Piarm

Release 1.0

www.sunfounder.com

Aug 02, 2022

CONTENTS

1 Component List and Assembly Instructions **3**

2 Hardware Introduction **5**
- 2.1 Arm . . . 5
- 2.2 Shovel Bucket . . . 8
- 2.3 Hanging Clip . . . 12
- 2.4 Electromagnet . . . 16
- 2.5 Dual Joystick Module . . . 18
- 2.6 About Robot HAT . . . 21

3 Play with Ezblock **23**
- 3.1 Quick Guide on Ezblock . . . 23
- 3.2 Test 3 EoATs . . . 25
- 3.3 Sound Effects . . . 33
- 3.4 Dual Joystick Module . . . 37
- 3.5 Remote Control . . . 49
- 3.6 Coordinate Mode . . . 65
- 3.7 Memory Function . . . 69
- 3.8 GAME - Catching Dolls . . . 73
- 3.9 GAME - Iron Collection . . . 81

4 Play with Python **89**
- 4.1 Quick Guide on Python . . . 89
- 4.2 Test 3 EoATs . . . 110
- 4.3 Sound Effects . . . 112
- 4.4 Dual Joystick Module Control . . . 114
- 4.5 Keyboard Control . . . 121
- 4.6 Coordinate Mode . . . 129
- 4.7 Memory function . . . 132
- 4.8 GAME - Catching Dolls . . . 135
- 4.9 GAME - Iron Collection . . . 138

5 Appendix **143**
- 5.1 I2C Configuration . . . 143
- 5.2 Remote Desktop . . . 145
- 5.3 About the Battery . . . 154

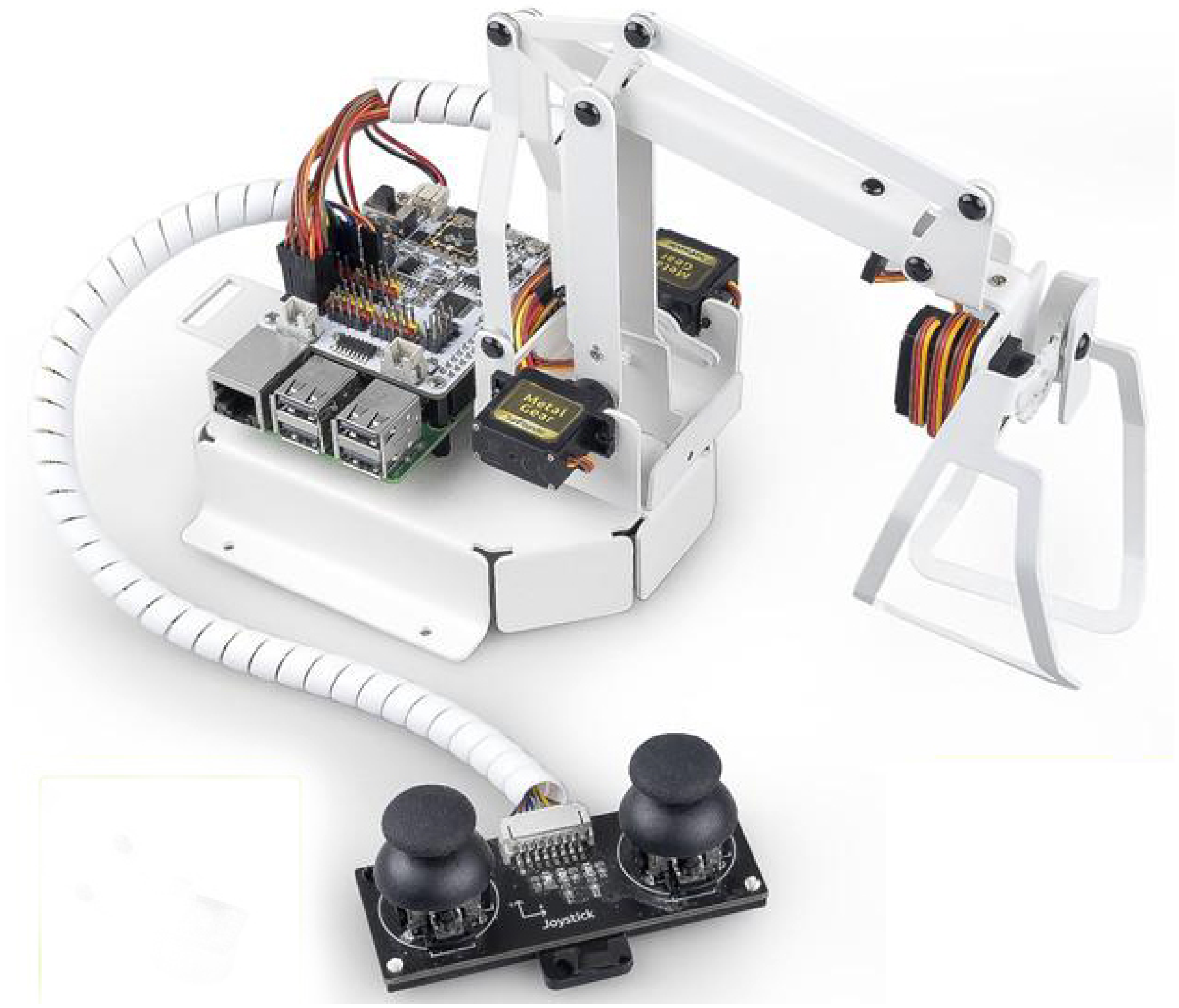
6 Thank You **157**

7 Copyright Notice **159**

Thank you for choosing our PiArm.

PiArm is a three-degree-of-freedom robotic arm for Raspberry Pi. It has 3 interchangeable parts - bucket, hanging clip and solenoid - to help you perform different tasks.

In addition, PiArm offers both remote control and built-in dual joystick module control.



This tutorial includes several parts: device list, assembly guide, programming and appendices. The programming section is divided into two chapters: Playing in Ezblock and Playing in Python, each of which allows you to make PiArm work the way you want it to.

- *Play with Ezblock*

If you are new to programming, check out this chapter as it introduces Ezblock Studio, a block-based visual programming software that allows you to make PiArm move and implement some interesting projects by simply dragging and dropping blocks.

- *Play with Python*

If you prefer to program in a more popular programming language - python, you can refer to this section. The chapter covers starting from burning the Raspberry Pi OS, to configuring the Raspberry Pi and finally getting the code running to see the effects, even if you don't have any Python foundation, you can get PiArm working quickly.

CHAPTER

ONE

COMPONENT LIST AND ASSEMBLY INSTRUCTIONS

You need to check whether there are missing or damaged components according to the list first. If there are any problems, please contact us and we will solve them as soon as possible.

Please follow the steps on the PDF to assemble.

Note:

1. Before assembling, you need to buy 2 18650 batteries and fully charge them, refer to *About the Battery*.
2. Robot HAT cannot charge the battery, so you need to buy a battery charger at the same time.

- `PiArm Assembly Instructions (.pdf)`

Warning: If the kit you received includes a clear Robot HAT Case, please do not mount it so as not to affect the PiArm left and right rotation range.

CHAPTER

TWO

HARDWARE INTRODUCTION

2.1 Arm

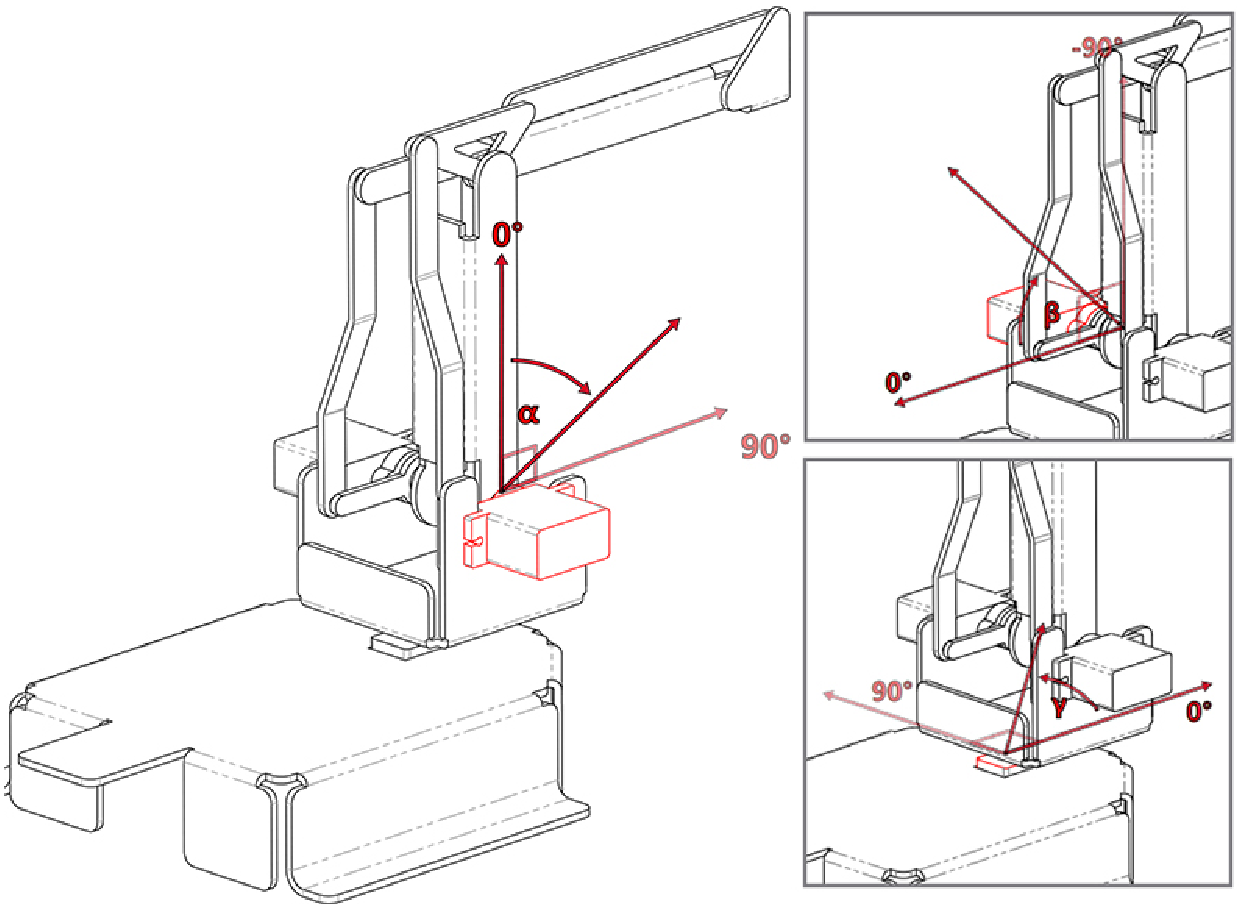
PiArm's arm can be controlled in two ways: *Angle Mode* and *Coordinate Mode*.

- *Angle Mode*: Writes a certain angle to the three servos on the arm, thus rotating the arm to a specific position.
- *Coordinate Mode*: Create a spatial right-angle coordinate system for the arm and set the control point. Set the coordinates of the control point so that the arm can reach a specific position.

2.1.1 Angle Mode

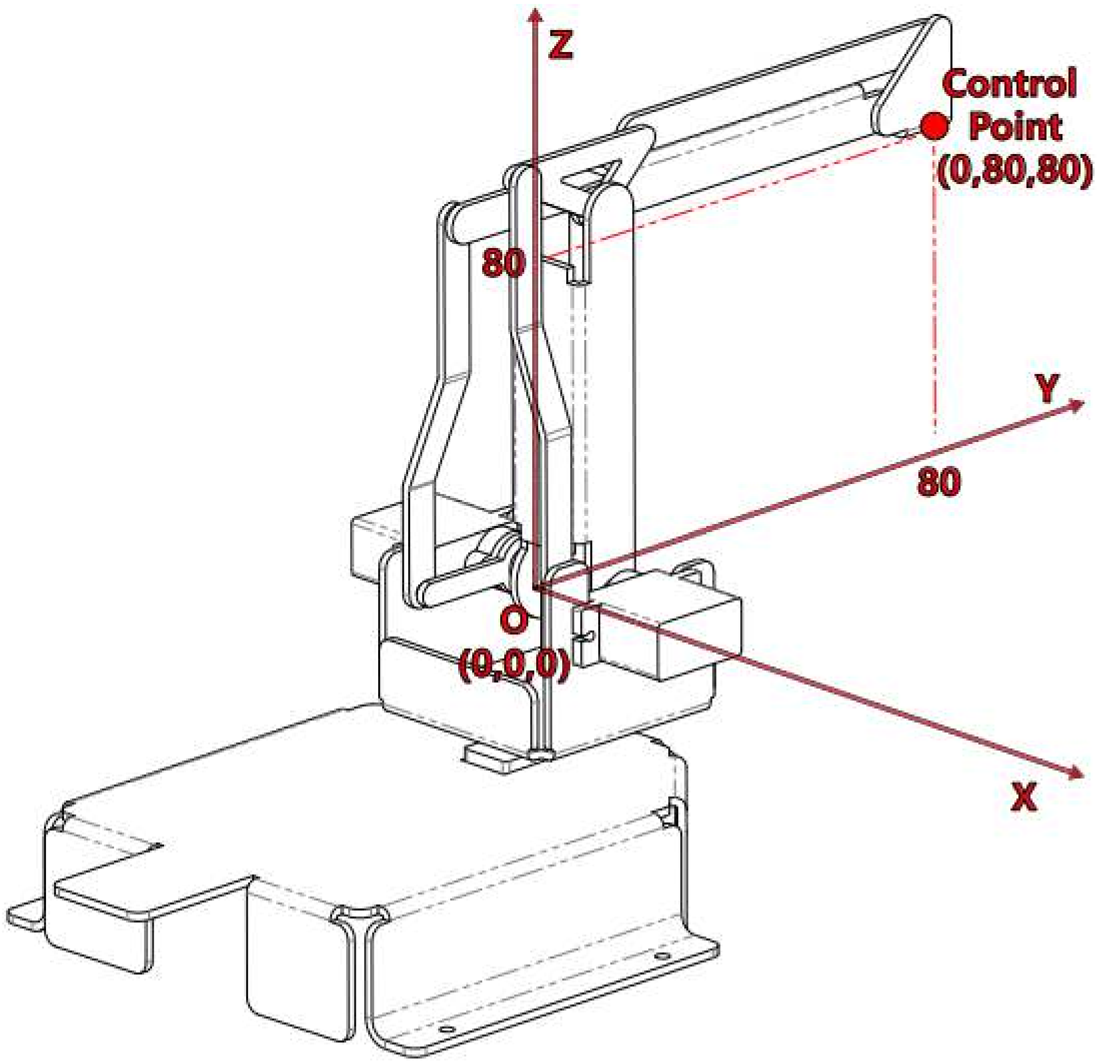
The arm has three servos to control its up and down, left and right, and front and back. We use , and to represent their rotation angles, as shown below.

- `(alpha)`: represents the front-to-back rotation angle of the arm, due to the limitation of the structure, the recommended rotation range is: -30 ~ 60.
- `(beta)`: represents the up and down rotation angle of the arm, due to the limitation of the structure, the recommended rotation range is: -60 ~ 30.
- `(gamma)`: represents the left and right rotation angle of the arm, the range is: -90 ~ 90.



2.1.2 Coordinate Mode

PiArm has a spatial rectangular coordinate system with its origin located at the center of the output axis of the servos on both sides. The Control Point is located at the top of the arm and is scaled in millimeters. In the initial state, the coordinates of the Control Point are (0, 80, 80).



It is important to note that the arm length of PiArm is finite, and if the coordinate values are set beyond the limits of its mechanical motion, PiArm will rotate to an unpredictable position.

In other words, the total arm length of PiArm is 160 mm, which means that the limit value of the control points moving along the Y-axis should be between (0,0,0) and (0,160,0). However, due to the limitations of the structure itself, the range of movement should be much smaller than this.

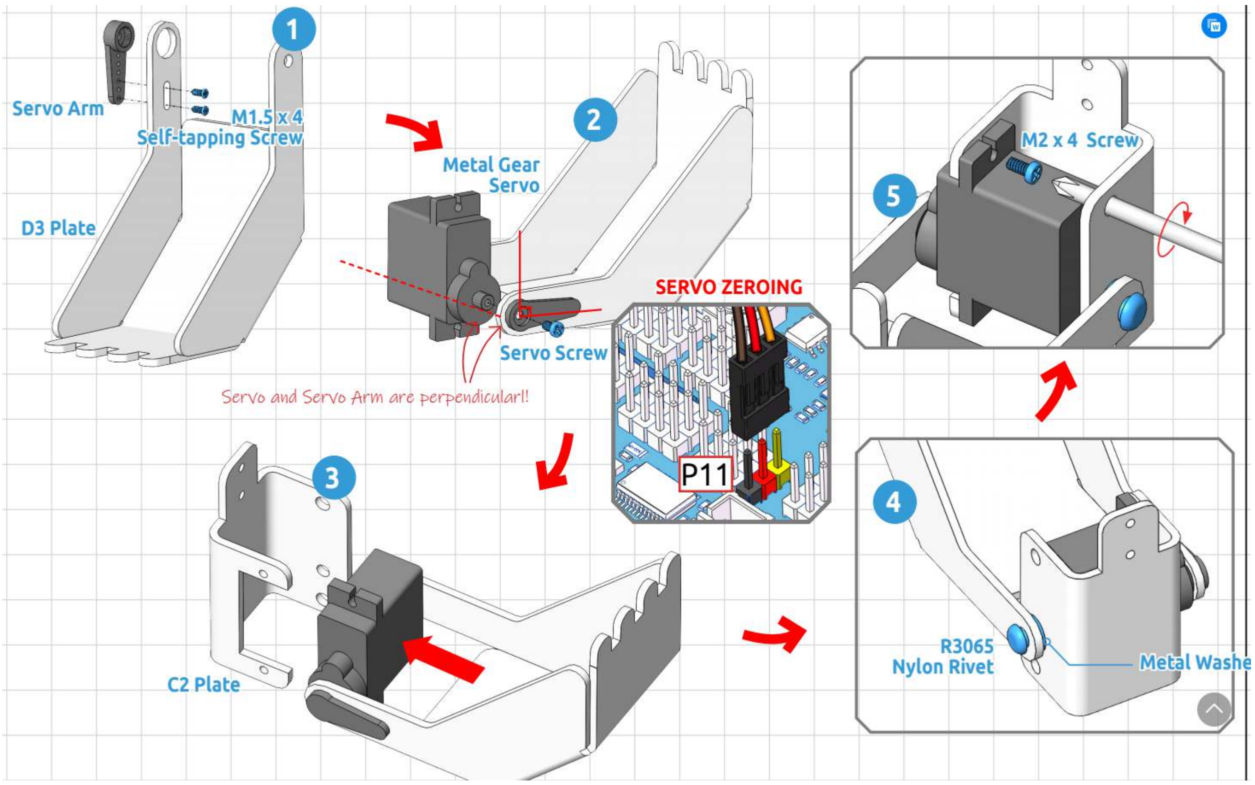
- The recommended range for the X coordinate is -80 ~ 80.
- The recommended range for Y coordinate is 30 ~ 130.
- The recommended range for Z coordinate is 0 ~ 80.

2.2 Shovel Bucket

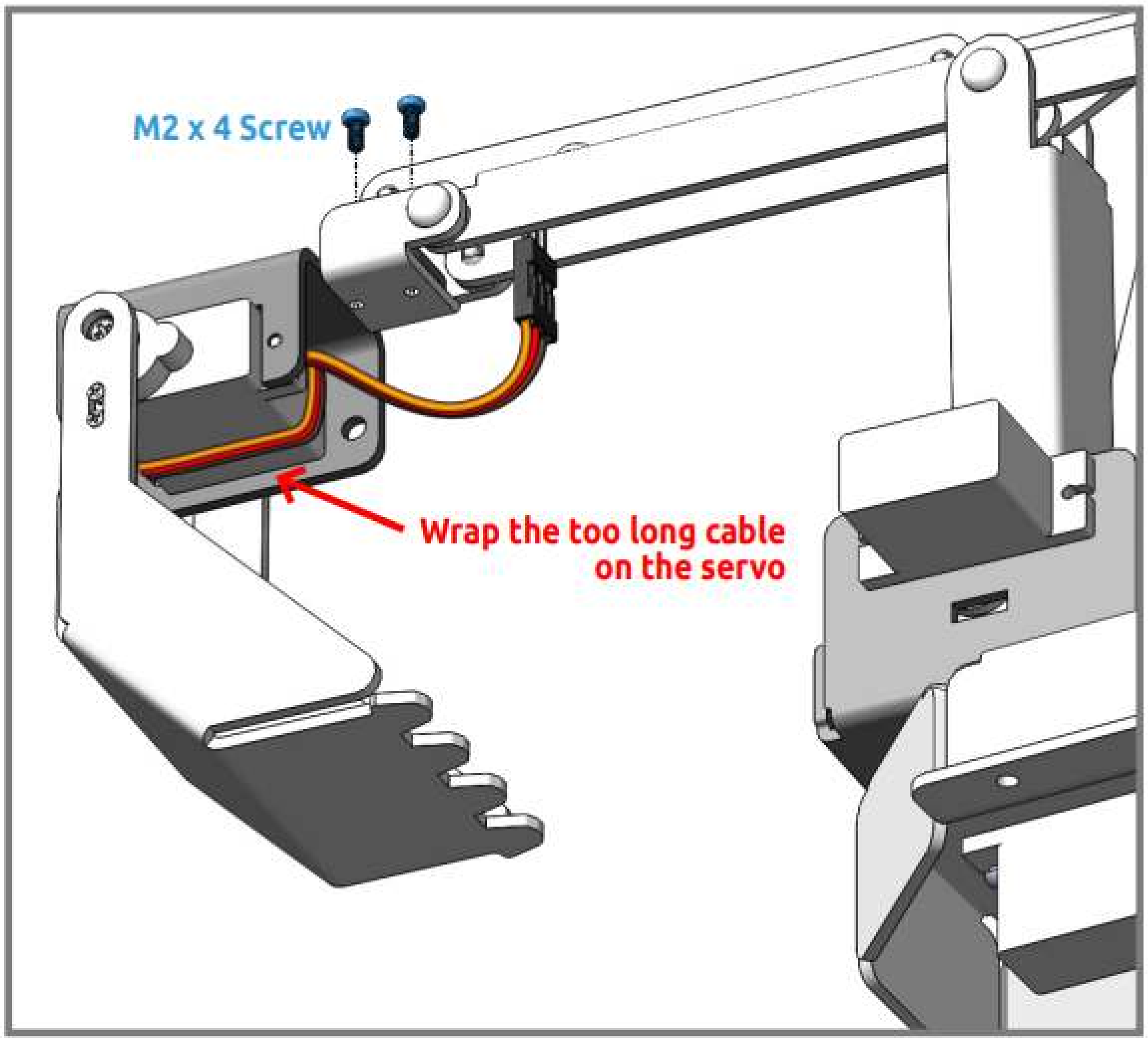
Assembling the Shovel Bucket

Assemble the Shovel Bucket as shown below.

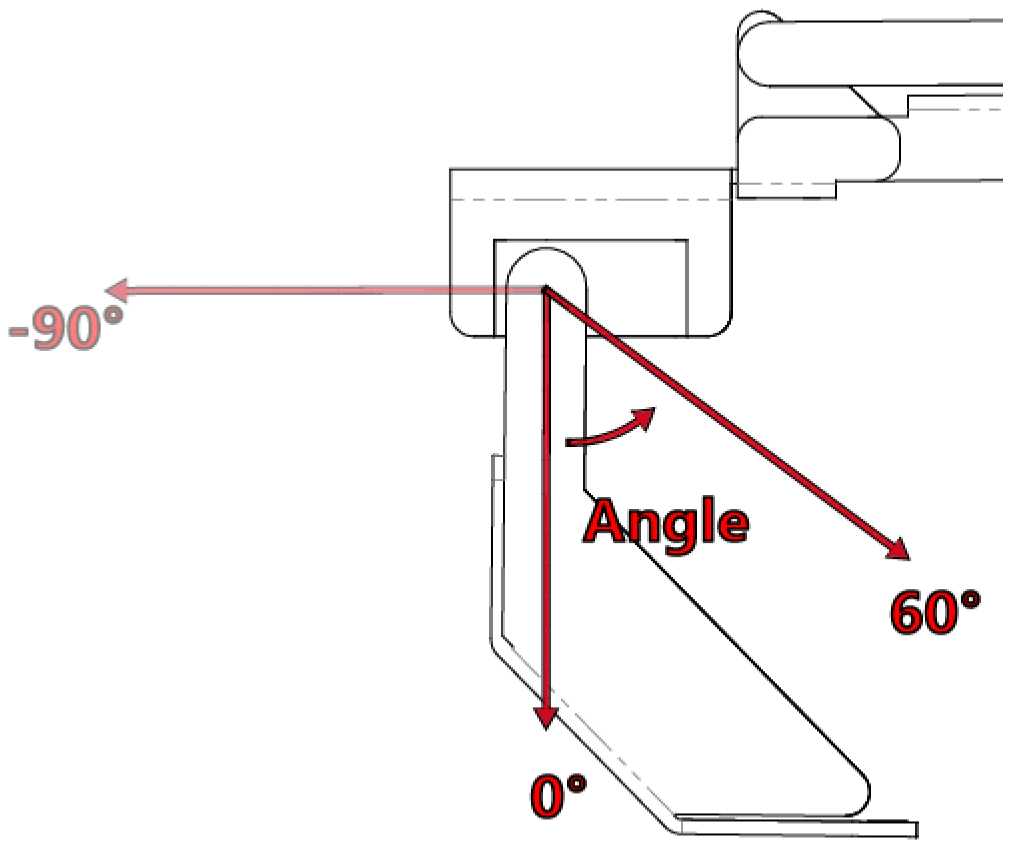
Note: In step 2 you need to insert the servo into P11 for zeroing before inserting the D3 plate into the servo shaft in a vertical orientation.



Assemble the Shovel Bucket to the end of the PiArm with M2x4 screws.



The Shovel Bucket has a rotation range of -90 ~ 60.



Use range

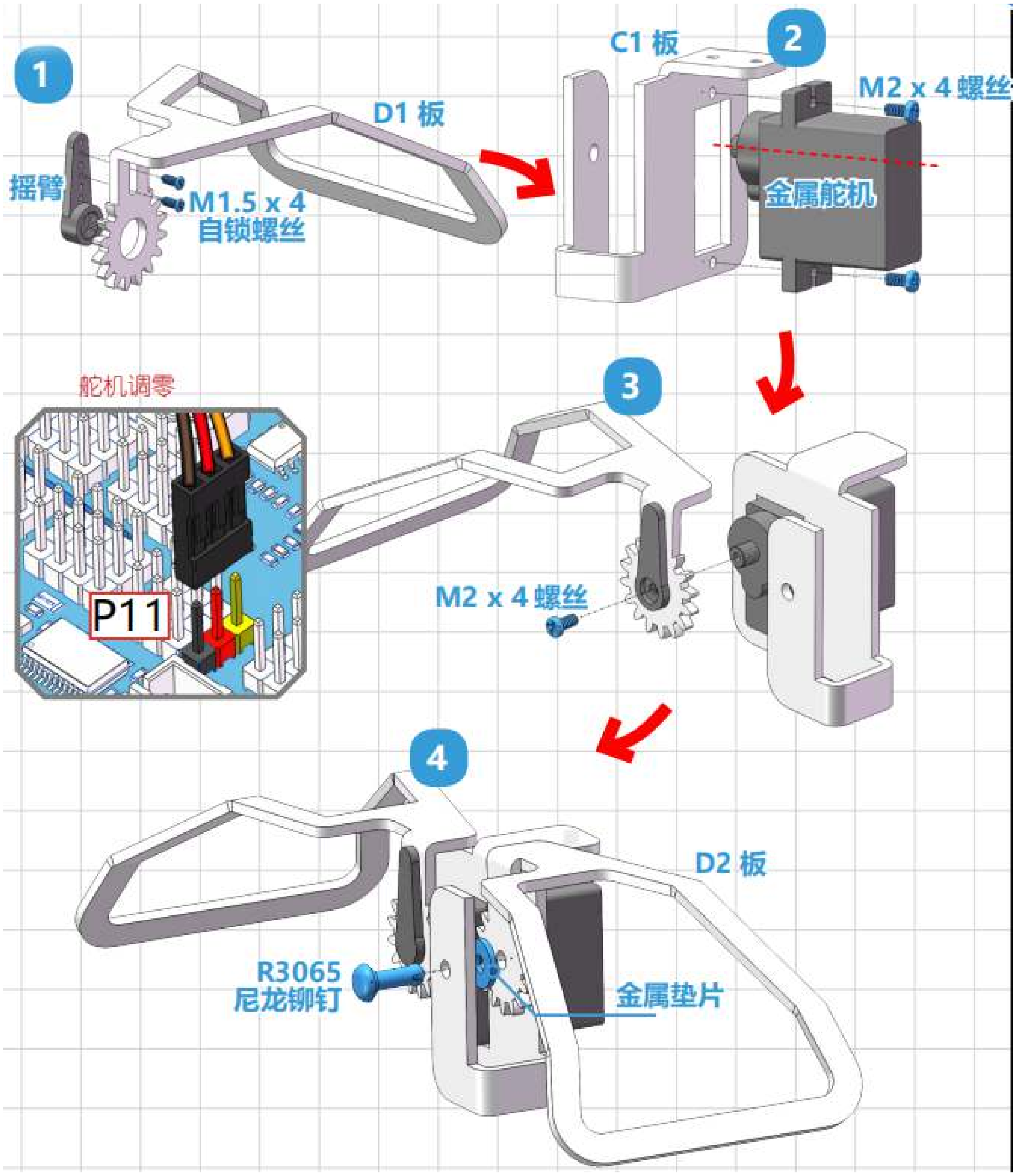
Can’t dig water, can be used to dig sand and gravel.

2.3 Hanging Clip

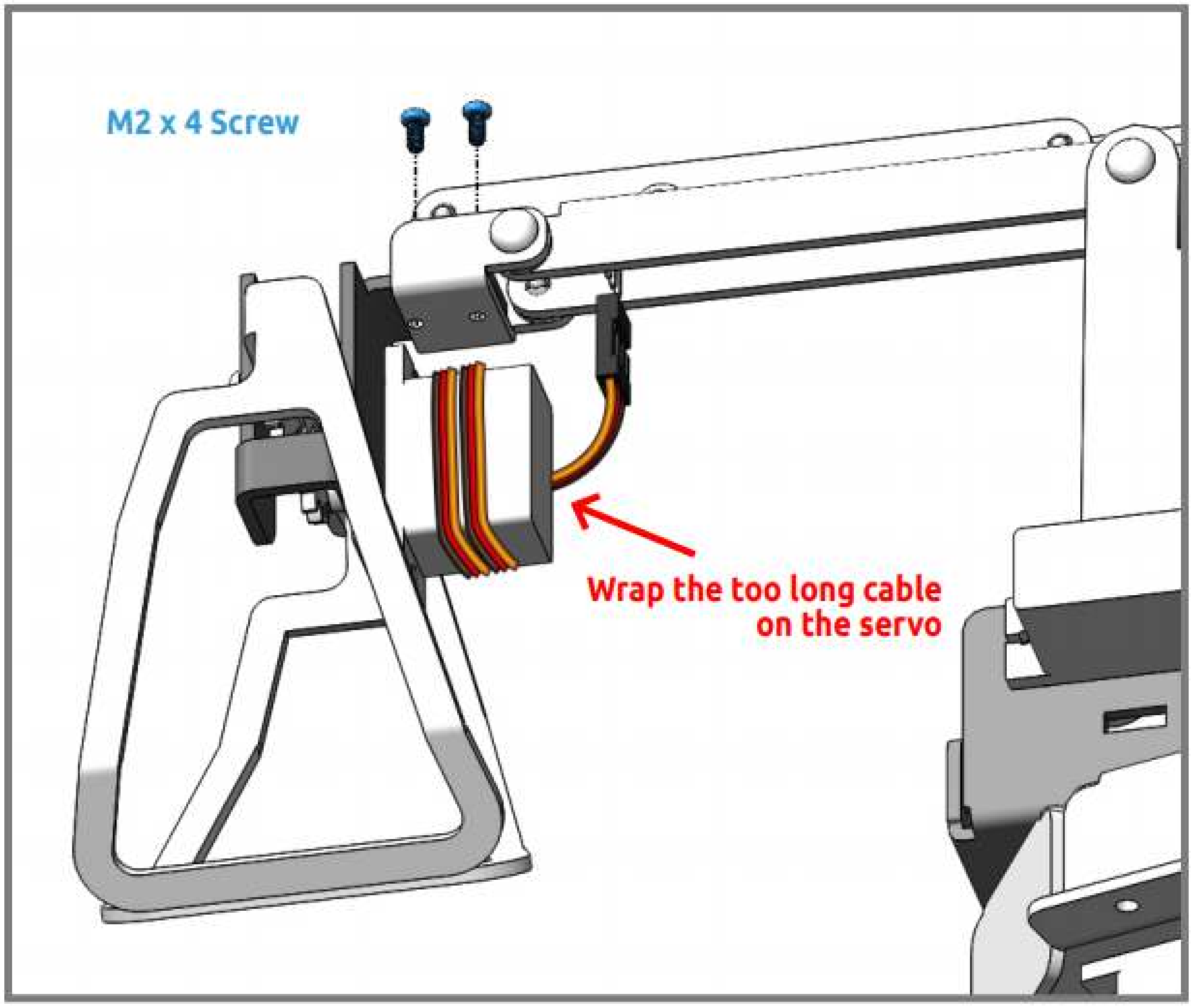
Assembly

Assemble the Hanging Clip as shown below.

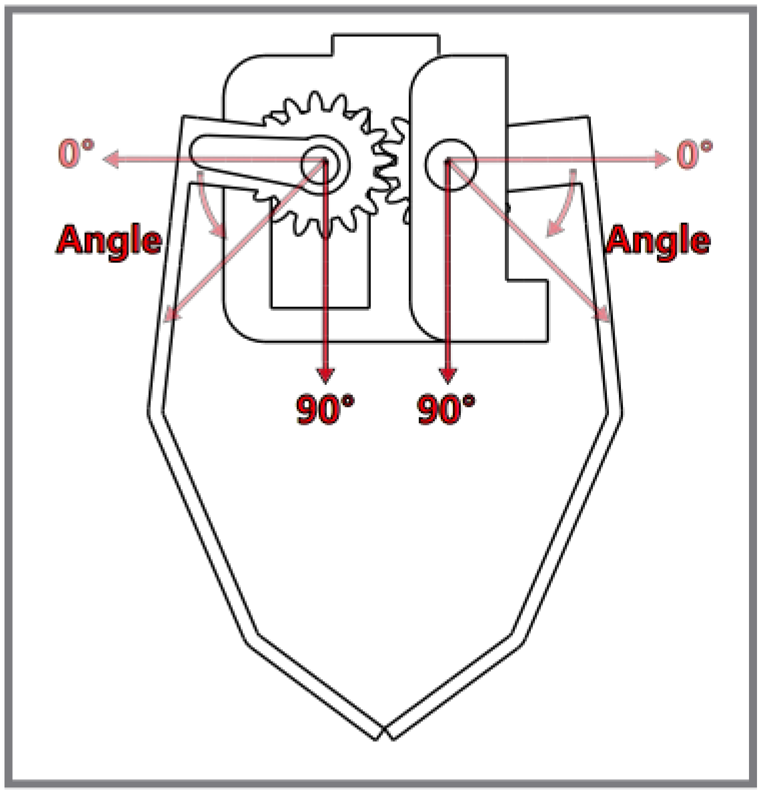
Note: Note that in step 3 you need to insert the servo into the P11 for zeroing before inserting the D1 plate into the servo shaft in a vertical orientation.



Attach the Hanging Clip to the end of the PiArm with the M2x4 screw.



The angle range of the Hanging Clip is 0-90°.



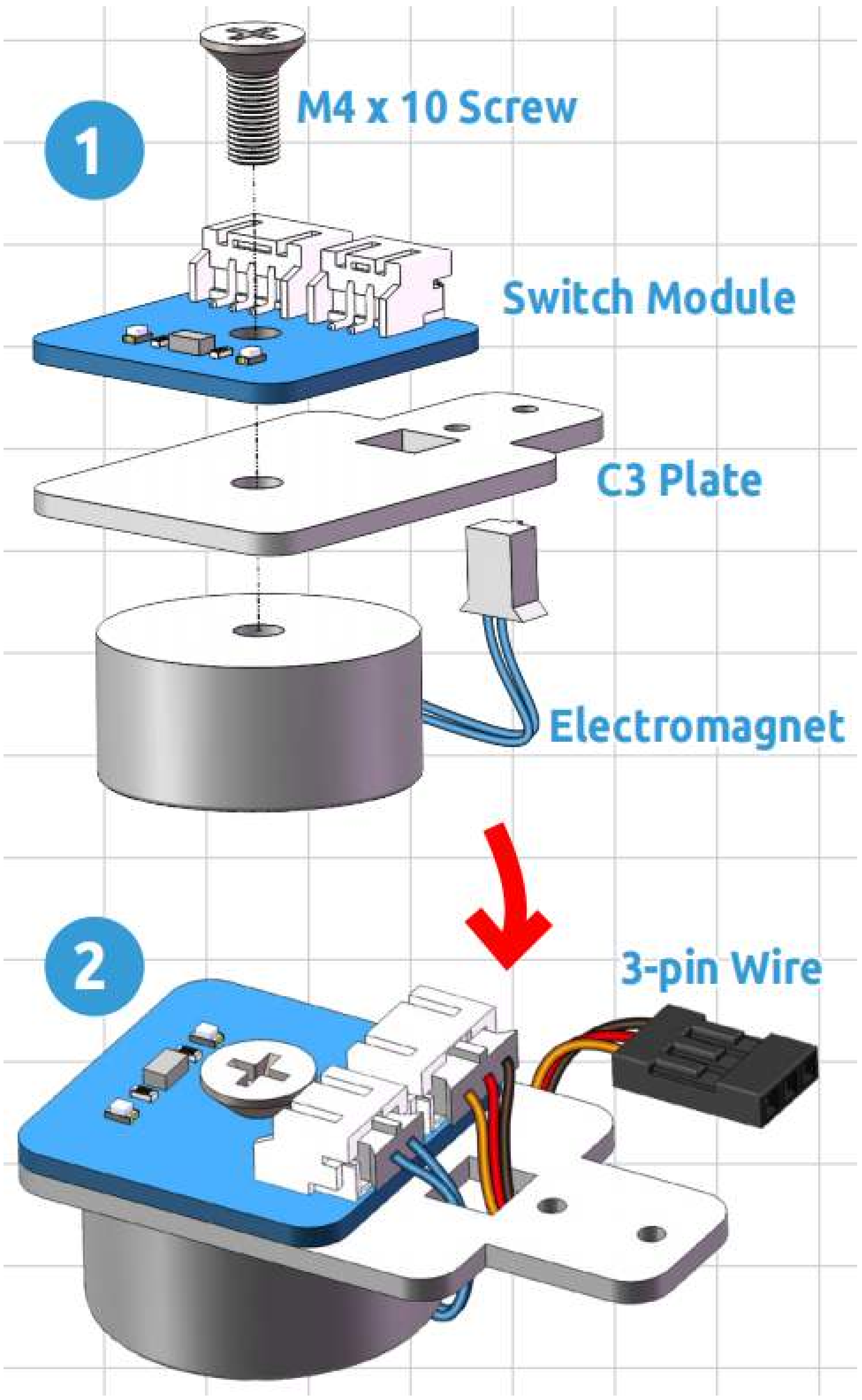
Using range

- The weight of the clamped object should be less than 150g.
- The recommended height of the object to be clamped should be less than 4cm, width less than 8.5cm.
- Slender objects need to find the right angle to clip up.

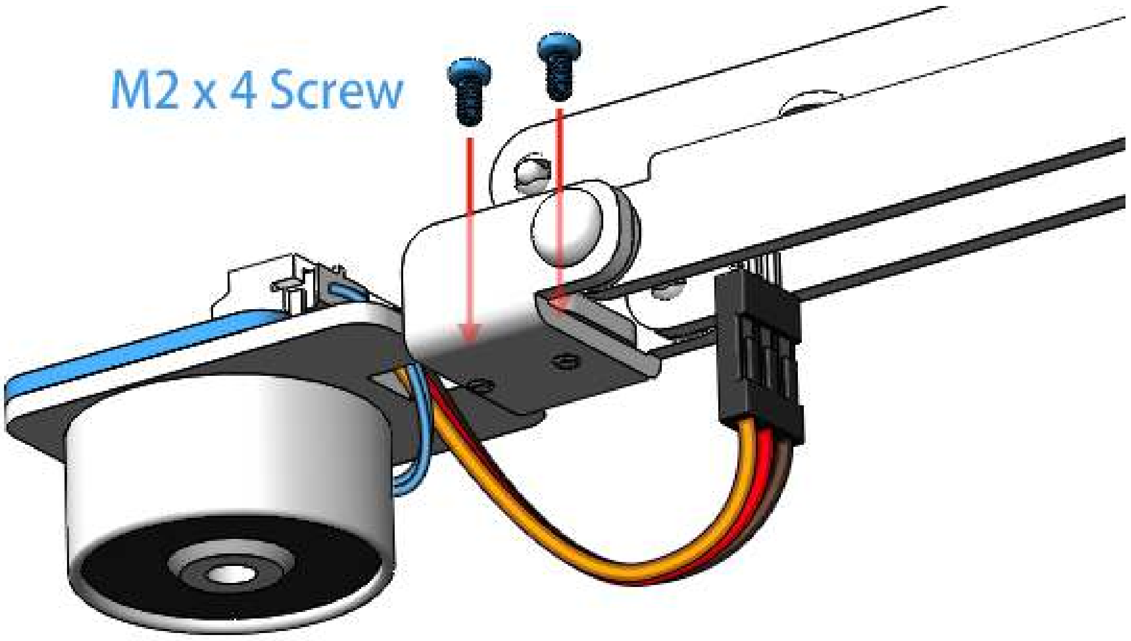
2.4 Electromagnet

Assembly

Assemble the electromagnet module according to the diagram below.



Then secure the electromagnet to the end of the **PiArm** with the M2x4 screws.

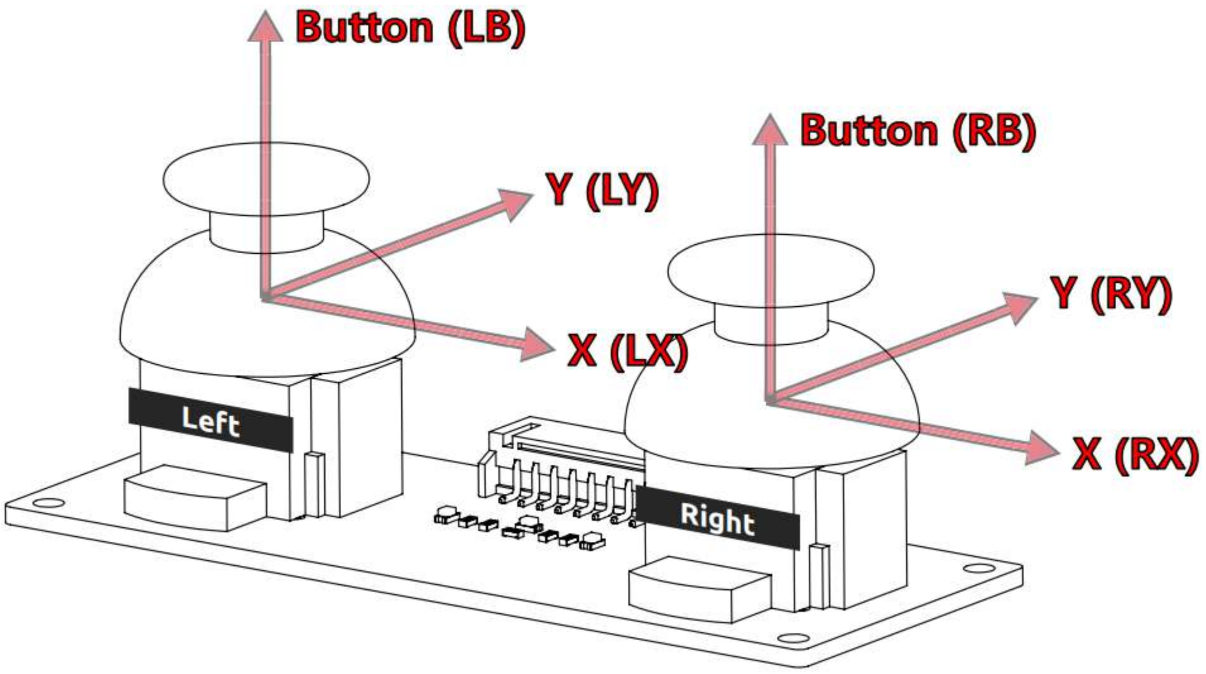


Range of use

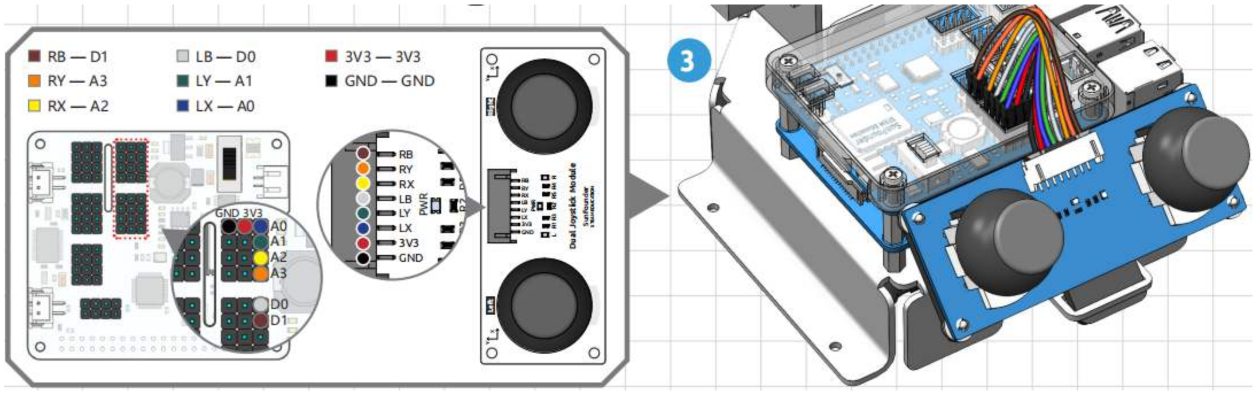
- Can only be used to suck ferrous products.
- The larger the surface area of the iron product, the stronger the adsorption.
- It is recommended that the weight of iron objects is less than 150g.

2.5 Dual Joystick Module

Dual joystick module, as the name implies, consists of two joysticks, each of which can output electrical signals in X, Y and Z directions.

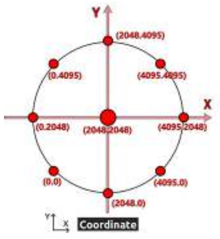


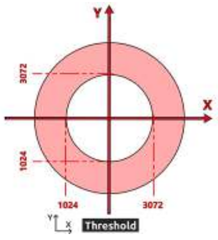
Before you can use the dual joystick module, you need to connect its 8 wires to the corresponding pins of the Robot HAT as shown in the picture below.



The joystick reads in a plane coordinate system from 0 to 4095, with the origin (0,0) in the lower left corner.

As an example, the coordinate value when the joystick is not pushed is (2048,2048). If the joystick is pushed to the left, the coordinates are (0,2048). When the joystick is pushed down, the coordinates are (2048,0), as shown below.

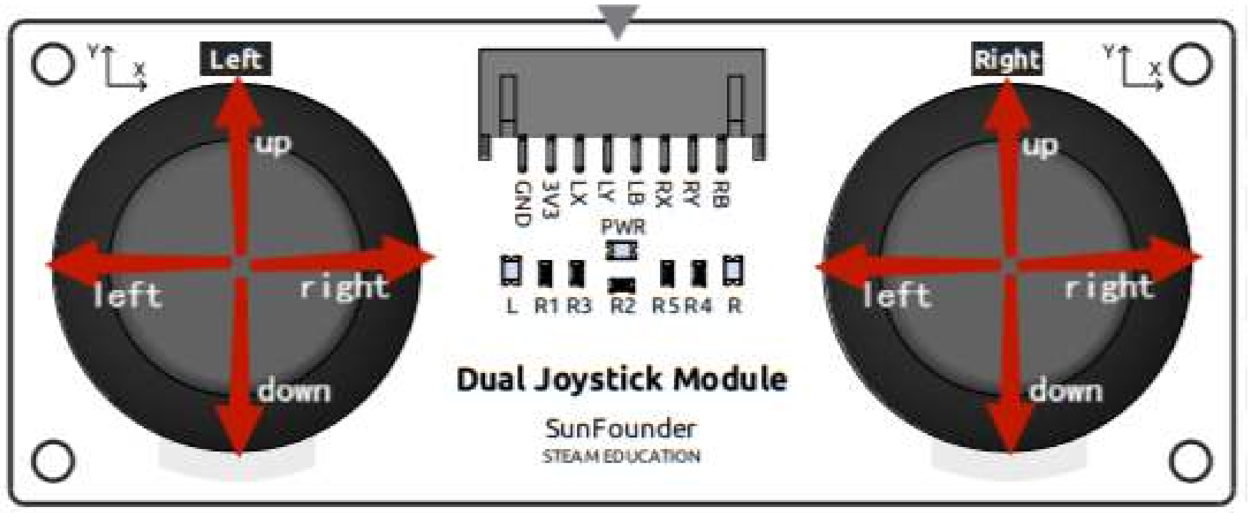




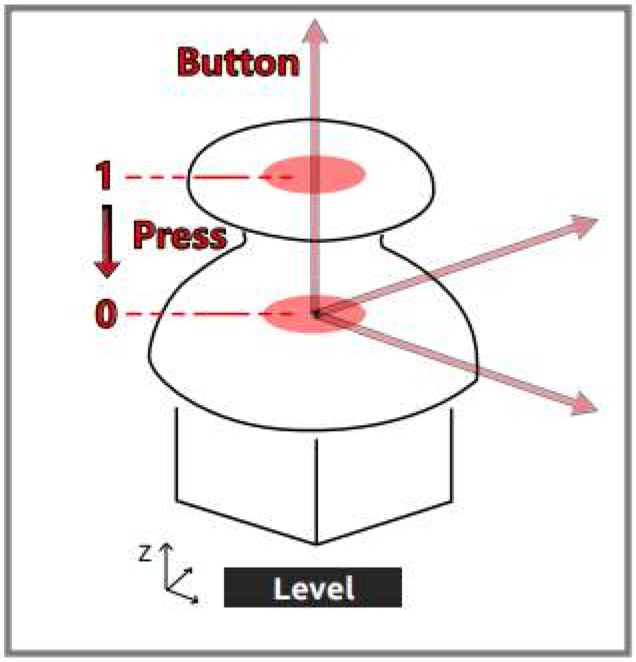
However, electrical signals tend to fluctuate and it is difficult to get an absolutely stable reading, so we usually set a value interval to determine where the joystick is currently located.

The recommended boundary values are set to 3072 and 1024. when the joystick reading is greater than 3072, the joystick is considered to be pushing up (or right); if the reading is less than 1024, the joystick is considered to be pushing down (or left).

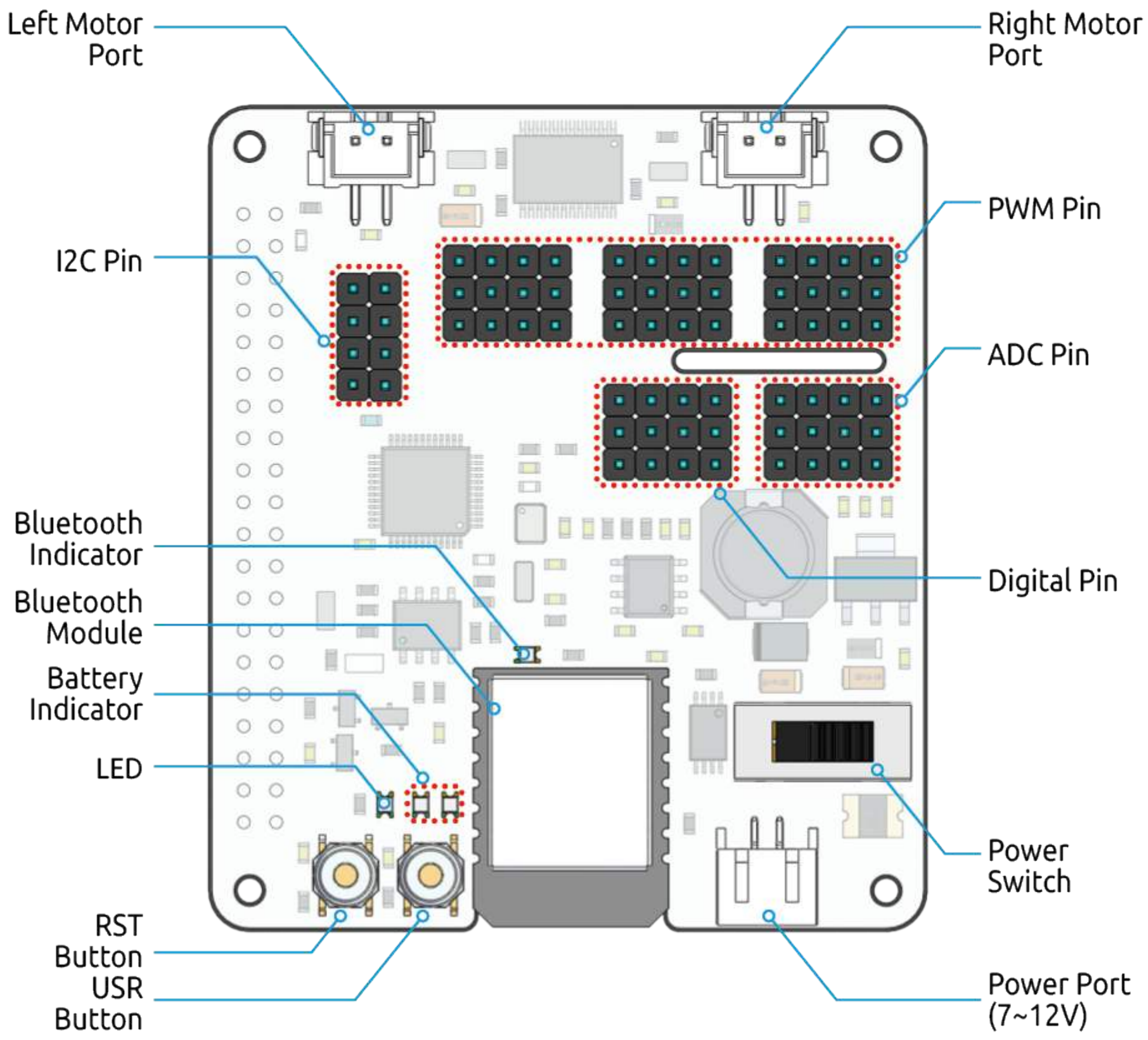
Note: In the Python library, these values have been processed into directional indications as follows.



The Z-axis button outputs a low level (0) when pressed and a high level (1) when released.



2.6 About Robot HAT



RST Button

- A short-press of the RST Button will cause any running programs to reset.
- A long-press of the RST Button until the LED lights up, and then releasing will disconnect the Robot HAT's Bluetooth chip.

USR Button

- The functions of the USR Button can be configured through programming. (Pressing down leads to a input of `0`, and releasing produces a input of `1`)

LED

- Configured through programming (Outputting `1` turns the LED on, Outputting `0` turns the LED off.)

Battery Indicator

- Battery voltage above 7.8V will light up the two indicator LEDs. Battery voltage ranging from 6.7V to 7.8V will only light up one LED, voltage below 6.7V will turn both LEDs off.

Bluetooth Indicator

- The Bluetooth indicator LED will stay on with a solid Bluetooth connection, and blink rapidly during a signal transmission. The LED will blink at 1-second intervals if the Bluetooth is disconnected.

Note: You can see more details in the Robot HAT Documentation.

CHAPTER

THREE

PLAY WITH EZBLOCK

Ezblock is a development platform developed by SunFounder designed for beginners to lower the barriers to getting started with Raspberry Pi. It has two programming languages: Graphical and Python, and available on almost all different types of devices. With Bluetooth and Wi-Fi support, you can download code, remote control a Raspberry Pi, on Ezblock Studio.

3.1 Quick Guide on Ezblock

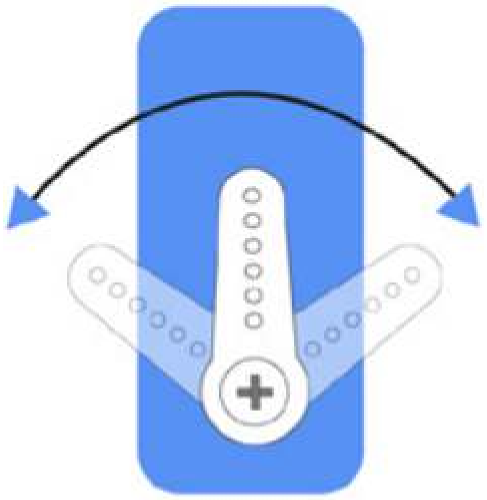
There are 2 parts here:

- *Servo Adjust* allows you to keep all the servos at 0 degrees to complete a proper and safe assembly (otherwise you will probably damage the servos).
- *Install and Configure EzBlock Studio* will guide you to download Ezblock Studio to play with your robot.

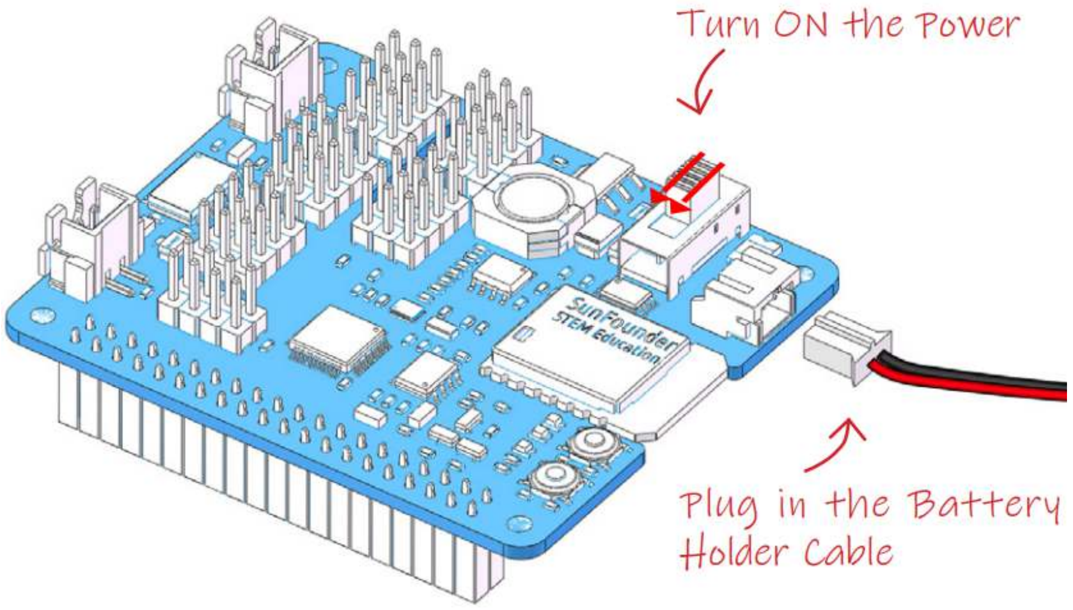
3.1.1 Servo Adjust

When assembling to the part with the servo, you need to keep the servo at 0° and secure it with the servo screw. Please follow the tutorial below to do this.

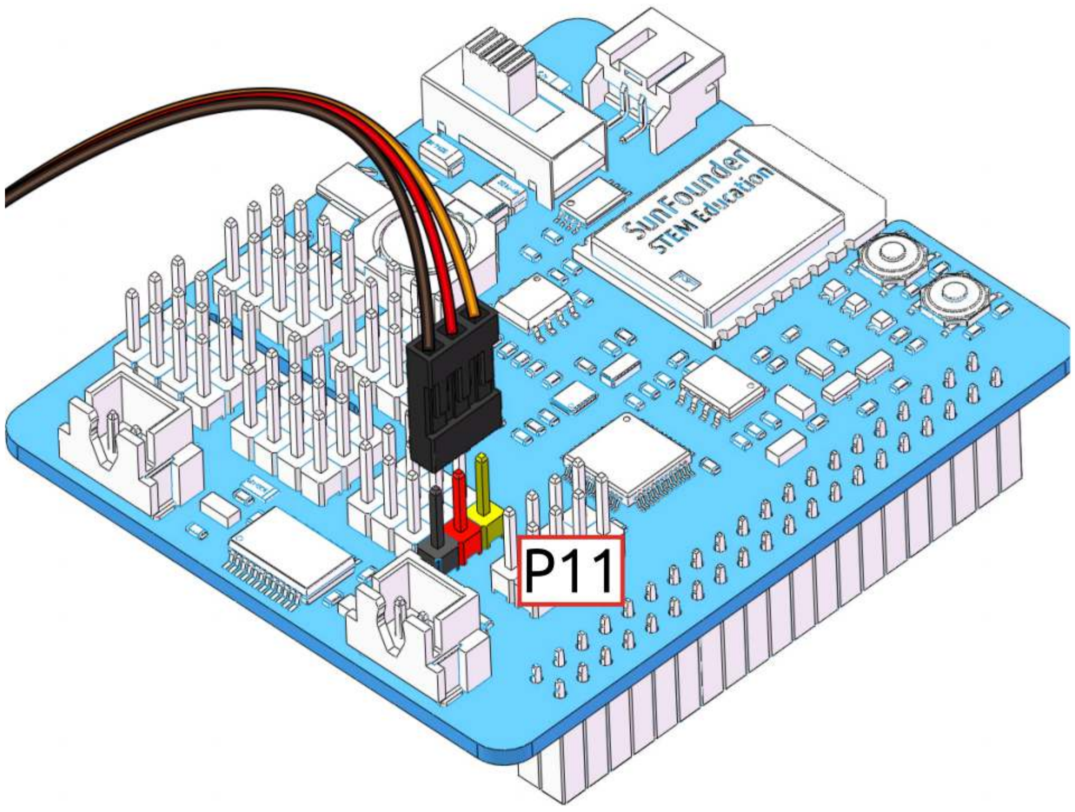
1. Firstly, Install EzBlock OS(3.1) onto a Micro SD card, once the installation is complete, insert it into the Raspberry Pi.
2. To ensure that the servo has been properly set to 0°, first insert the rocker arm into the servo shaft and then gently rotate the rocker arm to a different angle.



3. Follow the instructions on the assembly foldout, insert the battery holder cable and turn the power switch to the ON. Wait for 1-2 minutes, there will be a sound to indicate that the Raspberry Pi boots successfully.



4. Next, plug the servo cable into the P11 port as follows.



5. At this point you will see the servo arm rotate to a specific position (0°). If the servo arm does not return to 0°, press the RST button to restart the Robot HAT.

6. Now you can continue the installation as instructed on the assembly foldout.

Note:

- Do not unplug this servo cable before fastening this servo with the servo screw, you can unplug it after fastening.
- Do not turn the servo while it is powered on to avoid damage; if the servo shaft is inserted at the wrong angle, pull out the servo and reinsert it.
- Before assembling each servo, you need to plug the servo cable into P11 and turn on the power to set its angle to 0°.
- This zeroing function will be disabled if you download a program to the robot later with the EzBlock APP.

3.1.2 Install and Configure EzBlock Studio

As soon as the robot is assembled, you will need to carry out some basic operations.

- Install EzBlock Studio(3.1): Download and install EzBlock Studio on your device or use the web-based version.
- Connect the Product and EzBlock(3.1): Configure Wi-Fi, Bluetooth and calibrate before use.
- Open and Run Examples(3.1): View or run the related example directly.

Projects

Here, we show you the projects of playing Piarm on Ezblock Studio. If you are new to these, you can refer to the code images inside each project to program, and can learn the use of blocks according to TIPS.

If you don't want to write these projects one by one, we have uploaded them to Ezblock Studio's Examples page and you can run them directly or edit them and run them later.

3.2 Test 3 EoATs

This is the first program and the one you must see.

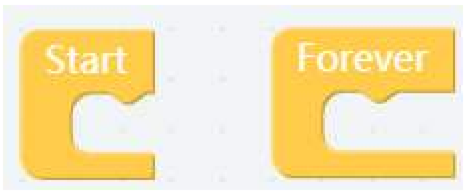
In this project, you will learn how to assemble and use PiArm's 3 End of Arm Tooling (EoAT, replaced by this abbreviation later.).

Before programming, you need to learn the basic usage of Ezblock Studio from here.

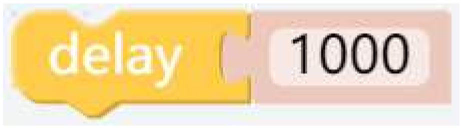
- How to Create a New Project?

3.2.1 Tips on basic blocks

- This is the basic structure of the program, the [Start] block is used to do some initialization (even if no block is placed, it cannot be deleted) and the [Forever] block is, as the name suggests, a continuous loop that allows your program to change and respond.

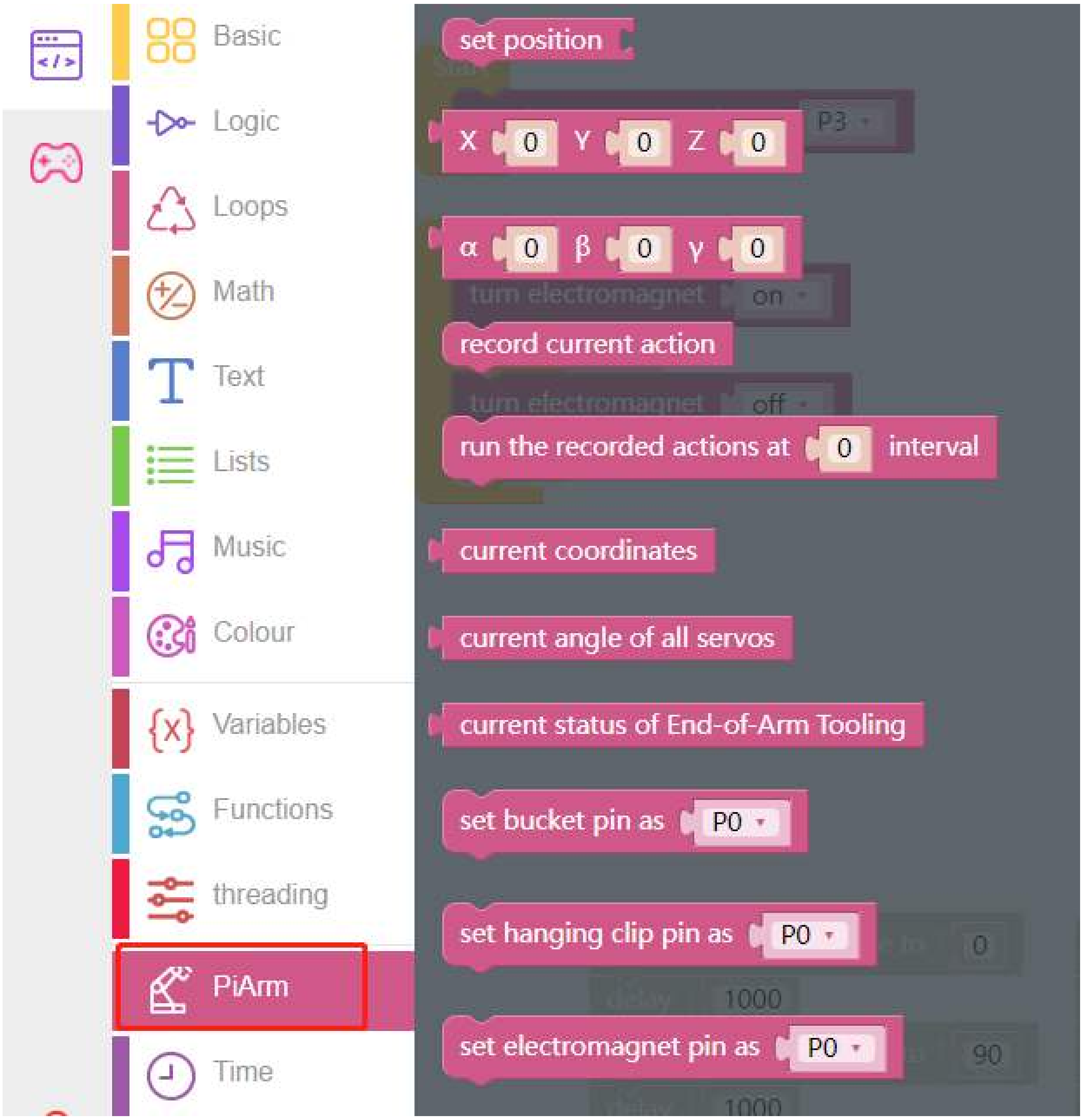


- This block is used to set an interval time in milliseconds.



3.2.2 Tips on PiArm blocks

Here you can find some blocks needed to make PiArm work.



3.2.3 Shovel Bucket

Step 1

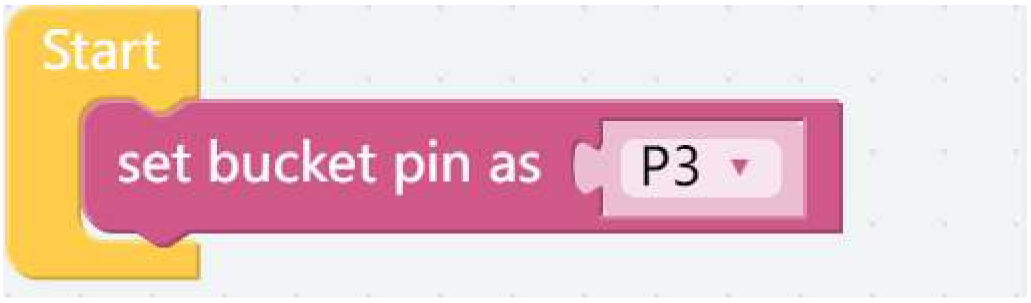
Assemble the *Shovel Bucket* to the end of **PiArm**.

Step 2

Now start writing the code to make Shovel Bucket work.

Put [set bucket pin as ()] in the [Start] block to initialize the bucket pin as P3.

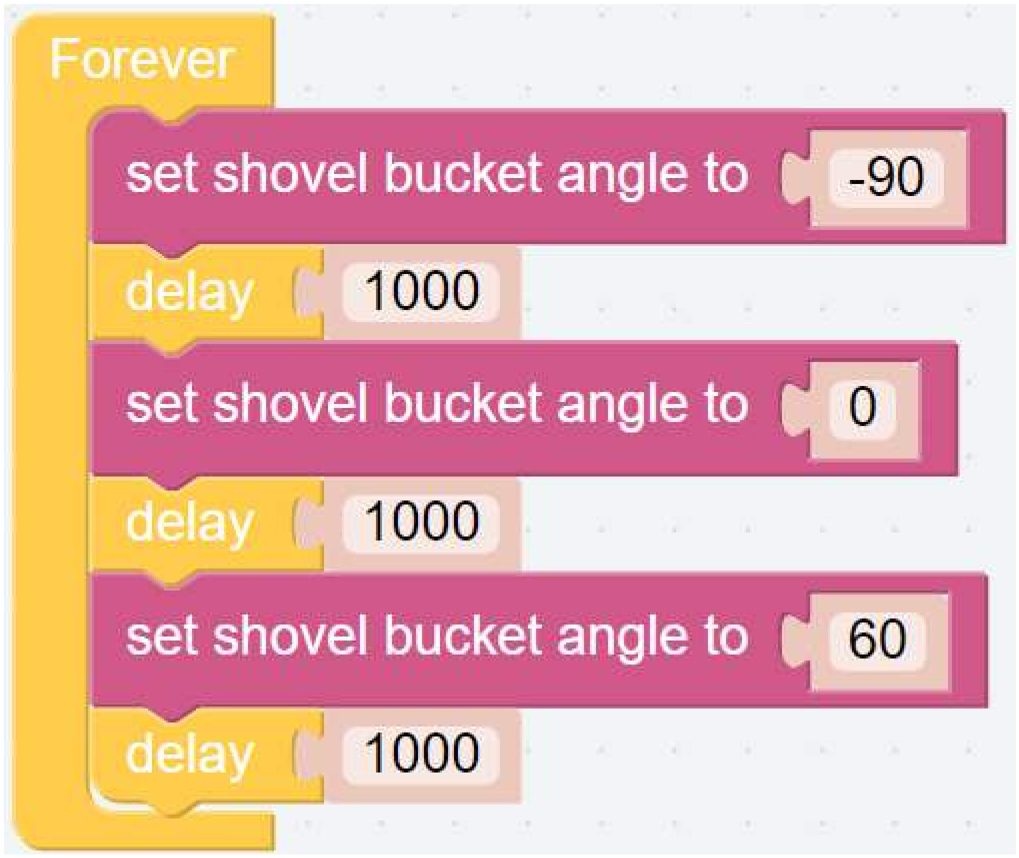
Note: Because in the assembly diagram above, it is connected to the Transfer Module, which is already connected to P3 during the PiArm assembly. Of course you can also connect it to other spare pins.


```
Start
  set bucket pin as P3
```

Step 3

Toggles the angle of the Shovel Bucket between 0° and 90° with an interval of 1s.

- [set shovel bucket angle to ()]: Used to set the angle of Shovel Bucket, the range is 0-90.
- [delay ()]: From the **Basic** category, used to set the time interval between 2 block runs, in: ms.

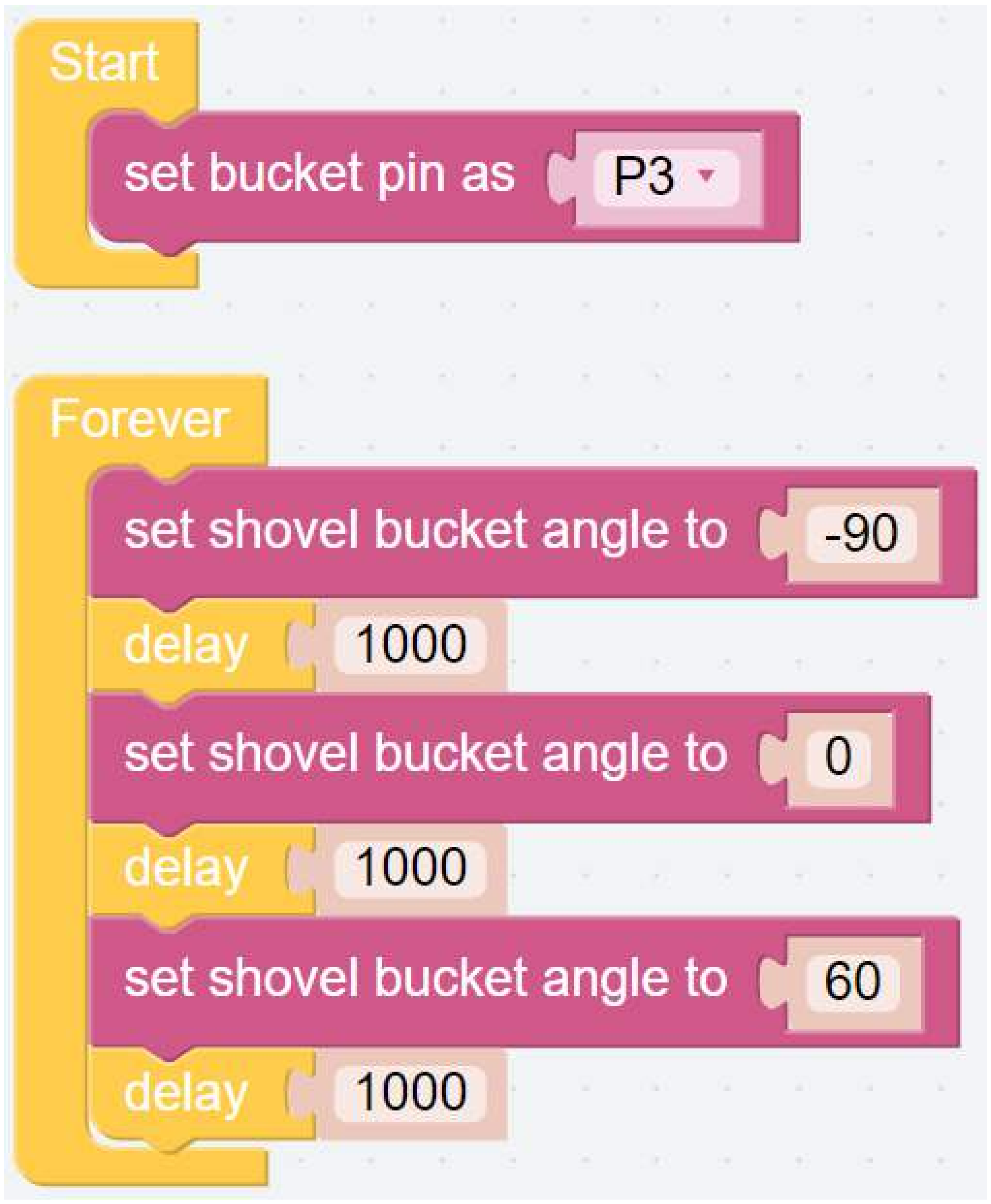


```
Forever
    set shovel bucket angle to -90
    delay 1000
    set shovel bucket angle to 0
    delay 1000
    set shovel bucket angle to 60
    delay 1000
```

Step 4

Once the code is written, click the **Download** button in the bottom right corner to download it to the PiArm.

Now you will see the **Shovel Bucket** moving back and forth, and you can click the **Run** button to stop the code from running.



```
Start
  set bucket pin as  P3

Forever
  set shovel bucket angle to  -90
  delay  1000
  set shovel bucket angle to  0
  delay  1000
  set shovel bucket angle to  60
  delay  1000
```

3.2.4 Hanging Clip

Step 1

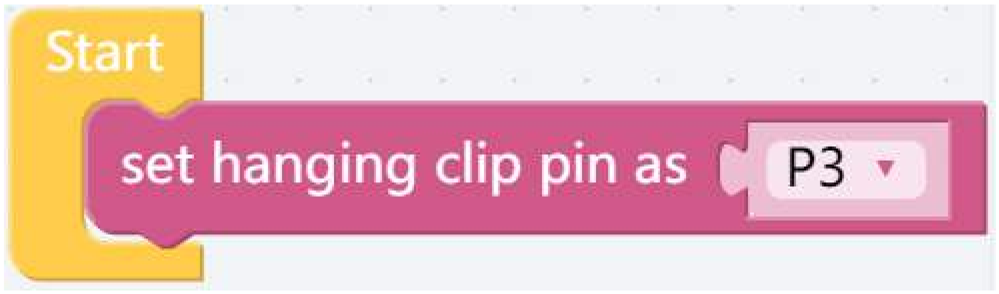
Assemble *Hanging Clip* to the end of **PiArm**.

Step 2

Now start writing the code to make Shovel Bucket work.

Put [set hanging clip pin as ()] in the [Start] block to initialize the hanging clip pin as P3.

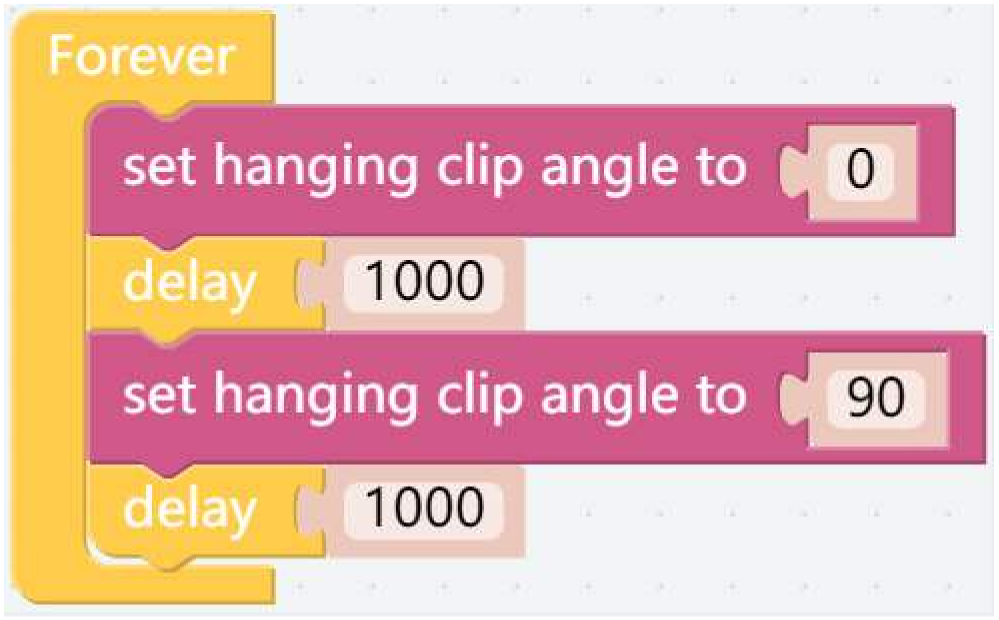
Note: Because in the assembly diagram above, it is connected to the Transfer Module, which is already connected to P3 during the PiArm assembly. Of course you can also connect it to other spare pins.



Step 3

Toggles the angle of the **Hanging Clip** between 0° and 90° with an interval of 1s.

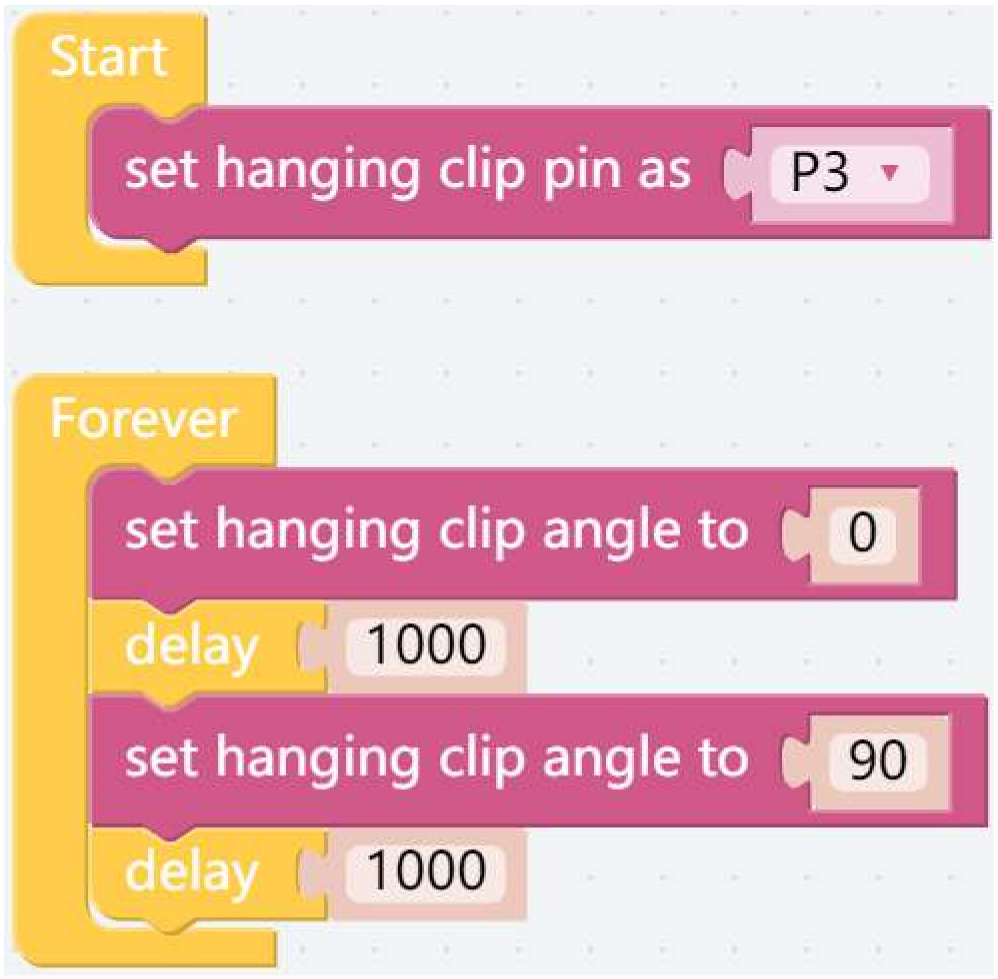
- [set hanging clip angle to ()]: Used to set the angle of **Hanging Clip**, the range is 0-90.
- [delay ()]: From the **Basic** category, used to set the time interval between 2 block runs, in: ms.



Step 4

Once the code is written, click the **Download** button in the bottom right corner to download it to the PiArm.

Now you will see the **Hanging Clip** repeatedly open/close, and you can click the **Run** button to stop the code from running.



```
Start
  set hanging clip pin as [P3 ▾]
Forever
  set hanging clip angle to (0)
  delay (1000)
  set hanging clip angle to (90)
  delay (1000)
```

3.2.5 Electromagnet

Step 1

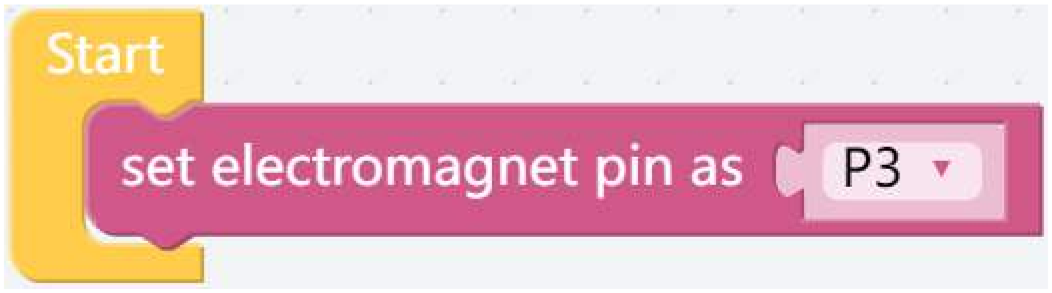
Assemble *Electromagnet* to the end of **PiArm**.

Step 2

Now start writing the code to make Shovel Bucket work.

Put [set electromagnet pin as ()] in the [Start] block to initialize the electromagnet pin as P3.

Note: Because in the assembly diagram above, it is connected to the Transfer Module, which is already connected to P3 during the PiArm assembly. Of course you can also connect it to other spare pins.

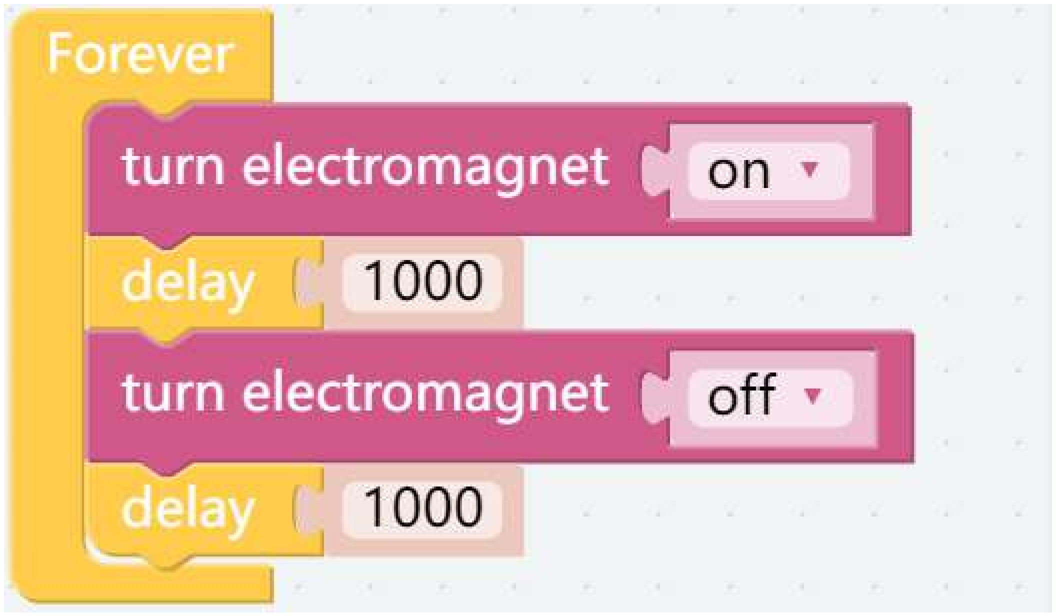


Start
set electromagnet pin as P3

Step 3

Let the electromagnet be repeatedly energized and de-energized at 1 second intervals.

- [turn electromagnet (on/off)]: Used to energize (on) or de-energize (off) the Electromagnet.
- [delay ()]: From the **Basic** category, used to set the time interval between 2 block runs, in: ms.

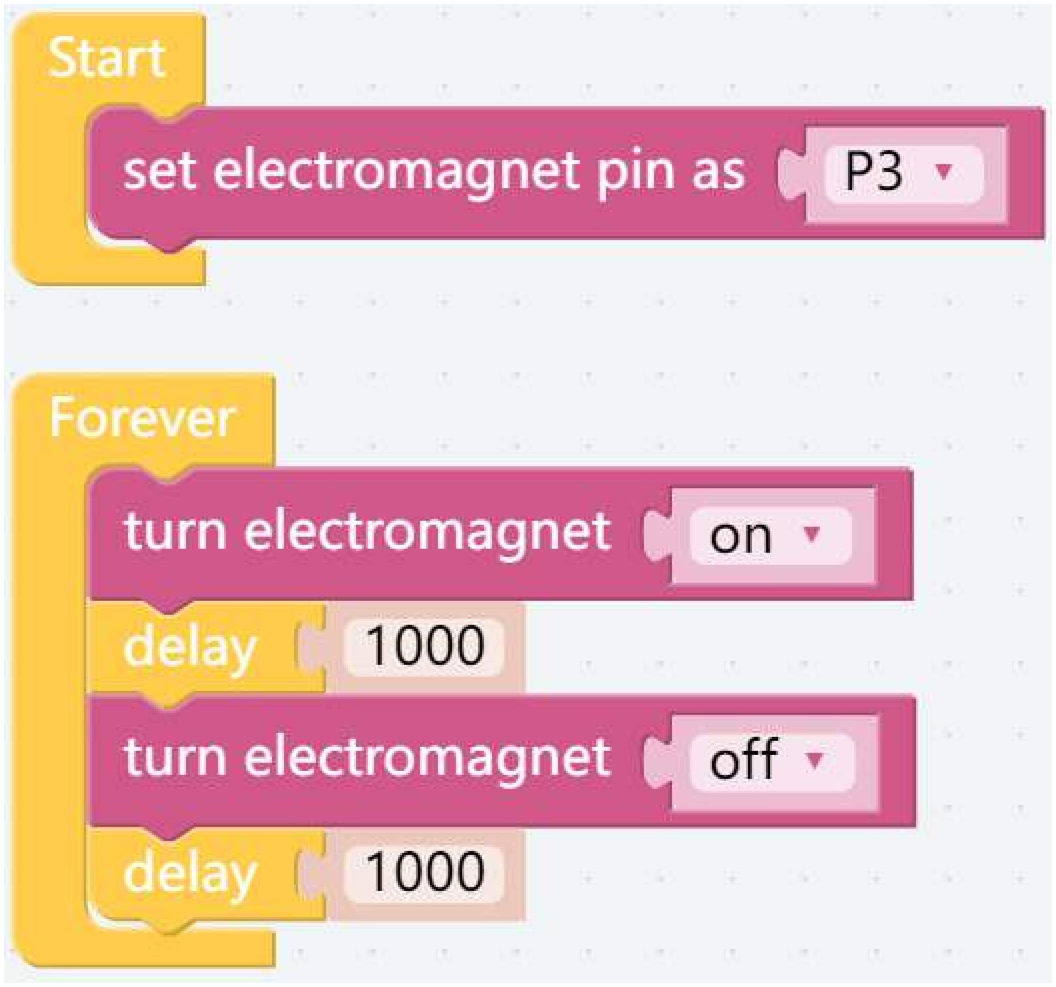


Forever
turn electromagnet on
delay 1000
turn electromagnet off
delay 1000

Step 4

Once the code is written, click the **Download** button in the bottom right corner to download it to the PiArm.

Now you will find that the **Electromagnet** is energized every second (the LED (D2) on the electromagnet lights up, indicating that it is energized, at which time it can be used to adsorb some materials with iron.).

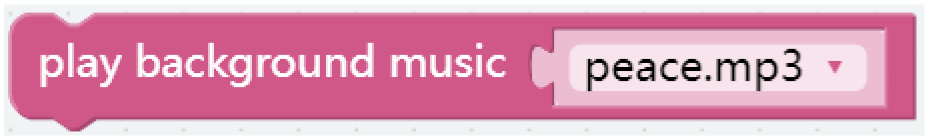


3.3 Sound Effects

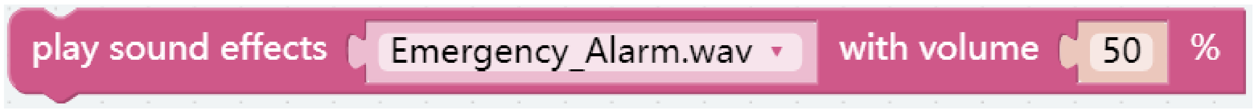
There is a built-in speaker in Robot HAT that can be used to play some music and sound effects, as well as to implement TTS functions.

3.3.1 Tips on Blocks

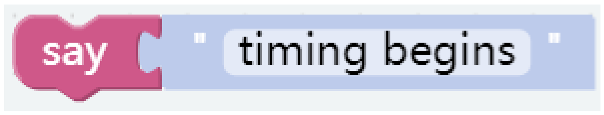
- This block is a separate thread and can play some built-in background music.



- This block can play some built-in sound effects.



- You can write some text in this block and let PiArm speak them.

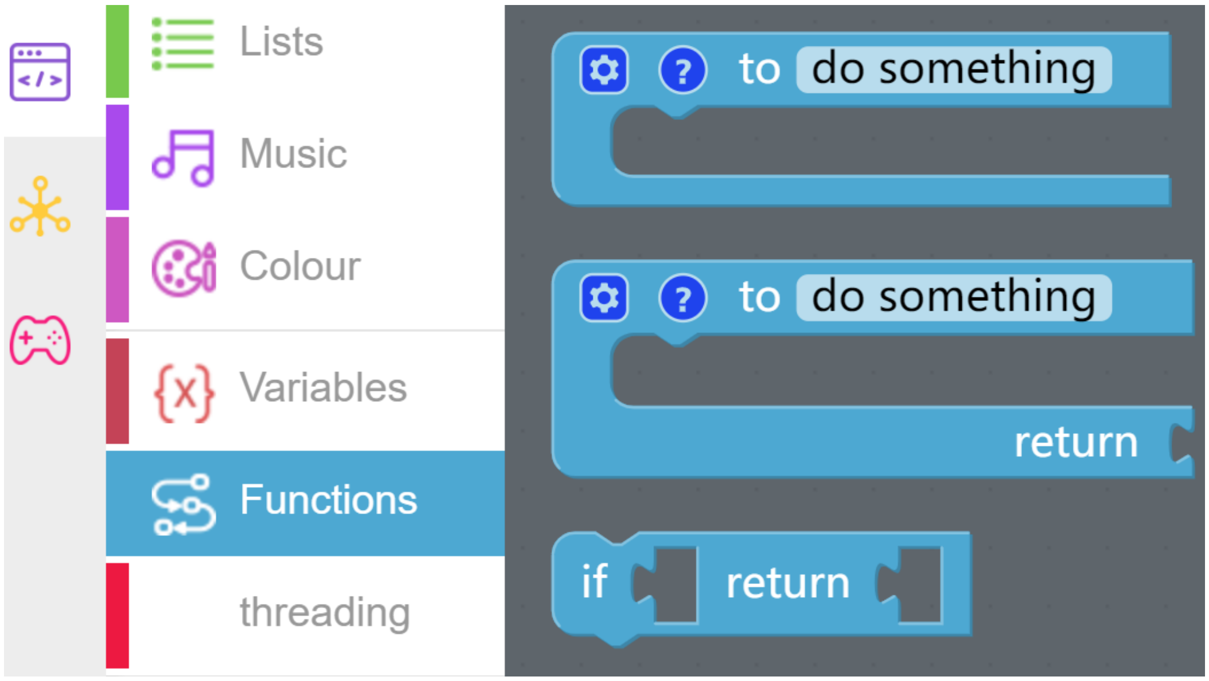


3.3.2 Programming

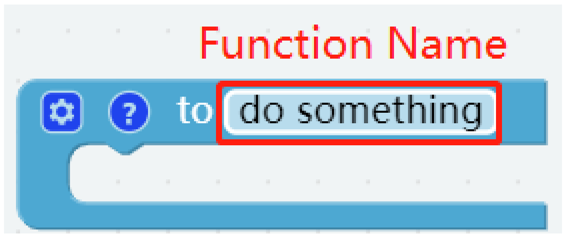
Step 1

You may want to simplify the program with **Functions**, especially when you perform the same operation multiple times. Putting these operations into a newly declared function can greatly facilitate your use.

Click on the **Functions** category and select the appropriate function block, the function you created will also appear here.



The **Function** block without output is used here.

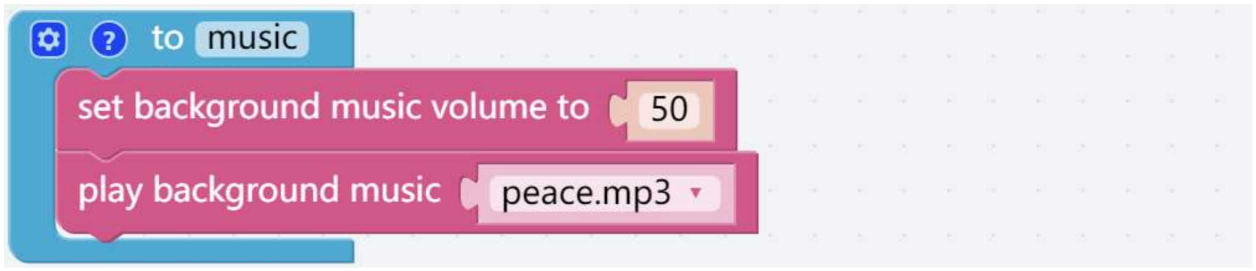


Step 2

Create a function named [music], after creating it you will see it in the **Functions** category.

Now let the [music] function implement playing background music at 50% volume.

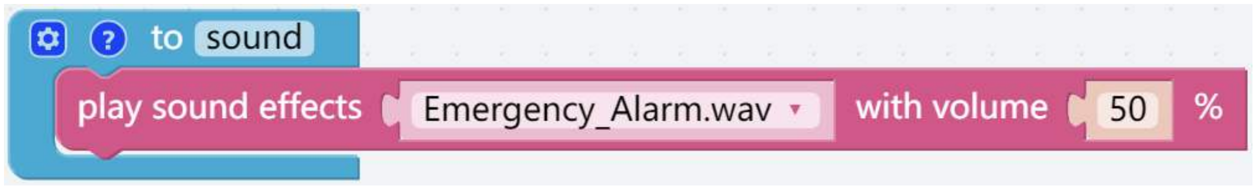
- [set background music volume to ()]: Used to set the volume of the background music, in the range of 0%-100%.
- [play background music ()]: This block is a separate thread and can play some built-in background music.


to music
set background music volume to 50
play background music peace.mp3

Step 3

Create a function named [sound] to make PiArm play a specific sound effect at a certain volume.

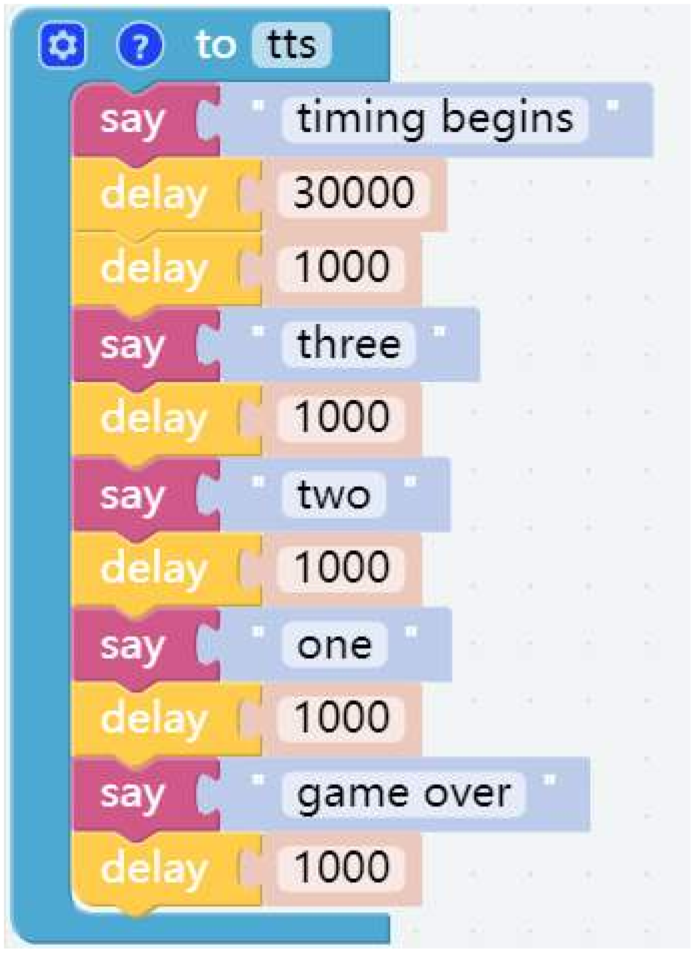
- [play sound effects () with volume to () %]: This block can be used to play built-in sound effects with a volume range of 0%-100%.



Step 4

Similarly create a function named [tts] that will be used to make PiArm say something.

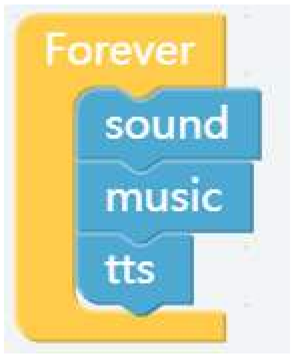
- [say ()]: This block converts the text you type into speech for PiArm to speak.



```
to tts
    say "timing begins"
    delay 30000
    delay 1000
    say "three"
    delay 1000
    say "two"
    delay 1000
    say "one"
    delay 1000
    say "game over"
    delay 1000
```

Step 5

From the **Functions** category, drag out these 3 functions into the [Forever] block to have them executed in order.



```
Forever
    sound
    music
    tts
```

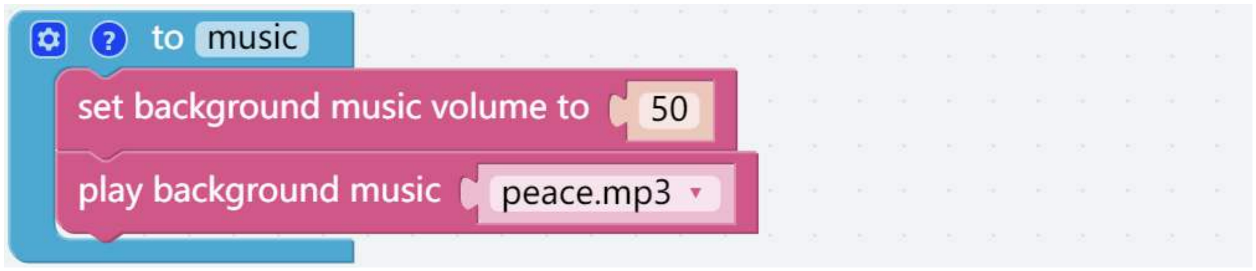
Step 6

Once the code is written, click the **Download** button in the bottom right corner to download it to the PiArm.

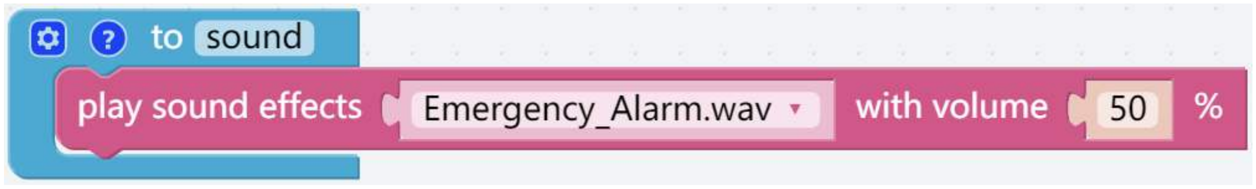
Now you will find that piarm first plays the sound effect in the sound function, and then plays the background music in the [music] function. When the background music is played, the [tts] function is run for timing, and the countdown voice broadcast will be performed after 30 seconds.

Note: You can also find the code with the same name on the **Examples** page of **Ezblock Studio** and click **Run** or

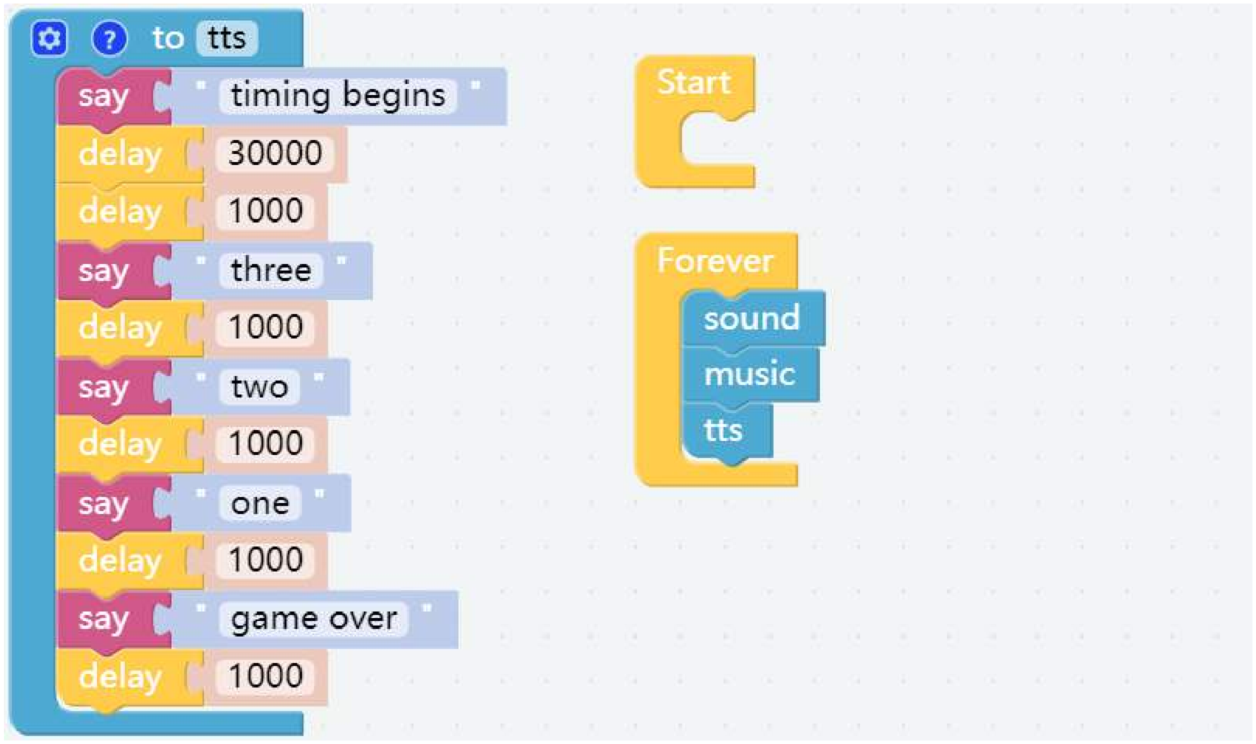
Edit directly to see the results.



```
to music
    set background music volume to 50
    play background music peace.mp3
```



```
to sound
    play sound effects Emergency_Alarm.wav with volume 50 %
```



```
to tts
    say "timing begins"
    delay 30000
    delay 1000
    say "three"
    delay 1000
    say "two"
    delay 1000
    say "one"
    delay 1000
    say "game over"
    delay 1000

Start

Forever
    sound
    music
    tts
```

3.4 Dual Joystick Module

We can control PiArm in 2 parts, Arm and EoAT. In the first project, you have learned how to *Test 3 EoATs* of PiArm's separately.

In this project, first the arm is controlled by *Angle Mode* and dual joystick module. Then the control code for the three EoATs was added to this so that the dual joystick module can control both arm and EoAT.

- *Arm - Joystick Control*
- *Shovel Bucket - Joystick Control*

- *Hanging Clip - Joystick Control*
- *Electromagnet - Joystick Control*

3.4.1 Arm - Joystick Control

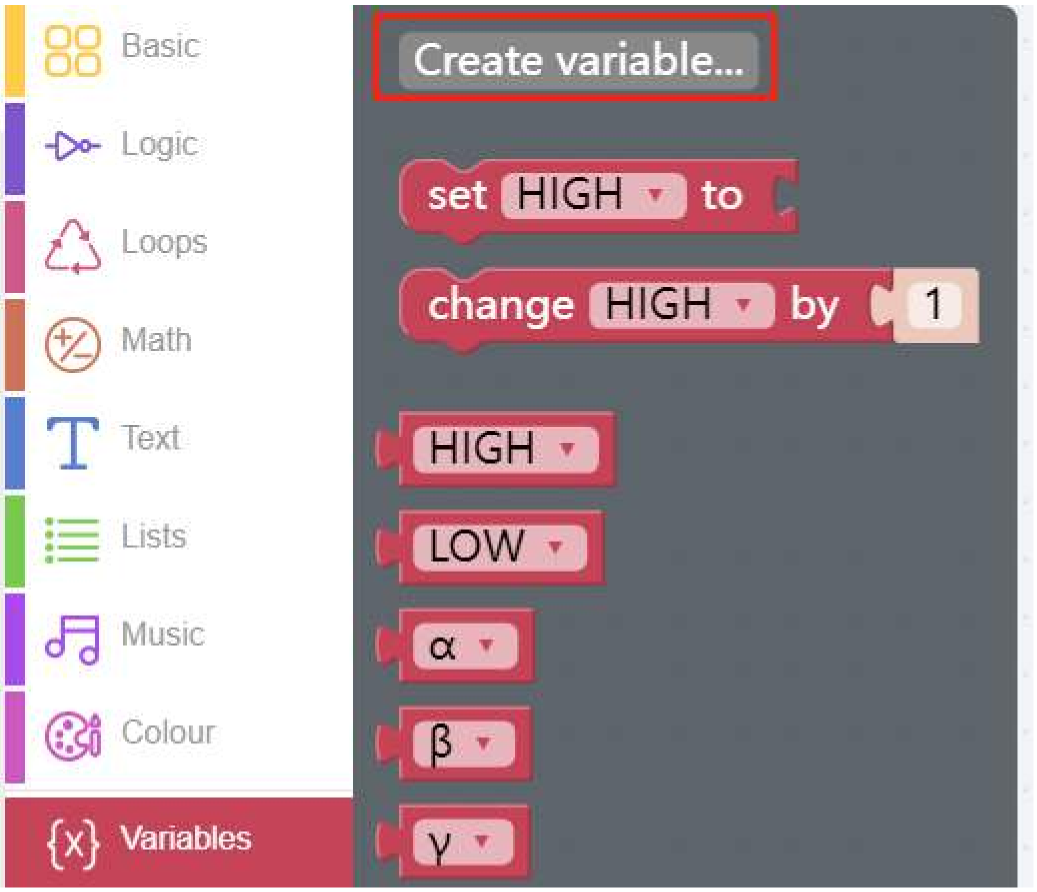
PiArm's arm can be controlled in two ways: *Angle Mode* and *Coordinate Mode*.

- *Angle Mode*: Writes a certain angle to the three servos on the arm, thus rotating the arm to a specific position.
- *Coordinate Mode*: Create a spatial right-angle coordinate system for the arm and set the control point. Set the coordinates of the control point so that the arm can reach a specific position.

Step 1

You may want to simplify your program with variables, now click the **Create variable** button on the **Variables** category to create 5 variables (`HIGH`, `LOW`, , and).

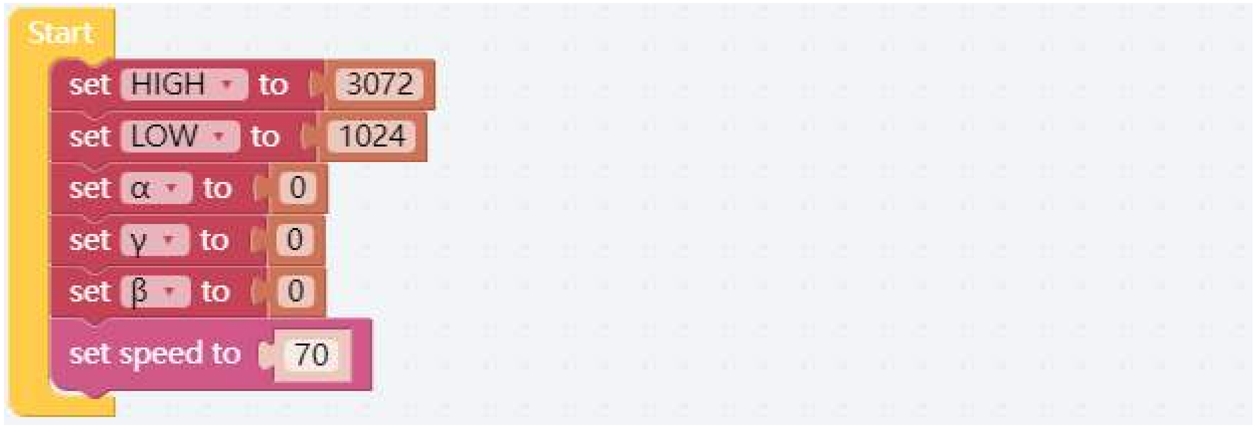
Note: The created variables are also stored in the **Variables** category.



Step 2

Set the initial values for these variables and set the servo rotation speed to 70%.

Note: For the reason of the values of the **HIGH** and **LOW** variables, please refer to *Dual Joystick Module*.

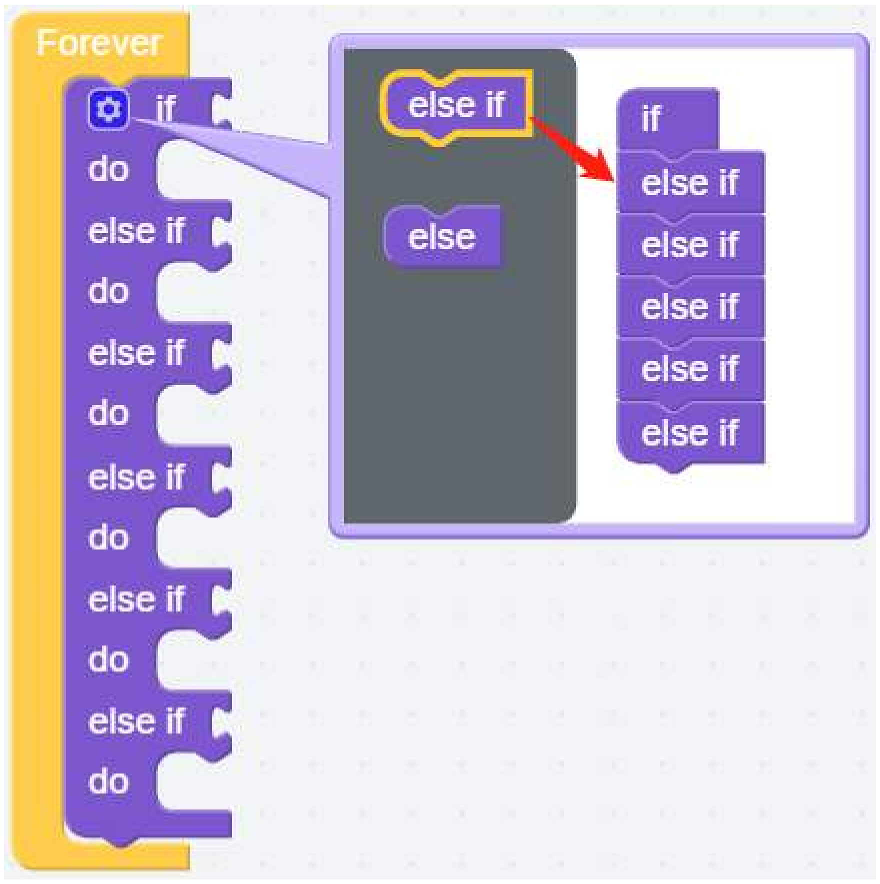


```
set HIGH to 3072
set LOW to 1024
set α to 0
set γ to 0
set β to 0
set speed to 70
```

Step 3

Use [if else] block to do some conditional judgment cases (drag 5 [else if] blocks from the left to below the [if] block).

- [if else]: Conditional judgment block, you can create multiple conditional judgments by clicking the set icon and dragging [else] or [else if] to the right below the [if].

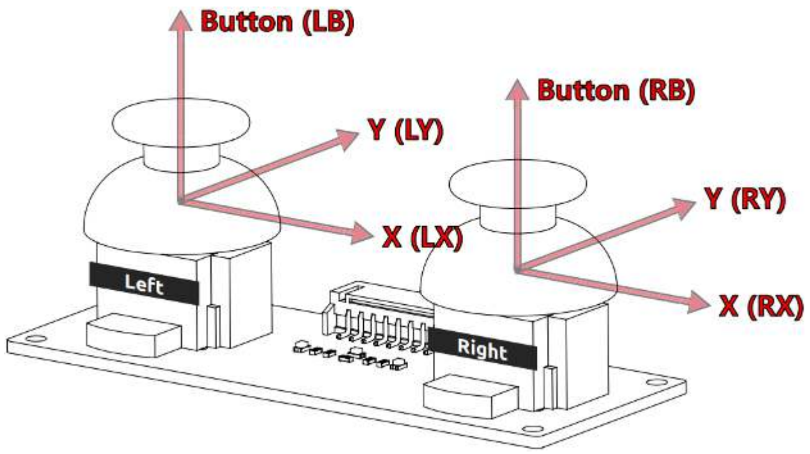


Step 4

The left and right joystick connections for the dual joystick module are shown below, refer to *Dual Joystick Module*.

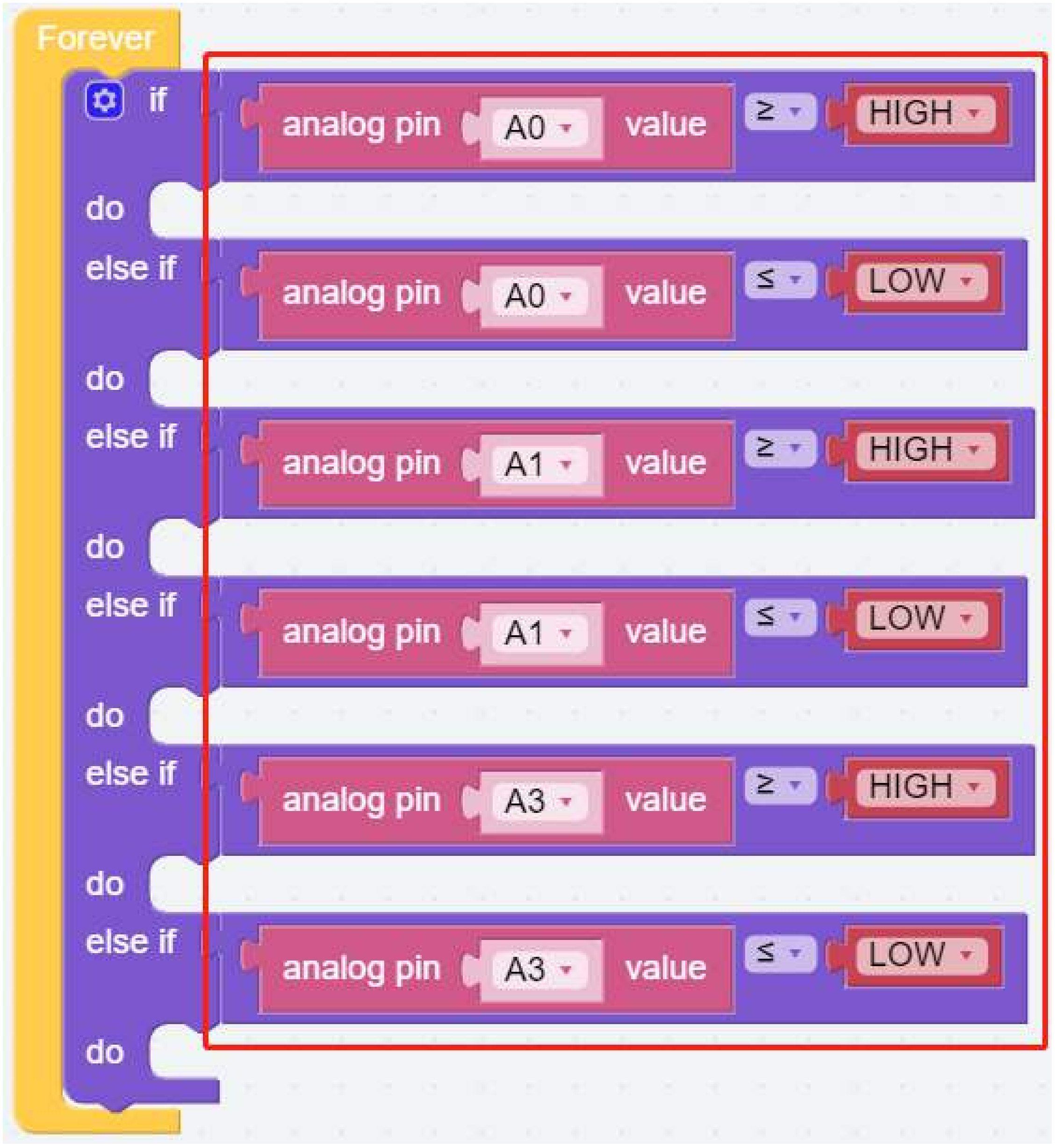
- The X of the left joystick is connected to A0 and the Y is connected to A1.
- The X of the right joystick is connected to A2, and the Y is connected to A3.

Assume that the X and Y of the left joystick and the Y of the right joystick are used to control the 3 servos of PiArm respectively, now first set the judgment condition to determine whether the left and right joysticks are toggled or not.



- If **A0 (LX)** is greater than **HIGH (3072)**, it means that the **left joystick** is toggled to the right.
- If **A0 (LX)** is less than **LOW (1024)**, it means the **left joystick** is toggled to the left.

- If **A1 (LY)** is greater than **HIGH (3072)**, it means the **left joystick** is toggled forward.
- If **A1 (LY)** is less than **LOW (1024)**, it means the **left joystick** is toggled backward.
- If **A3 (RY)** is greater than **HIGH (3072)**, it means the **right joystick** is toggled forward.
- If **A3 (RY)** is less than **LOW (1024)**, it means the **right joystick** is toggled backward.



```
Forever
  if      analog pin A0 value ≥ HIGH
  do
  else if analog pin A0 value ≤ LOW
  do
  else if analog pin A1 value ≥ HIGH
  do
  else if analog pin A1 value ≤ LOW
  do
  else if analog pin A3 value ≥ HIGH
  do
  else if analog pin A3 value ≤ LOW
  do
```

Step 4

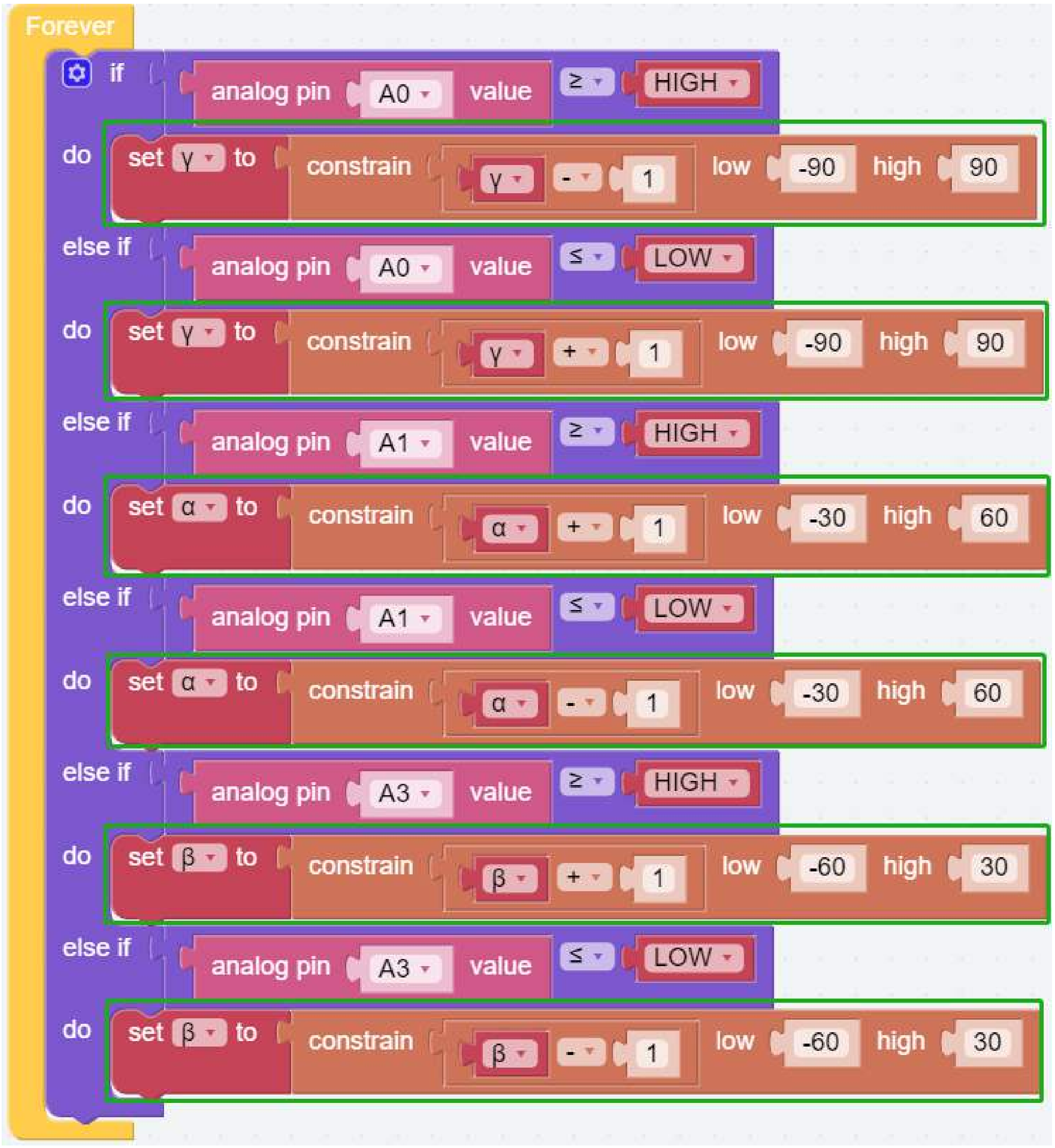
Now set the rotation effect of PiArm according to the toggle of the left and right joysticks.

- If the **left joystick** is toggled to the right, the Arm will turn right.
- If the **left joystick** is toggled to the left, the Arm will turn left.
- If the **left joystick** is toggled forward, the Arm will extend forward.
- If the **left joystick** is toggled backward, the Arm will retract backward.

- If the **right joystick** is toggled forward, the Arm will lower down.
- If the **right joystick** is toggled backward, the Arm will raise up.

Note:

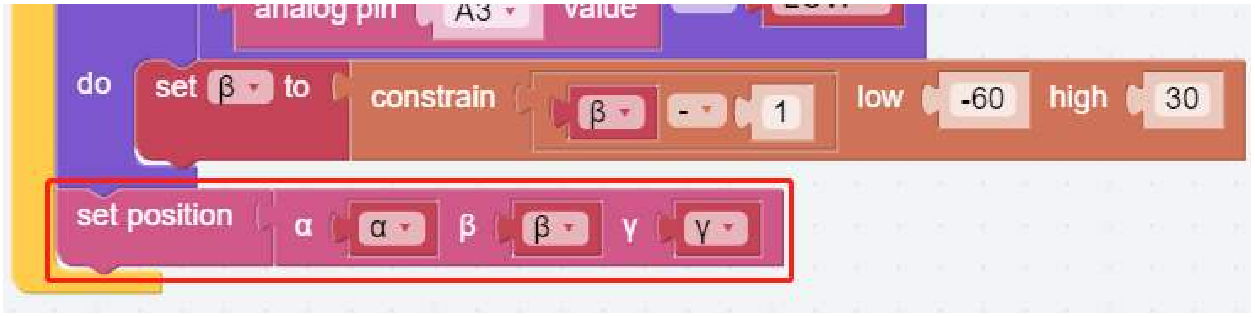
- , and represent the 3 servo rotation ranges on PiArm, refer to: *Angle Mode*.
- [constrain () low () high ()]: From Math category for setting the variation of a constant to a certain range.



```
Forever
  if analog pin A0 value ≥ HIGH
  do set γ to constrain γ - 1 low -90 high 90
  else if analog pin A0 value ≤ LOW
  do set γ to constrain γ + 1 low -90 high 90
  else if analog pin A1 value ≥ HIGH
  do set α to constrain α + 1 low -30 high 60
  else if analog pin A1 value ≤ LOW
  do set α to constrain α - 1 low -30 high 60
  else if analog pin A3 value ≥ HIGH
  do set β to constrain β + 1 low -60 high 30
  else if analog pin A3 value ≤ LOW
  do set β to constrain β - 1 low -60 high 30
```

Step 5

Store the obtained , and angle values into the [() () ()] block, and then use the [set positon] block to make PiArm rotate this position.

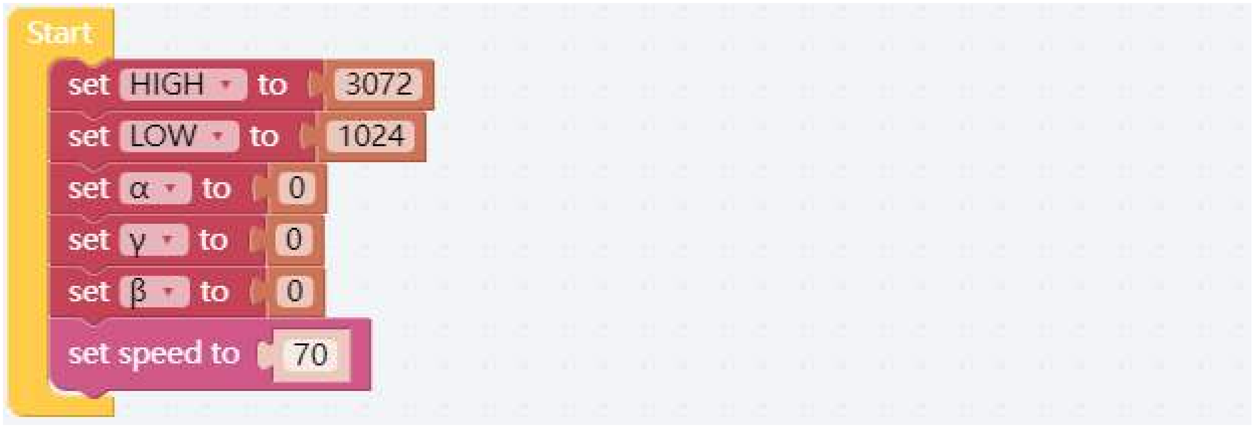


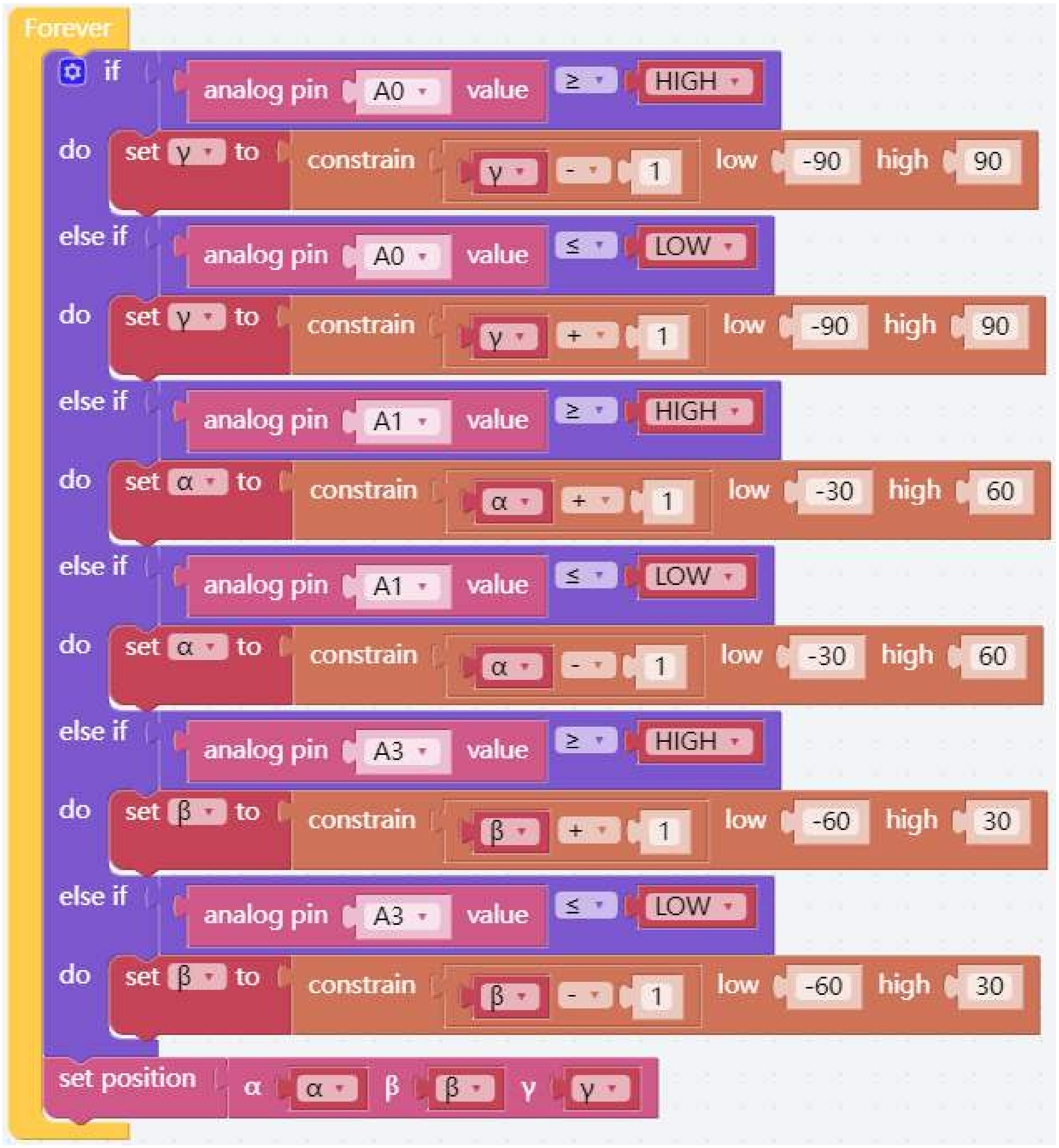
Step 7

Once you click the download button, you can use the Dual Joystick Module to control PiArm.

- Left joystick toggle left or right, the arm will turn to the left or right.
- Left joystick toggle forward or backward, the arm will extend forward or retract backward.
- Right joystick toggle forward or backward, the arm will raise up or lower down.

Note: You can also find the code with the same name on the **Examples** page of **Ezblock Studio** and click **Run** or **Edit** directly to see the results.



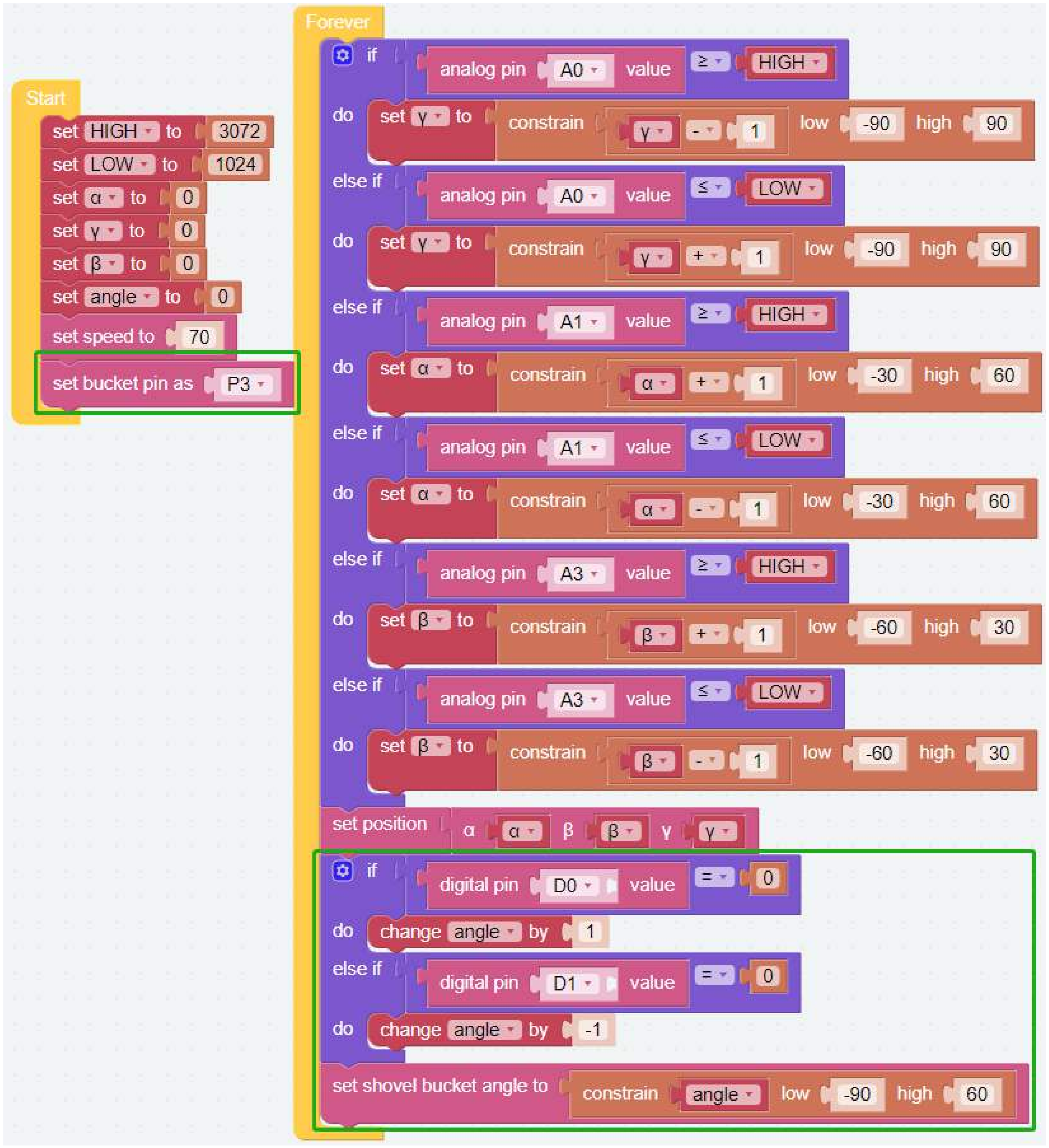


```
Forever
  if  analog pin A0 value ≥ HIGH
  do  set γ to constrain  γ - 1  low -90  high 90
  else if  analog pin A0 value ≤ LOW
  do  set γ to constrain  γ + 1  low -90  high 90
  else if  analog pin A1 value ≥ HIGH
  do  set α to constrain  α + 1  low -30  high 60
  else if  analog pin A1 value ≤ LOW
  do  set α to constrain  α - 1  low -30  high 60
  else if  analog pin A3 value ≥ HIGH
  do  set β to constrain  β + 1  low -60  high 30
  else if  analog pin A3 value ≤ LOW
  do  set β to constrain  β - 1  low -60  high 30
  set position  α α  β β  γ γ
```

3.4.2 Shovel Bucket - Joystick Control

Now add the control code for the Shovel Bucket.

Note: You can also find the code with the same name on the **Examples** page of Ezblock Studio and click Run or Edit directly to view the code.



```
Start
  set HIGH to 3072
  set LOW to 1024
  set α to 0
  set γ to 0
  set β to 0
  set angle to 0
  set speed to 70
  set bucket pin as P3

Forever
  if analog pin A0 value ≥ HIGH
  do set γ to constrain γ - 1 low -90 high 90
  else if analog pin A0 value ≤ LOW
  do set γ to constrain γ + 1 low -90 high 90
  else if analog pin A1 value ≥ HIGH
  do set α to constrain α + 1 low -30 high 60
  else if analog pin A1 value ≤ LOW
  do set α to constrain α - 1 low -30 high 60
  else if analog pin A3 value ≥ HIGH
  do set β to constrain β + 1 low -60 high 30
  else if analog pin A3 value ≤ LOW
  do set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
  if digital pin D0 value = 0
  do change angle by 1
  else if digital pin D1 value = 0
  do change angle by -1
  set shovel bucket angle to constrain angle low -90 high 60
```

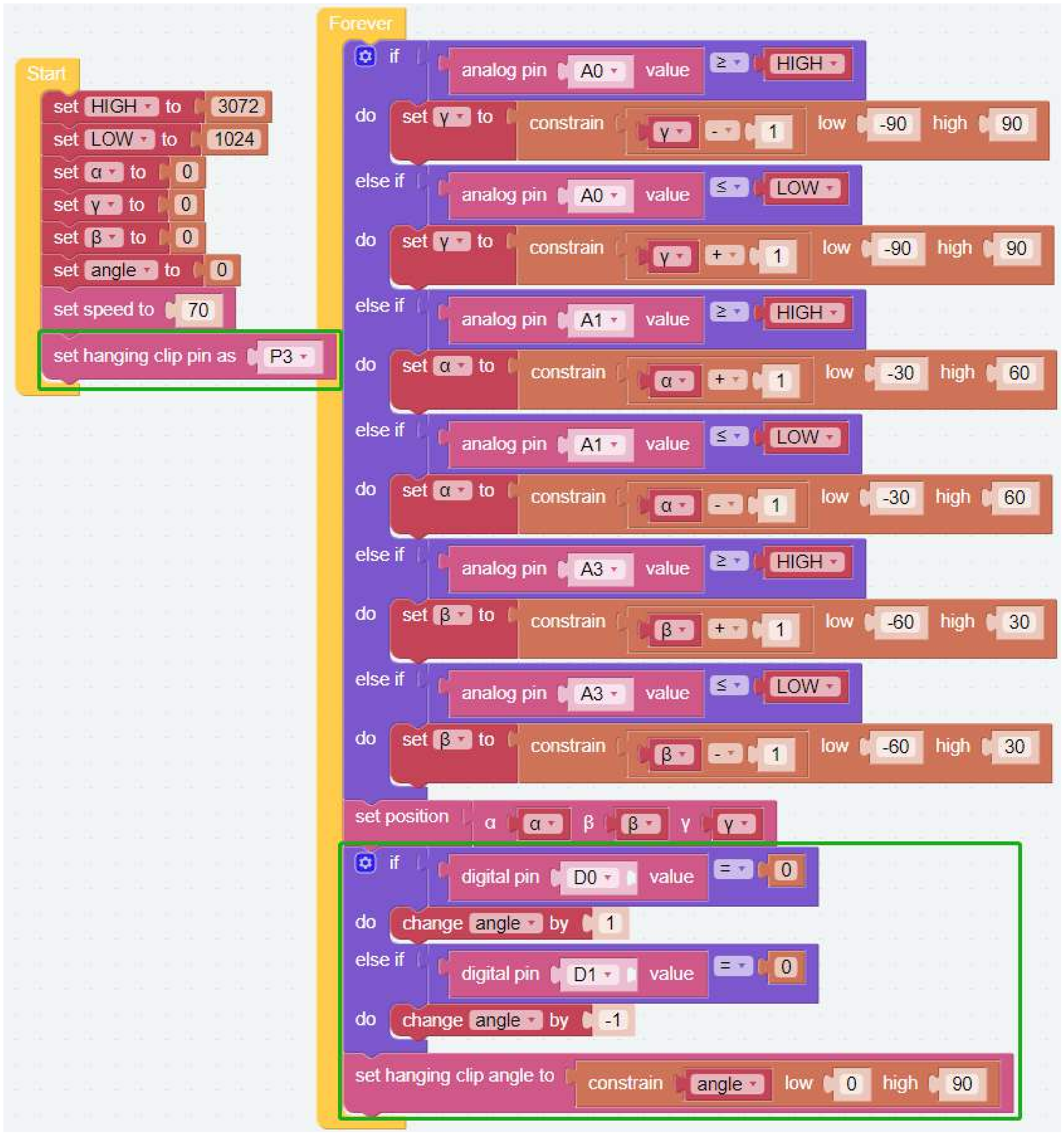
Once the code is run, you can control both the PiArm's arm and Shovel Bucket with the dual joystick module. But you need to install *Shovel Bucket* to the PiArm first.

- Push the left joystick to the left or right, the arm will turn to the left or right.
- Push the left joystick forward or backward, the arm will extend or retract.
- Push the right joystick forward or backward, the arm will be raised or lowered.
- Push the left joystick to rewind the Shovel Bucket inward.
- Press the right joystick to extend the Shovel Bucket outward.

3.4.3 Hanging Clip - Joystick Control

Now add the control code for the Hanging Clip to the code for the control arm.

Note: You can also find the code with the same name on the **Examples** page of Ezblock Studio and click Run or Edit directly to view the block.



```
Start
  set HIGH to 3072
  set LOW to 1024
  set α to 0
  set γ to 0
  set β to 0
  set angle to 0
  set speed to 70
  set hanging clip pin as P3

Forever
  if analog pin A0 value ≥ HIGH
  do set γ to constrain γ - 1 low -90 high 90
  else if analog pin A0 value ≤ LOW
  do set γ to constrain γ + 1 low -90 high 90
  else if analog pin A1 value ≥ HIGH
  do set α to constrain α + 1 low -30 high 60
  else if analog pin A1 value ≤ LOW
  do set α to constrain α - 1 low -30 high 60
  else if analog pin A3 value ≥ HIGH
  do set β to constrain β + 1 low -60 high 30
  else if analog pin A3 value ≤ LOW
  do set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
  if digital pin D0 value = 0
  do change angle by 1
  else if digital pin D1 value = 0
  do change angle by -1
  set hanging clip angle to constrain angle low 0 high 90
```

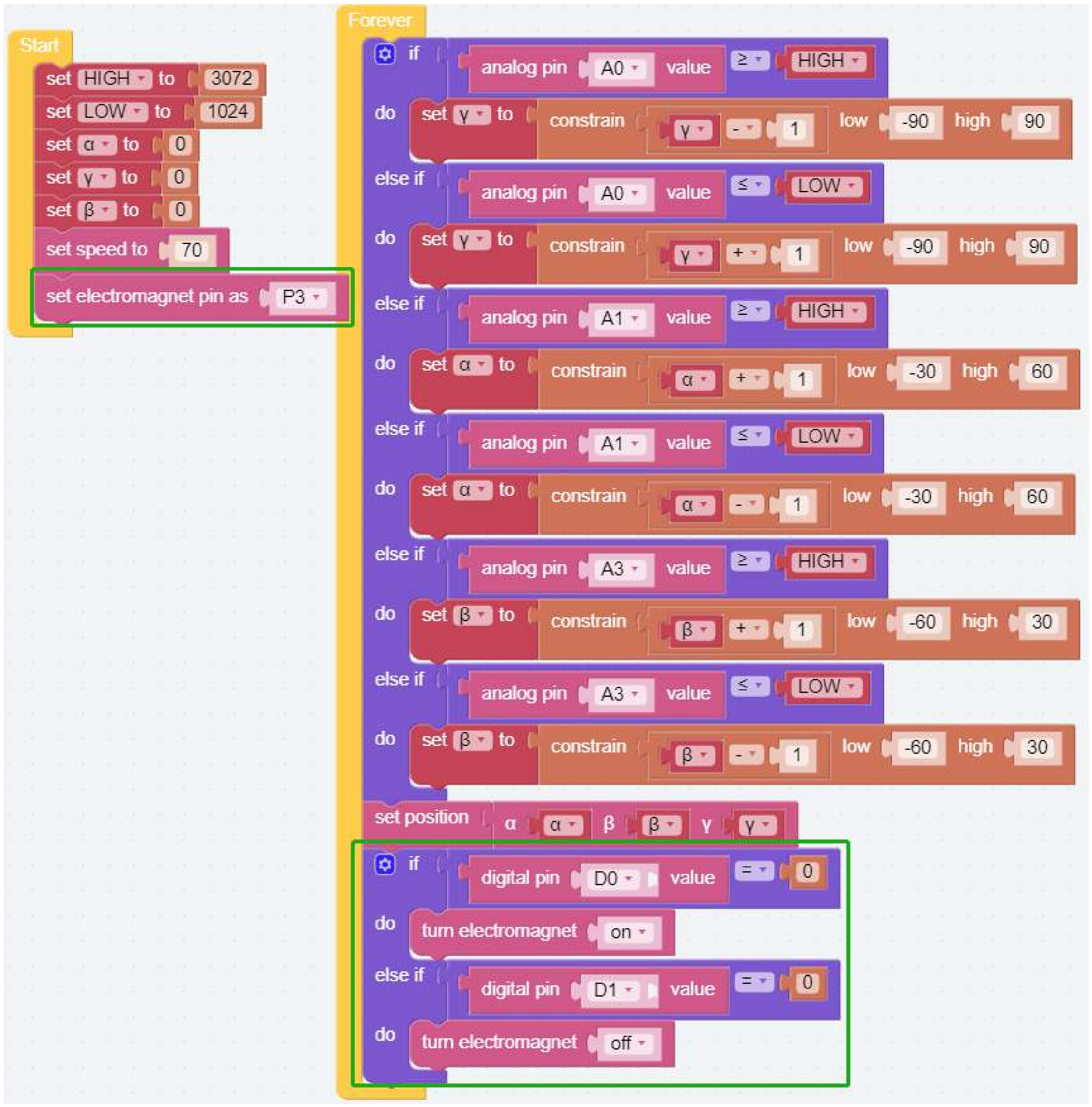
After the code is run, you can use the dual joystick module to control PiArm's arms and vertical clips at the same time. But you need to install *Hanging Clip* to PiArm first.

- Push the left Joystick to the left or right, the arm will turn to the left or right.
- Push the left Joystick forward or backward, the arm will extend or retract.
- Push the right Joystick forward or backward, the arm will be raised or lowered.
- Press the left Joystick to close the Hanging Clip.
- Press the right Joystick to open the Hanging Clip.

3.4.4 Electromagnet - Joystick Control

Now add the control code for the Electromagnet to the code for the control arm.

Note: You can also find the code with the same name on the **Examples** page of Ezblock Studio and click Run or Edit directly to view the block.



```
Start
  set HIGH to 3072
  set LOW to 1024
  set α to 0
  set γ to 0
  set β to 0
  set speed to 70
  set electromagnet pin as P3

Forever
  if analog pin A0 value ≥ HIGH
  do set γ to constrain γ - 1 low -90 high 90
  else if analog pin A0 value ≤ LOW
  do set γ to constrain γ + 1 low -90 high 90
  else if analog pin A1 value ≥ HIGH
  do set α to constrain α + 1 low -30 high 60
  else if analog pin A1 value ≤ LOW
  do set α to constrain α - 1 low -30 high 60
  else if analog pin A3 value ≥ HIGH
  do set β to constrain β + 1 low -60 high 30
  else if analog pin A3 value ≤ LOW
  do set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
  if digital pin D0 value = 0
  do turn electromagnet on
  else if digital pin D1 value = 0
  do turn electromagnet off
```

After the code is run, you can use the dual joystick module to control both PiArm's arm and the Electromagnet. But you need to install *Electromagnet* to PiArm first.

- Push the left joystick to the left or right, the arm will turn to the left or right.
- Push the left joystick forward or backward, the arm will extend or retract.

- Push the right joystick forward or backward, the arm will be raised or lowered.
- Press the left joystick to turn on the Electromagnet.
- Press the right joystick to turn the Electromagnet off.

3.5 Remote Control

In addition to the dual joystick module, we can also use the widgets on the Remote Control page in **EzBlock Studio** to control **PiArm** movement.

- *Arm - Remote Control*
- *Create a Library*
- *Shovel Bucket - Remote Control*
- *Hanging Clip - Remote Control*
- *Electromagnet - Remote Control*

3.5.1 Arm - Remote Control

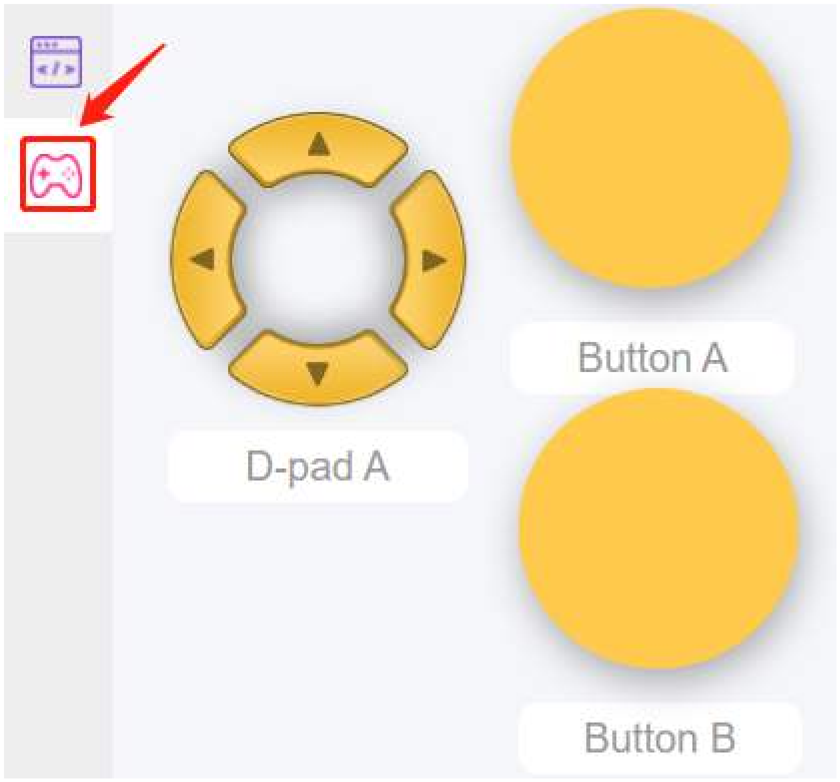
PiArm's arm can be controlled in two ways: *Angle Mode* and *Coordinate Mode*.

- *Angle Mode*: Writes a certain angle to the three servos on the arm, thus rotating the arm to a specific position.
- *Coordinate Mode*: Create a spatial right-angle coordinate system for the arm and set the control point. Set the coordinates of the control point so that the arm can reach a specific position.

The *Angle Mode* is used here.

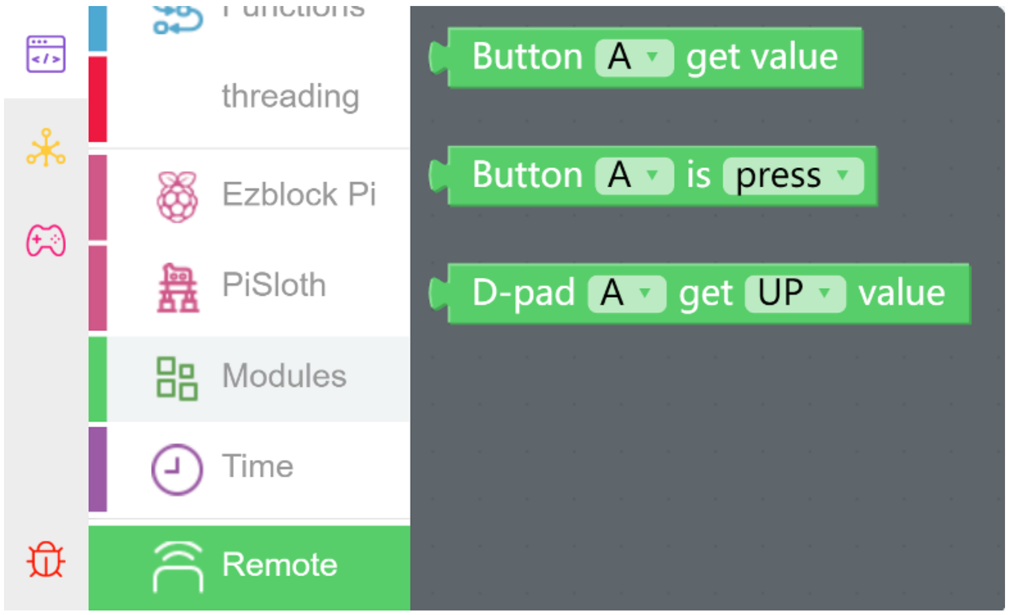
Step 1

To use the remote control function, you need to enter the **Remote Control** page from the left side of main page, and then drag one D-pad and 3 buttons to the central area.



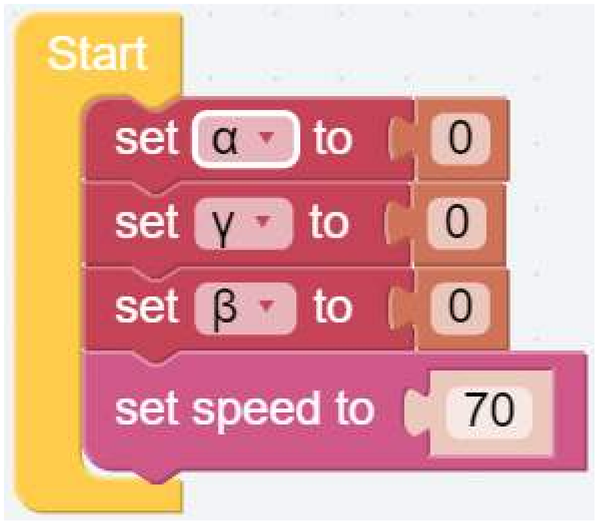
Back in the programming page, you will see an additional Remote category, and the D-pad and Button block appear in it.

- [Button () get value]: This block is used to read the value of the button, press is 1, release is 0.
- [Button () is (press/release)]: This block and `Button () get value = (0/1)` have the same effect and can be used directly to determine whether a button is pressed or not.
- [D-pad () get () value]: This block is used to read the up/down/left/right (selected through the drop-down menu) pad values, press for 1 and release for 0.



Step 2

Create 3 variables (, and) and set the initial values, and set the rotation speed of PiArm to 70%.



Step 3

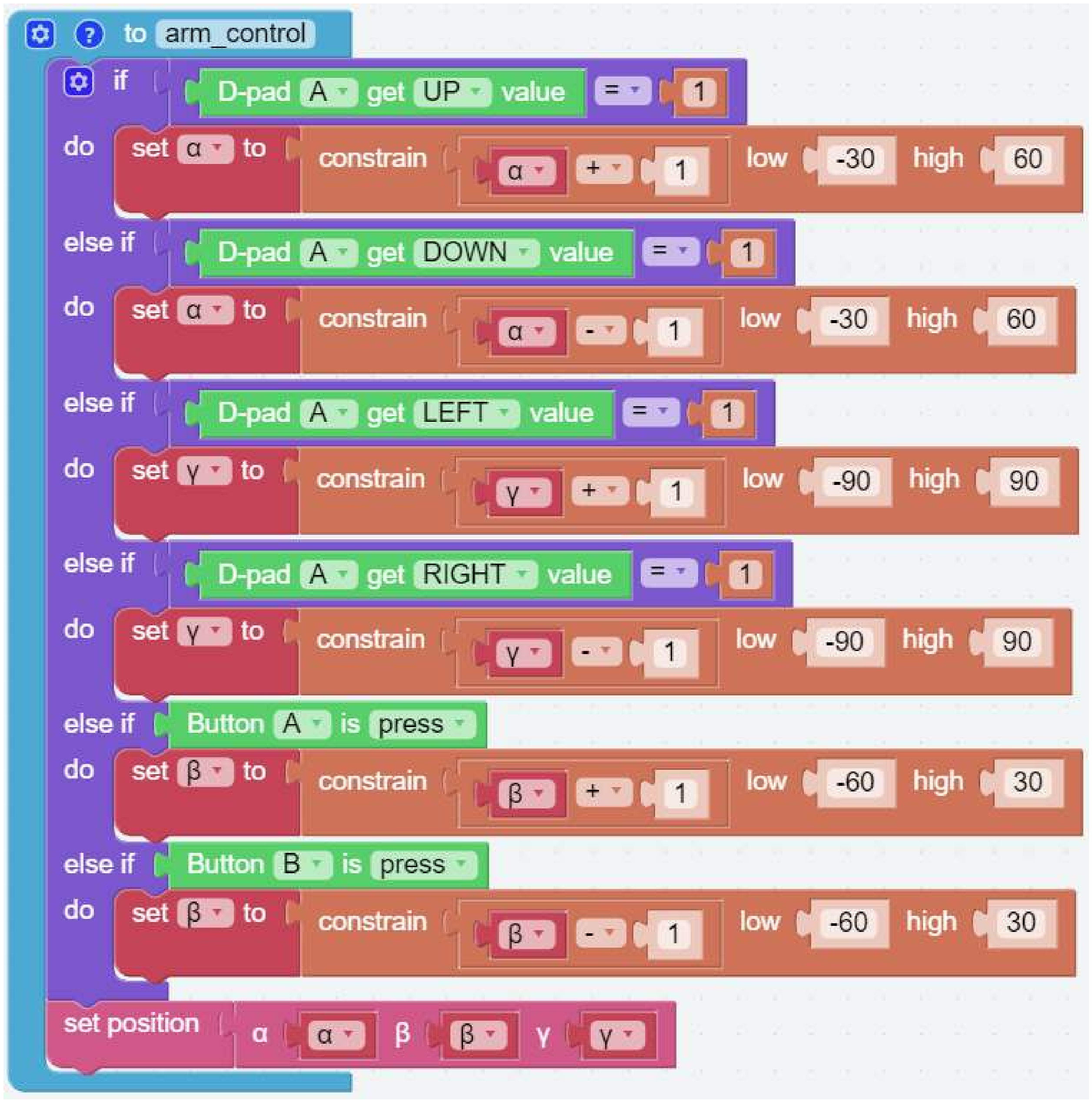
Create a function called [arm_control] to set the rotation direction of the PiArm based on the arrow keys and button values.

Note: The function name cannot contain spaces, and two words can be connected by _.

Note:

- , and represent the 3 servo rotation ranges on PiArm, refer to: *Angle Mode*.
- [constrain () low () high ()]: From Math category for setting the variation of a constant to a certain range.
- [if else]: Conditional judgment block, you can create multiple conditional judgments by clicking the set icon

and dragging [else] or [else if] to the right below the [if].



```
to arm_control
  if      D-pad A get UP value = 1
  do      set α to constrain α + 1 low -30 high 60
  else if D-pad A get DOWN value = 1
  do      set α to constrain α - 1 low -30 high 60
  else if D-pad A get LEFT value = 1
  do      set γ to constrain γ + 1 low -90 high 90
  else if D-pad A get RIGHT value = 1
  do      set γ to constrain γ - 1 low -90 high 90
  else if Button A is press
  do      set β to constrain β + 1 low -60 high 30
  else if Button B is press
  do      set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
```

- If the UP button () of D-pad is pressed, the Arm will extend forward.
- If the Down button () of D-pad is pressed, the Arm will retract backward.
- If the LEFT button () of D-pad is pressed, the Arm will turn left.
- If the RIGHT button () of D-pad is pressed, the Arm will turn right.
- If Button A is pressed, the Arm will lower down.
- If Button B is pressed, the Arm will raise up.

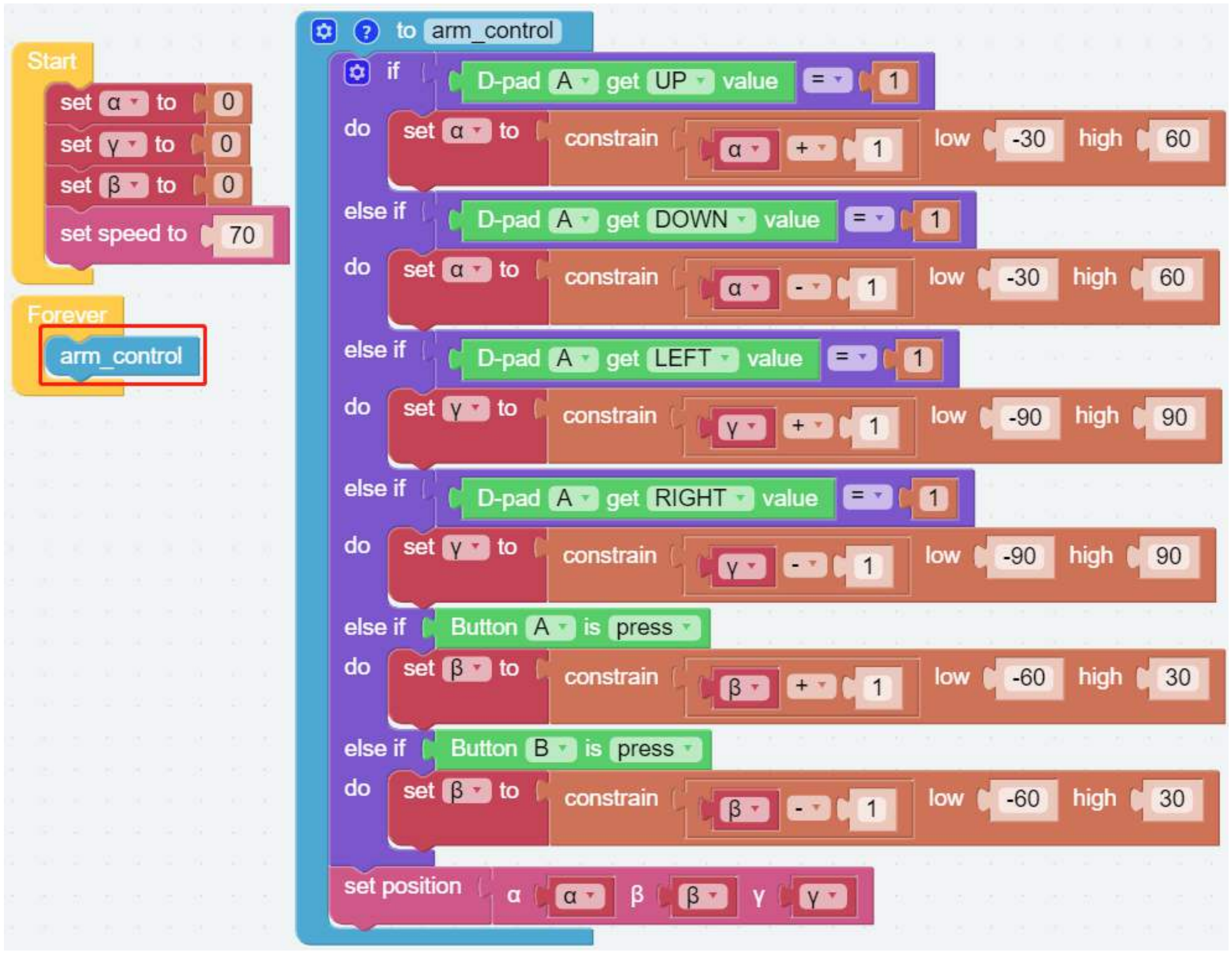
Step 4

Put the function [arm_control] into [Forever] for loop execution, and finally click the **Download** button to run the code.

After that you can use the D-pad and Button A/B on the **Remote Control** page to control the movement of the Arm.

Note:

- The functions must be placed before the [start] and [Forever] blocks.
- You can also find the code with the same name on the EzBlock Studio **Examples** page and click Run or Edit directly to view the result.



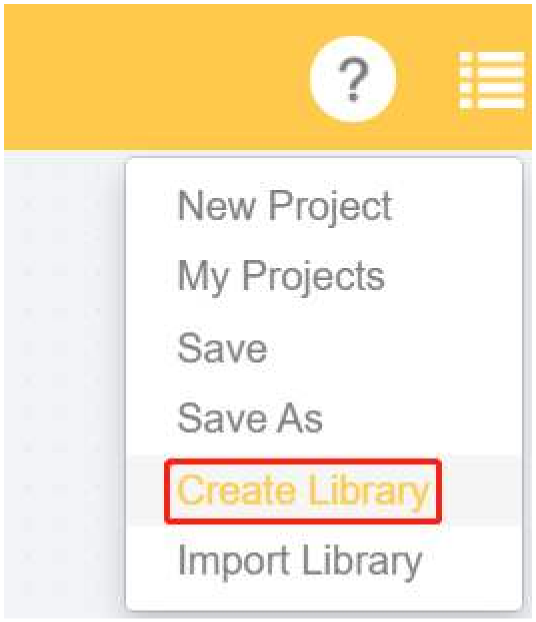
```
Start
  set α to 0
  set γ to 0
  set β to 0
  set speed to 70

Forever
  arm_control

to arm_control
  if D-pad A get UP value = 1
  do set α to constrain α + 1 low -30 high 60
  else if D-pad A get DOWN value = 1
  do set α to constrain α - 1 low -30 high 60
  else if D-pad A get LEFT value = 1
  do set γ to constrain γ + 1 low -90 high 90
  else if D-pad A get RIGHT value = 1
  do set γ to constrain γ - 1 low -90 high 90
  else if Button A is press
  do set β to constrain β + 1 low -60 high 30
  else if Button B is press
  do set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
```

3.5.2 Create a Library

To be able to use the function - [arm_control] in other code later, you can create it as a library and import it when you need to use it.

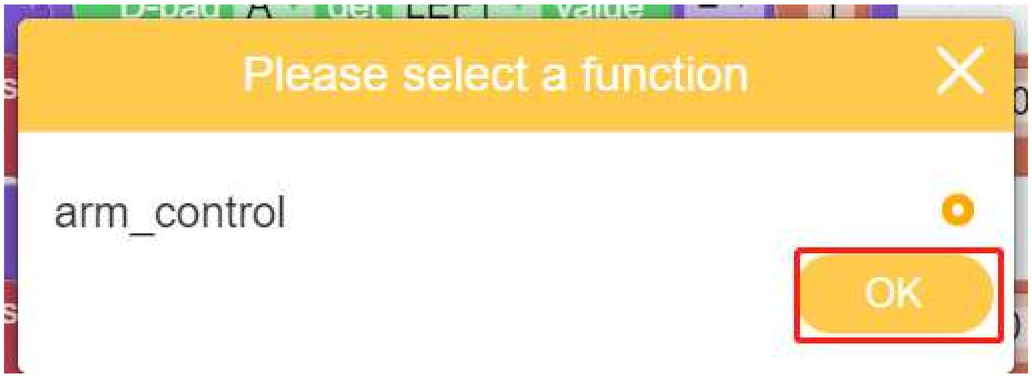
Step 1

Open the menu icon in the upper right corner and select **Create Library**.



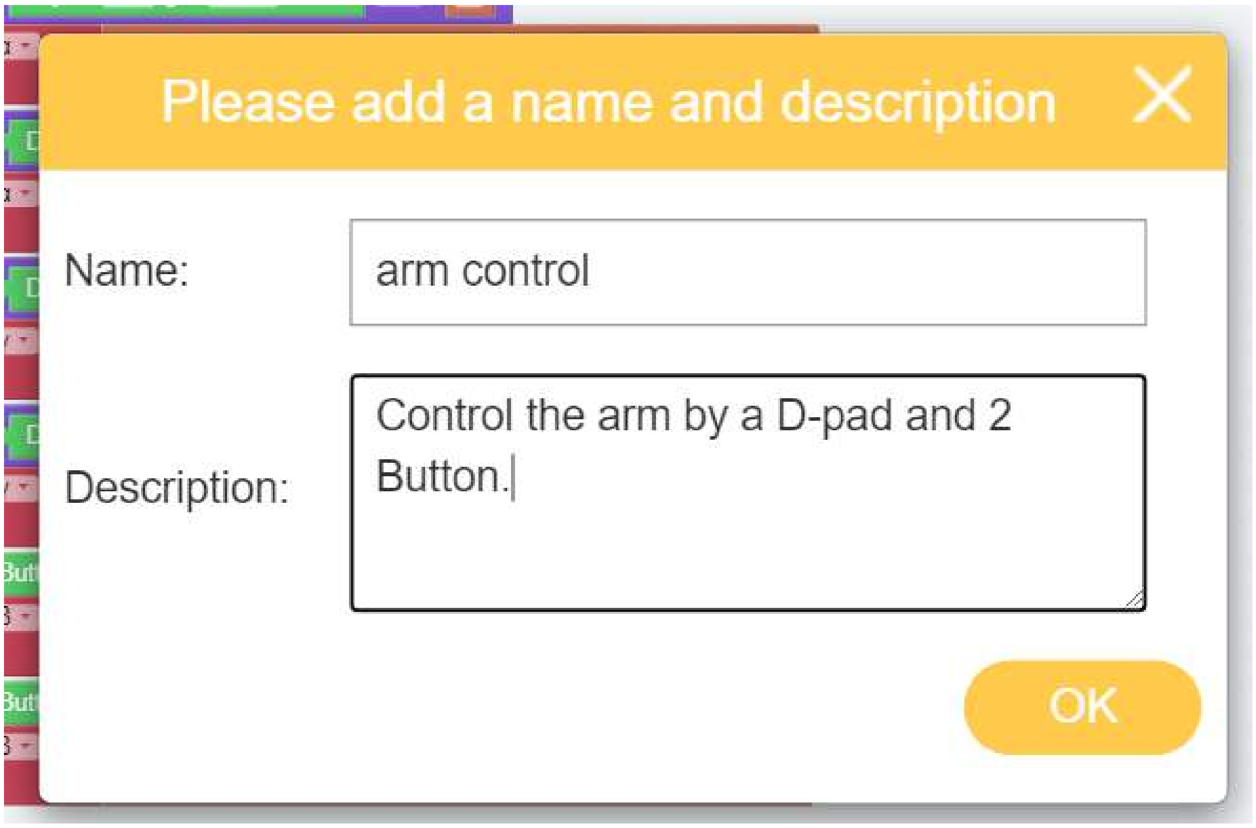
Step 2

Select the function, there is only one function here, so **arm_control** is selected by default.



Step 3

Name the library and fill in the description so that it can be better distinguished later.

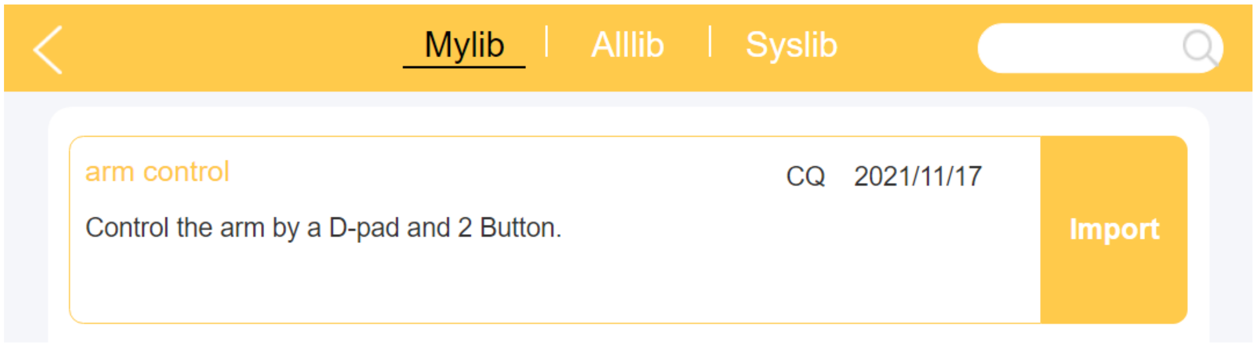


Name: arm control

Description: Control the arm by a D-pad and 2 Button.

Step 4

Wait for the prompt to save successfully and the library will be saved in **My Library** on your personal page. You can also see it when you click **Import Library**.



arm control CQ 2021/11/17

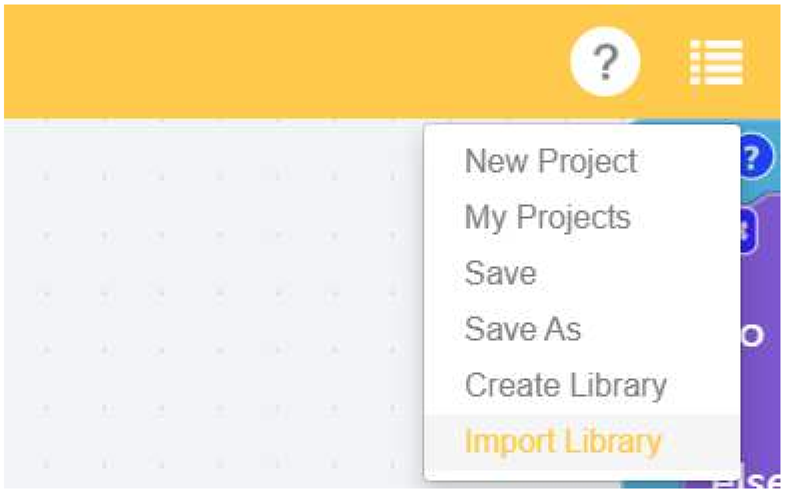
Control the arm by a D-pad and 2 Button.

3.5.3 Shovel Bucket - Remote Control

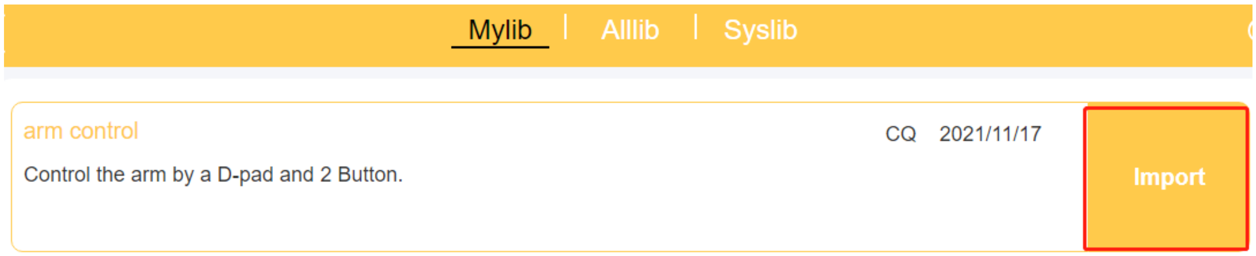
Create a new project and write code for it so that we can control the *Shovel Bucket* while controlling the arm.

Step 1

Import [arm_control] library, if you have not created this library before, please refer to: *Create a Library*.



In the **Mylib** page, select the library you created and click **Import**.

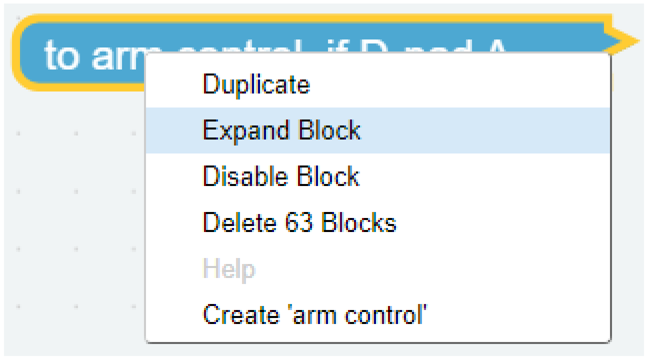


arm control

CQ 2021/11/17

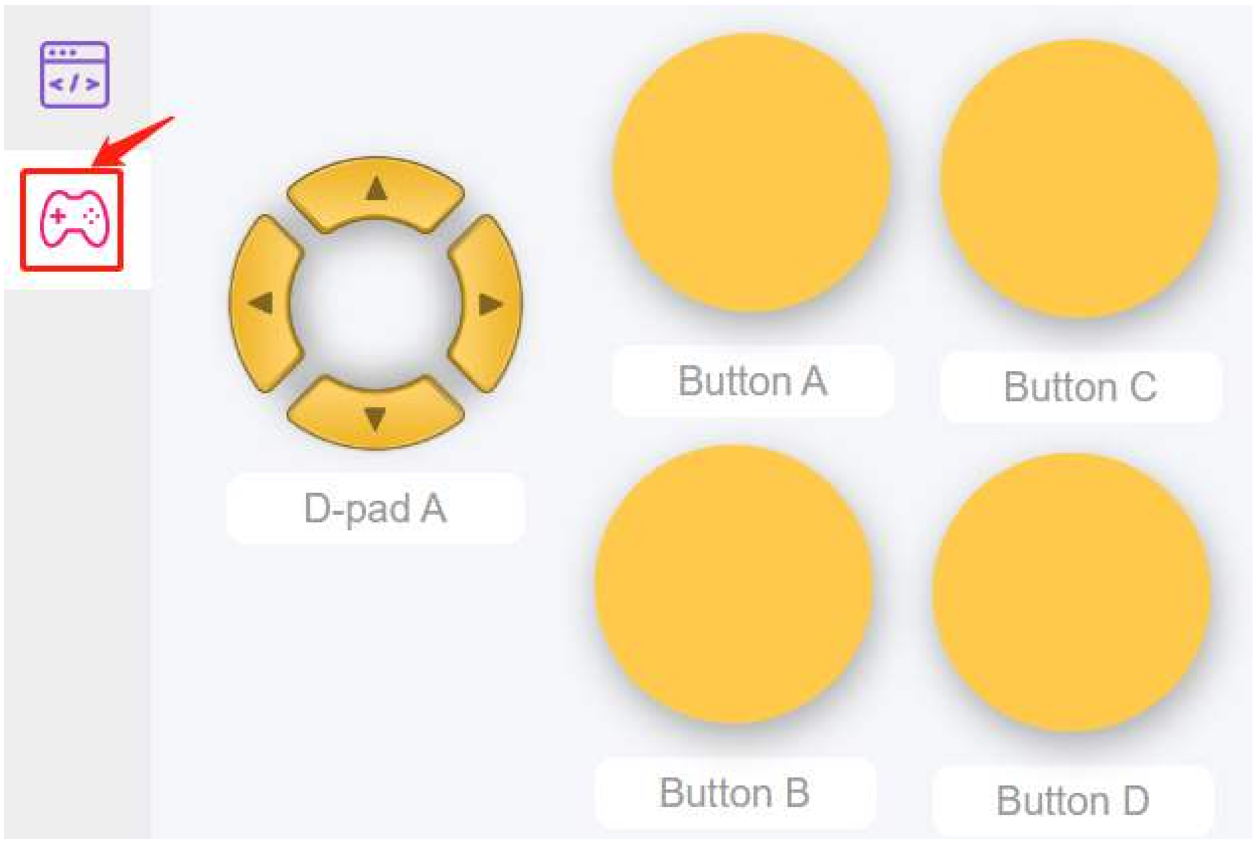
Control the arm by a D-pad and 2 Button.

After importing, this library is in a collapsed style. You can right-click on it and click **Expand Block**, so that you can see its internal code.



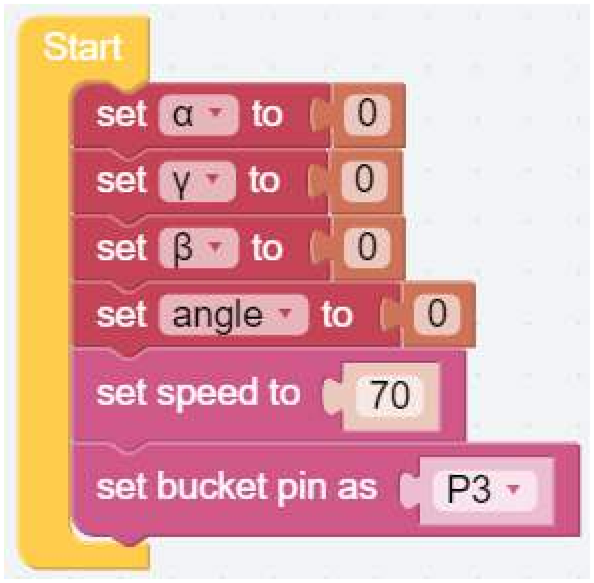
Step 2

Go to the remote control page and drag a D-pad and two buttons out again, because the import library will not import the widgets, so you need to drag them in again. Add two more buttons to control the angle of the Shovel Bucket.



Step 3

Create the variables (, , and `angle`) and set the initial values to 0, then initialize the PiArm rotation speed and the pin of Shovel Bucket.



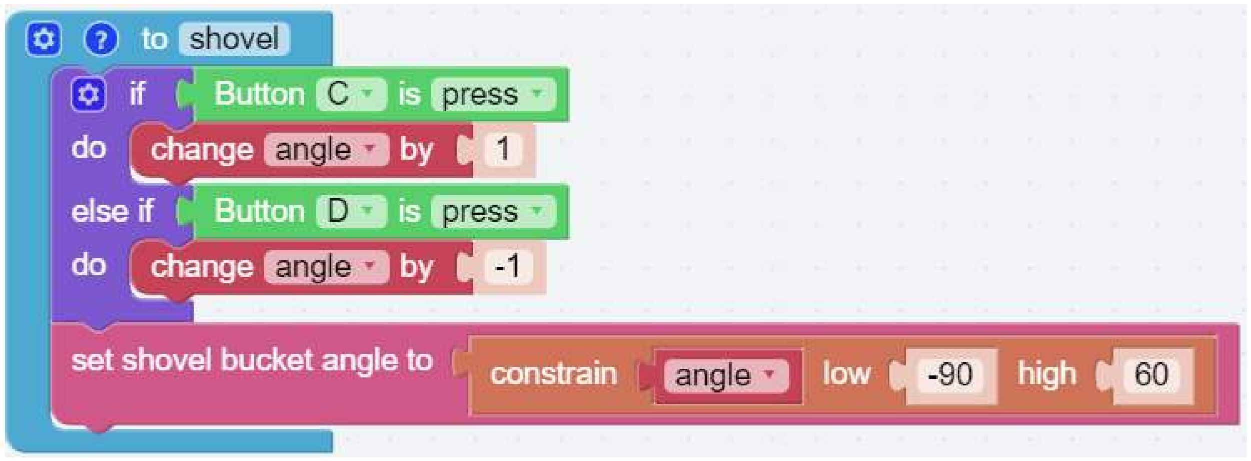
```
Start
  set α to 0
  set γ to 0
  set β to 0
  set angle to 0
  set speed to 70
  set bucket pin as P3
```

Step 4

Create a new function [shovel], and write the code as follows to control *Shovel Bucket* with two buttons.

- Use [if () else ()] block as a judgment condition. If button C is pressed, the variable `angle` is added by 1; if button D is pressed, the variable `angle` is subtracted by 1.

- Constrain the value of variable `angle` to -90 ~ 60 with [constrain () low() high ()] block.
- Set the angle of *Shovel Bucket* according to the variable `angle`.



```
to shovel
  if Button C is press
  do change angle by 1
  else if Button D is press
  do change angle by -1
  set shovel bucket angle to constrain angle low -90 high 60
```

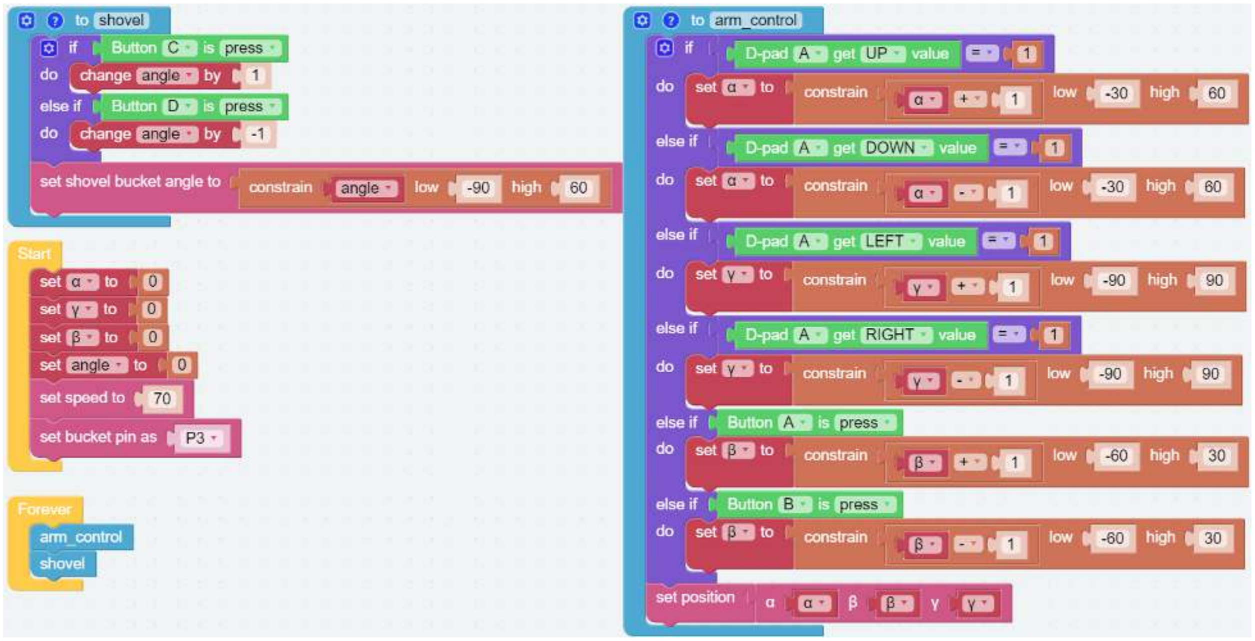
Step 5

Drag the [arm_control] and [shovel] functions from the **Functions** category to the [Forever] block respectively.

After clicking the download button, use the D-pad and buttons A/B on the remote control page to control the movement of the arm, and then use buttons C/D to control the addition/decrease of the bucket angle.

Note:

- The functions must be placed before the [start] and [Forever] blocks.
- You can also find the code with the same name on the EzBlock Studio **Examples** page and click Run or Edit directly to view the result.



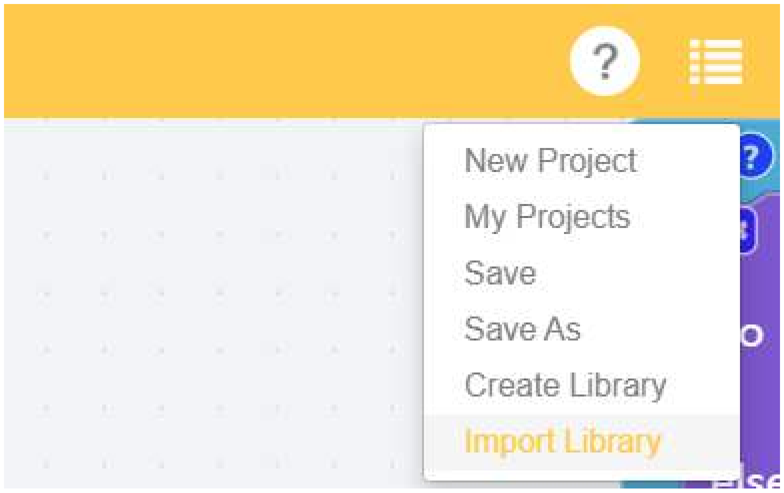
```
to shovel
  if Button C is press
  do change angle by 1
  else if Button D is press
  do change angle by -1
  set shovel bucket angle to constrain angle low -90 high 60

Start
  set α to 0
  set γ to 0
  set β to 0
  set angle to 0
  set speed to 70
  set bucket pin as P3

Forever
  arm_control
  shovel

to arm_control
  if D-pad A get UP value = 1
  do set α to constrain α + 1 low -30 high 60
  else if D-pad A get DOWN value = 1
  do set α to constrain α - 1 low -30 high 60
  else if D-pad A get LEFT value = 1
  do set γ to constrain γ + 1 low -90 high 90
  else if D-pad A get RIGHT value = 1
  do set γ to constrain γ - 1 low -90 high 90
  else if Button A is press
  do set β to constrain β + 1 low -60 high 30
  else if Button B is press
  do set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
```

3.5.4 Hanging Clip - Remote Control

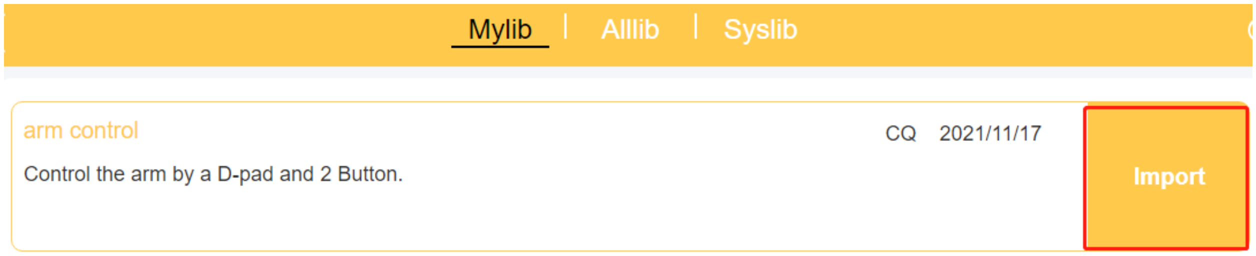
Create a new project and write code for it so that we can control the *Hanging Clip* while controlling the arm.

Step 1

Import [arm_control] library, if you have not created this library before, please refer to: *Create a Library*.



In the **Mylib** page, select the library you created and click **Import**.

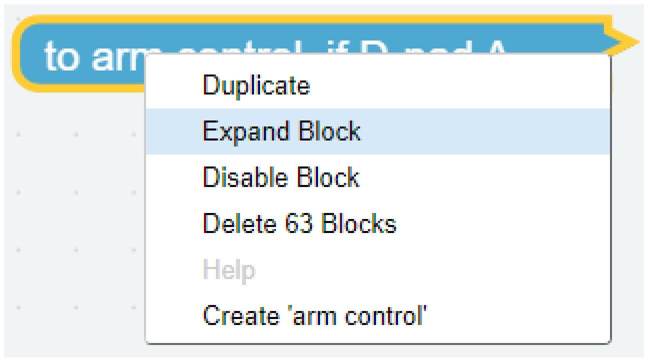


arm control

Control the arm by a D-pad and 2 Button.

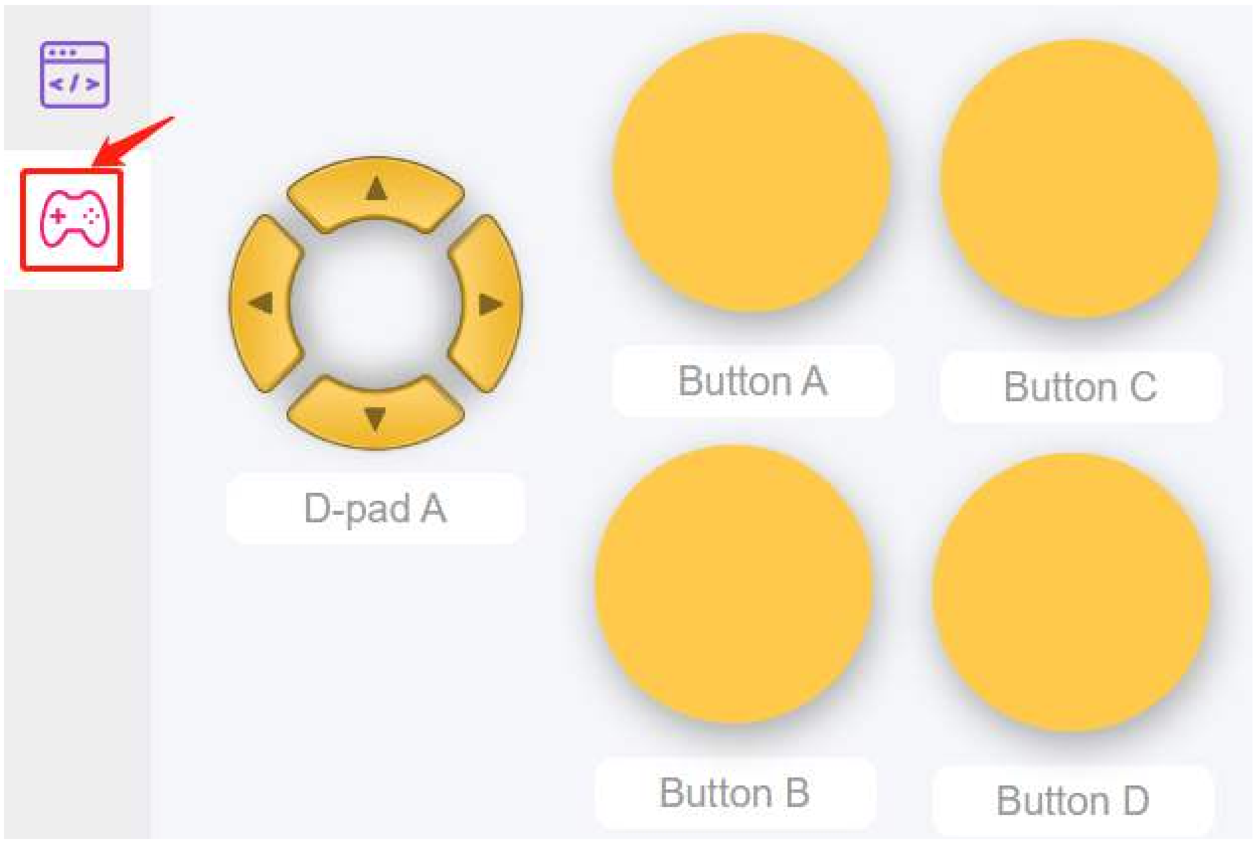
CQ 2021/11/17

After importing, this library is in a collapsed style. You can right-click on it and click **Expand Block**, so that you can see its internal code.



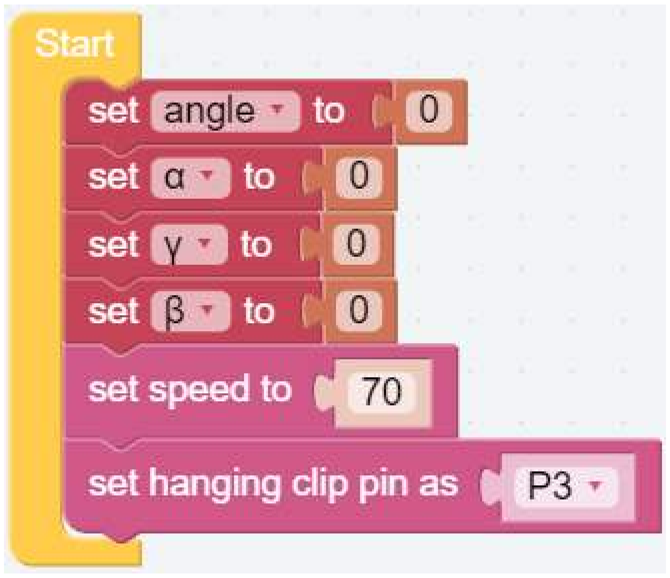
Step 2

Go to the remote control page and drag a D-pad and two buttons out again, because the import library will not import the widgets, so you need to drag them in again. Add two more buttons to control the angle of the Hanging Clip.



Step 3

Create the variables (, , and `angle`) and set the initial values to 0, then initialize the PiArm rotation speed and the pin of Hanging Clip.



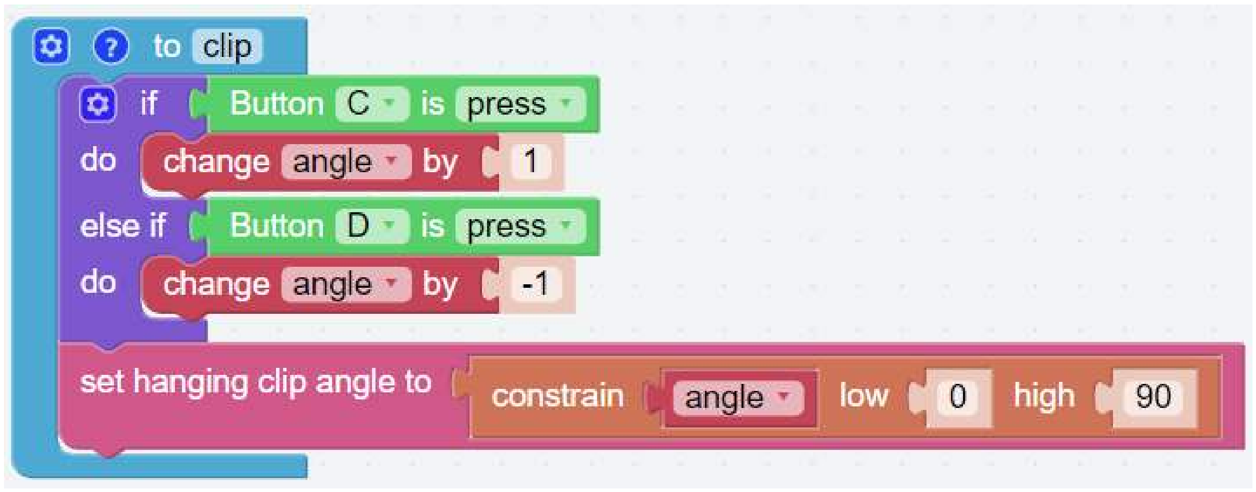
```
Start
  set angle to 0
  set α to 0
  set γ to 0
  set β to 0
  set speed to 70
  set hanging clip pin as P3
```

Step 4

Create a new function [clip], and write the code as follows to control *Hanging Clip* with two buttons.

- Use [if () else ()] block as a judgment condition. If button C is pressed, the variable `angle` is added by 1; if button D is pressed, the variable `angle` is subtracted by 1.

- Constrain the value of variable `angle` to 0 ~ 90 with [constrain () low() high ()] block.
- Set the angle of *Hanging Clip* according to the variable `angle`.



```
to clip
    if Button C is press
    do change angle by 1
    else if Button D is press
    do change angle by -1
    set hanging clip angle to constrain angle low 0 high 90
```

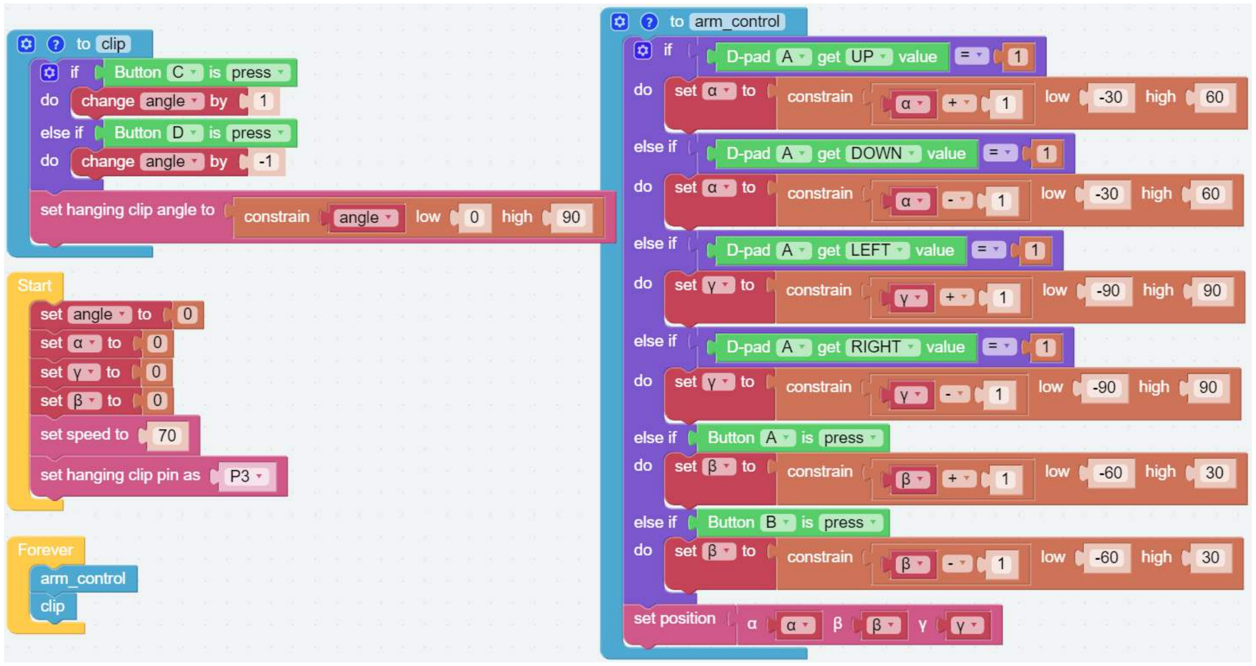
Step 5

Drag the [arm_control] and [clip] functions from the **Functions** category to the [Forever] block respectively.

After clicking the download button, use the D-pad and buttons A/B on the remote control page to control the movement of the arm, and then use buttons C/D to control the opening and closing of the Hanging Clip.

Note:

- The functions must be placed before the [start] and [Forever] blocks.
- You can also find the code with the same name on the EzBlock Studio **Examples** page and click Run or Edit directly to view the result.



```
to clip
    if Button C is press
    do change angle by 1
    else if Button D is press
    do change angle by -1
    set hanging clip angle to constrain angle low 0 high 90

Start
    set angle to 0
    set α to 0
    set γ to 0
    set β to 0
    set speed to 70
    set hanging clip pin as P3

Forever
    arm_control
    clip
```

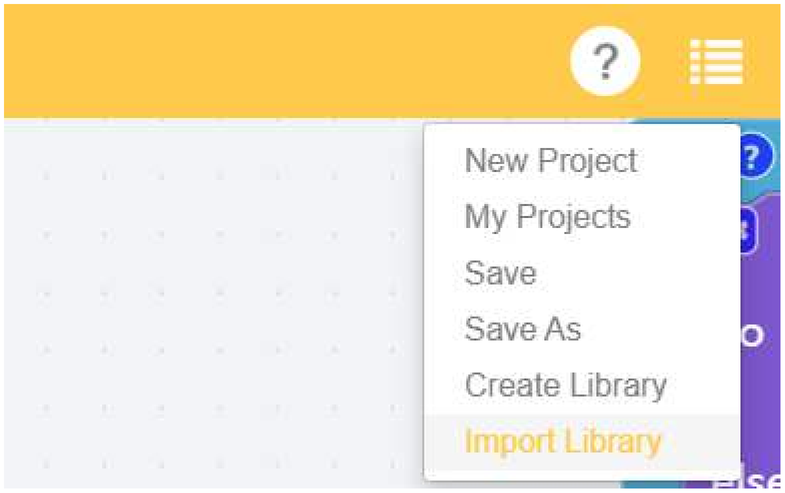
```
to arm_control
    if D-pad A get UP value = 1
    do set α to constrain α + 1 low -30 high 60
    else if D-pad A get DOWN value = 1
    do set α to constrain α - 1 low -30 high 60
    else if D-pad A get LEFT value = 1
    do set γ to constrain γ + 1 low -90 high 90
    else if D-pad A get RIGHT value = 1
    do set γ to constrain γ - 1 low -90 high 90
    else if Button A is press
    do set β to constrain β + 1 low -60 high 30
    else if Button B is press
    do set β to constrain β - 1 low -60 high 30
    set position α α β β γ γ
```

3.5.5 Electromagnet - Remote Control

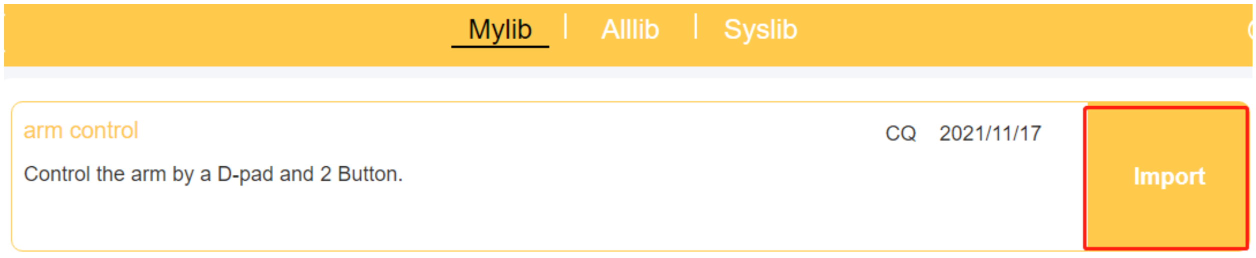
Create a new project and write code for it so that we can control the *Electromagnet* while controlling the arm.

Step 1

Import [arm_control] library, if you have not created this library before, please refer to: *Create a Library*.



In the **Mylib** page, select the library you created and click **Import**.

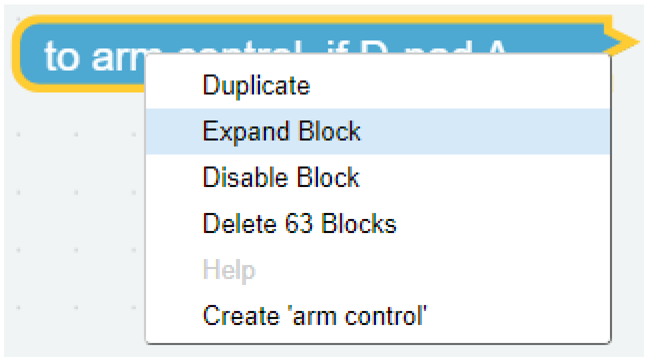


arm control

Control the arm by a D-pad and 2 Button.

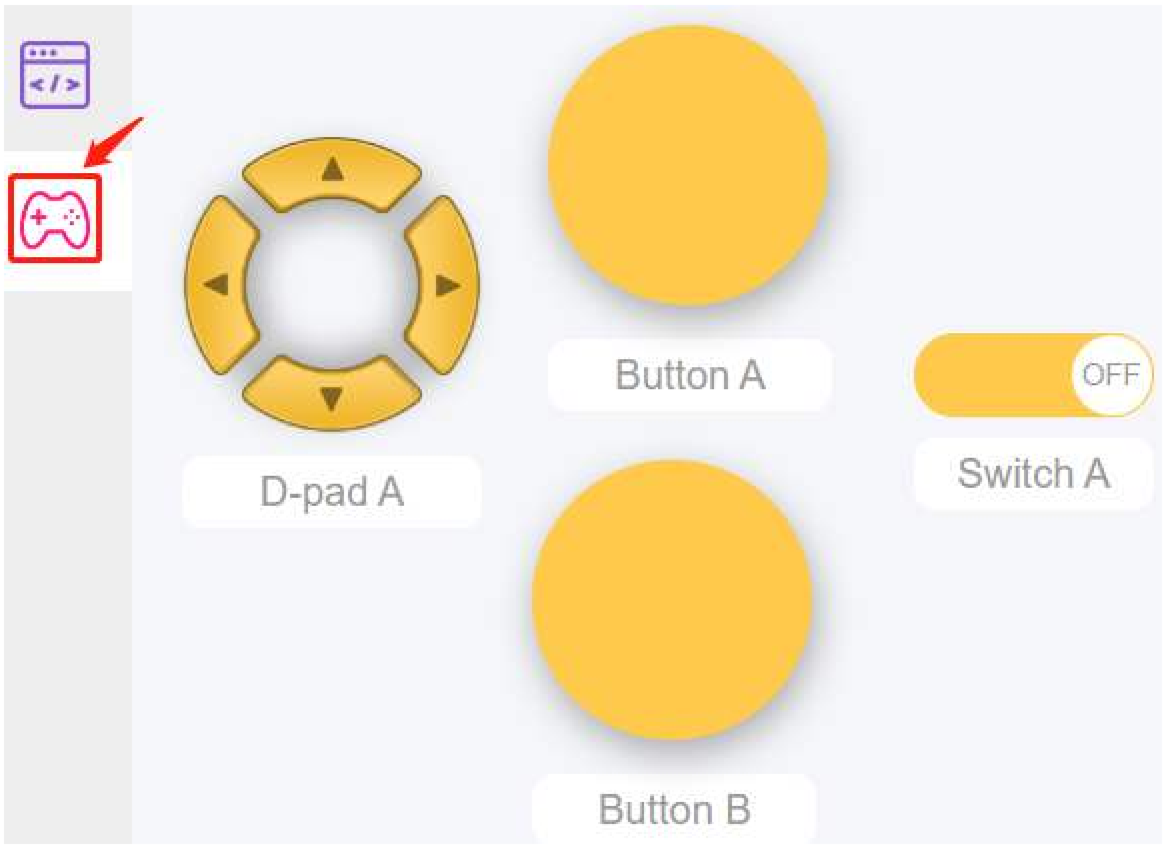
CQ 2021/11/17

After importing, this library is in a collapsed style. You can right-click on it and click **Expand Block**, so that you can see its internal code.



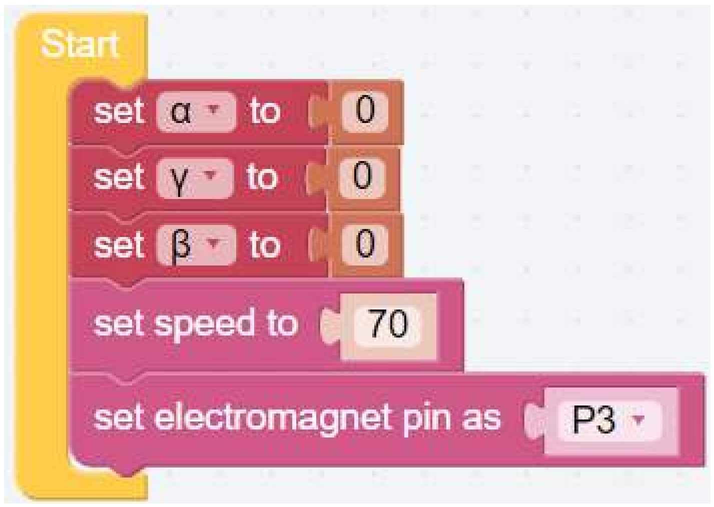
Step 2

Go to the remote control page and drag a D-pad and two buttons out again, because the import library will not import the widgets, so you need to drag them in again. In addition, add a **switch** widget to turn the Electromaget on/off.



Step 3

Create the variables (, , and) and set the initial values to 0, then initialize the PiArm rotation speed and the pin of Electromaget.

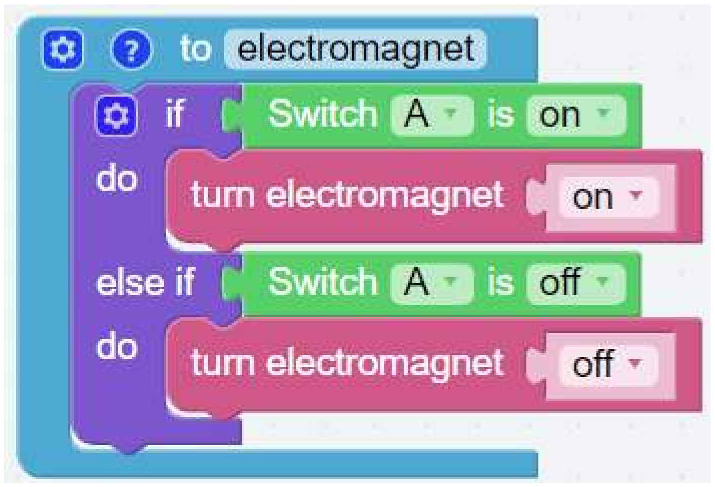


```
Start
  set α to 0
  set γ to 0
  set β to 0
  set speed to 70
  set electromagnet pin as P3
```

Step 4

Create a new function [electromagnet], and write code for it as the following steps, so that you can control *Electromagnet* by the **Switch** widget.

- Use [if () else ()] block as a judgment condition. If the switch is on, the Electromaget is activated; if the switch is off, the Electromaget is turned off.



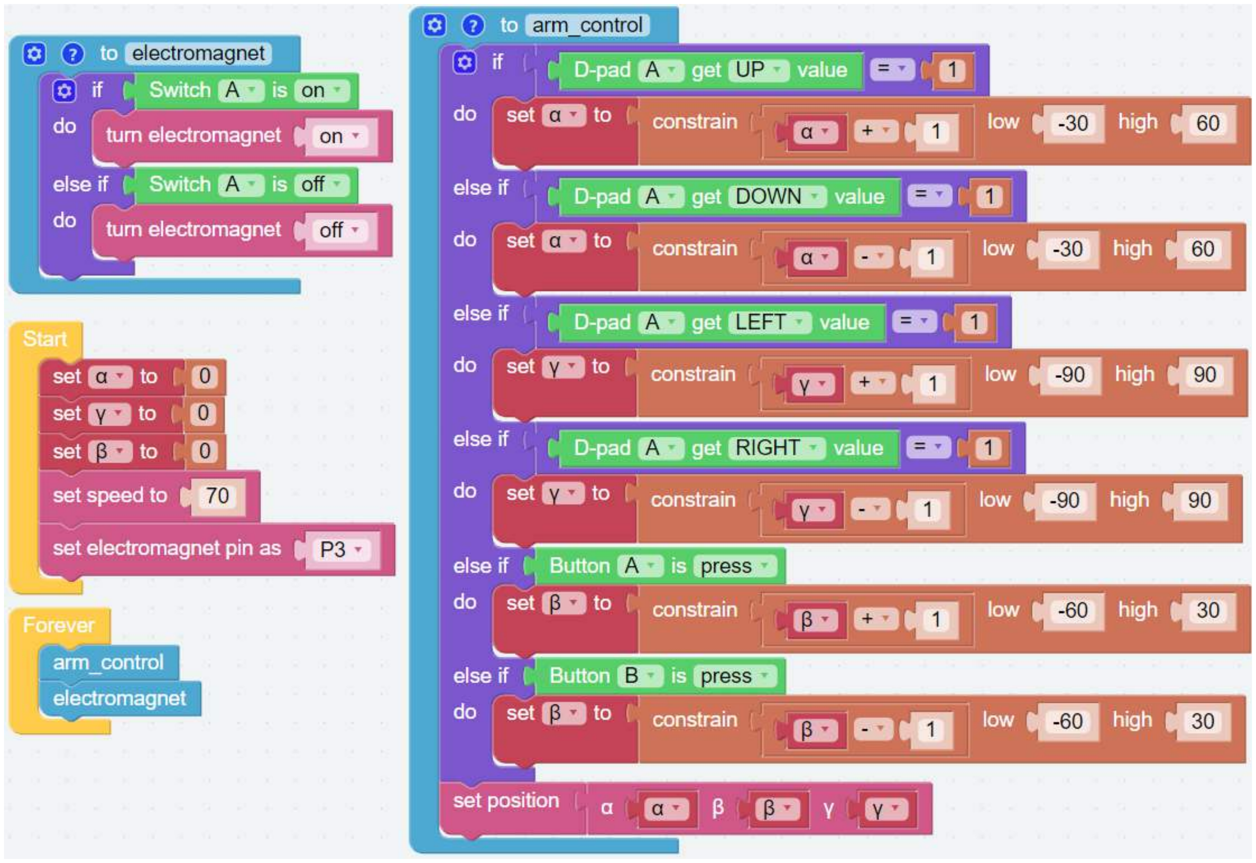
Step 5

Drag the [arm_control] and [electromaget] functions from the **Functions** category to the [Forever] block respectively.

After clicking the download button, when the switch toggles to on, the electromagnet turns on (it is magnetic at this time, you can use iron adsorption material); when the switch toggles to off, the electromagnet turns off. At the same time, you can use the D-pad and buttons A/B on the remote control page to control the movement of the arm.

Note:

- The functions must be placed before the [start] and [Forever] blocks.
- You can also find the code with the same name on the EzBlock Studio **Examples** page and click Run or Edit directly to view the result.



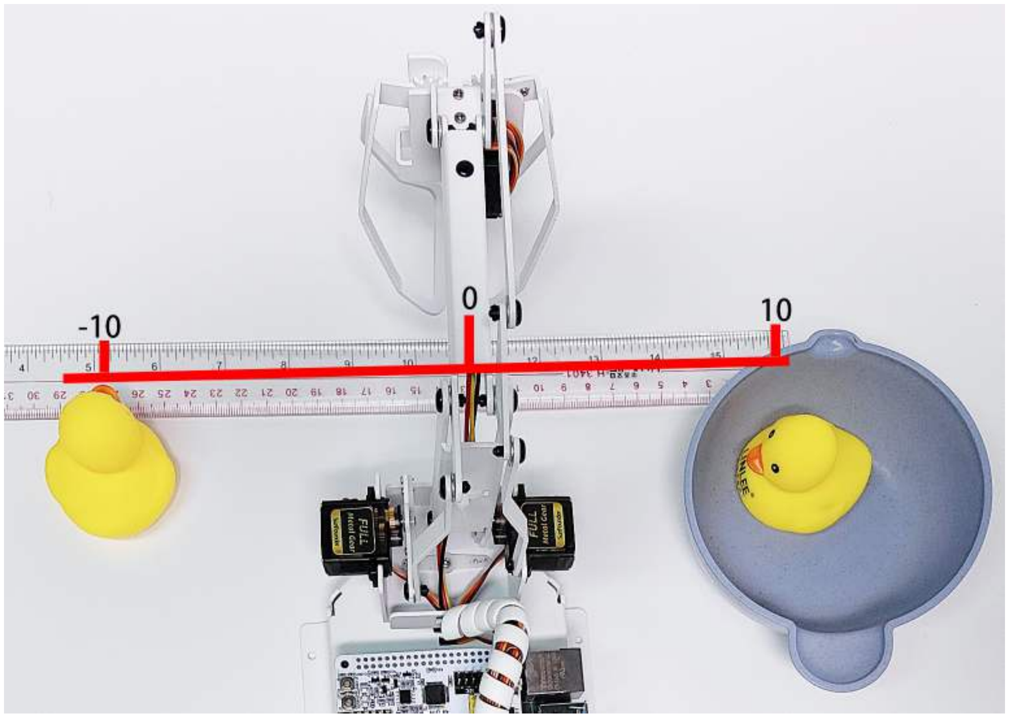
```
to electromagnet
    if Switch A is on
    do turn electromagnet on
    else if Switch A is off
    do turn electromagnet off

Start
    set α to 0
    set γ to 0
    set β to 0
    set speed to 70
    set electromagnet pin as P3

Forever
    arm_control
    electromagnet

to arm_control
    if D-pad A get UP value = 1
    do set α to constrain α + 1 low -30 high 60
    else if D-pad A get DOWN value = 1
    do set α to constrain α - 1 low -30 high 60
    else if D-pad A get LEFT value = 1
    do set γ to constrain γ + 1 low -90 high 90
    else if D-pad A get RIGHT value = 1
    do set γ to constrain γ - 1 low -90 high 90
    else if Button A is press
    do set β to constrain β + 1 low -60 high 30
    else if Button B is press
    do set β to constrain β - 1 low -60 high 30
    set position α α β β γ γ
```

3.6 Coordinate Mode

PiArm's arm can be controlled in two ways: *Angle Mode* and *Coordinate Mode*.

- *Angle Mode*: Writes a certain angle to the three servos on the arm, thus rotating the arm to a specific position.
- *Coordinate Mode*: Create a spatial right-angle coordinate system for the arm and set the control point. Set the coordinates of the control point so that the arm can reach a specific position.

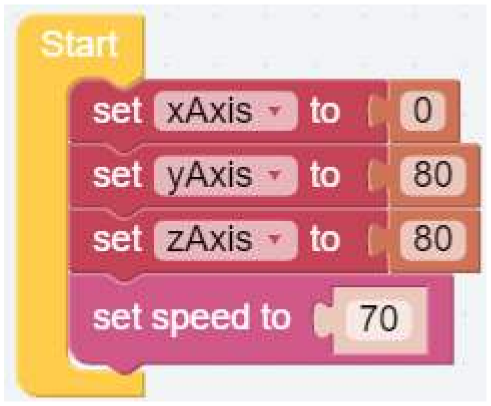
This project sets 2 coordinate points by coordinate mode, and let the PiArm clip the rubber duck on the left to the bowl on the right. But you need to mount *Hanging Clip* to the PiArm first.



3.6.1 Programming

Step 1

Initialize the pin of the Hanging Clip and set the speed of the robot arm to 60%.



```
Start
  set xAxis to 0
  set yAxis to 80
  set zAxis to 80
  set speed to 70
```

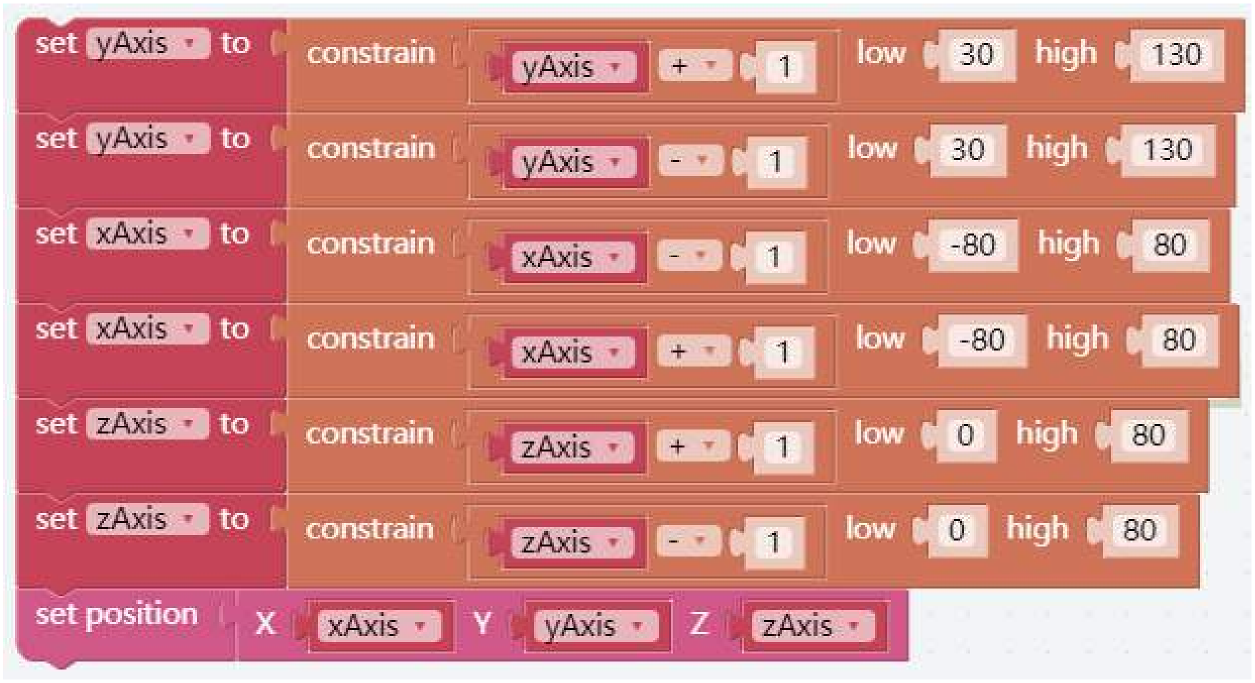
Step 2

Set the coordinates of the 2 points. Since the rubber duck on the left and the bowl on the right are on the same line, you will find that their Y coordinate values are the same.

- [start_coord]: The coordinates of the left rubber duck.
- [start_coord_up]: The coordinate of straight above the left rubber duck.
- [end_coord]: The coordinates of the bowl.
- [end_coord_up]: The coordinates straight above the bowl.

Note:

- All coordinates here refer to the coordinates of the control points, but the actual distance between the X and Y coordinates is a little larger when the end-of-arm tool is mounted.
- The tolerance distance is different for each end of arm tool. For example, 3-4cm for Hanging Clip and Electromagnet, 6-7cm for Shovel Bucket.
- For example, here the X coordinate is written as 100, but the actual distance is 13-14cm.
- It is generally recommended that the X coordinate is -80 ~ 80, but since the Y coordinate value here is small (the recommended range is 30~130), it is possible to reach to 100. However, if you increase the Y coordinate value, the X coordinate value needs to be reduced according to the actual situation due to the linkage action.

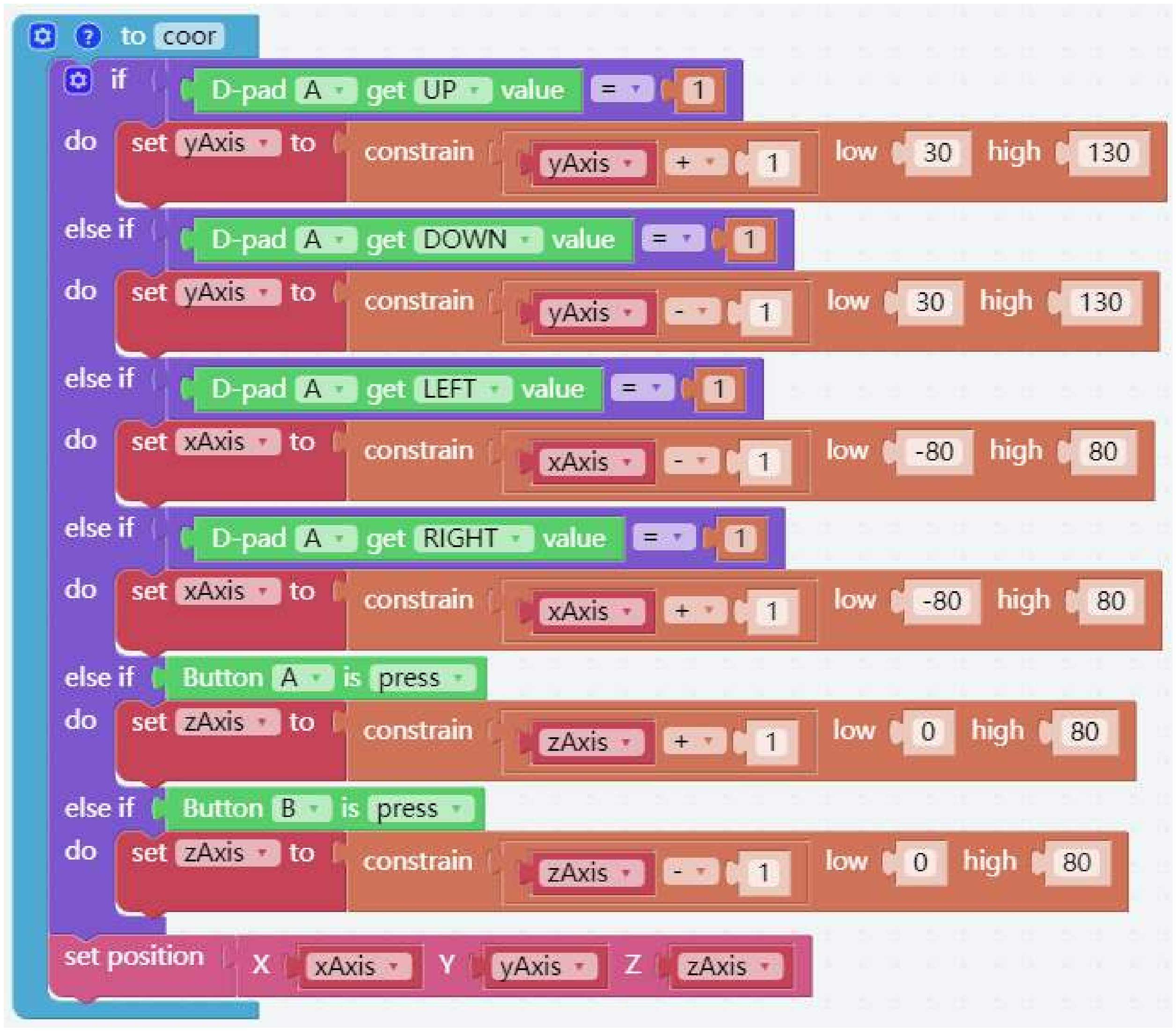


```
set yAxis to constrain yAxis + 1 low 30 high 130
set yAxis to constrain yAxis - 1 low 30 high 130
set xAxis to constrain xAxis - 1 low -80 high 80
set xAxis to constrain xAxis + 1 low -80 high 80
set zAxis to constrain zAxis + 1 low 0 high 80
set zAxis to constrain zAxis - 1 low 0 high 80
set position X xAxis Y yAxis Z zAxis
```

Step 3

In the [loop] block, let PiArm do the following.

- PiArm opens the Hanging Clip (20°), then rotates to the left rubber duck position (start_coord), then closes the Hanging Clip (90°).
- PiArm raises his head (start_coord_up) and then turns to the right side above the bowl (end_coord_up).
- PiArm lowers his head (end_coord_up), then opens the Hanging Clip (20°) to let the rubber duck fall into the bowl, and finally raises his head again (end_coord_up).

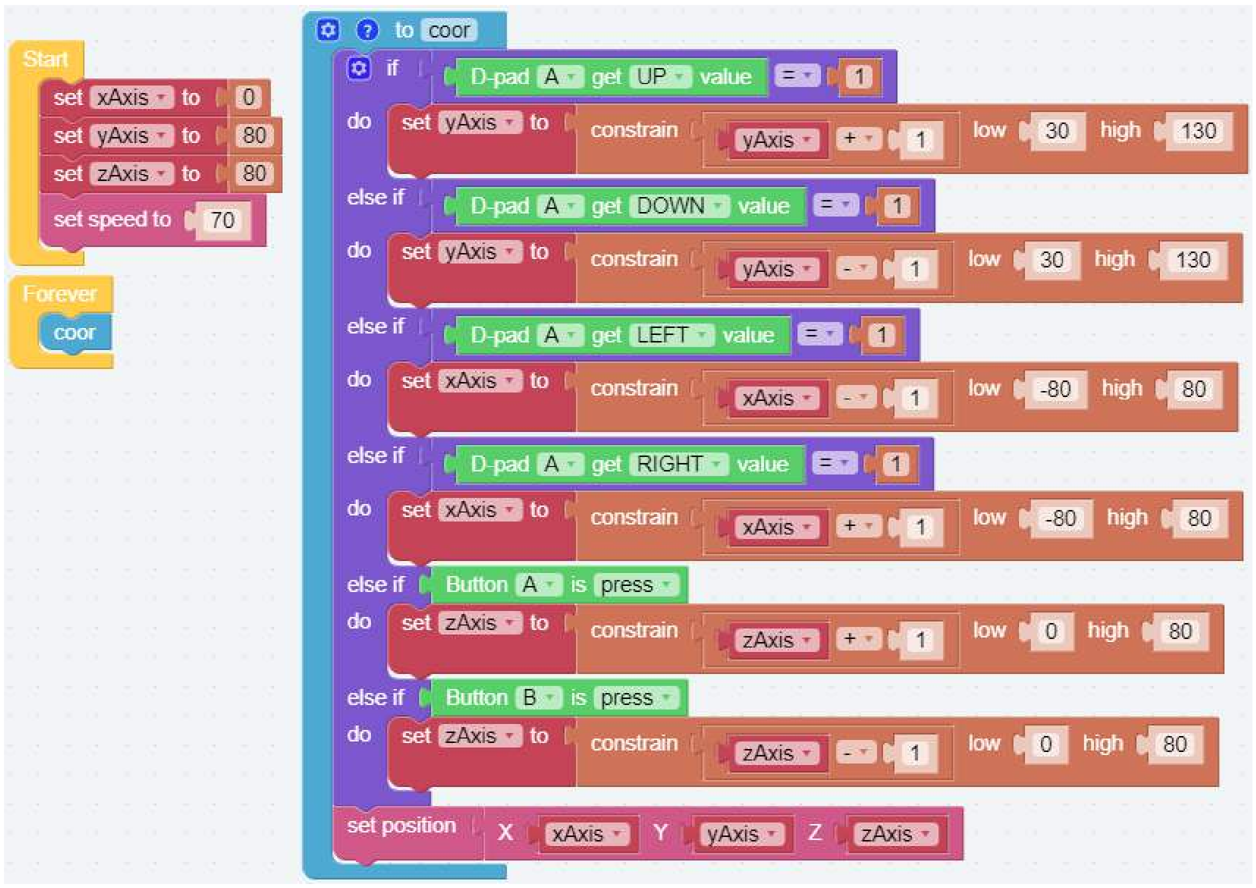


```
to coor
  if   D-pad A get UP value = 1
  do   set yAxis to constrain yAxis + 1 low 30 high 130
  else if   D-pad A get DOWN value = 1
  do   set yAxis to constrain yAxis - 1 low 30 high 130
  else if   D-pad A get LEFT value = 1
  do   set xAxis to constrain xAxis - 1 low -80 high 80
  else if   D-pad A get RIGHT value = 1
  do   set xAxis to constrain xAxis + 1 low -80 high 80
  else if   Button A is press
  do   set zAxis to constrain zAxis + 1 low 0 high 80
  else if   Button B is press
  do   set zAxis to constrain zAxis - 1 low 0 high 80
  set position X xAxis Y yAxis Z zAxis
```

Step 4

Click on the **Download** button in the bottom right corner and you will see PiArm repeating the actions described above.

Note: You can also find the code with the same name on the **Examples** page of Ezblock Studio and click Run or Edit directly to view the code.

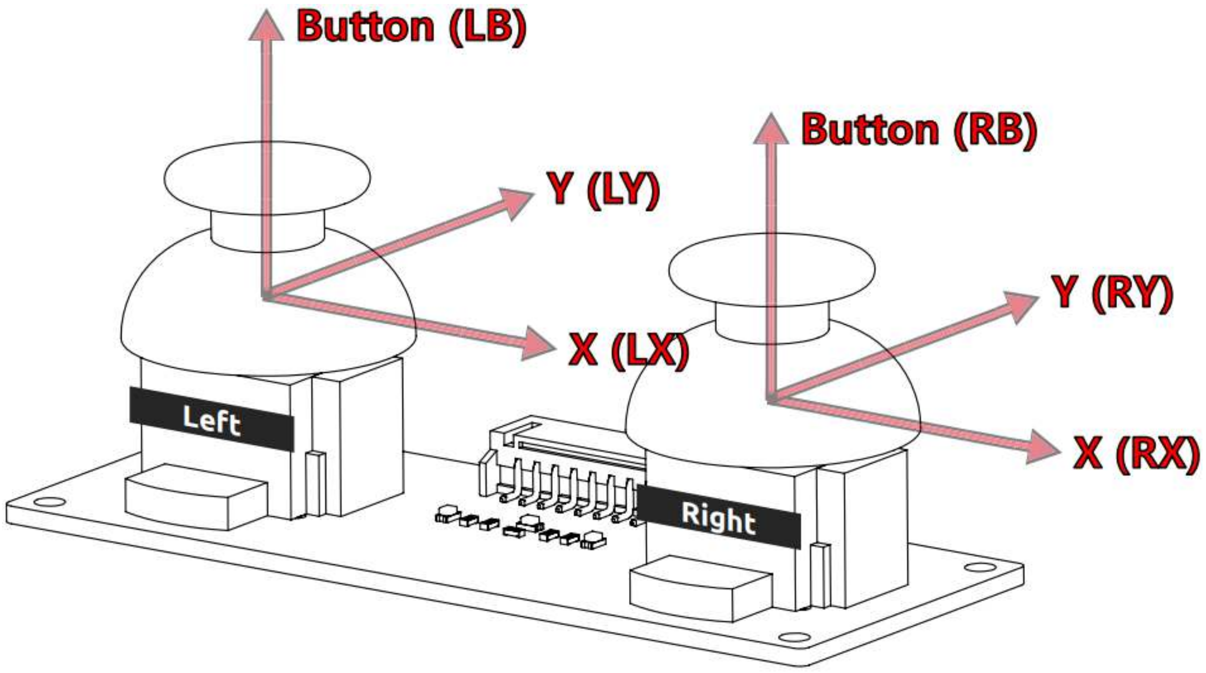


```
Start
  set xAxis to 0
  set yAxis to 80
  set zAxis to 80
  set speed to 70

Forever
  coor

to coor
  if D-pad A get UP value = 1
  do set yAxis to constrain yAxis + 1 low 30 high 130
  else if D-pad A get DOWN value = 1
  do set yAxis to constrain yAxis - 1 low 30 high 130
  else if D-pad A get LEFT value = 1
  do set xAxis to constrain xAxis - 1 low -80 high 80
  else if D-pad A get RIGHT value = 1
  do set xAxis to constrain xAxis + 1 low -80 high 80
  else if Button A is press
  do set zAxis to constrain zAxis + 1 low 0 high 80
  else if Button B is press
  do set zAxis to constrain zAxis - 1 low 0 high 80
  set position X xAxis Y yAxis Z zAxis
```

3.7 Memory Function

Piarm provides a function of recording actions, which can be used to record the actions that PiArm has done.

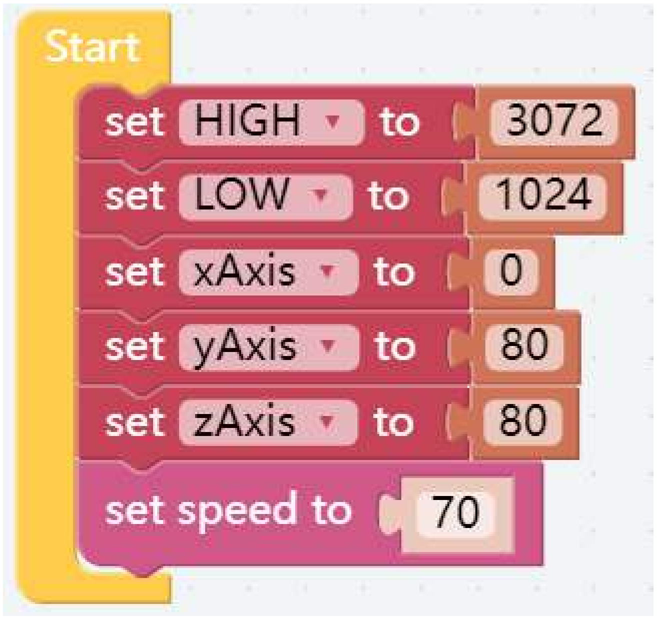
In this project, we will use the Dual Joystick Module to control the movement of the Arm of PiArm in Coordinate Control mode, and record the motion trajectory of the Arm through the joystick buttons so that PiArm can move repeatedly along the recorded trajectory.



3.7.1 Programming

Step 1

Create five variables (`HIGH`, `LOW`, `xAxis`, `yAxis` and `zAxis`) and set their initial values.



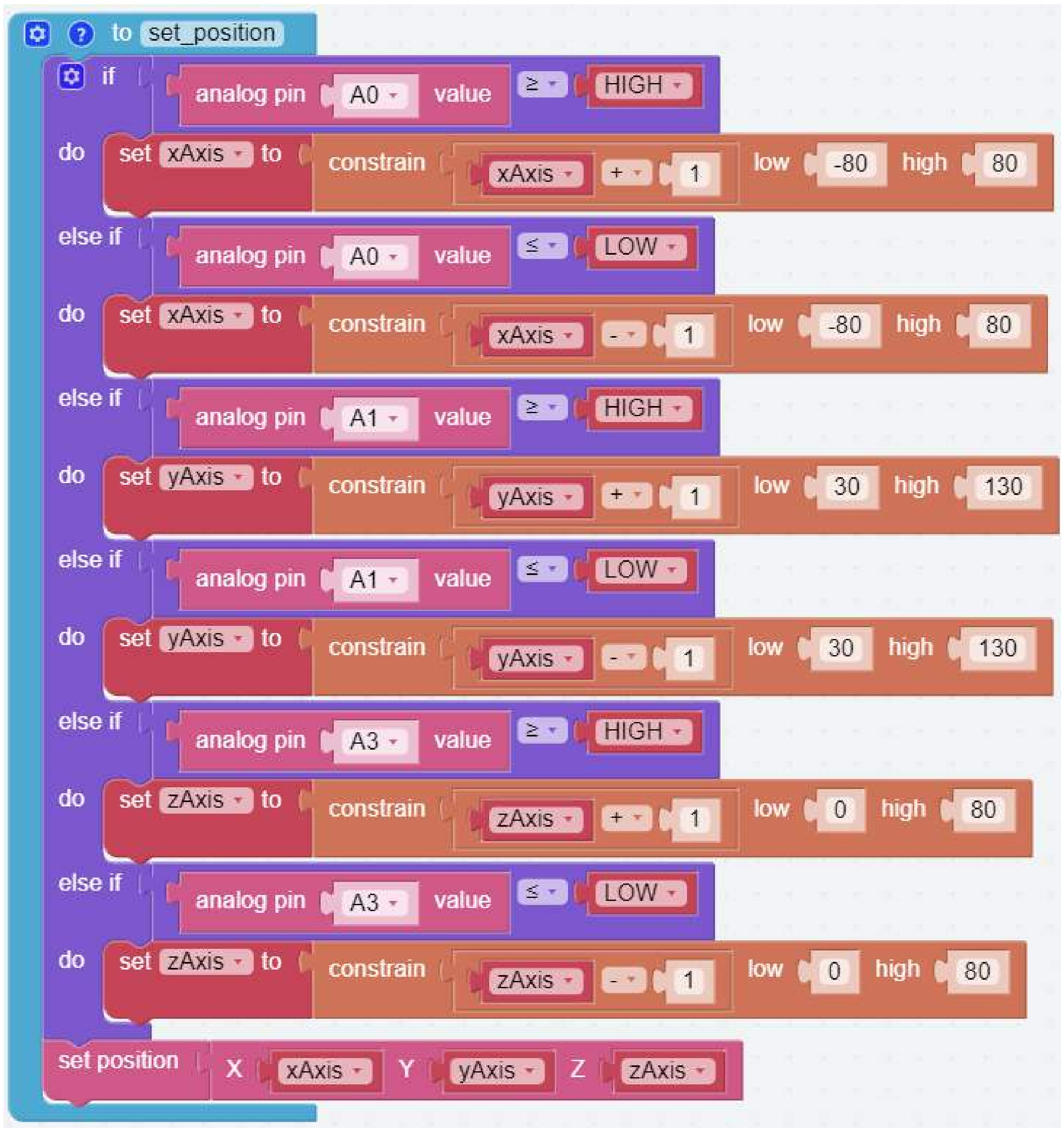
```
Start
  set HIGH to 3072
  set LOW to 1024
  set xAxis to 0
  set yAxis to 80
  set zAxis to 80
  set speed to 70
```

Step 2

Create a function named [set_position] to make the **Dual Joystick Module** move the PiArm in *Coordinate Mode*.

- If the **left joystick** is toggled to the right, the Arm will turn right.
- If the **left joystick** is toggled to the left, the Arm will turn left.

- If the **left joystick** is toggled forward, the Arm will extend forward.
- If the **left joystick** is toggled backward, the Arm will retract backward.
- If the **right joystick** is toggled forward, the Arm will raise up.
- If the **right joystick** is toggled backward, the Arm will lower down.



```
to set_position
  if analog pin A0 value ≥ HIGH
  do set xAxis to constrain xAxis + 1 low -80 high 80
  else if analog pin A0 value ≤ LOW
  do set xAxis to constrain xAxis - 1 low -80 high 80
  else if analog pin A1 value ≥ HIGH
  do set yAxis to constrain yAxis + 1 low 30 high 130
  else if analog pin A1 value ≤ LOW
  do set yAxis to constrain yAxis - 1 low 30 high 130
  else if analog pin A3 value ≥ HIGH
  do set zAxis to constrain zAxis + 1 low 0 high 80
  else if analog pin A3 value ≤ LOW
  do set zAxis to constrain zAxis - 1 low 0 high 80
  set position X xAxis Y yAxis Z zAxis
```

Note:

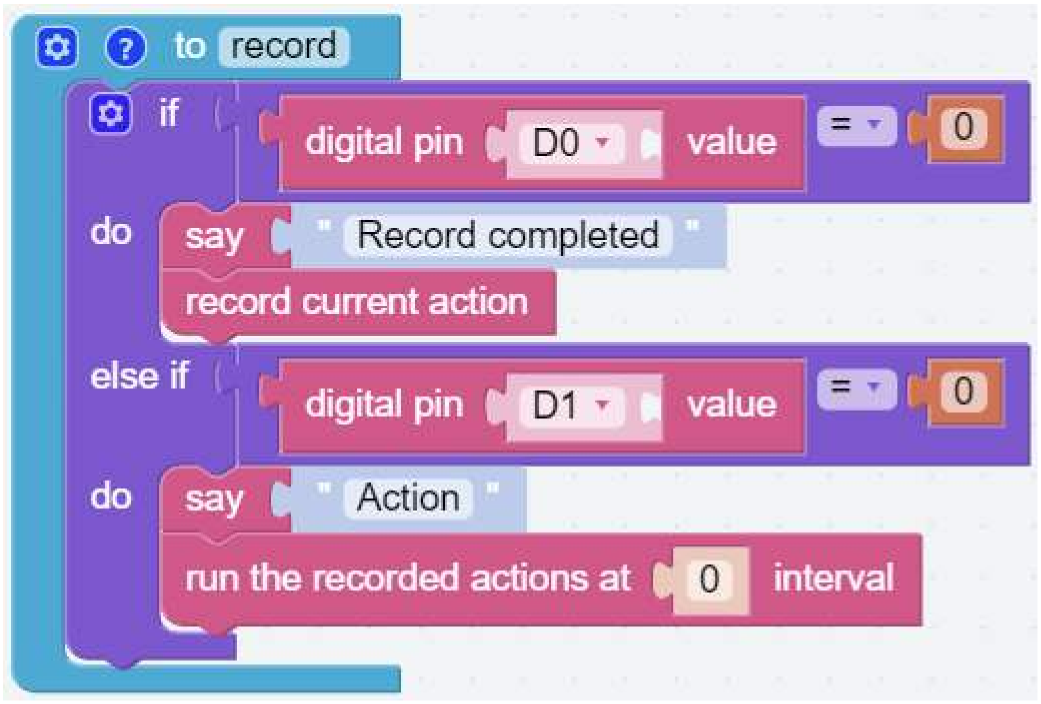
- About X, Y, Z coordinate directions, please refer to: *Coordinate Mode*.
- For the connection and direction of the dual joystick, refer to *Dual Joystick Module*.

- [constrain () low () high ()]: From Math category for setting the variation of a constant to a certain range.
- [if else]: Conditional judgment block, you can create multiple conditional judgments by clicking the set icon and dragging [else] or [else if] to the right below the [if].

Step 3

A new function, [record], is created to record the current actions and to allow PiArm to reproduce them.

- The left and right buttons of the Dual Joystick Module are connected to **D0 (Left Button)**, **D1 (Right Buttbon)** respectively.
- The buttons will output low level (0) when pressed and output high level (1) when released.
- When the **button of the left joystick** is pressed, the action of PiArm will be recorded at this time, and there will be a voice prompt to indicate the completion of recording.
- When the **button of the right joystick** is pressed, PiArm will reproduce these recorded actions.



```
to record
  if digital pin D0 value = 0
  do say "Record completed"
     record current action
  else if digital pin D1 value = 0
  do say "Action"
     run the recorded actions at 0 interval
```

Note:

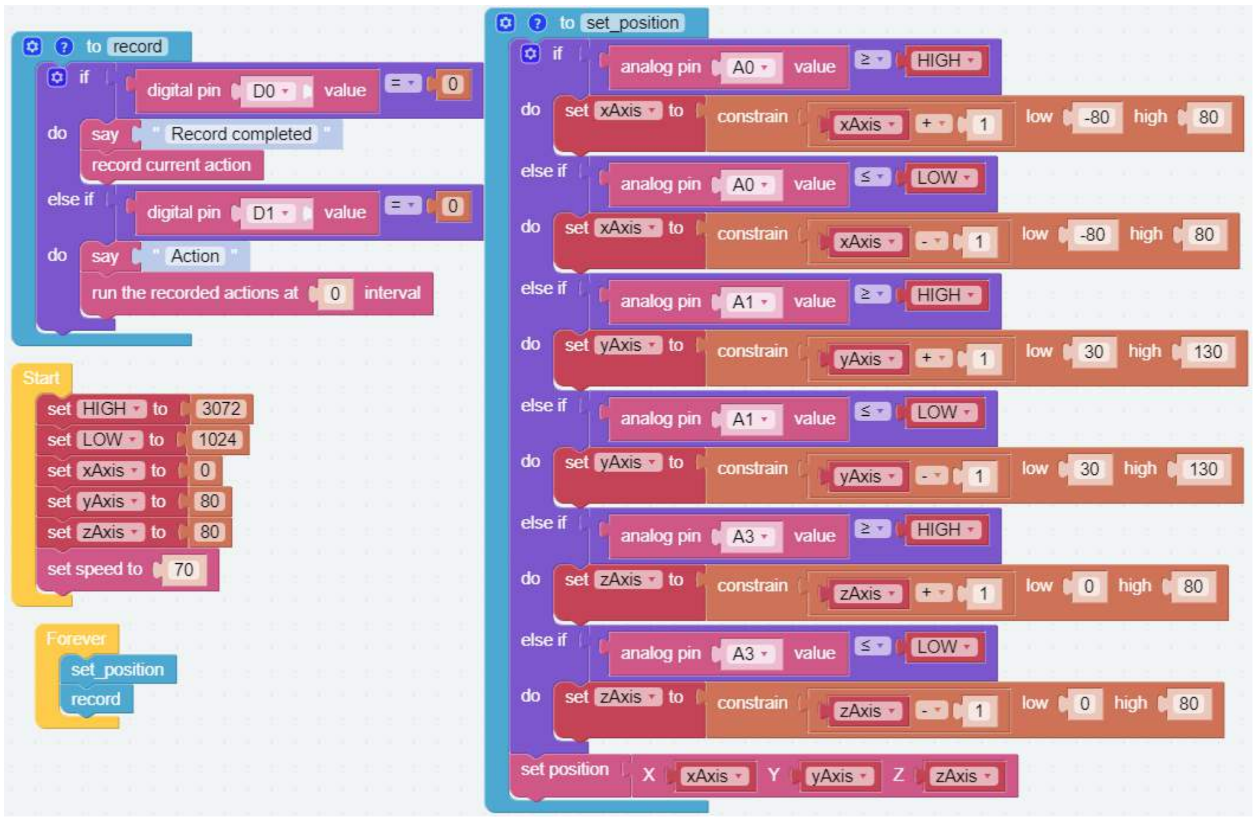
- The [if else], [and] and [=] blocks are all from the **Logic** category.
- [run the recorded actions at () internal]: This block is used to set the time interval for each set of recorded actions, if it is 0 it will reproduce each set of actions continuously.

Step 4

Put the [set_position] and [record] functions into the [Forever] block to execute them sequentially, and finally click the **Download** button to run the code.

Now you can use the joystick to control PiArm, press the **button of the left joystick** to record the desired actions, and after recording a few groups, press the **button of the right joystick** to make PiArm reproduce these actions.

Note: You can also find the code with the same name on the **Examples** page of **Ezblock Studio** and click **Run** or **Edit** directly to see the results.



```
to record
  if digital pin D0 value = 0
  do say "Record completed"
     record current action
  else if digital pin D1 value = 0
  do say "Action"
     run the recorded actions at 0 interval

Start
  set HIGH to 3072
  set LOW to 1024
  set xAxis to 0
  set yAxis to 80
  set zAxis to 80
  set speed to 70

Forever
  set_position
  record

to set_position
  if analog pin A0 value ≥ HIGH
  do set xAxis to constrain xAxis + 1 low -80 high 80
  else if analog pin A0 value ≤ LOW
  do set xAxis to constrain xAxis - 1 low -80 high 80
  else if analog pin A1 value ≥ HIGH
  do set yAxis to constrain yAxis + 1 low 30 high 130
  else if analog pin A1 value ≤ LOW
  do set yAxis to constrain yAxis - 1 low 30 high 130
  else if analog pin A3 value ≥ HIGH
  do set zAxis to constrain zAxis + 1 low 0 high 80
  else if analog pin A3 value ≤ LOW
  do set zAxis to constrain zAxis - 1 low 0 high 80
  set position X xAxis Y yAxis Z zAxis
```

3.7.2 What's More

You can also add separate EoAT control code to this project, so that you can control the **Arm** and **EoAT** of the PiArm at the same time.

- If you want to control *Shovel Bucket*, please refer to *Shovel Bucket - Joystick Control* to write the code.
- If you want to control *Hanging Clip*, please refer to *Hanging Clip - Joystick Control* to write the code.
- If you want to control *Electromagnet*, please refer to *Electromagnet - Joystick Control* to write the code.

3.8 GAME - Catching Dolls

Now let's play a game of catching dolls and see who can catch more dolls with PiArm in the given time. In order to play this game, we need to implement two functions, the first one is to control PiArm with the dual joystick module, and the second one is to timing, when the countdown is over, we can't control PiArm anymore. These two parts must be executed simultaneously.

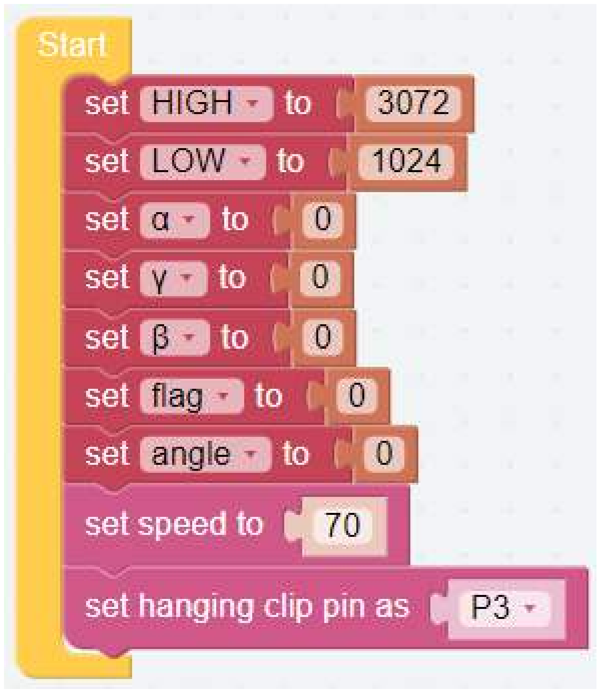
3.8.1 Programming

Step 1

Create five variables (`HIGH`, `LOW`, , , , `flag`, `angle`) and and set initial values for them. Then initialize the PiArm rotation speed and **Hanging Clip** pin.

Note:

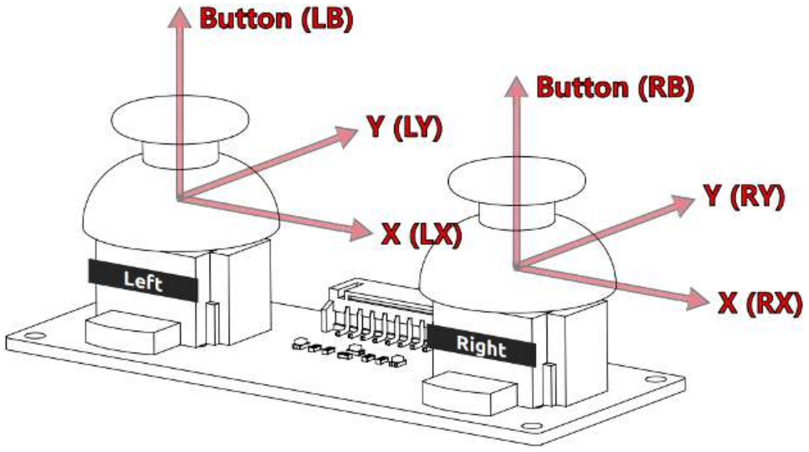
- For the reason of the values of the `HIGH` and `LOW` variables, please refer to *Dual Joystick Module*.
- , and represent the 3 servo rotation ranges on PiArm, refer to: *Angle Mode*.

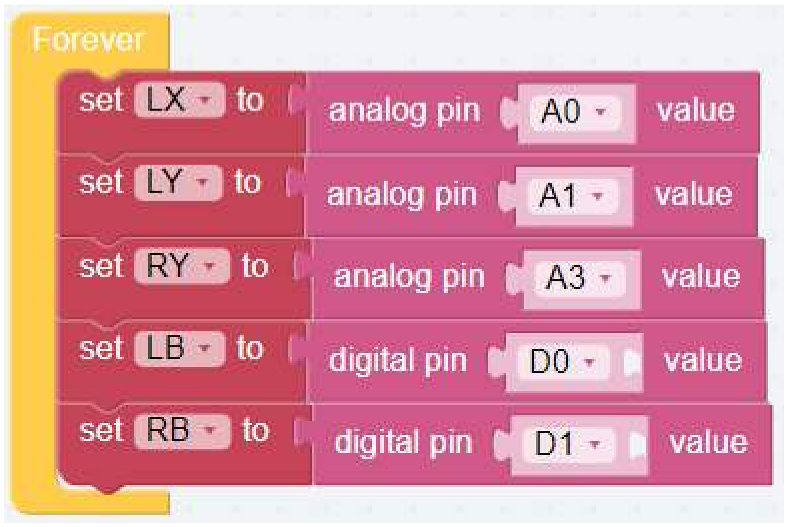


```
Start
  set HIGH to 3072
  set LOW to 1024
  set α to 0
  set γ to 0
  set β to 0
  set flag to 0
  set angle to 0
  set speed to 70
  set hanging clip pin as P3
```

Step 2

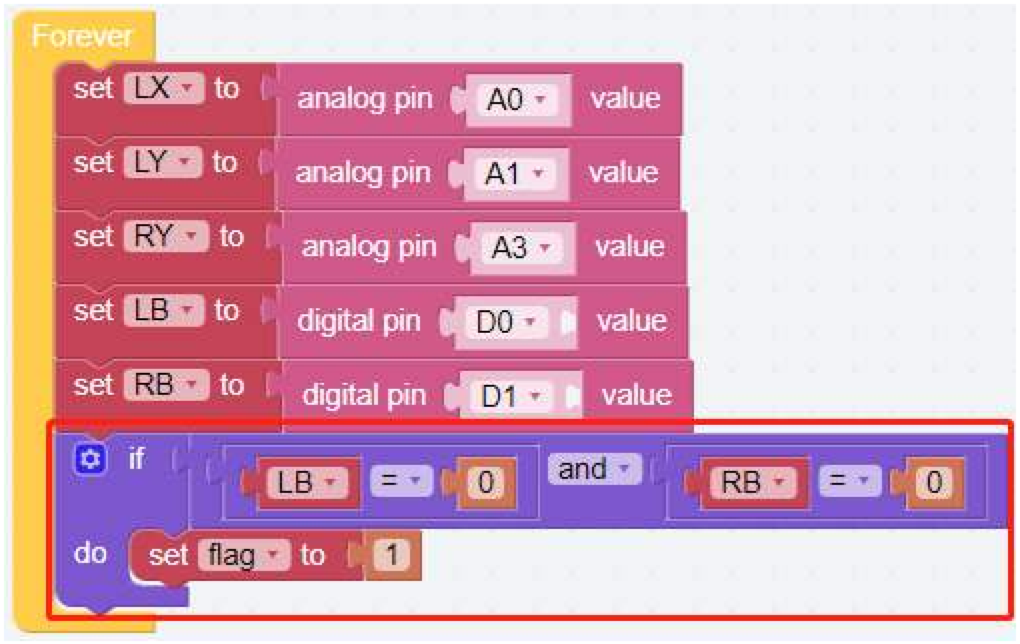
Create another 5 variables (`LX`, `LY`, `RY`, `LB`, `RB`) to read the X, Y and pressed values of the Dual Joystick Module respectively.




```
Forever
  set LX to analog pin A0 value
  set LY to analog pin A1 value
  set RY to analog pin A3 value
  set LB to digital pin D0 value
  set RB to digital pin D1 value
```

Step 3

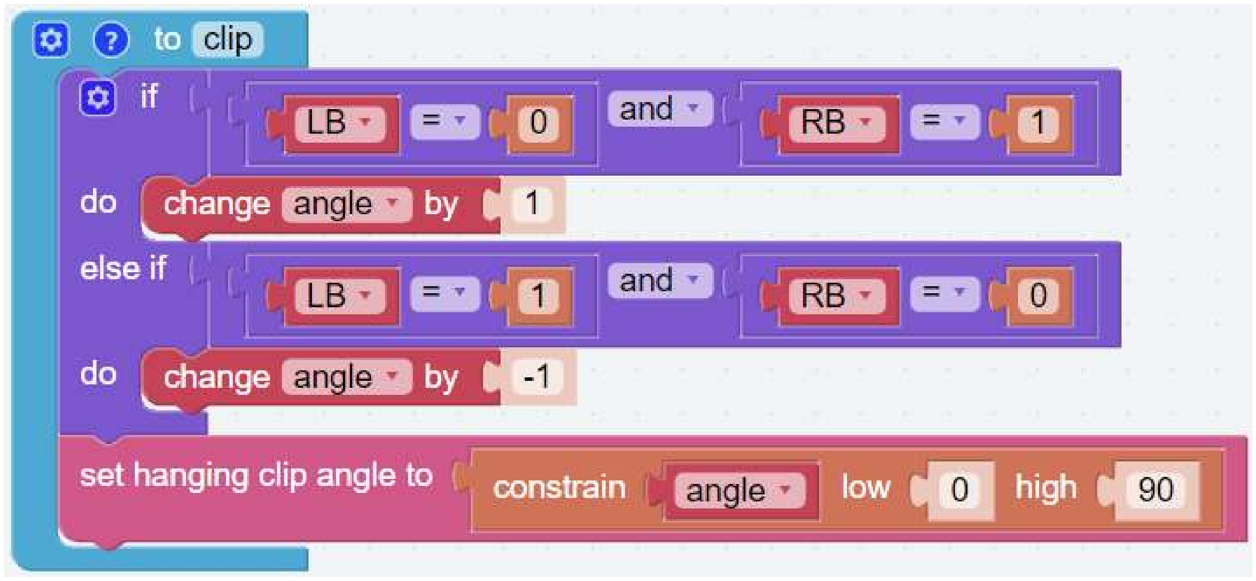
Set pressing the left and right joysticks at the same time as the game start action, so if `LB` and `RB` are read as 0 at the same time, it means the left and right joysticks are pressed, then the timing starts and the flag is set to 1.


```
Forever
  set LX to analog pin A0 value
  set LY to analog pin A1 value
  set RY to analog pin A3 value
  set LB to digital pin D0 value
  set RB to digital pin D1 value
  if LB = 0 and RB = 0
  do set flag to 1
```

Step 4

Create a function named [clip] to control the Hanging Clip.

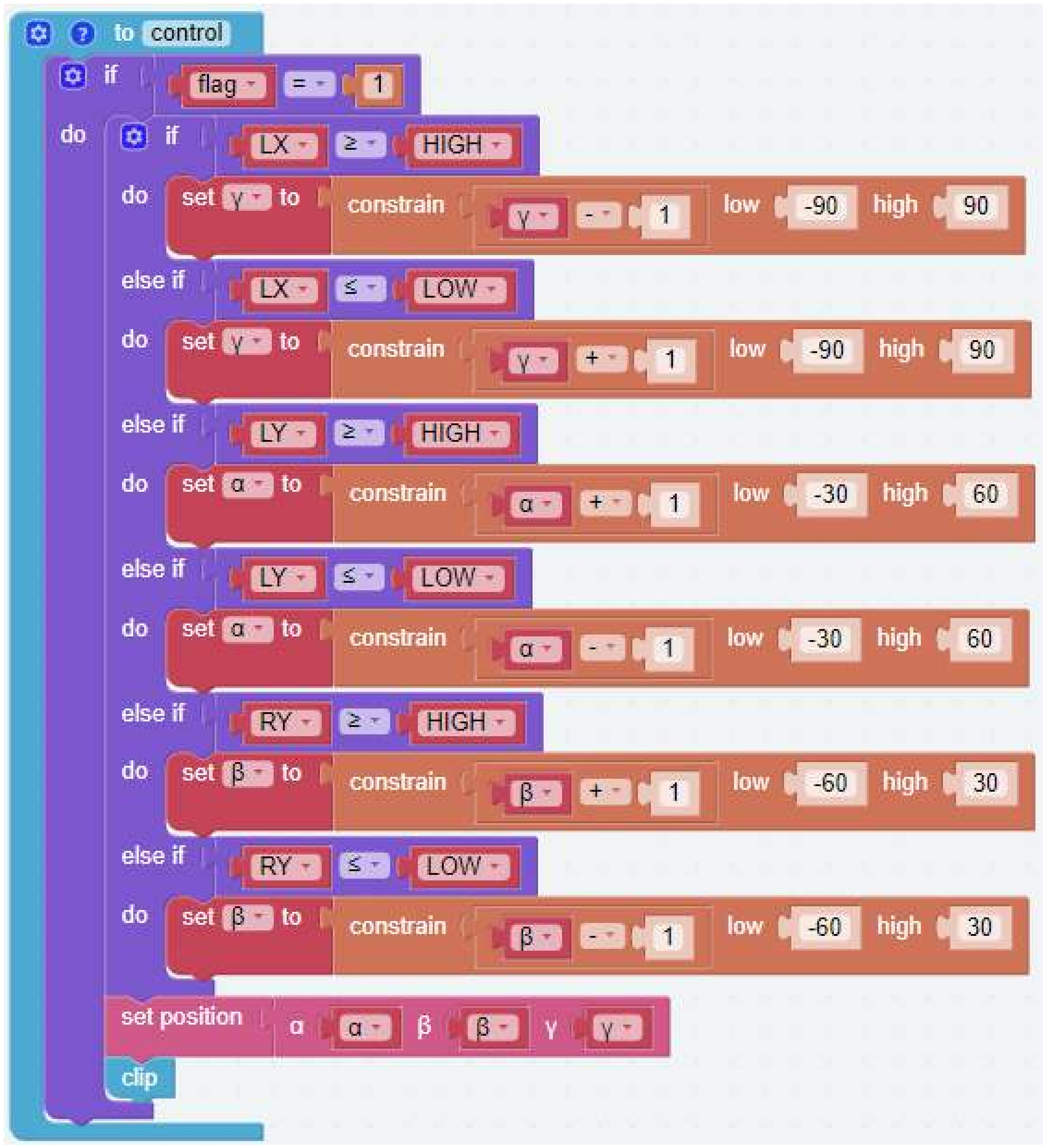
- When the left joystick is pressed and the right joystick is released, the *Hanging Clip* will slowly closed.
- When the left joystick is released and the right joystick is pressed, the *Hanging Clip* will slowly opened.



Step 5

Create a function [control] to set the rotation effect of PiArm based on the Dual Joystick Module.

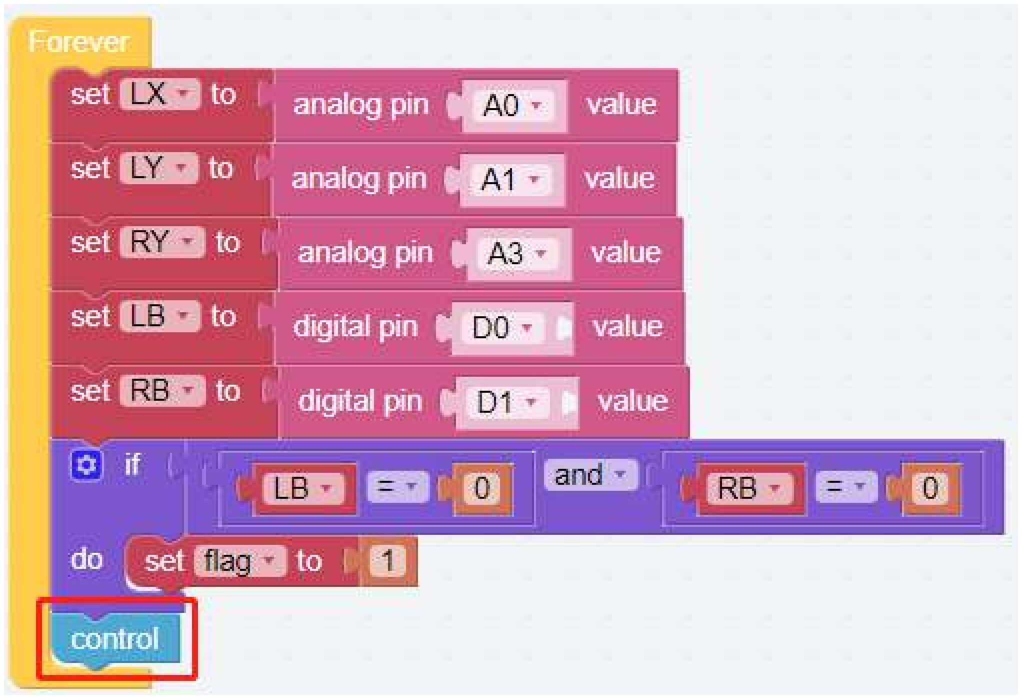
- When flag is 1, it means the game starts. At this time you can start to control PiArm.
- If the left joystick (`LX`) is toggled to the right, the Arm will turn right.
- If the left joystick (`LX`) is toggled to the left, the Arm will turn left.
- If the left joystick (`LY`) is toggled forward, the Arm will extend forward.
- If the left joystick (`LY`) is toggled backward, the Arm will retract backward.
- If the right joystick (`RY`) is toggled forward, the Arm will lower down.
- If the right joystick (`RY`) is toggled backward, the Arm will raise up.
- The Hanging Clip control function is also called here. This allows you to control both the Arm and Hanging Clip of the PiArm.



```
to control
  if flag = 1
  do  if LX ≥ HIGH
      do  set γ to constrain γ - 1 low -90 high 90
      else if LX ≤ LOW
      do  set γ to constrain γ + 1 low -90 high 90
      else if LY ≥ HIGH
      do  set α to constrain α + 1 low -30 high 60
      else if LY ≤ LOW
      do  set α to constrain α - 1 low -30 high 60
      else if RY ≥ HIGH
      do  set β to constrain β + 1 low -60 high 30
      else if RY ≤ LOW
      do  set β to constrain β - 1 low -60 high 30
      set position α α β β γ γ
      clip
```

Step 6

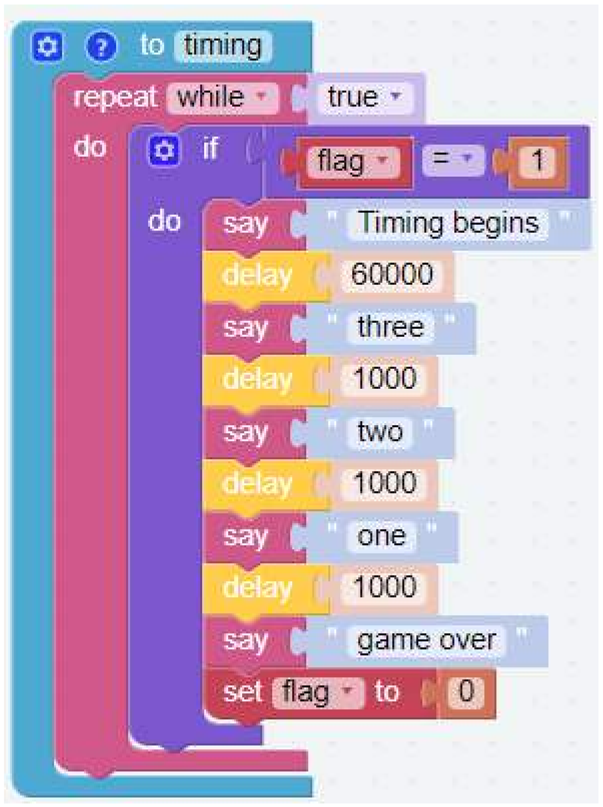
Put the [control] function into the [Forever] block.



```
Forever
  set LX to analog pin A0 value
  set LY to analog pin A1 value
  set RY to analog pin A3 value
  set LB to digital pin D0 value
  set RB to digital pin D1 value
  if LB = 0 and RB = 0
  do set flag to 1
  control
```

Step 7

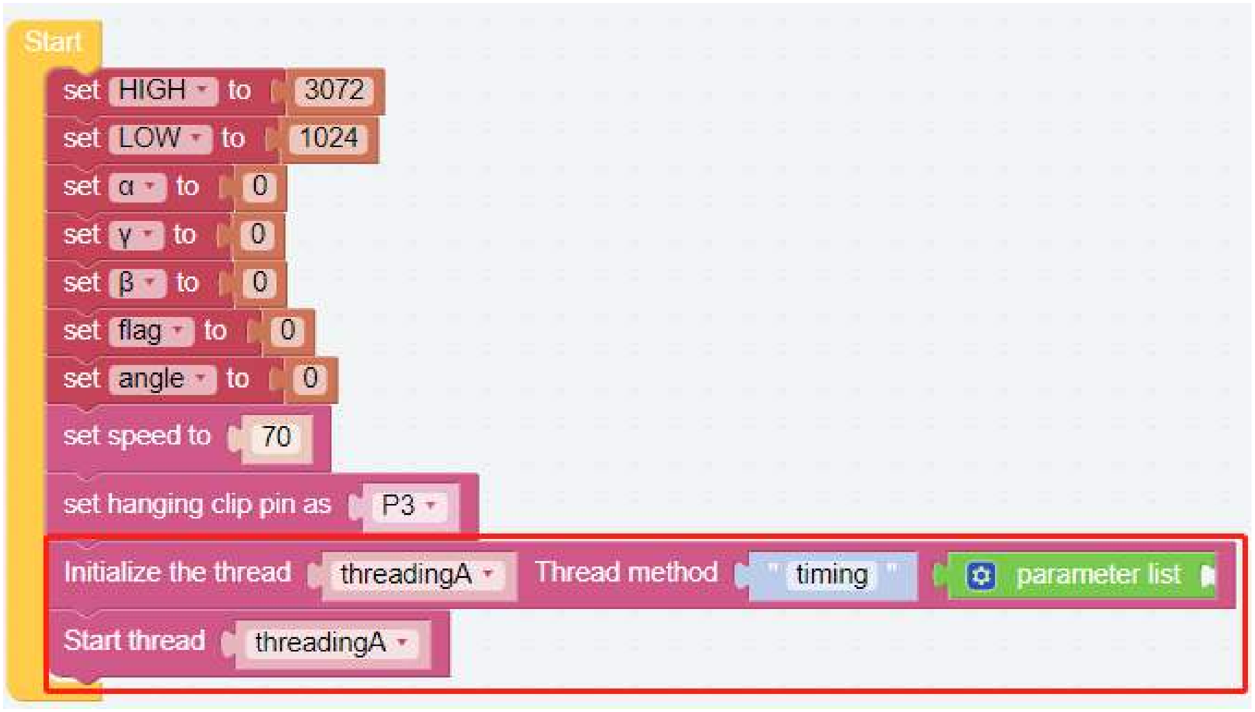
Create a function named [timing] to use for timing. The game time is set to 60 seconds (60000), and a countdown will chime in the last 3 seconds to let you know that time is almost up.



```
to timing
  repeat while true
  do if flag = 1
     do say " Timing begins "
        delay 60000
        say " three "
        delay 1000
        say " two "
        delay 1000
        say " one "
        delay 1000
        say " game over "
        set flag to 0
```

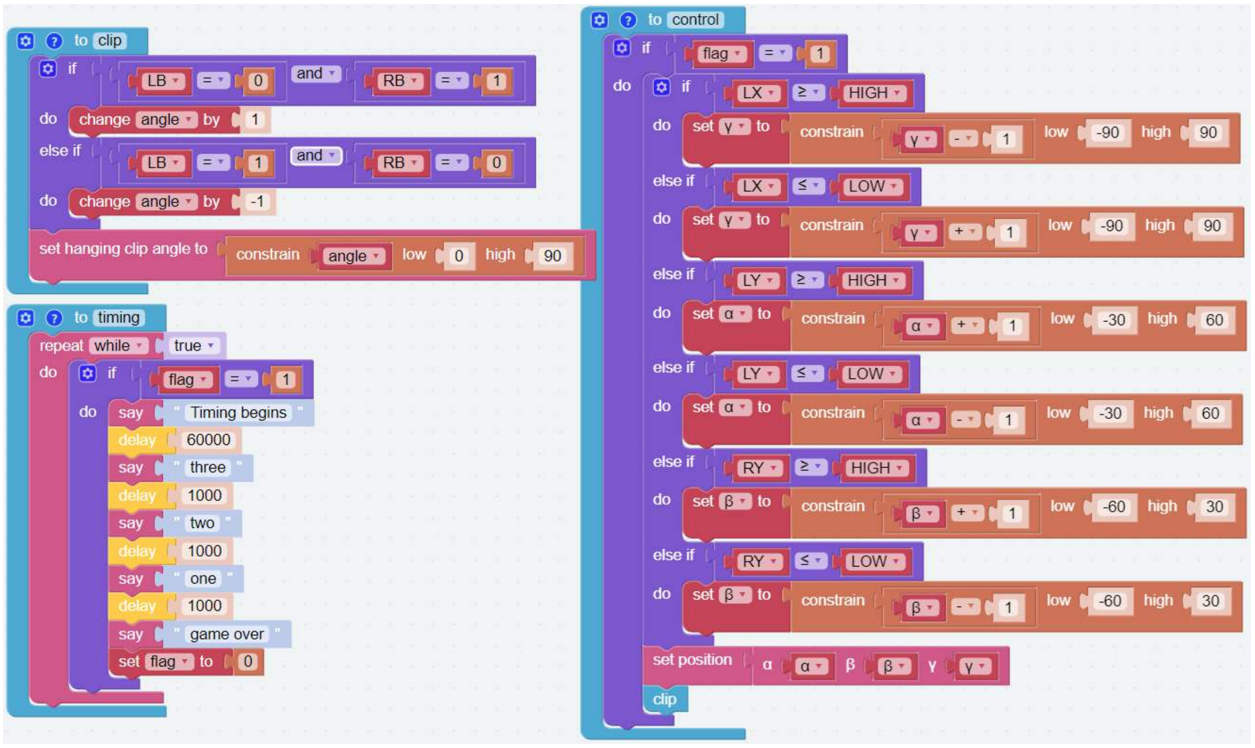
Step 8

Let the [timing] function run in a separate thread. This allows you to control PiArm while counting down.

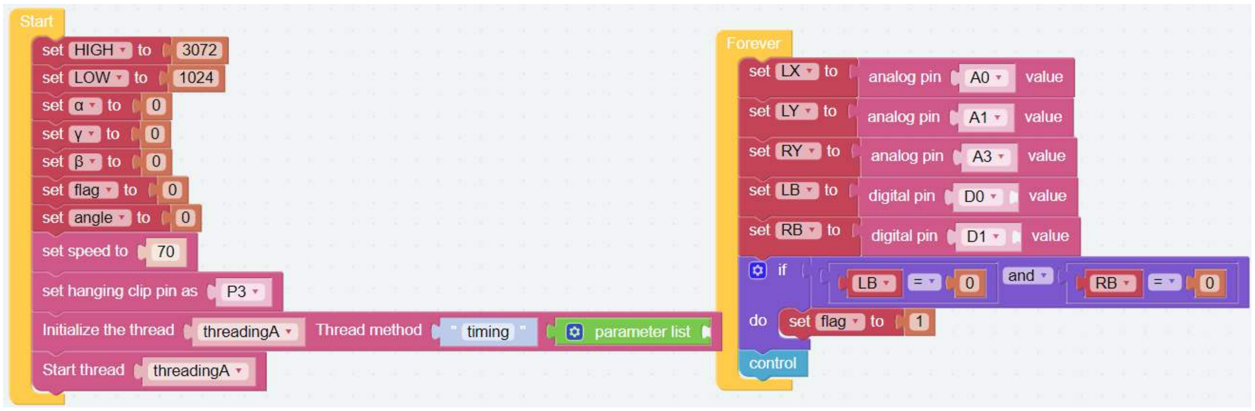


```
Start
  set HIGH to 3072
  set LOW to 1024
  set α to 0
  set γ to 0
  set β to 0
  set flag to 0
  set angle to 0
  set speed to 70
  set hanging clip pin as P3
  Initialize the thread threadingA Thread method " timing " parameter list
  Start thread threadingA
```

The complete code is as follows:



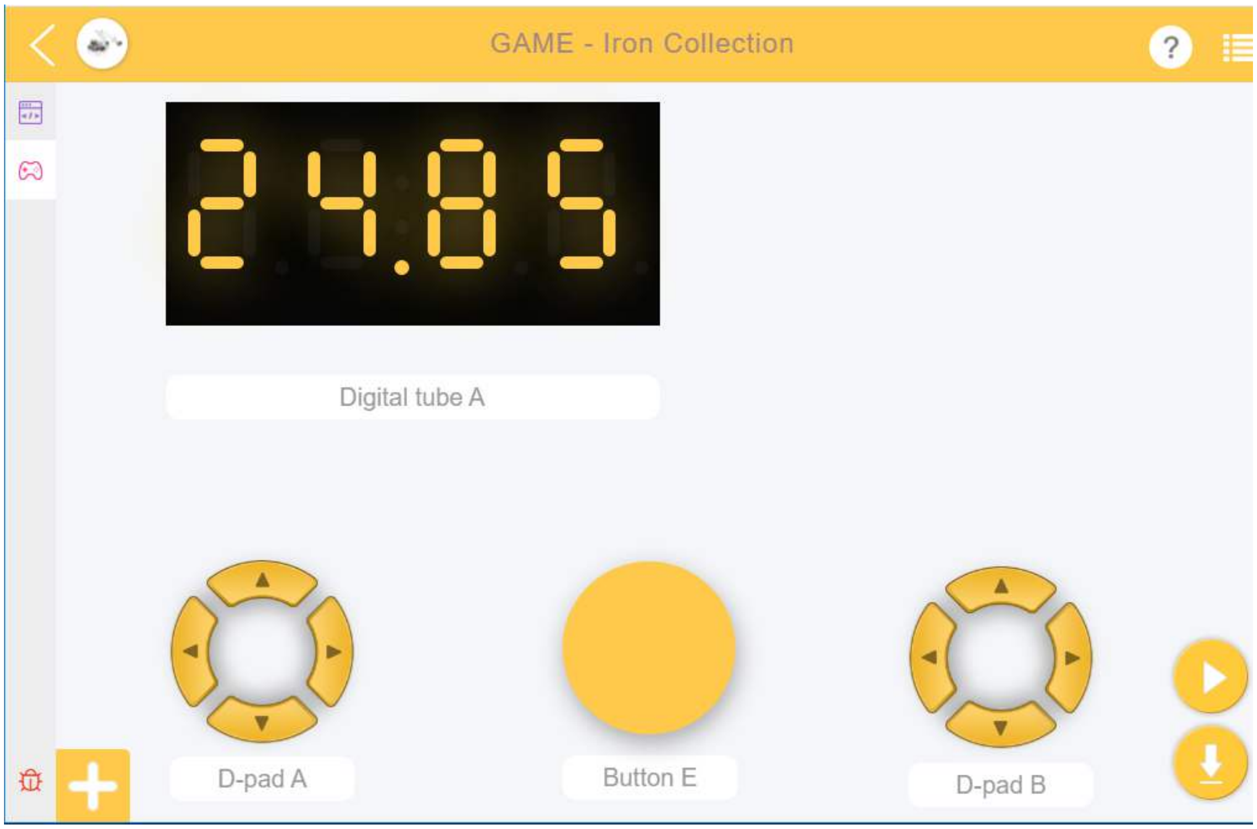
```
to clip
  if LB = 0 and RB = 1
  do  change angle by 1
  else if LB = 1 and RB = 0
  do  change angle by -1
  set hanging clip angle to constrain angle low 0 high 90

to timing
  repeat while true
  do  if flag = 1
      do  say " Timing begins "
          delay 60000
          say " three "
          delay 1000
          say " two "
          delay 1000
          say " one "
          delay 1000
          say " game over "
          set flag to 0

to control
  if flag = 1
  do  if LX ≥ HIGH
      do  set γ to constrain γ - 1 low -90 high 90
      else if LX ≤ LOW
      do  set γ to constrain γ + 1 low -90 high 90
      else if LY ≥ HIGH
      do  set α to constrain α + 1 low -30 high 60
      else if LY ≤ LOW
      do  set α to constrain α - 1 low -30 high 60
      else if RY ≥ HIGH
      do  set β to constrain β + 1 low -60 high 30
      else if RY ≤ LOW
      do  set β to constrain β - 1 low -60 high 30
      set position α α β β γ γ
      clip
```



```
Start
  set HIGH to 3072
  set LOW to 1024
  set α to 0
  set γ to 0
  set β to 0
  set flag to 0
  set angle to 0
  set speed to 70
  set hanging clip pin as P3
  Initialize the thread threadingA Thread method " timing " parameter list
  Start thread threadingA

Forever
  set LX to analog pin A0 value
  set LY to analog pin A1 value
  set RY to analog pin A3 value
  set LB to digital pin D0 value
  set RB to digital pin D1 value
  if LB = 0 and RB = 0
  do set flag to 1
control
```

3.9 GAME - Iron Collection

In this project, prepare 3 shapes of iron pieces: triangle, circle and square, PiArm will randomly say a shape, you need to control PiArm to put the corresponding shape of iron pieces into the corresponding box in the specified time, you will not be able to control PiArm when the time is over.



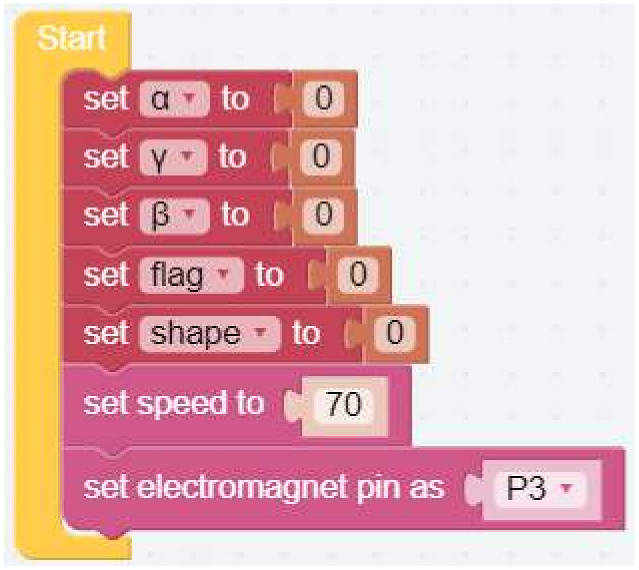
3.9.1 Programming

Step 1

Create five variables (, , , `flag`, `shape`) and and set initial values for them. Then initialize the PiArm rotation speed and **Electromagnet** pin.

Note:

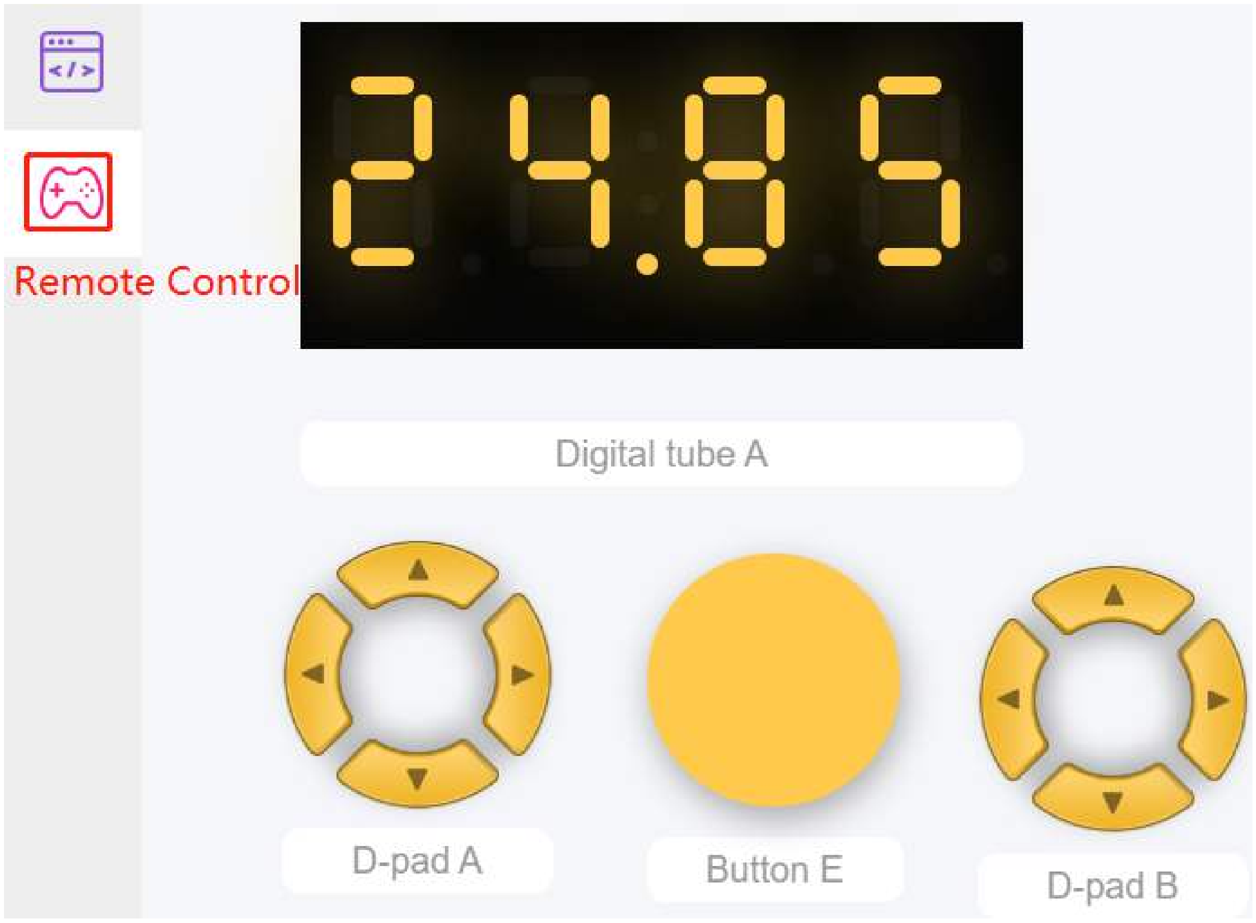
- , and represent the 3 servo rotation ranges on PiArm, refer to: *Angle Mode*.



```
Start
  set α to 0
  set γ to 0
  set β to 0
  set flag to 0
  set shape to 0
  set speed to 70
  set electromagnet pin as P3
```

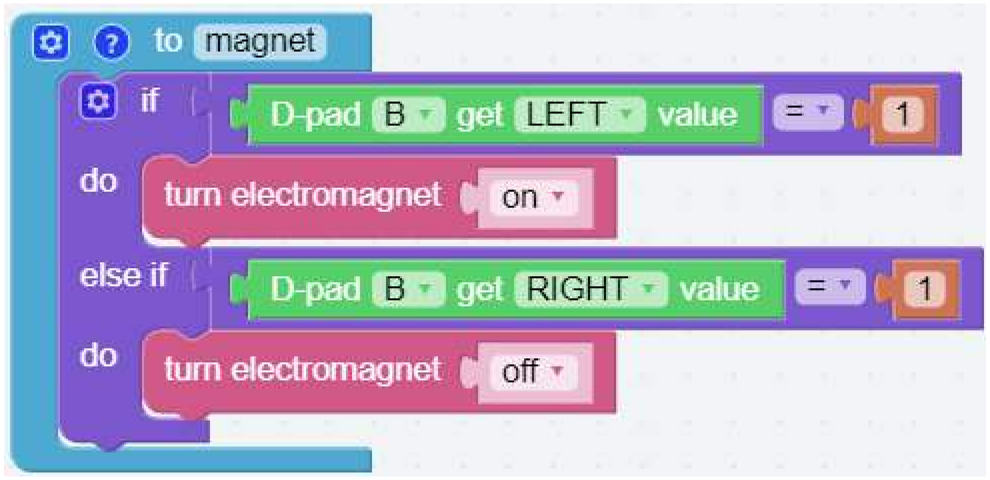
Step 2

Drag 2 D-pads from the remote control page to control PiArm, a button to start the game, and a Digital Tube to display the time.



Step 3

Create a function named [magnet] to enable the left and right control of the D-pad B to turn the electromagnet on and off.



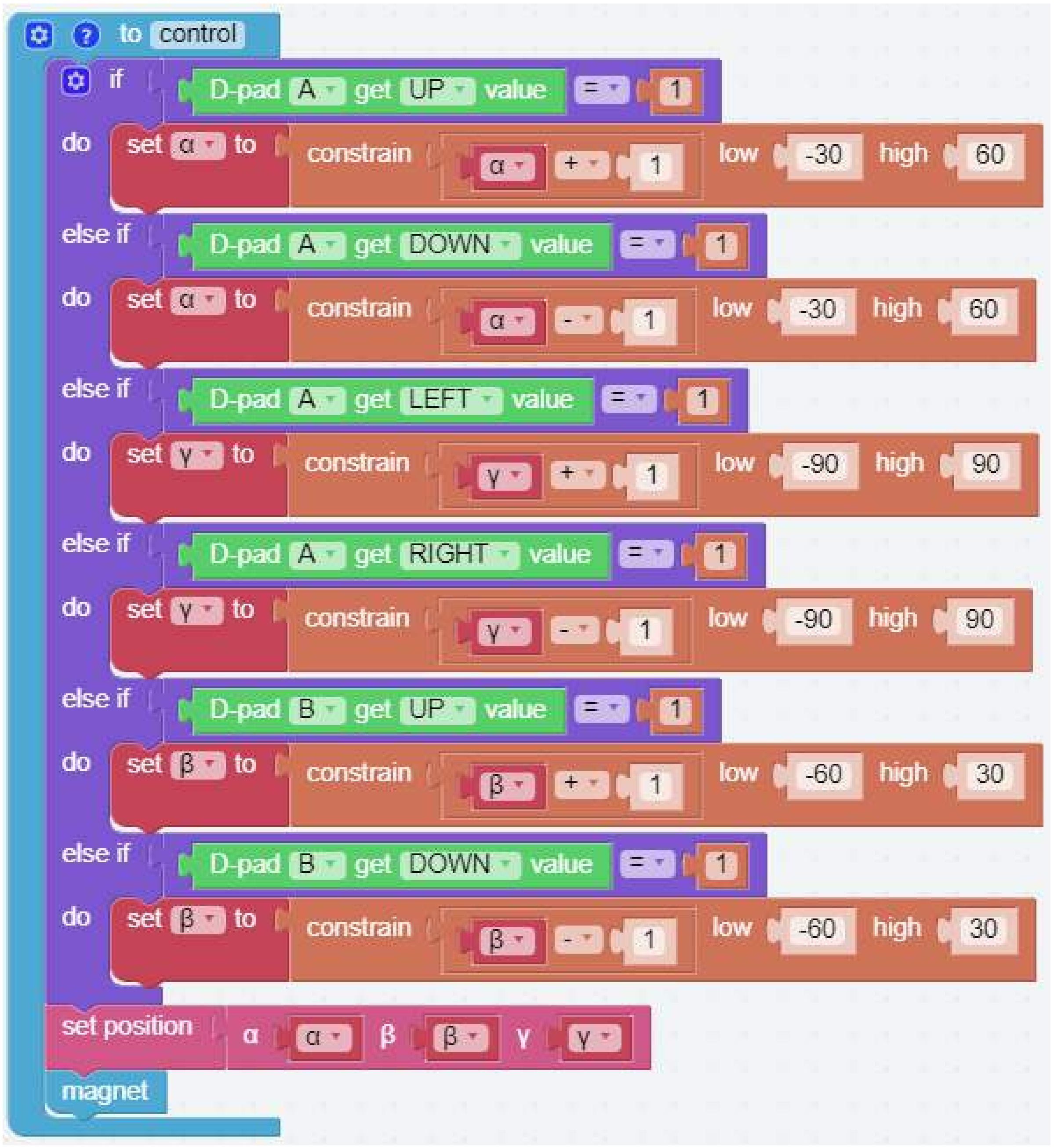
```
to magnet
  if D-pad B get LEFT value = 1
  do turn electromagnet on
  else if D-pad B get RIGHT value = 1
  do turn electromagnet off
```

Step 4

Create a function named [control] to implement the Arm of PiArm to be controlled by the D-pad A and D-pad B.

Note:

- , and represent the 3 servo rotation ranges on PiArm, refer to: *Angle Mode*.
- [constrain () low () high ()]: From Math category for setting the variation of a constant to a certain range.
- [if else]: Conditional judgment block, you can create multiple conditional judgments by clicking the set icon and dragging [else] or [else if] to the right below the [if].



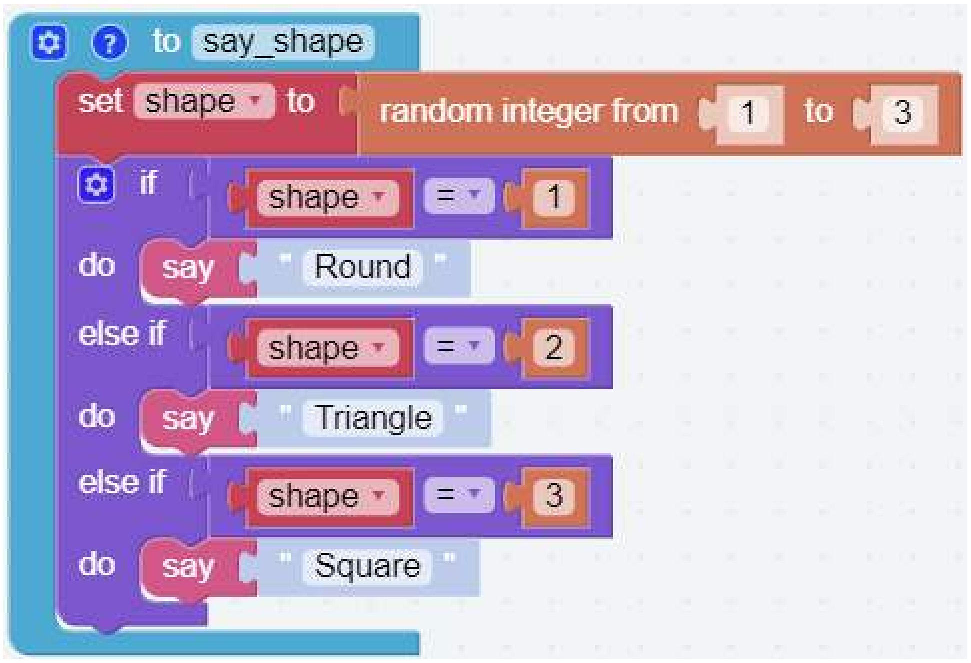
```
to control
  if D-pad A get UP value = 1
  do set α to constrain α + 1 low -30 high 60
  else if D-pad A get DOWN value = 1
  do set α to constrain α - 1 low -30 high 60
  else if D-pad A get LEFT value = 1
  do set γ to constrain γ + 1 low -90 high 90
  else if D-pad A get RIGHT value = 1
  do set γ to constrain γ - 1 low -90 high 90
  else if D-pad B get UP value = 1
  do set β to constrain β + 1 low -60 high 30
  else if D-pad B get DOWN value = 1
  do set β to constrain β - 1 low -60 high 30
  set position α α β β γ γ
  magnet
```

- If the UP button () of D-pad **A** is pressed, the Arm will extend forward.
- If the Down button () of D-pad **A** is pressed, the Arm will retract backward.
- If the LEFT button () of D-pad **A** is pressed, the Arm will turn left.
- If the RIGHT button () of D-pad **A** is pressed, the Arm will turn right.

- If the UP button () of D-pad **B** is pressed, the Arm will raise up.
- If the Down button () of D-pad **B** is pressed, the Arm will lower down.

Step 5

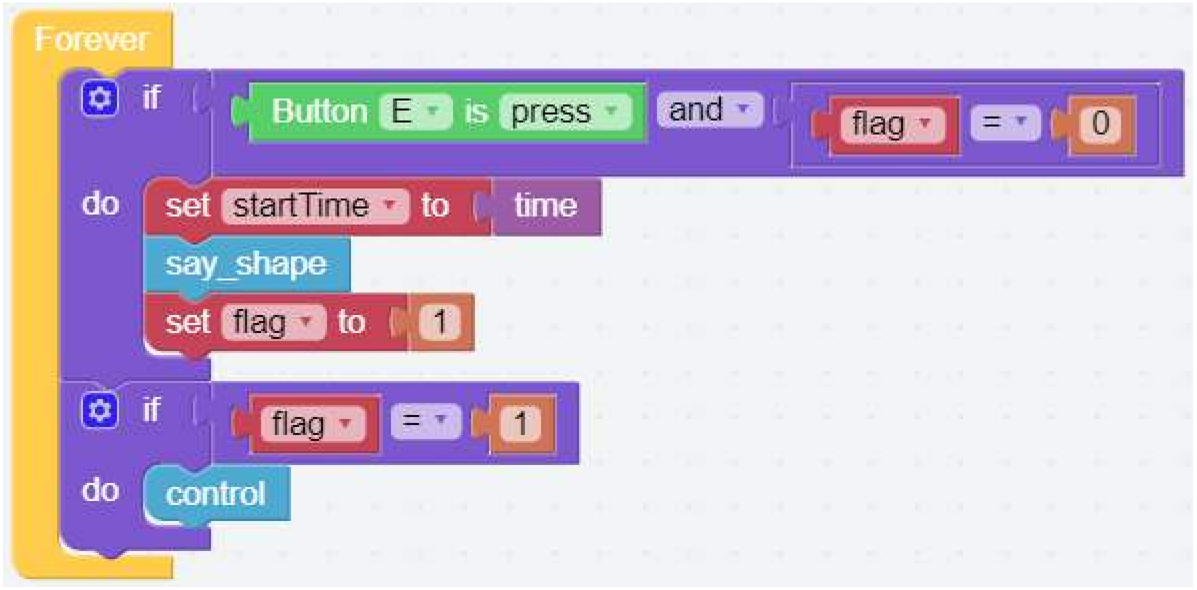
Create the function [say_shape] to have PiArm speak a random shape.



```
to say_shape
    set shape to random integer from 1 to 3
    if shape = 1
    do  say " Round "
    else if shape = 2
    do  say " Triangle "
    else if shape = 3
    do  say " Square "
```

Step 6

The main flow of the code: when **button E** is pressed, the timer starts and PiArm will say a random shape. `flag` is used to represent the start of the countdown and you can control PiArm.

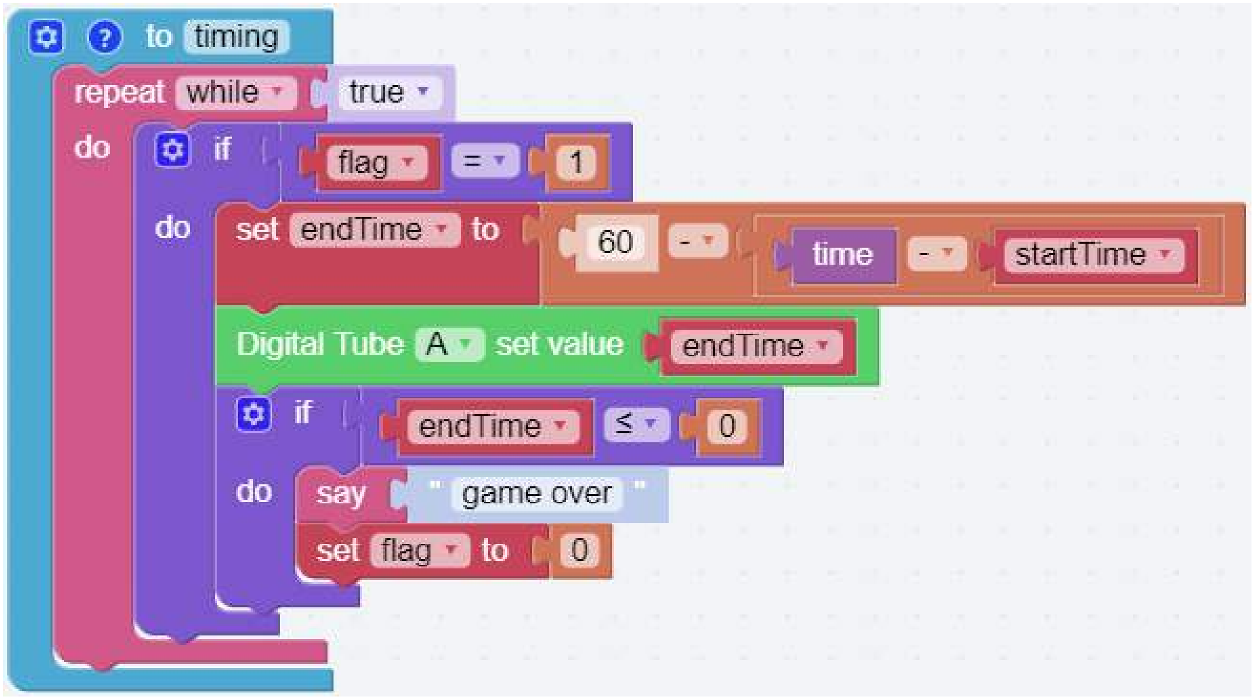


```
Forever
    if Button E is press and flag = 0
    do  set startTime to time
        say_shape
        set flag to 1
    if flag = 1
    do  control
```

Step 7

Create a function named [timing] to use for timing. The game time is set to 60 seconds, after the time is up, PiArm will say Game over and you will no longer be able to control it.

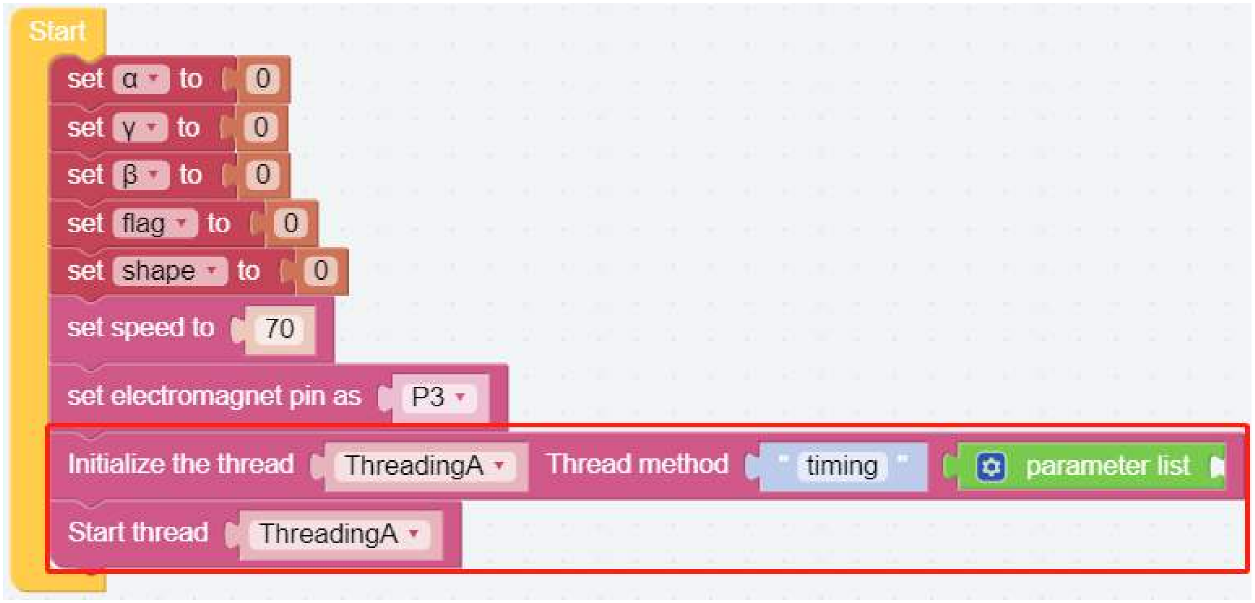
Here the [time] block is used for timing, in Forever, when button E is pressed, the timing starts and [time - startTime] represents how many seconds have passed since then.



```
to timing
  repeat while true
  do  if flag = 1
      do  set endTime to 60 - time - startTime
          Digital Tube A set value endTime
          if endTime ≤ 0
          do  say " game over "
              set flag to 0
```

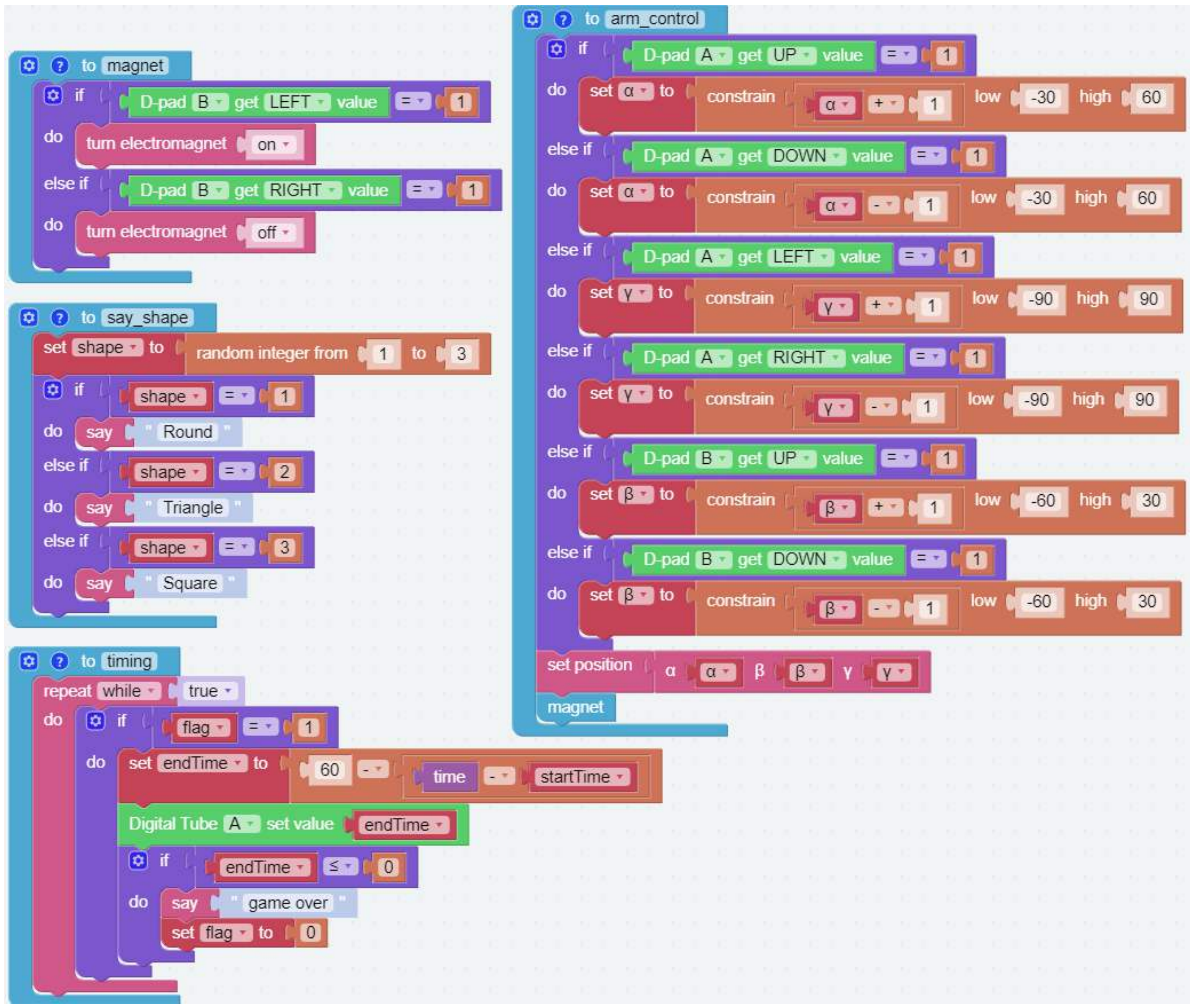
Step 8

Let the [timing] function run in a separate thread. This allows you to control PiArm while counting down.

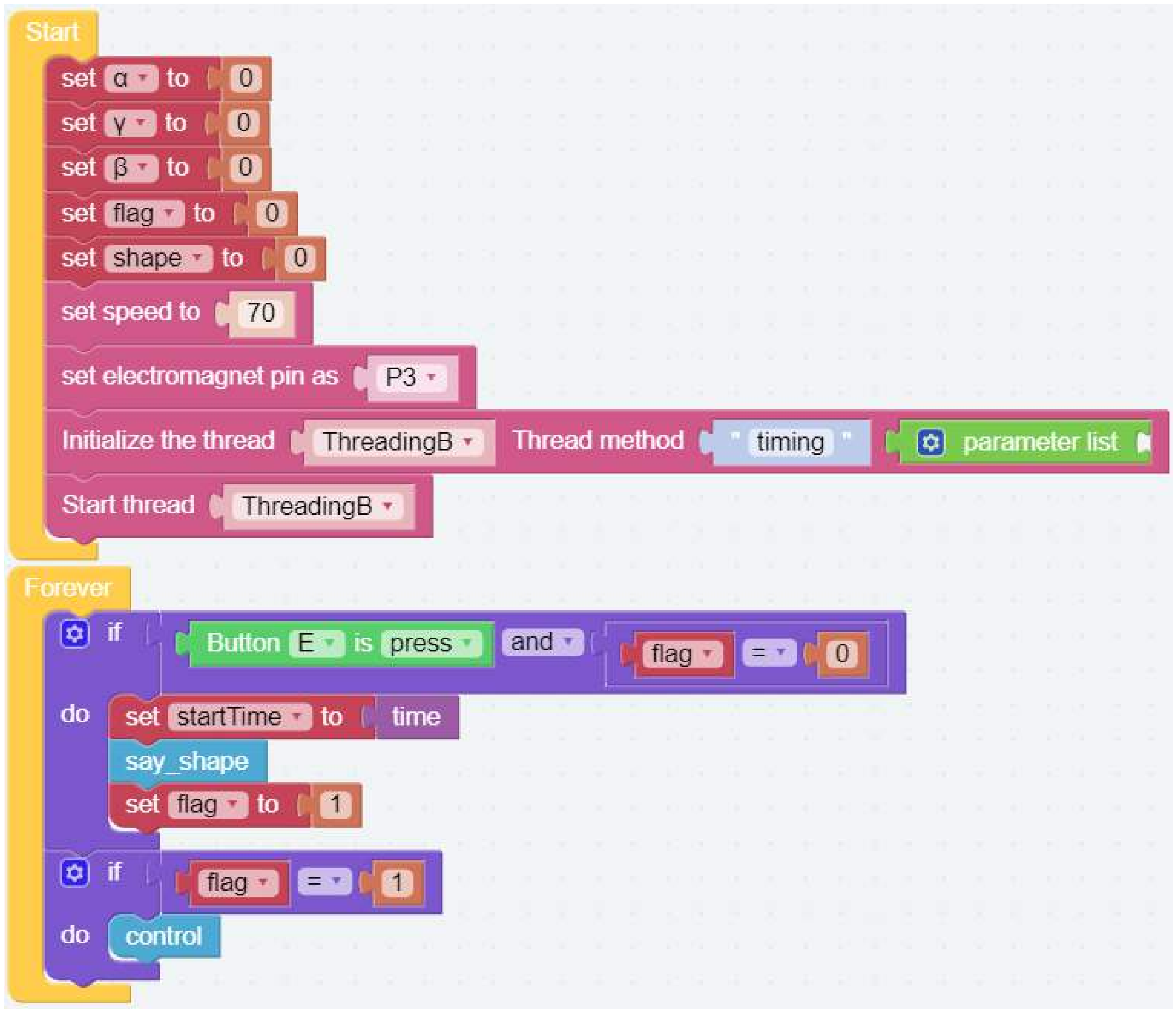


```
Start
  set α to 0
  set γ to 0
  set β to 0
  set flag to 0
  set shape to 0
  set speed to 70
  set electromagnet pin as P3
  Initialize the thread ThreadingA Thread method " timing " parameter list
  Start thread ThreadingA
```

The complete code is as follows:



```
to magnet
    if D-pad B get LEFT value = 1
    do turn electromagnet on
    else if D-pad B get RIGHT value = 1
    do turn electromagnet off

to say_shape
    set shape to random integer from 1 to 3
    if shape = 1
    do say " Round "
    else if shape = 2
    do say " Triangle "
    else if shape = 3
    do say " Square "

to timing
    repeat while true
    do  if flag = 1
        do  set endTime to 60 - time - startTime
            Digital Tube A set value endTime
            if endTime ≤ 0
            do  say " game over "
                set flag to 0

to arm_control
    if D-pad A get UP value = 1
    do set α to constrain α + 1 low -30 high 60
    else if D-pad A get DOWN value = 1
    do set α to constrain α - 1 low -30 high 60
    else if D-pad A get LEFT value = 1
    do set γ to constrain γ + 1 low -90 high 90
    else if D-pad A get RIGHT value = 1
    do set γ to constrain γ - 1 low -90 high 90
    else if D-pad B get UP value = 1
    do set β to constrain β + 1 low -60 high 30
    else if D-pad B get DOWN value = 1
    do set β to constrain β - 1 low -60 high 30
    set position α α β β γ γ
    magnet
```



```
Start
  set α to 0
  set γ to 0
  set β to 0
  set flag to 0
  set shape to 0
  set speed to 70
  set electromagnet pin as P3
  Initialize the thread ThreadingB Thread method " timing " parameter list
  Start thread ThreadingB
Forever
  if Button E is press and flag = 0
  do set startTime to time
     say_shape
     set flag to 1
  if flag = 1
  do control
```

CHAPTER FOUR

PLAY WITH PYTHON

If you want to program in python, then you will need to learn some basic Python programming skills and basic knowledge of Raspberry Pi, please configure the Raspberry Pi first according to *Quick Guide on Python*.

4.1 Quick Guide on Python

This section is to teach you how to install Raspberry Pi OS, configure wifi to Raspberry Pi, remote access to Raspberry Pi to run the corresponding code.

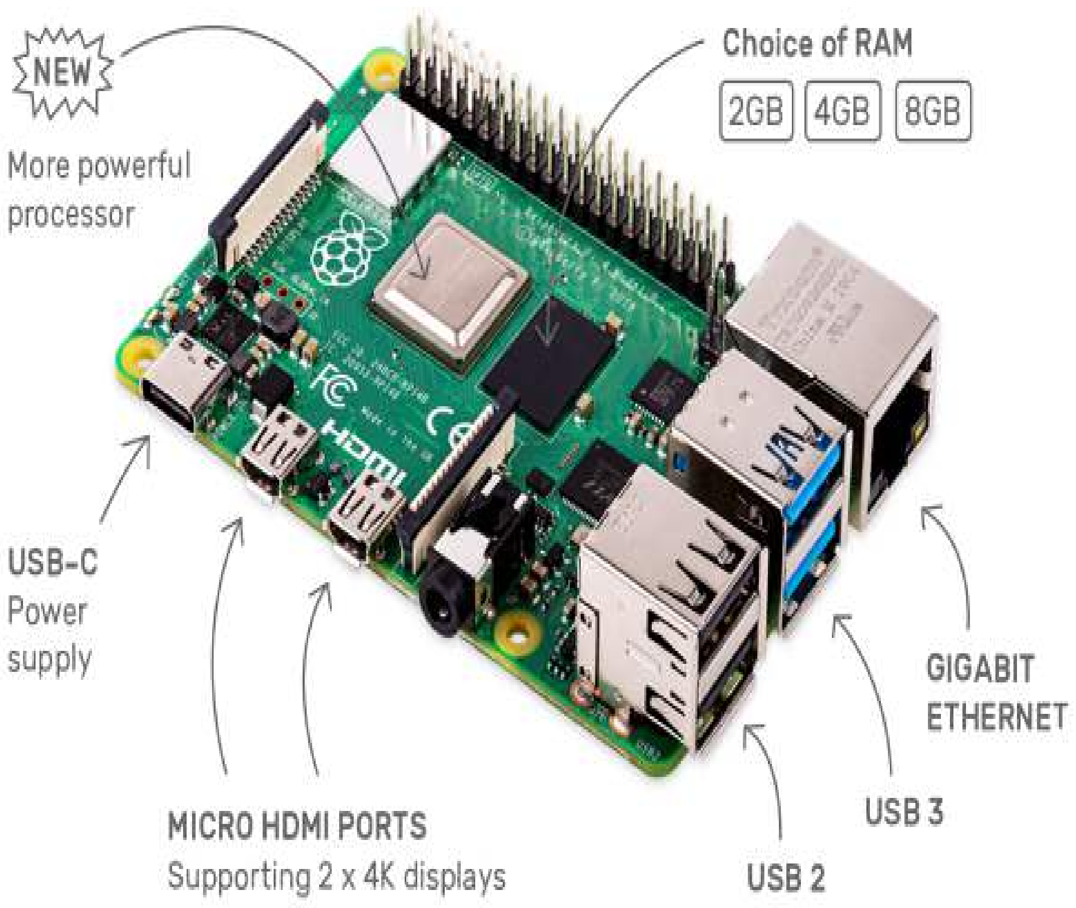
If you are familiar with Raspberry Pi and can open the command line successfully, then you can skip the first 3 parts and then complete the last part.

4.1.1 What Do We Need?

Required Components

Raspberry Pi

The Raspberry Pi is a low cost, credit-card sized computer that plugs into a computer monitor or TV, and uses a standard keyboard and mouse. It is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python.



Power Adapter

To connect to a power socket, the Raspberry Pi has a micro USB port (the same found on many mobile phones). You will need a power supply which provides at least 2.5 amps.

Micro SD Card

Your Raspberry Pi needs an Micro SD card to store all its files and the Raspberry Pi OS. You will need a micro SD card with a capacity of at least 8 GB

Optional Components

Screen

To view the desktop environment of Raspberry Pi, you need to use the screen that can be a TV screen or a computer monitor. If the screen has built-in speakers, the Pi plays sounds via them.

Mouse & Keyboard

When you use a screen , a USB keyboard and a USB mouse are also needed.

HDMI

The Raspberry Pi has a HDMI output port that is compatible with the HDMI ports of most modern TV and computer monitors. If your screen has only DVI or VGA ports, you will need to use the appropriate conversion line.

Case

You can put the Raspberry Pi in a case; by this means, you can protect your device.

Sound or Earphone

The Raspberry Pi is equipped with an audio port about 3.5 mm that can be used when your screen has no built-in speakers or when there is no screen operation.

4.1.2 Installing the OS

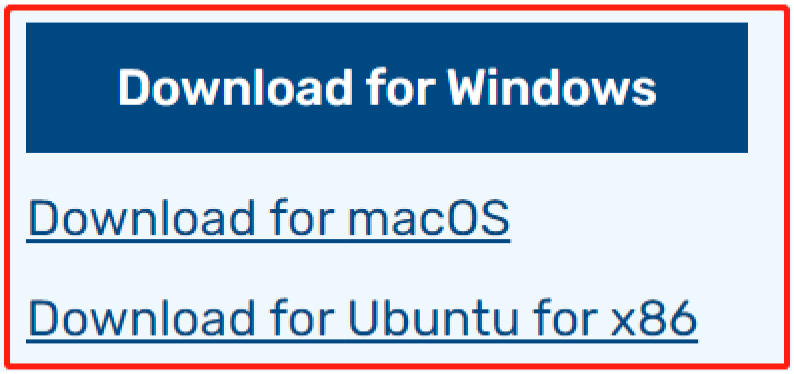
Required Components

Any Raspberry Pi	1 * Personal Computer
1 * Micro SD card	

Step 1

Raspberry Pi have developed a graphical SD card writing tool that works on Mac OS, Ubuntu 18.04 and Windows, and is the easiest option for most users as it will download the image and install it automatically to the SD card.

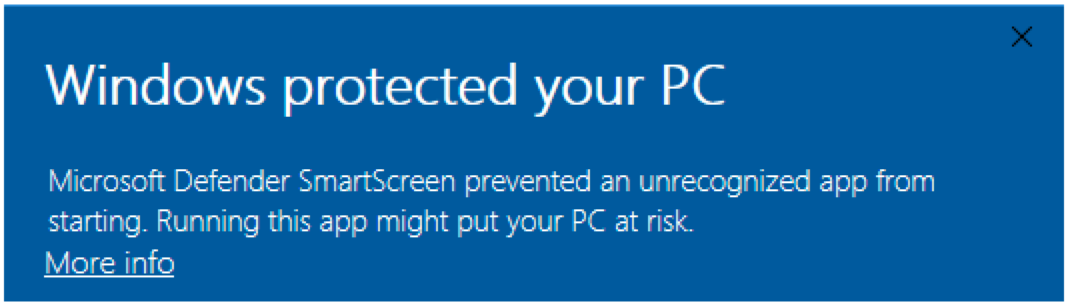
Visit the download page: https://www.raspberrypi.org/software/. Click on the link for the Raspberry Pi Imager that matches your operating system, when the download finishes, click it to launch the installer.



Step 2

When you launch the installer, your operating system may try to block you from running it. For example, on Windows I receive the following message:

If this pops up, click on **More info** and then **Run anyway**, then follow the instructions to install the Raspberry Pi Imager.


Windows protected your PC

Microsoft Defender SmartScreen prevented an unrecognized app from starting. Running this app might put your PC at risk.

More info

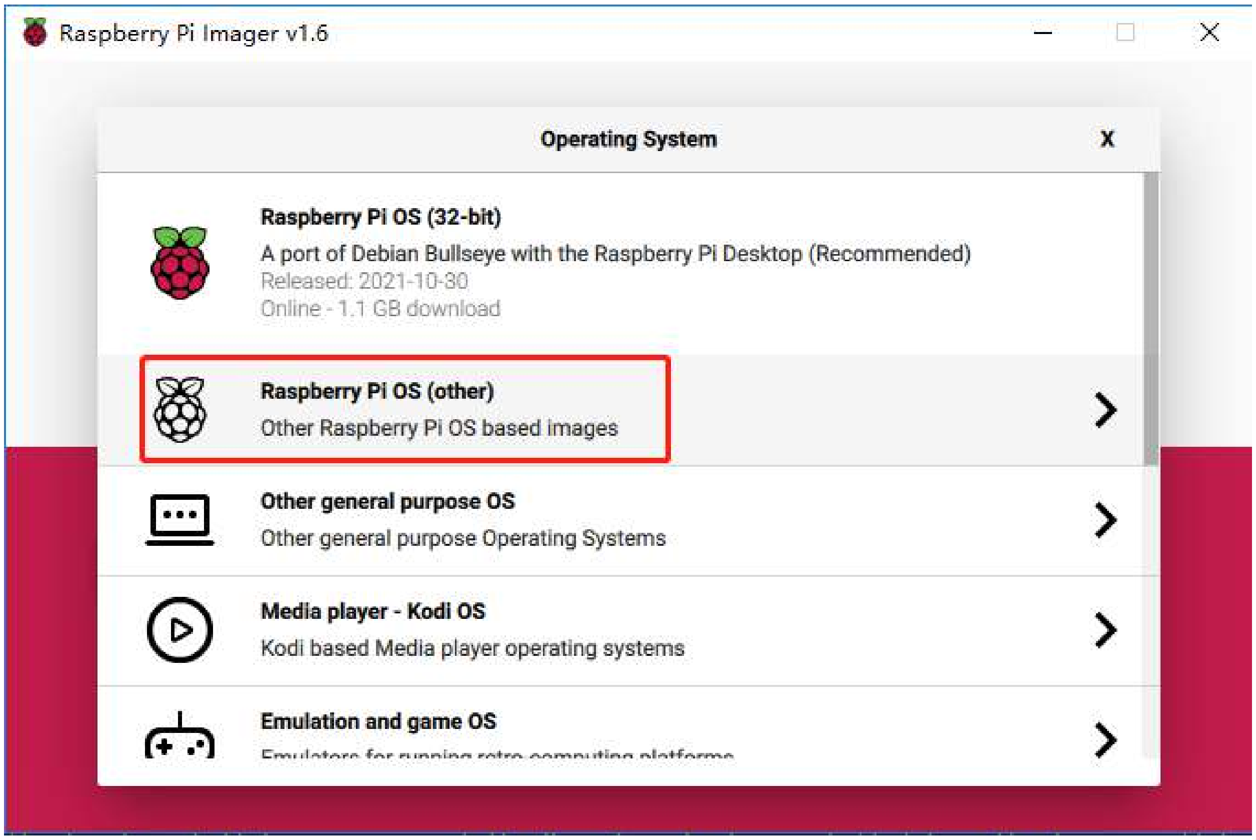
Step 3

Insert your SD card into the computer or laptop SD card slot.

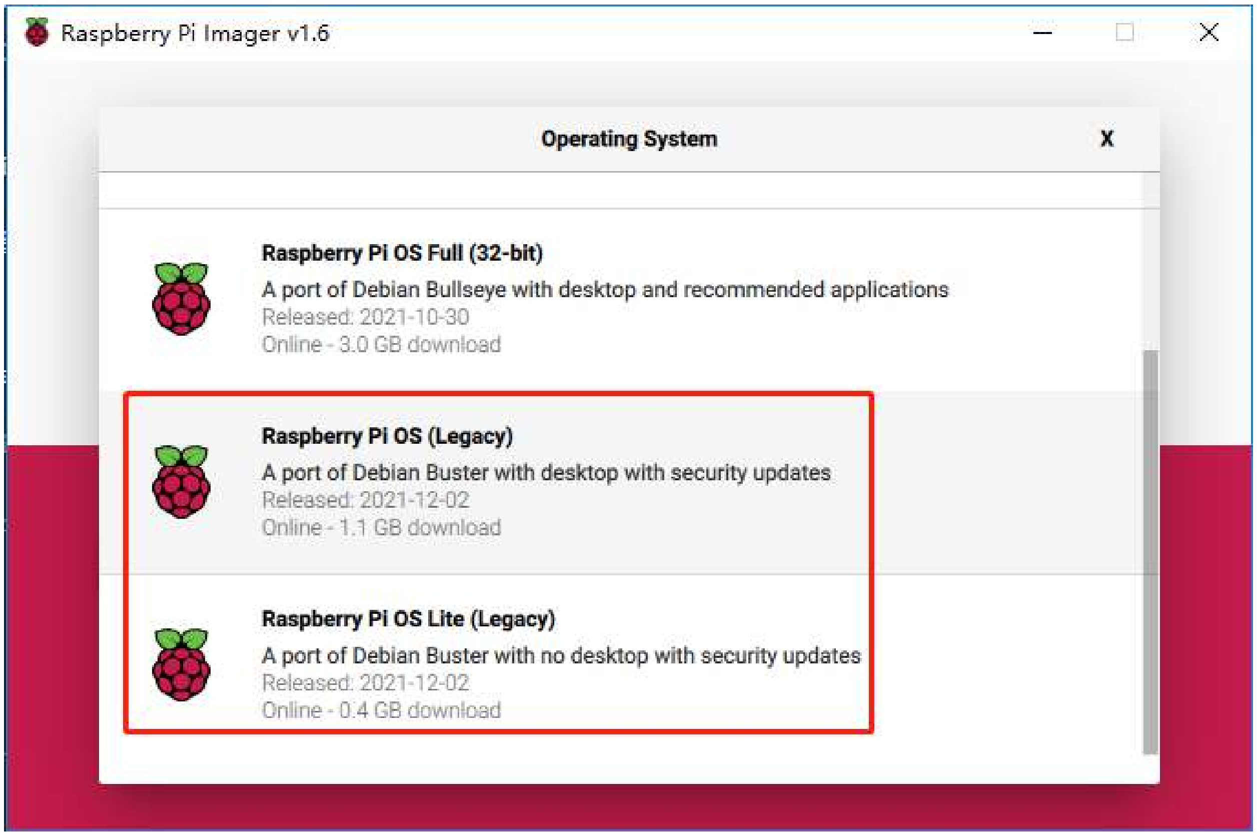
Step 4

Warning: Upgrading the Raspberry Pi OS to **Debian Bullseye** will cause some features to not work, so it is recommended to continue using the **Debian Buster** version.

In the Raspberry Pi Imager, click **CHOOSE OS** -> **Raspberry Pi OS(other)**.



Scroll down to the end of the newly opened page and you will see **Raspberry Pi OS(Legacy)** and **Raspberry Pi OS Lite(Legacy)**, these are security updates for Debian Buster, the difference between them is with or without the desktop. It is recommended to install **Raspberry Pi OS(Legacy)**, the system with the desktop.



Raspberry Pi Imager v1.6

Operating System

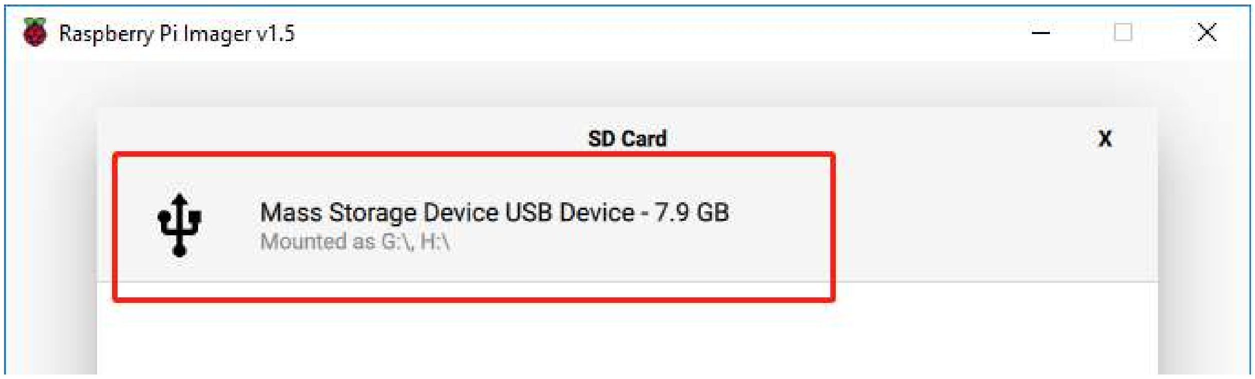
Raspberry Pi OS Full (32-bit)
A port of Debian Bullseye with desktop and recommended applications
Released: 2021-10-30
Online - 3.0 GB download

Raspberry Pi OS (Legacy)
A port of Debian Buster with desktop with security updates
Released: 2021-12-02
Online - 1.1 GB download

Raspberry Pi OS Lite (Legacy)
A port of Debian Buster with no desktop with security updates
Released: 2021-12-02
Online - 0.4 GB download

Step 5

Select the SD card you are using.



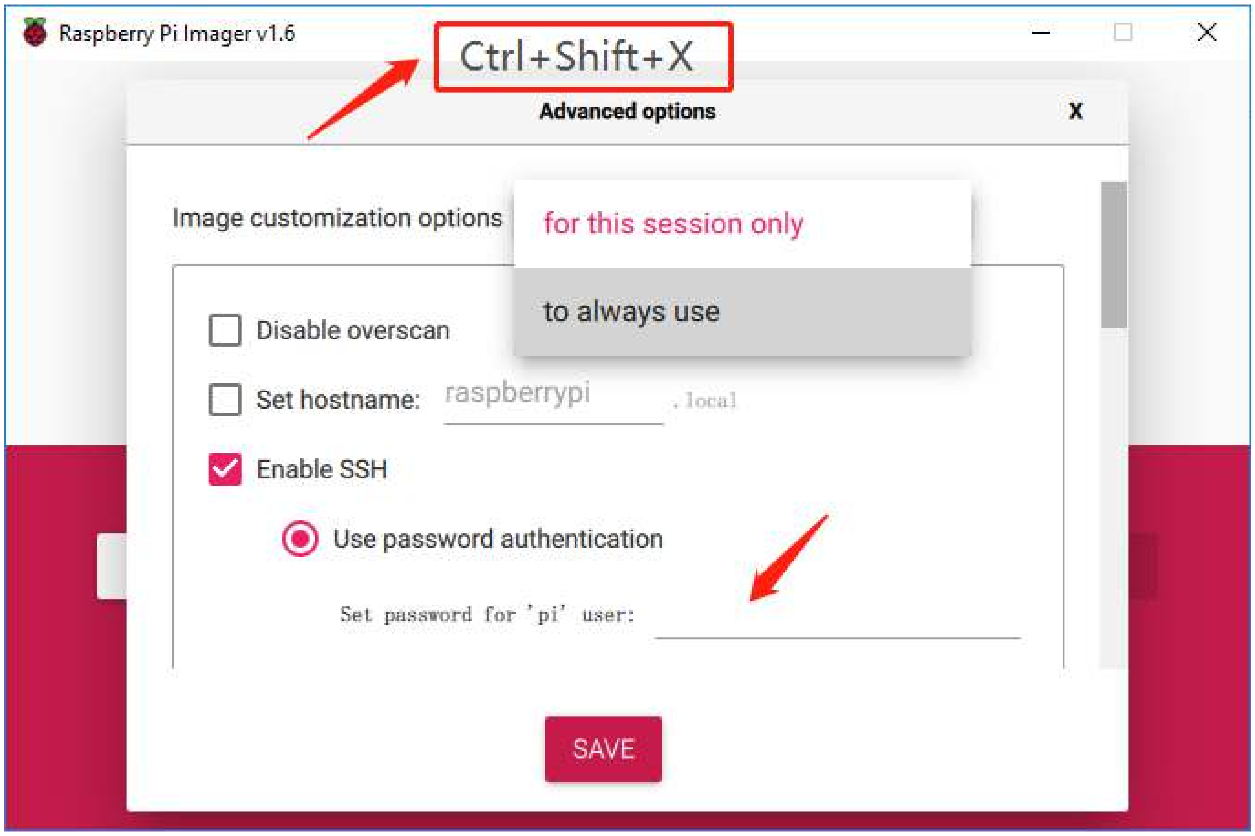
Raspberry Pi Imager v1.5

SD Card

Mass Storage Device USB Device - 7.9 GB
Mounted as G:\, H:\

Step 6

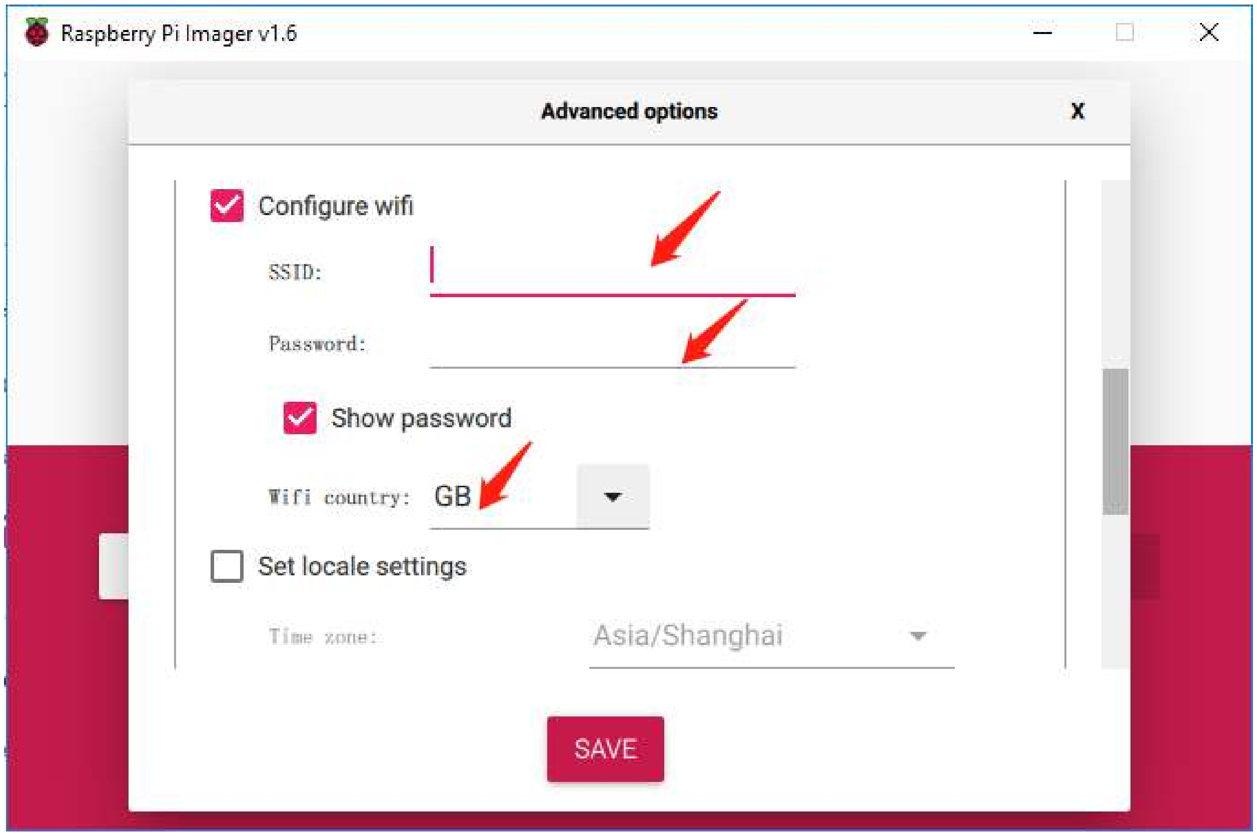
Press **Ctrl+Shift+X** or click the **setting ** button to open the **Advanced options** page to enable SSH and configure wifi, these 2 items must be set, the others depend on your choice . You can choose to always use this image customization options.



Disable overscan
Set hostname: raspberrypi .local
Enable SSH
Use password authentication
Set password for 'pi' user:

Then scroll down to complete the wifi configuration and click **SAVE**.

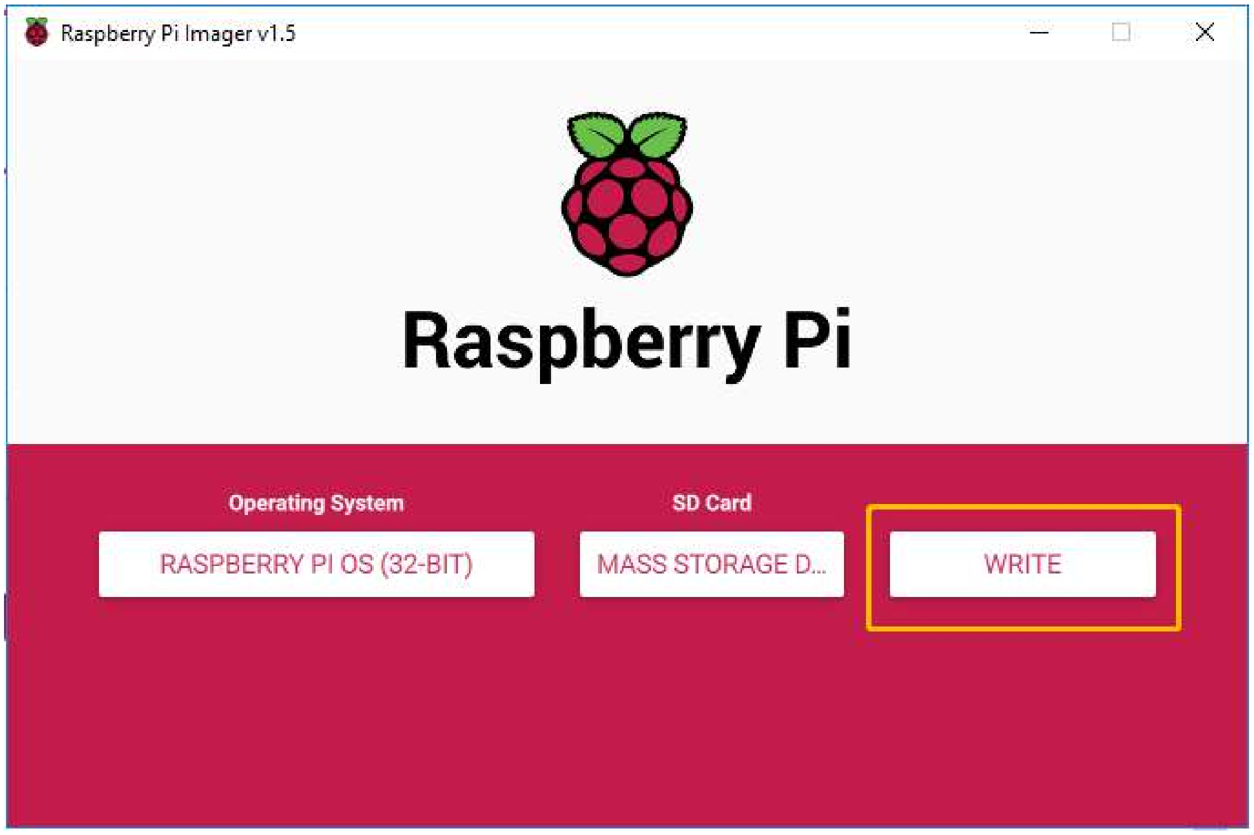
Note: **wifi country** should be set the two-letter ISO/IEC alpha2 code for the country in which you are using your Raspberry Pi, please refer to the following link: https://en.wikipedia.org/wiki/ISO_3166-1_alpha-2#Officially_assigned_code_elements



Configure wifi ✓
SSID:
Password:
Show password ✓
Wifi country: GB
Set locale settings
Time zone: Asia/Shanghai

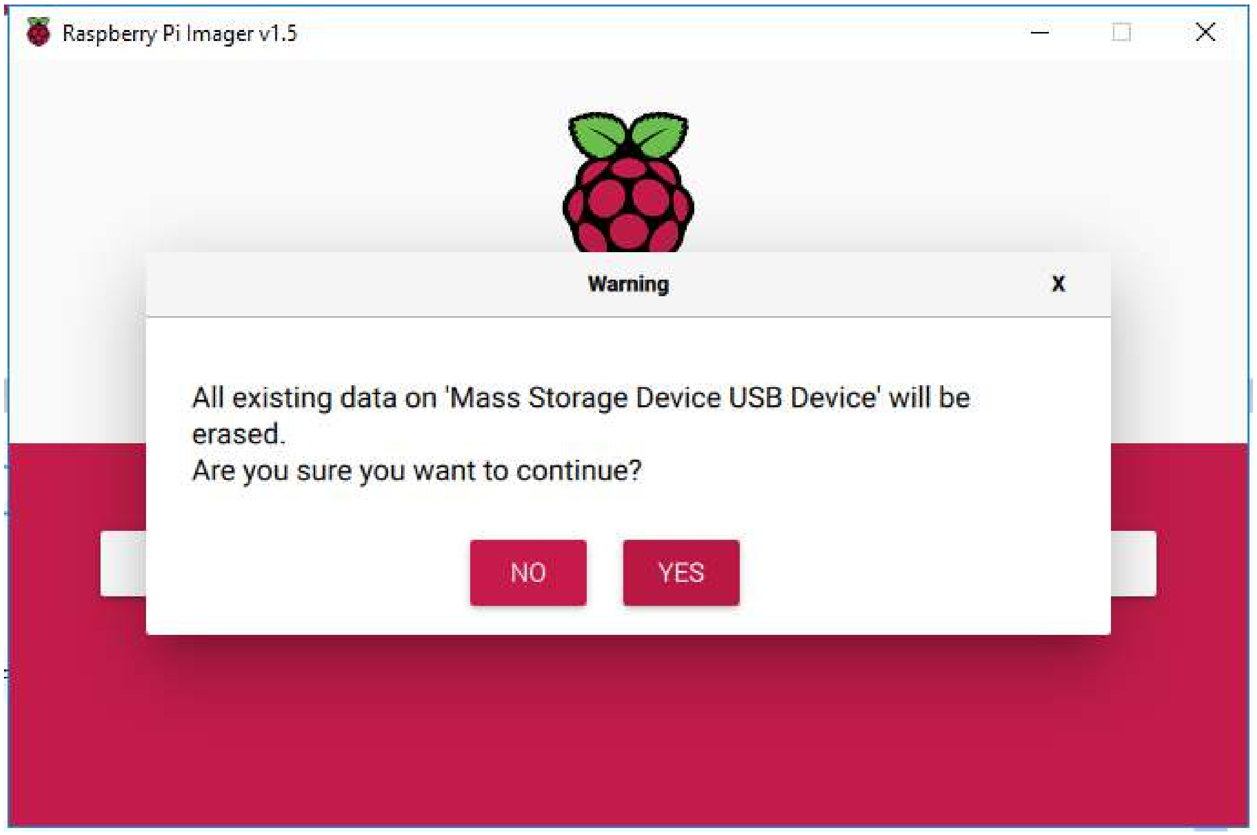
Step 7

Click the **WRITE** button.



Step 8

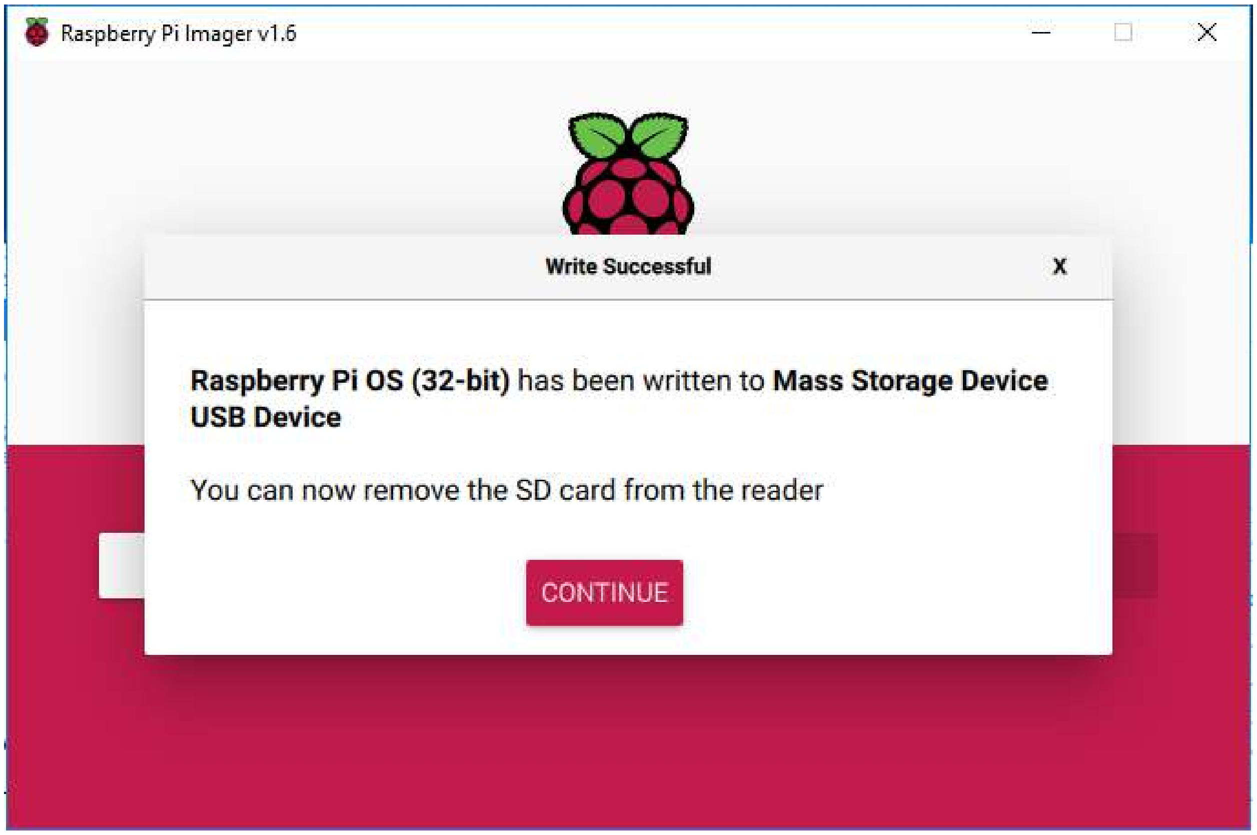
If your SD card currently has any files on it, you may wish to back up these files first to prevent you from permanently losing them. If there is no file to be backed up, click **Yes**.



All existing data on 'Mass Storage Device USB Device' will be erased.
Are you sure you want to continue?

Step 9

After waiting for a period of time, the following window will appear to represent the completion of writing.



Raspberry Pi OS (32-bit) has been written to **Mass Storage Device USB Device**

You can now remove the SD card from the reader

4.1.3 Set up Your Raspberry Pi

If You Have a Screen

If you have a screen, it will be easy for you to operate on the Raspberry Pi.

Required Components

Any Raspberry Pi	1 * Power Adapter
1 * Micro SD card	1 * Screen Power Adapter
1 * HDMI cable	1 * Screen
1 * Mouse	1 * Keyboard

1. Insert the SD card you've set up with Raspberry Pi OS into the micro SD card slot on the underside of your Raspberry Pi.
2. Plug in the Mouse and Keyboard.
3. Connect the screen to Raspberry Pi's HDMI port and make sure your screen is plugged into a wall socket and switched on.

> **Note:** If you use a Raspberry Pi 4, you need to connect the screen to the HDMI0 (nearest the power in port).

4. Use the power adapter to power the Raspberry Pi. After a few seconds, the Raspberry Pi OS desktop will be displayed.

If You Have No Screen

If you don't have a display, you can log in to the Raspberry Pi remotely, but before that, you need to get the IP of the Raspberry Pi.

Get the IP Address

After the Raspberry Pi is connected to WIFI, we need to get the IP address of it. There are many ways to know the IP address, and two of them are listed as follows.

1. Checking via the router

If you have permission to log in the router(such as a home network), you can check the addresses assigned to Raspberry Pi on the admin interface of router.

The default hostname of the Raspberry Pi OS is **raspberrypi**, and you need to find it. (If you are using ArchLinuxARM system, please find alarmpi.)

2. Network Segment Scanning

You can also use network scanning to look up the IP address of Raspberry Pi. You can apply the software, **Advanced IP scanner** and so on.

Scan the IP range set, and the name of all connected devices will be displayed. Similarly, the default hostname of the Raspberry Pi OS is **raspberrypi**, if you haven't modified it.

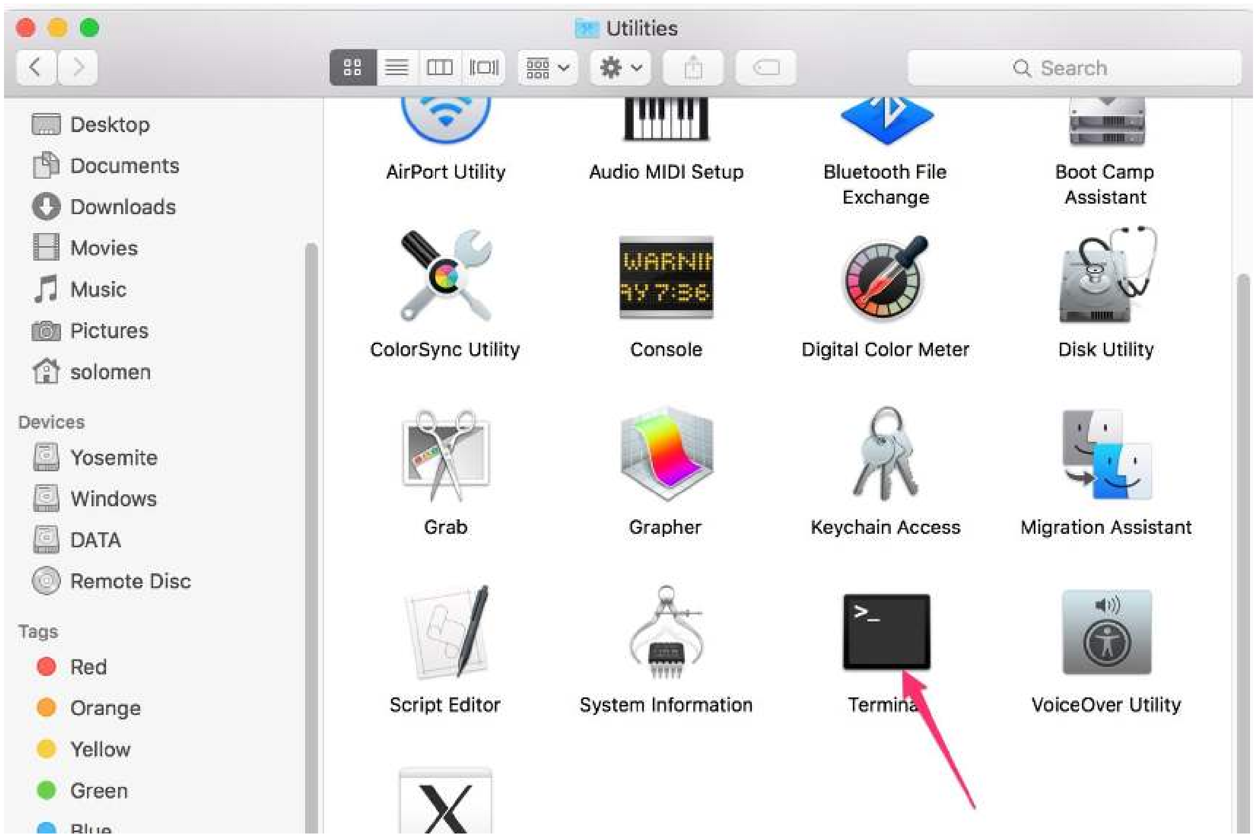
Use the SSH Remote Control

We can open the Bash Shell of Raspberry Pi by applying SSH. Bash is the standard default shell of Linux. The Shell itself is a program written in C that is the bridge linking the customers and Unix/Linux. Moreover, it can help to complete most of the work needed.

For Linux or/Mac OS X Users

Step 1

Go to **Applications->Utilities**, find the **Terminal**, and open it.



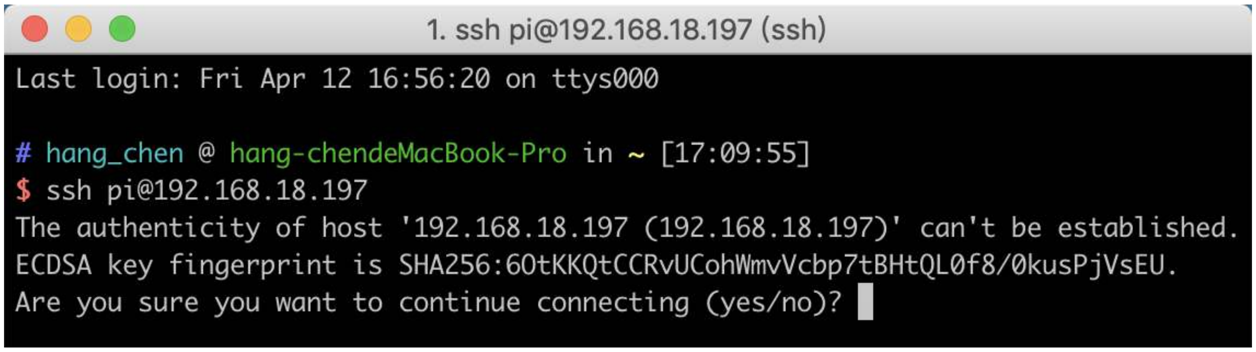
Step 2

Type in **ssh pi@ip_address** . “pi” is your username and “ip_address” is your IP address. For example:

```
ssh pi@192.168.18.197
```

Step 3

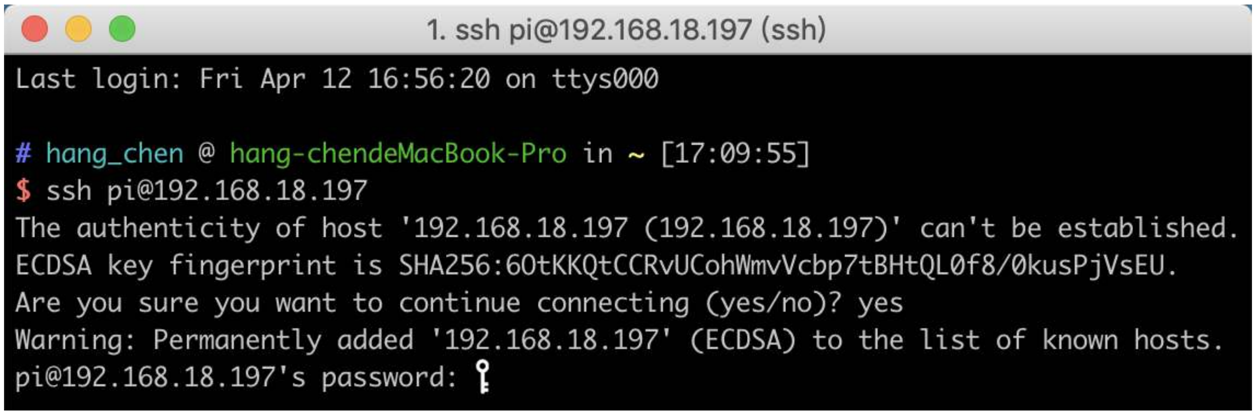
Input “yes”.



```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)?
```

Step 4

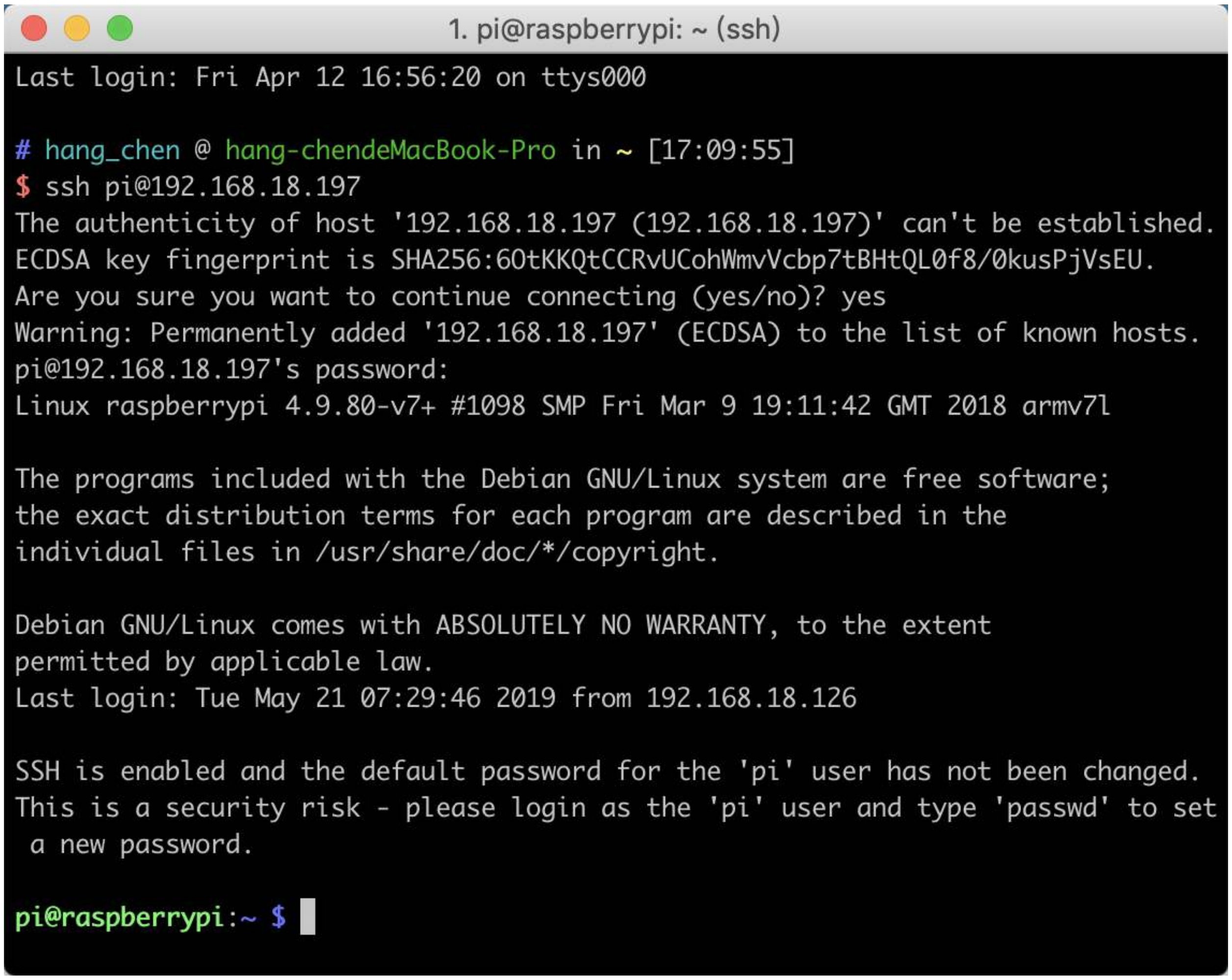
Input the passcode and the default password is **raspberry**.



```
1. ssh pi@192.168.18.197 (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
```

Step 5

We now get the Raspberry Pi connected and are ready to go to the next step.



```
1. pi@raspberrypi: ~ (ssh)
Last login: Fri Apr 12 16:56:20 on ttys000

# hang_chen @ hang-chendeMacBook-Pro in ~ [17:09:55]
$ ssh pi@192.168.18.197
The authenticity of host '192.168.18.197 (192.168.18.197)' can't be established.
ECDSA key fingerprint is SHA256:6OtKKQtCCRvUCohWmvVcbp7tBHtQL0f8/0kusPjVsEU.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '192.168.18.197' (ECDSA) to the list of known hosts.
pi@192.168.18.197's password:
Linux raspberrypi 4.9.80-v7+ #1098 SMP Fri Mar 9 19:11:42 GMT 2018 armv7l

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue May 21 07:29:46 2019 from 192.168.18.126

SSH is enabled and the default password for the 'pi' user has not been changed.
This is a security risk - please login as the 'pi' user and type 'passwd' to set
 a new password.

pi@raspberrypi:~ $
```

Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

For Windows Users

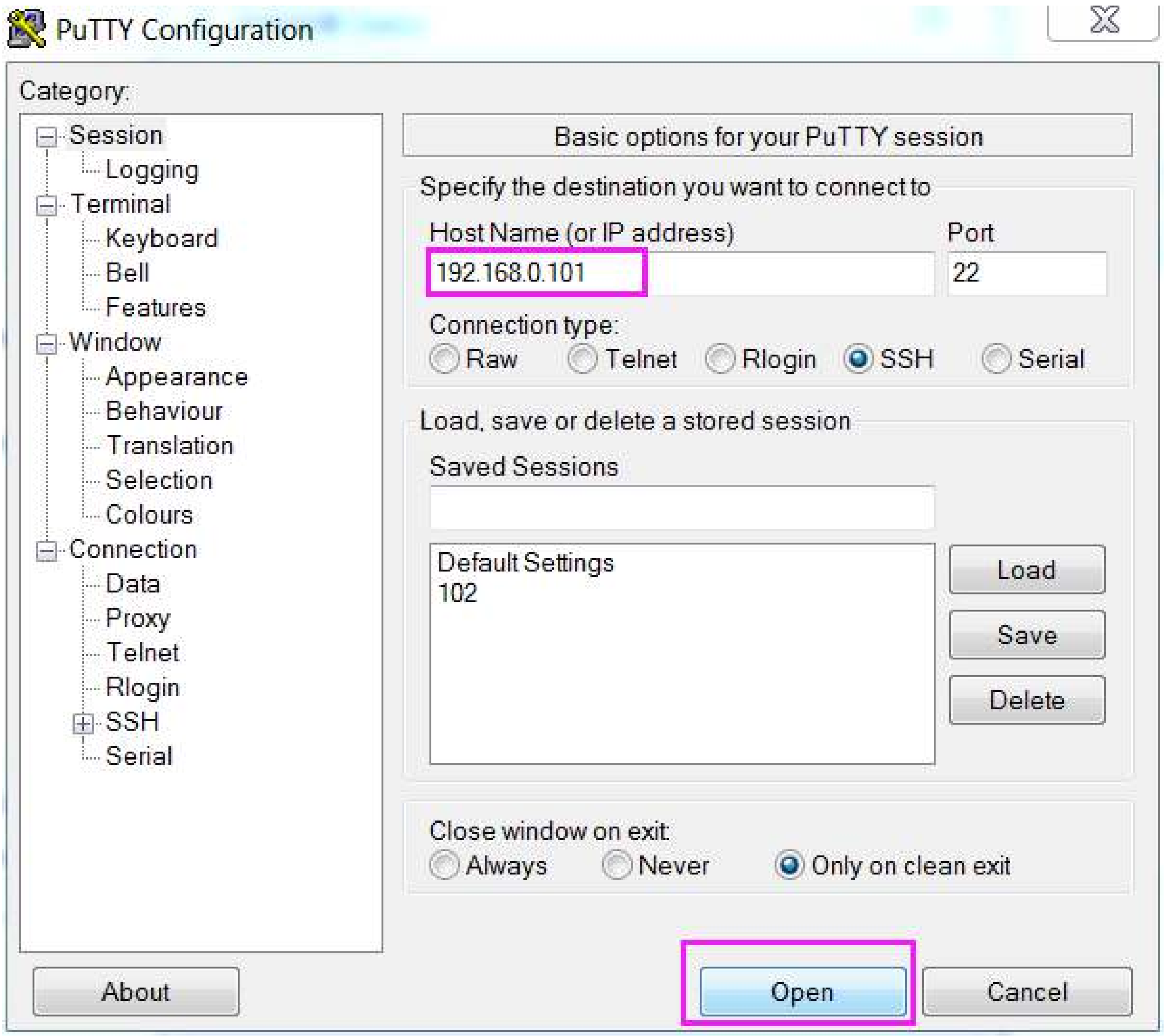
If you're a Windows user, you can use SSH with the application of some software. Here, we recommend **PuTTY**.

Step 1

Download PuTTY.

Step 2

Open PuTTY and click **Session** on the left tree-alike structure. Enter the IP address of the RPi in the text box under **Host Name (or IP address)** and **22** under **Port** (by default it is 22).



Specify the destination you want to connect to

Host Name (or IP address): 192.168.0.101

Port: 22

Connection type: Raw Telnet Rlogin [✓ SSH] Serial

Load, save or delete a stored session

Saved Sessions

Default Settings
102

Close window on exit: Always Never [✓ Only on clean exit]

Step 3

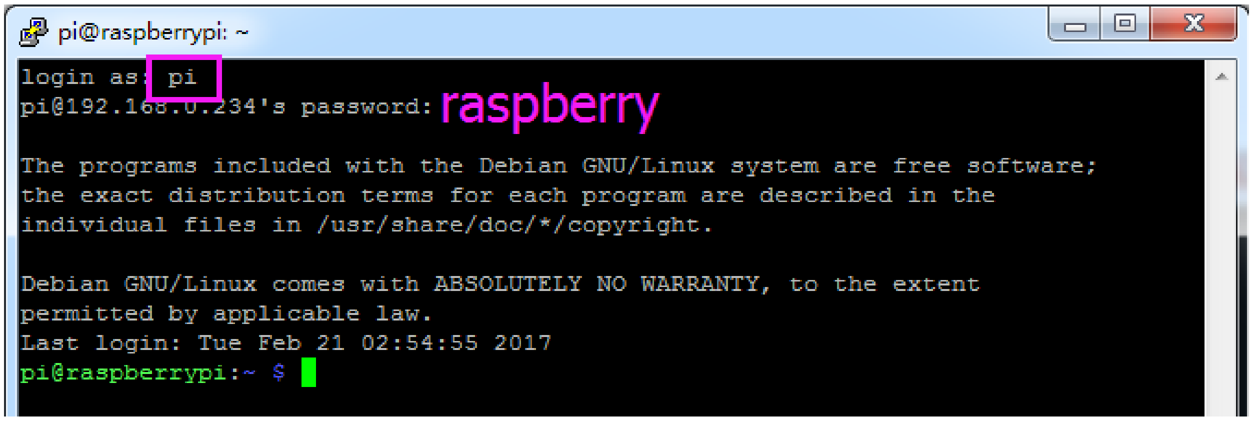
Click **Open**. Note that when you first log in to the Raspberry Pi with the IP address, there prompts a security reminder. Just click **Yes**.

Step 4

When the PuTTY window prompts "**login as:**", type in "**pi**" (the user name of the RPi), and **password:** "raspberry" (the default one, if you haven't changed it).

Note: When you input the password, the characters do not display on window accordingly, which is normal. What you need is to input the correct password.

If inactive appears next to PuTTY, it means that the connection has been broken and needs to be reconnected.



```
login as: pi
pi@192.168.0.234's password: raspberry

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Tue Feb 21 02:54:55 2017
pi@raspberrypi:~ $
```

Step 5

Here, we get the Raspberry Pi connected and it is time to conduct the next steps.

Note: If you are not satisfied with using the command window to control the Raspberry Pi, you can also use the remote desktop function, which can help us manage the files in the Raspberry Pi easily.

For details on how to do this, please refer to *Remote Desktop*.

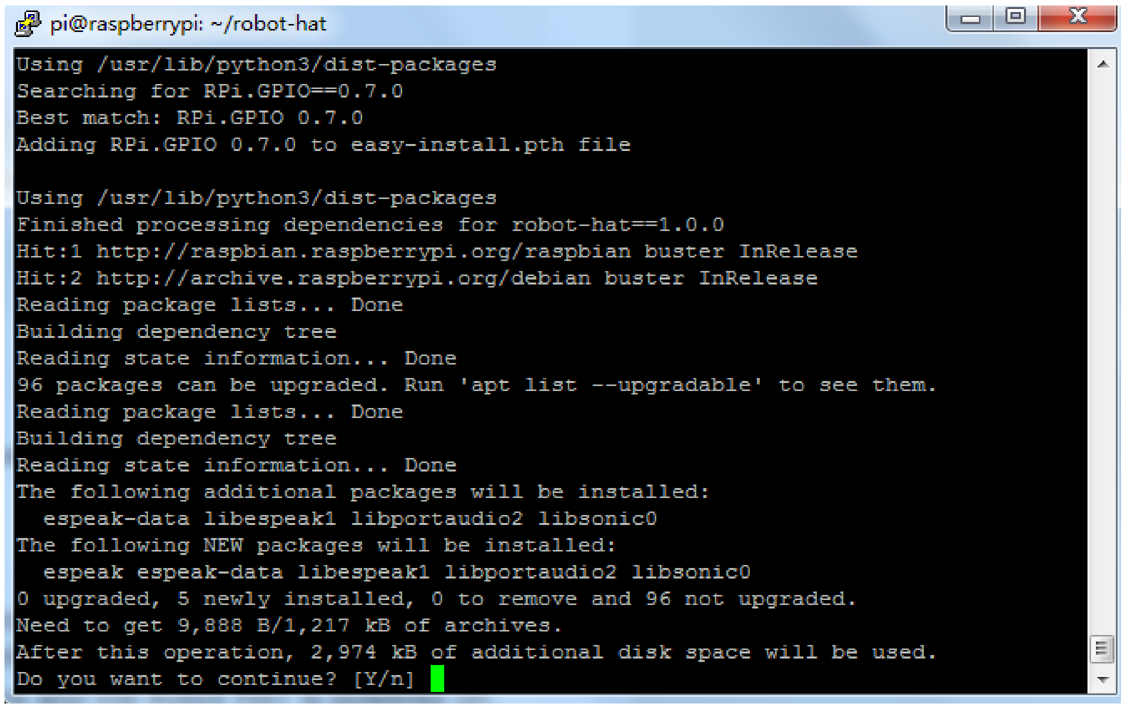
4.1.4 Download and Run the Code

First download and run the `robot-hat` module.

```
cd /home/pi/
git clone https://github.com/sunfounder/robot-hat.git
cd robot-hat
sudo python3 setup.py install
```

Note: Running `setup.py` will download some necessary components. Due to network problems, you may not be able to download successfully. You may need to download it again.

In this case, type Y and press Enter.



pi@raspberrypi: ~/robot-hat

```
Using /usr/lib/python3/dist-packages
Searching for RPi.GPIO==0.7.0
Best match: RPi.GPIO 0.7.0
Adding RPi.GPIO 0.7.0 to easy-install.pth file

Using /usr/lib/python3/dist-packages
Finished processing dependencies for robot-hat==1.0.0
Hit:1 http://raspbian.raspberrypi.org/raspbian buster InRelease
Hit:2 http://archive.raspberrypi.org/debian buster InRelease
Reading package lists... Done
Building dependency tree
Reading state information... Done
96 packages can be upgraded. Run 'apt list --upgradable' to see them.
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  espeak-data libespeak1 libportaudio2 libsonic0
The following NEW packages will be installed:
  espeak espeak-data libespeak1 libportaudio2 libsonic0
0 upgraded, 5 newly installed, 0 to remove and 96 not upgraded.
Need to get 9,888 B/1,217 kB of archives.
After this operation, 2,974 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

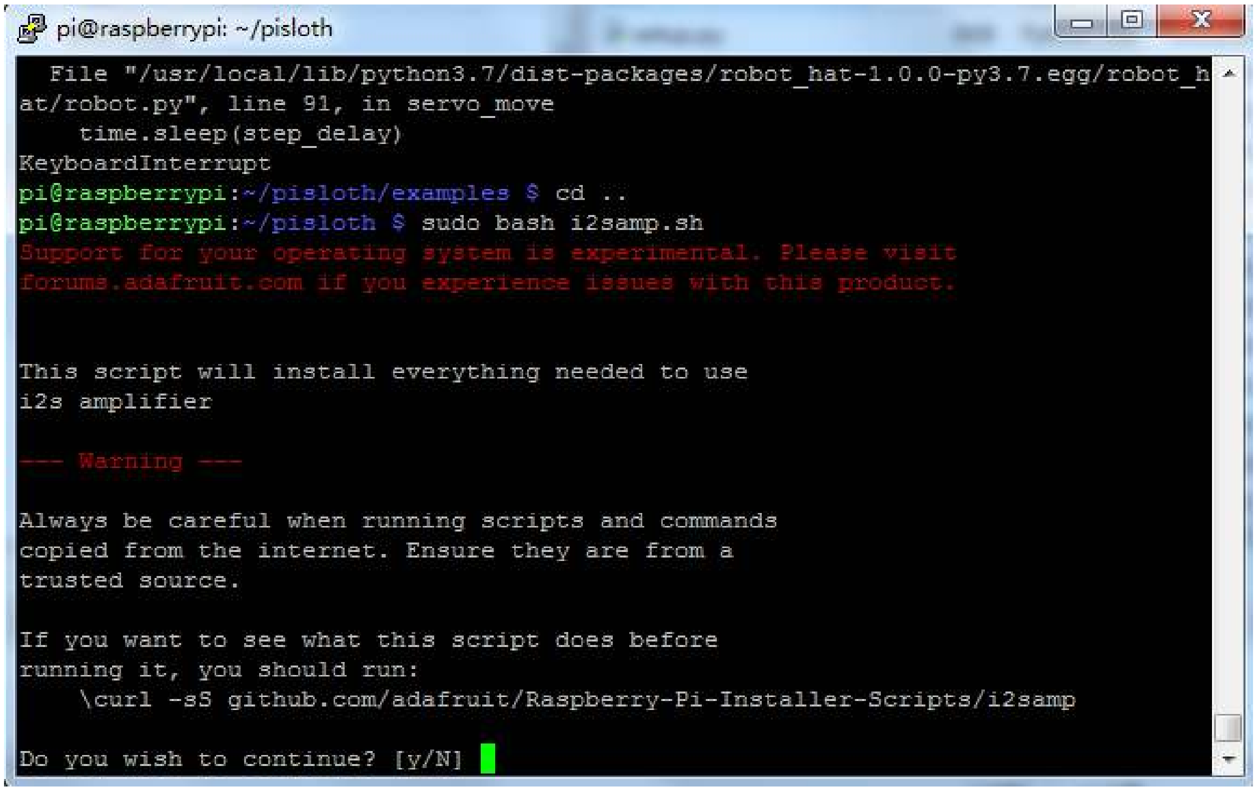
Then download the code and install the `piarm` library.

```
cd /home/pi/
git clone -b 2.0.0 https://github.com/sunfounder/piarm.git
cd piarm
sudo python3 setup.py install
```

This step will take a little time, so please be patient.

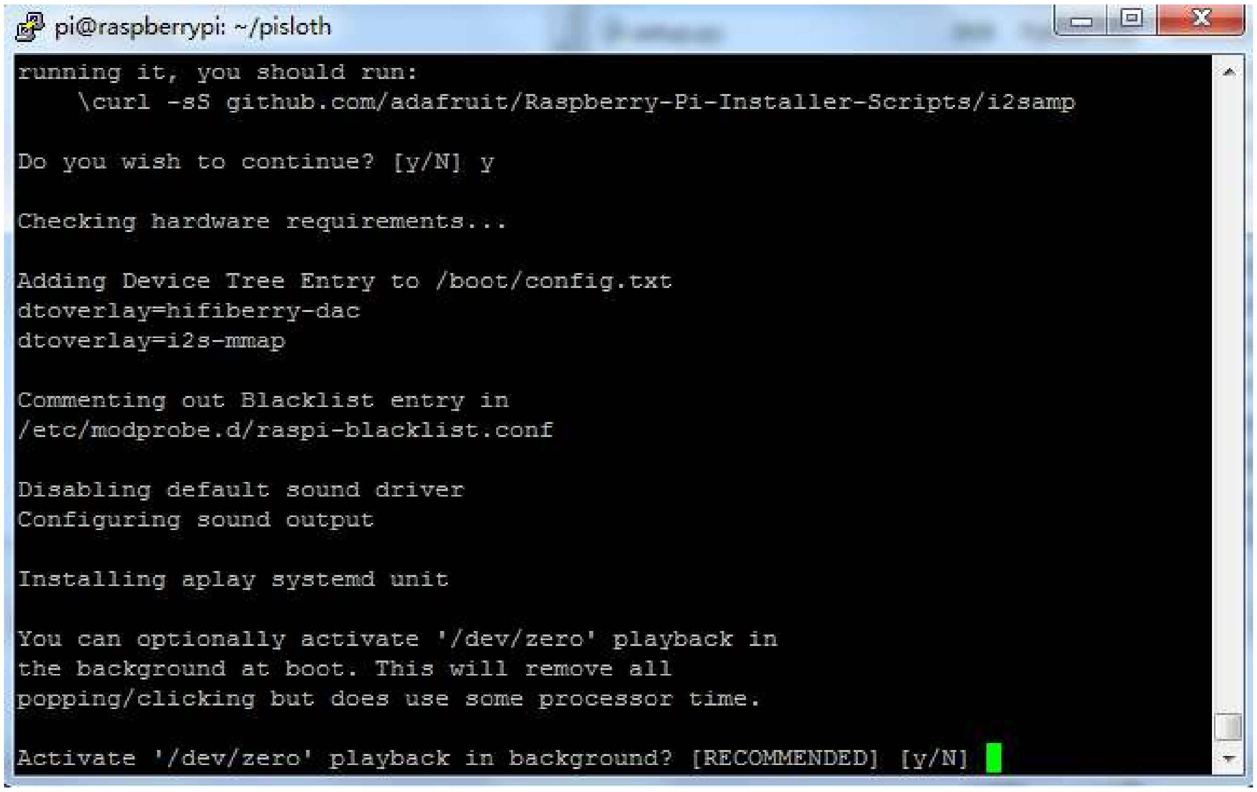
Finally you need to run the script `i2samp.sh` to install the components needed for the i2s amplifier, otherwise it may not have sound.

```
cd /home/pi/piarm
sudo bash i2samp.sh
```



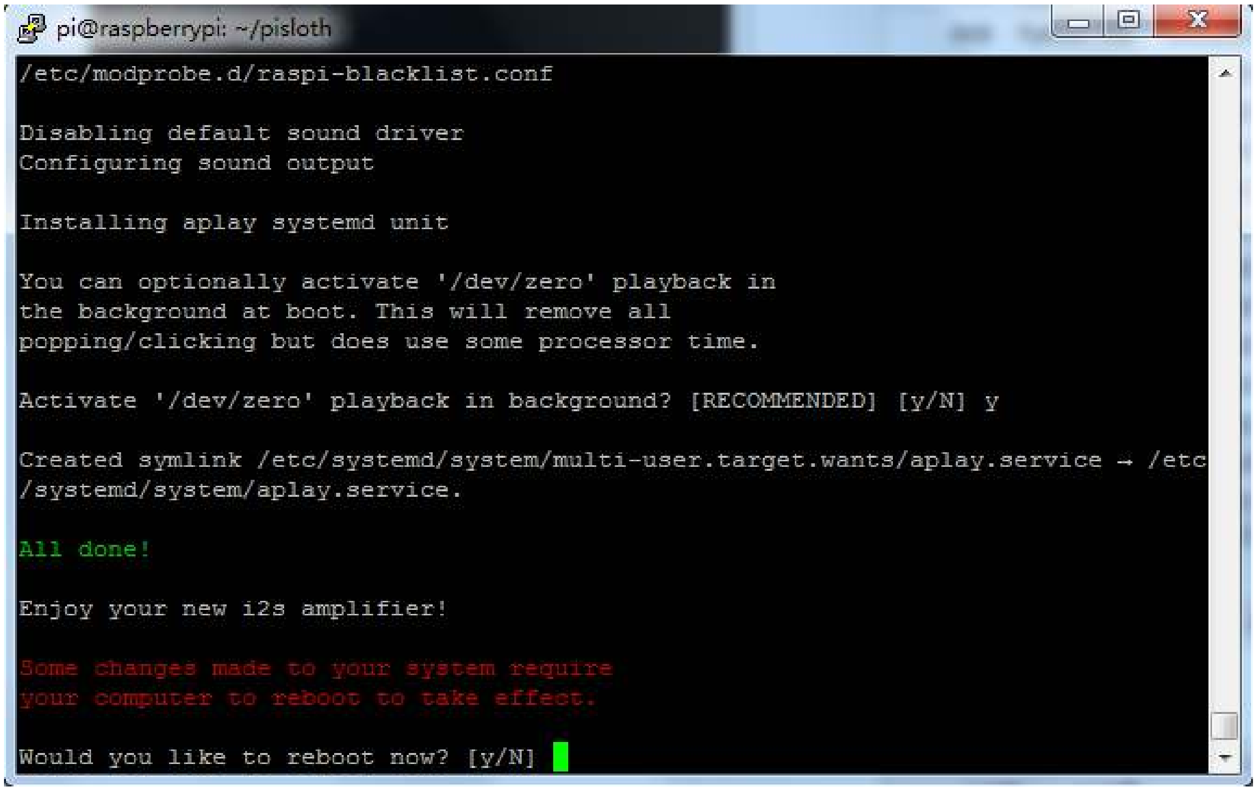
```
  File "/usr/local/lib/python3.7/dist-packages/robot_hat-1.0.0-py3.7.egg/robot_h
at/robot.py", line 91, in servo_move
    time.sleep(step_delay)
KeyboardInterrupt
pi@raspberrypi:~/pisloth/examples $ cd ..
pi@raspberrypi:~/pisloth $ sudo bash i2samp.sh
Support for your operating system is experimental. Please visit
forums.adafruit.com if you experience issues with this product.


This script will install everything needed to use
i2s amplifier

--- Warning ---

Always be careful when running scripts and commands
copied from the internet. Ensure they are from a
trusted source.

If you want to see what this script does before
running it, you should run:
    \curl -sS github.com/adafruit/Raspberry-Pi-Installer-Scripts/i2samp

Do you wish to continue? [y/N]
```

Type y and press Enter to continue running the script.



```
running it, you should run:
    \curl -sS github.com/adafruit/Raspberry-Pi-Installer-Scripts/i2samp

Do you wish to continue? [y/N] y

Checking hardware requirements...

Adding Device Tree Entry to /boot/config.txt
dtoverlay=hifiberry-dac
dtoverlay=i2s-mmap

Commenting out Blacklist entry in
/etc/modprobe.d/raspi-blacklist.conf

Disabling default sound driver
Configuring sound output

Installing aplay systemd unit

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N]
```

Type y and press Enter to make /dev/zero run in the background.



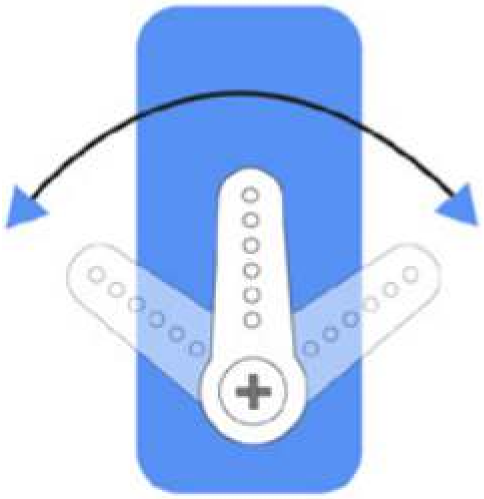
```
/etc/modprobe.d/raspi-blacklist.conf

Disabling default sound driver
Configuring sound output

Installing aplay systemd unit

You can optionally activate '/dev/zero' playback in
the background at boot. This will remove all
popping/clicking but does use some processor time.

Activate '/dev/zero' playback in background? [RECOMMENDED] [y/N] y

Created symlink /etc/systemd/system/multi-user.target.wants/aplay.service → /etc
/systemd/system/aplay.service.

All done!

Enjoy your new i2s amplifier!

Some changes made to your system require
your computer to reboot to take effect.

Would you like to reboot now? [y/N]
```

Enter `y` and press `Enter` to restart the robot.

Note: If there is no sound after a restart, you may need to run the i2samp.sh script several times.

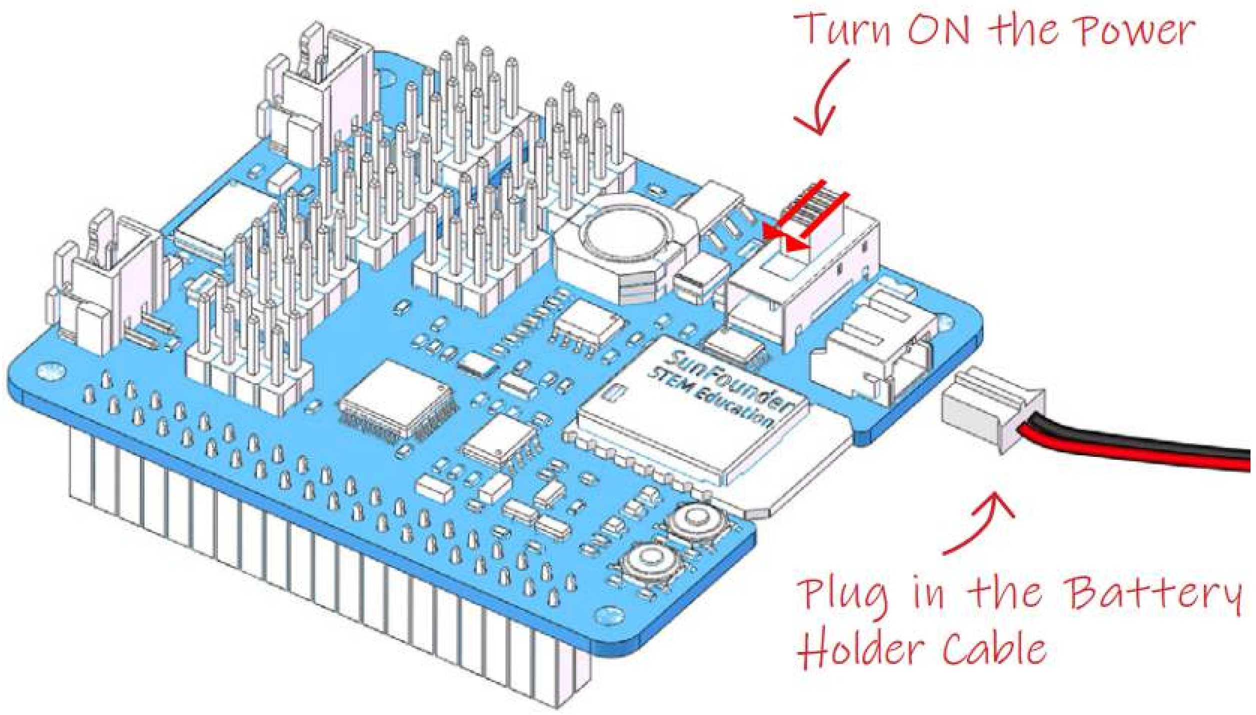
4.1.5 Servo Adjust

To ensure that the servo has been properly set to 0°, first insert the rocker arm into the servo shaft and then gently rotate the rocker arm to a different angle.



Follow the instructions on the assembly foldout, insert the battery holder cable and turn the power switch to the ON.

Wait for 1-2 minutes, there will be a sound to indicate that the Raspberry Pi boots successfully.

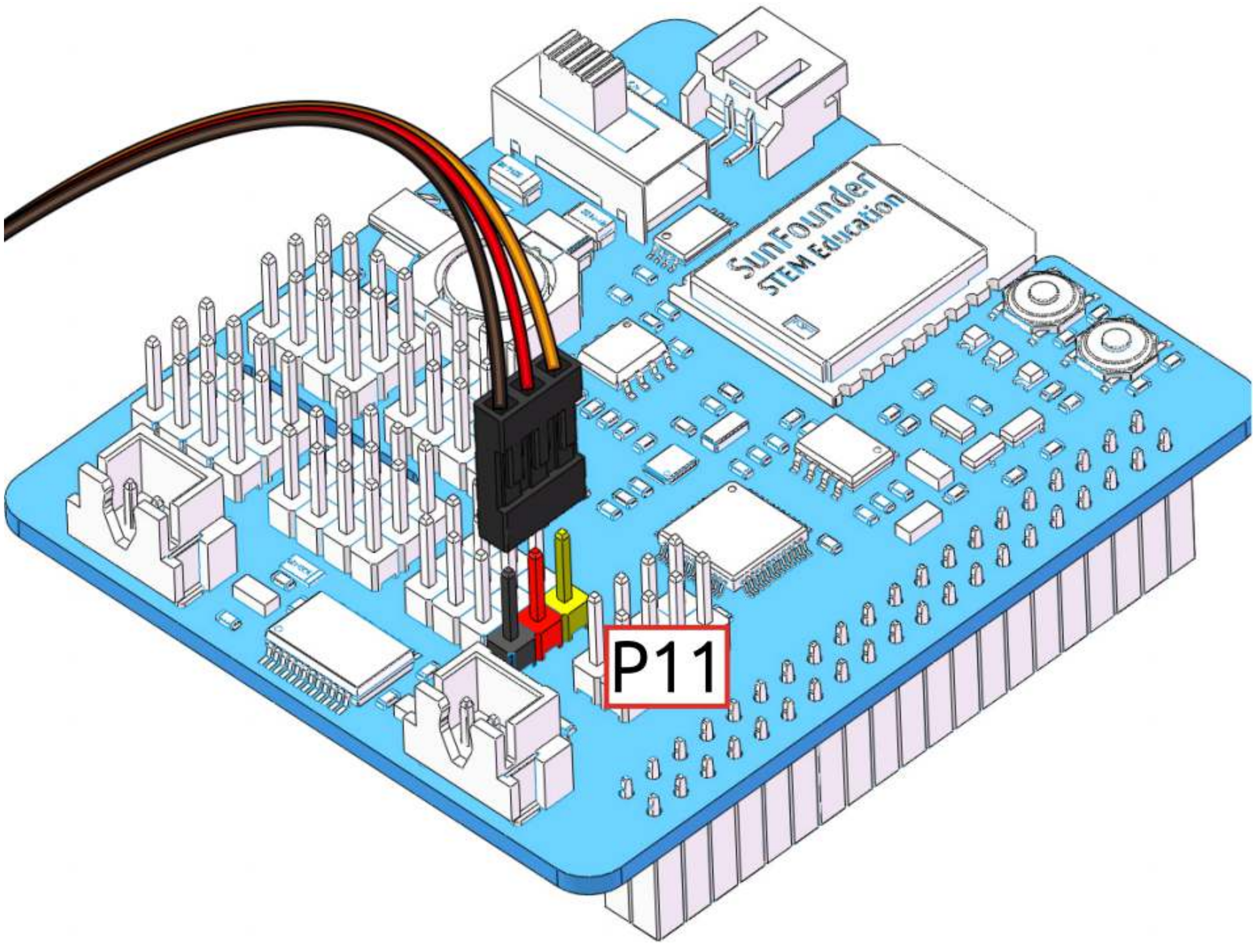


Now, run `servo_zeroing.py` in the `examples/` folder.

```
cd /home/pi/piarm/examples
sudo python3 servo_zeroing.py
```

Note: If you get an error, try re-enabling the Raspberry Pi's I2C port, see: *I2C Configuration*.

Next, plug the servo cable into the P11 port as follows.



At this point you will see the servo arm rotate to a specific position (0°). If the servo arm does not return to 0°, press the RST button to restart the Robot HAT.

Now you can continue the installation as instructed on the assembly foldout.

Note:

- Do not unplug this servo cable before fixing it with the servo screw, you can unplug it after fixing it.
- Do not rotate the servo while it is powered on to avoid damage; if the servo shaft is not inserted at the right angle, pull the servo out and reinsert it.
- Before assembling each servo, you need to plug the servo cable into P11 and turn on the power to set its angle to 0°.

After the assembly is complete, you can try to run the projects below.

4.2 Test 3 EoATs

This is the first program, and the one you must see.

In this program, you will learn how to assemble and use PiArm's 3 end-of-arm tools.

4.2.1 Shovel Bucket

Run the code

```
cd /home/pi/piarm/examples
sudo python3 shovel.py
```

After running the code, you will see the Shovel Bucket moving back and forth. But you need to assemble *Shovel Bucket* first.

Code

```
from robot_hat import Robot,Servo,PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)
arm = PiArm([1,2,3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0,0,0])

if __name__ == "__main__":
    while True:
        arm.set_bucket(-50)
        sleep(1)
        arm.set_bucket(90)
        sleep(1)
```

How it work

```
from robot_hat import Robot,Servo,PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm
```

- First, import the `Robot`, `servo`, and `PWM` classes from robot_hat.
- Import the `reset_mcu` class from the `robot_hat.utils` module, which is used to reset the MCU, to avoid conflicts between programs that can cause communication errors.
- Import the `sleep` class from the `time` module, which is used to implement the time delay function in seconds.
- Import the `PiArm` class from the `piarm` module, which is used to control PiArm.

```
reset_mcu()
sleep(0.01)
arm = PiArm([1,2,3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0,0,0])
```

Initialize the MCU first, then initialize the individual servo connection pins of PiArm and the connection pin of the bucket.

- `PiArm( )`: Initialize the 3 servo pins on the Arm.
- `bucket_init( )`: Set the pin of the bucket.
- `set_offset( )`: Set the offset value of the 3 servos on the Arm.

```
while True:
    arm.set_bucket(-50)
    sleep(1)
    arm.set_bucket(90)
    sleep(1)
```

This code is used to move the bucket back and forth between -50 and 90 degrees with a time interval of 1 second.

- `set_bucket()`: Used to control the rotation angle of the bucket.

4.2.2 Hanging Clip

Run the code

```
cd /home/pi/piarm/examples
sudo python3 clip.py
```

After running the code, you will see the Hanging Clip repeatedly opening and closing. But you need to assemble *Hanging Clip* first.

Code

```
from robot_hat import Robot,Servo,PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)
arm = PiArm([1,2,3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0,0,0])

if __name__ == "__main__":
    while True:
        arm.set_hanging_clip(-50)
        sleep(1)
        arm.set_hanging_clip(90)
        sleep(1)
```

- `hanging_clip_init( )`: Used to initialize the pin of the Hanging Clip.
- `set_hanging_clip( )`: used to set the rotation angle of the Hanging Clip.

4.2.3 Electromagnet

Run the code

```
cd /home/pi/piarm/examples
sudo python3 electromagnet.py
```

After running the code, you will see that **Electromagnet** is energized every second, the LED (D2) on the electromagnet lights up to indicate that it is energized, at which point it can attach some material with the iron.

But you need to assemble *Electromagnet* first.

Code

```
from robot_hat import Robot,Servo,PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)
arm = PiArm([1,2,3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0,0,0])

if __name__ == "__main__":
    while True:
        arm.set_electromagnet('on')
        sleep(1)
        arm.set_electromagnet('off')
        sleep(1)
```

- `electromagnet_init( )`: Used to initialize the connection of the Electromagnet.
- `set_electromagnet( )`: Used to control the Electromagnet on/off.

4.3 Sound Effects

In this example, we use the sound effects of PiArm (Robot HAT to be exact). It consists of three parts: Muisc, Sound, and Text to Speech.

Install i2samp

Before using this function, please activate the speaker so that it can produce sound.

Run `i2samp.sh`, this script will install everything you need to use the i2s amplifier.

```
cd /home/pi/piarm/
sudo bash i2samp.sh
```

There will be several prompts to confirm the request. Respond to all prompts with `Y`. After making changes to the Raspberry Pi system, you will need to reboot the computer for these changes to take effect.

After restarting, `i2samp.sh` runs the script again to test the amplifier. If the speaker successfully plays sound, the configuration is complete.

Run the code

```
cd /home/pi/piarm/examples
sudo python3 sound_effect.py
```

After the code is run, you will find that PiArm first plays the sound effect in the sound function, and then plays the background music. When the background music is played, the [tts] function is run for timing, and the countdown voice broadcast will be performed after 30 seconds.

Code

```
from robot_hat import Music,TTS
from time import sleep

m = Music()
t = TTS()

def sound():
    song = './sounds/sign.wav'
    m.music_set_volume(40)
    m.sound_play(song)

def background_music():
    music = './musics/sports-Ahjay_Stelino.mp3'
    m.music_set_volume(50)
    m.background_music(music)

def tts():
    t.say("timing begins")
    sleep(1)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("Stop music")
    sleep(1)

if __name__ == "__main__":
    background_music()
    sleep(10)
    #sound()
    #tts()
    while True:
        #background_music()
        sound()
        tts()
```

How it works?

The code is simple, it creates 3 functions `sound()`, `background()` and `tts()`, and then calls them separately to make PiArm play music and speak.

```
def sound():
    song = './sounds/sign.wav'
    m.music_set_volume(40)
    m.sound_play(song)
```

Play the sound effect `. /sounds/sign.wav` at 40% volume.

- `music_set_volume()`: Set volume, range is 0%-100%.
- `sound_play()`: Play a sound in a specific path.

```
def background_music():
    music = './musics/sports-Ahjay_Stelino.mp3'
    m.music_set_volume(50)
    m.background_music(music)
```

Play background music `./musics/sports-Ahjay_Stelino.mp3` at 50% volume.

- `background_music()`: Play the background music in a specific path.

```
def tts():
    t.say("timing begins")
    sleep(1)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("Stop music")
    sleep(1)
```

Write text to PiArm to make it speak.

- `say()`: Writing characters or strings in parentheses will make PiArm speak them out.

4.4 Dual Joystick Module Control

In this project, we will use the Dual Joystick Module that comes with the kit to control the PiArm.

- *Shovel Bucket - Joystick Control*
- *Hanging Clip - Joystick Control*
- *Electromagnet - Joystick Control*

4.4.1 Shovel Bucket - Joystick Control

```
cd /home/pi/piarm/examples
sudo python3 joystick_module1.py
```

Once the code is run, you will be able to control the rotation of PiArm's arm by toggling the left and right joysticks, and controlling the angle of the Shovel Bucket by pressing the left and right joysticks respectively.

But you need to assemble *Shovel Bucket* to PiArm first.

Code

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep

from piarm import PiArm

reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0,0,0])
```

(continues on next page)

(continued from previous page)

```python

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if leftJoystick.read_status() == "up":
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True
    if leftJoystick.read_status() == "pressed":
        bucket += 2
        flag = True
    elif rightJoystick.read_status() == "pressed":
        bucket -= 2
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s '%(arm.servo_positions,arm.
↪component_staus))

if __name__ == "__main__":
    while True:
        _angles_control()
        sleep(0.01)
```

How it works?

```python
leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))
```

Define the X,Y and Z pin connections for the left and right joysticks.

```python
def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if leftJoystick.read_status() == "up":
```

(continues on next page)

(continued from previous page)

```
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True
    if leftJoystick.read_status() == "pressed":
        bucket += 2
        flag = True
    elif rightJoystick.read_status() == "pressed":
        bucket -= 2
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s '%(arm.servo_positions,arm.
↪component_staus))
```

In this code, the `_angles_control()` function is created to control the PiArm.

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the Arm respectively, refer to: *Angle Mode*.
- If the **left** joystick is toggled up, `alpha` increases and the Arm will extend forward.
- If the **left** joystick is toggled down, `alpha` decreases and the Arm will retract backward.
- If the **left** joystick is toggled to the left, `gamma` increases and the Arm will turn left.
- If the **left** joystick is toggled to the right, `gamma` decreases and the Arm will turn right.
- If the **right** joystick is toggled up, `beta` increases and the Arm will raise up.
- If the **right** joystick is toggled down, `beta` decreases and the Arm will lower down.
- Finally, use the left and right joystick buttons to control the angle of the Shovel Bucket respectively.

4.4.2 Hanging Clip - Joystick Control

Run the code

```
cd /home/pi/piarm/examples
sudo python3 joystick_module2.py
```

Once the code is running, you will be able to control the rotation of PiArm's arm by toggling the left and right joysticks, and control the opening/closing of the Hanging Clip by pressing the left and right joysticks respectively.

But you need to assemble *Hanging Clip* to PiArm first.

Code

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep

from piarm import PiArm

reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0,0,0])

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    clip = arm.component_staus

    if leftJoystick.read_status() == "up":
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True

    if leftJoystick.read_status() == "pressed":
        clip += 2
        flag = True
    elif rightJoystick.read_status() == "pressed":
        clip -= 2
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_hanging_clip(clip)
        print('servo angles: %s , clip angle: %s '%(arm.servo_positions,arm.component_
↪staus))

if __name__ == "__main__":
    while True:
```

(continues on next page)

(continued from previous page)

```
        _angles_control()
        sleep(0.01)
```

In this code, the `_angles_control()` function is created to control the PiArm.

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the Arm respectively, refer to: *Angle Mode*.
- If the **left** joystick is toggled up, `alpha` increases and the Arm will extend forward.
- If the **left** joystick is toggled down, `alpha` decreases and the Arm will retract backward.
- If the **left** joystick is toggled to the left, `gamma` increases and the Arm will turn left.
- If the **left** joystick is toggled to the right, `gamma` decreases and the Arm will turn right.
- If the **right** joystick is toggled up, `beta` increases and the Arm will raise up.
- If the **right** joystick is toggled down, `beta` decreases and the Arm will lower down.
- Finally, use the left and right joystick buttons to control the angles of the Hanging Clip respectively.

4.4.3 Electromagnet - Joystick Control

Run the code

```
cd /home/pi/piarm/examples
sudo python3 joystick_module3.py
```

Once the code is run, you will be able to control the rotation of PiArm's arm by toggling the left and right joysticks, and controlling the on/off of the Electromagnet by pressing the left and right joysticks respectively.

But you need to assemble *Electromagnet* to PiArm first.

Code

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep

from piarm import PiArm

reset_mcu()
sleep(0.01)


leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0,0,0])

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    status = ""

    if leftJoystick.read_status() == "up":
```

(continues on next page)

(continued from previous page)

```
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True
    if leftJoystick.read_status() == "pressed":
        arm.set_electromagnet('on')
        status = "electromagnet is on"
    elif rightJoystick.read_status() == "pressed":
        arm.set_electromagnet('off')
        status = "electromagnet is off"

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        print('servo angles: %s , electromagnet status: %s '%(arm.servo_positions,
↪status))

if __name__ == "__main__":
    while True:
        _angles_control()
        sleep(0.01)
```

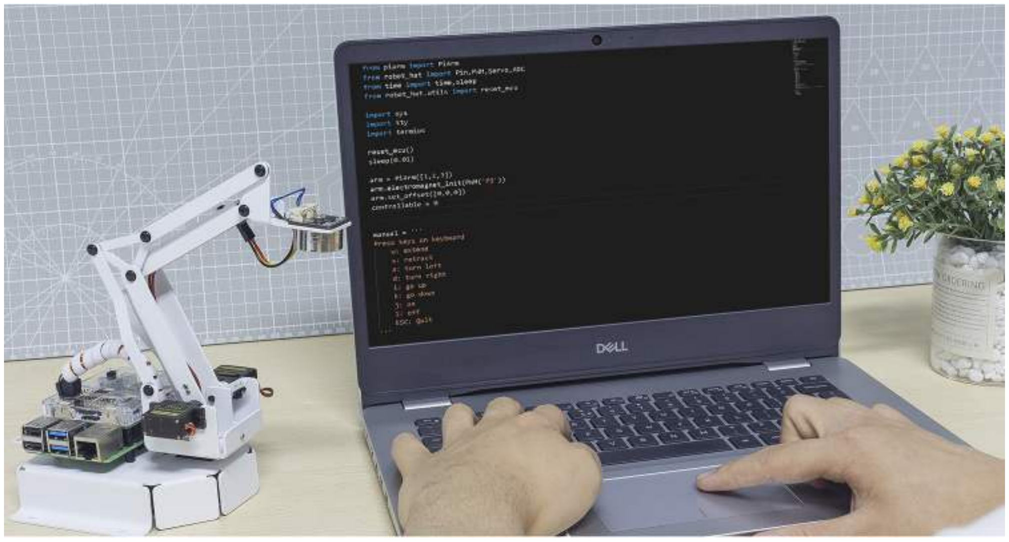
In this code, the `_angles_control()` function is created to control the PiArm.

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the Arm respectively, refer to: *Angle Mode*.
- If the **left** joystick is toggled up, `alpha` increases and the Arm will extend forward.
- If the **left** joystick is toggled down, `alpha` decreases and the Arm will retract backward.
- If the **left** joystick is toggled to the left, `gamma` increases and the Arm will turn left.
- If the **left** joystick is toggled to the right, `gamma` decreases and the Arm will turn right.
- If the **right** joystick is toggled up, `beta` increases and the Arm will raise up.
- If the **right** joystick is toggled down, `beta` decreases and the Arm will lower down.
- Finally, use the left and right joystick buttons to control the on/off of the Electromagnet respectively.

4.5 Keyboard Control

In this project, we will use `w`, `s`, `a`, `d`, `i`, `k`, `j` and `l` on the keyboard to control the PiArm.

- *Shovel Bucket - Keyboard Coboardntrol*
- *Hanging Clip - Keyboard Control*
- *Electromagnet - Keyboard Control*



```
from piarm import PiArm
from robot_hat import Pin,PWM,Servo,ADC
from time import time,sleep
from robot_hat.utils import reset_mcu

import sys
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1,2,3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0,0,0])
controllable = 0


manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: on
    l: off
    ESC: Quit
'''
```

4.5.1 Shovel Bucket - Keyboard Coboardntrol

Run the code

```
cd /home/pi/piarm/examples
sudo python3 keyboard_control1.py
```

After running the code, follow the prompts and press the keys on the keyboard to control the PiArm's arm and Shovel Bucket.

But you need to assemble *Shovel Bucket* on the PiArm first.

Note:

- To switch the keyboard to lowercase English input.
- `w`, `s`, `a`, `d`, `i` and `k` are used to control the rotation of the arm.
- `j` and `l` are used to control the angle of the Shovel Bucket.

Code

```python
from piarm import PiArm
from robot_hat import Pin,PWM,Servo,ADC
from time import time,sleep
from robot_hat.utils import reset_mcu

import sys
```

(continues on next page)

(continued from previous page)

```
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1,2,3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0,0,0])
controllable = 0


manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: open
    l: close
    ESC: Quit
'''

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
```

(continues on next page)

(continued from previous page)

```
    elif key == 'k':
        beta -= 3
        flag = True
    if key == 'j':
        bucket -= 1
        flag = True
    elif key == 'l':
        bucket += 1
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s '%(arm.servo_positions,arm.
↪component_staus))


if __name__ == "__main__":

    print(manual)

    while True:
        key = readchar().lower()
        control(key)
        if key == chr(27):
            break
```

How it works?

```
def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch
```

This function references the standard input stream and returns the first character of the read data stream.

- `tty.setraw(sys.stdin.fileno)` is to change the standard input stream to raw mode, i.e. all characters will not be escaped during transmission, including special characters.
- `old_settings = termios.tcgetattr(fd)` and `termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)` and acts as a backup and restore.

```
def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus

    if key == 'w':
        alpha += 3
        flag = True
```

(continues on next page)

(continued from previous page)

```
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
    elif key == 'k':
        beta -= 3
        flag = True
    if key == 'j':
        bucket -= 1
        flag = True
    elif key == 'l':
        bucket += 1
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s '%(arm.servo_positions,arm.
↪component_staus))
```

In this code, the `control()` function is created to control the PiArm by reading the key values on the keyboard.

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the arm respectively, refer to: *Angle Mode*.
- Press the `w` key on the keyboard, the `alpha` increases and the Arm will extend forward.
- Press the `s` key on the keyboard, the `alpha` decreases and the Arm will retract backward.
- Press the `a` key on the keyboard, the `gamma` increases and the Arm will turn left.
- Press the `d` key on the keyboard, the `gamma` decreases and the Arm will turn right.
- Press the `i` key on the keyboard, the `beta` increases and the Arm will raise up.
- Press the `k` key on the keyboard, the `beta` decreases and the Arm will lower down.
- Finally, use the `k` and `l` keys to control the angle of the Shovel Bucket respectively.

```
while True:
    key = readchar().lower()
    control(key)
    if key == chr(27):
        break
```

Call `readchar()` in the main program to read the key value, then pass the read key value into the `control()` function so that PiArm will move according to the different keys. `key == chr(27)` represents the key `Esc` key press.

4.5.2 Hanging Clip - Keyboard Control

Run the code

```
cd /home/pi/piarm/examples
sudo python3 keyboard_control2.py
```

After running the code, follow the prompts and press the keys on the keyboard to control the Arm and Hanging Clip of PiArm.

But you need to assemble *Hanging Clip* to PiArm first.

Note:

- To switch the keyboard to lowercase English input.
- `w`, `s`, `a`, `d`, `i` and `k` are used to control the rotation of the arm.
- `j` and `l` are used to control the opening and closing of the Hanging Clip.

Code

```python
from piarm import PiArm
from robot_hat import Pin,PWM,Servo,ADC
from time import time,sleep
from robot_hat.utils import reset_mcu

import sys
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1,2,3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0,0,0])
controllable = 0


manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: open
    l: close
    ESC: Quit
'''

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
```

(continues on next page)

(continued from previous page)

```
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    clip = arm.component_staus

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
    elif key == 'k':
        beta -= 3
        flag = True

    if key == 'j':
        clip -= 1
        flag = True
    elif key == 'l':
        clip += 1
        flag = True

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_hanging_clip(clip)
        print('servo angles: %s , clip angle: %s '%(arm.servo_positions,arm.component_
↪staus))


if __name__ == "__main__":

    print(manual)

    while True:
        key = readchar().lower()
        control(key)
        if key == chr(27):
            break
```

In this code, the `control()` function is created to control the PiArm by reading the key values on the keyboard.

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the arm respectively, refer to: *Angle Mode*.

- Press the w key on the keyboard, the alpha increases and the Arm will extend forward.
- Press the s key on the keyboard, the alpha decreases and the Arm will retract backward.
- Press the a key on the keyboard, the gamma increases and the Arm will turn left.
- Press the d key on the keyboard, the gamma decreases and the Arm will turn right.
- Press the i key on the keyboard, the beta increases and the Arm will raise up.
- Press the k key on the keyboard, the beta decreases and the Arm will lower down.
- Finally, use the k and l keys to control the opening and closing of the Hanging Clip respectively.

4.5.3 Electromagnet - Keyboard Control

Run the code

```
cd /home/pi/piarm/examples
sudo python3 keyboard_control1.py
```

After running the code, follow the prompts and press the keys on the keyboard to control the PiArm's arms and Electromagnet.

But you need to assemble *Electromagnet* to PiArm first.

Note:

- To switch the keyboard to lowercase English input.
- w, s, a, d, i and k are used to control the rotation of the arm.
- j and l are used to control the ON and OFF of the Electromagnet.

Code

```
from piarm import PiArm
from robot_hat import Pin,PWM,Servo,ADC
from time import time,sleep
from robot_hat.utils import reset_mcu

import sys
import tty
import termios

reset_mcu()
sleep(0.01)

arm = PiArm([1,2,3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0,0,0])
controllable = 0


manual = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
```

(continues on next page)

(continued from previous page)

```
    d: turn right
    i: go up
    k: go down
    j: on
    l: off
    ESC: Quit
'''

def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

def control(key):

    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    status = ""

    if key == 'w':
        alpha += 3
        flag = True
    elif key == 's':
        alpha -= 3
        flag = True
    if key == 'a':
        gamma += 3
        flag = True
    elif key == 'd':
        gamma -= 3
        flag = True
    if key == 'i':
        beta += 3
        flag = True
    elif key == 'k':
        beta -= 3
        flag = True

    if key == 'j':
        arm.set_electromagnet('on')
    elif key == 'l':
        arm.set_electromagnet('off')

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        print('servo angles: %s , electromagnet status: %s '%(arm.servo_positions,
↪status))


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    print(manual)

    while True:
        key = readchar().lower()
        control(key)
        if key == chr(27):
            break
```

In this code, the `control()` function is created to control the PiArm by reading the key values on the keyboard.

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the arm respectively, refer to: *Angle Mode*.
- Press the `w` key on the keyboard, the `alpha` increases and the Arm will extend forward.
- Press the `s` key on the keyboard, the `alpha` decreases and the Arm will retract backward.
- Press the `a` key on the keyboard, the `gamma` increases and the Arm will turn left.
- Press the `d` key on the keyboard, the `gamma` decreases and the Arm will turn right.
- Press the `i` key on the keyboard, the `beta` increases and the Arm will raise up.
- Press the `k` key on the keyboard, the `beta` decreases and the Arm will lower down.
- Finally, use the `k` and `l` keys to control the ON and OFF of the Electromagnet respectively.

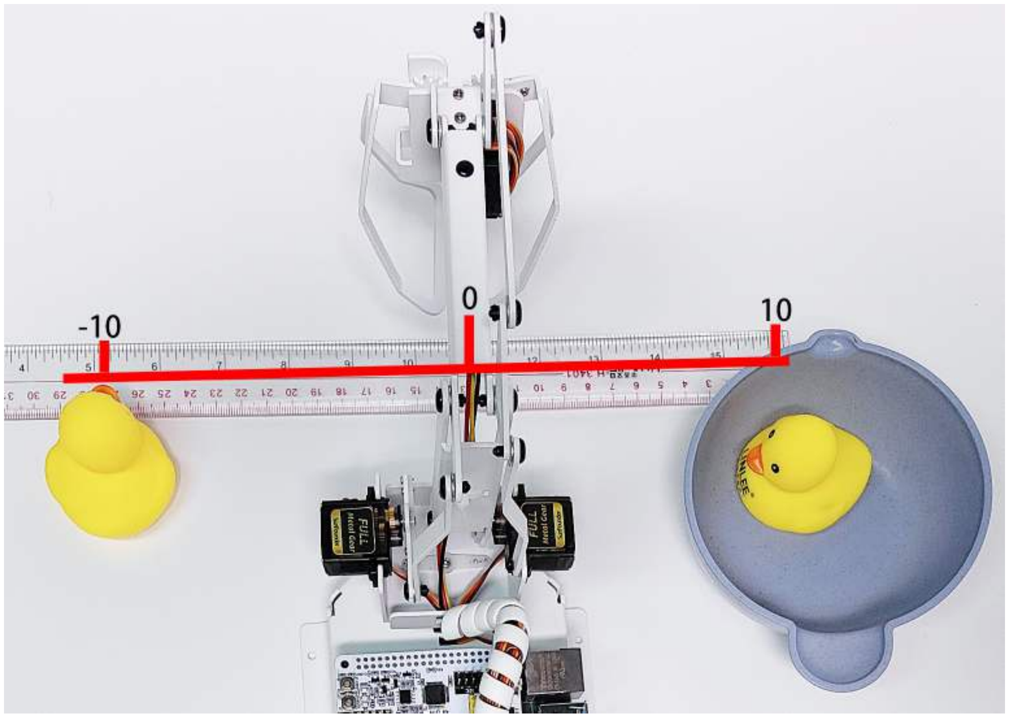
4.6 Coordinate Mode

PiArm's arm has 2 control modes: *Angle Mode* and *Coordinate Mode*.

- *Angle Mode*: Write a certain angle to the 3 servos of the arm to make the arm reach a specific position.
- *Coordinate Mode*: Create a spatial coordinate system for the arm, set a control point, and write 3D coordinates to this control point to make the arm reach a specific position.

The *Coordinate Mode* is used in this project.

This project sets 2 coordinate points by coordinate mode, and let the PiArm clip the rubber duck on the left to the bowl on the right. But you need to mount *Hanging Clip* to the PiArm first.



4.6.1 Programming

Run the code

```
cd /home/pi/piarm/examples
sudo python3 coordinate_mode.py
```

After the code is run, after the code is run, you will be able to control the rotation of PiArm's Arm by toggling the left and right joystick, and control the angle of the Shovel Bucket by pressing the left and right joystick respectively.

But you need to assemble *Hanging Clip* to PiArm first.

Code

```
from re import M
from robot_hat import PWM
from robot_hat.utils import reset_mcu
from time import sleep
from piarm import PiArm

reset_mcu()
sleep(0.01)

" Grab an object from one coordinate to another coordinate"

arm = PiArm([1,2,3])
arm.set_offset([0,0,0])
arm.hanging_clip_init(PWM('P3'))


if __name__ == "__main__":
```

(continues on next page)

(continued from previous page)

```
    start_coord = [-100, 40, 20] # x,y,z
    end_coord = [100, 40, 30] # x,y,z


    arm.set_speed(60)
    arm.set_hanging_clip(20)
    arm.do_by_coord(start_coord)
    arm.set_hanging_clip(90)

    start_coord_up = [start_coord[0], start_coord[1], 80]
    arm.do_by_coord(start_coord_up)

    end_coord_up = [end_coord[0], end_coord[1], 80]
    arm.do_by_coord(end_coord_up)

    arm.do_by_coord(end_coord)
    arm.set_hanging_clip(20)
    arm.do_by_coord(end_coord_up)
```

How it works?

```
start_coord = [-100, 40, 20] # x,y,z
end_coord = [100, 40, 30] # x,y,z
```

- `start_coord`The coordinates of the left rubber duck.
- `end_coord`: The coordinates of the bowl.

Note:

- All coordinates here refer to the coordinates of the control points, but the actual distance between the X and Y coordinates is a little larger when the end-of-arm tool is mounted.
- The tolerance distance is different for each end of arm tool. For example, 3-4cm for Hanging Clip and Electromagnet, 6-7cm for Shovel Bucket.
- For example, here the X coordinate is written as 100, but the actual distance is 13-14cm.
- It is generally recommended that the X coordinate is -80 ~ 80, but since the Y coordinate value here is small (the recommended range is 30~130), it is possible to reach to 100. However, if you increase the Y coordinate value, the X coordinate value needs to be reduced according to the actual situation due to the linkage action.

```
arm.set_speed(60)
arm.set_hanging_clip(20)
arm.do_by_coord(start_coord)
arm.set_hanging_clip(90)

start_coord_up = [start_coord[0], start_coord[1], 80]
arm.do_by_coord(start_coord_up)

end_coord_up = [end_coord[0], end_coord[1], 80]
arm.do_by_coord(end_coord_up)

arm.do_by_coord(end_coord)
arm.set_hanging_clip(20)
arm.do_by_coord(end_coord_up)
```

- PiArm opens the Hanging Clip (20°), then rotates to the left rubber duck position (`start_coord`), then closes the Hanging Clip (90°).
- PiArm raises his head (`start_coord_up`) and then turns to the right side above the bowl (`end_coord_up`).
- PiArm lowers his head (`end_coord_up`), then opens the Hanging Clip (20°) to let the rubber duck fall into the bowl, and finally raises his head again (`end_coord_up`).

4.7 Memory function

PiArm provides a function to record actions, which allows PiArm to do some repetitive actions automatically.

In this project, let's see how to implement this function.

Run the code

```
cd /home/pi/piarm/examples
sudo python3 memory_function.py
```

After the code is run, you can use the left and right joystick to control the rotation of PiArm and the Shovel Bucket (But you need to assemble *Shovel Bucket* to PiArm first), press the left joystick to record one movement of PiArm, after recording several sets of movements, you can press the right joystick to make PiArm to reproduce these movements.

Only record the changes between points, if the starting point and the end point are the same, and you do many moves in between, but only press once to record, it will go directly from the starting point to the end point, and will not record the middle process.

Code

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from robot_hat import TTS
from time import sleep
from piarm import PiArm

t = TTS()
reset_mcu()
sleep(0.01)

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.bucket_init(PWM('P3'))
arm.set_offset([0,0,0])

def _angles_control():
    arm.speed = 100
    flag = False
    alpha,beta,gamma = arm.servo_positions
    bucket = arm.component_staus
    global i

    if leftJoystick.read_status() == "up":
        alpha += 1
        flag = True
    elif leftJoystick.read_status() == "down":
```

(continues on next page)

(continued from previous page)

```
        alpha -= 1
        flag = True
    if leftJoystick.read_status() == "left":
        gamma += 1
        flag = True
    elif leftJoystick.read_status() == "right":
        gamma -= 1
        flag = True
    if rightJoystick.read_status() == "up":
        beta += 1
        flag = True
    elif rightJoystick.read_status() == "down":
        beta -= 1
        flag = True

    if rightJoystick.read_status() == "left":
        bucket += 2
        flag = True
    elif rightJoystick.read_status() == "right":
        bucket -= 2
        flag = True

    if leftJoystick.read_status() == "pressed":
        arm.record()
        t.say("record")
        print('step %s : %s'%(i,arm.steps_buff[i*2]))
        i += 1
        sleep(0.05)
    elif rightJoystick.read_status() == "pressed":

        t.say("action")
        arm.set_speed(80)
        arm.record_reproduce(0.05)
        arm.set_speed(100)

    if flag == True:
        arm.set_angle([alpha,beta,gamma])
        arm.set_bucket(bucket)
        print('servo angles: %s , bucket angle: %s '%(arm.servo_positions,arm.
↪component_staus))

if __name__ == "__main__":
    print(arm.servo_positions)
    i = 0
    while True:
        _angles_control()
        sleep(0.01)
```

How it works?

In this code, let's focus on the `_angles_control()` function, which is used to read the value of the dual joystick and then perform different operations.

1. control the movement of the Arm

```
if leftJoystick.read_status() == "up":
    alpha += 1
```

(continues on next page)

(continued from previous page)

```
    flag = True
elif leftJoystick.read_status() == "down":
    alpha -= 1
    flag = True
if leftJoystick.read_status() == "left":
    gamma += 1
    flag = True
elif leftJoystick.read_status() == "right":
    gamma -= 1
    flag = True
if rightJoystick.read_status() == "up":
    beta += 1
    flag = True
elif rightJoystick.read_status() == "down":
    beta -= 1
    flag = True
```

- `alpha`, `beta` and `gamma` refer to the angles of the 3 servos on the Arm respectively, refer to: *Angle Mode*.
- If the **left** joystick is toggled up, `alpha` increases and the Arm will extend forward.
- If the **left** joystick is toggled down, `alpha` decreases and the Arm will retract backward.
- If the **left** joystick is toggled to the left, `gamma` increases and the Arm will turn left.
- If the **left** joystick is toggled to the right, `gamma` decreases and the Arm will turn right.
- If the **right** joystick is toggled up, `beta` increases and the Arm will raise up.
- If the **right** joystick is toggled down, `beta` decreases and the Arm will lower down.

2. Control the angle of the Shovel Bucket

```
if rightJoystick.read_status() == "left":
    bucket += 2
    flag = True
elif rightJoystick.read_status() == "right":
    bucket -= 2
    flag = True
```

- Right joystick toggles to the left to allow the Shovel Bucket to rewind.
- Right joystick toggles to the right to extend the bucket outward.

3. Recording and reproducing actions

```
if leftJoystick.read_status() == "pressed":
    arm.record()
    t.say("record")
    print('step %s : %s'%(i,arm.steps_buff[i*2]))
    i += 1
    sleep(0.05)
elif rightJoystick.read_status() == "pressed":

    t.say("action")
    arm.set_speed(80)
    arm.record_reproduce(0.05)
    arm.set_speed(100)
```

- If the left joystick is pressed and the `record()` function is called to record the action, PiArm will say that it has recorded. The terminal will show the angle and the number of recorded moves at this point.
- If the right joystick is pressed, the `record_reproduce()` function is called to reproduce the recorded action, and PiArm will prompt to start doing the action.

4. Write the angles to PiArm

```
if flag == True:
    arm.set_angle([alpha,beta,gamma])
    arm.set_bucket(bucket)
    print('servo angles: %s , bucket angle: %s '%(arm.servo_positions,arm.component_
↪staus))
```

Write the angle of the Arm and the Shovel Bucket to PiArm and have it rotate to those angles.

If you have the Hanging Clip or Electromagnet connected to your PiArm, you can modify the above code by referring to the following parts.

- *Hanging Clip - Joystick Control*
- *Electromagnet - Joystick Control*

4.8 GAME - Catching Dolls

Now let's play a game of catching dolls and see who can catch more dolls with PiArm in the given time. In order to play this game, we need to implement two functions, the first one is to control PiArm with the dual joystick module, and the second one is to timing, when the countdown is over, we can't control PiArm anymore. These two parts must be executed simultaneously.

Run the code

```
cd /home/pi/piarm/examples
sudo python3 game_catching_dolls.py
```

After the code runs, press the left and right joystick at the same time to start the game. Then you can use the dual joystick module to control PiArm to catch the doll, please pay attention to the time, after 60 seconds, PiArm will tell the game is over and you will not be able to continue to control PiArm.

Code

```
from robot_hat import Servo,PWM,Joystick,ADC,Pin
from robot_hat.utils import reset_mcu
from time import sleep
from robot_hat import TTS
import threading

from piarm import PiArm

reset_mcu()
sleep(0.01)
t = TTS()

leftJoystick = Joystick(ADC('A0'),ADC('A1'),Pin('D0'))
rightJoystick = Joystick(ADC('A2'),ADC('A3'),Pin('D1'))

arm = PiArm([1,2,3])
arm.hanging_clip_init(PWM('P3'))
arm.set_offset([0,0,0])
arm.speed = 100
game_flag = 0

def control():

    alpha,beta,gamma = arm.servo_positions
    clip = arm.component_staus

    if leftJoystick.read_status() == "up":
        alpha += 1
    elif leftJoystick.read_status() == "down":
        alpha -= 1
    if leftJoystick.read_status() == "left":
        gamma += 1
    elif leftJoystick.read_status() == "right":
        gamma -= 1
    if rightJoystick.read_status() == "up":
        beta += 1
    elif rightJoystick.read_status() == "down":
        beta -= 1
    if leftJoystick.read_status() == "pressed":
        clip += 1
    elif rightJoystick.read_status() == "pressed":
        clip -= 1


    # if key_flag == True:
    arm.set_angle([alpha,beta,gamma])
    arm.set_hanging_clip(clip)
```

(continues on next page)

(continued from previous page)

```
        # print('coord: %s , servo angles: %s , clip angle: %s '%(arm.current_coord,
↪arm.servo_positions,arm.component_staus))

def timing():
    sleep(60)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("game over")
    global game_flag
    game_flag = 0

if __name__ == "__main__":

    thread1 = threading.Thread(target = timing)
    thread1.start()
    print("Press two joysticks at the same time to start the game")

    while True:
        if  leftJoystick.read_status() == "pressed" and rightJoystick.read_status()␣
↪== "pressed":
            t.say("timing begins")
            game_flag = 1
        if game_flag == 1:
            control()
```

How it works?

This code adds timing to the *Hanging Clip - Joystick Control* project.

```
def timing():
    sleep(60)
    t.say("three")
    sleep(1)
    t.say("two")
    sleep(1)
    t.say("one")
    sleep(1)
    t.say("game over")
    global game_flag
    game_flag = 0
```

Use the `sleep()` function to count down in 60 seconds, then let PiArm count down to 3, 2, 1, and when the time is over, let `game_flag` be 0, then PiArm will no longer be controlled.

```
if __name__ == "__main__":

    thread1 = threading.Thread(target = timing)
    thread1.start()
    print("Press two joysticks at the same time to start the game")
```

Let the `timing()` function run in a separate thread so that it can be timed while controlling PiArm.

```
while True:
    if  leftJoystick.read_status() == "pressed" and rightJoystick.read_status() ==
↪"pressed":
        t.say("timing begins")
        game_flag = 1
    if game_flag == 1:
        control()
```

This is the main flow of the code, when the left and right joysticks are pressed at the same time, PiArm says the timer starts, let `game_flag` be 1, then you can call `control()` function to control PiArm.

4.9 GAME - Iron Collection

In this project, prepare 3 shapes of iron pieces: triangle, circle and square. PiArm will randomly say a shape, and you need to control PiArm to put the corresponding shape of iron pieces into the corresponding box.

Run the code

```
cd /home/pi/piarm/examples
sudo python3 game_iron_collection.py
```

After the code is run, first press `p` on the keyboard to start the game, PiArm will prompt the game to start, then randomly say a shape (`Round`, `Triangle` and `Square`). You need to use `w`, `s`, `a`, `d`, `i` and `k` on the keyboard to control Arm, `j` and `l` to pick up the corresponding shape (you need to install *Electromagnet* to PiArm first.).

60 seconds later, the game will be prompted to end and you will no longer be able to control the PiArm. If you want to stop the code from running, you need to press the `Esc` key first, then press `Ctrl+C`.

Note:

- `w`, `s`, `a`, `d`, `i` and `k` are used to control the rotation of the Arm.
- `j` and `l` are used to control the ON and OFF of the Electromagnet.

Code

```
from piarm import PiArm
from robot_hat import Pin,PWM,Servo,ADC
from time import time,sleep
from robot_hat.utils import reset_mcu
from robot_hat import TTS

import threading
import sys
import tty
import termios
import random

reset_mcu()
sleep(0.01)
t = TTS()

arm = PiArm([1,2,3])
arm.electromagnet_init(PWM('P3'))
arm.set_offset([0,0,0])
```

(continues on next page)

(continued from previous page)

```
arm.speed = 100
flag = False



def readchar():
    fd = sys.stdin.fileno()
    old_settings = termios.tcgetattr(fd)
    try:
        tty.setraw(sys.stdin.fileno())
        ch = sys.stdin.read(1)
    finally:
        termios.tcsetattr(fd, termios.TCSADRAIN, old_settings)
    return ch

manual1 = '''
Press keys on keyboard
    p: Game Start
    ESC: Stop
'''

manual2 = '''
Press keys on keyboard
    w: extend
    s: retract
    a: turn left
    d: turn right
    i: go up
    k: go down
    j: on
    l: off
'''

# def control():

#     while flag == True:
#         arm.speed = 100
#         flag = False
#         alpha,beta,gamma = arm.servo_positions

def control(key):
    alpha,beta,gamma = arm.servo_positions

    if key == 'a':
        gamma += 3
    elif key == 'd':
        gamma -= 3
    if key == 's':
        alpha -= 3
    elif key == 'w':
        alpha += 3
    if key == 'i':
        beta += 3
    elif key == 'k':
        beta -= 3
    if key == 'j':
        arm.set_electromagnet('on')
```

(continues on next page)

(continued from previous page)

```
    elif key == 'l':
        arm.set_electromagnet('off')
    arm.set_angle([alpha,beta,gamma])


def timing():
    global flag
    while True:
        if flag == True:
            t.say("game start")
            sleep(60)
            t.say("three")
            sleep(1)
            t.say("two")
            sleep(1)
            t.say("one")
            sleep(1)
            t.say("game over")
            flag = False

def say_shape():
    k = random.randint(1,3)
    if k == 1:
        t.say("Round")
    if k == 2:
        t.say("Triangle")
    if k == 3:
        t.say("Square")

if __name__ == "__main__":

    print(manual1)

    thread1 = threading.Thread(target = timing)
    thread1.start()

    while True:
        key = readchar().lower()
        if  key == 'p':
            print(manual2)
            flag = True
            sleep(3)
            say_shape()
        if flag == True:
            control(key)
        if key == chr(27):
            print("press ctrl+c to quit")
            break
```

How it works?

This code is based on the project *Electromagnet - Keyboard Control* with the addition of timing and speaking random shapes.

```
def timing():
    global flag
    while True:
```

(continues on next page)

(continued from previous page)

```
        if flag == True:
            t.say("game start")
            sleep(60)
            t.say("three")
            sleep(1)
            t.say("two")
            sleep(1)
            t.say("one")
            sleep(1)
            t.say("game over")
            flag = False
```

This `timing()` function is used for timing. After prompting the game to start, the game is timed for 60 seconds, then a countdown of 3, 2, 1 is performed before the game is prompted to end and the `flag` is set to `False`.

```
def say_shape():
    k = random.randint(1,3)
    if k == 1:
        t.say("Round")
    if k == 2:
        t.say("Triangle")
    if k == 3:
        t.say("Square")
```

This `say_shape()` function is to make PiArm say a random shape.

```
if __name__ == "__main__":

    print(manual1)

    thread1 = threading.Thread(target = timing)
    thread1.start()

    while True:
        key = readchar().lower()
        if  key == 'p':
            print(manual2)
            flag = True
            sleep(3)
            say_shape()
        if flag == True:
            control(key)
        if key == chr(27):
            break
    print("press ctrl+c to quit")
```

This is the main flow of the code.

- Print out the key prompt in the terminal and let `timing()` run in a separate thread.
- Call the `readchar()` function to read the key value.
- If key `p` is read as being pressed, print out the key prompt and let `flag` be `True`, at which point the `timing()` function starts timing, and after 3 seconds, call the `say_shape()` function to make PiArm say a random shape.
- If `flag` is `True`, call the `control()` function to make the PiArm rotate according to the key value.
- `chr(27)` represents the `Esc` key, and if the `Esc` key is pressed, exit the main loop. This step is because the `readchar()` function is used to read the keyboard all the time, so you can't stop the code directly with

`Ctrl+C`.

- At this point, you can stop the code with `Ctrl+C`.

CHAPTER

FIVE

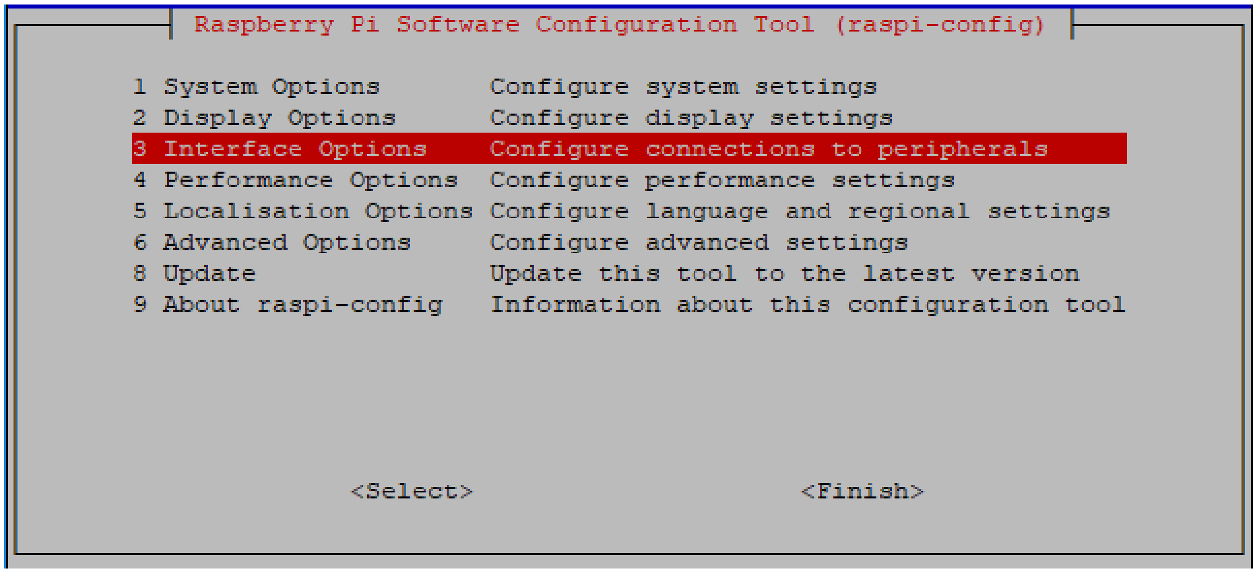
APPENDIX

5.1 I2C Configuration

Enable the I2C port of your Raspberry Pi (If you have enabled it, skip this; if you do not know whether you have done that or not, please continue).

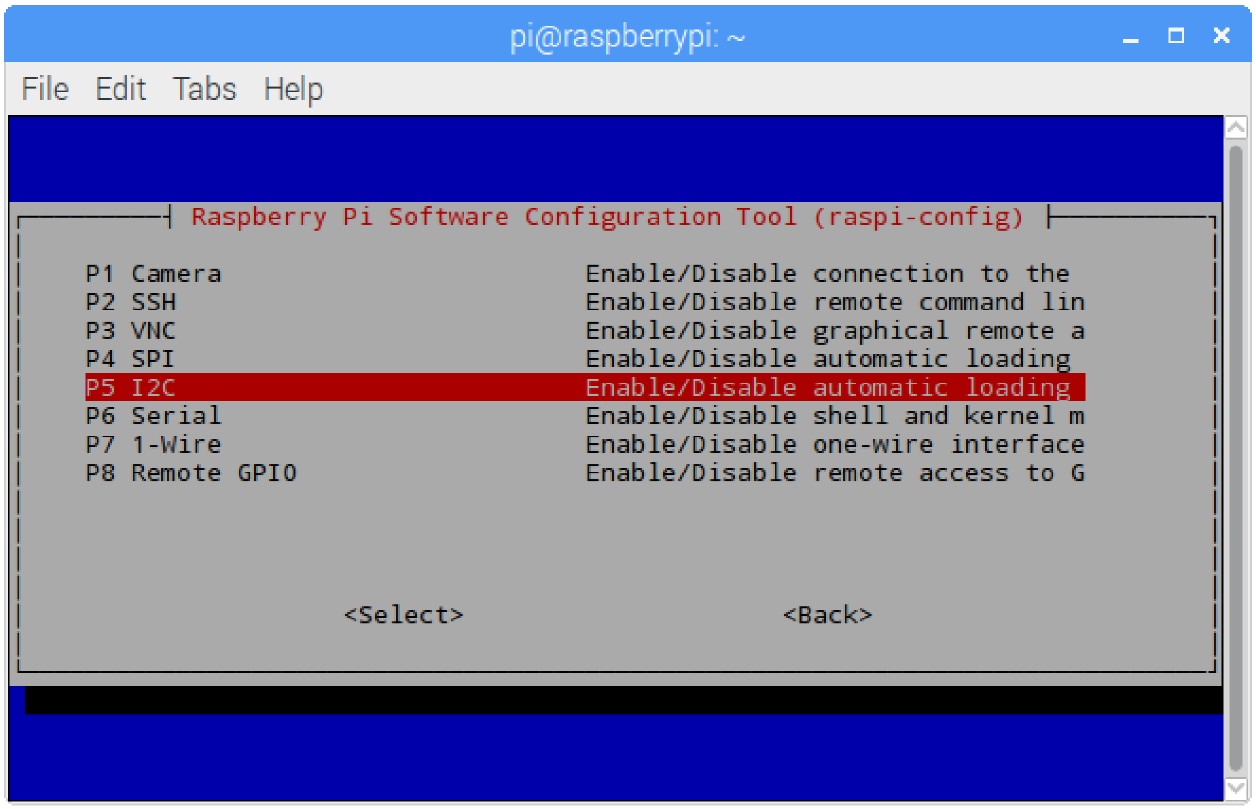
```
sudo raspi-config
```

3 Interfacing options



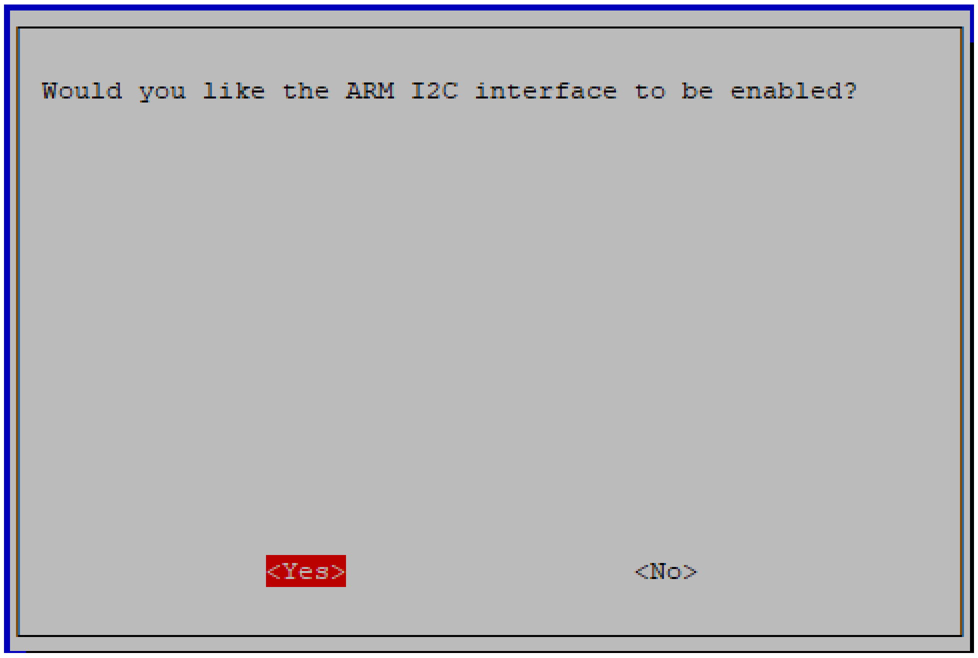
```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

              <Select>                     <Finish>
```

P5 I2C



```
┌──────────┤ Raspberry Pi Software Configuration Tool (raspi-config) ├──────────┐
│                                                                               │
│    P1 Camera                        Enable/Disable connection to the          │
│    P2 SSH                           Enable/Disable remote command lin         │
│    P3 VNC                           Enable/Disable graphical remote a         │
│    P4 SPI                           Enable/Disable automatic loading          │
│    P5 I2C                           Enable/Disable automatic loading          │
│    P6 Serial                        Enable/Disable shell and kernel m         │
│    P7 1-Wire                        Enable/Disable one-wire interface         │
│    P8 Remote GPIO                   Enable/Disable remote access to G         │
│                                                                               │
│                                                                               │
│                     <Select>                     <Back>                       │
│                                                                               │
└───────────────────────────────────────────────────────────────────────────────┘
```

<Yes>, then <Ok> -> <Finish>.



```
 Would you like the ARM I2C interface to be enabled?




              <Yes>                      <No>
```

5.2 Remote Desktop

There are two ways to control the desktop of the Raspberry Pi remotely:

VNC and **XRDP**, you can use any of them.

5.2.1 VNC

You can use the function of remote desktop through VNC.

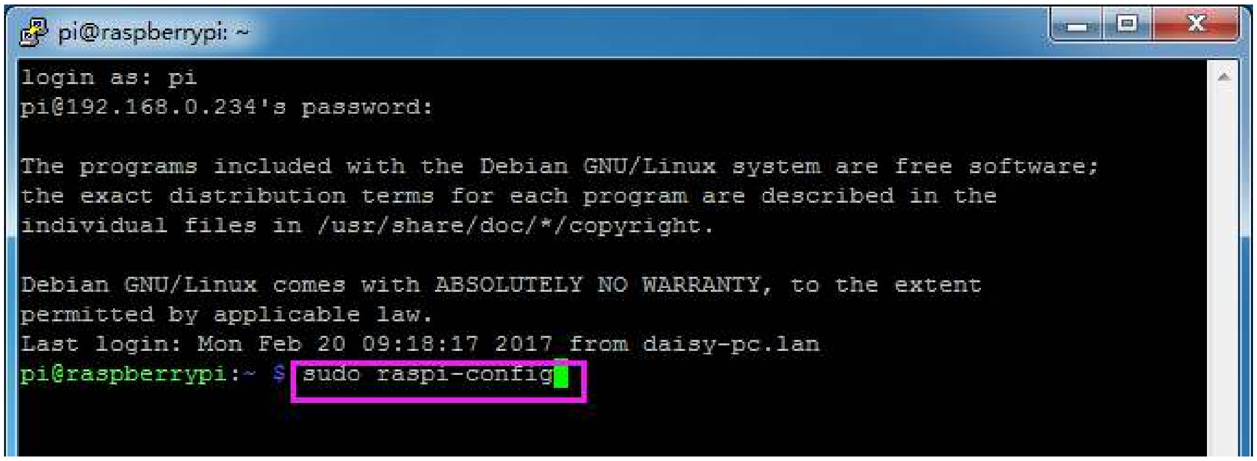
Enable VNC service

The VNC service has been installed in the system. By default, VNC is disabled. You need to enable it in config.

Step 1

Input the following command:

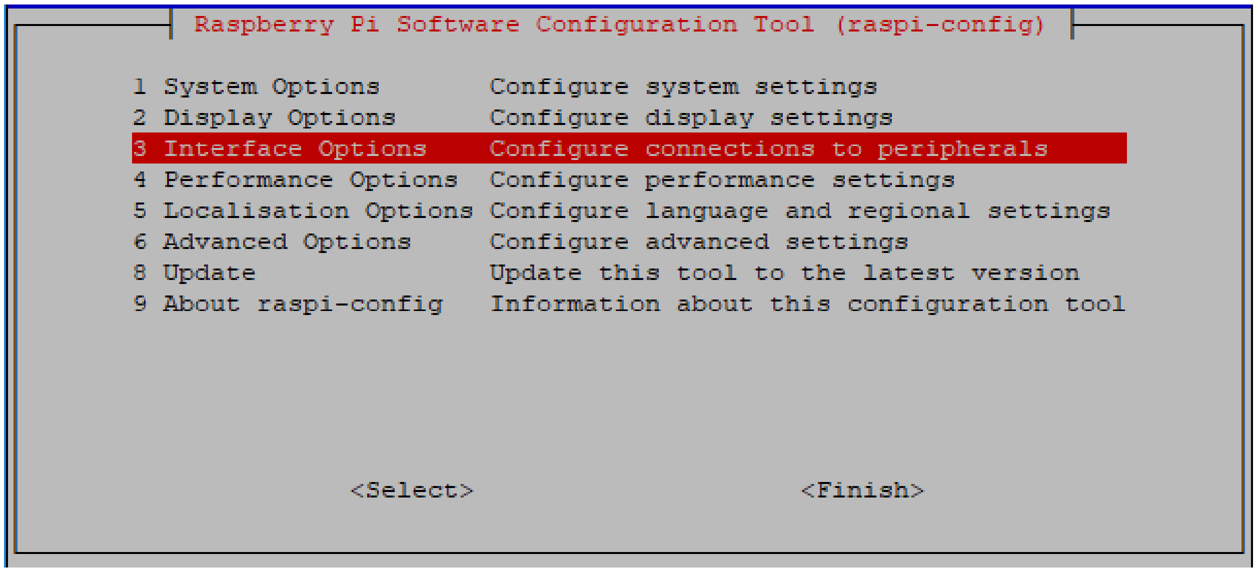
```
sudo raspi-config
```



```
login as: pi
pi@192.168.0.234's password:

The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
Last login: Mon Feb 20 09:18:17 2017 from daisy-pc.lan
pi@raspberrypi:~ $ sudo raspi-config
```

Step 2

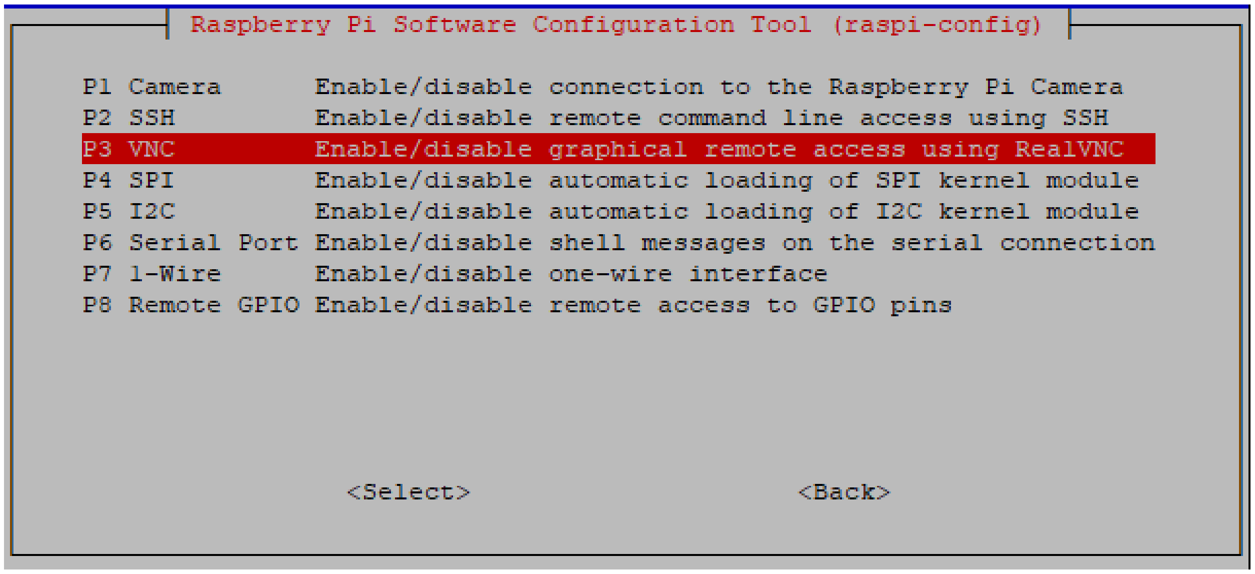
Choose **3 Interfacing Options** by press the down arrow key on your keyboard, then press the **Enter** key.



```
Raspberry Pi Software Configuration Tool (raspi-config)

1 System Options       Configure system settings
2 Display Options      Configure display settings
3 Interface Options    Configure connections to peripherals
4 Performance Options  Configure performance settings
5 Localisation Options Configure language and regional settings
6 Advanced Options     Configure advanced settings
8 Update               Update this tool to the latest version
9 About raspi-config   Information about this configuration tool

              <Select>                     <Finish>
```

Step 3

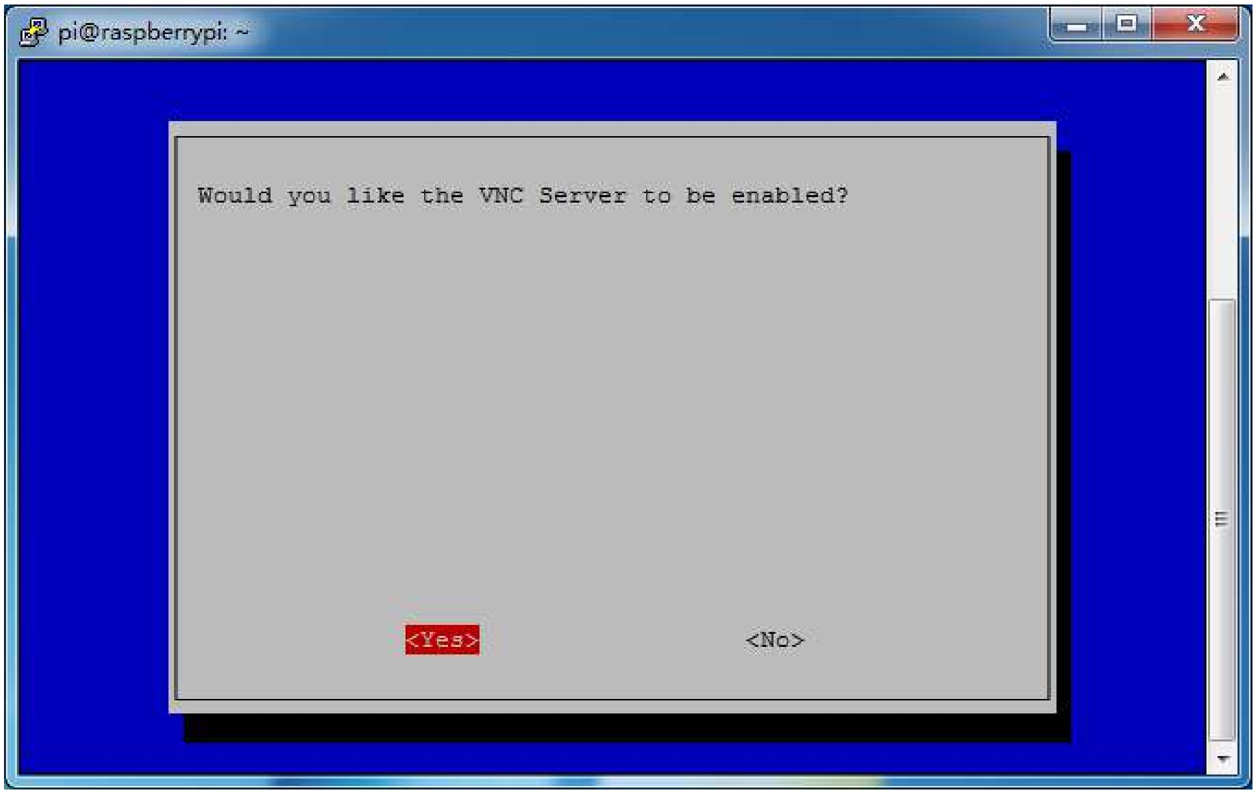
P3 VNC



```
Raspberry Pi Software Configuration Tool (raspi-config)

P1 Camera      Enable/disable connection to the Raspberry Pi Camera
P2 SSH         Enable/disable remote command line access using SSH
P3 VNC         Enable/disable graphical remote access using RealVNC
P4 SPI         Enable/disable automatic loading of SPI kernel module
P5 I2C         Enable/disable automatic loading of I2C kernel module
P6 Serial Port Enable/disable shell messages on the serial connection
P7 1-Wire      Enable/disable one-wire interface
P8 Remote GPIO Enable/disable remote access to GPIO pins

             <Select>                    <Back>
```

Step 4

Select **Yes -> OK -> Finish** to exit the configuration.



```
pi@raspberrypi: ~

Would you like the VNC Server to be enabled?

          <Yes>                 <No>
```

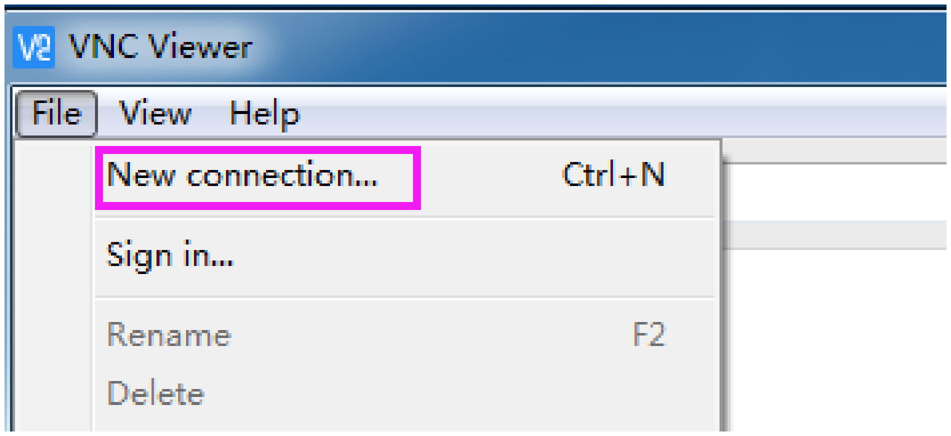
Login to VNC

Step 1

You need to download and install the VNC Viewer on personal computer. After the installation is done, open it.

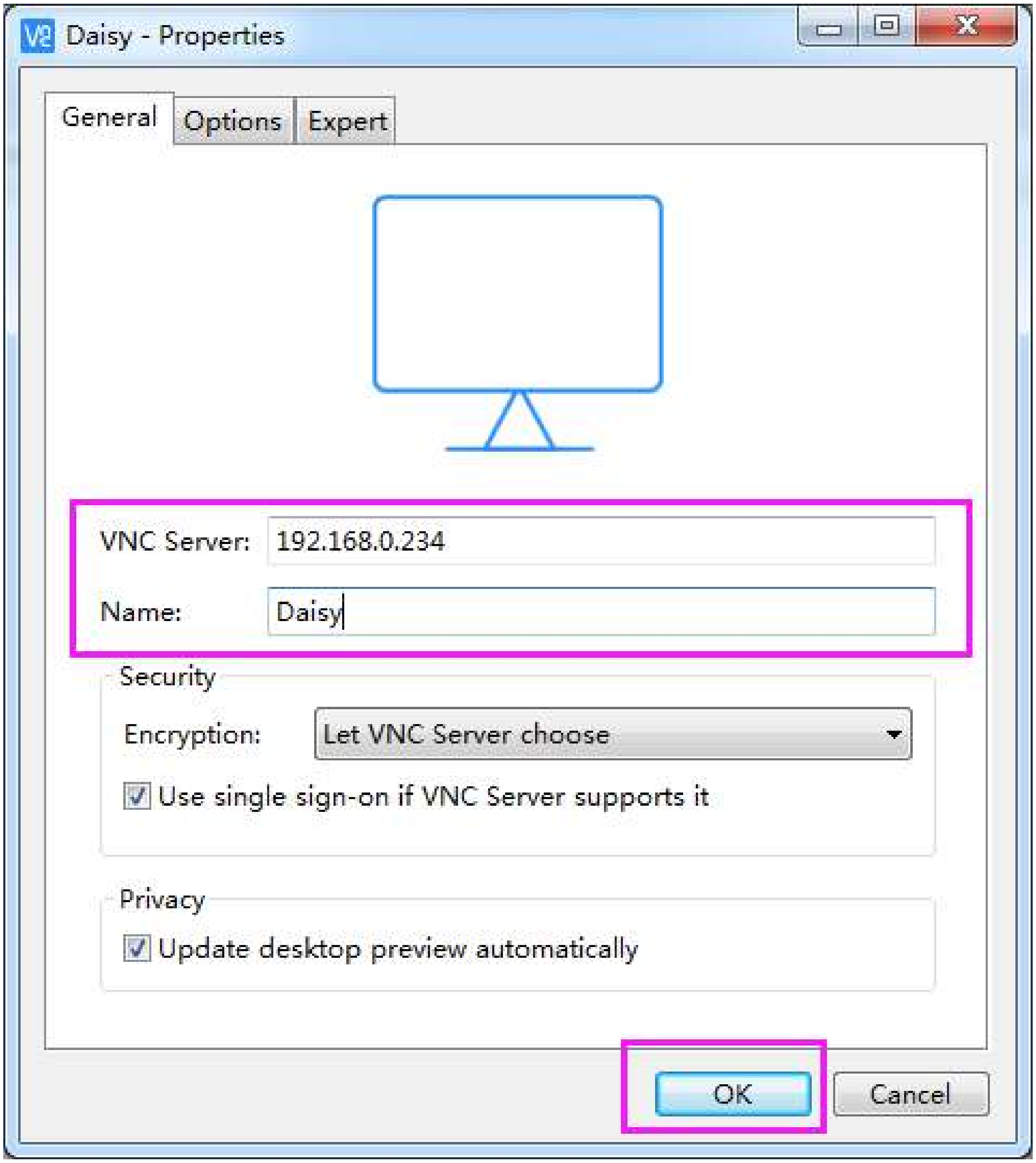
Step 2

Then select “**New connection**”.



Step 3

Input IP address of Raspberry Pi and any **Name**.



VNC Server: 192.168.0.234

Name: Daisy

Security

Encryption: Let VNC Server choose

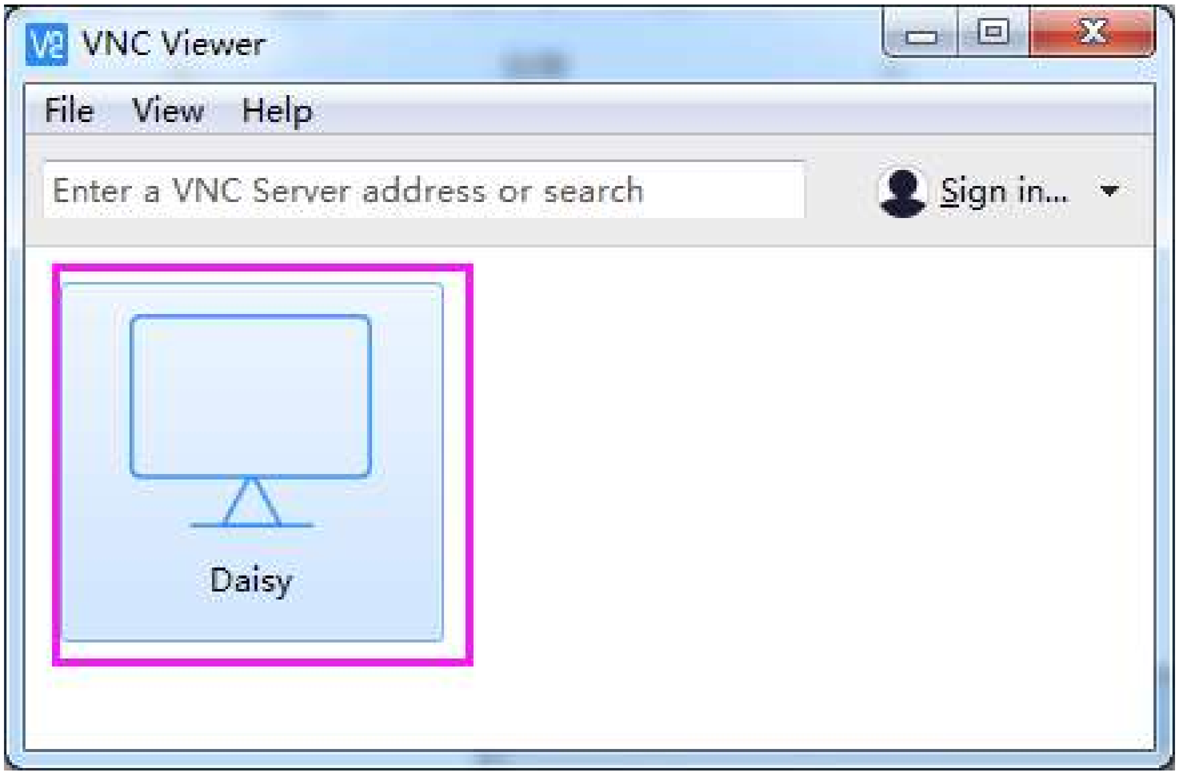
☑ Use single sign-on if VNC Server supports it

Privacy

☑ Update desktop preview automatically

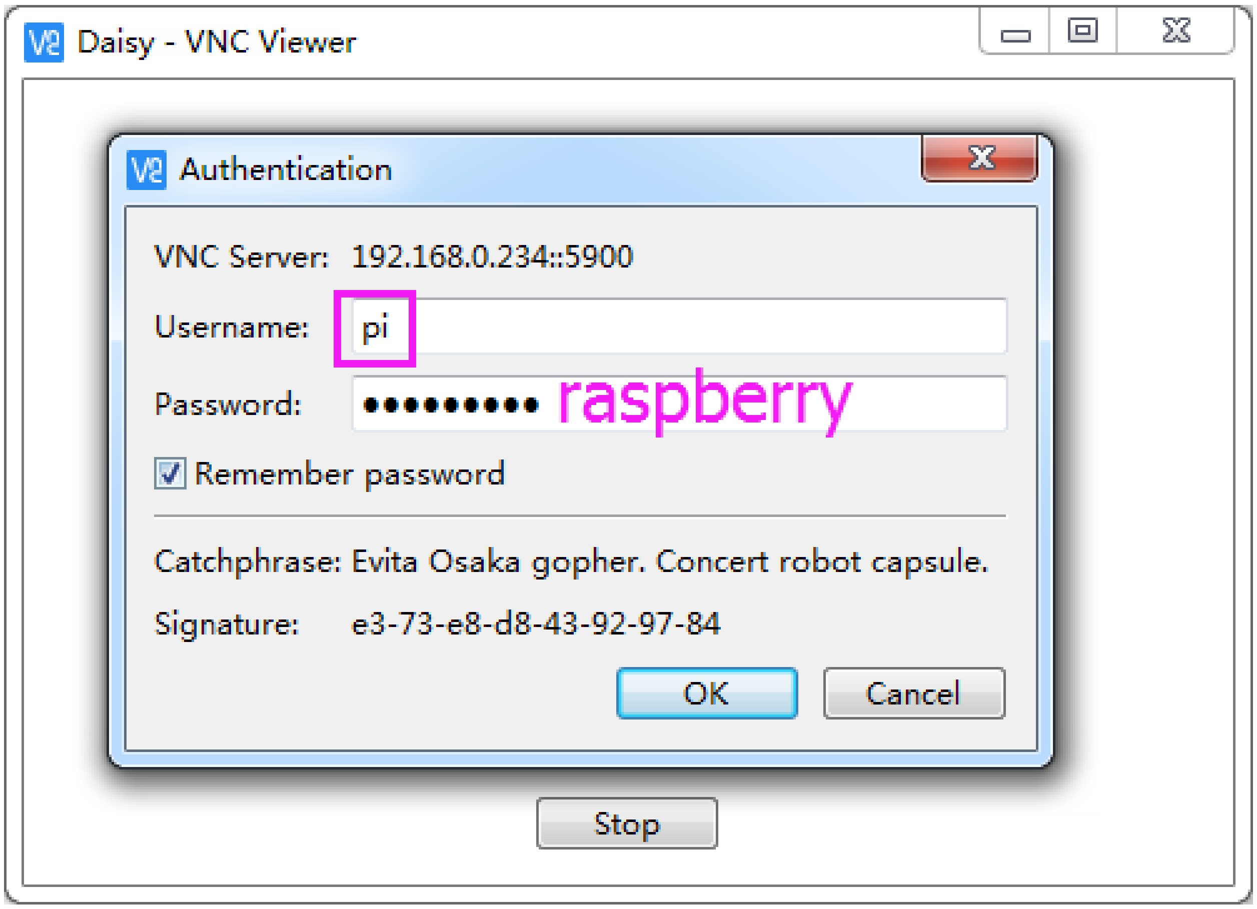
Step 4

Double click the **connection** just created:



Step 5

Enter Username (**pi**) and Password (**raspberry** by default).


Daisy - VNC Viewer

Authentication

VNC Server: 192.168.0.234::5900

Username: pi

Password: •••••••••• raspberry

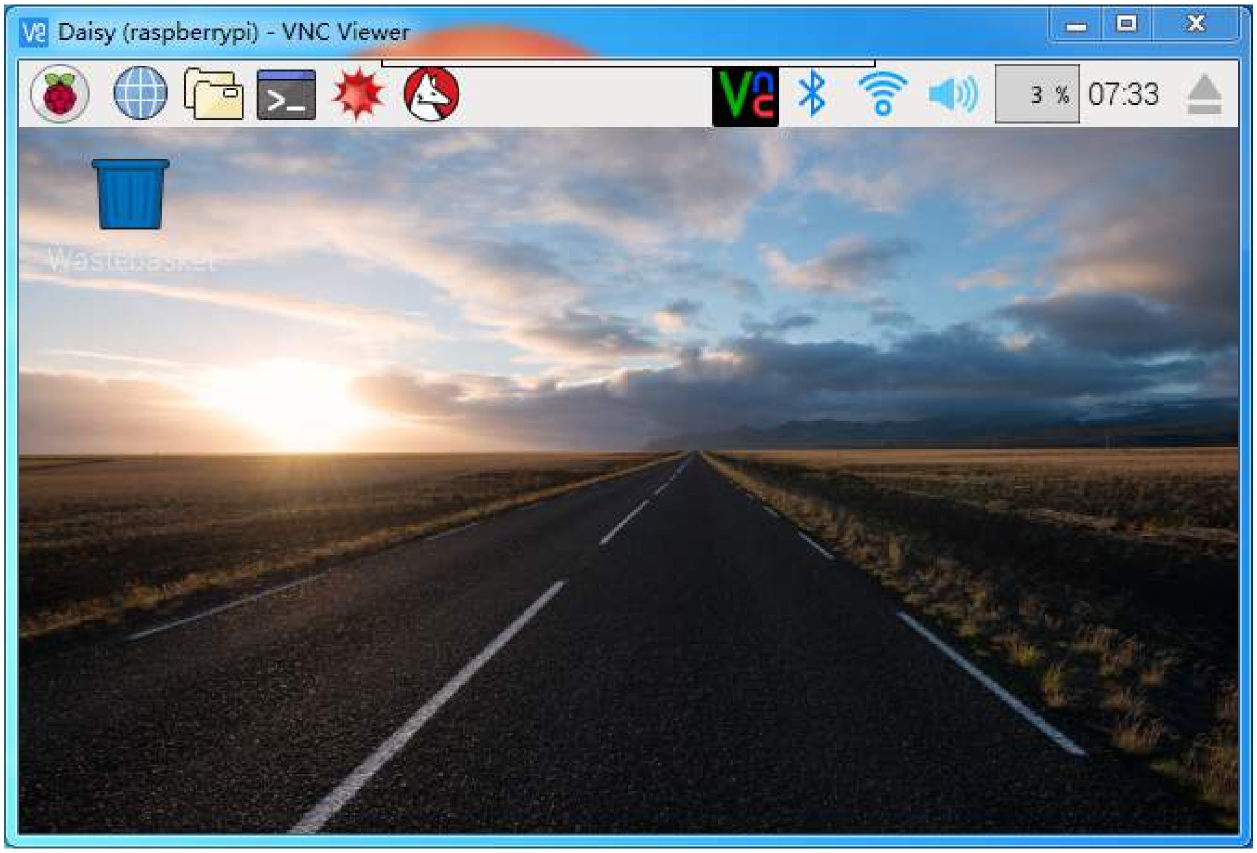
☑ Remember password

Catchphrase: Evita Osaka gopher. Concert robot capsule.

Signature: e3-73-e8-d8-43-92-97-84

Step 6

Now you can see the desktop of the Raspberry Pi:



That's the end of the VNC part.

5.2.2 XRDP

Another method of remote desktop is XRDP, it provides a graphical login to remote machines using RDP (Microsoft Remote Desktop Protocol).

Install XRDP

Step 1

Login to Raspberry Pi by using SSH.

Step 2

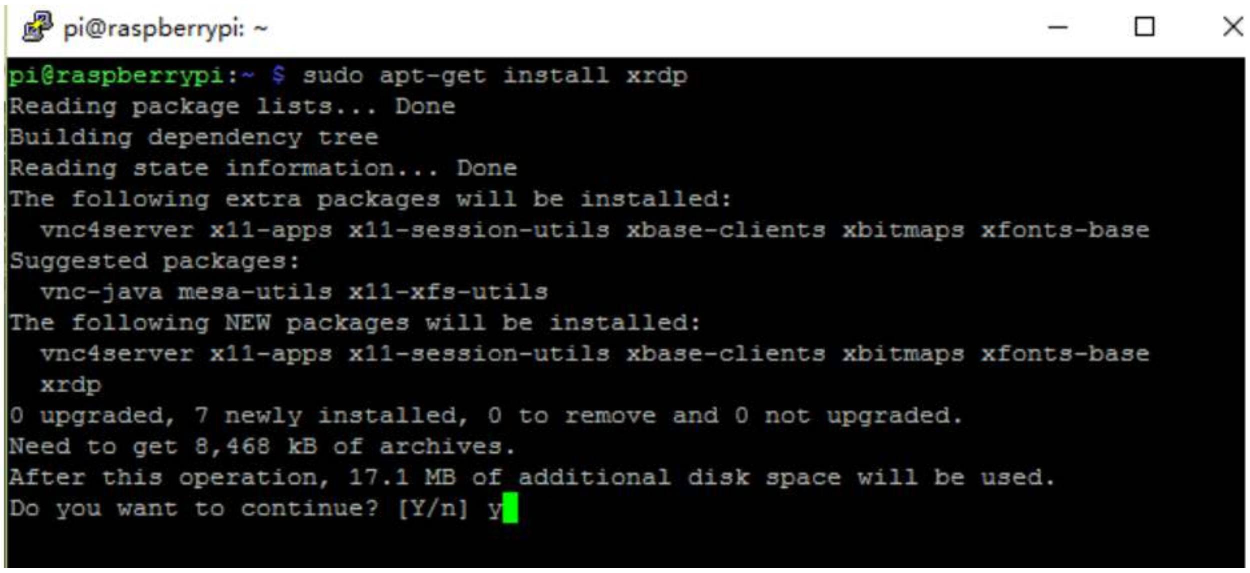
Input the following instructions to install XRDP.

```
sudo apt-get update
sudo apt-get install xrdp
```

Step 3

Later, the installation starts.

Enter "Y", press key "Enter" to confirm.


pi@raspberrypi: ~

```
pi@raspberrypi:~ $ sudo apt-get install xrdp
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following extra packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
Suggested packages:
  vnc-java mesa-utils x11-xfs-utils
The following NEW packages will be installed:
  vnc4server x11-apps x11-session-utils xbase-clients xbitmaps xfonts-base
  xrdp
0 upgraded, 7 newly installed, 0 to remove and 0 not upgraded.
Need to get 8,468 kB of archives.
After this operation, 17.1 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
```

Step 4

Finished the installation, you should login to your Raspberry Pi by using Windows remote desktop applications.

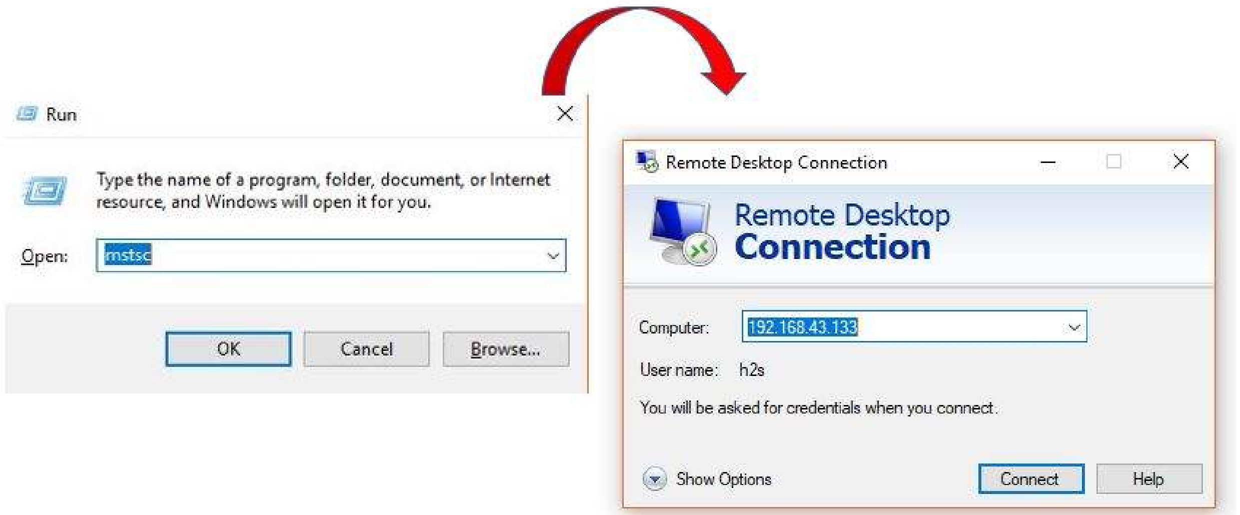
Login to XRDP

Step 1

If you are a Windows user, you can use the Remote Desktop feature that comes with Windows. If you are a Mac user, you can download and use Microsoft Remote Desktop from the APP Store, and there is not much difference between the two. The next example is Windows remote desktop.

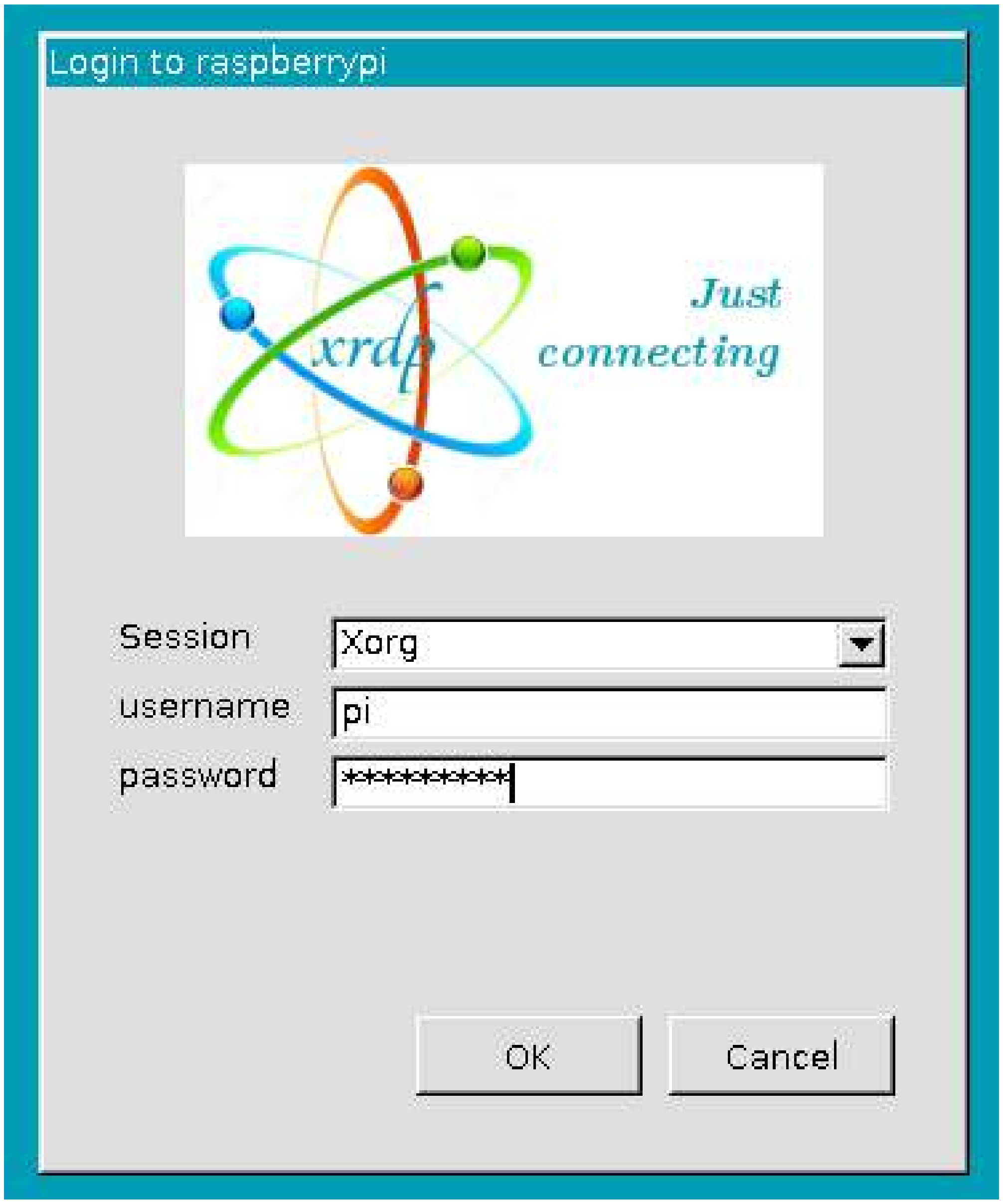
Step 2

Type in "**mstsc**" in Run (WIN+R) to open the Remote Desktop Connection, and input the IP address of Raspberry Pi, then click on "Connect".


Run
Type the name of a program, folder, document, or Internet resource, and Windows will open it for you.
Open: mstsc
OK
Cancel
Browse...
Remote Desktop Connection
Remote Desktop Connection
Computer: 192.168.43.133
User name: h2s
You will be asked for credentials when you connect.
Show Options
Connect
Help

Step 3

Then the xrdp login page pops out. Please type in your username and password. After that, please click "OK". At the first time you log in, your username is "pi" and the password is "raspberry".



Session Xorg
username pi
password **********

Step 4

Here, you successfully login to RPi by using the remote desktop.

5.3 About the Battery

Applicable Parameters

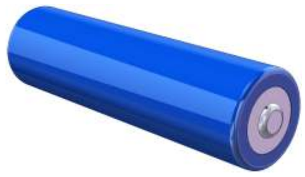
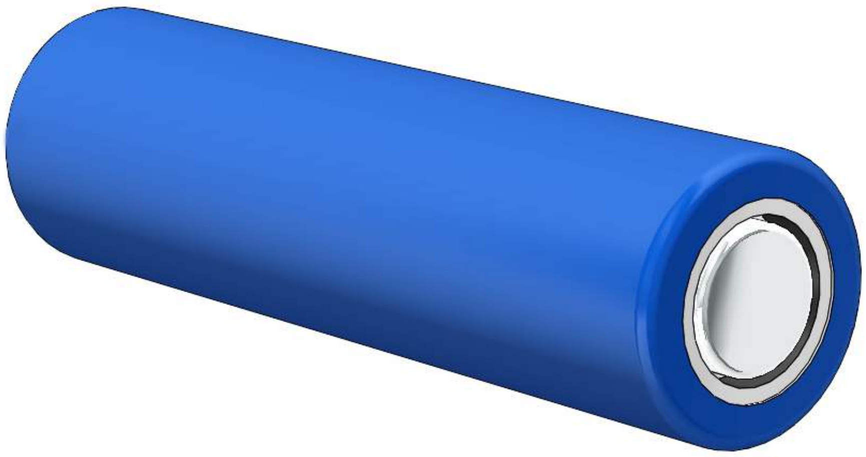
- 3.7V
- 18650
- Rechargeable
- Li-ion Battery
- Button Top
- No Protective Board

Note:

- Robot HAT cannot charge the battery, so you need to buy a battery charger.
- When the two power indicators on the Robot HAT are off, it means the power is too low and the batteries need to be charged.

Button Top vs Flat Top?

Please choose battery with button top to ensure a good connection between the battery and the battery holder.

Button Top	Flat Top
	

No protective board?

You are recommend to use 18650 batteries without a protective board. Otherwise, the robot may be cut power and stop running because of the overcurrent protection of the protective board.

Battery capacity?

In order to keep the robot working for a long time, use large-capacity batteries as much as possible. It is recommended to purchase batteries with a capacity of 3000mAh and above.

CHAPTER

SIX

THANK YOU

Thanks to the evaluators who evaluated our products, the veterans who provided suggestions for the tutorial, and the users who have been following and supporting us. Your valuable suggestions to us are our motivation to provide better products!

Particular Thanks

- Len Davisson
- Kalen Daniel
- Juan Delacosta

Now, could you spare a little time to fill out this questionnaire?

Note: After submitting the questionnaire, please go back to the top to view the results.

CHAPTER

SEVEN

COPYRIGHT NOTICE