Map Objects in a Grid

An Arduino and Perl Project

by

Tom Wolfe

March 2013

Table of Contents

Introduction

This project is a combination of an Arduino sketch and a Perl program. Existing Arduino sketchs were combined and modified for the project and a Perl program created to send commands and receive data from the Arduino.

This all came about after I purchased an Arduino and started playing with and modifying some of the existing sketches. I got to thinking what if... (The rest is history.)

The project uses an Arduino to control a servo and ultrasound sensor to collect data that can be used to locate objects in a grid of cells. Essentially it is a radar (sonar) sweep of the grid cells to locate objects. The Arduino receives commands from a Perl program and returns raw data for processing.

Note: This turned out to more of a programming project than a hardware project.

Copyright

What Next (If I feel like it)

1. Improve this documentation (polish the cannonball)
2. Make the GUI better looking (it's kind of clunky)
3. Modify the control program to run the ping sweep in a thread so it does no lock up the GUI.
4. Modify the ultrasound sensor with (paper?) cylinders around the receiver (and/or transmitter) to narrow its detection angle.
5. Test/calibrate the sensor to refine its distance calculations. I suspect that it may be slightly off.
6. Add a second servo to move the sensor vertically and determine how tall an object is.
7. Use a more accurate sensor. Microwave? Laser?
8. Scan the grid cells. Save the occupied counts for each cell. Move the sensor to a different location and scan again. Add the second counts to the first counts. This, in theory, should increase the accuracy. Move the sensor to a third location Etc.
9. Use Wi-Fi rather than connecting directly to the computer's USB port.
10. Mount the Arduino, servo, and sensor on a remotely controlled vehicle (car or helicopter). I need to some way determine the servo's location (coordinates) and the servo's orientation (zero direction).

Arduino Documentation

www.arduino.cc
Arduino Reference Manual
arduiniana.org

Perl

The project uses a program (written in PERL) to sends commands and process returned data. It also displays data graphically using Tkx.

Why PERL?

Why not? Perl can do everything I needed to do. (Python or Java could also have been used.)

Which Computer System?

The project was developed on a Windows 7 system. (A Linux or Macintosh system could also have been used.)

Which Perl?

There are several Perl implementations available. ActivePerl from ActiveState was used because:

- AcvtivePerl Community Edition is free
- ActivePerl runs on multiple systems
- ActivePerl is easy to install
- ActivePerl comes with several useful modules pre-installed
- Tkx is part of the standard installation
- A Win32::SerialPort module is available

Why Tkx and not Tk?

Tkx is included in the ActivePerl installation. Tk is not.

CPAN Tkx Documentation
See the web for more Tkx information and resources

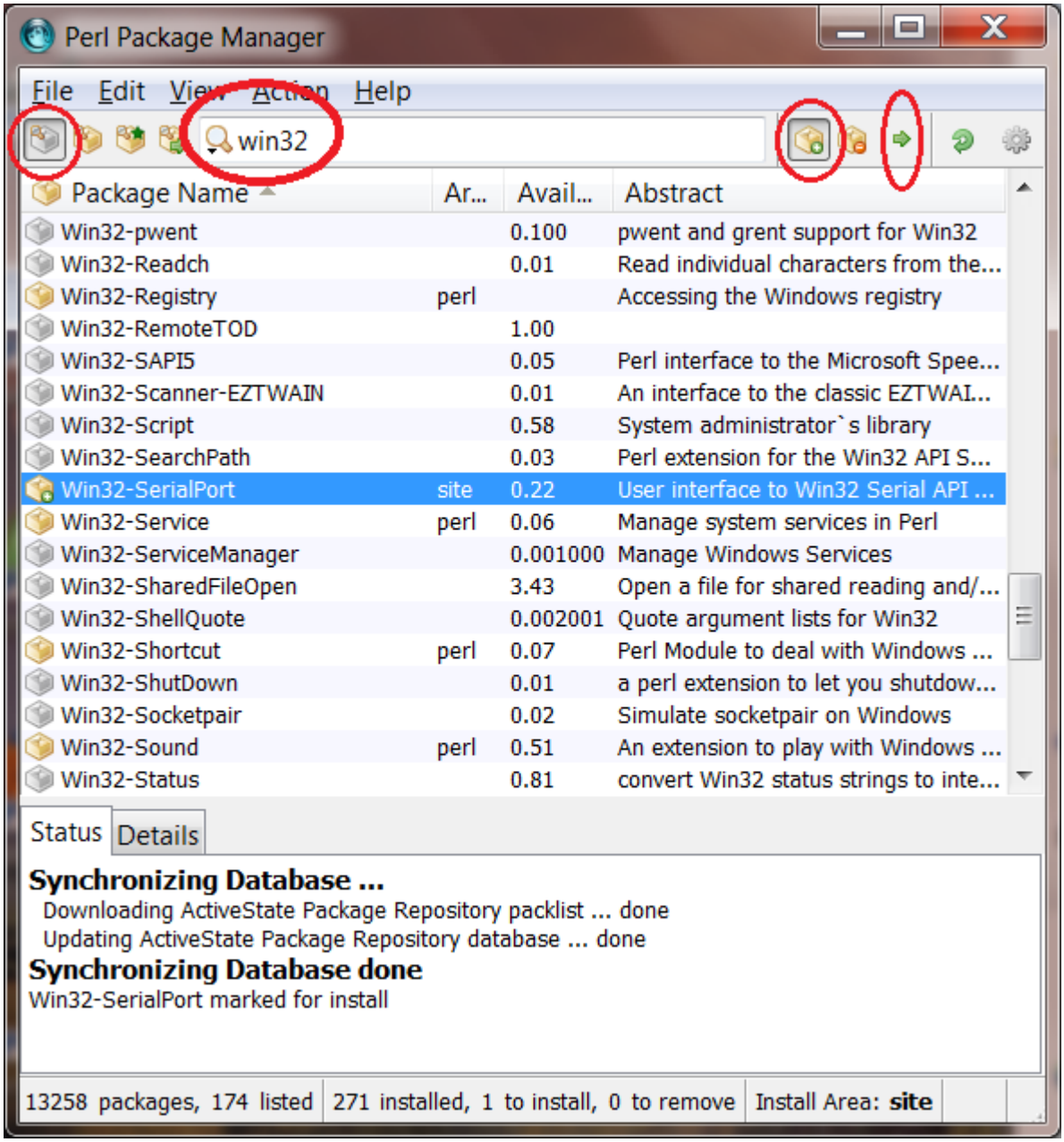
Win32::SerialPort

CPAN Win32::SerialPort Documentation

Setp 1 - Download and install ActivePerl

Download and install ActivePerl from here.

Step 2 - Use the Perl Package Manager to install the Win32::SerialPort module



Note: Place the cursor over the circled areas to see what they do.

Test Programs

It is always a good idea to test functionality in small programs to make sure it works as expected. It also makes debugging easier.

Do not ASS.U.ME!!

Several Perl programs (scripts) were written to test and verify ActivePerl's capabilities. There can be found the Perl Test Programs section.

Calculate Object Location

The purpose of the calculation is to determine if an object detected by the sensor is inside one of the grid squares to be mapped. The sensor can be located anywhere as long as:

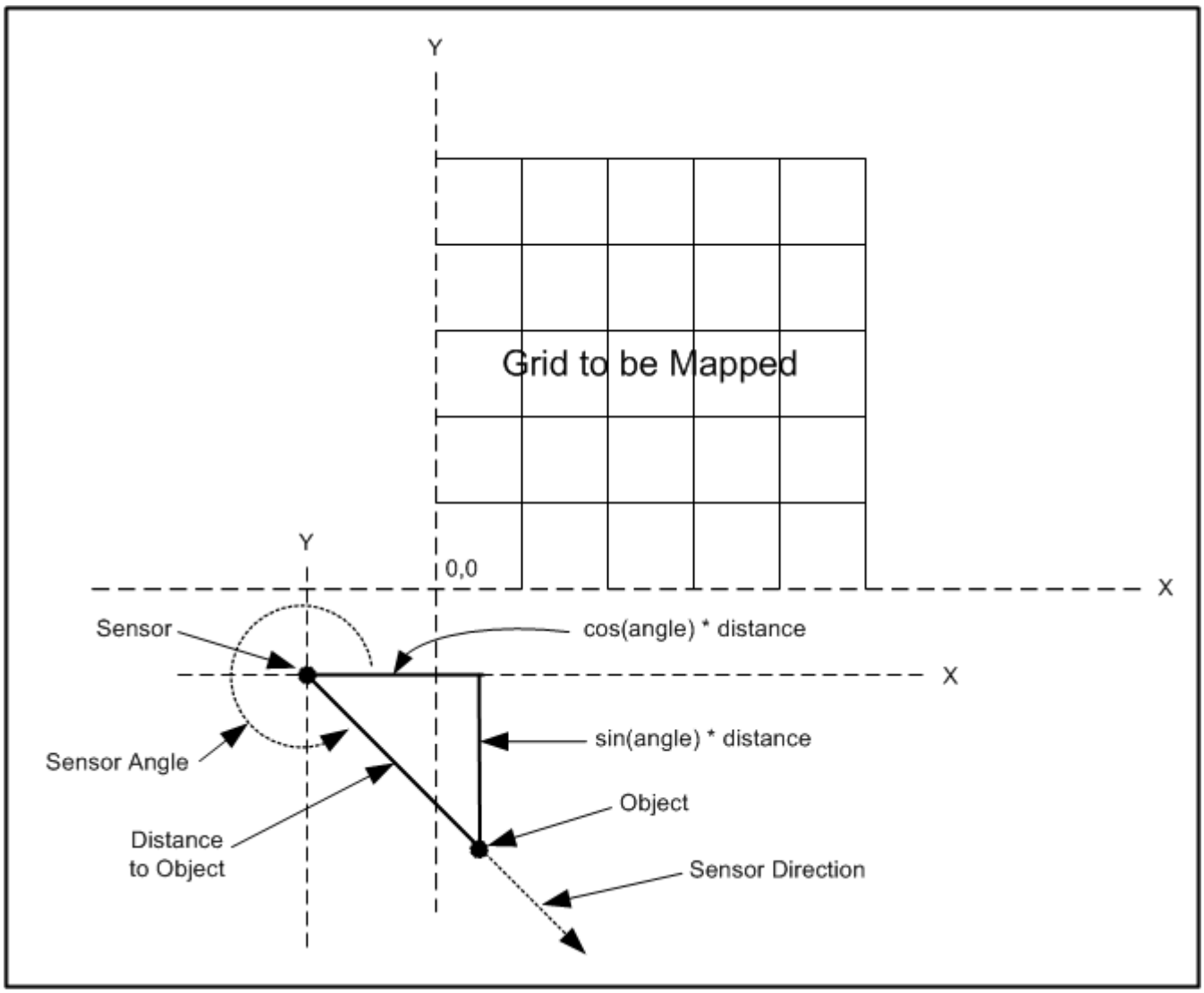
- The sensor's coordinates are known
- The sensor's X axis is parallel to the grid's X axis (If not, see the information below)
- Both the sensor's and grid's positive X and Y axes point in the same direction

The sensor's X axis is the servo's zero (0) direction. If the servo's zero (0) direction is not parallel to the grid's X axis (this is the usual case), see the servo section information on how to make adjustments.

The sensor angle is measured counterclockwise from the grid's positive X axis. This makes the sine and cosine of the angle have the correct sign (+,-) for the coordinate calculations. *Note: There is information about calculating the sensor angle on the servo section.*

Once an object's coordinates are known, it is easy to determine if the object falls within the grid to be mapped and which cell in the grid the object is in. *Note: the size of the grid and the size of each cell in the grid must be known.*

The servo's zero (0) direction is the result of setting the servo's position to zero (0). The sensor is mounted on the servo so that this is also the sensor's positive X axis. It does not have to be, but if it is not there are extra calculations required.



Calculations (pseudo code)

```
OX - object x coordinate
OY - object y coordinate
SX - sensor x coordinate
SY - sensor y coordinate
SA - sensor angle
D  - distance from sensor to object

OX = SX + (cos(SA) * D)
OY = SY + (sin(SA) * D)
GMINX - grid minimum x coordinate
GMINY - grid minimum y coordinate
GMAXX - grid maximum x coordinate
GMAXY - grid maximum y coordinate

if ((OX < GMINX) or (OX > GMAXX) or
    (OY < GMINY) or (OY > GMAXY))
{
  print "The object is outside of the grid."
}

CX  - cell X size
CY  - cell Y size
```

```
COL - object cell column
ROW - object cell row

COL = int((OX)/CX)
ROW = int((OY)/CY
```

PERL Code

```
use strict;
use Math::Trig;

# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# do not call this subroutine if the distance is zero (0)
# zero (0) means no object was detected
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

sub objCoordCalc
{
   my $sx = $_[0];             # sensor X coord
   my $sy = $_[1];             # sensor Y coord
   my $sa = $_[2];             # sensor angle (degrees)
   my $d  = $_[3];             # sensor to object distance

   my $r = $sa * (pi/180.0);   # sensor angle in radians

   my $x = 0;                  # object X coordinate
   my $y = 0;                  # object Y coordinate

   $x = $sx + (cos($r) * $d);
   $y = $sy + (sin($r) * $d);

   return ($x,$y);
}
```

Servo

The servo is used to turn the sensor so that it can scan the grid to be mapped.

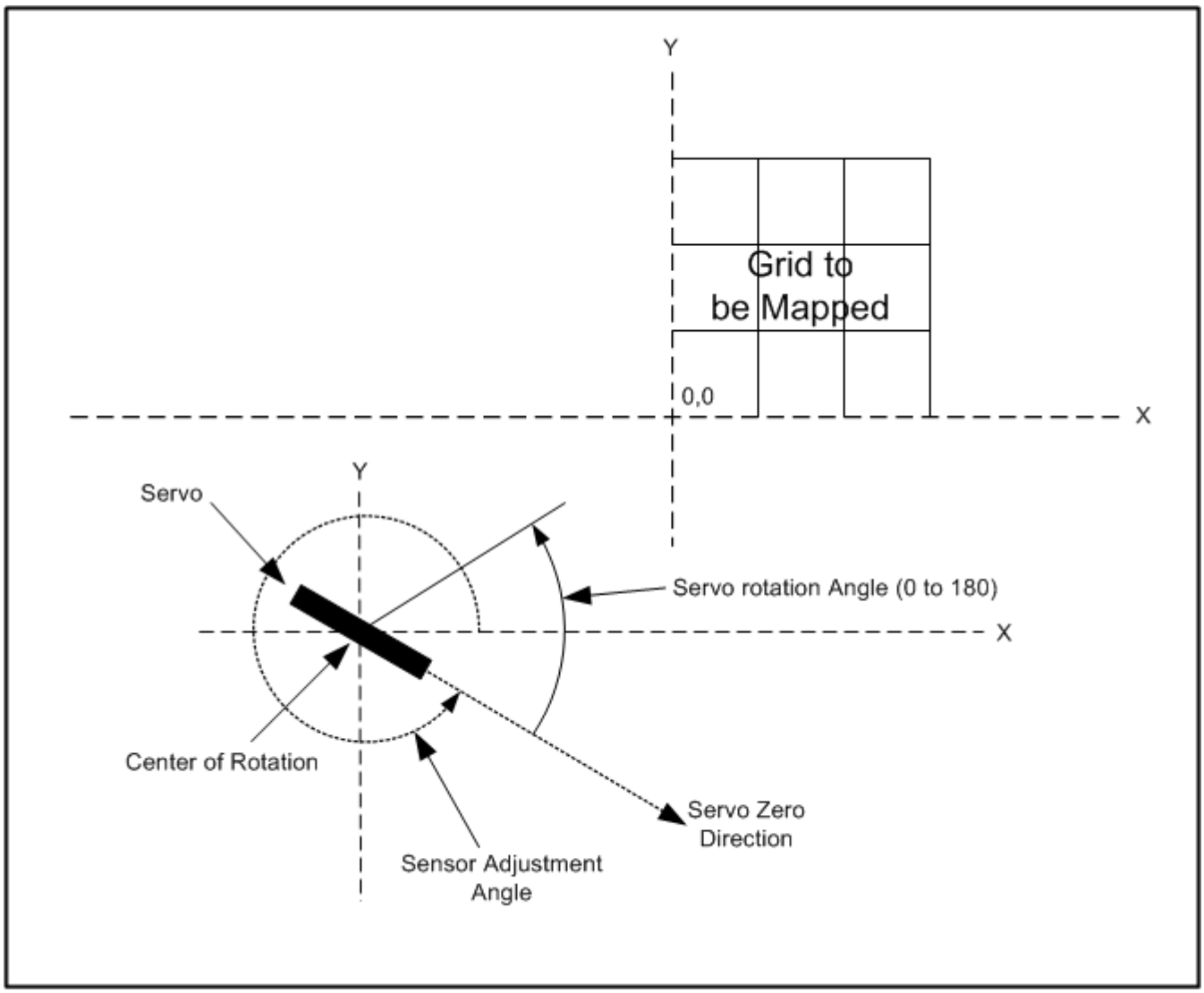
The servo may not be oriented with the gird X and Y axes. Therefore an adjustment must be made to the sensor angle used to calculate an object's coordinates.

I am using a servo that rotates counterclockwise from 0 to 180. Each position is assumed to be degrees from the zero direction.

The sensor is attached to the servo at the center of its rotation. The sensor point in the direction the servo has rotated to.

Because the sensor angle is measured from the positive X axis when calculating and objects coordinates, the sensor's adjustment angle must be used.

Note: See object coordinate calculation.



Calculations (pseudo code)

```
SRA - servo rotation angle (0 to 180)
      The sensor points in this direction.
SAA - sensor adjustment angle
      Counterclockwise from positive X axis.
SA  - sensor angle used in object coordinate calculations

SA = (SRA + SAA) % 360

Note: For integers only
359   % 360    = 359
360   % 360    = 0
361   % 360    = 1
359.1 % 360.0  = 359
360.1 % 360.0  = 0
361.1 % 360.0  = 1
```

PERL Code for Floating Point Numbers

```
use strict;

print "results: " . modFloatingPoint(359.1,360.0) . "\n";
```

```
print "results: " . modFloatingPoint(360.1,360.0) . "\n";
print "results: " . modFloatingPoint(361.1,360.0) . "\n";

sub modFloatingPoint
{
   my $n1 = $_[0];              # dividend
   my $n2 = $_[1];              # divisor

   my $n = int($n1/$n2);

   return $n1 - ($n * $n2);     # mod(n1/n2)
}
```

Test Output

```
results: 359.1
results: 0.100000000000023
results: 1.10000000000002
```

Project Sensor

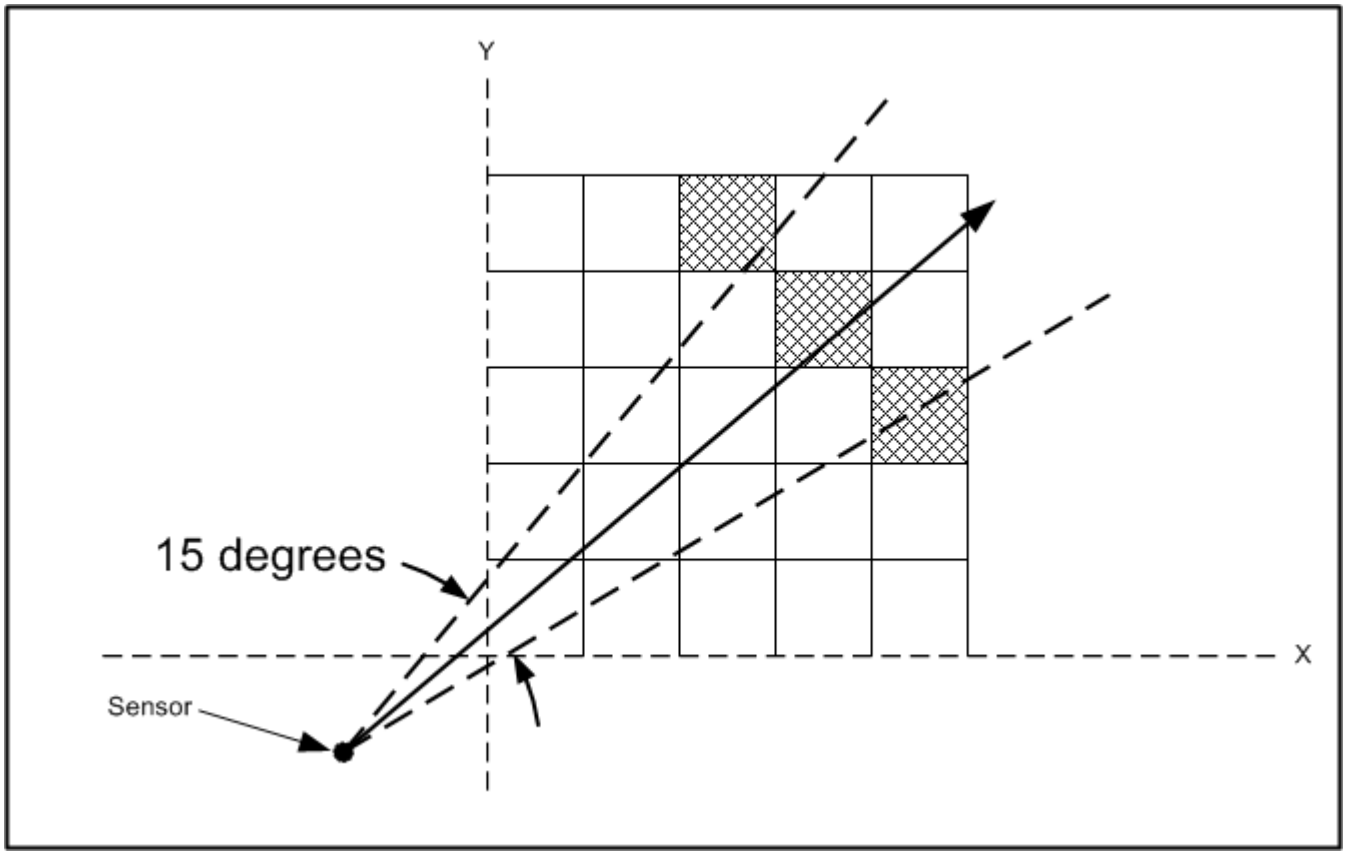
Ultrasonic Module HC-SR04 Distance Sensor

Detection	Value
Angle	<15°
Range	2cm ~ 500cm
Resolution	0.3 cm

Data from amazon.com

The sensor is used to locate objects in the grid to be mapped. The purpose of the calculation is to determine which cells contain objects. (Big objects could be in several cells.)

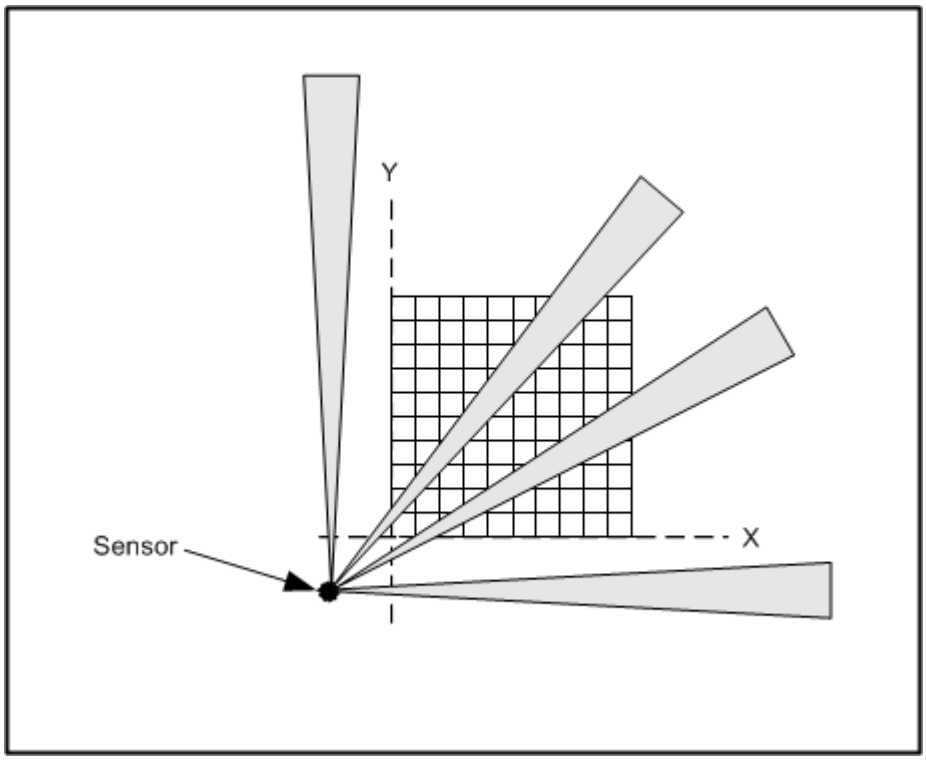
Because the sensor has a detection angle of < 15°, the distance to objects adds uncertainty as to which cell(s) are occupied and which ones are not. For Example, in the diagram it is uncertain which cell is actually occupied.



Scanning

Scans are done by rotating the sensor (1° at a time) keeping track of the number of times a cell appears to contain an object. Cell with low detection counts are probably empty.

The scan should begin and end completely outside of the grid.



Note: The size of the grid cells and the sensor-to-object distance determinates how many false positives are detected. The best values for eliminating false positives can be determined by experimentation.

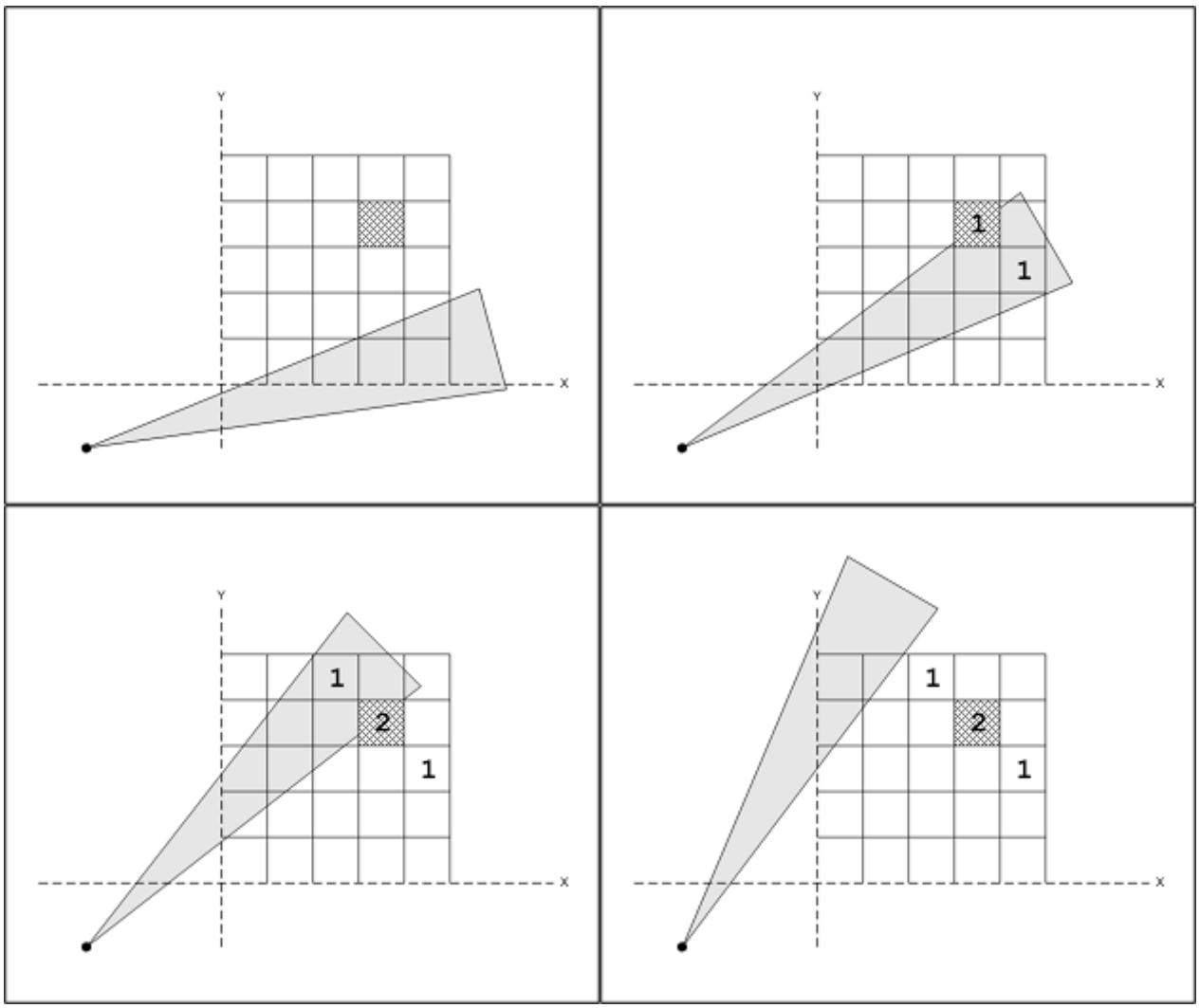
Which Cells Are Occupied?

Once an object is detected, you must determine which cell(s) the object might be in.

The sensor angle and the distance to the object are known. They can be used to determine which cell(s) are occupied.

I have chosen to use a simplification. Every time I move the sensor (1°) I use the sensor angle and distance to calculate which cell (if any) is occupied. I then modify the sensor angle ± 7° and calculate again to determine if any other cells appears to be occupied. If a cell is appears to be occupied, I increment its occupied count by one (1).

After scanning from one side of the grid to be mapped to the other, I see which cell(s) have the largest occupied count. These are probable occupied (contain an object). Lower counts are probably not occupied. For example:



I have chosen to start by changing the sensor angle ± 7°. Experimentation can be used to determine the optimal ± values. For example, using ± 3° and ± 7°.

Estimated Detection Angle Spread

7°		
5 ft	**10 ft**	**15 ft**
0.6 ft	1.2 ft	1.8 ft
12°		
5 ft	**10 ft**	**15 ft**
1.2 ft	2.4 ft	3.6 ft

sine of 7° = 0.1218693434
sine of 15° = 0.2588190451

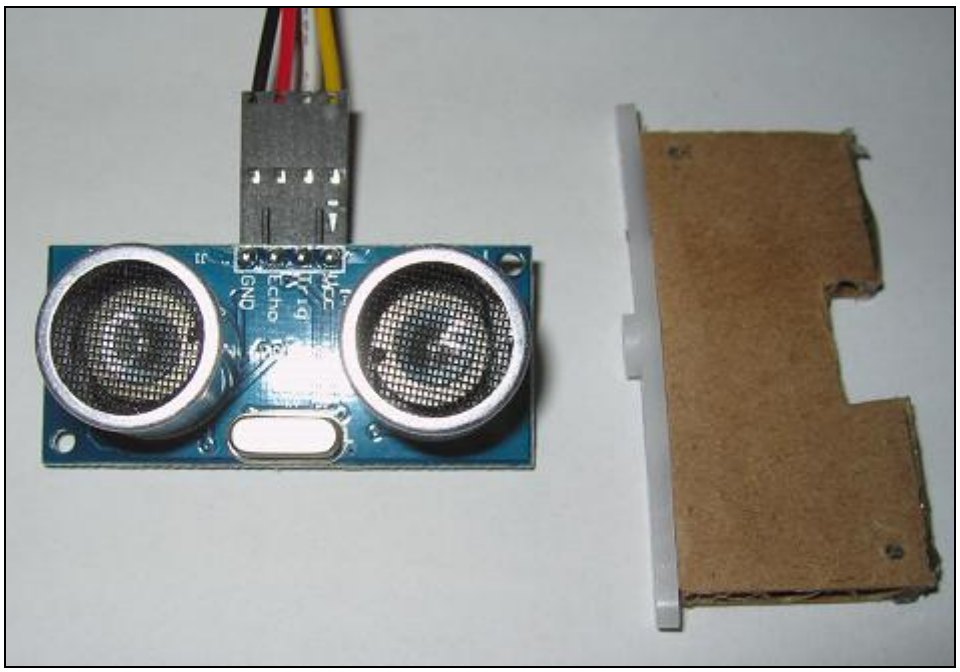
My Servo/Sensor Mounting Hardware

Description

With a glue gun and cardboard I created a mounting mechanism for the servo and sensor.

After setting the servo to its zero (0) position, I was able to "eyeball" the servo's zero direction and mark it with an arrow on the base.

Note: Someday I'll make something better and more accurate (it couldn't be worse). For now this is good enough to test my ideas.



LCD

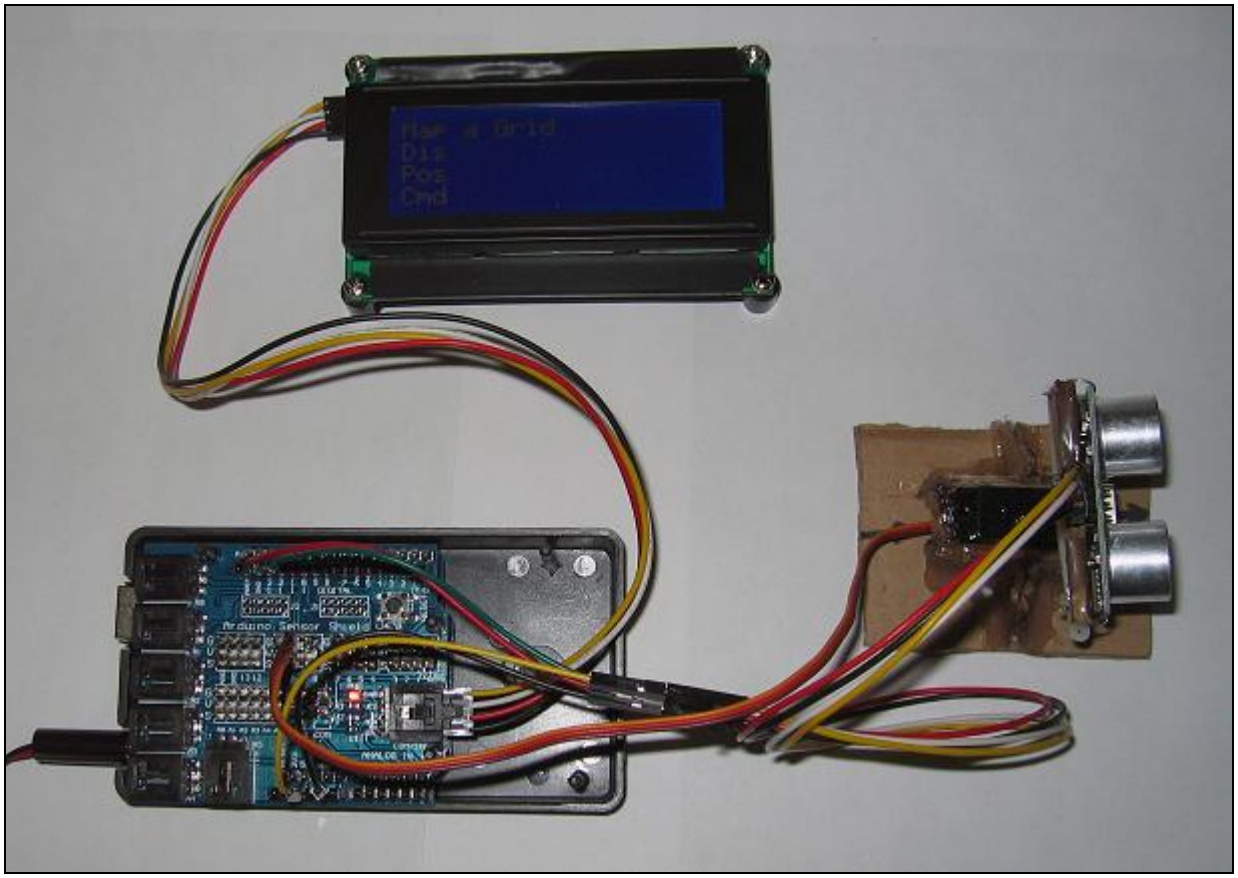
4 rows, 20 characters
Potentiometer contrast adjustment
I2C interface

The LCD is optional. I used it to help me debug the project. Plus, Its just fun to watch things while they are working.

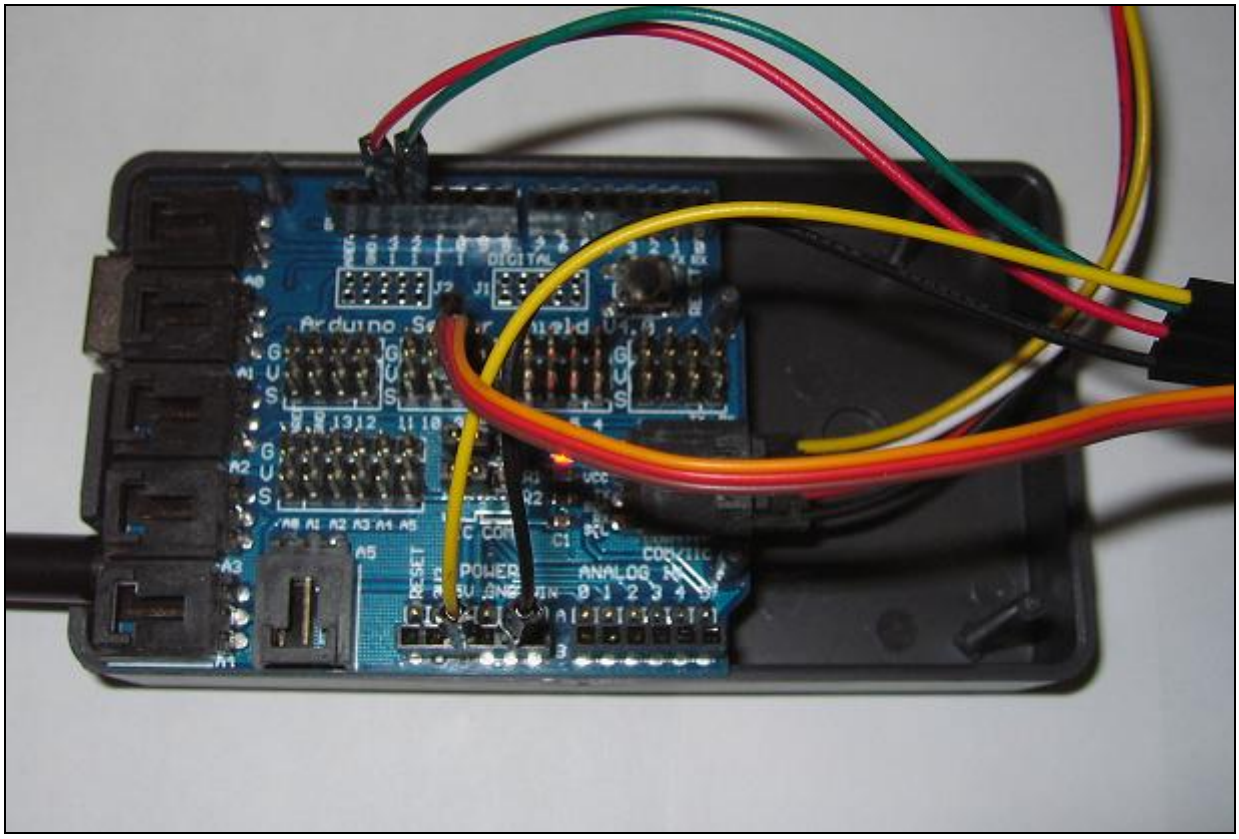
If you don't have a LCD, do the following in the project sketch:

Set LCDYES = false or remove the LCD related code.

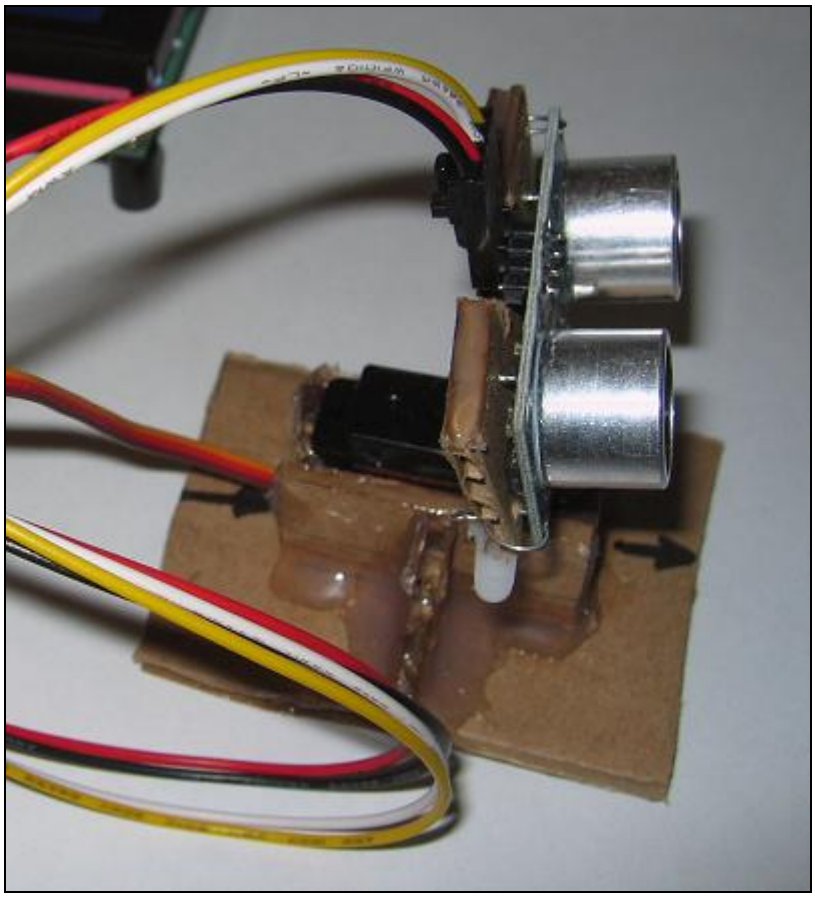
Project Hardware



Arduino with Sensor Shield



Sensor/servo Mount



Project Arduino Sketch

The Arduino sketch receives commands and returns data to the control program (written in Perl). The data returned is used by the control program to create displays, etc.

Note: See the comments in the sketch for documentation of the operations the sketch performs.

See appendix 1 for the project Arduino sketch.

Project Control Program

The control program sends commands to the Arduino and processes the returned data. It is written in Perl and uses Tkx to create a GUI interface. The returned command responses are displayed in graphic and text form.

The control program is used with the project's Arduino sketch and communications object (communications.pm).

Note: See the test programs on how to run the control program from a batch file or run it without a DOS window.

See appendix 2 for the control program source code.

Project Notes

The project uses an Arduino to control a servo and ultrasound sensor to collect data that can be used to locate objects in a grid of cells. Essentially it is a radar (sonar) sweep of the grid cells to locate objects. The Arduino receives commands from a control program (written in Perl) and returns raw data for processing.

See the other sections of this document for more documentation.

Conclusion

1. The project was interesting and I had fun developing it.
2. The premise the project was based on seems to be valid. The current cell and grid sizes give a rough estimate of which cells are occupied. Of course, I could not see behind objects. (See the "What Next" section of this document.)
3. There needs to be testing to determine the best grid and cell size to get the most accurate readings.
4. Because of the uncertainty in the detection angle (15°), testing need to be done to determine the most efficient subdivision of the detection angle to use. (± 7 or ± 7 and ± 3 or ...)

   Perhaps the farther away an object is from the sensor the more subdivisions you need? Perhaps dynamic subdivisions based on the distance from the sensor to the object?

5. There is much more testing to be done, but basically it works.

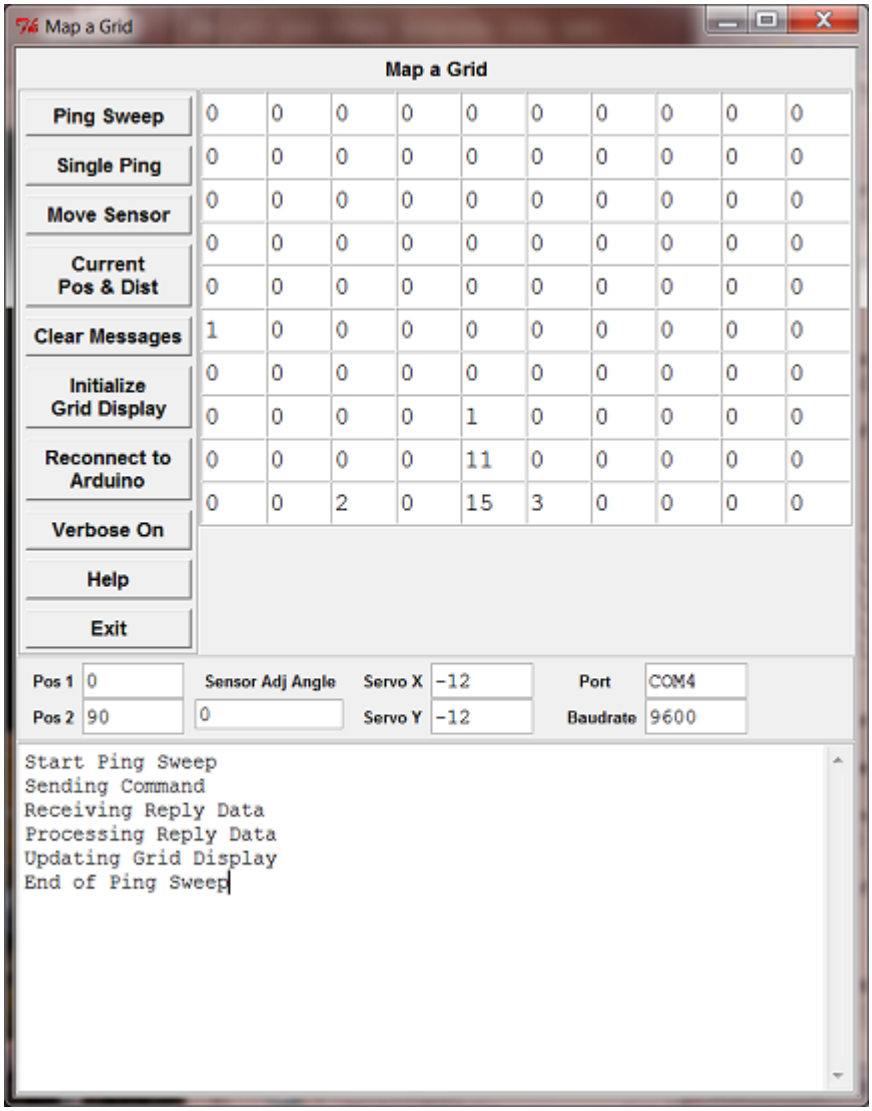
The Control Program (GUI)

The control program GUI is kind of clunky, but that is OK. After all, the GUI is not the point of the project.

The GUI contains:

- a set of buttons that execute actions
- an area that displays the cell occupied counts from ping sweeps
- operating parameters that can be modified by the users
- a text area where messages are displayed
- the source code also has a flag (TESTFLAG) that, if set to a non-zero value, will display extra buttons. These buttons were used for testing and debugging.

See the project's control program and Arduino sketch source code for documentation, comments, and other information.



GUI Fonts

The fonts used by the GUI are defined in the control program source code. To change the fonts, change these variables:

$btfont - button font
$gdfont - grid display font
$plfont - parameter label font
$pvfont - parameter entry font
$txfont - text window font
$wtfont - window title font

Communications.pm

All of the communications "stuff" is in the Perl module communications.pm. The control program creates a communications object using the module and "talks" to the Arduino with it.

Documentation, such as it is, is found in the module's source code. There are test Perl scripts that demonstrate the use of the communications.pm module and the window's Win32::SerialPort module.

Subroutine Return Status

Some of the subroutines in the communication.pm and the control program return an array rather than just a status. The first element in the array is a status and the second is information related to the status. (See the source code for examples.)

Note: Some people may not be familiar with this style of coding.

Design Notes

1. The ping sweep subroutine in the control program is a Tkx callback routine. It runs for a long time and will lock the GUI until it is finished. I was too lazy to code this differently so you will just have to wait. After all, the GUI is not the point of the project.

   Because the GUI locks up, the messages generated by the ping sweep will not be displayed until after it finishes.

2. Commands and messages used by the Arduino and control program are documented in the Project Arduino Sketch code.
3. The size of the grid cells and the number of rows and columns in the grid are defined as constants at the top of the control program source code.
4. Cell dimensions, coordinates, distances, etc. are in inches. This is determined by the ultrasound sensor which is currently set to return inches.

Communications.pm

A communication object is used by the control program to communicate with the Arduino.
Create the file communications.pm with the code in it.

Note: See the test programs for an example of using a communication object.

See appendix 3 for the source code.

Perl Test Programs

Windows Batch File

Each Perl program is executed by its own Windows batch file. This simplified development and testing. For example:

```
@echo off
rem ========================================================
rem
rem ========================================================

perl -w name-of-Perl-script-to-execute.pl

pause
```

How to Get Rid of the DOS Windows

In C:\Perl\bin there is an executable called wperl.exe that will run a Perl script without opening a DOS window. It is simply a Perl interpreter compiled as a Windows application rather than a console application.

Create a shortcut with...

```
wperl -w c:\path\to\script.pl

  or

C:\Perl\bin\wperl -w c:\path\to\script.pl
```

Double click it to run without a DOS window.

From: www.perlmonks.org/?node_id=271330

Test Project Commands

See the Arduino Test Sketches.

```
# ================================================================
# Test Project Commands (send predefined, hard coded commands)
# ================================================================

use strict;
use warnings;

require Win32::SerialPort;

my $portname = "COM4";

my $port = Win32::SerialPort->new($portname)
   or die "Can't open serial port (" . $portname . ")\n$^E\n";

# -- for debugging ------------------------------------------------
# prints hardware messages like "Framing Error"
# $port->error_msg(1);
# prints function messages like "Waiting for CTS"
# $port->user_msg(1);
# -----------------------------------------------------------------

$port->initialize();
$port->databits(8);
$port->baudrate(9600);
$port->parity("none");
$port->stopbits(1);
$port->debug(0);
```

```
#print "Serial port baudrate: " . $port->baudrate() . "\n";

# define line termination for $port->lookfor()
# Note: Arduino serial.println terminates each line/string
#       sent with a "\r\n"

$port->are_match("\r\n");

# -----------------------------------------------------------------
# send test S command
# -----------------------------------------------------------------

my $cmd;

$cmd = "S22,31";
print "\nSending Cmd $cmd\n";
sentCommand($cmd);
processReply();

# -----------------------------------------------------------------
# send test M command
# -----------------------------------------------------------------

$cmd = "M123";
print "\nSending Cmd $cmd\n";
sentCommand($cmd);
processReply();

# -----------------------------------------------------------------
# send test P command
# -----------------------------------------------------------------

$cmd = "P";
print "\nSending Cmd $cmd\n";
sentCommand($cmd);
processReply();

# -----------------------------------------------------------------
# send test C command
# -----------------------------------------------------------------

$cmd = "C";
print "\nSending Cmd $cmd\n";
sentCommand($cmd);
processReply();

# -----------------------------------------------------------------
# send test bad command
# -----------------------------------------------------------------

$cmd = "B,45,67,8";
print "\nSending Cmd $cmd\n";
sentCommand($cmd);
processReply();


$port->close();
exit 0;


# -----------------------------------------------------------------
# send command
# -----------------------------------------------------------------
```

```
sub sendCommand
{
   my $cmd = $_[0];

   my $l = length($cmd);

   $port->lookclear;           # empty buffer

   my $c = $port->write($cmd);

   if ($c == 0)
   {
      print "\nwrite failed\n";
   }
   elsif ($c != $l)
   {
      print "\nWrite incomplete (c=$c) (l=$l)\n";
   }

   return;
}


# ----------------------------------------------------------------
# process command reply
# ----------------------------------------------------------------

sub processReply
{
   my $data;

   $port->lookclear;           # empty buffer

   while(1)
   {
      # poll looking for data

      $data = $port->lookfor();

      # remove training carriage return (the default for
      #     are_match is "\n")
      # this code is not needed because of the $port->are_match
      # $data =~ s/\r$//;

      # if we get data, print it

      if ($data)
      {
         #print "String length:    " . length($data) . "\n";
         print "String Received: (" . $data . ")\n";
      }

      if ($data =~ /^x/) { return; }
   }

   return;
}
```

Send Project Commands

See the Arduino Test Sketches.

```
# ====================================================================
# Send Project Commands (ask the user for a command and send it)
# ====================================================================

use strict;
use warnings;

require Win32::SerialPort;

my $portname = "COM4";

my $port = Win32::SerialPort->new($portname)
   or die "Can't open serial port (" . $portname . ")\n$^E\n";


# -- for debugging ------------------------------------------------
# prints hardware messages like "Framing Error"
# $port->error_msg(1);
# prints function messages like "Waiting for CTS"
# $port->user_msg(1);
# -----------------------------------------------------------------

$port->initialize();
$port->databits(8);
$port->baudrate(9600);
$port->parity("none");
$port->stopbits(1);
$port->debug(0);

#print "Serial port baudrate: " . $port->baudrate() . "\n";

# define line termination for $port->lookfor()
# Note: Arduino serial.println terminates each line/string
#       sent with a "\r\n"

$port->are_match("\r\n");

# -----------------------------------------------------------------
# ask the user for a command and send it to the Arduino
# -----------------------------------------------------------------

my $cmd;

while(1)
{
   print "\nEnter commmand: ";
   $cmd = ;
   chomp $cmd;

   $cmd =~ s/^\s+//;          # trim white space
   $cmd =~ s/\s+$//;          # trim white space

   if (! $cmd) { last; }      # no command?

   if (($cmd =~ /^H/) || ($cmd =~ /^h/))
   {
      help();
      next;
   }
```

```
   sendCommand($cmd);
   processReply();
}

$port->close();
exit 0;


# ------------------------------------------------------------------
# send command
# ------------------------------------------------------------------

sub sendCommand
{
   my $cmd = $_[0];

   my $l = length($cmd);

   $port->lookclear;           # empty buffer

   my $c = $port->write($cmd);

   if ($c == 0)
   {
      print "\nwrite failed\n";
   }
   elsif ($c != $l)
   {
      print "\nWrite incomplete (c=$c) (l=$l)\n";
   }

   return;
}


# ------------------------------------------------------------------
# process the command's reply
# ------------------------------------------------------------------

sub processReply
{
   my $data;

   $port->lookclear;           # empty buffer

   while(1)
   {
      # poll looking for data

      $data = $port->lookfor();

      # remove training carriage return (the default for
      #    are_match is "\n")
      # this code is not needed because of the $port->are_match
      # $data =~ s/\r$//;

      # if we get data, print it

      if ($data)
      {
         #print "String length:    " . length($data) . "\n";
         print "String Received: (" . $data . ")\n";
      }
```

```
      if ($data =~ /^x/) { return; }
   }
}

# ---------------------------------------------------------------------
# display help
# ---------------------------------------------------------------------

sub help
{
   print << 'END';

Spos1,pos2 -- Do a ping sweep from servo position 1 to servo
              position 2 and return the resultant distances.

Mpos       -- Move the servo to a specified position.

P          -- Do a single ping for distance.

C          -- Return the current position and distance.

H or h     -- This help message.

END
}
```

Test Communication Object

```
# ====================================================================
# Test communication object
# ====================================================================

my $aref;
my $cmd;
my $status;

print "Creating communication object\n";

my $comm = new communications();

print "Opening communication port\n";

($status,$aref) = $comm->open('COM4',9600);

if (!$status)
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

# --------------------------------------------------------------------
# Sending test S command
# --------------------------------------------------------------------

my $aref;
my $cmd;
my $status;

$cmd = "s22,31";

print "\nSending Cmd $cmd\n";

($status,$aref) = $comm->sendMessage("S22,31");

if (!$status)
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

($status,$aref) = $comm->receiveReply();

DisplayReply($aref);


# --------------------------------------------------------------------
# Sending test M command
# --------------------------------------------------------------------

$cmd = "M123";

print "\nSending Cmd $cmd\n";

($status,$aref) = $comm->sendMessage($cmd);
```

```
if (!$status)
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

($status,$aref) = $comm->receiveReply();

DisplayReply($aref);

# ------------------------------------------------------------------
# Sending test P command
# ------------------------------------------------------------------

$cmd = "P";
print "\nSending Cmd $cmd\n";

($status,$aref) = $comm->sendMessage($cmd);

if (!$status)
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

($status,$aref) = $comm->receiveReply();

DisplayReply($aref);

# ------------------------------------------------------------------
# Sending test C command
# ------------------------------------------------------------------

$cmd = "C";

print "\nSending Cmd $cmd\n";

($status,$aref) = $comm->sendMessage($cmd);

if (!$status)
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

($status,$aref) = $comm->receiveReply();

DisplayReply($aref);

# ------------------------------------------------------------------
# Sending bad command
# ------------------------------------------------------------------

$cmd = "B,45,67,8";

print "\nSending Cmd $cmd\n";

($status,$aref) = $comm->sendMessage($cmd);

if (!$status)
```

```
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

($status,$aref) = $comm->receiveReply();

DisplayReply($aref);

# ------------------------------------------------------------------
# close communication port
# ------------------------------------------------------------------

print "\nClosing communication port\n";

$comm->close();


# ------------------------------------------------------------------
# Sending command to closed communication port
# ------------------------------------------------------------------

$cmd = "ZZZ";

print "\nSending Cmd $cmd\n";

($status,$aref) = $comm->sendMessage($cmd);

if (!$status)
{
   print "Communication failed\n";
   DisplayReply($aref);
   exit 0;
}

($status,$aref) = $comm->receiveReply();

DisplayReply($aref);

exit 0;

# ==================================================================
# display reply messages
# ==================================================================

sub DisplayReply
{
   my $aref = $_[0];         # reference to array of reply strings

   foreach my $m (@$aref) { print "$m\n"; }
}
```

Test Output

Note: See the Arduino Test Sketches.

```
Creating communication object
Opening communication port

Sending Cmd s22,31
S22,31
```

```
s22,220
s23,230
s24,240
s25,250
s26,260
s27,270
s28,280
s29,290
s30,300
s31,310
x

Sending Cmd M123
M123
x

Sending Cmd P
p124
x

Sending Cmd C
c45,90
x

Sending Cmd B,45,67,8
e66
x

Closing communication port

Sending Cmd ZZZ
Communication failed
Connection not open
```

Serial Port Read

```
# ======================================================================
# Listen for messages from the Arduino and displays them
# ----------------------------------------------------------------------
# Windows 7, ActivePerl (32 bit)
# install win32::SerialPort module
# ======================================================================

use strict;
use warnings;

require Win32::SerialPort;

my $portname = "COM4";

my $port = Win32::SerialPort->new($portname)
   or die "Can't open serial port (" . $portname . ")\n$^E\n";

# -- for debugging -----------------------------------------------
# prints hardware messages like "Framing Error"
# $port->error_msg(1);
# prints function messages like "Waiting for CTS"
# $port->user_msg(1);
# ----------------------------------------------------------------

$port->initialize();
$port->databits(8);
```

```
$port->baudrate(9600);
$port->parity("none");
$port->stopbits(1);
$port->debug(0);

print "Serial port baudrate: " . $port->baudrate() . "\n";

# define line termination for $port->lookfor()
# Note: Arduino serial.println terminates each line/string sent with a "\r\n"

$port->are_match("\r\n");

# loop forever

my $data;

while(1)
{
   # poll looking for data

   $data = $port->lookfor();

   # remove training carriage return (the default for are_match is "\n")
   # this code is not needed because of the $port->are_match
   # $data =~ s/\r$//;

   # if we get data, print it

   if ($data)
   {
      print "String length:    " . length($data) . "\n";
      print "String Received: (" . $data . ")\n";
   }

   # sleep for a second (look up usleep for a shorter period)

   sleep(1);
}
```

Serial Port Write

```
# ================================================================
# Write messages to the Arduino which displays them on a LCD
# ----------------------------------------------------------------
# Windows 7, ActivePerl (32 bit)
# install win32::SerialPort module
# ================================================================

use strict;
use warnings;

require Win32::SerialPort;

my $portname = "COM4";

my $port = Win32::SerialPort->new($portname,0)
   or die "Can't open serial port (" . $portname . ")\n$^E\n";

# -- for debugging ------------------------------------------
# prints hardware messages like "Framing Error"
# $port->error_msg(1);
# prints function messages like "Waiting for CTS"
```

```
# $port->user_msg(1);
# -------------------------------------------------------------

$port->baudrate(9600);
$port->parity("none");
$port->databits(8);
$port->stopbits(1);
$port->debug(0);

#print "Serial port baudrate  : " . $port->baudrate() . "\n";

$port->lookclear;

my $data = "ABC";

$port->write($data);

$port->close();
```

Test Sine and Cosine

```
#!/usr/local/bin/perl -w

use strict;
use Math::Trig;

my @a = (0,45,90,135,180,225,270,315,360,405);


foreach my $v (@a)
{
   displaySineCosine($v);
}

#-----------------------------------------------------

sub displaySineCosine
{
   my $angle = $_[0];

   my $radian = $angle * (pi/180.0);

   printf "Sine   of %3d  is % 2.6f",$angle,sin($radian);
   printf "Cosine of %3d  is % 2.6f",$angle,cos($radian);
   print "\n";

   return;
}
```

Test Output

```
Sine   of   0  is  0.000000  Cosine of   0  is  1.000000
Sine   of  45  is  0.707107  Cosine of  45  is  0.707107
Sine   of  90  is  1.000000  Cosine of  90  is  0.000000
Sine   of 135  is  0.707107  Cosine of 135  is -0.707107
Sine   of 180  is  0.000000  Cosine of 180  is -1.000000
Sine   of 225  is -0.707107  Cosine of 225  is -0.707107
Sine   of 270  is -1.000000  Cosine of 270  is -0.000000
Sine   of 315  is -0.707107  Cosine of 315  is  0.707107
Sine   of 360  is -0.000000  Cosine of 360  is  1.000000
Sine   of 405  is  0.707107  Cosine of 405  is  0.707107
```

Test Modulus Operator

```
#!/usr/local/bin/perl -w

use strict;

print "\nTest Mod Operator\n";

print "\nWith Integer\n\n";

print "   359 % 360  = " . 359 % 360 . "\n";
print "   360 % 360  = " . 360 % 360 . "\n";
print "   361 % 360  = " . 361 % 360 . "\n";

print "\nWith Floating Point\n\n";

print "   359.1 % 360.0  = " . 359.1 % 360.0 . "\n";
print "   360.1 % 360.0  = " . 360.1 % 360.0 . "\n";
print "   361.1 % 360.0  = " . 361.1 % 360.0 . "\n";

print "\nWith Floating Point Subroutine\n\n";

print "results: " . modFloatingPoint(359.1,360.0) . "\n";
print "results: " . modFloatingPoint(360.1,360.0) . "\n";
print "results: " . modFloatingPoint(361.1,360.0) . "\n";

print"\n";

#-----------------------------------------------------

sub modFloatingPoint
{
   my $n1 = $_[0];
   my $n2 = $_[1];

   my $n = int($n1/$n2);

   return $n1 - ($n * $n2);
}
```

Test Output

```
Test Mod Operator

With Integer

   359 % 360  = 359
   360 % 360  = 0
   361 % 360  = 1

With Floating Point

   359.1 % 360.0  = 359
   360.1 % 360.0  = 0
   361.1 % 360.0  = 1

With Floating Point Subroutine

results: 359.1
results: 0.100000000000023
```

```
results: 1.10000000000002
```

Test Responding To Commands

```
// ================================================================
// Test Responding To Commands
// ================================================================

#include <Wire.h>

#define BAUDRATE  9600

int  pos      = 0;             // servo position
int  posStart = 0;             // servo start posion
int  posEnd   = 0;             // servo end position
char cmd;                      // command

// ================================================================
// setup
// ================================================================

void setup()
{
   // initialize serial port

   Serial.begin(BAUDRATE);
}

// ================================================================
// main loop
// ================================================================

void loop()
{
  // process serial input -----------------------------------------

  if (Serial.available())
  {
    // wait for the entire message to arrive ---------------------

    delay(200);

    // get the command -------------------------------------------

    cmd = Serial.read();

    // return a command response---------------------------------

    if (cmd == 'S')            // ping sweep
    {
      posStart = Serial.parseInt();
      posStart = constrain(posStart,0,179);
      posEnd   = Serial.parseInt();
      posEnd   = constrain(posEnd,0,179);
      Serial.print("S");
      Serial.print(posStart,DEC);
      Serial.print(",");
      Serial.println(posEnd,DEC);
      sendTestPingSweep(posStart,posEnd);
      Serial.println("x");
      Serial.flush();
    }
```

```
    else if (cmd == 'P')       // ping
    {
      Serial.print("p");
      Serial.println(124,DEC);
      Serial.println("x");
      Serial.flush();
    }
    else if (cmd == 'M')       // move servo
    {
      pos = Serial.parseInt();
      pos = constrain(pos,0,179);
      Serial.print("M");
      Serial.println(pos,DEC);
      Serial.println("x");
      Serial.flush();
    }
    else if (cmd == 'C')       // current state
    {
      Serial.print("c");
      Serial.print(45,DEC);
      Serial.print(",");
      Serial.println(90,DEC);
      Serial.println("x");
      Serial.flush();
    }
    else                       // error
    {
      Serial.print("e");
      Serial.println(cmd,DEC);
      Serial.println("x");
      Serial.flush();
    }

    // remove the remaining characters in the buffer (if any) -----

    emptySerialInputBuffer();
  }
}

// ================================================================
// subroutines
// ================================================================

void sendTestPingSweep(int s, int e)
{
   for(int i = s; i<= e; i++)
   {
     Serial.print("s");
     Serial.print(i,DEC);
     Serial.print(",");
     Serial.println(i*10,DEC);
   }
}

void emptySerialInputBuffer()
{
   while(Serial.read() != -1);
   return;
}
```

I2C Scanner

```
// --------------------------------------------------------------
// i2c_scanner
//
// This program (or code that looks like it)
// can be found in many places.
// For example on the Arduino.cc forum.
// The original author is not know.
//
// This sketch tests the standard 7-bit addresses
// from 0 to 127. Devices with higher bit address
// might not be seen properly.
//
// Adapted to be as simple as possible by Arduino.cc user Krodal
//
// June 2012
// Using Arduino 1.0.1
// --------------------------------------------------------------

#include <Wire.h>

void setup()
{
  Wire.begin();

  Serial.begin(9600);
  Serial.println("\nI2C Scanner");
}

void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 0; address <= 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println(" !");

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
```

```
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(8000);           // wait 8 seconds for next scan
}
```

Read Serial Data and Display on LCD

```
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>

#define I2C_ADDR      0x3F  // Define the PCF8574A I2C Address

#define BACKLIGHT_PIN 3

#define En_pin        2
#define Rw_pin        1
#define Rs_pin        0
#define D4_pin        4
#define D5_pin        5
#define D6_pin        6
#define D7_pin        7

LiquidCrystal_I2C lcd(I2C_ADDR,En_pin,Rw_pin,Rs_pin,D4_pin,D5_pin,D6_pin,D7_pin);

void setup()
{
  lcd.begin(20,4);
  lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
  lcd.setBacklight(HIGH);
  lcd.home();
  lcd.print("Test Input:");
  Serial.begin(9600);
}

void loop()
{
  // when characters arrive over the serial port...
  if (Serial.available())
  {
    // wait a bit for the entire message to arrive
    delay(100);
    // clear the screen
    //lcd.clear();
    // read all the available characters
    while (Serial.available() > 0)
    {
      // display each character to the LCD
      lcd.write(Serial.read());
    }
  }
}
```

Appendix 1 – Project Arduino Sketch

```
// ================================================================
// Map a Grid using an ultrasonic Sensor
// ================================================================
//
// Respond to commands sent from a control program and return data
// if any.
//
// ----------------------------------------------------------------
//
// Commands:
//
// Spos1,pos2 -- Do a ping sweep from servo position 1 to servo
//               position 2 and return the resultant distances.
//               A ping is done at each position including the
//               start and stop positions. The command format is
//               ['S' + starting servo position + "," +
//               servo end position]. For example, "S100,200".
//
//               The returned distances are in the format ['s'
//               + servo position + ',' + distance]. For example,
//               "s120,34".
//
//               After the command ends, "x" is sent indicating
//               the end of the command.
//
// Mpos       -- Move the servo to a specified position.
//               The command format is ['M' + servo position].
//               For example, "M120".
//
//               After the command ends, "x" is sent indicating
//               the end of the command.
//
// P          -- Do a single ping for distance. The command format
//               is ['P']. The current servo position is used for
//               the ping.
//
//               The returned distance is in the format ['p'
//               + servo position + ',' + distance]. For example,
//               "p120,34".
//
//               After the command ends, "x" is sent indicating
//               the end of the command.
//
// C          -- Return the current position and distance.
//               The command format is ['C']. The returned
//               position and distance is in the format ['c' +
//               servo position + ',' + distance]. For example,
//               "c120,34".
//
//               After the command ends, "x" is sent indicating
//               the end of the command.
//
// Commands are an uppercase character + 0 or more parameters
// separated by commas. Responses (if any) are a lowercase
// character + 0 or more parameters seperated by commas.
//
// ----------------------------------------------------------------
//
// Notes:
```

```
//
// 1. The distance zero (0) indicates no object was
//    detected within the range of the sensor.
//
// 2. The arduino "println" terminates each string sent to
//    the command program with a "\r\n". (See the Arduino
//    documentation for more information.)
//
// 3. Once a command has been recognized, it starts executing and
//    cannot be interrupted. After the command finishes executing
//    any characters in the input serial buffer are destroyed.
//
// 4. If you do not have an LCD display, set LCDYES = 0 or remove
//    the LCD code.
//
// 5. This Arduino sketch does little to no error checking.
//    It is up to the command program to send the correct data
//    in the correct format.
//
// 6. If an unknown command is received, the control program is send
//    an error response. The error message format is
//    [the character 'e' + the command (character) in decimal
//    format]. For example, "e66".  Note: The command is in
//    decimal format because it may no be a printable character.
//
//    After the error response, the string "x" is sent indicating
//    the end of the command.
//
// ================================================================

#include <Servo.h>
#include <Wire.h>
#include <LCD.h>
#include <LiquidCrystal_I2C.h>
#include <Ultrasonic.h>

// ================================================================
// housekeeping
//=================================================================

// ----------------------------------------------------------------
// defines
// ----------------------------------------------------------------

// serial port
// (300, 600, 1200, 2400, 4800, 9600, 14400, 19200, 28800, 38400,
//  57600, 115200)

#define BAUDRATE      9600

// ultrasonic

#define  TRIG_PIN     12
#define  ECHO_PIN     13

// servo

#define SERVO_PIN     9

// LCD

#define I2C_ADDR      0x3F     // I2C Address
#define BACKLIGHT_PIN 3
```

```
#define RS_PIN        0
#define RW_PIN        1
#define EN_PIN        2
#define D4_PIN        4
#define D5_PIN        5
#define D6_PIN        6
#define D7_PIN        7
#define LCDYES        true     // LCD true/false flag

// ---------------------------------------------------------------
// objects and variables
// ---------------------------------------------------------------

Ultrasonic ultra(TRIG_PIN,ECHO_PIN);

LiquidCrystal_I2C lcd(I2C_ADDR,EN_PIN,RW_PIN,RS_PIN,
                      D4_PIN,D5_PIN,D6_PIN,D7_PIN);

Servo servo;

long dis      = ;             // distance to object
int  pos      = ;             // servo position
int  posStart = ;             // servo start position
int  posEnd   = ;             // servo end position
char cmd;                     // command

// ===============================================================
// setup
//================================================================

void setup()
{
   // initialize serial port

   Serial.begin(BAUDRATE);

   // initialize servo

   servo.attach(SERVO_PIN);
   servo.write();

  // initialize LCD display

  if (LCDYES)
  {
     // initialize the lcd
     lcd.begin(2,4);
     // Switch on the backlight
     lcd.setBacklightPin(BACKLIGHT_PIN,POSITIVE);
     lcd.setBacklight(HIGH);
     // set initial display
     lcd.home();
     lcd.print("Map a Grid");
     lcd.setCursor(,1);
     lcd.print("Dis ");
     lcd.setCursor(,2);
     lcd.print("Pos ");
     lcd.setCursor(,3);
     lcd.print("Cmd ");
   }
}

// ===============================================================
```

```
// main loop
// ================================================================

void loop()
{
  // process serial input buffer ---------------------------------

  if (Serial.available())
  {
    // wait for the entire message to arrive ----------------------

    delay(20);

    // get command ------------------------------------------------

    cmd = Serial.read();

    // ping sweep command -----------------------------------------

    if (cmd == 'S')
    {
      posStart = Serial.parseInt();
      posStart = constrain(posStart,0,179);
      posEnd   = Serial.parseInt();
      posEnd   = constrain(posEnd,0,179);
      lcdCommand(cmd);
      pingSweep(posStart,posEnd);
      Serial.println("x");
      Serial.flush();
    }

    // ping command -----------------------------------------------

    else if (cmd == 'P')
    {
      dis = pingDistance();
      lcdCommand(cmd);
      lcdPingDistance(dis);
      Serial.print("p");
      Serial.print(pos,DEC);
      Serial.print(",");
      Serial.println(dis,DEC);
      Serial.println("x");
      Serial.flush();
    }

    // move servo to a specified position command -----------------

    else if (cmd == 'M')
    {
      pos = Serial.parseInt();
      pos = constrain(pos,0,179);
      setServoPosition(pos);
      lcdCommand(cmd);
      lcdServoPosition(pos);
      Serial.print("m");
      Serial.println(pos,DEC);
      Serial.println("x");
      Serial.flush();
    }

    // current state command --------------------------------------
```

```
    else if (cmd == 'C')
    {
      lcdCommand(cmd);
      Serial.print("c");
      Serial.print(pos,DEC);
      Serial.print(",");
      Serial.println(dis,DEC);
      Serial.println("x");
      Serial.flush();
    }

    // error ----------------------------------------------------

    else
    {
      Serial.print("e");
      Serial.println(cmd,DEC);
      Serial.println("x");
      Serial.flush();
    }

    // remove the remaining characters in the buffer (if any) -----

    emptySerialInputBuffer();
  }

  // wait before checking for the next command --------------------

  delay(200);
}

// ================================================================
// subroutines
// ================================================================

// ----------------------------------------------------------------
// empty the serial input buffer
// ----------------------------------------------------------------

void emptySerialInputBuffer()
{
  while(Serial.read() != -1);
  return;
}

// ----------------------------------------------------------------
// get distance to an object (0 if no object)
// ----------------------------------------------------------------

long pingDistance()
{
  long d = ultra.Ranging(INC); // distance to object in inches
  d = constrain(d,0,255);      // constrain the distance
  delay(200);                  // let the echo dissipate
  return d;
}

// ----------------------------------------------------------------
// set the servo position
// ----------------------------------------------------------------

void setServoPosition(int pos)
{
```

```
  servo.write(pos);             // set servo position
  delay(200);                   // wait for servo to reach position
  return;
}


// ---------------------------------------------------------------
// ping sweep
// ---------------------------------------------------------------

void pingSweep(int start, int end)
{
  for(pos = start; pos <= end; pos++)
  {
    setServoPosition(pos);
    dis = pingDistance();
    lcdServoPosition(pos);
    lcdPingDistance(dis);
    Serial.print("s");
    Serial.print(pos,DEC);
    Serial.print(",");
    Serial.println(dis,DEC);
    Serial.flush();
  }
  return;
}

// ---------------------------------------------------------------
// display servo position
// ---------------------------------------------------------------

void lcdServoPosition(int pos)
{
  if (LCDYES)
  {
    lcd.setCursor(4,2);         // set cursor position
    lcd.print("     ");         // clear display
    lcd.setCursor(4,2);         // set cursor position
    lcd.print(pos,DEC);         // display servo position
  }
  return;
}

// ---------------------------------------------------------------
// display command
// ---------------------------------------------------------------

void lcdCommand(char cmd)
{
  if (LCDYES)
  {
    lcd.setCursor(4,3);         // set cursor position
    lcd.print("          ");    // clear display
    lcd.setCursor(4,3);         // set cursor position
    lcd.print(cmd);             // display command
  }
  return;
}

// ---------------------------------------------------------------
// display ping distance
// ---------------------------------------------------------------
```

```
void lcdPingDistance(long distance)
{
  if (LCDYES)
  {
    lcd.setCursor(4,1);         // set cursor position
    lcd.print("    ");          // clear display
    lcd.setCursor(4,1);         // set cursor position
    lcd.print(distance,DEC);    // display distance
  }
  return;
}
```

Appendix 2 – Project Control Program

```
#!/usr/bin/perl -w
# ============================================================================
# Map a Grid Project - Control Program
# ----------------------------------------------------------------------------
#
# Send commands to the Arduino and process the returned data.
# Arduino sketch are available that work with the control program.
#
# Note: The lower left corner of the grid is 0,0
#
# ============================================================================

use strict;
use warnings;
use Math::Trig;
use Tkx;

use communications;

# ----------------------------------------------------------------------------
# global constants and variables
# ----------------------------------------------------------------------------

use constant
{
   DEFAULTBAUDRATE =>   9600,  # default - baudrate
   DEFAULTPORT     => 'COM4',  # default - comm port
   DEFAULTPOS1     =>      0,  # default - position 1
   DEFAULTPOS2     =>     90,  # default - position 2
   DEFAULTSAA      =>      0,  # default - sensor adjustment angle
   DEFAULTSERVOX   =>    -12,  # default servo X coordinate
   DEFAULTSERVOY   =>    -12,  # default servo Y coordinate
   GCELLX          =>     12,  # grid cell X size     (inches)
   GCELLY          =>     12,  # grid cell Y size     (inches)
   GCOLS           =>     10,  # grid columns         (x axis)
   GROWS           =>     10,  # grid rows            (y axis)
   TCOLS           =>     66,  # text display columns
   TROWS           =>     14,  # text display rows
};

my $BAUDRATE;                   # baudrate text widget
my $COMM;                       # communication object
my $DEBUG           =      0;  # debug flag - display debug messages
my $FIRSTTEXTLINE   =      1;  # first line of text flag
my @GRIDCOUNT       =     ();  # This array is use during a ping
                               # sweep to accumulate the number
                               # of times a cell is found occupied
my @GRIDFLAG        =     ();  # For each ping in a ping sweep
                               # an array element (grid cell) is
                               # set to true (1) if it is occupied
                               # and false (0)if it is not. The
                               # value (true/false) determines if
                               # a grid cell's count is incremented.
my @GRIDWIDGET      =     ();  # grid display widgets
my $PORT;                      # comm port entry widget
my $P1ENTRY;                   # position 1 entry widget
my $P2ENTRY;                   # position 2 entry widget
my $SAAENTRY;                  # sensor adjustment angle entry widget
my $TESTFLAG        =      1;  # test flag - display test buttons
```

```
my $TEXT;                       # text display widget
my $SXENTRY;                    # servo X coordinate entry widget
my $SYENTRY;                    # servo Y coordinate entry widget
my $VERBOSE         =      0;  # display extra processing messages
my $VERBOSEBUTTON;              # verbose button widget

my $GXMIN = 0;                  # grid minimum X coordinate
my $GXMAX = GCELLX * GCOLS;     # grid maximum X coordinate
my $GYMIN = 0;                  # grid minimum Y coordinate
my $GYMAX = GCELLY * GROWS;     # grid maximum Y coordinate


# angles used to adjust the sensor angle to test if a cell
# is occupied (see the sensor documentation for more information)

my @SANGLE = (7,3.5,0,-3.5,-7);

# test ping sweep returned data

my @TESTPINGSWEEP = ('s45,25','s45,114','s45,178',
                     's45,160','s45,25',
                     's46,114','s47,114',
                     's80,127','s10,127');

# -----------------------------------------------------------------
# define fonts
# -----------------------------------------------------------------

my $cfont6   = Tkx::font_create(-family=>'Courier', -size=>6);
my $cfont8   = Tkx::font_create(-family=>'Courier', -size=>8);
my $cfont10  = Tkx::font_create(-family=>'Courier', -size=>10);
my $cfont12  = Tkx::font_create(-family=>'Courier', -size=>12);
my $cfont14  = Tkx::font_create(-family=>'Courier', -size=>14);
my $cfont16  = Tkx::font_create(-family=>'Courier', -size=>16);
my $cfont18  = Tkx::font_create(-family=>'Courier', -size=>18);
my $cfont20  = Tkx::font_create(-family=>'Courier', -size=>20);
my $cfont6b  = Tkx::font_create(-family=>'Courier', -size=>6,
                          -weight=>'bold');
my $cfont8b  = Tkx::font_create(-family=>'Courier', -size=>8,
                          -weight=>'bold');
my $cfont10b = Tkx::font_create(-family=>'Courier', -size=>10,
                          -weight=>'bold');
my $cfont12b = Tkx::font_create(-family=>'Courier', -size=>12,
                          -weight=>'bold');
my $cfont14b = Tkx::font_create(-family=>'Courier', -size=>14,
                          -weight=>'bold');
my $cfont16b = Tkx::font_create(-family=>'Courier', -size=>16,
                          -weight=>'bold');
my $cfont18b = Tkx::font_create(-family=>'Courier', -size=>18,
                          -weight=>'bold');
my $cfont20b = Tkx::font_create(-family=>'Courier', -size=>20,
                          -weight=>'bold');

my $hfont6   = Tkx::font_create(-family=>'Helvetica', -size=>6);
my $hfont8   = Tkx::font_create(-family=>'Helvetica', -size=>8);
my $hfont10  = Tkx::font_create(-family=>'Helvetica', -size=>10);
my $hfont12  = Tkx::font_create(-family=>'Helvetica', -size=>12);
my $hfont14  = Tkx::font_create(-family=>'Helvetica', -size=>14);
my $hfont16  = Tkx::font_create(-family=>'Helvetica', -size=>16);
my $hfont18  = Tkx::font_create(-family=>'Helvetica', -size=>18);
my $hfont20  = Tkx::font_create(-family=>'Helvetica', -size=>20);
my $hfont6b  = Tkx::font_create(-family=>'Helvetica', -size=>6,
                          -weight=>'bold');
```

```
my $hfont8b  = Tkx::font_create(-family=>'Helvetica', -size=>8,
                        -weight=>'bold');
my $hfont10b = Tkx::font_create(-family=>'Helvetica', -size=>10,
                        -weight=>'bold');
my $hfont12b = Tkx::font_create(-family=>'Helvetica', -size=>12,
                        -weight=>'bold');
my $hfont14b = Tkx::font_create(-family=>'Helvetica', -size=>14,
                        -weight=>'bold');
my $hfont16b = Tkx::font_create(-family=>'Helvetica', -size=>16,
                        -weight=>'bold');
my $hfont18b = Tkx::font_create(-family=>'Helvetica', -size=>18,
                        -weight=>'bold');
my $hfont20b = Tkx::font_create(-family=>'Helvetica', -size=>20,
                        -weight=>'bold');

my $tfont6   = Tkx::font_create(-family=>'Times', -size=>6);
my $tfont8   = Tkx::font_create(-family=>'Times', -size=>8);
my $tfont10  = Tkx::font_create(-family=>'Times', -size=>10);
my $tfont12  = Tkx::font_create(-family=>'Times', -size=>12);
my $tfont14  = Tkx::font_create(-family=>'Times', -size=>14);
my $tfont16  = Tkx::font_create(-family=>'Times', -size=>16);
my $tfont18  = Tkx::font_create(-family=>'Times', -size=>18);
my $tfont20  = Tkx::font_create(-family=>'Times', -size=>20);
my $tfont6b  = Tkx::font_create(-family=>'Times', -size=>6,
                        -weight=>'bold');
my $tfont8b  = Tkx::font_create(-family=>'Times', -size=>8,
                        -weight=>'bold');
my $tfont10b = Tkx::font_create(-family=>'Times', -size=>10,
                        -weight=>'bold');
my $tfont12b = Tkx::font_create(-family=>'Times', -size=>12,
                        -weight=>'bold');
my $tfont14b = Tkx::font_create(-family=>'Times', -size=>14,
                        -weight=>'bold');
my $tfont16b = Tkx::font_create(-family=>'Times', -size=>16,
                        -weight=>'bold');
my $tfont18b = Tkx::font_create(-family=>'Times', -size=>18,
                        -weight=>'bold');
my $tfont20b = Tkx::font_create(-family=>'Times', -size=>20,
                        -weight=>'bold');

my $btfont = $hfont10b;    # button font
my $gdfont = $cfont12;     # grid display font
my $plfont = $hfont8b;     # parameter label font
my $pvfont = $cfont10;     # parameter entry font
my $txfont = $cfont10;     # text window font
my $wtfont = $hfont10b;    # window title font

# ---------------------------------------------------------------------
# main window and frame
# ---------------------------------------------------------------------

my $mwrow = 0;                 # row for main window grid

my $mw = Tkx::widget->new(".");

$mw->g_wm_title("Map a Grid");

my $f = $mw->new_frame(-relief=>'groove', -borderwidth=>2);
$f->g_grid(-row=>$mwrow++, -column=>0, -sticky=>'nsew');

# ---------------------------------------------------------------------
# title
# ---------------------------------------------------------------------
```

```
my $l;

$l = $f->new_label(-text=>'Map a Grid',-font=>$wtfont);
$l->g_grid(-row=>$mwrow++, -column=>0, -padx=>4, -pady=>4,
           -sticky=>'new');

# ------------------------------------------------------------------
# container frames
# ------------------------------------------------------------------

# ---- buttons and grid display container frame

my $f1;
$f1 = $f->new_frame();
$f1->g_grid(-row=>$mwrow++, -column=>0);

#      buttons container frame

my $bf;
$bf = $f1->new_frame(-relief=>'groove', -borderwidth=>2);
$bf->g_grid(-row=>0, -column=>0, -sticky=>'nsew');

#      grid display container frame

my $gf;
$gf = $f1->new_frame(-relief=>'groove', -borderwidth=>2);
$gf->g_grid(-row=>0, -column=>1, -sticky=>'new');

# ---- test botton container frame

my $tbf;
if ($TESTFLAG)
{
   $tbf = $f->new_frame(-relief=>'groove', -borderwidth=>2);
   $tbf->g_grid(-row=>$mwrow++, -column=>0, -sticky=>'nsew');
}

# ---- parameter container frame

my $f2;
$f2 = $f->new_frame(-relief=>'groove', -borderwidth=>2);
$f2->g_grid(-row=>$mwrow++, -column=>0, -sticky=>'nsew');


# ---- message area container frame

my $f3;
$f3 = $f->new_frame(-relief=>'flat');
$f3->g_grid(-row=>$mwrow++, -column=>0, -sticky=>'nsew');

# ------------------------------------------------------------------
# add button
# ------------------------------------------------------------------

my $b;                          # button widget
my $bc = 0;                     # button row count

$b = $bf->new_button(-text=>'Ping Sweep', -font=>$btfont, -takefocus=>0,
                     -command=>\&ping_sweep);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'new');

$bc++;
```

```
$b = $bf->new_button(-text=>'Single Ping', -font=>$btfont, -takefocus=>0,
                     -command=>\&ping);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-text=>'Move Sensor', -font=>$btfont, -takefocus=>0,
                     -command=>\&move);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-text=>'Current' . "\n" . 'Pos & Dist',
                     -font=>$btfont, -takefocus=>0, -command=>\¤t);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-text=>'Clear Messages', -font=>$btfont, -takefocus=>0,
                     -command =>\&clear_text_display);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-text=>"Initialize\nGrid Display",
                     -font=>$btfont, -takefocus=>0,
                     -command =>\&init_grid_display);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-text=>"Reconnect to\nArduino",
                     -font=>$btfont, -takefocus=>0,
                     -command =>\&connect_reconnect);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-font=>$btfont, -takefocus=>0,
                     -command=>\&toggle_verbose);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');
if ($VERBOSE)
{ $b->configure(-text => 'Verbose Off'); }
else
{ $b->configure(-text => 'Verbose On'); }
$VERBOSEBUTTON = $b;

$bc++;

$b = $bf->new_button(-text=>'Help', -font=>$btfont, -takefocus=>0,
                     -command=>\&help);
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

$bc++;

$b = $bf->new_button(-text=>'Exit', -font=>$btfont, -takefocus=>0,
                     -command => sub{ $mw->g_destroy });
$b->g_grid(-row=>$bc, -column=>0, -padx=>4, -pady=>4, -sticky=>'ew');

# ---------------------------------------------------------------------
# add test buttons
# ---------------------------------------------------------------------
```

```
if ($TESTFLAG)
{
   $b = $tbf->new_button(-text=>"Test Grid\nCount Display",
                         -font=>$btfont, -takefocus=>0,
                         -command =>\&test_grid_count_display);
   $b->g_grid(-row=>0, -column=>0,-padx=>4, -pady=>4, -sticky=>'nsew');

   $b = $tbf->new_button(-text=>"Test Parameters",
                         -font=>$btfont, -takefocus=>0,
                         -command =>\&test_parameters);
   $b->g_grid(-row=>0, -column=>1, -padx=>4, -pady=>4, -sticky=>'nsew');

   $b = $tbf->new_button(-text=>"Display Grid Flags",
                         -font=>$btfont, -takefocus=>0,
                         -command =>\&test_display_grid_flags);
   $b->g_grid(-row=>0, -column=>2, -padx=>4, -pady=>4, -sticky=>'nsew');

   $b = $tbf->new_button(-text=>"Test Ping\nSweep Data",
                         -font=>$btfont, -takefocus=>0,
                         -command =>\&test_ping_sweep);
   $b->g_grid(-row=>0, -column=>3, -padx=>4, -pady=>4, -sticky=>'nsew');
}

# ------------------------------------------------------------------
# add grid cells text widgets to the grid container frame
# ------------------------------------------------------------------

for (my $c = 0; $c < GCOLS; $c++)
{
   for (my $r = 0; $r < GROWS; $r++)
   {
      my $cw = $gf->new_text(-height=>1, -width=>4,
                             -padx=>2, -pady=>6, -font=>$gdfont);
      $cw->g_grid(-row=>$r, -column=>$c, -sticky=>'nsew');
      $GRIDWIDGET[$c][GROWS - $r - 1] = $cw;
   }
}

# ------------------------------------------------------------------
# add parameters
# ------------------------------------------------------------------

my $ff  = $f2->new_frame(-relief=>'flat');
$ff->g_grid(-row=>0, -column=>0, -sticky=>'nsew');

# ------------------------------------------------------------------
# servo/sensor position widgets
# ------------------------------------------------------------------

my $fff;

$fff = $ff->new_frame();
$fff->g_grid(-row=>0, -column=>0, -padx=>4, -pady=>4, -sticky=>'nsew');

#---- position 1

$l = $fff->new_label(-text=>'Pos 1', -font=>$plfont);
$l->g_grid(-row=>0, -column=>0, -padx=>4, -pady=>4, -sticky=>'nsew');
$P1ENTRY = $fff->new_entry(-font=>$pvfont, -width=>8,
                           -borderwidth=>2, -relief=>'groove');
$P1ENTRY->g_grid(-row=>0, -column=>1, -sticky=>'nsew');
```

```
#---- position 2

$l = $fff->new_label(-text=>'Pos 2', -font=>$plfont);
$l->g_grid(-row=>1, -column=>0, -padx=>4, -pady=>4, -sticky=>'nsew');
$P2ENTRY= $fff->new_entry(-font=>$pvfont, -width=>8,
                          -borderwidth=>2, -relief=>'groove');
$P2ENTRY->g_grid(-row=>1, -column=>1, -sticky=>'nsew');

# ------------------------------------------------------------------
# sensor adjustment angle widgets
# ------------------------------------------------------------------

$fff = $ff->new_frame();
$fff->g_grid(-row=>0, -column=>1, -padx=>4, -pady=>4, -sticky=>'nsew');

$l = $fff->new_label(-text=>"Sensor Adj Angle",-font=>$plfont);
$l->g_grid(-row=>0, -column=>0, -padx=>4, -pady=>4,-sticky=>'nsew');
$SAAENTRY = $fff->new_entry(-font=>$pvfont, -width=>8,
                         -borderwidth=>2, -relief=>'groove');
$SAAENTRY->g_grid(-row=>1, -column=>0, -sticky=>'nsew');

# ------------------------------------------------------------------
# servo/sensor coordinate widgets
# ------------------------------------------------------------------

$fff  = $ff->new_frame();
$fff->g_grid(-row=>0, -column=>2, -padx=>4, -pady=>4, -sticky=>'nsew');

#---- servo/sensor X

$l = $fff->new_label(-text=>'Servo X',-font=>$plfont);
$l->g_grid(-row=>0, -column=>0, -padx=>4, -pady=>4,-sticky=>'nsew');
$SXENTRY = $fff->new_entry(-font=>$pvfont, -width=>8,
                         -borderwidth=>2, -relief=>'groove');
$SXENTRY->g_grid(-row=>0, -column=>1, -sticky=>'nsew');

#---- Servo/sensor Y

$l = $fff->new_label(-text=>'Servo Y',-font=>$plfont);
$l->g_grid(-row=>1, -column=>0, -padx=>4, -pady=>4, -sticky=>'nsew');
$SYENTRY= $fff->new_entry(-font=>$pvfont, -width=>8,
                         -borderwidth=>2, -relief=>'groove');
$SYENTRY->g_grid(-row=>1, -column=>1, -sticky=>'nsew');

# ------------------------------------------------------------------
# add comm port widgets
# ------------------------------------------------------------------

$fff = $ff->new_frame();
$fff->g_grid(-row=>0, -column=>3, -padx=>4, -pady=>4, -sticky=>'nsew');

#---- port

$l = $fff->new_label(-text=>'Port',-font=>$plfont);
$l->g_grid(-row=>0, -column=>0, -padx=>4, -pady=>4, -sticky=>'nsew');
$PORT = $fff->new_entry(-font=>$pvfont, -width=>8,
                         -borderwidth=>2, -relief=>'groove');
$PORT->g_grid(-row=>0, -column=>1, -sticky=>'nsew');

#---- baudrate

$l = $fff->new_label(-text=>'   Baudrate',-font=>$plfont);
$l->g_grid(-row=>1, -column=>0, -padx=>4, -pady=>4, -sticky=>'nsew');
```

```
$BAUDRATE = $fff->new_entry(-font=>$pvfont, -width=>8,
                            -borderwidth=>2, -relief=>'groove');
$BAUDRATE->g_grid(-row=>1, -column=>1, -sticky=>'nsew');

# ------------------------------------------------------------------
# add message area text widget and scrollbar
# ------------------------------------------------------------------

$ff = $f3->new_frame(-relief=>'groove', -borderwidth=>2);

$TEXT = $ff->new_text(-font=>$txfont, -padx=>4, -pady=>4,
                      -borderwidth=>0, -relief=>'flat',
                      -height=>TROWS, -width=>TCOLS);

my $sby = $ff->new_scrollbar(-orient=>'v', -command=>[$TEXT,'yview']);

$TEXT->configure(-yscrollcommand =>[$sby,'set']);

$TEXT->g_grid(-row=>0, -column=>0);

$sby->g_grid(-row=>0, -column=>1, -padx=>2, -pady=>2, -sticky=>'ns');

$ff->g_grid(-row=>0, -column=>0, -sticky=>'nsew');

# ------------------------------------------------------------------
# initialize
# ------------------------------------------------------------------

$BAUDRATE->m_insert('end',DEFAULTBAUDRATE);
$P1ENTRY->m_insert('end',DEFAULTPOS1);
$P2ENTRY->m_insert('end',DEFAULTPOS2);
$PORT->m_insert('end',DEFAULTPORT);
$SAAENTRY->m_insert('end',DEFAULTSAA);
$SXENTRY->m_insert('end',DEFAULTSERVOX);
$SYENTRY->m_insert('end',DEFAULTSERVOY);

init_grid_flag_array();
init_grid_count_array();
update_grid_count_display();
write_text_message('Text Messages');

# ------------------------------------------------------------------
# connect to the Arduino
# ------------------------------------------------------------------

$COMM = new communications();

my ($status,$aref) = $COMM->open(DEFAULTPORT,DEFAULTBAUDRATE);

if (!$status)
{
   clear_text_display();
   write_text_message('Connection to Arduino failed');
   write_text_message_array($aref);
}

# ------------------------------------------------------------------
# main loop
# ------------------------------------------------------------------

Tkx::MainLoop();
```

```
# ====================================================================
# ====================================================================
# subroutines
# ====================================================================
# ====================================================================

# --------------------------------------------------------------------
# connect/reconnect to Arduino
# --------------------------------------------------------------------

sub connect_reconnect
{
   clear_text_display();
   write_text_message('Connect/re-connect to the Arduino');

   # get and verify communication parameters

   my $port = $PORT->get();    # port

   $port =~ s/^\s+//;          # remove whitespace
   $port =~ s/\s+$//;          # remove whitespace

   if ($port eq '')
   {
      write_text_message("Error: no Port specified");
      return;
   }

   my $aref;
   my $baudrate;
   my $status;

   ($status,$baudrate) = verify_integer($BAUDRATE->get());
   if (!$status)
   {
      write_text_message("Error: Baudrate must be an integer (no sign)");
      return;
   }

   # close the connection (if there is one)

   $COMM->close();

   # open a connection

   ($status,$aref) = $COMM->open($port,$baudrate);
   if (!$status)
   {
      write_text_message_array($aref);
      return;
   }

   # success message

   write_text_message("Connected to Arduino");

   return;
}

# --------------------------------------------------------------------
# do a ping sweep from a start to and ending position
# return the position and distance at each step
# --------------------------------------------------------------------
```

```
sub ping_sweep
{
   my $pos1;                    # servo start position
   my $pos2;                    # servo end position
   my $saa;                     # sensor adjustment angle
   my $servox;                  # servo/sensor X coordinate
   my $servoy;                  # servo/sensor T coordinate
   my $status;                  # returned status

   clear_text_display();
   write_text_message('Start Ping Sweep');

   # verify parameters

   ($status,$pos1) = verify_integer($P1ENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Pos 1 must be an integer (no sign)");
      return;
   }

   ($status,$pos2) = verify_integer($P2ENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Pos 2 must be an integer (no sign)");
      return;
   }

   if ($pos2 < $pos1)
   {
      write_text_message("Error: Pos 2 < Pos 1");
      return;
   }

   ($status,$saa) = verify_integer($SAAENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Sensor Adj Angle must be an integer (no sign)");
      return;
   }

   ($status,$servox) = verify_signed_integer($SXENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Servo X must be an integer");
      return;
   }

    ($status,$servoy) = verify_signed_integer($SYENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Servo Y must be an integer");
      return;
   }

   #--- debug ----------------------------------------------------
   # write_text_message("Pos 1   = $pos1");
   # write_text_message("Pos 2   = $pos2");
   # write_text_message("SAA     = $saa");
   # write_text_message("Servo X = $servox");
   # write_text_message("Servo Y = $servoy");
   # --------------------------------------------------------------
```

```
   # send command

   write_text_message('Sending Command');

   ($status,$aref) = $COMM->sendMessage('S' . $pos1 . ',' . $pos2);
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   # receive reply

   write_text_message('Receiving Reply Data');

   ($status,$aref) = $COMM->receiveReply();
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   if ($DEBUG) { write_text_message_array($aref); }

   # process the reply data

   write_text_message('Processing Reply Data');

   process_ping_sweep_data($aref,$saa,$servox,$servoy);

   # update the grid display

   write_text_message('Updating Grid Display');

   update_grid_count_display();

   write_text_message('End of Ping Sweep');

   return;
}

# ---------------------------------------------------------------------
# process ping sweep data
# ---------------------------------------------------------------------

sub process_ping_sweep_data
{
   my $aref   = $_[0];
   my $saa    = $_[1];
   my $servox = $_[2];
   my $servoy = $_[3];

   init_grid_count_array();

   update_grid_count_display();

   foreach my $str (@$aref)
   {
      if ($str =~ /^x/) { last; }

      if (!process_a_ping($str,$saa,$servox,$servoy)) { return 0; }
   }
```

```
   return 1;
}

# ---------------------------------------------------------------------
# process a single ping from a ping sweep
# ---------------------------------------------------------------------

sub process_a_ping
{
   my $str    = $_[0];           # reply string
   my $saa    = $_[1];           # sensor adjustment angle
   my $servox = $_[2];           # servo X
   my $servoy = $_[3];           # servo y

   if ($VERBOSE) { write_text_message("Processing reply $str"); }

   if ($str =~ /^s(\d+),(\d+)$/)
   {
      my $pos = $1;            # servo position
      my $dst = $2;            # object distance

      if ($DEBUG) { write_text_message("$str pos=$pos  dst=$dst"); }

      if ($dst == 0) { return 1; }    # no object detected?

      # initialize the grid flag array

      init_grid_flag_array();

      # calculate the coordinates of the object

      my $x;                   # object's X coordinate
      my $y;                   # object's Y coordinate
      my $c;                   # object's grid column
      my $r;                   # object's grid row
      my $sa;                  # adjusted selsor angle

      foreach my $a (@SANGLE)
      {
         # calculate adjusted sensor angle to the object

         $sa = ($pos + $saa +$a) % 360;

         # calculate the object's coordinates

         ($x,$y) = calculate_coordinates($servox,$servoy,$sa,$dst);

         if ($DEBUG) { write_text_message("$str a=$sa x=$x y=$y"); }

         # is it outside the grid?

         if ($x < $GXMIN || $x > $GXMAX ||
             $y < $GYMIN || $y > $GYMAX)
         {
            if ($DEBUG) { write_text_message("$str outside the grid"); }
            next;
         }

         # calculate the object's cell

         $c = int($x / GCELLX);   # cell column
         $r = int($y / GCELLY);   # cell row
```

```
         if ($DEBUG) { write_text_message("$str col=$c row=$r"); }

         # set the cell flag

         $GRIDFLAG[$c][$r] = 1;
      }

      # increment the grid cell occupied count

      increment_grid_count_array();

      return 1;
   }

   write_text_message("Error: bad ping sweep data ($str)");
   return 0;
}

# ---------------------------------------------------------------------
# calculate coordinates
# given:
#  origin x,y coordinates
#  angle (clockwise angle from an axis parallel to the grid's X axis)
#  distance to object from origin
# ---------------------------------------------------------------------

sub calculate_coordinates
{
   my $ox = $_[0];             # origin X
   my $oy = $_[1];             # origin Y
   my $a  = $_[2];             # angle  (degrees)
   my $d  = $_[3];             # distance

   my $r = $a * (pi/180);      # radians

   my $x = $ox + (cos($r) * $d);
   my $y = $oy + (sin($r) * $d);

   return ($x,$y);
}

# ---------------------------------------------------------------------
# do a single ping using the currrent servo/sensor position
# return the position and distance
# ---------------------------------------------------------------------

sub ping
{
   my $pos1;
   my $saa;
   my $servox;
   my $servoy;
   my $status;

   clear_text_display();
   write_text_message('Ping');

   # verify parameters

   ($status,$pos1) = verify_integer($P1ENTRY->get());
   if (!$status)
   {
```

```
      write_text_message("Error: Pos 1 must be an integer (no sign)");
      return;
   }

   ($status,$saa) = verify_integer($SAAENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Sensor Adj Angle must be an integer (no sign)");
      return;
   }

   ($status,$servox) = verify_signed_integer($SXENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Servo X must be an integer");
      return;
   }

   ($status,$servoy) = verify_signed_integer($SYENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Servo Y must be an integer");
      return;
   }

   #--- debug ------------------------------------------------------
   # write_text_message("Pos 1   = $pos1");
   # write_text_message("SAA     = $saa");
   # write_text_message("Servo X = $servox");
   # write_text_message("Servo Y = $servoy");
   # ----------------------------------------------------------------

   # send command

   ($status,$aref) = $COMM->sendMessage('P' . $pos1);
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   # receive reply

   ($status,$aref) = $COMM->receiveReply();
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   # display reply

   write_text_message_array($aref);
   return;
}

# ------------------------------------------------------------------
# move the servo/sensor to a specfied position
# ------------------------------------------------------------------

sub move
{
   my $pos1;
```

```
   my $status;
   my $aref;

   clear_text_display();
   write_text_message('Move Servo/Sensor to Position');

   # verify parameters

   ($status,$pos1) = verify_integer($P1ENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Pos 1 must be an integer (no sign)");
      return;
   }

   # send move command

   ($status,$aref) = $COMM->sendMessage('M' . $pos1);
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   # receive reply

   ($status,$aref) = $COMM->receiveReply();
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   write_text_message_array($aref);
   return;
}

# ---------------------------------------------------------------------
# display current position and distance in the Arduino
# ---------------------------------------------------------------------

sub current
{
   my $status;
   my $aref;

   clear_text_display();
   write_text_message('Current Arduino Position and Distance');

   # send current command

   ($status,$aref) = $COMM->sendMessage('C');
   if(!$status)
   {
      write_text_message_array($aref);
      return;
   }

   # recieve reply

   ($status,$aref) = $COMM->receiveReply();
   if(!$status)
   {
```

```
      write_text_message_array($aref);
      return;
   }

   # display reply

   write_text_message_array($aref);
   return;
}

# ---------------------------------------------------------------------
# initialize the grid count array
# ---------------------------------------------------------------------

sub init_grid_count_array
{
   for(my $c = 0; $c < GCOLS; $c++)
   {
      for(my $r = 0; $r < GROWS; $r++)
      {
         $GRIDCOUNT[$c][$r] = 0;
      }
   }
   return;
}

# ---------------------------------------------------------------------
# initialize the grid flag array
# ---------------------------------------------------------------------

sub init_grid_flag_array
{
   for(my $c = 0; $c < GCOLS; $c++)
   {
      for(my $r = 0; $r < GROWS; $r++)
      {
        $GRIDFLAG[$c][$r] = 0;
      }
   }
   return;
}

# ---------------------------------------------------------------------
# increment grid count array using the value (0/1) grid flag array
# ---------------------------------------------------------------------

sub increment_grid_count_array
{
   for(my $c = 0; $c < GCOLS; $c++)
   {
      for(my $r = 0; $r < GROWS; $r++)
      {
         if ($GRIDFLAG[$c][$r]) { $GRIDCOUNT[$c][$r]++; }
      }
   }
   return;
}

# ---------------------------------------------------------------------
# update grid count display
# copy the grid counts to grid display
# ---------------------------------------------------------------------
```

```
sub update_grid_count_display
{
   for(my $c = 0; $c < GCOLS; $c++)
   {
      for(my $r = 0; $r < GROWS; $r++)
      {
         $GRIDWIDGET[$c][$r]->m_delete('0.0','end');
         $GRIDWIDGET[$c][$r]->m_insert('end',$GRIDCOUNT[$c][$r]);
      }
   }
   return;
}

# ---------------------------------------------------------------------
# init the grid display
# ---------------------------------------------------------------------

sub init_grid_display
{
   init_grid_flag_array();
   init_grid_count_array();
   update_grid_count_display();
   return;
}

# ---------------------------------------------------------------------
# write an array of messages to the message area
# (array passed by reference)
# ---------------------------------------------------------------------

sub write_text_message_array
{
   my $mref = $_[0];           # message array reference

   foreach my $m (@$mref)
   {
      write_text_message($m);
   }

   return;
}

# ---------------------------------------------------------------------
# write a message to the message area
# ---------------------------------------------------------------------

sub write_text_message
{
   my $m = $_[0];              # message

   if ($FIRSTTEXTLINE)
   {
      $TEXT->m_insert('end',$m);
      $FIRSTTEXTLINE = 0;
   }
   else
   {
      $TEXT->m_insert('end', "\n" . $m);
   }
   $TEXT->m_see('end');

   return;
}
```

```
# ---------------------------------------------------------------------
# clear the message area
# ---------------------------------------------------------------------

sub clear_text_display
{
   $TEXT->m_delete('0.0','end');
   $FIRSTTEXTLINE = 1;
   return;
}

# ---------------------------------------------------------------------
# verify integer string
# return validation status, integer string with leading and
# trailing whitespace
# ---------------------------------------------------------------------

sub verify_integer
{
   my $i = $_[0];

   $i =~ s/^\s+//;             # remove whitespace
   $i =~ s/\s+$//;             # remove whitespace

   if ($i !~ /^\d+$/)
   {
      return (0,$i);           # error
   }

   return (1,$i);              # OK
}

# ---------------------------------------------------------------------
# verify integer string with/without sign
# ---------------------------------------------------------------------

sub verify_signed_integer
{
   my $i = $_[0];

   $i =~ s/^\s+//;             # remove whitespace
   $i =~ s/\s+$//;             # remove whitespace

   if ($i !~ /^[-+]?\d+$/)
   {
      return (0,$i);           # error
   }

   return (1,$i);              # OK
}


# ---------------------------------------------------------------------
# toggle verbose flag
# ---------------------------------------------------------------------

sub toggle_verbose
{
   if ($VERBOSE)
   {

     $VERBOSEBUTTON->configure(-text => 'Verbose On');
```

```
      $VERBOSE = 0;
   }
   else
   {

      $VERBOSEBUTTON->configure(-text => 'Verbose Off');
      $VERBOSE = 1;
   }
}

# ---------------------------------------------------------------------
# display help information
# ---------------------------------------------------------------------

sub help
{
   clear_text_display();
   write_text_message(
      "The 'Ping Sweep' command uses Pos 1 and Pos 2 as the start\n" .
      "and stop positions for ping sweeps. Pos 1 must be less that\n" .
      "Pos 2 and they should not exceed the range of the servo.\n" .
      "\nThe 'Move' command uses Pos 1.\n" .
      "\nThe 'Ping' uses the current servo position.\n" .
      "\nThe 'Sensor Adj Angle' is the adjustment used when the servo's\n" .
      "zero direction does not align with the grid's X axis.\n" .
      "\n'Servo X' and 'Servo Y' are the servo/sensor coordinates.\n" .
      "\nPort is the Aruino communication port.\n" .
      "\nBaudrate is the communication port\'s baudrate.\n" .
      "\nAll of the numeric parameters are integers. " .
      "Parameters are\nonly validated when a command is " .
      "executed.\n" .
      "\nSee the documentation for more information on what the\n" .
      "parameters are and how they are used.");
   return;
}

# ---------------------------------------------------------------------
# test - ping sweep
# ---------------------------------------------------------------------

sub test_ping_sweep
{
   $DEBUG = 1;

   clear_text_display();

   process_ping_sweep_data(\@TESTPINGSWEEP,
                           $SAAENTRY->get(),
                           $SXENTRY->get(),
                           $SYENTRY->get());

   $DEBUG = 0;

   update_grid_count_display();

   return;
}

# ---------------------------------------------------------------------
# test - display grid flags
# ---------------------------------------------------------------------

sub test_display_grid_flags
```

```
{
   my $str;

   clear_text_display();

   write_text_message('Current grid flags');

   for(my $r = (GROWS - 1); $r >= 0; $r--)
   {
      $str = '';

      for(my $c = 0; $c < GCOLS; $c++)
      {
         $str .= ' ' . $GRIDFLAG[$c][$r];
      }

      write_text_message($str);
   }
   return;
}

# ---------------------------------------------------------------------
# test - generate and display random grid count data
# ---------------------------------------------------------------------

sub test_grid_count_display
{
   clear_text_display();

   write_text_message('Generating random grid cell counts');

   for(my $c = 0; $c < GCOLS; $c++)
   {
      for(my $r = 0; $r < GROWS; $r++)
      {
         $GRIDCOUNT[$c][$r] = int(180.0 * rand());
         update_grid_count_display();
      }
   }
   return;
}

# ---------------------------------------------------------------------
# test - parameters
# ---------------------------------------------------------------------
sub test_parameters
{
   clear_text_display();

   my $pos1;
   my $pos2;
   my $saa;
   my $servox;
   my $servoy;
   my $baudrate;
   my $status;

   # check Pos 1 and Pos 2

   ($status,$pos1) = verify_integer($P1ENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Pos 1 must be an integer (no sign)");
```

```
   }

   ($status,$pos2) = verify_integer($P2ENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Pos 2 must be an integer (no sign)");
   }

   if ($pos2 < $pos1)
   {
      write_text_message("Error: Pos 2 < Pos 1");
   }

   # check sensor adjustment angle

   ($status,$saa) = verify_integer($SAAENTRY->get());
   if (!$status)
   {
      write_text_message("Error: sensor adjustment angle must " .
                         "be an integer (no sign)");
   }

   # check Servo X

   ($status,$servox) = verify_signed_integer($SXENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Servo X must be an integer (no sign)");
   }

   # check Servo y

   ($status,$servoy) = verify_signed_integer($SYENTRY->get());
   if (!$status)
   {
      write_text_message("Error: Servo Y must be an integer (no sign)");
   }

   # check baudrate

   ($status,$baudrate) = verify_integer($BAUDRATE->get());
   if (!$status)
   {
      write_text_message("Error: baudrate must be an integer (no sign)");
   }

   write_text_message("Pos1     = $pos1");
   write_text_message("Pos2     = $pos2");
   write_text_message("SAA      = $saa");
   write_text_message("ServoX   = $servox");
   write_text_message("ServoY   = $servoy");
   write_text_message("Baudrate = $baudrate");

   return;
}
```

Appendix 3 – Communications.pm

```
package communications;
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
# Object methods:
# new           create a new object
# toString      return a string version of the object data
# open          open a connection to a port
# close         close a connection to a port
#               object is not destroyed and may be re-used
# sentMessage   send a message (string)
# receiveReply  receive reply nessages (strings)
# - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

use strict;
use warnings;

require Win32::SerialPort;

# ----------------------------------------------------------------------
# constructor
# ----------------------------------------------------------------------

sub new
{
   my ($class) = @_;
   my ($self) = { _port     => undef,
                  _portname => undef,
                  _baudrate => undef };
   bless($self,$class);
   return $self;
}

# ----------------------------------------------------------------------
# convert object to string
# ----------------------------------------------------------------------

sub toString
{
   my ($self) = @_;

   my $str = "";

   if (!defined($self->{_port}))
   {
      $str="Port not defined";
      return $str;
   }

   $str = "Port is open\n" .
          "Port name " . $self->{_portname} . "\n" .
          "Baudrate " . $self->{_baudrate};

   return $str;
}

# ----------------------------------------------------------------------
# open port connection
# ----------------------------------------------------------------------

sub open
```

```
{
   my ($self,$portname,$baudrate) = @_;

   # is the port already open?

   my @reply = ();

   if (defined($self->{_port}))
   {
      $reply[0] = "Connection already open";
      return (0,\@reply);
   }

   # open port connection

   my $port;
   my $quite = 1;

   if (!($port = Win32::SerialPort->new($portname,$quite)))
   {
      $reply[0] = "Can not open port ($portname)";
      $reply[1] = $^E;
      return (0,\@reply);
   }

   # save the port information

   $self->{_port}     = $port;
   $self->{_portname} = $portname;
   $self->{_baudrate} = $baudrate;


   # -- for debugging ------------------------------------------------
   # prints hardware messages like "Framing Error"
   # $self->{_port}->error_msg(1);
   # prints function messages like "Waiting for CTS"
   # $self->{_port}->user_msg(1);
   # -----------------------------------------------------------------

   $self->{_port}->initialize();
   $self->{_port}->databits(8);
   $self->{_port}->baudrate($self->{_baudrate});
   $self->{_port}->parity("none");
   $self->{_port}->stopbits(1);
   $self->{_port}->debug(0);
   $self->{_port}->are_match("\r\n");

   $reply[0] = "Connected to port ($portname)";
   return (1,\@reply);
}

# ------------------------------------------------------------------
# close port connection
# ------------------------------------------------------------------

sub close
{
  my ($self) = @_;
  if (!defined($self->{_port})) { return 1; }
  $self->{_port}->close();
  $self->{_port}     = undef;
  $self->{_portname} = undef;
  $self->{_baudrate} = undef;
```

```
  my @reply = ("Connection closed");
  return (1,\@reply);
}

# ------------------------------------------------------------------
# send message
# ------------------------------------------------------------------

sub sendMessage
{
   my ($self,$msg) = @_;

   if (! defined($self->{_port}))
   {
      my @reply = ("Connection not open");
      return (0,\@reply);
   }

   my $l = length($msg);       # length of message

   $self->{_port}->lookclear;  # empty buffer

   my $c = $self->{_port}->write($msg); # write message,
                                        # get characters sent count

   if ($c == 0)
   {
      my @reply = ("write failed - no characters sent");
      return (0,\@reply);
   }
   elsif ($c != $l)
   {
      my @reply = ("Write incomplete (sent=$c) (msglength=$l)");
      return (0,\@reply);
   }

   my @reply = ("Write complete (sent=$c)");
   return (1,\@reply);
}

# ------------------------------------------------------------------
# receive reply messages
# ------------------------------------------------------------------

sub receiveReply
{
   my ($self) = @_;

   my @reply = ();             # array to collect reply messages

   if (! defined($self->{_port}))
   {
      $reply[0] = "Connection not open";
      return (0,\@reply);
   }

   $self->{_port}->lookclear;  # empty buffer

   my $msg;                    # reply message

   while(1)
   {
      # poll looking for data
```

```
      $msg = $self->{_port}->lookfor();

      # if we get data, save it

      if ($msg) { push(@reply,$msg); }

      # last reply messages?

      if ($msg =~ /^x/) { return (1,\@reply); }
   }

   unshift(@reply,'Received reply terminated early. ' .
                  'Did not receive "x" message.');

   return (0,\@reply);
}

1;
```