**Digital Communications** **Laboratory Report**

Carrier Recovery using

Costas Loop

by JIE TIAN

2802461t

Rankine 329

Fubruary 9, 2024

1. **Importing libraries and setting input data**

Before we start the Demodulation with Carrier Recovery, we need to import the necessary Python libraries into the project, we can use the functions in these libraries to draw graphs, generate some random numbers, filter, and some array operations. The code below shows the libraries we need to use in this project.

*import numpy as np*

*from matplotlib import pyplot as plt*

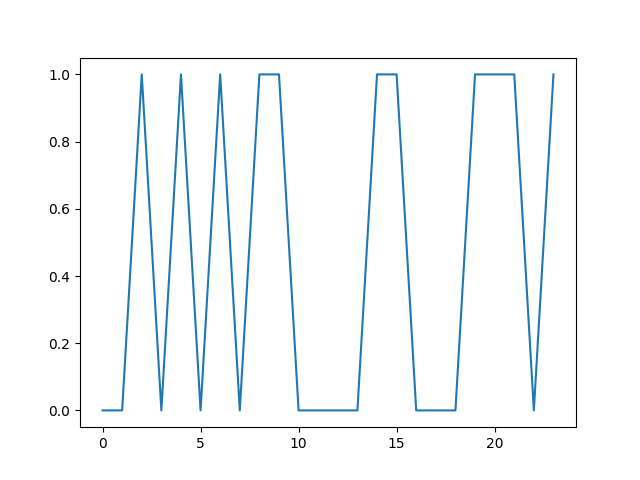
*from scipy import signal*

*from numpy import random*

For the input data, we will use the 24-bit binary number to represent our student number. The following function converts our student number to a 24-bit binary number for our later modulation.

*def bin\_array(num, m):  
 """Convert a positive integer num into an m-bit bit vector"""  
 return np.array(list(np.binary\_repr(num).zfill(m))).astype(np.bool\_)  
  
# import 24 bit digital data  
id\_num = 2802461  
Nbits = 24  
tx\_bin = bin\_array(id\_num, Nbits)*

The following is a 24-bit binary display of the student number: 2802461.



1. **VCO Cordic Digital Clock and initialize constants and variables**

The current state of the clock is signified by 2 floating point numbers representing the in-phase and quadrature components sampled values. Then each successive state is obtained by the Cordic transformation using cos 𝑓0 and sin 𝑓0, where 𝑓0 = 𝑓𝑖 + 𝛼𝑣 is the scaled voltage-dependent frequency, including a linear applied voltage dependence. Take the previous value for the carrier frequency as the initial value 𝑓𝑖, here is the relevant code.

*# VCO Cordic Digital Clock and initialize constants and variables*

*myClock = np.array([1.0, 0.0])*

*bit\_period = 128*

*fc = 1 / 32*

*fref = fc \* (1 + 0.02 \* (random.rand() - 0.5))*

*pref = 2 \* np.pi \* random.rand()*

*volt = 0*

*vout = np.array(volt)*

*cout = myClock[0]*

*rout = np.cos(pref)*

*dout0 = np.empty(0)*

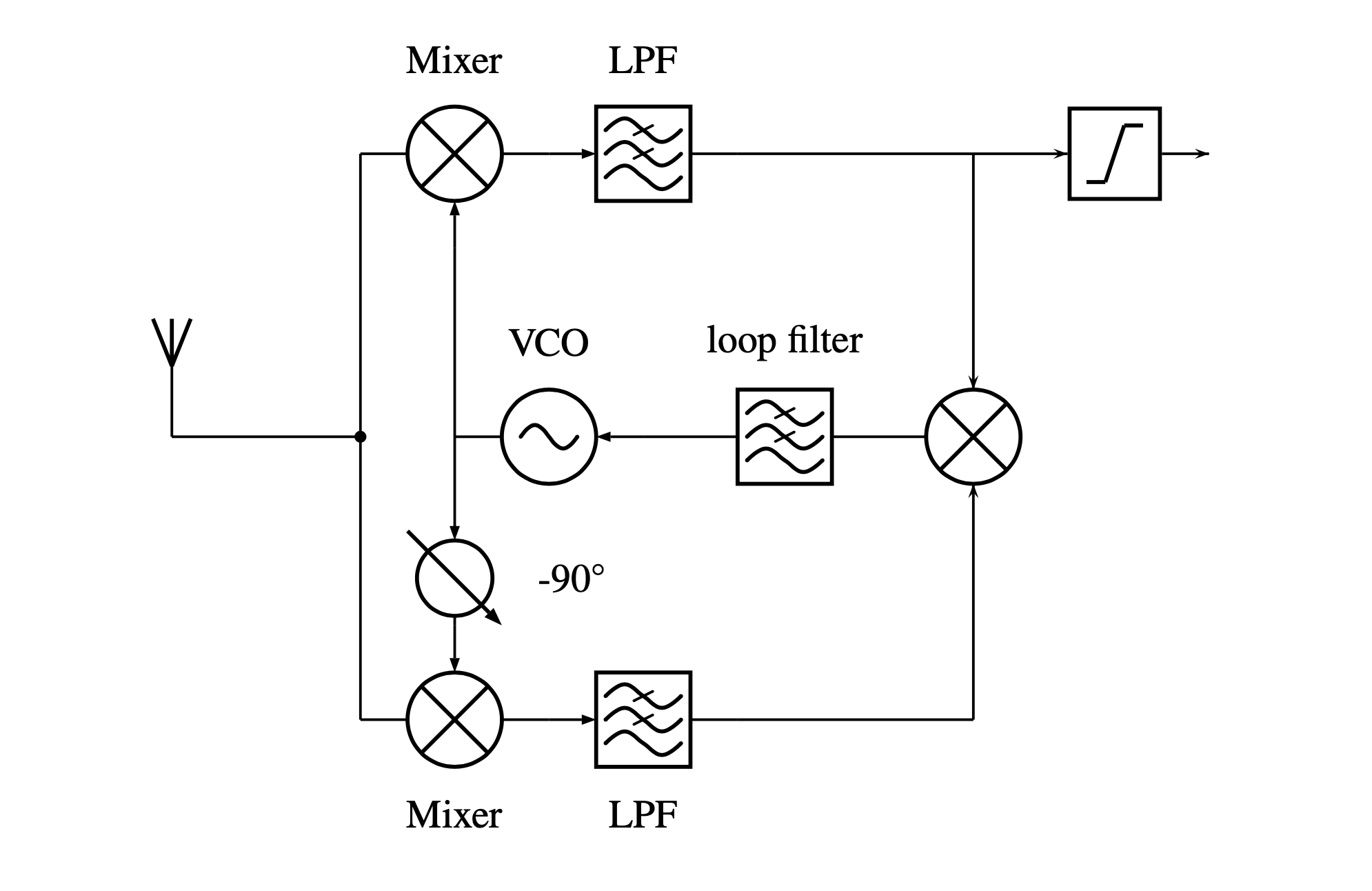
*tx\_mod = np.empty(0)*

*c = np.cos(2 \* np.pi \* fc \* (1 + 0.25 \* volt))*

*s = np.sin(2 \* np.pi \* fc \* (1 + 0.25 \* volt))*

*myClock = np.matmul(np.array([[c, -s], [s, c]]), myClock)*

1. **Demodulation with Carrier Recovery**

****

It starts with splitting the incoming modulated signal into two paths, each going through a mixer. One path is mixed with a sine component of a voltage-controlled oscillator (VCO), and the other with a cosine component, to produce in-phase (I) and quadrature (Q) components. These components are then processed to generate a voltage signal that adjusts the VCO, ensuring it remains locked to the phase of the incoming signal’s carrier.

The final step in the demodulation of digital data is to select an appropriate sample point in dout0, and then use a thresholding function to convert the floating point into a boolean. We use the most obvious sample point by taking the midpoint of the bit, i.e. i \* bit\_len + bit\_len // 2. Therefore construct a for loop over Nbits and applying a threshold to the bit midpoint. In this case, an appropriate threshold function is the heaviside (step) function.

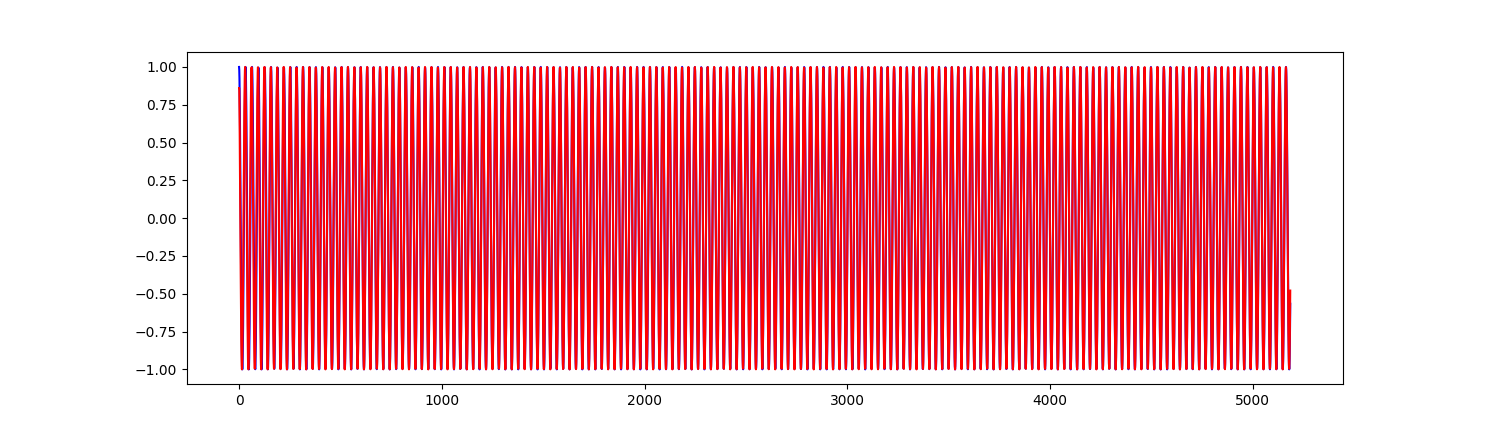
The following are the relevant code and resultant diagrams.

*for i in range(bit\_period \* (prelude.size + Nbits) + numtaps // 2):  
 mixed[0, :] = np.append(mixed[0, 1:], myClock[0] \* (2 \* tx\_bin[(i // bit\_period) % (prelude.size + Nbits)] - 1) \* np.cos(pref + 2 \* np.pi \* fref \* i))  
 mixed[1, :] = np.append(mixed[1, 1:], -myClock[1] \* (2 \* tx\_bin[(i // bit\_period) % (prelude.size + Nbits)] - 1) \* np.cos(pref + 2 \* np.pi \* fref \* i))  
  
 for j in range(2):  
 lpmixed[j] = np.sum(b1 \* mixed[j, :])  
 volt = lpmixed[0] \* lpmixed[1]  
  
 c = np.cos(2 \* np.pi \* fc \* (1 + 0.25 \* volt))  
 s = np.sin(2 \* np.pi \* fc \* (1 + 0.25 \* volt))  
 myClock = np.matmul(np.array([[c, -s], [s, c]]), myClock)*

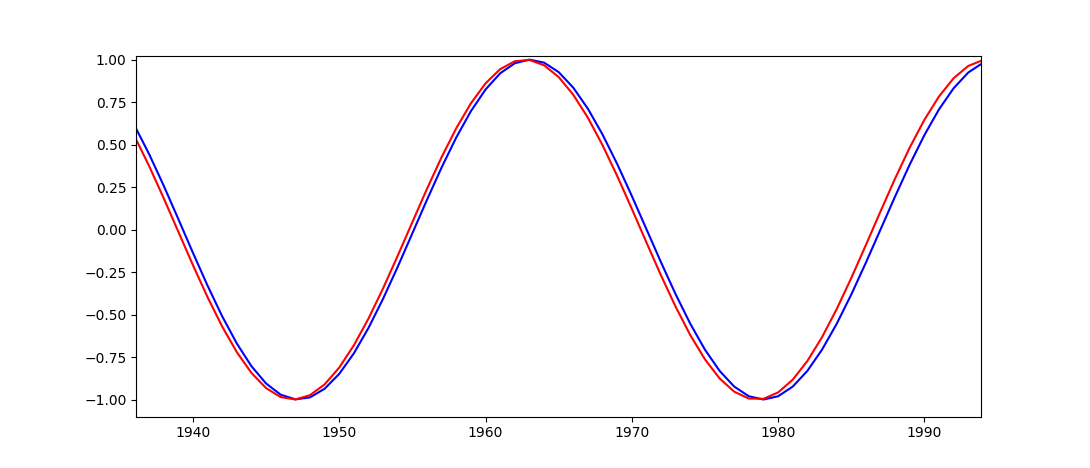
*for i in range(prelude.size + Nbits):  
 rx\_bin = np.append(rx\_bin, np.uint8(np.heaviside(dout0[(2 \* i + 1) \* bit\_period // 2 + numtaps // 2], 0)))*



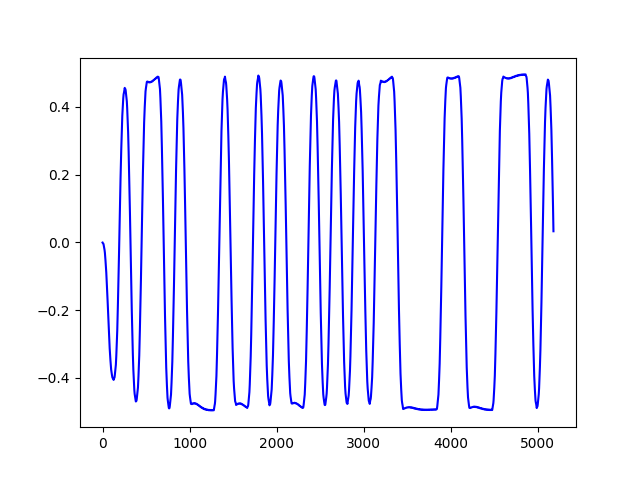
*Voltage variation diagram*



*Reference waveform and the output of our oscillator*

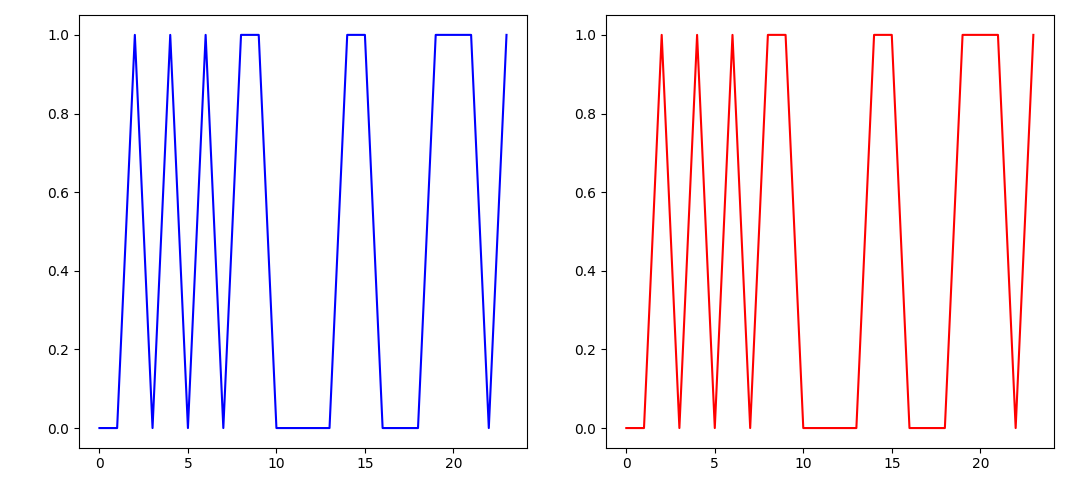


*Reference waveform and the output of our oscillator(zoom in)*

**

*Waveform obtained after filter*

The demodulated waveform is consistent with the original waveform.



1. **carrier frequency range**

I tested different ranges of carrier frequencies, the following bit\_period is set to 128 and when the amplitude exceeds 2%, an error occurs.

*/usr/bin/python3 /Users/george/Project/DC Project/Carrier Recovery/Test.py*

*carrier frequency range is 0.25 %, Error rate is 0 %*

*carrier frequency range is 0.5 %, Error rate is 0 %*

*carrier frequency range is 0.75 %, Error rate is 0 %*

*carrier frequency range is 1.0 %, Error rate is 0 %*

*carrier frequency range is 1.25 %, Error rate is 0 %*

*carrier frequency range is 1.5 %, Error rate is 0 %*

*carrier frequency range is 1.75 %, Error rate is 0 %*

*carrier frequency range is 2.0 %, Error rate is 0 %*

*carrier frequency range is 2.25 %, Error rate is 11 %*

*carrier frequency range is 2.5 %, Error rate is 19 %*

*carrier frequency range is 2.75 %, Error rate is 28 %*

*carrier frequency range is 3.0 %, Error rate is 39 %*

*carrier frequency range is 3.25 %, Error rate is 39 %*

*carrier frequency range is 3.5 %, Error rate is 44 %*

*carrier frequency range is 3.75 %, Error rate is 49 %*

*carrier frequency range is 4.0 %, Error rate is 55 %*

*carrier frequency range is 4.25 %, Error rate is 51 %*

*carrier frequency range is 4.5 %, Error rate is 55 %*

*carrier frequency range is 4.75 %, Error rate is 52 %*

The following bit\_period is set to 256, and when the amplitude exceeds 2.5%, an error occurs.

*carrier frequency range is 0.25 %, Error rate is 0 %*

*carrier frequency range is 0.5 %, Error rate is 0 %*

*carrier frequency range is 0.75 %, Error rate is 0 %*

*carrier frequency range is 1.0 %, Error rate is 0 %*

*carrier frequency range is 1.25 %, Error rate is 0 %*

*carrier frequency range is 1.5 %, Error rate is 0 %*

*carrier frequency range is 1.75 %, Error rate is 0 %*

*carrier frequency range is 2.0 %, Error rate is 0 %*

*carrier frequency range is 2.25 %, Error rate is 0 %*

*carrier frequency range is 2.5 %, Error rate is 0 %*

*carrier frequency range is 2.75 %, Error rate is 9 %*

*carrier frequency range is 3.0 %, Error rate is 16 %*

*carrier frequency range is 3.25 %, Error rate is 25 %*

*carrier frequency range is 3.5 %, Error rate is 23 %*

*carrier frequency range is 3.75 %, Error rate is 38 %*

*carrier frequency range is 4.0 %, Error rate is 32 %*

*carrier frequency range is 4.25 %, Error rate is 40 %*

*carrier frequency range is 4.5 %, Error rate is 43 %*

*carrier frequency range is 4.75 %, Error rate is 41 %*