Simulación de un Lazo Enganchado de Fase utilizando Python

J.J. Cáceres[1], F. Caviglia[2]
Instituto Balseiro. Centro Atómico Bariloche. 8400-S.C. de Bariloche, Río Negro, Argentina
8 de agosto de 2020

Resumen

En este informe se presenta y describe el desarrollo de un programa para la simulación de un circuito PLL. El lenguaje de programación utilizado fue Python en su versión 3.8.4. Inicialmente se implementó cada uno de los tres módulos principales que componen un PLL: el comparador de fase, el filtro pasa bajos y el oscilador controlado por voltaje. Cada componente se probó primero por separado y luego como un todo. Se observó el comportamiento de la simulación indicando distintos parámetros y con distintas señales de entrada. Se determinó el comportamiento cuando la señal tenía un error en fase y en frecuencia. Utilizando la implementación del PLL, se simuló también un par emisor-receptor donde la señal transmitida consistía en una portadora modulada en amplitud por una serie de pulsos binarios.

1. Introducción

El lazo de seguimiento de fase, bucle de enganche de fase, o sencillamente PLL (acrónimo del inglés *phase-locked loop*) es un sistema de control que genera una señal de salida cuya fase y frecuencia está relacionada con la fase y frecuencia de una señal de entrada. Hay varios tipos diferentes, el que se buscó desarrollar en este trabajo consta de una señal de entrada que pasa a través de un detector de fase junto con la señal que viene del oscilador controlado por tensión (VCO). El resultado se pasa por un filtro paso bajo, y la tensión resultante se aplica al VCO que modificará entonces su frecuencia de oscilación. Un esquema en bloques del PLL se muestra en la Fig. 1 [1].

Mantener la fase de entrada y salida iguales también implica mantener las frecuencias de entrada y salida iguales. En consecuencia, además de las señales de sincronización, un bucle de seguimiento de fase puede rastrear una frecuencia de entrada, o puede generar una frecuencia que es un múltiplo de la frecuencia de entrada. Estas propiedades se utilizan por ejemplo para la sincronización del reloj de la computadora, la demodulación y la síntesis de frecuencia.

Los lazos de seguimiento de fase se emplean ampliamente en la radio, las telecomunicaciones, computadoras y otras aplicaciones electrónicas. Se pueden usar para demodular una señal, recuperar una señal de un canal de comunicación ruidoso, generar una frecuencia estable a múltiplos de una frecuencia de entrada (síntesis de frecuencia) o distribuir pulsos de reloj temporizados con precisión en circuitos lógicos digitales como los microprocesadores. Dado que un solo circuito integrado puede proporcionar un bloque de construcción de circuito cerrado de fase completo, la técnica se usa ampliamente en dispositivos electrónicos modernos, con frecuencias de salida que van desde una fracción de hertz hasta muchos gigahercios.

1.1. Componentes del PLL

1.1.1. Detector de fases

El detector de fases más simple consiste en un multiplicador ideal de señales. Los multiplicadores de señales son circuitos con dos entradas v_x y v_y tales que a su salida generan

$$v_{\text{out}} = K v_x v_y,$$

donde K es una constante con dimensiones de V^{-1}. En función de la polaridad de las entradas que acepta el multiplicador, se puede clasificar como

1. Multiplicador de cuatro cuadrantes: admite entradas bipolares,
2. Multiplicador de dos cuadrantes: admite una entrada unipolar y la otra bipolar,
3. Multiplicador de un cuadrante: admite sólo ambas entradas unipolares.

Los multiplicadores se caracterizan en base a su precisión y su linealidad. Definiciones precisas de estas características son

- La precisión de un multiplicador representa la máxima desviación de la salida del multiplicador ideal respecto de la función de transferencia ideal.
- La linealidad de un multiplicador se mide como la máxima desviación de salida, relativa a la recta que mejor aproxima a la curva de salida del multiplicador respecto de una de las entradas, cuando la otra es mantenida constante en su valor máximo.

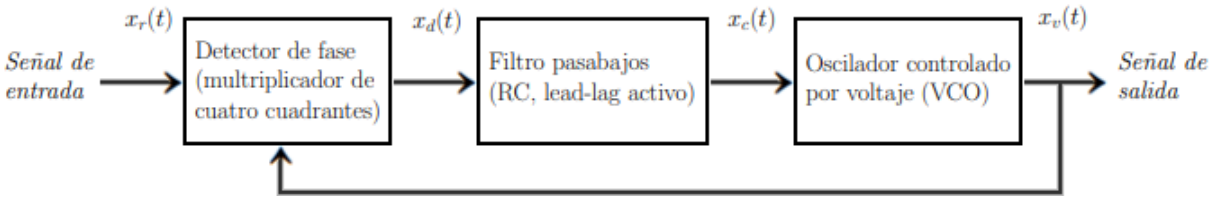

Figura 1: Esquema en bloques de un PLL básico. La flecha indica el camino que sigue la señal de entrada a través de cada componente.

[1]Email: joan.caceres@ib.edu.ar
[2]Email: franco.caviglia@ib.edu.ar

De entre los usos típicos de los multiplicadores se encuentran la amplificación y filtrado controlado por tensión, los circuitos moduladores y demoduladores, el control automático de la ganancia y el cálculo analógico.

En el procesamiento de señales analógicas, con frecuencia se requiere un circuito que toma dos entradas analógicas y produce una salida proporcional a su producto. Sin embargo es común que cuando se realiza la multiplicación de dos señales analógicas intervengan los tres elementos básicos que caracterizan dichas señales:

1. la magnitud, 2. la frecuencia, 3. la fase.

En particular, en este trabajo se medirá la fase empleando un multiplicador de cuatro cuadrantes ideal. Como las señales de entrada serán en general sinusoidales, se tiene que la respuesta la del detector será

$$\begin{aligned}
V_{(t)} &\propto \operatorname{sen}(h(t))\operatorname{sen}(h(t)+\phi(t)) \\
&= \operatorname{sen}(h(t))\left[\operatorname{sen}(h(t))\cos(\phi(t)) + \cos(h(t))\operatorname{sen}(\phi(t))\right] \\
&= \operatorname{sen}^2(h(t))\cos(\phi(t)) + \operatorname{sen}(h(t))\cos(h(t))\operatorname{sen}(\phi(t)) \\
&= \frac{1-\cos(2x)}{2}\cos(\phi(t)) + \operatorname{sen}(h(t))\cos(h(t))\operatorname{sen}(\phi(t)) \\
&= \frac{\cos(\phi(t))}{2} - \frac{\cos(2h(t))\cos(\phi(t))}{2} + \frac{\operatorname{sen}(2h(t))\operatorname{sen}(\phi(t))}{2} \\
&= \frac{\cos(\phi(t))}{2} - \frac{\cos(2h(t)+\phi(t))}{2}
\end{aligned}$$

Si la función $h(t)$ es lineal, entonces el valor medio de la salida del comparador va como la mitad del coseno del defasaje $\phi(t)$. Dicho comportamiento se observa en el **Ejercicio 5**, donde la componente continua de la señal de salida va como $0,354 \simeq \cos(\pi/4)/2$. Se observa también que si $\phi(t) = (2n+1)\pi/2$, entonces no se detecta diferencia de fase, y que si ambas señales están en fase entonces el valor medio de la salida es 1/2 (siempre que la amplitud de ambas señales de entrada es 1).

1.1.2. Filtro

Un filtro digital es un tipo de filtro que opera sobre señales discretas y cuantizadas, las cuales pueden ser analógicas o digitales, y devuelve a su salida otra señal analógica o digital de distinta amplitud, frecuencia o fase según la naturaleza del filtro. Está implementado con tecnología digital, bien como un circuito digital o como un programa informático. En general los filtros se pueden clasificar

- Según la parte del espectro sobre la cual actúan:
  - Pasa alto
  - Pasa bajo
  - Pasa banda
  - Pasa todo
  - Elimina banda
  - Multibanda
- Según su orden (primer orden, segundo orden, etc.).
- Según su respuesta ante una entrada unitaria:
  - FIR (Finite Impulse Response)
  - IIR (Infinite Impulse Response)
  - TIIR (Truncated Infinite Impulse Response)
- Según con la estructura con que se implementa:
  - Laticce
  - Varios en cascada
  - Varios en paralelo

Cada uno de estos filtros se encuentra completamente caracterizado por su función de transferencia $H(s)$, la cual toma en general valores complejos e indica en el dominio de frecuencias como afecta el filtro a la amplitud y fase de una dada onda sinusoidal.

En el circuito PLL, el elemento que le sigue al detector de fases está formado por un filtro pasa bajos, el cual se encarga de eliminar las componentes armónicas que aparecen a la salida del detector de frecuencias. La salida del filtro será entonces una tensión prácticamente continua que servirá de entrada para oscilador (VCO). En la implementación del filtro llevada a cabo se consideraron dos tipos distintos de filtros pasa bajos: un filtro RC de primer orden y un filtro tipo lead-lag activo de segundo orden.

1.1.3. Oscilador controlado por voltaje

Un oscilador controlado por voltaje (o simplemente VCO, por sus siglas en inglés) es un oscilador electrónico a cuya salida de obtiene una señal en principio sinusoidal que posee una frecuencia de oscilación que es controlada por la propia entrada del oscilador. En general la señal de salida que este oscilador va a generar se puede escribir como

$$x_v(t) = \sin(\theta(t)),$$

donde $\theta(t)$ es la fase del VCO, y dependerá en general de la señal de entrada, la cual es a su vez una función del tiempo t.

La frecuencia de esta señal será la frecuencia de una portadora, que denotaremos por f_0, con una desviación proporcional a la tensión que exista en la entrada del VCO $(x_c(t))$, es decir

$$f(t) = f_0 + Kx_c(t), \tag{1}$$

donde K es la ganancia de lazo del VCO y $x_c(t)$ es la tensión a la salida del filtro. La frecuencia f_0 se denomina también frecuencia de oscilación natural del VCO, pues es la frecuencia a la que oscila la salida en ausencia de una tensión de entrada (también llamada señal de control, $x_c(t) = 0$). Esto es lo que se observa en el **Ejercicio 13**.

En orden de recuperar $\theta(t)$, y con ello $x_v(t)$, a partir de conocer en cada instante de tiempo $x_c(t)$, se tiene la relación elemental entre frecuencia y fase

$$f(t) = \frac{1}{2\pi}\frac{d\theta(\theta)}{dt}. \tag{2}$$

Sustituyendo la Ecuación 1 e integrando se obtiene una expresión para la fase en cada instante t

$$\theta(t) = 2\pi\int^t f(u)\,du = 2\pi f_0 t + 2\pi K\int^t x_c(u)\,du,$$

donde se observa que la fase será aquella de una señal sinusoidal junto a un término extra que depende de la entrada. Dado que en la implementación del VCO cada señal de entrada se tratará de forma discretizada en el tiempo, se tiene que de la expresión integral anterior puede obtenerse

la diferencia en la fase $\theta(t)$ entre dos momentos sucesivos como

$$\begin{aligned}\theta(t+T_s) &= 2\pi f_0(t+T_s) + 2\pi K \int^{t+T_s} x_c(u)\,du \\ &= 2\pi f_0 t + 2\pi f_0 T_s \\ &\quad + 2\pi K \int^{t} x_c(u)\,du + 2\pi K \int_{t}^{t+T_s} x_c(u)\,du \\ &= \theta(t) + 2\pi f_0 T_s + 2\pi K x_c(t) T_s \\ &= \theta(t) + 2\pi f(t) T_s. \end{aligned} \tag{3}$$

Con esto se tiene una forma iterativa de calcular en cada instante de tiempo la fase a la salida del VCO conociendo una fase inicial, K, f_0 y la entrada $x_c(t)$.

1.2. Demodulador coherente

Una vez con el PLL se probó su comportamiento siendo parte de un simulador de un par emisor-receptor. Más precisamente, se lo utilizó dentro del receptor para realizar parte de la demodulación coherente de una señal digital transmitida por medio con modulación de amplitud. Un esquema en bloques del par se muestra en la Figura 3.

Una señal modulada en amplitud (ASK por sus siglas inglés Amplitude-shift keying) se encuentra compuesta por datos digitales, representados por pulsos o escalones, que sirven de modulación para una onda portadora cuya frecuencia y fase permanecen constantes. Puede pensarse como el análogo digital para la transmisión AM. Para su implementación primero se construirá la señal con un generador de código, devolverá una cantidad prefijada de pulsos temporalmente equiespaciados. Ya que está señal estará compuesta por una cantidad arbitraria de frecuencias, se la debe tratar con un filtro antes de enviarla por el canal de transmisión. Uno de los objetivos de esto es disminuir los requerimientos de ancho de banda del canal de transmisión. De los posibles filtros se utilizó un filtro de coseno alzado. Este es un filtro pasa bajos tiene la particularidad de minimizar la interferencia entre símbolos (pulsos representando 0 o 1), la cual consiste en un efecto distorsión temporal del pulso vía su expansión, y que provoca que cada uno invada los marcos de temporales de los símbolos adyacentes (la interferencia se abrevia ISI).

En la la gráfica de la Figura 2 se muestra la respuesta del filtro tipo coseno alzado a un impulso unitario en $t = 0$. Con la información provista por esta respuesta se puede primero obtener la respuesta en frecuencia realizando la transformada de Fourier a la señal, y segundo conocer la respuesta a cualquier señal de entrada haciendo la convolución. Esto último surge del llamado Teorema de la Convolución, el cual implica para la transformada de Fourier (también Laplace) que

$$\mathcal{F}[f \cdot g] \propto \mathcal{F}[f] * \mathcal{F}[g] \qquad f * g \propto \mathcal{F}^{-1}[\mathcal{F}[f] \cdot \mathcal{F}[g]], \tag{4}$$

donde la proporcionalidad indica que se requiere un factor de normalización que dependerá en general de la convención usada y el contexto. Ya que se trabaja con señales discretas, la operación de realizar la convolución entre la respuesta al impulso (cuya transformada da la función de transferencia del sistema) y la señal de entrada se transforma en

$$v{*}w\,(t) = \int_{-\infty}^{+\infty} v(u)w(t-u)\,du \Longrightarrow (v{*}w)[t] = \sum_{-\infty}^{+\infty} v[u]w[t-u]$$

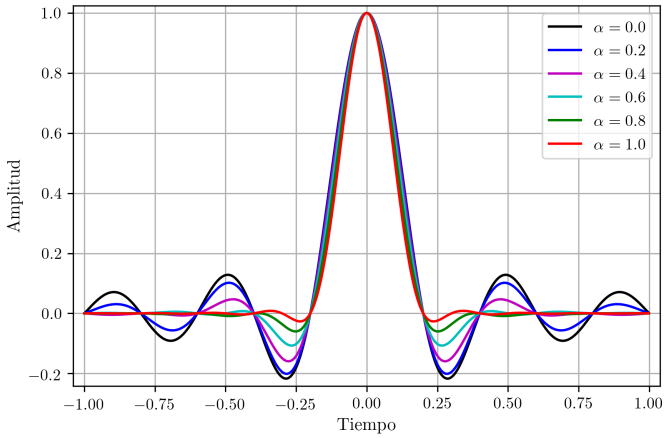

Figura 2: Respuesta al impulso para el filtro tipo coseno alzado con distintos factores de roll-off α.

Una vez con la señal creada y filtrada, se junta con la moduladora multiplicando punto a punto con dicha señal. La salida pasará entonces al canal de comunicación. Al llegar al receptor se requiere que la señal sea decodificada quitando la señal moduladora. La forma de hacer esto junto al PLL consiste primeramente en recuperar la frecuencia y fase de la moduladora. Luego, multiplicando el resultado con la señal modulada se obtiene una señal cuya portadora puede filtrarse por medio de un filtro pasa bajos.

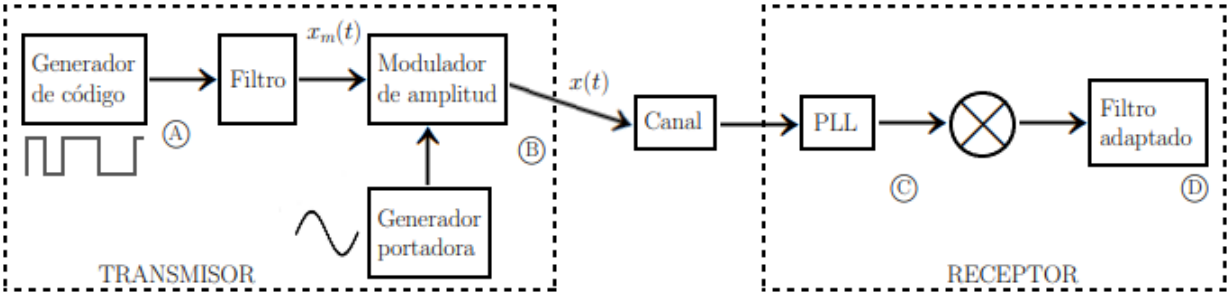

Figura 3: Esquema en bloques de un sistema de comunicaciones digital básico empleando un PLL. Dentro de cada uno de los elementos principales (transmisor, canal, receptor), se muestra el camino que sigue la señal con el código a través de la modulación y demodulación.

2. Método computacional

En la implementación del simulador se utilizaron funciones que forman parte de los paquetes NumPy[1] (para operaciones matemáticas), SciPy[2] (para el procesamiento de señales) y commPy[3] (para usar el filtro de coseno alzado). A su vez se hizo uso de los paquetes matplotlib[4] para el armado de las gráficas y Tkinter[5] para mostrar en un entorno gráfico.

Para simular el comportamiento de un PLL, se comenzó generando una señal de entrada $x_r(t)$ mediante la función `fuente`

```
def fuente(f0, Ts, fase_inicial, N):
    t = 0
    salida = []
    for i in range(N):
        salida.append([t,
                sin(2*pi*f0*t+fase_inicial)])
        t+=Ts
    return salida
```

donde f_0 representa la frecuencia de la señal, T_s el periodo de muestreo de la señal (intervalo entre cada muestra) y N la cantidad de muestras de la señal. La salida tiene la forma de una lista de pares [x,y].

Luego, se simuló el detector de señal, creando una función "detector" que hace de multiplicador de cuatro cuadrantes entre una muestra temporal de la señal de entrada $x_r(t)$ y una muestra temporal de la señal del VCO $x_v(t)$:

```
def detector(xr, xv, K=1):
    producto = []
    for i in range(len(xr)):
        producto.append([xr[i][0],K*xr[i][1]*xv[i
    ][1]])
    return producto
```

En este punto es importante notar que:

$$\sin(\omega t)\sin(\omega t+\phi) = \frac{1}{2}(\cos(\phi) - \cos(\phi + 2\omega t))$$

Por lo que uno una señal constante dada por $\cos(\phi)$ y una señal cosenoidal con una frecuencia del doble del de las señales de entrada, así que para eliminar aquella señal, se hizo uso de un filtro pasa-bajo.

En Python es posible implementar un filtro digital cuando se conoce la función de transferencia en el dominio de Laplace del filtro. Esto se realiza utilizando la función `bilinear`[6], que implementa sobre la función de transferencia $H(s)$ la transformación bilineal

$$s = \frac{2}{T_s}\frac{1-z^{-1}}{1+z^{-1}} \tag{5}$$

Los argumentos del comando bilinear pueden verse en la ayuda de Python. Esencialmente, sed deben especificar dos vectores. El primero es un vector que contiene los coeficientes de las potencias de s del numerador de Laplace. El segundo vector debe contener los coeficientes de las potencias de s del denominador de Laplace. El comando tiene un tercer argumento, que debe ser la frecuencia de muestreo que se utilice para pasar del filtro continuo en el dominio de Laplace al filtro discreto (dominio z). El comando devuelve los coeficientes a_k y b_k del correspondiente filtro digital que implementa la función de transferencia especificada.

La función de transferencia $H(s)$ es la relación entre la tensión de entrada y la tensión de salida en el espacio de las transformadas de Laplace. En el caso de un filtro RC la función de transferencia se escribe [2]

$$H(s) = \frac{V_{\text{out}}(s)}{V_{\text{in}}(S)} = \frac{V_C(s)}{V_{\text{in}}(S)} = \frac{1}{RC\,s+1} \tag{6}$$

En el caso de un filtro tipo lead-lag activo, la función de transferencia toma la forma

$$H(s) = \frac{1+R_2C\,s}{R_1C\,s} \tag{7}$$

Conociendo ambas expresiones, se definió entonces función `coef_filtro`, la cual devuelve los coeficientes según el tipo de filtro. Se incluye también, aunque no se usa, la opción para coeficientes del filtro tipo lead-lag pasivo [3].

```
def coef_filtro(tipo, C, R1, R2, Ts):
    if tipo == 'rc':
        return signal.bilinear([1], [R1*C,1], 1/
    Ts)
    elif tipo == 'lead-lag pasivo':
        return signal.bilinear([R2*C,1], [(R1+R2)
    *C,1], 1/Ts)
    elif tipo == 'lead-lag activo':
        return signal.bilinear([R2*C,1], [R1*C
    ,0], 1/Ts)
    else:
        return 0
```

Con estos coeficientes ya pueden utilizarse en el comando `lfilter` para realizar el filtrado de la señal.

La última consideración que hay que tener en cuenta es que se desea realizar un filtrado muestra a muestra de la señal por lo tanto es necesario guardar el estado final del filtro en una variable, después de filtrar cada muestra. Este estado final se especificará como estado inicial del filtro al realizar el filtrado de la siguiente muestra.

Con esta información, se definió la función `filtro`:

```
def filtro(xd, zi, tipo, C, R1, R2, Ts):
    entrada = []        # Para separar la
    variable dependiente en los pares (x,y)
    salida  = []        # Para armar la lista con
     los pares (x,y) filtrados

    for elem in xd:     # Primero construye un
    array para poder pasarlo al filtro
        entrada.append(elem[1])

    coeficientes = coef_filtro(tipo, C, R1, R2,
    Ts)

    if zi == []:        # Por si se desea filtrar
     toda una se al en una  nica  ejecuci n.
        zi = entrada[0]*signal.lfiltic(
    coeficientes[0], coeficientes[1],[0])
    else:
        pass

    for i in range(len(xd)):

```

[1] Más información disponible en www.numpy.org
[2] Más información disponible en www.scipy.org
[3] Más información disponible pypi.org/project/scikit-commpy
[4] Más información disponible en matplotlib.org
[5] Más información disponible en wiki.python.org/moin/TkInter
[6] Dentro del paquete `scypy`

```
        filtrado, zi = signal.lfilter(
    coeficientes[0], coeficientes[1], [entrada[i
    ]], zi=zi )
        salida.append([xd[i][0], filtrado[0]]) #
    Arma la lista con coordenadas (t,V)

    return salida, zi
```

donde "xd" es una lista con pares de la forma [[x,y]] que representa la señal a filtrar, "zi" es un array con la condición inicial para el filtro, el "tipo" define el tipo de filtro, "Ts" es el período de muestreo de la señal a filtrar en segundos y finalmente también recibe como argumento el valor de las resistencias y del capacitor. Esta función retorna una lista con la señal filtrada de la forma [[x,y]].

Generalmente los filtros se trabajan con los parámetros de entrada $\tau_1 \equiv R_1C$, $\tau_2 \equiv R_2C$, por lo que se definió una función que convierta estos dos parámetros en las capacitancias y resistencias y luego una función auxiliar `filtro_traducido` que usa estos dos parámetros como entradas:

```
def traductor(tipo, omega, xi):
    if tipo == 'rc':
        R = 1/sqrt(2*xi*omega)
    elif tipo == 'lead-lag activo':
        R = sqrt(2*xi*omega)
    else:
        return 0
    return R

def filtro_traducido(xd, estado_inicial, tipo,
    omega, xi, Ts):
    R = traductor(tipo, omega, xi)
    return filtro(xd, estado_inicial, tipo, R, R,
     R, Ts)
```

Luego, para simular el oscilador controlado por voltaje, se definió una función que dada la frecuencia actual ($f = f_0 + Kx_c(t)$) y la fase en el instante t, calcule la fase en el instante siguiente ($t + T_s$) según la Ec. 3:

```
def fase(Ts,f,fase_inicial):
    fase = fase_inicial + 2*pi*f*Ts  # Calcula la
     fase segun la formula (10)
    fase_final = mod(fase, 2*pi)     # Asegura
    que el valor este en (0, 2pi)

    return fase_final
```

Con esta función, se procedió a simular VCO a partir de la función `vco`:

```
def vco(v, Ts, K, f0):
    if ph == None:
        ph = 2*pi*f0*xc[0][0]
    else:
        pass

    V_vco = []

    for i in range(len(xc)):
        ph = fase(Ts, f0+K*xc[i][1], ph)
        V_vco.append( [ xc[i][0]+Ts, sin(ph) ] )

    return V_vco, ph
```

Finalmente, a partir de las funciones previamente definidas, se construyó el PLL a partir de la siguiente función:

```
def PLL(xr, tipo, xi, omega, f0, K, Ts):

    R = traductor(tipo, omega, xi)
    coeficientes = coef_filtro(tipo, R, R, R, Ts)

    xd = [[0,0]]
    xc = [[0,0]] # Antes de entrar la se al, los
     pasos intermedios son cero
    xv = [[0,0]] # Para el detector de fase
    ph = 0       # fase de la salida inicial del
    VCO, xv

    zi = xd[0][1]*signal.lfilter_zi(coeficientes
    [0], coeficientes[1])

    for i in range(len(xr)):
            # Itera sobre cada elemento
        xd.append(detector([xr[i]], [xv[-1]])
    [-1])        # 1ro. el detector de fase
        xc_, zi = filtro([xd[-1]], zi, tipo, R, R
    , R, Ts) # 2do. el filtro
        xc.append(xc_[-1])
        xv_, ph = vco([xc[-1]], Ts, K, f0, ph)
            # 3ro. el VCO
        xv.append(xv_[-1])

    return xr, xd, xc, xv
```

Donde "xr" es la señal de entrada representada con una lista con N elementos de la forma [x,y], "tipo" indica el tipo de filtro que utiliza el PLL, "xi" el coeficiente de amortiguamiento del filtro, "omega" la pulsación propia del filtro, "f0" es la frecuencia de oscilación natural del VCO, "K" la ganancia de lazo del VCO y "Ts" el período de muestreo de la señal de entrada y esto finalmente retorna "vs" que es la señal de salida del VCO.

3. Aplicaciones

3.1. Errores de fase

Considerando una señal de entrada con igual frecuencia a aquella del VCO, se introdujo un error de fase $\pi/4$ en la muestra $N/2$ y se obtuvieron para el caso de un filtro 'rc' definido con $\xi = 2{,}0$ y $\omega_n = 220$ la gráfica de errores de la Fig. 4 y el diagrama de Lissajous de la Fig. 5. Cuando se disminuye el ξ a $\sqrt{2}/2$ se obtiene una gráfica de errores similar y la curva de Lissajous es la de la Fig. 6. Con aquellas gráficas uno puede observar que el PLL no engancha con la frecuencia dada.

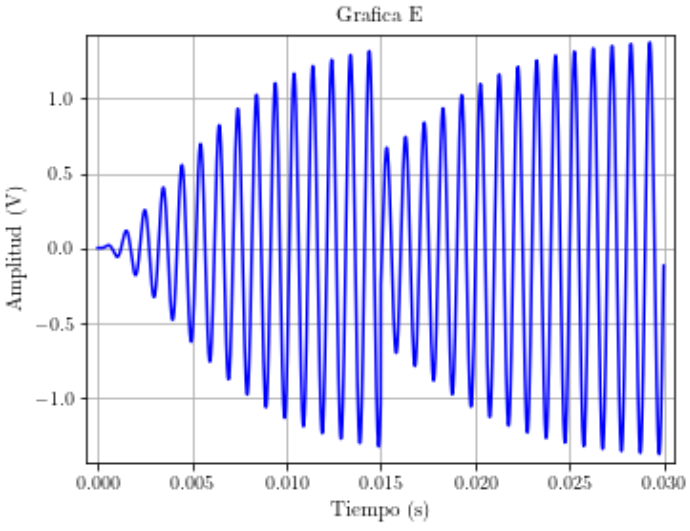

Figura 4: Errores del filtro 'rc' a $\xi = 2{,}0$ y $\omega_n = 220$.

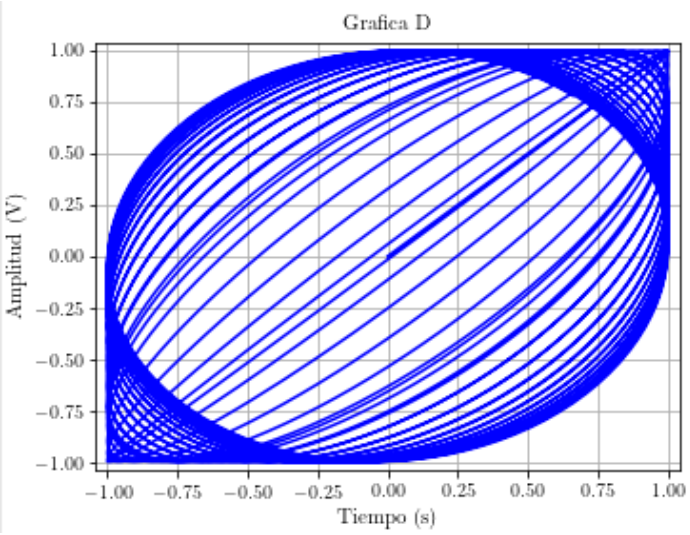

Figura 5: Curva de Lissajous del filtro 'rc' a $\xi = 2{,}0$ y $\omega_n = 220$.

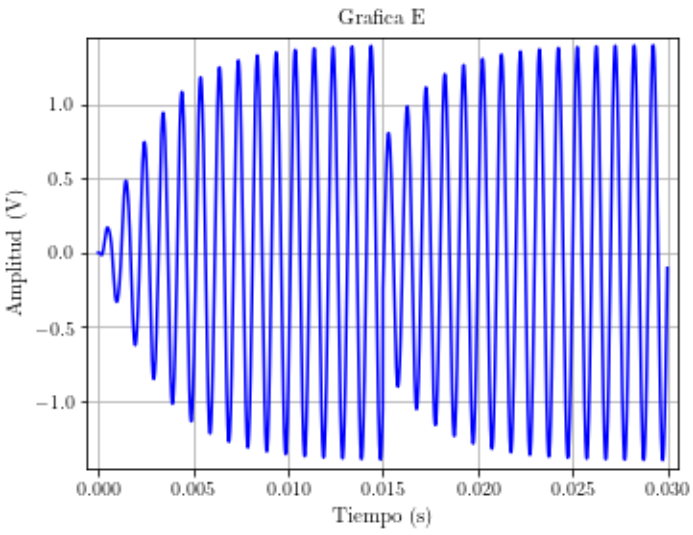

Figura 7: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$.

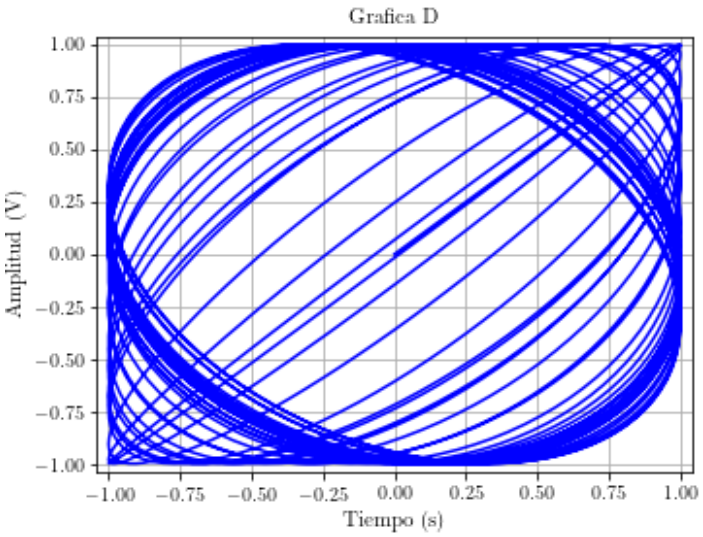

Figura 6: Curva de Lissajous del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 220$.

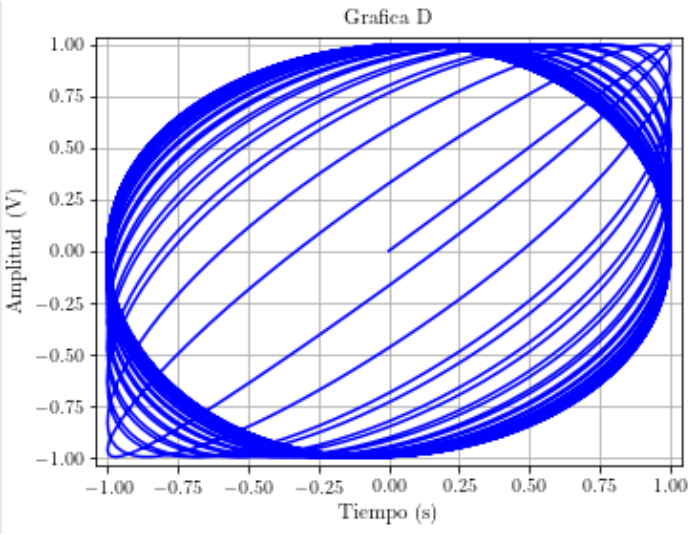

Figura 8: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$.

Luego, con el filtro 'lead-lag activo' se repitió el proceso con $\xi = \sqrt{2}/2$ y $\omega_n = 80$ y se observa la gráfica de errores de la Fig. 7 y el diagrama de Lissajous de la Fig. 8. Con esto, se concluye lo mismo que en el caso del filtro 'rc'.

3.2. Errores de frecuencia

Generando una señal a la entrada del PLL con un error de fase de $\sqrt{\pi}/2$ y un error de frecuencia del 10 % se usó un filtro 'rc' con parámetros $\xi = \sqrt{2}/2$ y $\omega_n = 200$ obteniendo la gráfica de errores de la Fig. 9 y la curva de Lissajous de la Fig. 10. En este se puede observar el PLL no engancha con la fase de la señal de entrada.

Cambiando el parámetro ω_n a 450 se obtiene la gráfica de errores de la Fig. 11 y el diagrama de Lissajous de la Fig. 12. En estas gráfica se observa que el PLL engancha con la fase de la señal de entrada llegando a un máximo de enganche hasta que luego vuelve a haber un desfasaje constante.

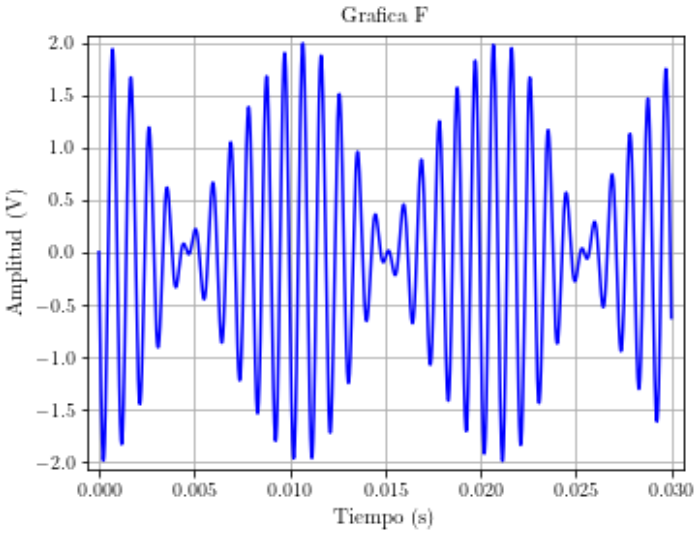

Figura 9: Errores del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$.

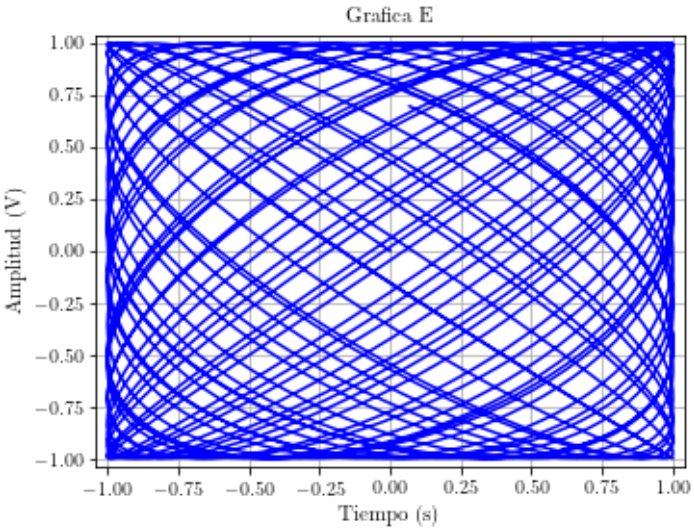

Figura 10: Curva de Lissajous del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$.

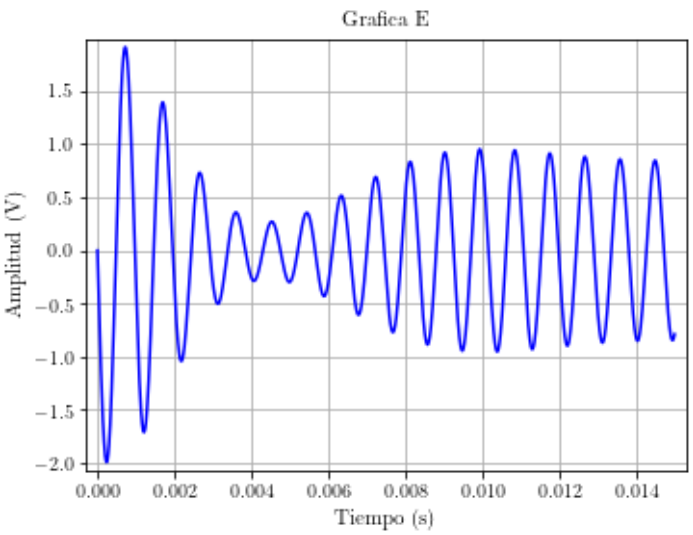

Figura 11: Errores del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 450$.

Cambiando de filtro al filtro 'lead-lad activo' con parámetros $\xi = \sqrt{2}/2$ y $\omega_n = 180$ se obtuvo la gráfica de errores de la Fig. 13 y la curva de Lissajous de la Fig. 14.

En este se observa que el PLL engancha con la fase de la señal de entrada y no vuelve a ocurrir un desenganche como en el primer ejemplo con el filtro 'rc'.

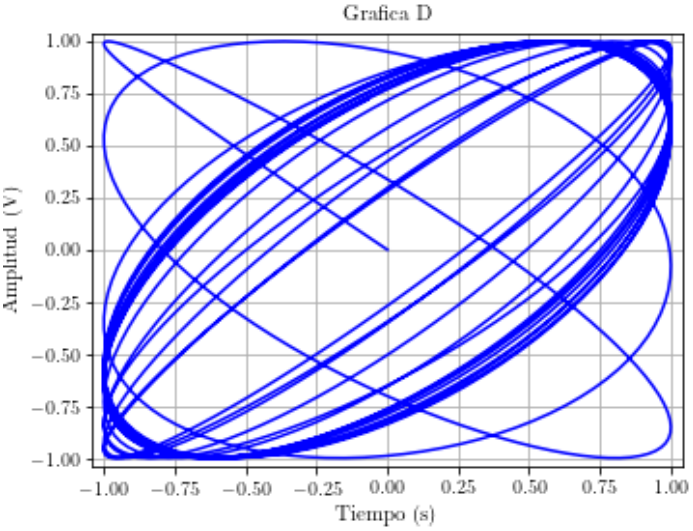

Figura 12: Curva de Lissajous del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 450$.

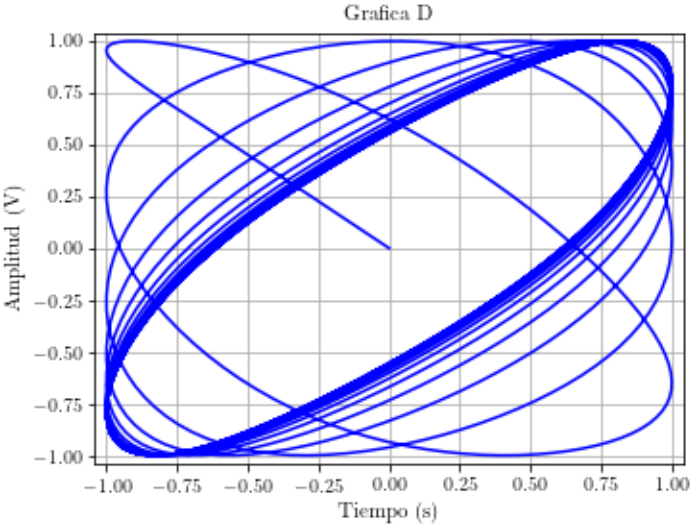

Figura 14: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/$ y $\omega_n = 180$.

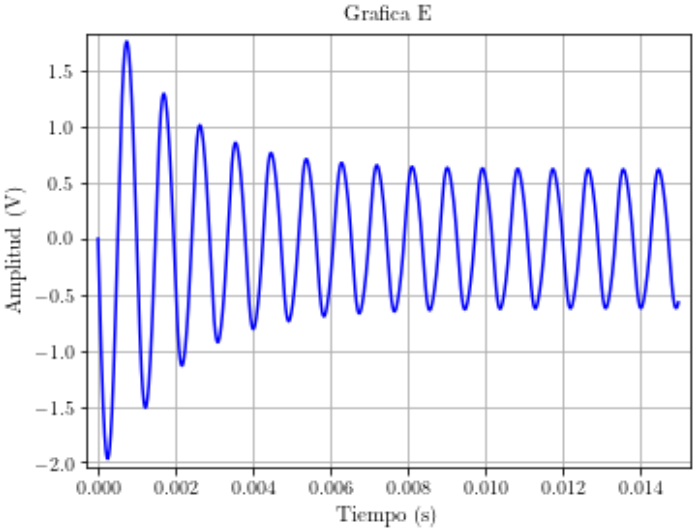

Figura 13: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 180$.

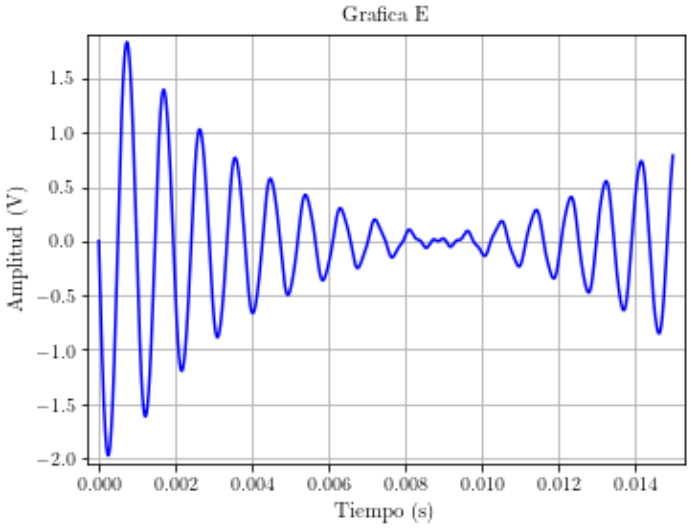

Figura 15: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 120$.

Cambiando este último filtro a $\omega_n = 120$ se obtuvo la gráfica de errores de la Fig. 15 y la curva de Lissajous de la Fig. 16. Acá se observó que nuevamente existe un periodo en el que engancha con la fase de entrada y luego vuelve a desengancharse.

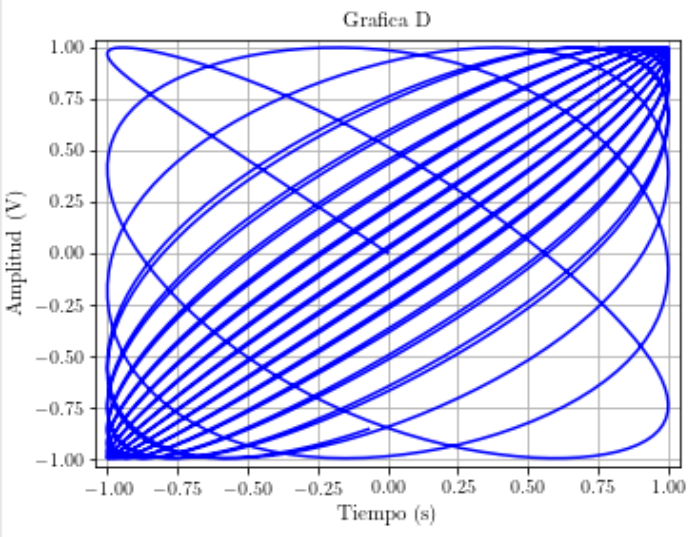

Figura 16: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 120$.

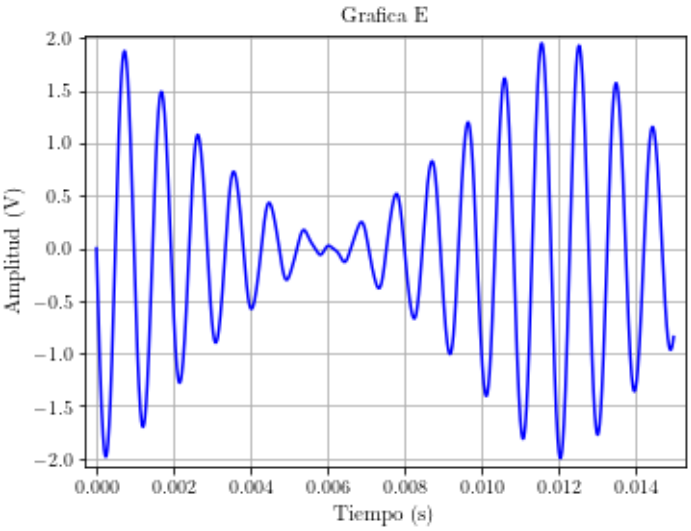

Figura 17: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$.

Finalmente, se redujo el error de frecuencia al 3 % con los mismos parámetros del ejemplo anterior y se obtuvo la gráfica de errores de la Fig. 19 y la curva de Lissajous de la Fig. 20. Con esto, se puede inferir que es el desenganche ha de ocurrir por el error inicial entre ambas frecuencias.

3.3. Rampa en frecuencia

En orden de implementar una señal de entrada que tuviera una frecuencia linealmente creciente (rampa de frecuencia), se implementó la función $\mathbf{fuente\_var(f_0, f_{final}, T_s, fase_{inicial}, N)}$, la cual tiene la forma:

Adicionalmente, con un $\omega_n = 80$ se observó la gráfica de errores de la Fig. 17 y la curva de Lissajous de la Fig. 18. Se concluye en este caso lo mismo que para el caso anterior.

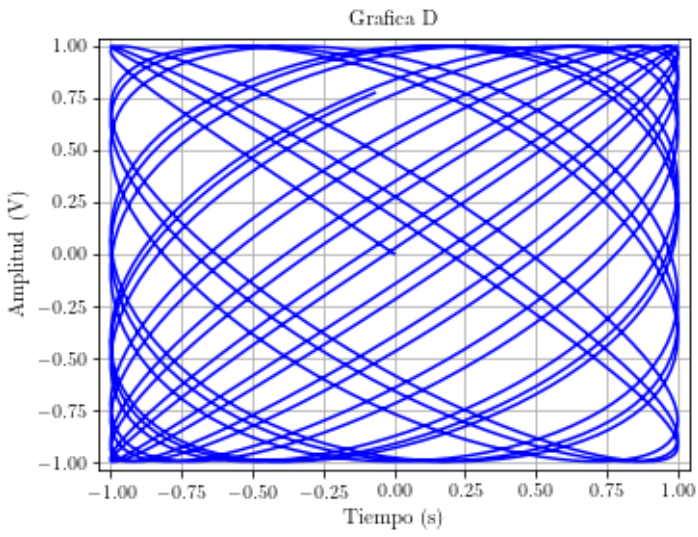

Figura 18: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$.

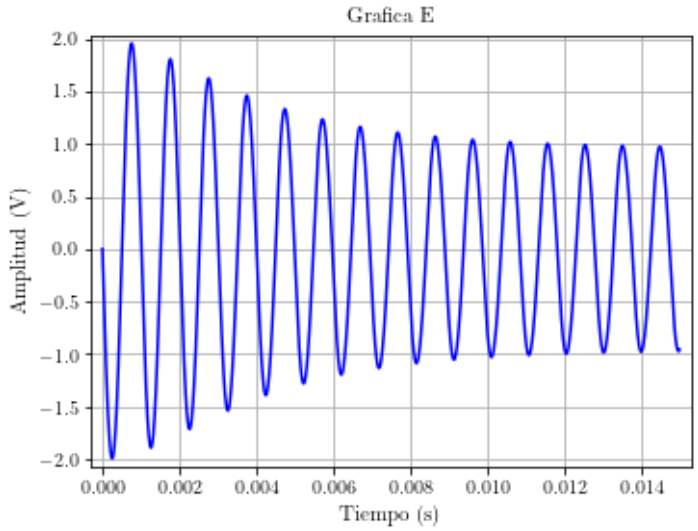

Figura 19: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 80$ con un 3 % menor de error de frecuencia.

```
def fuente_fvar(f_inicial, f_final, Ts,
    fase_inicial, N):

    salida = []
    m = (f_final-f_inicial)/(N-1)

    for i in range(N):

        f = f_inicial+m*i
        salida.append([i*Ts,sin(2*pi*f*i*Ts
                    + fase_inicial)])
        # A ade un nuevo punto a la lista.

```

```
13      return salida
```

Y con esta función se generó una señal con error de fase inicial de $\pi/2$, y una rampa en frecuencia del 10 %, tomando como frecuencia inicial: $f_0 = 1$ kHz. Luego, se pasó por el filtro 'rc' con los parámetros $\xi = \sqrt{2}/2$ y $\omega_n = 450$ obteniendo la gráfica de errores en la Fig. 21 y el diagrama de Lissajous en la Fig. 22. Como se observa, el filtro no consigue enganchar a la rampa y se presencia una envolvente ondulatoria en el gráfico de los errores.

Al pasar a un filtro 'lead-lag activo' con parámetros $\xi = \sqrt{2}/2$ y $\omega_n = 180$ se obtiene la gráfica de errores en la Fig. 23 y el diagrama de Lissajous en la Fig. 24.

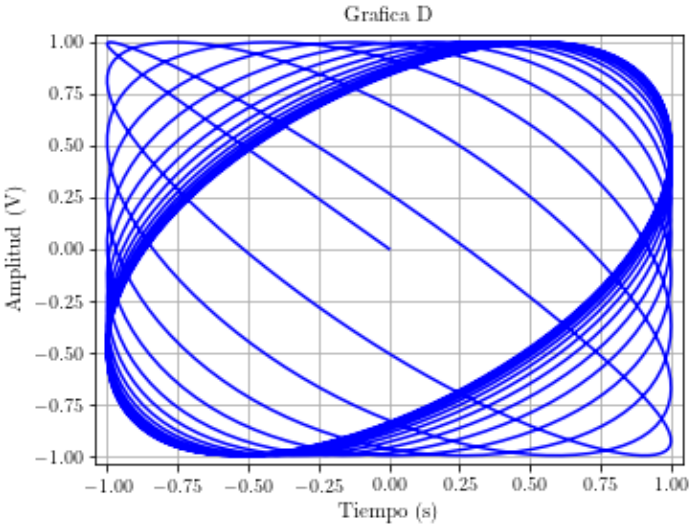

Figura 20: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/$ y $\omega_n = 80$ con un 3 % menor de error de frecuencia.

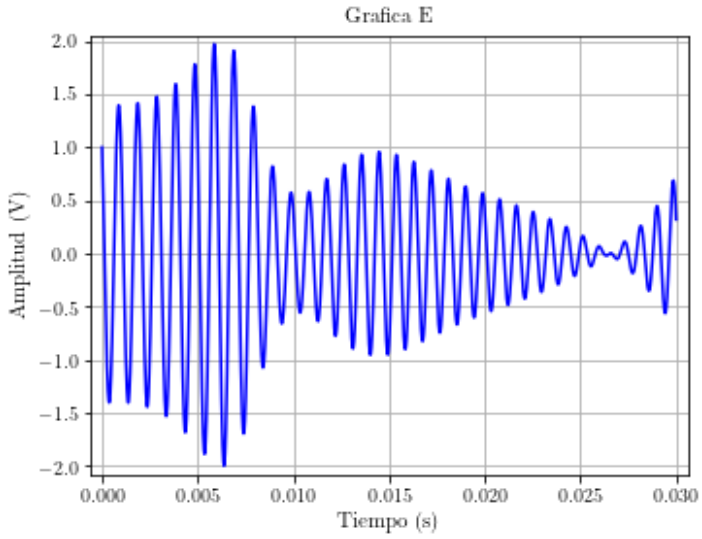

Figura 21: Errores del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 450$.

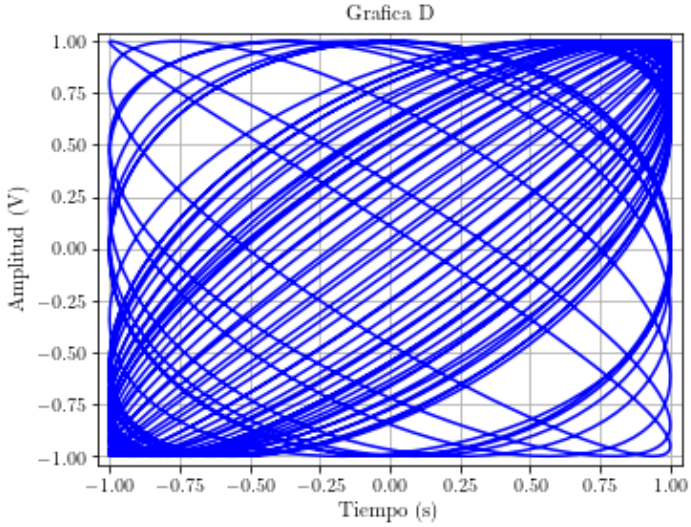

Figura 22: Curva de Lissajous del filtro 'rc' a $\xi = \sqrt{2}/2$ y $\omega_n = 450$.

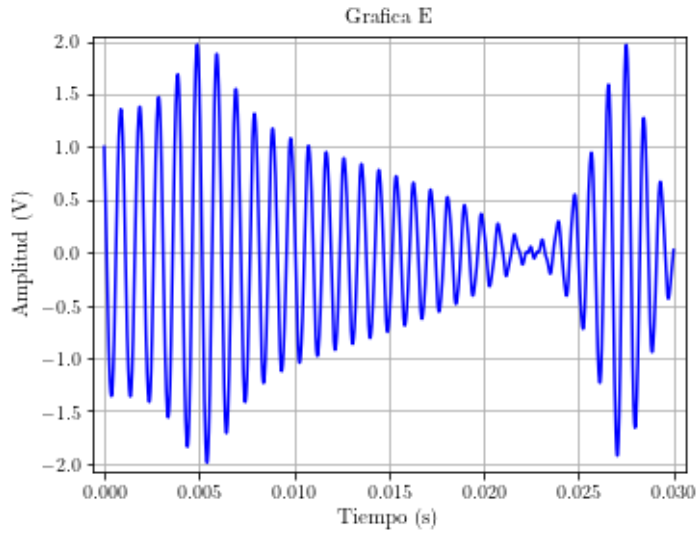

Figura 23: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 180$.

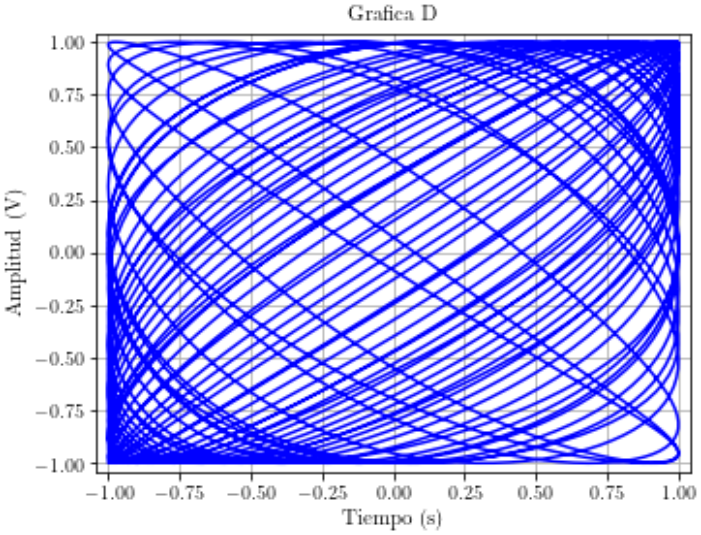

Figura 24: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 180$.

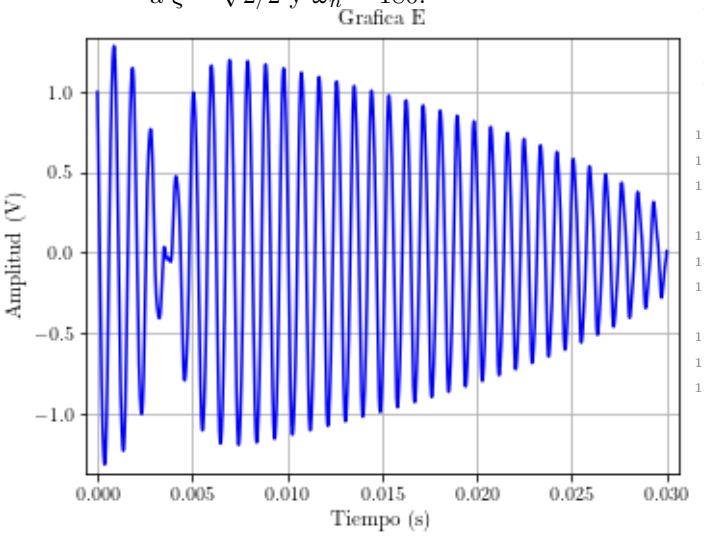

Figura 25: Errores del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 280$.

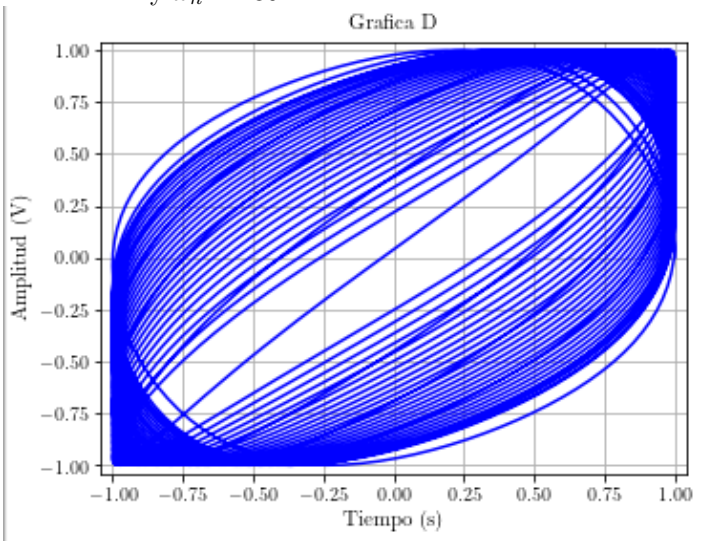

Figura 26: Curva de Lissajous del filtro 'lead-lag activo' a $\xi = \sqrt{2}/2$ y $\omega_n = 280$.

Finalmente, cuando a este mismo filtro, se le modifica el ω a $\omega = 280$ se obtiene la gráfica de errores de la Fig. 25 y la curva de Lissajous de la Fig. 26. Como se aprecia, con estos parámetros el filtro engancha con la fase de entrada.

3.4. Demodulación Coherente de una Señal

Para realizar una demodulación coherente, se tomó en consideración el esquema de la Fig. 3. Entonces para ello se procedió primero en definir una función generadora de códigos `gen_cod` que tenga como argumentos la cantidad de símbolos N_d, el período de muestreo de la señal de salida T_s y el período entre símbolos de la salida T_b y que retorne una señal de salida representada con una lista con $N = N_d T_b/T_s$ elementos de la forma [x,y].

```
def gen_cod(Nd, Ts, Tb):
    salida = []
    step = int(Tb/Ts)

    for j in range(0, int(step/2)):
            salida.append([j*Ts, 0])

    for i in range(0, Nd):
        salida.append( [ (i*step+int(step/2))*Ts,
     round(rand()) ] )

        if i<(Nd-1):
            for j in range(i*step+int(step/2)+1,
    (i+1)*step+int(step/2)):
                salida.append([j*Ts, 0])
        if i==(Nd-1):
            for j in range(i*step+int(step/2)+1,
    (i+1)*step):
                salida.append([j*Ts, 0])

    return salida
```

Luego, para minimizar la interferencia de símbolos se hizo uso de un filtro en coseno alzado, para ello, se definió la función `respuesta_filtro` con argumentos la longitud de la respuesta en cantidad de muestras (N), el parámetro del filtro de coseno alzado (a) comprendido entre 0 y 1, el período entre símbolos de la señal de salida (T_b) y la frecuencia de muestreo del filtro (F_s). Esta función hará uso del comando de Python `rcosfilter`. Esta función retorna una señal de salida representada con una lista con N elementos de la forma [x,y].

```
def respuesta_filtro(N, a, Tb, Fs):
    lista = rcosfilter(N, a, Tb, Fs)

    respuesta = []

    for i in range(N):
        respuesta.append([lista[0][i], lista[1][i
    ]])

    return respuesta
```

Con estas funciones definidas, se procedió a simular el transmisor como se muestra en la Fig. 3. Para ello se generaron un máximo de 10 pulsos como se muestra en la Fig. 27 y se realizó una convolución de estos pulsos con el filtro en coseno alzado con el comando `convolve` de Python, esto resulta lo observado en la Fig. 28. Luego, se generó una señal moduladora la cual genera la señal de radiofrecuencia (f_0) modulada en amplitud para ser transmitida por el canal, este resultado genera una señal plana como se observa en la Fig. 29.

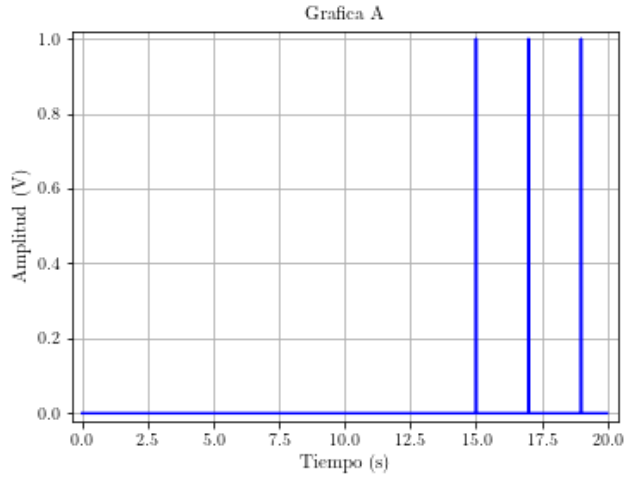

Figura 27: Gráfica de la convolución de los pulsos generado con el filtro en coseno alzado.

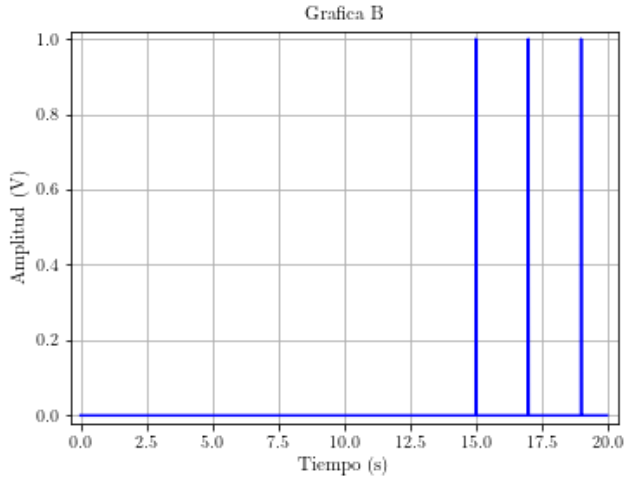

Figura 28: Salida de la señal emisora con la señal filtrada y modulada.

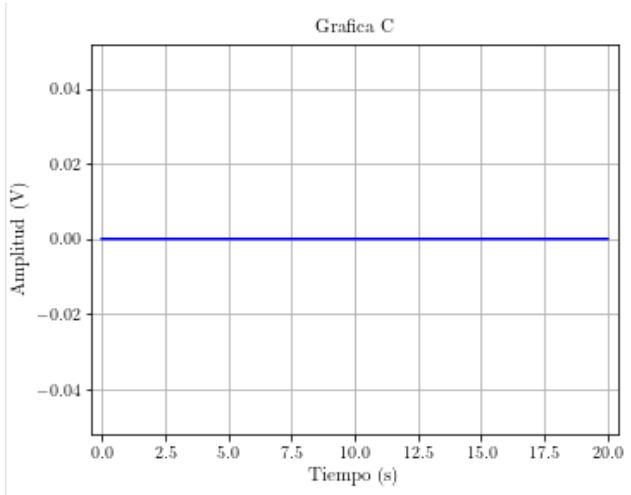

Figura 29: Salida del filtro 'rc' del PLL del receptor.

Luego de esto, pasando a la parte del receptor, se pasó esta señal que sale del transmisor por un filtro PLL y se multiplicó el resultado por la señal de salida del transmisor, como se explicó en la sección 1.2 y finalmente se hizo una convolución de esta señal que resulta con un filtro en coseno alzado. Esto, producto de la señal plana anterior, resulta también en una señal plana.

4. Conclusiones

Se simuló un circuito PLL con el lenguaje de programación Python 3.8.4 detallando el comportamiento de cada una de sus partes. Asimismo, se aplicó este PLL ante señales de entrada con errores de fase, errores de frecuencia y también con una rapa de frecuencia como entrada notando como con ciertos parámetros del PLL este engancha con la señal de entrada y con otros no. Finalmente se simuló el comportamiento del PLL al usarse para la demodulación coherente.

Referencias

[1] F. M. Gardner, *Phaselock Techniques*, 3.ª ed. New Jersey: John Wiley y Sons, Inc., 2005.

[2] E. Córdoba, «Transformada de Laplace aplicada en filtros analógicos», 2015, disponible en lcr.uns.edu.ar/fvc/NotasDeAplicacion/FVC-Emanuel%20Cordoba.pdf.

[3] J. H. Saunders, «Place 2.0-An Interactive Program for PLL Analysis and Design», *ATT Technical Journal*, n.º 64(5), 1985, doi:10.1002/j.1538-7305.1985.tb00458.x.