Radial distance gauge

What is a servo?

*Definitions of words in **bold** are on the last page*

- A servo is a special type of motor that uses a built in sensor to hold a specified angle.
- Most servos have 3 wires: **5V+**, **Sig**, **GND**
- The signal wire uses **PWM** to control the servos angle, the range of motion of a servo is typically limited to 180 or 360 degrees.
- For most servos the **PWM** signal has a 1ms to 2ms **pulse width** with a **frequency** of 50Hz. The Pico uses a **Duty cycle** to set the pulse width.

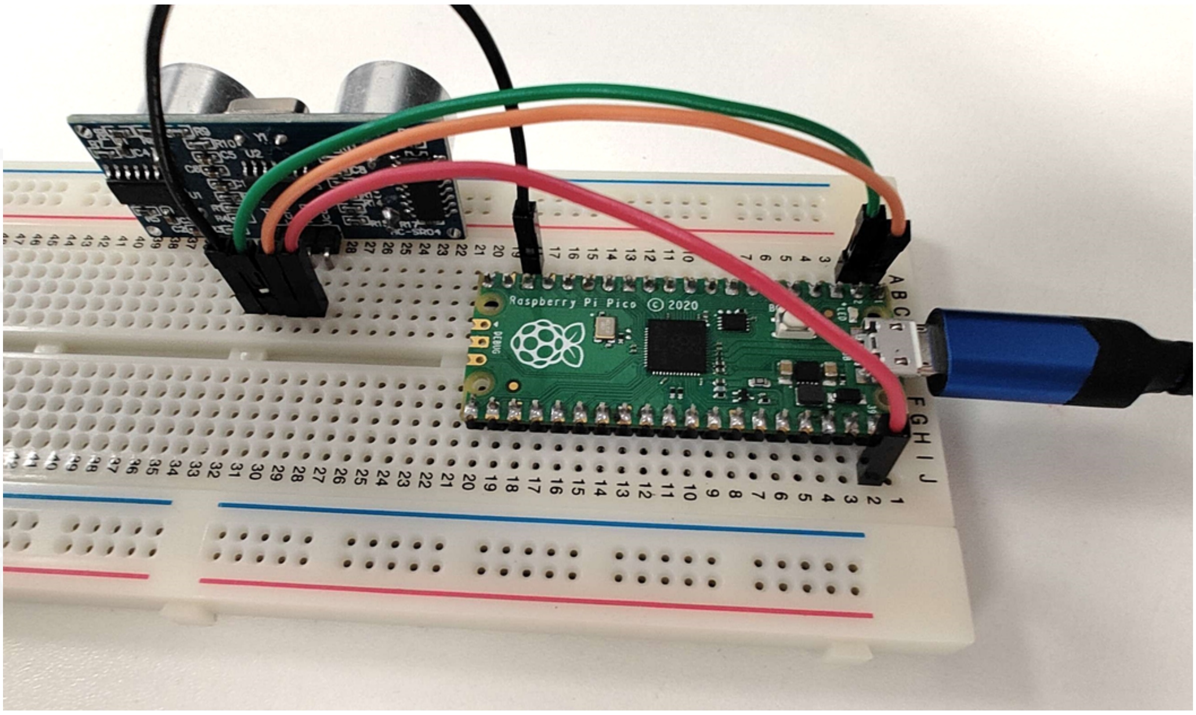
What is an ultrasonic sensor?

- An ultrasonic sensor uses a sonar pulse to measure the time it takes to recieve the echo of that pulse, that time can be converted to distance.
- Ultrasonic sensors have 4 wires: **V++**, **Trig**, **Echo**, **GND**
- The sensor will send a sonar pulse on the **falling edge** of a pulse applied to the trigger wire.
- Once the echo is recieved the echo wire will go **high** for a period of time proportional to the sound wave travel time.

This task is done using a Pi Pico, if you want to use an arduino follow the arduino worksheet (to be created)

Get distance from the ultrasonic sensor:

- Start by connecting the ultrasonic sensor to the Pi Pico, because the sensor has pins built-in you can plug it directly into the breadboard.
- Now with some wires connect sensor GND to Pico GND, Echo to GPIO 1, Trig to GPIO 0 and Vcc to VSYS as shown on the image on the next page:



- Once you've connected the sensor up correctly you can work on the code as shown bellow, the code is explained with its comments so you can understand it while you write it, if something doesnt make sense then please ask:

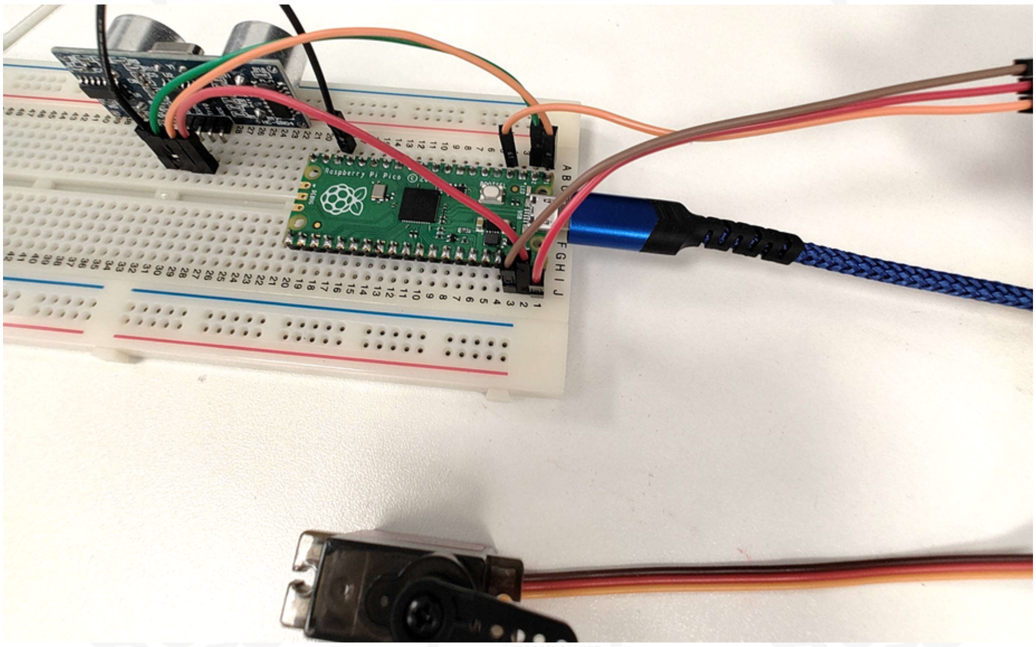
```
from machine import Pin # import the Pin class
import time # import the time library

trig = Pin(0, Pin.OUT) # setup the trigger pin to output on GPIO 0
echo = Pin(1, Pin.IN, Pin.PULL_DOWN) # setup the echo pin to input on GPIO 1

while True:
    # Recieving ultrasonic sensor data

    trig.off() # Make sure the trigger pin is off.
    time.sleep(0.1) # Give the sensor time to settle.

    trig.on() # start the trigger pulse
    time.sleep_us(2) # keep the pin high for 2 microseconds
    trig.off() # end the trigger pulse

    while echo.value() == 0: # keep doing nothing until the echo value rises
        start = time.ticks_us() # the start variable will keep updating to the current time

    while echo.value() == 1: # this is the pulse we want to record
        end = time.ticks_us() # so we keep updated the time the pulse ends until it actually does and breaks the loop

    distance = (end - start) / 58.8 # find the difference in time and divide by a constant to find the distance in cm
    print(round(distance, 1), "cm") # round to 1 d.p and print the result to the console

```

- Once you run the code you should have a working sensor!

Making a servo spin and the final product:

- Now the is sensor working we can work on getting the servo to move, but before we do that you should comment out your code in the while loop using 2 tripple quotes """ so we can work on the servo seperately.

- Now you need to connect your servo to your pi pico. Because servos have female pins you will need some male to male wires to do this.
- You will need to connect the servo's GND to the Pi Pico's GND, 5V+ to VBUS and signal to GPIO 2 like so:



- Now you have finished the hardware, we can go back to the code, you will need to add a new pin definition and also create a corresponding PWM object like so:

```
servo = Pin(2, Pin.OUT) # setup the PWM output pin on GPIO 2 for the servo
servoPWM = PWM(servo) # create a PWM object with the servo variable
servoPWM.freq(50) # set the PWM frequency, 50Hz is the standard for servos
```

- And then you can add some more code to the end of your while loop:

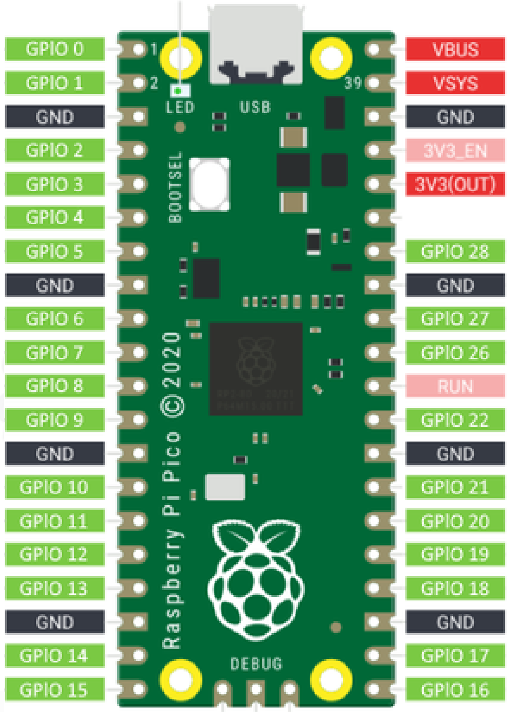
```
    """
    pulseWidth = 1000 # set the PWM pulse width in microseconds to a variable (must be in range of 0 to 2000)
    dutyCycle = int(65025 * (pulseWidth + 600) / 20000) # math to map the pulseWidth to a value between 0 and 65025
    servoPWM.duty_u16(dutyCycle) # set the duty cycle to the servo pin
```

- Running the code now should bring the servo to its center position, if it isnt aligned properly ask me to change the location of the arm.
- Now we can remove the comments from our sensor code and change the pulseWidth definition to convert the ultrasonic sensors distance output to a valid pulse width:

```
    pulseWidth = 2000 - min(distance * 33.3, 2000) # convert the distance in cm to a valid pulse width with a max of 2000
```

Done!

PWM - Pulse width modulation is a technique to represent the amplitude of an analog signal, it can also be used to control devices like servos with a **pulse width** between 1 and 2 milliseconds
Pulse width - The length of a **high** pulse inbetween a **low** signal, typically measured in seconds or milliseconds.
Frequency - How frequent something happens, measured in hertz (Hz). for something with 10Hz it would happen 10 times a second.
Duty Cycle - Pulse width as a percentage of the **period**.
Period - A period is the amount of time something takes for one cylce. It can be calculated by **period** = 1/**frequency**.
Falling edge - When a **high** signal changes to a **low** signal (voltage falls). For when **low** goes to **high**, that is called a rising edge.
High - Electrical terminology for true or binary 1 when powering a wire.
Low - Electrical terminology for false or binary 0 for a wire with ~0V.
data pins - This refers to any pin or wire carrying some sort of information that isnt a GND or V++ pin/wire.



Use the machine.Pin class:

```
from machine import Pin

p0 = Pin(0, Pin.OUT)    # create output pin on GPIO0
p0.on()                 # set pin to "on" (high) level
p0.off()                # set pin to "off" (low) level
p0.value(1)             # set pin to on/high

p2 = Pin(2, Pin.IN)     # create input pin on GPIO2
print(p2.value())       # get value, 0 or 1

p4 = Pin(4, Pin.IN, Pin.PULL_UP) # enable internal pull-up resistor
p5 = Pin(5, Pin.OUT, value=1) # set pin high on creation
```

PWM (pulse width modulation)

There are 8 independent channels each of which have 2 outputs making it 16 PWM channels in total which can be clocked from 7Hz to 125Mhz.

Use the `machine.PWM` class:

```
from machine import Pin, PWM

pwm0 = PWM(Pin(0))      # create PWM object from a pin
pwm0.freq()             # get current frequency
pwm0.freq(1000)         # set frequency
pwm0.duty_u16()         # get current duty cycle, range 0-65535
pwm0.duty_u16(200)      # set duty cycle, range 0-65535
pwm0.deinit()           # turn off PWM on the pin
```

Pi Pico Code Docs