Arduino – Bumblebee Radar Displacement Detector

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. The diagram below shows a block diagram.

<< 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 and decides if displacement is happening. The program sends displacement decisions to the PC over the serial port. 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 >>.

# 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. Optional but useful. Preferably high-intensity for better visibility.
6. (3) resistors, \*\*\* ohm. For interfacing the LEDs.
7. PC running Windows.
8. Miscellaneous supplies & tools such as jumpers, solder, soldering iron, shrink-wrap tubing, electrician’s tape.

## Wiring

<< diagram >>

## 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 we’ve included 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.

# How Displacement Detection Works: An Overview

The Bumblebee produces two analog power values called In-Phase (I) and Quadrature (Q); see Section 5 for detail. 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 : 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 2.6 cm of distance; this is also covered in the tutorial.

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 I (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’ve chosen 6; you can choose your own), we declare that displacement has occurred. Since negative and positive cuts cancel each other out, there have to be at least 6 net cuts in the same direction. If 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 about 4 \* 2.6 cm = 10.4 cm instead. This we’ve found 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. If you do, be aware of the fact that these computations will take more time and it might not be possible to get it done in the time between two samples.

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 one displacement can occur each 4 seconds, say, 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 we’ve just described can also 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.

# Arduino Displacement Detector Program

The Bumblebee Displacement Detector sketch has 5 parts.

## 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 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.

### 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.

### 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 to do such things as send parameter values (sample rate, snippet size, M-of-N values).

### Using an SD Card

Code is present to log to an SD card using a 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.

## Detector.ino

This sketch contains methods to check for a cut, detect displacement and do M-of-N confirmation. If you consult the Bumblebee tutorial << link >> you’ll see that the trig function atan2 is used calculate the angles of consecutive points in order to tell whether motion has taken place, how much and in what direction. However, all we need to do is determine whether a cut has taken place and if so, in what direction.

A cut takes place if the angle between the vectors of any successive pair of I-Q power pairs crosses the negative Q axis. If we know the direction of the angle then we know a cut has taken place if the direction is counter-clockwise, the Q value of the previous pair is positive and the Q value of the current pair is negative. There is a converse rule for clockwise direction, going from negative Q to positive. In Fig << fig>>, if P is a pair and C2 is the successor, a cut has taken place because the rotation is counter-clockwise and the Q values have changed from positive (4) to negative (-3).

To determine rotation, we use a bit of trig. Don’t let this dismay you. You can skip this discussion if you want, or you can dig into the references provided. Refer to << fig >>. You see a Q-I axis with four vectors. P represents an I-Q power value pair, (4, 2). Suppose we have a successor C, such as C0 (‑1, 3). We want to know whether C is clockwise or counter-clockwise from P. To do this, we make use of the following fact:

Before explaining this formula, note that on the right we have the sine of Θ which is the angle from P to C. The sine of an angle is zero at 00 and 1800; is positive between 00 and 1800; and is negative between 1800 and 3600. (3600 is of course equivalent to 00 so the sine is 0). See <https://www.mathsisfun.com/algebra/trig-four-quadrants.html>. This is what we’re interested in. If the sine of Θ­ is negative then the direction is counter-clockwise; if positive it is clockwise. We ignore the cases when it’s zero. If it’s at 00 then no motion has taken place. If it’s at 1800 then we don’t know whether it’s clockwise or counter-clockwise. This case is rare and ignoring it is better than guessing.

Now as to the formula itself. On the right hand side we have the unsigned length of the vector of P (from the origin (0, 0) to P) times the unsigned length of the vector of C times the sine of Θ­­. Since neither length is negative, the sign of the left hand side will be the same as the sign of the sine of Θ­­.

To calculate the left hand side we multiply the two vectors for P and C and get the signed length. The formula for this is:

Here, P.Q is the Q value of the P pair and similarly for I and for C. For more information about this, see

* <https://www.youtube.com/watch?v=Tesvs6xCWZA>
* <https://www.khanacademy.org/math/linear-algebra/vectors_and_spaces/dot_cross_products/v/proof-relationship-between-cross-product-and-sin-of-angle>

Hence two multiplications and a subtraction give us the sine of Θ and therefore the direction of rotation. Let’s see how this works out for the values in the figure.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **I** | **Q** | **LHS** | **Direction** |
| **P** | 2 | 4 |  |  |
| **C0** | 3 | -1 | 14 | counterclockwise |
| **C1** | -1 | 3 | -10 | clockwise |
| **C2** | -1 | -3 | 2 | counterclockwise |

So to summarize.

* If LHS is positive and P.Q is negative and C.Q is positive then we have a cut in the clockwise direction.
* If LHS is negative and P.Q is positive and C.Q is negative then we have a cut in the counter-clockwise direction.

# PC Host Program

\*\*\* tbd

# Data Collection

\*\*\* tbd