Contents

IDE

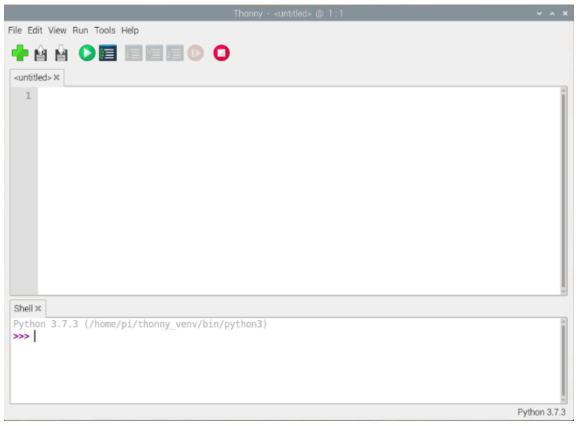
A selection of IDEs (Integrated Development Environments) can be used with the PICO. I tried VS code, Arduino but settled with Thonny which is simple to use and has more than enough functionality for this project.

THONNY INSTALL

Install Thonny

- On Windows, macOS, and Linux, you can install the latest Thonny IDE or update an existing version
- In a web browser, navigate to thonny.org
- In the top right-hand corner of the browser window, you will see download links for Windows and macOS, and instructions for Linux

Opening Thonny



```
Python 3.7.3 (/home/pi/thonny_venv/bin/python3)
>>> |
```

Open Thonny from your application launcher. It should look something like this:

You can use Thonny to write standard Python code. Type the following in the main window, and then click the Run button (you will be asked to save the file).

```
print('Hello World!')
```

PicoBot

Overview

PicoBot contains a RP2040 processor (Pico W), CYW43439 Wi-Fi, 2 x TB6612FNG DC Motor Drivers along with all required power supplies to enable operation via a 12v external power supply.

It is a bare processor; great care is required to prevent damage due to over current or over voltage.

We only use 3V3 for all our sensors - higher voltages are available explicitly for use with drive motor 12V and actuators 6V5.

The 5V supply is exclusively used for the processor. Power supply is not accessible from the I/O and is not for general use.

PLEASE NOTE YOUR PICOBOT IS NOT USER REPAIRABLE. IT MUST BE RETURNED TO THE EIETL OFFICE ALONG WITH A FAULT REPORT FOR REPAIR/REPLACEMENT. IT IS NOT TO BE TAKEN APART - THIS WILL DELAY REPAIR/REPLACEMENT.

PROCESSOR SPECIFICATIONS

- RP2040 microcontroller
- Dual-core Arm Cortex M0+ processor, flexible clock running up to 133 MHz
- 264KB of SRAM, and 2MB of on-board flash memory
- USB 1.1 with device and host support
- Low-power sleep and dormant modes
- Drag-and-drop programming using mass storage over USB
- 26 × multi-function GPIO pins
- 2 × SPI, 2 × I2C, 2 × UART, 3 × 12-bit ADC, 16 × controllable PWM channels
- Accurate clock and timer on-chip
- Temperature sensor
- Accelerated floating-point libraries on-chip
- 8 × Programmable I/O (PIO) state machines for custom peripheral support

Raspberry Pi Pico W adds on-board single-band 2.4GHz wireless interfaces (802.11n) using the Infineon CYW43439 while retaining the Pico form factor. The on-board 2.4GHz wireless interface has the following features:

- Wireless (802.11n), single band (2.4 GHz)

- WPA3
- Soft access point supporting up to four clients
- Bluetooth 5.2
    - Support for Bluetooth LE Central and Peripheral roles
    - Support for Bluetooth Classic

The antenna is an onboard antenna licensed from ABRACON (formerly ProAnt). The CYW43439 wireless chip is connected via SPI to the RP2040 microcontroller.

Due to pin limitations, some of the wireless interface pins are shared. The CLK is shared with VSYS monitor, so only when there isn't an SPI transaction in progress can VSYS be read via the ADC. The Infineon CYW43439 DIN/DOUT and IRQ all share one pin on the RP2040. Only when an SPI transaction isn't in progress is it suitable to check for IRQs. The interface typically runs at 33MHz.

For best wireless performance, the antenna should be in free space. For instance, putting metal under or close by the antenna can reduce its performance both in terms of gain and bandwidth. Adding grounded metal to the sides of the antenna can improve the antenna's bandwidth.

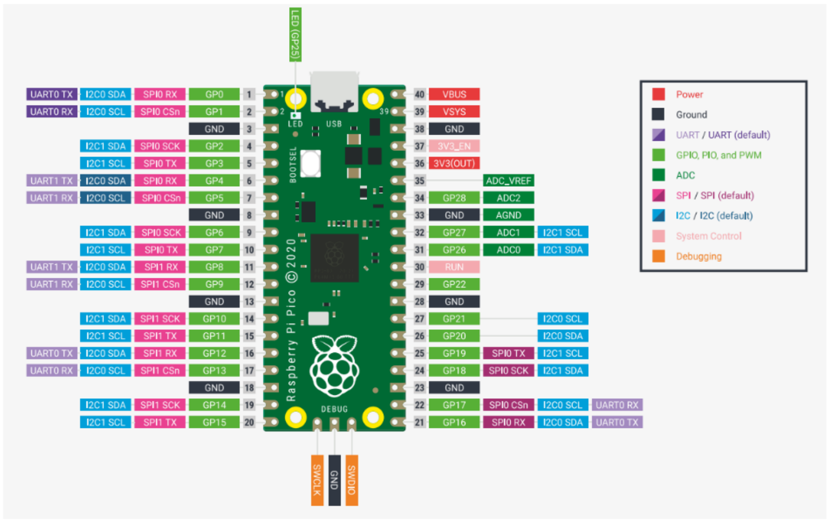
- Max current per pin 8mA
- Overhaul max current 300mA
- 3V3, 6V5 and 12v supply 500mA each
- Max voltage on i/O 3V3

MOTOR DRIVE SPECIFICATIONS

- 2 x Fermion TB6612FNG 2x1.2A DC Motor Driver breakouts
- VCC Operating Voltage: 2.7V~5.5V
- VM Input Power: 2.5V~12V
- Driver Channel: 2 channels
- Output Current: 1.2A (single-channel continuous drive current)
- Start / Peak Current: 2A (continuous pulse) / 3.2A (single pulse)
- Dimension: 0.79 x 0.77(in) / 20 x 19.50(mm)

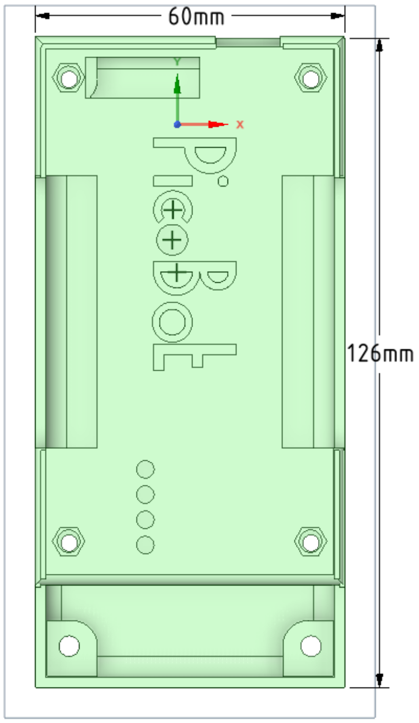
PINOUT

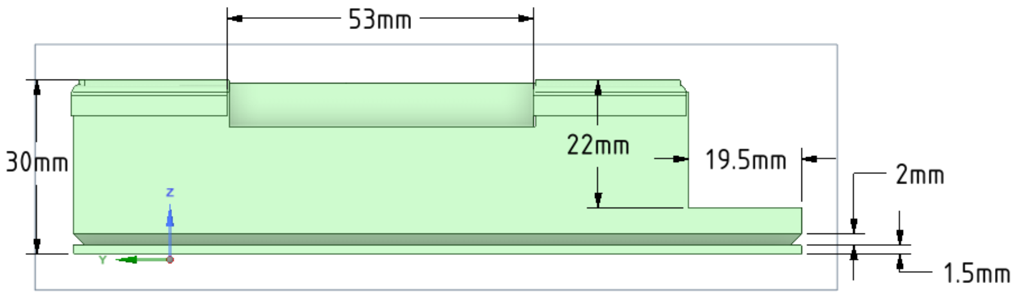
THIS IS A TEMPORARY PICTURE SECTION STILL UNDER CONSTRUCTION

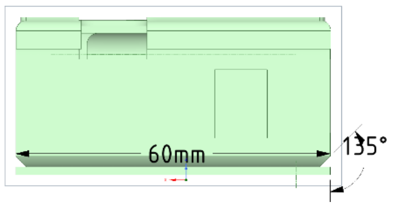


LED (GP25)
UART0 TX · I2C0 SDA · SPI0 RX · GP0 · 1
UART0 RX · I2C0 SCL · SPI0 CSn · GP1 · 2
GND · 3
I2C1 SDA · SPI0 SCK · GP2 · 4
I2C1 SCL · SPI0 TX · GP3 · 5
UART1 TX · I2C0 SDA · SPI0 RX · GP4 · 6
UART1 RX · I2C0 SCL · SPI0 CSn · GP5 · 7
GND · 8
I2C1 SDA · SPI0 SCK · GP6 · 9
I2C1 SCL · SPI0 TX · GP7 · 10
UART1 TX · I2C0 SDA · SPI1 RX · GP8 · 11
UART1 RX · I2C0 SCL · SPI1 CSn · GP9 · 12
GND · 13
I2C1 SDA · SPI1 SCK · GP10 · 14
I2C1 SCL · SPI1 TX · GP11 · 15
UART0 TX · I2C0 SDA · SPI1 RX · GP12 · 16
UART0 RX · I2C0 SCL · SPI1 CSn · GP13 · 17
GND · 18
I2C1 SDA · SPI1 SCK · GP14 · 19
I2C1 SCL · SPI1 TX · GP15 · 20
40 · VBUS
39 · VSYS
38 · GND
37 · 3V3_EN
36 · 3V3(OUT)
35 · ADC_VREF
34 · GP28 · ADC2
33 · GND · AGND
32 · GP27 · ADC1 · I2C1 SCL
31 · GP26 · ADC0 · I2C1 SDA
30 · RUN
29 · GP22
28 · GND
27 · GP21 · I2C0 SCL
26 · GP20 · I2C0 SDA
25 · GP19 · SPI0 TX · I2C1 SCL
24 · GP18 · SPI0 SCK · I2C1 SDA
23 · GND
22 · GP17 · SPI0 CSn · I2C0 SCL · UART0 RX
21 · GP16 · SPI0 RX · I2C0 SDA · UART0 TX
LED
USB
BOOTSEL
Raspberry Pi Pico ©2020
DEBUG
SWCLK
GND
SWDIO
Power
Ground
UART / UART (default)
GPIO, PIO, and PWM
ADC
SPI / SPI (default)
I2C / I2C (default)
System Control
Debugging

DIMENSIONS

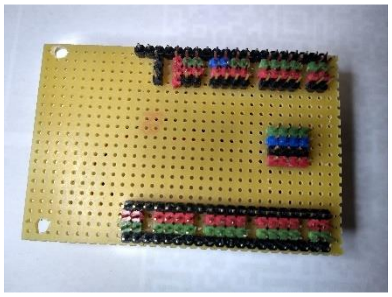
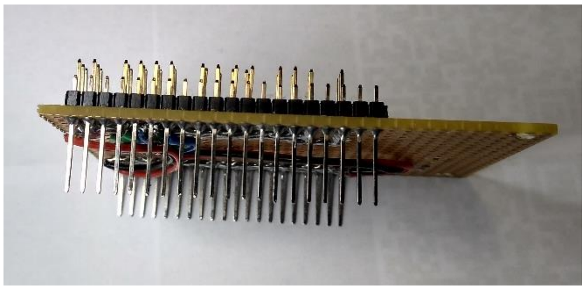
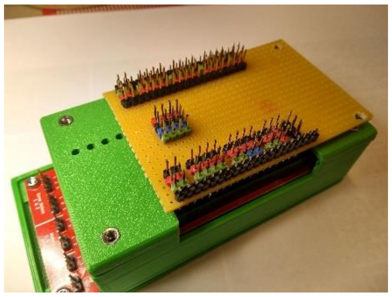






INTERFACE BOARD

One of the team’s first tasks is to construct an interface board to allow easy direct connection of I/O and sensors. Below is just an example. You should not require all the I/O - this was just built for testing.

The interface board is constructed using solder point/matrix board.

Once constructed and before applying power as with all I/O, check sensors with a bench Multi-meter for shorts, resistance, and continuity to/from/between all pins before use, then when power is applied check again.

Hints and Tips
Even though you will not need all the i/O it is a good idea to have some redundancy in the form of one or two spare connections.
Use colours to help confirm plug orientation where possible.

IMPORTANT NOTE
The team will only get 1 piece of board and 1 set of long pin headers so ensure the team think carefully what the layout should be before committing to construction.

I/O

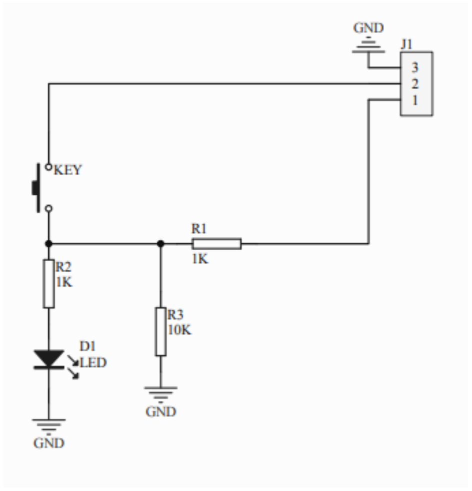
PUSH BUTTON 1 max per team

ALWAYS USE PLASTIC/NYLON SCREWs TO AFFIX MODULES/SENSORS

SPECIFICATION

- Supply Voltage: 3.3V to 5V
- Indicator LED on board
- Interface: Digital
- Size:22x30mm(0.87"x1.18")

CIRCUIT DIAGRAM



Hints and Tips

Check orientation before plugging in.

Interfacing with a bare processor requires consideration to current - use Ohms law to calculate what the push button could potentially draw.

LED 1 max per team

The digital pin (LOW on, HIGH off) is used to control it. The brightness of the LED can be controlled via PWM output.

ALWAYS USE PLASTIC/NYLON SCREWs TO AFFIX MODULES/SENSORS

SPECIFICATION

- Voltage: 3.3~5v
- Light color: Yellow
- Wavelength: 585~594nm
- Viewing angle: 65°
- Size: 30x22mm
- Weight: 5g

Pin Definition

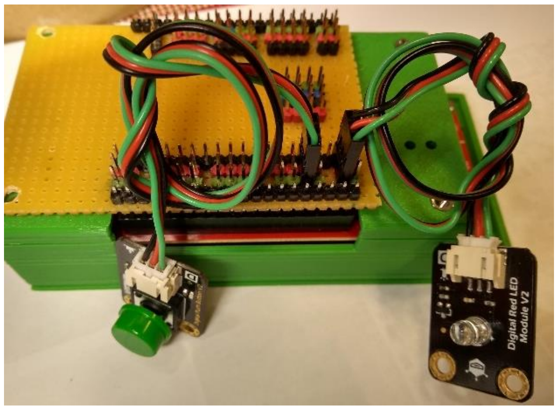
Digital control pin (support PWM)

Power Supply

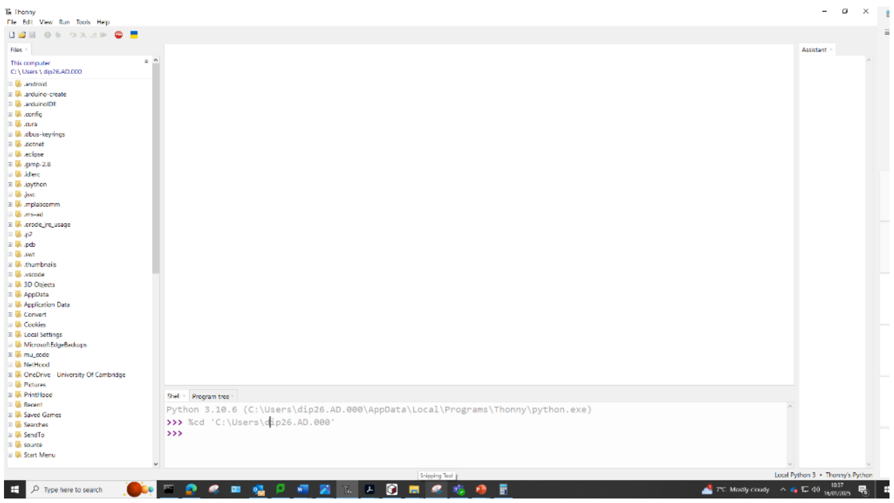
GND

PUSH BUTTON AND LED TESTING

Set up the test circuit using your interface board. Alternatively you could use a breadboard and jumpers though this is fraught with danger and is unreliable.

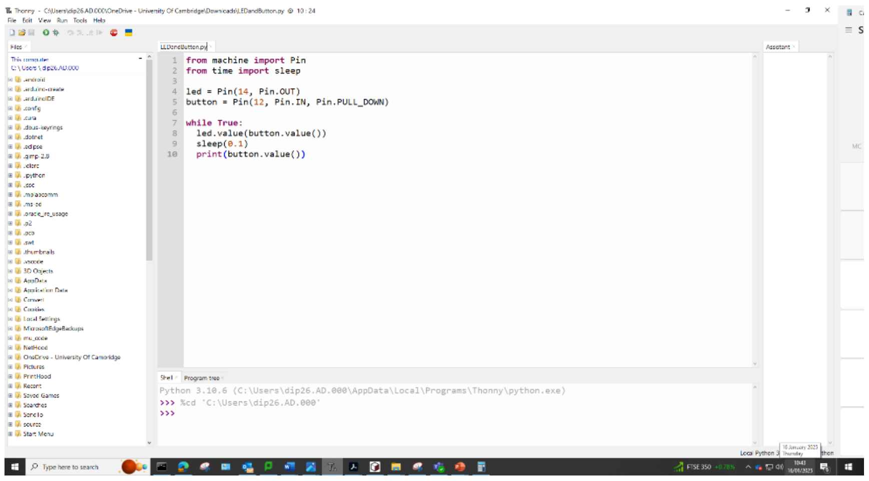


1 Open Thonny. You should see something like this. Go to file open new and load the example.



```
Python 3.10.6 (C:\Users\dip26.AD.000\AppData\Local\Programs\Thonny\python.exe)
>>> %cd 'C:\Users\dip26.AD.000'
>>>
```

2 You should have something like this

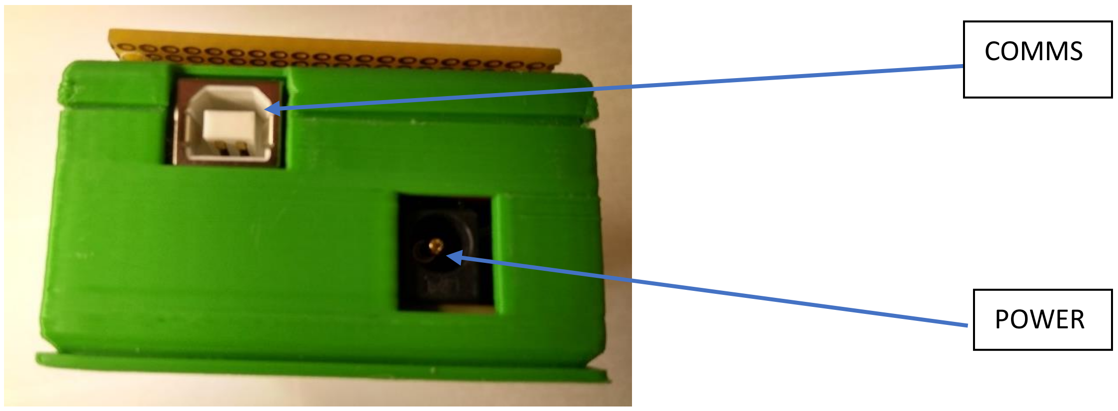


Thonny - C:\Users\dip26.AD.000\OneDrive - University Of Cambridge\Downloads\LEDandButton.py @ 10 : 24

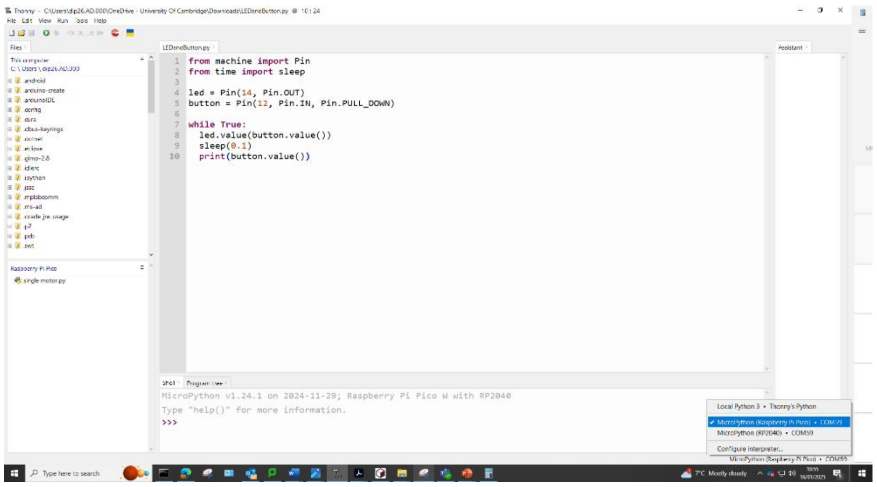
```
from machine import Pin
from time import sleep

led = Pin(14, Pin.OUT)
button = Pin(12, Pin.IN, Pin.PULL_DOWN)

while True:
    led.value(button.value())
    sleep(0.1)
    print(button.value())
```

```
Python 3.10.6 (C:\Users\dip26.AD.000\AppData\Local\Programs\Thonny\python.exe)
>>> %cd 'C:\Users\dip26.AD.000'
>>>
```

3. Your PicoBot is already configured for MicroPython. Plug in the comms cable PC to PicoBot and power either mains adaptor or battery 12V Max.

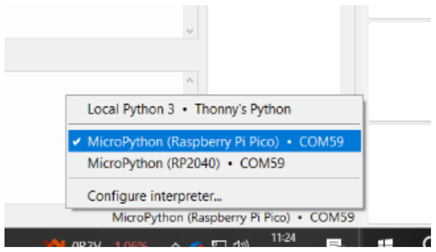


4. In Thonny (bottom right), click on local to open comms window. Click to connect to Raspberry Pi Pico. The comms port will have been set. This will open a second window on the left-hand side which is your Pico menu.

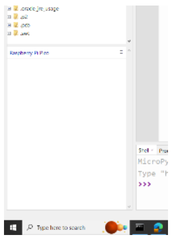


```
from machine import Pin
from time import sleep

led = Pin(14, Pin.OUT)
button = Pin(12, Pin.IN, Pin.PULL_DOWN)

while True:
    led.value(button.value())
    sleep(0.1)
    print(button.value())
```

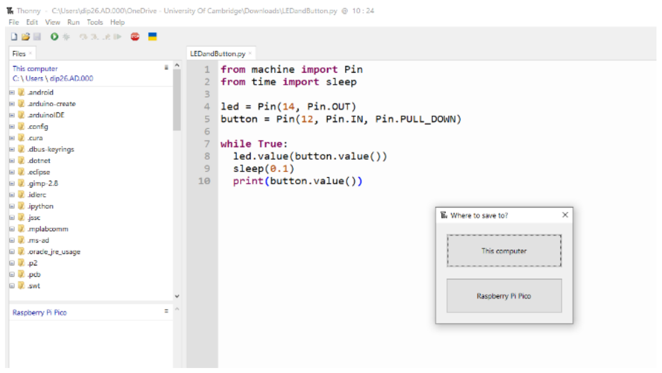
```
MicroPython v1.24.1 on 2024-11-29; Raspberry Pi Pico W with RP2040
Type "help()" for more information.
>>>
```



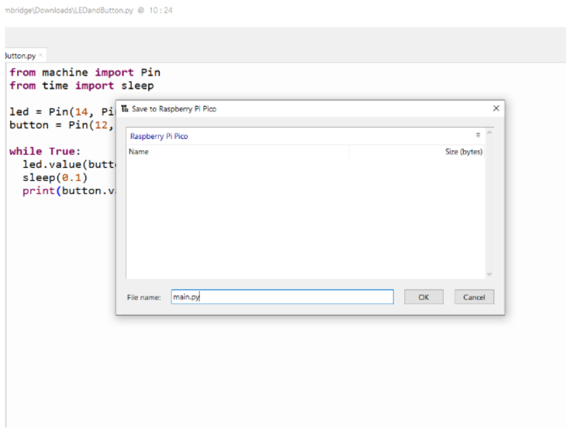
Local Python 3 • Thonny's Python
✓ MicroPython (Raspberry Pi Pico) • COM59
MicroPython (RP2040) • COM59
Configure interpreter...
MicroPython (Raspberry Pi Pico) • COM59



5. Click file > save as select Raspberry Pi Pico and save as main.py



```
from machine import Pin
from time import sleep

led = Pin(14, Pin.OUT)
button = Pin(12, Pin.IN, Pin.PULL_DOWN)

while True:
  led.value(button.value())
  sleep(0.1)
  print(button.value())
```


mbridge\Downloads\LEDandButton.py @ 10 : 24
Button.py
Save to Raspberry Pi Pico
Raspberry Pi Pico
Name
Size (bytes)
OK
Cancel

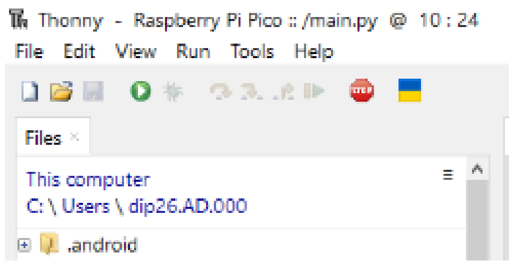
```
from machine import Pin
from time import sleep

led = Pin(14, Pi
button = Pin(12,

while True:
  led.value(butt
  sleep(0.1)
  print(button.v
```

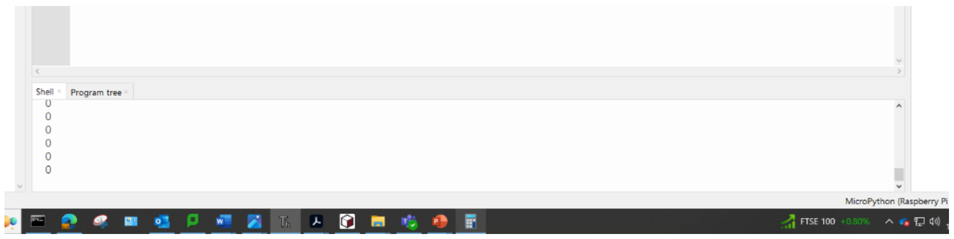
File name: main.py

Hints and Tips

Saving the file as main.py means that it will automatically run each time the pico is powered up. If you have a module, this would be saved as the name of the module.

6. If you now select run and press the button, the LED should go off and on and in the shell window the status should be given


Thonny - Raspberry Pi Pico :: /main.py @ 10 : 24
File Edit View Run Tools Help
Files
This computer
C: \ Users \ dip26.AD.000
.android



```
0
0
0
0
0
0
```

PUSH BUTTON LED EXAMPLE CODE

from time import sleep
from machine import Pin

Time is a big module – to save processor space/speed only use the bit you want.

led = Pin(14, Pin.OUT)
button = Pin(12, Pin.IN, Pin.PULL_DOWN)

Set led to be on Pin 14
Set the push button to be on pin 12

```
while True:
  led.value(button.value())
  sleep(0.1)
  print(button.value())
```

While true, anything below the while true will run continuously unless stopped
Set the led to the push button value
Sleep stops the processor from looping for 0.1
Prints out the value to the shell screen

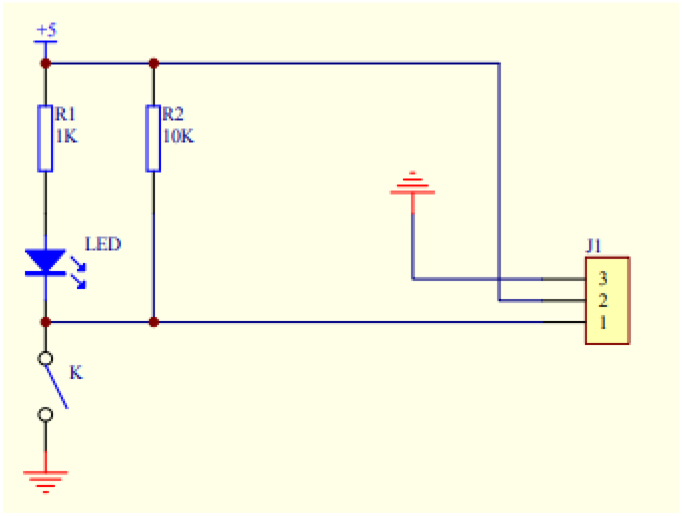
SUMMARY OF PUSH BUTTON LED EXAMPLE

- Connect to PicoBot via comms setting bottom right
- Saving as main.py means it will continue to run even if you disconnect the comms
- While true will continue to run code unless stopped

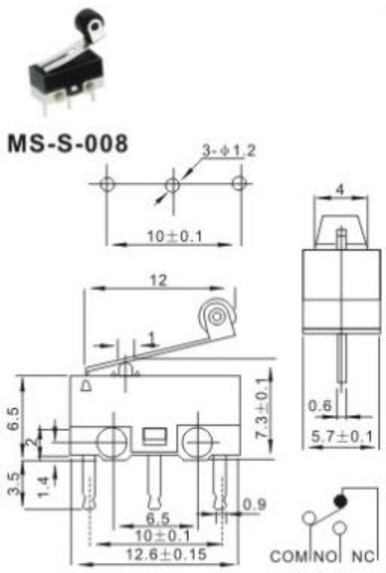
MICRO SWITCH LEFT/RIGHT 1 left, 1 right max per team

ALWAYS USE PLASTIC/NYLON SCREWs TO AFFIX MODULES/SENSORS

CIRCUIT DIAGRAM



SPECIFICATION



- Working Voltage: 5v
- Pinout:
- 1 Digital output
- 2 VCC
- 3 GND
- Onboard status indicator LED
- Directly connected to the IO Expansion shield for Arduino
- M3 mounting hole x2
- Size: 30x20x8mm (1.18x0.79x0.31")

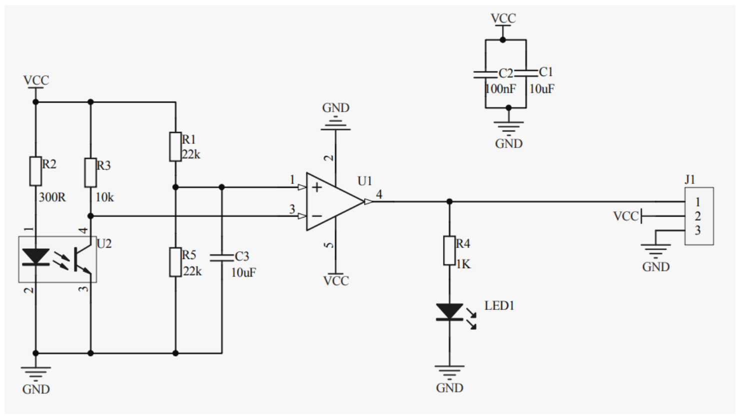
MICRO SWITCH EXAMPLE

Replace the push button with the micro switch. You should still be able to switch the LED off/on like the push button but not the same. Why?

LINE SENSOR 4 Max per team

Line tracking (following) sensor to guide your robot by telling white from black quickly and accurately.

ALWAYS USE PLASTIC/NYLON SCREWs TO AFFIX MODULES/SENSORS



SPECIFICATION

- Supply Voltage: 3.3V to 5V
- Detecting Range: 1-2cm
- Interface:Digital
- Operating current: 20mA @ 5V
- Operating temperature range: 0°C ~ + 50°C
- Output: TTL(Black for LOW output, White for HIGH output)
- Size:28x10mm (1.1x 0.4 in)

LINE SENSOR EXAMPLE

Replace the micro switch with the line sensor you should be able to switch the LED off/on when the line sensor is above light/dark surfaces. Complete the table below to optimize line sensor height on your robot.

Line sensor number	Operational Min. Height	Operational Max. Height	Average operational height

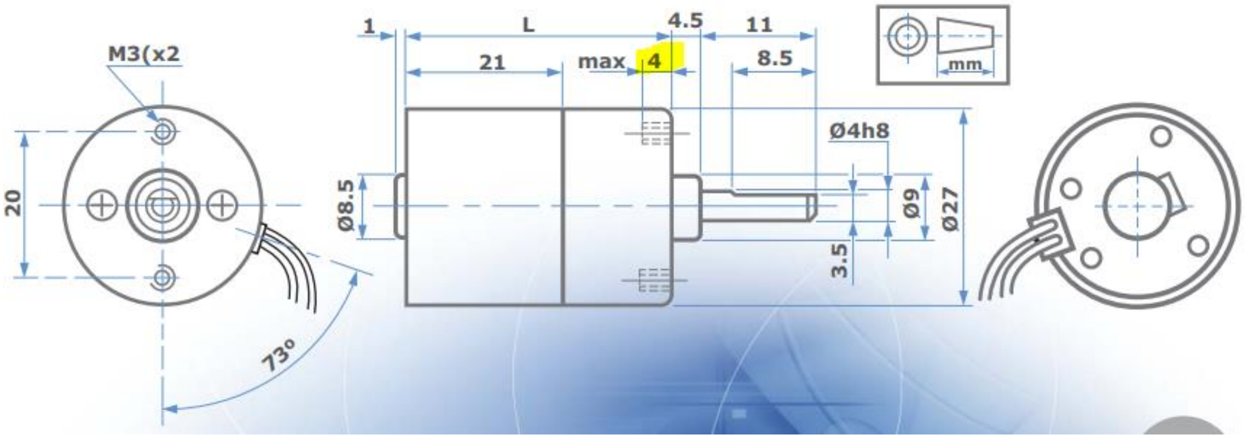
MOTORS and ACTUATORS

DC MOTORS 2 max per team

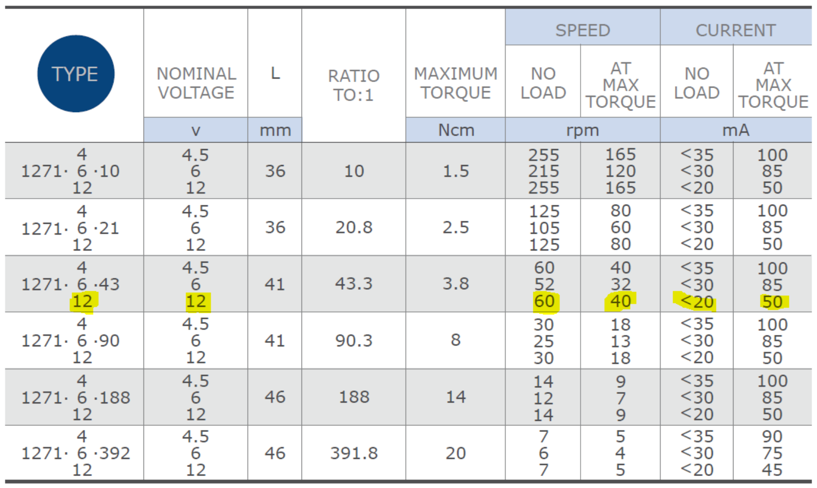
Teams are provided with 2 x 12-volt 40 RPM geared dc motors. The motors are supplied with 5mm screws giving a max. protrusion into the motor of 4mm and a Min. thickness of motor bracket of 1mm. All motors have been fully tested before issue.

WARNING
The maximum insertion depth of screws into the motor is 4mm - any further and you will destroy the motor. There is no excuse for this and getting a replacement will require a full explanation. Note we will know when we strip the motor how it has been damaged.

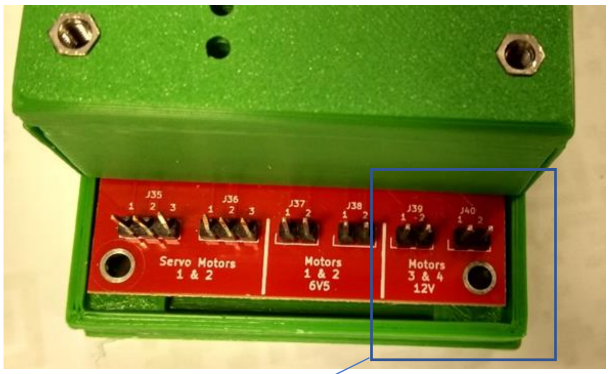
DC MOTOR DIMENSIONS



DC MOTOR SPECIFICATION



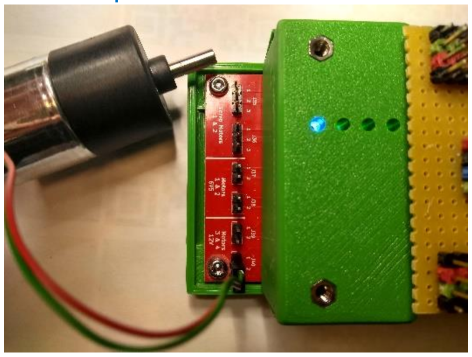
TYPE	NOMINAL VOLTAGE	L	RATIO TO:1	MAXIMUM TORQUE	SPEED		CURRENT	
					NO LOAD	AT MAX TORQUE	NO LOAD	AT MAX TORQUE
	v	mm		Ncm	rpm		mA	
4 1271· 6 ·10 12	4.5 6 12	36	10	1.5	255 215 255	165 120 165	<35 <30 <20	100 85 50
4 1271· 6 ·21 12	4.5 6 12	36	20.8	2.5	125 105 125	80 60 80	<35 <30 <20	100 85 50
4 1271· 6 ·43 12	4.5 6 12	41	43.3	3.8	60 52 60	40 32 40	<35 <30 <20	100 85 50
4 1271· 6 ·90 12	4.5 6 12	41	90.3	8	30 25 30	18 13 18	<35 <30 <20	100 85 50
4 1271· 6 ·188 12	4.5 6 12	46	188	14	14 12 14	9 7 9	<35 <30 <20	100 85 50
4 1271· 6 ·392 12	4.5 6 12	46	391.8	20	7 6 7	5 4 5	<35 <30 <20	90 75 45



Drive Motors fed from Motors 3 & 4 12V

DC MOTOR EXAMPLE

Connect one of your motors to motor 4 connection, load the DC Motor Example Code and run. Your motor should spin forwards and backwards.



DC MOTOR EXAMPLE CODE

```
from machine import Pin, PWM
from utime import sleep
class Motor:
    def __init__(self):
        self.m1Dir = Pin(7 , Pin.OUT)         # set motor direction
        self.pwm1 = PWM(Pin(6))               # set speed
        self.pwm1.freq(1000)                  # set max frequency
        self.pwm1.duty_u16(0)                 # set duty cycle
    def off(self):
        self.pwm1.duty_u16(0)
    def Forward(self):
        self.m1Dir.value(0)                   # forward = 0 reverse = 1 motor 1
        self.pwm1.duty_u16(int(65535*100/100))  # speed range 0-100 motor 1
    def Reverse(self):
        self.m1Dir.value(1)
        self.pwm1.duty_u16(int(65535*30/100))
motor=Motor()
while True:
    motor.Forward()
```

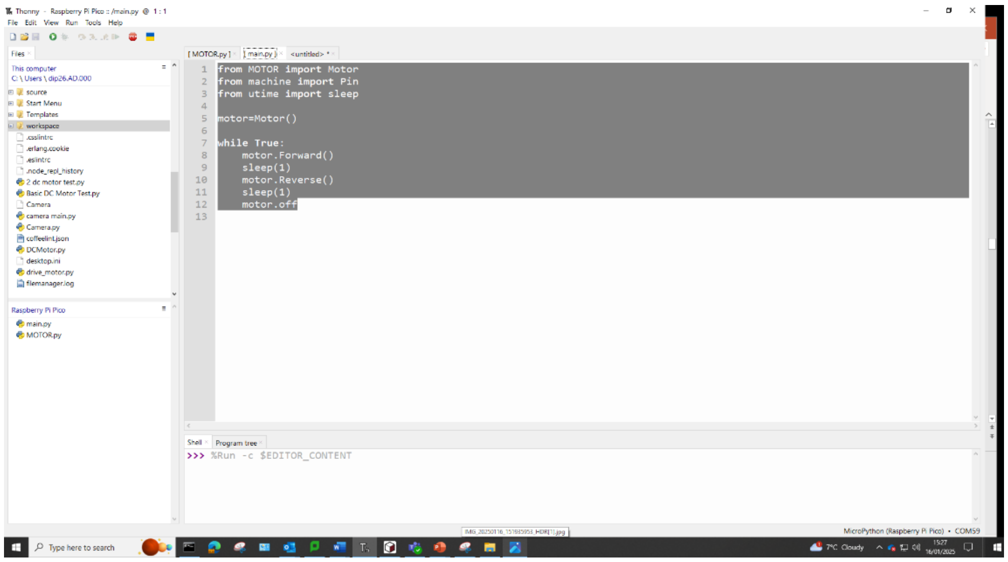
```
   sleep(1)
   motor.Reverse()
   sleep(1)
```

Note you have 22 lines of code to read through. Create a DC Motor Module, load this onto your PicoBot saving as MOTOR.py.

DC MOTOR MODULE EXAMPLE (MOTOR.py)

```
from machine import Pin, PWM
class Motor:
   def __init__(self):
      self.m1Dir = Pin(7 , Pin.OUT)        # set pin left wheel
      self.pwm1 = PWM(Pin(6))
      self.pwm1.freq(1000)
      self.pwm1.duty_u16(0)
   def off(self):
      self.pwm1.duty_u16(0)
   def Forward(self):
      self.m1Dir.value(0)                  # forward = 0 reverse = 1 motor 1
      self.pwm1.duty_u16(int(65535*100/100))  # speed range 0-100 motor 1
   def Reverse(self):
      self.m1Dir.value(1)
      self.pwm1.duty_u16(int(65535*30/100))
```

Now generate your main code again. Save it to your PicpBot as main.py. Your Thonny should look something like this.

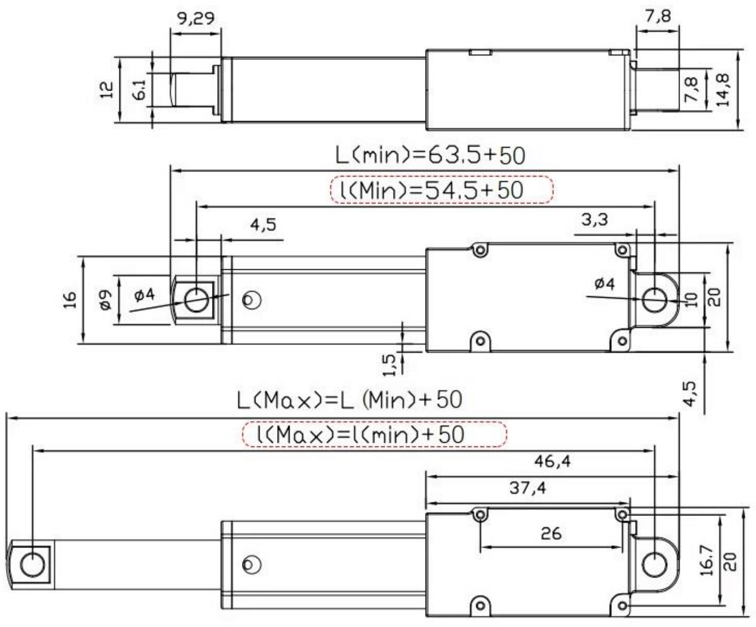


```python
from MOTOR import Motor
from machine import Pin
from utime import sleep

motor=Motor()

while True:
    motor.Forward()
    sleep(1)
    motor.Reverse()
    sleep(1)
    motor.off
```

```
>>> %Run -c $EDITOR_CONTENT
```

LINEAR ACTUATOR (1 max per team)

DIMENSIONS



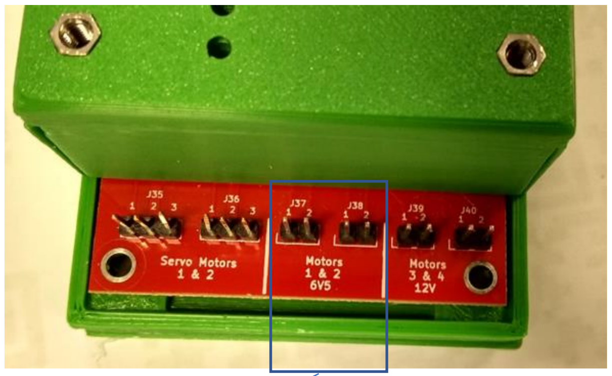
https://www.dfrobot.com/product-2368.html

SPECIFICATION

Rated voltage: 6V

Stroke: 50mm

Thrust: 128N



Linear Actuators can connect to either 1 0r 2 6V5 MAX

LINEAR ACTUATOR CODE EXAMPLE

Use the DC Motor example code extending sleep times to allow full linear extension or as required.

SERVO

DESCRIPTION

DSS-M15S is a heavy-duty metal geared standard servo with 270°wide angle, high torque power, improved stability and durability.

DF Metal Geared 15Kg Standard Servo 270° (DSS-M15S) - DFRobot

SPECIFICATION 1 per team

Electronic specifications

Operating voltage: 4.8-7.2V

6V test environment

Operating speed (no load): 0.18 sec/60 degrees

Resting current: 80mA

Locking torque: 11.5KG*cm

Stall current: 1.4A

Standby current: 4mA

7V test environment

Operating speed (no load): 0.16sec/60 degrees

Resting current: 100mA

Locking torque: 12KG*cm

Stall current: 1.76A

Standby current: 5mA

Mechanical specifications

Gear material: metal gear

Operating angle: 270 degrees

Wiring gauge: 28PVC

Data line length: 320mm

Gear bracket spline: 25T/5.80

Gear ratio: 310:1

Size: 54.5*20*47.5mm

Control specifications

Feedback signal: 0-3.3V

Control signal: RC PWM

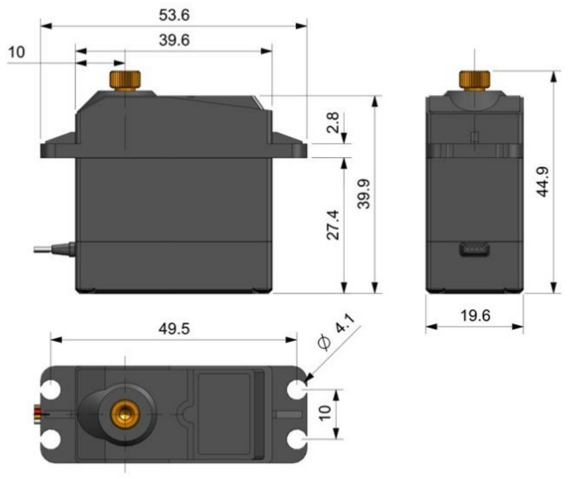
Pulse range: 500-2500 us

Median signal value: 1500us

Clockwise rotation: <1500us

Contro frequency: 50-330Hz (Arduion compatible)

DIMENSIONS



CONNECTIONS

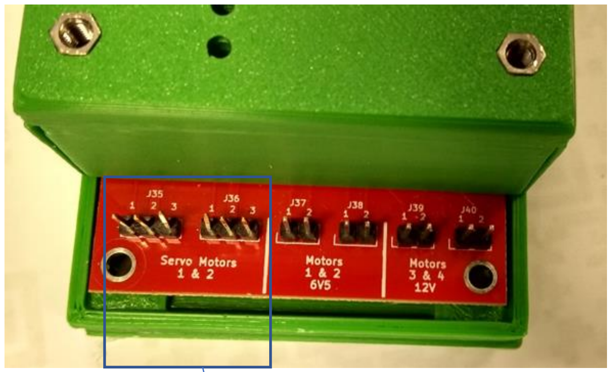
SERVO CONNECTIONS

Orange ---pin 1--- linked to PICO pin 13 channel 1

and linked to PICO pin 15 for channel 2

Red ---pin 2

Brown ---pin3

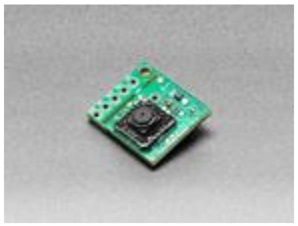


Servo connections you can use either one.

SERVO CODE EXAMPLE

```
from machine import Pin, PWM
from time import sleep
# Set up PWM Pin for servo control
servo_pin = machine.Pin(15)
servo = PWM(servo_pin)
# Set Duty Cycle for Different Angles
max_duty = 7864
min_duty = 1802
half_duty = int(max_duty/2)
#Set PWM frequency
frequency = 50
servo.freq (frequency)
try:
    while True:
        #Servo at 0 degrees
        servo.duty_u16(min_duty)
        sleep(2)
        #Servo at 90 degrees
        servo.duty_u16(half_duty)
        sleep(2)
        #Servo at 180 degrees
        servo.duty_u16(max_duty)
        sleep(2)
except KeyboardInterrupt:
    print("Keyboard interrupt")
    # Turn off PWM
    servo.deinit()
```

TINY CODE READER



DESCRIPTION

This module is designed to read QR codes from screens, paper, or e-ink displays, and transmit the encoded text information to a controlling microcontroller. It transmits this information over I2C and has an LED that indicates the state of the scanning process.

SPECIFICATION

- Power Supply 3.3 Volts
- Logic Voltage 3.3 Volts
- Minimum Operating Temperature -20.0 Celsius
- Maximum Operating Temperature 80.0 Celsius
- Maximum I2C Frequency 400 kilohertz
- Maximum Current 40 milliamp

CONNECTIONS

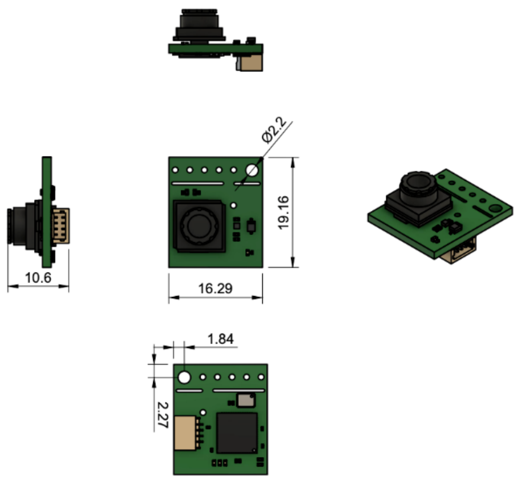
- 3.3v Power 3.3 Volt power supply. RED
- GND Ground Connection to ground. BLACK
- SDA Data I2C pin for data transmission
- SDC Clock I2C pin driven with clock from main controller.
- INT Interrupt Not used on this module.

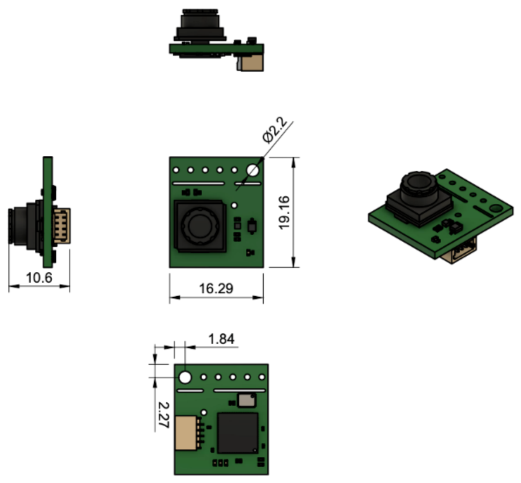
tiny_code_reader_docs/README.md at main · usefulsensors/tiny_code_reader_docs · GitHub

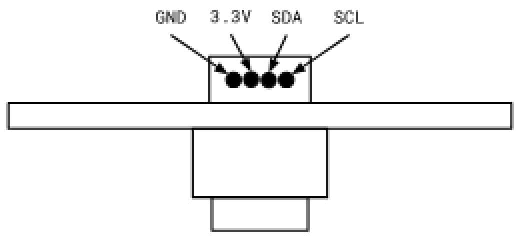
Hint and Tips

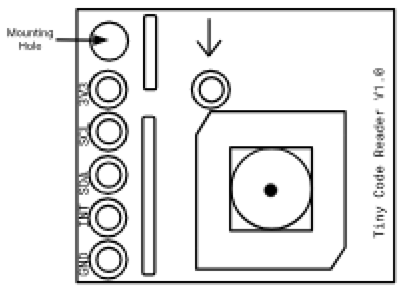
During testing my sensor read the QR code repeatably at around 180mm. Check the optimum distance for your sensor - they are all slightly different.

DIMENSIONS


Ø2.2
19.16
10.6
16.29
1.84
2.27


Ø2.2
19.16
10.6
16.29
1.84
2.27


GND
3.3V
SDA
SCL


Mounting Hole
3V3
SCL
SDA
INT
GND
Tiny Code Reader V1.0

TINY CODE READER EXAMPLE CODE

```
import network

import struct # This module converts between Python values and C structs represented as Python bytes objects.

from time import sleep

import machine


# NOTE INCORRECT CONNECTIONS WILL DESTROY THE SENSOR. CHECK WITH BENCH MULTIMETER BEFORE POWER/USE

# Red---3v3

# Black---ground

# Blue---sda

# Yellow---scl


# The code reader has the I2C ID of hex 0c, or decimal 12.

TINY_CODE_READER_I2C_ADDRESS = 0x0C


# How long to pause between sensor polls.

TINY_CODE_READER_DELAY = 0.05


TINY_CODE_READER_LENGTH_OFFSET = 0

TINY_CODE_READER_LENGTH_FORMAT = "H"

TINY_CODE_READER_MESSAGE_OFFSET = TINY_CODE_READER_LENGTH_OFFSET +
struct.calcsize(TINY_CODE_READER_LENGTH_FORMAT)

TINY_CODE_READER_MESSAGE_SIZE = 254

TINY_CODE_READER_MESSAGE_FORMAT = "B" * TINY_CODE_READER_MESSAGE_SIZE

TINY_CODE_READER_I2C_FORMAT = TINY_CODE_READER_LENGTH_FORMAT +
TINY_CODE_READER_MESSAGE_FORMAT

TINY_CODE_READER_I2C_BYTE_COUNT = struct.calcsize(TINY_CODE_READER_I2C_FORMAT)


# Set up for the Pico, pin numbers will vary according to your setup.

i2c = machine.I2C(1,
            scl=machine.Pin(19), # yellow
            sda=machine.Pin(18), # blue
            freq=400000)

print(i2c.scan())

# Keep looping and reading the sensor results until we get a QR code
while True:
  sleep(TINY_CODE_READER_DELAY)
  read_data = i2c.readfrom(TINY_CODE_READER_I2C_ADDRESS,
              TINY_CODE_READER_I2C_BYTE_COUNT)
#   print('raw data',read_data)
  message_length,  = struct.unpack_from(TINY_CODE_READER_LENGTH_FORMAT, read_data,
TINY_CODE_READER_LENGTH_OFFSET)
  message_bytes = struct.unpack_from(TINY_CODE_READER_MESSAGE_FORMAT, read_data,
TINY_CODE_READER_MESSAGE_OFFSET)

  if message_length == 0:
#     print('nothing')
    continue
  try:
    message_string = bytearray(message_bytes[0:message_length]).decode("utf-8")
    print('barcode:', message_string)

  except:
    print("Couldn't decode as UTF 8")
    pass
```

ADDITIONAL SENSORS

You will also be supplied with a 1 x ultrasonic distance sensor and 1 x time of flight sensor. You will need to research these to develop your own code.

Ultrasonic Distance sensor

DESCRIPTION

DFRobot URM09 is an ultrasonic sensor specially designed for fast ranging and obstacle avoidance applications. Its measuring frequency can reach up to 30Hz. The sensor adopts built-in temperature compensation and analog output. Meanwhile, it can provide accurate distance measurement within 500cm. The sensor is compatible with Arduino, Raspberry Pi, or other main-control boards with 3.3V or 5V logic level.

Gravity: URM09 Ultrasonic Distance Sensor (2～500cm, Analog) - DFRobot

SPECIFICATION

- Power Supply: 3.3~5.5V DC
- Operating Current: 20mA
- Operating Temperature: -10℃～＋70℃
- Measurement Range: 2cm～500cm
- Resolution: 1cm
- Accuracy: 1%
- Frequency: 30Hz Max
- Dimension: 47mm × 22 mm/1.85× 0.87”

VL53L0X ToF DISTANCE SENSOR (30-2000mm)

DESCRIPTION

The VL53L0X laser range finder is a high-precision distance sensor that based on new Time-of-Flight (ToF) principle. VL53L0X provides accurate distance measurement whatever the target reflection unlike traditional technology. It can measure absolute distances up to 2m.

DFRobot has introduced VL53L0X into Gravity Series. It provides Gravity-I2C interface, plug and play, supports 3.3V~5V power supply, and is compatible with more microcontrollers, and adapt to more application scenarios than others.

The VL53L0X integrates a leading-edge SPAD array (Single Photon Avalanche Diodes) and embeds ST's second generation FlightSenseTM patented technology. Its accuracy is ±3%, response time is less than 30ms, power consumption is only 20mW in normal operation mode, stand-by power consumption is 5uA.

The VL53L0X's 940nm VCSEL emitter (vertical cavity surface emitting laser) is totally invisible to the human eye, coupled with internal physical infrared filters, it enables longer ranging distance, higher immunity to ambient light and better robustness to cover-glass optical cross-talk.

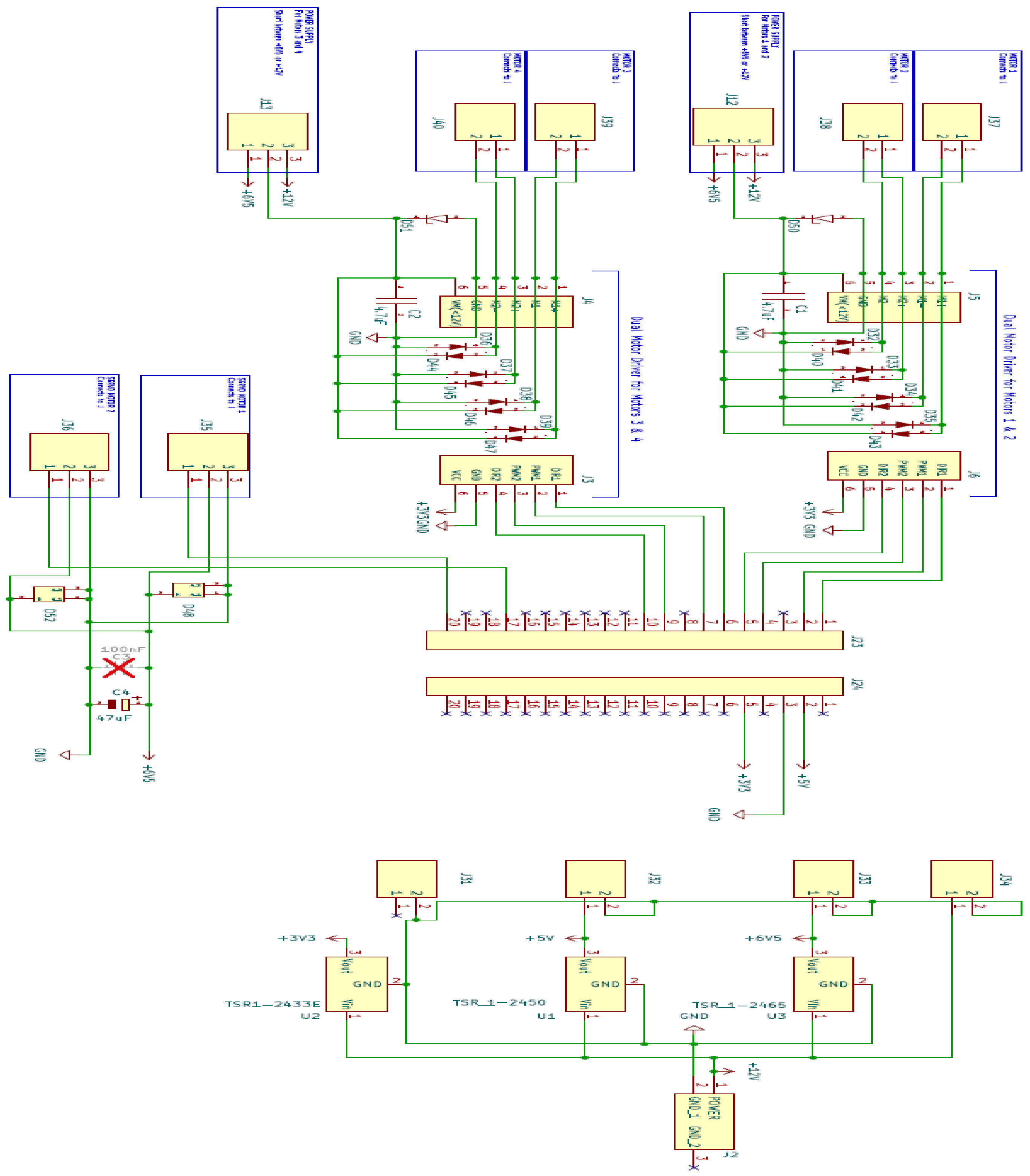
Gravity: VL53L0X ToF Distance Sensor (30-2000mm) - DFRobot

SPECIFICATION

- Power supply: 3.3-5V DC
- Operating Voltage: 2.8V
- Infrared emitter: 940nm
- Range: 30-2000mm
- FOV: 25°
- Ranging Accuracy: ±3%
- Sampling Time: <= 30ms
- Operating Temperature: -20-70 °C
- Interface Type: Gravity-I2C
- Product Size: 20*22 mm/0.787*0.866 inches

ANNEX

PicoBot DATA LOWER PCB


POWER SUPPLY For Motors 3 and 4
J13
+6V5
+12V
MOTOR 4
J40
MOTOR 3
J39
POWER SUPPLY For Motors 1 and 2
J12
MOTOR 2
J38
MOTOR 1
J37
D51
D50
C2
4.7uF
C1
4.7uF
GND
J4
J5
Dual Motor Driver for Motors 3 & 4
Dual Motor Driver for Motors 1 & 2
J3
J6
VCC
GND
DIR2
PWM2
PWM1
DIR1
+3V3
SERVO MOTOR 2
J36
SERVO MOTOR 1
J35
D52
D48
100nF
C3
C4
47uF
J23
J24
+3V3
+5V
J31
J32
J33
J34
TSR1-2433E
U2
TSR_1-2450
U1
TSR_1-2465
U3
Vout
GND
Vin
+6V5
POWER
GND_1
GND_2
J2