

Download example

Open a terminal and run the following command to download the examples and unzip them.

```
wget https://www.waveshare.com/w/upload/0/00/PicoGo_Code.zip
unzip PicoGo_Code.zip
```

Test Motors

- Open the motor.py in Thonny IDE, and run it
- The PicoGo will move forward, then backward, turn left and turn right after running the codes.

Note: You need to turn the power switch to ON, and make sure that the PicoGo has enough place to move.

- Codes:

```
. . .

    M = PicoGo()      #Instantiate the PicoGo class, which has defined
the functions of motions (forward, backward, left, right stop and the
intialiation.

    M.forward(50)     #Move the motor forward in half speed (0- 100)
    utime.sleep(0.5) #Set sleep time to let the Picogo keep moving
for 0.5s
    M.backward(50)    #Move the motor backward in half speed (0- 100)
    utime.sleep(0.5)
    M.left(30)        #Let the motor turn left in speed 30 (0-100)
    utime.sleep(0.5)
    M.right(30)       #Let the motor turn right in speed 30 (0-100)
    utime.sleep(0.5)
    M.stop()          #Stop the motor
```

Infrared Remote Control

- Open the IRremote.py in ThonnyIDE and run it.

- Press the Infrared controller to control the PicoGo
- 2 , 8 , 4 , 6 , 5 are used for forwarding, backward, turn left, turn right and stop. You can press the - or + keys to adjust the speed and press EQ to restore the setting.
- Different infrared remote controllers may have different key codes, if you use other controllers, you may need to modify the codes.

Note: If you need to make the PicoGo run without cable, you need to rename the IRremote.py as main.py and save it to Raspberry Pi Pico. The codes also need to call the Motor.py, you need to save it to Raspberry Pi Pico as well.

- Codes:

```

. . .
while True:
    key = getkey()          #use the getkey function to read the
    singal of Infrated controller.
    if(key != None):
        n = 0
        if key == 0x18:
            M.forward(speed) #If the value of controlled is 0x18,
            move the motor forward
            print("forward")
        if key == 0x08:
            M.left(20)       #If the value of controlled is 0x08,
            turn the Picogo toward left
            print("left")
        if key == 0x1c:
            M.stop()        #If the value of controlled is 0x1C,
            stop the motor
            print("stop")
        if key == 0x5a:
            M.right(20)     #If the value of controlled is 0x5a,
            turn the Picogo toward right
            print("right")
        if key == 0x52:

```

```

        M.backward(speed) #If the value of controlled is 0x52,
move the motor backward

        print("backward")

    if key == 0x09:
        speed = 50 #If the value of controlled is 0x09,
set the speed to 50

        print(speed)

    if key == 0x15:
        if(speed + 10 < 101):
            speed += 10 #If the value of controlled is 0x15,
speed up the motor in 10, max is 100

            print(speed)

    if key == 0x07:
        if(speed - 10 > -1):
            speed -= 10 #If the value of controlled is 0x07,
slow down the motor in 10, min is 0

            print(speed)

    else:
        n += 1

        if n > 800:
            n = 0

            M.stop() #If the controller doesn't be operated
for a certain time, stop the motors.

```

Infrared Obstacle Avoidance

- Open the Infrared-Obstacle-Avoidance.py in Thonny IDE, rename it as main.py, and save it to Pico. Disconnect the USB cable and run it.
- When there is no obstacle in front of the car, the green LED light in front of the car will be off. When the car meets an obstacle, the green LED light in front will be on.
- If the LED light is not bright or keeps brightening, you can adjust two potentiometers on the bottom of the PicoGo, so that the LED is just out of state. The detection distance is the farthest.
- Procedure phenomenon is no obstacle when the car straight, encountered obstacles when the car to turn right.

- Code:

```
...
while True:

    DR_status = DSR.value()    # Read the value of the right Infrared
    sensor

    DL_status = DSL.value()    # Read the value of the left Infrared
    sensor

    if((DL_status == 0) and (DR_status == 0)):        #If the value of
    both the sensor are 0, there is obstacle in the front, turn left

        M.left(10)

    elif((DL_status == 0) and (DR_status == 1)):        #If the DL value
    is 0 and the DR value is 1, there is obstacle in the left side, turn
    right

        M.right(10)

    elif((DL_status == 1) and (DR_status == 0)):        #If the DL value
    is 1 and the DR value is 0, there is obstacle in the right side, turn
    left.

        M.left(10)

    else:

        M.forward(20)                                #else, there is
    no obstacle, keep moving forward.

    utime.sleep_ms(10)
```

Ultrasonic Ranging

- Open the Ultrasonic_Ranging.py in Thonny IDE, the detected distance will be shown on the shell.
- Because the ultrasonic wave will be reflected, the front side of the obstacle plane is not in front of the ultrasonic wave but with the ultrasonic wave formed an Angle of the measured distance may be inaccurate.

```
[ Ultrasonic_Ranging.py ] X
1 import utime
2 from machine import Pin
3
4
5 Echo = Pin(15, Pin.IN)
6 Trig = Pin(14, Pin.OUT)
7 Trig.value(0)
8 Echo.value(0)
9
10 def dist():
11     Trig.value(1)
12     utime.sleep_us(10)
13     Trig.value(0)
14     while(Echo.value() == 0):
15         pass
16     ts=utime.ticks_us()
17     while(Echo.value() == 1):
18         pass
19     te=utime.ticks_us()
-- ..

Shell X
Distance: 39.02 cm
Distance: 38.56 cm
Distance: 38.71 cm
Distance: 38.28 cm
Distance: 38.86 cm
Distance: 38.61 cm
Distance: 38.17 cm
Distance: 38.05 cm
Distance: 38.20 cm

MicroPython (Raspberry Pi Pico)
```

- Codes:

```
...
def dist():                                #Function to read the sensor data and
caculate the distance

    Trig.value(1)
    utime.sleep_us(10)
    Trig.value(0)
    while(Echo.value() == 0):
        pass
    ts=utime.ticks_us()
    while(Echo.value() == 1):
        pass
    te=utime.ticks_us()
    distance=((te-ts)*0.034)/2
    return distance
```

```

while True:
    print("Distance:%6.2f cm" % dist()) # Print the Distance data to
the console.
    utime.sleep(1)

```

Ultrasonic Obstacle Avoidance

- Open the Ultrasonic-Obstacle-Avoidance.py in Thonny IDE, rename it as main.py, and save it to Raspberry Pi Pico.
- Run the program after disconnecting the USB cable. Go straight when there is no obstacle, and turn right when there is an obstacle.
- Codes ;

```

...
def dist(): #Function for reading data of Ultrasonic sensor
and calculate the distance
    Trig.value(1)
    utime.sleep_us(10)
    Trig.value(0)
    while(Echo.value() == 0):
        pass
    ts=utime.ticks_us()
    while(Echo.value() == 1):
        pass
    te=utime.ticks_us()
    distance=((te-ts)*0.034)/2
    return distance

while True:
    D = dist() #read the distance data
    if(D <= 20): #Turn right if there the distance of obstace is
shorteer than 20
        M.right(20)
        #Ab.left()

```

```

else:          #else keep moving forward
    M.forward(20)

utime.sleep_ms(20)

```

Ultrasonic Infrared Obstacle Avoidance

- Open the Ultrasonic-Infrared-Obstacle-Avoidance.py in Thonny, rename it as main.py, and save it to Raspberry Pi Pico.
- Run the program after disconnecting the USB cable. Go straight when there is no obstacle, and turn right when there is an obstacle. The combination of ultrasonic and infrared has a better obstacle avoidance effect and a higher success rate.
- Codes:

```

...
while True:
    D = dist()          #read the distance data of Ultrasonic
    sensor

    DR_status = DSR.value()    #read the distance data of right
    Infrared sensor

    DL_status = DSL.value()    #read the distance data of left
    Infrared sensor

    if((D <= 20) or (DL_status == 0) or (DR_status == 0)):    #If there
    is obstace detected, turn right

        M.right(20)

        #Ab.left()

    else:

        M.forward(40)          #else, keep moving forward.

    utime.sleep_ms(20)

```

RGB LED

- Open the WS2812.py in Thonny and run it.

- Four colored LED lights at the bottom of the car will show red, yellow, green, clear color, blue, purple, white, and then show the color light effect.
- Codes:

```

...
if __name__=='__main__':
    strip = NeoPixel()      #instantiate the NeoPixel class which is
used to initial LED controller, and set LED

    print("fills")

    for color in strip.COLORS:      #Set the LED to show color in loop
(BLACK, RED, GREEN, CYAN, BLUE, PURPLE, WHITE)

        strip.pixels_fill(color)    #Set the LED color

        strip.pixels_show()        # Turn on the RGB LED

        time.sleep(0.5)

    print("chases")

    for color in strip.COLORS:      #Turn on the RGB LED one by one

        strip.color_chase(color, 0.05)

    print("rainbow")              #Change the color in loop like rainbow

    while(1):

        strip.rainbow_cycle(0.02)

```

1.14inch LCD

- Open the ST7789.py in Thonny IDE and run it.
- After the program runs normally, LCD will display the string.
- Codes:

```

...
if __name__=='__main__':
    lcd = ST7789()      #instantiate the function of LCD controlling

    lcd.fill(0xFFFF)   #Set the backlight color

    lcd.show()         #display

    lcd.text("Raspberry Pi Pico",10,5,0xFF00)      #Draw text on the
buffer with coordination 10(x), 5(y)

```



```

    lcd.text("PicoGo",10,15)                #Draw text on the
buffer
    lcd.text("Waveshare.com",10,25,0x07E0)  #Draw text on the
buffer
    lcd.show()                               #Display the content
from buffer

```

Battery Voltage Detection

- Open the Battery_Voltage.py in Thonny IDE and run it.
- LCD will display chip temperature, battery voltage, and power percentage. The percentage of electric quantity is obtained by simple linear conversion of voltage. The actual battery voltage and electric quantity are not linear, so there will be some error in this percentage.
- Codes:

```

...
while (1):
    utime.sleep(1)

    reading = temp.read_u16() * 3.3 / (65535)          #Read
the temperature data from register
    temperature = 27 - (reading - 0.706)/0.001721
#calculate the temperature data

    v = bat.read_u16()*3.3/65535 * 2                  #Read
the voltage data from register

    p = (v - 3) * 100 / 1.2
#Calculat the battery data

    if(p < 0):p=0
    if(p > 100):p=100

    lcd.fill_rect(145,50,65,40,0xF232)
#Display the temperature, voltage and battery data to LCD,

    lcd.text("temperature : {:.2f}
C".format(temperature),30,50,0xFFFF) #use the ST7789.py as libraries

    lcd.text("Voltage      : {:.2f} V".format(v),30,65,0xFFFF)

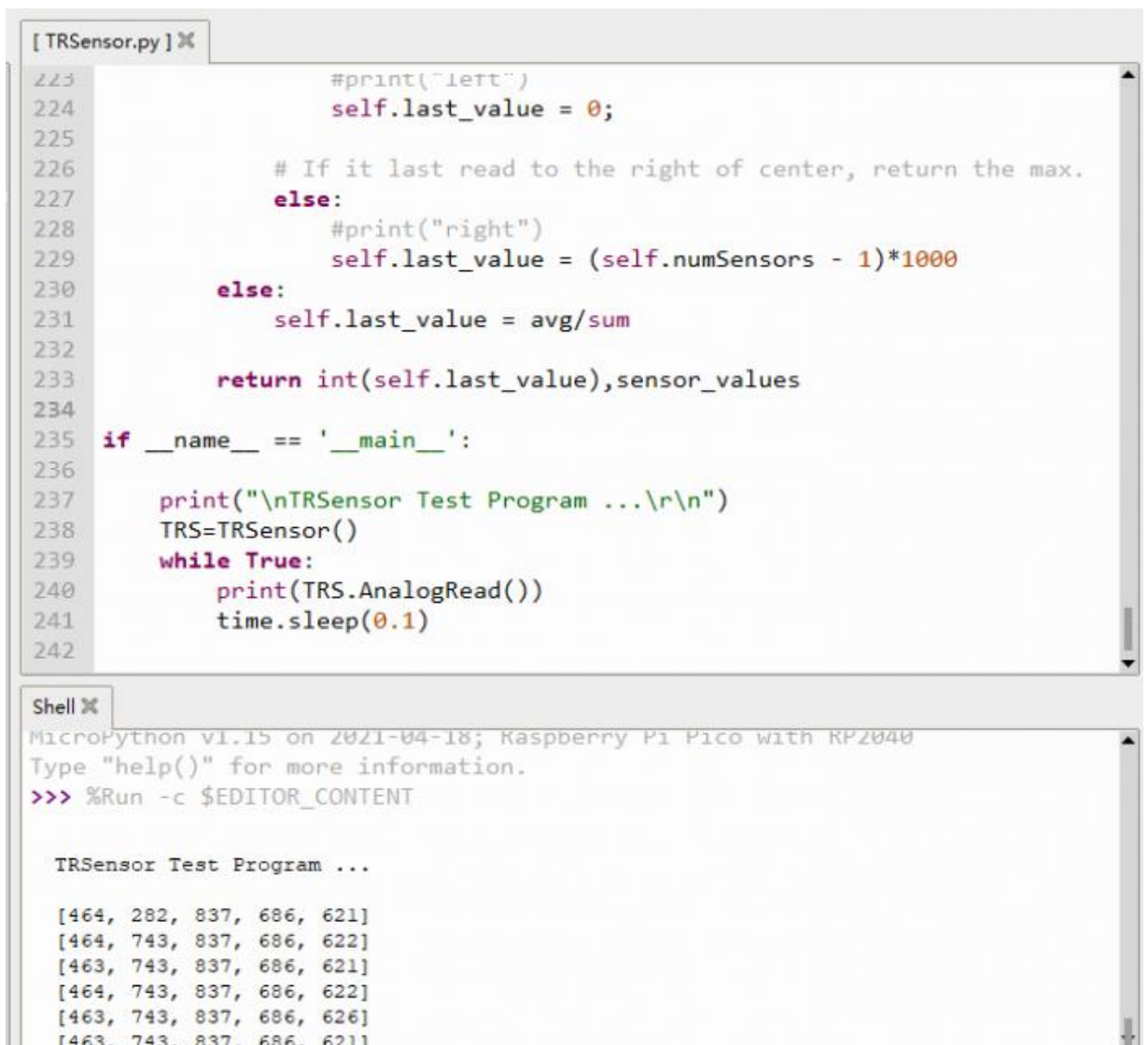
    lcd.text("percent     : {:.1f} %".format(p),30,80,0xFFFF)

```

```
lcd.show()
```

Tracking Sensor Test

- Open the TRsensor.py in Thonny IDE and run it.
- The shell interface will display the values of the five tracking sensors. The data range is 600~900 when the PicoGo is put on the white paper, and the data range is 0~50 when the PicoGo is put in the air.



```
[ TRSensor.py ] X
223         #print("left")
224         self.last_value = 0;
225
226         # If it last read to the right of center, return the max.
227         else:
228             #print("right")
229             self.last_value = (self.numSensors - 1)*1000
230     else:
231         self.last_value = avg/sum
232
233     return int(self.last_value),sensor_values
234
235 if __name__ == '__main__':
236
237     print("\nTRSensor Test Program ...\r\n")
238     TRS=TRSensor()
239     while True:
240         print(TRS.AnalogRead())
241         time.sleep(0.1)
242
Shell X
MicroPython v1.15 on 2021-04-18; Raspberry Pi Pico with RP2040
Type "help()" for more information.
>>> %Run -c $EDITOR_CONTENT

TRSensor Test Program ...

[464, 282, 837, 686, 621]
[464, 743, 837, 686, 622]
[463, 743, 837, 686, 621]
[464, 743, 837, 686, 622]
[463, 743, 837, 686, 626]
[463, 743, 837, 686, 621]
```

- Codes:

```
...
if __name__ == '__main__':

    print("\nTRSensor Test Program ...\r\n")
```

```

    TRS=TRSensor()    #Instantiate the TRSensor class, whic features
    fucntions ofr read analog data and calibrates...

    while True:

        print(TRS.AnalogRead())    #Print the analog data red
        time.sleep(0.1)

```

Infrared Tracking

- Open the Line-Tracking.py file in Thonny IDE, rename it as main.py, and save it to Raspberry Pi Pico.
- Tracking sensor can detect black line with white background (or white line with a black background, need to modify program).
- The tracking board can be made by sticking black tape in the white KT board. The width of the black track is 15cm. If the background color is too dark, the tracking effect will be affected.
- After disconnecting the USB cable, running the program, and putting the car in the black line, the car will rotate left and right, this is the car calibration stage. If the calibration phase operation error will directly affect the tracking effect.
- Codes:

```

...
while True:

    #print(TRS.readCalibrated())

    #print(TRS.readLine())

    position,Sensors = TRS.readLine()    #Use the TRsensor.py as
    libraries function, read the data of Infrared tracking sensor

    #time.sleep(0.1)

    if((Sensors[0] + Sensors[1] + Sensors[2]+ Sensors[3]+ Sensors[4]) >
    4000):    #Check the data of sensors

        M.setMotor(0,0)

    else:

        # The "proportional" term should be 0 when we are on the line.
        proportional = position - 2000

```

```

    # Compute the derivative (change) and integral (sum) of the
    position.
    derivative = proportional - last_proportional
    integral += proportional

    # Remember the last position.
    last_proportional = proportional

    ...

    // Compute the difference between the two motor power settings,
    // m1 - m2.  If this is a positive number the robot will turn
    // to the right.  If it is a negative number, the robot will
    // turn to the left, and the magnitude of the number determines
    // the sharpness of the turn.  You can adjust the constants by
which
    // the proportional, integral, and derivative terms are
multiplied to
    // improve performance.
    ...
    power_difference = proportional/30 + derivative*2;

    if (power_difference > maximum):
        power_difference = maximum
    if (power_difference < - maximum):
        power_difference = - maximum

    if (power_difference < 0):
        M.setMotor(maximum + power_difference, maximum)
    else:
        M.setMotor(maximum, maximum - power_difference)

```

Infrared Tracking-Integrated

- Open a Line-Tracking2.py in Thonny IDE, rename it as main.py, and save it to Raspberry Pi Pico.

- After disconnecting the USB cable, running the program, and putting the car in the black line, the car will rotate left and right to calibrate. After calibration, the black line will be run.
- When there is an obstacle in front of the car, the car will stop and the buzzer will sound. After the obstacle is cleared, the car will continue to run. Pick up the car and the motor will stop.
- During the calibration stage of the car, the four RGBS display red, green, and blue respectively. Change. The RGB LED will display the color light effect when tracking is running.
- Codes:

```

...
#This following function combines the Infrared sensor to detect
obstacles while following the line.
while True:
    position,Sensors = TRS.readLine()
    DR_status = DSR.value()
    DL_status = DSL.value()

    if((Sensors[0] + Sensors[1] + Sensors[2]+ Sensors[3]+ Sensors[4]) >
4000):
        Buzzer.value(0)
        M.setMotor(0,0)
    elif((DL_status == 0) or (DR_status == 0)):
        Buzzer.value(1)
        M.setMotor(0,0)
    else:
        Buzzer.value(0)
        # The "proportional" term should be 0 when we are on the line.
        proportional = position - 2000

        # Compute the derivative (change) and integral (sum) of the
position.
        derivative = proportional - last_proportional
        #integral += proportional

```

```

# Remember the last position.
last_proportional = proportional

'''
// Compute the difference between the two motor power settings,
// m1 - m2.  If this is a positive number the robot will turn
// to the right.  If it is a negative number, the robot will
// turn to the left and the magnitude of the number determines
// the sharpness of the turn.  You can adjust the constants by
which
// the proportional, integral, and derivative terms are
multiplied to
// improve performance.
'''
power_difference = proportional/30 + derivative*2;

if (power_difference > maximum):
    power_difference = maximum
if (power_difference < - maximum):
    power_difference = - maximum

if (power_difference < 0):
    M.setMotor(maximum + power_difference, maximum)
else:
    M.setMotor(maximum, maximum - power_difference)

for i in range(strip.num):
    strip.pixels_set(i, strip.wheel(((i * 256 // strip.num) + j) &
255))
    strip.pixels_show()
    j += 1
if(j > 256):
    j = 0

```

Ultrasonic Infrared Following

- Open the Ultrasonic-Infrared-follow.py in Thonny, rename it as main.py, and save it to Raspberry Pi Pico.
- Run the program after disconnecting the USB cable, place the object in the sensor of the car, and the car will automatically follow the object to move.
- The following distance of the car can be set, the default following distance is 5cm, the car will stop when it is 5cm away from the object, the car will continue to run when it is larger than 5cm and smaller than 7cm.
- Turn left and right by infrared.
- When the car is running, the RGB LED will display the color light effect.
- Codes:

```
...
#Combine Ultrasonic and infrareas sensor to follow lines and obstacing,
the LCD is used to display text

while True:

    D = dist()

    # print("Distance:%6.2f cm" % dist())

    # utime.sleep(1)

    DR_status = DSR.value()
    DL_status = DSL.value()

    if((utime.ticks_ms() - t) > 3000):
        t=utime.ticks_ms()
        reading = temp.read_u16() * 3.3 / (65535)
        temperature = 27 - (reading - 0.706)/0.001721
        v = bat.read_u16()*3.3/65535 * 2
        p = (v - 3) * 100 / 1.2
        if(p < 0):p=0
        if(p > 100):p=100

        lcd.fill_rect(145,50,50,40,0xF232)
```

```

        lcd.text("temperature : {:.2f}
C".format(temperature),30,50,0xFFFF)

        lcd.text("Voltage      : {:.2f} V".format(v),30,65,0xFFFF)
        lcd.text("percent      : {:.1f} %".format(p),30,80,0xFFFF)
        lcd.show()

print(D)
if(D<5):
    M.stop()
elif((DL_status == 0) and (DR_status == 1)):
    M.left(20)
elif((DL_status == 1) and (DR_status == 0)):
    M.right(20)
elif(((D>5) and( D<7)) or ((DL_status == 0) and (DR_status == 0))):
    M.forward(30)
else:
    M.stop()

utime.sleep_ms(20)

for i in range(strip.num):
    strip.pixels_set(i, strip.wheel(((i * 256 // strip.num) + j) &
255))
    strip.pixels_show()
    j += 1
if(j > 256):
    j = 0

```

Bluetooth Remote Control

- Open the bluetooth.py in the Thonny IDE, rename it as main.py, and save it to Raspberry Pi Pico
- Install [PicoGo APP](#) in your phone (only support Android)

- Start the APP, select Bluetooth control, and click "Search" in the upper right corner. After about a few seconds, the corresponding Bluetooth device will be displayed in the list normally.



- Select JDY-33-SPP. If you select "JDy-33-ble", the connection to the device will fail. Go to the next page and select remote Control



- Press the button to control the car, but also can control the buzzer sound, and RGB LED display different colors.
- Codes:

```

...
while True:
    s=uart.read()      #Use serial port to read the data from Bluetooth
    module
    if(s != None):
        try:
            j=ujson.loads(s)      #use ujson librariries to handle the
            bluetooth data
            #print(j)

            cmd=j.get("Forward")
            if cmd != None:
                if cmd == "Down":
                    M.forward(speed)
                    uart.write("{\"State\":\"Forward\"}")
                elif cmd == "Up":
                    M.stop()
                    uart.write("{\"State\":\"Stop\"}")

            cmd = j.get("Backward")
            if cmd != None:
                if cmd == "Down":
                    M.backward(speed)
                    uart.write("{\"State\":\"Backward\"}")
                elif cmd == "Up":
                    M.stop()
                    uart.write("{\"State\":\"Stop\"}")

            cmd = j.get("Left")
            if cmd != None:
                if cmd == "Down":
                    M.left(20)
                    uart.write("{\"State\":\"Left\"}")
                elif cmd == "Up":
                    M.stop()

```

```

        uart.write("{\"State\":\"Stop\"}")

cmd = j.get("Right")
if cmd != None:
    if cmd == "Down":
        M.right(20)
        uart.write("{\"State\":\"Right\"}")
    elif cmd == "Up":
        M.stop()
        uart.write("{\"State\":\"Stop\"}")

cmd = j.get("Low")
if cmd == "Down":
    uart.write("{\"State\":\"Low\"}")
    speed = 30

cmd = j.get("Medium")
if cmd == "Down":
    uart.write("{\"State\":\"Medium\"}")
    speed = 50

cmd = j.get("High")
if cmd == "Down":
    uart.write("{\"State\":\"High\"}")
    speed = 100

cmd = j.get("BZ")
if cmd != None:
    if cmd == "on":
        BUZ.value(1)
        uart.write("{\"BZ\":\"ON\"}")
        uart.write("{\"State\":\"BZ:ON\"}")
    elif cmd == "off":
        BUZ.value(0)

```

```

        uart.write("{\"BZ\": \"OFF\"}")
        uart.write("{\"State\": \"BZ: OFF\"}")

cmd = j.get("LED")
if cmd != None:
    if cmd == "on":
        led.value(1)
        uart.write("{\"LED\": \"ON\"}")
        uart.write("{\"State\": \"LED: ON\"}")
    elif cmd == "off":
        led.value(0)
        uart.write("{\"LED\": \"OFF\"}")
        uart.write("{\"State\": \"LED: OFF\"}")

cmd = j.get("RGB")
if cmd != None:
    rgb=tuple(eval(cmd))
    strip.pixels_set(0, rgb)
    strip.pixels_set(1, rgb)
    strip.pixels_set(2, rgb)
    strip.pixels_set(3, rgb)
    strip.pixels_show()
    uart.write("{\"State\": \"RGB: (" +cmd+ ")\"}")
except:
    print("err")

...

```