EENG18020 Ultrasonic Lab
Week 5 - Distance Measure

1 Introduction

In this lab, you will finish the last part of your system - detect the ultrasonic signal with the micro-controller's ADC and calculate distance. You will be using the **range_detect** project.

2 Rising edge detection

Now that you have generated a pulsed ultrasonic signal and sampled it with an ADC in the amplitude and time domains, you can start to measure distance. To do this, first we need to be able to detect the rising edge of the received signal.

The signal we receive will have three components:

1. A DC offset that has been added by our amplifier.
2. Gaussian white noise introduced at various stages in the system (e.g thermal noise in the amplifier).
3. The 40kHz desired sine wave, delayed by the round trip time of the signal.

Due to the DC offset, which will vary in magnitude by configuration, we cannot use a simple threshold to detect our signal. Similarly, an adaptive threshold (that places itself relative to the DC offset) is similarly inappropriate as the spurious noise will trigger it. For this reason, we will use a digital filter to detect the signal. As a sine wave will have a high auto-correlation with itself when in-phase, yet a low auto-correlation with anything else, we can create a filter which follows the 40kHz sine wave and auto-correlate it with the received signal. Should the output of the auto-correlation be high, we know there is a 40kHz sine wave present in the signal.

3 Implementation

We have given you a function named **autocorrelate()**. This function takes in a sample buffer and position as parameters. It then calculates the auto-correlation between a 40kHz sinusoid and your signal for 8 samples starting from the position. If your signal (stored in the sample buffer) contains a 40kHz signal starting before position i and ending after position i+7, then the returned value will have a large magnitude. The returned value is a struct (Complex32) that contains the real and imaginary parts of the correlation.

Your first task is to construct a function, **is_it_here()**, which takes in a buffer and a position, then applies **autocorrelate()** to the buffer at that position. If the magnitude of the return value is greater than a global variable named **SENSITIVITY**, then you should return TRUE, else return FALSE. The magnitude can be calculated as $(num.real * num.real) + (num.imag * num.imag)$ where **num** is the return value from **autocorrelate()**. The function prototype is in **adcSingleChannel.c** at line 145 as follows:

```
Bool is_it_here(volatile uint32_t *buffer, int i);
```

Task 1: Complete the is_it_here() function on line 145 of adcSingleChannel.c, and use the test logic at line 69-90 to test it. You should see one line of "This should always happen" come out for each PWM pulse transmitted by the application

Now that we are able to tell whether the signal is in a particular position, we need to search through the buffer for it. To do this, you will need to loop a variable i from 0 to **ADCBUFFERSIZE-FILTER_COEFFS**, and for each i call **is_it_here()** with i as the position and **microVoltBuffer** as the sample buffer. You will then need to break from the loop (using the "break;" statement) so that the post-loop logic can pick up the value of i and display the distance.

Task 2: Delete the test code and implement a loop on line 66 of adcSingleChannel.c to search for the response. Run your application to measure the distance to an object (large and flat is best). Is the measured distance accurate? Plot the measured versus real distance on a graph. Determine the limit at which the system stops detecting the received signal. Show a lab demonstrator.

4 Extension: Calibration

You may have seen in the previous section that the measured and actual distances do not exactly match. This will be due to a number of factors:

- The horizontal/vertical distance between the detectors (this distortion is most prominent at low distances).
- The time delay in the electronic components (this distortion produces a constant offset).
- The inaccurate timestamping of the buffer return time or PWM start (this will cause a random error).
- Spurious noise causing higher or lower responses at certain times (this will cause a random error).

While some of these (random errors) cannot be removed, the systematic errors (offsets, distortion) can be removed by calibration. This involves measuring some pairs of perceived and actual values, converting these measurements into a model of the systematic error, and then inverting this model to convert from measured values to real values. The simplest model is a linear model, which consists of a scaling factor and an offset ($y = mx + c$), but other options are available.

Extension 1: Decide on a model and calibrate your ultrasonic system according to this model. Show a lab demonstrator.

5 Extension: Phase detection

Up until now, the highest resolution we have been able to achieve in the time domain is 1 sample = 6.25μs. This is equivalent to 0.9375mm. If we want even higher resolution than this, we can use phase detection. By detecting the phase of the returned signal, we can determine the relative delay by comparing the phase, although this becomes ambiguous past one full cycle (25us = 3.75mm). As we can remove this ambiguity with the pulsed operation of the ultrasonics, we can then use the phase to refine the measurement.

Extension 2: Obtain the phase of the received signal using the results from the autocorrelate() function and combine this with your coarse distance measurement to increase the resolution of your rangefinder. Measure the accuracy and precision. Show a lab demonstrator..

6 Extension: Amplitude Detection

In addition to pulsed time-of-flight measurements, amplitude-based measurements can be performed. In this mode of operation, the received amplitude of the ultrasonic signal is tested to determine the distance. As the distance between the transmitter and receiver of a wave increases, the amplitude decreases inversely proportional to the square of the distance. This is known as the inverse square law. This can be exploited to give us a distance measurement based on the returned signal's amplitude.

Extension 3: Measure the amplitude of the received signal (using the magnitude output by the autocorrelate() function) and calibrate it according to the inverse square law. Test your amplitude measurement alongside your time measurement. How accurate is it? Try rotating the object the ultrasonic waves are reflecting off. How does this affect your measurements? Show a lab demonstrator.

7 Extension: Velocity Detection

Sometimes, we may not only be interested in the distance to an object, but also what speed it is travelling at relative to us. This is particularly important in automotive applications, where the relative speed of an object is as important as its distance. As an object moves towards or away from the source of the signal, it will compress or decompress waves in the spatial/time domain, thereby causing doppler shift. Upon reception, it would be seen that the received frequency would be slightly different to the transmitted frequency, with the frequency shift ΔF related as in equation 1. f_0 is the frequency of the transmitted wave, c is the speed of the wave in the medium (speed of sound in air), and Δv is the speed of the moving object.

$$\Delta F = \frac{\Delta v f_0}{c} \tag{1}$$

$$\frac{\Delta v}{c} << 1 \tag{2}$$

To determine the frequency of the received signal, an algorithm called a Fast Fourier Transform (FFT) can be used. This algorithm converts a signal into its frequency components using autocorrelation with sine waves. Once converted, the peak magnitude in the FFT spectrum can be found to determine the frequency and hence the doppler shift.

Extension 4: Implement code that detects the doppler frequency of a moving object and calculates the speed. Test it by moving an object back and forth in front of the ultrasonics. How accurately does it work? Show a lab demonstrator.

8 Extension: Multi-Object Detection

If we have multiple objects in the environment, we will get a reflection from each object, with different delays to match each object. The returned signals will add to each other, meaning the amplitude may either increase or decrease with each reflection. The change in amplitude can be detected by a differing magnitude return from the autocorrelate() function.

Extension 5: Detect multiple objects using your ultrasonic rangefinder. How accurate can you be? What is the minimum distinguishable distance? Show a lab demonstrator.

9 Extension: Angular Detection

If we have multiple transmitters or receivers, we can use the phase difference between them to determine the angle at which the reflection occurs. If the reflection is to the left, for example, the reflection would arrive at a receiver further to the left before one further to the right. This time / phase difference can then be utilised to calculate the angle through equation 3, where B, the baseline, is the distance between the receivers, c is the speed of sound in air, Δt is the time difference between the reception of the two signals at the two receivers (which may be calculated by relating it to the phase difference, $\Delta\Phi$, and θ is the incident angle of the signal. This equation is valid only when the object is sufficiently far away that the incoming wavefront appears to be flat.

$$\theta = sin^{-1}\left(\frac{c\Delta t}{B}\right) \tag{3}$$

$$\Delta t = \frac{\Delta\Phi}{2\pi f_0} \tag{4}$$

Extension 6: Use multiple ultrasonic receivers and the signal phase difference between these receivers to detect the angle of objects. Can you combine this with multi-object detection to give multiple objects at different angles? What is the minimum discernable angle? Show a lab demonstrator.

10 Extension: Depth Detection

As ultrasonic signals are partially reflected and partially transmitted from a surface, it is possible to penetrate through objects to detect other objects (a bit like seeing through a translucent material). This is often used in subterranean sonar for geological surveyancing. The implementation is the same as multi-object detection, so implement the multi-object detection first.

Extension 7: Place the ultrasonics against a solid object with different objects the other side. Can you detect the other objects? Compare the maximum range to the maximum range of the free-space ultrasonics; how does it differ? Show a lab demonstrator.