E.T.S. de Ingeniería Industrial,
Informática y de Telecomunicación

Estudio, comparación e implementación de algoritmos PLL. Nuevo algoritmo PLL robusto ante perturbaciones en la red.

Máster Universitario en
Ingeniería Industrial

Trabajo Fin de Máster

Asier Irigoyen Indave

David Arricibita de Andrés

Ernesto Barrios Rípodas

Pamplona, 29/06/2016

upna Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

RESUMEN

En los últimos años, debido al incremento de sistemas electrónicos de potencia que trabajan conectados a la red eléctrica debido a la implantación de las energías renovables, ha habido un interés creciente en el estudio de diferentes topologías de PLL aplicables a sistemas trifásicos.

La correcta sincronización con la red es un aspecto importante para el adecuado funcionamiento del control de convertidores conectados a la red eléctrica. Esta tarea normalmente es realizada por el PLL (Phase Locked Loop). El objetivo de los PLL es la sincronización exacta del control del convertido, con la fase de la secuencia positiva de la armónica fundamental de la tensión de red. Normalmente el PLL es un algoritmo realimentado por la frecuencia que, a partir de la tensión trifásica de la red y el ángulo obtenido a partir del propio algoritmo, realiza la transformación de Park y trata de hacer 0 una de las dos componentes (normalmente la q) para asegurar que el ángulo obtenido es el correcto.

La red puede presentar alteraciones que provoquen que el ángulo generado por el PLL no sea el adecuado. Estos errores pueden ser debidos a:

- Presencia de componente continua en la red.
- Escalones en la amplitud de tensión o frecuencia.
- Presencia de armónicos en la red.
- Desequilibrios en la red.
- Derivas en frecuencia.

Cuando una de estas alteraciones o varias está presentes en la red el PLL obtiene un ángulo retorcido o desfasado que no es el que nos interesa.

En el proyecto se determinará por qué estas alteraciones en la red hacen que el PLL no cumpla su función correctamente, se hará un análisis de las soluciones que actualmente se están utilizando con sus ventajas e inconvenientes, se propondrá una nueva y se programará en ARDUINO para hacer un análisis real y compararla con las otras 2 que mejores resultados obtienen hoy en día.

PALABRAS CLAVE

- PLL
- Arduino
- Filtro digital
- Perturbación en la red
- PSIM

ÍNDCE

1 INTRODUCCIÓN

La correcta sincronización con la red es un aspecto importante para el adecuado funcionamiento del control de convertidores conectados a la red eléctrica. Esta tarea normalmente es realizada por el PLL (Phase Locked Loop). El objetivo de los PLL es la sincronización exacta del control del convertido, con la fase de la secuencia positiva de la armónica fundamental de la tensión de red. Normalmente el PLL es un algoritmo realimentado por la frecuencia que, a partir de la tensión trifásica de la red y el ángulo obtenido a partir del propio algoritmo, realiza la transformación de Park y trata de hacer 0 una de las dos componentes (normalmente la q) para asegurar que el ángulo obtenido es el correcto.

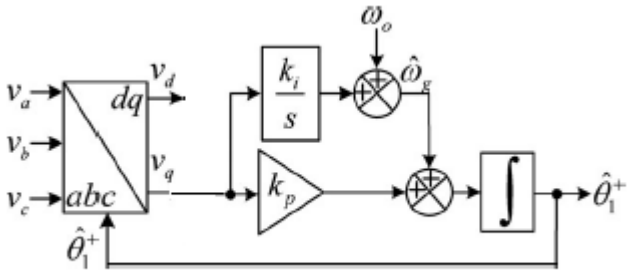


Figura 1.1

La red puede presentar alteraciones que provoquen que el ángulo generado por el PLL no sea el adecuado. Estos errores pueden ser debidos a:

- Presencia de componente continua en la red.
- Escalones en la amplitud de tensión o frecuencia.
- Presencia de armónicos en la red.
- Desequilibrios en la red.
- Derivas en frecuencia.

Cuando una de estas alteraciones o varias está presentes en la red el PLL obtiene un ángulo retorcido o desfasado que no es el que nos interesa.

En el proyecto se determinará por qué estas alteraciones en la red hacen que el PLL no cumpla su función correctamente, se hará un análisis de las soluciones que actualmente se están utilizando con sus ventajas e inconvenientes, se propondrá una nueva y se programará en ARDUINO para hacer un análisis real y compararla con las otras 2 que mejores resultados obtienen hoy en día.

En primer lugar se hará una introducción a los PLL explicando el funcionamiento del PLL más sencillo, el algoritmo que se utiliza y cómo éste es capaz de sincronizarse con la red.

Posteriormente se analizarán los problemas que puede presentar la red y que hacen que este algoritmo por sí sólo no sea adecuado para aplicaciones reales. A continuación se analizarán las soluciones adoptadas hoy en día y se compararán en función de unos parámetros como la velocidad de sincronismo ante variaciones de frecuencia.

Tras observar los problemas que presentan los PLL, hacer un análisis de los filtros digitales y analizar las soluciones adoptadas se propondrá una nueva solución.

Se implementará ésta y las dos mejores de las analizadas anteriormente en el programa de simulación P-SIM.

Finalmente se programarán los 3 PLL en ARDUINO, se explicarán las funciones utilizadas y se hará un breve resumen de las características del ARDUINO utilizado. Se analizará el funcionamiento de cada PLL primero alimentando con una fuente de tensión y posteriormente con una máquina trifásica se realizará un ensayo de potencia.

Se concluirá con un análisis del comportamiento que presenta cada una de las 3 soluciones ante diferentes ensayos y se valorará objetivamente cuál es el mejor PLL.

2 ANTECEDENTES

En los últimos años, debido al incremento de sistemas electrónicos de potencia que trabajan conectados a la red eléctrica debido a la implantación de las energías renovables, ha habido un interés creciente en el estudio de diferentes topologías de PLL aplicables a sistemas trifásicos.

La detección de la tensión de secuencia positiva es una cuestión crucial en el control de convertidores electrónicos de potencia conectados a la red eléctrica. Esta detección resulta imprescindible en el control de sistemas como acondicionadores activos, compensadores estáticos de potencia reactiva, sistemas de alimentación ininterrumpida, sistemas flexibles de transmisión en corriente alterna (FACTS: Flexible AC Transmission Systems), o sistemas emergentes de generación distribuida. Generalmente, la información obtenida acerca de la componente de secuencia positiva de la tensión de red se utiliza para sincronizar ésta con las variables de salida de los convertidores, para calcular el flujo de potencia activa y reactiva, o para expresar diferentes variables internas del sistema de control sobre ejes de referencia síncronos. En los últimos años se han presentado diferentes enfoques para la detección del módulo y la fase de la componente de secuencia positiva de la tensión de red. Con independencia de la técnica utilizada, y a partir de la lectura instantánea de las tensiones de red, el objetivo del sistema de detección es obtener, con el menor error y la mayor rapidez posibles, las magnitudes características de la componente de frecuencia fundamental de secuencia positiva, incluso cuando las tensiones de la red se encuentren distorsionadas o desequilibradas. Cuando las tensiones de red se caractericen por describir un conjunto de sinusoides perfectas y equilibradas, la obtención del módulo y la fase de las mismas se puede realizar mediante el uso de detectores de valor de pico y de paso por cero. Sin embargo, teniendo en cuenta que ambas magnitudes se detectan sólo una vez cada medio ciclo de la tensión de red, el seguimiento de las mismas durante instantes intermedios resulta imposible, con lo que siempre existirá un retardo en la respuesta del sistema. Además, la aparición de armónicos o desequilibrios en las tensiones de red falseará la detección realizada mediante este procedimiento, impidiendo su utilización en la mayoría de las aplicaciones anteriormente mencionadas.

En determinados enfoques se supone que la frecuencia de la red es una magnitud constante y conocida. Bajo este supuesto, resulta común la utilización del método de las componentes simétricas instantáneas. Otro enfoque menos común consiste en la utilización de un algoritmo de determinación de coeficientes mediante métodos numéricos iterativos. En ambos métodos

se introduce un desfase constante en las tensiones sensadas para así poder determinar la magnitud de las componentes de secuencia positiva y negativa. Este desfase es función de la frecuencia prevista en la red. Ambos métodos de detección darán lugar a errores de régimen permanente cuando la frecuencia de red experimente variaciones respecto a su valor nominal. Existen otros métodos que, mediante el uso de sistemas adaptativos, más o menos complejos, intentan solventar este problema, aunque su dinámica de adaptación es generalmente lenta. Las variaciones de frecuencia se deben principalmente a desviaciones transitorias entre la capacidad de generación y la carga, y suelen estar originadas por la maniobra de interruptores de interconexión de la red. La norma EN-50160 especifica que, durante un 5% de la semana, estas variaciones pueden llegar a ser de hasta un -6% para sistemas interconectados, y de hasta un ±15% en sistemas aislados. Cuando se supone que la frecuencia de las tensiones de red puede experimentar variaciones, es habitual usar un lazo de enganche de fase (PLL – Phase Locked Loop) dentro del sistema de detección. En los últimos años, debido al incremento de sistemas electrónicos de potencia que trabajan conectados a la red eléctrica, ha habido un interés creciente en el estudio de diferentes topologías de PLL aplicables a sistemas trifásicos. Generalmente, estas estructuras de PLL se apoyan en un sistema de referencia síncrono (SRF-PLL – Synchronous Reference Frame PLL) sobre el cual se proyecta el vector de tensión de red. En estos sistemas, mediante un controlador, se modifica la posición angular del SRF para que la proyección del vector de tensión sobre el eje en cuadratura (q) del SRF sea nula. De esta manera, se conseguirá que la proyección del vector sobre el eje directo (d) del SRF coincida con el módulo del mismo, y la posición angular del SRF coincidirá con el ángulo de fase del vector de tensión.

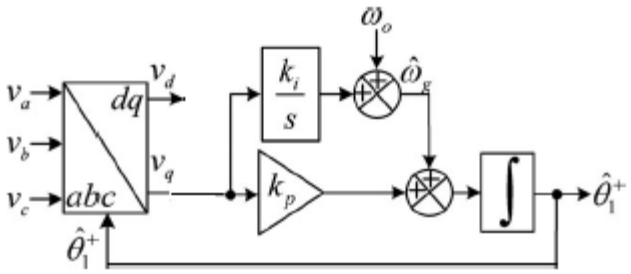


Figura 2.1

El algoritmo PLL se alimenta con una frecuencia de 50Hz: ώ₀=2*pi*50. De esta forma se inicializa el algoritmo PLL con la frecuencia de red o con un valor cercano a ella. Al integrar la frecuencia se obtiene el ángulo y con éste se realiza la transformada a dq. El valor de la tensión en el eje de cuadratura (q) se hace pasar por un PI que hará que Vq tienda a 0 modificando la frecuencia y por tanto el ángulo. Cuando Vq sea 0 la frecuencia se mantendrá estable y será igual que la de la red.

En las siguientes figuras se muestra el procedimiento del PLL paso a paso suponiendo que la frecuencia de la red son 50Hz y el PLL se inicializa con esa misma frecuencia:

En primero lugar el vector tensión de red está girando a una frecuencia de 50Hz y el eje de coordenadas dq comienza a girar con una frecuencia w también de 50Hz ya que se ha inicializado con este valor. Al realizar la transformada y obtener un valor de Vq que no es 0, como se puede ver en la figura 2.2, la frecuencia del PLL aumentará para que el eje d se aproxime al vector tensión de red.

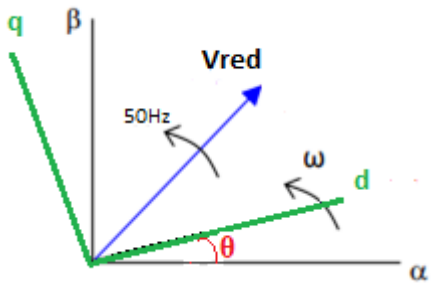


Figura 2.2

Al cabo de un tiempo el eje de coordenadas dq que gira a una frecuencia algo mayor que la de la red se aproximará al vector tensión de red (figurtra 2.3). Conforme se acerca Vq se va haciendo 0 y la frecuencia a la que gira el eje de coordenadas dq va disminuyendo hasta acercarse de nuevo a 50Hz.

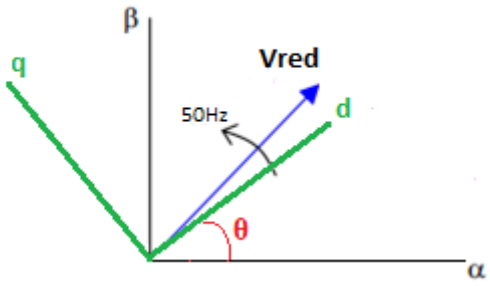


Figura 2.3

Una vez que el eje d se ha alineado con el vector tensión de red Vq se hace 0 y el PI hace que se mantenga la frecuencia del eje coordenado dq igual a la frecuencia de la red (50Hz) para que Vq siga siendo 0. Así se obtiene un eje coordenado dq girando a la frecuencia de la red en el que el ángulo **θ** obtenido es en todo momento el ángulo del vector tensión de red (figura 2.4).

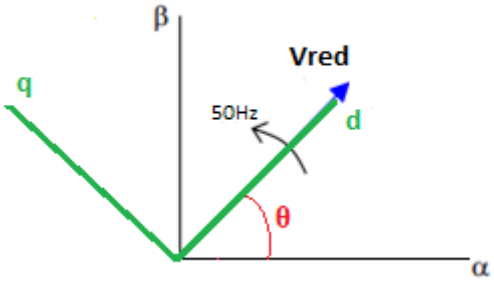


Figura 2.4

2.1 Problema

En condiciones ideales de red, sin desequilibrios ni armónicos de tensión, esta técnica simple conduce a buenos resultados. Cuando la red presenta armónicos de tensión de orden elevado, se puede reducir el ancho de banda del SRF-PLL para atenuar el efecto de estos armónicos sobre el módulo y la fase detectadas. Sin embargo, cuando en la red existen desequilibrios en las tensiones de frecuencia fundamental y cuando hay presencia de componente continua, la reducción del ancho de banda no es una solución aceptable, ya que la respuesta dinámica del PLL será muy pobre, y siempre existirá un error residual en régimen permanente en las magnitudes detectadas. Por tanto, en aquellas aplicaciones en las que se necesite un sistema de detección preciso y con un buen comportamiento dinámico, aun cuando existan desequilibrios a frecuencia fundamental, la utilización del SRF-PLL convencional no es la solución más apropiada.

A continuación se exponen los problemas más comunes que puede presentar la red y que hacen que la solución del PLL convencional no pueda generar el ángulo adecuado:

- **Presencia de componente continua en la red.** Provoca oscilaciones en la fase y frecuencia estimada por el PLL difíciles de eliminar por su baja frecuencia. Debido a esto puede ocurrir que los convertidores inyecten corriente continua a la red debido al mal funcionamiento del PLL y la normativa es estricta en este aspecto. Por ejemplo, la norma IEC 61727 limita la aplicación de corriente continua por los inversores fotovoltaicos conectados a la red a menos del 1% de su corriente nominal de salida
  Motivos de presencia de corriente continua:
  - Fallos en la red.
  - Fallos en el sistema de medición.
  - Fallos en la conversión A/D.
  - Inyección de corriente continua por los sistemas de generación.
  - Fenómenos geomagnéticos.

  La componente continua (0Hz) se ve en el sistema de referencia síncrono dq como una componente a la frecuencia de 50Hz (en el caso de una red europea sin derivas en frecuencia). Esto hace que el ángulo obtenido por el PLL no sea el adecuado.

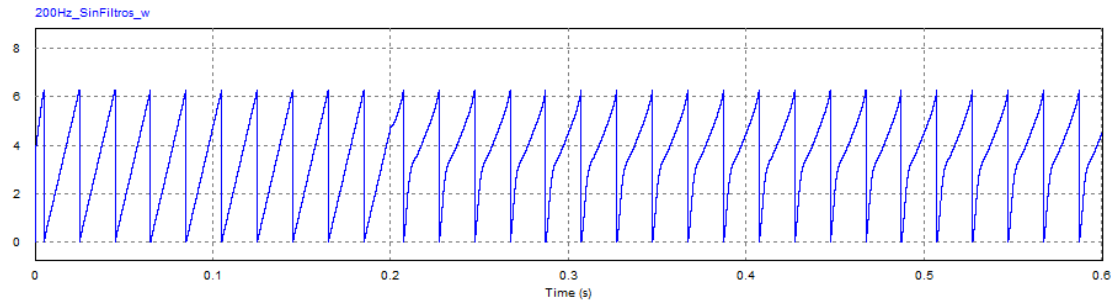


Figura 2.5

En la figura 2.5 se introduce un escalón de corriente continua por la fase R a los 0.2 segundos y se observa cómo el ángulo obtenido por el PLL se deforma.

Para atenuar el efecto de la componente continua habría que reducir el ancho de banda del PLL. En la figura 2.5 está realizado con un ancho de banda de 200Hz. Si se reduce el ancho de banda a 10Hz (figura 2.6) la respuesta es mejor pero sigue sin ser adecuado. Habría que reducir todavía más el ancho de banda pero la dinámica de éste sería muy pobre.

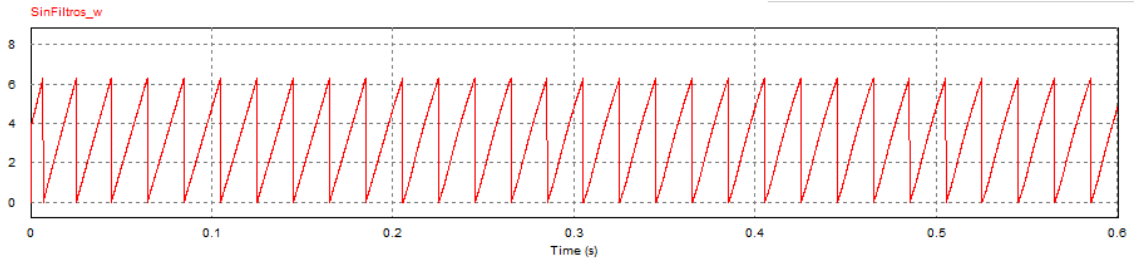


Figura 2.6

- **Escalones en la amplitud de tensión.** Un escalón en la amplitud de la tensión puede provocar que ésta se salga de los límites asumibles por el algoritmo PLL y no sea capaz de generar el ángulo adecuado. Esto se soluciona fácilmente incluyendo un mecanismo de normalización de amplitud (ANM) para ser insensible a las variaciones de amplitud del voltaje. Existen diferentes soluciones con el mismo resultado. En las figuras 2.7 y 2.8 se observan dos típicas soluciones. En el proyecto se adoptó la primera.

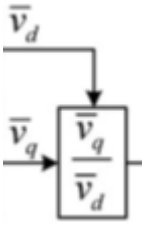


Figura 2.7

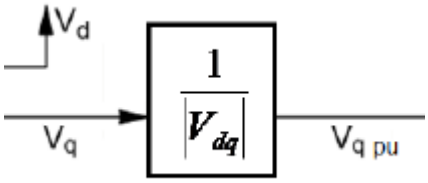


Figura 2.8

- **Presencia de armónicos en la red**. Cuando la red presenta armónicos de tensión de orden elevado, se puede reducir el ancho de banda del SRF-PLL para atenuar el efecto de estos armónicos sobre el módulo y la fase detectadas. Sin embargo, cuando en la red existen desequilibrios en las tensiones de frecuencia fundamental, la reducción del ancho de banda no es una solución aceptable, ya que la respuesta dinámica del PLL será muy pobre, y siempre existirá un error residual en régimen permanente en las magnitudes detectadas.
- **Desequilibrios en la red**. Los desequilibrios en la red se presentan como una secuencia inversa a la frecuencia de la red. Lo que nos interesa es conocer el ángulo de la secuencia directa. En el sistema dq aparecen como secuencia inversa al doble de la frecuencia de la red (100Hz). Esto provoca que el PLL no sea capaz de generar el ángulo adecuado y para ello habría que reducir el ancho de banda empobreciendo la respuesta dinámica del PLL.

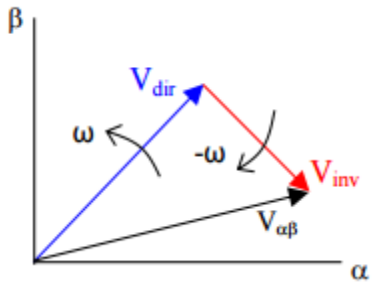


Figura 2.9

El objetivo es aislar las secuencias directa e inversa para quedarnos sólo con la directa ya que la inversa es la que proporciona el desequilibro.
Durante desequilibrios en la tensión de red la aparición de la secuencia inversa dificulta la obtención del ángulo y la frecuencia de dicha tensión. Mediante el uso de filtros y la transformada de Park se pueden aislar la secuencia directa y la inversa. Alimentando a la PLL únicamente con la secuencia directa es posible utilizar el control habitual también durante los desequilibrios.

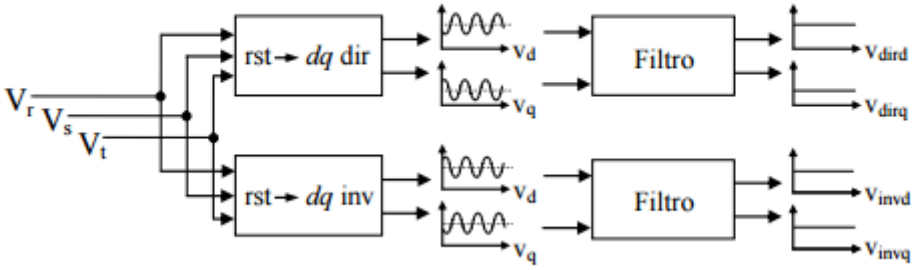


Figura 2.10

- **Derivas en frecuencia.** Las derivas en frecuencia no afectan al PLL pero sí a las soluciones que se incluyen para eliminar los problemas anteriores ya que estas incluyen filtros que se diseñan para una determinada frecuencia de corte. Cuando se produce una deriva en frecuencia de la red estos filtros pueden introducir un desfase. Para solucionarlo se puede realimentar el filtro con la frecuencia y recalcular los coeficientes como se verá más adelante.

3 ESTADO DEL ARTE

En primer lugar se estudian los tipos de filtros digitales que se pueden implementar en la actualidad y a continuación se detallan las soluciones PLL utilizadas actualmente para hacer frente a los problemas analizados en los apartados anteriores.

3.1 Filtros digitales

Un filtro digital es un tipo de filtro que opera sobre señales discretas y cuantizadas, implementado con tecnología digital, bien como un circuito digital o como un programa informático. Existen dos tipos de filtros digitales: filtros FIR y filtros IIR.

Un filtro FIR es aquel que tiene una respuesta finita al impulso y que se caracterizan por ser sistemas no recursivos.
Algunas de las características de este tipo de filtros son las siguientes:

- Un filtro FIR puede ser diseñado para tener fase lineal.
- Siempre son estables porque son hechos únicamente con ceros en el plano complejo.
- Los errores por desbordamiento no son problemáticos porque la suma de productos en un filtro FIR es desempeñada por un conjunto finito de datos.
- Un filtro FIR es fácil de comprender e implementar.
- La salida siempre es una combinación lineal de los valores presentes y pasados de la señal de entrada.
- Tiene memoria finita.

Un filtro IIR es aquel que tiene una respuesta infinita al impulso y que se caracterizan por tener una retroalimentación de la señal de salida.
En los filtros IIR, la salida es función no sólo de la entrada actual y de las precedentes, sino también de las salidas anteriores. Es decir, se trata de filtros recursivos (poseen realimentación), y por tanto se espera que (en general) posean una respuesta impulsional infinita.

Los filtros IIR son usualmente mucho más eficientes que los FIR, en términos de conseguir ciertas características de calidad para un mismo orden de filtro dado.

	IIR Filters	FIR Filters
Phase (grp delay)	difficult to control, no particular techniques available	linear phase always possible
Stability	can be unstable, can have limit cycles	always stable, no limit cycles
Order	less	more
History	derived from analog filters	no analog history
Others		polyphase implementation possible can always be made causal

Figura 3.1

Los filtros IIR requieren menos memoria y menos instrucciones para implementar su función de transferencia.

Un filtro IIR se diseña mediante el cálculo de polos y ceros en el plano complejo. El uso de polos confieren a un filtro IIR la capacidad de implementar funciones de transferencia que es imposible de realizar mediante filtros FIR.

Elegimos los filtros IIR porque nos interesa el menor orden posible para simplificar el coste computacional con la mayor calidad posible.

Existen diferentes tipos de filtros IIR:

Filtros Butterworth

La aproximación de Butterworth se obtiene al imponer que la respuesta en magnitud del filtro sea máximamente plana en la banda pasante y en la banda no pasante.

$$|H(\Omega)|^2 = \frac{1}{1+(\Omega/\Omega_c)^{2N}}$$

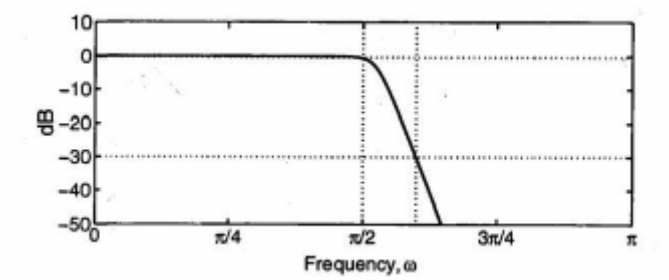


Figura 3.2 Referencia [2]

Presentan una respuesta en frecuencia monótona decreciente y una caída suave en frecuencias bajas.
A mayor orden más rápido cae la banda. En las siguientes figuras se diseñan los filtros con el programa MATLAB (filtros pasobajo con frecuencia de muestreo de 20KHz y frecuencia de corte de 60Hz) y se comprueba su diagrama de Bode para observar sus características:

Orden 2:

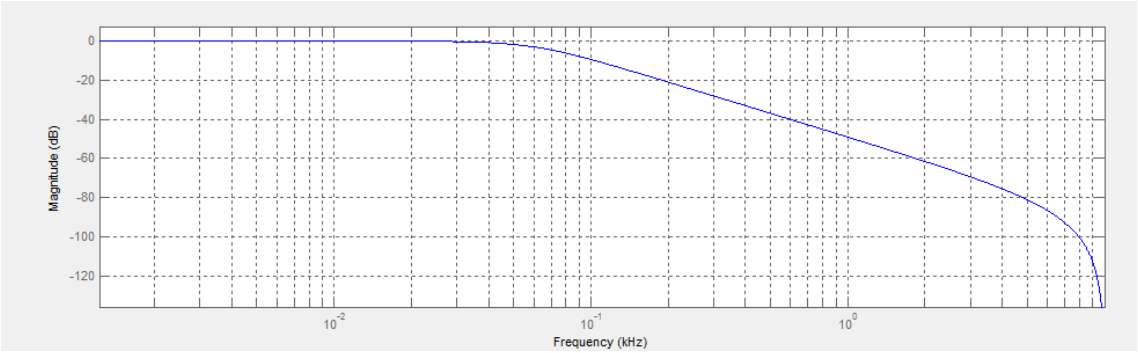


Figura 3.3

En la figura 5.3 se observa que tiene una ganancia de -40dB a 60Hz.

Orden 10:

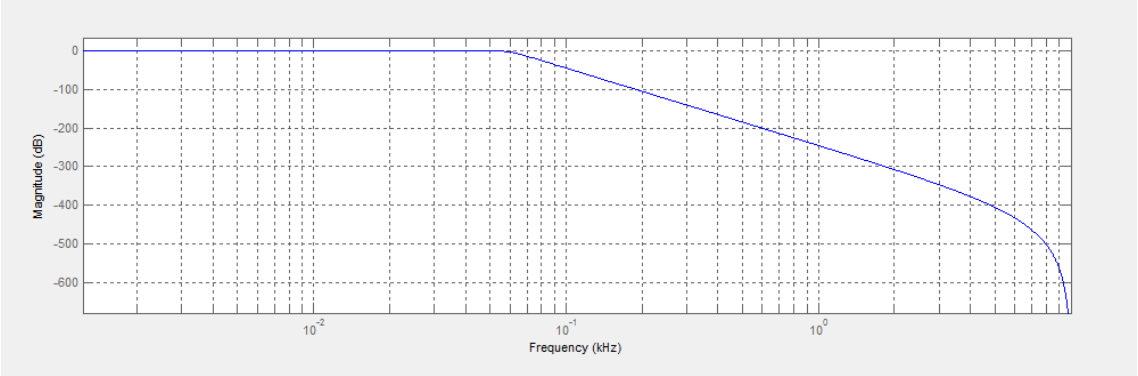


Figura 3.4

Ventajas: la banda pasante tiene ganancia la unidad exactamente.

Inconvenientes: se necesita un orden alto para eliminar bien la frecuencia deseada.

Filtros Chebyshev

Los filtros de Tchebyshev consiguen una caída más abrupta a frecuencias bajas en base a permitir un rizado de la respuesta en frecuencia en alguna de las bandas. Existen dos tipos:

- Tipo I
  Son filtros que solo tienen polos.
  Presentan rizado constante en la banda pasante.
  Presentan una caída monotónica en la banda no pasante.

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon^2 \cdot T_N^2(\Omega/\Omega_c)}$$

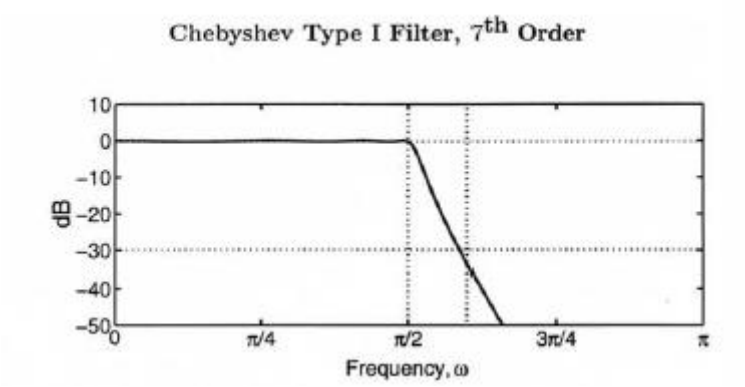


Figura 3.5 Referencia [2]

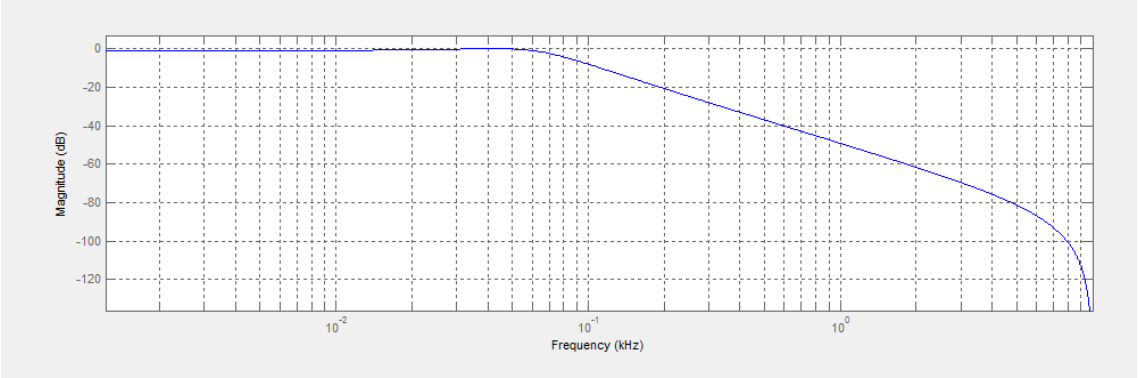


Figura 3.6

En la figura 5.6 se observa que tiene una ganancia de -40dB a 60Hz al igual que el de tipo Butterworth.

- Tipo II
  Presentan ceros y polos.
  Presentan rizado constante en la banda no pasante.
  Presentan una caída monotónica en la banda pasante.

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon^2 \cdot \frac{T_N^2(\Omega_s/\Omega_c)}{T_N^2(\Omega_s/\Omega)}}$$

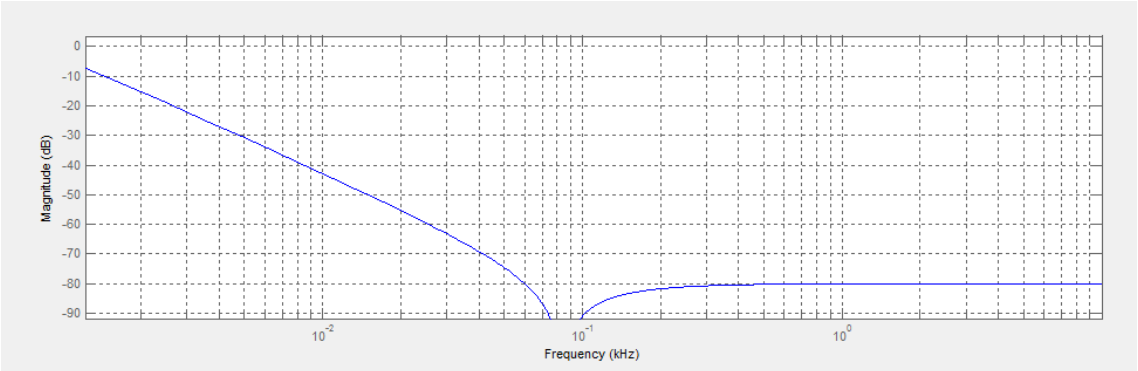


Figura 3.7

Filtros Elliptic

Los filtros elípticos o de Cauer consiguen estrechar la zona de transición permitiendo un rizado constante en ambas bandas.

$$|H(\Omega)|^2 = \frac{1}{1+\varepsilon^2 \cdot U_N(\Omega/\Omega_c)}$$

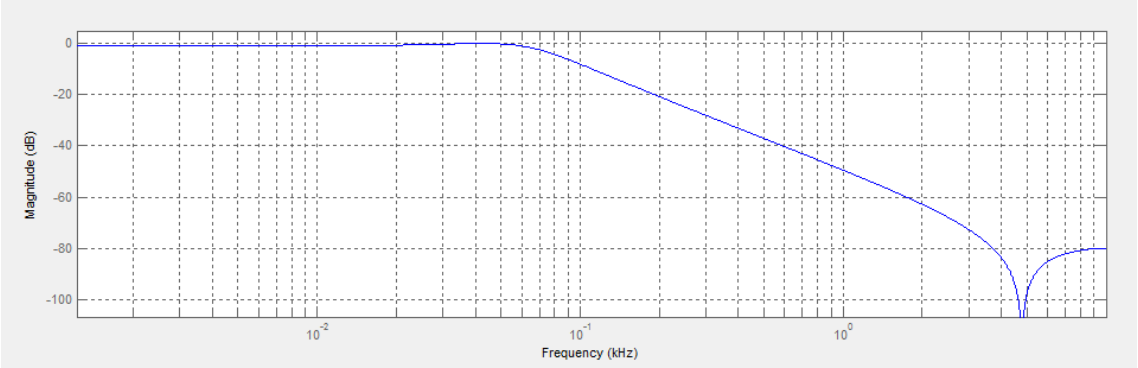
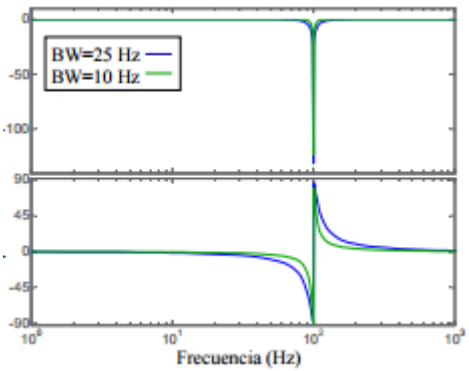


Figura 3.8

Se elegirá el filtro digital de tipo Butterworth porque no afecta a la frecuencia pasante y elimina la no deseada con un orden no demasiado elevado.

Filtro Notch o antirresonante

Rango estrecho de frecuencias.
Sólo afecta a la que se quiere eliminar.
A mayor ancho de banda respuesta más oscilante y menor tiempo de estabilización.



$$F(z) = \frac{1+\alpha}{2} \frac{z^2 - 2\beta \cdot z + 1}{z^2 - \beta(1+\alpha) \cdot z + \alpha}$$

Figura 3.9 Referencia [5]

Filtro de ventana deslizante

Elimina la frecuencia deseada y todos sus múltiplos.
Necesita la mitad del periodo de la fundamental para que la salida alcance el valor correcto (10ms en caso de una red a 50Hz).

$$F(z)=\frac{1-z^{-n}}{n(1-z^{-1})}$$

$$F(s)\approx\frac{1}{\tau_f s+1}$$

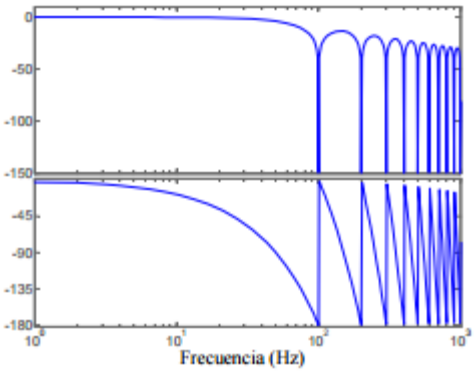


Figura 3.10 Referencia [5]

Filtro de retardo de cuarto de periodo o DSC

Se basa en sumar a una señal su valor una fracción de periodo antes.
Elimina las componentes impares.
El DSC4 elimina la componente al doble de la fundamental (100Hz).
Respuesta más rápida que los anteriores, un cuarto de periodo de la fundamental (5ms para red de 50Hz).

$$F(z)=\frac{1+z^{-n}}{2}$$

$$n=\frac{T_{fun}}{4T_m}$$

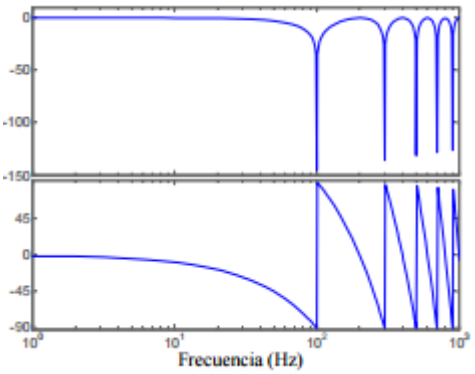


Figura 3.11 Referencia [5]

Filtro	Descripción	Velocidad	Respuesta ante hueco
Notch	Elimina sólo la frecuencia deseada.	Lento	Aísla frecuencias correctamente pero da respuesta oscilante. Solución: filtro paso bajo a 10Hz
Ventana deslizante	Elimina la frecuencia deseada y sus múltiplos.	Medio	Aísla secuencia directa e inversa bien
DSC4	Elimina múltiplos impares	Rápido	Aísla bien

A continuación se detallan algunas de las soluciones adoptadas en la actualidad para hacer frente a los problemas explicados anteriormente **[1]**.

3.2 *PLL dq*DSC

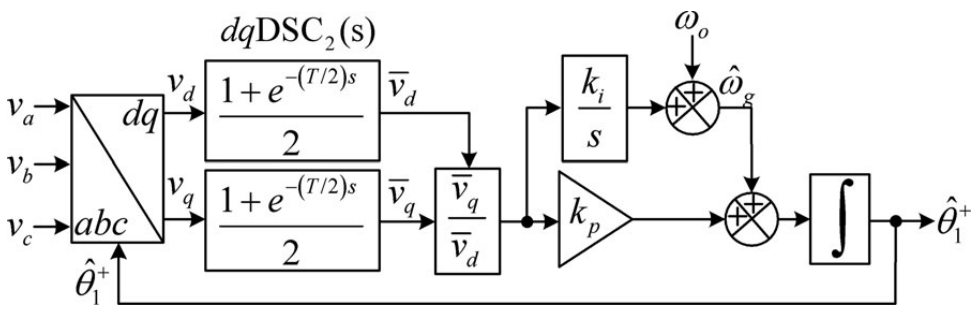


Figura 3.12

El PLL dqDSC utiliza filtros DSC para eliminar los armónicos no deseados. Al hacerlo en dq la secuencia inversa de los desequilibrios aparece al doble de la frecuencia fundamental y la componente continua a la frecuencia fundamental. Una vez eliminadas las frecuencias no deseadas con los filtros DSC se normaliza la tensión para que no haya problemas con tensiones demasiado altas y se utiliza el algoritmo SFR-PLL convencional.

Presencia de componente continua:

El filtro DSC elimina la componente de frecuencia fundamental y todos los armónicos de orden impar.

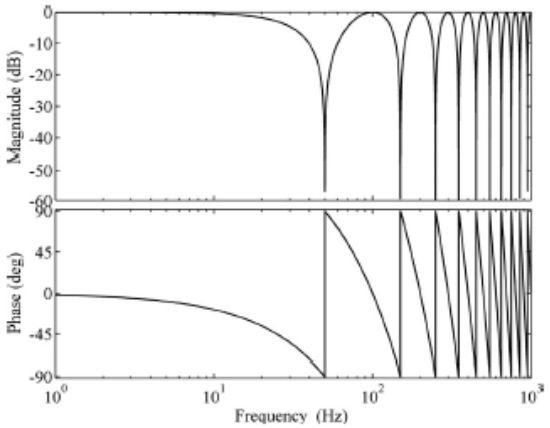


Figura 3.13

Bloquea frecuencias $f = \frac{n}{T} * (k + \frac{1}{2})$. Para bloquear la frecuencia fundamental f=1/T entonces $n = \frac{2}{2k+1}$

Problemas: añade un retardo inversamente proporcional a n. Por lo tanto n debe ser lo mayor posible (n=2) Cuando se producen cambios en la frecuencia de la red o saltos de ángulo, tiene una respuesta lenta (del orden de 50-70ms). Por lo tanto, el PLL dqDSC puede ser útil en aplicaciones donde se espera un comportamiento dinámico lento y amortiguado del PLL.

Además cuando hay componente continua y la frecuencia no es exactamente igual a la nominal se produce un pequeño error porque los filtros han sido diseñados para una frecuencia determinada.

En la figura 3.3 se observa el error en grados del ángulo obtenido por el PLL para diferentes frecuencias. Cuanto más se aleja de la frecuencia para la que ha sido diseñado el filtro mayor es el error obtenido.

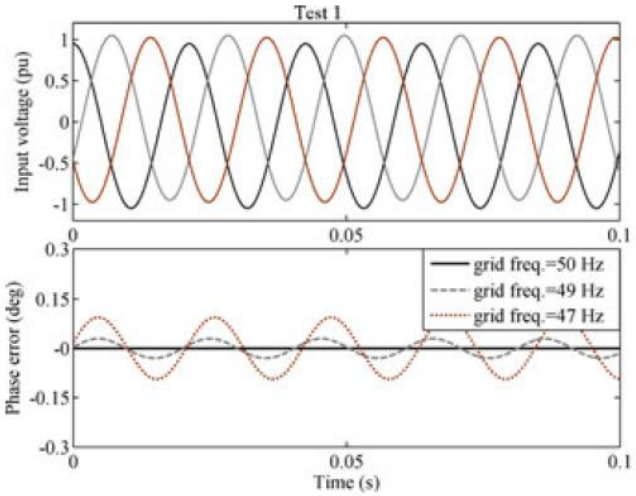


Figura 3.14

Variación de amplitud:

Al algoritmo PLL visto no le afecta ya que incluye un mecanismo de normalización de amplitud (ANM) para ser insensible a las variaciones de amplitud del voltaje.

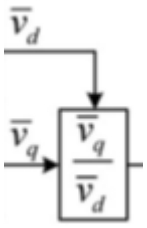


Figura 3.15

Presencia de armónicos a la entrada y red desequilibrada:

Bloquea los armónicos de orden par. Para bloquear el resto de armónicos se pueden incluir bloques DSC4 a diferentes frecuencias.

3.3 PLL $\alpha\beta$DSC

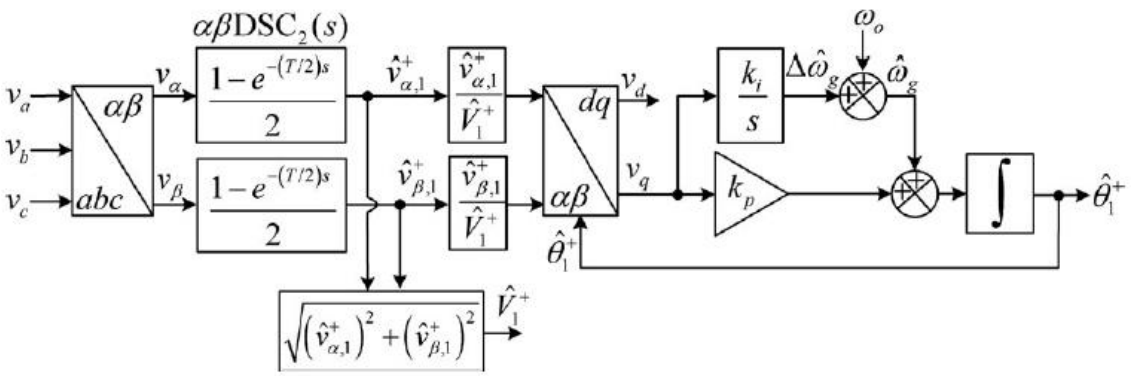


Figura 3.16

Utiliza el mismo PLL (standard synchronous reference frame PLL (SRF-PLL)) y, al igual que el PLl dqDSC utiliza filtros DSC para eliminar la componente continua. A diferencia del anterior éste lo hace en αβ.

Presencia de componente de corriente continua:

Con n=2 se bloquea la componente continua a la entrada del PLL

El αβ DSC deja pasar la fundamental y rechaza la componente continua (0Hz en αβ) y los armónicos de orden par.

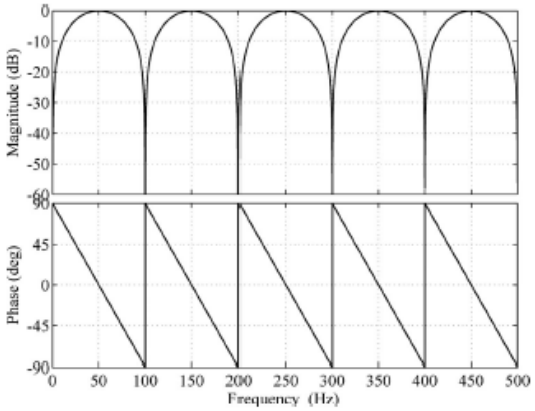


Figura 3.17

Problemas: cuando la frecuencia de red no está en su valor nominal (en derivas de frecuencia) se produce un desplazamiento de fase. Para compensar esto se suele realimentar el operador $\alpha\ \beta$ DSC con la frecuencia para que se adapte. Esto hace que sea no lineal y que difícilmente sea estable en todas las circunstancias. Para evitar este problema se corrige el error a la salida del PLL.

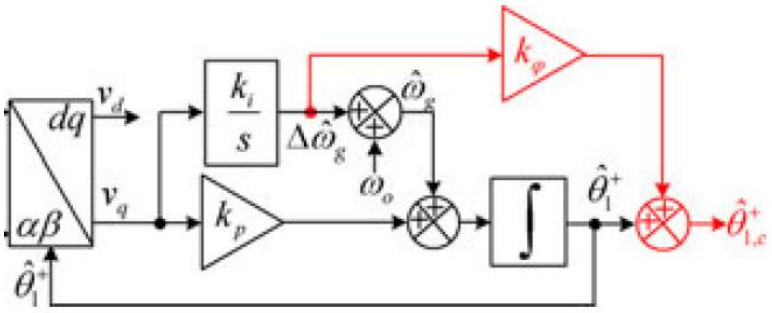


Figura 3.18

Ventaja con respecto al PLL dqDSC: el anterior PLL presenta una pequeña desviación de la frecuencia cuando hay componente continua y hay deriva en frecuencia, sin embargo este PLL no gracias a la ganancia cero a frecuencia 0.

Además presenta una respuesta dinámica más rápida que el PLL1 dqDSC.

Variación de amplitud:

Al algoritmo PLL visto no le afecta ya que incluye un mecanismo de normalización de amplitud (ANM) para ser insensible a las variaciones de amplitud del voltaje.

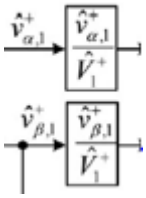


Figura 3.19

Presencia de armónicos a la entrada y red desequilibrada:

El operador αβDSC2 sólo puede bloquear los componentes armónicos de orden par, para eliminar los componentes negativos y los de orden impar se necesitan 3 operadores αβDSC2 en cascada con factor de retraso (n) 4, 8 y 16. (En este caso la Kp debería ser 15T/32).

3.4 PLL NF

Un NF es un filtro de rechazo de banda que atenúa significativamente las señales dentro de una banda de frecuencias y pasa todas las otras frecuencias casi sin cambio.

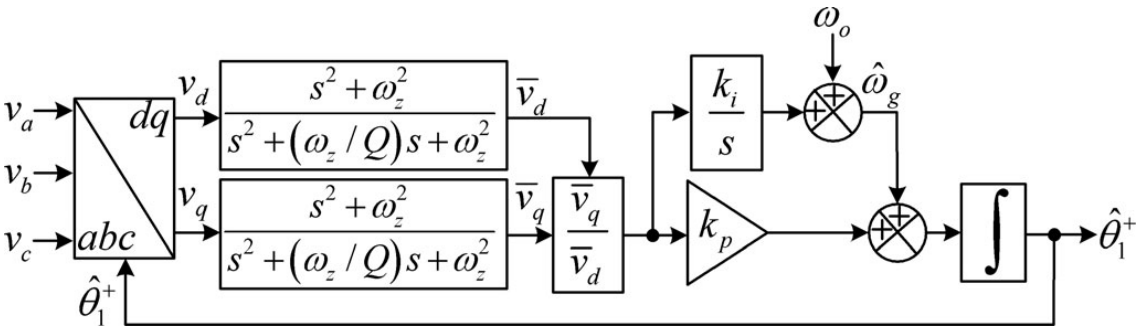


Figura 3.20

Presencia de componente de corriente continua:

Ventaja: Se puede diseñar para que bloquee efectivamente la componente continua incluso en presencia de grandes variaciones en la frecuencia de red.

Problema: Esta ventaja es acosta de inducir un retardo de fase considerable que incluso puede llegar a poner en peligro la estabilidad. Su velocidad es parecida a la del PLL dqDSC.

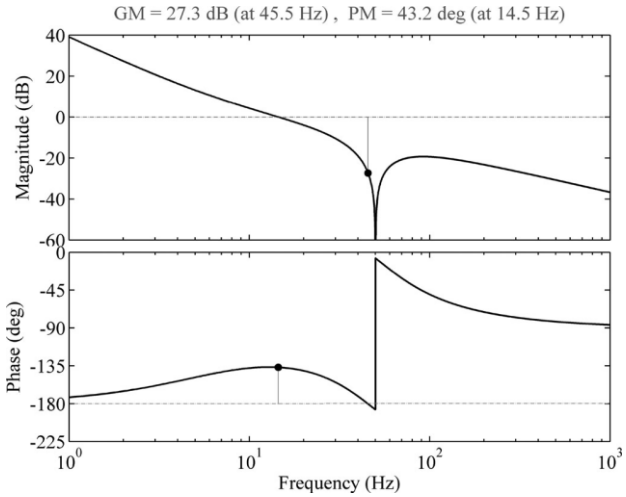


Figura 3.21

Comparación error cuando la red tiene componente continua, PLL dqDSC vs PLL NF:

PLL dqDSC:

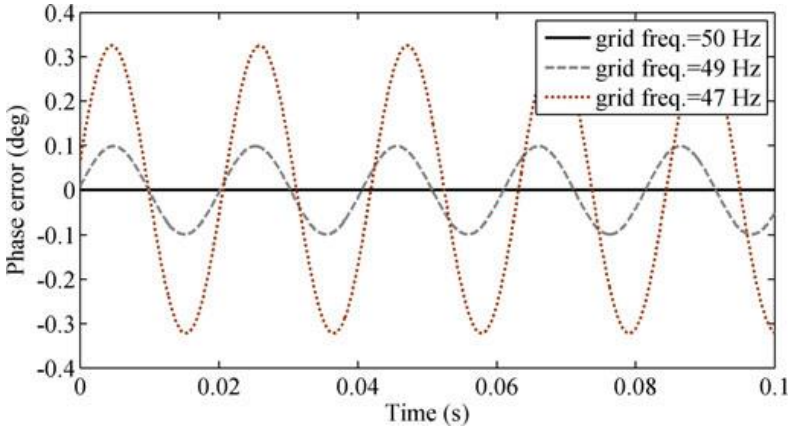


Figura 3.22

PLL NF:

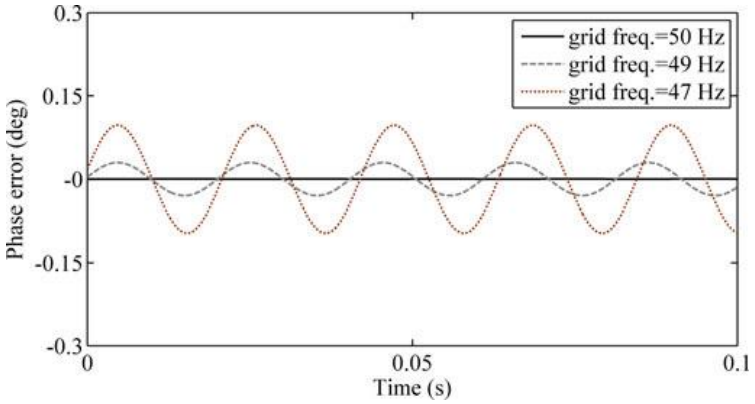


Figura 3.23

Se observa cómo el error es mucho mayor en el caso del PLL dqDSC.

Variación de amplitud:

Al algoritmo PLL visto no le afecta ya que incluye un mecanismo de normalización de amplitud (ANM) para ser insensible a las variaciones de amplitud del voltaje.

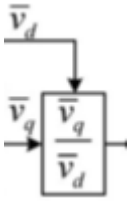


Figura 3.24

Presencia de armónicos a la entrada y red desequilibrada:

Se pueden incluir NFs adicionales para mejorar la respuesta frente a armónicos y desequilibrio en la red. Sobre todo las componentes de orden -5 y +7 que son las que suele llevar la red. Estas

componentes son vistas por el PLL con h=6 y 12. La componente negativa aparece como una componente al doble de la frecuencia.

Por lo tanto el uso de tres NFs con frecuencias de corte de 2π (2*50), 2π (6*50) y 2π (12*50) rad/s es suficientemente bueno en los casos prácticos.

3.5 PLL CFN

El filtro paso bajo que se utiliza puede ser cualquiera, en este caso consideramos uno con función de transferencia:

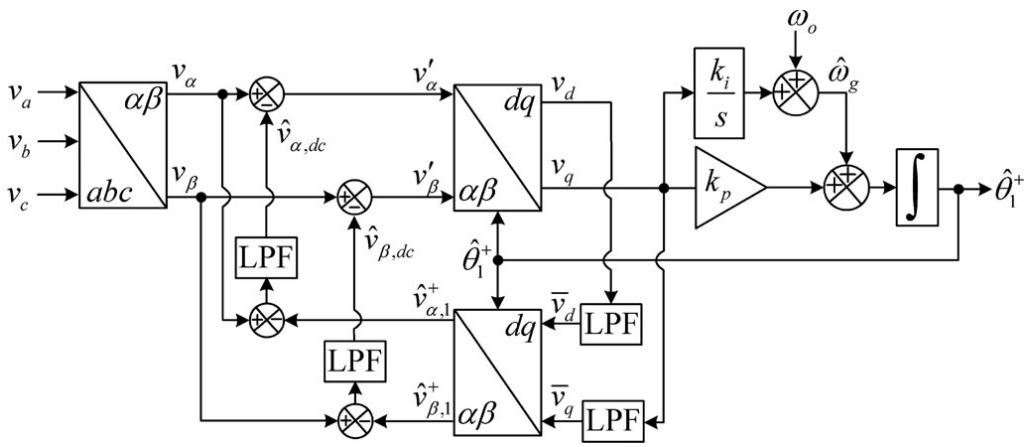


Figura 3.25

- Primero las componentes de tensión en d y q se pasan a través de dos filtros LPF para eliminar sus posibles alteraciones. Sólo pasa la componente a 0Hz, que como está en dq es la de 50Hz real.
- Se vuelven a transformar al marco de referencia estacionario alfa-beta.
- Estas componentes se restan a la tensión de red y así sólo queda la componente continua de la red. Eso se pasa por un filtro paso bajo, que da una estimación de la componente continua de la tensión de red.
- Finalmente eso se resta a la tensión de red y así se elimina el offset de continua de la entrada al PLL.

Presencia de componente de corriente continua:

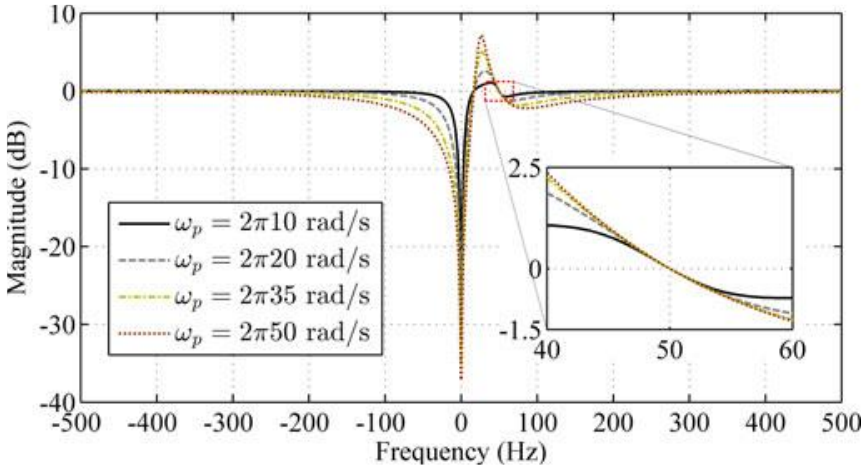


Figura 3.26

Se puede observar cómo a 50Hz la ganancia es la unidad.

El error de seguimiento será cero aunque varíe la frecuencia, como en el caso del PLL2, ya que se aplica en el eje estacionario alfa-beta y no sobre dq. Además es relativamente rápido (40ms).

Variación de amplitud:

Al algoritmo PLL visto no le afecta ya que incluye un mecanismo de normalización de amplitud (ANM) para ser insensible a las variaciones de amplitud del voltaje. Se divide por Vd que es una estimación de la amplitud de la tensión de red.

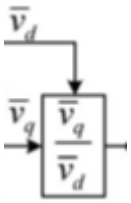


Figura 3.27

Presencia de armónicos a la entrada y red desequilibrada:

Este PLL no presenta la facilidad de los otros para evitar armónicos mediante la puesta en cascada de varios bloques. Sí que se puede conseguir evitar los armónicos a bajas frecuencias mediante la extensión del CFN pero hay que tener en cuenta que el esfuerzo computacional aumenta mucho (2 funciones trigonométricas más, 2 filtros de paso bajo más y varias sumas y multiplicaciones).

3.6 PLL CCF

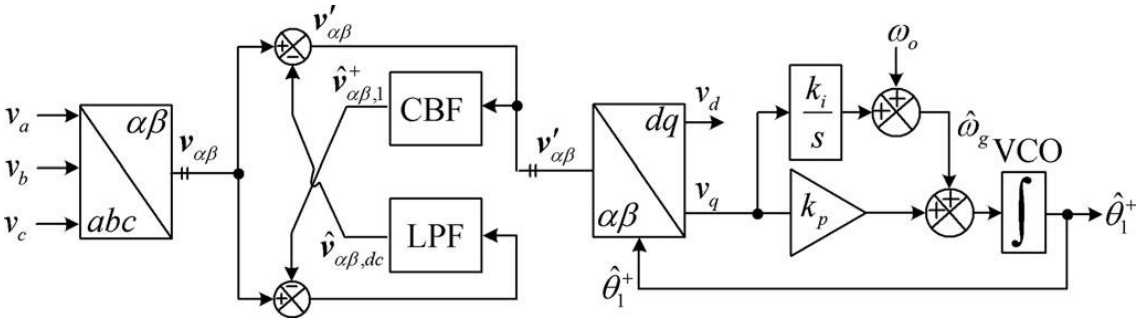


Figura 3.28

Los CCF tienen una respuesta en frecuencia asimétrica alrededor de cero, lo que permite hacer distinción entre la secuencia positiva y la negativa de la misma componente de frecuencia. Esta característica mejora la capacidad del filtrado.

En este caso se estima el vector tensión de red (FFPS) utilizando un filtro paso banda complejo (CBF) con la frecuencia central en la frecuencia fundamental en secuencia positiva como se puede observar en la figura 3.18. De esta manera sólo deja pasar la componente a 50Hz.

- Pasa la tensión de red por un paso banda a 50Hz. Lo que deja una componente a 50Hz y el resto atenuadas.
- Esto se resta a la tensión de red, eliminando así la componente a 50Hz de la red.
- Se hace pasar por un paso bajo y así queda sólo la continua.
- Se resta a la tensión de red y ya se ha eliminado el offset de continua de la tensión de red. Esto se lleva a la entrada del PLL.

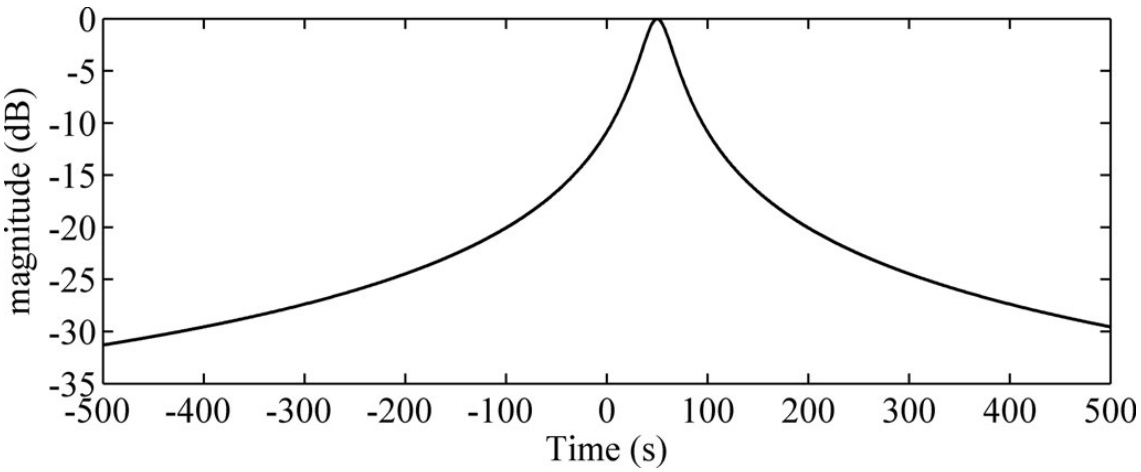


Figura 3.29

Para implementar el CBF:

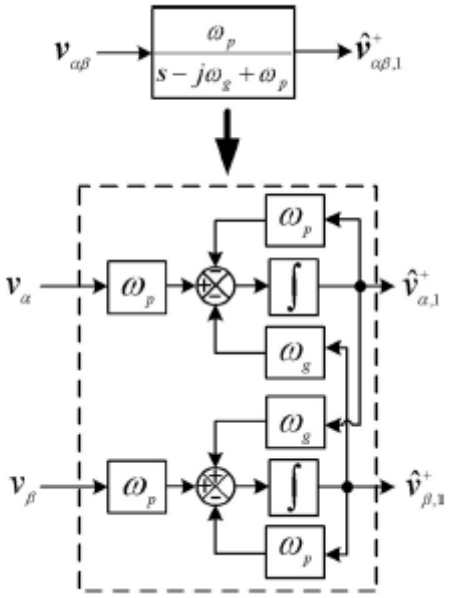


Figura 3.30

Presencia de componente de corriente continua:

El rendimiento dinámico y la capacidad de rechazo de perturbaciones es igual al de la CFN.

Variación de amplitud:

Al algoritmo PLL visto no le afecta ya que incluye un mecanismo de normalización de amplitud (ANM) para ser insensible a las variaciones de amplitud del voltaje. Se divide por Vd que es una estimación de la amplitud de la tensión de red.

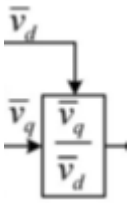


Figura 3.31

Presencia de armónicos a la entrada y red desequilibrada:

Se puede eliminar componentes de la frecuencia que queramos añadiendo CBFs centrados en dicha frecuencia y restándosela a la de la red.

Se implementarán los PLL dqDSC y el CCF para compararlos con un nuevo PLL propuesto ya que son representativos de los dos tipos de PLL que existen, los que utilizan filtros en dq y los que los utilizan en αβ.

3.7 Resumen PLL analizados

Solución	Descripción	Veocidad	Componente continua	Variación de amplitud	Derivas en frecuencia	Armónicos y redes desequilibradas
dqDSC	Filtro de respuesta finita de impulso que rechaza la componente continua. No deja pasar la componente a 50Hz en dq.	Lento	La elimina aunque da pequeño error con derivas en frecuencia	No le afecta, incluye un mecanismo de normalización de amplitud (ANM)	Pequeña desviación cuando la frecuencia no es la nominal	Bloquea los armónicos de orden par de forma natural, para el resto hay que incluir más bloques.
Figura 3.33 **αβDSC**	Similar al anterior pero utilizando el filtro en alfa-beta. No deja pasar la componente a 0Hz en alfa-beta.	Rápido	La elimina totalmente	No le afecta, incluye un mecanismo de normalización de amplitud (ANM)	Se incluye solución para que el error sea cero aun cuando la frecuencia no sea la nominal.	Bloquea los armónicos de orden par de forma natural, para el resto hay que incluir más bloques. No elimina desequilibrios.
NF	Filtro de paso banda que atenúa la señal de continua	Lento	La elimina y se puede diseñar para que la bloquee efectivamente incluso con grandes variaciones en la frecuencia de red.	No le afecta, incluye un mecanismo de normalización de amplitud (ANM)	Producen un pequeño error pero mucho menor que en el caso del dqDSC	Usar varios NFs con dierentes frecuecnias de corte para eliminar las componentes que se requiera
CFN	Utiliza filtros paso bajo para eliminar la componente continua	Rápido	La elimina totalmente	No le afecta, incluye un mecanismo de normalización de amplitud (ANM)	El error es cero aun cuando la frecuencia no es la nominal	Aumenta mucho el coste computacional si se pretende eliminar armónicos.
CCF	Utiliza un filtro paso banda y otro paso bajo para eliminar la componente continua	Rápido	La elimina totalmente	No le afecta, incluye un mecanismo de normalización de amplitud (ANM)	El error es cero aun cuando la frecuencia no es la nominal	Se puede eliminar componentes de la frecuencia que se requiera añadiendo CBFs centrados en dicha frecuencia y restándoselo a la red.

Figura 3.32

4 PROPUESTA DE UN NUEVO ALGORITMO PLL

Se propone un nuevo diseño que garantice que el ángulo generado no se va a ver afectado por los desequilibrios, la componente continua, las derivas en frecuencia ni las variaciones de tensión y que además tenga una respuesta dinámica rápida.

Para ello se pretende eliminar la componente continua en el sistema alfa-beta, ya que aparece como una componente a 0Hz, y eliminar los desequilibrios en el sistema dq, que aparecen a 100Hz en secuencia inversa. Se propone por tanto un sistema híbrido entre los analizados.

Para usar la menor cantidad de filtros posible y que estos sean capaces de eliminar todos los armónicos no deseados se pretende incluir en αβ un filtro pasobanda que elimine todos los armónicos que no se encuentren a 50Hz. Además se incluirá un filtro pasobajo en dq que elimine el resto de armónicos como la secuencia inversa de los desequilibrios que aparece a 50Hz.

Para diseñar el PLL completo hay que tener en cuenta todos los problemas presentados anteriormente y que éste sea capaz de solucionarlo.

- Presencia de componente continua: para que no le afecte se eliminará utilizando un filtro pasobanda en alfa-beta a 50Hz.
- Desequilibrios: en alfa-beta no es posible eliminarlos ya que aparecen a la frecuencia de red. Por lo tanto se eliminarán en dq ya que al necesitar eliminar la secuencia inversa aparecerán a una frecuencia el doble de la de la red (100Hz). Para ello se utilizará un filtro pasobajo a una frecuencia de 70Hz suficiente para eliminarlos y tener una respuesta dinámica rápida.
- Distorsión: que sea capaz de eliminar los armónicos de la red no deseados, con los dos filtros se consigue esto.

Después de realizar las pruebas pertinentes y realizar los cambios oportunos que se verán con detalle en el siguiente apartado, el diseño definitivo es el de la figura 4.1.

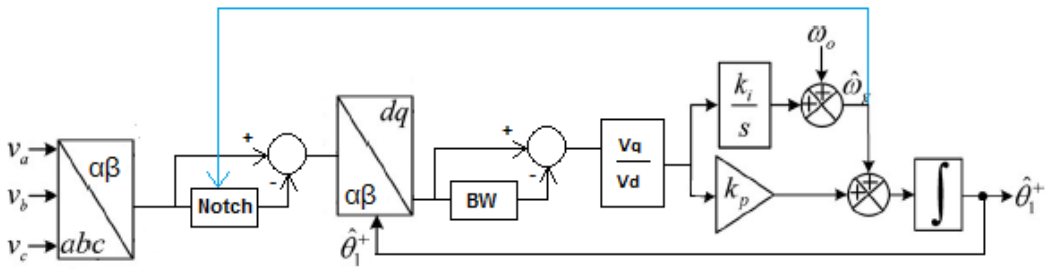


Figura 4.1

5 DISEÑO

A continuación se hace un análisis de los filtros digitales **[2]** y se escogerá el más adecuado en cada caso para la aplicación que se necesite. Para el diseño de los filtros se utilizará la herramienta *fdatool* de Matlab que proporciona la información necesaria para su implementación dando los coeficientes del filtro en Z y el diagrama de Bode del filtro.

5.1 PLL1

En primer lugar como algoritmo PLL se utiliza el SFR-PLL que es el que se utiliza comúnmente en la mayoría de aplicaciones con PLL. En este caso se diseña para una frecuencia de corte de 10Hz que va a permitir que el PLL no tenga en cuenta las frecuencias superiores. Algo interesante ya que la frecuencia fundamental aparece en dq a 0Hz.

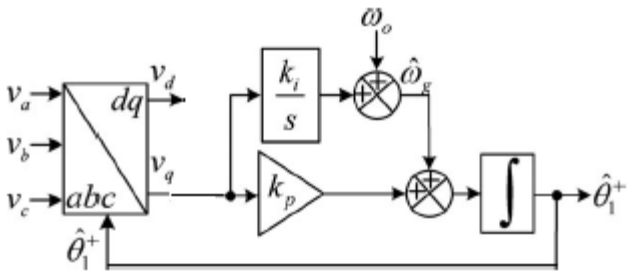


Figura 5.1

La función de transferencia del algoritmo SFR-PLL es la siguiente:

$$FT_{LA}(s) = K_p \frac{T_n s + 1}{T_n s} \cdot \frac{1}{s} \cdot \frac{1}{1{,}5 \cdot T_m \cdot s + 1}$$

Con una frecuencia de corte de 10Hz, un margen de fase de 60º para asegurar la estabilidad y una frecuencia de muestreo de 20KHz se obtienen los siguientes parámetros para el diseño del proporcional integrador:

TN	**0.02786891**
Kp	**54.562025**

Para diseñar el PLL completo hay que tener en cuenta todos los problemas presentados anteriormente y que éste sea capaz de solucionarlo.

- Presencia de componente continua: para que no le afecte se eliminará utilizando un filtro pasobanda en alfa-beta a 50Hz.

- Desequilibrios: en alfa-beta no es posible eliminarlos ya que aparecen a la frecuencia de red. Por lo tanto se eliminarán en dq ya que al necesitar eliminar la secuencia inversa aparecerán a una frecuencia el doble de la de la red (100Hz). Para ello se utilizará un filtro pasobajo a una frecuencia de 70Hz suficiente para eliminarlos y tener una respuesta dinámica rápida.
- Distorsión: que sea capaz de eliminar los armónicos de la red no deseados.

Para ambos casos se utilizarán filtros Butterworth ya que en la banda pasante la ganancia es exactamente la unidad.

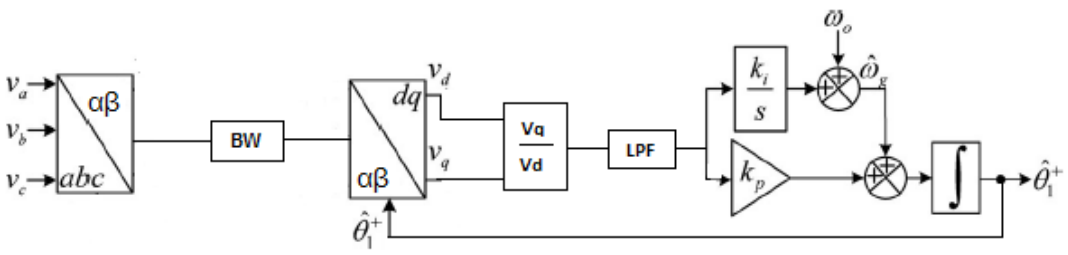


Figura 5.2

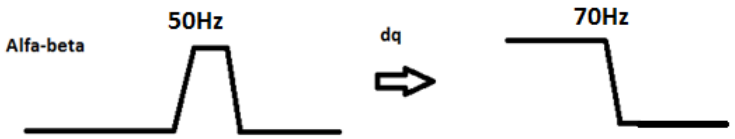


Figura 5.3

Filtro pasobanda a 50Hz para eliminar la componente continua:

De nuevo se utiliza la herramienta para diseñar filtros digitales de MATLAB utilizando un filtro de tipo Butterworth con una frecuencia de muestreo de 20KHz y el menor orden posible para minimizar el coste computacional.

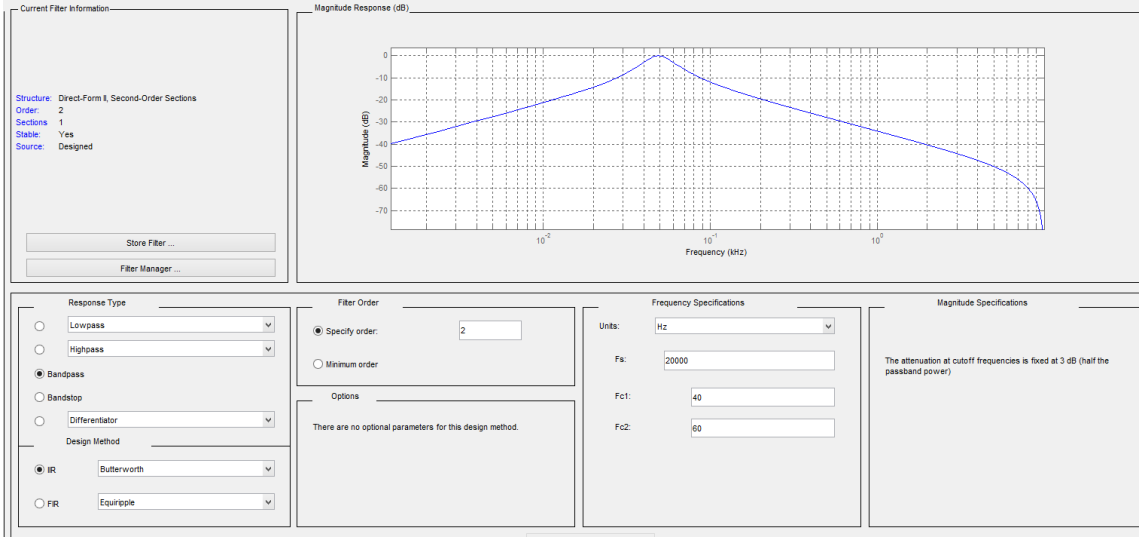


Figura 5.4

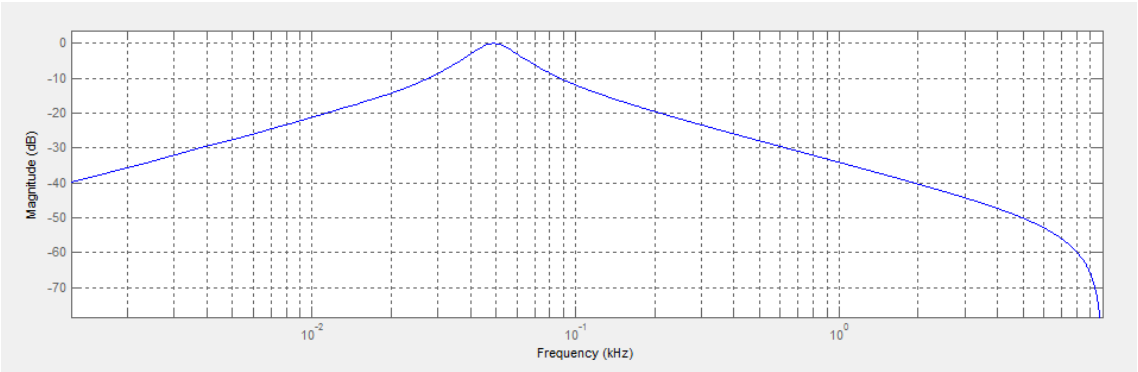


Figura 5.5

```
--------------------------
Section #1
--------------------------
Numerator:
 1
 0
-1
Denominator:
 1
-1.9935003467427104
 0.9937364715416203 7
Gain:
0.0031317642291898312
--------------------------
Output Gain:
1
```

Figura 5.6

En la figura 5.17 se encuentran los coeficientes de la función de transferencia en Z del filtro.

En este caso para comprobar el funcionamiento del filtro programado en C se suma una fuente senoidal a 50Hz y 220V con una continua na 220V en el programa PSIM. El filtro debe rechazar la continua y dejar intacta a senoidal:

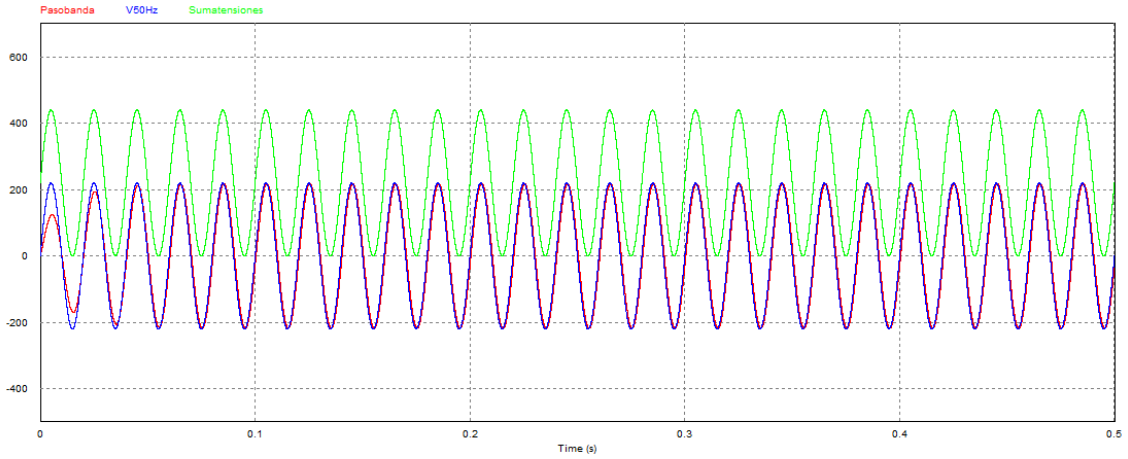


Figura 5.7

En la figura 5.18 se observa cómo el filtro funciona correctamente. En verde la tensión de entrada al filtro, la suma de una senoidal a 50Hz y una señal continua. En rojo la señal filtrada y en azul la señal senoidal a 50Hz sin la continua sumada. La señal filtrada coincide con la senoidal a 50Hz por lo que se comprueba que es capaz de eliminar la componente continua correctamente.

Aunque aparece un pequeño retraso:

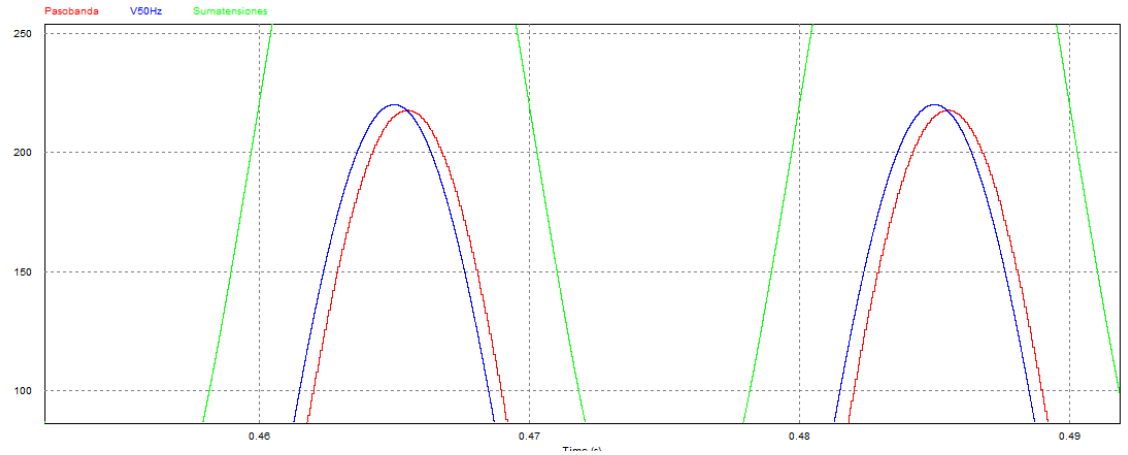


Figura 5.8

Se prueba con un filtro pasobanda de mayor orden, 4 en lugar de 2:

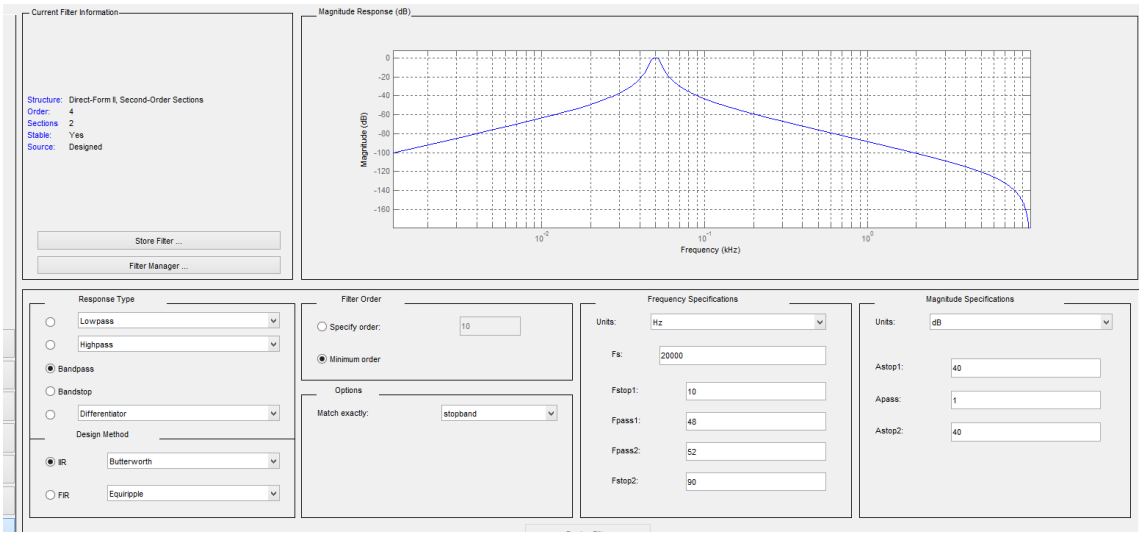


Figura 5.9

```
--------------------------
Section #1
--------------------------
Numerator:
 1
 0
-1
Denominator:
 1
-1.9982879151587798
 0.99855677281431143
Gain:
0.0009774756818565029
--------------------------
--------------------------
Section #2
--------------------------
Numerator:
 1
 0
-1
Denominator:
 1
-1.998453116015676
 0.99867851282493636
Gain:
0.0009774756818565029
--------------------------
Output Gain:
1
```

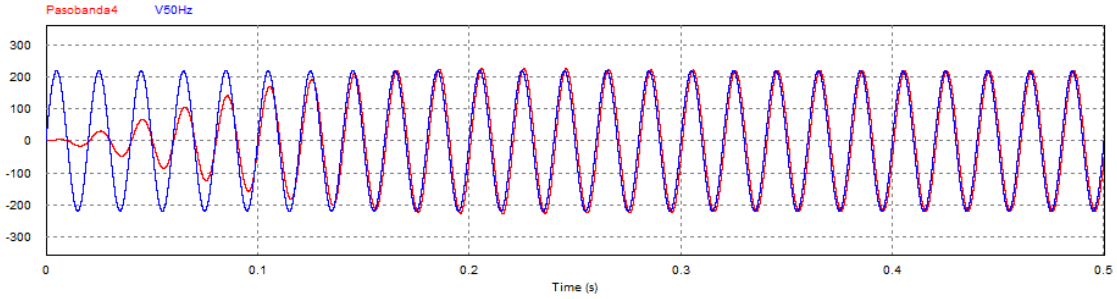


Figura 5.10

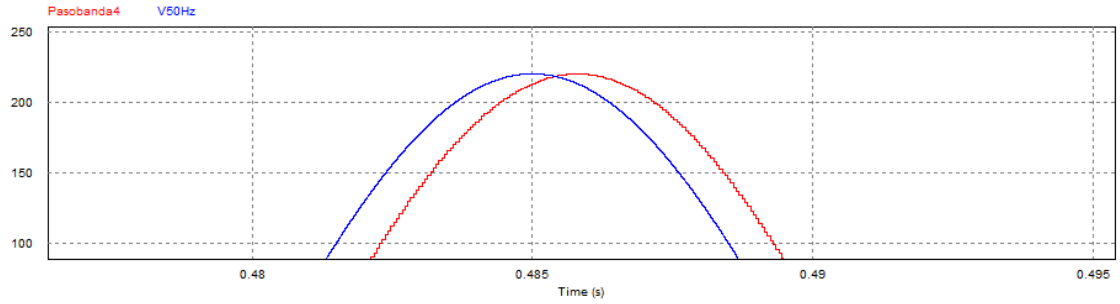


Figura 5.11

Comparación pasobandas:

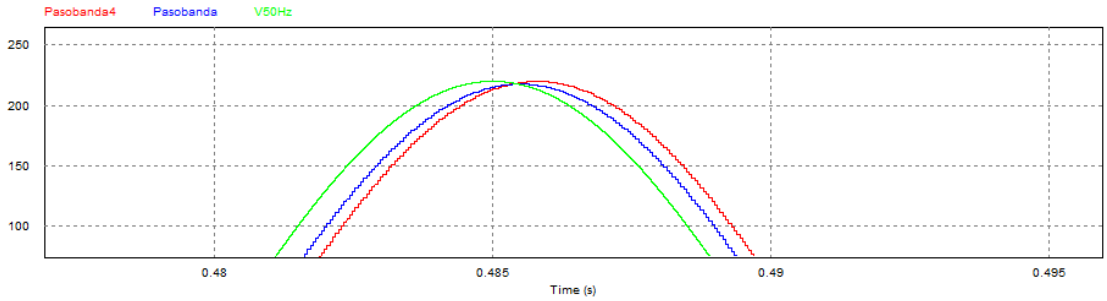


Figura 5.12

El de orden 4 se ajusta más al valor (ya que elimina mejor la continua) pero tiene mayor retraso por lo que se implementará el pasobanda de orden 2.

Filtro pasobajo para eliminar la secuencia inversa en los desequilibrios:

Se escogerá una frecuencia pasante de hasta 3Hz por si se producen derivas en frecuencia en la red y una frecuencia de corte de 70Hz con una ganancia de -45dB (algo menos de 0.01). Así se consiguen eliminar los desequilibrios que aparecen en dq a una frecuencia del doble de la fundamental que en condiciones normales serán 100Hz. (A 100Hz la ganancia es de -50dB).

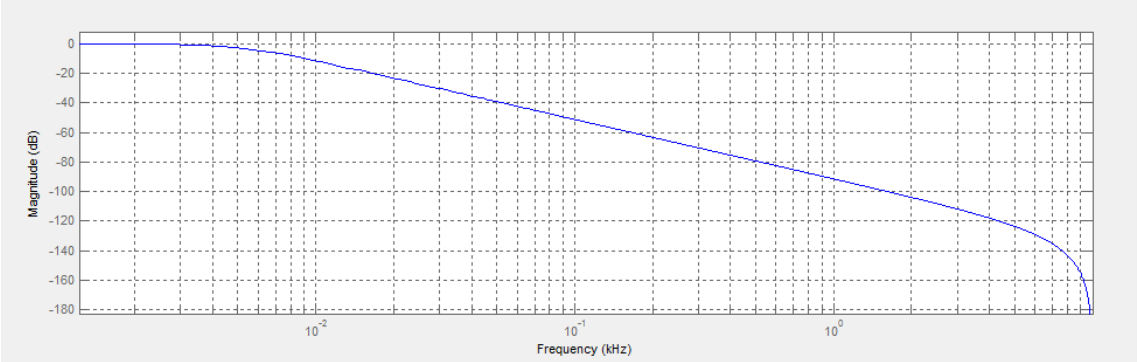


Figura 5.13

```
--------------------------
Section #1
--------------------------
Numerator:
1
2
1
Denominator:
 1
-1.9976677046200626
 0.99767042125477967
Gain:
0.00000067915867934128506
--------------------------
Output Gain:
1
```

Prueba de implementación en C:

Se hace la prueba sumando 2 tensiones, una a 100Hz y de amplitud 220V y otra continua de 220V, se hace pasar por el filtro que debe eliminar la componente a 100Hz como así lo hace:

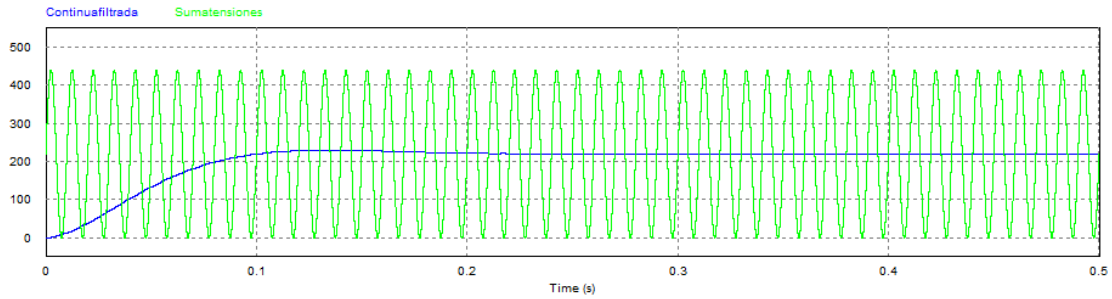


Figura 5.14

Tiene un pequeño rizado debido a que la ganancia a la que se ha diseñado el filtro es de unos -45dB a 100Hz. Se podría eliminar esto pero el orden subiría y no compensa el coste computacional con la precisión que se necesita.

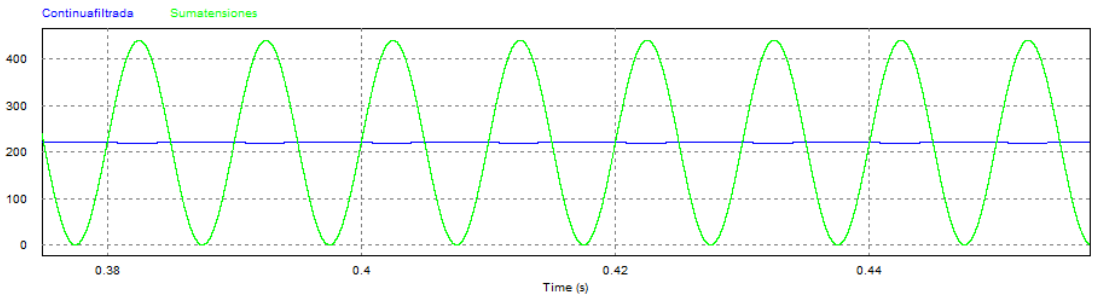


Figura 5.15

Problema:

Como el algoritmo del PLL hace que Vq sea 0 y la Vq se filtra previamente por un pasobajo, la frecuencia no es la real sino una muy alta porque de esa forma la Vq filtrada que llega al algoritmo PLL también es 0:

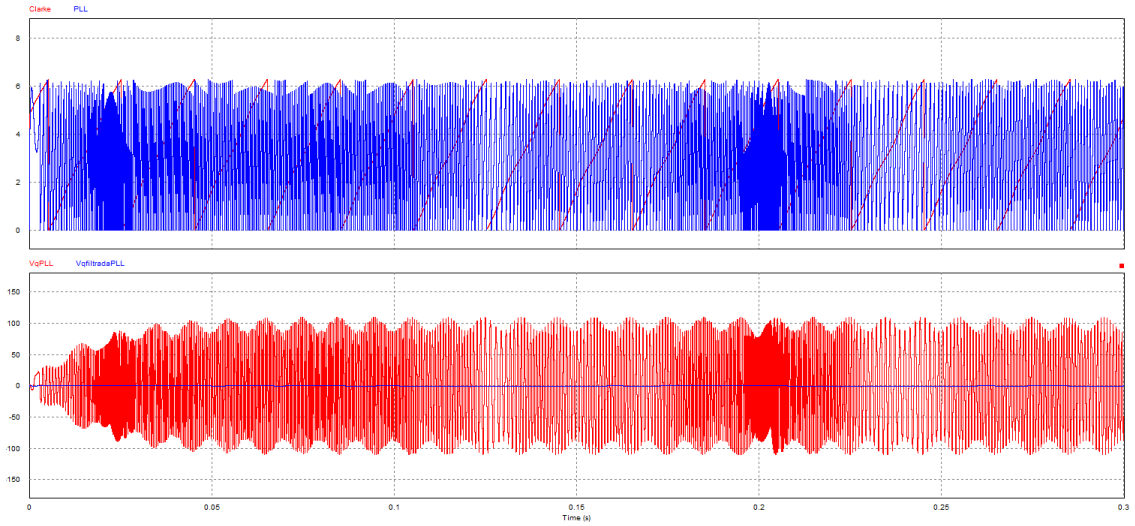


Figura 5.16

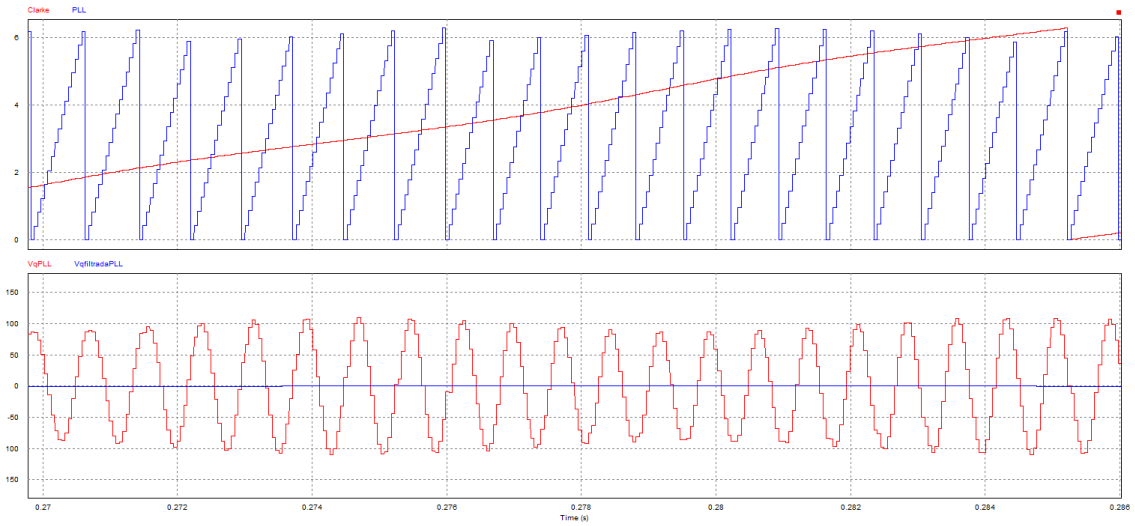


Figura 5.17

Posible solución: limitador de frecuencia entre 40 y 60Hz

```
/**Limitador**/

if(w_1CCF>2*pi*60)
{w_1CCF=2*pi*60;
error_acumCCF=error_acumCCF-VCCF*Tm;  //Al hacer esto la integral se queda como estaba.
}
if(w_1CCF<2*pi*40)
{w_1CCF=2*pi*40;
error_acumCCF=error_acumCCF-VCCF*Tm;  //Al hacer esto la integral se queda como estaba.
}
```

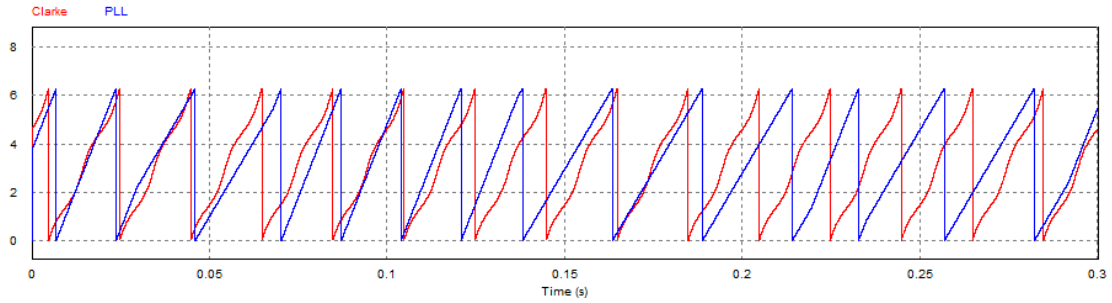


Figura 5.18

Como se ve en la figura 5.29 no funciona bien ya que la frecuencia obtenida no es la correcta que debería coincidir con la obtenida a partir de Clarke.

Solución:

- Utilizar un filtro Notch en lugar de un pasobajo. Inconveniente: malo ante derivas de frecuencia.
- Utilizar un filtro pasobanda a 100Hz para dejar pasar esa componente (desequilibrios) y restársela a la fundamental.

Se opta por la solución del pasobanda ya que de esta forma se puede escoger el ancho de banda adecuado para que no le afecten las derivas en frecuencia.

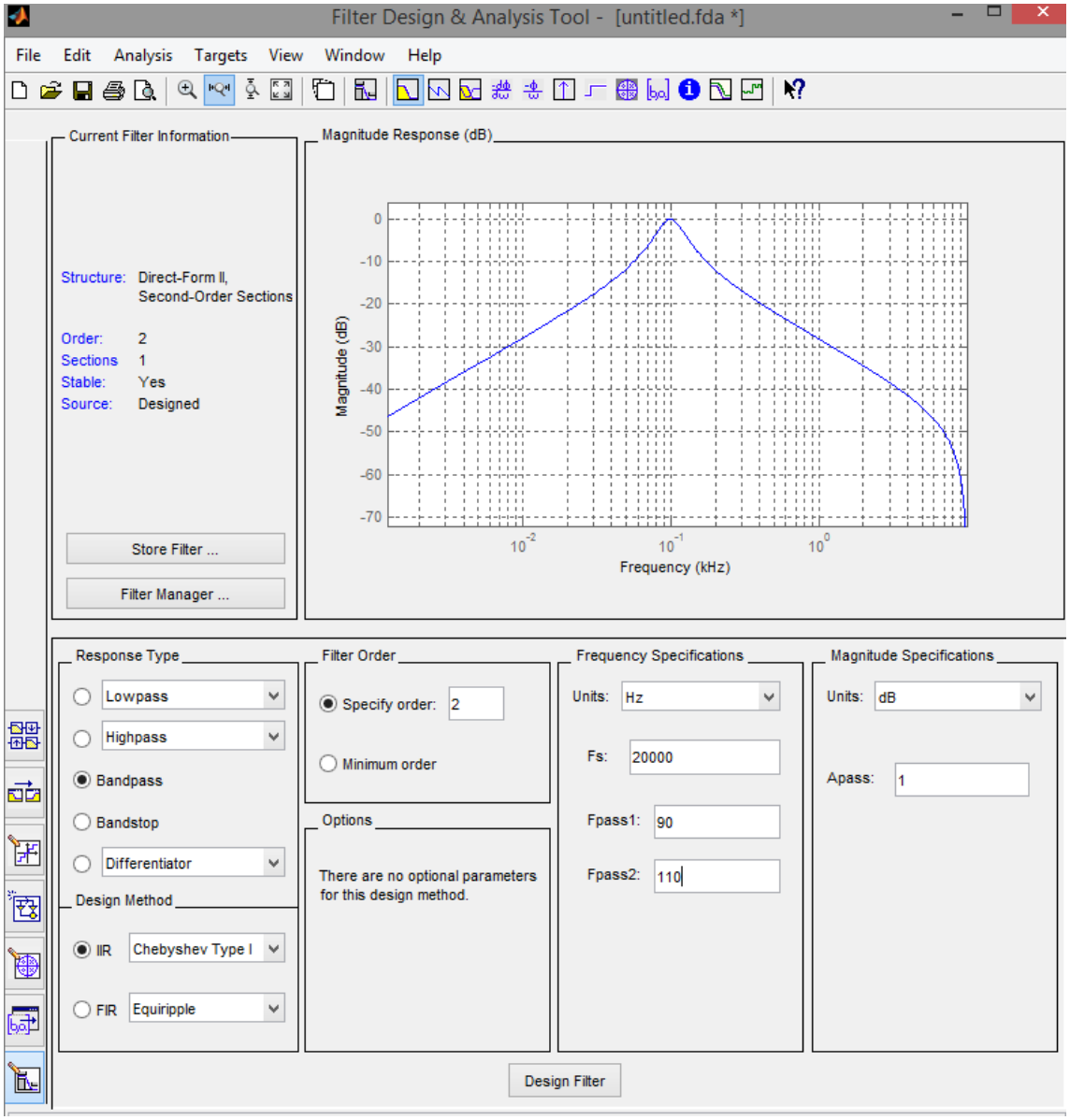


Figura 5.19

Prueba de funcionamiento del pasobanda:

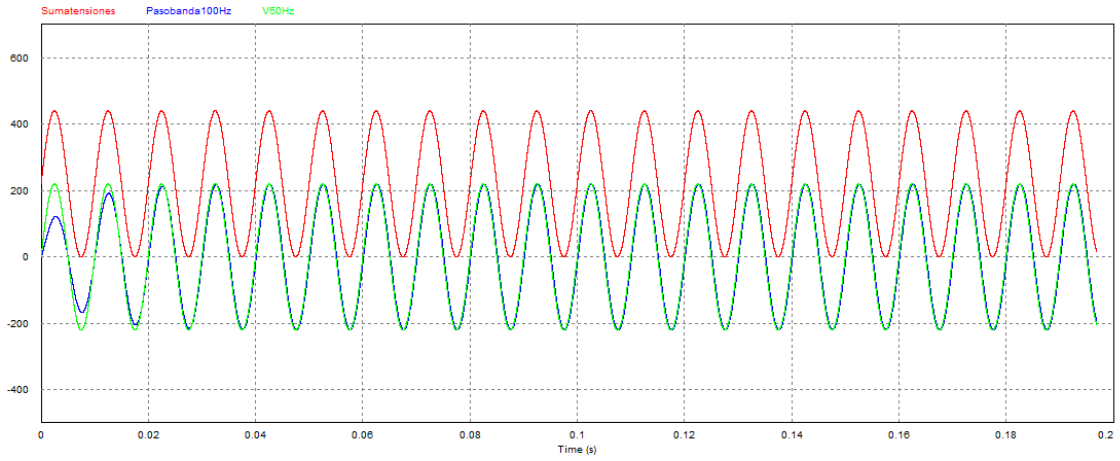


Figura 5.20

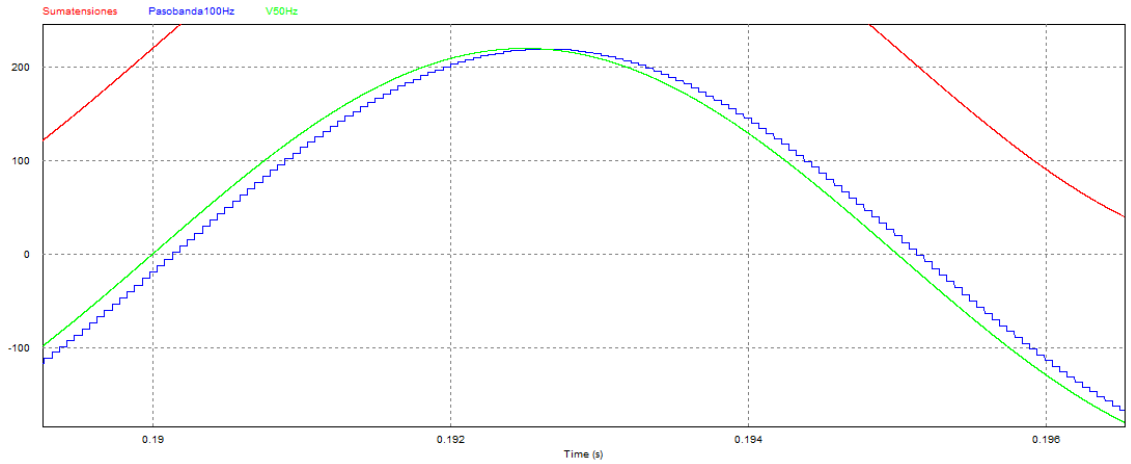


Figura 5.21

Tiene un retraso. Si lo comparamos con la senoidal muestreada:

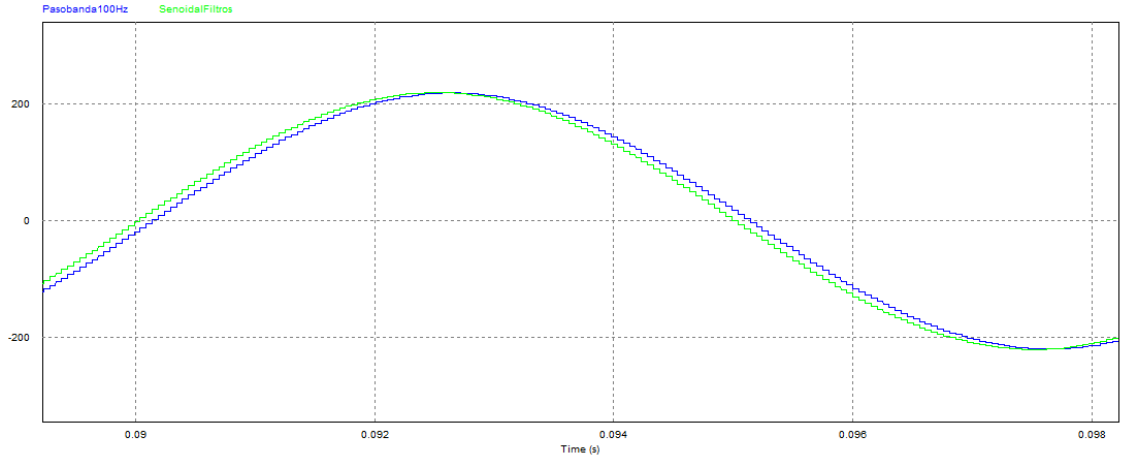


Figura 5.22

Sigue teniendo un retraso de aproximadamente 0.16ms.

Por lo tanto el PLL1 con los filtros diseñados toma la siguiente forma:

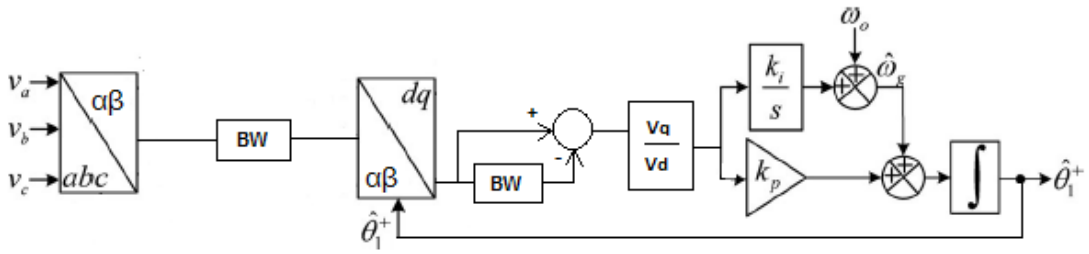


Figura 5.23

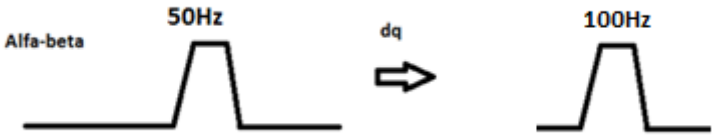


Figura 5.24

Se comprueba si la solución adoptada funciona utilizando el programa PSIM y simulando una red sin desequilibrios ni continua:

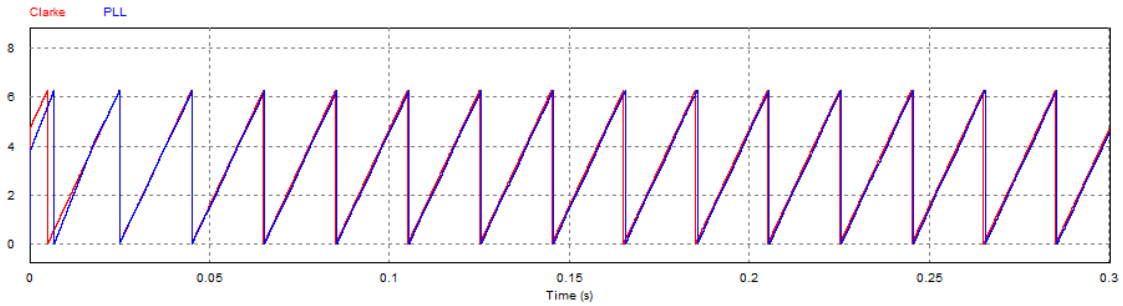


Figura 5.25

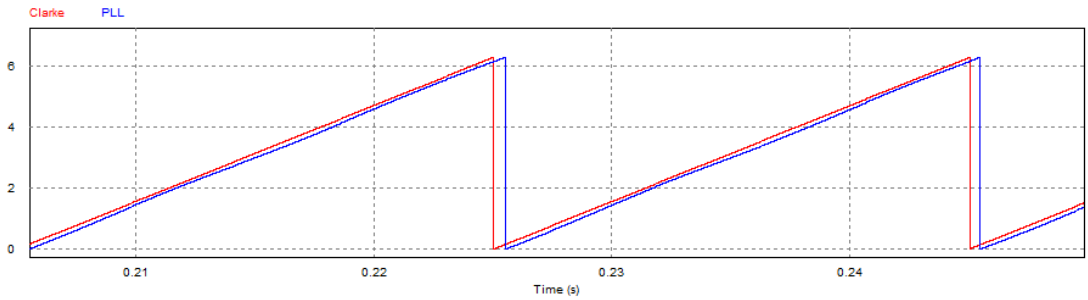


Figura 5.26

Como se puede observar se ha solucionado el problema y ahora la frecuencia es la correcta. Pero hay un pequeño desfase introducido por el filtro pasobanda en alfa-beta a 50Hz para eliminar la componente continua.

Para corregirlo se utilizará un filtro de tipo Notch cuyo retraso es mucho menor, se diseñará para una frecuencia de corte de 50Hz, y esta señal se le restará a la fundamental obteniendo así sólo la componente a 50Hz, filtrando de esta forma la componente continua y el resto de componentes que no interesan que lleguen al PLL.

Filtro Notch

Utilizar un filtro Notch en alfa-beta a 50Hz y restárselo para eliminar todas las componentes que no estén a 50Hz. Como el pasobanda tiene un pequeño desfase y el Notch no, se prueba esta opción.

Se escoge el siguiente filtro digital Notch:

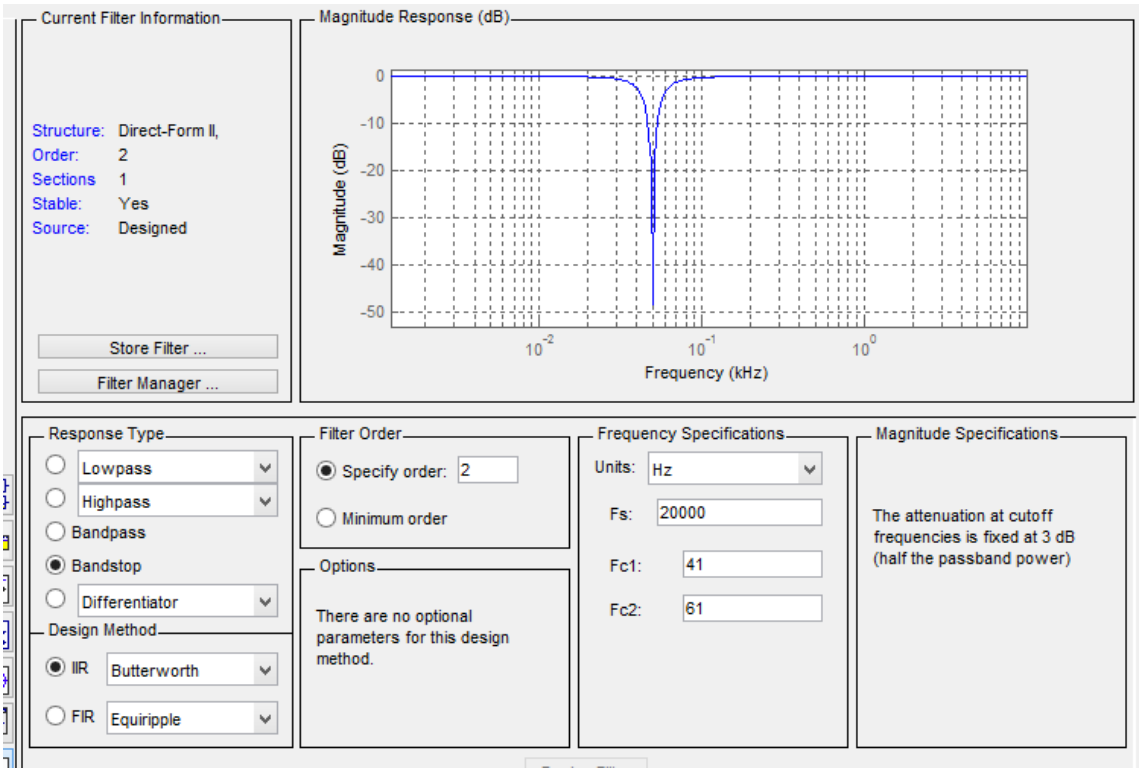


Figura 5.27

En la siguiente imagen se observa cómo el resultado con el filtro Notch presenta menor desfase que con el filtro pasobanda.

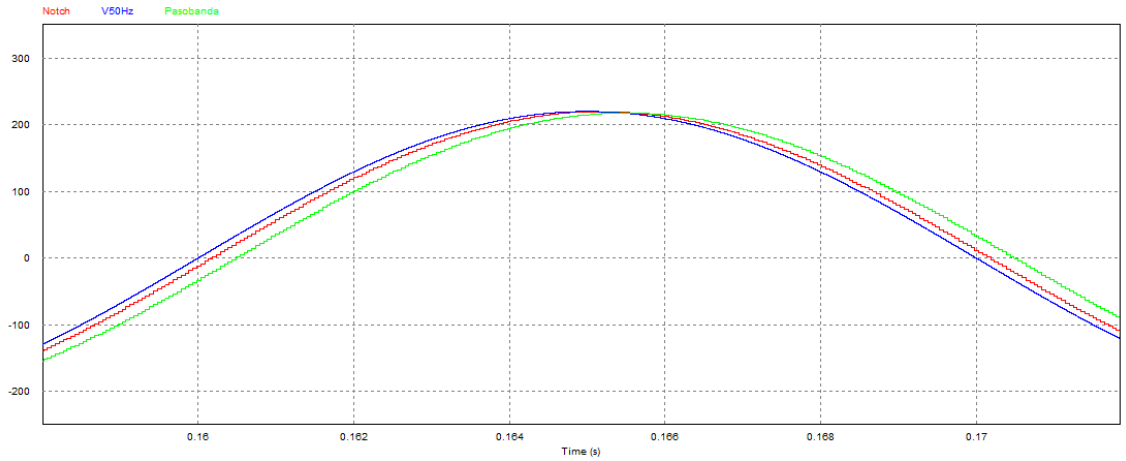


Figura 5.28

Se programa para observar la respuesta del PLL en PSIM. En la siguiente imagen se compara la senoidal real con una señal senoidal generada con el ángulo obtenido por el PLL1 utilizando el filtro pasobanda y con una senoidal utilizando el filtro Notch:

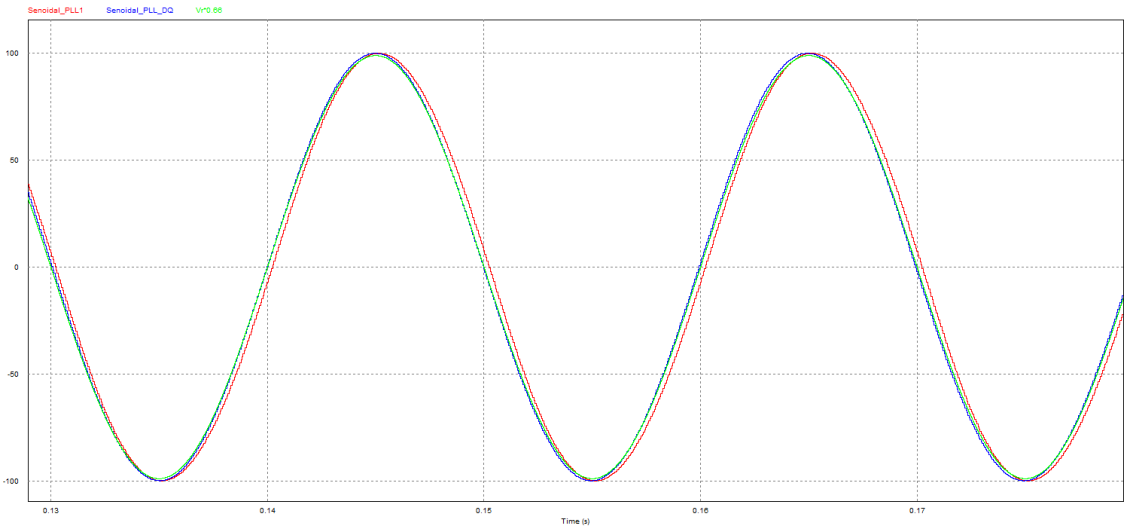


Figura 5.29

Cuando se usaba el pasobanda:

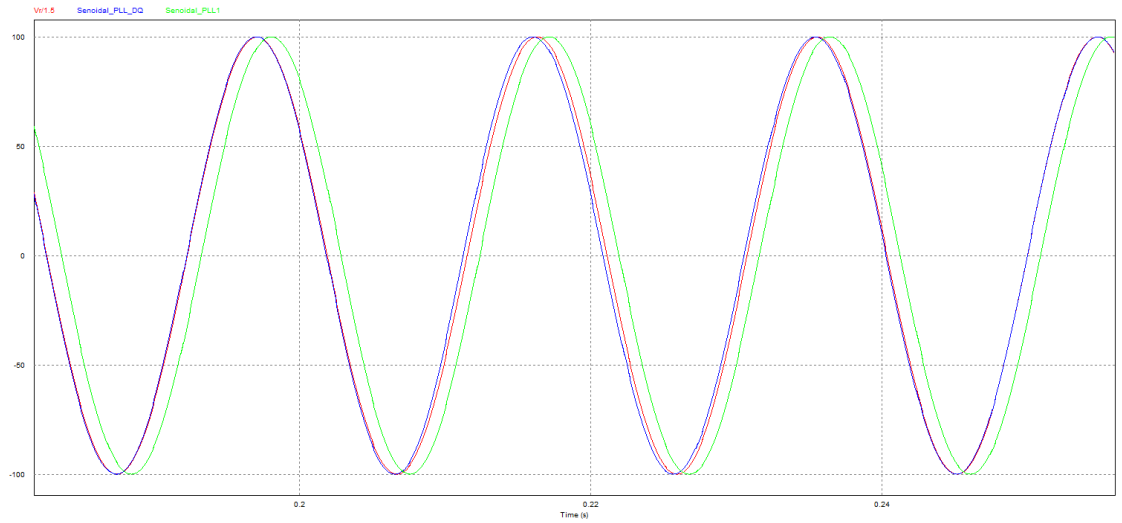


Figura 5.30

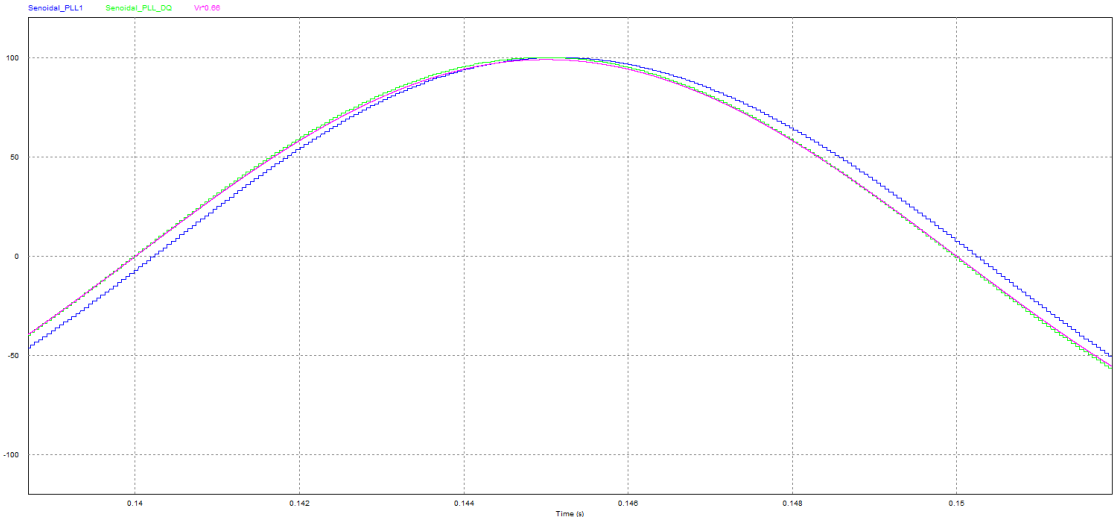


Figura 5.31

Como vemos se mejora el comportamiento respecto al pasobanda.

Por lo tanto el PLL completo queda de la siguiente forma:

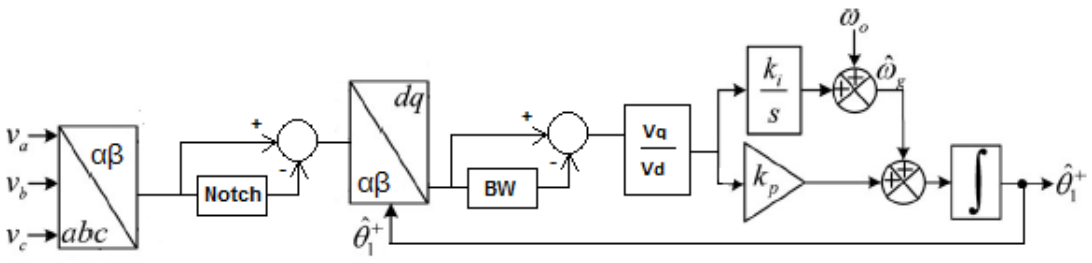


Figura 5.32

Ahora el ángulo obtenido por el PLL no tiene el retraso permanente que introducía el pasobanda:

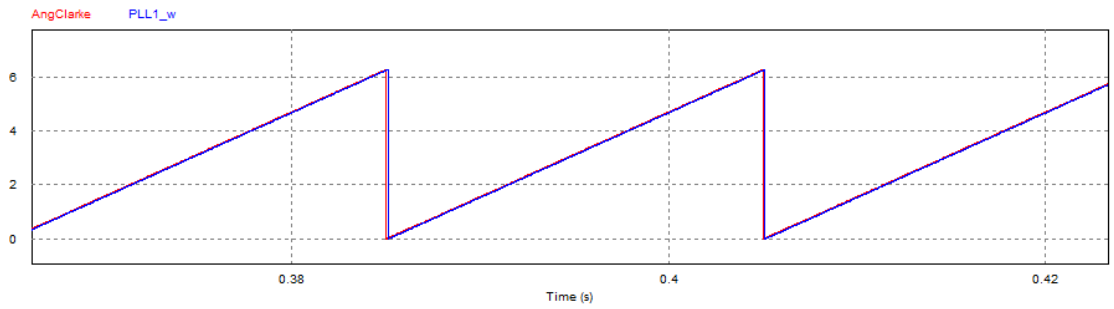


Figura 5.33

Pero al haber diseñado los filtros para una frecuencia de 50Hz si se produce una deriva en frecuencia de la red los filtros pueden introducir un desfase permanente en el ángulo obtenido por el PLL además de no ser capaz de filtrar correctamente los armónicos.

Se comprueba la respuesta del PLL1 ante una red equilibrada a una frecuencia de 56Hz:

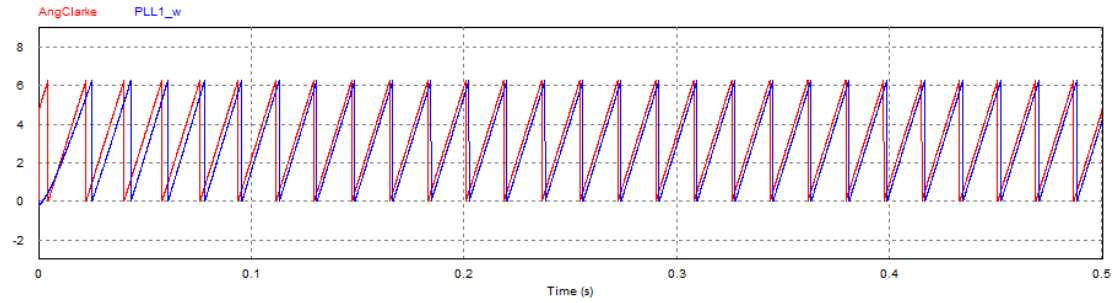


Figura 5.34

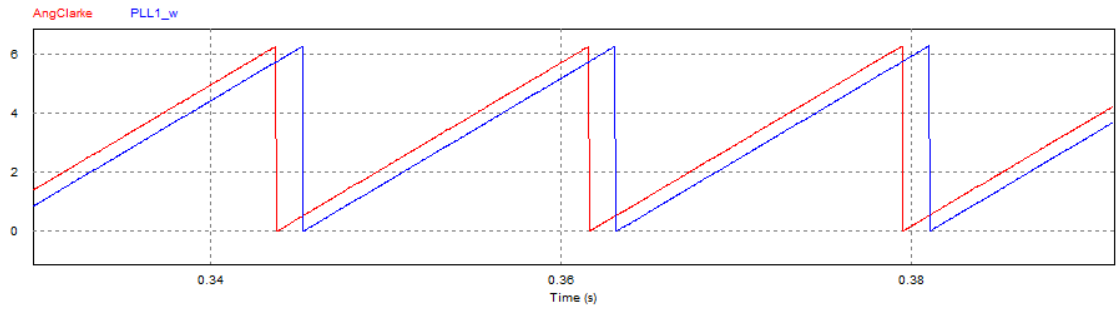


Figura 5.35

Como se suponía los filtros introducen un desfase entre el ángulo obtenido por el PLL1 y el ángulo real.

Para solucionarlo se hace un estudio del filtro Notch a diferentes frecuencias. El filtro Notch utilizado es de segundo orden. Se comprueban sus coeficientes a diferentes frecuencias para observar cuál de ellos es función de esta:

Realimentación de frecuencia para recalcular los coeficientes del filtro Notch y tener mayor amplitud de frecuencia:

Primero se calculan los coeficientes del Notch para diferentes frecuencias de corte para comprobar qué coeficiente cambia y con qué función lo hace:

- 41-61 con fc=50Hz

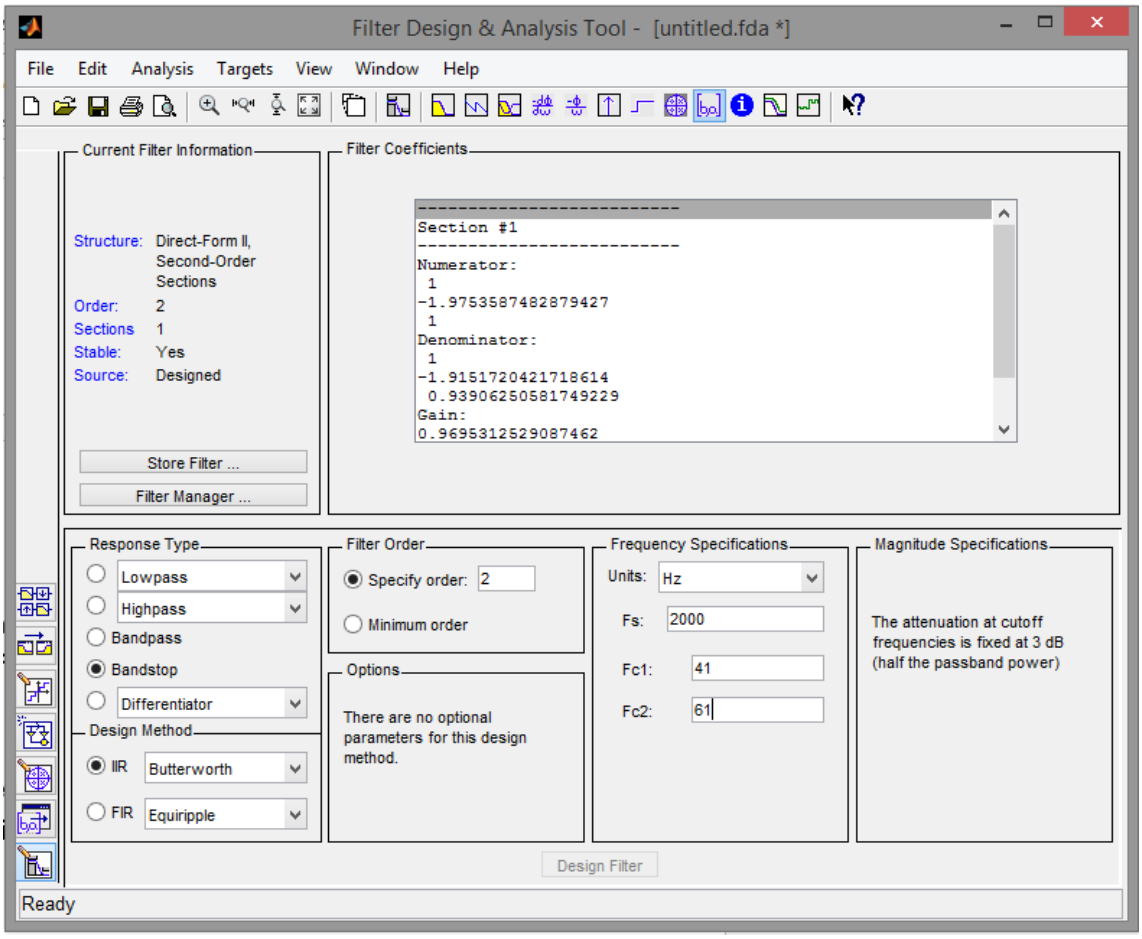


Figura 5.36

- 45-65 con fc=54Hz

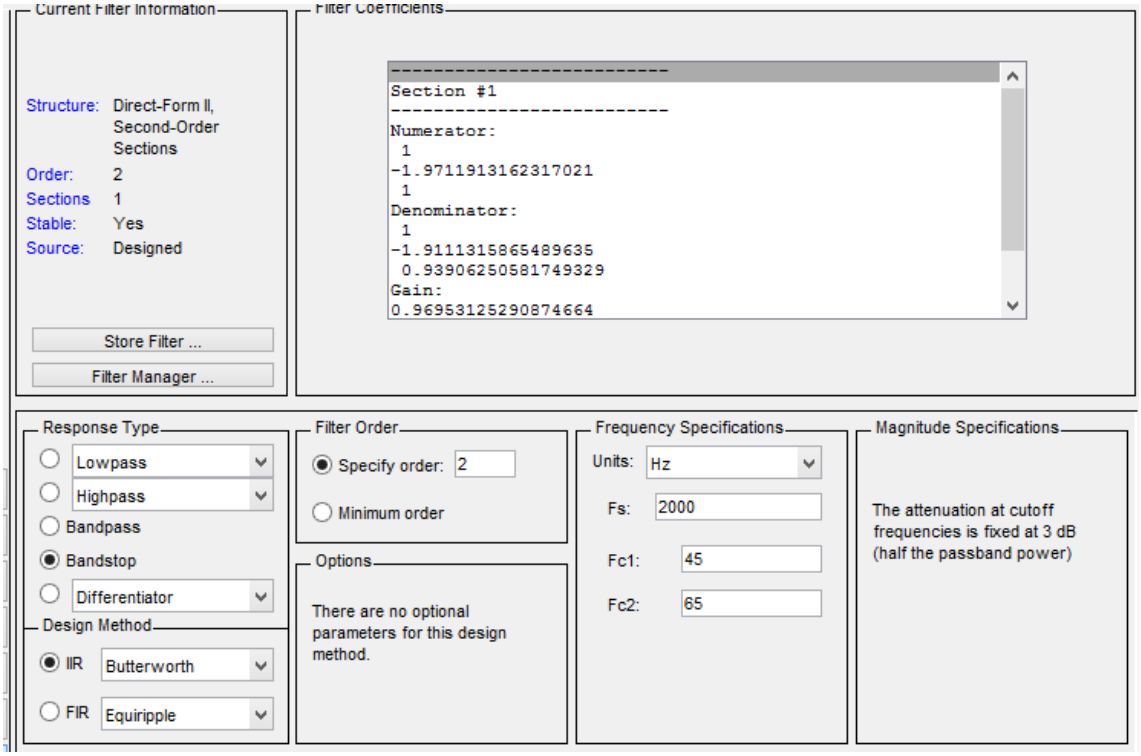


Figura 5.37

- 37-57 con 45.89844Hz

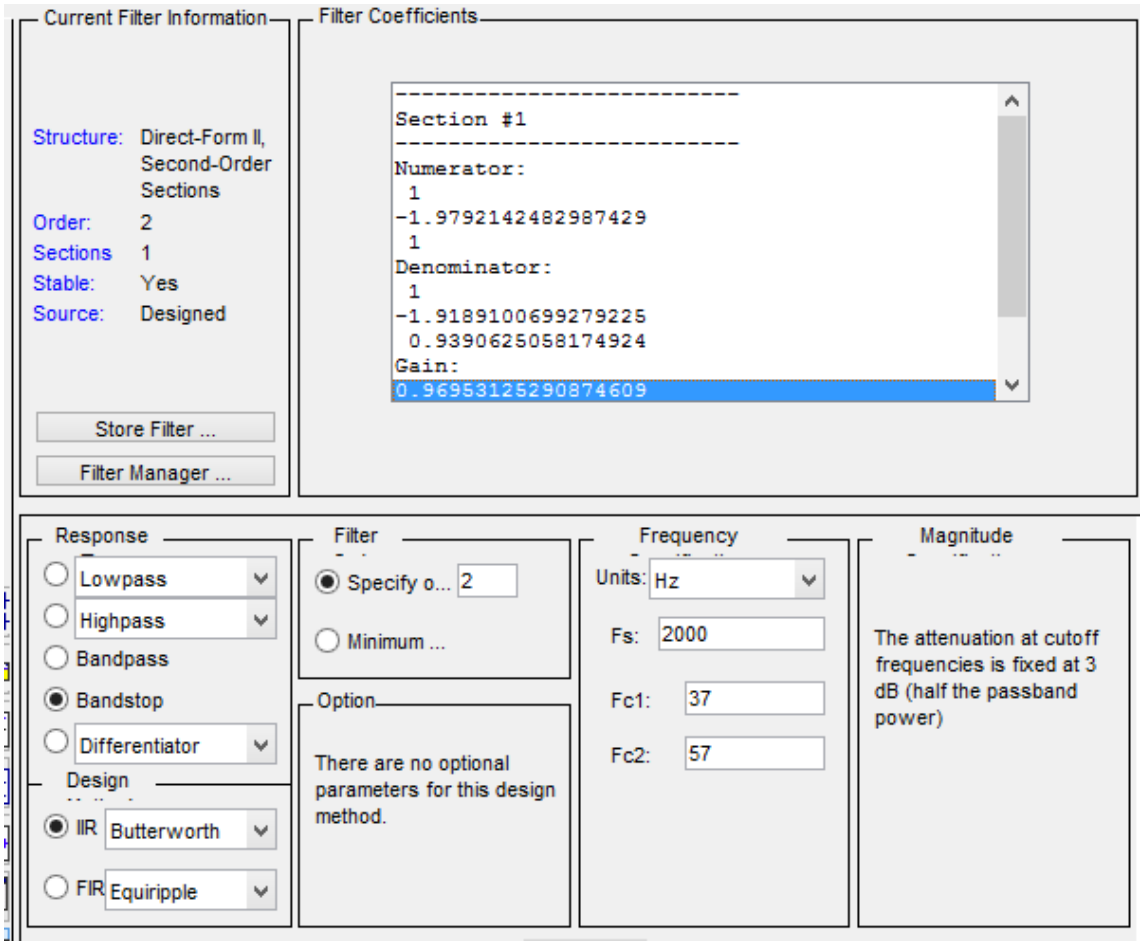


Figura 5.38

	Denominador			Numerador			
Frecuencia	**1**	**2**	**3**	**1**	**2**	**3**	**Ganancia**
50,04883	1	-1,91517204217186	0,93906250581749	1,00000000000000	-1,97535874828794	1,00000000000000	0,969531253
54,07715	1	-1,91113158654896	0,93906250581749	1,00000000000000	-1,97119131623170	1,00000000000000	0,96953125290875
45,89844	1	-1,91891006992792	0,93906250581749	1,00000000000000	-1,97921424829874	1,00000000000000	0,96953125290875

Como vemos sólo cambian los dos coeficientes que multiplican la z-1 y lo hacen de forma lineal:

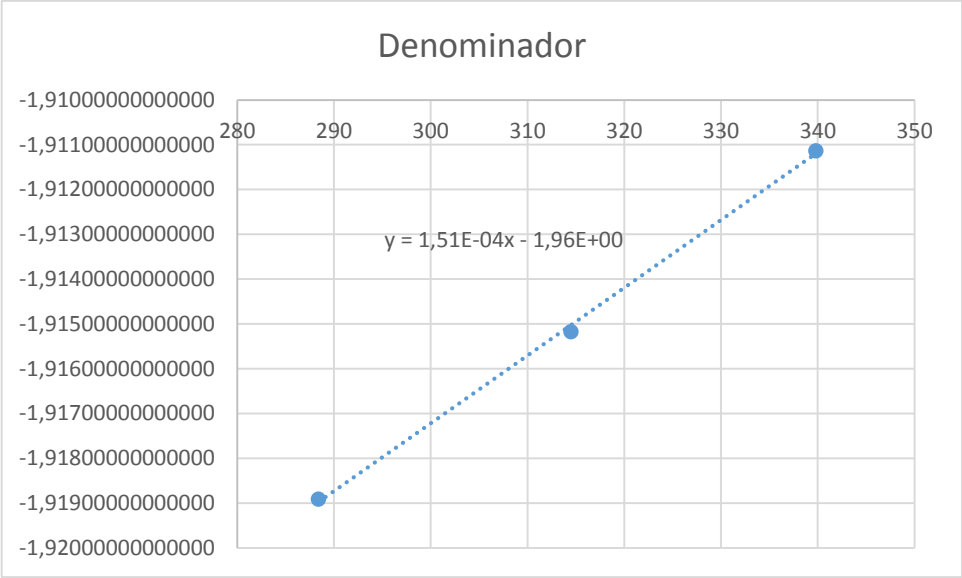


Figura 5.39

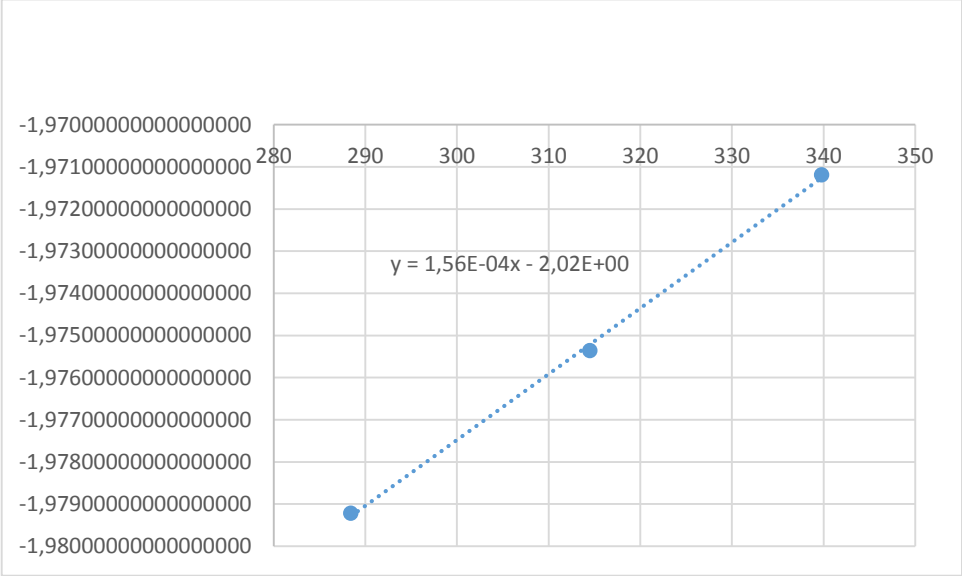


Figura 5.40

MATLAB nos permite generar la función del filtro Notch de una forma diferente que nos puede ayudar a obtener con más exactitud la función que siguen los coeficientes al variar con la frecuencia. Con el siguiente código se obtienen los coeficientes del filtro para una frecuencia de a 40 a 60Hz.

```
>> Fs=2000;
k=0;
for fc=40:60
    k=k+1;
    fl=fc-10;
    fh=fc+10;
    d=fdesign.bandstop('N,F3dB1,F3dB2',2,fl,fh,Fs);
    Filtro=design(d,'butter');
    coef=Filtro.SOS;
    f(k)=fc;
    n1(k)=coef(2);
    n2(k)=coef(5);
end
```

Figura 5.41

N1 corresponde con el numerador que cambia de valor con la frecuencia y n2 con el denominador.

Para obtener la función con la que varía se prueba primero con una de grado 1 y otra de grado 2. Si no se ajusta bien y se necesita mayor precisión se irá aumentado el grado del polinomio.

Con las siguientes líneas de código se representa el numerador y el denominador:

```
figure
plot(f,n1,f,n2)
```

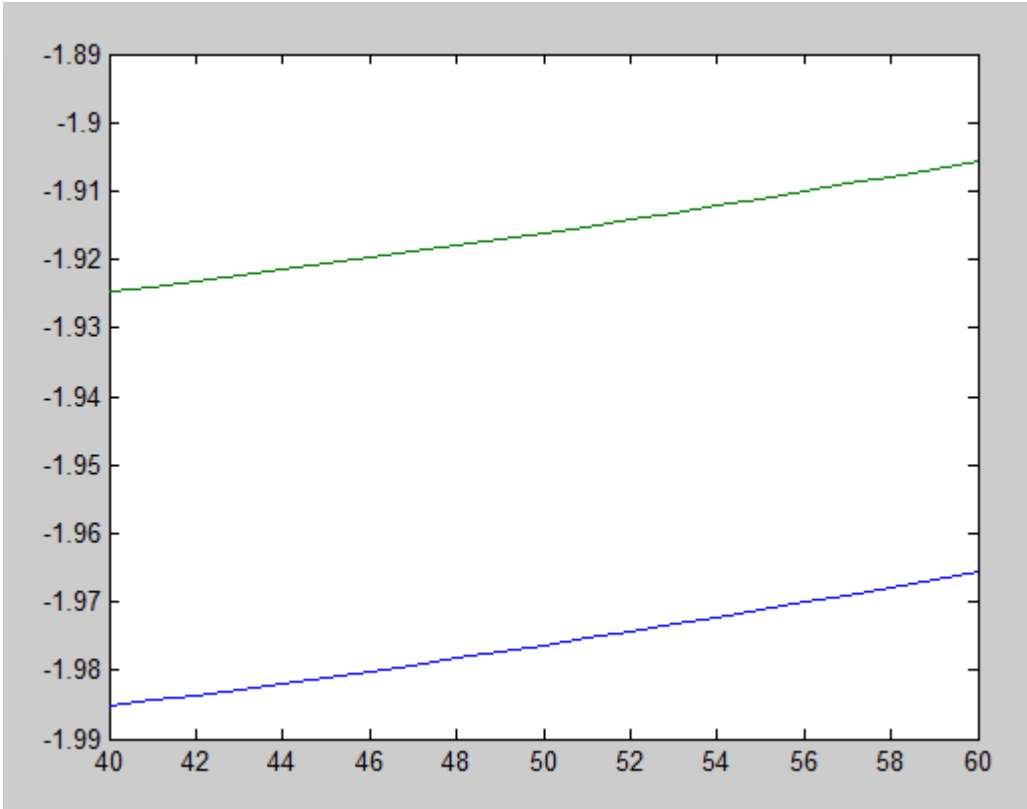


Figura 5.42

A continuación se utiliza un polinomio de orden 1 para el numerador y se representa para ver si se ajusta correctamente:

```
>> q=polyfit(f,n1,1);
figure
plot(f,n1,f,q(1)*f+q(2))
```

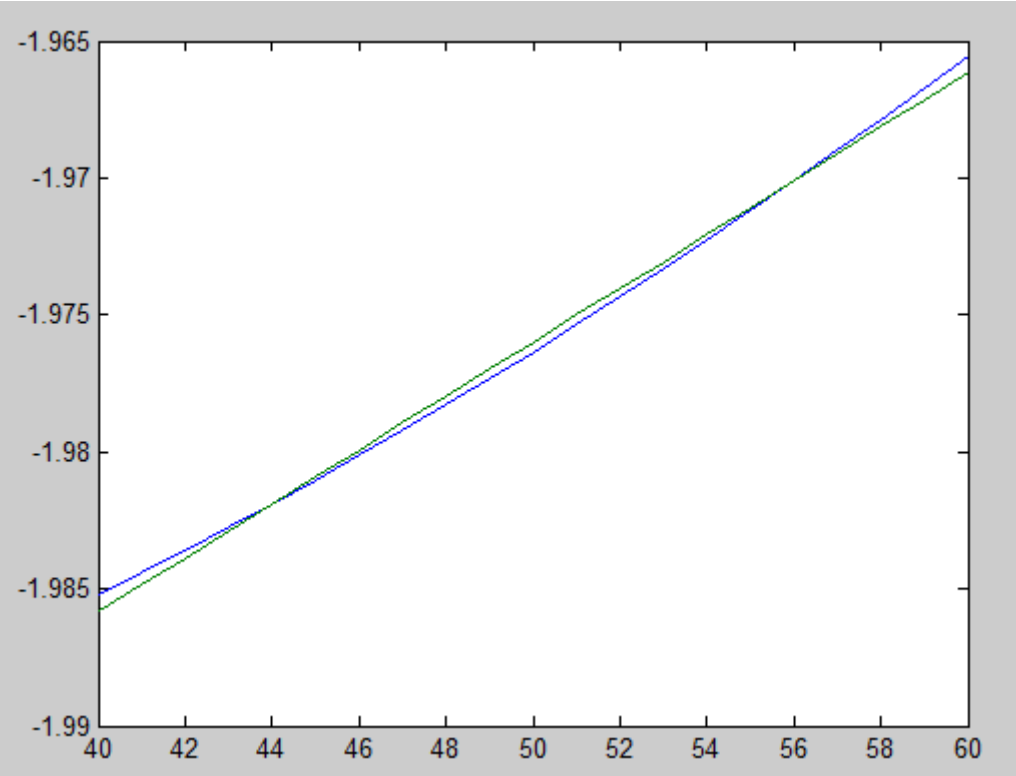


Figura 5.43

Como hay bastante error se prueba utilizando un polinomio de grado 2:

```
>> p=polyfit(f,n1,2);
figure
plot(f,n1,f,p(1)*f.*f+p(2)*f+p(3))
```

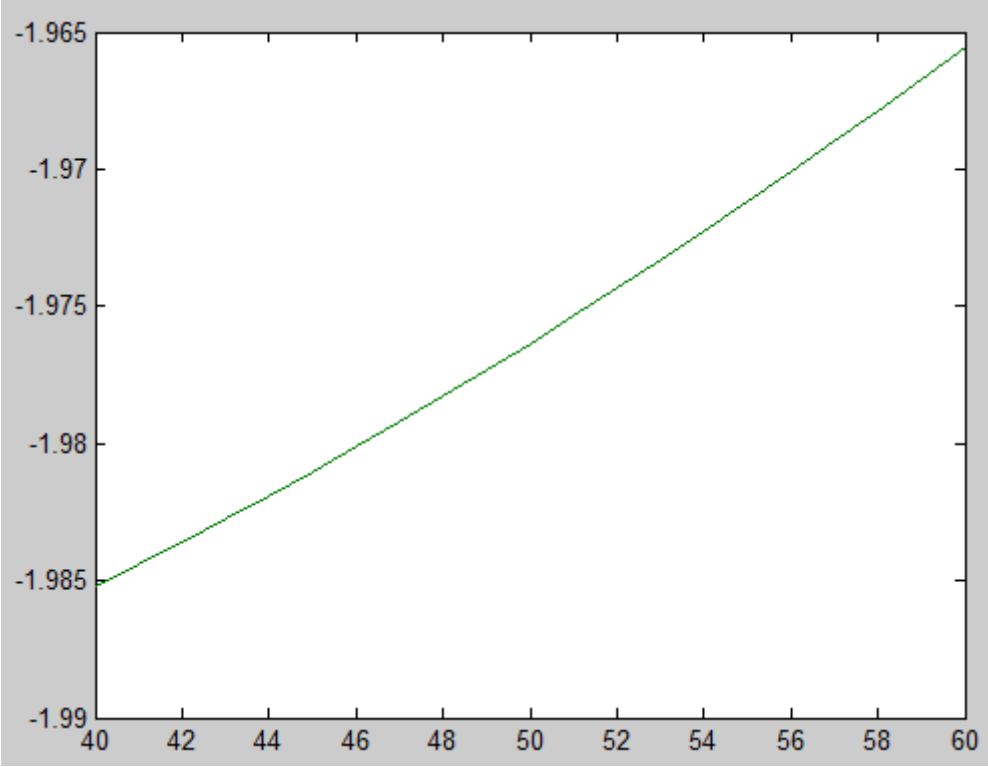


Figura 5.44

El error es mínimo así que se utilizará éste.

9,75215510792633e-06 8,07003109938107e-06 -2,00113577270181

Se hace lo mismo con el denominador.

```
p=polyfit(f,n2,2);
figure
plot(f,n2,f,p(1)*f.*f+p(2)*f+p(3))
```

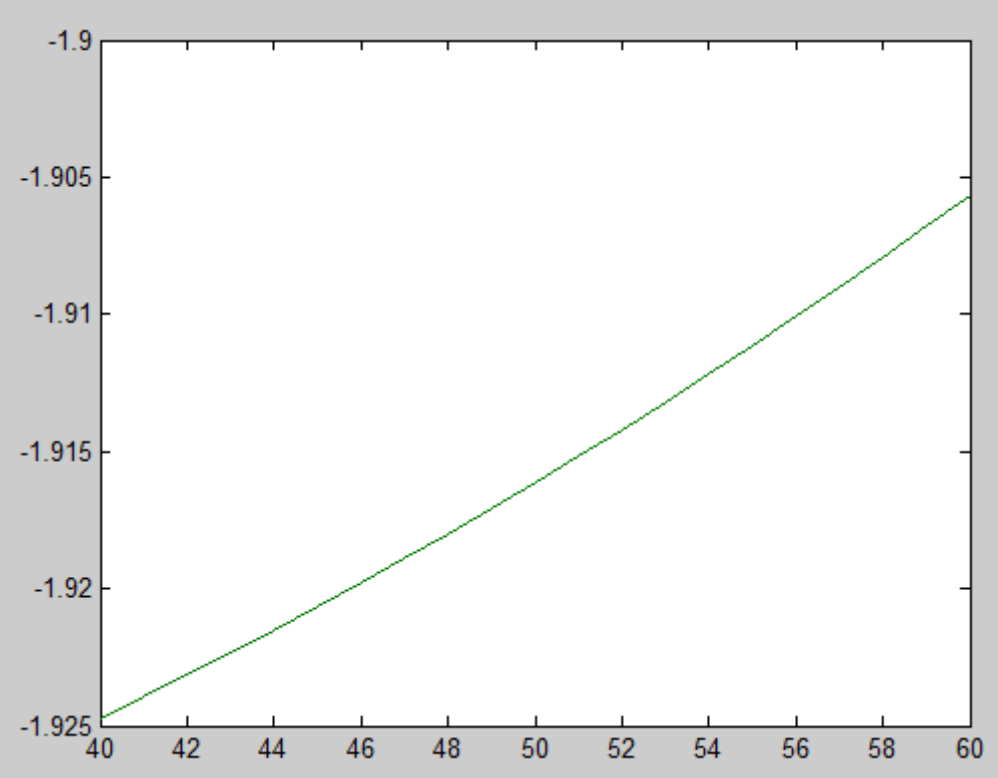


Figura 5.45

9,45501916033719e-06 7,82414736398593e-06 -1,94016367294813

De esta forma obtenemos cómo varían el numerador y el denominador del filtro en función de la frecuencia. Como la frecuencia se obtiene en el algoritmo PLL se puede utilizar para ajustar el filtro a la frecuencia real y, siempre que ésta esté entre 40 y 60Hz, el filtro no producirá desfase alguno en el ángulo obtenido por el PLL y además será capaz de eliminar correctamente las frecuencias no deseadas.

Antes de la programación del filtro Notch se añaden las siguientes líneas:

nun1=0.0000097522*(w_piCCF*w_piCCF/(2*pi*2*pi))+0.00000807*w_piCCF/(2*pi)-2.0011;

denn1=0.000009455*(w_piCCF*w_piCCF/(2*pi*2*pi))+0.0000078241*w_piCCF/(2*pi)-1.9402;

w_piCCF es la frecuencia en rad/s obtenida del algoritmo PLL.

El PLL definitivo tiene la siguiente forma:

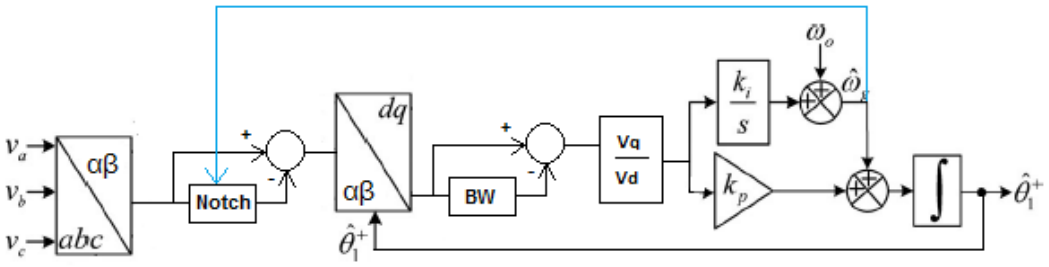


Figura 5.46

Se vuelve a simular una red equilibrada a 56Hz en PSIM y se observa cómo ahora no se produce desfase:

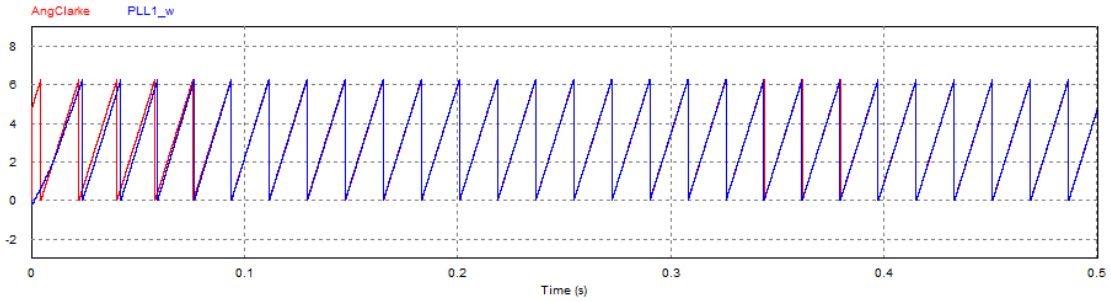


Figura 5.47

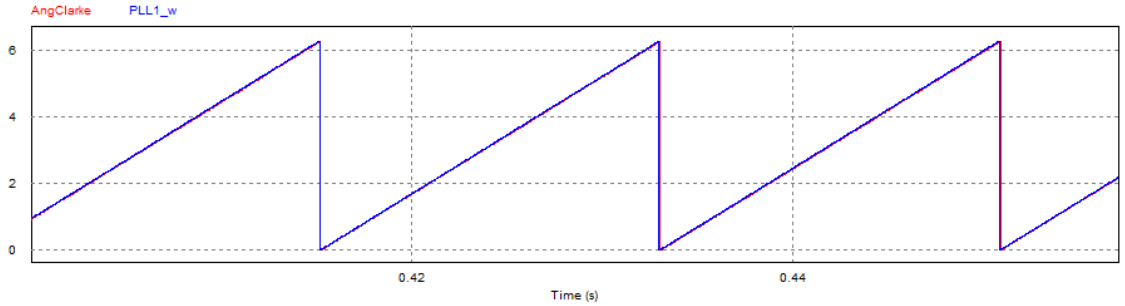


Figura 5.48

A continuación se explica brevemente cómo se ha implementado la programación de los otros dos diseños de PLL con los que se va a comparar

5.2 Diseño de PLL del estado del arte

PLL dqDSC

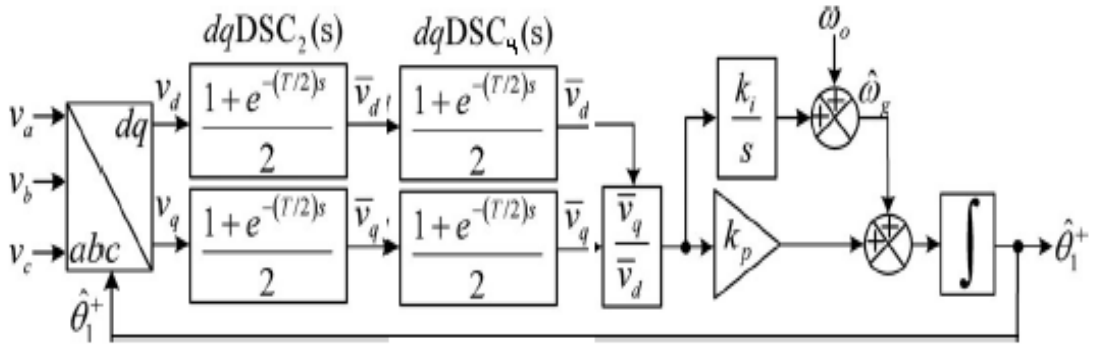


Figura 5.49

La principal dificultad de construir este PLL de forma digital consiste en implementar los filtros DSC. Para ello se utiliza su transformada Z.

$$F(z)=\frac{1+z^{-n}}{2}$$

$$n=\frac{T_{fun}}{4T_m}$$

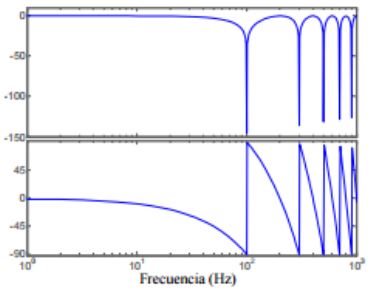


Figura 5.50 Referencia [5]

La Z-1 significa un retraso de un periodo de muestreo. Se va a utilizar un periodo de muestreo de 20KHz para todos los PLL. Como la frecuencia que se pretende eliminar es la de 50Hz y la de 100Hz se obtiene una n=100 y n=200.

Filtro DSC4 para eliminar la componente a 100Hz, prueba:

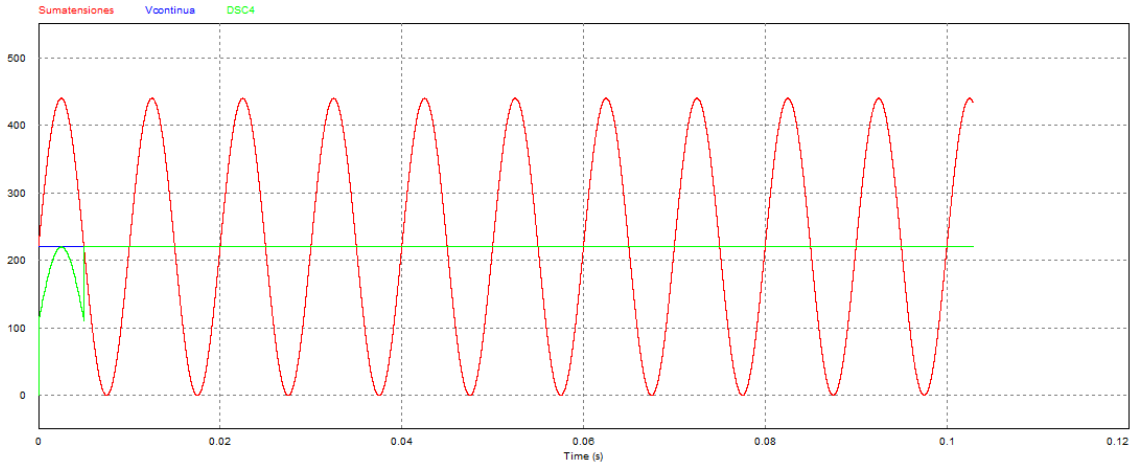


Figura 5.51

PLL CCF

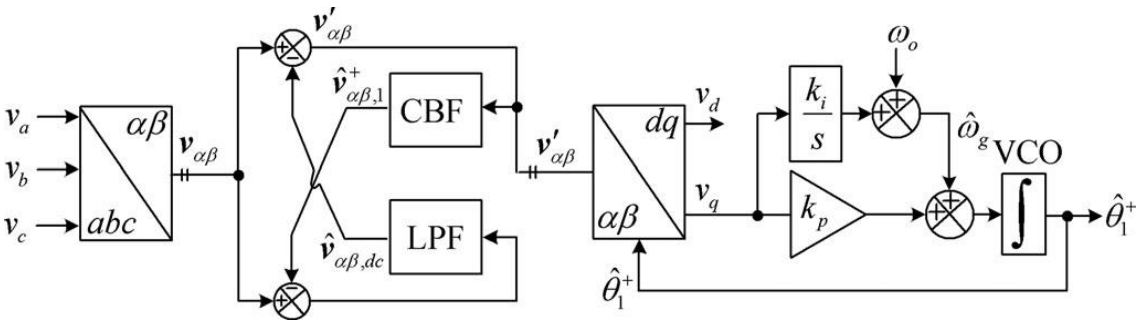


Figura 5.52

Como filtro pasobajo se diseña uno de orden 2 y frecuencia de corte de 3Hz para que sólo pase la componente continua.

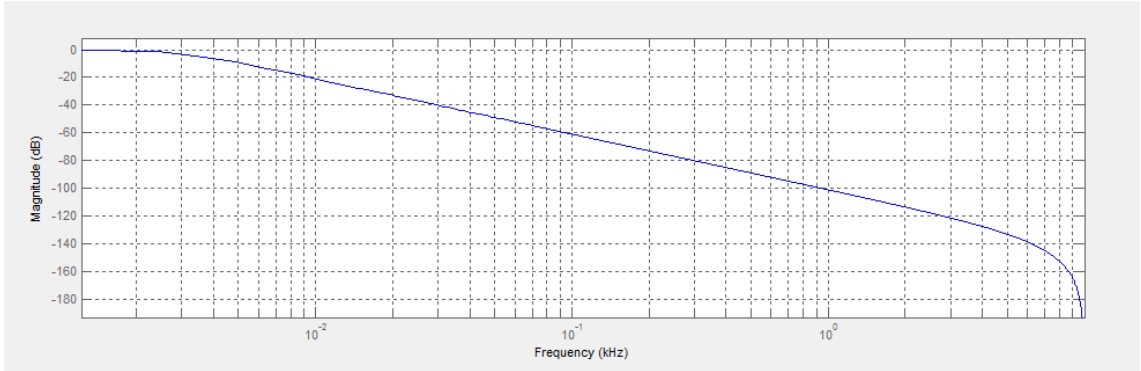


Figura 5.53

```
Section #1
--------------------------
Numerator:
1
2
1
Denominator:
 1
-1.998667135315678
 0.99866802298843449
Gain:
0.00000022191818912819991
--------------------------
Output Gain:
1
```

Figura 5.54

$$\mathrm{CBF}(s) = \frac{\omega_p}{s - j\omega_g + \omega_p}$$

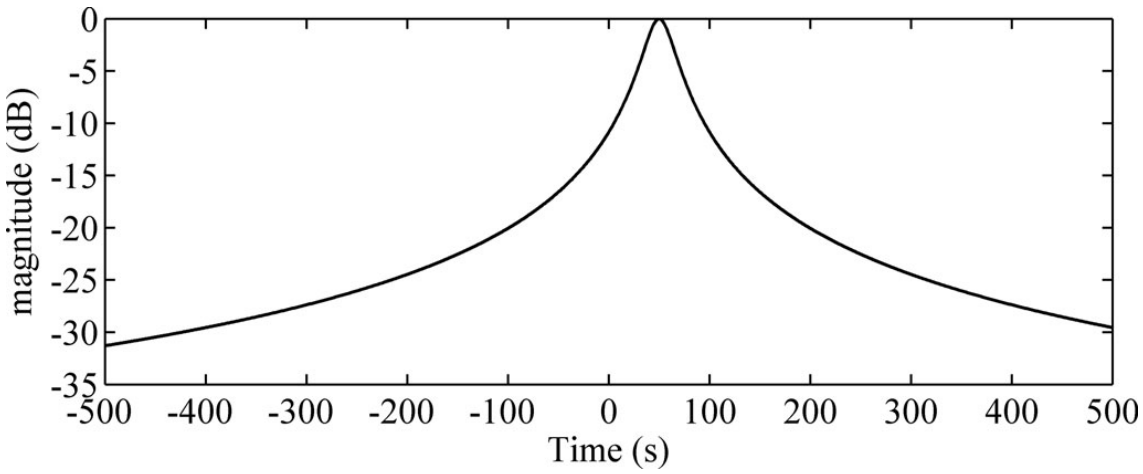


Figura 5.55

Para implementar el CBF:

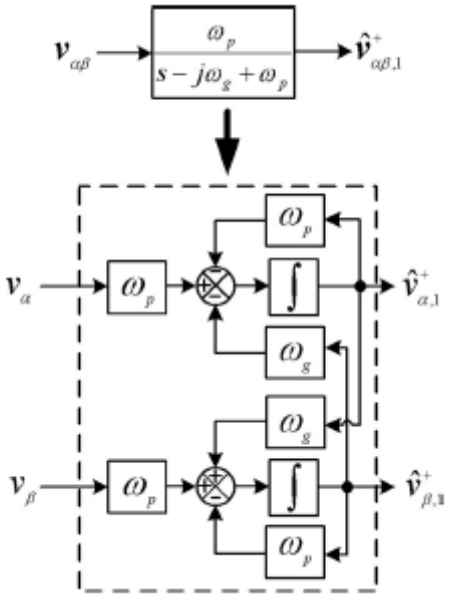


Figura 5.56

6 IMPLEMENTACIÓN EN ARDUINO

6.1 Arduino Due

El Arduino Due es la primera placa de desarrollo de Arduino basado en ARM. Esta placa está basada en un potente microcontrolador ARM CortexM3 de 32bit, programable mediante IDE de Arduino.

El Arduino Due dispone de 54 pines digitales de entrada / salida (de los cuales 12 pueden utilizarse para salidas PWM), 12 entradas analógicas, 4 UARTs (puertas seriales), un reloj de 84 MHz, una conexión USB OTG, 2 DAC (digital a analógico), 2 TWI, un conector de alimentación, un cabezal SPI, un cabezal JTAG, un botón de reinicio y un botón de borrado. También hay algunas características interesantes como DACs, Audio, DMA, una biblioteca multitarea experimental y más.

Para compilar el código para el procesador ARM, se necesita la última versión del IDE de Arduino: v1.5.

A diferencia de otras placas Arduino, la placa Arduino Due funciona a 3.3V. El voltaje máximo que los pines de I/O pueden tolerar es 3.3V. Proporcionar voltajes más altos, como 5V a un pin I/O podría dañar la placa.

6.1.1 Características:

- Microcontrolador: AT91SAM3X8E
- Voltaje de operación: 3.3V
- Voltaje de entrada recomendado: 7-12V
- Voltaje de entrada min/max: 6-20V
- Digital I/O Pins: 54 (de los cuales 12 proveen salida PWM)
- Analog Input Pins: 12
- Analog Outputs Pins: 2
- Corriente total de salida DC en todas las lineas I/O: 130 mA
- Corriente DC para el Pin de 3.3V: 800 mA
- Corriente DC para el Pin de 5V: 800 mA
- Memoria Flash: 512 KB disponibles para aplicaciones del usuario
- SRAM: 96 KB (two banks: 64KB and 32KB)
- Clock Speed: 84 MHz

6.2 Introducción a Arduino

6.2.1 Entorno de programación

Todo programa para Arduino presenta una estructura básica **[3]**.

1. Primera parte: declarar las variables.
2. Segunda parte: configuración de Arduino.

   En este bloque habrá que especificar:

   Qué pines van a ser empleados como entrada y cuáles como salida.

   Las entradas analógicas no hacen falta incluirlas en el setup, puesto que esos pines (A0, A1, A2, A3, A4, A5) solo pueden ser entradas analógicas.
3. Tercera parte: comandos que regirán el comportamiento de Arduino.

6.2.2 Lenguaje de Arduino

El lenguaje del Arduino es una versión reducida y mucho más sencilla de manejar que el lenguaje C. El objetivo de este lenguaje es que se pueda programar de una manera intuitiva concentrándose en lo que se quiere hacer más que en la manera de hacerlo.

Arduino fue desarrollado originalmente en el Interactive Design Institute en Ivrea (Italia) donde los estudiantes estaban familiarizados con un lenguaje llamado Processing. Este lenguaje estaba orientado a estudiantes de arte y diseño y utilizaba un entorno de desarrollo visual e intuitivo en el cual se basó el entorno de desarrollo del Arduino y su lenguaje de programación.

Trabajar con un Arduino consiste fundamentalmente en interactuar con los diferentes puertos de entrada y salida del Arduino. A fin de evitar al programador el engorro y la complejidad de programar estos puertos (ya sean analógicos, digitales o de cualquier otro tipo) el lenguaje de Arduino usa una serie de librerías (de las que no te tienes que preocupar ya que forman parte del lenguaje). Estas librerías permiten programar los pins digitales como puertos de entrada o salida, leer entradas analógicas, controlar servos o encender y apagar motores de continua. La mayor parte de estas librerías de base ("core libraries") forman parte de una macro librería llamada Wiring desarrollada por Hernando Barragán.

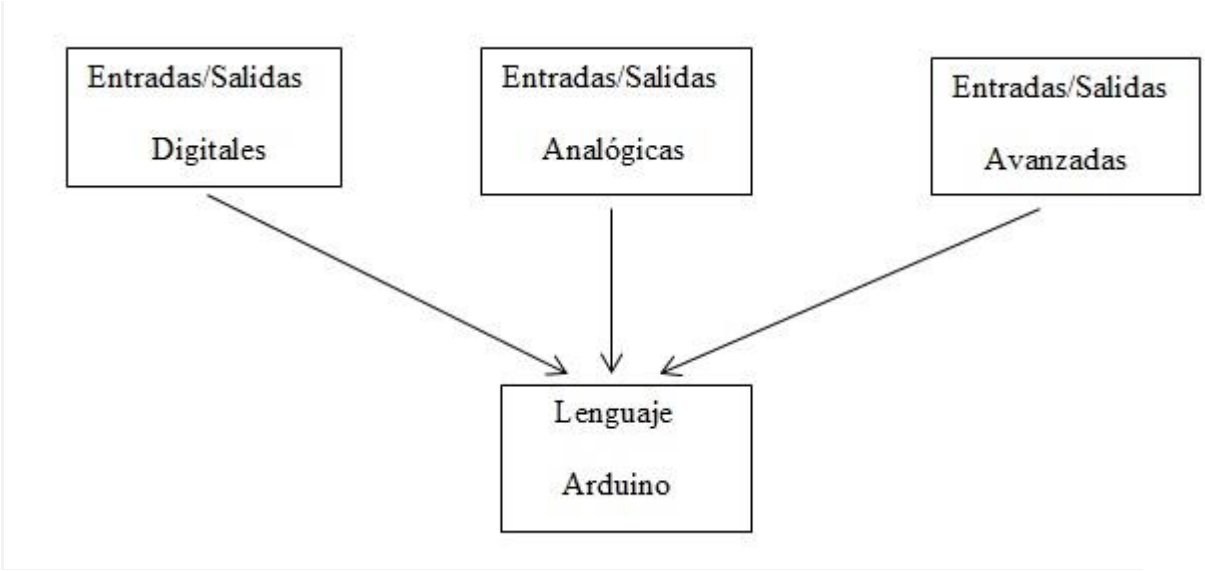


Figura 6.1

Cada vez que un nuevo puerto de entrada o salida es añadido al Arduino, un nuevo conjunto de librerías especializadas en ese puerto es suministrada y mantenida por los desarrolladores del nuevo puerto.

El problema del lenguaje de Arduino es que para funciones simples puede estar utilizando varios pasos y la velocidad se ve reducida.

6.2.3 Funciones y tipos de variables utilizadas

A continuación se citan y se explican brevemente las funciones utilizadas en la programación de los diferentes PLLs:

- analogWriteResolution(12): define la resolución en bytes de la escritura analógica.
- analogReadResolution(12): define la resolución en bytes de la lectura analógica.
- atan2(y,x): devuelve el ángulo en radianes entre –pi y +pi.
- Micros(): devuelve el número de microsegundos desde que ha empezado a ejecutarse el programa. El número se resetea aproximadamente a los 70 minutos.
- Funciones para ampliar las salidas analógicas: puesto que ARDUINO DUE sólo tiene dos salidas puramente analógicas se pueden utilizar más filtrando una señal de salida PWM. Para ello se emplea la función diseñada por la UPNA **[4]** que genera una señal PWM a través de 3 canales de un periférico del microcontrolador y lo hace con un ciclo constante que se puede definir:
    - Pwm3_setup (frecuencia, deadtime, int_sync): inicializa los canales 0, 1 y 2 del módulo de PWM y los configura con la frecuencia (Hz) y el tiempo de muestreo (microsegundos) especificados.
    - Pwm3_start: arranca el módulo y comienza a generar las órdenes de conmutación.
    - Pwm3_setDutyCycles (CicloA, CicloB, Cicloc): fija el ciclo de trabajo (entre 0 y 1) de las tres fases.

En cuanto al tipo de variables, se utiliza para las variables que manejan datos leídos de las entradas analógicas y las operaciones que se realizan con ellos así como los dato de salida (ángulo, frecuencia…) la variable float que almacena un número decimal con un rango entre -3.4028235 · 1038 y 3.4028235 · 1038 (4 bytes) ya que no es necesario ahorrar en memoria para el programa y con la variable float se obtiene la máxima precisión posible.
También se utiliza el tipo de variable int que almacena un número entero entre -32769 y 32767 (2 bytes) siempre que la variable se utilice como contador.

6.3 Programación de los PLL

En la primera parte del programa se declaran las variables que se van a utilizar.

Las variables declaradas son diferentes para cada uno de los 3 PLLs pero para los 3 se presentan las variables para las lecturas analógicas de la tensión, variables utilizadas en los filtros, y variables utilizadas en el algoritmo PLL y las variables de salida como el ángulo obtenido o la frecuencia.

En las siguientes líneas de programación se aprecia la declaración de variables para el PLLdq:

```
/**Declaración de varibles**/
double Suma,pi=3.14159265358979323846264338327950288419716939937751;
```

```
/**PLL**/
double Va, Vb,AngClarke, Senoidal1, Senoidal2;
double VaFiltrada, VbFiltrada, prueba=0;
double VdCCF, VqCCF, w_2CCF;
double VqFiltrada, VdFiltrada;
double Kpl2=1147.49724, Tnl2=0.00173685;
double Kpl3=16.3375182, Tnl3=0.0928889;
double Kpl4=30.3375182, Tnl4=0.0528889;
double Kpl5=75.3375182, Tnl5=0.0228889;
double Kpl=55.8944216,Tnl=0.03083307;
double KplR=54.81747361, TnlR=0.028408946;
double Tm=1./2000;
double Vd3, Vq3, w3,w4, VCCF3, error_acumCCF3, w_piCCF3,w_1CCF3,Division,Seno,Coseno,Angulo;
long Tmuestreo=500, Pasado=0, Ahora, Cambio;
unsigned char ciclo=0;
int Vr, Vs, Vt;
double Vd_red,Vq_red,principio=0;
double Vdq,Vq_pu,error_acum,w_pi,w_1,w_2,w=2*pi, w_dq;

/**Variables PLL DQ**/
double Vd_red3,Vd_red4, Vq_red3, w_2DQ=0, sum, sum2, lleno, lleno2, Vbusmed, Vbusmed2, VDQ, error_acumDQ;
double w_piDQ, w_1DQ;

/**Variables filtro PLLdq**/
int j, k, e, q;
double suma, suma1,suma2,suma3,suma4,Vdfiltrada1,Vdfiltrada2,Vqfiltrada1,Vqfiltrada2;
double Vent[20], Vent2[20], Vent5[10],Vent6[10];

/**Coeficientes Filtros**/
```

```
double xa2,xa1,xa0,ya2,ya1,ya0;
double ca2,ca1,ca0,ua2,ua1,ua0;
double va2,va1,va0,oa2,oa1,oa0;
double ba2,ba1,ba0,pa2,pa1,pa0;

double xb2,xb1,xb0,yb2,yb1,yb0;
double cb2,cb1,cb0,ub2,ub1,ub0;
double vb2,vb1,vb0,ob2,ob1,ob0;
double bb2,bb1,bb0,pb2,pb1,pb0;

double xq2,xq1,xq0,yq2,yq1,yq0;
double cq2,cq1,cq0,uq2,uq1,uq0;
double vq2,vq1,vq0,oq2,oq1,oq0;
double bq2,bq1,bq0,pq2,pq1,pq0;

double xd2,xd1,xd0,yd2,yd1,yd0;
double cd2,cd1,cd0,ud2,ud1,ud0;
double vd2,vd1,vd0,od2,od1,od;
double bd2,bd1,bd0,pd2,pd1,pd0;

/**Pasobajo a 3Hz**/
double num2=1,num1=2,num0=1,den2=1,den1=-
1.998667135315678,den0=0.99866802298843449,ganancia=0.00000022191818912819991;

/**Señal de referencia para calcular el error)**/
long double Frecuencia, Clarkeantiguo,SenoidalError;
long double correcto, FrecuenciaReal,AnguloReal2, F,FrecuenciaReal2,
AnguloReal,AnguloReal3;
unsigned long Suma5, An, An2, suma12, con;
long double Continua, VrFiltrada,tiempo6,tiempo2, tiempoFrecuencia, tiempo5;
unsigned long  Suma7,reinicio;
```

En la segunda parte del programa se configura el Arduino: entradas, salidas y resolución de las lecturas analógicas y de las salidas.

Las entradas analógicas no hacen falta incluirlas en el setup, puesto que esos pines (A0-A11) solo pueden ser entradas analógicas.

Las salidas analógicas que se utilizan son las 2 que tiene el Arduino Due capaces de dar una señal analógica pura y no una señal PWM, así que tampoco es necesario incluirlas en el setup puesto que sólo pueden ser utilizadas de esa forma. Son los pines de salida DAC0 Y DAC1.

Se incluye la salida digital del pin 13 para usarlo como flag para calcular el tiempo que tarda en ejecutarse todo el programa o cualquier tiempo que se necesite dentro del programa poniendo la salida en nivel alto y bajo y midiendo su frecuencia.

En cuanto a la resolución, Arduino Due permite utilizar una resolución de hasta 12 bits en las lecturas de señales analógicas y en las salidas analógicas. Pero esto hay que configurarlo porque si no por defecto utiliza sólo 10 bits. Se configura mediante la función analogWriteResolution(12) y analogReadResolution(12). De esta forma conseguimos la máxima resolución posible. Las variables asignadas a la entrada podrán tomar un valor de entre 0 y 4096.

El siguiente código es general para los programas de los 3 PLL:

```
/**Definición de entradas**/
/**Defino la salida 13 digital**/
  pinMode(13, OUTPUT);
  analogWriteResolution(12); // Define resolución del DAC
analogReadResolution(12); // Define resolución de la entrada analógica;
pwm3_setup(10000,1,0);
pwm3_start();
```

La tercera parte de la programación es donde se encuentra el programa principal y se repite en bucle continuamente.

Se irá explicando la programación del PLL1 parte por parte.

Primera parte: lectura de las entradas e inicialización

Al Arduino entra una tensión senoidal de entre 0 y 3.3V lo que se traduce en un valor de entre 0 y 4096. Para centrar en 0 la senoidal se le resta el offset a la tensión que es un valor de 1.65V que se traducen en 2048. De esta forma las variables Vr, Vs y Vt toman valores entre -2048 y +2048.

Es necesario que el tiempo de muestreo sea constante porque se ha diseñado el controlador para un tiempo de muestreo determinado. En primer lugar se utilizó Tm=20KHz ya que el reloj interno del Arduino Due es de 84MHz y se creía que sería capaz de ejecutar el programa a una

velocidad mayor de 50 microsegundos. Pero se comprobó que el PLL funcionaba de manera incorrecta. Tras estudiar el caso y utilizar el pin digital 13 como flag que se poía a 1 al principio del programa y a 0 cuando terminaba de ejecutarlo se comprobó que el tiempo medio era de unos 400 microsegundos. El tiempo de muestreo no puede llegar a 20KHz porque Arduino no es capaz de ejecutar el programa entero a esa velocidad. Por lo tanto se modificó a 2KHz, cambiando los parámetros del controlador y los coeficientes de los filtros digitales utilizados que varían en función de la frecuencia de muestreo.

Para mantener constante la frecuencia de muestreo se utilizó las siguientes líneas de código:

```
Ahora=micros();
Cambio=Ahora-Pasado;
if(Cambio>=Tmuestreo)
{
Vr=analogRead(A0)-2048;
Vs=analogRead(A1)-2048;
Vt=analogRead(A2)-2048;
```

La variable Ahora tiene el valor de la variable de Arduino micros que va tomando el valor de los microsegundos que han pasado desde que se ha producido la inicialización del Arduino o se ha pulsado el Reset.

La variable Pasado toma el valor de la variable Ahora al final del programa. De esta forma si el programa tarda en ejecutarse 300 microsegundos la resta de Ahora menos Pasado, la variable Cambio tomará este valor. Tmuestreo es una variable constante que toma el valor de 500, ya que una frecuencia de 2KHz son 500 microsegundos.

Sólo cuando la variable Cambio es igual al tiempo de muestreo significa que han pasado 500 microsegundos desde la anterior lectura de datos y el programa pasa a ejecutarse. Mientras la variable Cambio sea menor a los 500 microsegundos el resto del programa no se ejecuta ni se lee el valor de la tensión de entrada.

Segunda parte: transformada de Clarke

Para la programación del PLL1 y del PLLccf se necesita hacer la transformada de Clarke ya que los filtros para eliminar continua actúan en alfa-beta. Para el PLLdq también se realiza la transformada ya que será necesario para otras cosas que se verán más adelante.

Se utiliza la transformada de Clarke con convenio americano para hacerlo invariante en tensión.

$$\begin{bmatrix} u_\alpha \\ u_\beta \\ u_0 \end{bmatrix} = C \begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix} \begin{bmatrix} u_a \\ u_b \\ u_c \end{bmatrix}$$

Figura 6.2

También se obtiene el ángulo del vector tensión realizando la arcotangente de la proyección de la tensión en alfa y beta. Para ello se utiliza la función atan2 que devuelve el ángulo en radianes entre –pi y +pi.

```
/**Cálculo del ángulo mediante Clarke para comprobación del PLL**/


Va=(2./3)*(Vr-0.5*Vs-0.5*Vt);

Vb=(2./3)*((sqrt(3)/2)*Vs-(sqrt(3)/2)*Vt);

AngClarke=atan2(Vb,Va);


if(AngClarke<0)

{

AngClarke=AngClarke+2*pi;

}
```

Tercera parte: filtro para la eliminación de componente continua

En este apartado se programa el filtro para eliminar la componente continua de cada PLL. En el caso del PLL1 es un filtro Notch de frecuencia variante que no deja pasar la componente a la frecuencia de la red. Esto se resta a la señal original y de esta forma la señal final queda sin todo lo que no esté a 50Hz (o la frecuencia de la red en cada momento en un rango de 40-60Hz).

Los filtros digitales programados son de segundo orden para que el coste computacional sea pequeño. Por ello su función de transferencia sólo tiene z-2, z-1 y 1. Lo que significa que el máximo retraso es de 2 periodos de muestreo. Al estar trabajando en el sistema de coordenadas alfa-beta se pasan por el filtro ambas componentes por lo que se repite el filtro 2 veces.

Para el PLL1 además es de frecuencia variable para que esté sintonizado siempre a la frecuencia de la red. Esto se consigue con dos líneas de código previas a al filtro que adaptan los coeficientes de z-1 según un polinomio de segundo orden que depende de la frecuencia obtenido anteriormente.

Para los retrasos de los filtros al ser el mayor de 2 periodos de muestreo se utilizan variables que toman el valor de la tensión en el periodo de muestreo anterior. Si los retrasos fueran mayores sería necesario crear una tabla de valores para simplificar la programación. Esto sucede con el filtro DSC para el PLLdq que se puede ver en el ANEXO.

```
/**Filtro notch a 50Hz para elimianar frecuencia continua. Orden 2**/

nun1=0.0000097522*(w_piCCF*w_piCCF/(2*pi*2*pi))+0.00000807*w_piCCF/(2*pi)-2.0011;
denn1=0.000009455*(w_piCCF*w_piCCF/(2*pi*2*pi))+0.0000078241*w_piCCF/(2*pi)-1.9402;
/**Va**/
cna2=cna1;
cna1=cna0;
cna0=Va;
una2=una1;
una1=una0;
una0=ganancian2*(nun0*cna2+nun1*cna1+nun2*cna0)-denn1*una1-denn0*una2;
VaFiltrada=Va-una0;
/**Vb**/
cnb2=cnb1;
cnb1=cnb0;
cnb0=Vb;
unb2=unb1;
unb1=unb0;
unb0=ganancian2*(nun0*cnb2+nun1*cnb1+nun2*cnb0)-denn1*unb1-denn0*unb2;
VbFiltrada=Vb-unb0;
```

Cuarta parte: transformada de Park

Es necesario realizar una segunda transformación desde el sistema de ejes fijos a-b a un sistema de ejes rotatorio d-q, manteniendo la misma componente nula. Dicho de otro modo, debe pasarse del sistema de coordenadasa-b-0, a otro rotatorio de coordenadas dq-0.

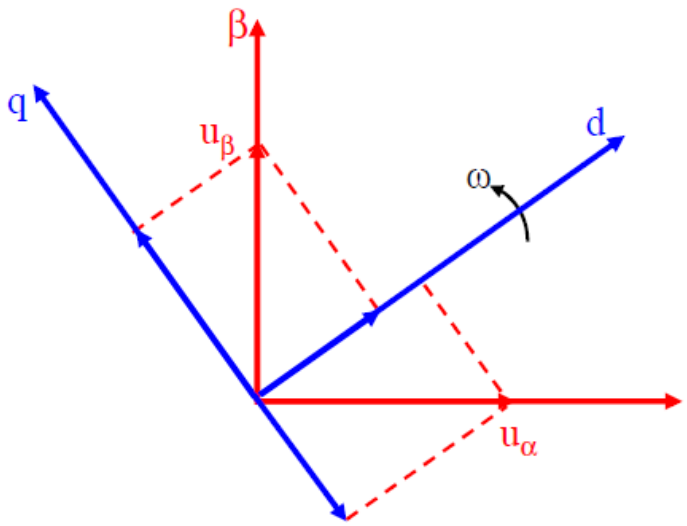


Figura 6.3

$$\begin{bmatrix} u_d \\ u_q \end{bmatrix} = C_{rot} \begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \begin{bmatrix} \cos\omega t & sen\omega t \\ -sen\omega t & \cos\omega t \end{bmatrix} \begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix}$$

Figura 6.4

```
/**Transformada alfa-beta a dq**/
VdCCF=cos(w_2CCF)*VaFiltrada+sin(w_2CCF)*VbFiltrada;
VqCCF=-sin(w_2CCF)*VaFiltrada+cos(w_2CCF)*VbFiltrada;
```

Quinta parte: filtro para eliminar desequilibrios

En el eje de coordenadas dq se pueden eliminar los desequilibrios ya que estos aparecen al doble de la frecuencia de la red. Para ello cada PLL utiliza diferentes filtros. El PLL1 utiliza un filtro pasobanda de segundo orden que se programa de forma idéntica al filtro Notch pero con diferentes coeficientes. El filtro pasobanda sintonizado a 100Hz deja pasar la componente del desequilibrio y luego se resta a la señal original para obtener una señal sin componentes debidos a desequilibrios. En este caso también es necesario hacer pasar cada componente (en d y en q) por el filtro.

```
/**Filtro pasobanda a 100Hz y resta**/
/**Vd**/
xd2=xd1;
```

```
xd1=xd0;

xd0=VdCCF;

yd2=yd1;

yd1=VdBanda;

VdBanda=ganancia9*(numerad0*xd2+numerad1*xd1+numerad2*xd0)-denomin1*yd1-
denomin0*yd2;

VdFiltrada=VdCCF-VdBanda;

/**Vq**/

xq2=xq1;

xq1=xq0;

xq0=VqCCF;

yq2=yq1;

yq1=VqBanda;

VqBanda=ganancia9*(numerad0*xq2+numerad1*xq1+numerad2*xq0)-denomin1*yq1-
denomin0*yq2;

VqFiltrada=VqCCF-VqBanda;
```

Sexta parte: algoritmo PLL

Una vez eliminadas las componentes debidas a armónicos que no nos interesan se programa el algoritmo para obtener el ángulo. Para ello se normaliza la tensión realizando la división Vq/Vd y se hace la Vq=0 para alinear el eje d con el vector de tensión.

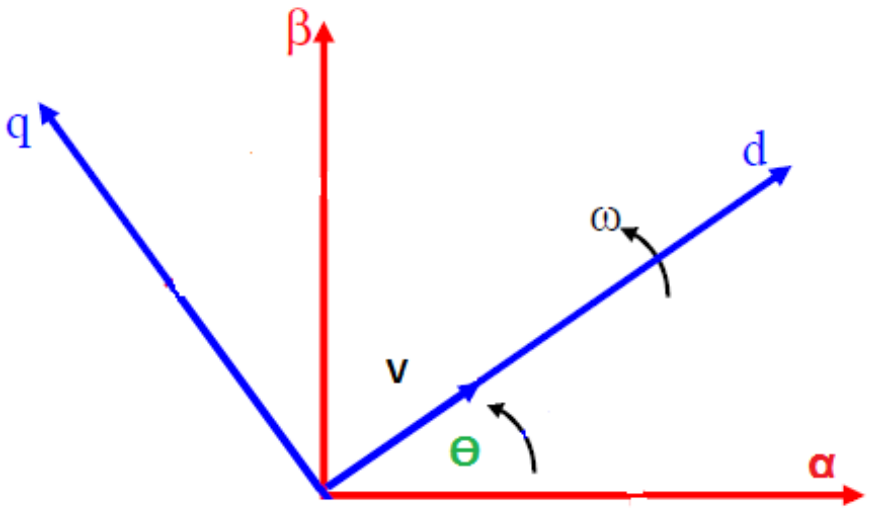


Figura 6.5

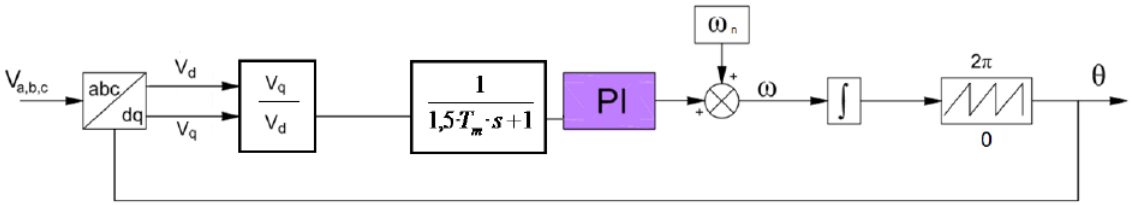


Figura 6.6

Se utiliza un controlador PI. Para su digitalización se usa el método de Euler en su versión Forward.

Forward Euler:

$$Int(k) = Int(k-1) + T_m \varepsilon(k)$$

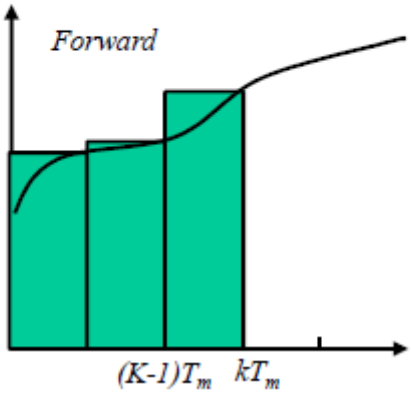


Figura 6.7

De este modo el algoritmo para todos los PLL se programa de la siguiente forma:

```
/**Algoritmo PLL**/
VCCF=VqFiltrada/VdFiltrada;
error_acumCCF=error_acumCCF+VCCF*Tm;
w_piCCF=Kpl*( (error_acumCCF)/Tnl);
w_1CCF=w_piCCF+Kpl*VCCF;
w_2CCF=w_2CCF+w_1CCF*Tm;
if(w_2CCF>2*pi)
{
w_2CCF=0;
}
```

Séptima parte: alineación de la tensión con el eje d positivo

Tal y como está realizada la programación no hay nada que impida que el vector de tensión de red se alinee con el eje negativo d. La Vq sigue siendo 0 pero el ángulo queda desfasado 180 grados. Para evitar esta situación se utiliza el ángulo obtenido anteriormente mediante la transformada de Clarke. Este ángulo no es el adecuado ya que cuando hay desequilibrios, componente continua o armónicos se tiene un pequeño desfaso y se "tuerce". Pero es muy útil para alinear el vector tensión con el eje d positivo ya que el desfase es muy pequeño y el ángulo obtenido siempre está correctamente alineado.

Se programa de forma que mientras el ángulo de Clarke esté entre 0.6 y 1.4pi radianes, se realice la resta entre este ángulo y el obtenido por el PLL. Si esta resta está entre –pi/2 y +pi/2 quiere decir que se ha alineado el vector tensión con el eje positivo del sistema coordenado dq y es correcto. Si no significa que el vector tensión se ha alineado con el eje d negativo y hay que desfasar el ángulo 180 grados (pi radianes) para que el sistema coordenado dq se sitúe correctamente.

```
/**Código para que Vd se alinee en con el eje positivo**/

if(AngClarke>0.6){

if(AngClarke<1.4*pi){

if((AngClarke-w_2CCF)<-pi/2){w_2CCF=w_2CCF+pi;}

else{

if((AngClarke-w_2CCF)>pi/2){w_2CCF=w_2CCF+pi;}}}}

if(w_2CCF>2*pi){w_2CCF=w_2CCF-2*pi;}
```

Octava parte: salidas

Las salidas de Arduino utilizadas son las analógicas DAC0 y DAC1. Para tener la máxima precisión se utilizan los 12 bits. Esto quiere decir que si se da a la salida el valor de 4096 ésta dará 3.3V.

Como se quiere tener la máxima amplitud, sabiendo que el ángulo está en radianes y varía entre 0 y 2pi se hace la transformación para que el 2pi sea 4096.

En la salida DAC0 se obtiene la frecuencia. La frecuencia está en radianes por segundo por lo tanto toma un valor de 2pi*50. Como se van a realizar pruebas en las que la frecuencia sea algo mayor no se hará la transformación para que a 50Hz dé el máximo de Arduino (que sería multiplicar por 13) sino algo menos.

```
/**Escribo en el pin 1 de salidas analógicas el valor del ángulo**/

w4=w_2CCF*4096/(2*pi);

analogWrite(DAC1,w4);

analogWrite(DAC0,w_1CCF*12);
```

7 ANÁLISIS COMPARATIVO DE LOS 3 PLL

7.1 Circuito de simulación

Para simular una red trifásica se utiliza una fuente de voltaje capaz de dar una tensión senoidal con una amplitud y frecuencia deseadas. Pero sólo tiene una salida así que es necesario crear las otras 2 para generar una tensión trifásica.

Para ello se creó un circuito con dos filtros pasotodo de primer orden en serie con ganancia la unidad para bajas frecuencias que son a las que vamos a trabajar y -1 a altas frecuencias. La fase varía entre -180 y 0º. Los filtros pasotodo se diseñan para tener una fase de -120º a 50Hz. De esta forma de la salida de uno se toma la tensión Vs desfasada 120º respecto a la original, Vr, y de la otra la Vt, desfasada 240º.

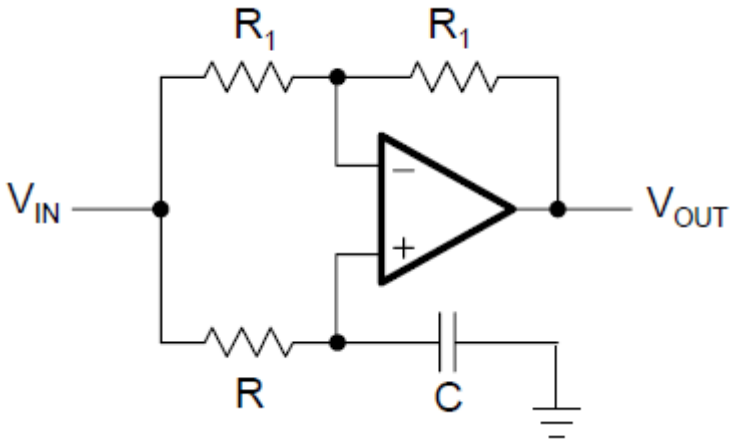


Figura 7.1

$$A(s) = \frac{1 - RC\omega_c \cdot s}{1 + RC\omega_c \cdot s}$$

Figura 7.2

A 50Hz la fase es:

$$\propto = -2Arctg(w * RC) = -120$$

$$RC * 2 * pi * 50 = 1.732$$

Se despeja el valor de la resistencia a partir de la capacidad de un condensador determinado que se escoge de 150nF. Por tanto la resistencia es de 36750Ω

Por último se monta el circuito:

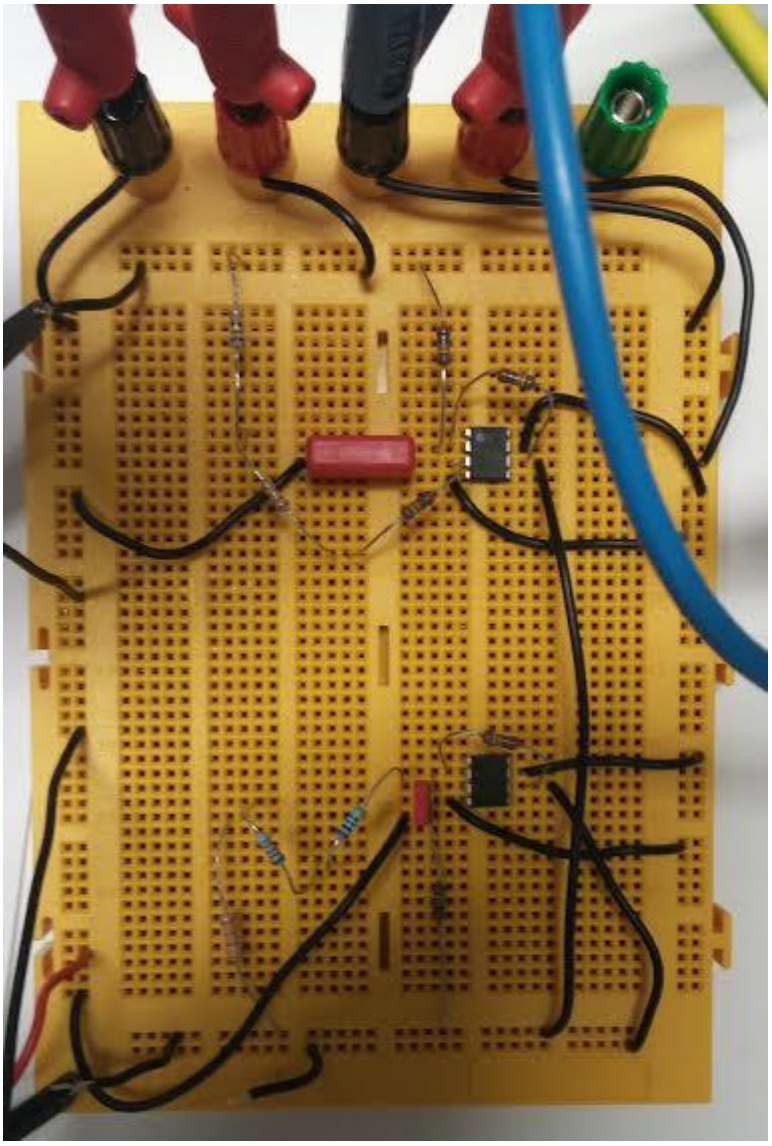


Figura 7.3

7.2 Pruebas realizadas

Para comparar los 3 PLL se estudia su respuesta ante diferentes estados de la red. Se realizarán pruebas de calidad y pruebas de respuesta temporal.

Los problemas que pueden surgir en la red eléctrica y que hacen que los PLL tradicionales sin filtros no sean capaces de obtener el ángulo del vector tensión correcto son los siguientes:

- Componente continua.
- Armónicos en la red.
- Desequilibrios.
- Derivas en frecuencia.

De estos casos el tener componente continua en una de las fases no se puede simular en el circuito así que se estudiará posteriormente en los ensayos de potencia.

Las pruebas realizadas se dividen en pruebas de calidad y pruebas de respuesta temporal.

Pruebas de calidad: se mide el error cuadrático medio mediante la creación de un ángulo de referencia que simula el ángulo real que debería dar el PLL y el ángulo obtenido por cada PLL.

- Armónicos en la red: la fuente de tensión utilizada es capaz de generar una señal con armónicos.
- Desequilibrios: el circuito utilizado para obtener las 2 fases que no es capaz de dar la fuente de tensión hace posible que variando la frecuencia por encima o por debajo de 50Hz el desfase sea diferente a 120 grados y tengamos desequilibrio en la red trifásica.
- Funcionamiento normal: red equilibrada a 50Hz sin armónicos.

Pruebas de respuesta temporal: es importante que el PLL tenga una dinámica rápida y sea capaz de obtener el ángulo correcto ante:

- Escalones de frecuencia.
- Reset de Arduino.

7.2.1 Pruebas de calidad

Mediante una señal de referencia se calcula el error cuadrático medio:

$$\frac{\sqrt{\sum(\text{abs}(\Theta\text{ref} - \Theta))\text{^}2}}{2\pi n}$$

Señal de referencia para calcular el error:

- Red equilibrada:

Cuando la red está equilibrada y no presenta armónicos el ángulo obtenido mediante la transformada de Clarke y la arcotangente de las componentes alfa y beta es el real. Se utiliza este ángulo como referencia para calcular el error del ángulo obtenido por el PLL.

```
/**Cálculo del ángulo mediante Clarke para comprobación del PLL**/
Va=(2./3)*(Vr-0.5*Vs-0.5*Vt);
Vb=(2./3)*((sqrt(3)/2)*Vs-(sqrt(3)/2)*Vt);
AngClarke=atan2(Vb,Va);
if(AngClarke<0)
{
AngClarke=AngClarke+2*pi;
}
```

- Armónicos:

Los armónicos en la red provocan que el ángulo obtenido mediante la transformación de Clarke y la función arcotangente esté torcido pero no produce ningún desfase. Es posible utilizar los pasos por 0 de este ángulo para generar una señal de la misma frecuencia pero en la que el ángulo no esté torcido. Para ello se primero se obtiene la frecuencia calculando el tiempo que tarda entre cada paso por 0 y, se obtiene el ángulo multiplicando la frecuencia por el tiempo y reiniciando el ángulo cuando pasa de 2pi.

```
/**Prueba 2 para sacar la señal de referencia para armónicos**/

Suma5++;

if(AngClarke>3){

correcto=1;}

tiempo5=micros()-tiempo6;

if(AngClarke<0.2){

if(correcto>0){

F++;reinicio++;tiempoFrecuencia=micros()-tiempo6;tiempo6=micros(); if (reinicio<2){F=0;tiempo2=millis();}

if(reinicio<2){AnguloReal=AngClarke;}else{FrecuenciaReal=1/(tiempoFrecuencia/1000000);AnguloReal=0;}

}

correcto=0;

}

}

AnguloReal=FrecuenciaReal*2*pi*tiempo5/1000000;

if(AnguloReal>2*pi){AnguloReal=AnguloReal-(2*pi);}**/
```

- Desequilibrios en la red:

Un desequilibrio en la red produce que el ángulo obtenido mediante la transformada de Clarke tenga un pequeño desfase y esté torcido. Este pequeño desfase hace que no sea posible utilizar el mismo método que en los armónicos para obtener el ángulo de referencia. Como los desequilibrios sólo están presentes en Vs y Vt es posible utilizar los pasos por 0 de Vr para obtener la frecuencia y de ahí el ángulo. Como Vr puede tener algo de ruido que dé varios pasos por 0 falsos se crea una variable que calcula el tiempo que ha pasado desde el último paso por 0 y si es menor de 5 milisegundos no aumenta el contador de paso por 0. Cada vez que Vr pasa por 0 se calcula el tiempo hasta el siguiente paso por 0, la inversa de ese tiempo es la frecuencia y con ella se obtiene el ángulo que se resetea en cada paso por 0 de Vr.

```
/**Cálculo de ángulo de referencia para desequilibrios**/
tiempo5=micros()-tiempo6;
if(Vr<150){if(Vr>-150){
 if(con<1){
  if(tiempo5>5000){con++;}}
 if(con>0){
  if(tiempo5>15000){tiempoFrecuencia=micros()-
tiempo6;tiempo6=micros();con=0;AnguloReal2=0;digitalWrite(13,LOW);}}}}
FrecuenciaReal2=1./(tiempoFrecuencia/1000000);
AnguloReal2=FrecuenciaReal2*2*pi*tiempo5/1000000;
AnguloReal3=AnguloReal2+(pi/2);
if(AnguloReal3>(2*pi)){AnguloReal3=AnguloReal3-(2*pi);}
```

A continuación se muestran los resultados de las pruebas de calidad realizadas:

- Red equilibrada:

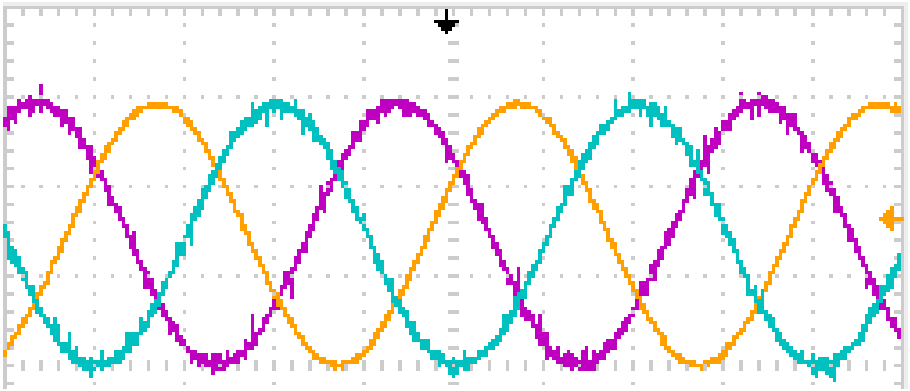


Figura 7.4

PLLdq:

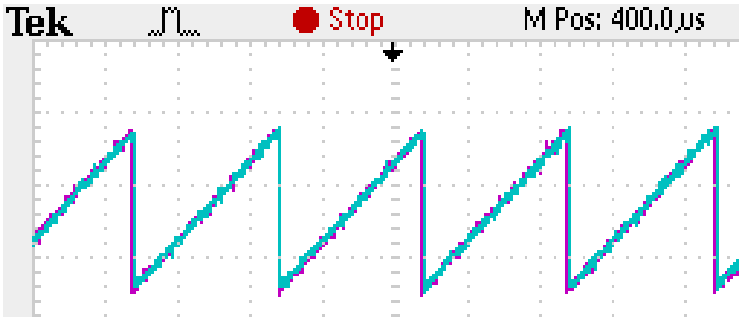


Figura 7.5

PLLccf:

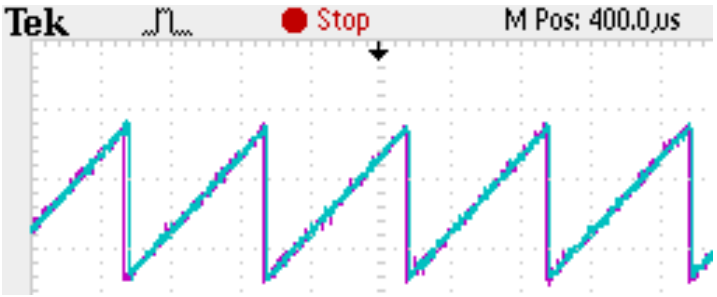


Figura 7.6

PLL1:

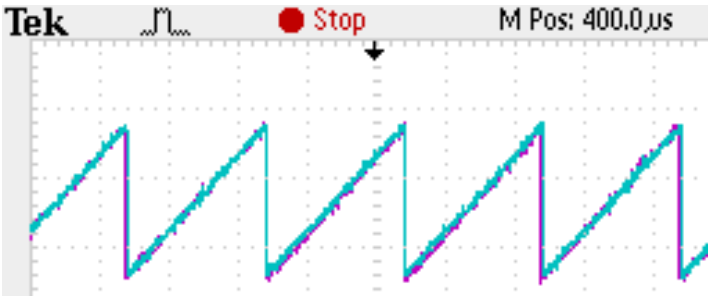


Figura 7.7

Error cuadrático medio (en porcentaje):

PLL DQ	0.067501568
PLL CCF	0.056142172
PLL 1	0.035555349

El error es mínimo y el ruido de la lectura del osciloscopio puede alterar los resultados. La conclusión es que los 3 PLL son capaces de obtener el ángulo correctamente para una red equilibrada y sin armónicos.

- Red desequilibrada a 43Hz:

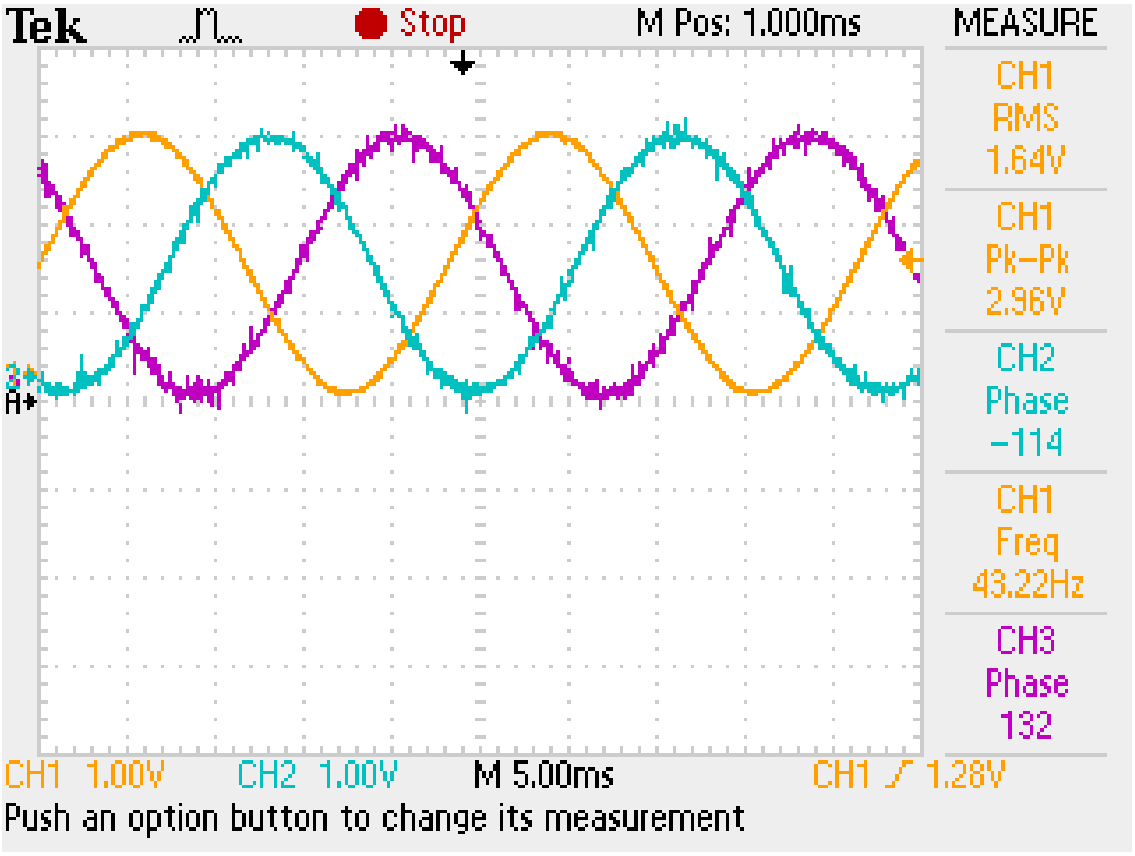


Figura 7.8

PLLdq:

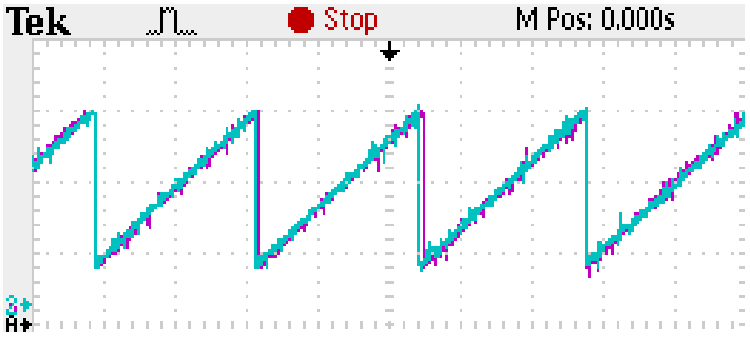


Figura 7.9

PLLccf:

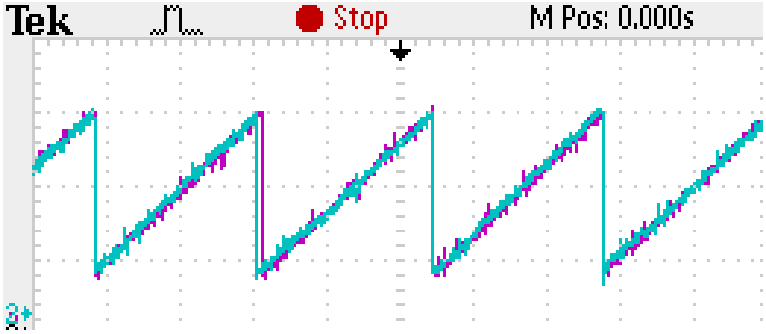


Figura 7.10

PLL1:

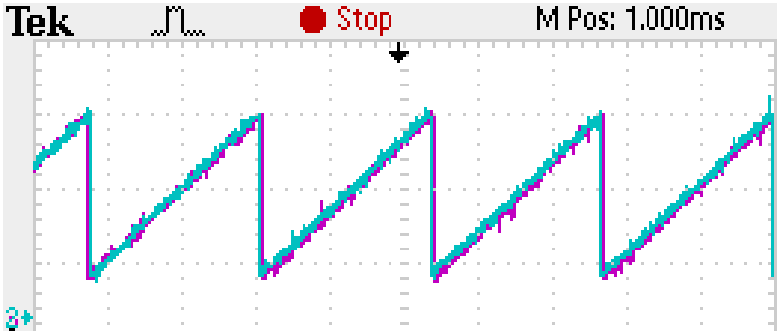


Figura 7.11

Error cuadrático medio (en porcentaje):

PLL DQ	0.030761768
PLL CCF	0.026653799
PLL 1	0.027125275

El error es mínimo y el ruido de la lectura del osciloscopio puede alterar los resultados. La conclusión es que los 3 PLL son capaces de obtener el ángulo correctamente para una red desequilibrada y sin armónicos.

- Red desequilibrada a 57Hz:

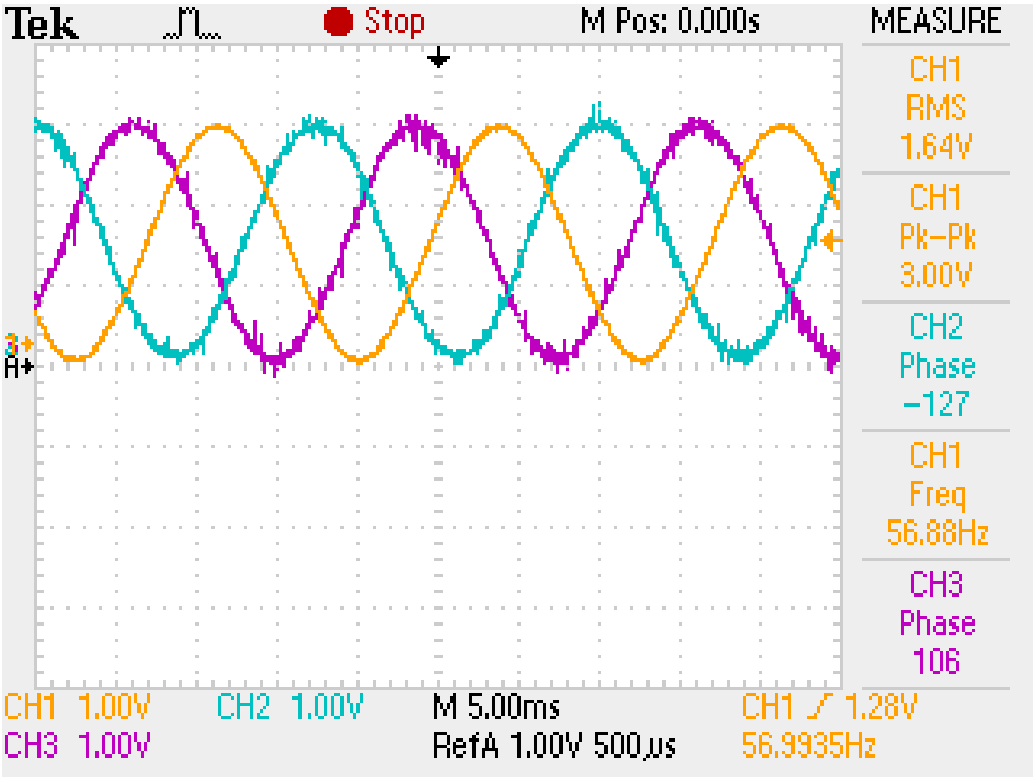


Figura 7.12

PLLdq:

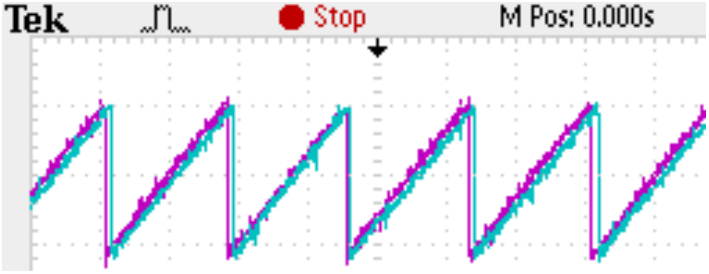


Figura 7.13

PLLccf:

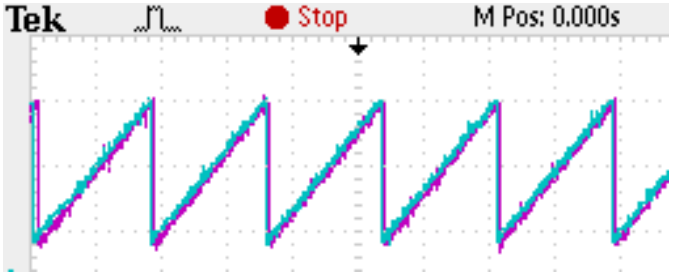


Figura 7.14

PLL1:

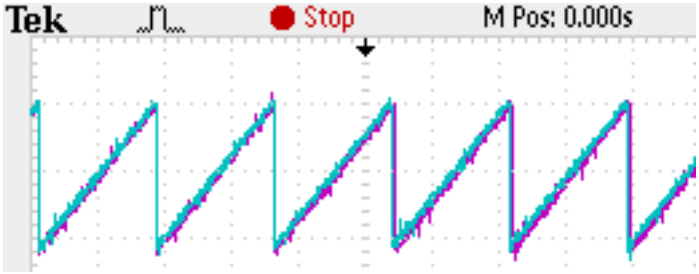


Figura 7.15

Error cuadrático medio (en porcentaje):

PLL DQ	0.196142274
PLL CCF	0.105289847
PLL 1	0.071759653

El error es mínimo y el ruido de la lectura del osciloscopio puede alterar los resultados. La conclusión es que los 3 PLL son capaces de obtener el ángulo correctamente para una red desequilibrada y sin armónicos aunque en este caso sí que se aprecia un mayor error en el PLLdq.

- Armónicos a 45Hz:

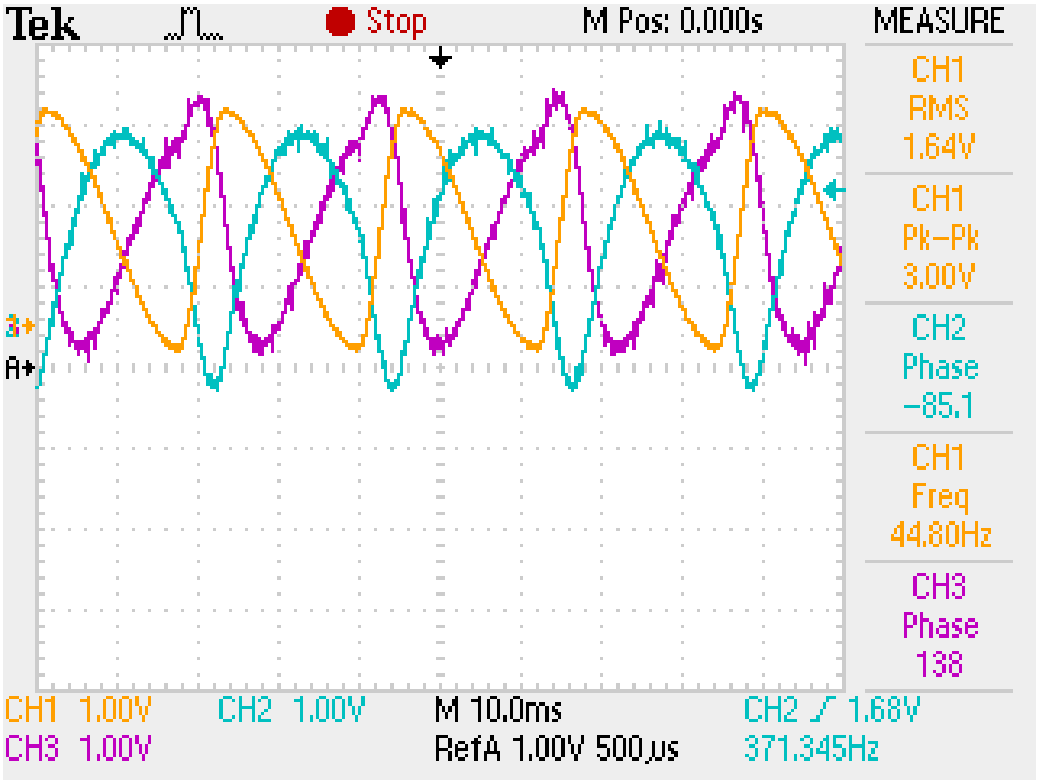


Figura 7.16

PLLdq:

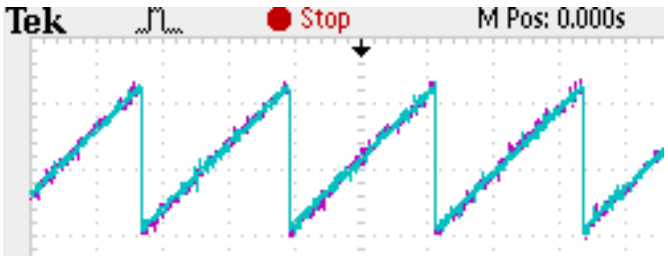


Figura 7.17

PLLccf:

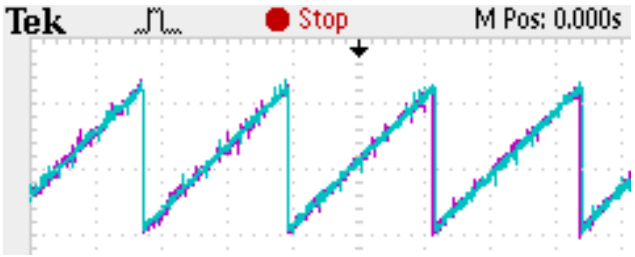


Figura 7.18

PLL1:

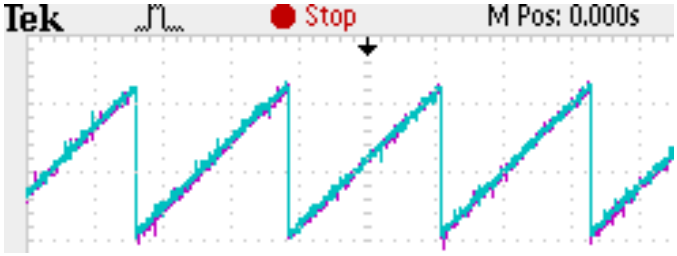


Figura 7.19

Error cuadrático medio (en porcentaje):

PLL DQ	0.051461721
PLL CCF	0.053847216
PLL 1	0.052957881

El error es mínimo y el ruido de la lectura del osciloscopio puede alterar los resultados. La conclusión es que los 3 PLL son capaces de obtener el ángulo correctamente para una red desequilibrada y con armónicos.

- Armónicos a 50Hz:

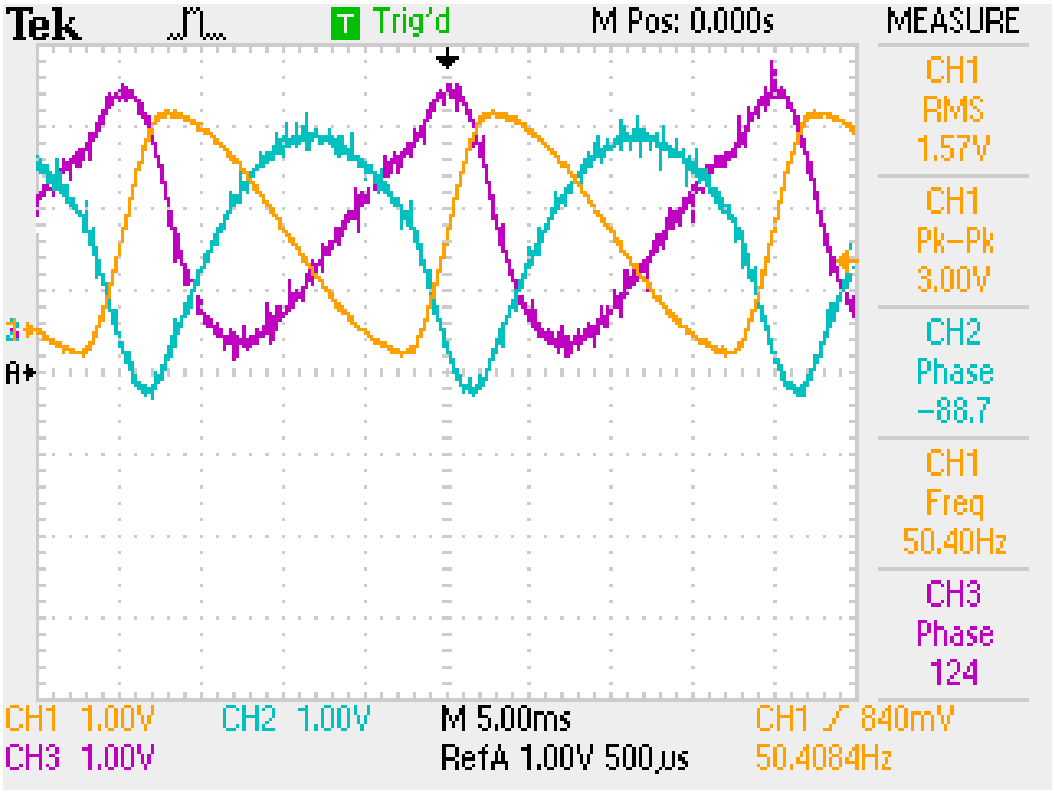


Figura 7.20

PLLdq:

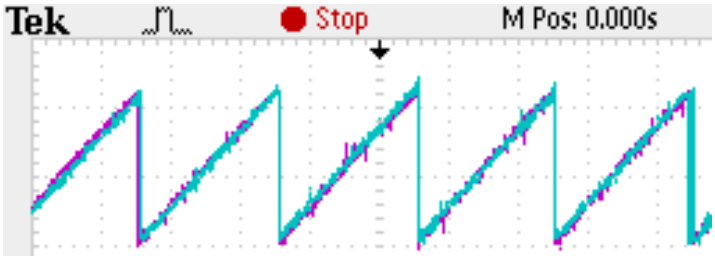


Figura 7.21

PLLccf:

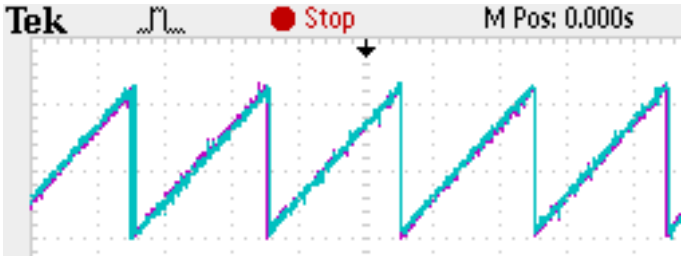


Figura 7.22

PLL1:

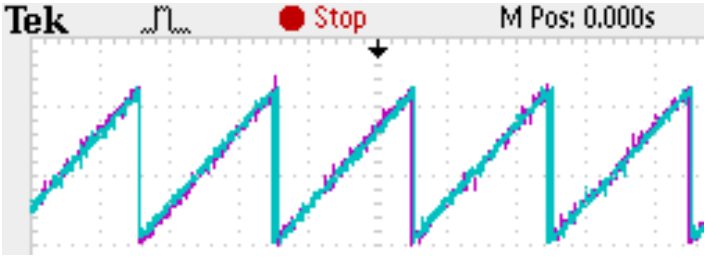


Figura 7.23

Error cuadrático medio (en porcentaje):

PLL DQ	0.067232626
PLL CCF	0.059315680
PLL 1	0.069304418

El error es mínimo y el ruido de la lectura del osciloscopio puede alterar los resultados. La conclusión es que los 3 PLL son capaces de obtener el ángulo correctamente para una red desequilibrada y con armónicos.

- Armónicos a 55Hz:

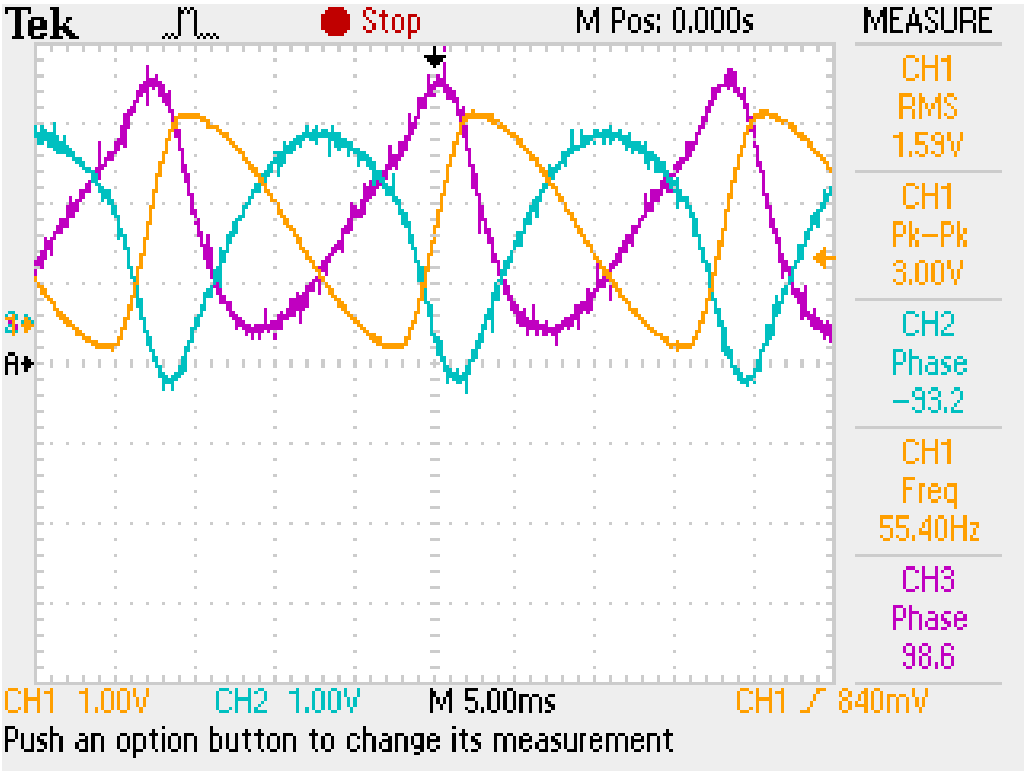


Figura 7.24

PLLdq:

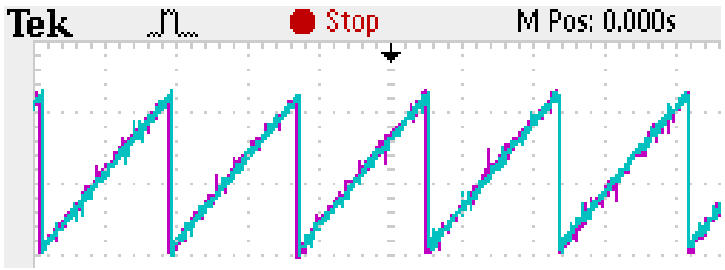


Figura 7.25

PLLccf:

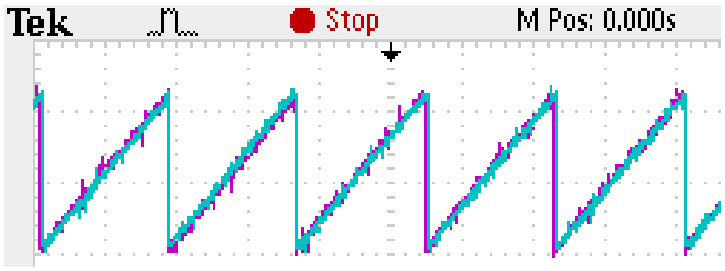


Figura 7.26

PLL1:

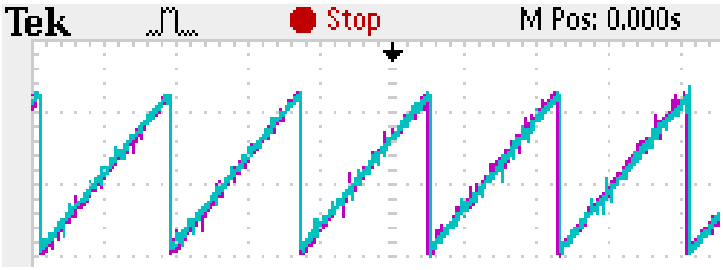


Figura 7.27

Error cuadrático medio (en porcentaje):

PLL DQ	0.096176377
PLL CCF	0.081707711
PLL 1	0.050960170

El error es mínimo y el ruido de la lectura del osciloscopio puede alterar los resultados. La conclusión es que los 3 PLL son capaces de obtener el ángulo correctamente para una red desequilibrada y con armónicos.

7.2.2 Pruebas de respuesta temporal

Para comprobar la respuesta temporal de los PLL se observa cuánto tarda la frecuencia en estabilizarse dentro del 5% de su valor final. Se hacen dos tipos de pruebas, el reseteo de Arduino y los escalones de frecuencia.

- Reset:

Arduino tiene un botón que Resetea todas las variables y ejecuta el programa desde el principio. Se hacen 2 pruebas, la primera con los PLL normales a los cuales se les inicializaba a 50Hz y una prueba en las cuales se les inicializa desde 0Hz por lo que tardan más en alcanzar su valor final.

  - Reset normal:

PLLdq:

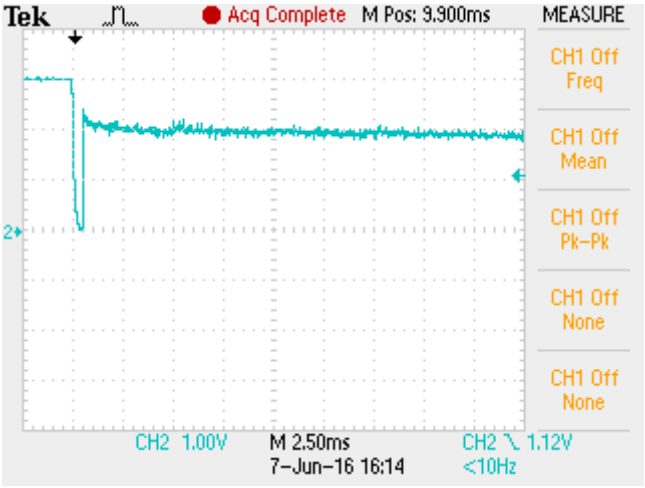


Figura 7.28

PLLccf:

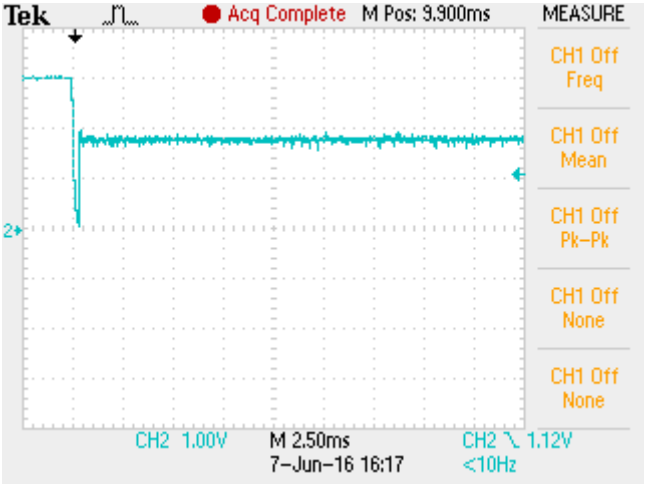


Figura 7.29

PLL1:

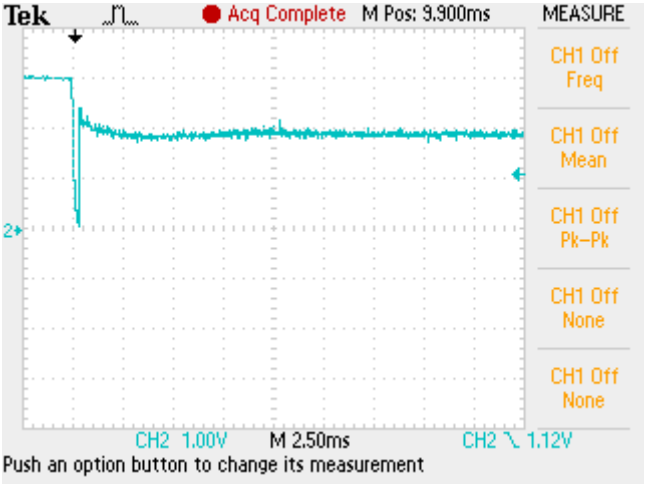


Figura 7.30

Tiempo de estabilización en milisegundos:

PLL DQ	10.6800
PLL CCF	1.1100
PLL 1	1.8700

- Reset sin inicializar los PLL a 50Hz:

PLLdq:

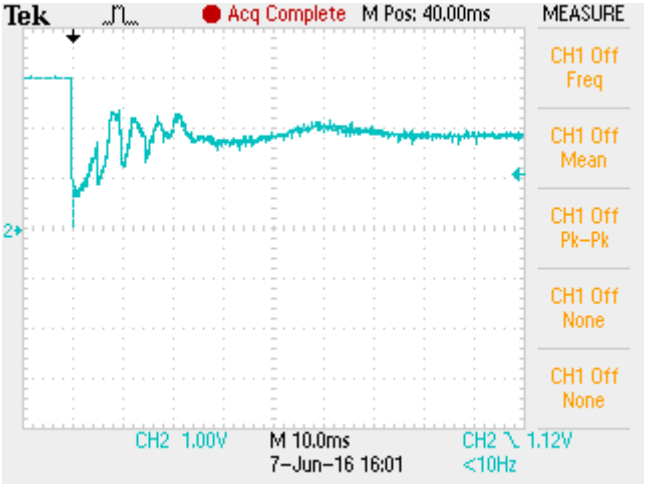


Figura 7.31

PLLccf:

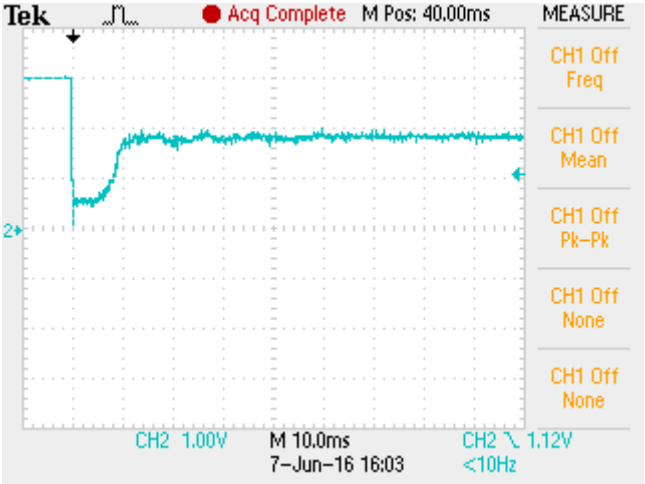


Figura 7.32

PLL1:

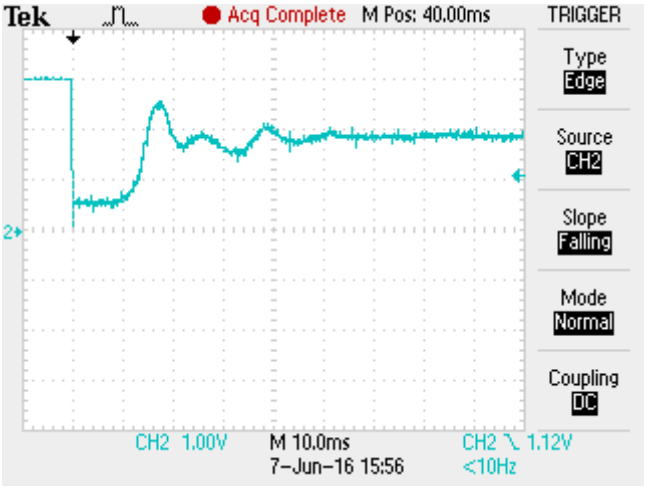


Figura 7.33

Tiempo de estabilización en milisegundos:

PLL DQ	62.04
PLL CCF	24.04
PLL 1	40.16

- Escalones de frecuencia:

Para comprobar la respuesta de los PLL ante un escalón de frecuencia se debe hacer la prueba con el mismo escalón para cada PLL. Para ello se programaron los 3 PLL en un mismo programa de Arduino pero el tiempo de ejecución era mayor de 500 milisegundos y sería necesario bajar la frecuencia de muestreo a al menos 1KHz para que funcionaran. Por tanto para realizar la prueba se utilizaron 3 Arduinos Due diferentes cada uno con un PLL programado y a los cuales se alimentaba con la misma fuente de tensión.

Se realizó la prueba para varios escalones de frecuencia, uno de frecuencia ascendente y otro descendente y se obtuvieron los siguientes resultados:

- 47-53Hz:

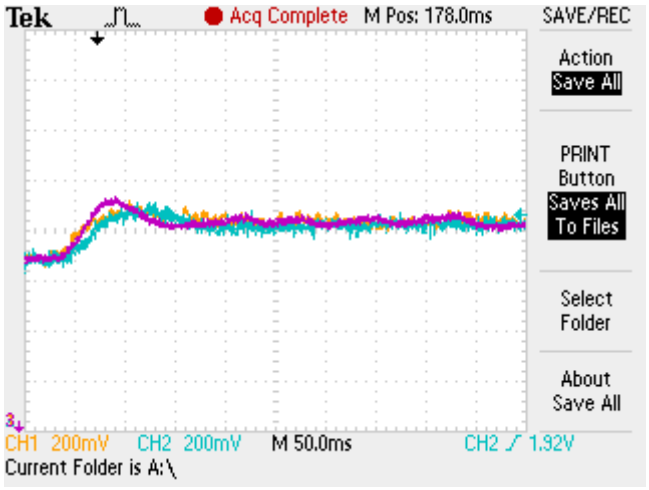


Figura 7.34

PLLdq:

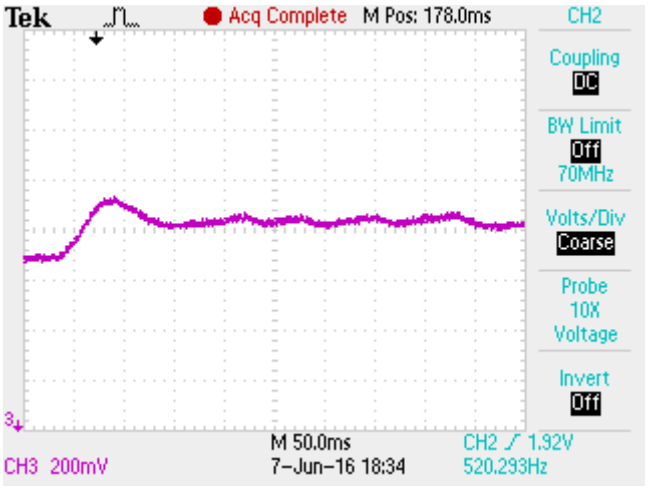


Figura 7.35

PLLccf:

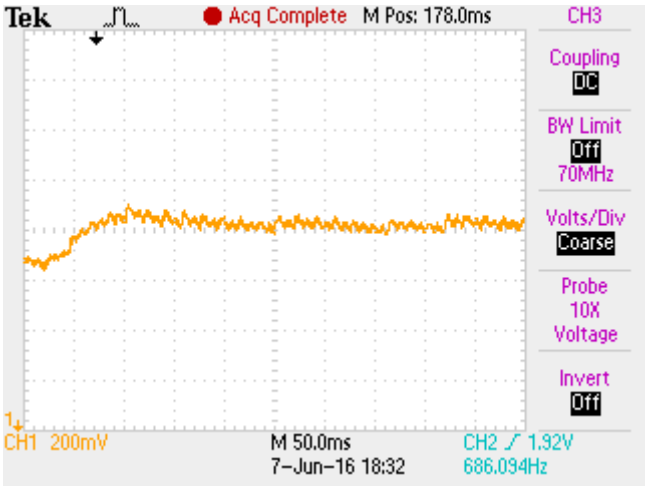


Figura 7.36

PLL1:

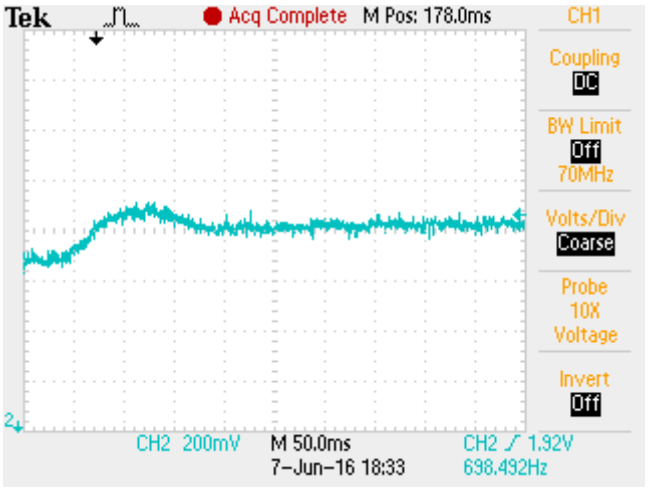


Figura 7.37

Como se observa todos los PLL tienen una respuesta similar aunque el PLLdq tiene una oscilación a baja frecuencia y el PLLccf a alta.

Se calcula el tiempo que tarda cada PLL en alcanzar el valor final con un error del 2%:

PLL DQ	361.3
PLL CCF	73.7
PLL 1	81.9

El que tiene una respuesta más rápida ante un escalón de frecuencia es el PLLccf aunque el PLL1 no se queda muy atrás. El más lento es el PLLdq con diferencia.

- 47-53Hz

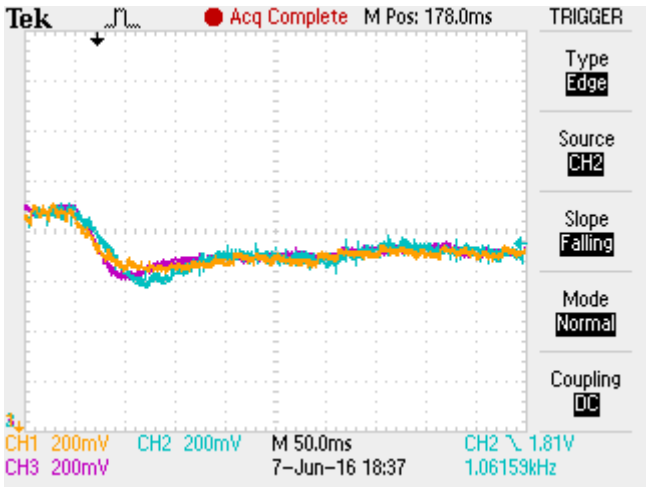


Figura 7.38

PLLdq:

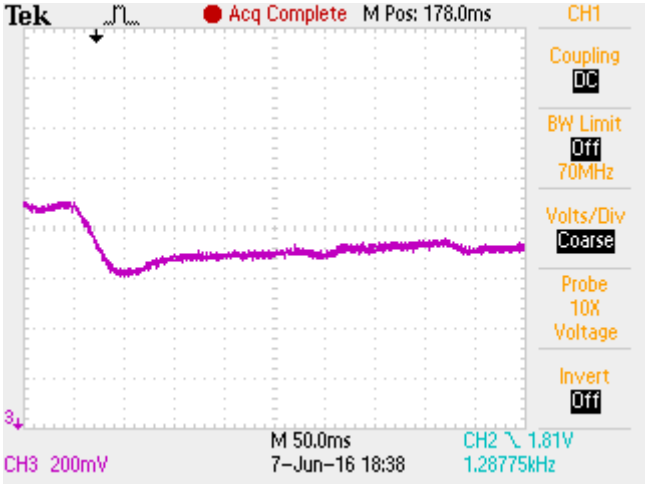


Figura 7.39

PLLccf:

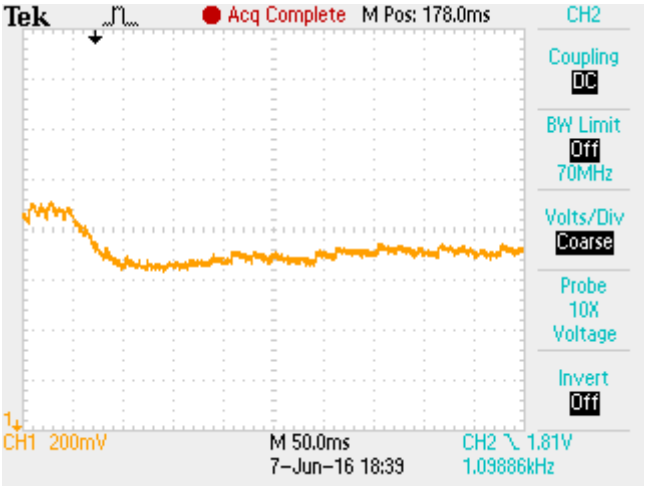


Figura 7.40

PLL1:

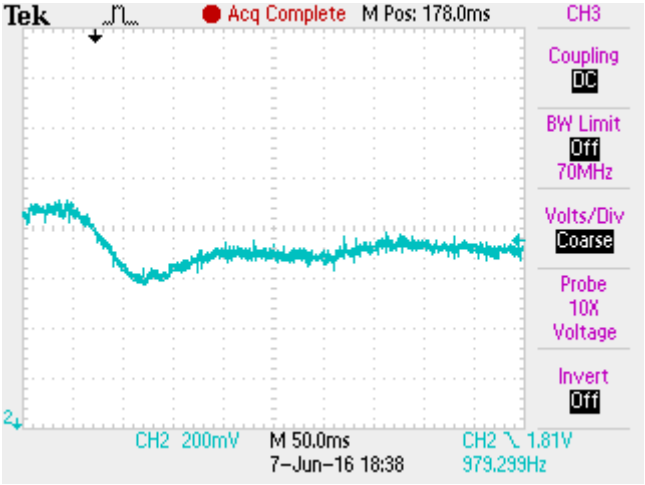


Figura 7.41

Como se observa todos los PLL tienen una respuesta similar aunque el PLLdq tiene una oscilación a baja frecuencia y el PLLccf a alta.

Se calcula el tiempo que tarda cada PLL en alcanzar el valor final con un error del 2%:

PLL DQ	169.4
PLL CCF	172
PLL 1	138.4

RESUMEN

	Normal	***Desequilibrios***		***Distorsión***		
	50Hz	***43Hz***	***57Hz***	***45Hz***	***50Hz***	***55Hz***
PLL DQ	0.06750156	0.030761768	0.196142274	0.051461721	0.067232626	0.096176377

PLL CCF	0.05614217	0.026653799	0.105289847	0.053847216	0.059315680	0.081707711
PLL 1	0.035555349	0.027125275	0.071759653	0.052957881	0.069304418	0.050960170

	Tiempo Reset (ms)		***Escalones de frecuencia (ms)***	
	Inicializado	***Sin iniciar***	***47-53Hz***	***54-49***
PLL DQ	10.6800	62.04	361.3	169.4
PLL CCF	1.1100	24.04	73.7	172
PLL 1	1.8700	40.16	81.9	138.4

8 ENSAYO DE POTENCIA

Para poder comprobar el funcionamiento real del PLL1 se utilizaron máquinas eléctricas de las cuales, a partir de un captador de tensiones que adapta la tensión trifásica a las especificaciones necesarias por el ARDUINO (0-3.3V), se alimentaron las entradas del ARDUINO. A partir del ángulo obtenido por el PLL1 se realizó la transformada de Park de la tensión trifásica y se obtuvieron las señales transformadas Vd y Vq. Conociendo las características de la tensión trifásica es posible saber qué forma debe tener la tensión una vez aplicada la transformada de Park. Esto se comprueba mediante los armónicos y, si el ángulo obtenido por el PLL1 es el correcto, la transformada de Park de las tensiones debería ser la ideal.

Para comprobar el funcionamiento del PLL1 en diferentes condiciones se realizaron los siguientes ensayos:

- Tensiones equilibradas.
- Tensiones desequilibradas.
- Tensiones con componente continua.
- Tensiones equilibradas con armónicos.
- Tensiones desequilibradas con armónicos.
- Tensiones desequilibradas con armónicos y componente continua.

Todos estos ensayos se realizaron a 45, 50 y 55Hz.

Montaje:

Máquina de corriente continua: se alimenta un transformador variable mediante una tensión trifásica y se rectifica con un rectificador AC/DC para alimentar el rotor. El estator se excita rectificando la tensión de una fase de la red. Esta máquina moverá la máquina síncrona.

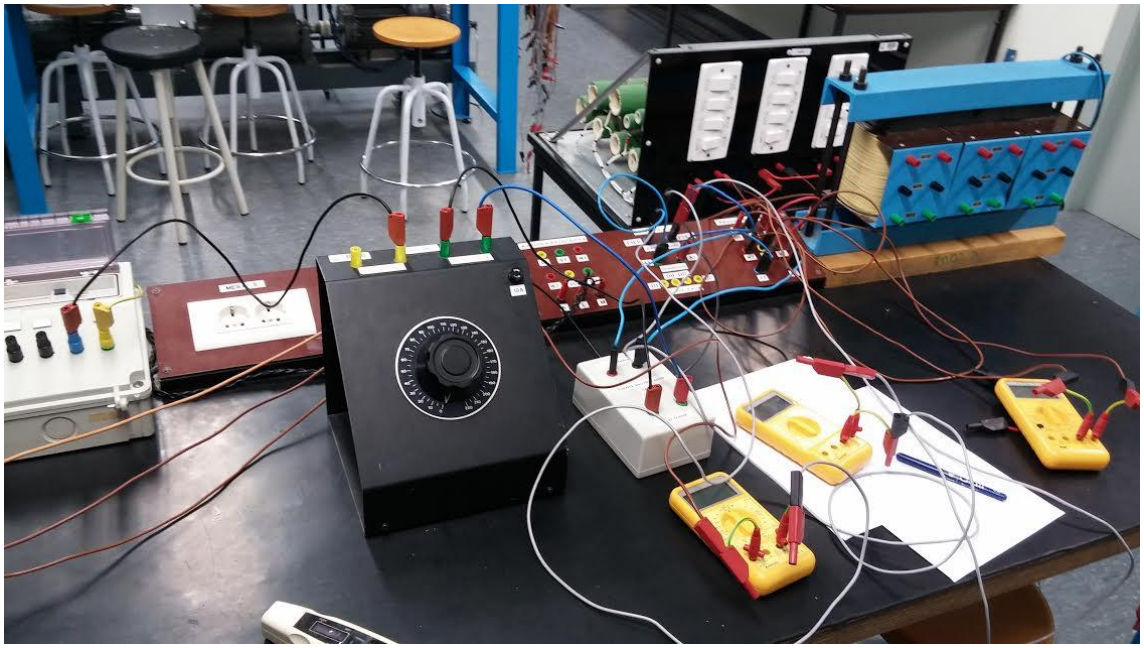


Figura 8.1

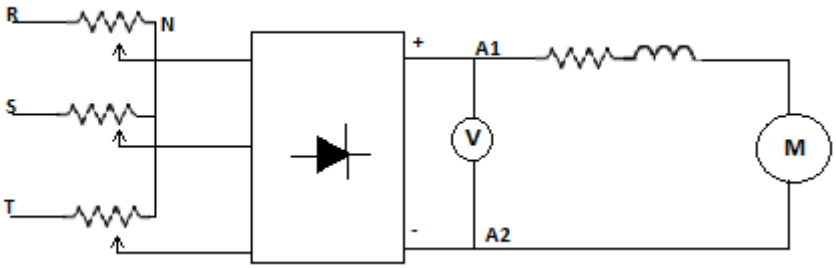


Figura 8.2

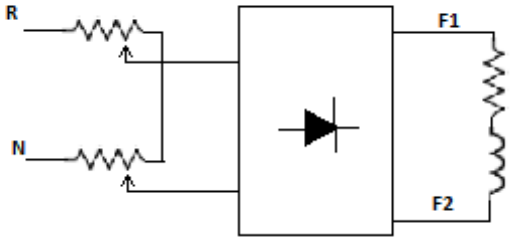


Figura 8.3

Máquina síncrona: se excita el rotor y el estator se conecta en estrella a unas resistencias con las que se regulará la tensión de salida. La excitación se controla rectificando la tensión de una fase de la red.

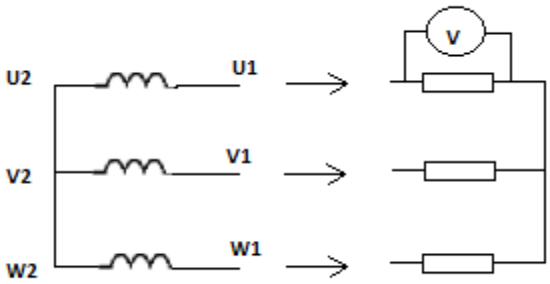


Figura 8.4

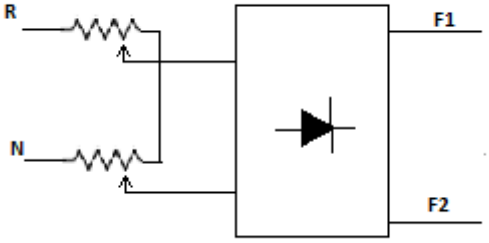


Figura 8.5

8.1 Tensiones equilibradas

Se conectan resistencias del mismo valor en todas las fases del estator de la máquina síncrona conectando dos bancos de resistencias en serie. Se mide la tensión en la resistencia del banco 1 en cada fase para crear las tensiones de red que se introducen a ARDUINO.

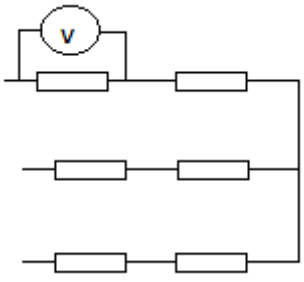


Figura 8.6

- 45Hz

Fuentes de tensión:

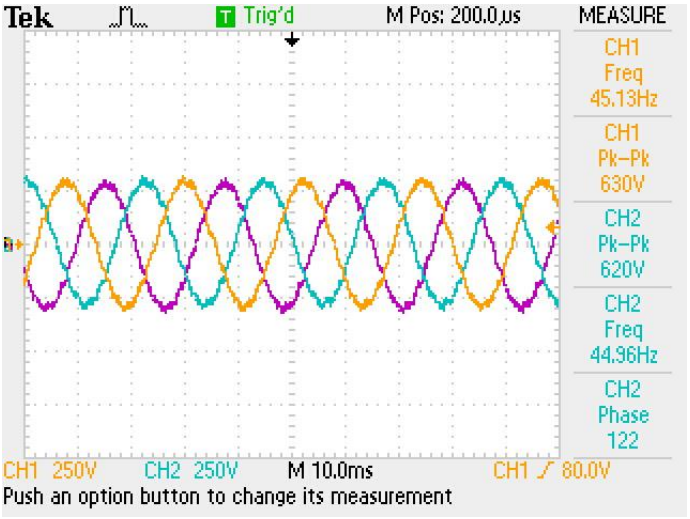


Figura 8.7

Vd y vq:

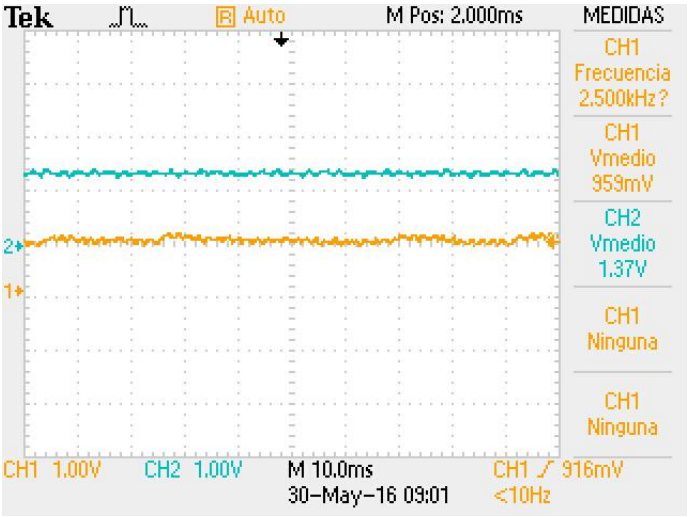


Figura 8.8

Si el ángulo obtenido por el PLL1 en esta situación es el correcto, a 45Hz y con las tensiones equilibradas las tensiones en Vd y Vq se deberían ver como tensiones continuas tal y como se ve en la imagen. El pequeño rizado se debe al ruido.

También se puede observar en los armónicos:

Vr:

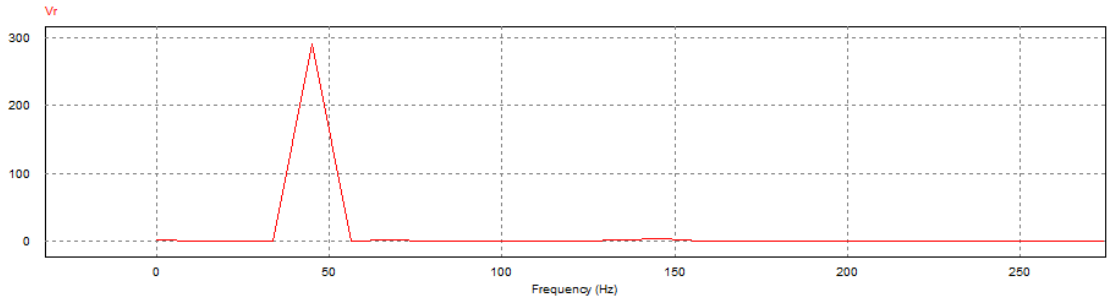


Figura 8.9

Vd:

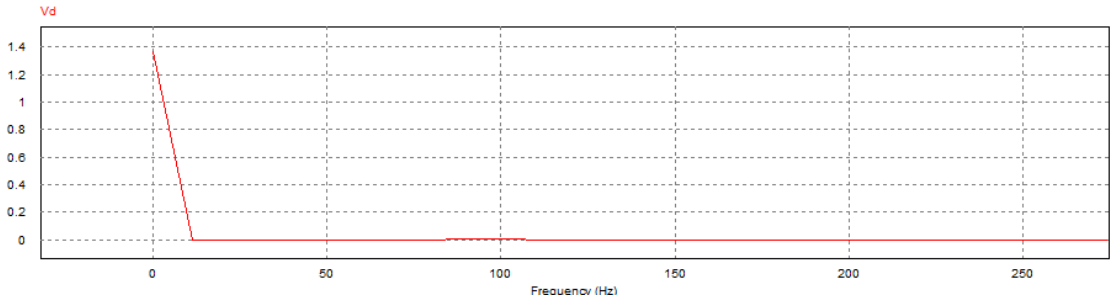


Figura 8.10

8.2 Tensiones desequilibradas

Para desequilibrar las tensiones de entrada al ARDUINO se utilizan dos bancos de resistencias. La suma de resistencias en serie para cada fase es la misma, sin embargo en la fase S la resistencia que se pone en cada banco es diferente al de las otras dos fases (aunque la suma es la misma), de este modo la tensión Vs que se coge para llevar al ARDUINO es de diferente valor a las otras dos fases creando así el desequilibrio.

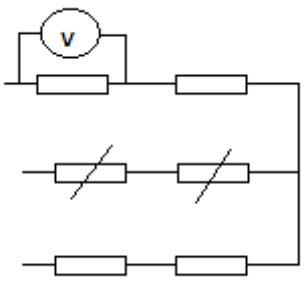


Figura 8.11

Los desequilibrios en la red se presentan como una secuencia inversa a la frecuencia de la red. Lo que nos interesa es conocer el ángulo de la secuencia directa. En el sistema dq aparecen como secuencia inversa al doble de la frecuencia de la red (100Hz).

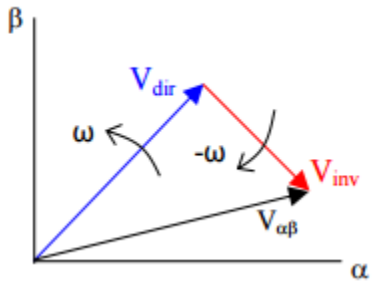


Figura 8.12

El ángulo que interesa obtener del PLL es el de la secuencia directa para que los desequilibrios no le afecten ya que de otra forma la frecuencia variaría, no sería en todo momento la real de la red y la potencia calculada mediante las transformadas de Park de la tensión y de la intensidad no sería la correcta. Por lo tanto los desequilibrios se deberían observar en las tensiones Vd y Vq como la suma de una tensión continua (la de la secuencia directa) y una tensión senoidal a una frecuencia del doble de la de la red (ya que al ser secuencia inversa, en los ejes d y q que giran a la frecuencia de la red se verá como una tensión que gira al doble de la frecuencia de la red y en sentido inverso). La componente en q debido a la secuencia directa es 0 y la debida a la secuencia inversa es una senoidal como ya se ha dicho. La componente en d debido a la secuencia directa es una continua con el mismo valor que la tensión trifásica ya que se ha utilizado el convenio invariante en tensión y una componente senoidal del mismo valor que Vq debido a la secuencia inversa del desequilibrio.

50Hz:

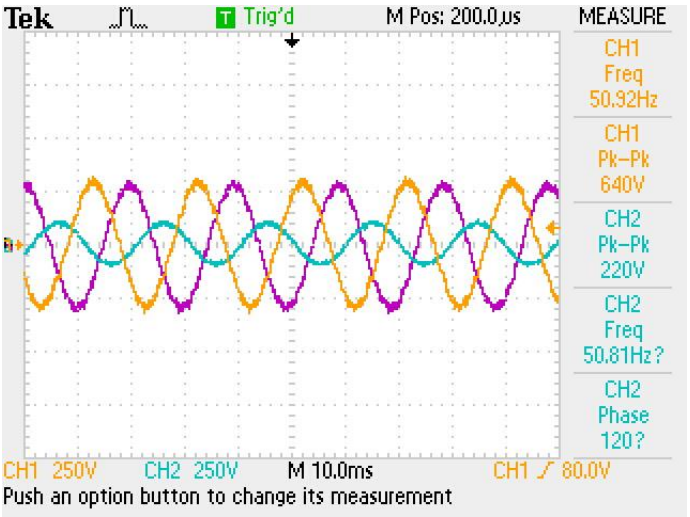


Figura 8.13

Vd y Vq:

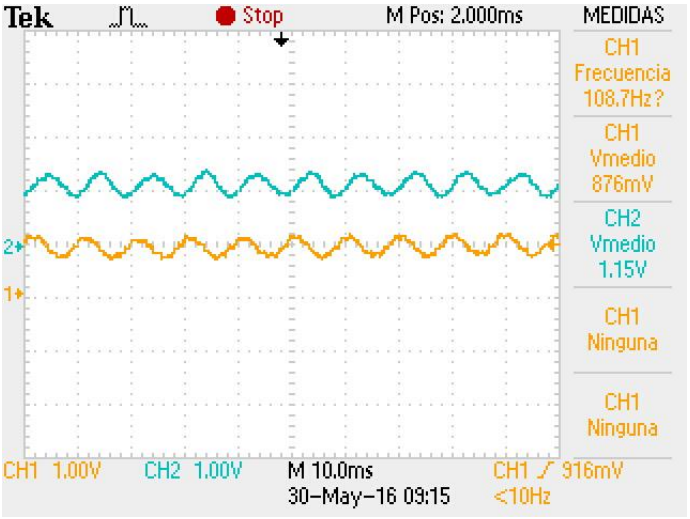


Figura 8.14

Armónicos:

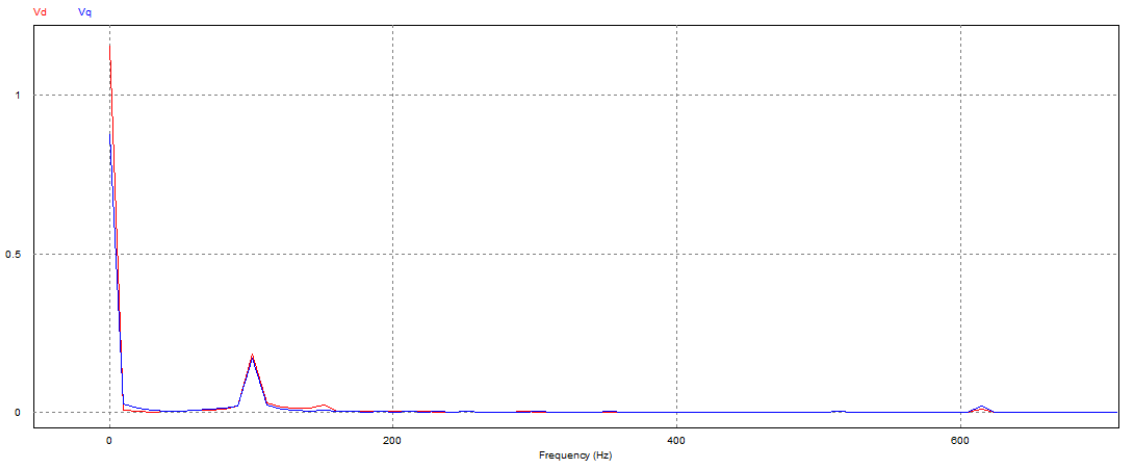


Figura 8.15

Cabe destacar que como ARDUINO no es capaz de dar tensión negativa, a la Vq se le ha sumado 0.8V para poder observar todos los valores. De forma que tiene un offset permanente en 0.8V que es el valor que se ve en continua y que realmente es 0.

La Vd tiene una componente continua de 1.088V debida a la secuencia directa.

También se puede observar cómo existe un armónico a 100Hz (doble de la frecuencia de red) que es el mismo para Vd y Vq y es el debido al desequilibrio.

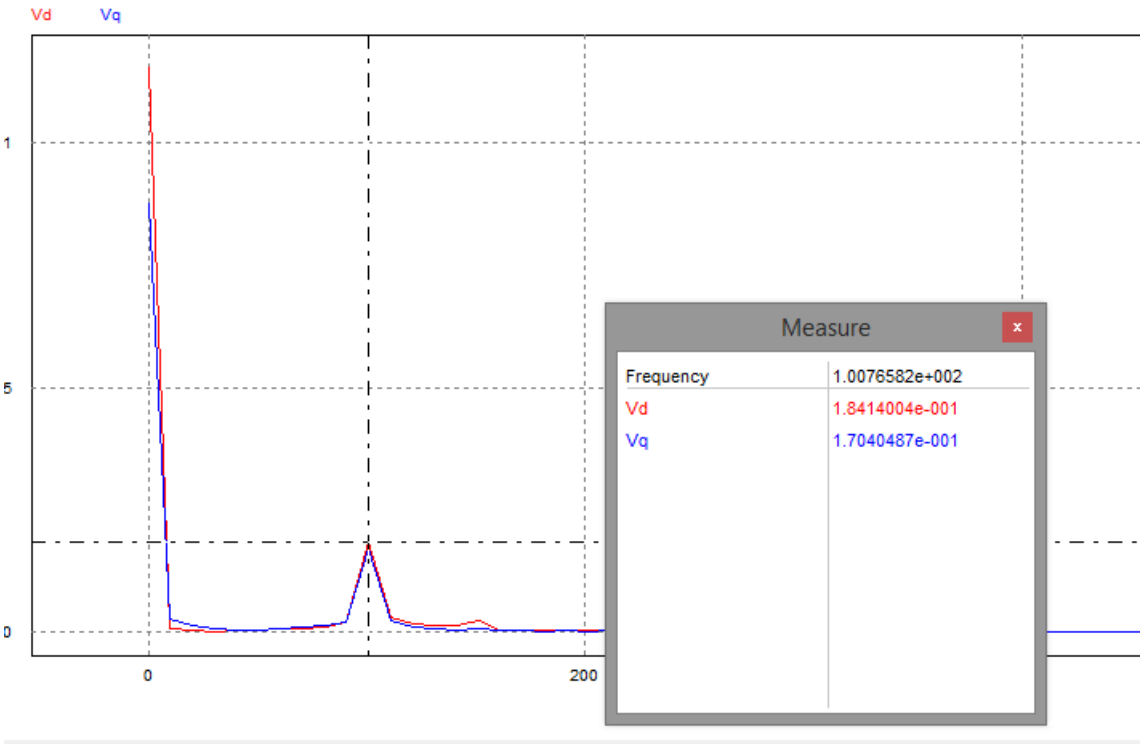


Figura 8.16

8.3 Tensiones equilibradas con armónicos

Para simular una red equilibrada en la que las tensiones tienen armónicos se utilizó en cada resistencia de salida de la máquina síncrona un rectificador y un condensador en paralelo con la resistencia. El rectificador rectifica la tensión haciendo que tome valores sólo positivos. El condensador "aplana" esta tensión, de modo que cuando la tensión del estator es positiva y mayor que la tensión del condensador pasa corriente y la tensión medida es la que cae en la resistencia. Cuando la tensión del condensador es mayor que la de la máquina no pasa corriente y la tensión que cae en la resistencia es 0.

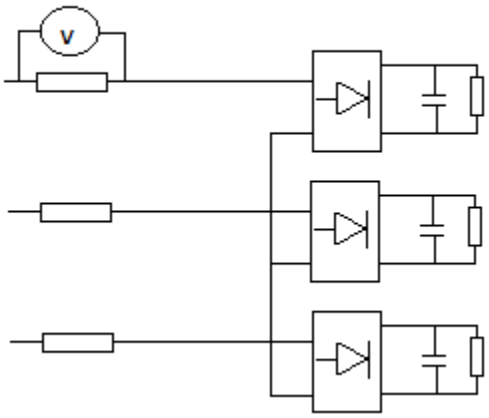


Figura 8.17

El ángulo que se obtiene del PLL no debe tener en cuenta estos armónicos para que la frecuencia sea constante dentro de un periodo. Al igual que en los desequilibrios donde existía un armónico a la frecuencia de red pero en secuencia inversa que se veía en dq al doble de la frecuencia, con los armónicos pasará lo mismo. Todos los armónicos que no estén a la frecuencia de red se verán en dq como armónicos en (n-1)*w ya que el sistema de coordenadas dq gira a w.

55Hz:

Fuentes:

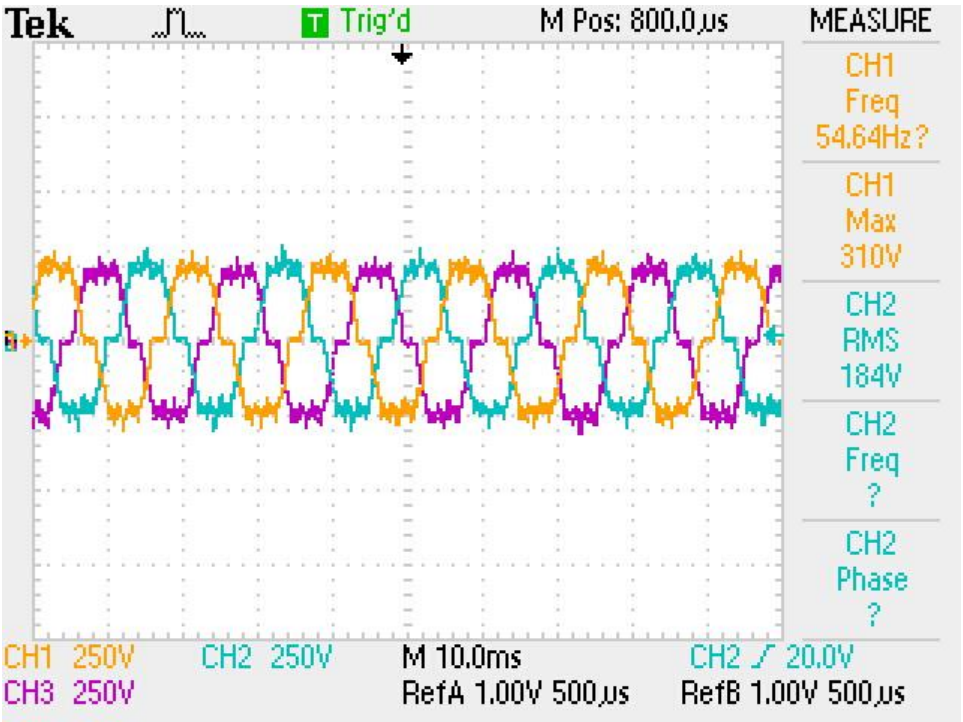


Figura 8.18

Armónicos:

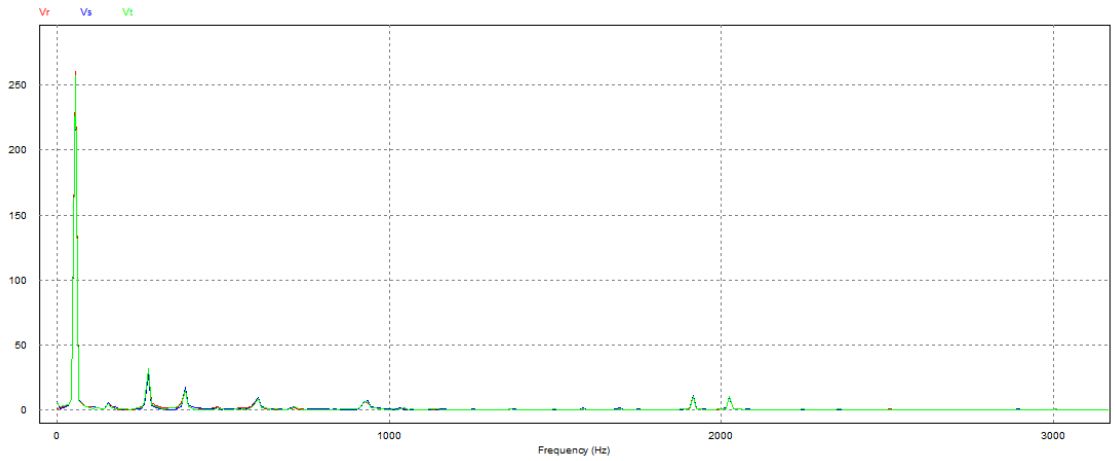


Figura 8.19

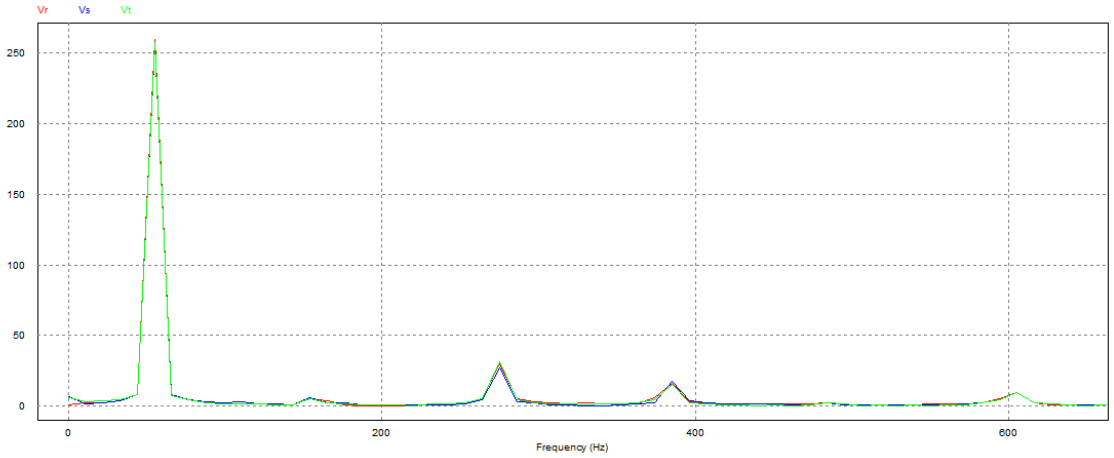


Figura 8.20

El armónico principal en las fuentes está a 55Hz, y los siguientes a 275Hz y 385Hz. Armónicos 5 y 7. Armónicos impares debido a la distorsión de la red.

Por lo tanto en Vd y Vq deberían aparecer el armónico principal a 0Hz (en Vq será 0) y otros armónicos a 275-55=220Hz y 385-55=330Hz. O lo que es lo mismo (5-1)*55=220Hz y (7-1)*55=330Hz. Armónicos pares.

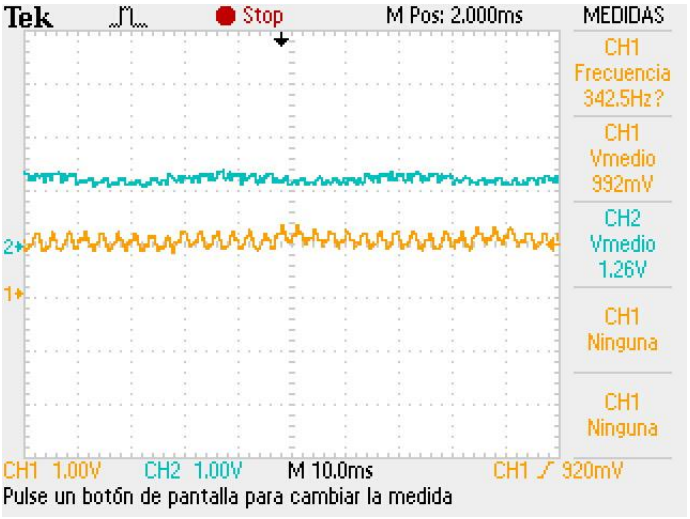


Figura 8.21

Armónico 6 a 330Hz:

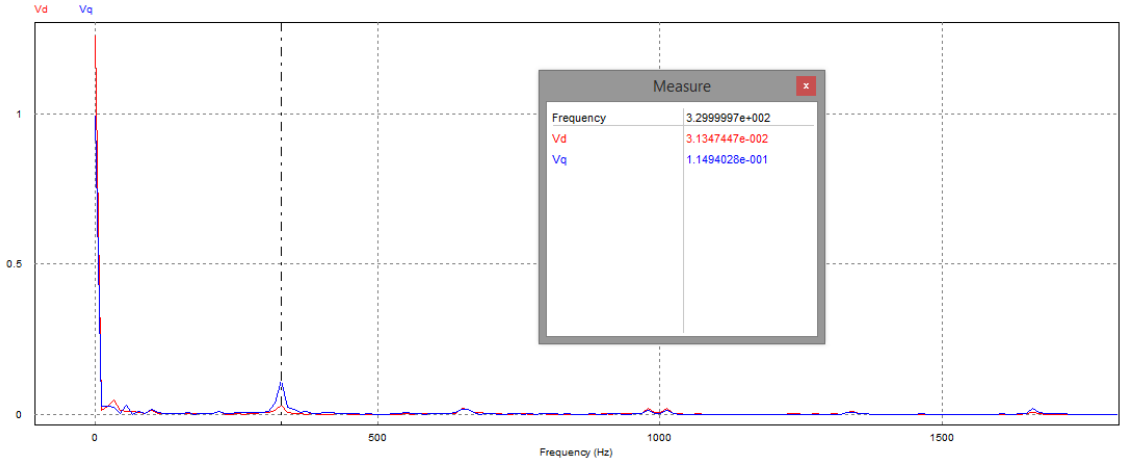


Figura 8.22

Armónico 4 a 220Hz:

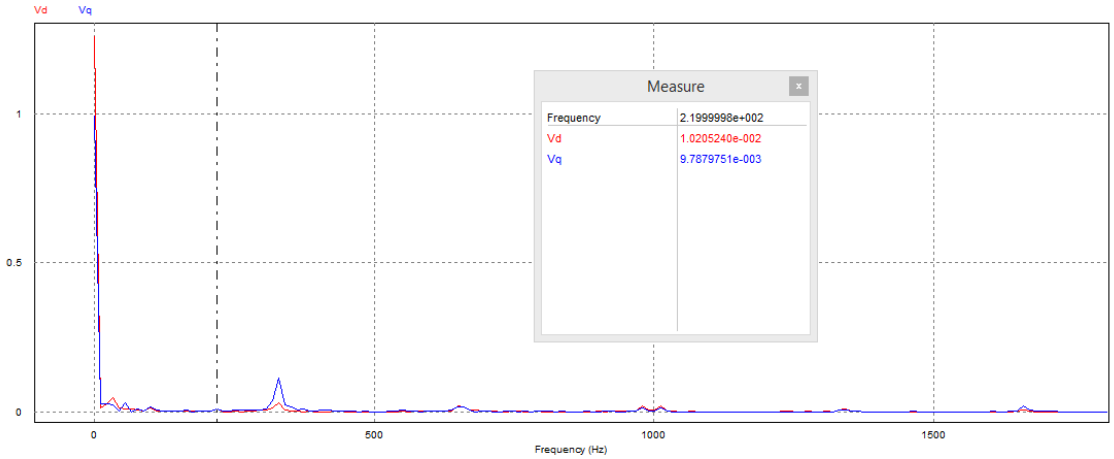


Figura 8.23

Como se puede observar en Vd aparece la componente fundamental de la tensión a 0Hz ya que el sistema coordenado dq gira en sincronismo con la frecuencia de la red y en Vq aparece a 0Hz la componente continua que se le sumaba de 0.8V para que ARDUINO sea capaz de dar salida a la señal completa y algo más de tensión que es debida a la presencia de armónicos en la red que al ser impares aparecen en dq como pares.

También se aprecia cómo el armónico 5 que aparecía en las fuentes de amplitud importante en d y q aparece mucho más pequeño. Esto es debido a que es de secuencia inversa y entonces en dq no aparece como el cuarto armónico sino como el sexto, sumándose al séptimo armónico de las fuentes que también aparece en dq como sexto al ser de secuencia directa.

50Hz:

A 50Hz los armónicos fundamentales 5 y 7 están a 250Hz y 350Hz. En Vd y Vq se deberían ver a 200 y 300Hz respectivamente si ambos fueran de secuencia directa. Pero como el quinto armónico su mayor parte es de secuencia inversa ambos aparecen a 300Hz en Vd y Vq.

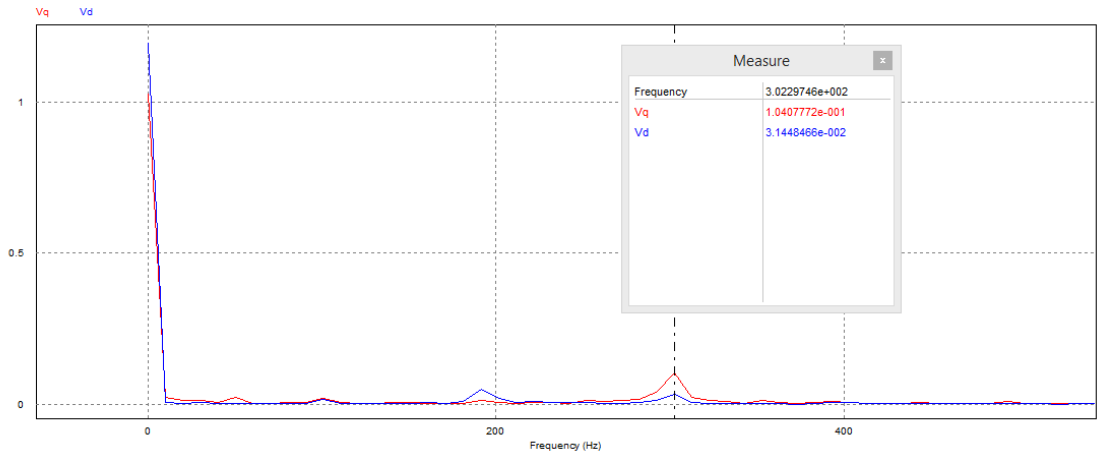


Figura 8.24

45Hz:

A 45Hz los armónicos fundamentales 5 y 7 están a 225Hz y 315Hz. En Vd y Vq se deberían ver a 180 y 270Hz respectivamente si ambos fueran de secuencia directa. Pero como el de 225Hz es de secuencia inversa, ambos aparecerán como el sexto armónico en Vd y Vq.

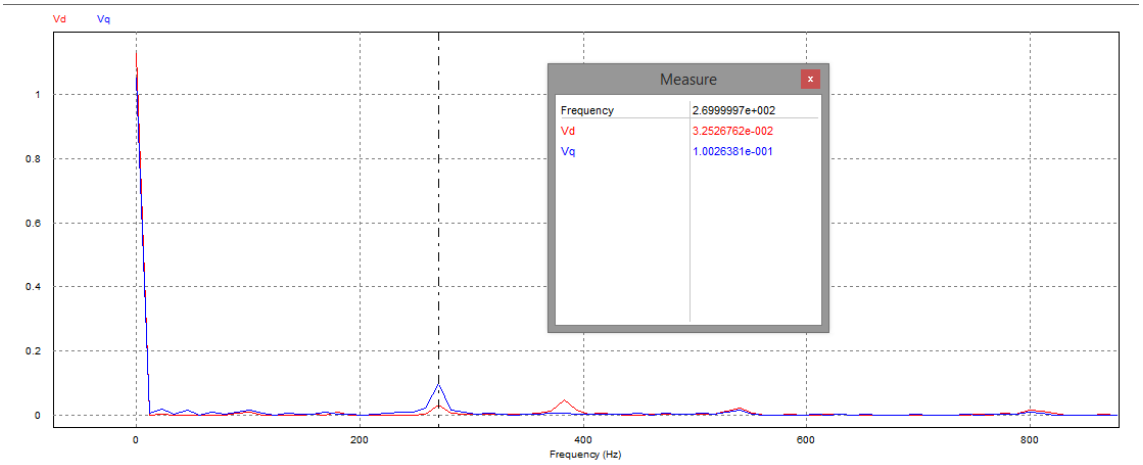


Figura 8.25

8.4 Tensiones desequilibradas con armónicos

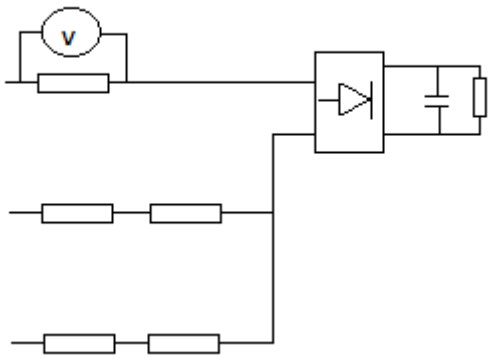


Figura 8.26

Igual que en los casos anteriores el PLL calcula el ángulo eliminando los desequilibrios y los armónicos y esto se observa en las tensiones en d y q. Al haber desequilibrios aparecerá un armónico al doble de la frecuencia fundamental y al tener armónicos aparecerán los impares que deberían aparecer en d y q como pares.

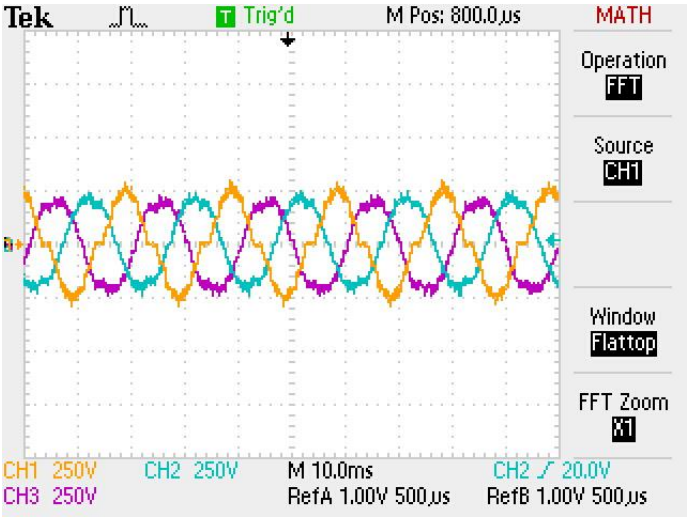


Figura 8.27

45Hz:

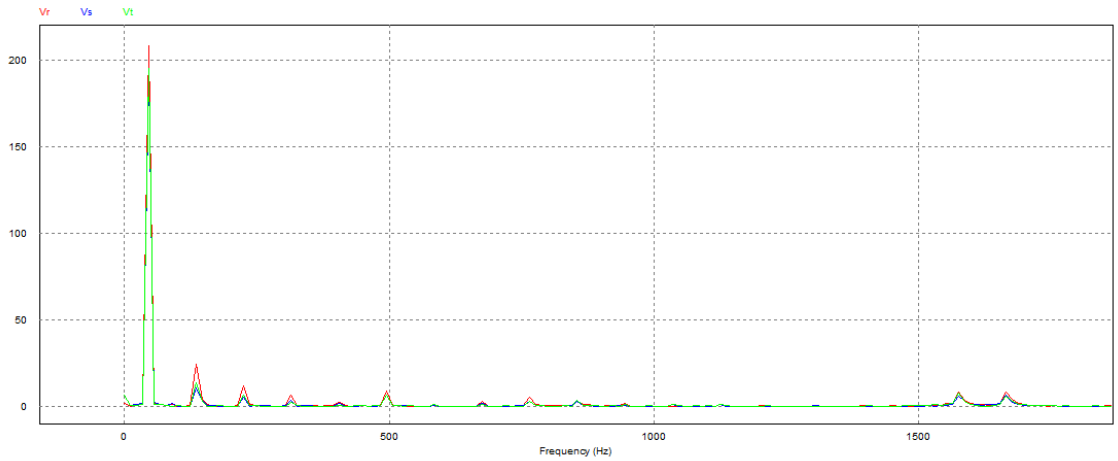


Figura 8.28

Se observan los armónicos 3, 5, 7… los impares.

En Vd y Vq se deberían ver los armónicos pares y uno al doble de la frecuencia fundamental debido al desequilibrio.

El armónico 2 (90Hz) debido al desequilibrio en la red:

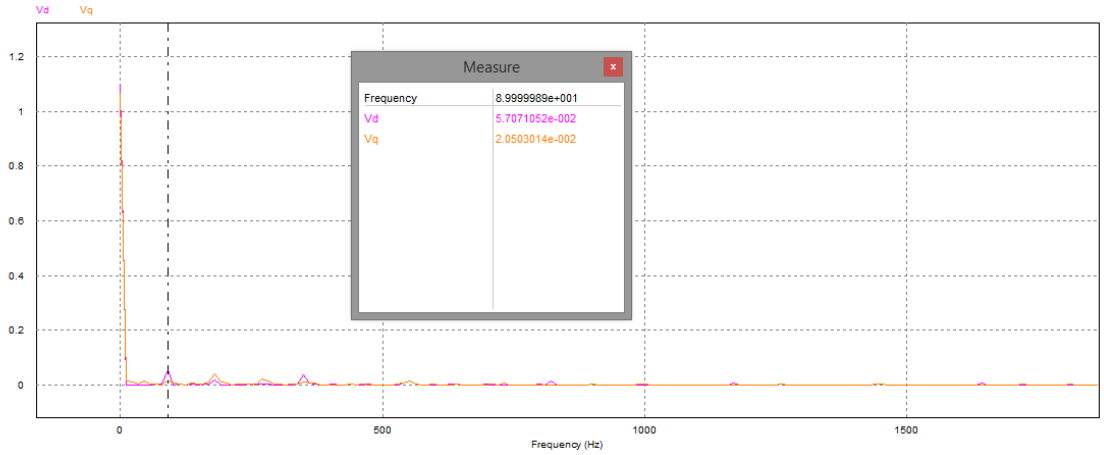


Figura 8.29

El armónico 4 (180Hz) debido a la distorsión de la red:

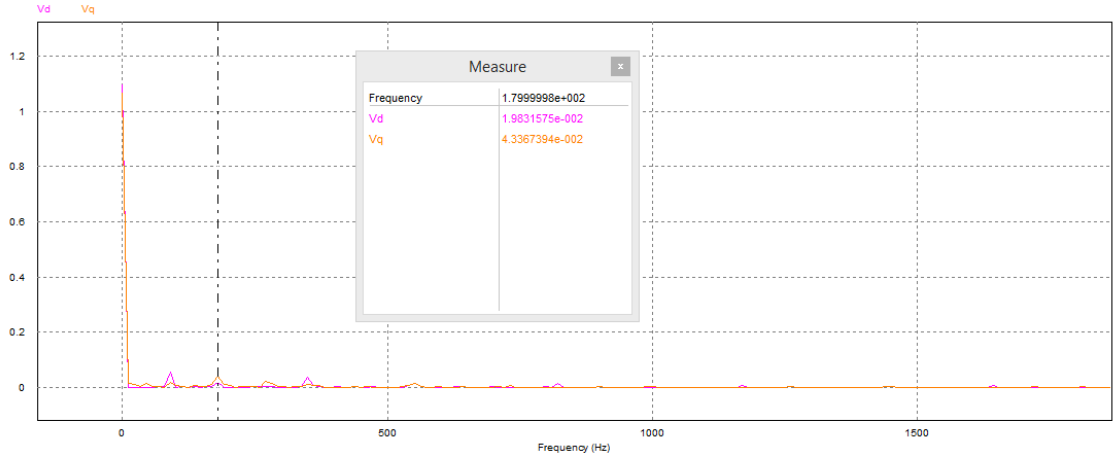


Figura 8.30

El armónico 6 (270Hz) debido a la distorsión de la red:

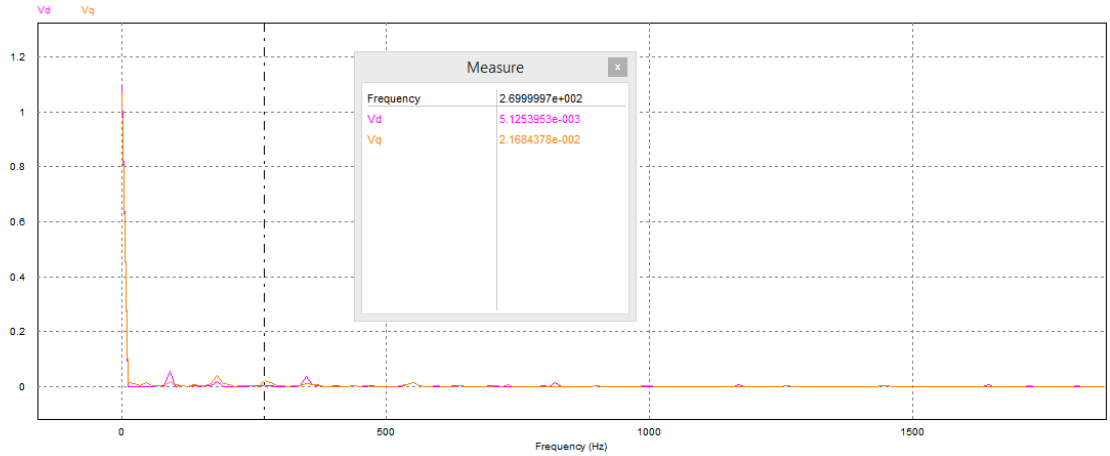


Figura 8.31

8.5 Tensiones con componente continua

Además de componente continua en esta prueba hay tensiones desequilibradas y presencia de armónicos. En consecuencia en d y q deberían aparecer los armónicos pares por la presencia de armónicos en la red, armónico al doble de la frecuencia fundamental debido a los desequilibrios y un armónico a la frecuencia fundamental debido a la presencia de componente continua.

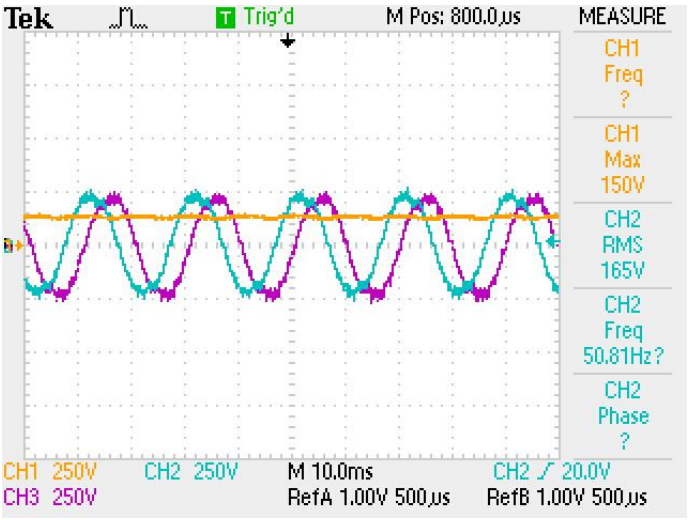


Figura 8.32

55Hz:

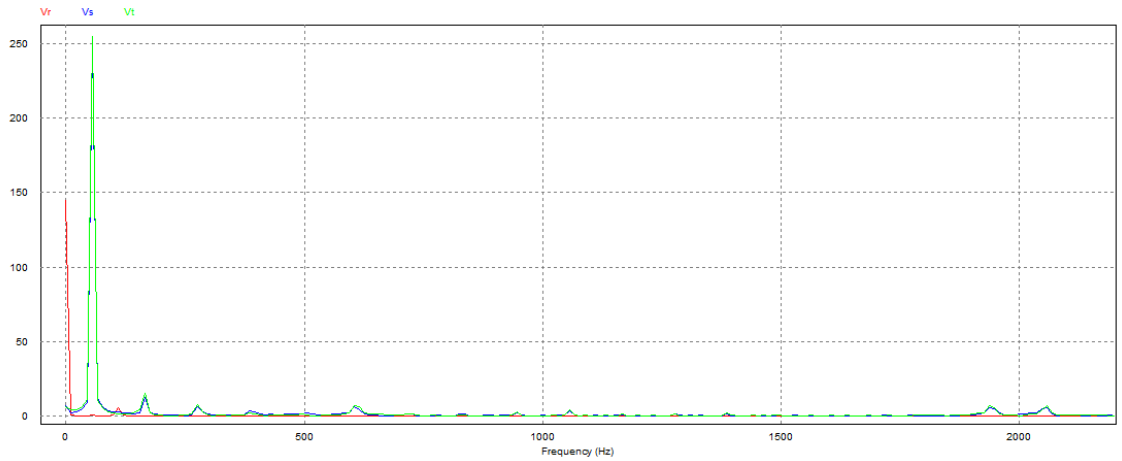


Figura 8.33

La componente continua se debería ver como un armónico a 55Hz. El resto de armónicos (como el 3 y el 5) a n-1. Además aparecerán a 0Hz la componente fundamental en Vd y la suma de la componente debida a los armónicos de la red de orden 1 y el valor de tensión de offset que se puso para poder trabajar con ARDUINO en Vq.

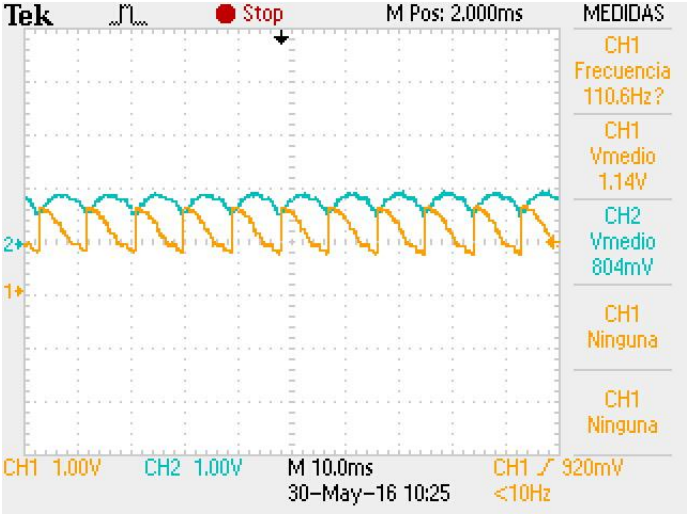


Figura 8.34

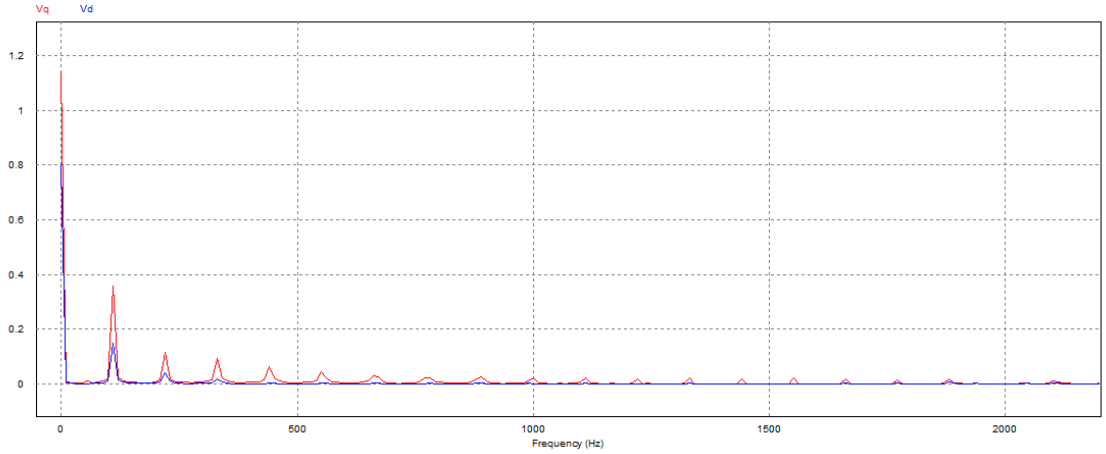


Figura 8.35

Armónico a 55Hz debido a la componente continua:

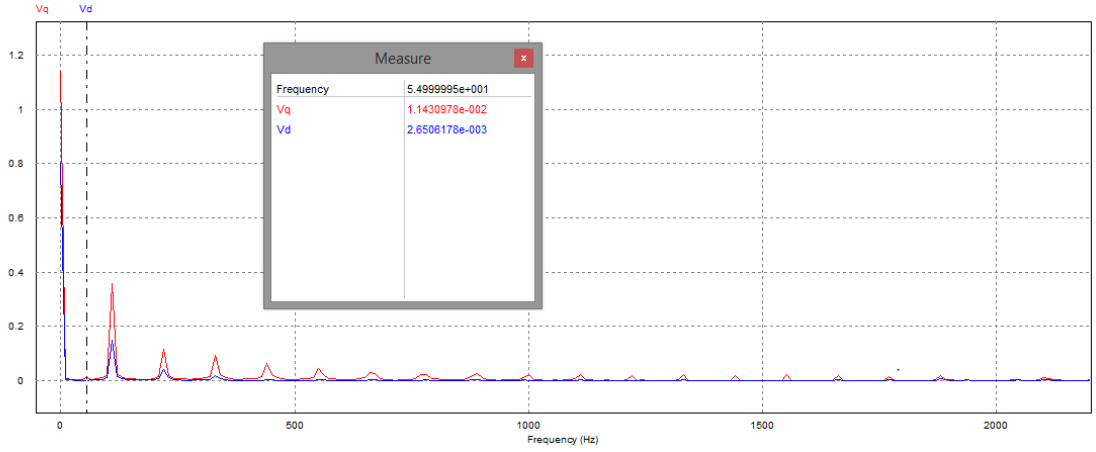


Figura 8.36

Armónico 2 (110Hz) debido a los desequilibrios:

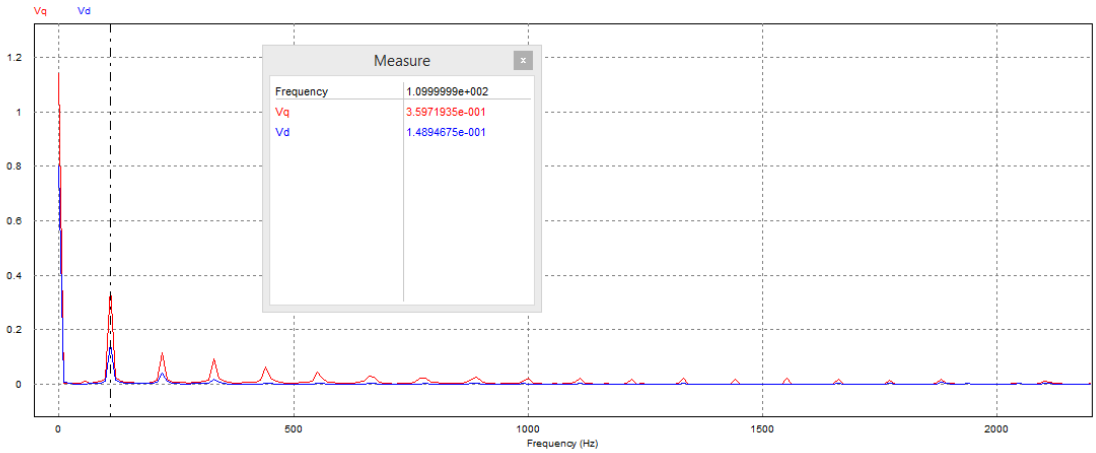


Figura 8.37

Armónico 4 (220Hz), armónico par debido a la presencia de armónicos en la red:

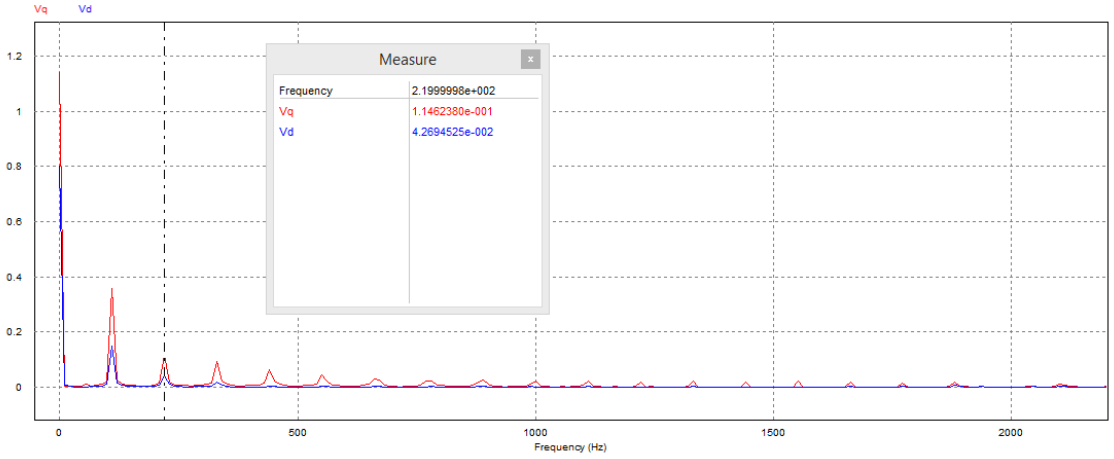


Figura 8.38

8.6 Tensiones con componente continua, armónicos y desequilibrio

Este caso es similar al anterior y los resultados obtenidos deberán ser parecidos.

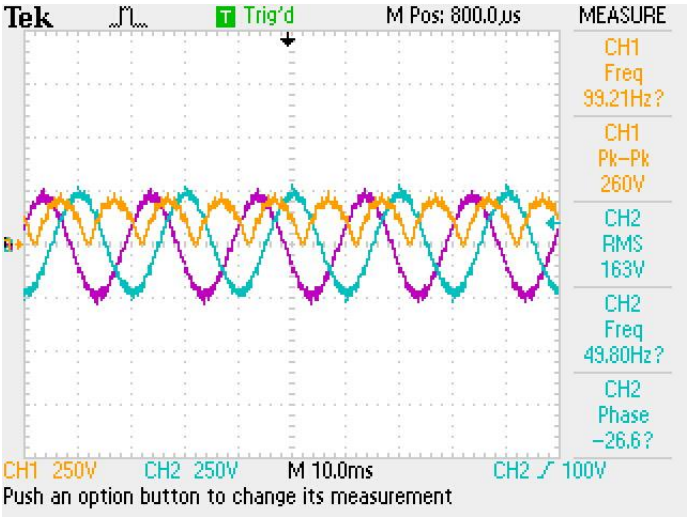


Figura 8.39

50Hz:

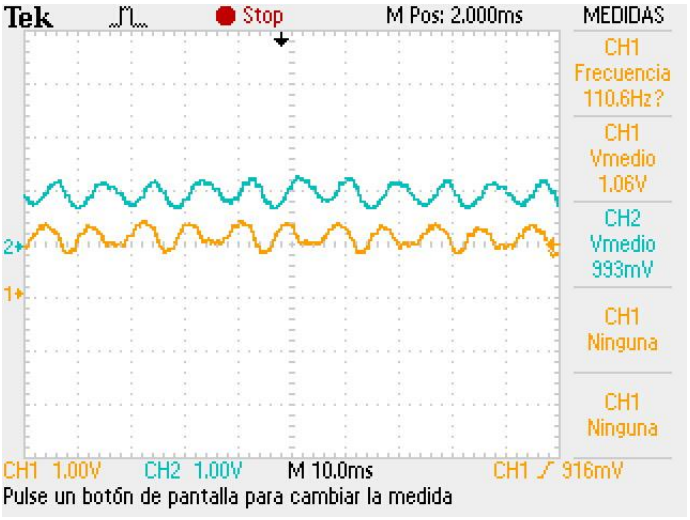


Figura 8.40

Armónicos de las fuentes:

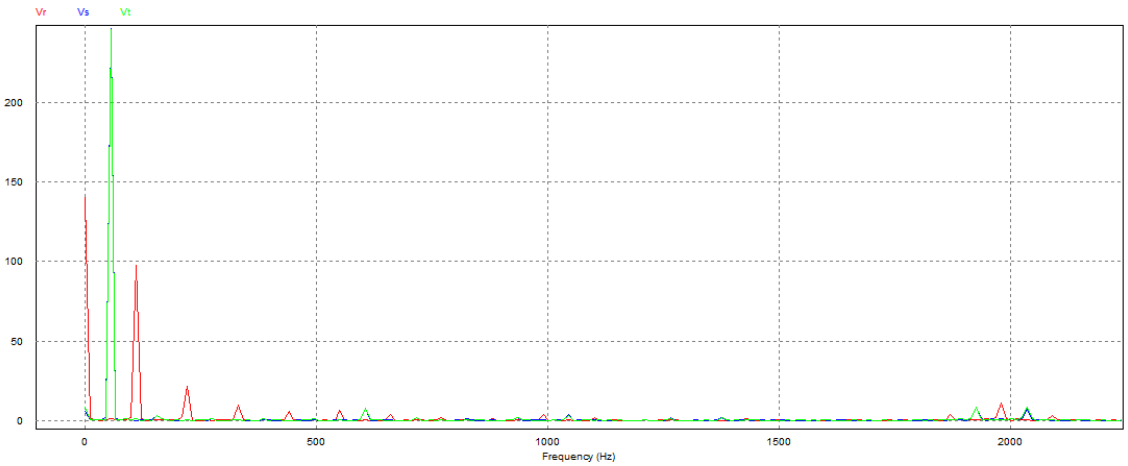


Figura 8.41

En las fuentes aparecen el armónico 0, en la que tiene componente continua y los pares (2, 4, 6…) debido a la presencia de armónicos en la red.

Vd y Vq:

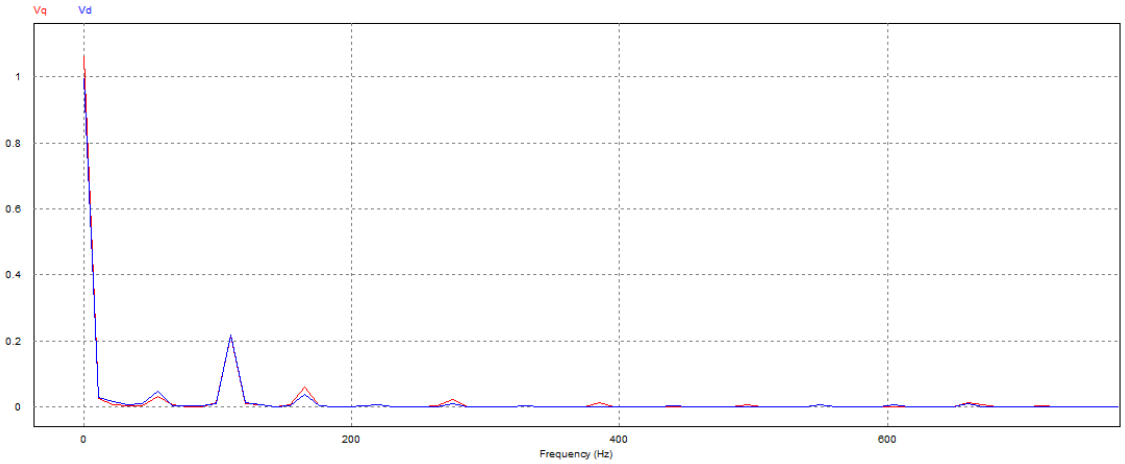


Figura 8.42

En Vd aparece la componente fundamental a 0Hz, continua, que es como se tiene que ver la tensión en Vd ya que los ejes d y q giran a la frecuencia de la red. En Vq aparece una componente a 0Hz que es la que se le suma para que ARDUINO pueda dar la salida completa y la debida al armónico 1 de la red.

En Vd y Vq aparece un armónico a 100Hz, doble de la frecuencia fundamental, debido al desequilibrio en la red. Además aparece un armónico a 50Hz debido a la componente continua y por último los armónicos impares debido a la presencia de armónicos en la red.

55Hz: deberán aparecer los mismos

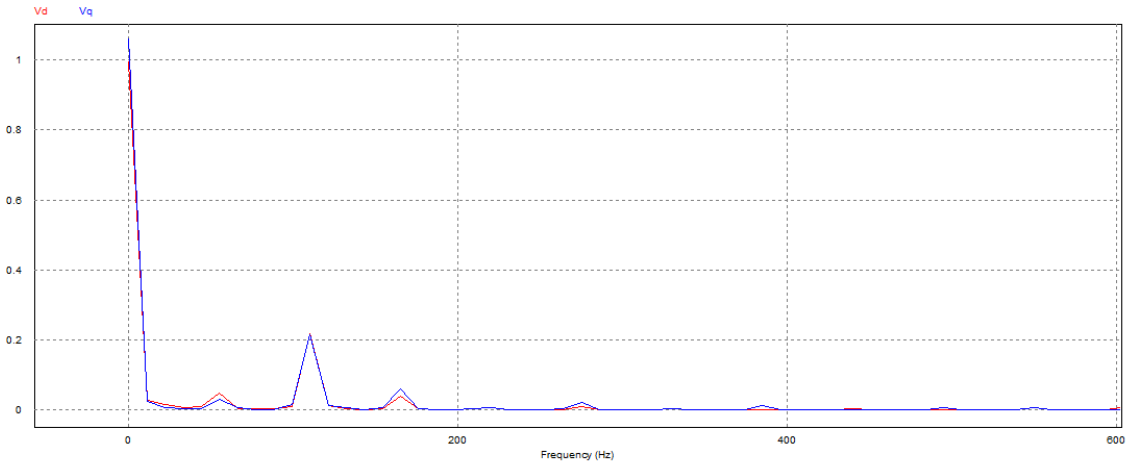


Figura 8.43

9 CONCLUSIONES

El algoritmo PLL es crucial para aplicaciones de electrónica de potencia como el control de convertidores electrónicos. La detección del ángulo y la frecuencia reales de la red se puede complicar cuando esta presenta componente continua, desequilibrios o distorsión armónica. Para que no le afecten es necesario filtrar la tensión de red antes de calcular el ángulo mediante el PLL. En la actualidad existen dos formas de hacerlo, una utilizando filtros en αβ y otra haciéndolo en dq. Se propone una nueva forma híbrida utilizando un filtro en αβ para eliminar la componente continua y uno en dq para eliminar la secuencia inversa de los desequilibrios.

Los tres PLL se programan en ARDUINO DUE. La conclusión es que pese a ser mucho más potente que ARDUINO UNO sigue teniendo una velocidad pequeña para aplicaciones que requieren una alta frecuencia de muestreo y un coste computacional elevado como en este caso. De modo que no fue posible programar los PLL para una frecuencia de muestreo de 20KHz como se pretendía sino a 2KHz. La ventaja de usar ARDUINO es su sencillez de programación.

En cuanto a los resultados obtenidos en las pruebas de comparación de los 3 PLL se observa que todos son capaces de generar el ángulo correctamente aunque la red presente alteraciones. Se ha podido comprobar cómo el PLL propuesto no tiene nada que envidiar en su respuesta a los otros 2 que se utilizan actualmente e incluso en la pruebas de calidad presenta mejores resultados que estos. Aunque los errores son tan pequeños que no es concluyente dado que el ruido en la medición a esta escala tan pequeña de error pasa a ser un factor importante.

En cuanto a las pruebas de respuesta temporal el PLL diseñado se encuentra en un segundo puesto en velocidad a poca distancia del primero que es el algoritmo PLL con filtros en αβ. Mientras que el PLL con filtros en dq es el más lento con diferencia.

Por último mediante las pruebas de potencia se comprueba que los armónicos más comunes en la red son el -5 y el 7. Los de secuencia directa aparecen en dq a (n-1)w y los de secuencia inversa a (n+1)w. La componente continua en una fase aparece en dq a la frecuencia fundamental y la secuencia inversa de los desequilibrios al doble de la frecuencia fundamental. Todo ello es observable en las tensiones Vd y Vq siempre y cuando el ángulo obtenido por el PLL sea el correcto. Analizando los resultados de las pruebas de potencia se puede concluir con que todos los valores observados son los correctos lo que indica que el algoritmo PLL diseñado trabaja correctamente en cualquier situación de la red.

10 BIBLIOGRAFÍA

[1] "Five Approaches to Deal With Problem of DC Offset in Phase-Locked Loop Algorithms: Design Considerations and Performance Evaluations". Saeed Golestan, Senior Member, IEEE, Josep M. Guerrero, Fellow, IEEE, and Gevork B. Gharehpetian, Senior Member, IEEE.

[2] G. Miramontes, ISBN 968-5923-15-9 PDS: "Introducción con teoría y práctica". Diseño de filtros de respuesta finita.

[3] "Prácticas de laboratorio con ARDUINO DUE". M.C.Gilberto Santillán Tovar y DR. Daniel U. Campos Delgado, Facultad de ciencias UASLP.

[4] "Prácticas ampliación de accionamientos: programación de microcontrolador". UPNA

[5] Xabier Juankorena Saldias, tesis doctoral realizada en la Universidad Pública de Navarra. "Conversión de energía en generadores eólicos con MSIP de gran potencia", Pamplona 2014.

11 ANEXO 1: Progrmación en ARDUINO del PLL1

```
/**Declaración de varibles**/

double Suma,pi=3.14159265358979323846264338327950288419716939937 51;

/**PLL**/

double AngClarke, Senoidal1, Senoidal2;
double VaFiltrada, VbFiltrada, prueba=0;
double VdCCF, VqCCF, w_2CCF;
double VqFiltrada, VdFiltrada;
double Kpl2=1147.49724, Tnl2=0.00173685;
double Kpl3=16.3375182, Tnl3=0.0928889;
double Kpl=55.8944216,Tnl=0.03083307;
double Tm=1./2000;
double Vd3, Vq3, w3,w4, VCCF3, error_acumCCF3, w_piCCF3,w_1CCF3,Division;
long Tmuestreo=500, Pasado=0, Ahora, Cambio;
unsigned char ciclo=0;
int Vr, Vs, Vt;
long double Va, Vb;

double Vd_red,Vq_red,principio=0;
double Vdq,Vq_pu,error_acum,w_pi,w_1,w_2,w=2*pi, w_dq;

/**Variables PLL DQ**/

double Vd_red3,Vd_red4, Vq_red3, w_2DQ, sum, sum2, lleno, lleno2, Vbusmed, Vbusmed2,
VDQ, error_acumDQ;
double w_piDQ, w_1DQ;

/**Variables filtro PLL**/
```

```
int j, k, e, q, n, m,s;
double suma, suma1,suma2,suma3,suma4,Vdfiltrada1,Vdfiltrada2,Vqfiltrada1,Vqfiltrada2;
double Vent[200], Vent2[200], Vent5[100],Vent6[100];



/**Coeficientes Filtros**/
double xa2,xa1,xa0,ya2,ya1,ya0;
double ca2,ca1,ca0,ua2,ua1,ua0;
double va2,va1,va0,oa2,oa1,oa0;
double ba2,ba1,ba0,pa2,pa1,pa0;

double xb2,xb1,xb0,yb2,yb1,yb0;
double cb2,cb1,cb0,ub2,ub1,ub0;
double vb2,vb1,vb0,ob2,ob1,ob0;
double bb2,bb1,bb0,pb2,pb1,pb0;

double xq2,xq1,xq0,yq2,yq1,yq0;
double cq2,cq1,cq0,uq2,uq1,uq0;
double vq2,vq1,vq0,oq2,oq1,oq0;
double bq2,bq1,bq0,pq2,pq1,pq0;

double xd2,xd1,xd0,yd2,yd1,yd0;
double cd2,cd1,cd0,ud2,ud1,ud0;
double vd2,vd1,vd0,od2,od1,od;
double bd2,bd1,bd0,pd2,pd1,pd0;
```

```
/**FILTRO**/



/**PLL**/



double  error_acumCCF, VCCF, w_piCCF, w_1CCF, VqCCF1, VdCCF1;



/**Pasobanda a 50Hz**/

double nu2=1,nu1=0,nu0=-1,de2=1,de1=-1.9935003467427104,de0=0.99373647154162037,
ganancia2=0.0031317642291898312;



/**Pasobanda a 50Hz de orden 4**/

double nume2=1,nume1=0,nume0=-1,deno2=1,deno1=-
1.9982879151587798,deno0=0.99855677281431143, ganancia3=0.0009774756818565029;

double numer2=1,numer1=0,numer0=-1,denom2=1,denom1=-
1.998453116015676,denom0=0.99867851282493636, ganancia4=0.0009774756818565029;



/**Notch a 100Hz**/

double numera2=1,numera1=-1.9990229855063772,numera0=1,denomi2=1,denomi1=-
1.9927625168270415,denomi0=0.9937364715416217,ganancia8=0.99686823577081085;



/**Pasobanda a 100Hz**/

long double numerad2=1,numerad1=0,numerad0=-1,denomin2=1,denomin1=-
1.7923565998958046,denomin0=0.88366532316014657,ganancia9=0.058167338419926731;

double VqBanda, VdBanda;
```

```
/**Filtro DSC4**/

double Vent7[100],Vent8[100];



/**Filtro DSC para eliminar continua**/

double Ven[2], Ven2[2];

/**Notch a 50Hz**/
long double nun2=1,nun1=-1.9753587482879427,nun0=1,denn2=1,denn1=-
1.9151720421718614,denn0=0.93906250581749229,ganancian2=0.9695312529087462;



long double cna2, cna1, cna0, una2,una1;
long double una0;
long double cnb2, cnb1, cnb0, unb2,unb1;
long double unb0;

double pasadofrecuencia=2*pi*50,anteriorfrecuencia=2*pi*50,kanterior=-
1.9151720421718614,kanterior2=-1.9753587482879427;

/**Señal de referencia para calcular el error)**/
long double Frecuencia, Clarkeantiguo,SenoidalError;
long double  correcto, FrecuenciaReal,AnguloReal2, F,FrecuenciaReal2, AnguloReal;
unsigned long Suma5, An, An2, suma12, con;
long double Continua, VrFiltrada,tiempo6,tiempo2, tiempoFrecuencia, tiempo5, AnguloReal3;
unsigned long  Suma7,reinicio;
```

```
void setup() {

/**Definición de entradas**/
/**Las entradas analógicas no hacen falta incluirlas en el setup, puesto que
esos pines (A0-A11) solo pueden ser entradas analógicas.**/
/**Defino la salida 4 digital**/
  pinMode(13, OUTPUT);
 //iniciamos el puerto de serie
  Serial.begin(9600);
analogWriteResolution(12); // Define resolución del DAC
analogReadResolution(12); // Define resolución del DAC

}

void loop() {



Ahora=micros();
Cambio=Ahora-Pasado;
if(Cambio>=Tmuestreo)
{

digitalWrite(13,HIGH);

Vr=analogRead(A0)-2048;
Vs=analogRead(A1)-2048;
Vt=analogRead(A2)-2048;

prueba++;

/**PLL**/
```

```
/**PLL1**/


/**Cálculo del ángulo mediante Clarke para comprobación del PLL**/

Va=(2./3)*(Vr-0.5*Vs-0.5*Vt);
Vb=(2./3)*((sqrt(3)/2)*Vs-(sqrt(3)/2)*Vt);
AngClarke=atan2(Vb,Va);

if(AngClarke<0)
{
AngClarke=AngClarke+2*pi;
}


nun1=0.0000097522*(w_piCCF*w_piCCF/(2*pi*2*pi))+0.00000807*w_piCCF/(2*pi)-2.0011;
denn1=0.000009455*(w_piCCF*w_piCCF/(2*pi*2*pi))+0.0000078241*w_piCCF/(2*pi)-1.9402;



/**Filtro notch a 50Hz para elimianar frecuencia continua. Orden 2**/

cna2=cna1;
cna1=cna0;
cna0=Va;
una2=una1;
una1=una0;

una0=ganancian2*(nun0*cna2+nun1*cna1+nun2*cna0)-denn1*una1-denn0*una2;
```

```
VaFiltrada=Va-una0;



cnb2=cnb1;
cnb1=cnb0;
cnb0=Vb;
unb2=unb1;
unb1=unb0;

unb0=ganancian2*(nun0*cnb2+nun1*cnb1+nun2*cnb0)-denn1*unb1-denn0*unb2;

VbFiltrada=Vb-unb0;

/**
Continua=sqrt(una0*una0+unb0*unb0);
VrFiltrada=Vr-Continua;
tiempo5=millis()-tiempo6;
if(VrFiltrada<200){if(VrFiltrada>-200){if(tiempo5<5){}else{tiempoFrecuencia=(millis()-
tiempo6);tiempo6=millis();}}}
FrecuenciaReal2=1./(2*tiempoFrecuencia);
AnguloReal2=double(FrecuenciaReal2)*2*pi*millis()-An2*2*pi;
if(AnguloReal2>2*pi){AnguloReal2=AnguloReal2-2*pi;An2++;}**/



/**Cálculo de ángulo de referencia para desequilibrios**/

tiempo5=micros()-tiempo6;
if(Vr<150){if(Vr>-150){
```

```
 if(con<1){
  if(tiempo5>5000){con++;}}
 if(con>0){
  if(tiempo5>15000){tiempoFrecuencia=micros()-
tiempo6;tiempo6=micros();con=0;AnguloReal2=0;digitalWrite(13,LOW);}}}}

FrecuenciaReal2=1./(tiempoFrecuencia/1000000);



AnguloReal2=FrecuenciaReal2*2*pi*tiempo5/1000000;
AnguloReal3=AnguloReal2+(pi/2);
if(AnguloReal3>(2*pi)){AnguloReal3=AnguloReal3-(2*pi);}
/**if(AnguloReal2>2*pi){AnguloReal2=AnguloReal2-(2*pi);An2++;}**/



/** if(con<1){
   if(tiempo5>5000){con++;{if(con>1){tiempoFrecuencia=micros()-
tiempo6;tiempo6=micros();con=0;}}}}**/



/**Transformada alfa-beta a dq**/

VdCCF=cos(w_2CCF)*VaFiltrada+sin(w_2CCF)*VbFiltrada;
VqCCF=-sin(w_2CCF)*VaFiltrada+cos(w_2CCF)*VbFiltrada;



/**Filtro pasobanda a 100Hz y resta**/
```

```
xd2=xd1;

xd1=xd0;

xd0=VdCCF;

yd2=yd1;

yd1=VdBanda;


VdBanda=ganancia9*(numerad0*xd2+numerad1*xd1+numerad2*xd0)-denomin1*yd1-
denomin0*yd2;

VdFiltrada=VdCCF-VdBanda;


xq2=xq1;

xq1=xq0;

xq0=VqCCF;

yq2=yq1;

yq1=VqBanda;


VqBanda=ganancia9*(numerad0*xq2+numerad1*xq1+numerad2*xq0)-denomin1*yq1-
denomin0*yq2;


VqFiltrada=VqCCF-VqBanda;




/**Algoritmo PLL**/


VCCF=VqFiltrada/VdFiltrada;

error_acumCCF=error_acumCCF+VCCF*Tm;

w_piCCF=Kpl*( (error_acumCCF)/Tnl);

w_1CCF=w_piCCF+Kpl*VCCF;

w_2CCF=w_2CCF+w_1CCF*Tm;
```

```
if(w_2CCF>2*pi)
{
w_2CCF=0;
}


/**Código para que Vd se alinee en con el eje positivo


if(AngClarke>0.6){
if(AngClarke<1.4*pi){
if((AngClarke-w_2CCF)<-pi/2){w_2CCF=w_2CCF+pi;}
else{
if((AngClarke-w_2CCF)>pi/2){w_2CCF=w_2CCF+pi;}}}}
if(w_2CCF>2*pi){w_2CCF=w_2CCF-2*pi;}**/




/**Prueba 2 para sacar la señal de referencia para armónicos
Suma5++;

if(Suma7<100000000){

if(AngClarke>3){
correcto=1;}
tiempo5=micros()-tiempo6;
if(AngClarke<0.4){
if(correcto>0){
F++;reinicio++;tiempoFrecuencia=micros()-tiempo6;tiempo6=micros(); if
(reinicio<2){F=0;tiempo2=millis();}
```

```
if(reinicio<2){AnguloReal=AngClarke;}else{FrecuenciaReal=1/(tiempoFrecuencia/1000000);Ang
uloReal=0;}


}
correcto=0;
}
}

AnguloReal=FrecuenciaReal*2*pi*tiempo5/1000000;
if(AnguloReal>2*pi){AnguloReal=AnguloReal-(2*pi);}**/

/**Cálculo de Vd y Vq
VdCCF1=cos(w_2CCF)*Va+sin(w_2CCF)*Vb;
VqCCF1=-sin(w_2CCF)*Va+cos(w_2CCF)*Vb;**/

/**w3=w_2CCF;
w4=w3*4096/(2*pi);

Senoidal1=cos((w3));
Senoidal2=(Senoidal1+1)*2000;**/

/**w4=w_2CCF+(pi/2);
if(w4>(2*pi)){w4=w4-2*pi;}**/

/**Escribo en el pin 1 de salidas analógicas el valor del ángulo**/
analogWrite(DAC1,w_2CCF*4000/(2*pi));
analogWrite(DAC0,w_1CCF*7);
Pasado=Ahora;
}
}
```

12 ANEXO 2: Progrmación en ARDUINO del PLLdq

```
/**Declaración de varibles**/

double Suma,pi=3.14159265358979323846264338327950288419716939937751;

/**PLL**/

double Va, Vb,AngClarke, Senoidal1, Senoidal2;
double VaFiltrada, VbFiltrada, prueba=0;
double VdCCF, VqCCF, w_2CCF;
double VqFiltrada, VdFiltrada;
double Kpl2=1147.49724, Tnl2=0.00173685;
double Kpl3=16.3375182, Tnl3=0.0928889;
double Kpl4=30.3375182, Tnl4=0.0528889;
double Kpl5=75.3375182, Tnl5=0.0228889;
double Kpl=55.8944216,Tnl=0.03083307;
double KplR=54.81747361, TnlR=0.028408946;
double Tm=1./2000;
double Vd3, Vq3, w3,w4, VCCF3, error_acumCCF3,
w_piCCF3,w_1CCF3,Division,Seno,Coseno,Angulo;
long Tmuestreo=500, Pasado=0, Ahora, Cambio;
unsigned char ciclo=0;
int Vr, Vs, Vt;



double Vd_red,Vq_red,principio=0;

double Vdq,Vq_pu,error_acum,w_pi,w_1,w_2,w=2*pi, w_dq;
```

```
/**Variables PLL DQ**/

double Vd_red3,Vd_red4, Vq_red3, w_2DQ=0, sum, sum2, lleno, lleno2, Vbusmed, Vbusmed2,
VDQ, error_acumDQ;
double w_piDQ, w_1DQ;

/**Variables filtro PLLdq**/

int j, k, e, q;
double suma, suma1,suma2,suma3,suma4,Vdfiltrada1,Vdfiltrada2,Vqfiltrada1,Vqfiltrada2;
double Vent[20], Vent2[20], Vent5[10],Vent6[10];



/**Coeficientes Filtros**/
double xa2,xa1,xa0,ya2,ya1,ya0;
double ca2,ca1,ca0,ua2,ua1,ua0;
double va2,va1,va0,oa2,oa1,oa0;
double ba2,ba1,ba0,pa2,pa1,pa0;

double xb2,xb1,xb0,yb2,yb1,yb0;
double cb2,cb1,cb0,ub2,ub1,ub0;
double vb2,vb1,vb0,ob2,ob1,ob0;
double bb2,bb1,bb0,pb2,pb1,pb0;

double xq2,xq1,xq0,yq2,yq1,yq0;
double cq2,cq1,cq0,uq2,uq1,uq0;
double vq2,vq1,vq0,oq2,oq1,oq0;
double bq2,bq1,bq0,pq2,pq1,pq0;

double xd2,xd1,xd0,yd2,yd1,yd0;
```

```
double cd2,cd1,cd0,ud2,ud1,ud0;

double vd2,vd1,vd0,od2,od1,od;

double bd2,bd1,bd0,pd2,pd1,pd0;



/**Pasobajo a 3Hz**/

double num2=1,num1=2,num0=1,den2=1,den1=-
1.998667135315678,den0=0.99866802298843449,ganancia=0.00000022191818912819991;



/**Señal de referencia para calcular el error)**/

long double Frecuencia, Clarkeantiguo,SenoidalError;

long double  correcto, FrecuenciaReal,AnguloReal2, F,FrecuenciaReal2,
AnguloReal,AnguloReal3;

unsigned long Suma5, An, An2, suma12, con;

long double Continua, VrFiltrada,tiempo6,tiempo2, tiempoFrecuencia, tiempo5;

unsigned long  Suma7,reinicio;



void setup() {

/**Definición de entradas**/

/**Las entradas analógicas no hacen falta incluirlas en el setup, puesto que

esos pines (A0-A11) solo pueden ser entradas analógicas.**/

/**Defino la salida 4 digital**/

  pinMode(13, OUTPUT);

analogWriteResolution(12); // Define resolución del DAC

analogReadResolution(12); // Define resolución del DAC
```

```
}

void loop() {



Ahora=micros();
Cambio=Ahora-Pasado;
if(Cambio>=Tmuestreo)
{

digitalWrite(13,HIGH);

Vr=analogRead(A0)-2048;
Vs=analogRead(A1)-2048;
Vt=analogRead(A2)-2048;

prueba++;




/**Cálculo de ángulo de referencia para desequilibrios**/
tiempo5=micros()-tiempo6;
if(Vr<150){if(Vr>-150){
  if(con<1){
    if(tiempo5>5000){con++;}}
  if(con>0){
    if(tiempo5>15000){tiempoFrecuencia=micros()-
tiempo6;tiempo6=micros();con=0;AnguloReal2=0;digitalWrite(13,LOW);}}}}

FrecuenciaReal2=1./(tiempoFrecuencia/1000000);
```

```
AnguloReal2=FrecuenciaReal2*2*pi*tiempo5/1000000;
AnguloReal3=AnguloReal2+(pi/2);
if(AnguloReal3>(2*pi)){AnguloReal3=AnguloReal3-(2*pi);}


/**PLL dq**/


/**Transformada park Vred**/

Vd_red3=sqrt(2./3)*(cos(w_2DQ)*Vr+cos(w_2DQ-(2*pi/3))*Vs+cos(w_2DQ+(2*pi/3))*Vt);
Vq_red3=sqrt(2./3)*(-sin(w_2DQ)*Vr-sin(w_2DQ-(2*pi/3))*Vs-sin(w_2DQ+(2*pi/3))*Vt);


/**Filtro DSC a 50Hz para eliminar la continua**/

/**Vd**/

Vent[j]=Vd_red3;
j++;
if(j>19){j=0;}
Vdfiltrada1=(Vd_red3)/2+(Vent[j])/2;

/**Vq**/


Vent2[k]=Vq_red3;
k++;
```

```
if(k>19){k=0;}
Vqfiltrada1=Vq_red3/2+Vent2[k]/2;


/**Filtro DSC a 100Hz para eliminar desequilibrios**/

/**Vd**/

Vent5[q]=Vdfiltrada1;
q++;
if(q>9){q=0;}
Vdfiltrada2=Vdfiltrada1/2+Vent5[q]/2;


/**Vq**/

Vent6[e]=Vqfiltrada1;
e++;
if(e>9){e=0;}
Vqfiltrada2=Vqfiltrada1/2+Vent6[e]/2;


VDQ=Vqfiltrada2/Vdfiltrada2;
error_acumDQ=error_acumDQ+VDQ*Tm;
w_piDQ=Kpl*(VDQ+ (error_acumDQ)/Tnl);
w_1DQ=w_piDQ+2*pi*50;
w_2DQ=w_2DQ+w_1DQ*Tm;

if(w_2DQ>2*pi)
{
w_2DQ=0;
```

```
}

/****/
Va=(2./3)*(Vr-0.5*Vs-0.5*Vt);
Vb=(2./3)*((sqrt(3)/2)*Vs-(sqrt(3)/2)*Vt);
AngClarke=atan2(Vb,Va);

if(AngClarke<0)
{
AngClarke=AngClarke+2*pi;
}

/**Codigo para Vd positiva
if(AngClarke>0.4){
if(AngClarke<1.6*pi){
if((AngClarke-w_2DQ)<-pi/2){w_2DQ=w_2DQ+pi;}
else{
if((AngClarke-w_2DQ)>pi/2){w_2DQ=w_2DQ+pi;}}}}
if(w_2DQ>2*pi){w_2DQ=w_2DQ-2*pi;}

Prueba 1 para saar la señal error
if((AngClarke-Clarkeantiguo)<0){Frecuencia=(AngClarke+(2*pi-Clarkeantiguo))/0.0005;}
else{
Frecuencia=(AngClarke-Clarkeantiguo)/0.0005;

Clarkeantiguo=AngClarke;
SenoidalError=cos(Frecuencia*double(millis())/1000);**/


/**Prueba 2 para sacar la señal de referencia para armónicos
Suma5++;
```

```
if(Suma7<100000000){

if(AngClarke>3){
correcto=1;}
tiempo5=micros()-tiempo6;
if(AngClarke<0.5){
if(correcto>0){
tiempoFrecuencia=micros()-tiempo6;tiempo6=micros();
FrecuenciaReal=1/(tiempoFrecuencia/1000000);AnguloReal=0;
}
correcto=0;
}
}
AnguloReal=FrecuenciaReal*2*pi*tiempo5/1000000;
if(AnguloReal>2*pi){AnguloReal=AnguloReal-(2*pi);}**/

/**w3=w_2DQ;
w4=w3*4096/(2*pi);**/

/**Escribo en el pin 1 de salidas analógicas el valor del ángulo**/
analogWrite(DAC1,AnguloReal3*4000/(2*pi));
analogWrite(DAC0,w_1DQ*7);
Pasado=Ahora;
digitalWrite(13,LOW);
}
}
```

13 ANEXO 3: Progrmación en ARDUINO del PLLccf

```
#include "pwm3.h"
#include "math.h"



/**Declaración de varibles**/


double Suma,pi=3.14159265358979323846264338327950288419716939937 51;


/**PLL**/


double Va, Vb,AngClarke, Senoidal1, Senoidal2, CicloA,CicloB,CicloC;
double VaFiltrada, VbFiltrada, prueba=0;
double VdCCF, VqCCF, w_2CCF;
double VqFiltrada, VdFiltrada;
double Kpl2=1147.49724, Tnl2=0.00173685;
double Kpl3=16.3375182, Tnl3=0.0928889;
double Kpl=55.8944216,Tnl=0.03083307;
double Tm=1./2000;
double Vd3, Vq3, w3,w4, VCCF3, error_acumCCF3, w_piCCF3,w_1CCF3,Division,
Angulo,Angulo2, Seno, Coseno;
long Tmuestreo=500, Pasado=0, Ahora, Cambio;
unsigned char ciclo=0;
int Vr, Vs, Vt;



double Vd_red,Vq_red,principio=0;


double Vdq,Vq_pu,error_acum,w_pi,w_1,w_2,w=2*pi, w_dq;
```

```
/**Coeficientes Filtros**/
double xa2,xa1,xa0,ya2,ya1,ya0;
double ca2,ca1,ca0,ua2,ua1,ua0;
double va2,va1,va0,oa2,oa1,oa0;
double ba2,ba1,ba0,pa2,pa1,pa0;

double xb2,xb1,xb0,yb2,yb1,yb0;
double cb2,cb1,cb0,ub2,ub1,ub0;
double vb2,vb1,vb0,ob2,ob1,ob0;
double bb2,bb1,bb0,pb2,pb1,pb0;

double xq2,xq1,xq0,yq2,yq1,yq0;
double cq2,cq1,cq0,uq2,uq1,uq0;
double vq2,vq1,vq0,oq2,oq1,oq0;
double bq2,bq1,bq0,pq2,pq1,pq0;

double xd2,xd1,xd0,yd2,yd1,yd0;
double cd2,cd1,cd0,ud2,ud1,ud0;
double vd2,vd1,vd0,od2,od1,od;
double bd2,bd1,bd0,pd2,pd1,pd0;


/**Variables PLL CCF**/

double Va2, Va3, Va4, Va5, Va6, Vb2, Vb3,Vb4,Vb5,Vb6, sum3, vent3[400], sum4,
vent4[400],VCCF,error_acumCCF,w_piCCF,w_1CCF;
double lleno3, lleno4, l, z,error_acumA=0, error_acumB=0, Wg=2*pi*50, Wp=2*pi*15;
```

```
/**Pasobajo a 3Hz**/
double num2=1,num1=2,num0=1,den2=1,den1=-1.998667135315678,den0=0.99866802298843449,ganancia=0.00000022191818912819991;


/**Señal de referencia para calcular el error)**/
long double Frecuencia, Clarkeantiguo,SenoidalError;
long double  correcto, FrecuenciaReal,AnguloReal2, F,FrecuenciaReal2, AnguloReal, AnguloReal3;
unsigned long Suma5, An, An2, suma12, con;
long double Continua, VrFiltrada,tiempo6,tiempo2, tiempoFrecuencia, tiempo5;
unsigned long  Suma7,reinicio;



void setup() {

/**Definición de entradas**/
/**Las entradas analógicas no hacen falta incluirlas en el setup, puesto que
esos pines (A0-A11) solo pueden ser entradas analógicas.**/
/**Defino la salida 4 digital**/
  pinMode(13, OUTPUT);
  pinMode(34, OUTPUT);

analogWriteResolution(12); // Define resolución del DAC
analogReadResolution(12); // Define resolución de la entrada analógica;
pwm3_setup(10000,1,0);
pwm3_start();
```

```
}

void loop() {



Ahora=micros();
Cambio=Ahora-Pasado;
if(Cambio>=Tmuestreo)
{
digitalWrite(13,HIGH);



Vr=analogRead(A0)-2048;
Vs=analogRead(A1)-2048;
Vt=analogRead(A2)-2048;

prueba++;

/**Cálculo de ángulo de referencia para desequilibrios**/

tiempo5=micros()-tiempo6;
if(Vr<150){if(Vr>-150){
  if(con<1){
    if(tiempo5>5000){con++;}}
  if(con>0){
    if(tiempo5>15000){tiempoFrecuencia=micros()-
tiempo6;tiempo6=micros();con=0;AnguloReal2=0;digitalWrite(13,LOW);}}}}
```

```
FrecuenciaReal2=1./(tiempoFrecuencia/1000000);



AnguloReal2=FrecuenciaReal2*2*pi*tiempo5/1000000;
AnguloReal3=AnguloReal2+(pi/2);
if(AnguloReal3>(2*pi)){AnguloReal3=AnguloReal3-(2*pi);}
/**if(AnguloReal2>2*pi){AnguloReal2=AnguloReal2-(2*pi);An2++;}**/

/**Cálculo del ángulo mediante Clarke para comprobación del PLL**/

Va=(2./3)*(Vr-0.5*Vs-0.5*Vt);
Vb=(2./3)*((sqrt(3)/2)*Vs-(sqrt(3)/2)*Vt);
AngClarke=atan2(Vb,Va);

if(AngClarke<0)
{
AngClarke=AngClarke+2*pi;
}



/**PLL CCF**/

Va2=Va-Va4;
Vb2=Vb-Vb4;
Va6=Va-Va5;
Vb6=Vb-Vb5;



/**Filtro paso bajo**/
```

```
xd2=xd1;
xd1=xd0;
xd0=Va2;
yd2=yd1;
yd1=Va5;

Va5=ganancia*(num0*xd2+num1*xd1+num2*xd0)-den1*yd1-den0*yd2;



xq2=xq1;
xq1=xq0;
xq0=Vb2;
yq2=yq1;
yq1=Vb5;

Vb5=ganancia*(num0*xq2+num1*xq1+num2*xq0)-den1*yq1-den0*yq2;



/**Filtro paso banda**/



Va3=Va6*Wp-Va4*Wp-Vb4*Wg;
error_acumA=error_acumA+Va3*Tm;
Va4=Kpl*(Va3+ (error_acumA)/Tnl);



Vb3=Vb6*Wp-Vb4*Wp+Va4*Wg;
error_acumB=error_acumB+Vb3*Tm;
Vb4=Kpl*(Vb3+ (error_acumB)/Tnl);
```

```
VdCCF=cos(w_2CCF)*Va6+sin(w_2CCF)*Vb6;
VqCCF=-sin(w_2CCF)*Va6+cos(w_2CCF)*Vb6;


VCCF=VqCCF/VdCCF;
error_acumCCF=error_acumCCF+VCCF*Tm;
w_piCCF=Kpl*(VCCF+ (error_acumCCF)/Tnl);
w_1CCF=w_piCCF+2*pi*50;
w_2CCF=w_2CCF+w_1CCF*Tm;

if(w_2CCF>2*pi)
{
w_2CCF=0;
}
```

```
/**Código para Vd positiva**/
if(AngClarke>0.3){
if(AngClarke<1.7*pi){
if((AngClarke-w_2CCF)<-pi/2){w_2CCF=w_2CCF+pi;}
else{
if((AngClarke-w_2CCF)>pi/2){w_2CCF=w_2CCF+pi;}}}}
if(w_2CCF>2*pi){w_2CCF=w_2CCF-2*pi;}
```

```
/** Prueba para sacar el ángulo de Vr
Seno=Vr/((2/3)*(sqrt((Vr-Vs*sin(pi/6)-Vt*sin(pi/6))*(Vr-Vs*sin(pi/6)-Vt*sin(pi/6))+(-
Vs*cos(pi/6)+Vt*cos(pi/6))*(-Vs*cos(pi/6)+Vt*cos(pi/6)))));
Coseno=sqrt(1-Seno*Seno);
Angulo=atan2(Seno,Coseno);
if(Angulo<0)
{
Angulo=Angulo+2*pi;
}

if((Angulo-w_2DQ)<(-pi/6-pi/2)){w_2DQ=Angulo-pi/2;}
if((Angulo-w_2DQ)>(pi/6-pi/2)){w_2DQ=Angulo-pi/2;}
if(w_2DQ>2*pi)
{
w_2DQ=w_2DQ-2*pi;
}**/


/**Prueba 2 para sacar la señal de referencia para armónicos
Suma5++;

if(Suma7<100000000){

if(AngClarke>3){
correcto=1;}
tiempo5=micros()-tiempo6;
if(AngClarke<0.5){
if(correcto>0){
```

```
F++;reinicio++;tiempoFrecuencia=micros()-tiempo6;tiempo6=micros(); if
(reinicio<2){F=0;tiempo2=millis();}

if(reinicio<2){AnguloReal=AngClarke;}else{FrecuenciaReal=1/(tiempoFrecuencia/1000000);Ang
uloReal=0;}



}

correcto=0;

}

}



AnguloReal=FrecuenciaReal*2*pi*tiempo5/1000000;

if(AnguloReal>2*pi){AnguloReal=AnguloReal-(2*pi);}**/



/**Angulo2=AngClarke*4096/(2*pi);**/

/**Escribo en el pin 1 de salidas analógicas el valor del ángulo**/

analogWrite(DAC1,AnguloReal3*4000/(2*pi));

analogWrite(DAC0,w_1CCF*7);

/**CicloA=(Senoidal1+1)/2;

CicloB=(Senoidal1+1)/2;

CicloC=VdCCF/4096;

pwm3_setDutyCycles(CicloA, CicloB, CicloC);**/

Pasado=Ahora;

}

}
```