Using IR Beacons and IR Seeker

FTC Technical Document

Thomas Eng

9/12/2012

“Seeing” the Light…

Background

The 2012-2013 FIRST Tech Challenge **Ring It Up!** provides infrared (IR) beacons that can be used as navigational aids for the autonomous operation of a robot. The light that is emitted by these beacons is not visible to the human eye. A special device, such as the HiTechnic IR Seeker (v2), can be used to detect the IR energy and navigate towards the beacon. This document outlines the basic use of the IR beacon and Seeker sensor.

Components

IR Beacon

The IR Beacon is made by HiTechnic (part number FTCBCN). It is powered by a 9V battery and is equipped with two status LEDs and three IR LEDs. The red LED illuminates when the battery is running low. The green LED indicates that power is on. The three IR LEDs flash on and off at a frequency of 1200 Hz. The light from the IR LEDs is not visible to the human eye, but can be detected by devices such as the HiTechnic IR Seeker sensor, or even a common digital camera that does not have an IR filter installed.

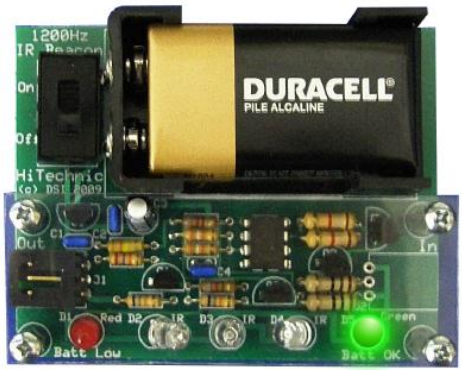

Figure 1 - Hitechnic IR Beacon (Part # FTCBCN)

The IR beacon has a switch to power the unit on or off. A fresh 9V battery should be sufficient to power the beacon continuously for several hours (approximately 4 according to the instruction sheet). Competition organizers, however, should keep a reserve supply of fresh 9V alkaline batteries on hand during a match.

Note that unlike the light emitted by a traditional incandescent light bulb, the light from the IR LEDs is not omnidirectional. The strongest intensity will be measured when the sensor is directly facing an LED. The two outer IR LEDs on the beacon have been intentionally bent inwards at the factory to provide a stronger signal when the detector is facing the beacon at an indirect angle.

IR Seeker Sensor

HiTechnic makes a sensor known as the IR Seeker (v2, part number NSK1042) that can be used to detect and navigate towards an infrared source of energy. The sensor detects un-modulated (DC) infrared energy, such as the light emitted from the sun or an IR heat lamp. The sensor also detects modulated (AC) IR signals, such as the light that is pulsed from the IR beacon. In AC mode, the sensor will "tune-in" to IR signals pulsing at either a 1200 Hz or 600 Hz frequency and ignore other IR signals. For the FTC competitions, the IR beacons pulse at 1200 Hz, so the IR seekers should be set to the 1200 Hz AC mode.

Figure 2 - HiTechnic IR Seeker v2

The IR Seeker is equipped with five internal infrared detectors. Each detector monitors a specific sector around the Seeker and provides an output value of 0 to 255, with 0 indicating no measured signal and 255 indicating the highest measurable value.

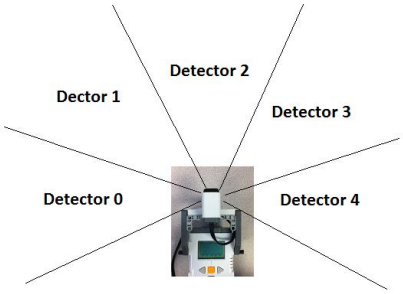

Figure 3 – The IR Seeker has 5 Internal Detectors with 5 Sectors of Coverage

The IR Seeker also provides directional information. It returns a value from 0 to 9. A value of 0 implies that an IR source was not detected. A value of 1 to 9 corresponds to 1 of 9 sectors around the Seeker (see Figure 4).

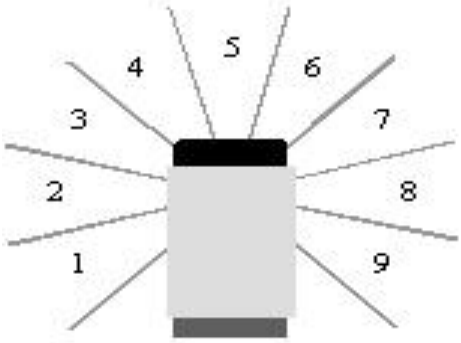

Figure 4 - Seeker Provides Directional Information on the Location of the IR Signal

Note that the energy that is emitted by the IR beacon is not omnidirectional. In order to get the strongest measured signal, it is important that the IR Seeker have the most direct line-of-sight possible. If the IR beacon is placed at a certain height above the floor, then a robot designer should consider placing their IR sensor at the same level, to ensure a good, direct line-of-sight to the IR LEDs. Also, in order to measure the strongest possible signal, the IR sensor should not have any objects in the way that obstruct the sensor's line of sight to the LEDs.

Sample LabVIEW Programs

Displaying Output of IR Seeker Sensor

LabVIEW for NXT 2010 and later versions include the VI's needed for using the IR Seeker v2 sensor. The VI's for the IR Seeker can be found in a HiTechnic sub menu for NXT I/O. Specifically, the VI's to use the IR Seeker can be found under the NXT Robotics -> NXT I/O -> Additional Sensors and Motors -> HiTechnic Sensors -> HT Complete.

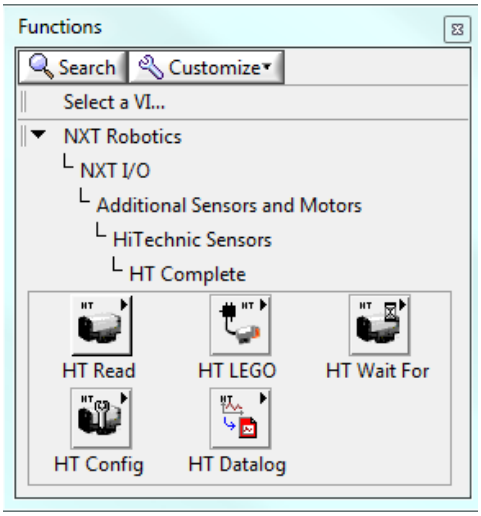

Figure 5 - HT Complete Menu

The HiTechnic menu includes controls to configure the IR Seeker (and put it in 600 Hz or 1200 Hz modes). It also includes controls to read data from the IR Seeker (AC or DC modes).

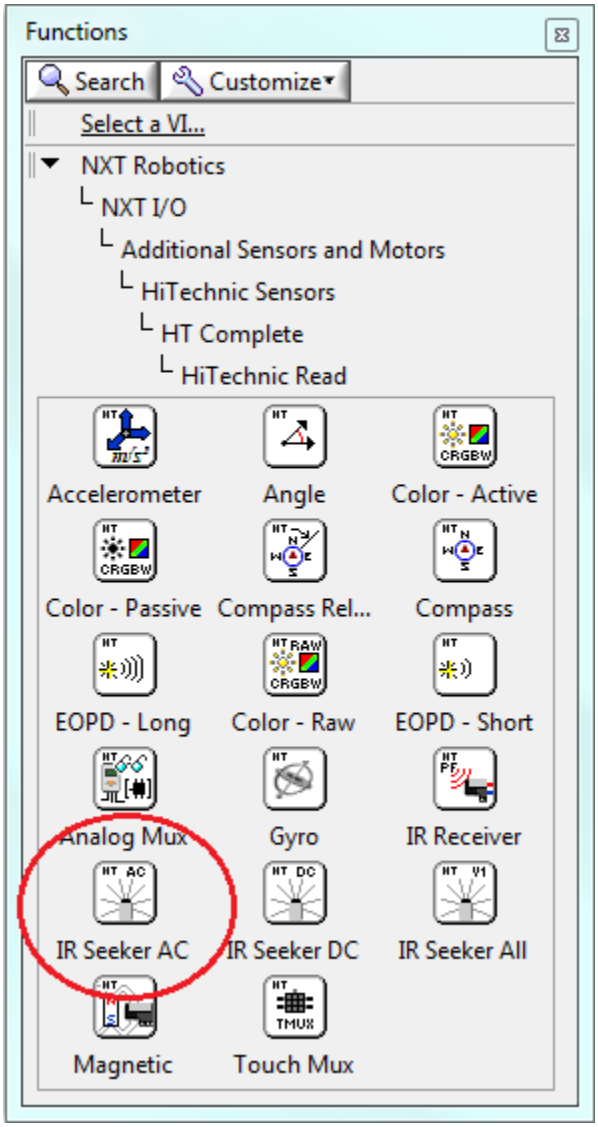

Figure 6 - HT Control to Read AC Data from IR Seeker

The block diagram depicted in Figure 7 is a program that continuously displays the output from the IR Seeker onto an NXT's LCD display. The first line of the LCD display ranges in value from 0 to 9 and corresponds to the sector where the IR signal is the strongest (see Figure 4) with zero indicating no source detected. The remaining lines of the LCD display show the signal strength for each of the five internal IR detectors (numbered 0 through 4).

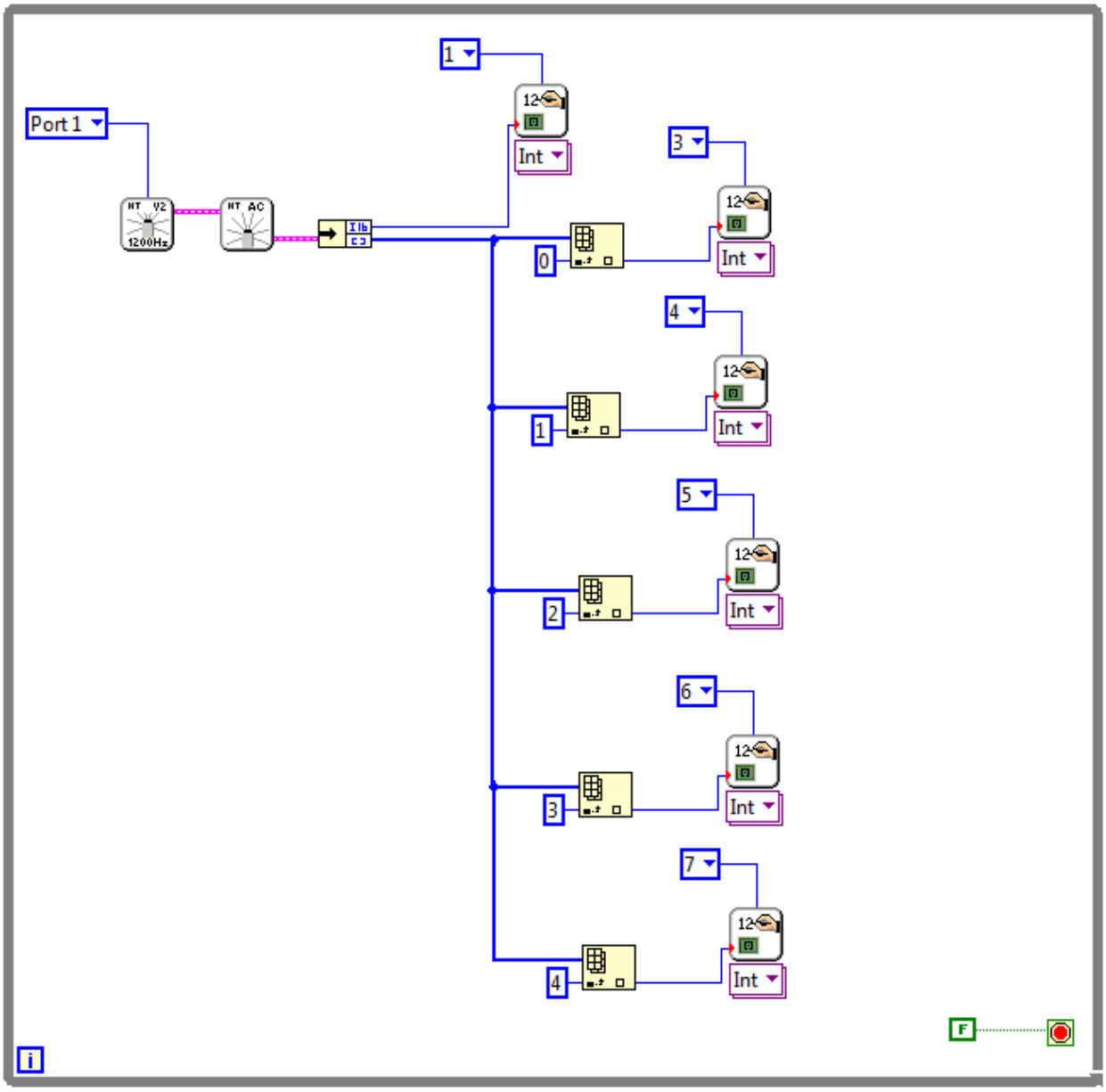

Figure 7 - Sample LabView Program that Displays Beacon Direction and Signal Strength Array

In this sample program the IR Seeker sensor is configured for sensor port 1 and set to 1200 Hz AC Mode (see Figure 8). The AC mode output from the IR sensor is a clustered data stream. It includes an integer value that indicates the direction of the IR source (0 means no signal, 1-9 corresponds to the sector map in Figure 4). The AC mode output also includes output data from the five IR detectors that are inside of the Seeker.

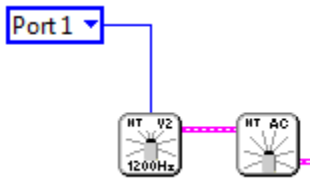

Figure 8 - IR Seeker is Configured for Port 1 and 1200 Hz Mode

In order to process the output from the IR Seeker, the sample program uses an Unbundle block to separate the data for processing (see Figure 9). The directional information is sent to a Display block which lists the directional value on line 1 of the NXT's LCD.

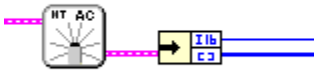

Figure 9 - An Unbundle Control is Used to Extract Data from Seeker Output

The data from the Signal Strength array is sent to a series of Index Array Controls to extract the values for the individual detectors (numbered 0 through 4). Each indexed value is then sent to a Display control to display the value on the NXT's LCD (see Figure 10).

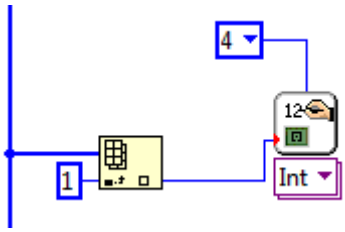

Figure 10 - Index Array Control Gets Value from Array and then Passes Data to Display Control

Drive to Ball Sample Program

LabVIEW for NXT 2010 also includes a sample program that shows how to use the IR Seeker sensor to make a robot follow an infrared soccer ball. To view the sample program, select "Create Program" from the main LabVIEW screen. A "New Program" window should appear. Select "Virtual Instruments" under the Templates section and then expand the HiTechnic folder and select "HTIRSeekerACDriveToBall.vit" to view the sample code (see Figure 11).

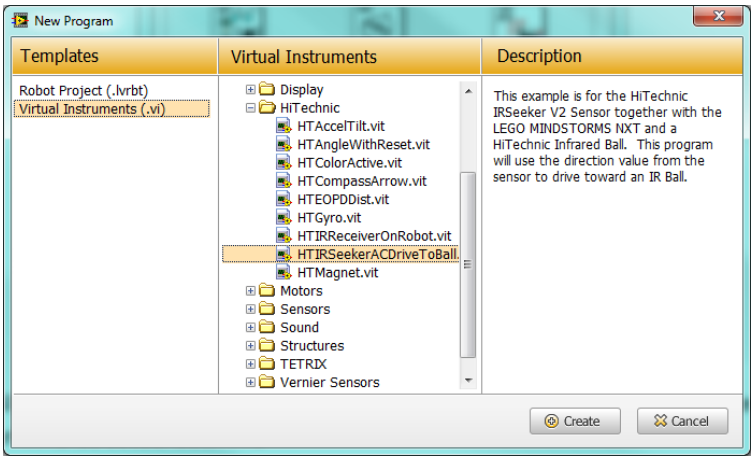

Figure 11 – Select HTIRSeekerACDriveToBall.vit

The sample program shows how to take the output from the HiTechnic IR Seeker sensor and use it to navigate towards an infrared source. If a source is not detected, the robot will point turn to the left, looking for a signal.

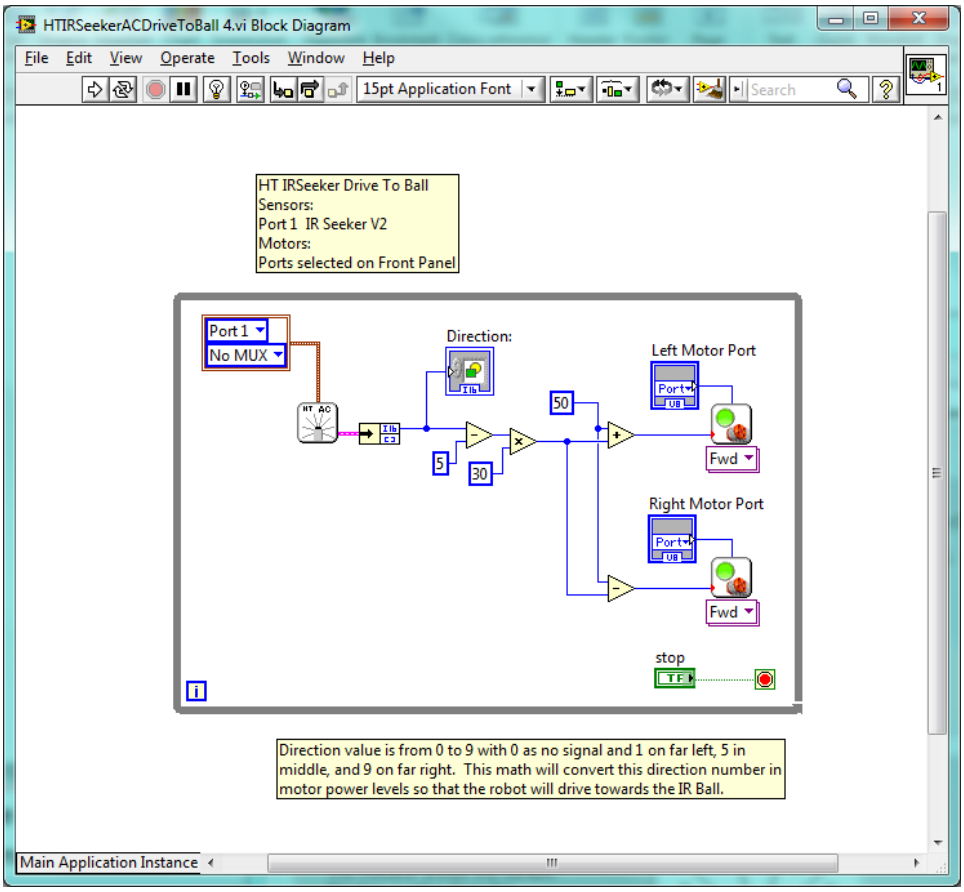

Figure 12 - Sample IR Seeker LabView Program

Sample ROBOTC Programs

HiTechnic ROBOTC Driver

ROBOTC includes a driver for the HiTechnic IR Seeker V2 sensor. The driver filename is "HTIRS2-driver.h" and it resides in a subdirectory called "drivers" in the NXT sample programs subdirectory of ROBOTC.

For a 64-bit version of Windows, the path to the 3rd party drivers should be as follows,

```
C:\Program Files (x86)\Robomatter Inc\ROBOTC Development
Environment\Sample Programs\NXT\3rd Party Sensor Drivers\drivers
```

For a 32-bit version of Windows, the path is similar, but without the (x86) designation,

```
C:\Program Files\Robomatter Inc\ROBOTC Development
Environment\Sample Programs\NXT\3rd Party Sensor Drivers\drivers
```

In ROBOTC, you can use the Detailed Preferences menu (see Figure 13) to specify the path to include the HiTechnic drivers.

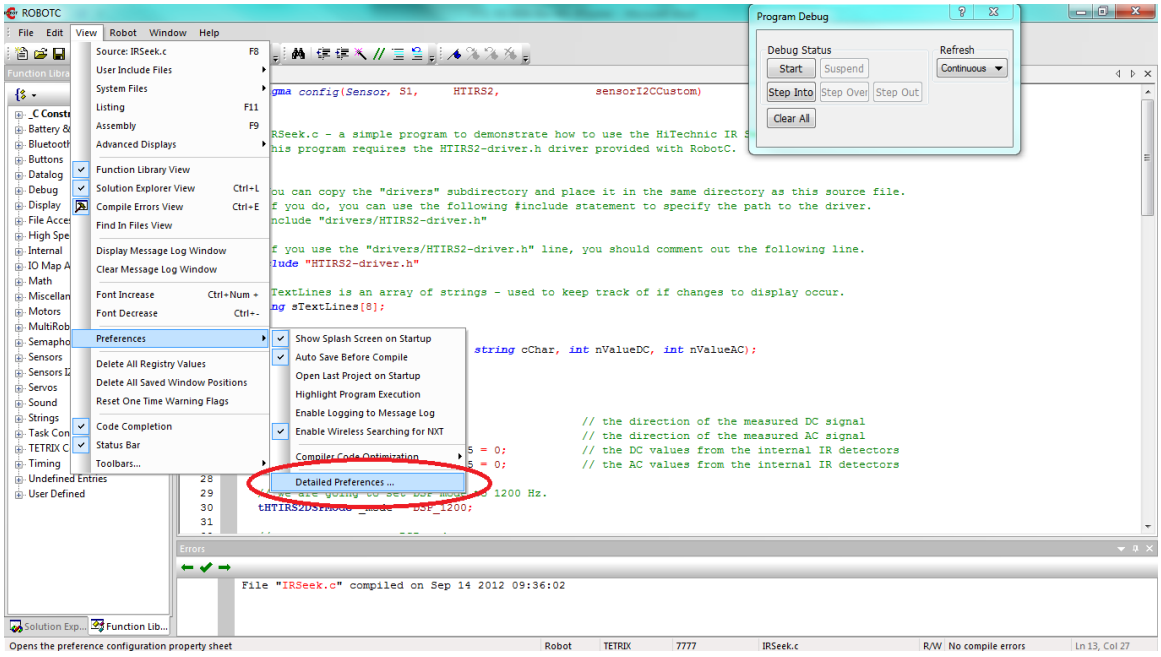

Figure 13 - Select Detailed Preferences Menu

Switch to the “Directories” tab of the Detailed Preferences menu to specify the include file path for the 3rd party drivers (see Figure 14).

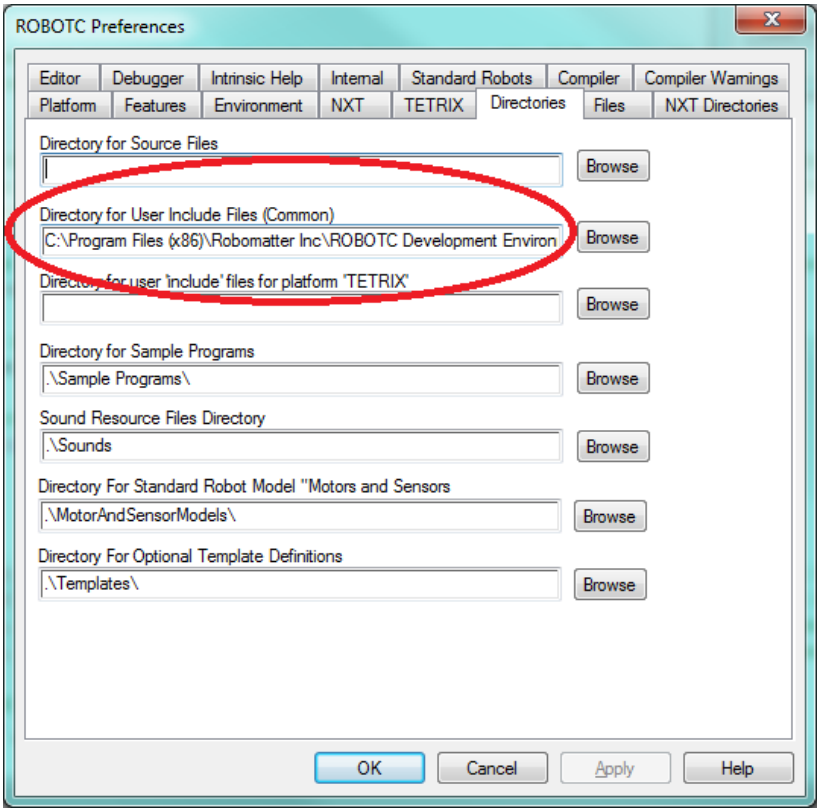

Figure 14 – Specify the Common Directory for User Include Files

Once you have your directory properly specified, be sure to put the following statement,

```
#include "HTIRS2-driver.h"
```

in your source code.

Display IR Seeker Data

The following code (from "IRSeek.c") is a sample program that polls the IR Seeker sensor and displays the direction and signal strength from the 5 internal detectors on the screen. The "DC" values are the output values for the un-modulated signal. The "AC" values are the output values for the modulated signal (1200 Hz).

```
#pragma config(Sensor, S1,     HTIRS2,              sensorI2CCustom)

//
// IRSeek.c - a simple program to demonstrate how to use the
// HiTechnic IR Seeker V2 sensor.  This program requires
// the HTIRS2-driver.h driver provided with RobotC.
//

// You can copy the "drivers" subdirectory and place it
// in the same directory as this source file. if you do,
// you can use the following #include statement to specify
// the path to the driver.
//#include "drivers/HTIRS2-driver.h"

// if you use the "drivers/HTIRS2-driver.h" line,
// then you should comment out the following line.
#include "HTIRS2-driver.h"

// sTextLines is an array of strings - used to keep
// track of if changes to display occur.
string sTextLines[8];

// function prototype
void displayText(int nLineNumber, string cChar, int nValueDC, int nValueAC);

// main task
task main ()
{
  // dc and ac directional values.
  int _dirDC = 0;
  int _dirAC = 0;

  // DC and AC values from 5 internal detectors.
       int dcS1, dcS2, dcS3, dcS4, dcS5 = 0;
       int acS1, acS2, acS3, acS4, acS5 = 0;

       // we are going to set DSP mode to 1200 Hz.
  tHTIRS2DSPMode _mode = DSP_1200;

  // attempt to set to DSP mode.
  if (HTIRS2setDSPMode(HTIRS2, _mode) == 0)
  {
    // unsuccessful at setting the mode.
    // display error message.
    eraseDisplay();
    nxtDisplayCenteredTextLine(0, "ERROR!");
    nxtDisplayCenteredTextLine(2, "Init failed!");
    nxtDisplayCenteredTextLine(3, "Connect sensor");
    nxtDisplayCenteredTextLine(4, "to Port 1.");

    // make a noise to get their attention.
    PlaySound(soundBeepBeep);

    // wait so user can read message, then leave main task.
```

```
    wait10Msec(300);
    return;
  }

  // initialize the array sTextLines.
  for (int i = 0; i < 8; ++i)
    sTextLines[i] = "";

  // display some header info at top of screen
  eraseDisplay();
  nxtDisplayTextLine(0, "       DC    AC");
  nxtDisplayTextLine(1, "-----------------");

  // loop continuously and read from the sensor.
  while(true)
  {

    // Read the current non modulated signal direction
    _dirDC = HTIRS2readDCDir(HTIRS2);
    if (_dirDC < 0)
      break; // I2C read error occurred

    // read the current modulated signal direction
    _dirAC = HTIRS2readACDir(HTIRS2);
    if (_dirAC < 0)
      break; // I2C read error occurred

    // Read the individual signal strengths of the internal sensors
    // Do this for both unmodulated (DC) and modulated signals (AC)
    if (!HTIRS2readAllDCStrength(HTIRS2, dcS1, dcS2, dcS3, dcS4, dcS5))
      break; // I2C read error occurred
    if (!HTIRS2readAllACStrength(HTIRS2, acS1, acS2, acS3, acS4, acS5 ))
      break; // I2C read error occurred

    displayText(2, "D", _dirDC, _dirAC);
    displayText(3, "0", dcS1, acS1);
    displayText(4, "1", dcS2, acS2);
    displayText(5, "2", dcS3, acS3);
    displayText(6, "3", dcS4, acS4);
    displayText(7, "4", dcS5, acS5);

    // wait a little before resuming.
    wait10Msec(5);
  }
}

// Minimize LCD screen flicker by only updating LCD when data has changed
void displayText(int nLineNumber, string cChar, int nValueDC, int nValueAC)
{
  string sTemp;
  StringFormat(sTemp, "%4d  %4d", nValueDC, nValueAC);

  // Check if the new line is the same as the previous one
  // Only update screen if it's different.
  if (sTemp != sTextLines[nLineNumber])
  {
    string sTemp2;
    sTextLines[nLineNumber] = sTemp;
    StringFormat(sTemp2, "%s:  %s", cChar, sTemp);
    nxtDisplayTextLine(nLineNumber, sTemp2);
  }
}
```

The code listed above will continuously display the output from the IR Seeker on the LCD.

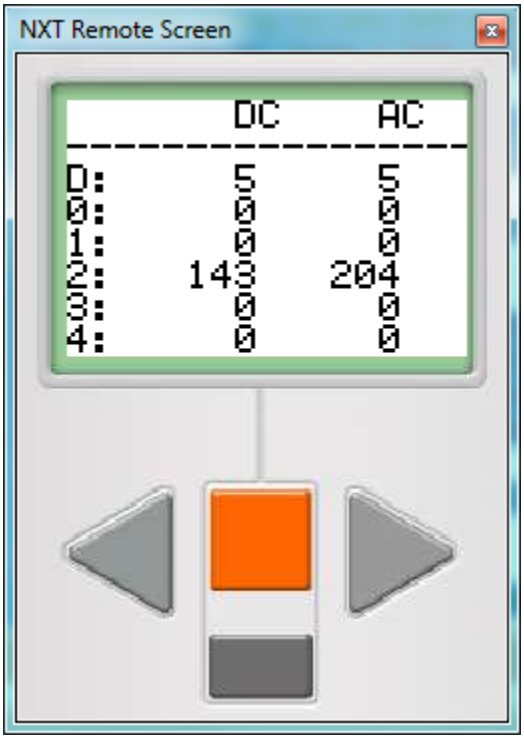

Figure 15 – The Program IRSeek.c Continuously Displays Output from Sensor

Following an IR Source

The following code (from "RoboSeek.c") uses the IR Seeker sensor to follow an infrared source (such as an IR soccer ball or beacon) flashing at 1200Hz.

```
#pragma config(Sensor, S1,     HTIRS2,              sensorI2CCustom)

//
// RoboSeek.c - This program uses the HiTechnic IR Seeker V2
// sensor to follow an IR source.  This program requires the
// HTIRS2-driver.h driver provided with RobotC.
//

// You can copy the "drivers" subdirectory and place it
// in the same directory as this source file. if you do,
// you can use the following #include statement to specify
// the path to the driver.
//#include "drivers/HTIRS2-driver.h"

// if you use the "drivers/HTIRS2-driver.h" line,
// then you should comment out the following line.
#include "HTIRS2-driver.h"

// main task
task main ()
{
  int _dirAC = 0;
       int acS1, acS2, acS3, acS4, acS5 = 0;

       int maxSig = 0;    // the max signal strength from the seeker.
       int val = 0;       // the translated directional value.

       // we are going to set DSP mode to 1200 Hz.
  tHTIRS2DSPMode _mode = DSP_1200;

  // attempt to set to DSP mode.
  if (HTIRS2setDSPMode(HTIRS2, _mode) == 0)
  {
    // unsuccessful at setting the mode.
    // display error message.
    eraseDisplay();
    nxtDisplayCenteredTextLine(0, "ERROR!");
    nxtDisplayCenteredTextLine(2, "Init failed!");
    nxtDisplayCenteredTextLine(3, "Connect sensor");
    nxtDisplayCenteredTextLine(4, "to Port 1.");

    // make a noise to get their attention.
    PlaySound(soundBeepBeep);

    // wait so user can read message, then leave main task.
    wait10Msec(300);
    return;
  }

  eraseDisplay();

  // loop continuously and read from the sensor.
  while(true)
  {
    // read the current modulated signal direction
    _dirAC = HTIRS2readACDir(HTIRS2);
    if (_dirAC < 0)
```

```
    {
      // error! - write to debug stream and then break.
      writeDebugStreamLine("Read dir ERROR!");
      break;
    }

    // Get the AC signal strength values.
    if (!HTIRS2readAllACStrength(HTIRS2, acS1, acS2, acS3, acS4, acS5 ))
    {
      // error! - write to debug stream and then break.
      writeDebugStreamLine("Read sig ERROR!");
      break;
    }  else {
      // find the max signal strength of all detectors.
      maxSig = (acS1 > acS2) ? acS1 : acS2;
      maxSig = (maxSig > acS3) ? maxSig : acS3;
      maxSig = (maxSig > acS4) ? maxSig : acS4;
      maxSig = (maxSig > acS5) ? maxSig : acS5;
    }

    // display info
    nxtDisplayCenteredBigTextLine(1, "Dir=%d", _dirAC);
    nxtDisplayCenteredBigTextLine(4, "Sig=%d", maxSig);

    // figure out which direction to go...
    // a value of zero means the signal is not found.
    // 1 corresponds to the far left (approx. 8 o'clock position).
    // 5 corresponds to straight ahead.
    // 9 corresponds to far right.

    // first translate directional index so 0 is straight ahead.
    val = _dirAC - 5;

    // calculate left and right motor speeds.
    motor[motorC] = 50 + 30 * val;
    motor[motorB] = 50 - 30 * val;

    // wait a little before resuming.
    wait10Msec(2);
  }
}
```

As the robot follows around the IR source, the directional output and the magnitude of the strongest measured signal are displayed on the screen.

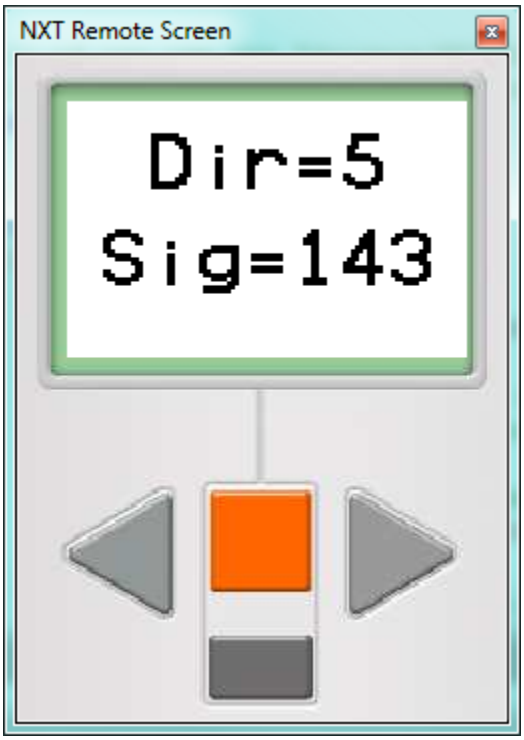

Figure 16 - RoboSeek.c Will Follow an AC IR Signal and Display the Direction and Max Intensity on the LCD

Figure 17 - Robot with IR Seeker and IR Soccer Ball