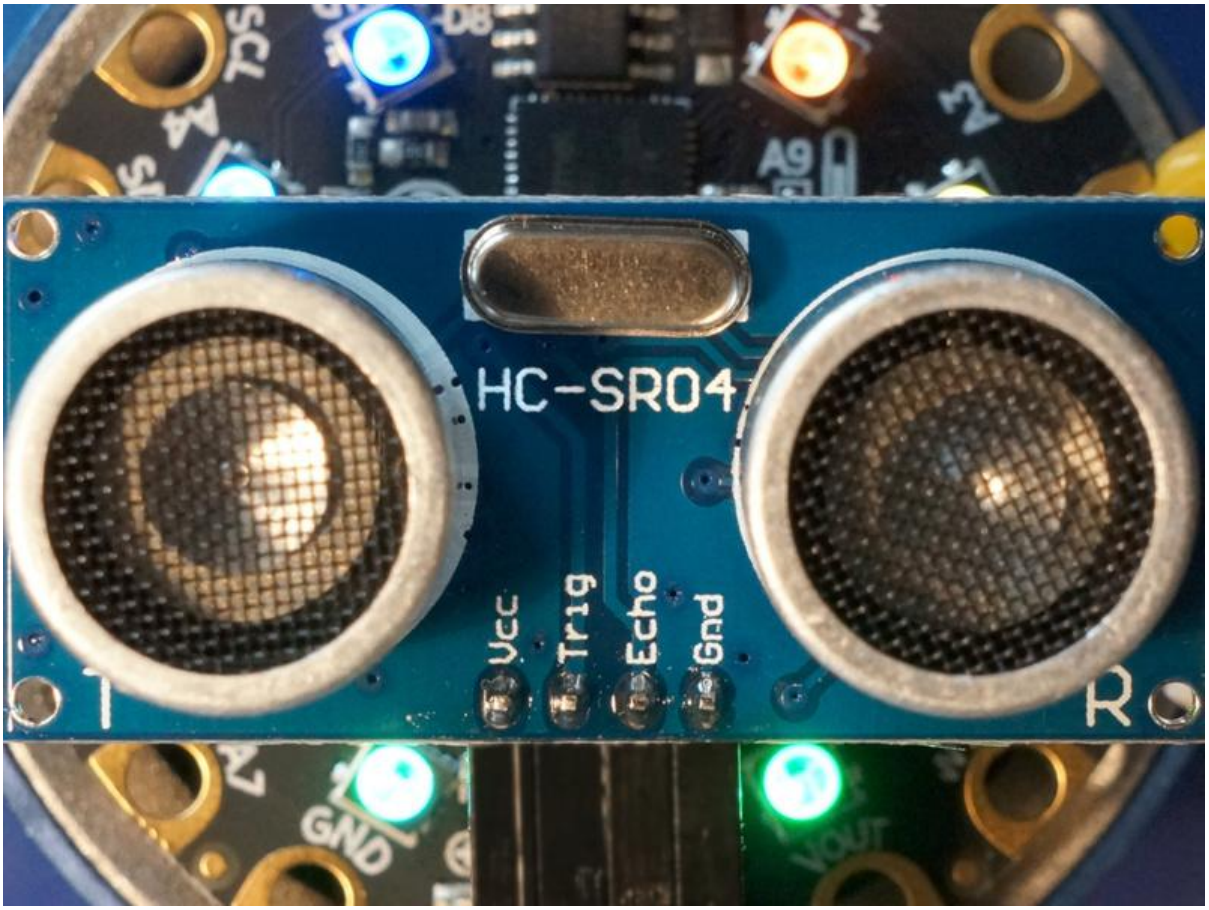
Distance Measurement with Ultrasound

Created by Kevin Walters



https://learn.adafruit.com/distance-measurement-ultrasound-hcsr04

Last updated on 2021-11-15 07:21:34 PM EST

Table of Contents

Overview 3
- Parts 4

Connect the Sensor 4

Simple MakeCode 6
- Description and Discussion 8

More Advanced MakeCode 8

Speed of Sound 8
- Description and Discussion 9

Range Selection 10
- Description and Discussion 11

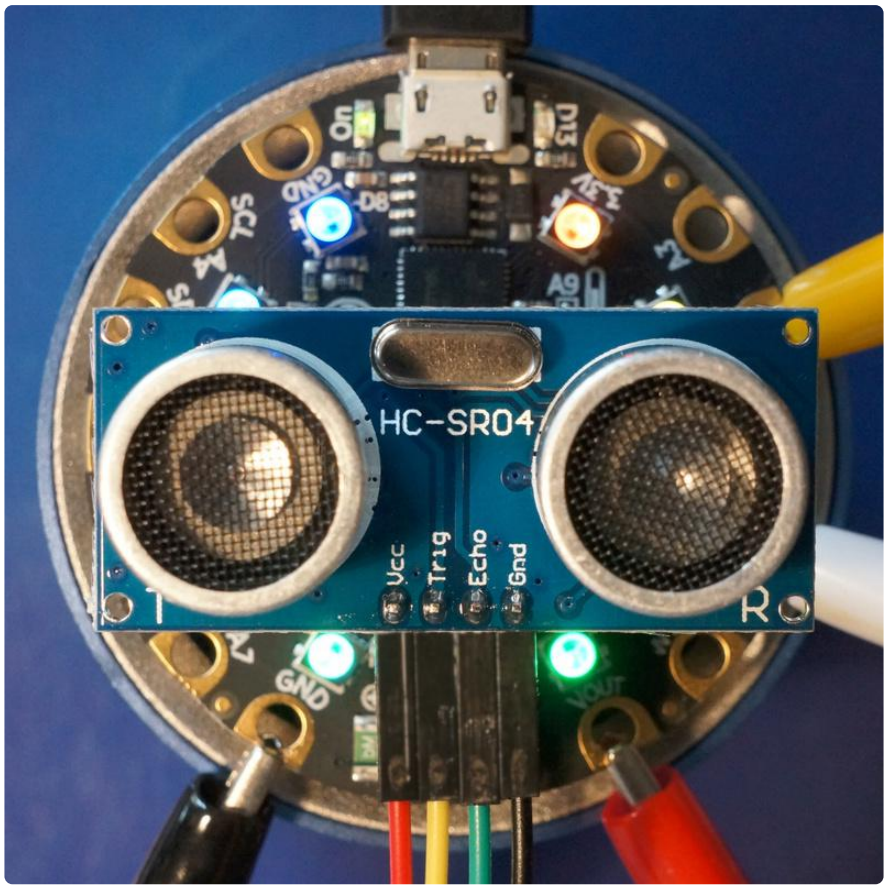
New Graph v1 12

New Graph v2 14
- Description and Discussion 16

The Ultrasound Pulses 17

Going Further 19
- Ideas for areas to explore 19
- Further Reading 19

Overview



The HC-SR04 is an inexpensive distance sensor based on a pair of ultrasonic transducers (https://adafru.it/CxB) with a straightforward TTL (https://adafru.it/CxC) level interface and a claimed range of 2cm to 4m. Similar sensors are now commonplace with the widespread use of car parking assistance sensors.

The sensor's 5V Echo output cannot be directly connected to the inputs of a board like the Circuit Playground Express (CPX) but some trivial electronics can remedy this incompatibility.

This project demonstrates distance measurement in Microsoft MakeCode (https://adafru.it/BUu) inspired by the BBC micro:bit example (https://adafru.it/CxD) starting with a very simple program and then showing more features of MakeCode and the Circuit Playground Express (CPX) board.

If you are new to using MakeCode, check out the guide to getting started (https://adafru.it/wWd) first.

Thank-you to George and Edward for the loan of their CPX board.

Parts

1 x Circuit Playground Express https://www.adafruit.com/product/3333
A great sensor-packed development board supporting many languages.

---

1 x HC-SR04 Ultrasound sensor https://www.adafruit.com/product/3942
HC-SR04 Ultrasound sensor with TTL 5V interface and 2x 10K Resistors for 3.3V use.

---

1 x Half-size Breadboard https://www.adafruit.com/product/64
A breadboard to place the sensor and resistors on.

---

1 x Premium Male/Male Jumper Wires - 40 x 6" (150mm) https://www.adafruit.com/product/758
Four jumper wires for breadboard connections.

---

1 x Small Alligator Clip to Male Jumper Wire Bundle - 6 Pieces https://www.adafruit.com/product/3448
Four alligator (crocodile) clips to connect to pads on CPX.

Adafruit now sell the US-100 Ultrasonic Distance Sensor which is directly compatible with 3.3V levels.

1 x US-100 Ultrasound Sensor https://www.adafruit.com/product/4019
US-100 Ultrasonic Distance Sensor - 3V or 5V Logic. Easier to use alternative, since it does not require output level conversion.

Connect the Sensor

The common HC-SR04 boards are designed for 5V TTL voltage levels. Since the advent of CMOS (https://adafru.it/CxE) many circuits started using 3.3V levels including the CPX board, this can create compatibility problems. The guide to CPX Pinouts (https://adafru.it/AM9) states:

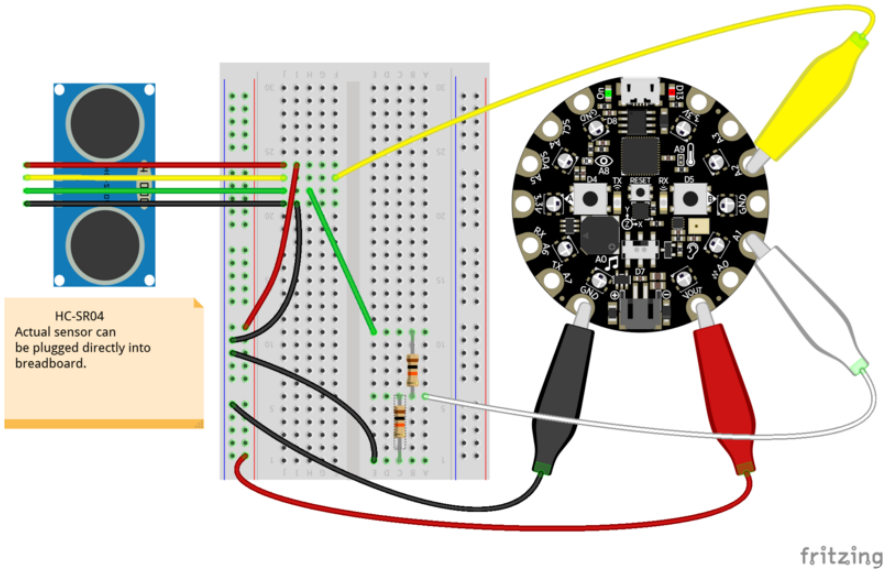
> All of the GPIO pads are 3.3V output level, and should not be used with 5V inputs. In general, most 5V devices are OK with 3.3V output though.

This creates a requirement to reduce the voltage from the Echo output of the HC-SR04 rather than simply directly connecting all of the sensor's pins to the CPX board.

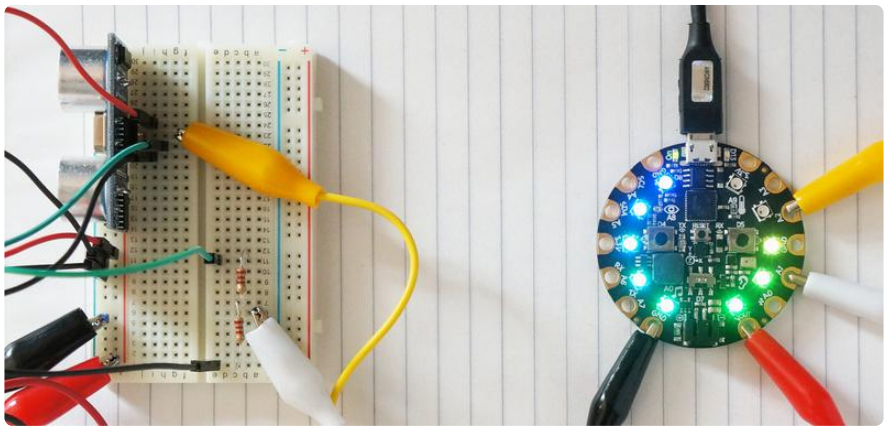
The easiest way to reduce a single voltage is to use two resistors as a potential divider (https://adafru.it/osc). The example shown uses two equal resistors (10 K ohms, they come with the HC-SR04 that Adafruit sells) to halve the voltage. 2.5V is clearly lower than 3.3V but this is high enough to work. The sum of the two resistors should be above 1 kiloohm to keep the current within the limits of the sensor's output.

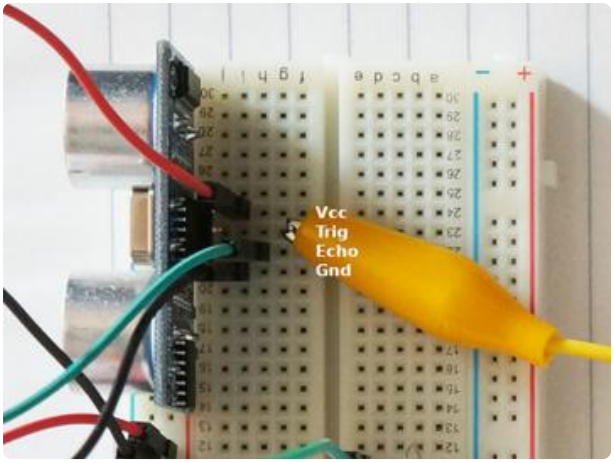
The slightly more complicated alternative to converting the voltages is to use a semiconductor solution like the unidirectional 74LVC245 (https://adafru.it/CxF) or 74AHCT125 (https://adafru.it/e5g) or the bidirectional TXB0104 (https://adafru.it/dre).

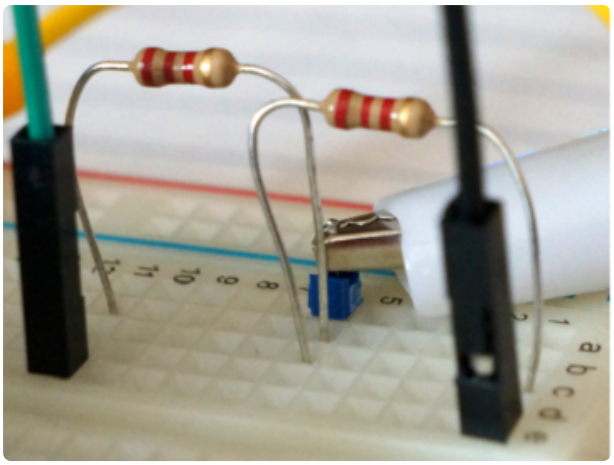
The diagram below shows how to connect the components together. The HC-SR04 can be plugged directly into the breadboard without the four wires shown in the diagram.



The picture below shows how it looks implemented on a breadboard. The only differ ences are: the sensor is plugged into the breadboard; the use of double-ended alligator (crocodile) clip leads with header pins aiding connection to the breadboard. The alligator clip to male jumper wire (https://adafru.it/CuB) is an easier alternative.







The connections are:

- Black: Gnd to GND via bus strip. Row 1 is also connected to ground.
- Red: Vcc to VOUT via bus strip.
- Green: Echo to potential divider (a pair of 10k resistors across).
- White: divided voltage (row 6) to A1.
- Yellow: Trig to A2.

Note: Pictures show two 2.2K resistors, Adafruit supplies two 10K resistors, both configurations function the same).

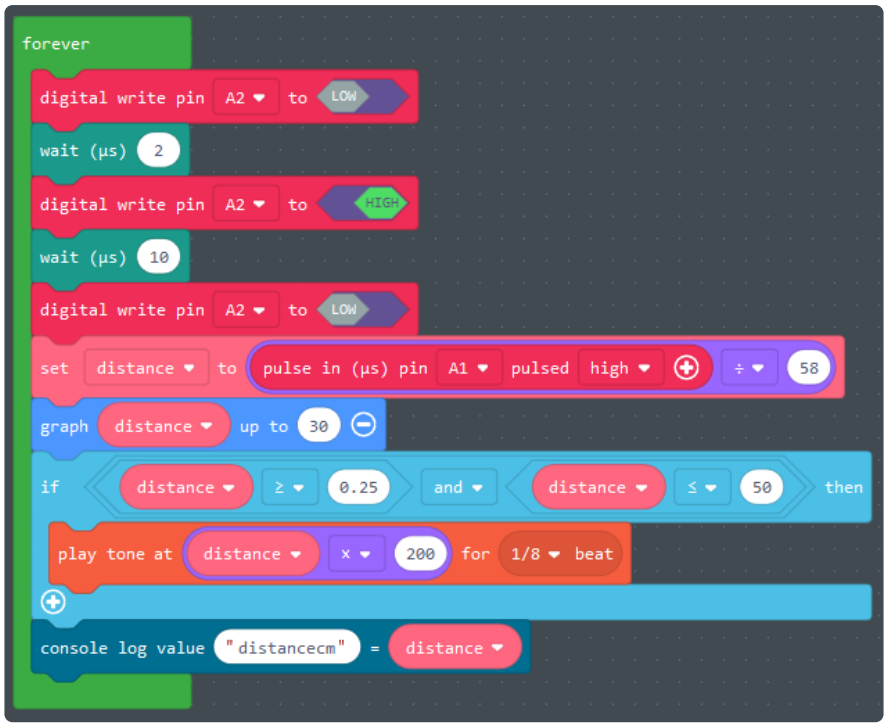
The A0 input can be used as an output for the sensor's Trig pin if you want the surprise of hearing the trigger pulses. The CPX board has A0 hard wired up to its small speaker.

For comparison: the Raspberry Pi has the same 3.3V limitation on GPIO inputs; many of the Arduino boards like the Uno are 5V tolerant for inputs, allowing direct connection (https://adafru.it/C7Y) of this sensor.

The standard HC-SR04 will appear to work at 3.3V but apparently is far less accurate (https://adafru.it/CxG). There's a detailed discussion on some of the variations of this sensor on David Pilling's HC-SR04 page (https://adafru.it/CxI). Adafruit now sell the US-100 Ultrasonic Distance Sensor (https://adafru.it/GqE) which is 3.3V compatible.

Simple MakeCode

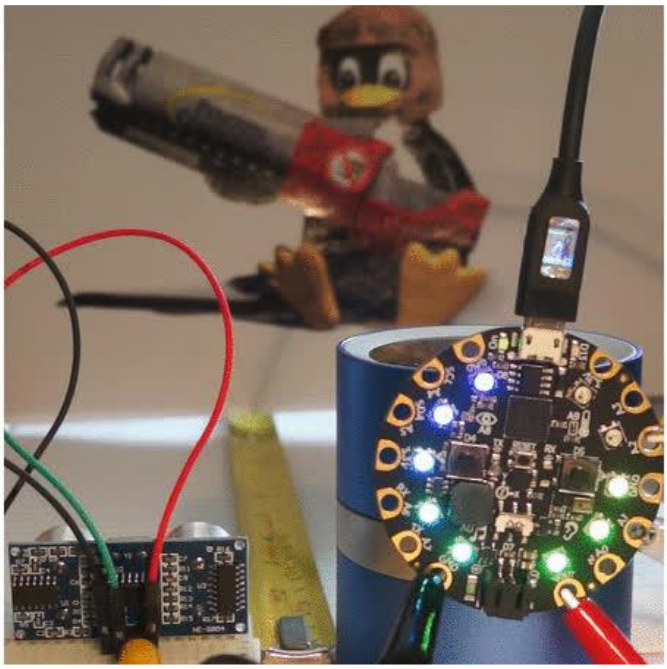
This example calculates the distance from the response from the sensor, graphs it on the NeoPixels, beeps with a pitch based on distance and logs the value.



Open this example in MakeCode

https://adafru.it/CxJ

The video (animated gif) below shows the sensor measuring the distance to a moving piece of paper and indicating this via the number of NeoPixels illuminated. The paper doesn't move far enough away to light up the tenth NeoPixel. There is a brief reading over the expected value, this could be due to the first reflection being missed and the sensor picking up sound from the the third reflection.



Description and Discussion

The code is a simple loop sending a pulse to A1 output pin and then measuring the pulse on A2 input pin with the MakeCode pulse in pin block. The division by 58 converts the timed value into centimetres and this value is then shown on the NeoPixels up to a value of 30cm (12"). The distance is also represented by a short beep with frequency (pitch) between 50Hz (for 0.25cm) and 10kHz (for 50cm). The distance is also logged but this is only visible if you are using the Windows 10 app version of MakeCode (https://adafru.it/CxK).

It's not clear if this sensor has a maximum measuring rate or a required pause after the Echo pulse before the next measurement. If it does then the code has a minor flaw as it's relying on the console log as a delay if the distance is out of range for the beep to occur.

The value of 58 or 58.2 is often seen in example code. It's worth exploring where this "magic" number comes from. The Echo high pulse duration represents the time for the first detected reflection. The speed of sound (https://adafru.it/CxL) varies mainly with temperature and to a far lesser degree by humidity. The chirp travels to the target and is reflected back, its journey is twice as long as the distance. 58 represents an assumption that the speed of sound is 1000000 / 58 * 2 / 100 = 344.8 metres per second. 344.8m/s would equate to a 21.6 degrees Celsius (70.8 Fahrenheit), 40% relative humidity (https://adafru.it/CxM) day.

The graph block (https://adafru.it/CxN) logs the distance value to the console as an unnamed value making the use of console log value (https://adafru.it/CxO) block a little superfluous.

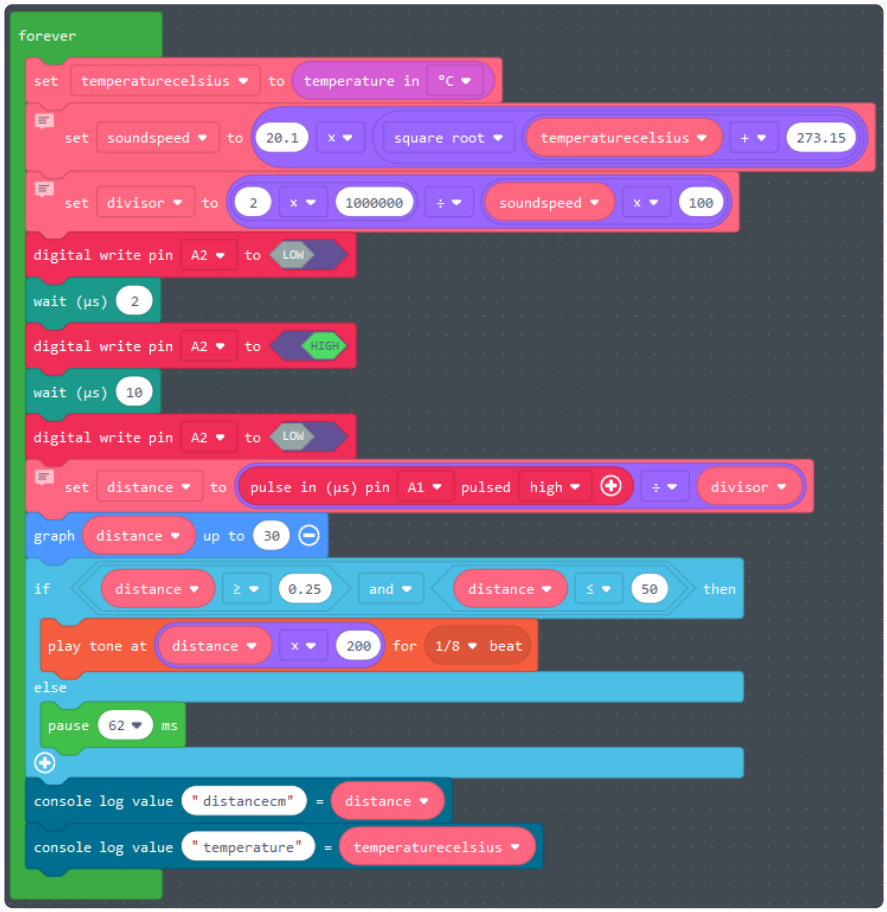
More Advanced MakeCode

These are some more advanced examples showing how to use the temperature from the on-board thermistor, how to use the buttons to add extra functionality and how to implement your own graph on the NeoPixels.

Speed of Sound

This example is the same as the first simple example but with added code to calculate the speed of sound based on the temperature from the CPX's thermistor (https://

adafru.it/CxP). That temperature is then used to calculate the divisor for distance calculations.



Open this example in MakeCode

https://adafru.it/CxQ

Description and Discussion

This code introduces a new variable to store the temperature read from the thermistor in degrees Celsius. The speed of sound is then calculated using a formula. The number 273.15 may be familiar - it's being used to convert the temperature to Kelvin (https://adafru.it/xxc).

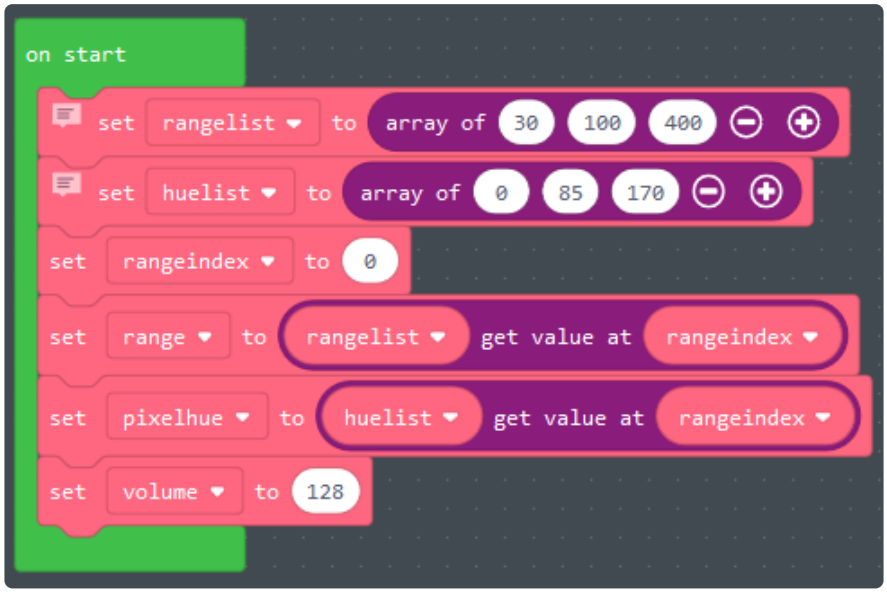
The potential flaw from the original code is fixed with the addition of a 62 millisecond delay when the distance is out of range for a beep. At 120bpm, a demisemiquaver (1/32 note) (https://adafru.it/CDZ) is 4 * 60 / 120 / 32 * 1000 = 62.5ms.

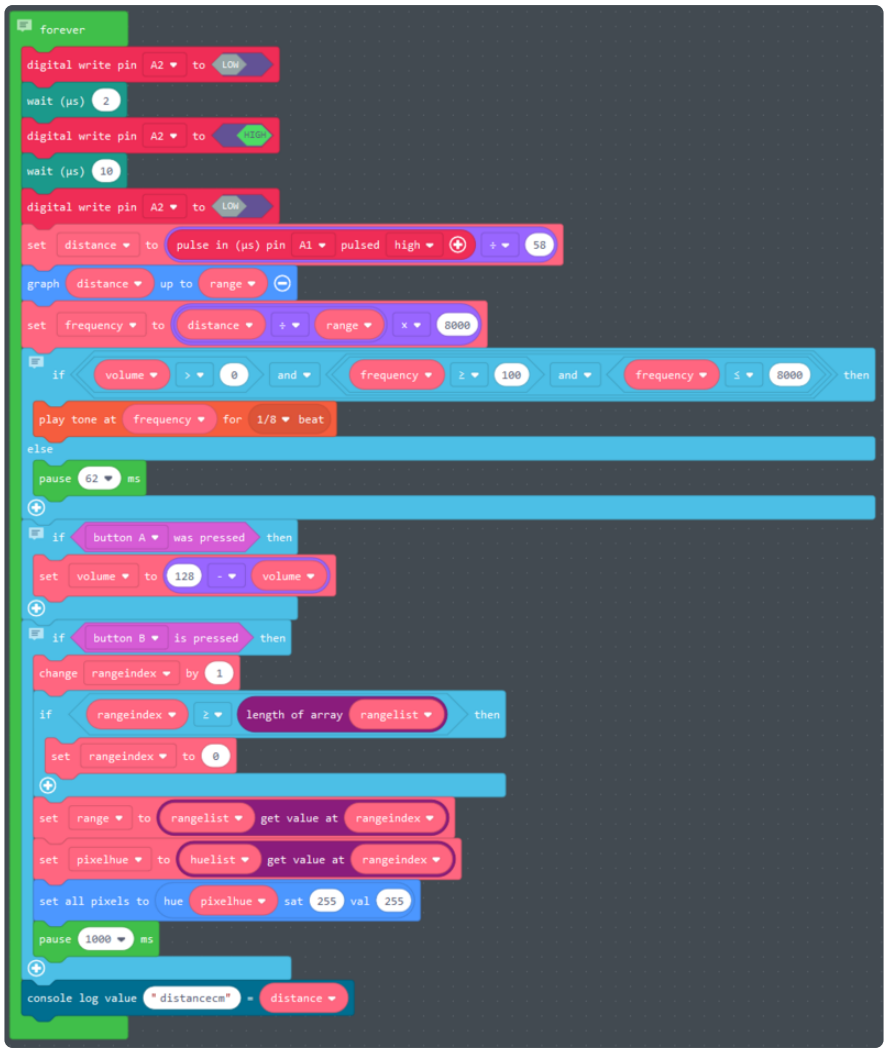
The temperature is also logged to allow it to inspected to ensure the value is reasonable and accurate.

A good question is whether the temperature needs to be calculated every time a new distance is measured. As temperature does not typically change much this could be performed once. A single calculation would be problematic if the code was expected to run for a long time or the temperature was expected to change rapidly. At the cost of increased complexity of code, one compromise solution would be to calculate it periodically with the period based on the likely rate of change. Sample rate decisions like this become more relevant when the calculation is more intensive and there are limited resources.

Range Selection

This example is the same as the first simple example but with added code to allow the left button to mute the beeping and the right button to change the maximum range. The on start (https://adafru.it/CxS) block is introduced to initialise some variables include two arrays (https://adafru.it/CxT).





Open this example in MakeCode

https://adafru.it/CxU

Description and Discussion

The three maximum range distances are stored in the `rangelist` array with some associated colours in `huelist` (0 is red, 85 is green, 170 is blue in this HSV model (https://adafru.it/CxV)). `rangeindex` is used to select the current range from `rangelist`. The first element in an array in MakeCode is 0, in other languages this may be 1.

The loop is similar to the first example but with the addition of two `if` blocks to deal with buttons `A` (left) and `B` (right) and a more sophisticated condition for beeping which now uses the same range as graphing and constrains the frequency between 100Hz and 8kHz.

The check for `button A` uses a common subtraction technique to toggle between the two volume values, 0 and 128. `button B` looks more complicated but is only

adding 1 (incrementing) the `rangeindex` and then returning to the first element if it has gone past the end of the array. The code flashes all the NeoPixels the `pixelhue` colour for 1 second to indicate to the user which range has been selected, e.g. green for 100cm.

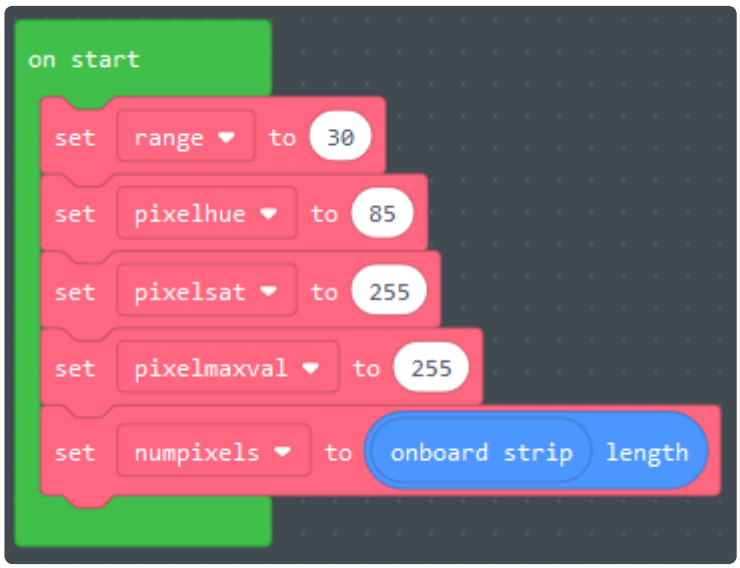
There is a subtle but important difference between the code for the two buttons. The first uses was pressed (https://adafru.it/CxW) and the second uses is pressed (https://adafru.it/CxX). `was pressed` checks for any press since the last check but `is pressed` only checks as the block is executed. In this case the code before the `if` blocks executes quickly hence everything loops frequently allowing the use of the instantaneous `is pressed`. Holding down either button for a few seconds will reveal a difference in behaviour relating to this.

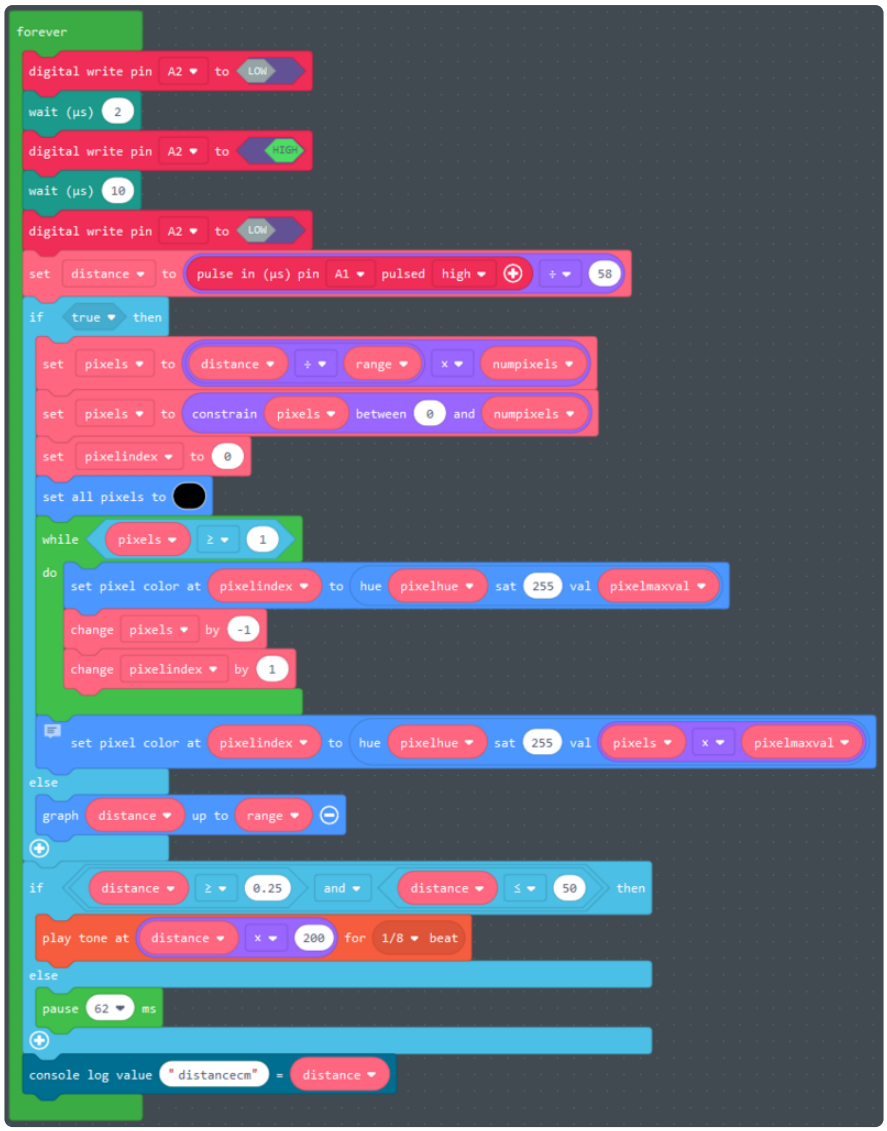
MakeCode also has an on (https://adafru.it/CxY) block which is an alternative method for coding for buttons. The volume mute could also have been implemented using a boolean (https://adafru.it/CxZ) value and controlled using set volume (https://adafru.it/Cx-) although the CPX default level sounds like 20 rather than 128. It is common in computer languages for there to be more than one way to achieve the same goal.

The set volume block does not work on all browsers in the simulator.

New Graph v1

This example is the same as the first simple example but with the native graph (https://adafru.it/CxN) block replaced by a new one with a higher precision representation on the NeoPixels.

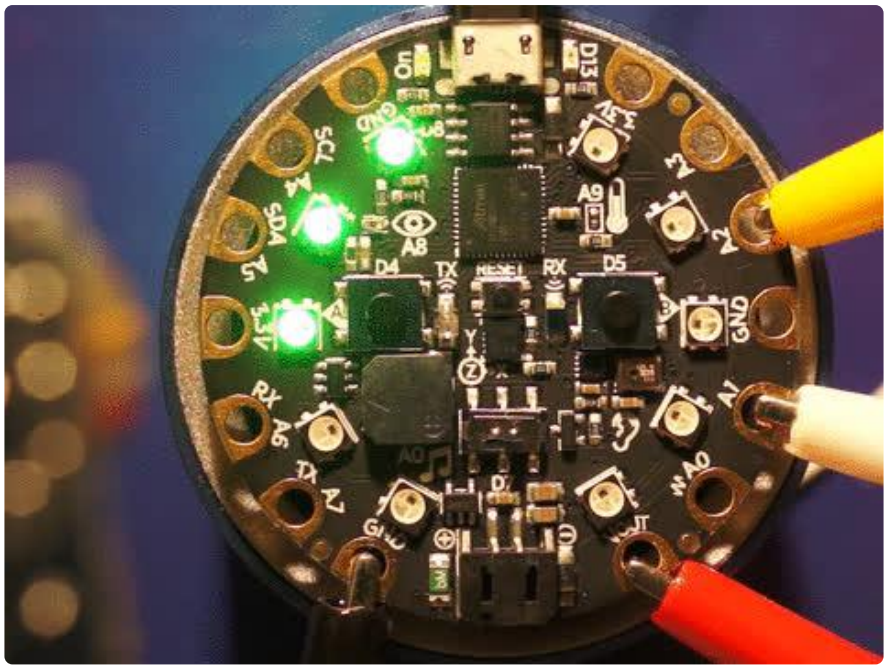




Open this example in MakeCode

https://adafru.it/Cy0

The new graphing code calculates the value in terms of number of pixels but keeps the remainder from the calculation and varies the brightness of the final pixel based on that remainder to show values "between" the NeoPixels. The video below demonstrates the sensor being moved (just visible on the left) to indicate how the NeoPixels represent the distance values.

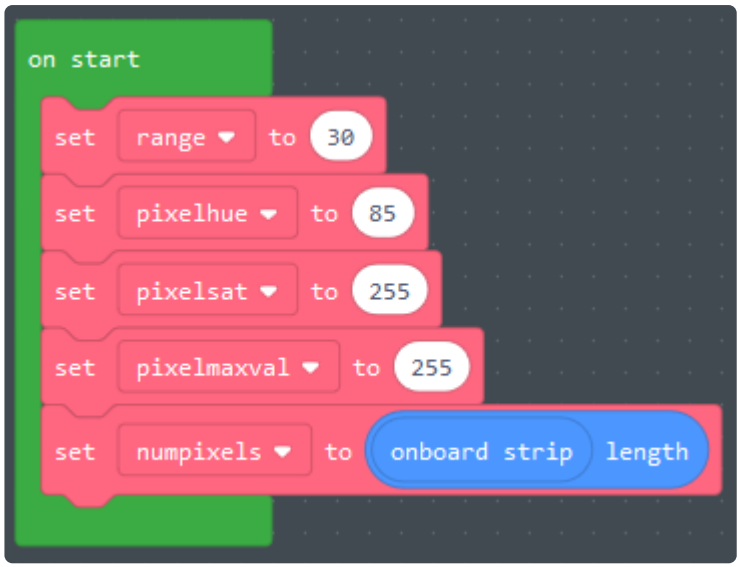


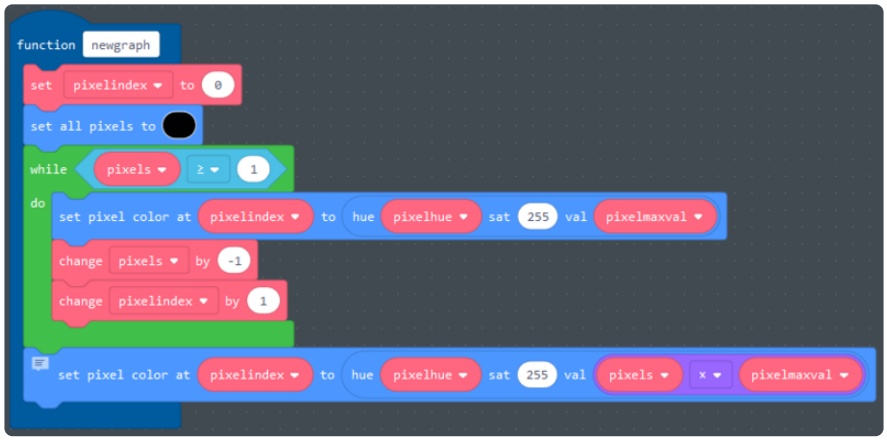
The code has an `if` block with a `true` value. This looks a little odd but allows the programmer to switch between the two versions of the graphing for comparison. This could be removed if the programmer has made a decision to only use one type of graphing.

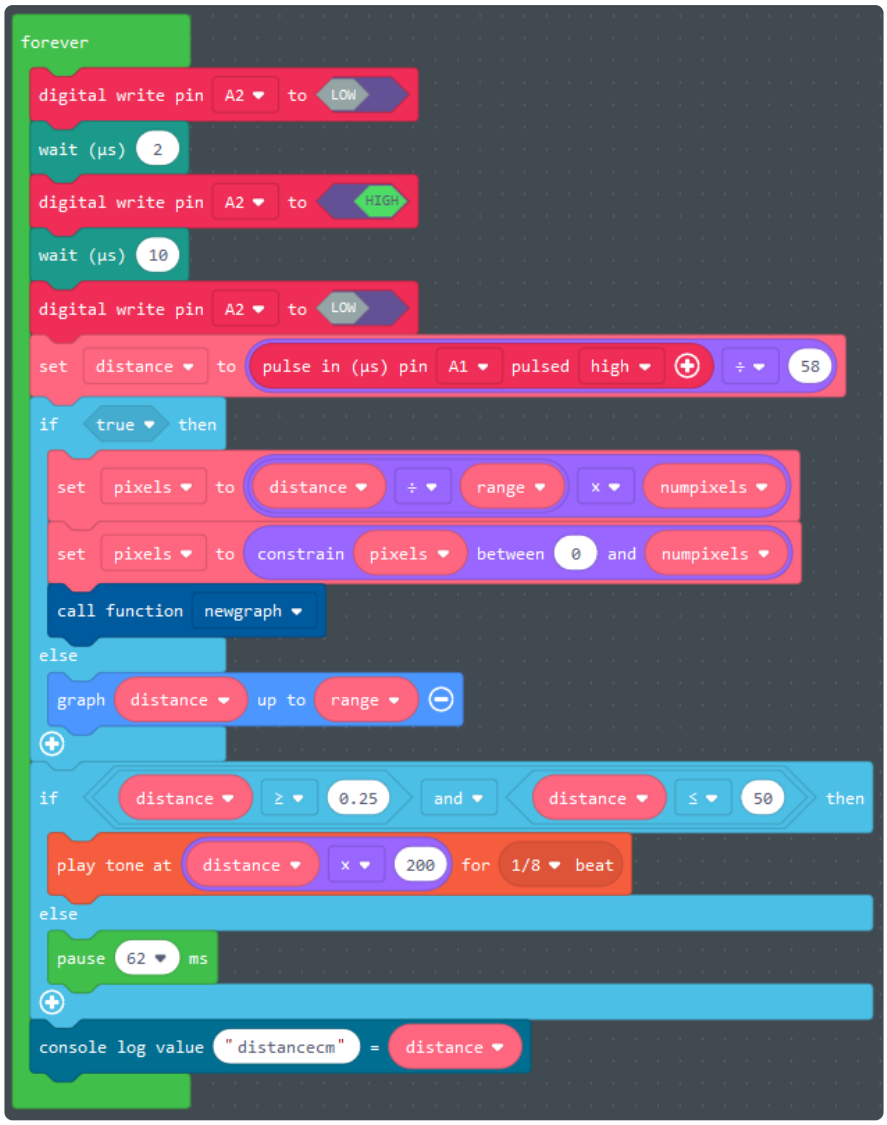
The NeoPixels visibly flicker when many of them are lit. The code sets them all to black and then sets each one that needs to be on or partially on. There are more efficient approaches to this. A key change is to use the `set buffered` and `show` blocks to apply the changes in one go when they are all done.

New Graph v2

This example is the same as the first graphing version (https://adafru.it/Cy1) but places the graphing code in a function (https://adafru.it/Cy2) called `newgraph`.


on start
set range to 30
set pixelhue to 85
set pixelsat to 255
set pixelmaxval to 255
set numpixels to onboard strip length


function newgraph
set pixelindex to 0
set all pixels to
while pixels ≥ 1
do
set pixel color at pixelindex to hue pixelhue sat 255 val pixelmaxval
change pixels by -1
change pixelindex by 1
set pixel color at pixelindex to hue pixelhue sat 255 val pixels x pixelmaxval



Open this example in MakeCode

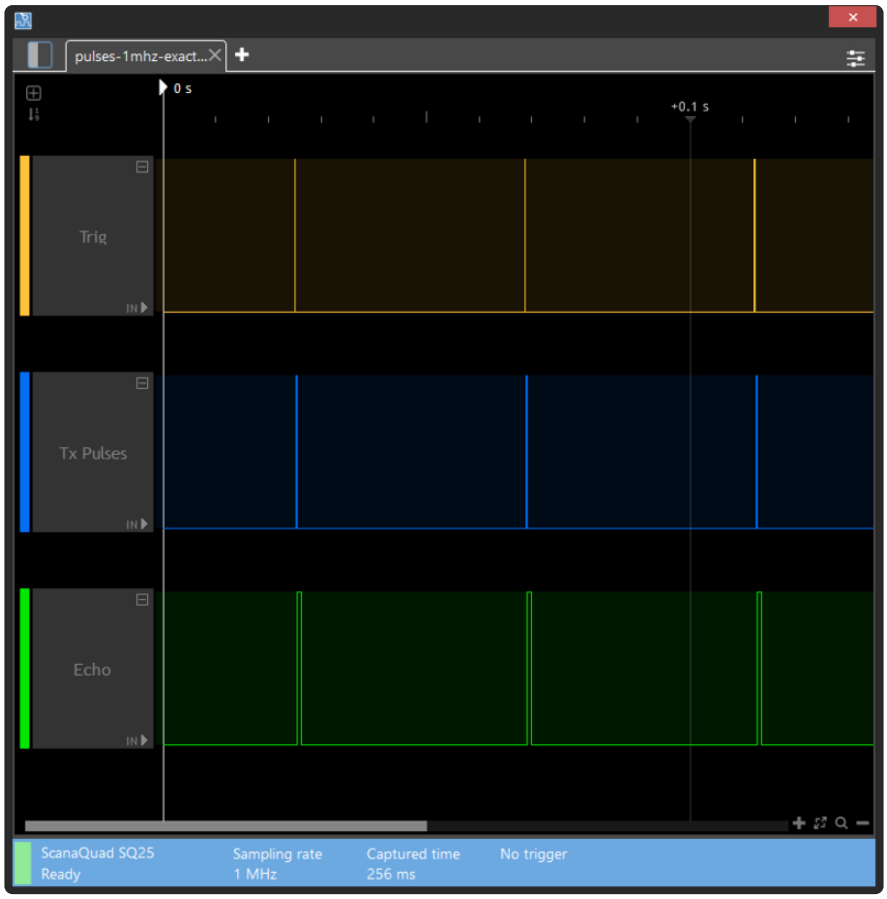
https://adafru.it/Cy3

Description and Discussion

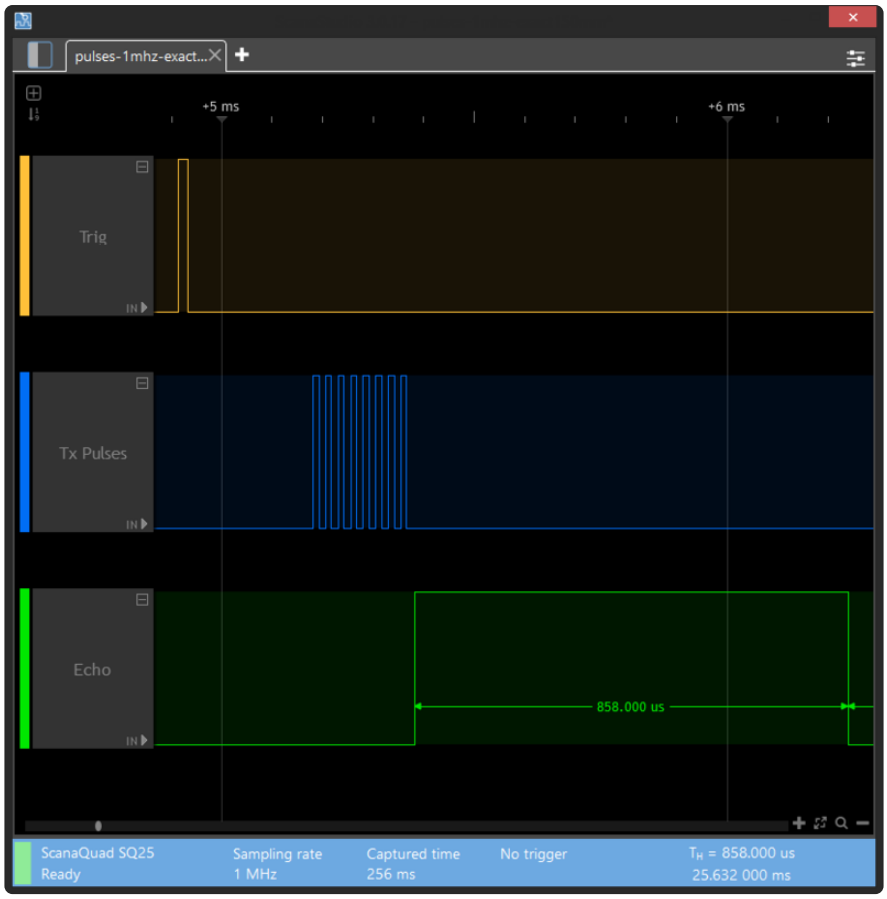
In this particular case the use of a `function` does not make a huge amount of difference. It makes the `forever` loop more compact and a little easier to read. If all of the code from the previous examples were combined then the `forever` loop would become unwieldy. If the code is difficult to understand and review then bugs (https://adafru.it/Cy4) are more likely to creep in.

Functions become more useful in a program when they are used (called) multiple times as they reduce code duplication. They can also be the first step towards sharing code between different programs.

The Ultrasound Pulses



A logic analyser can be used to look at the communication between the CPX board and sensor. It can also be connected to the ultrasonic transmitter to see the high frequency chirp but won't work on the unamplified receiver output. A basic logic analyser cannot show the full detail of the analogue voltages as it makes everything appear rectangular. There are some examples of what the signal really looks like on David Pilling's HC-SR04 page (https://adafru.it/CxI).



The screenshot at the top of the page shows three distance measurements. The screenshot just above is one of the measurements zoomed in showing:

- Trig (yellow): the pulse requesting the sensor to make a measurement,
- Tx Pulses (blue): the actual burst of eight pulses which constitute the ultrasonic chirp,
- Echo (green): the sensor's output measured by the logic analyser as 858 microseconds.

The 858 microseconds divided by 58 gives 14.8cm. The target was placed at 15.0cm which suggests this can be very accurate. The voltage thresholds for low and high on the logic analyser may differ by a tiny amount compared to the CPX board.

The TX Pulses can be used to determine the audio frequency. Using a higher resolution trace made at 25MHz, measuring seven pulses from the leading edge for best accuracy gives 173.24 microseconds, 7 / 173.24 * 1000 = 40.41kHz, clearly in the ultrasound range.

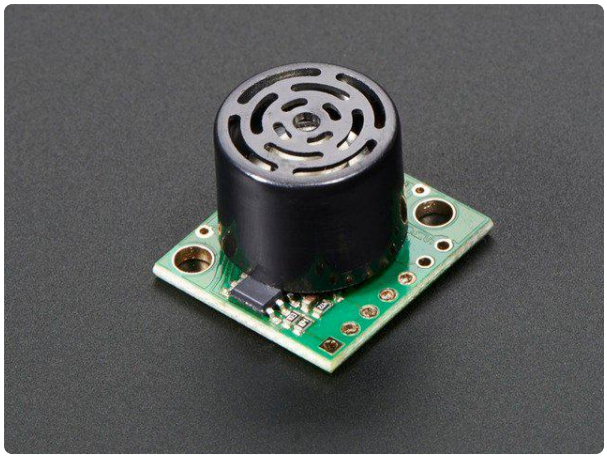
Going Further

Ideas for areas to explore

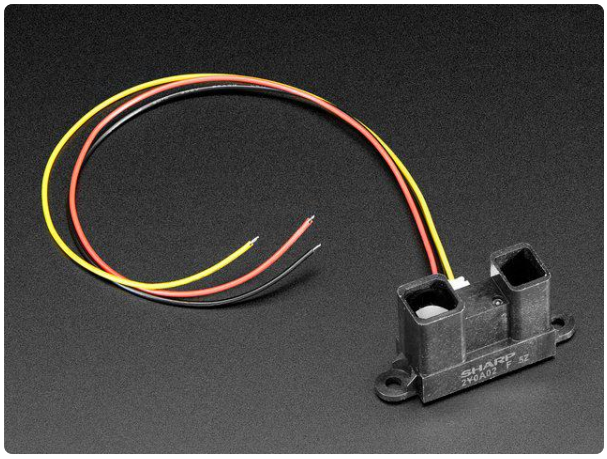
- Finding something fun to measure!
- Seeing if your pet can hear the sensor, some animals can hear ultrasound.
- Checking the maximum range for different objects and the behaviour of the sensor for no reflection ("infinity").
- Checking and explaining the behaviour for very close objects.
- Adding calibration offsets and coefficients to the program based on testing with one or more sensors.
- Determining if sensors can be confused by other sensors operating in the same space.
- Measuring the cone angle for the sensor and exploring physical techniques for reducing this.
- Testing if fast movement affects sensor's ability to measure distance.
- Exploring techniques for enhancing accuracy for stationary objects.
- Checking the behaviour of the standard HC-SR04 at 3.3V.
- Testing the 3.3V tolerant HC-SR04 variants. These may be red and labelled as "1652".
- Explore other similar sensors like the (single transducer) Maxbotix range and ones that use different techniques:
    - SHARP sensors with infrared arrays,
    - STMicroelectronics Time of Flight (ToF) sensors.

Further Reading

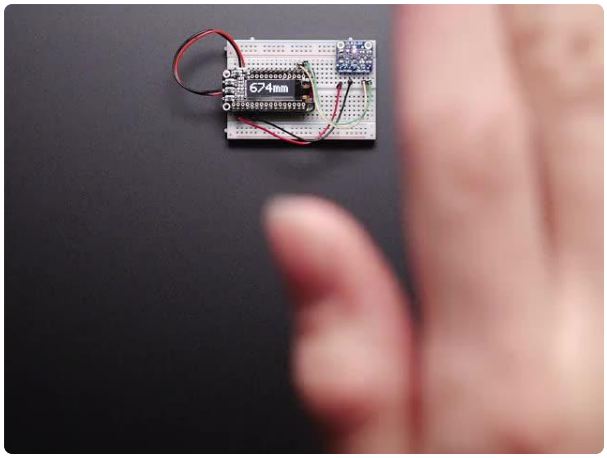
- Element 14: The Learning Circuit 85: How Do Ultrasonic Distance Sensors Work? (https://adafru.it/Qfo) (video)



[Maxbotix Ultrasonic Rangefinder - LV-EZ1]
LV-EZ1 Maxbotix Ultrasonic Rangefinder provides very short to long-range detection and ranging, in an incredibly small package. It can detect objects from 0-inches to 254-inches...
https://www.adafruit.com/product/172



[IR distance sensor includes cable (20cm-150cm)]
This SHARP distance sensor bounces IR off objects to determine how far away they are. It returns an analog voltage that can be used to determine how close the nearest object is. Comes...
https://www.adafruit.com/product/1031



[Adafruit VL53L0X Time of Flight Distance Sensor - ~30 to 1000mm]
The VL53L0X is a Time of Flight distance sensor like no other you've used! The sensor contains a very tiny invisible laser source, and a...
https://www.adafruit.com/product/3317