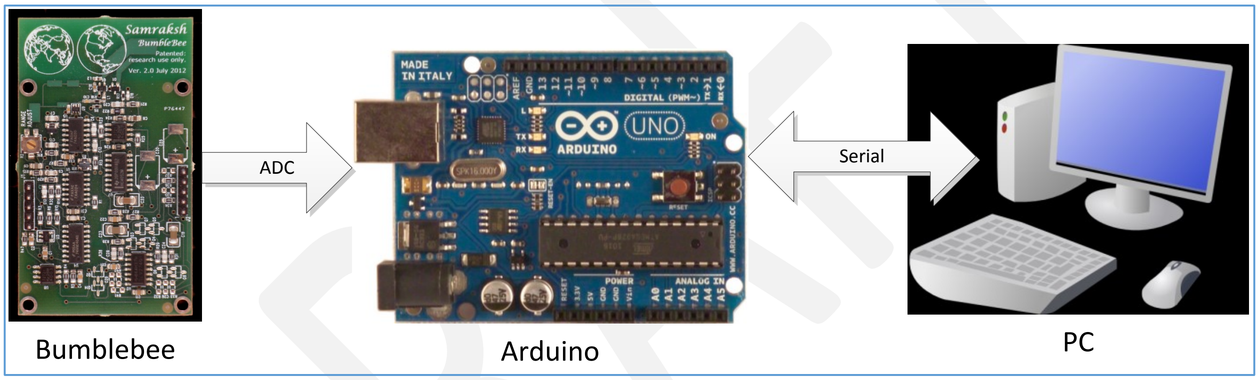
Arduino – Bumblebee Radar Displacement Detector

Version 1.0, 5/29/2015

The Bumblebee, made and sold by Samraksh, is a small, inexpensive, low-power phased pulsed Doppler radar that can be used to detect various kinds of physical motion, including displacement (movement in one direction) and periodic (movement back and forth). If it's used to detect displacement then it can detect the motion of an animal, person, vehicle or some other object without being confused by periodic motion such as a bush or tree in the wind.

In this write-up we'll describe a project that uses a combination Arduino UNO and Bumblebee as a displacement detector along with a PC that acts as a base station. Here's a block diagram.



The Bumblebee is powered by the Arduino and sends sensed data to it on two ADC lines. The Arduino runs a program that interprets the Bumblebee data, decides if displacement is happening, and does confirmation. The program sends displacement and confirmation decisions to the PC over the serial port, optionally with sample-level detail. The PC runs a program that receives the Arduino output, displays it on a log, changes the display and plays a sound when displacement is occurring.

For a tutorial on the Bumblebee radar, please see << link >>.

1 Setting Up the Displacement Detector

Parts List:

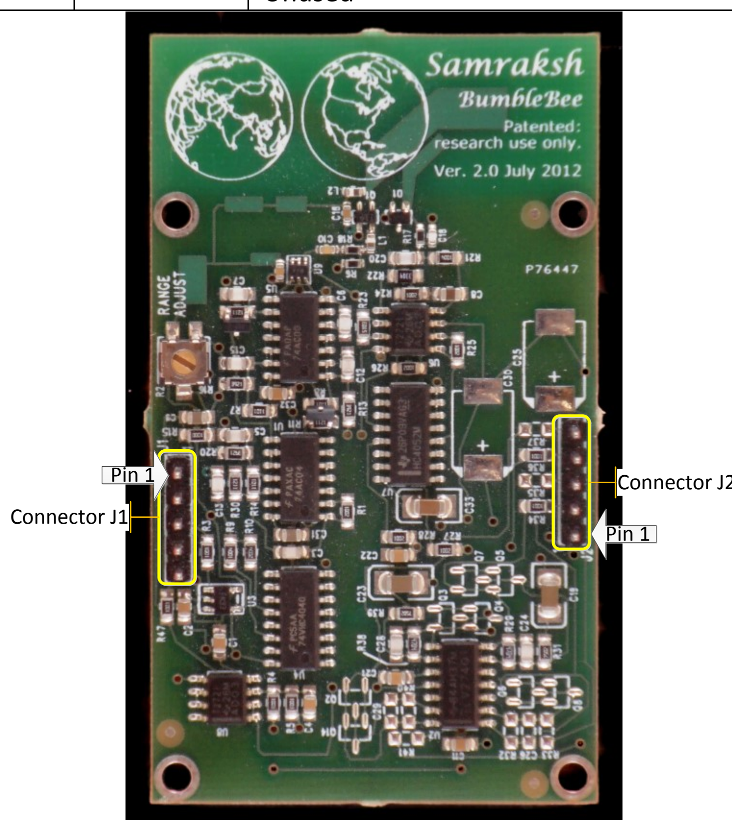
1. Arduino UNO. Other versions of Arduino might also serve.
2. Bumblebee radar.
3. Bumblebee stand.
4. Breadboard. Optional but useful.
5. (3) LEDs with resistors. Optional but useful. Preferably high-intensity for better visibility. For the size of the resistor, see http://www.instructables.com/id/Choosing-The-Resistor-To-Use-With-LEDs/.

6. SPST momentary contact switch. Optional but useful for SD output. Closes SD output buffer and idles the Arduino.
7. PC running Windows.
8. Miscellaneous supplies & tools such as jumpers, solder, soldering iron, shrink-wrap tubing, electrician's tape.

1.1 Bumblebee Pinouts

The Bumblebee pinouts are shown below. The two ground pins are internally connected so only one need be connected to the Arduino.

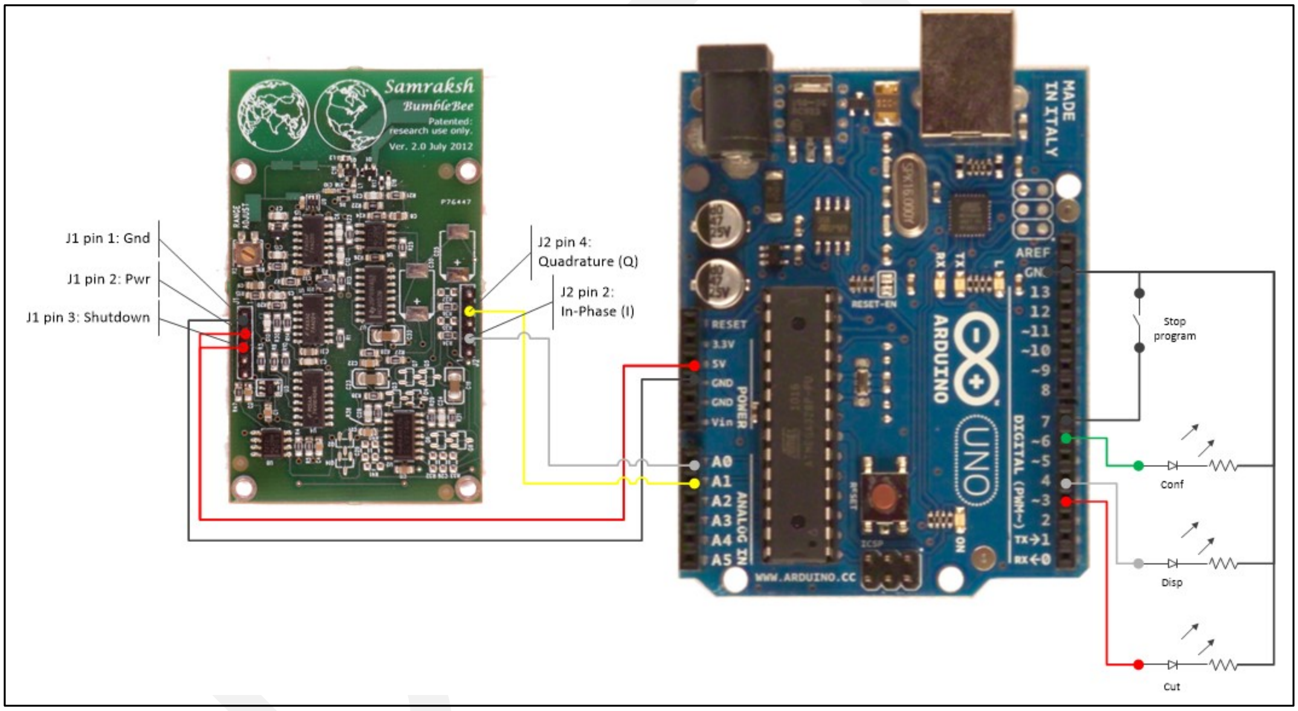
Pin	Type	Designation	Remarks
Connector J1			
1	In	Ground	
2	In	Power	[3.65v, 16v]
3	In	Shutdown	Assert high to enable, assert low to shutdown
Connector J2			
1	Out	Ground	
2	Out	In Phase	Analog [0,3.3v]
3			Unused
4	Out	Quadrature	Analog [0,3.3v]
5			Unused



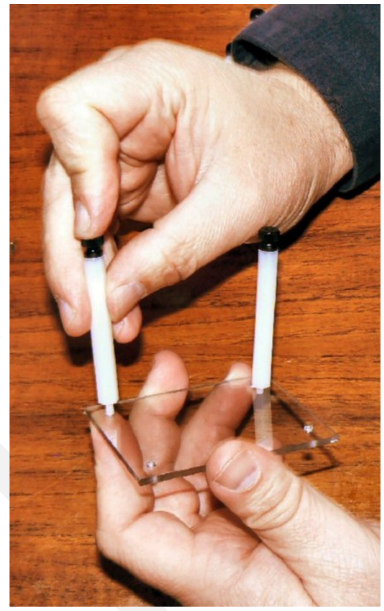
1.2 Wiring

The Bumblebee is connected to the Arduino as shown. In addition, you can wire the optional stop-program switch and the LEDs. The Bumblebee's output voltage range is [0, 3.3v] whereas the Arduino's GPIO is set for [0, 5v]. This means that the digitized values will be over a smaller range but this does not affect the displacement detection.

Bumblebee	Arduino
J1 pin 1 (Gnd)	Gnd
J1 pin 2 (Pwr)	+5v
J1 pin 3 (Shutdown)	+5v
J2 pin 2 (In-Phase)	A0
J2 pin 4 (Quadrature)	A1



1.3 Mounting the Bumblebee onto a Stand



A grounded object within one wavelength of an antenna will load the antenna in such a way as to dramatically diminish the effectiveness of the antenna. The Bumblebee’s center frequency is 5.8 GHz, which corresponds to a wavelength of about 5.2 cm. As a result it is ideal to position the radar so that its antenna is at least 5.2 cm away from any large metal objects, especially the batteries. To make this easier to do the Bumblebee comes with a plastic stand.

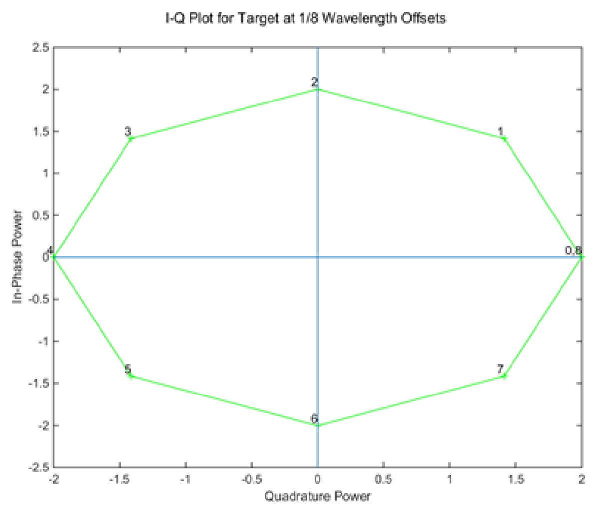
The stand is assembled by screwing the four plastic posts into the base as shown in the figure. The plastic thread can easily be striped with excess force. In addition avoiding cross threading requires a steady downward force and carful perpendicular alignment. Once all 4 posts are secured, remove the black thumb screw and washers on the top of each post. Place the board on top of the posts and refasten each of the thumb screws, making sure that the washers are on top of the board. The threaded posts allow you to assemble and disassemble the stand many times. However the plastic threads are striped more easily than metal threads. Once your setup is finalized you can improve the strength by gluing the posts into the base.

2 How Displacement Detection Works: An Overview

The Bumblebee produces two analog power values called In-Phase (I) and Quadrature (Q). The internal values are over a positive-negative range that can vary depending on variability in components in the Bumblebee. To reduce error (and to be compatible with ADCs that only accept non-negative voltages), the I and Q power values are each shifted so as to be non-negative. As the Arduino program samples the I and the Q power values via the ADC it calculates a running average for each and subtracts it from the respective power value sampled. Over time this gives sample power values that are accurately displaced in the positive-negative range.

The Bumblebee tutorial <<link>> gives detail on how it can be used to detect motion and direction. In Figure 1, taken from the tutorial, we have a target that is moving away from the Bumblebee. The sample power values are shown from sample 0 through sample 8. As the target moves away, the I-Q power values change as shown. For example, sample 1 is counter-clockwise from sample 0 and similarly sample 2 with respect to sample 1. At sample 8 the I-Q power values are the same as for sample 0. As the target keeps moving, rotation of sample power values continues.

Figure 1: Plot of I-Q Power Values



If the target is moving towards the Bumblebee, we'll see the same thing happen except the rotation on the graph will be clockwise. Because of the frequency the Bumblebee uses for its radio broadcast, each rotation represents about 2.6 cm of distance.

Our interest is to detect when something is moving in a steady fashion towards or away from the Bumblebee, ignoring things that are moving back and forth. We'll do this by "chunking" the movement into units of one rotation—2.6 cm—assigning a positive value if it's clockwise (motion towards the Bumblebee) and negative otherwise. We arbitrarily choose the right half of the Q (horizontal) axis as our "cut" point, when we declare that a rotation has taken place.

Next we sum up the cuts over the course of a time interval called a "snippet"; in the program we've chosen a snippet size of 1 second. If the sum of the cuts is at least some *minimum cumulative cuts* value we declare that displacement has occurred. We've chosen 6 as the minimum; you can choose your own. Since negative and positive cuts cancel each other out, there have to be at least 6 net cuts in the same direction in one snippet. Since a cut is 2.6 cm, 6 cuts is a displacement of 15.6 cm.

As you might have noticed, a target's motion could begin just before a cut boundary and, upon the 6th cut, end just after the boundary, making the displacement a bit more than 4 * 2.6 cm = 10.4 cm instead. We've found this isn't usually a problem, but if you care, you're free to modify the program to keep track of the distance between each successive pair of samples rather than cuts. If you do, be aware of the fact that this will take more time, might not be possible to get it done in the time between two samples, and will take more power, reducing lifetime if using battery power.

Suppose we have a bush being blown by the wind. If it's gusty wind, the bush will be blown back and forth so the positive and negative cuts will cancel each other out and displacement will not be detected. As you reflect on this, you'll see that there are ways in which a false detection could occur. For example, a large bush might be blown more than the 15.6 cm and held there steadily for a while, causing displacement detection; later the wind might slacken and another displacement in the other direction might occur. Sensors aren't perfect and neither are detection algorithms, so to add to our confidence we include an *M-of-N confirmation*: In the last window of N snippets, has displacement occurred at in least M of them? If we choose M = 2 and N = 8, then two displacements can occur at a 4 second interval, say, or twice in a row, and M-of-N confirmation will be satisfied. As with minimum cumulative cuts, M and N can be adjusted to your taste.

The displacement detection and confirmation algorithms themselves be adjusted. For example, instead of dividing time into fixed snippets, you could try a sliding window, so that a snippet would start only when you detect a cut. The M-of-N confirmation is agnostic to whether the M cuts are positive or negative or a mix, so displacement forward and backwards would each qualify to help satisfy the confirmation requirement. You could change it so that all displacements would have to be in the same direction. You can be as creative as you like on your detector algorithm and/or confirmation algorithms; you can bias it towards minimizing false detections by making the minimum cumulative cuts larger at the expense of missing some actual detection; and conversely making it smaller to minimize misses at the expense of false detections. Just bear in mind that each choice comes with a trade-off and you'll need to decide what's important to you.

3 Arduino Displacement Detector Program

The Arduino program was developed using Visual Micro, an add-in for Visual Studio that facilitates Arduino sketch development and debugging. However, the program does not depend on it and you're free to use the development tools of your choice.

3.1 Bumblebee_Displacement_Detector.ino

This is the main sketch. It handles the overall orchestration of displacement detection. Broadly speaking, the process is as follows.

- The setup function does the usual initialization. It also initializes a semaphore and starts a timer at 250 Hz.
- The timer callback function (interrupt service routine) reads alternately from ADC0 (Q power) and ADC1 (I power). To form a sample, it applies the running average to the channel sampled and interpolates the value for the other channel (described in more detail below). When a sample is ready it sets the semaphore.
- The loop function waits on the semaphore. When it is set, it resets it and processes the sample, checking for displacement and for M-of-N confirmation. The results are optionally sent via serial to the PC.

A number of GPIO lines are used to give alerts and provide information for debugging with a logic analyzer or oscilloscope.

3.1.1 Interpolation

Sample	I	Q
1	4	
2		7
3	5	
4		4

Since we are alternating between sampling the I and Q channels, we calculate the value for the unsampled channel as the average of the last and the next values. In the example shown, we can't interpolate for Q in the first sample because there is no previous value. For sample 2, we can interpolate for I as (4 + 5) / 2 = 4.5. Hence for sample 2 the I-Q pair formed is (4.5, 7). Similarly, for sample 3 the pair is (5, 5.5). To interpolate we have to read ahead one sample in order to have a next value available.

3.1.2 Serial Interaction

The program can optionally send sample detail or snippet-level information to an attached PC. It can also receive commands from the PC. One command has been implemented. When the Arduino program receives "*1", it responds by sending parameter values to the PC. You can, of course, implement other commands.

3.1.3 Using an SD Card

Code is present to log to an SD card using the FAT file system. However, write is blocking so the program cannot proceed when data is being written to the card. This delay is sufficient to cause samples to be lost. You may want to experiment with this to see if you can overcome the limitation.

3.2 Detector.ino

The functions in this file handle cut analysis, displacement detection and MofN confirmation.

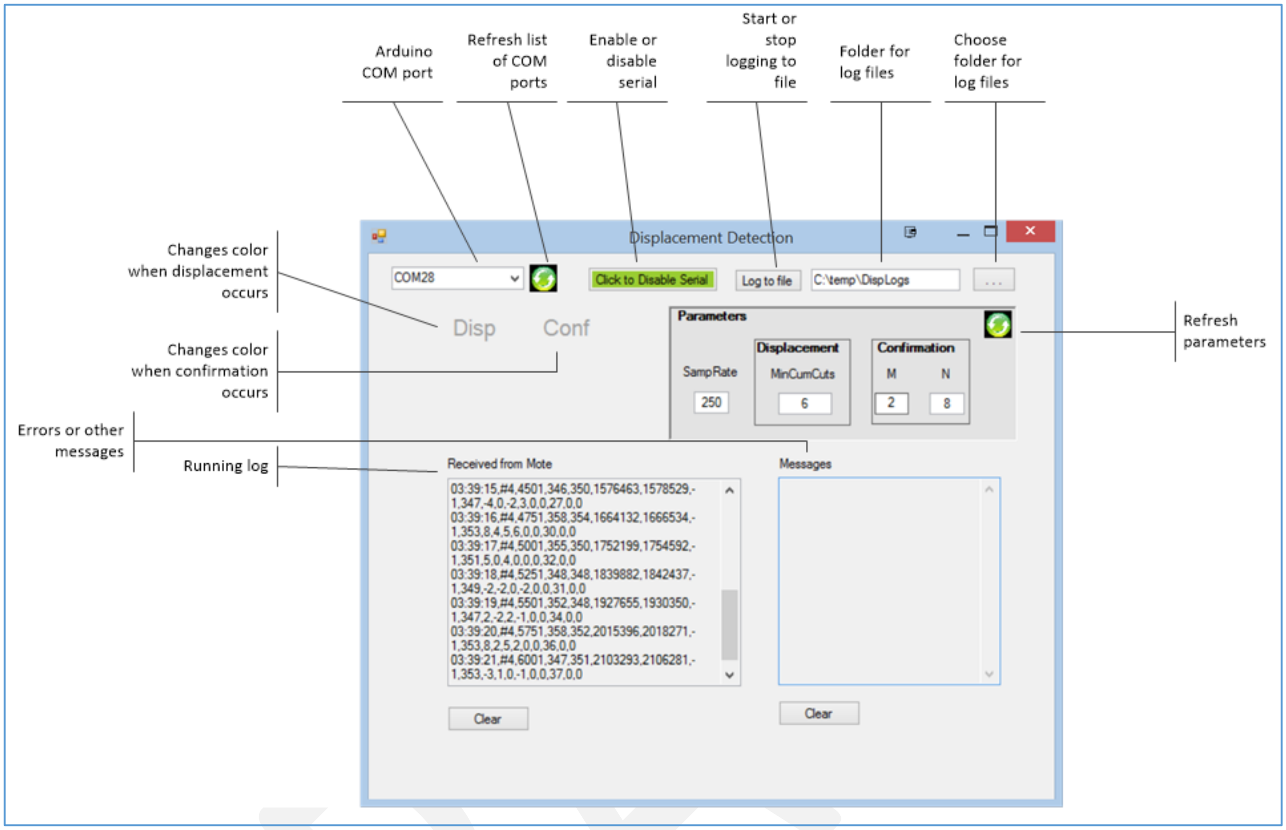
3.3 Power Management

The program does not do any power management on the Arduino: it is running at full power all the time. However, there are tools you can use to make it sleep between events. For example, http://playground.arduino.cc/Learning/ArduinoSleepCode.

4 PC GUI Program

The PC GUI program for the displacement detector is a Windows Forms C# program. You can of course modify the program to suit your needs. If you have a Windows PC but don't have Visual Studio

(commonly used for C# programming), you can get a free version from
https://www.visualstudio.com/en-us/products/free-developer-offers-vs.aspx.



The program is used to connect via serial to an Arduino running the displacement detection program. To get started, select the COM port that the Arduino is attached to; refresh the COM list if necessary. Click "Enable Serial" to start. If you want to log, choose a folder and click "Log to file". You can populate the Parameters panel by clicking the refresh button; the first time you might have to click it twice.

When a displacement occurs, the Disp control changes color and a sound plays; and similarly for confirmation.

5 Validation

The Arduino program was validated by comparing with a MatLab program. The PC program was used to collect raw sample data. The MatLab program was used to check sample interpolation, and results for cut, displacement and confirmation. The algorithms used by the MatLab program were high-level, making use of the availability of history and future data for interpolation, trig functions for cut analysis, and a sliding window for MofN confirmation.