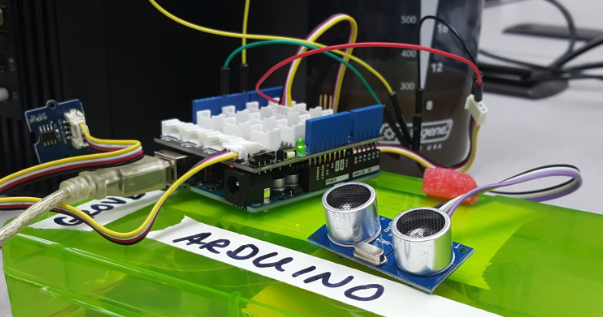
Sarah Wood

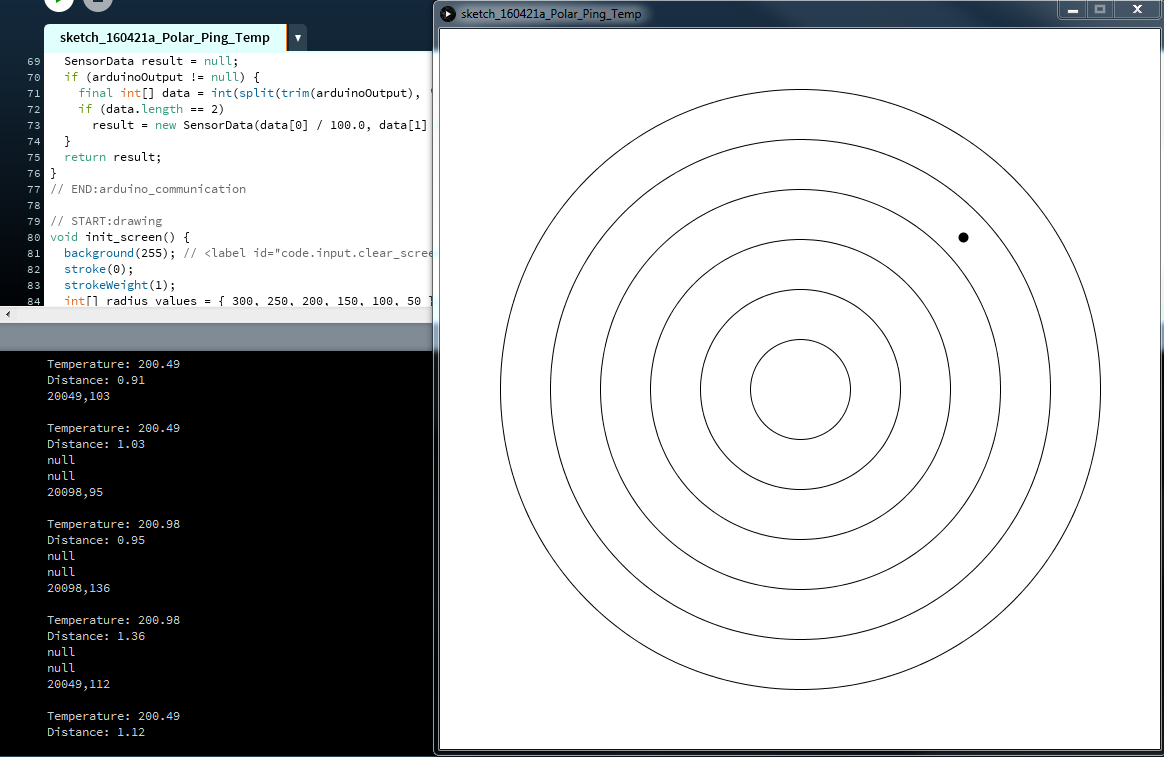
Embedded Programming

C5-A1

Ping & “Radar”

Screen Caps





I couldn’t figure out why my Processing visualizer wasn’t working. I added “print line” code to it to verify serial I/O. That’s what is shown in the console output. Then I had my Eureka moment. I saw the black dot twitching – I thought I was seeing things. Then in the debugger, the polar Y value was changing, but not enough. I adjusted the scale of my graph, and Hallelujah – it worked!

Source Code

The Arduino sensor code is borrowed under GNU license from Tim Eckel. His info is in “NewPing.h”. I have annotated my adjustments.

The Processing code is from the book library, but modified.

Main

/\*

\* Sarah Wood

\* Embedded Programming

\* Chapter 5

\*

\* This is an amalgam of the code from the book, and a program written by

\* Tim Eckel, which I have used under the terms of the GNU license. Tim's original "rap sheet"

\* is included unaltered in "NewPing.h".

\*

\* Thank you, Tim.

\*

\* In places where I have substantially modified Tim's program, I have labelled

\* the changes with my initials: SJW .

\*

\*

\*

\*/

// ---------------------------------------------------------------------------

// Example NewPing library sketch that does a ping about 20 times per second.

// ---------------------------------------------------------------------------

// SJW: I modified this. I wanted to try different delay settings.

#include "NewPing.h"

#define TRIGGER\_PIN 12 // Arduino pin tied to trigger pin on the ultrasonic sensor.

#define ECHO\_PIN 8 // Arduino pin tied to echo pin on the ultrasonic sensor.

#define MAX\_DISTANCE 500 // Maximum distance we want to ping for (in centimeters). Maximum sensor distance is rated at 400-500cm.

NewPing sonar(TRIGGER\_PIN, ECHO\_PIN, MAX\_DISTANCE); // NewPing setup of pins and maximum distance.

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

// SJW: Temperature sensor support

//&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&

const unsigned int TEMP\_SENSOR\_PIN = 0;

float current\_temperature = 0.0;

unsigned long last\_measurement = millis();

const float SUPPLY\_VOLTAGE = 5.0;

void setup() {

Serial.begin(9600); // Open serial monitor at 115200 baud to see ping results.

//SJW: Adjusted to comply with my hardware.

}

void loop() {

//SJW: Temp support added

unsigned long current\_millis = millis();

if (abs(current\_millis - last\_measurement) >= 1000) {

current\_temperature = get\_temperature();

last\_measurement = current\_millis;

}

delay(500); // Wait 500ms between pings (about 2 pings/sec). 29ms should be the shortest delay between pings.

Serial.print(scaled\_value(current\_temperature));

Serial.print(",");

Serial.print(sonar.ping\_cm()); // Send ping, get distance in cm and print result (0 = outside set distance range)

Serial.println("");

}

//SJW: Temp support functions

long scaled\_value(const float value) {

float round\_offset = value < 0 ? -0.5 : 0.5;

return (long)(value \* 100 + round\_offset);

}

const float get\_temperature() {

const int sensor\_voltage = analogRead(TEMP\_SENSOR\_PIN);

const float voltage = sensor\_voltage \* SUPPLY\_VOLTAGE / 1024;

return (voltage \* 1000 - 500) / 10;

}

NewPing.h

// ---------------------------------------------------------------------------

// NewPing Library - v1.7 - 09/29/2015

//

// AUTHOR/LICENSE:

// Created by Tim Eckel - teckel@leethost.com

// Copyright 2015 License: GNU GPL v3 http://www.gnu.org/licenses/gpl.html

//

// LINKS:

// Project home: https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home

// Blog: http://arduino.cc/forum/index.php/topic,106043.0.html

//

// DISCLAIMER:

// This software is furnished "as is", without technical support, and with no

// warranty, express or implied, as to its usefulness for any purpose.

//

// BACKGROUND:

// When I first received an ultrasonic sensor I was not happy with how poorly

// it worked. Quickly I realized the problem wasn't the sensor, it was the

// available ping and ultrasonic libraries causing the problem. The NewPing

// library totally fixes these problems, adds many new features, and breaths

// new life into these very affordable distance sensors.

//

// FEATURES:

// \* Works with many different ultrasonic sensors: SR04, SRF05, SRF06, DYP-ME007, URM37 & Parallax PING)))™.

// \* Compatible with the entire Arduino line-up, Teensy family (including $19 96Mhz 32 bit Teensy 3.0) and derivatives.

// \* Interface with all but the SRF06 sensor using only one Arduino pin.

// \* Doesn't lag for a full second if no ping/echo is received.

// \* Ping sensors consistently and reliably at up to 30 times per second.

// \* Timer interrupt method for event-driven sketches.

// \* Built-in digital filter method ping\_median() for easy error correction.

// \* Uses port registers for a faster pin interface and smaller code size.

// \* Allows you to set a maximum distance where pings beyond that distance are read as no ping "clear".

// \* Ease of using multiple sensors (example sketch with 15 sensors).

// \* More accurate distance calculation (cm, inches & uS).

// \* Doesn't use pulseIn, which is slow and gives incorrect results with some ultrasonic sensor models.

// \* Actively developed with features being added and bugs/issues addressed.

//

// CONSTRUCTOR:

// NewPing sonar(trigger\_pin, echo\_pin [, max\_cm\_distance])

// trigger\_pin & echo\_pin - Arduino pins connected to sensor trigger and echo.

// NOTE: To use the same Arduino pin for trigger and echo, specify the same pin for both values.

// max\_cm\_distance - [Optional] Maximum distance you wish to sense. Default=500cm.

//

// SYNTAX:

// sonar.ping() - Send a ping and get the echo time (in microseconds) as a result.

// sonar.ping\_in() - Send a ping and get the distance in whole inches.

// sonar.ping\_cm() - Send a ping and get the distance in whole centimeters.

// sonar.ping\_median(iterations) - Do multiple pings (default=5), discard out of range pings and return median in microseconds.

// sonar.convert\_in(echoTime) - Convert echoTime from microseconds to inches (rounds to nearest inch).

// sonar.convert\_cm(echoTime) - Convert echoTime from microseconds to centimeters (rounds to nearest cm).

// sonar.ping\_timer(function) - Send a ping and call function to test if ping is complete.

// sonar.check\_timer() - Check if ping has returned within the set distance limit.

// NewPing::timer\_us(frequency, function) - Call function every frequency microseconds.

// NewPing::timer\_ms(frequency, function) - Call function every frequency milliseconds.

// NewPing::timer\_stop() - Stop the timer.

//

// HISTORY:

// 09/29/2015 v1.7 - Removed support for the Arduino Due and Zero because

// they're both 3.3 volt boards and are not 5 volt tolerant while the HC-SR04

// is a 5 volt sensor. Also, the Due and Zero don't support pin manipulation

// compatibility via port registers which can be done (see the Teensy 3.2).

//

// 06/17/2014 v1.6 - Corrected delay between pings when using ping\_median()

// method. Added support for the URM37 sensor (must change URM37\_ENABLED from

// false to true). Added support for Arduino microcontrollers like the $20

// 32 bit ARM Cortex-M4 based Teensy 3.2. Added automatic support for the

// Atmel ATtiny family of microcontrollers. Added timer support for the

// ATmega8 microcontroller. Rounding disabled by default, reduces compiled

// code size (can be turned on with ROUNDING\_ENABLED switch). Added

// TIMER\_ENABLED switch to get around compile-time "\_\_vector\_7" errors when

// using the Tone library, or you can use the toneAC, NewTone or

// TimerFreeTone libraries: https://bitbucket.org/teckel12/arduino-toneac/

// Other speed and compiled size optimizations.

//

// 08/15/2012 v1.5 - Added ping\_median() method which does a user specified

// number of pings (default=5) and returns the median ping in microseconds

// (out of range pings ignored). This is a very effective digital filter.

// Optimized for smaller compiled size (even smaller than sketches that

// don't use a library).

//

// 07/14/2012 v1.4 - Added support for the Parallax PING)))™ sensor. Interface

// with all but the SRF06 sensor using only one Arduino pin. You can also

// interface with the SRF06 using one pin if you install a 0.1uf capacitor

// on the trigger and echo pins of the sensor then tie the trigger pin to

// the Arduino pin (doesn't work with Teensy). To use the same Arduino pin

// for trigger and echo, specify the same pin for both values. Various bug

// fixes.

//

// 06/08/2012 v1.3 - Big feature addition, event-driven ping! Uses Timer2

// interrupt, so be mindful of PWM or timing conflicts messing with Timer2

// may cause (namely PWM on pins 3 & 11 on Arduino, PWM on pins 9 and 10 on

// Mega, and Tone library). Simple to use timer interrupt functions you can

// use in your sketches totally unrelated to ultrasonic sensors (don't use if

// you're also using NewPing's ping\_timer because both use Timer2 interrupts).

// Loop counting ping method deleted in favor of timing ping method after

// inconsistent results kept surfacing with the loop timing ping method.

// Conversion to cm and inches now rounds to the nearest cm or inch. Code

// optimized to save program space and fixed a couple minor bugs here and

// there. Many new comments added as well as line spacing to group code

// sections for better source readability.

//

// 05/25/2012 v1.2 - Lots of code clean-up thanks to Arduino Forum members.

// Rebuilt the ping timing code from scratch, ditched the pulseIn code as it

// doesn't give correct results (at least with ping sensors). The NewPing

// library is now VERY accurate and the code was simplified as a bonus.

// Smaller and faster code as well. Fixed some issues with very close ping

// results when converting to inches. All functions now return 0 only when

// there's no ping echo (out of range) and a positive value for a successful

// ping. This can effectively be used to detect if something is out of range

// or in-range and at what distance. Now compatible with Arduino 0023.

//

// 05/16/2012 v1.1 - Changed all I/O functions to use low-level port registers

// for ultra-fast and lean code (saves from 174 to 394 bytes). Tested on both

// the Arduino Uno and Teensy 2.0 but should work on all Arduino-based

// platforms because it calls standard functions to retrieve port registers

// and bit masks. Also made a couple minor fixes to defines.

//

// 05/15/2012 v1.0 - Initial release.

// ---------------------------------------------------------------------------

#ifndef NewPing\_h

#define NewPing\_h

#if defined (ARDUINO) && ARDUINO >= 100

#include <Arduino.h>

#else

#include <WProgram.h>

#include <pins\_arduino.h>

#endif

#if defined (\_\_AVR\_\_)

#include <avr/io.h>

#include <avr/interrupt.h>

#endif

// Shouldn't need to change these values unless you have a specific need to do so.

#define MAX\_SENSOR\_DISTANCE 500 // Maximum sensor distance can be as high as 500cm, no reason to wait for ping longer than sound takes to travel this distance and back. Default=500

#define US\_ROUNDTRIP\_CM 57 // Microseconds (uS) it takes sound to travel round-trip 1cm (2cm total), uses integer to save compiled code space. Default=57

#define US\_ROUNDTRIP\_IN 146 // Microseconds (uS) it takes sound to travel round-trip 1 inch (2 inches total), uses integer to save compiled code space. Defalult=146

#define ONE\_PIN\_ENABLED true // Set to "false" to disable one pin mode which saves around 14-26 bytes of binary size. Default=true

#define ROUNDING\_ENABLED false // Set to "true" to enable distance rounding which also adds 64 bytes to binary size. Default=false

#define URM37\_ENABLED false // Set to "true" to enable support for the URM37 sensor in PWM mode. Default=false

#define TIMER\_ENABLED true // Set to "false" to disable the timer ISR (if getting "\_\_vector\_7" compile errors set this to false). Default=true

// Probably shouldn't change these values unless you really know what you're doing.

#define NO\_ECHO 0 // Value returned if there's no ping echo within the specified MAX\_SENSOR\_DISTANCE or max\_cm\_distance. Default=0

#define MAX\_SENSOR\_DELAY 5800 // Maximum uS it takes for sensor to start the ping. Default=5800

#define ECHO\_TIMER\_FREQ 24 // Frequency to check for a ping echo (every 24uS is about 0.4cm accuracy). Default=24

#define PING\_MEDIAN\_DELAY 29000 // Microsecond delay between pings in the ping\_median method. Default=29000

#define PING\_OVERHEAD 5 // Ping overhead in microseconds (uS). Default=5

#define PING\_TIMER\_OVERHEAD 13 // Ping timer overhead in microseconds (uS). Default=13

#if URM37\_ENABLED == true

#undef US\_ROUNDTRIP\_CM

#undef US\_ROUNDTRIP\_IN

#define US\_ROUNDTRIP\_CM 50 // Every 50uS PWM signal is low indicates 1cm distance. Default=50

#define US\_ROUNDTRIP\_IN 127 // If 50uS is 1cm, 1 inch would be 127uS (50 x 2.54 = 127). Default=127

#endif

// Conversion from uS to distance (round result to nearest cm or inch).

#define NewPingConvert(echoTime, conversionFactor) (max(((unsigned int)echoTime + conversionFactor / 2) / conversionFactor, (echoTime ? 1 : 0)))

// Detect non-AVR microcontrollers (Teensy 3.x, Arduino DUE, etc.) and don't use port registers or timer interrupts as required.

#if (defined (\_\_arm\_\_) && defined (TEENSYDUINO))

#undef PING\_OVERHEAD

#define PING\_OVERHEAD 1

#undef PING\_TIMER\_OVERHEAD

#define PING\_TIMER\_OVERHEAD 1

#elif !defined (\_\_AVR\_\_)

#undef PING\_OVERHEAD

#define PING\_OVERHEAD 1

#undef PING\_TIMER\_OVERHEAD

#define PING\_TIMER\_OVERHEAD 1

#undef TIMER\_ENABLED

#define TIMER\_ENABLED false

#endif

// Disable the timer interrupts when using ATmega128 and all ATtiny microcontrollers.

#if defined (\_\_AVR\_ATmega128\_\_) || defined(\_\_AVR\_ATtiny24\_\_) || defined(\_\_AVR\_ATtiny44\_\_) || defined(\_\_AVR\_ATtiny84\_\_) || defined(\_\_AVR\_ATtiny25\_\_) || defined(\_\_AVR\_ATtiny45\_\_) || defined(\_\_AVR\_ATtiny85\_\_) || defined(\_\_AVR\_ATtiny261\_\_) || defined(\_\_AVR\_ATtiny461\_\_) || defined(\_\_AVR\_ATtiny861\_\_) || defined(\_\_AVR\_ATtiny43U\_\_)

#undef TIMER\_ENABLED

#define TIMER\_ENABLED false

#endif

// Define timers when using ATmega8 microcontrollers.

#if defined (\_\_AVR\_ATmega8\_\_)

#define OCR2A OCR2

#define TIMSK2 TIMSK

#define OCIE2A OCIE2

#endif

class NewPing {

public:

NewPing(uint8\_t trigger\_pin, uint8\_t echo\_pin, unsigned int max\_cm\_distance = MAX\_SENSOR\_DISTANCE);

unsigned int ping();

unsigned long ping\_cm();

unsigned long ping\_in();

unsigned long ping\_median(uint8\_t it = 5);

unsigned int convert\_cm(unsigned int echoTime);

unsigned int convert\_in(unsigned int echoTime);

#if TIMER\_ENABLED == true

void ping\_timer(void (\*userFunc)(void));

boolean check\_timer();

unsigned long ping\_result;

static void timer\_us(unsigned int frequency, void (\*userFunc)(void));

static void timer\_ms(unsigned long frequency, void (\*userFunc)(void));

static void timer\_stop();

#endif

private:

boolean ping\_trigger();

#if TIMER\_ENABLED == true

boolean ping\_trigger\_timer(unsigned int trigger\_delay);

boolean ping\_wait\_timer();

static void timer\_setup();

static void timer\_ms\_cntdwn();

#endif

uint8\_t \_triggerBit;

uint8\_t \_echoBit;

volatile uint8\_t \*\_triggerOutput;

volatile uint8\_t \*\_echoInput;

volatile uint8\_t \*\_triggerMode;

unsigned int \_maxEchoTime;

unsigned long \_max\_time;

};

#endif

NewPing.cpp

// ---------------------------------------------------------------------------

// Created by Tim Eckel - teckel@leethost.com

// Copyright 2015 License: GNU GPL v3 http://www.gnu.org/licenses/gpl.html

//

// See "NewPing.h" for purpose, syntax, version history, links, and more.

// ---------------------------------------------------------------------------

#include "NewPing.h"

// ---------------------------------------------------------------------------

// NewPing constructor

// ---------------------------------------------------------------------------

NewPing::NewPing(uint8\_t trigger\_pin, uint8\_t echo\_pin, unsigned int max\_cm\_distance) {

\_triggerBit = digitalPinToBitMask(trigger\_pin); // Get the port register bitmask for the trigger pin.

\_echoBit = digitalPinToBitMask(echo\_pin); // Get the port register bitmask for the echo pin.

\_triggerOutput = portOutputRegister(digitalPinToPort(trigger\_pin)); // Get the output port register for the trigger pin.

\_echoInput = portInputRegister(digitalPinToPort(echo\_pin)); // Get the input port register for the echo pin.

\_triggerMode = (uint8\_t \*) portModeRegister(digitalPinToPort(trigger\_pin)); // Get the port mode register for the trigger pin.

#if ROUNDING\_ENABLED == false

\_maxEchoTime = min(max\_cm\_distance + 1, (unsigned int) MAX\_SENSOR\_DISTANCE + 1) \* US\_ROUNDTRIP\_CM; // Calculate the maximum distance in uS (no rounding).

#else

\_maxEchoTime = min(max\_cm\_distance, (unsigned int) MAX\_SENSOR\_DISTANCE) \* US\_ROUNDTRIP\_CM + (US\_ROUNDTRIP\_CM / 2); // Calculate the maximum distance in uS.

#endif

#if defined (\_\_arm\_\_) && defined (TEENSYDUINO)

pinMode(echo\_pin, INPUT); // Set echo pin to input (on Teensy 3.x (ARM), pins default to disabled, at least one pinMode() is needed for GPIO mode).

pinMode(trigger\_pin, OUTPUT); // Set trigger pin to output (on Teensy 3.x (ARM), pins default to disabled, at least one pinMode() is needed for GPIO mode).

#endif

#if defined (ARDUINO\_AVR\_YUN)

pinMode(echo\_pin, INPUT); // Set echo pin to input on the Arduino Yun, not sure why it doesn't default this way.

#endif

#if ONE\_PIN\_ENABLED != true

\*\_triggerMode |= \_triggerBit; // Set trigger pin to output.

#endif

}

// ---------------------------------------------------------------------------

// Standard ping methods

// ---------------------------------------------------------------------------

unsigned int NewPing::ping() {

if (!ping\_trigger()) return NO\_ECHO; // Trigger a ping, if it returns false, return NO\_ECHO to the calling function.

#if URM37\_ENABLED == true

while (!(\*\_echoInput & \_echoBit)) // Wait for the ping echo.

if (micros() > \_max\_time) return NO\_ECHO; // Stop the loop and return NO\_ECHO (false) if we're beyond the set maximum distance.

#else

while (\*\_echoInput & \_echoBit) // Wait for the ping echo.

if (micros() > \_max\_time) return NO\_ECHO; // Stop the loop and return NO\_ECHO (false) if we're beyond the set maximum distance.

#endif

return (micros() - (\_max\_time - \_maxEchoTime) - PING\_OVERHEAD); // Calculate ping time, include overhead.

}

unsigned long NewPing::ping\_cm() {

unsigned long echoTime = NewPing::ping(); // Calls the ping method and returns with the ping echo distance in uS.

#if ROUNDING\_ENABLED == false

return (echoTime / US\_ROUNDTRIP\_CM); // Call the ping method and returns the distance in centimeters (no rounding).

#else

return NewPingConvert(echoTime, US\_ROUNDTRIP\_CM); // Convert uS to centimeters.

#endif

}

unsigned long NewPing::ping\_in() {

unsigned long echoTime = NewPing::ping(); // Calls the ping method and returns with the ping echo distance in uS.

#if ROUNDING\_ENABLED == false

return (echoTime / US\_ROUNDTRIP\_IN); // Call the ping method and returns the distance in inches (no rounding).

#else

return NewPingConvert(echoTime, US\_ROUNDTRIP\_IN); // Convert uS to inches.

#endif

}

unsigned long NewPing::ping\_median(uint8\_t it) {

unsigned int uS[it], last;

uint8\_t j, i = 0;

unsigned long t;

uS[0] = NO\_ECHO;

while (i < it) {

t = micros(); // Start ping timestamp.

last = ping(); // Send ping.

if (last != NO\_ECHO) { // Ping in range, include as part of median.

if (i > 0) { // Don't start sort till second ping.

for (j = i; j > 0 && uS[j - 1] < last; j--) // Insertion sort loop.

uS[j] = uS[j - 1]; // Shift ping array to correct position for sort insertion.

} else j = 0; // First ping is sort starting point.

uS[j] = last; // Add last ping to array in sorted position.

i++; // Move to next ping.

} else it--; // Ping out of range, skip and don't include as part of median.

if (i < it && micros() - t < PING\_MEDIAN\_DELAY)

delay((PING\_MEDIAN\_DELAY + t - micros()) / 1000); // Millisecond delay between pings.

}

return (uS[it >> 1]); // Return the ping distance median.

}

// ---------------------------------------------------------------------------

// Standard and timer interrupt ping method support function (not called directly)

// ---------------------------------------------------------------------------

boolean NewPing::ping\_trigger() {

#if ONE\_PIN\_ENABLED == true

\*\_triggerMode |= \_triggerBit; // Set trigger pin to output.

#endif

\*\_triggerOutput &= ~\_triggerBit; // Set the trigger pin low, should already be low, but this will make sure it is.

delayMicroseconds(4); // Wait for pin to go low.

\*\_triggerOutput |= \_triggerBit; // Set trigger pin high, this tells the sensor to send out a ping.

delayMicroseconds(10); // Wait long enough for the sensor to realize the trigger pin is high. Sensor specs say to wait 10uS.

\*\_triggerOutput &= ~\_triggerBit; // Set trigger pin back to low.

#if ONE\_PIN\_ENABLED == true

\*\_triggerMode &= ~\_triggerBit; // Set trigger pin to input (when using one Arduino pin this is technically setting the echo pin to input as both are tied to the same Arduino pin).

#endif

#if URM37\_ENABLED == true

if (!(\*\_echoInput & \_echoBit)) return false; // Previous ping hasn't finished, abort.

\_max\_time = micros() + \_maxEchoTime + MAX\_SENSOR\_DELAY; // Maximum time we'll wait for ping to start (most sensors are <450uS, the SRF06 can take up to 34,300uS!)

while (\*\_echoInput & \_echoBit) // Wait for ping to start.

if (micros() > \_max\_time) return false; // Took too long to start, abort.

#else

if (\*\_echoInput & \_echoBit) return false; // Previous ping hasn't finished, abort.

\_max\_time = micros() + \_maxEchoTime + MAX\_SENSOR\_DELAY; // Maximum time we'll wait for ping to start (most sensors are <450uS, the SRF06 can take up to 34,300uS!)

while (!(\*\_echoInput & \_echoBit)) // Wait for ping to start.

if (micros() > \_max\_time) return false; // Took too long to start, abort.

#endif

\_max\_time = micros() + \_maxEchoTime; // Ping started, set the time-out.

return true; // Ping started successfully.

}

#if TIMER\_ENABLED == true

// ---------------------------------------------------------------------------

// Timer interrupt ping methods (won't work with Teensy 3.0, ATmega8/128 and all ATtiny microcontrollers)

// ---------------------------------------------------------------------------

void NewPing::ping\_timer(void (\*userFunc)(void)) {

if (!ping\_trigger()) return; // Trigger a ping, if it returns false, return without starting the echo timer.

timer\_us(ECHO\_TIMER\_FREQ, userFunc); // Set ping echo timer check every ECHO\_TIMER\_FREQ uS.

}

boolean NewPing::check\_timer() {

if (micros() > \_max\_time) { // Outside the time-out limit.

timer\_stop(); // Disable timer interrupt

return false; // Cancel ping timer.

}

#if URM37\_ENABLED == false

if (!(\*\_echoInput & \_echoBit)) { // Ping echo received.

#else

if (\*\_echoInput & \_echoBit) { // Ping echo received.

#endif

timer\_stop(); // Disable timer interrupt

ping\_result = (micros() - (\_max\_time - \_maxEchoTime) - PING\_TIMER\_OVERHEAD); // Calculate ping time including overhead.

return true; // Return ping echo true.

}

return false; // Return false because there's no ping echo yet.

}

// ---------------------------------------------------------------------------

// Timer2/Timer4 interrupt methods (can be used for non-ultrasonic needs)

// ---------------------------------------------------------------------------

// Variables used for timer functions

void (\*intFunc)();

void (\*intFunc2)();

unsigned long \_ms\_cnt\_reset;

volatile unsigned long \_ms\_cnt;

#if defined(\_\_arm\_\_) && defined(TEENSYDUINO)

IntervalTimer itimer;

#endif

void NewPing::timer\_us(unsigned int frequency, void (\*userFunc)(void)) {

intFunc = userFunc; // User's function to call when there's a timer event.

timer\_setup(); // Configure the timer interrupt.

#if defined (\_\_AVR\_ATmega32U4\_\_) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).

OCR4C = min((frequency>>2) - 1, 255); // Every count is 4uS, so divide by 4 (bitwise shift right 2) subtract one, then make sure we don't go over 255 limit.

TIMSK4 = (1<<TOIE4); // Enable Timer4 interrupt.

#elif defined (\_\_arm\_\_) && defined (TEENSYDUINO) // Timer for Teensy 3.x

itimer.begin(userFunc, frequency); // Really simple on the Teensy 3.x, calls userFunc every 'frequency' uS.

#else

OCR2A = min((frequency>>2) - 1, 255); // Every count is 4uS, so divide by 4 (bitwise shift right 2) subtract one, then make sure we don't go over 255 limit.

TIMSK2 |= (1<<OCIE2A); // Enable Timer2 interrupt.

#endif

}

void NewPing::timer\_ms(unsigned long frequency, void (\*userFunc)(void)) {

intFunc = NewPing::timer\_ms\_cntdwn; // Timer events are sent here once every ms till user's frequency is reached.

intFunc2 = userFunc; // User's function to call when user's frequency is reached.

\_ms\_cnt = \_ms\_cnt\_reset = frequency; // Current ms counter and reset value.

timer\_setup(); // Configure the timer interrupt.

#if defined (\_\_AVR\_ATmega32U4\_\_) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).

OCR4C = 249; // Every count is 4uS, so 1ms = 250 counts - 1.

TIMSK4 = (1<<TOIE4); // Enable Timer4 interrupt.

#elif defined (\_\_arm\_\_) && defined (TEENSYDUINO) // Timer for Teensy 3.x

itimer.begin(NewPing::timer\_ms\_cntdwn, 1000); // Set timer to 1ms (1000 uS).

#else

OCR2A = 249; // Every count is 4uS, so 1ms = 250 counts - 1.

TIMSK2 |= (1<<OCIE2A); // Enable Timer2 interrupt.

#endif

}

void NewPing::timer\_stop() { // Disable timer interrupt.

#if defined (\_\_AVR\_ATmega32U4\_\_) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).

TIMSK4 = 0;

#elif defined (\_\_arm\_\_) && defined (TEENSYDUINO) // Timer for Teensy 3.x

itimer.end();

#else

TIMSK2 &= ~(1<<OCIE2A);

#endif

}

// ---------------------------------------------------------------------------

// Timer2/Timer4 interrupt method support functions (not called directly)

// ---------------------------------------------------------------------------

void NewPing::timer\_setup() {

#if defined (\_\_AVR\_ATmega32U4\_\_) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).

timer\_stop(); // Disable Timer4 interrupt.

TCCR4A = TCCR4C = TCCR4D = TCCR4E = 0;

TCCR4B = (1<<CS42) | (1<<CS41) | (1<<CS40) | (1<<PSR4); // Set Timer4 prescaler to 64 (4uS/count, 4uS-1020uS range).

TIFR4 = (1<<TOV4);

TCNT4 = 0; // Reset Timer4 counter.

#elif defined (\_\_AVR\_ATmega8\_\_)

timer\_stop(); // Disable Timer2 interrupt.

ASSR &= ~(1<<AS2); // Set clock, not pin.

TCCR2 = (1<<WGM21 | 1<<CS22); // Set Timer2 to CTC mode, prescaler to 64 (4uS/count, 4uS-1020uS range).

TCNT2 = 0; // Reset Timer2 counter.

#elif defined (\_\_arm\_\_) && defined (TEENSYDUINO)

timer\_stop(); // Stop the timer.

#else

timer\_stop(); // Disable Timer2 interrupt.

ASSR &= ~(1<<AS2); // Set clock, not pin.

TCCR2A = (1<<WGM21); // Set Timer2 to CTC mode.

TCCR2B = (1<<CS22); // Set Timer2 prescaler to 64 (4uS/count, 4uS-1020uS range).

TCNT2 = 0; // Reset Timer2 counter.

#endif

}

void NewPing::timer\_ms\_cntdwn() {

if (!\_ms\_cnt--) { // Count down till we reach zero.

intFunc2(); // Scheduled time reached, run the main timer event function.

\_ms\_cnt = \_ms\_cnt\_reset; // Reset the ms timer.

}

}

#if defined (\_\_AVR\_ATmega32U4\_\_) // Use Timer4 for ATmega32U4 (Teensy/Leonardo).

ISR(TIMER4\_OVF\_vect) {

intFunc(); // Call wrapped function.

}

#elif defined (\_\_AVR\_ATmega8\_\_) // ATmega8 microcontrollers.

ISR(TIMER2\_COMP\_vect) {

intFunc(); // Call wrapped function.

}

#elif defined (\_\_arm\_\_)

// Do nothing...

#else

ISR(TIMER2\_COMPA\_vect) {

intFunc(); // Call wrapped function.

}

#endif

#endif

// ---------------------------------------------------------------------------

// Conversion methods (rounds result to nearest cm or inch).

// ---------------------------------------------------------------------------

unsigned int NewPing::convert\_cm(unsigned int echoTime) {

#if ROUNDING\_ENABLED == false

return (echoTime / US\_ROUNDTRIP\_CM); // Convert uS to centimeters (no rounding).

#else

return NewPingConvert(echoTime, US\_ROUNDTRIP\_CM); // Convert uS to centimeters.

#endif

}

unsigned int NewPing::convert\_in(unsigned int echoTime) {

#if ROUNDING\_ENABLED == false

return (echoTime / US\_ROUNDTRIP\_IN); // Convert uS to inches (no rounding).

#else

return NewPingConvert(echoTime, US\_ROUNDTRIP\_IN); // Convert uS to inches.

#endif

}

Processing code

import processing.serial.\*;

class SensorData {

private float temperature;

private float distance;

SensorData(float temperature, float distance) {

this.temperature = temperature;

this.distance = distance;

}

float getTemperature() {

return this.temperature;

}

float getDistance() {

return this.distance;

}

}

final int WIDTH = 720;

final int HEIGHT = 720;

final int xCenter = WIDTH / 2;

final int yCenter = HEIGHT / 2;

final int LINE\_FEED = 10;

Serial arduinoPort;

SensorData sensorData;

int degree = 0;

int radius = 0;

// END:init\_sonar

// START:arduino\_communication

void setup() {

size(720,720);

printArray(Serial.list()) ;

String arduinoPortName = Serial.list()[2];

arduinoPort = new Serial(this, arduinoPortName, 9600); // <label id="code.input.serial\_port"/>

// arduinoPort.bufferUntil('\n');

println(arduinoPort);

println(arduinoPortName);

}

void serialEvent() {

sensorData = getSensorData();

if (sensorData != null) {

println("Temperature: " + sensorData.getTemperature());

println("Distance: " + sensorData.getDistance());

radius = min(300, int(sensorData.getDistance() \* 200));

}

}

SensorData getSensorData() {

SensorData result = null;

if (arduinoPort.available() > 0) { // <label id="code.input.data\_available"/>

final String arduinoOutput = arduinoPort.readStringUntil(LINE\_FEED);

println(arduinoOutput);

result = parseArduinoOutput(arduinoOutput);

}

return result;

}

SensorData parseArduinoOutput(final String arduinoOutput) {

SensorData result = null;

if (arduinoOutput != null) {

final int[] data = int(split(trim(arduinoOutput), ',')); // <label id="code.input.parse\_output"/>

if (data.length == 2)

result = new SensorData(data[0] / 100.0, data[1] / 100.0); // <label id="code.input.rescale\_data"/>

}

return result;

}

// END:arduino\_communication

// START:drawing

void init\_screen() {

background(255); // <label id="code.input.clear\_screen"/>

stroke(0);

strokeWeight(1);

int[] radius\_values = { 300, 250, 200, 150, 100, 50 };

for (int r = 0; r < radius\_values.length; r++) {

final int current\_radius = radius\_values[r] \* 2;

ellipse(xCenter, yCenter, current\_radius, current\_radius);

}

strokeWeight(10);

}

void draw() {

init\_screen();

serialEvent();

sensorData = getSensorData();

int x = (int)(radius \* Math.cos(degree \* Math.PI / 180));

int y = (int)(radius \* Math.sin(degree \* Math.PI / 180));

point(xCenter + x, yCenter + y);

if (++degree == 360)

degree = 0;

}

// END:drawing