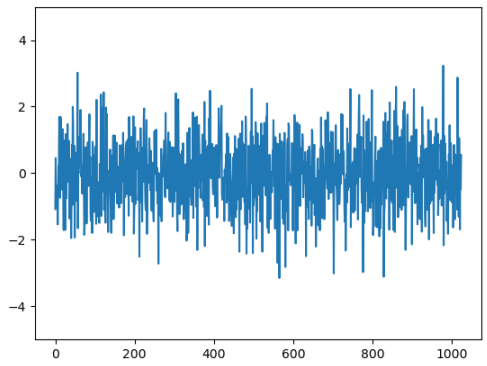
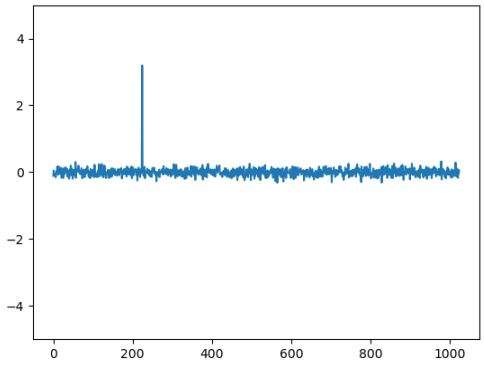
BWSI AUV Challenge 2021

Lab 12
Detection in Noise





Introduction

In the AUV Challenge obstacle course you will be required to detect buoys while traveling in an underwater environment. You will have knowledge of exactly what the buoys look like, so that should be easy, right? Not so fast - while the buoys may not change, the background on which you are expected to detect them varies constantly, as a function of time and look direction. In order to contend with these varying background conditions, we will seek to make our detection algorithms robust to noise that we can model.

The key feature we will use to help combat background noise is the assumption that the signal we are looking for (the buoy in this case) remains relatively constant while the background changes as a function of time and space. In the images at the top of the Lab you can see the same signal in a noisy background. On the left, the noise background is so high that it is not obvious the signal is there at all. On the right, the noise background is lower and the signal is obvious to any observer.

Basics of Noise

All measurements are contaminated by noise, which can often be modeled as an independent random signal added to the signal that you are trying to measure. A classic expression in detection theory is *y(t) = s(t) + w(t)*, where *y(t)* is the measurement, *s(t)* is the true signal that is being observed, and *w(t)* is the contaminating noise. The noise is typically represented as zero mean, with a specific probability distribution. The strength of the noise is characterized by its *variance* or its *standard deviation* (which is the square root of the variance). The most common strategy to reduce noise in a detection problem is to average over independent samples. Let's take a look at what effect that has on the noise itself.

1. Use the np.random.randn() function to create a noise vector that is 1024 samples long. The randn generator makes zero-mean noise that is distributed according to the bell curve, or Gaussian distribution. What is its variance (use np.var() to compute the variance)? _____
2. Now generate 2 random sequences of the same length and average them. What is the variance of the averaged random sequence? Continue to average 3, 4, 5 and fill in the following table. What do you notice?

Gaussian

Number of sequences	**1**	**2**	**3**	**4**	**5**
Variance					

3. Now try that again with a different random distribution. Use np.random.random() to create a 1024-element vector that draws from the uniform distribution. The uniform distribution draws random values between 0 and 1, so it does not have a zero mean. You should subtract out the mean before calculating the variance as you fill out the same table as we just filled out for the Gaussian distribution.

Uniform

Number of sequences	1	2	3	4	5
Variance					

4. Sometimes we don't have a series of measurements to average together. Another way to average as a function of time is to run a smoothing filter over a single time series. Since we will use OpenCV for our image analysis code, let's practice using their 1D smoothing function. If you've made a vector called *noise* that is (1024,1) in size, then a call to create an n-point smoothing function in OpenCV would look like the following: *noise_smoothed = cv2.boxFilter(noise, cv2.CV_32F, (1, n))*. Vary n from 1 (no filter) up to 5, and compute the variance of the resulting output. Do you notice anything?

Smoothing

Filter length	1	2	3	4	5
Variance					

Detection

We've now seen that noise variance goes down as it is averaged. Reduced noise is great, but what we really want is to improve our signal-to-noise ratio (SNR), which will help us detect buoys (or anything else we are looking for) with greater confidence and less false alarms. In this section we will see the effect of our smoothing effects on the signal. For the purpose of this lab we will define the signal-to-noise ratio as the square of the maximum signal level divided by the variance of the noise.

5. Create a measurement signal that is a 1024-element vector with all zeros, except a 1 in position 27 of the vector. Make copies of this vector and average them together, and take the peak value. Then assume this signal had the additive noise from step 2, and fill out the table below.

Number of sequences	1	2	3	4	5
Signal max					
Noise variance					
SNR					

6. Now let's try the smoothing option. Generate the same 1024-element signal vector with a single point equal to one (and the rest zeros). Apply the box filter to this signal for filter lengths > 1. Now the variance is given by the table you made in step 4 as the noise and signal have gone through the same processing step. Complete the table below. Do you notice anything about the SNR that is different from the averaging case in step 5?

Filter length	1	2	3	4	5
Signal max					
Noise variance					
SNR					

7. Finally, let's consider what would have happened if the signal was not a single point, but was instead a constant value for a period of time. Create a signal vector that is all zeros, and then add 10 consecutive points that have a value of 1. Repeat the table from step 6 and see what changes, if anything.

Filter length	1	2	3	4	5

Signal max					
Noise variance					
SNR					

Receiver Operating Characteristic

The Receiver Operating Characteristic (ROC) curve is a classic measure of the performance of a given detector (and associated sensor). The ROC is actually a family of curves that specify a system's *probability of detection* versus *probability of false alarm* at a given SNR. Let's see how having a higher SNR impacts our ability to detect objects (like buoys!)

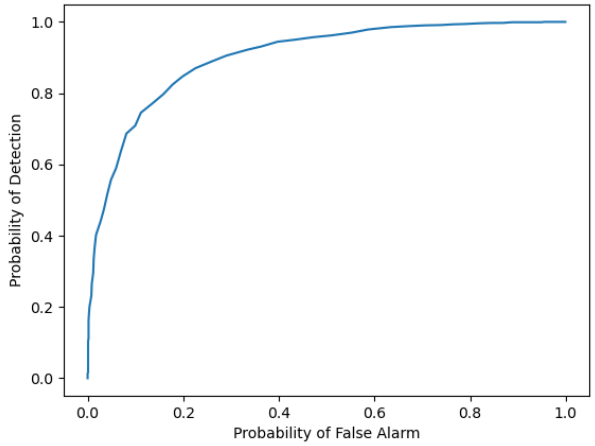
8. Consider the following code snippet that calculates and plots a ROC curve for a given SNR (in this case 3). In this function, a known signal of height sqrt(SNR) is created and added to 1024 random samples of noise. We then vary a threshold from a very low value - in which all the noise and the target will be above threshold (so probability of detection (PD) = 1 but probability of false alarm (PFA) also = 1!) - to a very high value, in which neither noise values nor signal values will be above the threshold (so detection=0 and false alarm=0). In between, the detector will make some detections and some false alarms. The more detections it can make with fewer false alarms, the better the detector.:

```
# ROC
SNR = 3
noise = np.random.randn(1024,1)
sigp = np.sqrt(SNR) + noise

thresh = np.linspace(np.min(noise), np.max(sigp), num=100)
pd = np.zeros((100,1))
pf = np.zeros((100,1))
for n in range(100):
    pd[n] = len(sigp[sigp>thresh[n]])/1024
    pf[n] = len(noise[noise>thresh[n]])/1024

plt.plot(pf, pd)
plt.ylabel('Probability of Detection')
plt.xlabel('Probability of False Alarm')
```

The ideal detector would have a ROC curve that has 0 PFA and 1 PD, so it would follow (0,0) to (0,1) to (1,1). The straight line that connects (0,0) to (1,1) is known as the *guessing line*, because it is the ROC curve you would get by randomly guessing whether or not a signal is present.



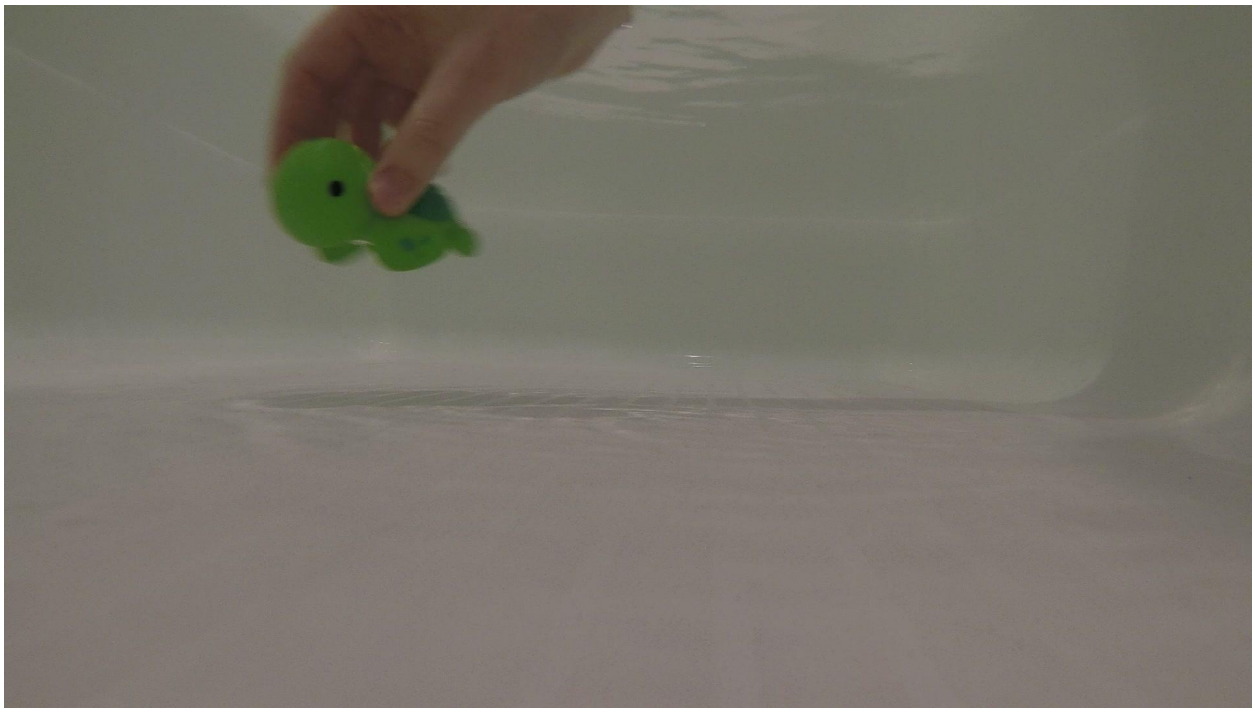
The resulting ROC curve from our code snippet looks like the image above. Here we can see the range of performance we could expect from our detector, and that it is somewhere between the guessing line and the ideal detector.

Try this function for the SNR values you calculated in step 7. What do you notice about the ROC curves as the SNR increases? What does that say about the performance of the detector?

9. The ROC curve shows the PD and PFA of a detector for all possible choices of detection threshold. In a practical detection algorithm we select a single threshold that lies at a desirable point on this curve, which we call the *operating point* of the detector. If your navigation controller is highly sensitive to false buoy detections, then you would choose a high threshold so the false alarm rate is low (at the cost of less detections). On the other hand, if you really need to detect the buoys every time and can deal with false alarms in your controller, then you could choose a lower threshold to increase the PD and allow more false alarms. Let's say you can accept a PFA of 0.1. How does PD change as the SNR increases from 1 to 10?

Detection on Imagery

For our navigation problem we will need to detect (and identify) buoys of specific colors. Reducing noise by averaging multiple measurements will not be a good option because the frame rate of the Pi camera is only about 1 frame per 2 seconds, and we are moving through the water so the scene is changing relatively quickly. However, these buoys will be collections of pixels bundled together with similar color so there may be an opportunity to average in the spatial dimensions. This is a 2-dimensional version of our signal in step 7. Can our smoothing approach help us here? Let's try it.



10. Open our familiar green turtle image from Lab 9 with cv2 and repeat the conversion to HSV and plotting the Hue channel that was found to be the most effective for detecting the turtle. Now apply a 40x40 pixel two-dimensional smoothing filter over that Hue channel using *hfilt = cv2.boxFilter(hsv[:,:,1], cv2.CV_32F, (40,40))*. Compare the outputs. In which image does the turtle stand out more from the background?
11. Compare the SNR in the two images by finding and squaring the max value of the hue on the turtle body and dividing by the variance of the image background, which you can approximate by selecting a section of the image that is considered

background and calculating the variance among those pixels. Does the calculation agree with what you see in the images?