Homework 4

1. Write a controller (matlab or Simulink) for a simple $\frac{1}{s}$ plant. Assume a unit step input for the reference, r(t).

(a) What type of controller did you use. Provide the Gain Margin, Phase Margin, closed-loop eigenvalues, and steady state error.
(b) What is the steady state error if the reference r(t) is a unit ramp input?
(c) Redesign the controller to track the ramp input and repeat *part a*.

Solution:
A proportional-integral (PI) controller was used for each part of this problem. The controller was designed to be under damped with a 1 second time to steady-state. Assuming, $t_{ss} = 1$ and $\zeta = 0.7071$, the controller gains were determined in *Equation 1*.

$$\begin{aligned} \omega &= \frac{4.6}{\zeta t_{ss}} = 6.5054 \\ K_i &= \omega^2 = 42.32 \\ K_p &= 2\zeta\omega = 9.2 \end{aligned} \tag{1}$$

The addition of integral control to an integrator plant makes our system Type II, and therefore has zero steady-state error for both a step input or a ramp input. The outputs of both scenarios are shown in *Figure 1*.

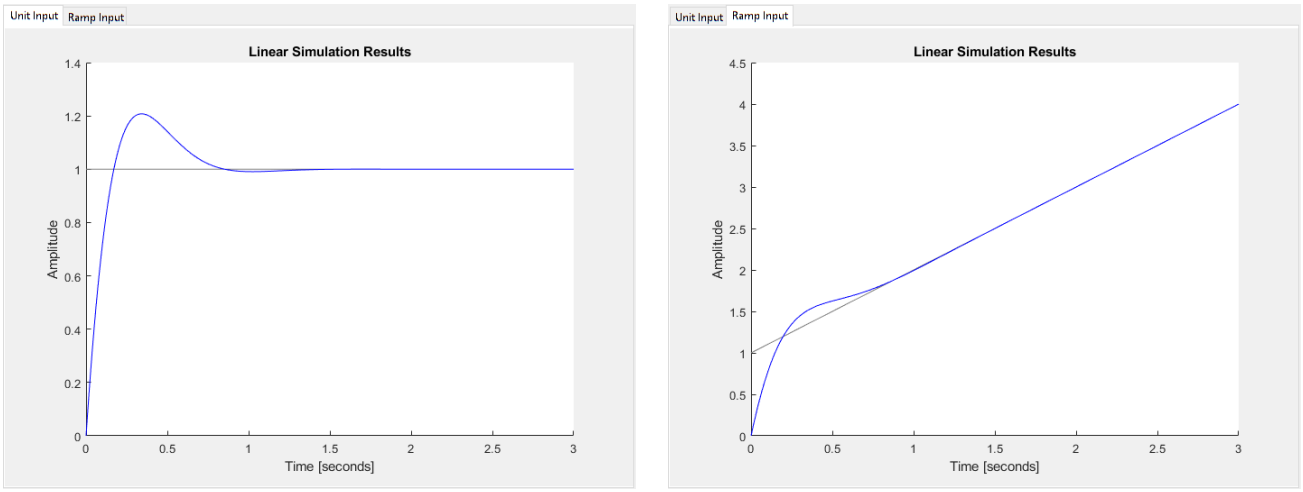

Figure 1: Simulation of Step and Ramp Input on Designed System.

Using the closed-loop transfer function (*Equation 2*):

$$G(s) = \frac{9.2s + 42.32}{s^2 + 9.2s + 42.32} \tag{2}$$

MATLAB's *eig* and *margin* functions can be used to determine the closed-loop eigenvalues as well as gain

and phase margin, *Equation 3* and *Figure 2*.

$$s = -4.6 \pm 4.6i \tag{3}$$

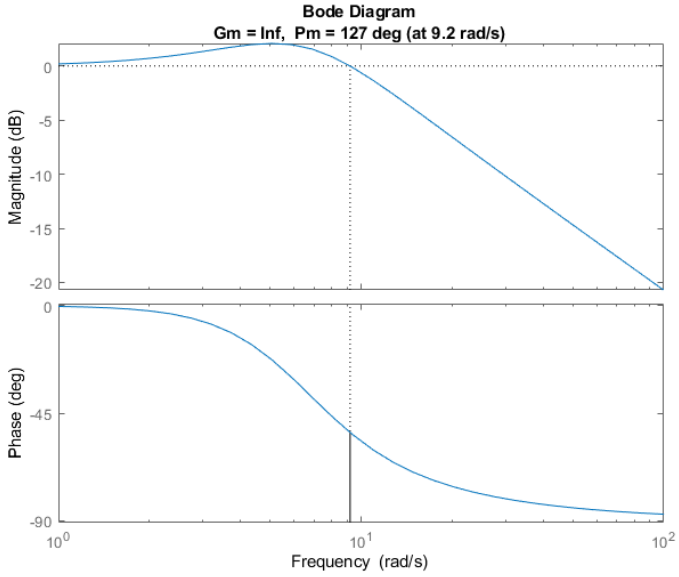

Figure 2: Gain and Phase Margin of Closed Loop System.

2. Take the sampled 100 Hz sine wave (generate_signal(1)) from the website (sampled at 1 MHz, i.e. Ts=1e-6).

(a) Develop a simple (1 Hz) PLL to track the phase of the signal. Provide plots of phase, phase error, as well as the estimated signal vs. true signal.
(b) Double the PLL bandwidth and repeat *part a*.
(c) Determine the true frequency and repeat *part a*.
(d) Modify your PLL to work from the unknown frequency and repeat part a assuming 10 Hz signal.

Solution:
To create a controller with a bandwidth of 1 Hz, $\zeta = 0.7071$, $BW = 1$ and open loop gain of $K = 4$ were chosen. Using *equation* **??**, the controller values were determined:

$$\begin{aligned} \omega &= \frac{4BW\zeta}{1+\zeta^2} \\ K_i &= \omega^2 \\ K_p &= \zeta\omega \end{aligned} \tag{4}$$

The following code was used for the PLL and costas loops.

```
for i = 2:length(loop)
  % data signal
```

```
    sig = signal(beg:fin);
    % Inphase and Quadrature
    ssig = sin(2*pi*f(i-1).*t' + theta(i-1));
    csig = cos(2*pi*f(i-1).*t' + theta(i-1));
    I(i) = sig * ssig;
    Q(i) = sig * csig;
    % error
    disc(i) = atan(Q(i) / I(i));
    % propagate
    f(i) = f(i-1) + K * (Ki*t_int*disc(i) + Kp*(disc(i) - disc(i-1)));
    theta(i) = rem(theta(i-1) + 2*pi*f(i-1)*t_int, 2*pi);
    beg = beg + numSamp;
    fin = fin + numSamp;
end
```

Figure 3 contains the outputs of the PLL for *part a*. As shown, the controller locks on while the frequency of the incoming signal is 100 Hz. As soon as the frequency changes, the controller is no longer able to track the signal completely and ends up oscillating wildly around a frequency of 110 Hz.

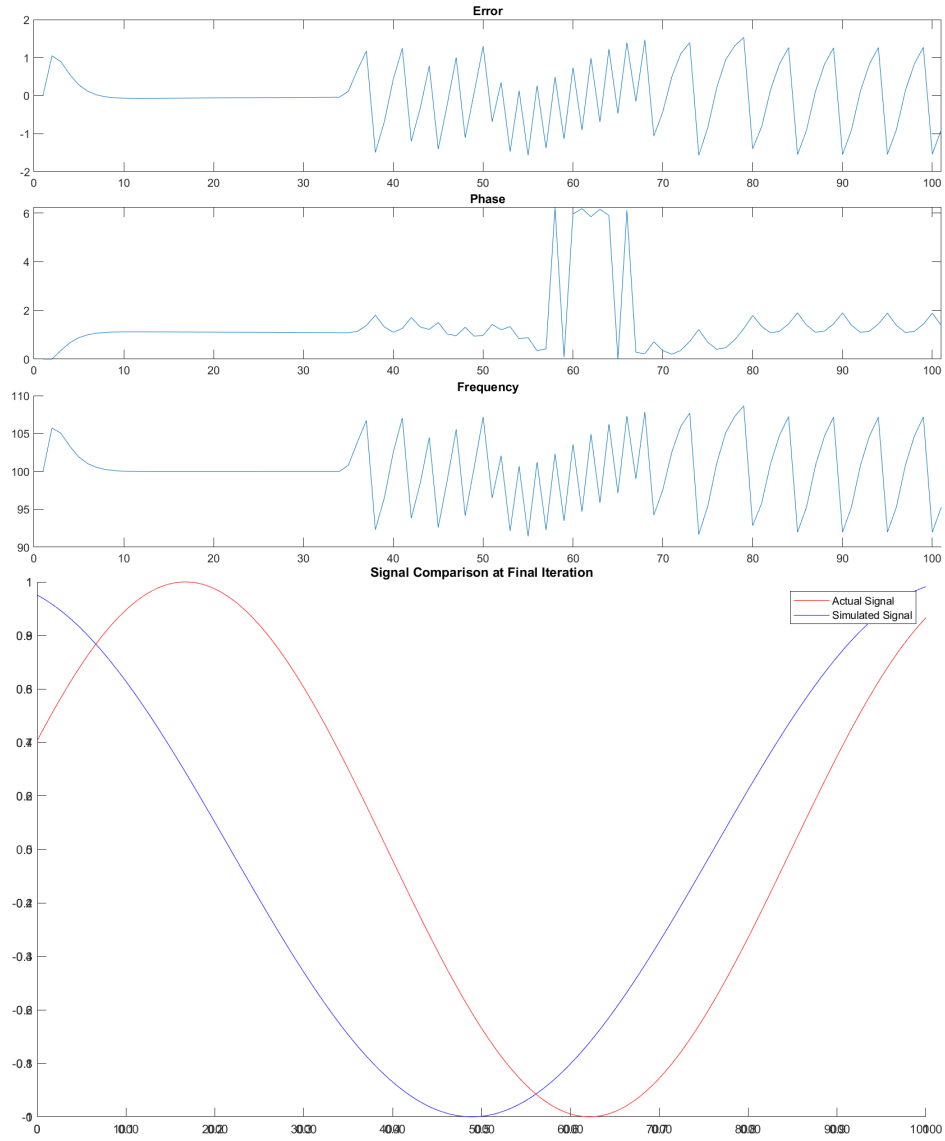

Figure 3: PLL Part A.

A similar response is received when the bandwidth is doubled to 2 Hz. The only difference is more initial oscillation happens, shown in *Figure 4*. Additionally, the overshoot at 110 Hz is reduced.

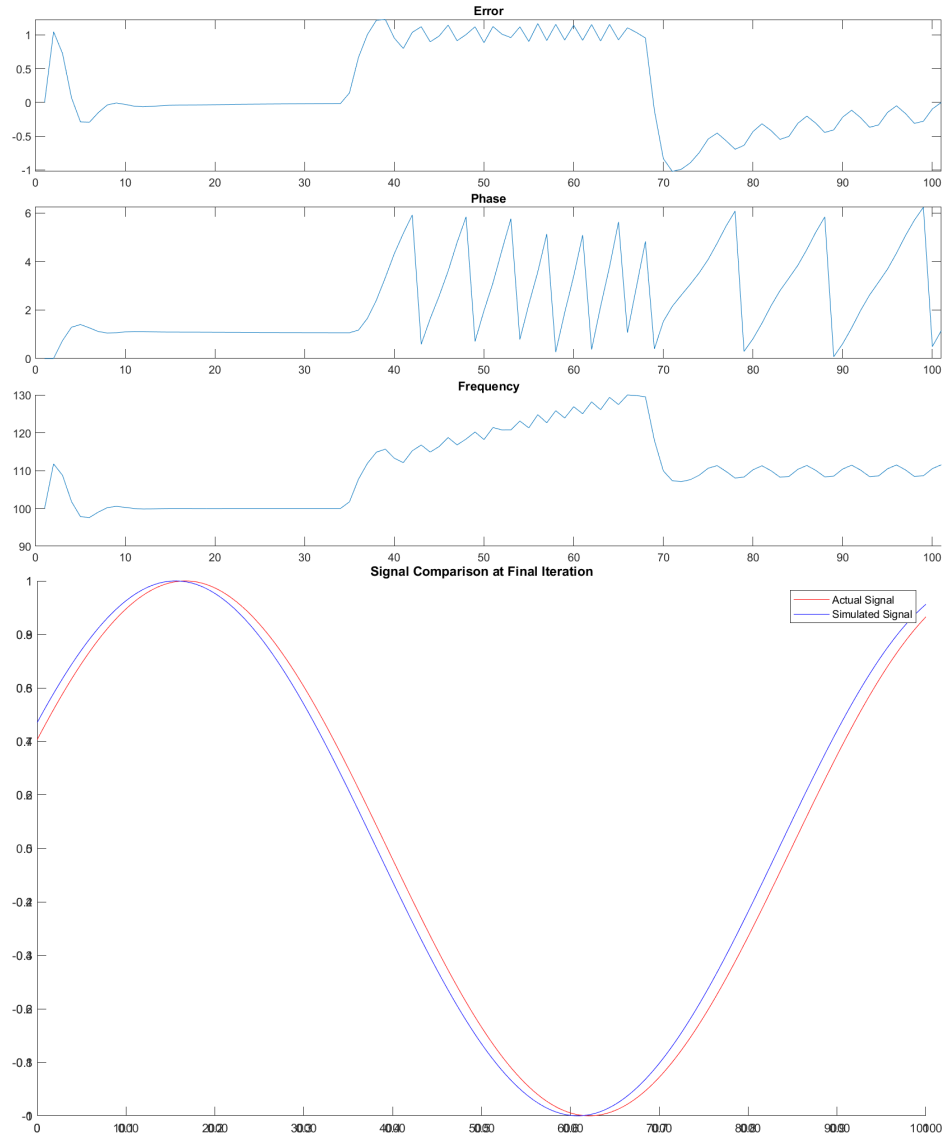

Figure 4: PLL Part B.

Switching the nominal frequency to 110 Hz instead of 100 Hz results in *Figure 5*. There is a higher phase error magnitude, but the system still converges to 100 Hz. Again the frequency change causes oscillation around the 110 Hz frequency.

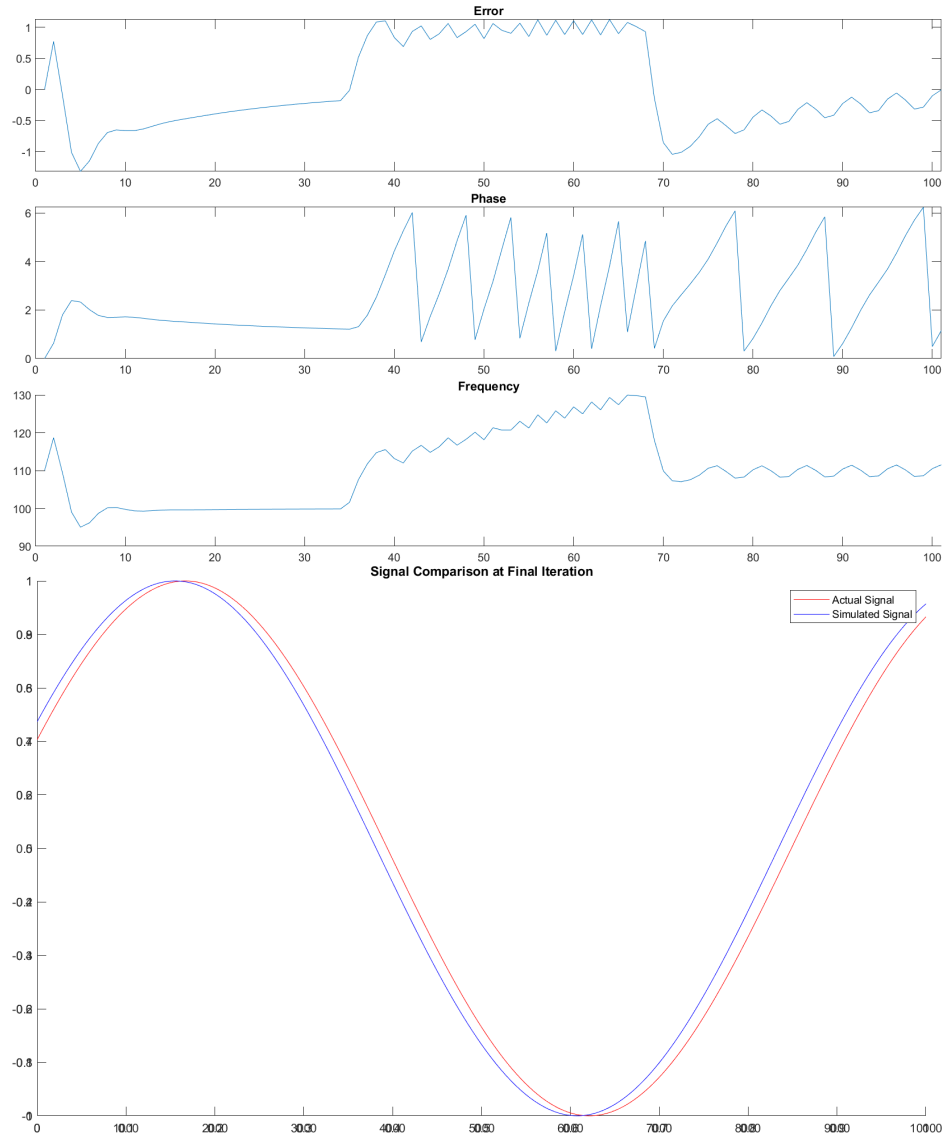

Figure 5: PLL Part C.

Lastly, assuming a nominal frequency of 10 Hz, the PLL designed in *part a* completely fails (*Figure 6*).

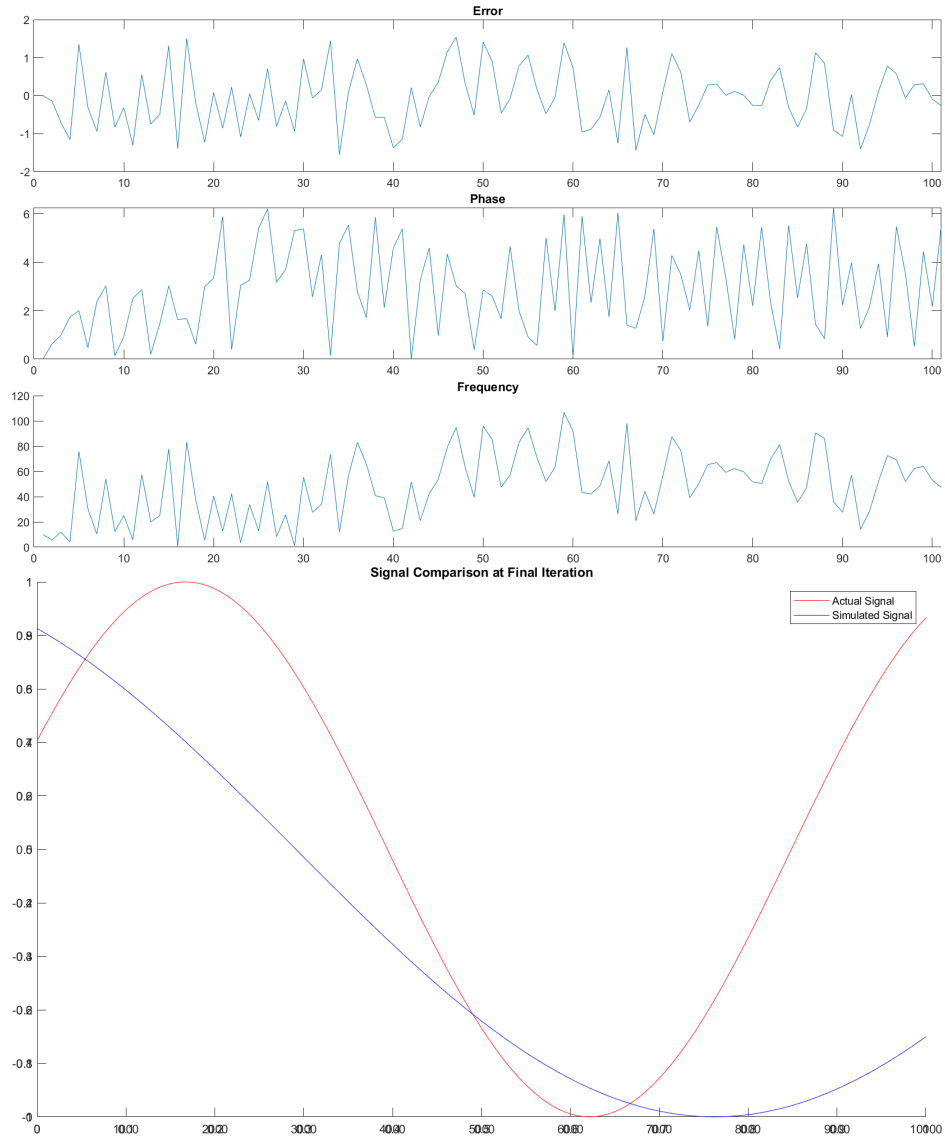

Figure 6: PLL Part D.

3. Modify your PLL from problem #2 to operate as a Costas loop filter. Take the 88 second data (generate_signal(2)) sampled at 1 MHz and decode the data message on the 100 Hz sinusoid using your Costas loop filter. The data bits are 1 second wide and are comprised of 8 bit ascii characters.

Solution:

The same controller designed and discriminator used in *Problem 2* was used for the Costas Loop. The only addition was using of the ”sign” of the Inphase values as the values for the binary sequence. The resulting message was ”**AU WarEagle**”.

4. Develop a simple DLL to phase align the sequence shown below with the digital signal (generate_signal(3)) provided on the website. The sequence is sampled at 10 samples per chip. Plot the delay vs. time (or frequency vs. time depending on your implementation).

$$Sequence = \begin{bmatrix} 1 & -1 & -1 & -1 & 1 & -1 & 1 & 1 \end{bmatrix}$$

Solution:

For this, a proportional only controller works well with $K_p = 5$. The overall setup for a basic DLL is very similar to the PLL except it utilizes a different discriminator. For 1/2 chip spacing, the discriminator is $0.5\dfrac{E-L}{E+L}$. The following code was used for the DLL and *Figure 7* shows the output.

```
for i = 2:length(loop)
  % signal chunk
  sig = signal(beg:fin);
  % autocorrelatation
  E = correlation(sig, sequence, 'standard', tau(i-1)+s);
  L = correlation(sig, sequence, 'standard', tau(i-1)-s);
  % error
  disc(i) = 0.5 * (E-L)/(E+L);
  tau(i) = round(tau(i-1) + Kp*disc(i));
  beg = beg + 80;
  fin = fin + 80;
end
```

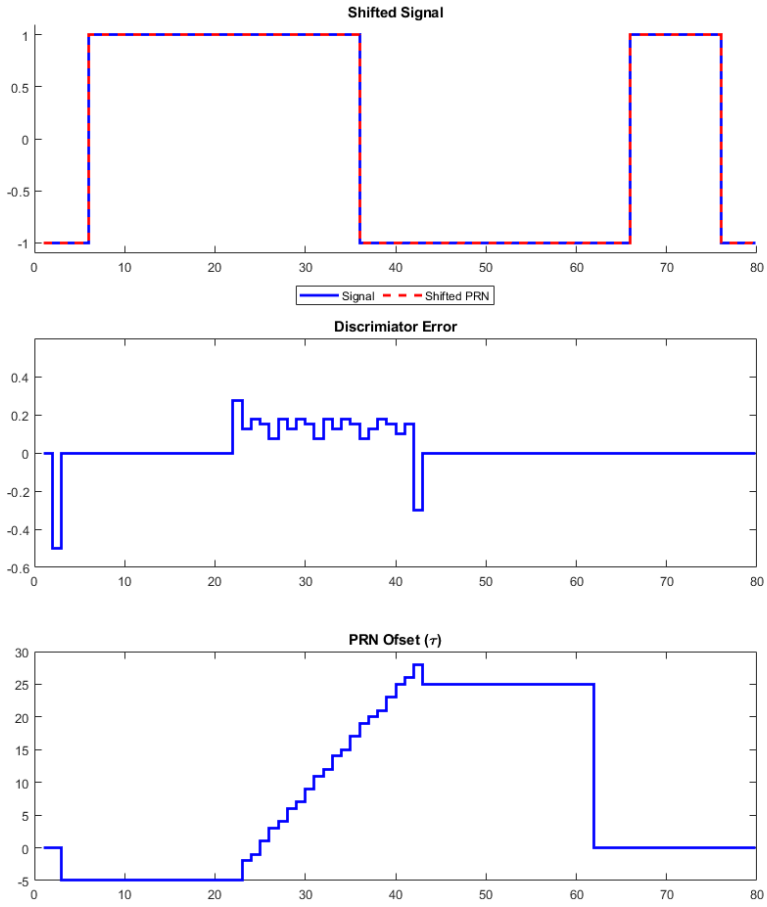

Figure 7: Basic DLL.

5. Combine your Costas Loop filter from problem #3 and your DLL from problem #4 to decode the data signal (generate_signal(4)).

Solution:

Signal 4 was incomplete, therefore this problem could not be completed.

6. Take the PRN code for SV #4 or #7 (i.e. from HW #3) and upsample it such that there are 16 samples at each chip (i.e. the length of this vector will be 1023*16 long).

(a) Show the autocorrelation calculation from -5 chips to +5 chips in 1/16 chip increments.

(b) Repeat with noise ($\sigma = 0.2$) added to the non-shifted signal

Solution:

For this problem, the PRN could be easily upsampled by an even number using MATLAB's *repelem* function. Running the autocorrelatation for this upsampled PRN and an upsampled PRN with added noise results in *Figure 8*. As shown, adding noise has visually no effect on the correlation of the sequences proving even a noisy signal, like GPS, can be correlated properly.

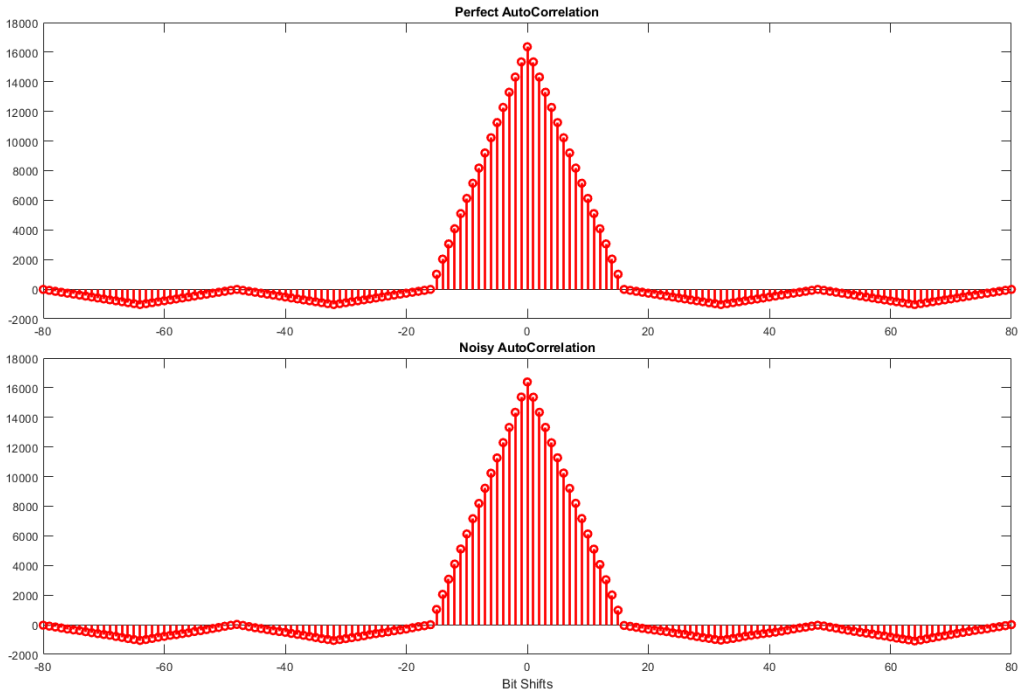

Figure 8: Upsampled Autocorrelatation.

7. Write your own acquisition software to acquire a single satellite from the IFEN IF data file. You can write a serial or parallel search algorithm. Provide a plot of the acquisition plane (Code and Doppler) and provide the code phase and Doppler results. The C/A (Gold) codes for several satellites in view are on the website.

Solution:

Using Tanner Watts's thesis, the technique for parallel satellite acquisition was learned and utilized in the following code for satellite 7:

```
idx = ceil(1/nSamples : 1023/nSamples : 1023-(1/nSamples));
f_dopp = -10000:100:10000;
[sig1, ~] = fread(fid, n_ms*nSamples, 'int8');
% PRNS IN SEARCH
for k = 1:32
  prn_up = ca_code(k, idx)';
  % DOPPLER BINS
```

```
    for j = 1:length(f_dopp)
        % tanner watts's thesis
        f = f_if + f_dopp(j);
        I = sig1 .* sin(2*pi*f*t_samp);
        Q = sig1 .* cos(2*pi*f*t_samp);
        R1(:,j,k) = abs(ifft(fft(I + Q.*1j) .* conj(fft(prn_up)))).^2;
    end
end
```

This PRN was acquired at a code shift of 989 (-34) chips and a doppler offset of -500 Hz using only 1 ms of data. The acquisition plane is shown in *Figure 9*. Using a 10 ms acquisition time results in a much cleaner acquisition but still gets acquired at the same code and doppler shifts (*Figure 10*). Interesting to note, over the 10 ms acquisition period, 10 maximum peaks are observed at the same PRN offset in each of the 10 replications of the PRN. Also iterating through all 32 satellites it can be determined that satellites **1**, **7**, **14**, **17**, **19**, **21**, and **30** are all in view after the first 1 ms of data.

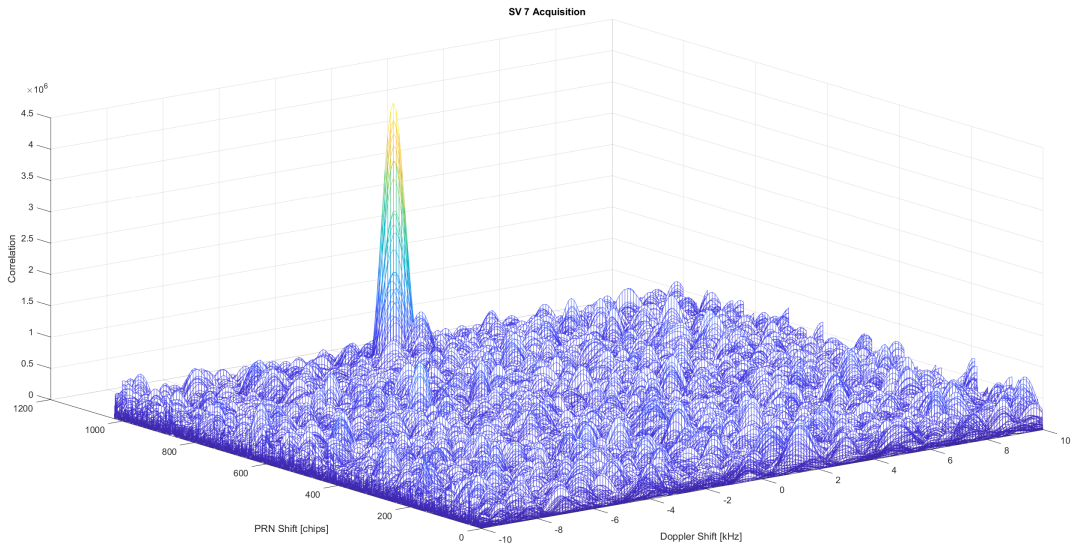

Figure 9: Satellite 7 Acquisition over 1 ms.

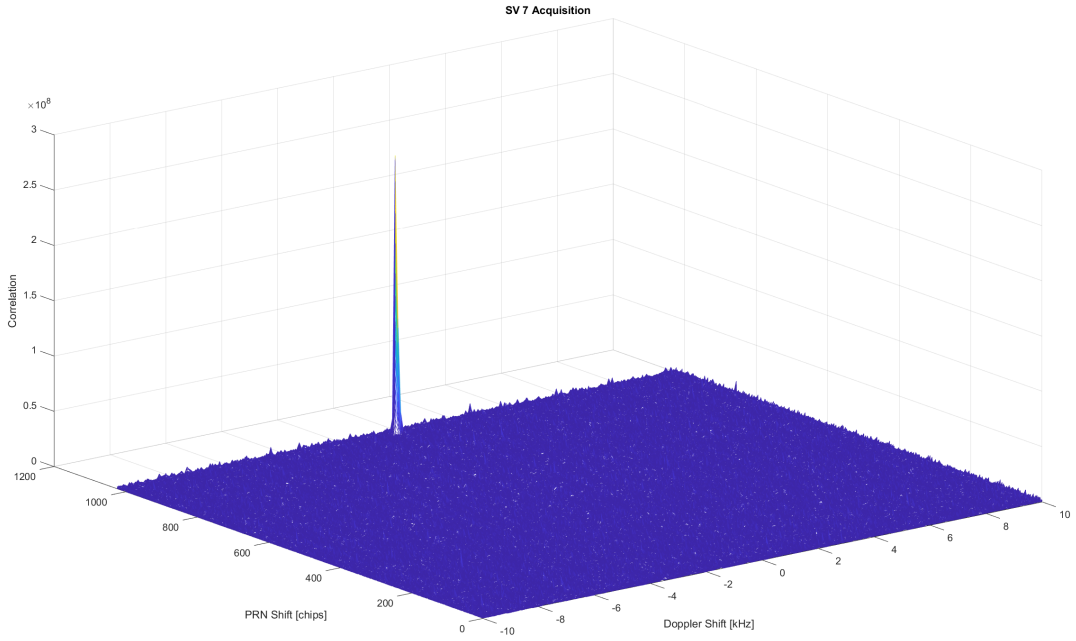

Figure 10: Satellite 7 Acquisition over 10 ms.