Experiment on Electrical and Information Basics

Development of Line tracing Car by using Micro Controller
（Applied Experiment of Micro Controller）

National Institute of Technology, Akita College
Department of Electrical and Information Technology

Date：
Name：

Contents

1 Purposes

Using the Raspberry Pi Pico, which has a wide range of input and output ports, students will create a line tracing car using infrared sensors and PWM control of DC motors. By customizing the hardware and software through group work, they aim to create a line tracing car that operates faster and more accurately. The purpose of this experiment is to master methods of controlling electronic devices using microcomputers and to develop the ability to apply and develop this knowledge.

Another objective of this experiment is to form groups, clarify each student's role, and achieve the goal through group work.

2 Introduction

2.1 Rasberry Pi Pico

Raspberry Pi Pico (hereafter referred to as Pico in this experiment) is a microcomputer board equipped with an RP2040 microcontroller chip. The RP2040 contains a CPU, multiple memories, and functions for connecting to external devices, allowing it to function as a small computer. The microcontroller board is equipped with this chip, resistors and capacitors to adjust the input and output electrical signals, pin headers for connecting to external devices, etc., and operates as a microcontroller when powered. The operating principle of Pico is shown in Fig. 1.

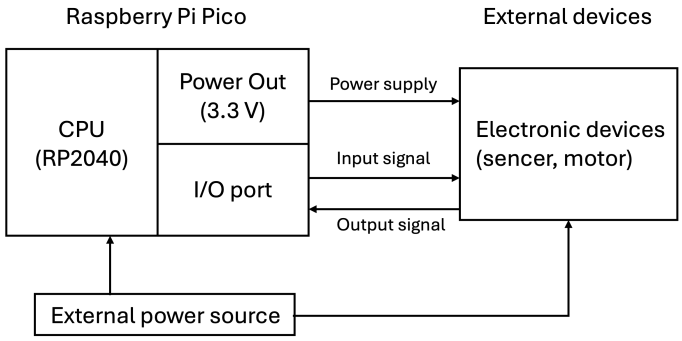


Figure 1: Schematics of Raspberry Pi Pico

Pico has 40 input/output terminals (pins), the layout of which is shown in Figure 1. As shown below, in addition to the power input/output, it also has general-purpose input/output ports (GPIO: General Purpose Input/Output) that can be used for input and output of digital signals.

- GPIO: I/O (input/output) pin. When writing to a program, specify this GPIO number instead of the number marked on the board.
- GND：GND pin
- VBUS：5 V output pin (When connected to a PC via USB, 5 V is output.)
- VSYS：1.8~5.5 V power input pin

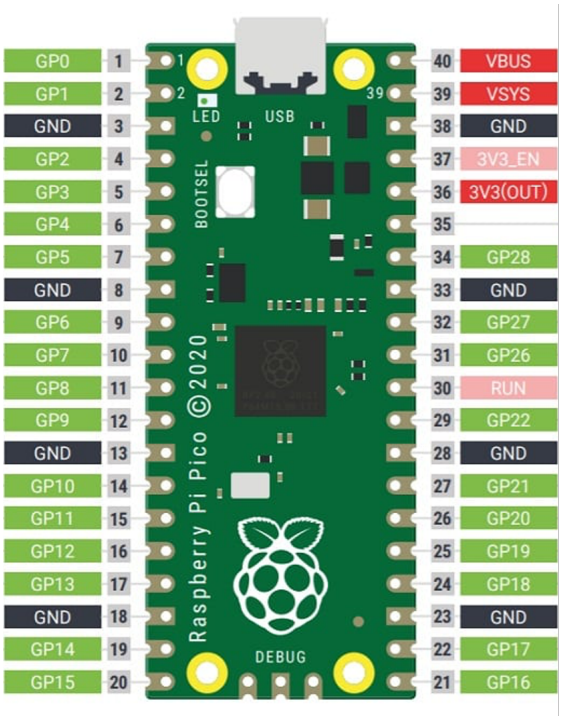


Figure 2: Input and output pins of Pico

- 3V3(OUT)：3.3 V output pint

The Raspberry Foundation, which develops Pico, recommends using the programming language ”Python” as the development environment for Pico. Furthermore, they recommend ”Thonny”, developed by the University of Tartu in Estonia, as the Python development environment. This experiment will also involve development using Python on Thonny.

2.2 Python

In conventional microcontroller programming using the C language, a large series of programs are converted (compiled) into machine language all at once using a PC development environment (compiler) before being written to the microcontroller.

On the other hand, if you load firmware called ”MicroPython” into the RP2040, a Python program written on a PC will be translated line by line into machine code that the Pico’s CPU can understand. This mechanism allows you to immediately try out the source code you’ve written on the Pico. In other words, it makes it easy to smoothly rewrite programs while watching Pico’s reactions. This type of development environment is called a REPL (Read-Eval-Print Loop). Watching Pico’s reactions in the REPL is ideal for learning, so we used MicroPython in this experiment.

Other reasons why Python is recommended include:

- used all over the world
- A wealth of ready-to-use libraries (modules, templates, samples) are available

If you want to use Pico to control electronic devices, having to read through a thick programming textbook can be discouraging. Python is relatively short and concise, making it an easy and convenient way to get started. There are countless Python users around the world, and you can find plenty of sample programs on the Internet.

2.3 Precautions

Before starting the experiment, please be sure to read the following precautions.

- The terminals of the element are very delicate and will be damaged if excessive force is applied. Use special tools when handling them.
- Do not apply voltages exceeding the specified range, as this may damage the devices and sensors.
- A short circuit can cause high heat to be generated in elements such as sensors and microcontrollers, which can be dangerous, so be sure to thoroughly check that the circuit is correct before applying voltage. A short circuit can cause a distinctive smell and the elements to heat up. If you notice anything abnormal, immediately turn off the power and check the wiring, etc. Also, do not touch elements that have become hot due to a short circuit, as there is a risk of burns if you touch them, so do not touch them until they have cooled down sufficiently.
- Carefully read the experimental procedure and connection examples shown in the circuit diagram to conduct the experiment safely.

3 Experiments

3.1 Preparing the Python development environment

Source code written in Python is written on the integrated development environment "Thonny" prepared on a PC, and then transferred to Pico to execute the program. MicroPython is firmware (translator) for microcontrollers that translates Python into machine code. If MicroPython is uploaded to Pico in advance, Python source code written on a PC can be sent directly to Pico. Each line of Python code is converted into binary code (machine language) that can be understood by the Pico's CPU (RP2040). The concept of this development environment is shown in Fig. 3.

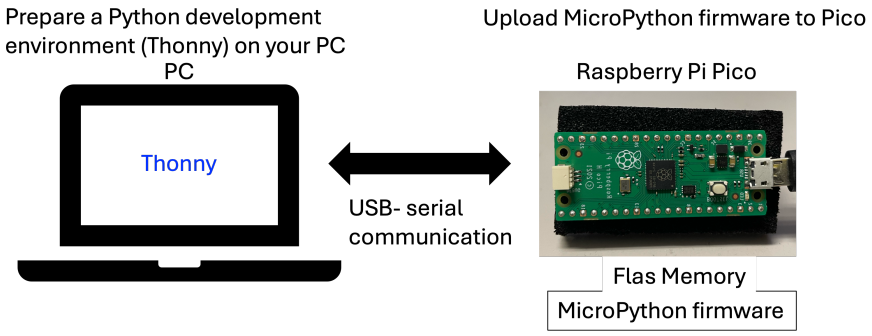
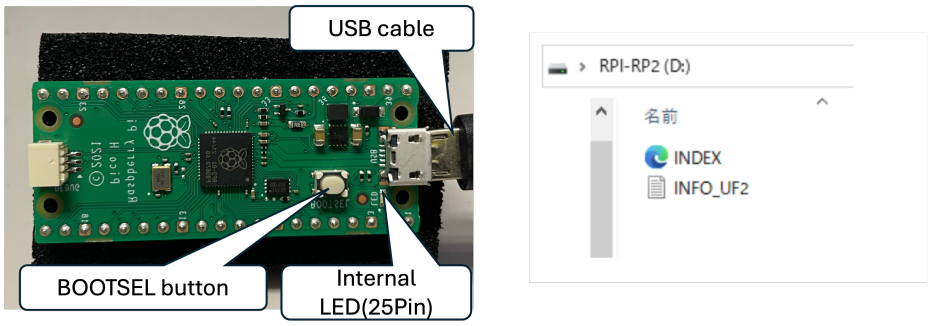


Figure 3: Development environment of Pico

Follow the steps below to set up your development environment.

3.1.1 Start Pico in BOOTSEL mode and access the CPU (RP2040)

As shown in Fig. 4(a), if you press the BOOTSEL button on the Pico and connect the USB cable to your PC, the Pico will start up in BOOTSEL mode, allowing you to access the Pico from your PC. When it starts up in BOOTSEL mode, you can view the Pico drive (RPI-RP2) from your PC's file browser (Windows Explorer like Fig. 4(b)). Clicking on "index.htm" in the RPI-RP2 folder will launch a web browser and access the Raspberry Pi RP2040 page shown in Fig. 5. From this page, you can download the MicroPython firmware (UF2 file) for the Pico.



(a) Booting the Pico in BOOTSEL mode

(b) Pico drive in BOOTSEL mode mounted on PC

Figure 4: Booting Pico in BOOTSEL mode (a) and mounting it on a PC (b)

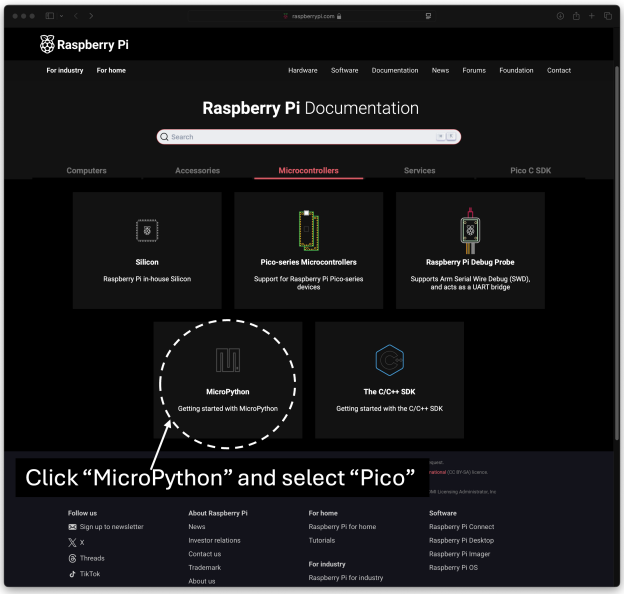


Figure 5: Download MicroPython firmware

3.1.2 Downloading the MicroPython firmware and copying it to the Pico

Copy the downloaded MicroPython firmware (UF2 file) to the RPI-RP2 drive. This will write the MicroPython firmware to the Pico, and it will be unmounted on your PC and will not be visible in Explorer.

3.1.3 Download and set up the development environment Thonny (if Thonny is not installed on your PC)

Visit the official Thonny website[1] shown in Fig. 7, download the installer that matches your PC's OS, and install it according to the instructions. After installation is complete, double-click the Thonny icon to launch a screen like Fig. 7. Click on communication settings in the bottom right and select Pico. If no errors are displayed in the shell, the connection between Pico and your PC has been established and your development environment is ready.

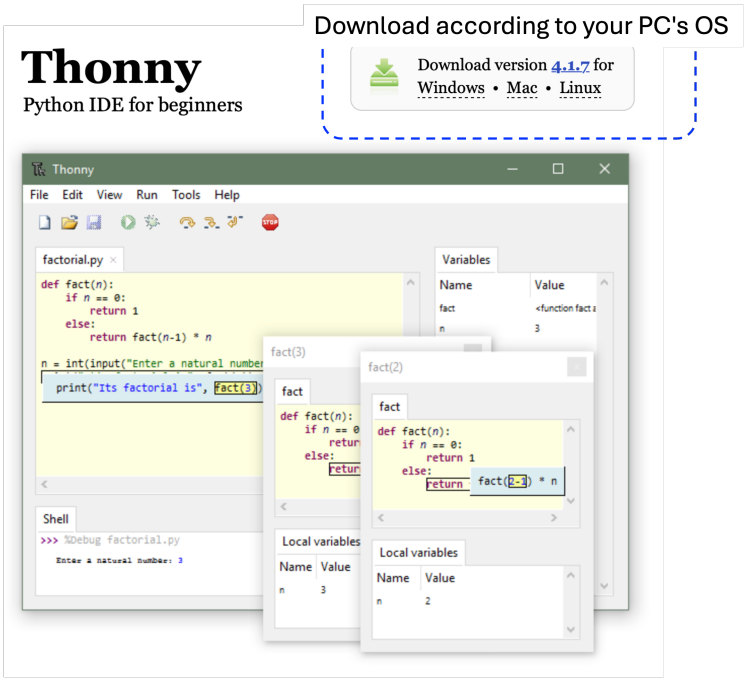


Figure 6: Download the Thonny installer from the official Thonny website.

[1] https://thonny.org

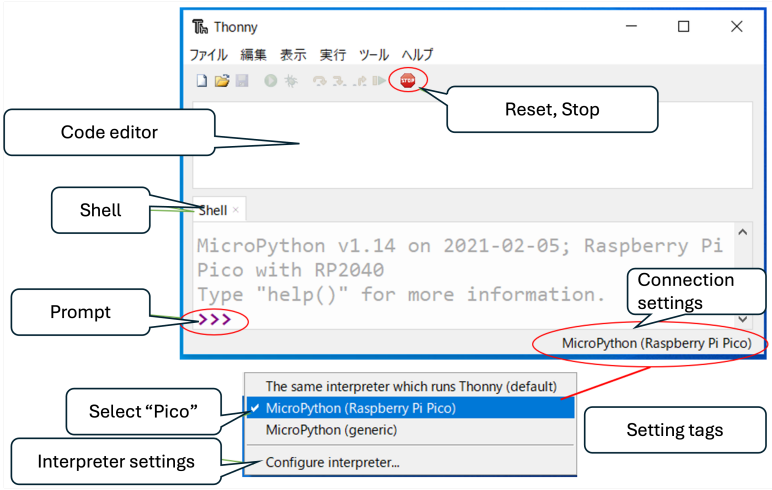


Figure 7: Thonny startup screen

3.2 LED lighting experiment1 (L blinking)

The Pico board is equipped with a green LED. This LED is connected to pin GPIO25, so we will try lighting up this LED by setting the output voltage of pin GPIO25 to ”High” or ”Low.” The list 1 is a program that repeatedly turns the LED on and off every 0.5 seconds (blinking the LED). In Python, lines starting with # are commented out.

List 1: LED lighting (blinking) program（L-chika.py）

```
1 import machine,utime  # import modules
2
3 led = machine.Pin(25, machine.Pin.OUT) # define GPIO25 as output pin
4
5 while True: # Iterative processing
6     led.value(1)       # LED ON
7     utime.sleep(0.5)   # wait 0.5 sec
8     led.value(0)       # LED OFF
9     utime.sleep(0.5)   # wait 0.5 sec
```

[Exercise 1] Try changing the LED lighting interval (ON and OFF time) as you like.

3.3 LED lighting experiment 2 (PWM Controll)

In LED lighting experiment 1, the LED will blink at a cycle of 1 second (ON: 0.5 seconds, OFF: 0.5 seconds). Consider changing utime.sleep(0.5) on lines 7 and 9 of the list 1 to 0.05 and 0.005. The blinking can be seen with the naked eye up to 0.05 seconds (cycle 0.1 seconds, frequency 10 Hz), but at 0.005 seconds (cycle 0.01 seconds, frequency 100 Hz), the eye cannot keep up and the LED appears to be emitting light continuously.

Here, as shown in Figure 8, the ON time per cycle is called the duty ratio D (Duty ratio), and is defined as follows: (3.1)

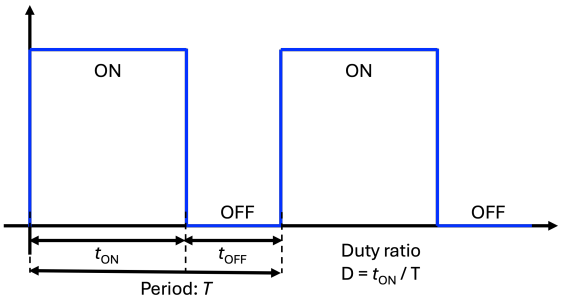


Figure 8: Duty ratio

$$D = \frac{t_{ON}}{t_{ON} + t_{OFF}} = \frac{t_{ON}}{T} \tag{3.1}$$

Let’s fix the blinking frequency to 100-Hz, which is too fast for the human eye to keep up, and change the duty ratio. Comparing a duty ratio of 10% (on time: 0.001 seconds, off time: 0.009 seconds) with a

duty ratio of 90% (on time: 0.009 seconds, off time: 0.001 seconds), it appears as if the light is emitting continuously, with the light intensity being weaker at 10% and stronger at 90%. This is called pulse width modulation (PWM) dimming.

The combination of Pico and Python comes with a library (module) for PWM control. Let's create a list 2 using this library and check the PWM dimming.

List 2: LED lighting program using PWM dimming (L-PWM.py)

```
1 from machine import Pin, PWM # import modules
2
3 pwm = PWM(Pin(25))          # define GPIO25 as output pin with PWM
4 pwm.freq(100)               # set the PWM frequency PWM as 100 Hz
5 duty = 10                   # Duty ratio (0~100%)
6 pwm.duty_u16(int(65536*duty/100))
```

[Exercise 2] Change the duty ratio to 0, 20, 40, 60, 80, 100% and confirm that the light intensity changes.

[Exercise 3] What does line 6 of the list 2 do?

3.4 How to use a breadboard

It is recommended to connect Pico to external components on a breadboard. While soldering is usually required to create electronic circuits, using a breadboard eliminates the need for soldering, making it easier to modify and test circuits. Breadboards are ideal for student experiments and the prototype stage of circuits.

As shown in Figure 9, a breadboard has small holes for inserting pins. The holes are spaced 1/10 of an inch apart, which is the same as the pin spacing on an IC (Integrated Circuit). Circuits are created by inserting ICs into these holes. Not only ICs, but also semiconductor components such as resistors, switches, and transistors can be inserted. The inserted components are connected with wiring (jumper wires) to complete the circuit.

To use a breadboard, it is important to understand the wiring inside it. As shown in Figure 9, several holes are electrically connected inside. By understanding this connection, you can use a breadboard to complete a circuit with neat wiring.

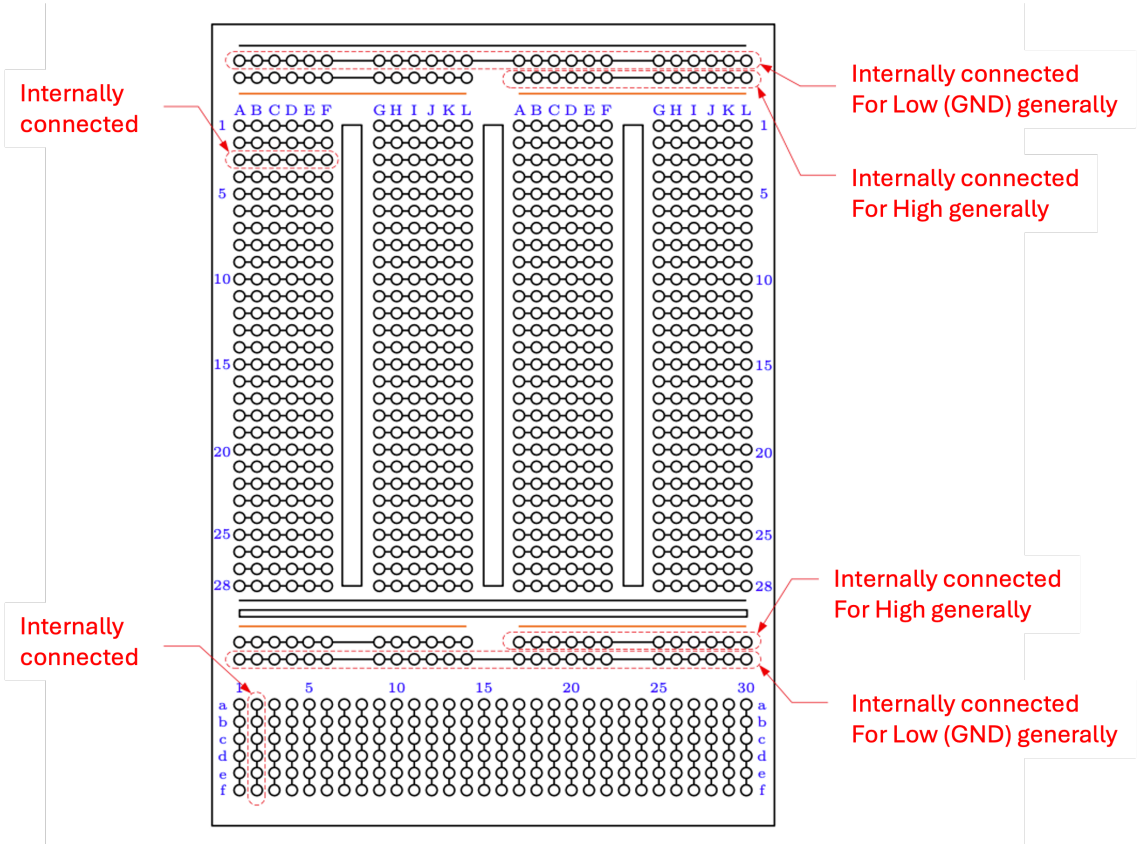


Figure 9: Internal wiring of the breadboard

3.5 Experiment on detecting approaching objects using infrared sensors

We will experiment with detecting approaching objects by applying conditional branching processing using a reflective photosensor.

The reflective photosensor used in this experiment has a light-emitting element and a light-receiving element attached in the same direction. As shown in Figure 10, light (infrared light) is emitted from the light-emitting element, and the presence or absence of an object is determined by how much of the reflected light hits the light-receiving element.

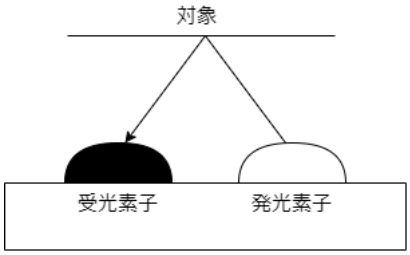


Figure 10: Photo sensor operating concept

3.5.1 Program

Create the program shown in Listing 3.

List 3: A program to detect approaching objects using a reflective photosensor（photo_sensor.py）

```
1  from machine import Pin
2  import utime
3
4  class Photo_sensor:
5      def __init__(self):
6
7          self.sensor = Pin(22,Pin.IN)
8
9      def read_sensor(self):
10
11         return self.sensor.value()
12
13 sensor = Photo_sensor()
14
15 while True:
16     if sensor.read_sensor() == 0:
17         print("Reflect")
18         utime.sleep(0.5)
19
20     else:
21         print("not Reflect")
22         utime.sleep(0.5)
```

3.5.2 Circuit

Create the circuit shown in Figure 11 on a breadboard. For this experiment, we will use the infrared reflective photosensor TCRT5000. The appearance of the TCRT5000 is shown in Figure 12.

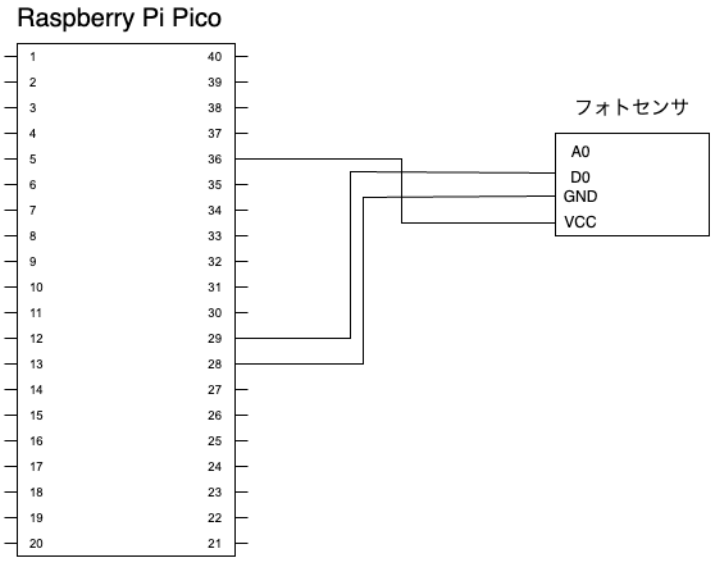


Figure 11: Photo sensor circuit for detecting approaching objects

Figure 12: Appearance of TCRT5000

[Exercise 4] Run the list 3 and check that it works.

[Exercise 5] Try adjusting the sensitivity of the photosensor by turning the variable resistor attached to it.

3.6 PWM control of DC motors

Connect a motor driver to Pico and try PWM control of the rotation speed of a DC motor. Generally, microcontrollers that issue control signals do not output enough electrical energy (current and voltage) to drive a motor. Therefore, the motor is controlled via a motor driver with a built-in amplifier.

3.6.1 Program

Create the program shown in the list 4.

List 4: DC motor PWM control program（motor_PWM.py）

```
from machine import PWM, Pin
import utime

IN1 = PWM(Pin(26))
IN2 = PWM(Pin(27))
IN1.freq(100)
IN2.freq(100)

max_duty = 65025

while True:

    IN2.duty_u16(0)
    for i in range(50,100):
        IN1.duty_u16(int(max_duty*i*0.01))
        utime.sleep(0.1)

    IN1.duty_u16(0)
    for i in range(50,100):
        IN2.duty_u16(int(max_duty*i*0.01))
        utime.sleep(0.1)

    IN1.duty_u16(0)
    IN2.duty_u16(0)
    utime.sleep(2)
```

3.6.2 Circuit

Create the circuit shown in Figure 13 on a breadboard. Connect 3 V batteries (two AA batteries) to the motor driver as the power source for the motor. For this experiment, use DRV8833 as the motor driver. The appearance of DRV8833 is shown in Figure 14.

[Exercise 6] Run the list 4 and check that it works.

[Exercise 7] In the list 4, the force applied to the motor is controlled by the combination of the AIN1 and AIN2 signals (”High” or ”Low”). Compare the results with the operation and make a table showing what combinations are controlled.

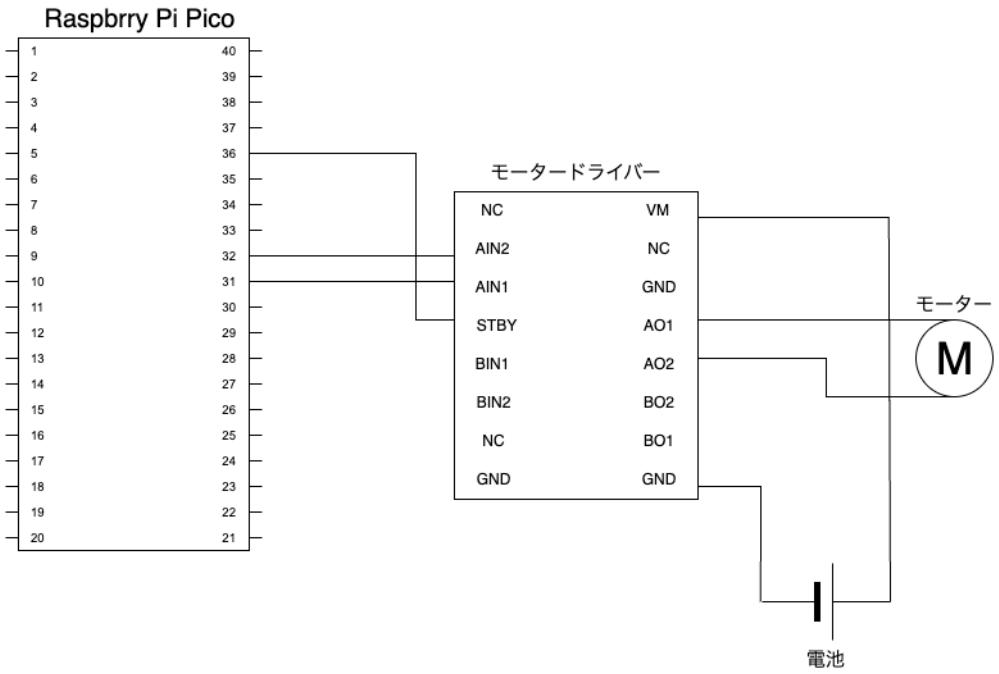


Figure 13: PWM control circuit for DC motor

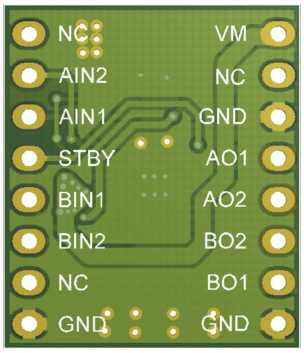


Figure 14: Appearance of DRV8833

3.7 Making a line tracing car using an infrared sensor and a DC motor

By combining an infrared sensor and a DC motor, students will create a line tracing car. Although there are an infinite number of possible combinations of hardware and software (programs), each group will use the tools provided to create an original line tracing car. The appearance of a completed example (prototype) is shown in Figure 15, the circuit diagram in Figure 16, and an example program in List 5.

When running the line trace car, the PC and USB cable must be disconnected. In this case, by saving the program name ”main.py” to Pico, the program written in main.py will be executed when the power is turned on.

This program allows for turning using a super-spin turn. There may be other more efficient ways to turn, so it’s a good idea to research and try them out.

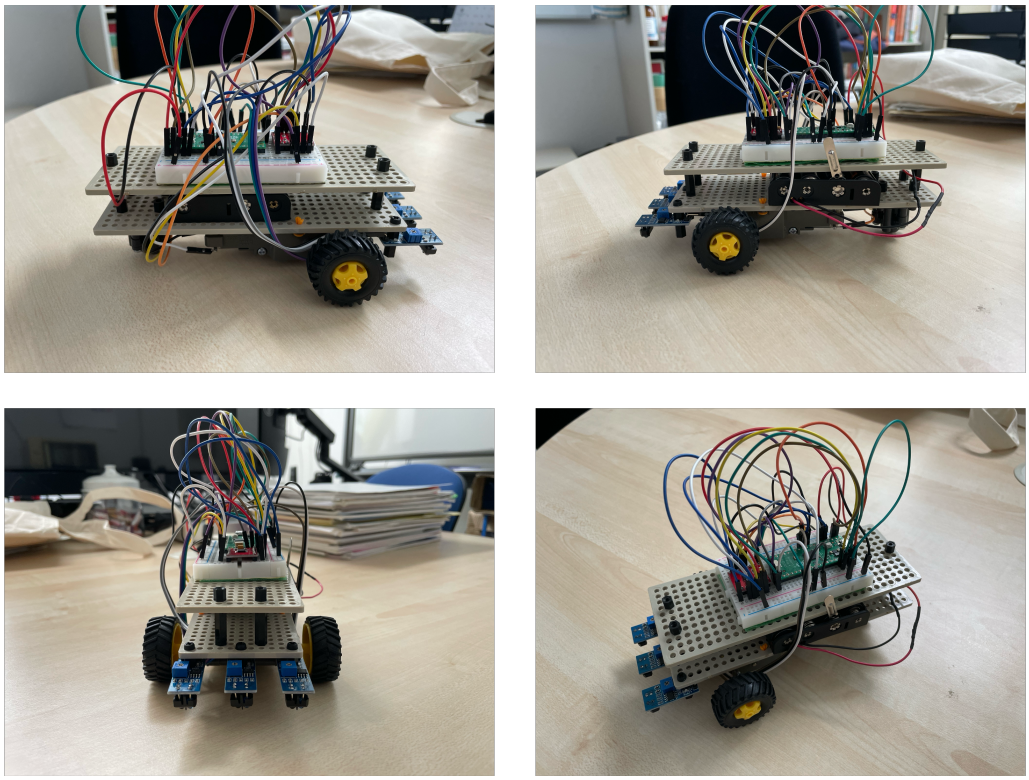


Figure 15: Appearance of the line trace car (prototype)

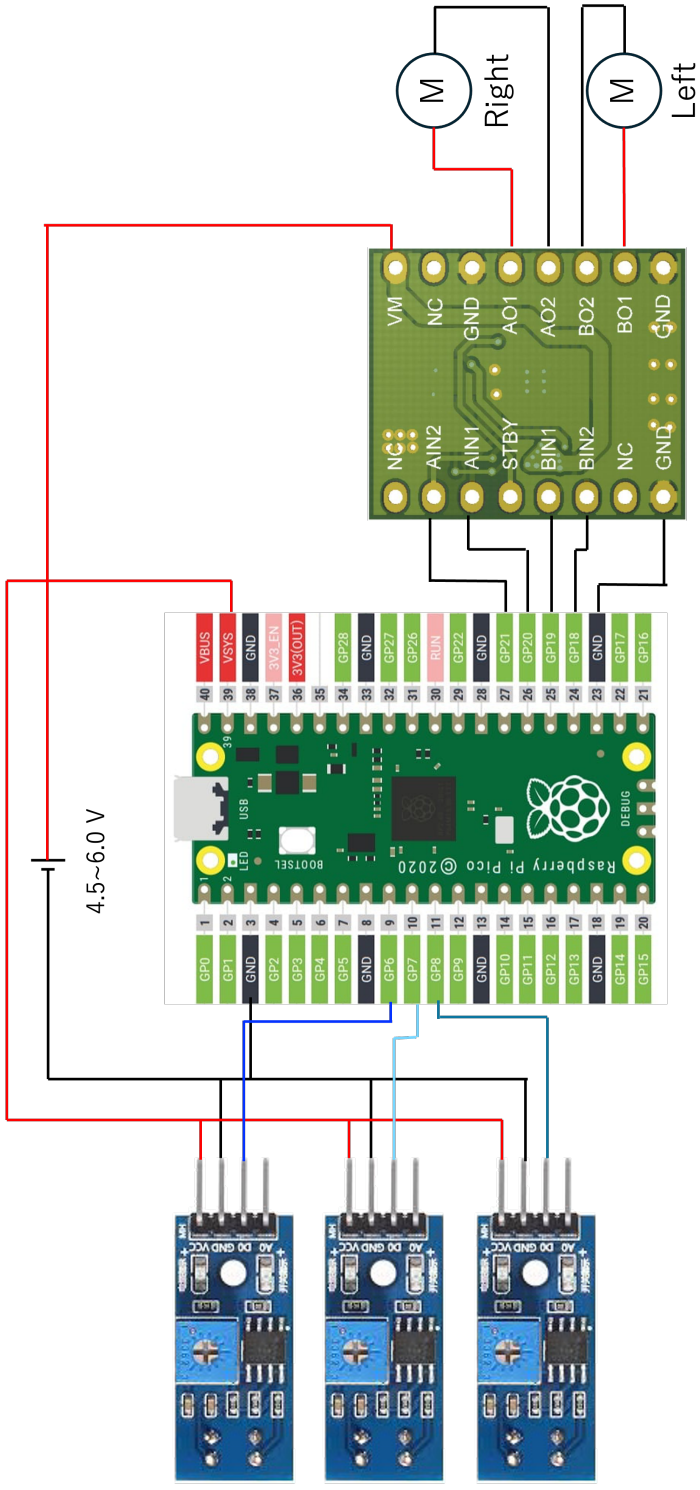


Figure 16: Circuit diagram of the line tracing car (prototype)

List 5: Line trace car control program example（main.py）

```
from machine import PWM, Pin
import utime

class Robot:
    def __init__(self):
        self.IN1 = PWM(Pin(20))
        self.IN2 = PWM(Pin(21))
        self.IN3 = PWM(Pin(19))
        self.IN4 = PWM(Pin(18))
        self.IN1.freq(1000)
        self.IN2.freq(1000)
        self.IN3.freq(1000)
        self.IN4.freq(1000)
    def forward(self,v):
        self.IN1.duty_u16(duty(v))
        self.IN2.duty_u16(0)
        self.IN3.duty_u16(duty(v))
        self.IN4.duty_u16(0)
    def right(self,v):
        self.IN1.duty_u16(duty(v))
        self.IN2.duty_u16(duty(v))
        self.IN3.duty_u16(duty(v))
        self.IN4.duty_u16(0)
    def left(self,v):
        self.IN1.duty_u16(duty(v))
        self.IN2.duty_u16(0)
        self.IN3.duty_u16(duty(v))
        self.IN4.duty_u16(duty(v))
    def right_spin(self,v):
        self.IN1.duty_u16(0)
        self.IN2.duty_u16(duty(v))
        self.IN3.duty_u16(duty(v))
        self.IN4.duty_u16(0)
    def left_spin(self,v):
        self.IN1.duty_u16(duty(v))
        self.IN2.duty_u16(0)
        self.IN3.duty_u16(0)
        self.IN4.duty_u16(duty(v))
    def back(self,v):
        self.IN1.duty_u16(0)
        self.IN2.duty_u16(duty(v))
        self.IN3.duty_u16(0)
        self.IN4.duty_u16(duty(v))
    def stop(self):
        self.IN1.duty_u16(duty(100))
        self.IN2.duty_u16(duty(100))
        self.IN3.duty_u16(duty(100))
        self.IN4.duty_u16(duty(100))
    def release(self):
        self.IN1.duty_u16(0)
        self.IN2.duty_u16(0)
        self.IN3.duty_u16(0)
```

```
53          self.IN4.duty_u16(0)
54
55  class Photo_sensor:
56      def __init__(self):
57          self.sensor_1 = Pin(6, Pin.IN)
58          self.sensor_2 = Pin(7, Pin.IN)
59          self.sensor_3 = Pin(8, Pin.IN)
60          self.value = [0]*3
61
62      def read_sensor(self):
63          self.value[0] = self.sensor_1.value()
64          self.value[1] = self.sensor_2.value()
65          self.value[2] = self.sensor_3.value()
66          return self.value
67
68  def duty(percent):
69      return int(65025*percent*0.01)
70
71  robot = Robot()
72  sensor = Photo_sensor()
73  value_old = [0, 0, 0]
74  utime.sleep(1)
75
76  while True:
77      value = sensor.read_sensor()
78
79      if value[1]==1:
80          if value[0]==value[2]:
81              robot.forward(80)
82          elif value[0]==1:
83              robot.left_spin(80)
84          elif value[2]==1:
85              robot.right_spin(80)
86
87      else:
88          if value[0]==1:
89              robot.left_spin(80)
90          elif value[2]==1:
91              robot.right_spin(80)
92          else:
93              if value_old[0]==1:
94                  robot.left_spin(100)
95                  utime.sleep(0.2)
96              elif value_old[2]==1:
97                  robot.right_spin(100)
98                  utime.sleep(0.2)
99              else:
100                 robot.forward(80)
101     value_old = value
```