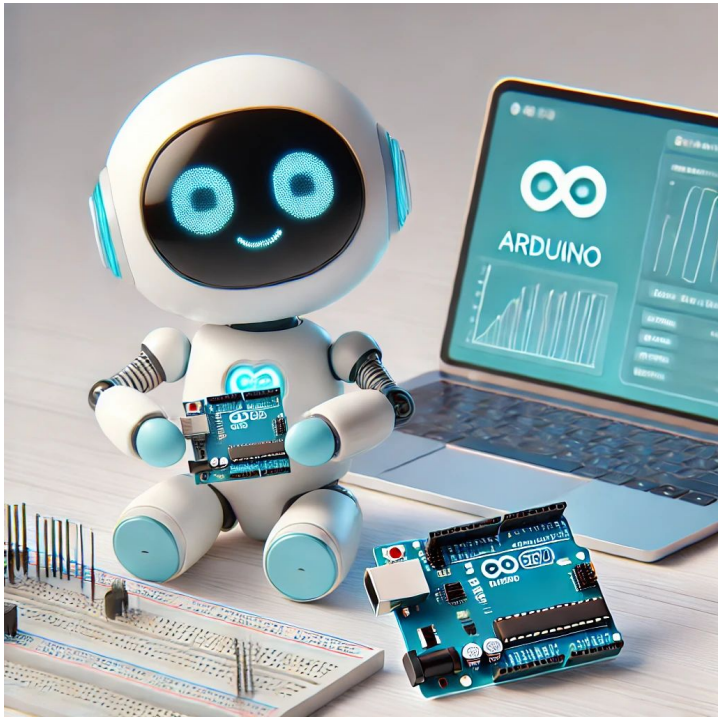
4WD Car

7/10/2025

MAYFLOWER
A.S.T.E.M.
ACADEMY



Arduino

Arduino is an open-source platform that uses a programming language based on C++, which makes it easy to use for beginners and professionals. Here, we will introduce the basics of the Arduino programming language, including its features such as a large library of pre-written code, the simplicity of its syntax, and its versatility in a wide variety of projects. We will also discuss the limitations of the language. Hence, this should help anyone interested in learning how to use the Arduino platform to build innovative and fun projects.

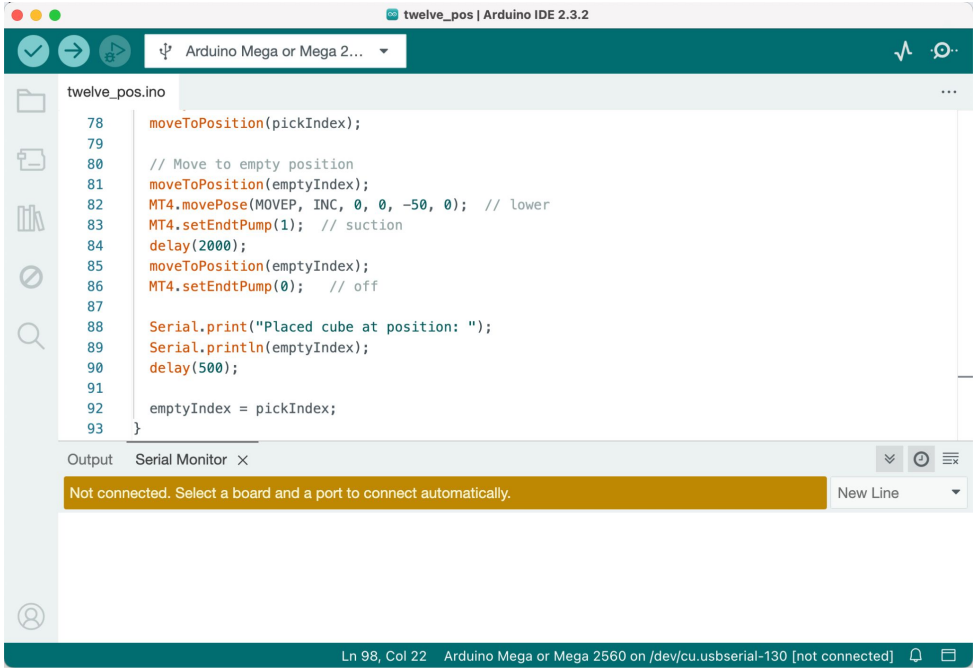
Arduino Programming Language

The Arduino programming language is used to program microcontroller boards such as the Arduino Uno to interact with sensors, actuators, and other devices connected to the board. In fact, the language is based on C++, and it is designed to be easy to use for beginners and non-programmers. Additionally, it is commonly used in projects involving robotics, home automation, and Internet of Things (IoT) applications.

Features

- **Open Source:** Software and Hardware: Both are open source.
- **Based on C++:** Programming Language: Uses a C++-based language.
- **Widely Known:** Leverages the popularity and familiarity of C++.
- **Arduino Library:** Extensive library for common tasks.
- **Support for PWM** (Pulse Width Modulation)

Arduino IDE

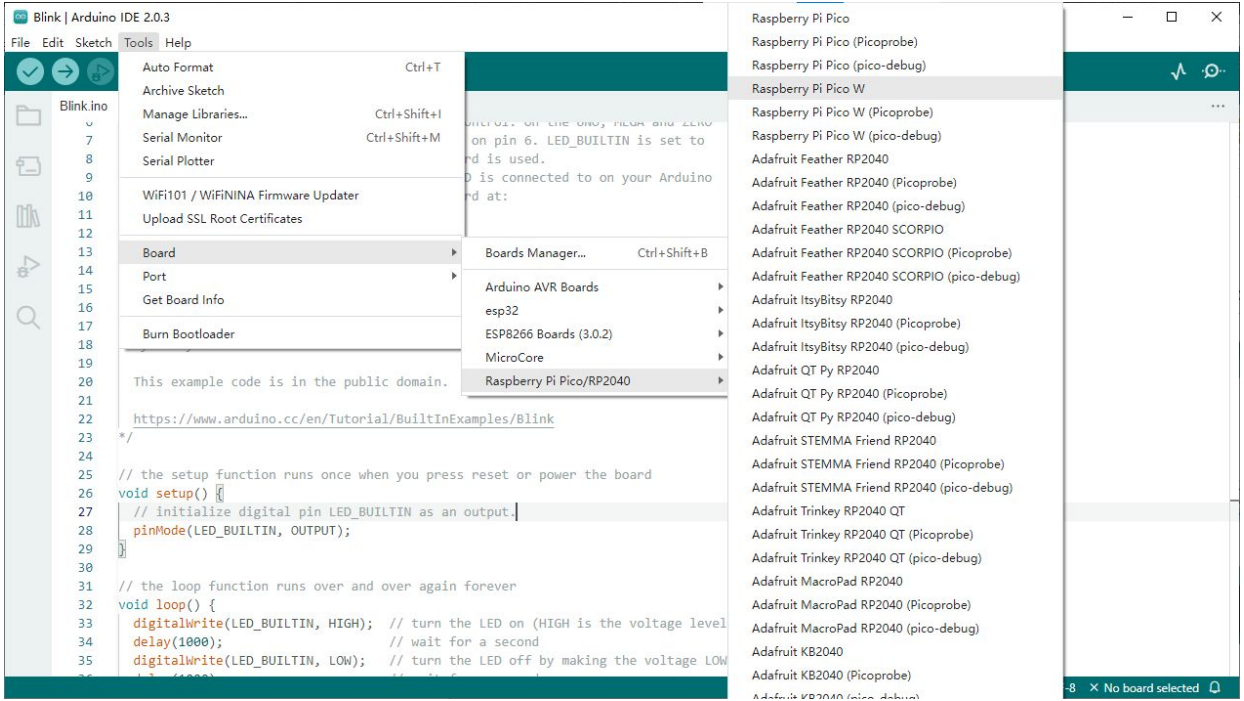


1. Toolbar
2. Sidebar
3. Text editor
4. Console controls
5. Text console

Upload to Board

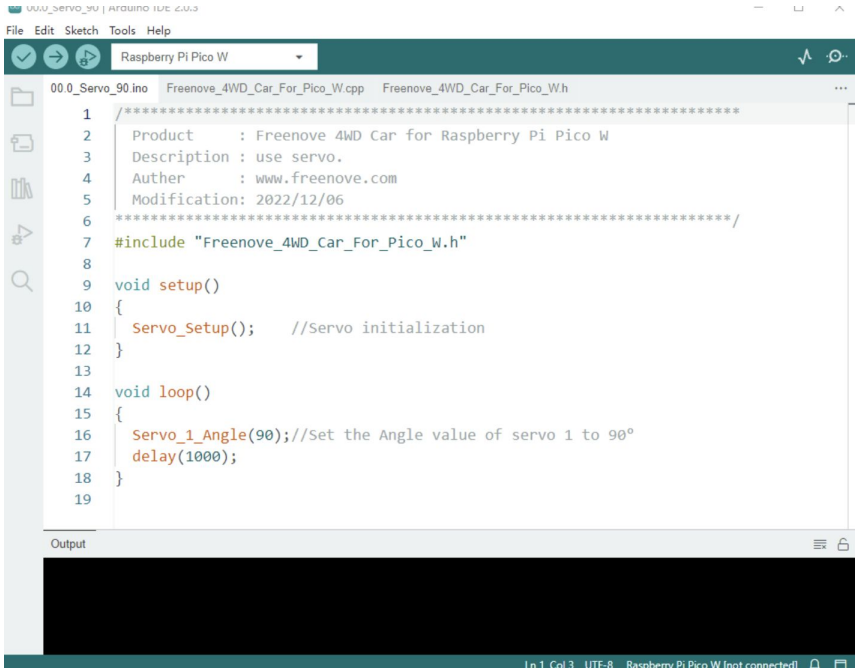
1. Connect board to computer

2. Select the board & port



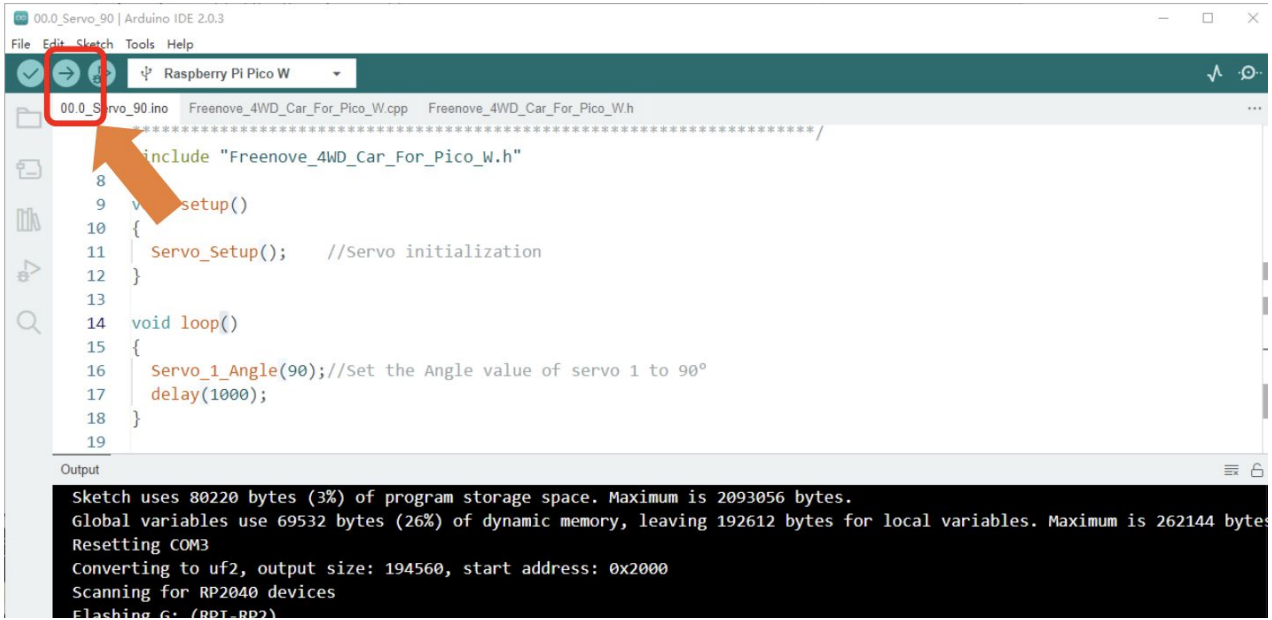
Upload to Board

3. Write sketch



Upload to Board

4. Upload to board



2 Arduino Basics

Structure

Method & Parameter	Description
`void loop()`	Main function for continuous code execution.
`void setup()`	Initialization function, called once at startup.

Control Structure

Method & Parameter	Description
`break`	Exits a loop or switch statement.
`continue`	Skips the rest of a loop iteration.
`do...while`	Executes a block of code repeatedly while a specified condition is true.
`else`	Part of the if-else statement.
`for`	Creates a loop with a specified initialization, condition, and increment.
`goto`	Transfers control to a labeled statement.
`if`	Conditional statement for decision-making.
`return`	Exits a function and optionally returns a value.
`switch...case`	Multi-way branch statement.
`while`	Creates a loop with a specified condition.

Further Syntax

Method & Parameter	Description
`#define (define)`	Macro definition for code substitution.
`#include (include)`	Includes a file in the source code.
`/* */ (block comment)`	Block comment for multiple lines.
`// (single line comment)`	Single line comment.
`; (semicolon)`	Statement terminator.
`{} (curly braces)`	Block of code, often used with control structures.

3

UART

UART

Universal Asynchronous Receiver-Transmitter (UART) is a serial communication protocol that can be used to send data between an Arduino board and other devices. This is the protocol used when you send data from an Arduino to your computer, using the classic Serial.print() method.

UART is one of the most used device-to-device (serial) communication protocols. It’s the protocol used by Arduino boards to communicate with the computer. It allows an asynchronous serial communication in which the data format and transmission speed are configurable.

Serial Class

With the Serial class, you can send / receive data to and from your computer over USB, or to a device connected via the Arduino's RX/TX pins.

- When sending data over USB, we use Serial. This data can be viewed in the Serial Monitor in the Arduino IDE.
- When sending data over RX/TX pins, we use Serial1.

Serial Class

The Serial class have several methods with some of the essentials being:

- begin() - begins serial communication, with a specified baud rate (many examples use either 9600 or 115200).
- print() - prints the content to the Serial Monitor.
- println() - prints the content to the Serial Monitor, and adds a new line.
- available() - checks if serial data is available (if you send a command from the Serial Monitor).
- read() - reads data from the serial port.
- write() - writes data to the serial port.

Serial

For example, to initialize serial communication on both serial ports, we would write it as:

```
Serial.begin(9600); //init communication over USB

Serial1.begin(9600); //communication over RX/TX pins
```

UART

UART frame format

- UART frames consist of:
  - Start / stop bits
  - Data bits
  - Parity bit (optional)
- High voltage ("mark") = 1, low voltage ("space") = 0

https://www.youtube.com/watch?v=sTHckUyxwp8

Basic Print Example

```
void setup() {
    Serial.begin(9600);
}

void loop() {
    Serial.println("Hello world!");
    delay(1000);
}
```

Basic Print Example

```
int incomingByte = 0; // for incoming serial data

void setup() {
    Serial.begin(9600); //initialize serial communication at a 9600 baud rate
}

void loop() {
    // send data only when you receive data:
    if (Serial.available() > 0) {
         // read the incoming byte:
         incomingByte = Serial.read();
         // say what you got:
         Serial.print("I received: ");
         Serial.println(incomingByte, DEC);
    }
}
```

4
Data and Operations

Data Types

Method & Parameter	Description
`array`	Collection of variables of the same type.
`bool`	Boolean data type.
`boolean`	Boolean data type (synonym for bool).
`byte`	8-bit unsigned data type.
`char`	8-bit character data type.
`double`	Double-precision floating-point data type.
`float`	Single-precision floating-point data type.

Data Types

Method & Parameter	Description
`int`	Integer data type.
`long`	Long integer data type.
`short`	Short integer data type.
`size_t`	Unsigned integer data type.
`string`	Sequence of characters (not a primitive type).
`String()`	String class in Arduino.
`unsigned char`	Unsigned 8-bit character data type.

Conversion

Method & Parameter	Description
`(unsigned int)`	Type casting to unsigned int.
`(unsigned long)`	Type casting to unsigned long.
`byte()`	Type casting to byte.
`char()`	Type casting to char.

Conversion

Method & Parameter	Description
`float()`	Type casting to float.
`int()`	Type casting to int.
`long()`	Type casting to long.
`word()`	Type casting to word.

Arithmetic Operators

Method & Parameter	Description
`% (remainder)`	Modulo operator for finding the remainder of a division.
`* (multiplication)`	Multiplication operator.
`+ (addition)`	Addition operator.
`- (subtraction)`	Subtraction operator.
`/ (division)`	Division operator.
`= (assignment operator)`	Assignment operator.

Comparison Operators

Method & Parameter	Description
`!= (not equal to)`	Checks if two values are not equal.
`< (less than)`	Checks if the left value is less than the right value.
`<= (less than or equal to)`	Checks if the left value is less than or equal to the right value.
`== (equal to)`	Checks if two values are equal.
`> (greater than)`	Checks if the left value is greater than the right value.
`>= (greater than or equal to)`	Checks if the left value is greater than or equal to the right value.

Boolean Operators

Method & Parameter	Description
`! (logical not)`	Inverts the logical value, true becomes false and vice versa.
`&& (logical and)`	Logical AND operator, returns true if both operands are true.
`|| (logical or)`	Logical OR operator, returns true if at least one operand is true.

Bitwise Operators

Method & Parameter	Description
`& (bitwise and)`	Performs bitwise AND operation.
`<< (bitshift left)`	Shifts bits to the left.
`>> (bitshift right)`	Shifts bits to the right.
`^ (bitwise xor)`	Performs bitwise XOR (exclusive OR) operation.
`| (bitwise or)`	Performs bitwise OR operation.
`~ (bitwise not)`	Inverts all bits.

Compound Operators

Method & Parameter	Description
`%= (compound remainder)`	Performs a modulo operation and assigns the result to the left operand.
`&= (compound bitwise and)`	Performs a bitwise AND operation and assigns the result to the left operand.
`*= (compound multiplication)`	Multiplies the left operand by the right operand and assigns the result to the left operand.
`++ (increment)`	Increments the value of the operand by 1.
`+= (compound addition)`	Adds the right operand to the left operand and assigns the result to the left operand.

Compound Operators

Method & Parameter	Description
`-- (decrement)`	Decrements the value of the operand by 1.
`-= (compound subtraction)`	Subtracts the right operand from the left operand and assigns the result to the left operand.
`/= (compound division)`	Divides the left operand by the right operand and assigns the result to the left operand.
`^= (compound bitwise xor)`	Performs a bitwise XOR operation and assigns the result to the left operand.
`|= (compound bitwise or)`	Performs a bitwise OR operation and assigns the result to the left operand.

5

Arduino API

Digital I/O

Method & Parameters	Description	Returns
`int digitalRead(int pin)`	Reads the state of a digital pin.	`int`
`void digitalWrite(int pin, int state)`	Writes a state to a digital pin.	Nothing
`void pinMode(int pin, int mode)*`	Define the mode of a pin.	Nothing

Analog I/O

Method & Parameters	Description	Returns
`int analogRead(int pin)`	Reads the value of an analog pin in a 10-bit resolution (0-1023).*	`int`
`void analogReadResolution(int resolution)`	Sets ADC read resolution in bits.	Nothing
`void analogReference(int reference)`	Changes the voltage reference for a board.**	Nothing
`void analogWrite(int pin, int value)`	Writes a value to a PWM supported pin in a 8-bit resolution (0-255).**	Nothing
`void analogWriteResolution(int resolution)`	Sets write resolution for a board.	Nothing

Time

Method & Parameters	Description	Returns
`void delay(long milliseconds)`	Freezes program execution for specified number of **milliseconds**.	Nothing
`void delayMicroseconds(int microseconds)`	Freezes program execution for specified number of **microseconds**.	Nothing
`long millis()`	Returns **milliseconds** passed since program start.	`long`
`long micros()`	Returns **microseconds** passed since program start.	`long`

Math

Method & Parameters	Description	Returns
`int abs(int value)`	Calculates the absolute value of a number.	`int`
`int constrain(int value, int min, int max)`	Constrains a number to be within a range.	`int`
`long map(long val, long min, long max, long newMin, long newMax)`	Re-maps a number from one range to another.	`long`
`double sqrt(double value)`	Calculates the square root of a number.	`double`

Math

Method & Parameters	Description	Returns
`int max(int val1, int val2)`	Returns the greater of two values.	`int`
`int min(int val1, int val2)`	Returns the smaller of two values.	`int`
`double pow(double base, double exponent)`	Raises a base to the power of an exponent.	`double`
`int sq(int value)`	Calculates the square of a number.	`int`

Trigonometry

Method & Parameters	Description	Returns
`cos(double angle)`	Calculates the cosine of an angle in radians.	`double`
`sin(double angle)`	Calculates the sine of an angle in radians.	`double`
`tan(double angle)`	Calculates the tangent of an angle in radians.	`double`

Characters

Method & Parameters	Description	Returns
`boolean isAlpha(char c)`	Checks if the character is an alphabetic character.	`boolean`
`boolean isAlphaNumeric(char c)`	Checks if the character is an alphanumeric character.	`boolean`
`boolean isAscii(char c)`	Checks if the character is a 7-bit ASCII character.	`boolean`
`boolean isControl(char c)`	Checks if the character is a control character.	`boolean`
`boolean isDigit(char c)`	Checks if the character is a digit (0-9).	`boolean`

Characters

Method & Parameters	Description	Returns
`boolean isGraph(char c)`	Checks if the character is a printable character, excluding space.	`boolean`
`boolean isHexadecimalDigit(char c)`	Checks if the character is a hexadecimal digit (0-9, A-F, a-f).	`boolean`
`boolean isLowerCase(char c)`	Checks if the character is a lowercase alphabetic character.	`boolean`
`boolean isPrintable(char c)`	Checks if the character is a printable character, including space.	`boolean`
`boolean isPunct(char c)`	Checks if the character is a punctuation character.	`boolean`

Characters

Method & Parameters	Description	Returns
`boolean isSpace(char c)`	Checks if the character is a whitespace character.	`boolean`
`boolean isUpperCase(char c)`	Checks if the character is an uppercase alphabetic character.	`boolean`
`boolean isWhitespace(char c)`	Checks if the character is a whitespace character according to `isSpaceChar()` method.	`boolean`

Random Numbers

Method & Parameters	Description	Returns
`int random()`	Generates a pseudo-random number between 0 and `RAND_MAX`.	`int`
`void randomSeed(unsigned long seed)`	Seeds the random number generator.	Nothing

Variables

Enum Type	Enumeration	Description
PinStatus	HIGH / LOW	Logical HIGH and LOW values (1 and 0).
PinMode	INPUT / OUTPUT / INPUT_PULLUP / INPUT_PULLDOWN / OUTPUT_OPENDRAIN	Constants for specifying pin modes (0, 1, 2, 3, 4).
	true / false	Boolean constants for true and false (1 and 0).

6

Using Variables

Using Variables

A variable is a place to store a piece of data. It has a name, a value, and a type. For example, this statement (called a *declaration*):

```
int pin = 13;
```

creates a variable whose name is pin, whose value is 13, and whose type is int. Later on in the program, you can refer to this variable by its name, at which point its value will be looked up and used. For example, in this statement:

```
pinMode(pin, OUTPUT);
```

it is the value of pin (13) that will be passed to the pinMode() function. In this case, you don't actually need to use a variable, this statement would work just as well:

```
pinMode(13, OUTPUT);
```

Variables

The advantage of a variable in this case is that you only need to specify the actual number of the pin once, but you can use it lots of times. So if you later decide to change from pin 13 to pin 12, you only need to change one spot in the code. Also, you can use a descriptive name to make the significance of the variable clear (e.g. a program controlling an RGB LED might have variables called redPin, greenPin, and bluePin).

A variable has other advantages over a value like a number. Most importantly, you can change the value of a variable using an assignment (indicated by an equals sign).

Variables

For example:

```
pin = 12;
```

will change the value of the variable to 12. Notice that we don't specify the type of the variable: it's not changed by the assignment. That is, the name of the variable is permanently associated with a type; only its value changes. [1] Note that you have to declare a variable before you can assign a value to it. If you include the preceding statement in a program without the first statement above, you'll get a message like: "error: pin was not declared in this scope".

When you assign one variable to another, you're making a copy of its value and storing that copy in the location in memory associated with the other variable. Changing one has no effect on the other. For example, after:

```
int pin = 13;

int pin2 = pin;

pin = 12;
```

Scope

It refers to the part of your program in which the variable can be used. This is determined by where you declare it. For example, if you want to be able to use a variable anywhere in your program, you can declare at the top of your code. This is called a global variable; here's an example:

```
int pin = 13;
void setup() {
    pinMode(pin, OUTPUT);
}

void loop() {
    digitalWrite(pin, HIGH);
}
```

Scope

As you can see, pin is used in both the setup() and loop() functions. Both functions are referring to the same variable, so that changing it one will affect the value it has in the other, as in:

```
int pin = 13;
void setup() {
    pin = 12;
    pinMode(pin, OUTPUT);
}

void loop() {
    digitalWrite(pin, HIGH);
}
```

Scope

If you only need to use a variable in a single function, you can declare it there, in which case its scope will be limited to that function. For example:

```
void setup() {
    int pin = 13;
    pinMode(pin, OUTPUT);
    digitalWrite(pin, HIGH);
}
```

In this case, the variable pin can only be used inside the setup() function. If you try to do something like this:

```
void loop() {
    digitalWrite(pin, LOW);// wrong: pin is not in scope here.
}
```

7

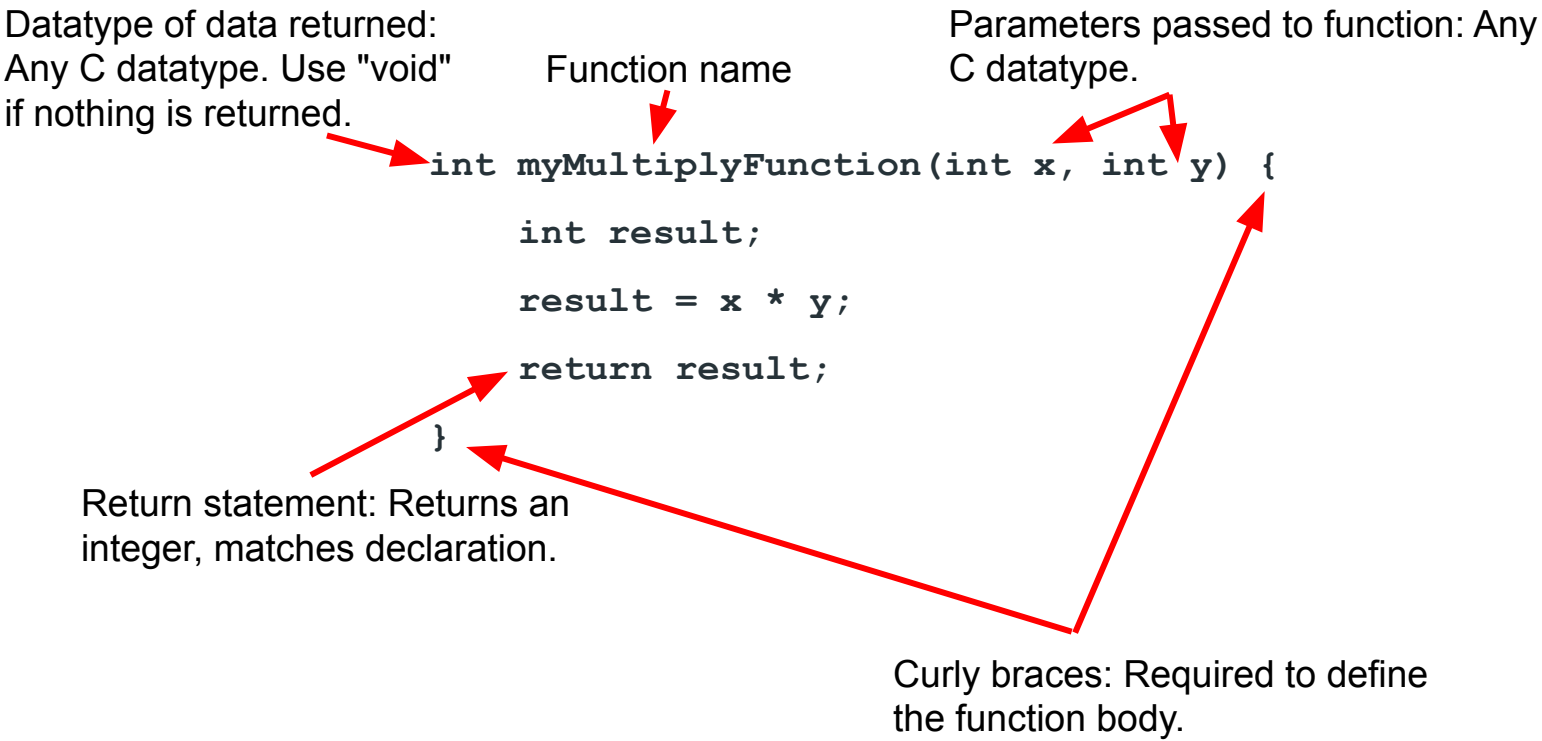
Functions

Code Segmenting

Standardizing code fragments into functions has several advantages

- Functions help the programmer stay organized. Often this helps to conceptualize the program.
- Functions codify one action in one place so that the function only has to be thought out and debugged once.
- This also reduces chances for errors in modification, if the code needs to be changed.

Function Example



Call a Function

To "call" our simple multiply function, we pass it parameters of the datatype that it is expecting:

```
void loop(){
 int i = 2;
 int j = 3;
 int k;
 k = myMultiplyFunction(i, j); // k now contains 6
}
```

The function needs to be declared outside any other function, so "myMultiplyFunction()" can go either above or below the "loop()" function.

Useful Functions

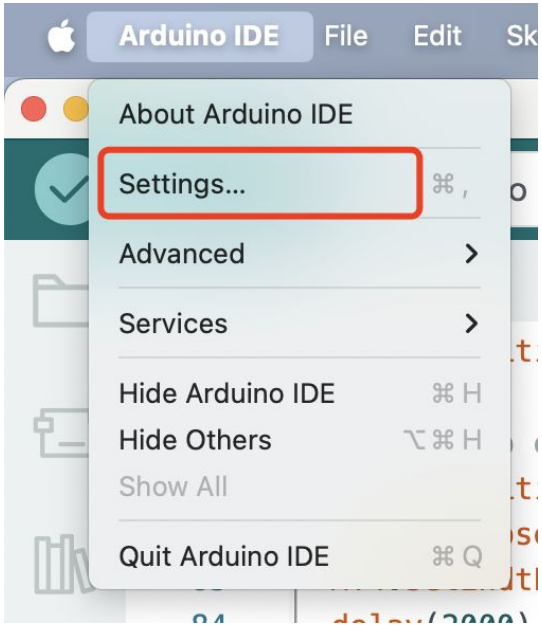
1. #include <Freenove.h>
2. Motor_Setup() //setup the motor
3. Motor_Move(int left_speed, int right_speed)
4. Tracking:
    a. pinMode(PIN_TRACKING_LEFT, INPUT);
    b. pinMode(PIN_TRACKING_CENTER, INPUT);
    c. pinMode(PIN_TRACKING_RIGHT, INPUT);
    d. int left = digitalRead(PIN_TRACKING_LEFT);
    e. int center = digitalRead(PIN_TRACKING_CENTER);
    f. int right = digitalRead(PIN_TRACKING_RIGHT);

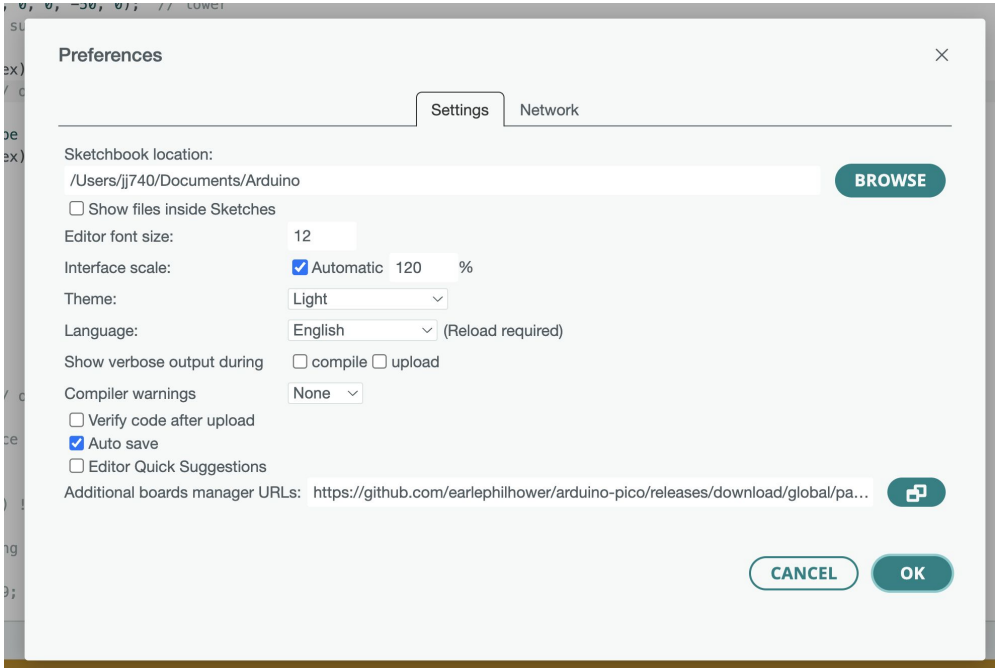
```
void loop() {
 LEFT, CENTER, RIGHT = READING
 if CENTER:
   if LEFT:
     if RIGHT:
       STOP
     else: // NOT RIGHT
       LEFT_TURN
   else: // NOT LEFT
     if RIGHT:
       RIGHT_TURN
     else: // NOT RIGHT
       FORWARD
  else: // NOT CENTER
   if LEFT:
     OFFTRACK_LEFT_ADJUSTMENT
   if RIGHT:
     OFFTRACK_RIGHT_ADJUSTMENT
   if NOT LEFT and NOT RIGHT:
     KEEP PREVIOUS MOVEMENT
}
```

Lab

Environment Configuration

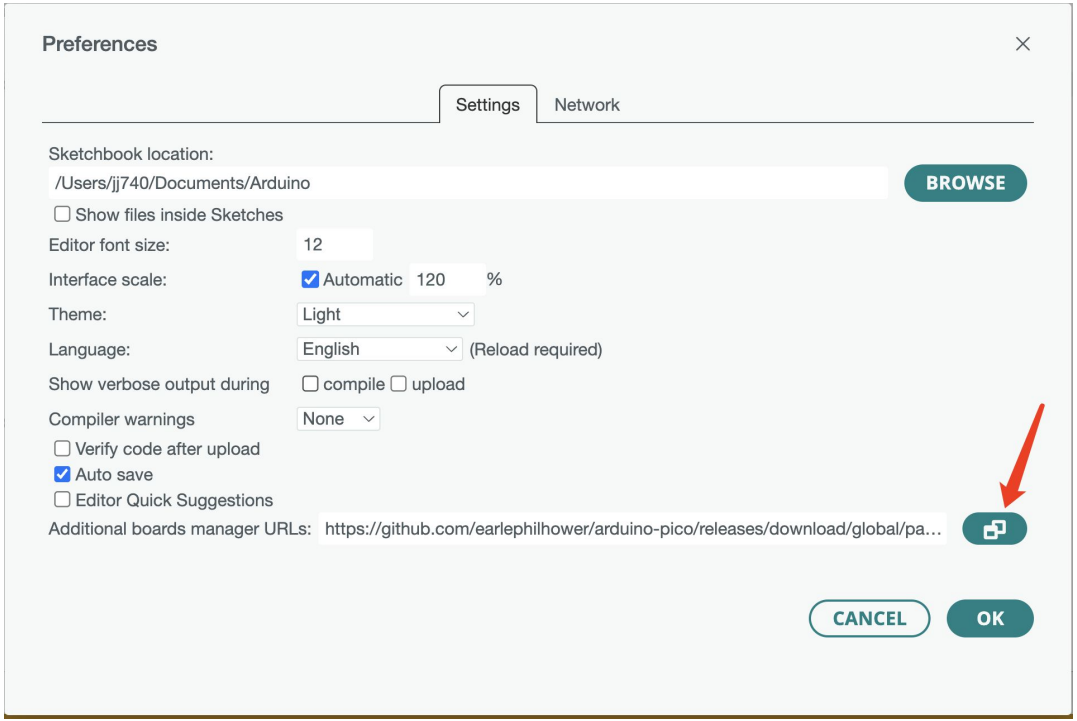
Open Arduino and click **File** in Menus and select **Preferences**





Environment Configuration

"Additional Boards Manager URLs"



Environment Configuration

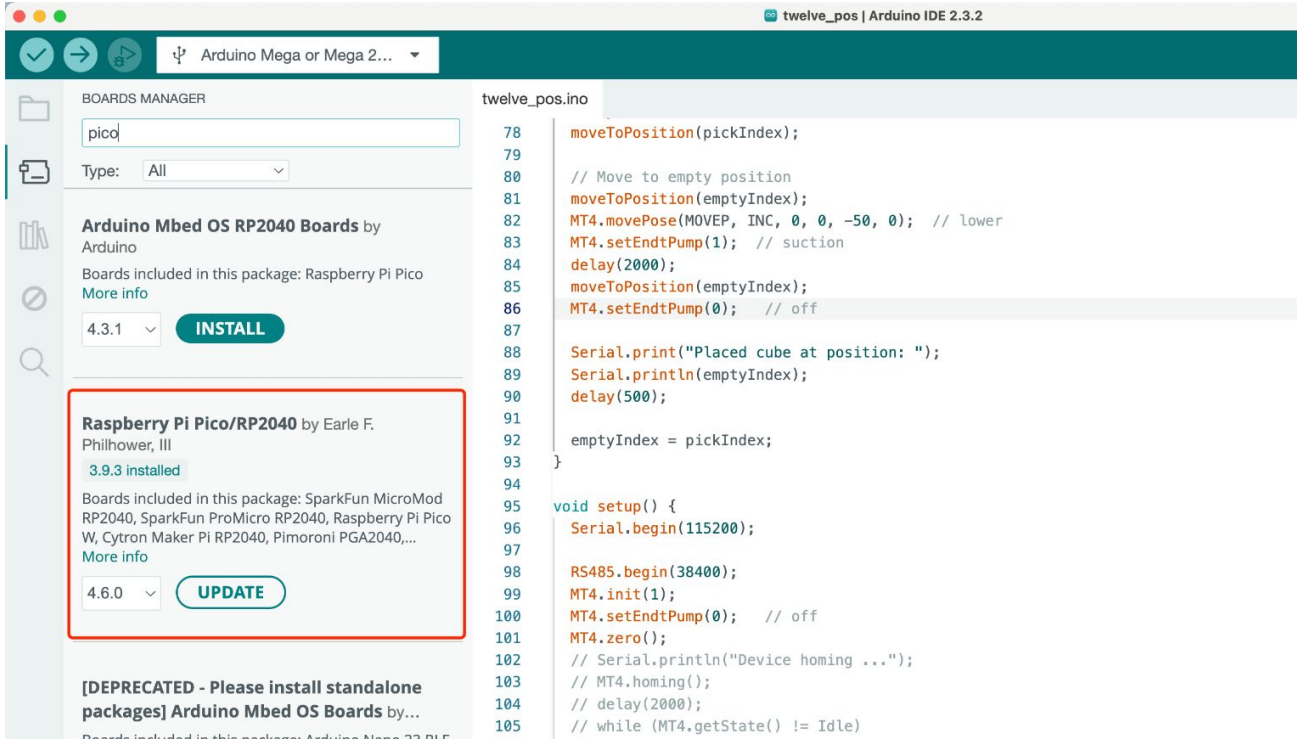
1. Fill https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json

   in the new window, click OK, and click OK on the Preferences window again

2. Open **Tools** in Menus, select **Board:"ArduinoUno"**, and then select "Boards Manager".

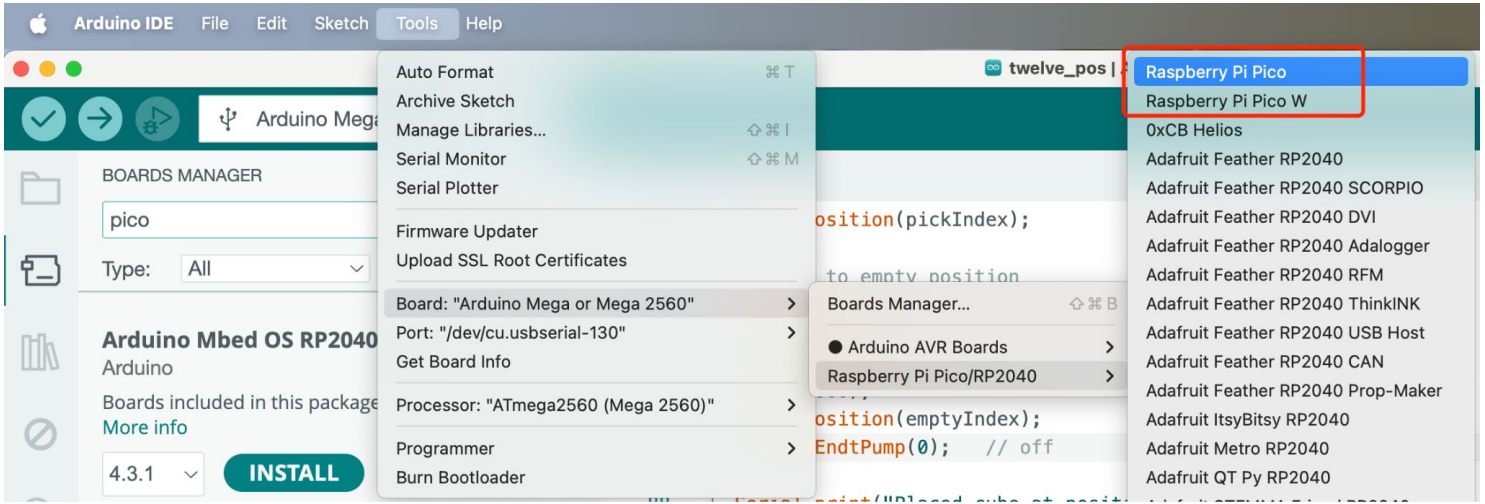
Environment Configuration

Input "Pico" in the window below, and press Enter. click "Install" to install.

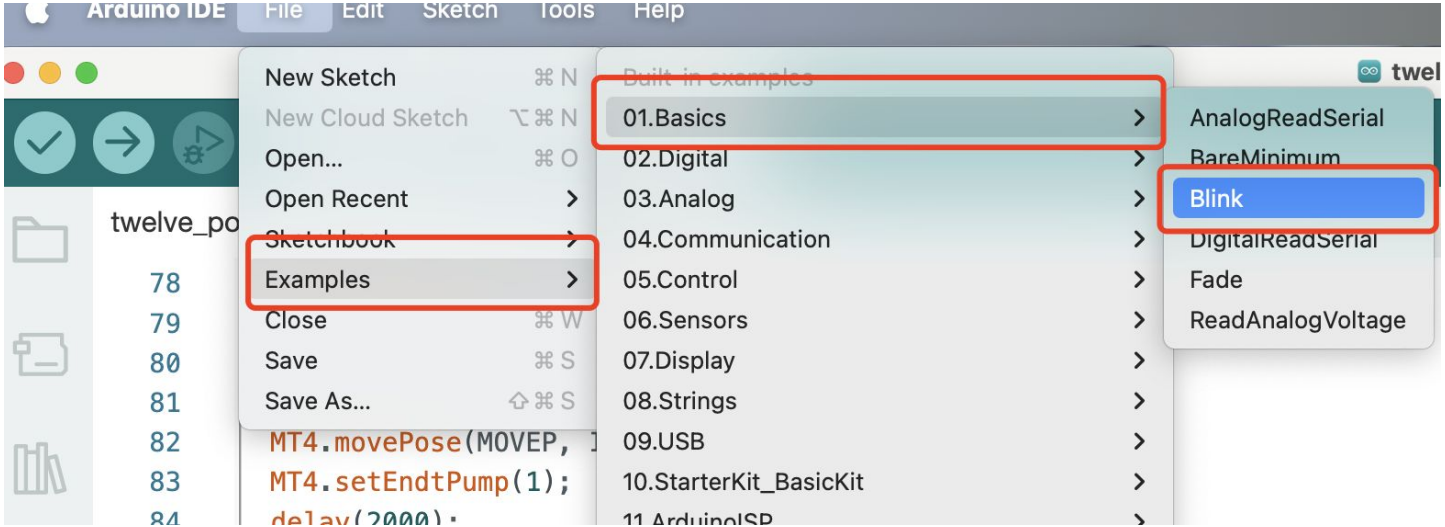


Environment Configuration

When finishing installation, click Tools in the Menus again and select Board: "Raspberry Pi Pico/RP2040", and then you can see information of Raspberry Pi Pico (W). Click "Raspberry Pi Pico W" so that the Raspberry Pi Pico W programming development environment is configured.

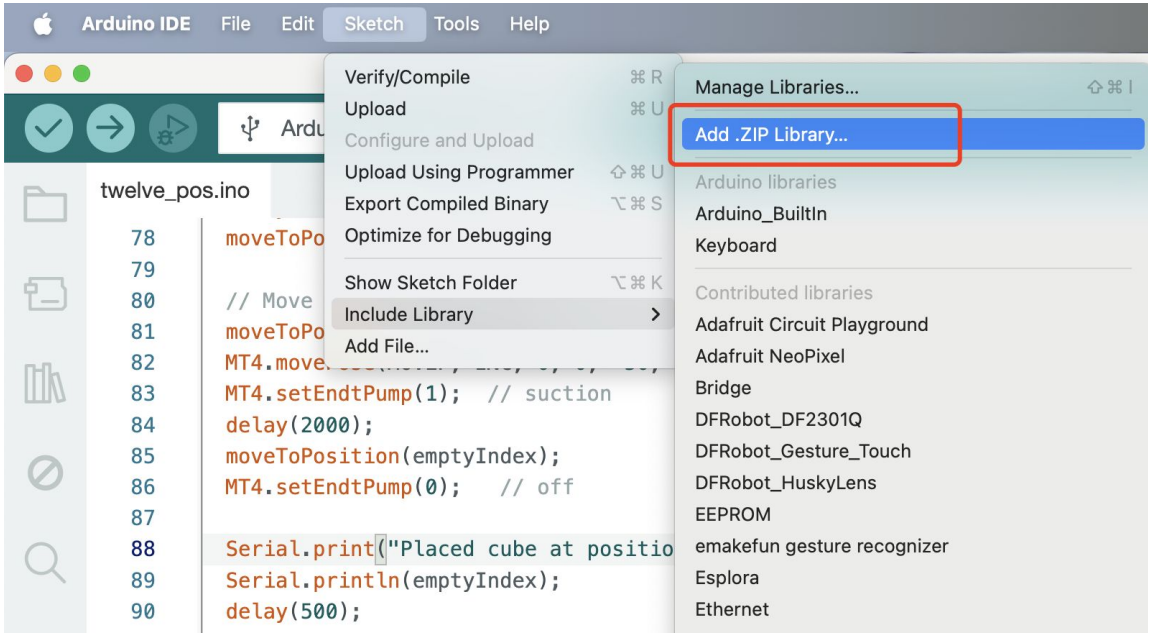


Test with Blink

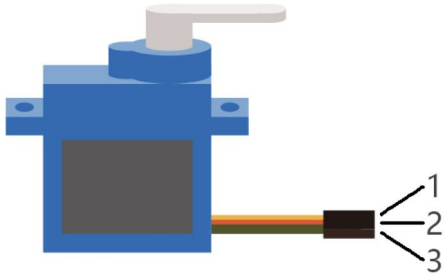


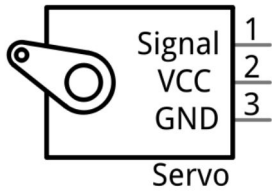
4WD Car Assembly and Library

Open Arduino IDE, click Sketch on Menu bar->Include Library ->Add .ZIP library



Turn Servo 90 Degree





hello.py -> python hello.py

Arduino:

folder name: hello

file name: hello.ino

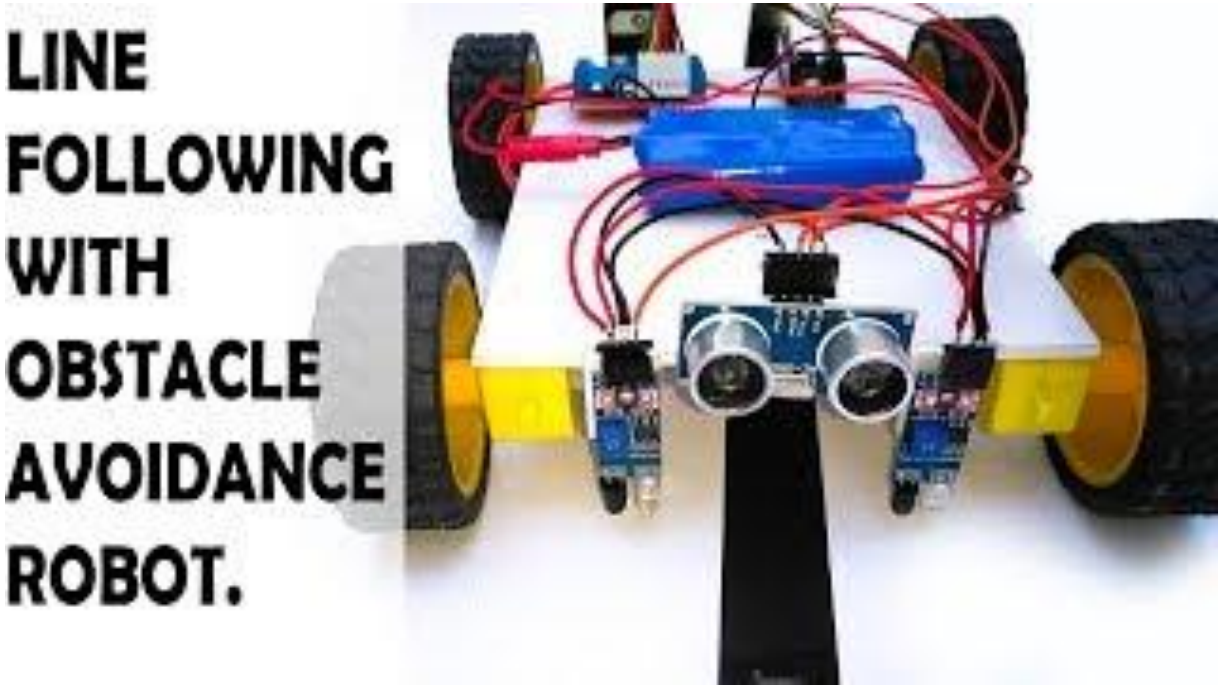
Turn Servo 90 Degree

```
#include "Freenove_4WD_Car_For_Pico_W.h"

void setup() {
 Servo_Setup();  //Servo initialization
}

void loop() {
 Servo_1_Angle(90);  //Set the Angle value of servo 1 to 90°
 delay(1000);
}
```

Line Following + Obstacle Avoiding

https://www.youtube.com/watch?v=ZiqAyuLpS3o

Line Following + Obstacle Avoiding



https://www.youtube.com/watch?v=Py2lBehF9rA

IR Remote Control

https://www.youtube.com/watch?v=ml2ZqmC_aIs

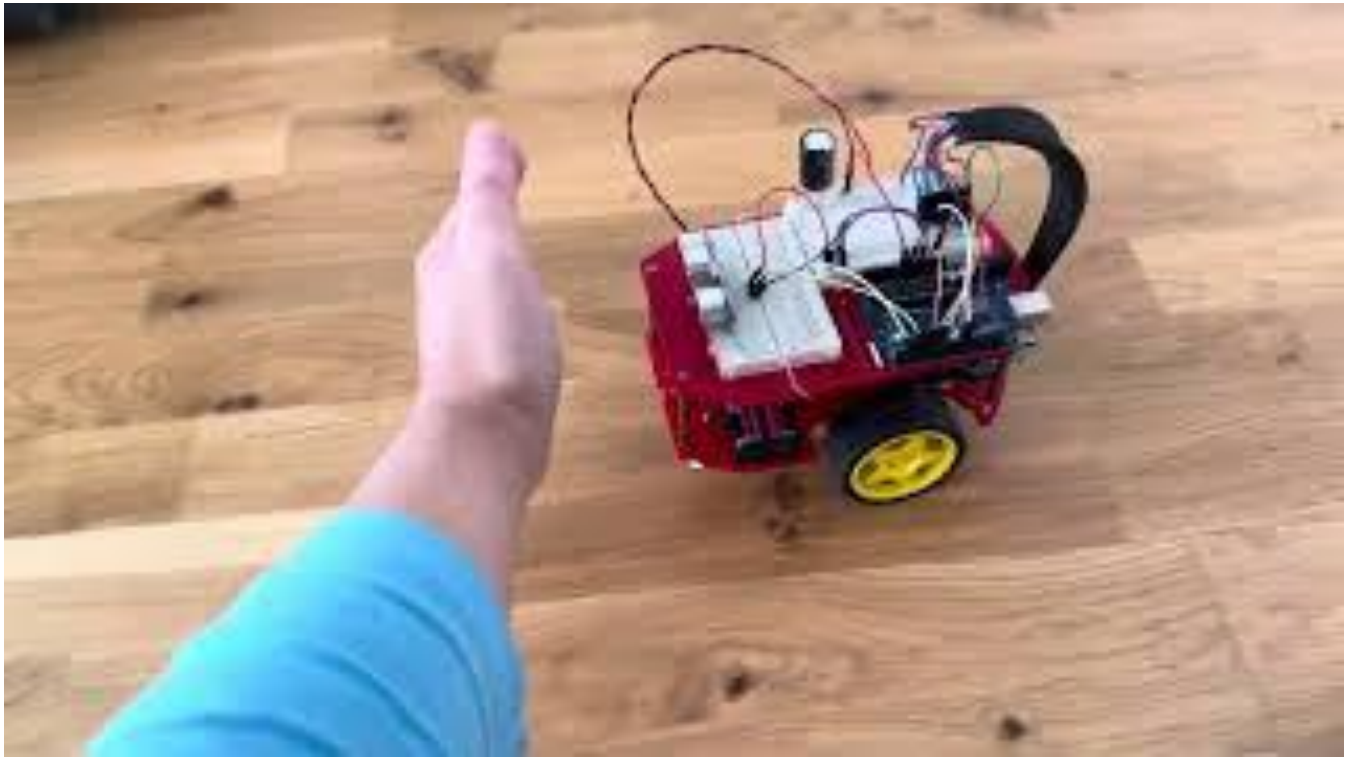
Remote Control + Obstacle Avoidance

https://www.youtube.com/watch?v=bUmWhUiorwQ

Object Following

https://www.youtube.com/watch?v=xF0DI_SjOAE

Robot with PID

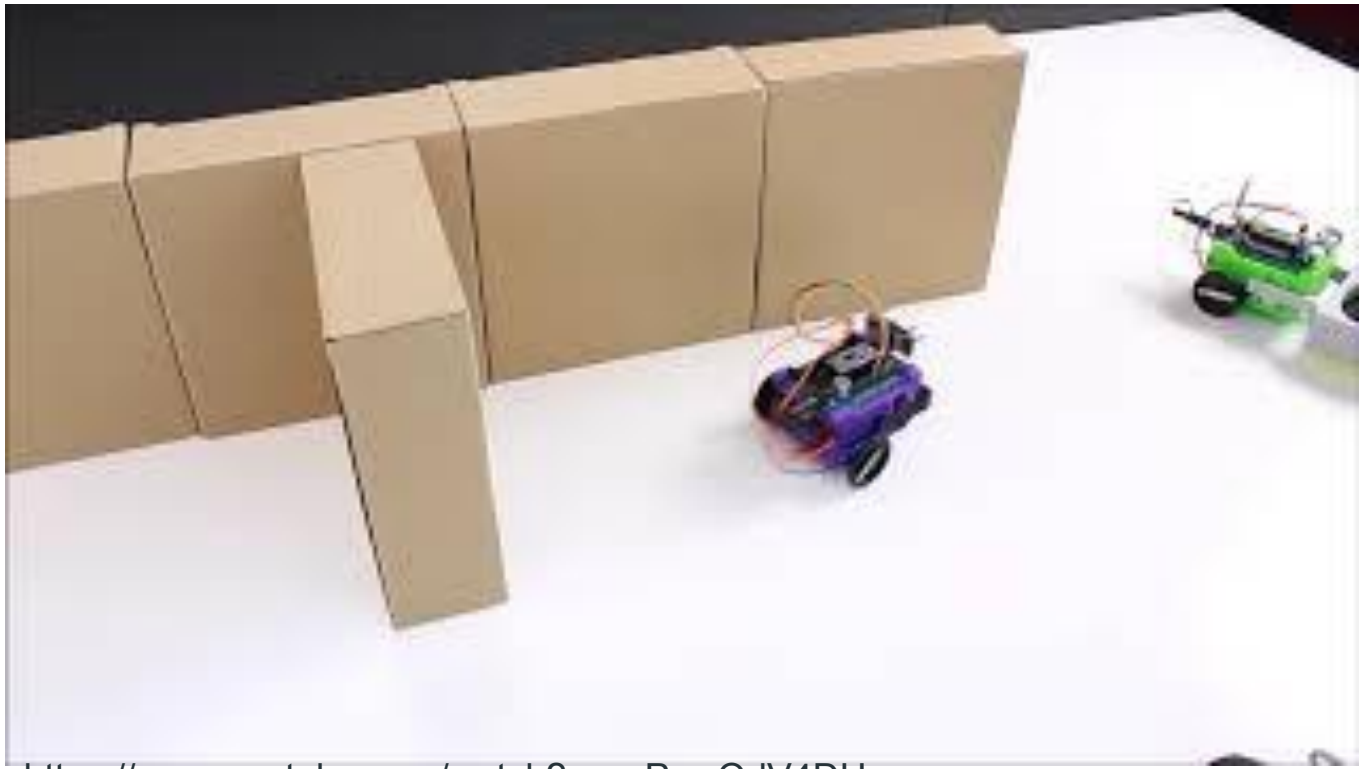


https://www.youtube.com/watch?v=vMdGYDRBIls

Wall Following with PID

https://www.youtube.com/watch?v=lqnzpmA6QDQ

Wall Following with PID

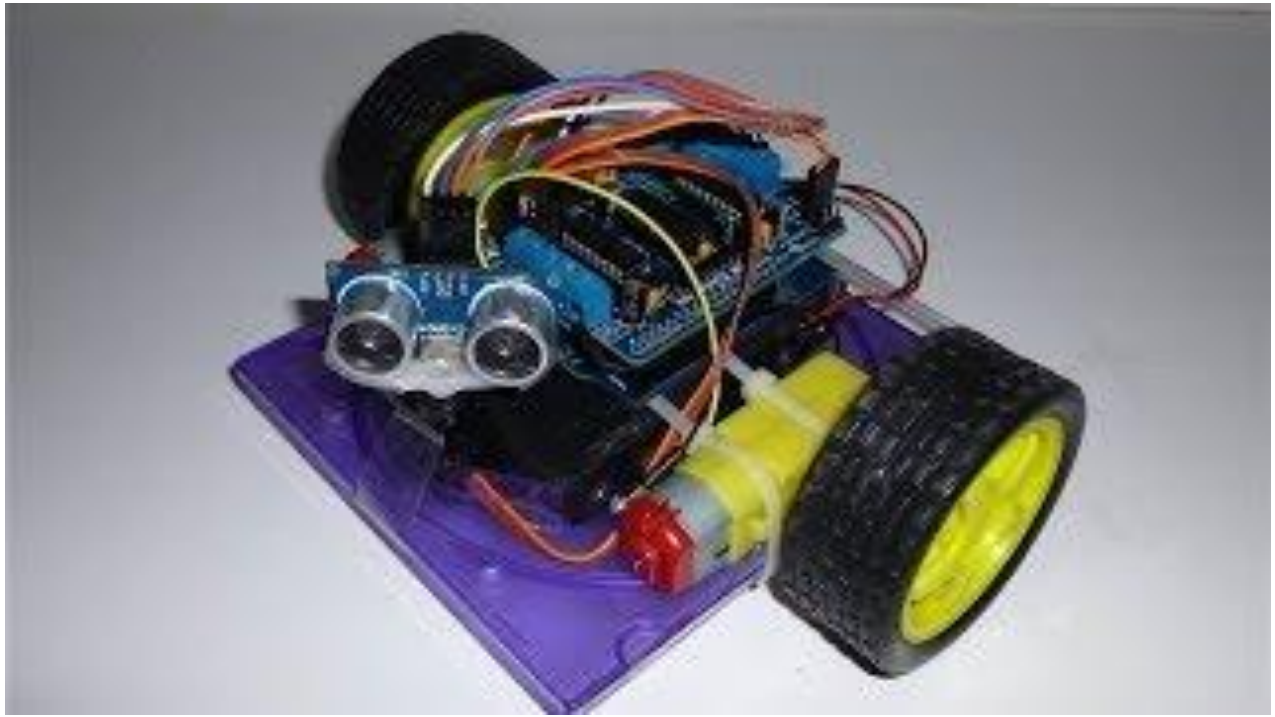


https://www.youtube.com/watch?v=vvPnwQdV4DU

Wall Following with PID

https://www.youtube.com/watch?v=tPZOjE9Vx8c

Explore with Obstacle Avoidance



https://www.youtube.com/watch?v=SLP5-1RtxUU

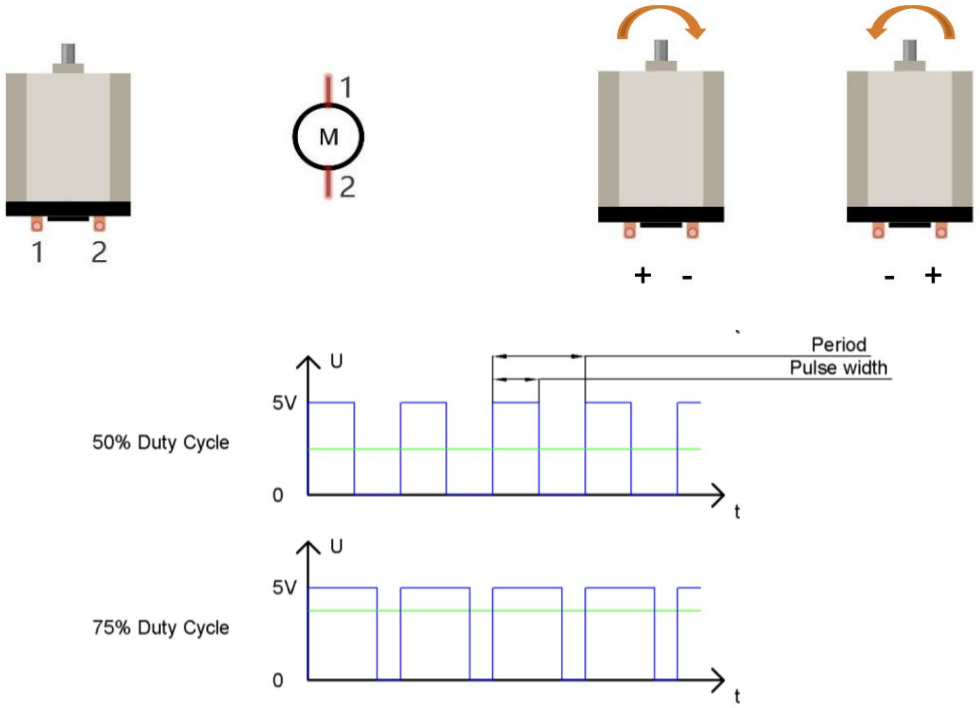
Vacuum Cleaner

https://www.youtube.com/watch?v=hoY2YxLGV98

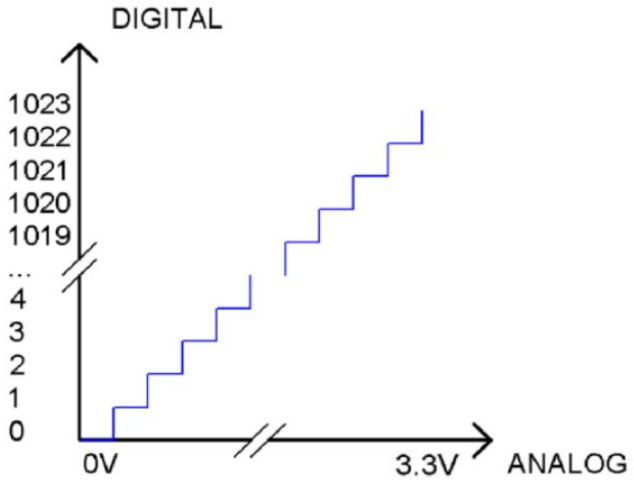
LiDAR with Robot

https://www.youtube.com/watch?v=XXnAnxWpPPs

Module test – Motor

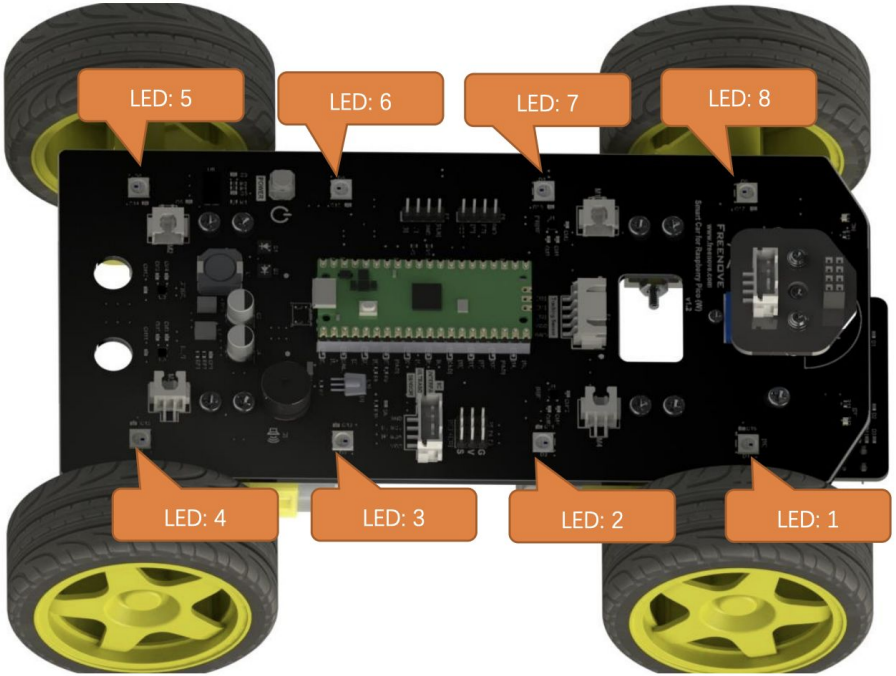


Module test – ADC

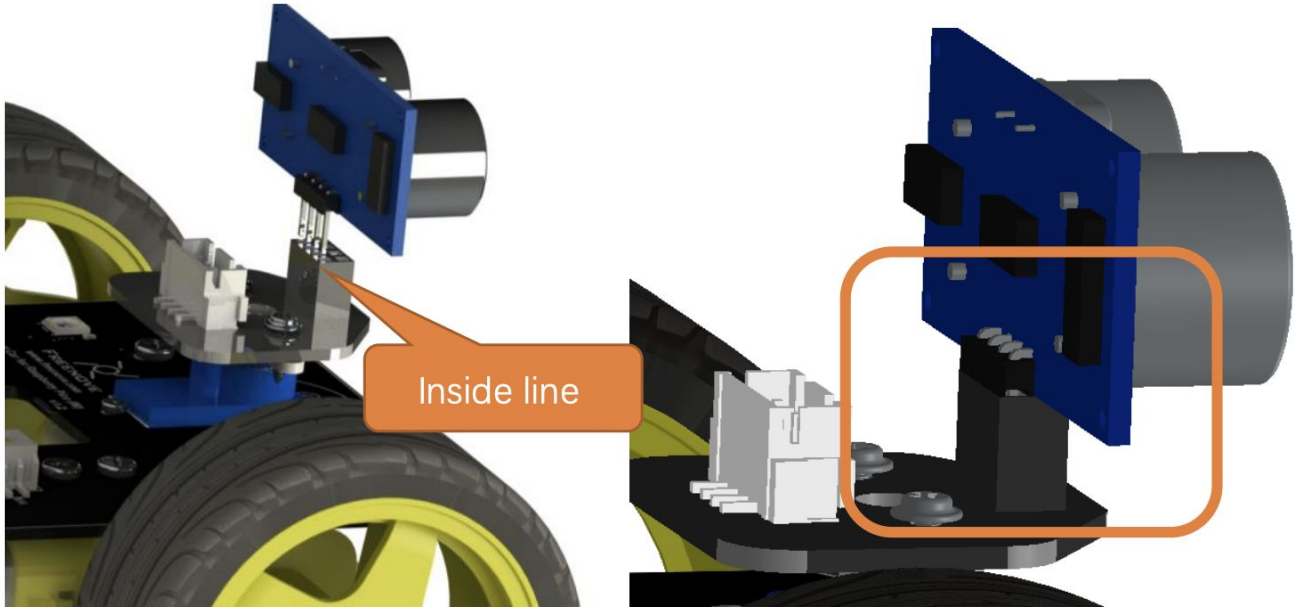


$$ADC\ Value = \frac{\text{Analog Voltage}}{3.3} * 1023$$

Module test – LED



Module test – Ultrasonic



Example Code

```
float Kp = 1.2;
float Ki = 0.02;
float Kd = 0.6;
float integral = 0;
float lastError = 0;
float TARGET_DISTANCE = 10;

float distance = Get_Sonar();
float error = distance - TARGET_DISTANCE;

integral += error;
float derivative = error - lastError;

float pidOutput = (Kp * error) + (Ki * integral) + (Kd * derivative);
lastError = error;

Motor_Move(pidOutput, pidOutput);
```

Module test – remote control

ICON	KEY Value	ICON	KEY Value
	BA45FF00	C	F20DFF00
MENU	B847FF00	1	F30CFF00
TEST	BB44FF00	2	E718FF00
+	BF40FF00	3	A15EFF00
	BC43FF00	4	F708FF00
⏮	F807FF00	5	E31CFF00
▶	EA15FF00	6	A55AFF00
⏭	F609FF00	7	BD42FF00
0	E916FF00	8	AD52FF00
–	E619FF00	9	B54AFF00

```
#include <IRremote.hpp>
#define IR_Pin 3
#define ENABLE_LED_FEEDBACK true
#define DISABLE_LED_FEEDBACK false

void setup() {
 // put your setup code here, to run once:
 Serial.begin(9600);
 IrReceiver.begin(IR_Pin, DISABLE_LED_FEEDBACK);
 Serial.println(IR_Pin);
}

void loop() {
 // put your main code here, to run repeatedly:
 if (IrReceiver.decode()){
   unsigned long value = IrReceiver.decodedIRData.decodedRawData;
   Serial.println(value, HEX);
   IrReceiver.resume();
 }
}
```

```
void handleCommands(unsigned long value) {
 // Handle the commands
 if (value == 0xBF40FF00) {
   Motor_Move(motor_speed, motor_speed);  //forward
   delay(200);
   Motor_Move(0, 0);
 } else if (value == 0xE619FF00) {
   Motor_Move(-motor_speed, -motor_speed); //Back
   delay(200);
   Motor_Move(0, 0);
 }}
```

onboard LED

```
#include <Arduino.h>
#include <Adafruit_NeoPixel.h>
#include "Freenove_4WD_Car_WS2812.h"
void setup() {
WS2812_Setup(); //WS2812 initialization
}
void loop() {
WS2812_Show(5);
}
```

IR remote control

```
void loop() {
 if (IrReceiver.decode()) {
   unsigned long value = IrReceiver.decodedIRData.decodedRawData;
   Serial.println(value, HEX);  // Print "old" raw data
   if (value != 0) {
     prev = value;
   } else {
     value = prev;
   }
   lastButtonTime = millis();
   handleControl(value);         // Handle the commands from remote
control
   IrReceiver.resume();     // Enable receiving of the next value
 }

 if ((millis() - lastButtonTime > timeoutDelay) && state == 1) {
   state = 0;
   Motor_Move(0 , 0);
 }
}
```

```
void handleCommands(unsigned long value)
{
 // Handle the commands
 if (value == 0xBF40FF00) {
   Motor_Move(motor_speed, motor_speed);
   delay(200);
   state = 1;
 } else if {...}
 }
}
```

Line_Tracking.ino

```ino
#include <Freenove.h>
void setup() {
  pinMode(PIN_TRACKING_LEFT, INPUT);
  pinMode(PIN_TRACKING_CENTER, INPUT);
  pinMode(PIN_TRACKING_RIGHT, INPUT);
  Motor_Setup();
}

void move(int speed) {
  Motor_Move(speed, speed);
}
void turnLeft(int speed) {
  Motor_Move(speed, -speed);
}
void turnRight(int speed) {
  Motor_Move(-speed, speed);
}
void wait() {
  delay(100);
}
int checkStop(int x, int y, int z) {
  for (int i = 0; i < 15; i++) {
    delay(10);
    if (x == 1 && y == 1 && z == 1) {
      move(0);
      delay(5000);
      return 1;
    }
  }
  return 0;
}

void loop() {
  int left = digitalRead(PIN_TRACKING_LEFT);
  int center = digitalRead(PIN_TRACKING_CENTER);
```

Line_Tracking.ino

```ino
void loop() {
  int left = digitalRead(PIN_TRACKING_LEFT);
  int center = digitalRead(PIN_TRACKING_CENTER);
  int right = digitalRead(PIN_TRACKING_RIGHT);

  int speed = 70;
  int turnSpeed = 50;

  //checkStop(left, center, right);
  move(30);
  if (left == 1 && right == 0) {
    if (center == 1) {
      if (checkStop(left, center, right) == 0) {
        turnRight(turnSpeed);
        delay(1500);
        move(speed);
      }
    } else {
      turnRight(turnSpeed);
      wait();
    }
  }
  if (right == 1 && left == 0) {
    if (center == 1) {
      if (checkStop(left, center, right) == 0) {
        turnLeft(turnSpeed);
        delay(1500);
        move(speed);
      }
    } else {
      turnLeft(turnSpeed);
      wait();
    }
```