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 one pair and C2 is the next one, 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 it further. See for instance << need reference >>. <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>

Refer to << fig >>. You see a Q-I axis with four vectors. P represents a previous I-Q power value pair, (4, 2). We consider what happens with different successor power values. Suppose the next one is C0 (-1,3).

# PC Host Program

\*\*\* tbd

# How the Bumblebee Works

We’ll begin with how a low-power phased pulsed Doppler radar works. The Doppler Effect, named after the 19th century Austrian physicist Christian Doppler, is illustrated by what we hear when, for instance, a train sounding a horn passes by us: as it approaches the sound waves are compressed, while after it passes they are expanded. See <http://science.howstuffworks.com/radar.htm> for more information. If we knew the frequency of the horn we could measure the frequency of what we’re hearing and compare. This would tell us whether the train is approaching or leaving and at what speed and we would have a displacement detector.

A radio signal is an electromagnetic wave that is similar to sound for our purposes; it just travels faster, at the speed of light. As a wave, we can observe the Doppler Effect as a target moves closer or further away from the receiver. A Doppler radar works on the same principal as our train example except that instead of just receiving a signal, it also sources it by sending a radio wave as a reference and listening to the response as it is reflected by a target.

Radars in general can be used for measuring range and speed. They can do this by measuring how long it took for the reference signal to return (distance) and the difference in frequency between reference and response (speed and direction). The problem for a low-power application like ours is that the speed of light is so large that measuring the reference and return times and frequencies accurately requires a lot more computing power than a battery can sustain for very long.

To spell this out some, light takes about 1 nanosecond to travel 1 foot <http://en.wikipedia.org/wiki/Nanosecond>. To measure the round-trip time to a foot of resolution would require a clock running at 2 GHz. That’s because the signal has to make a round-trip, so a difference of ½ foot distance means 1 foot round-trip difference. So to get distance to the nearest foot \*\*\* huh \*\*\*

This puts us into the PC range, with corresponding power requirements. By contrast, the Arduino UNO R3 has a 16 MHz clock, less than 2% that speed. Since Hence if we if we tried to use the Arduino clock to measure round-trip time we’d be at 100 feet of resolution. That is, we’d only know the target’s location to within 100 feet.

Doppler radars can be continuous-wave or pulsed. Continuous wave radars, as the term implies, have the transmitter on all the time. This is popular for some applications because it’s inexpensive. However, having the transmitter on all the time means that the receiver has to be isolated to avoid interference. A typical application is a police vehicle speed radar where the transmitter signal is narrowly focused so it does not broadcast directly to the receiver. In addition, having the transmitter and receiver on all the time takes energy, with negative impact for low-power applications.

\*\*\* a police radar measures the speed of a target; other than directionality issues, why couldn’t we do the same? \*\*\*

To avoid dealing with the interference between transmitter and receiver incurred by continuous wave (and, as a bonus, to save power), we use a pulsed Doppler approach. We’re not going to try to measure frequency change or round-trip time since that takes too much power. What we can do instead with low power is to determine where a target is relative to a wavelength. We can leverage the fact that when we get a return signal, we can mix it with the reference signal and tell how much out of phase the return is.

The Bumblebee radar transmits at a frequency of 5.8 GHz; this amounts to a wavelength (peak-to-peak distance) of about 5.2 cm. Now suppose the target is stationary at, say, 5.2 meters from the radar: exactly 100 times 5.2 cm. Assuming an ideal environment, the wave will travel 5.2 meters out and 5.2 meters back, for a total distance of 10.4 meters. Since 10.4 meters is exactly 200 \* 5.2 cm, the return wave will, at the radar, look just like the reference. See the figure below, where the blue is the reference and the red is the return:

<< also need a figure showing a wave leaving and returning >>



Suppose the target moves 1.8 cm closer to the radar. Now the total round-trip distance is (200 \* 5.2) – (2 \* 1.8) = 199.5 \* 5.2 cm. So the wave coming back will be offset by ½ wavelength or 180o out of phase as shown below.



What we’re interested in is the phase difference between the signal and the return. So suppose we take the reference signal and mix it with the return. Since waveforms add, we wind up with the following composites depending on where the target is in the 2.6 cm interval.

        

We can see that as the target moves closer to the radar, the combined waveforms will change until the target reaches 2.6 cm closer. Then it starts all over again. We can’t tell how far away the target is but we can tell where in a wavelength it is. (\*\*\* how exactly? Values after 180 start looking like those before, in reverse \*\*\*)

So how do we do this in practice? Broadly, the process is as follows:

1. The transmitter is turned on and transmits a 5.8 GHz wave for 30 ns (nanoseconds).
2. The transmitter is turned off for 16 ns.
3. The transmitter *and* receiver are turned on for 30 ns.
4. The receiver is turned off.

Note in step 3 that the transmitter is on. That’s to provide the receiver with the reference signal. The combination gives us composite waveforms that can help us know how far the target is. \*\*\* what exactly is I? Q? \*\*\*

There’s an issue we have addressed yet. Although the amplitude varies according to the distance of the target, it also varies according to the mass of the target. \*\*\* things getting fuzzy here … \*\*\* To handle this, we also “interfere” with the return by using a second signal that’s 90o out of phase with the reference. This gives us two values. I (for In-Phase) is the first “interference”: the return combined with the reference. Q (for quadrature) is the second “interference”: the return combined with the 90o out of phase reference. The phase difference of the return with respect to the reference is given by:

* I = amplitude of return \* cos (phase diff).
* Q = amplitude of return \* sin(phase diff).

Now we have two unknowns (amplitude and phase difference) and two equations, so we can solve for the phase difference.

(junk from here on)

We assume that the target’s mass and location don’t change much between the paired In-Phase and Quadrature signals. This gives us two equations in two unknowns (mass and interval position), which we can solve for position. (You can also solve it for mass, which is a way to detect the size of a target. Here, we’re only concerned with position.) \*\*\* need a lot more detail \*\*\* Now consider the situation of a target that is moving in one direction. We can take a sample from the radar and plot the Quadrature on the X-axis and the In-Phase on the Y-axis. \*\*\* continue example … show how axis transitions occur \*\*\*

We want the benefits of Doppler for displacement detection in a low-power context. There are two principal obstacles: standard approaches to Doppler detection require too much computing capability and continuous operation, even for a device like the Bumblebee, will drain batteries too quickly. We resolve these by pulsing the transmission so that most of the time it is not transmitting (\*\* How much \*\*) and by measuring the Doppler effect indirectly.

A Doppler radar sends signals at some frequency, which bounces off some target and returns. It’s possible to estimate range if the radar can measure how long it took the signal to return. That requires substantial computing power and is not suitable for low-power radars. It’s possible to estimate motion if the radar can measure the difference between the reference (outbound) signal frequency and the return frequency. When the target moves, the return frequency will be higher or lower than the reference if the target is moving towards or away from the radar, respectively. Comparing the frequencies again requires intensive computing and is not suitable for a low power radar. Finally, radars with a continuous power source will generally be transmitting all the time. As this would drain batteries quickly,

the Bumblebee instead transmits in pulses so that most of the time it is not transmitting. (\*\*\* Need details on pulse length, interval \*\*\*)

The approach the Bumblebee uses is to measure the power level of the return signal. The Bumblebee runs at a frequency of 5.8 GHz, which means the wavelength is about 5.2 cm. Since we’re working with both reference and return signals, we divide the value by 2. So 2.6 cm is effectively one wavelength. \*\*\* huh\* \*\*\* Think of a line to the target and then, beginning at the Bumblebee, mark off segments of 2.6 cm. For a given target, the power level of the return signal will vary according to the target position in a given interval: at the boundaries of an interval it will be minimum while in the middle it will be maximum. Hence as a target moves toward or away from the radar, the power level will oscillate as the target moves from interval one interval to another.

<< pic to illustrate >>

However, the power level is a function not only of position within the interval but also of the mass of the target and the time when a signal was sent (\*\*\*): larger targets will result in higher return power. To deal with this we send a second signal that is 90o out of phase with the first. The intervals for the second signal (which we call the Quadrature) are shifted relative to the first (In-Phase) by ½ interval. The power for the Quadrature is the same as it would be for In-Phase if the target were ½ interval closer or farther away.

Now we assume that the target has fixed mass and that it doesn’t move much between the paired In-Phase and Quadrature signals. This gives us two equations in two unknowns (mass and interval position), which we can solve for position. (You can also solve it for mass, which is a way to sense the size of a target. Here, we’re only concerned with position.) \*\*\* need a lot more detail \*\*\*

Bear in mind that we don’t know how far away from the radar the target is. All we know is where it’s located relative to some interval.

F: 5.8 GHz

Wavelength: 5.2 cm

Cut: 2.6 cm