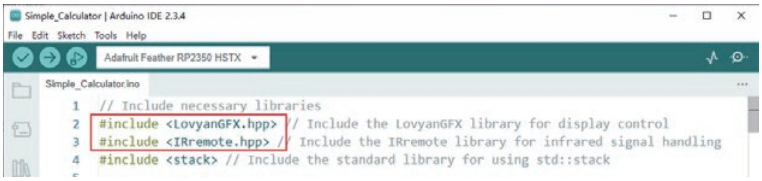
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use two additional library (**IRremote and LovyanGFX-develop**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

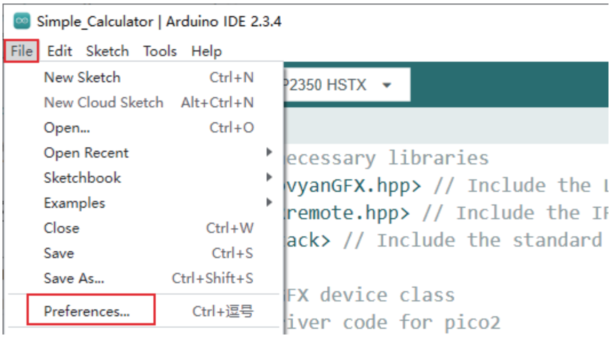
• Navigate to the **/example/libraries** directory and download the **IRremote and LovyanGFX-develop** folder.


Simple_Calculator | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 HSTX
Simple_Calculator.ino

```
// Include necessary libraries
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control
#include <IRremote.hpp> // Include the IRremote library for infrared signal handling
#include <stack> // Include the standard library for using std::stack
```

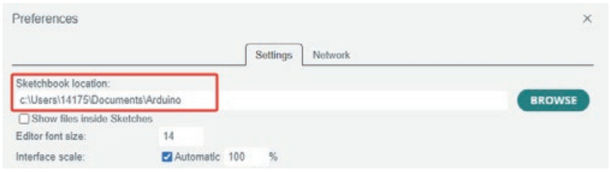
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



```
ecessary libraries
vyanGFX.hpp> // Include the L
remote.hpp> // Include the IF
ack> // Include the standard

FX device class
iver code for pico2
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



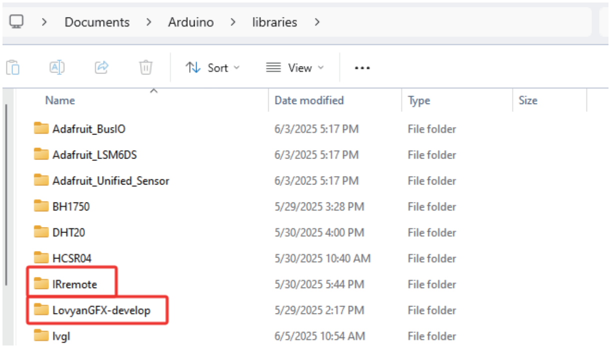
Sketchbook location:
c:\Users\14175\Documents\Arduino

☐ Show files inside Sketches

Editor font size: 14

Interface scale: ☑ Automatic 100 %

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
Adafruit_BusIO	6/3/2025 5:17 PM	File folder	
Adafruit_LSM6DS	6/3/2025 5:17 PM	File folder	
Adafruit_Unified_Sensor	6/3/2025 5:17 PM	File folder	
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
lvgl	6/5/2025 10:54 AM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

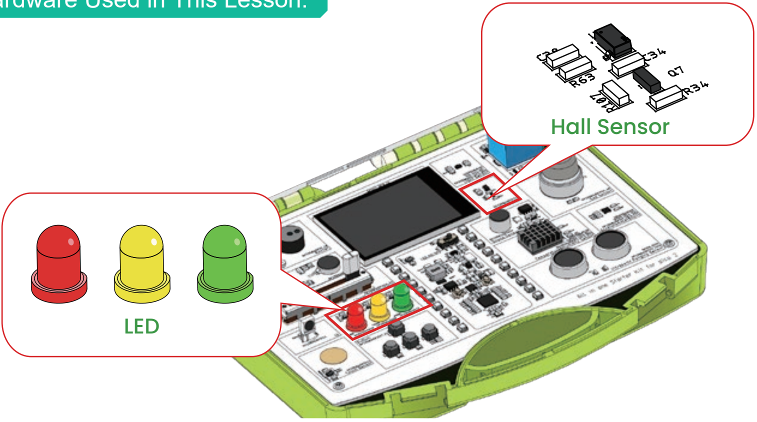
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board. Make sure the compilation completes without errors before running the program.

Lesson 20 - Hall Counter

Introduction

In this chapter, you'll learn the basic application of a Hall effect sensor. A magnet is used to trigger the sensor, enabling a counting function with the results displayed in real time on a TFT screen. This system is useful for detecting magnetic field changes and can be applied in scenarios such as access control counters and object detection.

Hardware Used in This Lesson:



Working Principle of Hall Sensor

A Hall sensor detects magnetic fields via the Hall effect. When current flows perpendicular to a magnetic field in a semiconductor plate, carriers deflect due to Lorentz force, generating a transverse potential difference (Hall voltage). This voltage, proportional to field strength, is amplified into analog/digital outputs. It measures field intensity, position, speed, etc., featuring non-contact operation and high reliability for motor control and position sensing.

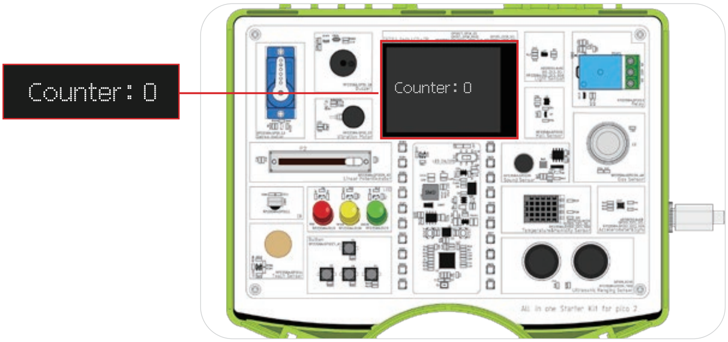
Working Principle of LED

At the heart of an LED (Light Emitting Diode) is a semiconductor PN junction. When a forward bias voltage is applied, electrons from the N-type region recombine with holes from the P-type region near the junction. During this recombination, electrons drop from a higher energy level to a lower one, releasing the excess energy in the form of photons—producing light. The color (or wavelength) of the emitted light is determined by the energy band gap of the semiconductor material. This process is a direct application of electroluminescence.

Operation Effect Diagram

1. Initial State:

After powering on, the TFT screen displays “Counter: 0,” and the system enters standby mode.

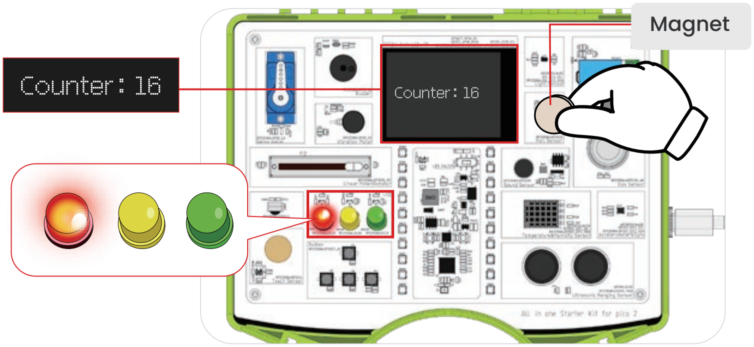


2. Count Trigger:

a. When a magnet is brought near the Hall sensor, it detects the change in magnetic field and outputs a high signal.

b. Upon receiving the signal, the microcontroller flashes a red LED once (turns it on and off), and the counter increments by 1.

c. The TFT screen updates in real time to display “Counter: X,” where X is the current



3. Repeating the Count Operation:

Each time the magnet is brought close to and then removed from the sensor, the counter increments following the same process (e.g., 1 → 2 → 3...), with the LED flashing and the display updating for every count.count.

Key Explanations

1. Display Initialization and Interface Update Function

```
void updateCounterDisplay(int count) {
  gfx.fillScreen(TFT_BLACK);
  gfx.setCursor(60, 100, 4);
  gfx.setTextSize(1);
  gfx.setTextColor(TFT_WHITE);
  gfx.printf("counter: %d", count);
}
```

- **gfx.fillScreen(TFT_BLACK);** This command fills the entire screen with black using the fillScreen() method. TFT_BLACK is a predefined color constant in the LovyanGFX library. This ensures that each screen update starts with a clean background.
- **setCursor(x, y, font):** This method sets the starting position and font for text display.

  • 60 is the X coordinate (horizontal offset from the left edge),

  • 100 is the Y coordinate (vertical offset from the top),

  • 4 is the font index (an optional parameter that likely specifies a particular font style).
- **gfx.setTextSize(1);** Sets the text scaling factor. A value of 1 keeps the text at its original size. Larger values scale the text proportionally.
- **gfx.setTextColor(TFT_WHITE);** Sets the text color to white. TFT_WHITE is a predefined white color constant in the LovyanGFX library.
- **gfx.printf("counter: %d", count);** Uses the printf() function to format and print the text on screen. It displays "counter: " followed by the value of count. The %d is a placeholder that inserts the integer value of count in decimal format.

2. Hall Sensor Detection Logic

```
if (digitalRead(hall) == LOW) {
  delay(50);
  if (digitalRead(hall) == LOW) {
    while (digitalRead(hall) == LOW);
    count++;
    updateCounterDisplay(count);
    digitalWrite(redLed, HIGH);
    delay(200);
    digitalWrite(redLed, LOW);
  }
}
```

- **Use a two-step digitalRead() check combined with a 50ms delay to implement debounce handling.**
- **while (digitalRead(hall) == LOW);** Wait until the magnet moves away to prevent counting a single trigger multiple times.
- **When a magnet is successfully detected:**

  • Increment the count (count++).

  • Call updateCounterDisplay() to refresh the screen.

  • Briefly flash the red LED to indicate a successful trigger.
- This process is crucial to ensure the Hall sensor triggers accurately, resists interference, and avoids duplicate counts.

3. Backlight Control and Display Initialization

```
gfx.init();
pinMode(0, OUTPUT);
digitalWrite(0, HIGH);
```

- **gfx.init():** This is the core call to initialize the LovyanGFX display system. It performs several key steps: first, it establishes SPI communication with the ST7789 display controller at an 80MHz clock speed, configures the display parameters for a 240x320 resolution, initializes the FT5x06 touch controller via I2C (address 0x38), and finally allocates memory for the display buffer. Notably, this function automatically sets up the hardware interfaces based on the SPI pin configuration defined earlier in the LGFX class (SCLK=6, MOSI=7, DC=16, etc.), and sends the factory default initialization command sequence to the screen to ensure proper color mode (RGB565) and scan direction.
- **pinMode(0, OUTPUT):** sets GPIO0 as a digital output to control the display backlight circuit. In the hardware design, this pin is usually connected to the gate of an N-MOSFET (such as AO3400). By controlling the MOSFET's conduction, it switches the power supply to the backlight LED. Setting the pin to OUTPUT mode configures the GPIO as a push-pull output, capable of driving up to 12mA according to the RP2040 chip specs, sufficient to directly drive a small MOSFET. It's important to note that pin 0 here refers to RP2040's GPIO0, not the physical pin number on the board.
- **digitalWrite(0, HIGH):** outputs a 3.3V high signal to GPIO0, activating the backlight circuit. When set high, the connected MOSFET fully conducts, allowing the backlight LED's anode to receive power (typically 5V supply). This action triggers two key effects: first, it enables the backlight boost circuit (if the screen uses a constant-current driver), and second, it sets the PWM duty cycle to 100%, achieving maximum brightness.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the code above, you have implemented a Hall effect sensor counting system that updates the count in real time on the TFT screen using the format "counter: X". When a magnet approaches the Hall sensor, the system triggers a count, while the red LED flashes for 200ms as visual feedback. The design employs dual detection and delayed debounce mechanisms to ensure counting is both accurate and reliable.

Programming Steps

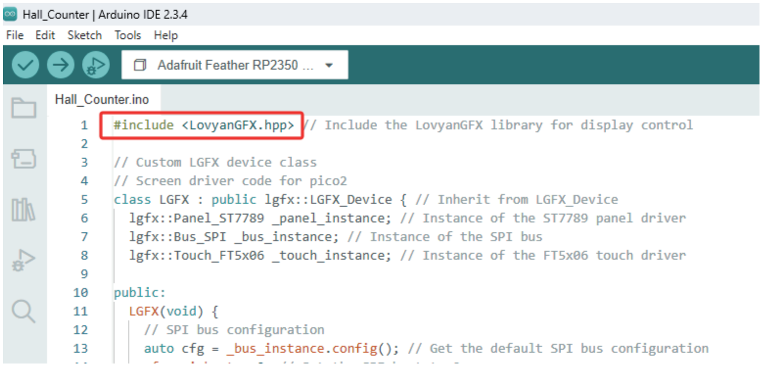
Note: For detailed programming steps, you can refer to the programming process in the first lesson.

In this lesson, we use an additional library (**LovyanGFX-develop**), so it's important to include it before running the code to avoid compilation errors.

1. Download the Library

• Click the link below:
https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2

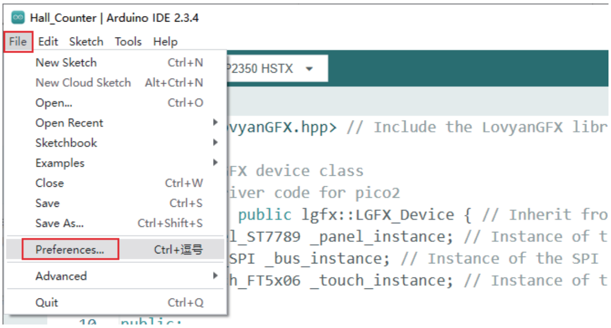
• Navigate to the **/example/libraries** directory and download the **LovyanGFX-develop** folder


Hall_Counter | Arduino IDE 2.3.4
File Edit Sketch Tools Help
Adafruit Feather RP2350 ...
Hall_Counter.ino

```
#include <LovyanGFX.hpp> // Include the LovyanGFX library for display control

// Custom LGFX device class
// Screen driver code for pico2
class LGFX : public lgfx::LGFX_Device { // Inherit from LGFX_Device
  lgfx::Panel_ST7789 _panel_instance; // Instance of the ST7789 panel driver
  lgfx::Bus_SPI _bus_instance; // Instance of the SPI bus
  lgfx::Touch_FT5x06 _touch_instance; // Instance of the FT5x06 touch driver

public:
  LGFX(void) {
    // SPI bus configuration
    auto cfg = _bus_instance.config(); // Get the default SPI bus configuration
```

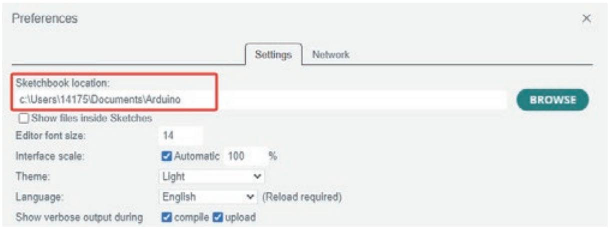
2. Add the Library to the Arduino Environment

• In the Arduino IDE, click on the **File** menu and select **Preferences**.



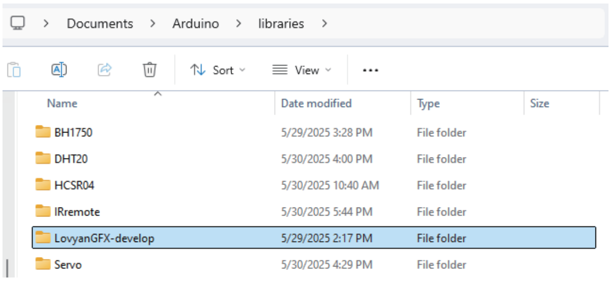
```
vyanGFX.hpp> // Include the LovyanGFX libr

FX device class
iver code for pico2
 public lgfx::LGFX_Device { // Inherit fro
l_ST7789 _panel_instance; // Instance of t
SPI _bus_instance; // Instance of the SPI
h_FT5x06 _touch_instance; // Instance of t
```

• Open the corresponding folder on your computer. Inside, you'll find a folder named libraries – this is where Arduino stores third-party libraries.



Sketchbook location:
c:\Users\14175\Documents\Arduino
☐ Show files inside Sketches
Editor font size: 14
Interface scale: ☑ Automatic 100 %
Theme: Light
Language: English (Reload required)
Show verbose output during ☑ compile ☑ upload

• Copy or move the downloaded library folder directly into the libraries directory. Once done, the Arduino IDE will automatically recognize and be able to use this library.



Name	Date modified	Type	Size
BH1750	5/29/2025 3:28 PM	File folder	
DHT20	5/30/2025 4:00 PM	File folder	
HCSR04	5/30/2025 10:40 AM	File folder	
IRremote	5/30/2025 5:44 PM	File folder	
LovyanGFX-develop	5/29/2025 2:17 PM	File folder	
Servo	5/30/2025 4:29 PM	File folder	

3. Upload and Run the Code(You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.)

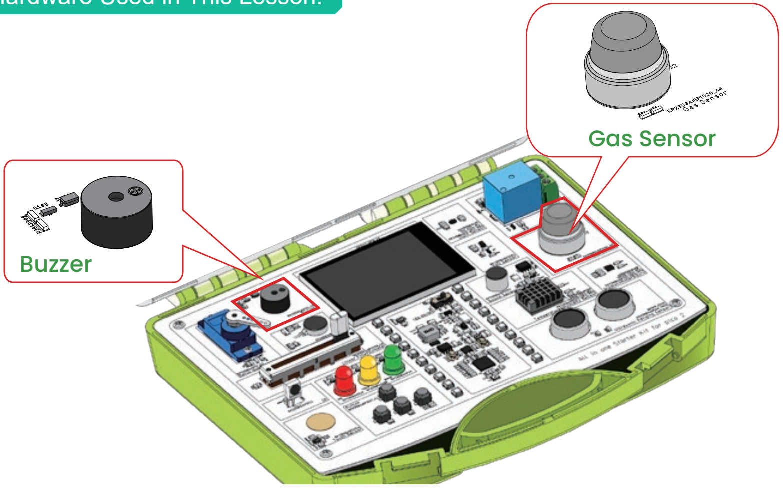
After placing the library correctly, follow the upload steps from Lesson 1 to compile and upload the code to your board.Make sure the compilation completes without errors before running the program.

Lesson 21 - Smoke Alarm

Introduction

In this chapter, you'll learn the principles and implementation of an alarm system based on the MQ2 smoke sensor. The system triggers a buzzer when it detects a certain level of smoke in the environment. This lesson focuses on the basics of sensor signal acquisition and alarm logic.

Hardware Used in This Lesson:



Working Principle of Gas Sensor

A gas sensor detects target gases via reactions that alter the electrical properties (eg: resistance/capacitance/current) of sensing materials. MOS types rely on resistance changes from gas adsorption; electrochemical sensors generate current via redox reactions; infrared models measure light absorption at specific wavelengths. Signal processing converts variations into concentration readings, offering high selectivity and rapid response.

Working Principle of Buzzer Operation

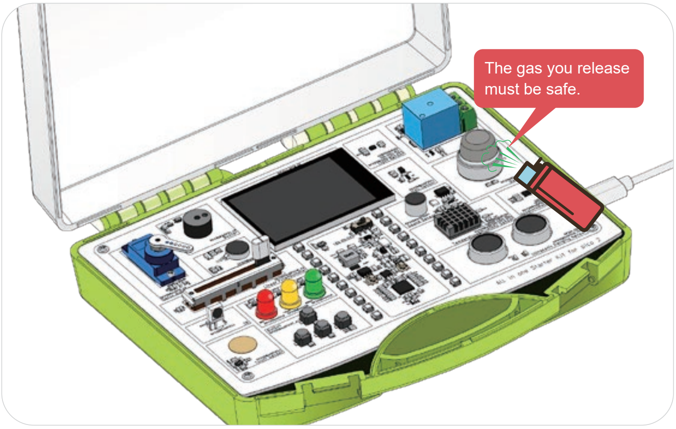
The buzzer generates sound by vibrating a diaphragm driven by an electrical signal. When an alternating current is applied, the diaphragm vibrates rapidly due to magnetic or piezoelectric effects, producing sound. The pitch and frequency are determined by the current frequency, and it is commonly used for alerts or alarms.

Operation Effect Diagram

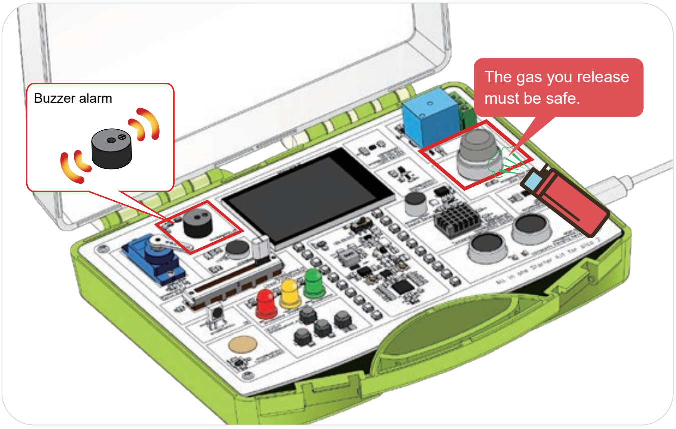
1. Prepare a safe smoke source (take extreme caution to avoid any fire hazard during the process).

(In this demonstration, I used a lighter and extinguished the flame, allowing it to emit only carbon monoxide.)

2. Slowly move the smoke source closer to the sensing head of the MQ-2 sensor.



3. Observe the system’s response: when the smoke concentration exceeds the preset threshold, the buzzer should sound continuously as an alarm.



Key Explanations

1. Analog Pin Reading and Voltage Conversion

```
float sensorValue = analogRead(gas_pin);
float sensor_volt = sensorValue / 1023.0 * 5.0;
```

- The analogRead() function returns an integer between 0 and 1023, corresponding to an input voltage range from 0 to 5 volts.
- To determine the actual voltage, the raw value is converted using the formula: sensorValue / 1023.0 * 5.0. This voltage is then used to estimate the gas concentration.
- When detecting gases such as smoke, methane, or propane, the MQ-2 sensor's analog output voltage increases proportionally with the concentration of gas.
- **Significance:** This is the core data-processing step for smoke detection—it determines whether the system should trigger an alarm.

2. Smoke Voltage Threshold Detection and Buzzer Alarm

```
if (sensor_volt > 1.0) {
  tone(buzzerPin, 1300);
} else {
  noTone(buzzerPin);
}
```

- When the MQ-2 sensor's output voltage exceeds 1.0V, it indicates the presence of a certain concentration of smoke or gas in the air.
- tone(pin, freq) sends a square wave of the specified frequency—1300Hz in this case—to the buzzerPin, producing an audible alarm.
- noTone(pin) stops the sound output from the buzzer.
- Significance: This forms the core logic of the alarm response—using a simple voltage threshold to deliver real-time audio warnings.

3. Serial Output for Monitoring and Debugging

```
Serial.print("Voltage: ");
Serial.print(sensor_volt);
Serial.println(" V");
```

- The current voltage reading is sent to the serial port, making it easy to debug or observe real-time data using a serial monitor tool.
- Based on the test results, the alarm threshold can be fine-tuned dynamically for better accuracy.
- **Significance:** This feature is essential for debugging and calibrating the sensor, especially useful during the development and testing phase.

Complete Code

Kindly click the link below to view the full code implementation.

https://github.com/Elecrow-RD/All-in-one-Starter-Kit-for-Pico-2/tree/master/example/pico_arduino_code

After studying the above code, you've built a basic alarm system using an MQ2 smoke sensor. It reads the analog voltage from the sensor and compares it to a 1.0V threshold. If the voltage exceeds this value—indicating the presence of smoke—the system triggers a 1300Hz alarm sound through the buzzer. If not, the buzzer remains silent. Meanwhile, the current voltage is continuously output via the serial port for monitoring and debugging purposes.

Programming Steps

You can refer to the flashing steps from Lesson 1 (page 10-15) as a guide.

ELECROW

MAKE YOUR MAKING EASIER