Signal Processing Is Easy (By Example)

In this exercise we will explore some of the communication signal processing pieces available in MATLAB and how they can be used with PlutoSDR.

Starting with script lab1part3.m:

```
Command Window
fx >> edit lab1part3.m
```

This script sets up a loopback-based system which utilizes a QPSK signal to transmit data through PlutoSDR. Since we control both the transmitter and receiver, and since they share the same clock, we can perform some useful testing with our algorithms.

The default script generates a signal, passes it through PlutoSDR, and performs Timing Recovery with the "comm.SymbolSynchronizer" System Object. Timing recovery is necessary since there is a random delay between transmitter and receiver and the transmitter and receiver LO's on Pluto have random phase differences.

Run the unedited script and inspect the constellation diagrams produced.

```
Command Window
>> lab1part3
Overflow events: 0 of 10
Performing Timing Synchronization
fx >>
```

Running the script multiple times, you will observe different constellations each time. This is due to the impairments already discussed. An example constellation set is shown in Figure 5.

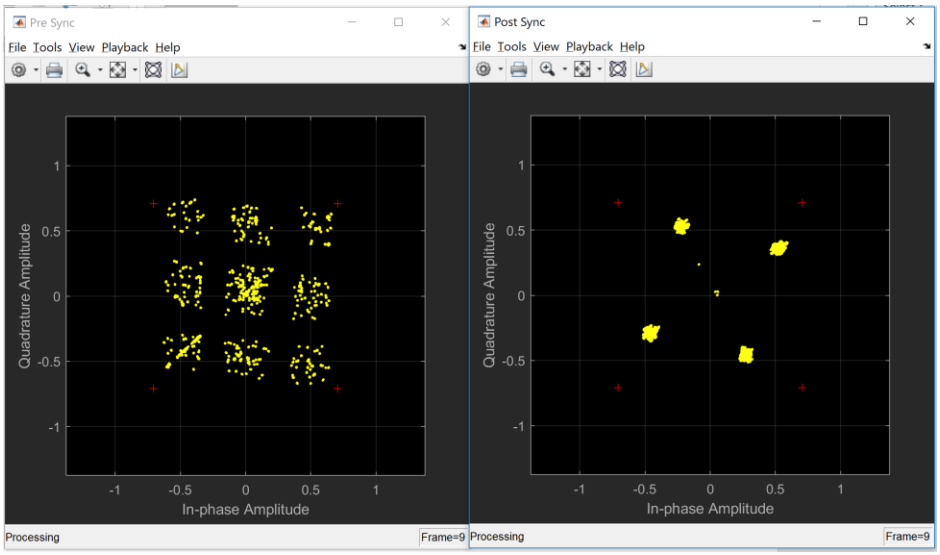

Figure 5 Constellation of QPSK signal before and after timing recovery

Next modify line 10 in the provided script to change the frequency offset between radios to 100Hz:

```
6    %% CHANGE ME HERE
7 —  SampleRate = 1e6;
8 —  SamplesPerRXFrame = 2^16;
9 —  FramesToCollect = 10;
10 — FrequencyOffset = 100;
```

Now run the script again with this change.

```
Command Window
>> lab1part3
Overflow events: 0 of 10
Performing Timing Synchronization
fx >>
```

Looking at the constellations in Figure 6 we can notice a rotation, which is expected due to the frequency difference between the receiver and transmitter. Since PlutoSDR has independent LO's for the transmitter and receiver, we can configure the radio in this mismatched configuration.

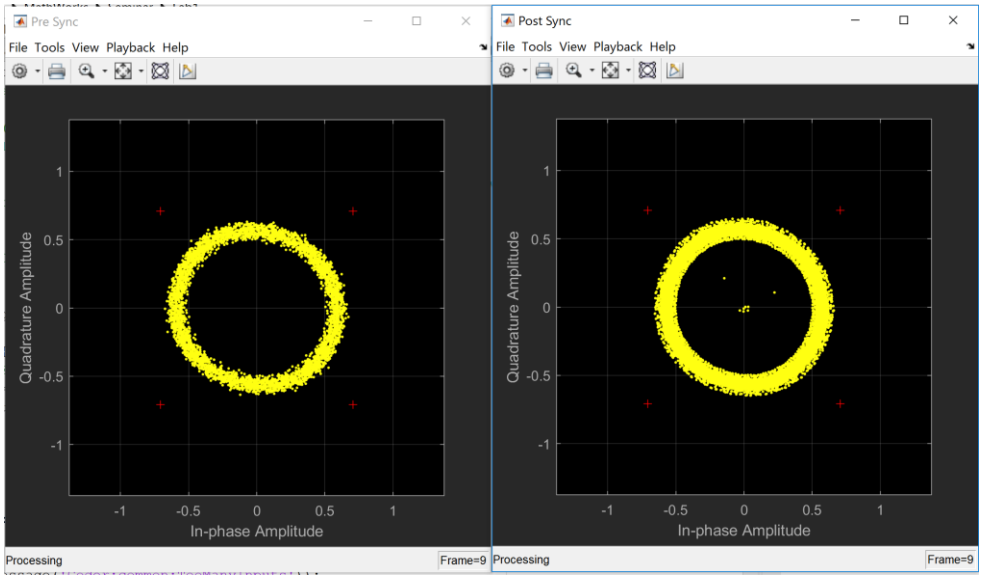

Figure 6 Constellation of signal before and after timing recovery with remaining carrier offset.

When transmitting between two different radios, we will always have to compensate for frequency offset since they will have different LO's. We are emulating this effect in a single radio by selecting different center frequencies for receive and transmit.

We can compensate for this carrier frequency offset by performing carrier synchronization. For this we will use the comm.CarrierSynchronizer System object, which implements the necessary algorithms to perform carrier recovery on a timing synchronized signal.

To enable this synchronization, uncomment lines 47-49 so they appear as:

```
46      %% Insert Carrier Synchronization here
47 -    fprintf('Performing Carrier Synchronization\n');
48 -    cs = comm.CarrierSynchronizer('SamplesPerSymbol',1);
49 -    savedPost = cs(savedPost);
```

Rerun the script with these modifications to visualize the timing- and frequency-synchronized signal. This should look similar to Figure 7 below, which has the correct phase orientation to the constellation for QPSK.

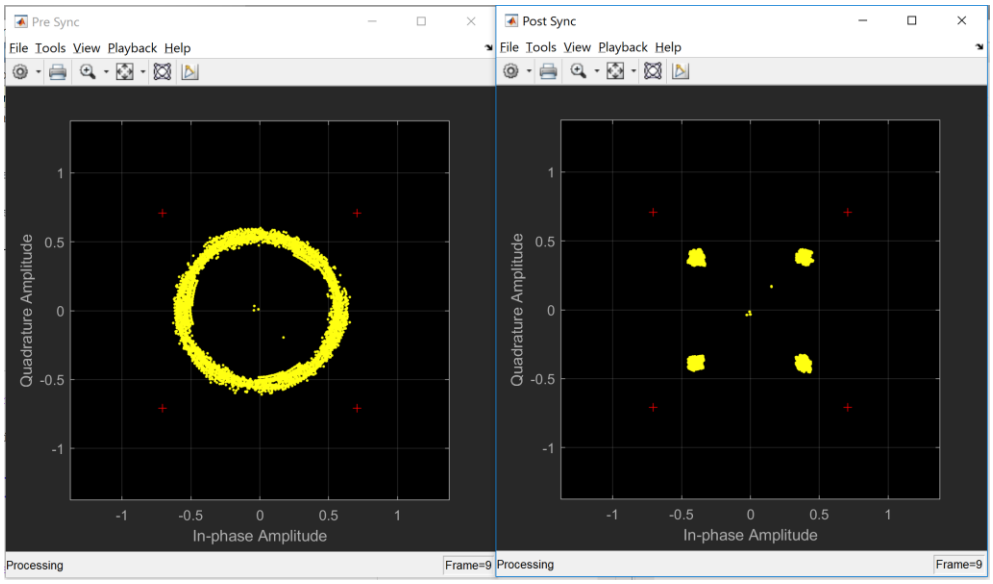

Figure 7 Constellation of QPSK recovery signal before and after carrier and timing synchronization

Now modify line 10 to increase the offset to 4Khz and rerun the script.

We now observe there is still rotation even after synchronization, producing a constellation similar to Figure 6 as in Figure 8.

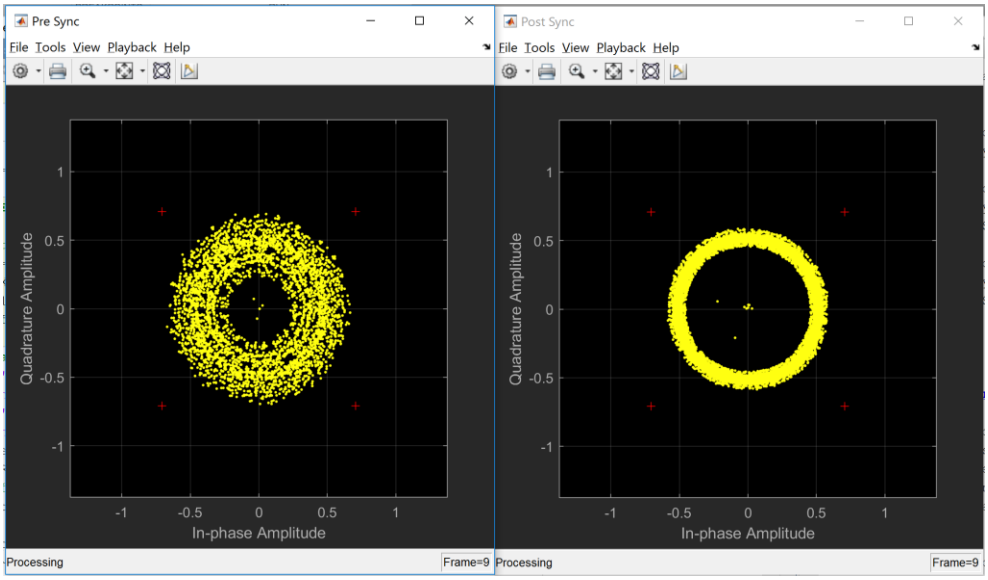

Figure 8 Constellation of QPSK recovery signal before and after carrier and timing synchronization with remaining offset

It is now your task to update the comm.CarrierSynchronization System object's parameters to produce a stable (non-rotating) constellation. If you can accomplish this task, increase the offset to 5kHz and repeat this process if you are feeling adventurous.