UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DEPARTAMENTO DE INGENIERÍA ELECTRÓNICA

# UNIVERSITAT POLITÈCNICA DE VALÈNCIA

## Dpto. de Ingeniería Electrónica

# Diseño e implementación de un control híbrido Six-Step - FOC mediante SOGI-FLL de motores BLDC con sensores de efecto hall

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Sistemas Electrónicos

AUTOR/A: Martinez Lopez, Pablo

Tutor/a: González Medina, Raúl

CURSO ACADÉMICO: 2023/2024

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

# DISEÑO E IMPLEMENTACIÓN DE UN CONTROL HÍBRIDO TRAPEZOIDAL-FOC MEDIANTE SOGI-FLL DE MOTORES BLDC CON SENSORES DE EFECTO HALL

**Pablo Martínez López**

**Tutor: Raúl González Medina**

Trabajo Fin de Máster presentado en la Escuela Técnica Superior de Ingeniería de Telecomunicación de la Universitat Politècnica de València, para la obtención del Título de Máster Universitario en Ingeniería de Sistemas Electrónicos

Curso 2023-24

València, 08 de septiembre de 2024

Escuela Técnica Superior de Ingeniería de Telecomunicación
Universitat Politècnica de València
Edificio 4D. Camino de Vera, s/n, 46022 Valencia
Tel. +34 96 387 71 90, ext. 77190
**www.etsit.upv.es**

VLC/ CAMPUS
VALENCIA, INTERNATIONAL CAMPUS OF EXCELLENCE

EMAS
Gestión ambiental verificada
REG.NO.ES-CV-000030

## Resumen

En este trabajo de fin de Máster se trata el diseño e implementación de un sistema de control híbrido para motores BLDC (Brushless DC Motor), utilizando técnicas de modulación trapezoidal y FOC (Field Oriented Control).

En este ámbito se ha introducido el concepto de SOGI-FLL (Second-Order Generalized Integrator-Frequency-Locked Loop) como una forma de estimar parámetros en el funcionamiento del motor, simplificar la interfaz mecánica de este, y proporcionar una transición suave entre los dos métodos de control. Este algoritmo utiliza de manera directa, los sensores de efecto Hall incorporados en el motor BLDC.

Como contenido del trabajo, se justificará la elección del motor BLDC entre otras tipologías para un rango de aplicaciones propuesto, y se desarrollarán los principios de los métodos de control trapezoidal y FOC, así como del algoritmo SOGI-FLL. Esto dará paso a una implementación práctica del control híbrido en un entorno de simulación, donde se obtendrán resultados que demuestren la efectividad del sistema propuesto.

Por último, se realizará la implementación de algunas de las características desarrolladas sobre un entorno de evaluación real, utilizando una placa de evaluación con DSP e instrumentación de laboratorio.



## Resum

En aquest treball de fi de Màster es tracta el disseny i implementació d'un sistema de control híbrid per a motors BLDC (Brushless DC Motor), utilitzant tècniques de modulació trapezoidal i FOC (Field Oriented Control).

En aquest àmbit s'ha introduït el concepte de SOGI-FLL (Second-Order Generalized Integrator-Frequency-Locked Loop) com una forma d'estimar paràmetres en el funcionament del motor, simplificar la interfície mecànica d'aquest, i proporcionar una transició suau entre els dos mètodes de control. Aquest algoritme utilitza de manera directa, els sensors d'efecte Hall incorporats en el motor BLDC.

Com a contingut del treball, es justificarà l'elecció del motor BLDC entre altres tipologies per a un rang d'aplicacions proposat, i es desenvoluparan els principis dels mètodes de control tra- pezoidal i FOC, així com de l'algoritme SOGI-FLL. Això donarà pas a una implementació pràctica del control híbrid en un entorn de simulació, on s'obtindran resultats que demostrin l'efectivitat del sistema proposat.

Per últim, es realitzarà la implementació d'algunes de les característiques desenvolupades sobre un entorn d'avaluació real, utilitzant una placa d'avaluació amb DSP i instrumentació de laboratori.

# Abstract

In this Master's thesis, the design and implementation of a hybrid control system for BLDC (Brushless DC Motor) motors is discussed, using trapezoidal modulation techniques and FOC (Field Oriented Control).

In this field, the concept of SOGI-FLL (Second-Order Generalized Integrator-Frequency-Locked Loop) has been introduced as a way to estimate parameters in the operation of the motor, simplify its mechanical interface, and provide a smooth transition between the two control methods. This algorithm directly uses the Hall effect sensors incorporated in the BLDC motor.

As part of the work, the choice of the BLDC motor among other typologies for a proposed range of applications will be justified, and the principles of trapezoidal control methods and FOC, as well as the SOGI-FLL algorithm, will be developed. This will lead to a practical implementation of the hybrid control in a simulation environment, where results will be obtained that demonstrate the effectiveness of the proposed system.

Lastly, the implementation of some of the developed features will be carried out in a real evaluation environment, using an evaluation board with DSP and laboratory instrumentation.

A todos mis seres queridos, que han cuidado tanto de mí y me han hecho ser quien soy.

# Índice general

# Índice de figuras

# Índice de tablas

# Parte I

# Memoria

# Capítulo 1

# INTRODUCCIÓN

## 1.1. OBJETO DEL TRABAJO

No son pocos los ámbitos de nuestras vidas donde los motores eléctricos, cada vez más, están presentes. Desde electrodomésticos o juguetes, hasta dispositivos de radiocontrol o vehículos personales. Estos dispositivos no solo contribuyen a hacer nuestras vidas más cómodas, sino también más sostenibles, reduciendo nuestra huella de carbono de manera muy significativa.

El objetivo de este Trabajo de Fin de Máster es utilizar la tecnología de los motores BLDC[I] para desarrollar un sistema de control híbrido con dos estrategias de control diferentes (el control trapezoidal y el control FOC[II]), que actuarán de manera conjunta, permitiendo un aprovechamiento máximo de las características del motor, una alta eficiencia y un rendimiento máximo del sistema.

Además de esto, se propondrá el uso de algoritmos complejos como el SOGI-FLL[III] que permitirán calcular con total precisión parámetros útiles del motor en su funcionamiento (como RPM[IV] o posición angular), y permitirán una implementación más robusta de los mecanismos de control. Esta tecnología es común en el mundo de la generación de energía, pero está todavía en desarrollo en el ámbito de las máquinas eléctricas.

En línea con lo anterior, otro de los objetivos que pretende plantear este trabajo, es sentar la base de posibles investigaciones futuras en la misma dirección, y servir como una contribución al desarrollo tecnológico de los motores BLDC.

## 1.2. CONTEXTO Y PREMISAS TECNOLÓGICAS

Los motores BLDC representan una importante evolución en la tecnología de motores eléctricos. A diferencia de los tradicionales motores de CC con escobillas, estos no tienen contactos mecánicos en su interior, reduciendo el desgaste y mejorando en gran medida su eficiencia y rendimiento. En la figura 1.1 se muestra el esquema genérico de un motor CC tradicional.

[I] Motor brushless de DC
[II] Control por orientación de flujo
[III] Second Order Generalized Integrator - Frequency Locked Loop
[IV] Revoluciones por minuto

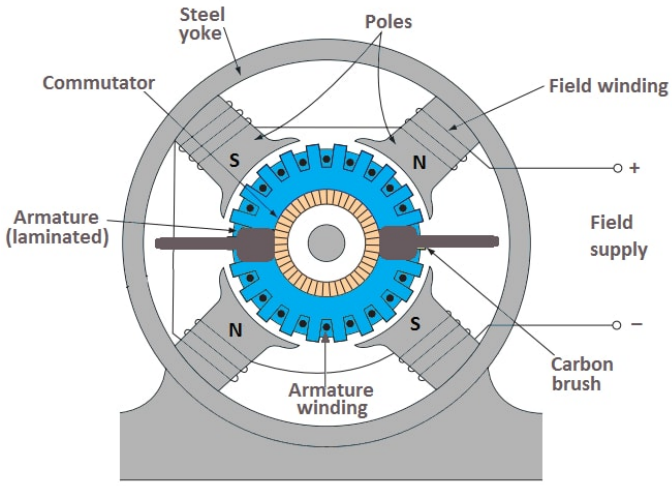


**Figura 1.1: Esquema genérico de motor CC [1]**

La adaptabilidad que presentan los motores BLDC dentro del contexto de las máquinas eléctricas viene propiciada por algunas de las características que se enumeran a continuación:

- **Eficiencia energética**: Los motores BLDC son mucho más eficientes que los motores CC con escobillas, principalmente bajo cargas lineales. No obstante, existen otras alternativas con mejores prestaciones en ciertos escenarios (como los motores PMSM), que se desarrollarán más adelante.
- **Mecanismos de control**: Existe una gran variedad de mecanismos de control disponibles para motores BLDC, aunque el algoritmo de control más frecuente es el Control Trapezoidal, que se explicará más adelante. Además de este, también se utilizará control FOC de manera conjunta con el trapezoidal.
- **Necesidad de mantenimiento**: Debido a que este tipo de motores no presentan escobillas, no es necesario un mantenimiento muy frecuente ni costoso. Esto es una ventaja significativa en aplicaciones de bajo coste y larga vida útil.
- **Ruido y vibraciones**: El ruido y las vibraciones afectan directamente a la experiencia de los usuarios. Si se utiliza un motor BLDC, este efecto podrá mitigarse mediante un diseño adecuado del control.

Otra característica muy descriptiva de los motores BLDC, es a menudo la presencia de sensores de efecto Hall integrados en su estátor. Estos facilitan la implementación de algoritmos de control y la medida de parámetros como la velocidad angular, aunque su presencia dependerá del tipo de motor BLDC escogido, y de la aplicación que se desea implementar. En apartados posteriores se discutirán estas dos configuraciones, y se discutirá qué ventajas e inconvenientes presentan de cara al objeto de este trabajo.

Además de todo lo anterior, los motores BLDC componen un sector de máquinas eléctricas con multitud de aplicaciones y una gran comunidad tanto de usuarios como desarrolladores. Al ser una tecnología tan madura y estable, es más oportuno que cada vez más aplicaciones cuenten con este tipo de motores en lugar de otras tecnologías, y se produzcan nuevos desarrollos tecnológicos.

## 1.3. APLICACIONES Y ENFOQUE

La versatilidad de los motores BLDC los convierte en unos candidatos perfectos para multitud de aplicaciones, pero la elección de un método de control u otro, está sujeta a la exigencia que presente cada sistema en cuanto a tiempos de respuesta, presencia de vibraciones o necesidad de precisión en el control.

Uno de los objetos de este trabajo es proponer una aplicación para la industria de la automoción, donde un sistema de control híbrido (con distintos tipos de modulación que aprovechen mejor las características del motor, en cada rango de velocidad) permitirá aumentar al máximo la eficiencia energética del vehículo eléctrico (aumentando de manera sustancial la autonomía de este), reduciendo vibraciones y ruido de modo que aumente el confort en el habitáculo del vehículo o aumente la precisión si la aplicación es industrial. En la figura 1.2 se muestra un ejemplo simplificado de un vehículo eléctrico, con una unidad de control central y dos etapas de potencia que actúan sobre los motores BLDC por pares, con uno en cada rueda.

A lo largo de este documento, se considerará esta aplicación de manera recurrente para el dimensionamiento de algunos componentes electrónicos en el entorno de simulación.

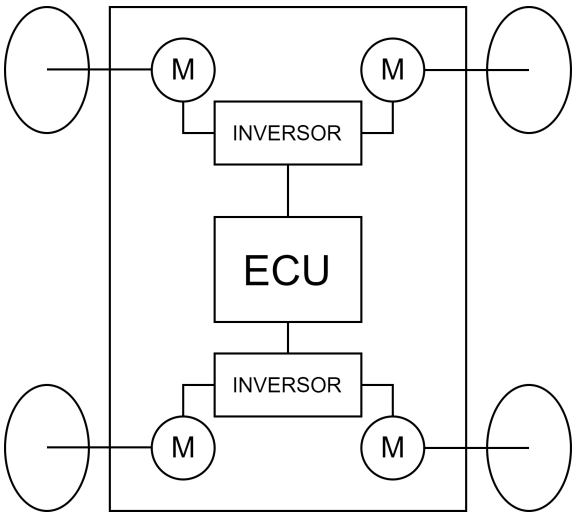


**Figura 1.2: Ejemplo de vehículo con cuatro motores BLDC (original)**

## 1.4. HERRAMIENTAS SOFTWARE UTILIZADAS

Para el desarrollo de este documento, se utilizarán diferentes herramientas informáticas para el procesado matemático, el desarrollo de código C, y la simulación del entorno hardware del motor. Todos estos recursos se encuentran enumerados a continuación:

1. **MATLAB**: Cálculos numéricos complejos en lazos de control, y obtención de resultados a partir de estos.
2. **Mathcad**: Cálculo simbólico para la discretización de sistemas continuos.

3. **Visual Studio Code**: Elaboración de documento LaTeX, y procesamiento de código en C para CCS y PSIM.
4. **Altair PSIM 9.3**: Simulación del inversor de tracción y el motor BLDC con los métodos de control y algoritmos propuestos.
5. **Code Composer Studio v3**: Programación de DSP TMS320F280049C de Texas Instruments.
6. **DrawIO**: Creación de diagramas de bloques, diagramas de flujo y otras figuras explicativas.

El proceso de diseño constará de varias fases, donde se pasará por la primera obtención de modelos matemáticos sobre el sistema, después la simulación de este en la herramienta PSIM, y por último, una implementación del sistema en un DSP, con la obtención de resultados experimentales.

## 1.5. ESTRUCTURA DEL TRABAJO

El documento de este trabajo está compuesto por varios bloques principales: En primer lugar, se incluye una memoria dividida en capítulos, donde se exponen todos los conocimientos técnicos necesarios para el correcto desarrollo del sistema híbrido de control. Este se aborda de manera teórica generando una serie de modelos y ecuaciones que serán de gran utilidad para la evaluación en el entorno de simulación de PSIM. Todo lo anterior da pie a un último apartado, que es la validación del diseño mediante una implementación en un entorno real. Para llevar esto a cabo, será necesario utilizar un modelo a escala del desarrollo realizado para la simulación, que permita demostrar el funcionamiento de algunas características de este. Las características que puedan codificarse en el DSP, dependerán de las especificaciones del hardware disponible para la implementación.

Aparte de esto se incluirá una sección de anexos, donde se añadirán todos los scripts de MATLAB utilizados en el cálculo de parámetros del diseño, y todo el código en C de elaboración propia desarrollado para el control del motor.

# Capítulo 2

# CARACTERÍSTICAS PRINCIPALES DE LOS MOTORES BLDC

## 2.1. JUSTIFICACIÓN DEL USO DE MOTORES BLDC

Los motores BLDC surgen como una alternativa a los moteres DC convencionales, donde se da un gran salto tecnológico al dejar de depender de un sistema con escobillas que generan fricción con el estátor, a un sistema cuyo principio de funcionamiento es la rotación de un campo magnético en el los devanados del estátor.

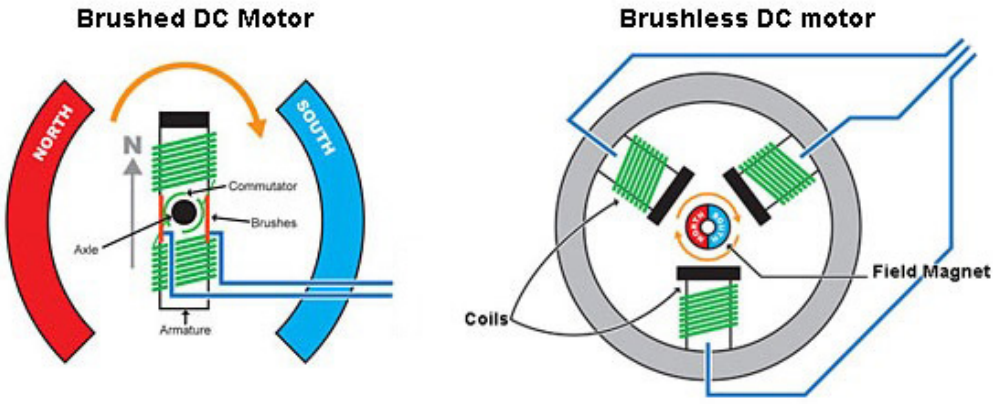


**Figura 2.1: Motor CC frente a motor BLDC [2]**

Los BLDC son una clase de máquinas síncronas de imanes permanentes o PMSM, que se caracterizan por su multitud de aplicaciones en electrónica de consumo. No obstante, cuando nos referimos a PMSM convencionales, solemos hacerlo a otro tipo de máquinas algo diferentes [3].

La diferencia entre una PMSM y un motor BLDC reside en la manera con que están configurados los devanados del motor, y la disposición de los imanes en el rotor [4]. En el primero, los devanados del estátor se encuentran dispuestos de manera uniforme por todo el contorno, mientras que en los BLDC estos son más salientes. Esta diferencia repercute en la forma de onda de la FEM[1], que

[1]Fuerza electromotriz

tenderá a ser senoidal en el caso de las PMSM, y será trapezoidal para los motores BLDC como se observa en la figura 2.2.

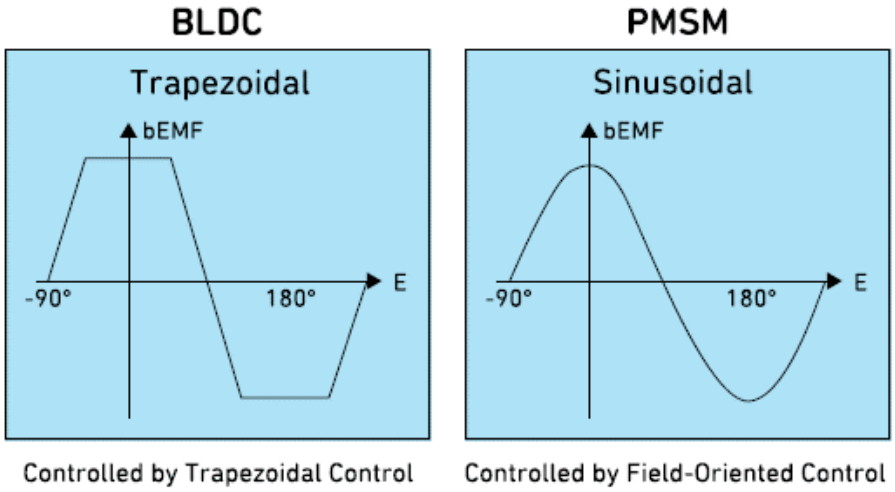


**Figura 2.2: Comparación de FEM en BLDC y PMSM [5]**

Esta diferencia provoca principalmente que los motores PMSM sean utilizados en aplicaciones de gran precisión, ya que contarán con un par más suave, con menor rizado, lo que sugiere que son, en términos generales, más silenciosos y con menos vibraciones que los BLDC. No obstante, esto también hace que las PMSM sean más caras que los BLDC, y que su relación de tamaño y par sea generalmente algo mayor.

Por su accesibilidad en el mercado, gran cantidad de opciones disponibles y bajo coste, este trabajo se fundamentará en el uso de motores BLDC.

### 2.1.1. TIPOLOGÍAS EXISTENTES DE MOTORES BLDC

Dentro de los motores BLDC hay dos tipologías principales, los motores **inrunner**[II] y los **outrunner**[III]. Estos dos tipos de motores presentan diferencias físicas muy notables, que se desarrollarán más adelante.

Los motores BLDC inrunner, también conocidos como motores de rotor interno, tienen el rotor en el centro de la carcasa. Todos los componentes rotativos, excepto el extremo del eje, están dentro del motor. El estator se construye alrededor del rotor y se fija a la carcasa del motor. Además, los imanes permanentes se encuentran en la superficie del rotor o están enterrados en el rotor. Algunas de las ventajas de este tipo de motores son:

- **Eficiencia y potencia**: Los motores inrunner son más eficientes y potentes.
- **Velocidad**: Se utilizan donde se requiere alta velocidad.
- **Tamaño**: Suelen ser más pequeños en diámetro pero axialmente más largos en longitud.

[II] Motor BLDC de rotor interno
[III] Motor BLDC de rotor externo

- **RPM por voltio**: Tienen un alto coeficiente de RPM por voltio.

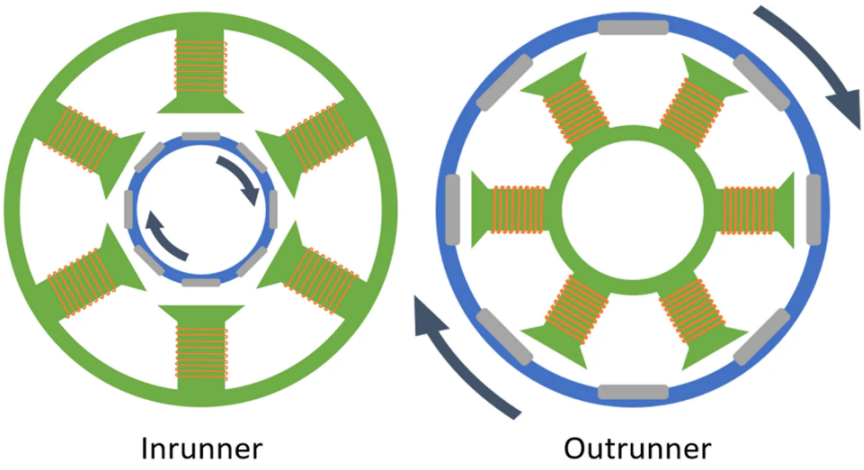


**Figura 2.3: Motor BLDC inrunner frente a outrunner [6]**

Por otro lado, están los motores BLDC outrunner. Estos tienen el estator montado en el exterior del rotor. Los imanes están pegados a la carcasa, que está más alejada del eje y el peso de la carcasa actúa como un volante de inercia. Esto les permite ser idóneos para aplicaciones de alto par y baja velocidad, como el movimiento de la hélice de un vehículo aéreo.

A pesar de las ventajas de los motores inrunner, es importante tener en cuenta que la elección entre un motor inrunner y outrunner depende en gran medida de la aplicación específica. Cada tipo de motor tiene sus propias ventajas y desventajas, y optar por uno u otro dependerá de factores como la necesidad de par, eficiencia, tamaño y velocidad. En la tabla 2.1, se muestran algunas características y aplicaciones específicos de estos dos tipos de motores BLDC.

Para poder elegir un tipo de motor, habrá que estimar qué aplicación principal se encontrará el sistema de control aquí diseñado. Una posibilidad es orientarlo hacia vehículos pequeños para mercancías ligeras, o vehículos personales de poca potencia. Para estas dos aplicaciones se requerirá una precisión mayor en el control de la velocidad, y un aprovechamiento máximo de la potencia disponible. Debido a esto, a su alta disponibilidad en el mercado y también su bajo coste, se ha preferido optar por una tipología de motor inrunner.

| Característica | Inrunner | Outrunner |
|---|---|---|
| Precisión | Precisión relativamente baja | Precisión relativamente alta |
| Eficiencia | Más alta debido a la baja inercia | Más baja debido a la alta inercia |
| TEM | Par generado relativamente bajo | Par generado relativamente alto |
| Estructura | Rotor presente dentro del estator | Rotor presente fuera del estator |
| Tamaño físico | Tamaño total es pequeño y el diámetro es más pequeño | El tamaño total es más extenso y el diámetro es más extenso |
| RPM por voltio | Altas RPM por voltio | Bajas RPM por voltio |
| Transferencia de calor | Alto | Bajo |
| Aplicaciones | Aeromodelismo, coches, camiones, lanchas acuáticas, etc | Aviones de combate, avionetas, helicópteros, drones, etc |
| Mantenimiento | Mantenimiento relativamente alto | Mantenimiento relativamente bajo |
| Pérdidas y riesgos | El alto riesgo de contaminación da como resultado una baja precisión | Bajos riesgos de contaminación, lo que da como resultado una alta precisión |
| Ruido | Muy ruidoso | Muy silencioso |
| Aplicaciones | Con requisitos de alta velocidad | Con requisitos de par elevado |
| Número de polos magnéticos | Generalmente menor | Generalmente mayor |

**Tabla 2.1: Comparación de tipologías motores BLDC**

## 2.2. METODOLOGÍAS DE CONTROL PARA MOTORES BLDC

Dentro de los posibles algoritmos de control aplicados a motores BLDC, se pueden encontrar tres tipos principales, el control trapezoidal (o six-step), el control senoidal, y el FOC. En este apartado, se describirá cada uno de ellos, destacando cuáles son sus diferencias, y la justificación de los elegidos para llevar a cabo este diseño.

### 2.2.1. CONTROL TRAPEZOIDAL

El control trapezoidal es el método de control más utilizado en motores BLDC, por la sencillez de su implementación, y por ser capaz de obtener un buen par y velocidad, con unas pérdidas de conmutación bajas. El origen de su nombre proviene de la forma de onda que se crea en las tensiones de línea del motor, que se muestran en la figura 2.4.

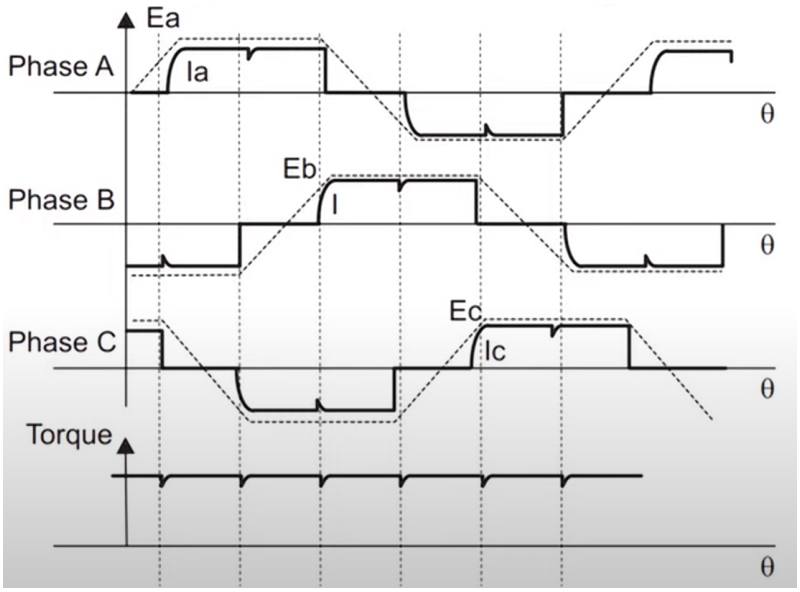


**Figura 2.4: Secuenia de conmutación en motor BLDC [7]**

Se observa que las formas de onda de las tensiones son una sucesión de seis estados que se repiten de manera periódica. Esto es algo de mucha utilidad para poder crear posteriormente una secuencia de conmutación que permita generar las señales de disparo del inversor de tracción, ya que el objetivo de este método de control será crear pequeños desequilibrios en el campo magnético de los imanes permanentes, de manera secuencial. Esto se traduce en que la implementación de este método de control estará caracterizada por una máquina de estados.

Aunque este método de control presenta muchas ventajas, es importante señalar algunos inconvenientes como pueden ser una menor eficiencia comparado con otros métodos de control, y un menor par electromecánico máximo que podrá desaprovechar las especificaciones del motor. Además, el rizado en el par mecánico del motor, puede provocar vibraciones que se traduzcan en ruidos audibles y poco agradables para los usuarios del sistema. Debido a que la forma de onda de las tensiones de fase $V_{ab}$, $V_{bc}$ y $V_{ca}$ serán un tren de pulsos, se introducirán componentes de alta frecuencia en el par de salida.

Algunas de las ventajas que este modo de modulación presenta respecto a otras alternativas, es su fácil implementación, la extracción de un alto par y velocidad con pérdidas de conmutación pequeñas, y una posibilidad de implementación del sistema en lazo abierto sin medidas de tensión, velocidad o posición del rotor.

### 2.2.2. CONTROL SENOIDAL

Otro tipo de estrategia de control para motores BLDC es el control senoidal. Este también es un método de control que al igual que el trapezoidal, puede describirse a través de las formas de onda de las tensiones y corrientes de línea. El objetivo será producir unas corrientes para cada fase del motor, con una forma de onda senoidal, como se muestra en la figura 2.5.

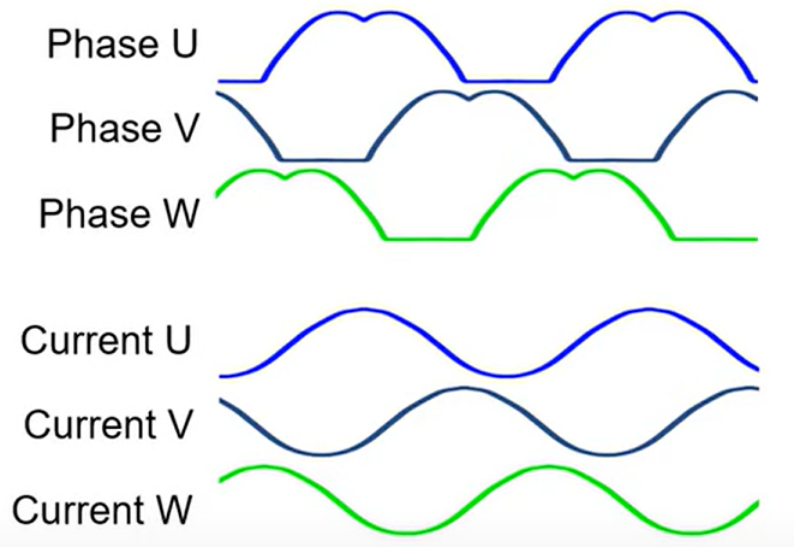


**Figura 2.5: Tensiones de línea en control trapezoidal [8]**

Entre las ventajas que ofrece este tipo de control están una eficiencia mucho mayor que el control trapezoidal, y un par electromecánico que presentará un bajo rizado para cargas lineales. Esto se traducirá en una mayor eficiencia del motor y un menor ruido audible. No obstante, este tipo de control requiere un mayor esfuerzo en su implementación, y sigue sin ofrecer una eficiencia máxima ni un par electromecánico con buena adaptación a cargas dinámicas. Esto se debe, de manera específica con motores BLDC, a que una modulación senoidal generará inevitablemente una discrepancia entre las tensiones de línea y la forma de onda de la $F_{em}$ inducida, lo que se traducirá, de nuevo, en armónicos en la corriente de entrada del inversor, y rizado en el par de carga.

### 2.2.3. CONTROL FOC

El control FOC (Control por Orientación de Campo), es una técnica más avanzada de control que consistirá en el control de la corriente que se aplica a los devanados del motor con el fin de que esta siempre esté en fase con el campo magnético generado por el rotor. Esto permite aplicar en todo momento el par electromecánico máximo, para cualquier posición del rotor. Por este motivo, el control FOC será ideal para aplicaciones con control de velocidad donde pueda haber cargas que no sean constantes. Para visualizar el funcionamiento del control FOC, se imagina el campo magnético de una máquina con un solo imán permanente. En un sistema de referencia móvil DQ, se podrá determinar que el par electromecánico aplicado sobre el rotor será máximo cuando el campo magnético creado por los devanados del estátor esté alineado con el campo del eje de cuadratura ’q’ del imán.

A diferencia del control trapezoidal, el control FOC requiere para su sincronización la medida de parámetros como las corrientes de línea (para calcular el error entre la corriente ideal y la medida), y la posición angular del rotor para estimar la intensidad del flujo magnético en esa posición.

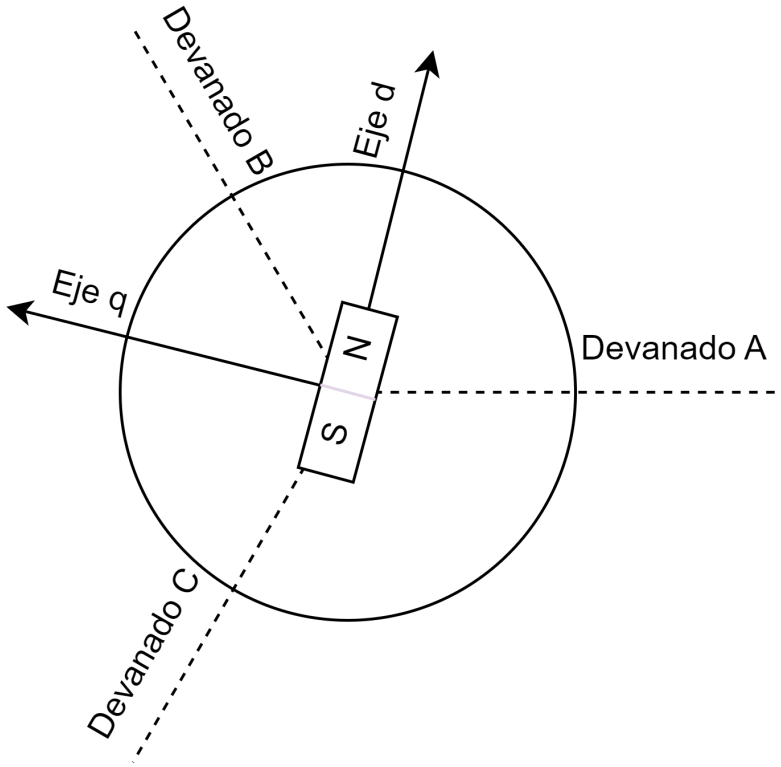


**Figura 2.6: Esquema de orientación del flujo magnético en motor BLDC ideal con un imán permanente (original)**

### 2.2.4. JUSTIFICACIÓN DE LOS ALGORITMOS DE CONTROL ELEGIDOS

La arquitectura del control que se plantea en este documento estará formada por el control trapezoidal para tareas de arranque y baja velocidad, y el control FOC para aplicaciones de alta velocidad y par elevado. El sistema desarrollado elegirá de manera autónoma qué método de control utilizar en cada momento.

El control trapezoidal funcionará como una máquina de seis estados de conmutación, donde a partir de las lecturas de los sensores de efecto Hall se podrá estimar qué secuencia de conmutación deberá aplicarse para provocar un giro en uno u otro sentido. Dado que este método crea un par de salida con un gran rizado y que no es máximo en ningún caso, se cambiará el control de trapezoidal a FOC tan pronto como sea posible. Para que esto suceda, deberá intervenir el algoritmo SOGI-FLL, que permitirá calcular una medida muy fiable de la fase del rotor y la velocidad angular del sistema. Este algoritmo es muy común en sistemas de generación energética, pero su implementación en un sistema de control de tracción es original y novedosa.

Con la jerarquía de control propuesta, se plantea un sistema óptimo en cuanto a control, y que previsiblemente será robusto ante escalones de carga o variaciones en la velocidad de referencia.

# Capítulo 3

# DISEÑO DEL SISTEMA DE CONTROL HÍBRIDO

Para afrontar el diseño del sistema de control híbrido, primero tendrán que plantearse unos requesitos generales para el sistema que permitan dimensionar algunos componentes, entre ellos el motor BLDC o la batería que se usará en el sistema. Para el posterior diseño de los lazos de control, será también necesario sentar las bases sobre el desarrollo de diferentes modelos de inversores trifásicos, que permitan obtener unas expresiones válidas para el diseño de ganancias de lazo y funciones de transferencia. Esto se afrontará de manera separada para el control trapezoidal y el control FOC, ya que en el diseño estos se comportarán de manera completamente independiente, aunque el diseño de sus lazos de control estará estrechamente relacionado.

Por otro lado, el control FOC hará uso del algoritmo SOGI-FLL. La implementación de este es original y novedosa, y no hay demasiada documentación relacionada con su aplicación a inversores de tracción para control de máquinas de imanes permanentes. Con este documento se pretende avanzar con esta tecnología, y arrojar luz sobre su aplicación al mundo de las máquinas eléctricas.

Cabe destacar que en todas las etapas del diseño se ha elegido una frecuencia de modulación de 10kHz. Esto se ha decidido tras observar que este valor es un valor típico de frecuencia de conmutación para motores BLDC, y es frecuentemente recomendado por fabricantes de controladores integrados como Infineon o Texas Instruments. Además, numerosos equipos de control de motores aplicados a vehículos radiocontrol utilizan esta misma frecuencia de conmutación.

## 3.1. ESTUDIO DE MODELOS PARA LA CREACIÓN DE LAZOS DE CONTROL

Para crear el lazo de control con la estrategia de control FOC, primero se tendrán que desarrollar una serie de modelos que surgirán de las consideraciones mecánicas y eléctricas del motor, y el inversor trifásico. En este apartado se sentarán las bases del posterior desarrollo de todos los algoritmos y lazos de control que permitirán implementar el sistema del motor en el software PSIM.

### 3.1.1. MODELO DE INVERSOR EN COORDENADAS ABC - DQO

El modelo en pequeña señal de un inversor trifásico genérico, con un motor BLDC conectado en su salida trifásica, es el que se observa en la figura 3.1. Este modelo está compuesto por una entrada de tensión continua, un filtro DC-LINK, y unas fuentes de corriente y tensión que permiten modelar cómo la conmutación de los semiconductores afecta a la etapa de entrada (fuentes dependientes de corriente), y a la salida (fuentes dependientes de tensión) [9]. En la parte derecha se pueden ver otros elementos, que son las inductancias y resistencias de cada uno de los devanados del motor BLDC en conexión de estrella, además de tres fuentes de tensión $\epsilon_a, \epsilon_b$ y $\epsilon_c$ que representan el aporte de tensión de la fuerza contraelectromotriz inducida sobre el estátor.

No obstante, trabajar en coordenadas ABC no es práctico de cara a la realización de lazos de control, por la complejidad de los cálculos, y por el poco control que se puede conseguir sobre magnitudes como el par del motor y el flujo magnético. Es por ello que generalmente se opta por recurrir a la transformada de Park, y transformar este sistema a un eje de referencia móvil D-Q, que será dependiente de la fase del rotor.

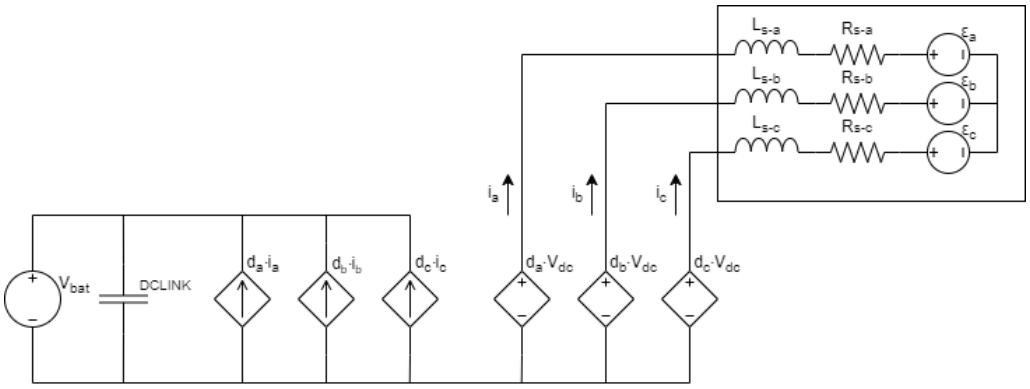


**Figura 3.1: Modelo de inversor en coordenadas ABC (original)**

Una vez se aplica la transformada directa de Park, el circuito resultante es el de la figura 3.2, que presenta dos lazos D y Q que están acoplados entre sí.

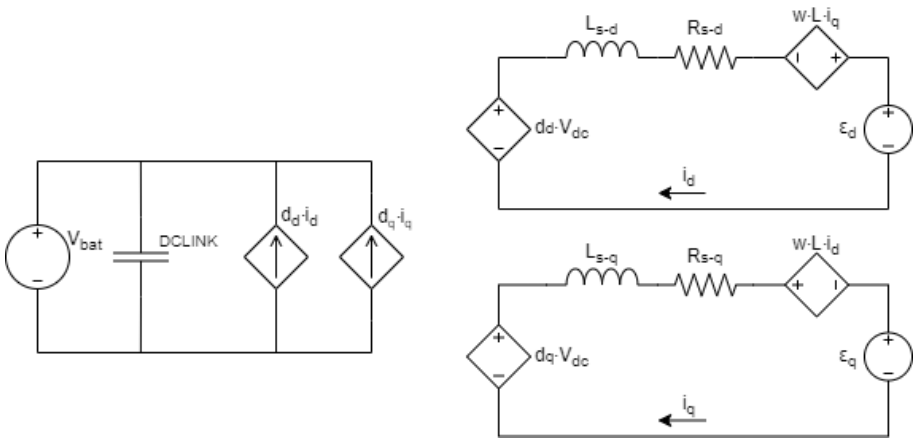


**Figura 3.2: Modelo de inversor en coordenadas DQ (original)**

Se observa que en el esquema anterior se han expresado también los términos de resistencia e inductancia del estátor en coordenadas D-Q. Esto se debe a la aproximación hecha en la expresión 3.1, debida a que en los motores BLDC la distribución de los devanados en el estátor e imanes permanentes en el rotor es simétrica. Además, estas aproximaciones simplificarán enormemente los cálculos, y permitirán considerar equivalentes los lazos de control diseñados en los ejes D y Q.

$$\begin{aligned} R_s &= R_d \simeq R_q \\ L_s &= L_d \simeq L_q \end{aligned} \tag{3.1}$$

Con el fin de diseñar los lazos de control, será necesario obtener las ecuaciones que caracterizan los circuitos D y Q. De acuerdo con lo planteado en la figura 3.2, las ecuaciones de las tensiones $V_d$ y $V_q$ son las que se muestran en la expresión 3.2. Si bien el criterio de elección de un sentido para las corrientes $i_d$ e $i_q$ es arbitrario, este determinará posteriormente en qué sentido se está controlando el giro del motor.

$$\begin{aligned} V_d &= d_d \cdot V_{dc} = R_s \cdot i_d + L_d \cdot i_d \cdot s + L_q * \omega * i_q \\ V_q &= d_q \cdot V_{dc} = R_s \cdot i_q + L_q \cdot i_q \cdot s - L_d * \omega * i_d - \omega \cdot \Phi_m \end{aligned} \tag{3.2}$$

Una expresión que será de vital importancia será la 3.3, donde se relaciona el cálculo del par electromecánico $T_{em}$ con la corriente del eje Q del inversor. Esta expresión permite relacionar los parámetros eléctricos generados en el estátor con la generación de un par de fuerzas en el rotor del motor, que se traducirá en una velocidad mecánica de giro. El problema que surge es cómo se podrá estimar el cálculo del flujo magnético del estátor, $\Phi_m$, puesto que este valor no es proporcionado por el fabricante del motor BLDC al estar este tipo de motores principalmente orientados hacia la implementación de un control trapezoidal.

$$T_{em} = \frac{3}{2} \cdot PP \cdot \Phi_m \cdot i_q \tag{3.3}$$

Para poder hacer una aproximación, se ha realizado un proceso de cálculo dimensional con los parámetros presentes en el motor, y se ha observado que la constante de velocidad $K_v$ relaciona un concepto parecido al flujo magnético, medido en Wb. Esto es debido a que un Wb es una unidad que cuantifica el flujo magnético que es capaz de inducir una FEM de 1V a través de una sola espira, cuando el flujo disminuye de manera constante a 0 en 1s. Haciendo análisis dimensional, se puede determinar que $1Wb$ es igual a $1V \cdot s$, y la constante de velocidad $K_v$ representa las RPM que alcanzará el motor BLDC ante una tensión fase-fase de 1V. Si esta se invierte y se transforman las RPM a rad/s, se obtiene que:

$$[\frac{1}{K_v}] = \frac{V}{RPM} = \frac{V \cdot s}{rad} = Wb \tag{3.4}$$

De este modo, se puede determinar que la constante de flujo magnético $\Phi_m$ será la que dicta la expresión 3.5.

$$K_m = \frac{1}{K_v \cdot PP}(\frac{Wb}{revolucin})$$
$$\Phi_m \approx K_m \cdot \frac{60}{2 \cdot \pi}(Wb) \tag{3.5}$$

## 3.2. DISEÑO DEL BUS DE CONTINUA PARA EL INVERSOR

En el contexto del desarrollo de una cadena de tracción que pueda ser aplicada a vehículos con cierta intervención humana, cualquier rizado o perturbación en la forma de onda de la tensión de entrada podría producir un rizado en el par de carga que podría traducirse en ruido o vibraciones. Además, debido a la conmutación de los semiconductores, sería posible que se produjesen otros efectos indeseados que afectarían a la durabilidad del sistema a largo plazo. El filtro DC-LINK se añade como una manera de mitigar todos estos inconvenientes [10].

### 3.2.1. PRINCIPIO DE FUNCIONAMIENTO DEL DC-LINK

Un DC-LINK es un elemento indispensable en la electrónica de conversión de energía, ya sea para generación energética, conversión para elementos de tracción o cualquier otro uso. Este es un filtro que se encarga de amortiguar el flujo energético entre dos etapas de energía.

En la figura 3.3 se pueden observar dos convertidores teóricos que han sido conectados a través de sus entradas/salidas de DC. En este caso, el filtro DC-LINK se encarga de almacenar energía cuando la demanda de la segunda etapa es baja, y liberarla cuando esta es más alta. Esto, traducido a valores cuantificables, provocará que el rizado en la tensión del bus de continua sea mucho menor.

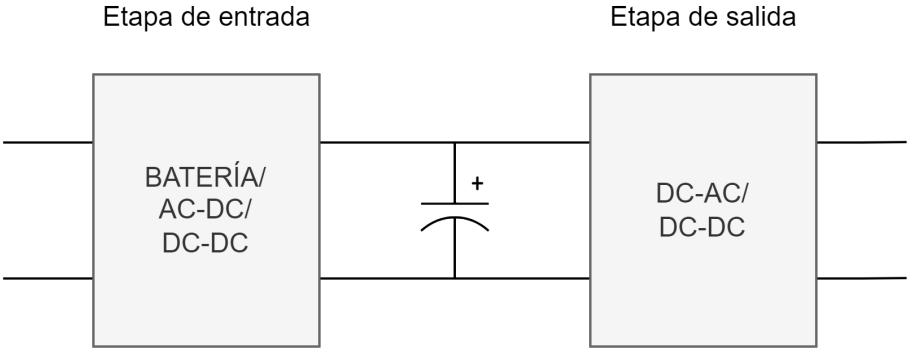


**Figura 3.3: Modelo genérico de filtro DCLINK (original)**

A continuación, se enumeran algunas de las mejoras que la inclusión de un filtro DC-LINK puede aportar al diseño:

- Atenúa los armónicos de conmutación. Estos se producen a múltiplos de $f_{sw}$ y provocan rizado en la corriente del bus de continua, $i_{dc}$. Esto se traduce de manera directa en un rizado del par electromagnético, $Tem$.

- Reduce el ruido electromagnético que proviene de los devanados del motor, o de EMI.
- Al reducir el rizado de tensión, se reducen las pérdidas generales de potencia en el convertidor, y aumenta su eficiencia.

### 3.2.2. DISEÑO TEÓRICO DEL DC-LINK

Para realizar el diseño teórico del filtro DC-LINK, se parte del hecho de que los armónicos de conmutación se producirán, de manera periódica, en múltiplos enteros de la frecuencia de conmutación del sistema. Estos serán los principales causantes de las perturbaciones en la tensión del bus de continua.

Para conseguir una atenuación en todas las componentes frecuenciales iguales o mayores que la frecuencia de conmutación, se plantea el diseño del filtro DC-LINK como un filtro paso bajo, donde la frecuencia de corte vendrá dada por la expresión 3.6. De este modo, se espera tener una atenuación grande para las frecuencias procedentes de la conmutación.

$$\begin{aligned} f_c &= \frac{fsw}{10} \\ \omega_c &= 2 \cdot \pi \cdot f_c \end{aligned} \tag{3.6}$$

El esquema del inversor en coordenadas DQ de la figura 3.4, sirve para modelar, de manera simplificada, la entrada del inversor considerando la resistencia serie equivalente (ESR) de la batería, y el conjunto completo DC-LINK con la capacidad $C_{dclink}$ y la resistencia $ESR_{dclink}$. El análisis de este circuito en contexto con el resto del inversor no es trivial, pero se puede simplificar asumiendo que toda la potencia de los armónicos de conmutación por encima de la frecuencia de conmutación de 10kHz, serán atenuados por el filtro DC-LINK. De este modo, el modelado de una función de transferencia para la tensión del bus de continua respecto a la tensión de la batería se reduce a la creación de un divisor de tensión, como el de la figura 3.5. Esto permite obtener la función de transferencia 3.7.

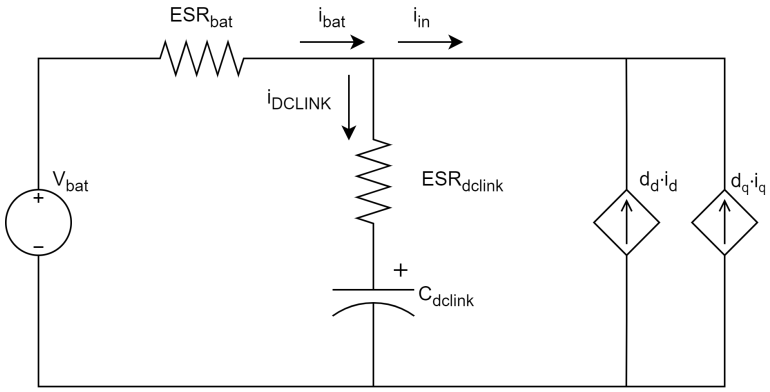


**Figura 3.4: Modelo simplificado de inversor con DC-LINK, en coordenadas D (original)**

$$H_v = \frac{V_i}{V_{bat}} = \frac{Z_{DCLINK}}{Z_{DCLINK} + ESR_{bat}} \tag{3.7}$$

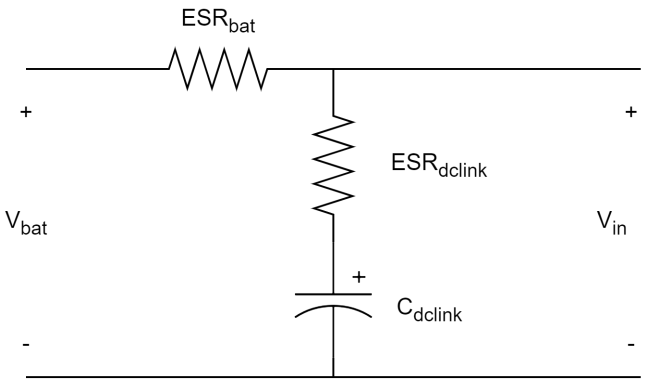


**Figura 3.5: Modelo de filtro de entrada al inversor (original)**

Planteando una expresión para la impedancia del condensador DC-LINK y conociendo la impedancia en serie de la batería, se puede desarrollar la función de transferencia anterior para obtener la expresión 3.9, que se muestra por simplicidad en el dominio de Laplace.

$$Z_{DCLINK}(s) = ESR_{DCLINK} + \frac{1}{C_{DCLINK} \cdot s} \tag{3.8}$$

$$H_v(s) = \frac{V_i}{V_{bat}} = \frac{s \cdot C \cdot ESR_{DCLINK}}{(ESR_{bat} + ESR_{DCLINK}) \cdot s \cdot C + 1} \tag{3.9}$$

Como se puede observar, el condensador DC-LINK tiene dos parámetros, la resistencia serie ESR y la capacidad, que intervendrán en la frecuencia de corte del filtro paso bajo.

### 3.2.3. DISEÑO DEL DC-LINK EN MATLAB

La obtención de unos parámetros de capacidad y resistencia ESR teóricos en MATLAB, pasará por dos etapas. En la primera etapa, se calculará la capacidad teórica del condensador, y en la segunda se utilizará este parámetro para dimensionar un valor esperado de resistencia ESR.

En primer lugar, con el fin de reducir el número de variables independientes de la función de transferencia 3.9, se realiza la aproximación de la expresión 3.10. Esto es debido a que las resistencias en serie para los condensadores electrolíticos de aluminio, suelen ser de decenas de miliohmios. De esta manera, ya se puede calcular un módulo para la función de transferencia H, que dependa únicamente del valor de capacidad del condensador DC-LINK.

$$ESR_{dclink} = ESR_{bat} \tag{3.10}$$

El módulo de la función de transferencia H vendrá dado por la expresión 3.11.

$$\mid H_v \mid = \frac{\mid V_i \mid}{\mid V_{bat} \mid} = \frac{\mid s \cdot C \cdot ESR_{DCLINK} \mid}{\mid (ESR_{bat} + ESR_{DCLINK}) \cdot s \cdot C + 1 \mid} \tag{3.11}$$

Al mismo tiempo, se sabe que en la frecuencia de cruce, el filtro paso bajo DC-LINK tendrá una caída de magnitud de 3dB, o lo que es lo mismo, la magnitud de $V_i$ respecto a la tensión de la batería $V_{bat}$ será de un 70.7 %.

$$\mid H_v \mid = \frac{1}{\sqrt{2}} = 0{,}707 \tag{3.12}$$

Desarrollando la expresión 3.11, se puede despejar una expresión para la reactancia del condensador DC-LINK.

$$\mid H_v \mid = \frac{\sqrt{ESR_{DCLINK}^2 + {X_{DCLINK}}^2}}{\sqrt{(ESR_{bat} + ESR_{DCLINK})^2 + {X_{DCLINK}}^2}} = \frac{1}{\sqrt{2}} \tag{3.13}$$

$$\mid H_v \mid^2 = \frac{ESR_{DCLINK}^2 + {X_{DCLINK}}^2}{(ESR_{bat} + ESR_{DCLINK})^2 + {X_{DCLINK}}^2} = \frac{1}{2} \tag{3.14}$$

$$X_{DCLINK} = \frac{1}{2 \cdot \pi \cdot \omega_c \cdot C} = \sqrt{{ESR_{bat}}^2 + 2 \cdot ESR_{bat} \cdot ESR_{DCLINK} - {ESR_{DCLINK}}^2} \tag{3.15}$$

Resolviendo la expresión 3.15, se obtiene un valor teórico de condensador DC-LINK de 3.8mF. Esta es la capacidad mínima que deberá tener el filtro DC-LINK.

Si ahora se realiza la curva de Bode del filtro diseñado, según la figura 3.3, se podrá comprobar que la frecuencia de cruce está en el lugar deseado, sobre 1kHz.

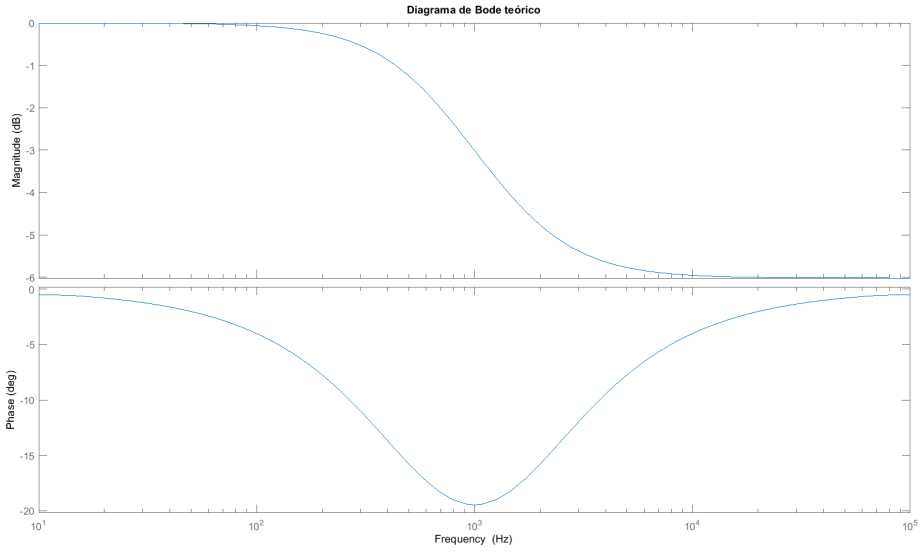


**Figura 3.6: Diagrama de bode del filtro DC-LINK teórico**

## 3.3. DESARROLLO DEL ARRANQUE CON CONTROL TRAPEZOIDAL

Una de las partes fundamentales del control híbrido será el desarrollo de un control de tipo trapezoidal para el arranque del motor. Esto se va a lograr mediante la implementación de un control en lazo cerrado, con un lazo único de regulación que generará un ciclo de trabajo para la creación de una secuencia de conmutación en los transistores de las tres ramas. Este es el método de control más básico para motores BLDC, que permitirá arrancar el motor con una gran facilidad.

### 3.3.1. USO DEL MOTOR COMO CARGA PARA LA ELABORACIÓN DE LA SECUENCIA DE CONMUTACIÓN

Para poder diseñar el algoritmo de control trapezoidal del motor BLDC, será necesario idear una secuencia de conmutación apropiada para los sensores Hall, en función de la tensión en las fases del motor, con el fin de poder actuar sobre estas para rotar el campo magnético que produce el estátor. Esto se puede conseguir observando qué estado presenta cada sensor Hall en función de la tensión fase-fase que produce cada estado de conmutación en los transistores del inversor, con uso de una fuente de par mecánico externa y haciendo que el motor BLDC se comporte como una carga RLE. Esto se ha conseguido mediante el diseño de la figura 3.7, creando la rampa de velocidad de la figura 3.8.

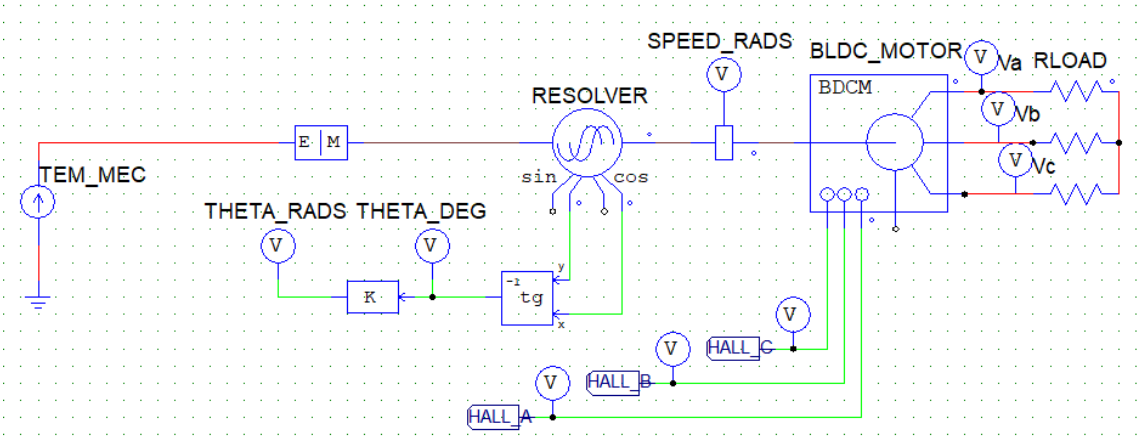


**Figura 3.7: Esquema en PSIM para la evaluación del motor BLDC como carga RLE**

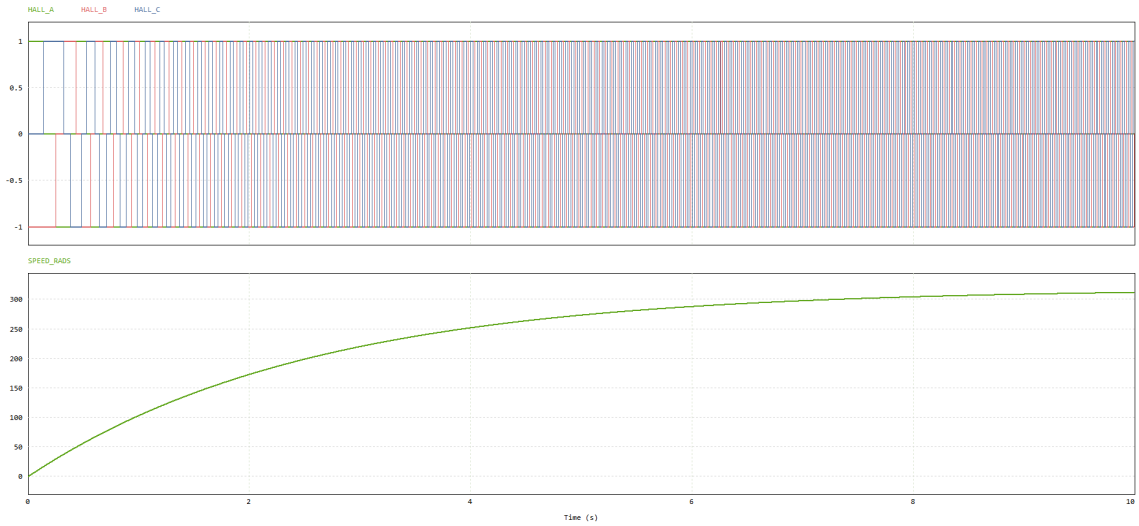


**Figura 3.8: Rampa de velocidad creada a partir de carga externa, frente a conmutaciones en sensores Hall**

Tras esto, se pueden observar todas las tensiones de línea frente a los sensores de efecto Hall, con el fin de traducir cada estado de conmutación de estos en un cierto estado de conmutación de los transistores de las ramas 1, 2 y 3. Si se observa por ejemplo la gráfica 3.9, se puede ver que cuando el sensor Hall A se encuentra a nivel alto, la tensión de fase Va se encuentra a nivel bajo. Esto sugiere que habrá dos estados principales, y que podrá predecirse cómo será la tensión de fase en función de los estados de conmutación de los sensores.

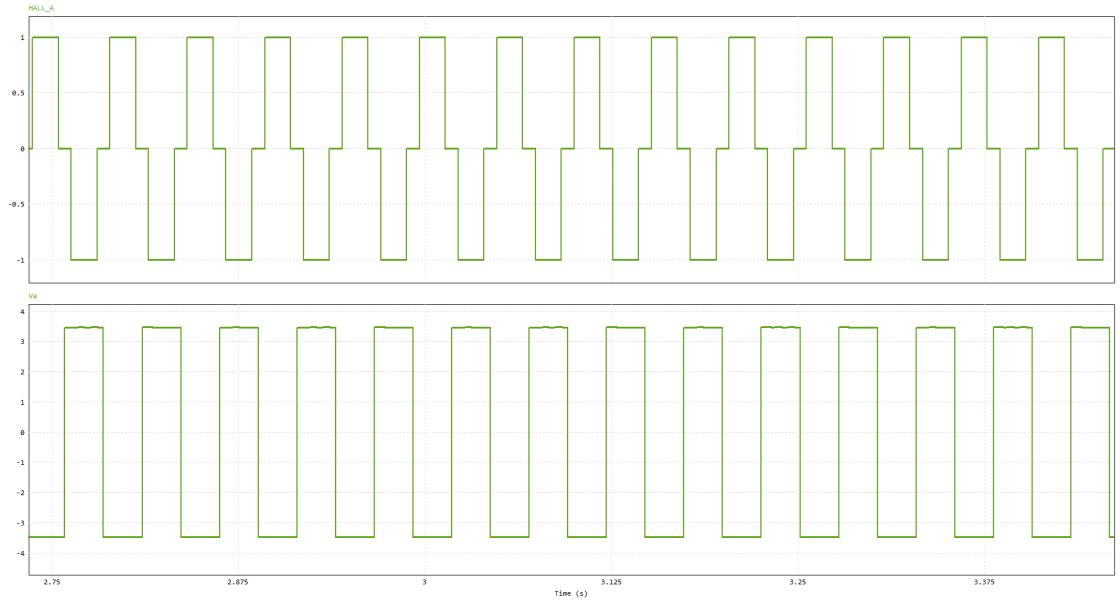


**Figura 3.9: Conmutación de sensores Hall frente a tensión de fase Va**

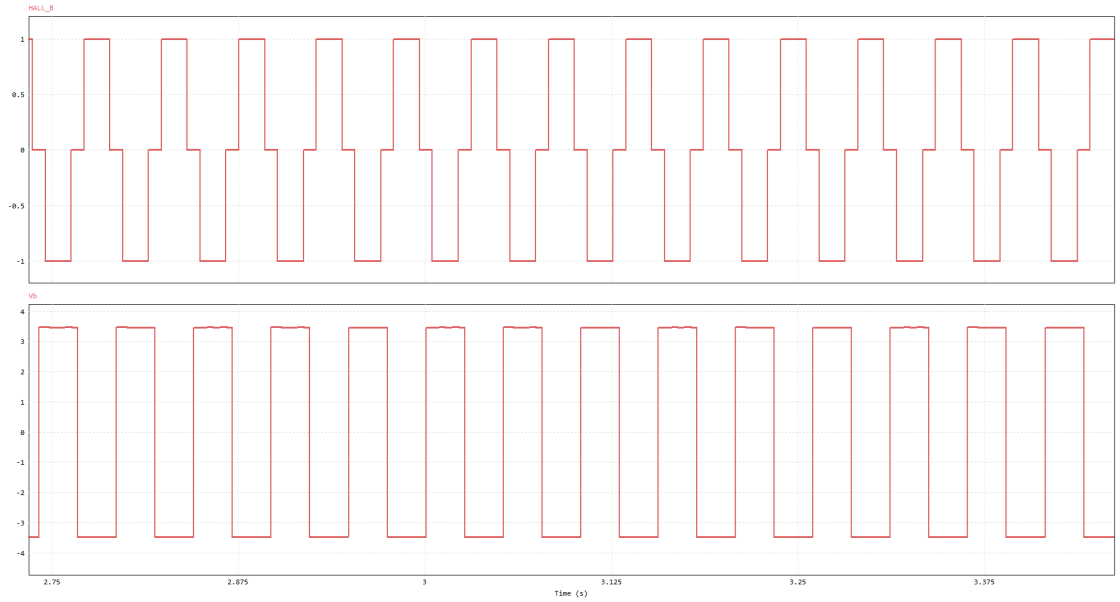


**Figura 3.10: Conmutación de sensores Hall frente a tensión de fase Vb**

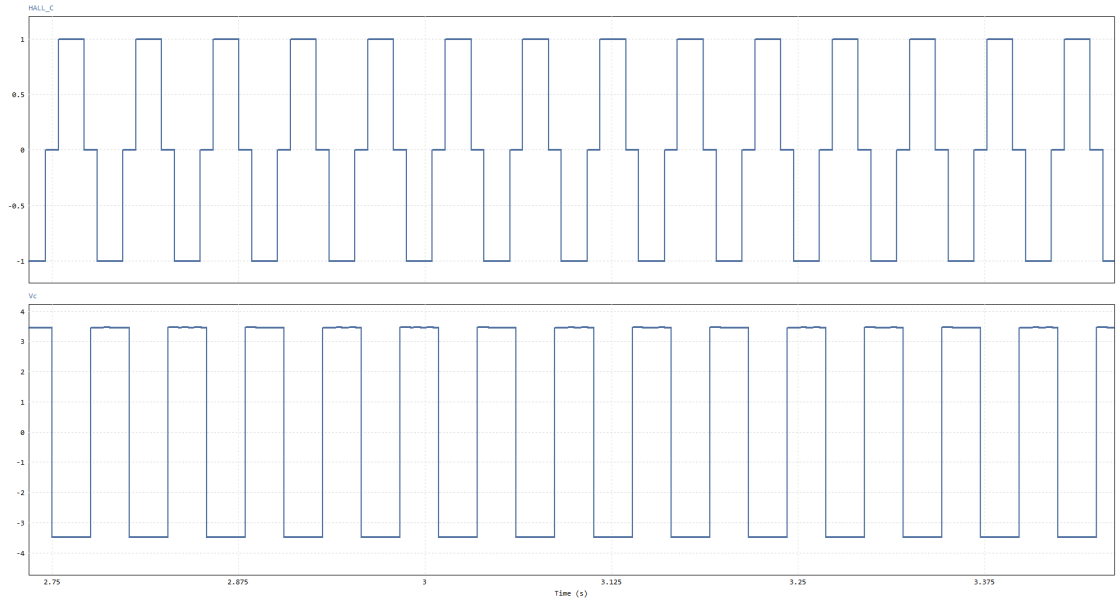


**Figura 3.11: Conmutación de sensores Hall frente a tensión de fase Vc**

Después de observar las imágenes anteriores, se podrán elaborar las diferentes tablas de verdad que relacionarán el estado de los transistores de cada rama, con su respectivo sensor de efecto Hall. De esta manera, se obtienen tres tablas con el estado que deberá tener cada sensor, a fin de que su tensión de fase correspondiente conmute. Si el motor se encuentra en un determinado estado, y se modifica el estado de conmutación de los transistores, podrá esperarse que se produzca un giro en el rotor, y que los sensores Hall vuelvan a conmutar. Las tablas de verdad para las tres ramas se muestran a continuación.

- **Tabla de conmutación para rama A**

| $S_1 1$ | $S_1 2$ | $Hall_A$ |
|---|---|---|
| 1 | 0 | $-1$ |
| 0 | 1 | 1 |

**Tabla 3.1: Secuencia de conmutación trapezoidal de rama A**

- **Tabla de conmutación para rama B**

| $S_2 1$ | $S_2 2$ | $Hall_B$ |
|---|---|---|
| 1 | 0 | $-1$ |
| 0 | 1 | 1 |

**Tabla 3.2: Secuencia de conmutación trapezoidal de rama B**

- **Tabla de conmutación para rama C**

| $S31$ | $S32$ | $Hall_C$ |
|---|---|---|
| 1 | 0 | $-1$ |
| 0 | 1 | 1 |

**Tabla 3.3: Secuencia de conmutación trapezoidal de rama C**

Por último, solo queda componer todos los resultados anteriores en una nueva tabla que recopile todos posibles estados de conmutación, evitando estados prohibidos y sin considerar estados nulos. Todo esto se recoge en la tabla 3.4.

| $State$ | $S_1 1$ | $S_1 2$ | $S_2 1$ | $S_2 2$ | $S_3 1$ | $S_3 2$ | $Hall_A$ | $Hall_B$ | $Hall_C$ |
|---|---|---|---|---|---|---|---|---|---|
| $S1$ | 1 | 0 | 0 | 0 | 0 | 1 | $-1$ | 0 | 1 |
| $S2$ | 1 | 0 | 0 | 1 | 0 | 0 | $-1$ | 1 | 0 |
| $S3$ | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | $-1$ |
| $S4$ | 0 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | $-1$ |
| $S5$ | 0 | 1 | 1 | 0 | 0 | 0 | 1 | $-1$ | 0 |
| $S6$ | 0 | 0 | 1 | 0 | 0 | 1 | 0 | $-1$ | 1 |

**Tabla 3.4: Secuencia de conmutación trapezoidal de ramas A, B y C**

Una vez definidos todos los estados de conmutación S1-S6, será sencillo estimar a qué estado se deberá saltar tras conocer los valores actuales de los sensores de efecto Hall. La frecuencia a la que se producirán estas conmutaciones vendrá marcada por un reloj externo, que se ha creado como una señal triangular unipolar unitaria de la frecuencia de conmutación del sistema, 10kHz. Este reloj se comparará con un ciclo de trabajo externo, y de esta manera se creará una señal PWM (figura 3.12) que generará conmutaciones en las tensiones de fase del motor solo cuando esta esté a nivel alto.

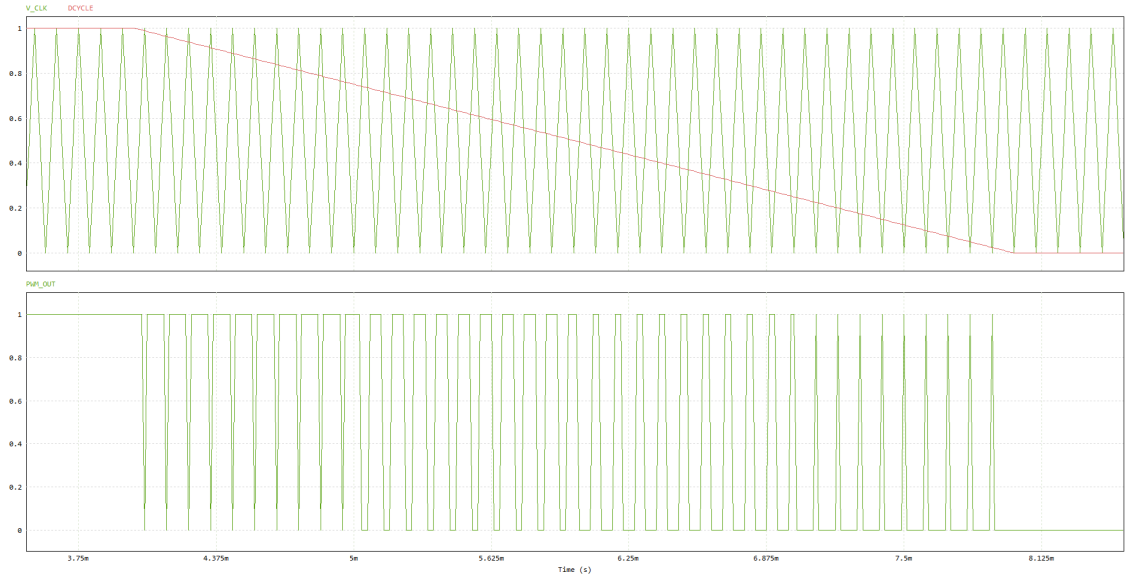


**Figura 3.12: Variación de ciclo de trabajo en PWM para conmutación trapezoidal**

Para la posterior implementación de este algoritmo de modulación trapezoidal en PSIM, se ha diseñado el diagrama de flujo de la figura 3.13. Se observa que cuando la señal PWM se encuentra a nivel alto, se producen conmutaciones en uno u otro sentido, pero cuando se encuentra a nivel bajo, todos los transistores se encuentran en circuito abierto.

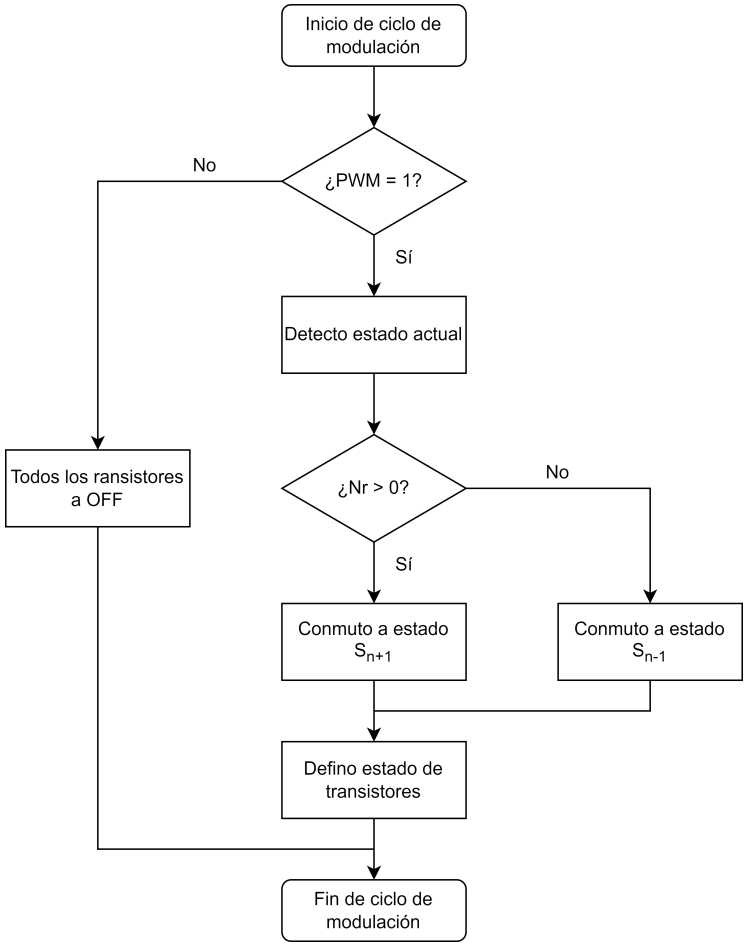


**Figura 3.13: Diagrama de flujo para modulador trapezoidal (original)**

## 3.3.2. DISEÑO DE LAZO CERRADO PARA CONTROL SIX-STEP

Con el fin de mejorar las prestaciones del control trapezoidal, se ha decidido añadir un regulador PI y se ha creado un lazo cerrado de control. Este permitirá controlar la velocidad con mayor precisión que en un lazo abierto.

Para lograr esto, se ha planteado el lazo de control de la figura 3.14. En él se relaciona directamente la corriente generada en el estátor $\frac{d \cdot V_{dc}}{Z_s}$ con el par electromecánico generado en el rotor (ecuación 3.3).

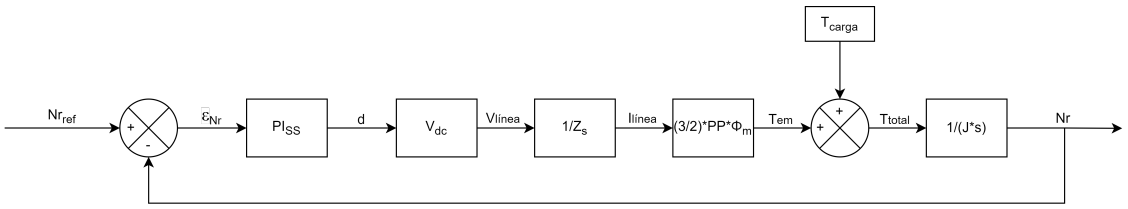


**Figura 3.14: Diagrama de bloques con modulación trapezoidal en lazo cerrado (original)**

Para tener un control en lazo cerrado lo suficientemente lento para poder leer los sensores Hall y actuar sobre la máquina de estados del control trapezoidal, se ha elegido una frecuencia de cruce del regulador PI de $\frac{f_{sw}}{60}$.

$$\begin{aligned} f_{ss} &= \frac{f_{sw}}{60} \\ \omega_{ss} &= 2 \cdot \pi \cdot f_{ss} \end{aligned} \tag{3.16}$$

Se observa que el diseño del regulador PI vendrá marcado por la presencia de polos complejos procedentes tanto del momento de inercia $\frac{1}{J \cdot s}$ como de la impedancia del estátor de la expresión 3.17. Esta situación no es deseada, ya que podría complicar mucho los cálculos y el análisis de estabilidad posterior.

$$Z_s = R_s + j \cdot \omega_{ss} \cdot L_s \tag{3.17}$$

Para simplificar los cálculos, se ha supuesto que esta impedancia se mantiene constante en torno al punto de operación del control trapezoidal. Esto significa que se ha linearizado el efecto del polo complejo de $Z_s$, y se ha supuesto constante a la frecuencia de cruce del regulador PI, según la expresión 3.18.

$$\left|Z_{slin}\right| = \sqrt{{R_s}^2 + L_s \cdot {\omega_{ss}}^2} \tag{3.18}$$

De este modo, la ganancia de lazo en función de la impedancia del estátor linearizada $Z_{slin}$ y la constante de flujo magnético $K_m$ en $\frac{V}{revolucin}$ será la de la expresión 3.19.

$$\begin{aligned} K_m &= \frac{1}{K_v \cdot PP} \\ G_{ss} &= \frac{V_{dc}}{2 \cdot Zs_{lin}} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J \cdot s} \end{aligned} \tag{3.19}$$

$$\begin{aligned} J_{min} &= J_{mot} \\ J_{max} &= 1000 \cdot J_{mot} \end{aligned} \tag{3.20}$$

Para calcular las constantes Kp y Ki, será necesario predecir qué aspecto tendrá la función de transferencia en lazo cerrado (figura 3.21). Esta presentará una atenuación inicial de -40dB/década, pero deberá cruzar por 0dB con -20dB/década para que sea estable. Esto se consigue calculando el módulo de la función de transferencia para la frecuencia de cruce del regulador PI, para la función de transferencia con momento de inercia $J_{min}$, que tendrá su codo sobre la frecuencia de cruce del regulador PI.

$$T_{ss} = Pi_{ss} \cdot \frac{V_{dc}}{2 \cdot Zs_{lin}} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J \cdot s} \tag{3.21}$$

De este modo, la expresión de la constante proporcional en la frecuencia $\omega_{ss}$, donde el efecto de Ki se considera nulo, es la que sigue la expresión 3.22.

$$K_{p_{ssmin}} = \frac{J_{min} \cdot \omega_{ss}}{\frac{V_{dc}}{2 \cdot Zs_{mot}} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP} \tag{3.22}$$

De manera similar, se puede calcular la frecuencia del codo de menor frecuencia, que se conseguirá cuando el momento de inercia en la función de transferencia sea $J_{max}$.

$$\omega_{ssmin} = K_{p_{ss}} \cdot \frac{V_{dc}}{2 \cdot Zs_{mot}} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J_{max}} \tag{3.23}$$

Por último, deberá dimensionarse la constante integral del regulador PI sabiendo que habrá dos funciones de transferencia para los dos valores límite de J, y que todo el rango de funciones desde $J_{min}$ hasta $J_{max}$ deberán ser estables. Por ello, se calcula *Ki* según la expresión 3.24, con el fin de lograr que el codo de la función de transferencia para $J_{max}$ esté al menos una década retrasado respecto del paso por 0dB, asegurando la estabilidad de esta.

$$K_{i_{ss}} = \frac{K_{p_{ss}} \cdot \omega_{ssmin}}{10} \tag{3.24}$$

Si una vez se tienen los dos coeficientes del regulador PI se calcula la respuesta en frecuencia de este, se podrá observar que toda la gama de funciones de transferencia desde $J_{min}$ hasta $J_{max}$ son estables. Esto se observa en la figura 3.15.

$$Pi_{ss} = K_{p_{ss}} + K_{i_{ss}} \cdot s \tag{3.25}$$

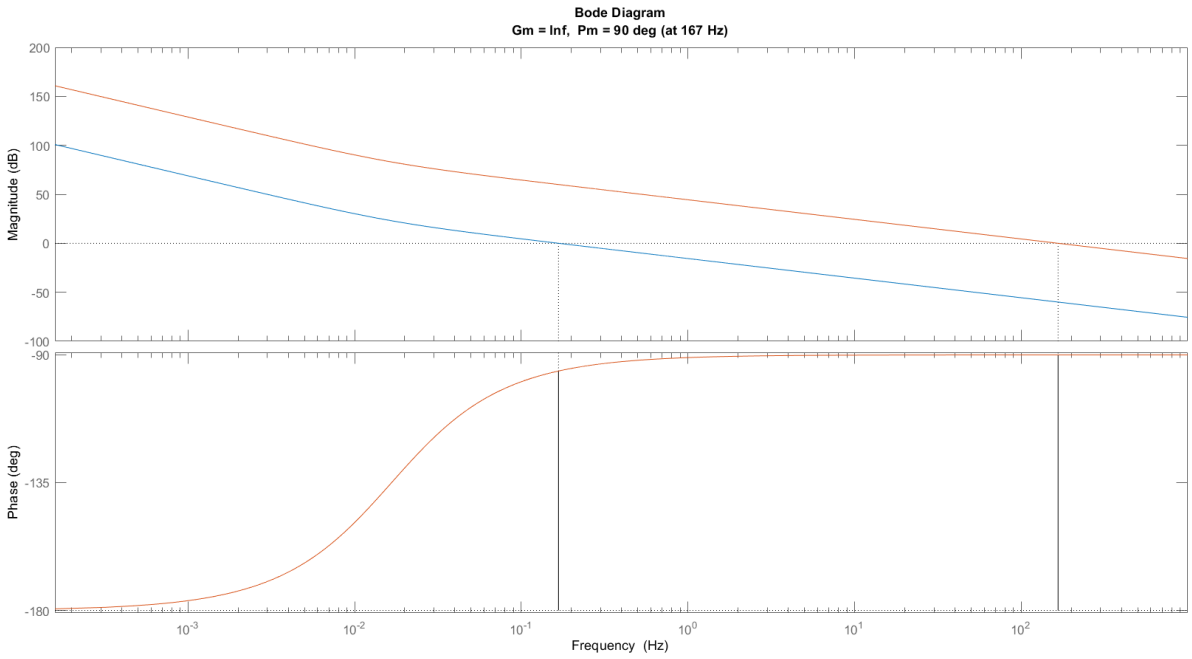


**Figura 3.15: Diagrama de Bode de funciones de transferencia en lazo abierto para control trapezoidal**

## 3.4. DESARROLLO DEL CONTROL FOC

El control FOC es el método de control más complejo de los dos que se han desarrollado, pero también ofrece ventajas claras sobre el trapezoidal como un control del par más preciso, o una mejor adaptación ante escalones de carga.

Como se ha querido simplificar la interfaz mecánica del motor con el fin de no requerir de ningún tipo de medidor externo de velocidad o posición angular, se ha tenido que recurrir a la implementación de otro algoritmo muy novedoso en el mundo de los convertidores de tracción, que es el SOGI-FLL. Este algoritmo será capaz de realizar la estimación de fase del sistema y velocidad angular, para crear el lazo cerrado del control FOC, y poder transformar las medidas de corriente trifásicas a coordenadas móviles de Park, D-Q.

Para el diseño de los lazos de regulación para control FOC y control trapezoidal, se ha creado un script en MATLAB, que se encuentra en el anexo A.1.

### 3.4.1. DESARROLLO DEL LAZO DE CORRIENTE EN COORDENADAS DQO

El desarrollo de los lazos de control de corriente para los ejes D y Q será diferente, ya que como se muestra en la figura 3.16, los dos ejes D y Q se encuentran acoplados, y el diseño de los lazos de corriente puede no ser trivial.

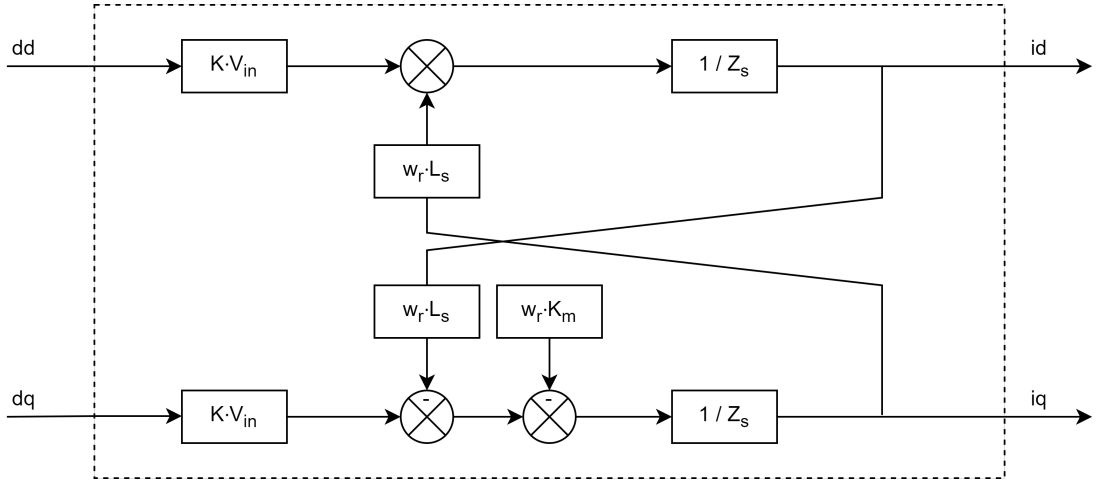


**Figura 3.16: Diagrama de flujo del inversor y motor BLDC en coordenadas D y Q (original)**

Por ello será importante incluir en el diseño de los lazos de control de $i_d$ e $i_q$ los términos de desacoplo necesarios para que el diseño del regulador PI pueda ser independiente de la otra coordenada. Los lazos de control diseñados son el de la figura 3.17 para el eje D, y el de la figura 3.18 para el eje Q.

Gracias a los términos de desacoplo, se obtiene una ganancia de lazo que será la de la ecuación 3.26 y se hace que los lazos de regulación de los ejes D y Q sean iguales.

$$G_i = \frac{K_{inv} \cdot V_{dc}}{Z_s} \tag{3.26}$$

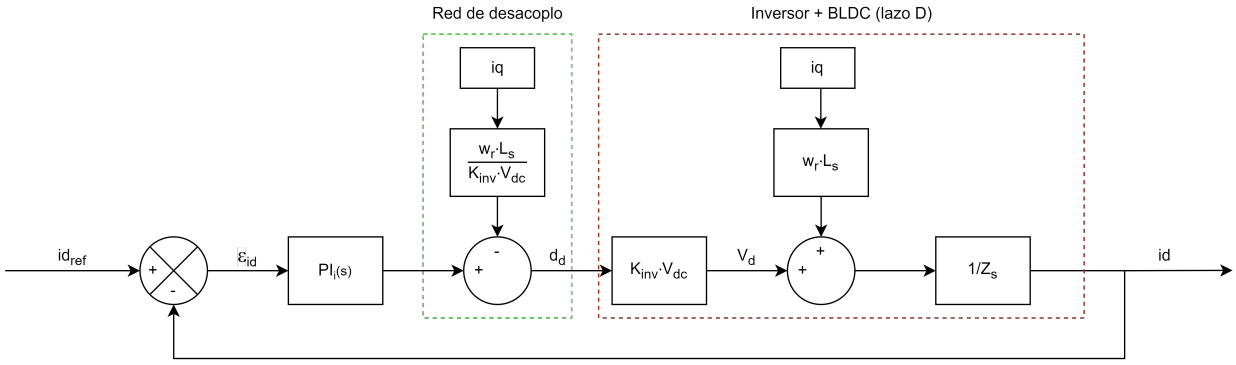


**Figura 3.17: Diagrama de bloques de lazo de corriente en coordenada D (original)**

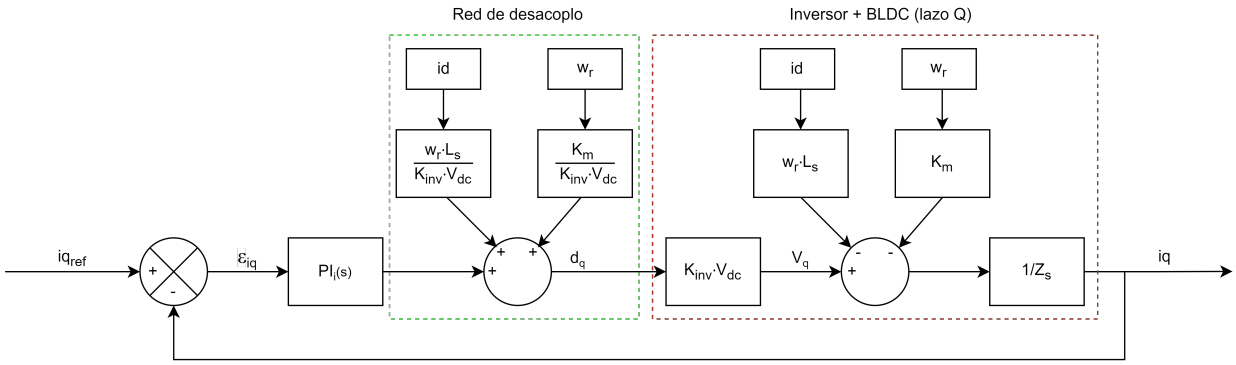


**Figura 3.18: Diagrama de bloques de lazo de corriente en coordenada Q (original)**

En esta ecuación destaca el término $K_{inv}$, que se supondrá que es $\frac{1}{sqrt(3)}$. Esta es la ganancia teórica típica que tendrá un inversor conectado a una máquina senoidal cuando se utiliza modulación SVPWM o se hace inyección del tercer armónico. Como las condiciones de este diseño son similares dado que la máquina que se está utilizando tiene tensiones de fase trapezoidales, y se está haciendo inyección del tercer armónico, se ha estimado que este valor es válido con las condiciones actuales.

Para elegir una frecuencia de corte a la que diseñar el lazo de corriente, habrá que tener en cuenta que el objetivo es que este sea lento, para que pueda adaptarse a cambios en el flujo magnético del rotor de manera adecuada. Esto se consigue de manera típica haciendo que la frecuencia de cruce del regulador sea una vigésima parte de la frecuencia de conmutación general del sistema, que es de 10kHz. De este modo, el resultado es una frecuencia de cruce de 500Hz.

$$\begin{aligned} f_{ci} &= \frac{f_{sw}}{20} = 500Hz \\ \omega_{ci} &= f_{ci} \cdot 2 \cdot \pi = 3141{,}1592 rad/s \end{aligned} \tag{3.27}$$

Una vez se ha hecho esto, se puede proceder al cálculo de las constantes $K_{p_i}$ y $K_{ii}$. Se sabe que en la frecuencia de cruce la ganancia de la función de transferencia en lazo abierto será 0dB, que equivale a una magnitud de 1. Si se calcula ahora el módulo de la función de transferencia, se obtiene la equivalencia en 3.28.

$$1 = \left| K_{p_i} + \frac{K_{ii}}{s} \right| \cdot \left| \frac{K_{inv} \cdot V_{dc}}{Z_s} \right| \tag{3.28}$$

De la equivalencia anterior, se pueden extraer las expresiones 3.29 y 3.30 que permitirán calcular las constantes del regulador PI.

$$K_{ii} = \frac{\omega_{ci} \cdot R_s}{V_{dc} \cdot K_{inv}} \tag{3.29}$$

$$K_{p_i} = K_{ii} \cdot \frac{L_s}{R_s} \tag{3.30}$$

Por último, se puede apreciar en la figura 3.19 la respuesta en frecuencia de la ganancia de lazo, y en la figura 3.20 la respuesta en frecuencia en lazo abierto del lazo de corriente al completo. Se observa que esta pasa por 0dB a 500Hz exactamente.

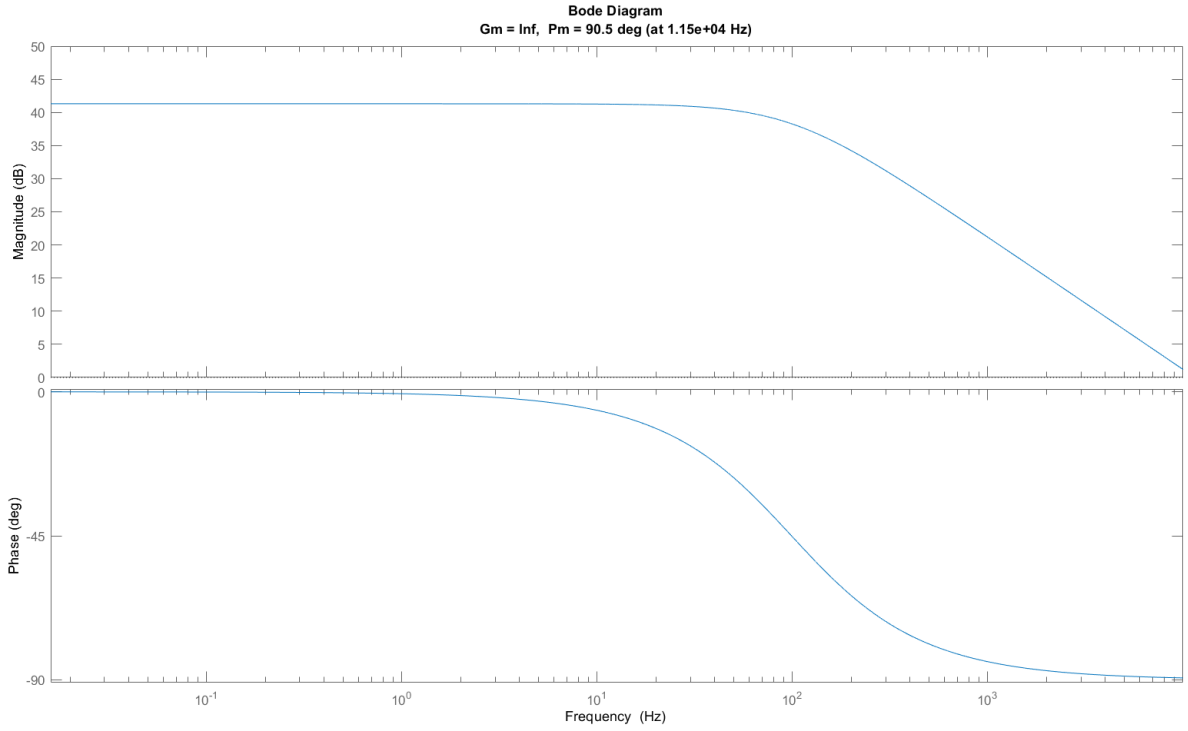


**Figura 3.19: Diagrama de Bode de ganancia de lazo de corriente**

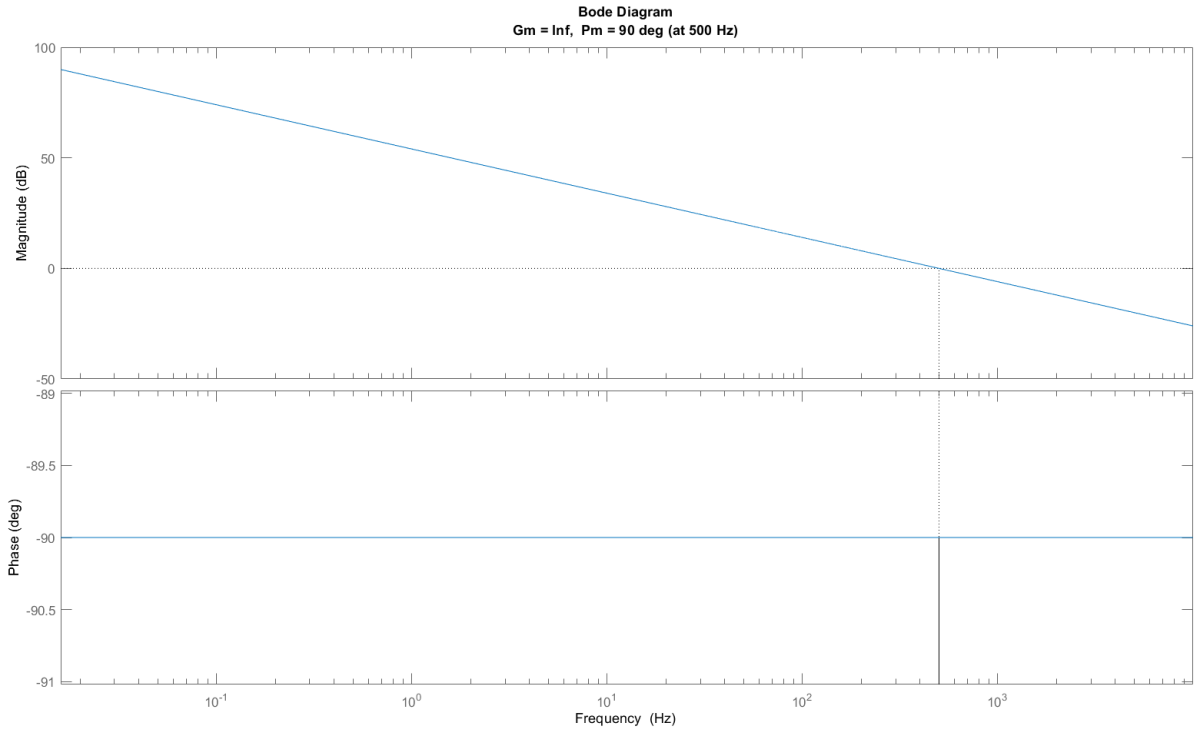


**Figura 3.20: Diagrama de Bode de función de transferencia de lazo de corriente**

Se observa cómo el regulador PI ha conseguido que la atenuación de la función de transferencia sea constante, con una caída de 20dB/década.

### 3.4.2. DESARROLLO DEL LAZO DE VELOCIDAD

Para el desarrollo del lazo de velocidad, habrá que plantear un nuevo lazo de control, que dependerá del lazo de corriente anteriormente diseñado.

Con el fin de simplificar los cálculos, se puede fijar una frecuencia de cruce al menos una década menor que la frecuencia de cruce del lazo de corriente, y de este modo hacer la aproximación $iq_{ref} = iq$. Esto permite suponer que la ganancia desde la entrada hasta la salida del lazo de regulación de la figura 3.18 es unitaria, ya que se supone que el lazo de corriente, al ser más rápido, habrá tenido tiempo suficiente para reducir su error de entrada. De este modo, se puede plantear el lazo de velocidad de la figura 3.21.

Según lo anterior, en este caso se ha fijado una frecuencia de cruce de $\frac{1}{12}$ respecto a la frecuencia del lazo de corriente.

$$\begin{aligned} f_{cn} &= \frac{f_{ci}}{12} = 41{,}6667Hz \\ \omega_{cn} &= f_{cn} \cdot 2 \cdot \pi = 261{,}7996rad/s \end{aligned} \tag{3.31}$$

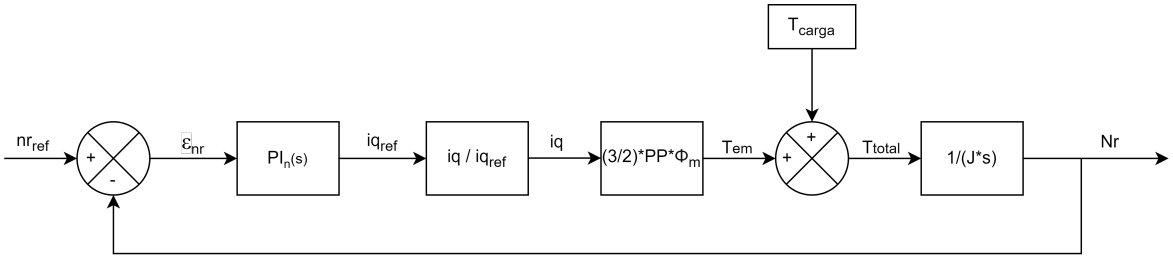


**Figura 3.21: Diagrama de bloques de lazo de velocidad (original)**

De este modo, y teniendo en cuenta las aproximaciones hechas, la ganancia del lazo de velocidad es la que describe la ecuación 3.32. Esta función de transferencia en el dominio de Laplace, será dependiente del momento de inercia J que se aplique en la carga del motor.

$$G_n = \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J \cdot s} \tag{3.32}$$

Si se tiene en cuenta la variación del momento de inercia dentro de los límites máximo y mínimo definidos en la expresión 3.20, podrá definirse todo el rango de funcionamiento del lazo de velocidad. El regulador diseñado tendrá que asegurar que la función de transferencia del sistema en lazo abierto sea estable y tenga un margen de fase próximo a 90 grados para cualquier variación de J dentro del rango definido.

$$\begin{aligned} G_{nmax} &= \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J_{min} \cdot s} \\ G_{nmin} &= \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J_{max} \cdot s} \end{aligned} \tag{3.33}$$

Para calcular las constantes *Kp* y *Ki*, habrá que tener en cuenta que el cruce de la función de transferencia en lazo abierto, cuando se aplica un factor proporcional *Kp*, estará situado originalmente en la frecuencia de cruce. Esta es la frecuencia donde se cruza por 0dB, o lo que es lo mismo, una ganancia unitaria. En las expresiones de 3.34 se puede observar el desarrollo seguido hasta hallar la expresión que caracteriza a $K_{p_n}$.

$$\begin{aligned} T_{nmax} &= Pi_n \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J_{min} \cdot s} \\ T_{nmax}\,|_{wcn} &= 1 = K_{p_n} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J_{min} \cdot \omega_{cn}} \\ K_{p_n} &= \frac{J_{min} \cdot \omega_{cn}}{\frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP} \end{aligned} \tag{3.34}$$

De manera análoga al desarrollo hecho en 3.34, se puede calcular la frecuencia a la que se producirá el cruce por 0dB de la función $T_{nmin}$, sustituyendo la constante $K_{p_n}$ recién calculada, con el valor de momento de inercia máximo Jmax. Esto se detalla en la expresión 3.35.

$$\omega_{cnmin} = K_{p_n} \cdot \frac{60}{2 \cdot \pi} \cdot \frac{3}{2} \cdot K_m \cdot PP \cdot \frac{1}{J_{max}} \tag{3.35}$$

Por último, la constante $K_{in}$ es la que crea el codo del sistema, que hará que la función de transferencia en lazo abierto cruce por 0dB con una ganancia de -20dB/década, y sea estable. Debido a esto, el objetivo será situar el codo del sistema a una proporción de la frecuencia de cruce original del sistema, suficientemente pequeña como para que el sistema se estabilice antes del cruce por 0dB, pero tampoco excesivamente pequeña, porque esto haría que el regulador PI fuese demasiado lento. Además, habrá que contemplar el caso más restrictivo, o lo que es equivalente, la frecuencia de cruce más pequeña dentro de todos los valores de $T_n$ para todos los valores de momento de inercia. En este caso, se ha situado el codo del sistema a 1/5 de la frecuencia de cruce mínima.

$$K_{in} = \frac{K_{p_n} \cdot \omega_{cnmin}}{5} \tag{3.36}$$

Por último, es interesante observar las respuestas en frecuencia de la ganancia de lazo en la figura 3.22, donde se observan las frecuencias de cruce originales del sistema, y de la función de transferencia en lazo abierto en la figura 3.23, donde se observa que el margen de fase es bastante bueno para todo el rango de funcionamiento del control.

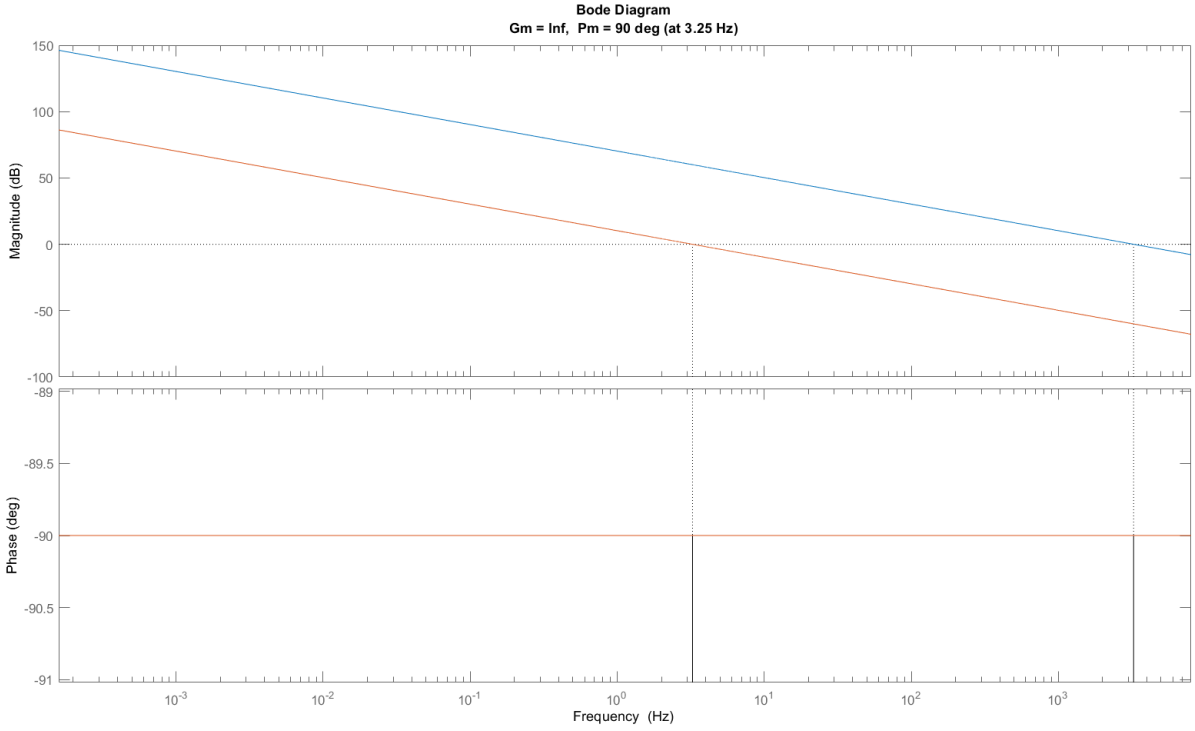


**Figura 3.22: Diagrama de Bode de ganancia de lazo de velocidad**

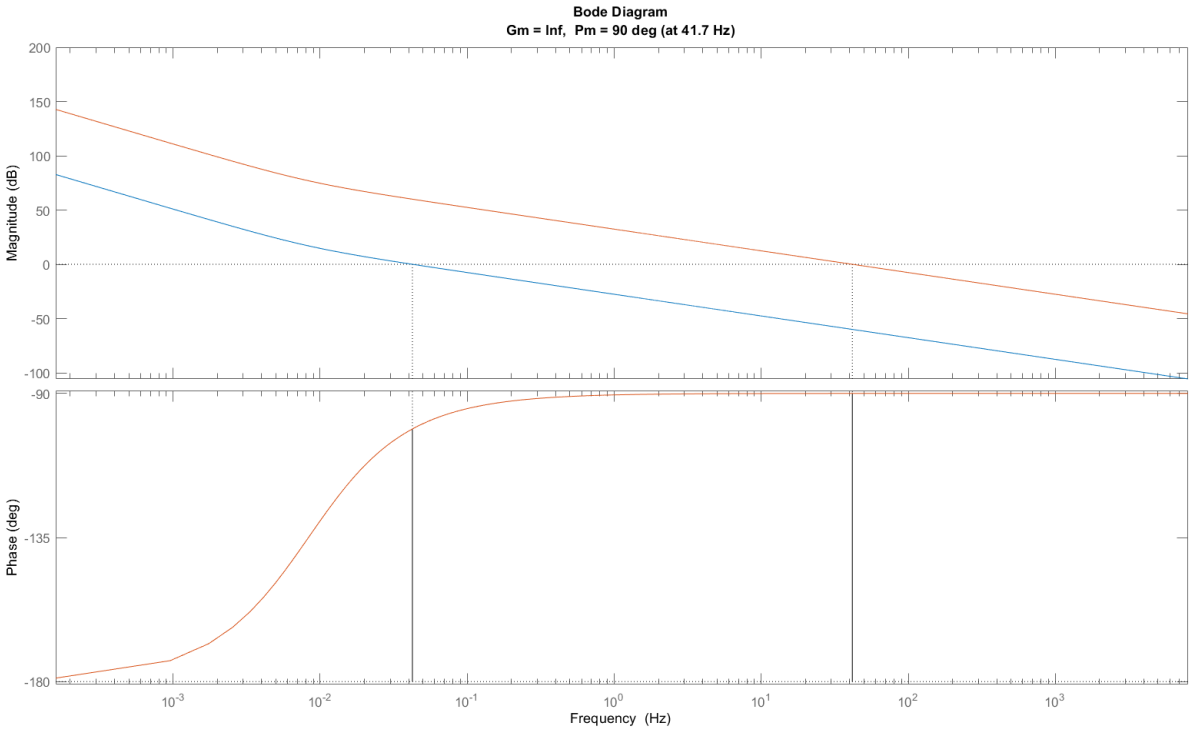


**Figura 3.23: Diagrama de Bode de función de transferencia de lazo de velocidad**

## 3.5. IMPLEMENTACIÓN DEL SOGI-FLL TRIFÁSICO PARA ESTIMACIÓN DE FASE Y VELOCIDAD ANGULAR

En este apartado se desarrollarán todos los conceptos teóricos y prácticos sobre la tecnología que cohesiona el completo diseño del motor, permitiendo el arbitraje entre los métodos de control, la realimentación de velocidad, y la creación de una fase general del sistema. Este es el SOGI-FLL, y en este diseño se utiliza para obtener estimaciones muy precisas sobre la fase del motor, y su velocidad angular. Aunque es un mecanismo común en conversores de potencia para generación energética en su tarea de sincronismo con la red eléctrica, su uso para el sincronismo con motores es mucho menos frecuente.

### 3.5.1. CONCEPTOS GENERALES SOBRE SOGI-FLL

Un SOGI-FLL es un sistema que se compone por dos componentes principales, un SOGI-QSG [I], y un FLL [11].

De manera resumida, el SOGI-QSG es un filtro paso banda adaptativo, que resuena a la frecuencia $\omega'$ que se muestra en la parte inferior de la figura 3.24. Esta frecuencia de resonancia se puede controlar de manera externa, pero en la realidad no es demasiado práctico, y por eso se opta por añadir un elemento externo que realice esa función. Esto es tarea del FLL, o ”Bucle de seguimiento de frecuencia”, que creará la frecuencia de resonancia del sistema a partir del error de la salida respecto de la entrada $\epsilon_v$, y la componente de salida en cuadratura.

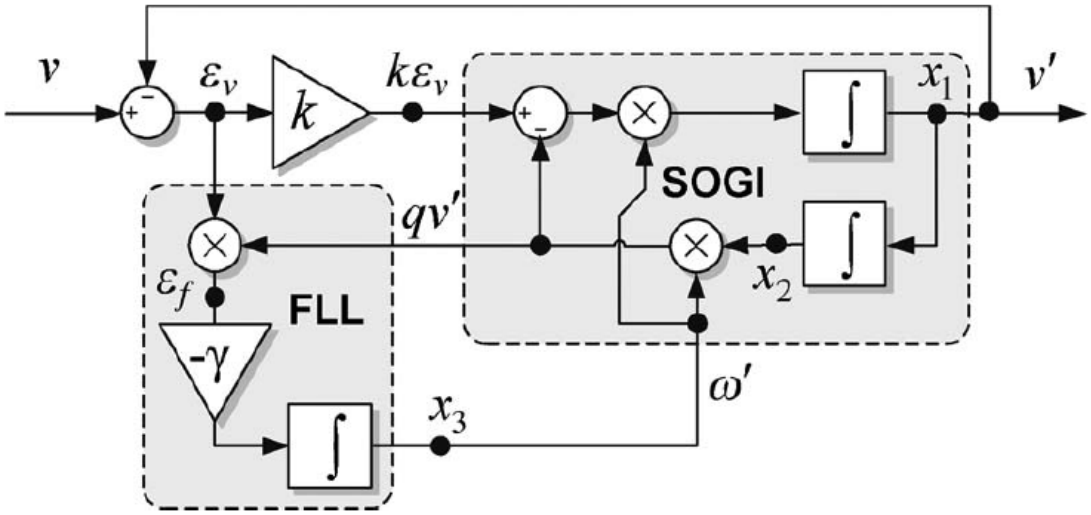


**Figura 3.24: Esquema de SOGI-FLL monofásico [12]**

El SOGI-QSG, junto con el FLL, serán bloques que permitirán obtener a su salida la componente fundamental de una señal de entrada con distorsiones. Esto significa, aplicado al diseño de este documento, que podrán obtenerse tres señales senoidales casi perfectas a partir de los sensores de efecto Hall del motor BLDC. Esto se logra con dos integradores en cascada, que generan una señal directa y otra en cuadratura (desfase de 90 grados).

En el diseño que se ha desarrollado, se utilizan las tres componentes A, B y C de los sensores de efecto Hall para obtener unas componentes Alfa y Beta en coordenadas de Clarke. Si bien se podría haber implementado un SOGI-FLL con una sola coordenada, de manera similar a la figura 3.24, esto hubiese sido una solución mucho menos precisa al suponer la pérdida de ciclos de conmutación de los sensores en la estimación de la fase y la velocidad angular.

El segundo bloque que se observa es el FLL. Este es un algoritmo que se encarga de sincronizarse con una cierta frecuencia de entrada. Este es el bloque que se utilizará para averiguar la frecuencia angular del rotor del motor con control FOC. En su versión trifásica, el FLL admite como parámetros de entrada el error y salida de cuadratura del SOGI-QSG Alfa, y el error y salida de cuadratura

[I] Second Order Generalized Integrator - Quadrature Signal Generator

del SOGI-QSG Beta. Esto significa, que la frecuencia de salida será una media de la frecuencia central de las ramas Alfa y Beta, ponderadas por sus respectivos errores.

El último bloque es el PNSC, que actuará como un promediador entre las componentes directas y de cuadratura de los SOGI-QSG Alfa y Beta. A su salida se encuentran dos pares de señales, $V^{+'}{}_{\alpha}$, $V^{+'}{}_{\beta}$, $V^{-'}{}_{\alpha}$ y $V^{-'}{}_{\beta}$ [13]. Las dos primeras representarán las componentes fundamentales en coordenadas Alfa y Beta de los sensores Hall, sin distorsiones.

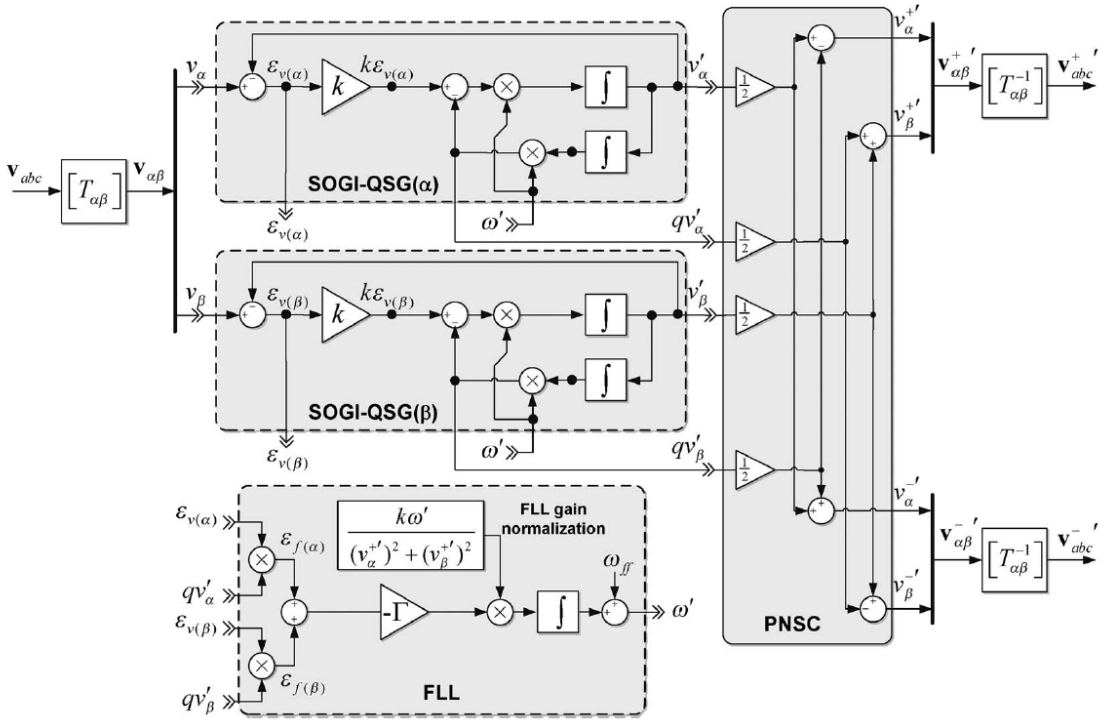


**Figura 3.25: Esquema de SOGI-FLL trifásico [13]**

Una ventaja muy clara del SOGI-QSG respecto de otros elementos de sincronismo como los PLL será que el primero no utiliza funciones trigonométricas, lo que acelerará en gran medida su posterior implementación en DSP.

### 3.5.2. DESARROLLO DEL SOGI-FLL TRIFÁSICO EN MATLAB

El SOGI-QSG presenta dos funciones de transferencia, $D(s)$ y $Q(s)$, con las que se obtendrán la componente directa ($v'$) y componente de cuadratura ($qv'$) respectivamente frente a la entrada. Estas dos funciones se muestran en 3.37 y 3.38 [14].

$$D(s) = \frac{K_{qsg} \cdot \omega_{sogi} \cdot s}{s^2 + K_{qsg} \cdot \omega_{sogi} \cdot s + \omega_{sogi}^2} \tag{3.37}$$

$$Q(s) = \frac{K_{qsg} \cdot \omega_{sogi}^2}{s^2 + K_{qsg} \cdot \omega_{sogi} \cdot s + \omega_{sogi}^2} \tag{3.38}$$

El parámetro de ganancia $K_{QSG}$ controlará el tiempo de establecimiento del SOGI-QSG. Este se podrá hacer más rápido a costa de disminuir el ancho de banda del filtro. Si se elige un número de establecimiento de 5 revoluciones del motor en su velocidad máxima (condición más restrictiva de sincronismo), se podrá calcular el valor del periodo de establecimiento $Ts_{qsg}$ tal y como marca la expresión 3.40. Para la elaboración de los cálculos, se supondrá que el motor que se elija tendrá una velocidad máxima en torno a 2000 RPM.

$$n_{rmax} = 2000RPM \tag{3.39}$$

$$\begin{aligned} f_{qsg} &= \frac{n_{rmax} \cdot PP}{60} \\ Ts_{qsg} &= \frac{1}{f_{qsg}} \end{aligned} \tag{3.40}$$

$$K_{qsg} \approx \frac{10}{Ts_{qsg} \cdot 2 \cdot \pi \cdot f_{qsg}} \tag{3.41}$$

Ahora se deberá ajustar el FLL de acuerdo con la función de transferencia descrita en 3.42. Esta indica una relación entre la frecuencia de conmutación de las componentes Alfa y Beta de los sensores Hall ($\omega$), con la frecuencia angular del rotor estimada por el FLL ($\omega'$).

$$FLL(s) = \frac{\omega'}{\omega} = \frac{\Gamma_{FLL}}{\Gamma_{FLL} \cdot s} \tag{3.42}$$

De manera similar al SOGI-QSG, el objetivo será dimensionar $\Gamma_{FLL}$ para que el tiempo de establecimiento del FLL sea de 5 ciclos de red. Esto se consigue mediante el desarrollo de las expresiones en 3.43 [15].

$$\begin{aligned} Ts_{FLL} &\approx \frac{5}{\Gamma_{FLL}} \\ \Gamma_{FLL} &\approx \frac{5}{Ts_{FLL}} \end{aligned} \tag{3.43}$$

## 3.6. ESTIMACIONES EN EL DISEÑO DEL ENTORNO DE SIMULACIÓN

El objetivo que se plantea antes de comenzar el diseño del sistema de control híbrido, será realizarlo de modo que este sea versátil en su funcionalidad, y fácilmente integrable dentro de sistemas más complejos. Por ello, se propone la creación de un sistema de control híbrido-trapezoidal con un solo motor BLDC. El método de control principal será un control en lazo cerrado por velocidad, que será de extrema utilidad en aplicaciones como vehículos personales, vehículos radiocontrol o vehículos para mercancías ligeras en un entorno industrial, de un almacén o hasta de una biblioteca.

Con el fin de tener un primer acercamiento a unas especificaciones reales, se han tomado como ejemplo algunos modelos de 'quads' eléctricos, por ser estos vehículos diseñados para tener la máxima versatilidad en la conducción. La potencia de uno de estos vehículos puede variar entre 1kW y 3kW, por lo que si se establece una potencia de en torno a 1.4kW, podrá diseñarse una cadena de tracción distribuida en cuatro partes, de en torno a 345W cada una, de acuerdo con lo descrito en la figura 1.2. Para posteriores cálculos, se considerará esta especificación como la referencia para el diseño.

Si hablamos del tipo de control que se va a implementar, este será siempre un control en lazo cerrado. En cualquiera de las dos metodologías de control que se van a utilizar el funcionamiento será similar.

1. Obtener un ciclo de trabajo en el DSP, que se usará para modular por ancho de pulso (PWM) las tensiones de línea del motor (Va, Vb, Vc).
2. Leer los parámetros disponibles en el motor. En el caso de esta implementación, serán los sensores de efecto Hall de las ramas A, B y C.
3. Procesar adecuadamente los datos recogidos de las lecturas de los sensores Hall para generar nuevos ciclos de trabajo para las tres ramas.

Un esquema que ejemplifica esto es el de la figura 3.26.

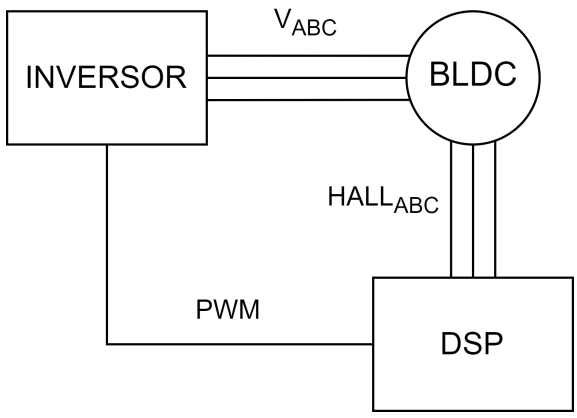


**Figura 3.26: Diagrama simplificado del lazo cerrado (original)**

### 3.6.1. ELECCIÓN DE UN MOTOR BLDC COMERCIAL

De acuerdo con lo planteado anteriormente, se ha realizado la búsqueda de un motor BLDC que se adapte a las necesidades de potencia. Para ello se ha contactado con proveedores de motores BLDC internacionales, y se ha conseguido una máquina que cumple con los requisitos anteriormente planteados. El uso de unas características reales en el diseño del sistema en un entorno de simulación será fundamental para poder obtener resultados coherentes y precisos, que demuestren el correcto funcionamiento de los algoritmos de control.

El motor seleccionado, de 345W de potencia, es el que se muestra en la hoja de datos de la figura 3.27. Se puede observar que las características del motor aparecen representadas a menudo en forma de constantes como la constante de par $K_t$ o la constante de velocidad $K_v$.

| Model | D80BLD350-48V-10S | |
|---|---|---|
| ITEM | UNIT | SPEC |
| PHASE | PHS | 3 |
| VOLTAGE | VDC | 48 |
| ROTOR POLES | POLES | 8 |
| RESISTANCE @25° C | Ω | 0.596(1±10%) |
| INDUCTANCE | mH | 0.96(1±20%) |
| ROTOR INERTIA | g.cm$^2$ | 168 |
| SPEED CONSTANT | RPM/V | 41.7 |
| TORQUE CONSTANT | N.m/A | 0.33 |
| NOLOAD SPEED | RPM | 2000REF |
| NOLOAD CURRENT | A | 1.1REF |
| RATED SPEED | RPM | 1000 |
| RATED POWER | W | 345 |
| RATED TORQUE | N.m | 3.3 |
| RATED CURRENT | A | 11 |
| INSULATING STRENGTH | VAC | 500 |
| IP CLASS | | IP40 |
| INSULATION CLASS | | B |
| WEIGHT | Kg | 3.0 |

**Figura 3.27: Hoja de datos del motor BLDC**

Añadido a los datos anteriores, se puede observar que el motor, dibujado en la figura 3.28 tiene unas dimensiones muy reducidas, de 13x8x8 cm. Esta es una de las ventajas que se obtienen del planteamiento de la cadena de tracción distribuida, además de una reducción en el tamaño de los semiconductores de potencia.

Como se comentó en el apartado 3.1.1, no se proporciona en la hoja de características ningún valor para el flujo magnético $\Phi_m$. Esto se debe a que los fabricantes a menudo asumen que sus máquinas se controlarán con modulación trapezoidal.

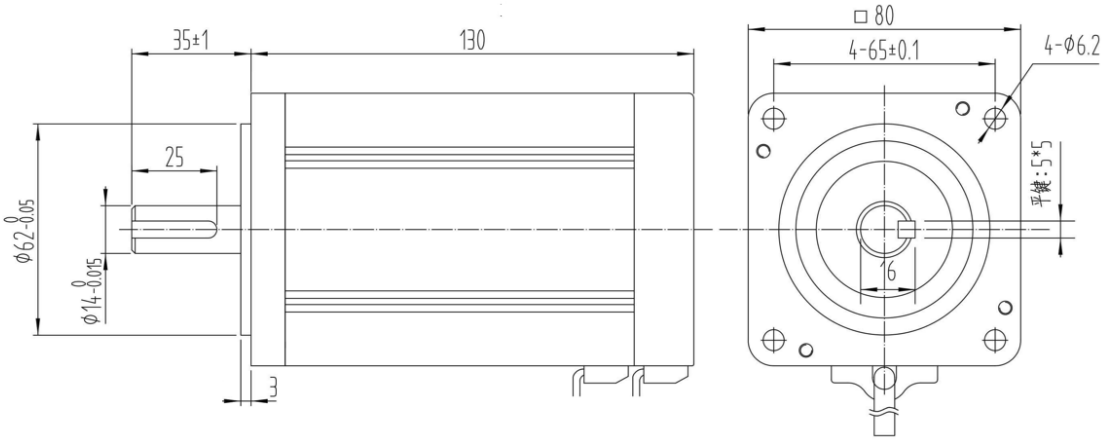


**Figura 3.28: Motor BLDC según hoja de datos [16]**

### 3.6.2. DIMENSIONAMIENTO DE UNA BATERÍA

La elección de una batería estará condicionada por la autonomía que se espera tener en el sistema final. Suponiendo la aplicación de cuatro motores de la figura 1.2 con 345W por motor, funcionando a máxima potencia, habrá un consumo de potencia de 1.4kW despreciando las de elementos pasivos.

Como el motor seleccionado admite una tensión entre fases de 48V, se dimensionará la batería con una tensión algo superior a esta. De este modo, se mitigarán las pérdidas de rendimiento que puedan provocar las variaciones de carga en el eje del motor, o el rizado en la tensión de entrada. En este caso, se seleccionarán baterías de 60V.

Si se pretende alimentar cuatro cadenas de tracción de manera simultánea, con esta batería de 60V, y trabajando con potencia máxima de 1.4kW, se tendrá un consumo de corriente de 25A. Es por esto que de entre las opciones disponibles, se ha elegido una batería de 60V y 40Ah, de ión de litio [17].

Dentro de la familia de las baterías de ión de litio, la tecnología que utiliza esta es la NMC ($LiNiMnCoO_2$). Este tipo de celdas son muy utilizadas habitualmente en convertidores de potencia de herramientas industriales, bicicletas eléctricas, y vehículos eléctricos [18]. Si bien sus características son muy equilibradas, destaca sobre todo la energía específica de esta, que será la energía por unidad de masa que contendrá la batería. Por otro lado, es una tecnología también muy utilizada en el almacenamiento energético para sistemas de generación energética, debido a que conserva muy bien sus propiedades ante ciclos de carga y descarga repetitivos [19].

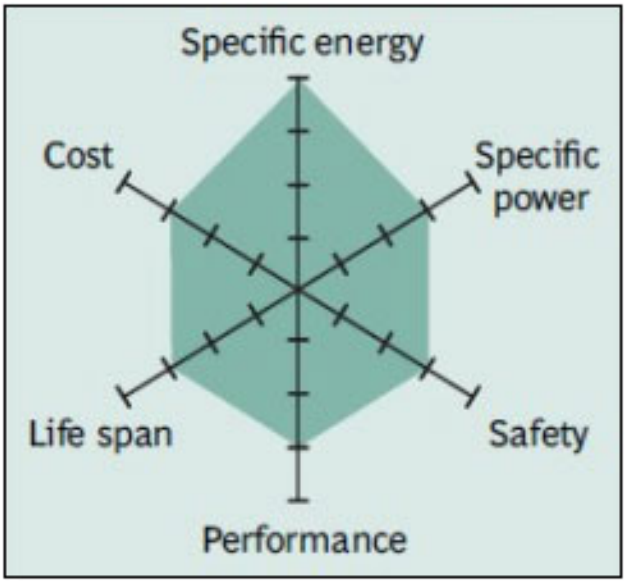


**Figura 3.29: Características de las baterías Li-ion con tecnología NMC [19]**

### 3.6.3. SELECCIÓN DE UN CONDENSADOR DC-LINK COMERCIAL

Para elegir un condensador apropiado para el diseño, se han consultado diversas opciones de fabricantes de condensadores electrolíticos de aluminio. Los objetivos son que este condensador presente una ESR lo más baja posible, y una capacidad lo más alta posible, siendo esta como mínimo la capacidad calculada en el apartado anterior. Una capacidad más alta, solo hará que la frecuencia de corte sea todavía más restrictiva.

En este caso, se ha optado por los condensadores electrolíticos del fabricante KEMET[20]. Se ha elegido un condensador de 4.7mF, con una tensión máxima de 100V, una $ESR\ |_{100Hz}= 46\Omega$ y una $ESR\ |_{10000Hz}= 36\Omega$. Como se prevé que la frecuencia de corte del filtro con los componentes seleccionados sea similar a la frecuencia de 1kHz teórica, será necesario calcular el valor de ESR a 1kHz. No obstante, el fabricante no proporciona este valor, por lo que observando las curvas típicas de impedancia de los condensadores electrolíticos, se ha decidido utilizar el valor de $ESR$ a 1kHz, que además será el caso más desfavorable de funcionamiento.

Con el fin de aumentar la capacidad total del filtro y disminuir el valor de resistencia ESR, se prevé colocar dos de estos condensadores en paralelo, de modo que su impedancia total sea la que dicta la ecuación 3.44.

$$Z_{DCLINK} = Z_c \,||\, Z_C = (2 \cdot ESR_C) + \frac{1}{j \cdot \omega \cdot 2 \cdot C} \tag{3.44}$$

Si se plantea la ecuación de transferencia 3.9 de nuevo con esta nueva impedancia, se puede obtener la gráfica 3.30, donde la nueva frecuencia de corte es de 406Hz, y la atenuación total es de en torno a 8dB.

Todos los cálculos relativos al diseño del DC-LINK se encuentran en el script de MATLAB del anexo A.2.

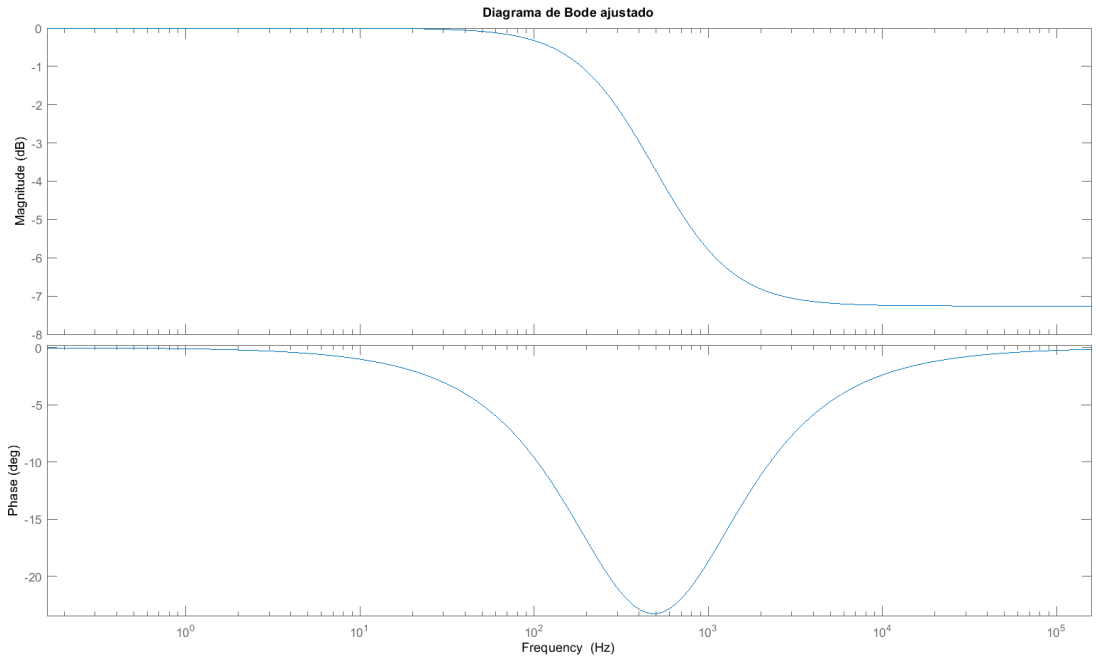


**Figura 3.30: Diagrama de bode del filtro DC-LINK con componentes reales**

## 3.7. IMPLEMENTACIÓN Y SIMULACIÓN DEL SISTEMA EN PSIM

En este apartado se desarrollará toda la implementación del sistema propuesto en el software de simulación PSIM. En él se creará el inversor trifásico con el motor BLDC a su salida, el SOGI-FLL y los lazos de control. En la figura 3.31 se puede observar un diagrama de bloques general que representa, de manera muy visual, toda la implementación de PSIM. A continuación, se detallan los elementos principales con los que cuenta este diagrama:

- **Generador de error**: Este es el elemento que crea el error que proporcionará a los PI del control trapezoidal y FOC para que lleven a cabo la regulación. En ambos casos, el error generado es un error de velocidad, $Nr - Nr_{ref}$.
- **Lazos de control**: Se aprecian los lazos de control trapezoidal y FOC, en ramas separadas. Cada uno de ellos generará unas señales de disparo, que se seleccionarán en estados de funcionamiento alternos.
- **FSM central y multiplexores**: La FSM central del sistema se utilizará para definir los diferentes escenarios de funcionamiento del motor, y arbitrará el uso de las dos medidas de velocidad posibles, además del uso de uno u otro mecanismo de control de velocidad.
- **Cálculo de velocidad por conmutaciones**: Este bloque tiene un periodo de muestreo de 20kHz (esto permite ganar ancho de banda al ser del doble de frecuencia que la frecuencia general del sistema, de 10kHz), y toma como entrada los tres sensores de efecto Hall. Si en

ninguno de los tres sensores se produce una conmutación de estados, se incrementa un contador. Cuando se produzca cualquier conmutación, se podrá calcular un paso en la velocidad del sistema como $N_r = \frac{60}{4 \cdot n_{samples} \cdot T_s \cdot 6}$. El motivo de la multiplicación por seis es que dentro de una revolución completa del rotor, habrá seis posibles estados de conmutación de los sensores Hall. Por eso en cada revolución del motor se realizan seis cálculos de velocidad, algo que como se verá más adelante, puede suponer una clara limitación a RPM bajas.

- **SOGI-FLL trifásico**: El SOGI-FLL es el núcleo del diseño, ya que permite desvincularlo de elementos mecánicos externos para medida de velocidad angular y fase, y también permite realizar el control FOC. La inclusión del SOGI-FLL en un diseño con motores BLDC es algo novedoso, ya que a diferencia de en los sistemas de generación, el SOGI-FLL tendrá que adaptarse ante variaciones de frecuencia muy grandes.

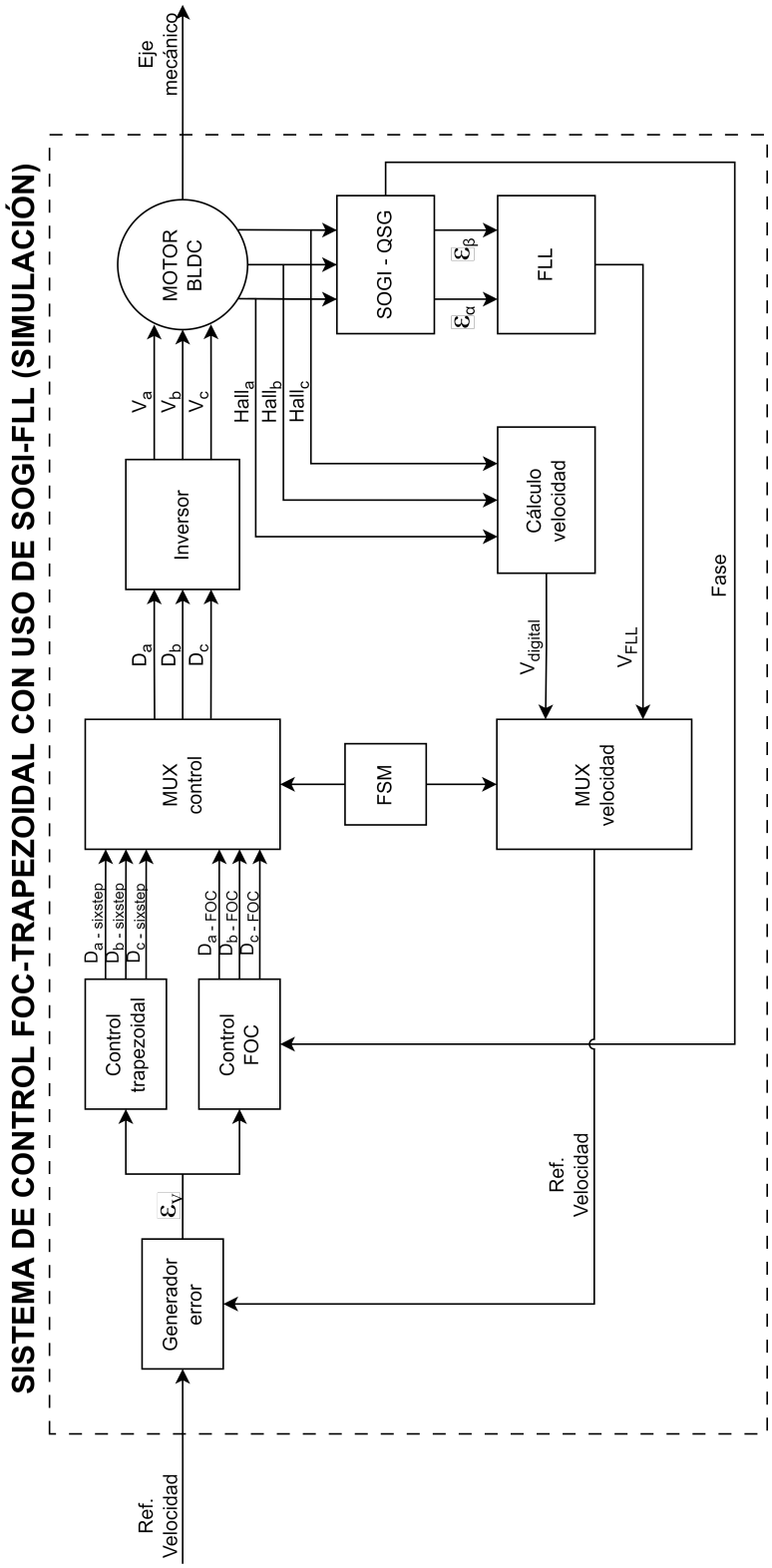


**Figura 3.31: Diagrama de bloques del modelo desarrollado para la simulación (original)**

### 3.7.1. ESQUEMA DESARROLLADO EN PSIM

En las siguientes figuras se presenta el esquema que ha sido desarrollado en PSIM. Este está compuesto por tres secciones principales:

- **Esquema del inversor en PSIM**: En la figura 3.32 se puede observar el inversor trifásico conectado al motor BLDC.

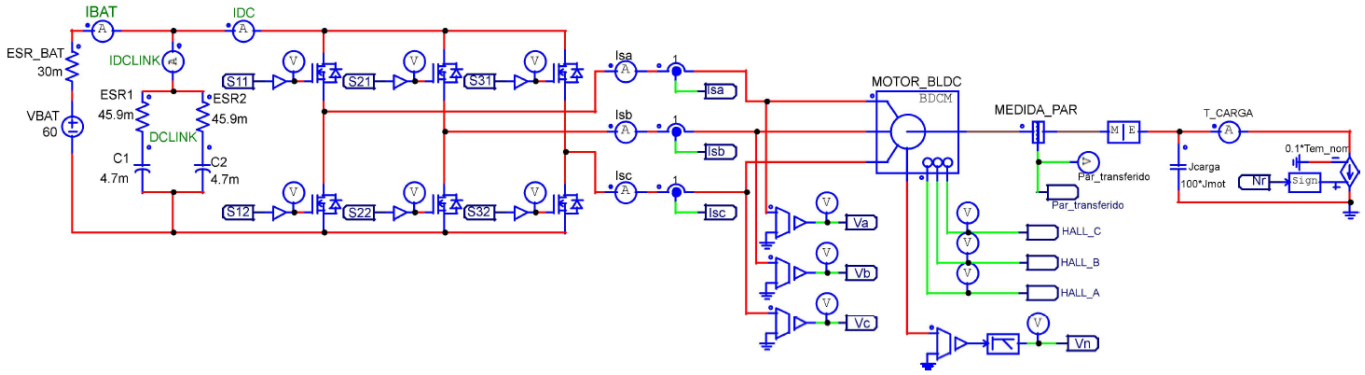


**Figura 3.32: Inversor de tracción modelado en PSIM**

- **Lazos de control y transformadas**: En la figura 3.33 se ha detallado la estructura de los lazos de control híbridos, junto con todos los otros bloques que servirán para hacer el arbitraje en el control, a excepción del SOGI-FLL que se detallará más adelante. El bloque 'FREQ_CALCULATOR_FSM' es uno de los más importantes, ya que es el que se encarga de gestionar la máquina de estados de la figura 3.34, y del cálculo de velocidad por conmutaciones de los sensores Hall.

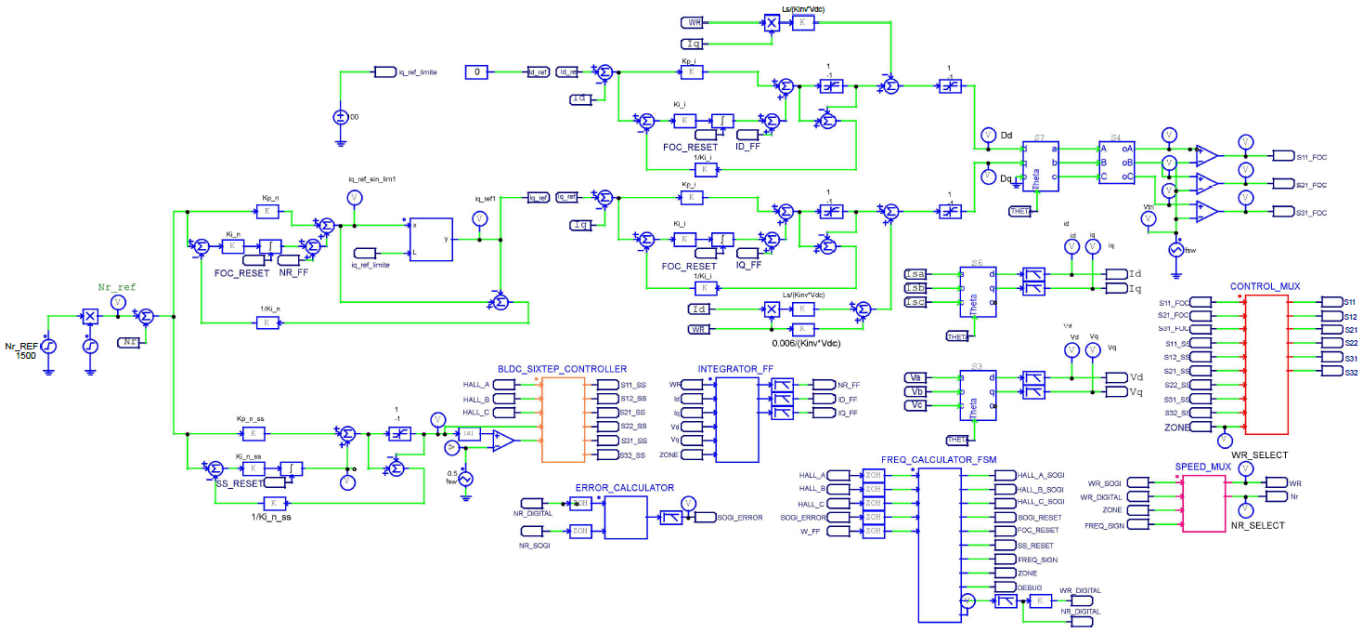


**Figura 3.33: Lazos de control implementados en PSIM**

El funcionamiento de la máquina de estados será sencillo. En primer lugar, sabemos que necesitamos una medida para determinar cuándo es un buen momento para hacer el cambio entre control FOC y control trapezoidal. Esa medida vendrá determinada por el cálculo del error proporcional 3.45 entre la velocidad por conmutaciones (en las simulaciones definida como 'NR_DIGITAL'), y la velocidad del SOGI-FLL (definida en simulación como 'NR_SOGI').

$$\epsilon_{nr} = || \frac{Nr_{sogi}}{Nr} \mid -1 \mid \cdot 100 \tag{3.45}$$

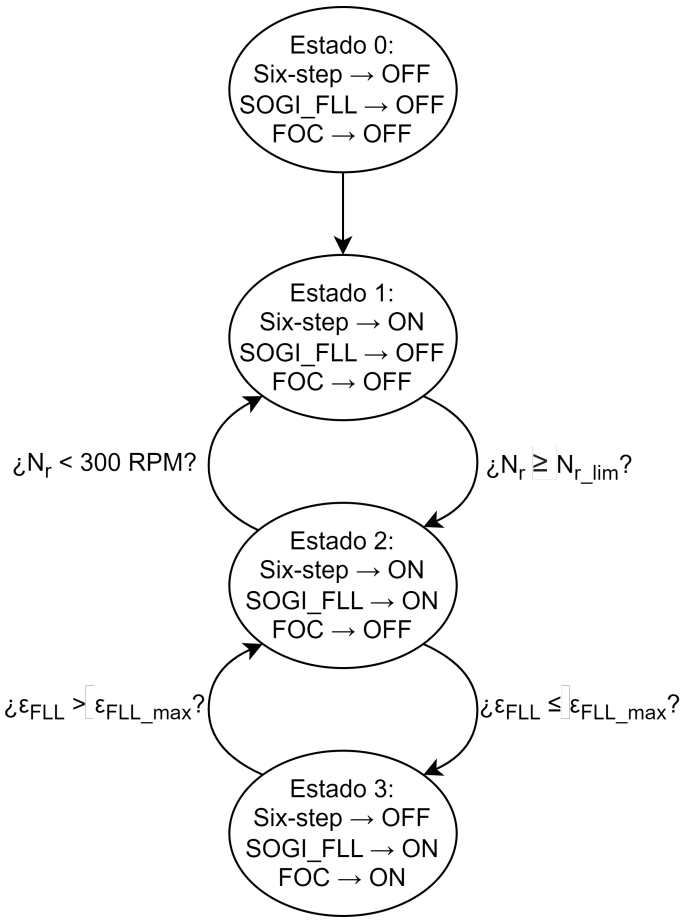


**Figura 3.34: FSM del arbitraje en el control de simulación (original)**

Suponiendo un escenario real de funcionamiento, cuando el sistema parta del reposo y se genere una curva de velocidad ascendente, en primer lugar se activará el control *Six-step*. Tras esto, cuando se alcance una velocidad límite que en esquema se ha definido como $Nr_{lim}$, se liberará el reset en los integradores del SOGI-FLL, se fijará el valor de sus alimentaciones de *feed-forward*, y comenzará su sincronización, aunque el control seguirá siendo trapezoidal porque el FOC todavía no está listo para funcionar (por no tener una buena estimación de fase para la transformada de Park). Cuando el error proporcional baja por debajo de un cierto umbral, se procede a realizar el cambio en el control, se deja el control trapezoidal en estado de *reset*, y se libera el control FOC. También se actúa sobre el bloque 'SPEED_MUX' para cambiar la medida de velocidad en la realimentación por la nueva medida del FLL, que

es mucho más precisa, y sobre el bloque 'CONTROL_MUX' para cambiar las señales de disparo a las del FOC.

- **SOGI-FLL trifásico y estimaciones de fase y velocidad**: El SOGI-FLL trifásico que se desarrolló en el apartado 3.5.1 ha sido implementado por completo en PSIM, para desarrollar el esquema de la figura 3.35. Se puede observar además, cómo el cálculo de la constante $K_{QSG}$ se realiza de manera dinámica. Esto es imprescindible para un sistema donde la variación de frecuencia será tan grande como en el rango de funcionamiento de este motor.

  Un punto importante a destacar es el cálculo de la fase del sistema. Este se realiza tomando las componentes de salida ${v_\alpha}'^{+}$ y ${v_\beta}'^{+}$, y calculando la arcotangente. Como el SOGI-FLL no es capaz de sincronizarse con secuencias de entrada con fase negativa (o lo que es lo mismo, velocidad angular negativa en el rotor), deberá tenerse en cuenta el signo de la velocidad para actuar sobre la fase de la componente ${v_\beta}'^{+}$. Así se obtendrá una medida de fase que representará el sentido de giro del rotor.

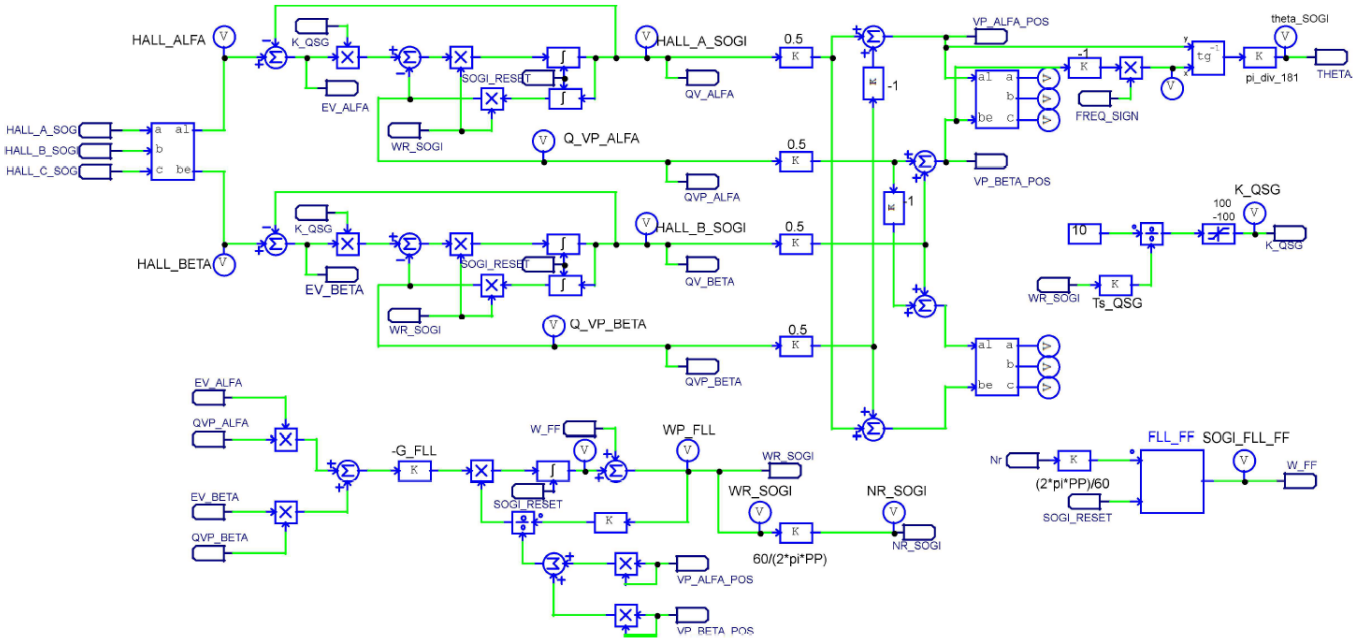


**Figura 3.35: SOGI-FLL implementado en PSIM**

## 3.7.2. DESARROLLO DE LA INYECCIÓN DE TERCER ARMÓNICO

En este diseño, la inyección del tercer armónico se convierte en un elemento fundamental para obtener el máximo rendimiento del motor BLDC con modulación PWM tradicional. Debido a que el motor BLDC es una máquina con una fuerza contraelectromotriz trapezoidal, será necesario considerar esto a la hora de diseñar las tensiones para cada rama. El objetivo será por tanto, que los ciclos de trabajo $D_a$, $Db$, y $D_c$ tengan una forma de onda lo más trapezoidal posible. Es muy frecuente la utilización de modulación SVPWM en motores BLDC, pero se espera que con la inyección de tercer armónico se consiga un efecto similar en cuanto a rendimiento.

En la figura 3.36 se puede encontrar el circuito de pruebas, que se ha evaluado con una entrada sinusoidal trifásica equilibrada.

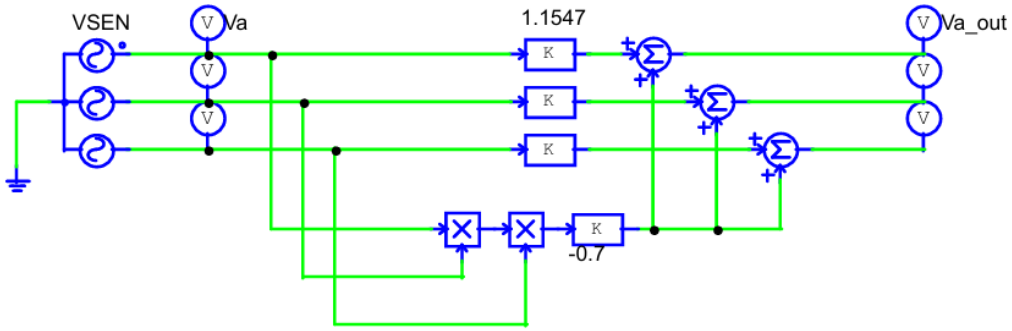


**Figura 3.36: Diseño de subcircuito para inyección de tercer armónico**

Como resultado se obtiene en la figura 3.37 una forma de onda similar a la trapezoidal, que permitirá tener un mayor número de conmutaciones a nivel alto, y obtener un mayor rendimiento en el motor.

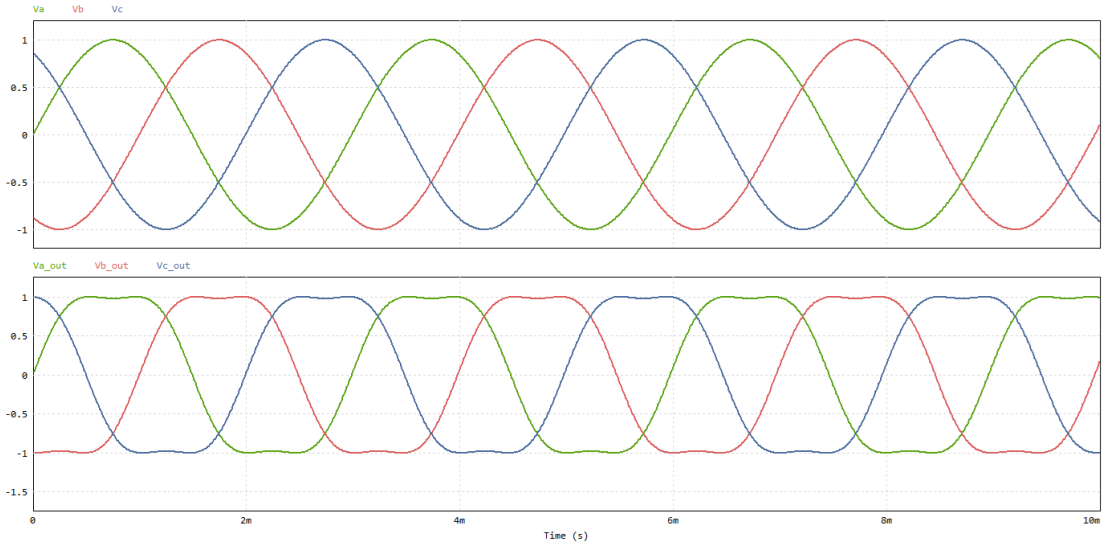


**Figura 3.37: Resultado de inyección de tercer armónico sobre onda trifásica equilibrada**

### 3.7.3. OBTENCIÓN DE RESULTADOS MEDIANTE SIMULACIÓN

La evaluación del sistema de control propuesto se realizará en PSIM, y contemplará diferentes escenarios de funcionamiento, con la observación de parámetros que ayudarán a determinar si el comportamiento del lazo cerrado es correcto en todos los casos. A continuación, se presentan cuatro escenarios de funcionamiento diferentes.

- **Evaluación del control *six-step* en lazo cerrado**

  Para la evaluación del funcionamiento del control *six-step* en lazo cerrado, se ha creado un modelo simplificado con este método de control únicamente. Se han definido distintas rampas de velocidad que se muestran en la figura 3.38 en la parte superior de la gráfica, junto con sus respectivas referencias de velocidad en la parte inferior.

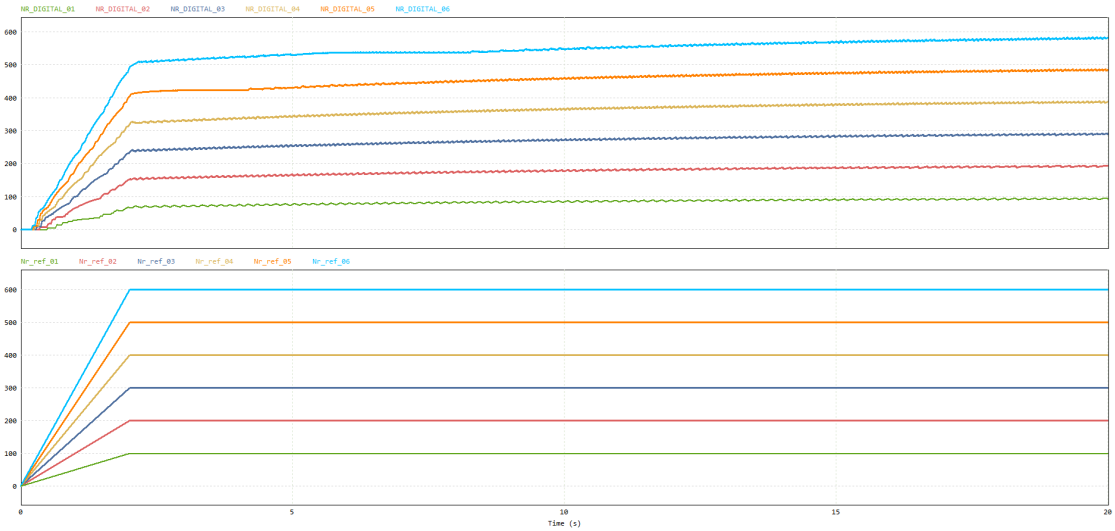


**Figura 3.38: Rampas de velocidad para evaluación del control trapezoidal en lazo cerrado**

  De la figura anterior se puede deducir que el sistema está sobreamortiguado, ya que el comportamiento de la velocidad medida respecto a la referencia es asintótico. No obstante, este comportamiento es deseado ya que el control *six-step* en lazo cerrado está diseñado para funcionar a bajas revoluciones, en el intervalo que transcurre desde el arranque hasta que el control FOC esté completamente sincronizado. Además, una respuesta demasiado violenta del control podría provocar errores en el seguimiento de velocidad, dado que la estimación de velocidad por conmutación de sensores Hall no es demasiado precisa a bajas revoluciones. Eventualmente, todas las curvas de velocidad llegarán a tener un error 0 respecto de su referencia.

- **Simulación con rampa de velocidad de 0 a 2000 RPM**

  Para este escenario, se ha creado una única rampa de velocidad, desde 0 hasta 2000RPM. Esta rampa de velocidad permite la visualización de fenómenos como el cambio de referencia en el multiplexor de velocidad, la zona de funcionamiento, o la sincronización del SOGI-FLL.

  En primer lugar, la figura 3.39 muestra la rampa que se produce en la velocidad de salida del multiplexor, respecto a su referencia. Se observa cómo el ajuste es casi perfecto para el caso del control FOC, aunque el seguimiento también es bueno para el control trapezoidal a bajas RPM. En la parte de abajo, se observa la medida de velocidad mecánica en el eje del rotor. Puede verse que el cambio entre métodos de control se produce en torno a las 300RPM, y que este es casi inapreciable.

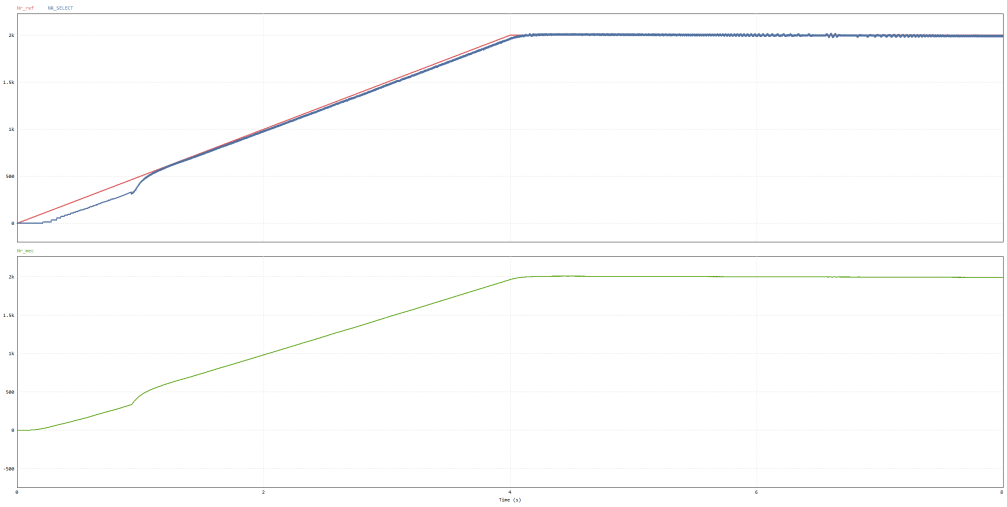

**Figura 3.39: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Velocidad mecánica. Modelo continuo**

En la figura 3.40 se observan las formas de onda de las corrientes en los ejes D y Q de Park, donde se aprecia cómo $i_q$ sigue a la referencia que genera el lazo de velocidad, y también cómo el valor medio de $i_d$ permanece en 0. En las dos componentes se puede apreciar un rizado de alta frecuencia, pero es un efecto esperado tal y como se predijo en el apartado 2.1, por tratarse de un motor con fuerza contraelectromotriz de carácter trapezoidal.

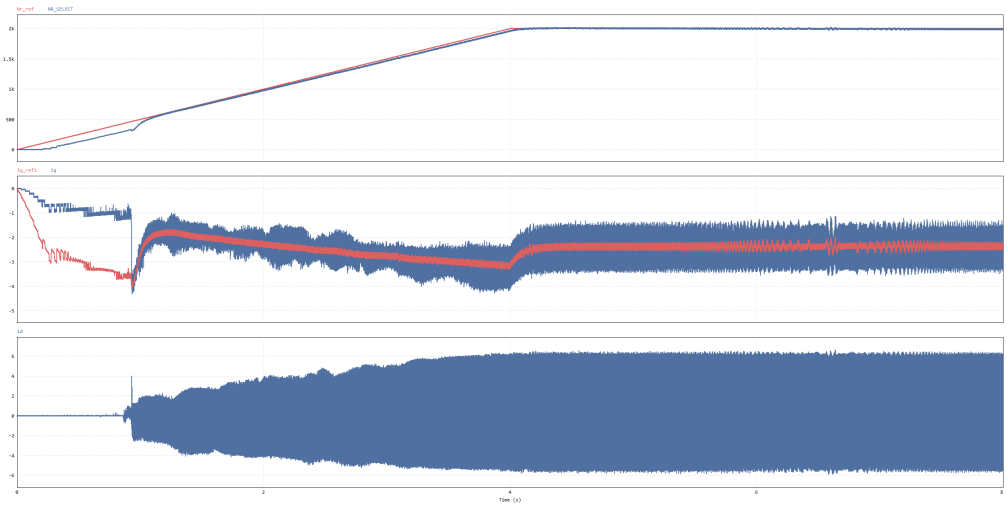

**Figura 3.40: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Corrientes de Clarke. Modelo continuo**

Algo interesante que se puede observar es cómo se produce el cambio entre métodos de control. En la figura 3.41 se puede apreciar cómo la máquina de estados 3.34 conmuta entre los tres posibles escenarios de funcionamiento, y cómo el indicador de zona de funcionamiento (sonda 'ZONA' en PSIM) conmuta de 0 a 1 cuando el control FOC está listo para ser activado.

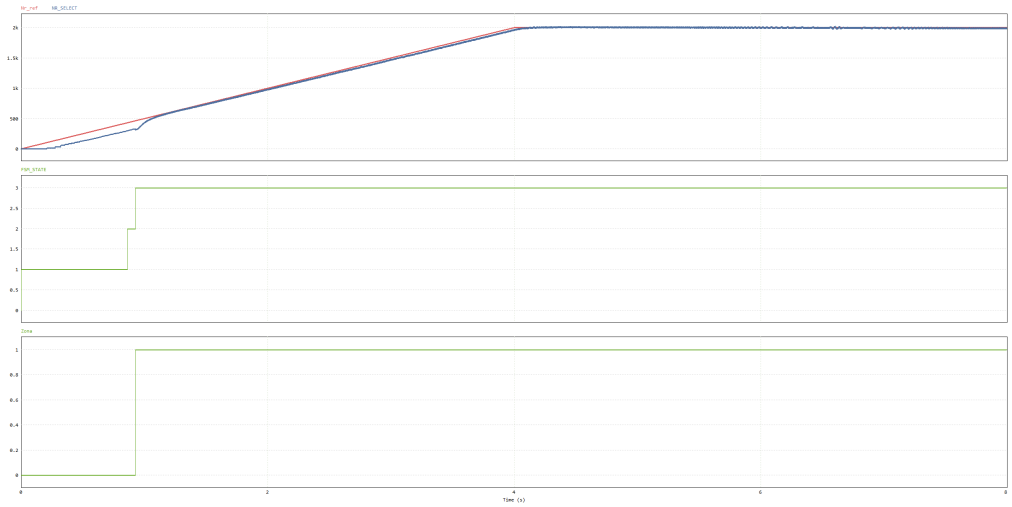

**Figura 3.41: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Estados de FSM y zona de funcionamiento. Modelo continuo**

Si se amplía cualquiera de las zonas de funcionamiento donde el control FOC está activo, se pueden observar algunos parámetros interesantes, como la forma de onda de los ciclos de trabajo $D_a$, $D_b$ y $D_c$. En la figura 3.42 se aprecia que para una velocidad de en torno a 1500 RPM, la forma de onda es trapezoidal, y el valor de pico está en torno a 0.75.

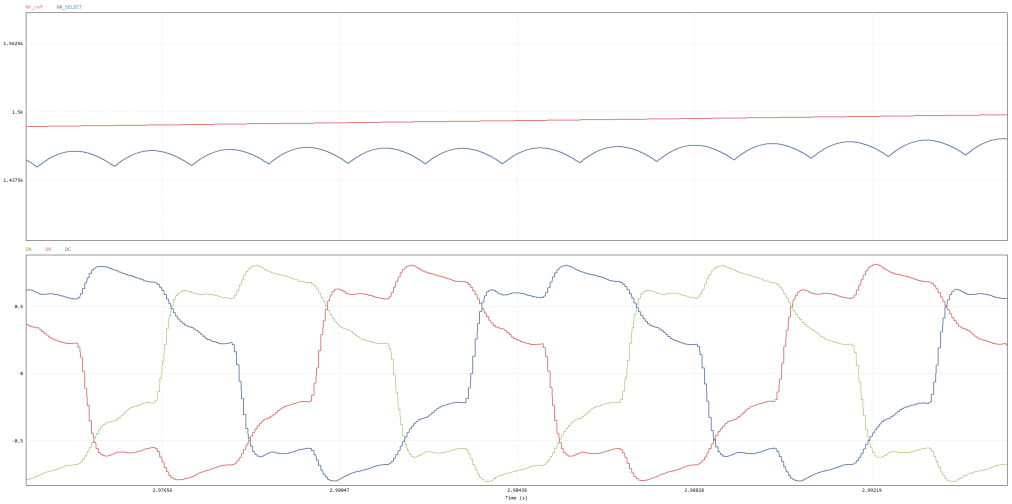

**Figura 3.42: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Ciclos de trabajo ABC. Modelo continuo**

En la figura 3.43 se aprecia, para una velocidad de 2000RPM constante, la forma de onda de la tensión de línea $V_{ab}$, y de la corriente por la fase A, $I_{sa}$. Ambos resultados son esperados, y esto es debido de nuevo a la forma trapezoidal de la FEM.

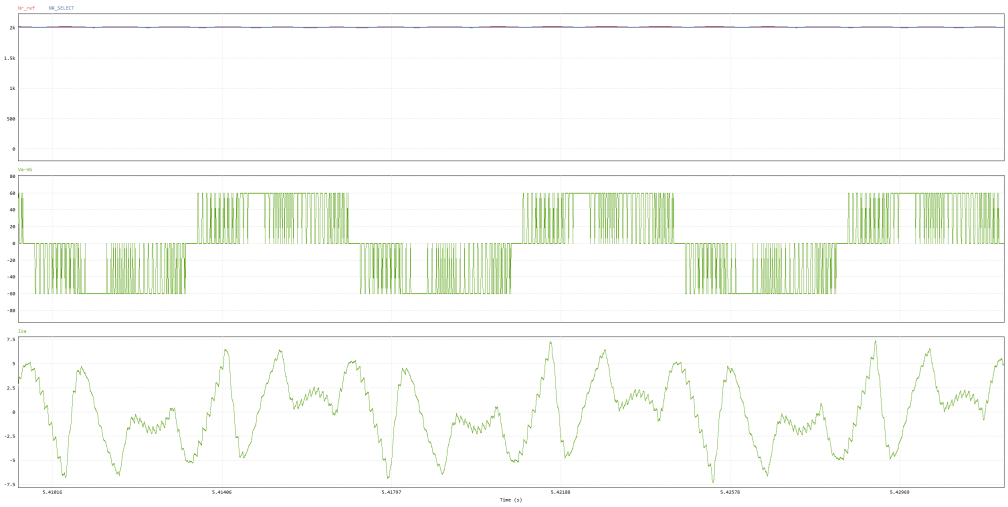

**Figura 3.43: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Tensión y corriente de línea. Modelo continuo**

Ahora se pasará a evaluar el comportamiento del SOGI-FLL para una velocidad constante de 2000RPM. El resultado esperado será la obtención de tres componentes senoidales, $Hall_{asogi}$, $Hall_{bsogi}$ y $Hall_{csogi}$. Estas componentes tendrán una frecuencia que deberá ser la componente fundamental de los sensores de efecto Hall. Esto se evidencia en la figura 3.44, donde se aprecia que las salidas del SOGI-QSG están perfectamente en fase con las entradas de los sensores. Las otras tres salidas que se representan abajo, $Hall_{asogi}$, $Hall_{bsogi}$ y $Hall_{csogi}$, representan las componentes de frecuencia que han sido filtradas en el SOGI-QSG.

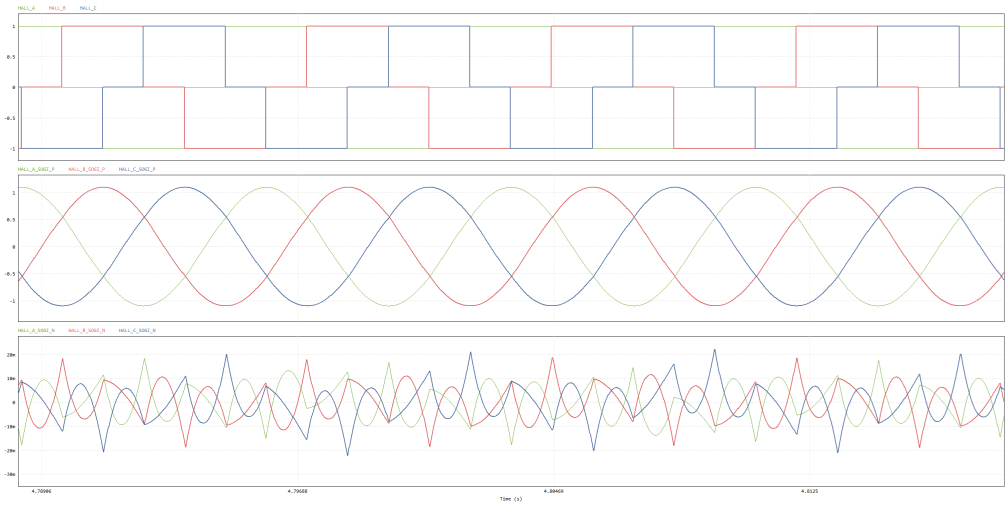

**Figura 3.44: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Sincronización de SOGI-FLL. Modelo continuo**

Para comprobar la cansidad de distorsión presente en las salidas del SOGI, se ha realizado el cálculo de FFT de la figura 3.45. En este se observa que las salidas positivas del SOGI-FLL no presentan ningún tipo de distorsión armónica más allá de la componente fundamental.

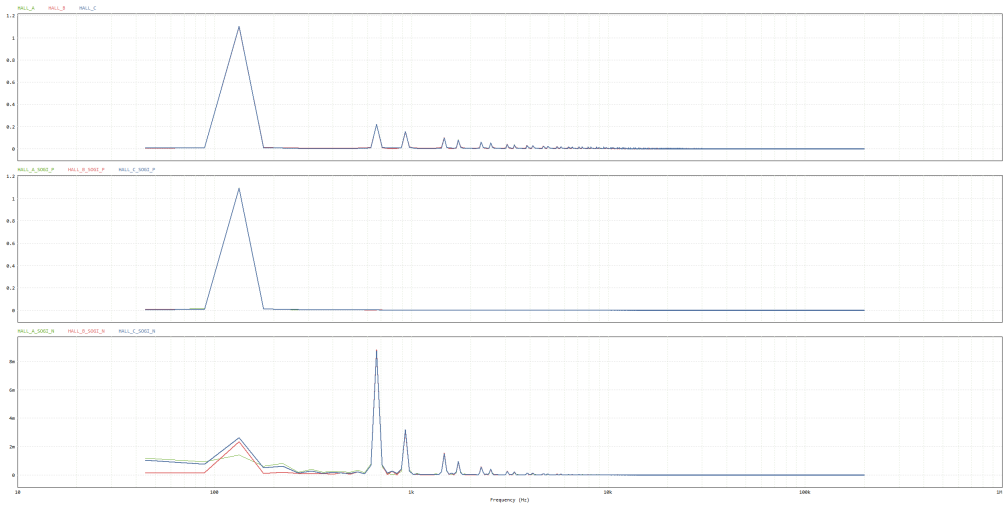

**Figura 3.45: Rampa de velocidad desde 0 RPM hasta 2000 RPM. FFT sobre entradas y salidas de SOGI-FLL. Modelo continuo**

- **Simulación con paso de 2000 RPM a -2000 RPM**

  El siguiente escenario de funcionamiento que se ha planteado es una rampa de velocidad que transita desde 0 a 2000RPM (velocidad máxima del motor sin carga), y después transita a -2000RPM. Este caso es muy interesante, ya que podrá determinarse la robustez del sistema en su paso por 0, cuando las estimaciones de velocidad son algo más imprecisas.

  En la figura 3.46 se aprecia cómo la velocidad se ajusta a la referencia, y cómo la velocidad mecánica apenas tiene distorsiones.

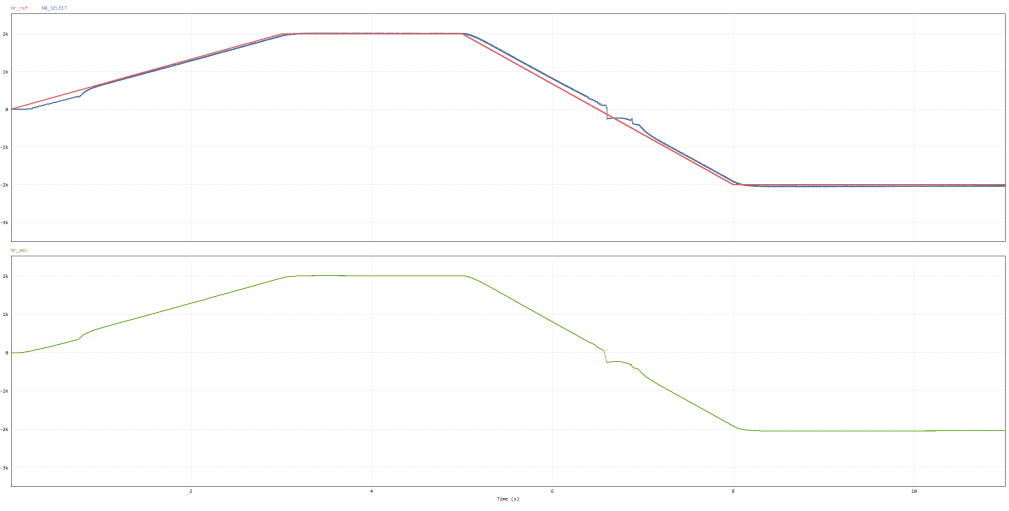

**Figura 3.46: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Medida de velocidad mecánica. Modelo continuo**

Por otro lado, se pueden observar los diferentes estados de la FSM y la zona de funcionamiento. En primer lugar, se transita en la rampa de aceleración hacia el estado de sincronismo del SOGI-FLL y después, al estado de activación del FOC. Como la deceleración es abrupta,

cerca del cruce por 0RPM se transita muy rápido entre los estados 3, 2 y 1, y posteriormente, cuando la velocidad ha cruzado el eje de abscisas, se vuelve a tener la misma secuencia que en el arranque. El pequeño error que se produce en la conmutación entre estados para RPM negativas se debe a lo anterior, aunque esta situación ha sido resuelta sin problema por el control.

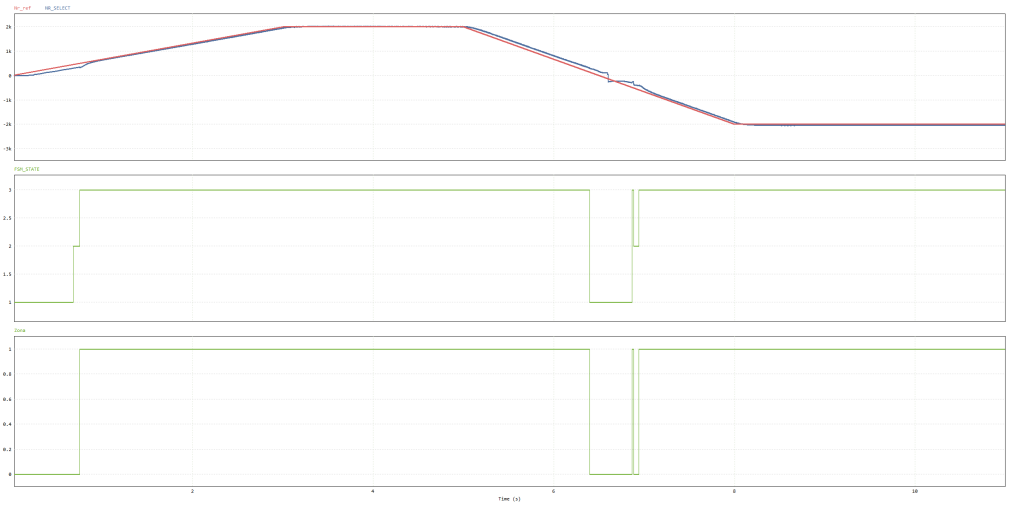

**Figura 3.47: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Estados de FSM y zona de funcionamiento. Modelo continuo**

Otro concepto importante es el funcionamiento del mecanismo de generación del error para el arbitraje de estados. Este toma las dos medidas de velocidad disponibles, 'NR_DIGITAL' y 'NR_SOGI', para calcular el error porcentual en valor absoluto. El motivo del uso del valor absoluto es que el SOGI-FLL sólo se sincronizará frecuencias positivas en la entrada de sensores Hall.

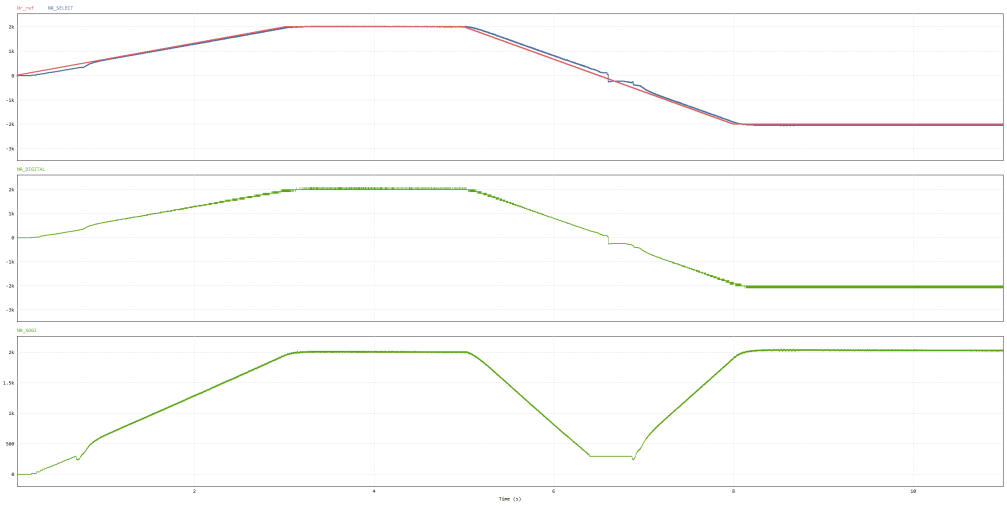

**Figura 3.48: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Medidas de velocidad del sistema. Modelo continuo**

A raíz de la figura anterior, surge la duda de cómo se comportará el SOGI-FLL en el paso por 0RPM. En la figura 3.49 se muestra cómo para un valor de zona de funcionamiento de '1', el SOGI-FLL está sincronizado y cómo para un valor de zona de '0', se encuentra en estado de *reset*.

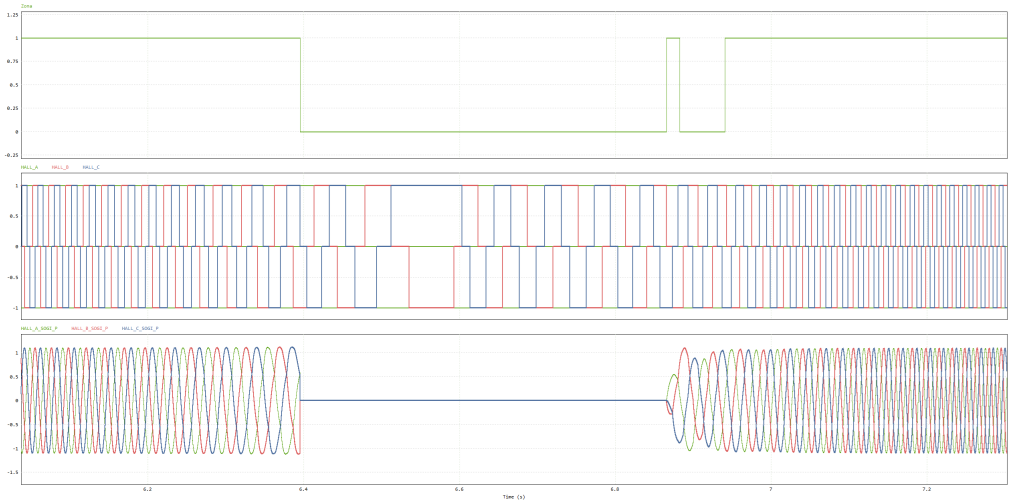

**Figura 3.49: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Sincronización del SOGI-FLL en paso por 0 RPM. Modelo continuo**

Puede observarse que las ondas sinusoidales de salida del SOGI-FLL están en fase con los sensores de efecto Hall solo antes del cruce por 0RPM. Esto es una consecuencia de la inversión de secuencia que sufren los sensores de efecto Hall. Para velocidades positivas, se tendrá secuencia directa $Hall_a$, $Hall_b$, $Hall_c$, pero para velocidades negativas, la secuencia será inversa, $Hall_a$, $Hall_c$, $Hall_b$. Ante esta última situación, una solución directa será siempre pasarle una secuencia directa al SOGI-FLL, invirtiendo $Hall_b$ y $Hall_c$ para velocidades negativas. Esto no afecta en nada a la medida de velocidad resultante en el FLL.

$$\begin{aligned} N_r(+) &\longrightarrow Hall_a, Hall_b, Hall_c \longrightarrow Hall_{asogi}, Hall_{bsogi}, Hall_{csogi} \\ N_r(-) &\longrightarrow Hall_a, Hall_c, Hall_b \longrightarrow Hall_{asogi}, Hall_{bsogi}, Hall_{csogi} \end{aligned} \tag{3.46}$$

En la figura 3.50 se representan las medidas de velocidad 'NR_DIGITAL' y 'NR_SOGI', y la evolución del error proporcional según las etapas de sincronismo.

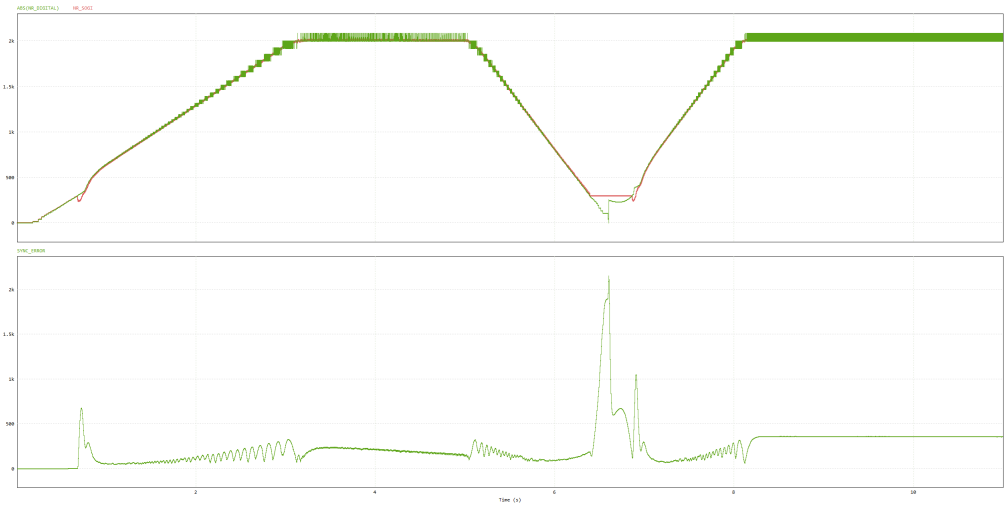

**Figura 3.50: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Error de sincronización para arbitraje del control. Modelo continuo**

- **Simulación para estabilidad para distintos valores de $J_{mot}$**

  Una condición fundamental que se ha impuesto en el diseño del control six-step y FOC, es que el sistema sea estable para cualquier rango de $J_{carga}$ que varíe entre $J_{mot}$ y $1000 \cdot J_{mot}$. En la gráfica 3.51 se muestra cómo para cuatro intervalos de $J_{carga}$ desde $100 \cdot J_{mot}$ hasta $950 \cdot J_{mot}$ el sistema es estable ante una rampa fija de velocidad de 1500RPM.

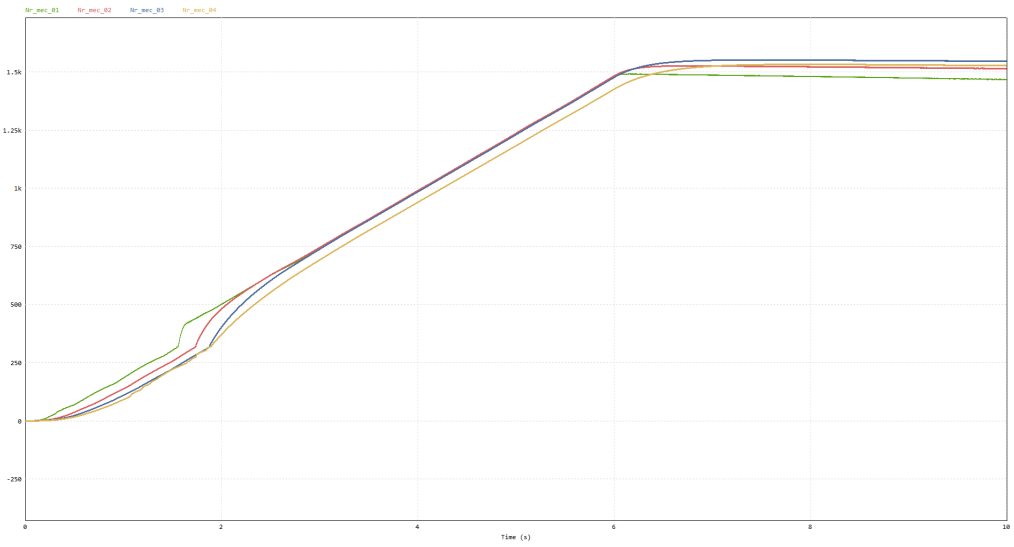

**Figura 3.51: Barrido paramétrico de momento de inercia J para evaluación de estabilidad del control. Modelo continuo**

## 3.8. DISCRETIZACIÓN DEL SISTEMA COMPLETO Y SIMULACIÓN

Con el propósito de acercar el diseño de PSIM a una futura implementación sobre un DSP real, se ha optado por discretizar todos los lazos de control, y también el SOGI-FLL trifásico. Esto ha permitido generar un código en lenguaje C que será de gran valor para la implementación en DSP, y también condensar mucho el diseño del modelo de PSIM.

En este apartado se detallará cómo se ha afrontado la discretización de cada elemento, y se proporcionarán resultados que serán muy cercanos a los que se obtvieron con el primer diseño de PSIM. Es importante destacar que los resultados pueden no ser completamente idénticos, pero eso es debido a que lo que se está planteando en este apartado no es una representación digital idéntica al modelo original, sino una digitalización de dicho modelo que conserve en la mayor medida su funcionalidad original. Añadido a lo anterior, el diagrama de bloques de la figura 3.31 seguirá estando representado de manera exacta por este modelo digital.

### 3.8.1. DISCRETIZACIÓN DE LAZOS DE CONTROL

Para la discretización de los lazos de control se ha utilizado el método de Tustin, o transformación bilineal. Este método consiste en la traslación de los números comlejos que se encuentran en el plano 's' del dominio de Laplace, a la circunferencia unitaria del plano 'z', mediante la equivalencia de la ecuación 3.47.

$$s(z) = \frac{2}{T_s} \cdot \frac{z-1}{z+1} \tag{3.47}$$

Para establecer las frecuencias de muestreo de los bloques 'C' en PSIM, se han utilizado elementos 'ZOH' con una frecuencia de muestreo de 20kHz.

### 3.8.2. DISCRETIZACIÓN DEL SOGI-FLL TRIFÁSICO

La discretización del SOGI-QSG trifásico no es algo trivial, ya que el sistema dependerá de la frecuencia angular del FLL, y el parámetro $K_{qsg}$, que según la expresión 3.41 también depende de la frecuencia del FLL. Por ello, realizar un cálculo de coeficientes directo como podría hacerse en el caso de los lazos de control no es algo viable. En su lugar, se pueden tomar las expresiones del SOGI-QSG en el dominio de Laplace 3.37 y 3.38, y discretizar únicamente los términos 's' de Laplace. Esto resulta en las dos expresiones $D(z)$ (función 3.48) y $Q(z)$ (función 3.55).

$$D(z) = \frac{2 \cdot T_s \cdot K_{qsg} \cdot \omega_{sogi} \cdot (z^2 - 1)}{(T_s^2 \cdot \omega_{sogi}^2 + 2 \cdot K_{qsg} \cdot T_s \cdot \omega_{sogi} + 4) \cdot z^2 + (2 \cdot T_s^2 \cdot \omega_{sogi}^2 - 8) \cdot z + (T_s^2 \cdot \omega_{sogi}^2 - 2 \cdot K_{qsg} \cdot T_s \cdot \omega_{sogi} + 4)} \tag{3.48}$$

De la ecuación 3.48 se pueden extraer los coeficientes C1 a C6 para la componente directa del SOGI-QSG. Estos serán dependientes de $\omega_{sogi}$, y podrán recalcularse de manera dinámica en cada iteración de cálculo del SOGI-QSG que realice el microcontrolador.

$$C1_D = 2 \cdot T_s \cdot k_{sogi} \cdot \omega_{sogi} \tag{3.49}$$

$$C2_D = 0 \tag{3.50}$$

$$C3_D = -2 \cdot T_s \cdot k_{sogi} \cdot \omega_{sogi} \tag{3.51}$$

$$C4_D = {T_s}^2 \cdot {\omega_{sogi}}^2 + 2 \cdot T_s \cdot k_{sogi} \cdot \omega_{sogi} + 4 \tag{3.52}$$

$$C5_D = 2 \cdot {T_s}^2 \cdot {\omega_{sogi}}^2 - 8 \tag{3.53}$$

$$C6_D = {T_s}^2 \cdot {\omega_{sogi}}^2 + 2 \cdot T_s \cdot k_{sogi} \cdot \omega_{sogi} + 4 \tag{3.54}$$

$$Q(z) = \frac{T_s^2 \cdot K_{qsg} \cdot \omega_{sogi}^2 \cdot (z+1)^2}{(T_s^2 \cdot \omega_{sogi}^2 + 2 \cdot K_{qsg} \cdot T_s \cdot \omega_{sogi} + 4) \cdot z^2 + (2 \cdot T_s^2 \cdot \omega_{sogi}^2 - 8) \cdot z + (T_s^2 \cdot \omega_{sogi}^2 - 2 \cdot K_{qsg} \cdot T_s \cdot \omega_{sogi} + 4)} \tag{3.55}$$

De igual manera que con la ecuación directa, se pueden obtener los coeficientes C1 a C6 para la función de transferencia en cuadratura 3.55.

$$C1_Q = {T_s}^2 \cdot k_{sogi} \cdot {\omega_{sogi}}^2 \tag{3.56}$$

$$C2_Q = 2 \cdot {T_s}^2 \cdot k_{sogi} \cdot {\omega_{sogi}}^2 \tag{3.57}$$

$$C3_Q = {T_s}^2 \cdot k_{sogi} \cdot {\omega_{sogi}}^2 \tag{3.58}$$

$$C4_Q = {T_s}^2 \cdot {\omega_{sogi}}^2 + 2 \cdot T_s \cdot k_{sogi} \cdot \omega_{sogi} + 4 \tag{3.59}$$

$$C5_Q = 2 \cdot {T_s}^2 \cdot {\omega_{sogi}}^2 - 8 \tag{3.60}$$

$$C6_Q = {T_s}^2 \cdot {\omega_{sogi}}^2 + 2 \cdot T_s \cdot k_{sogi} \cdot \omega_{sogi} + 4 \tag{3.61}$$

Una vez se ha discretizado el SOGI-QSG, puede pasarse a la discretización del FLL, que se ha afrontado de la misma manera, discretizando únicamente su integrador. Siguiendo el esquema de la figura 3.25 y aplicando la transformación bilineal, el resultado para la función de transferencia del FLL en el dominio de 'Z' es el de la expresión 3.62.

$$FLL(z) = (-\Gamma_{FLL} * ((\epsilon_\alpha \cdot qv'_\alpha) + (\epsilon_\beta \cdot qv'_\beta))) \cdot (\frac{\omega_{sogi} \cdot 0{,}3}{({v_\alpha}'^+)^2 + ({v_\beta}'^+)^2}) \tag{3.62}$$

### 3.8.3. ESQUEMA DISCRETIZADO DESARROLLADO EN PSIM

Para simular el comportamiento de los lazos de control y SOGI-FLL discretizados, se ha desarrollado un esquema independiente en PSIM. Este está compuesto por las mismas secciones que el anterior modelo propuesto, aunque con algunas modificaciones. Las partes principales de este son las siguientes:

- **Lazos de control y transformadas**: La figura 3.52 representa toda la parte relativa al arbitraje general del sistema, cálculo de error proporcional y modulación del mismo modo que se hizo en el apartado 3.7, pero con todo el control trapezoidal y FOC integrado en un único bloque C, que es el bloque 'DIGITAL_CONTROL'.

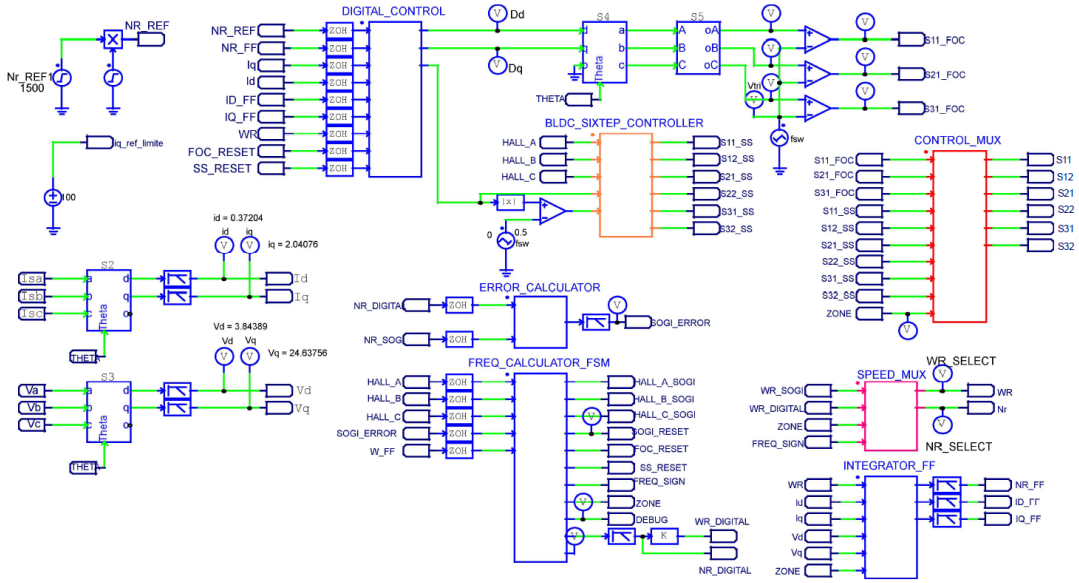


**Figura 3.52: Lazos de control discretizados implementados en PSIM**

- **SOGI-FLL trifásico digital**: La implementación digital del SOGI-FLL simplifica mucho el esquema de PSIM. En la figura 3.53, se han creado tres bloques de C, que se corresponderán con la transformada de Clarke para la generación de las componentes $Hall_\alpha$ y $Hall_\beta$ del SOGI-QSG, el bloque 'SOGI_FLL_DIGITAL' que es el encargado de ejecutar el cálculo de los dos SOGI-QSG y el FLL, y por último el bloque de cálculo de los *feed-forward*.

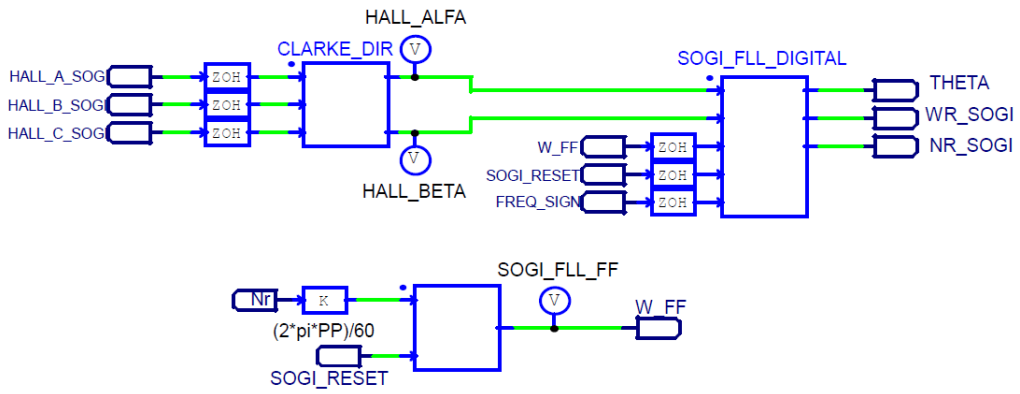


**Figura 3.53: SOGI-FLL discretizado implementado en PSIM**

### 3.8.4. OBTENCIÓN DE RESULTADOS MEDIANTE SIMULACIÓN

La obtención de resultados para el sistema discretizado se afrontará del mismo modo que en el apartado 3.7. Es por ello, que para entender mejor las pequeñas diferencias que puede haber entre los dos modelos de PSIM, se ha decidido respetar la secuencia en la obtención de resultados, con el fin de que la comparación entre los dos modelos sea más directa.

- **Simulación con rampa de velocidad de 0 a 2000 RPM**

  Se ha creado una rampa de velocidad de 0 a 2000RPM, que ha permitido observar en la figura 3.54 el correcto seguimiento de la velocidad por parte de la referencia, y la velocidad mecánica real del sistema.

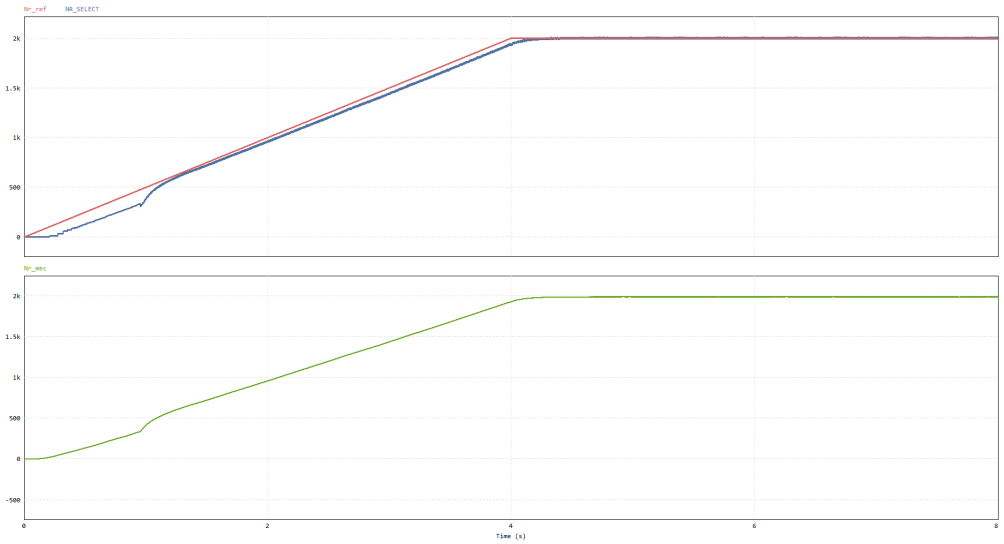

**Figura 3.54: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Velocidad mecánica. Modelo discreto**

Aparte de esto, se puede observar en la gráfica 3.55 que el aspecto de las corrientes en ejes

D y Q son muy similar al que se obtuvo en el apartado anterior, con un rizado superpuesto pero valores medios que se corresponden con la referencia.

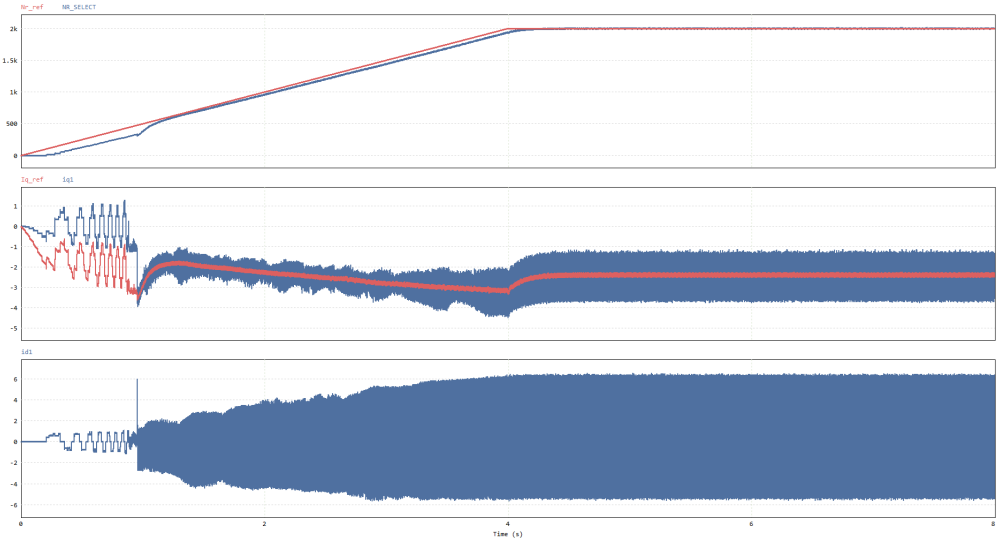


**Figura 3.55: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Corrientes de Clarke. Modelo discreto**

Por otro lado, en la figura 3.56 la conmutación de estados de la FSM sigue funcionando de la misma manera. Se observa cómo esta transita entre los estados 1, 2 y 3 cuando se sincroniza el SOGI-FLL y se libera el control FOC. Además, el indicador de zona de funcionamiento transita a '1' cuando el control FOC se activa.

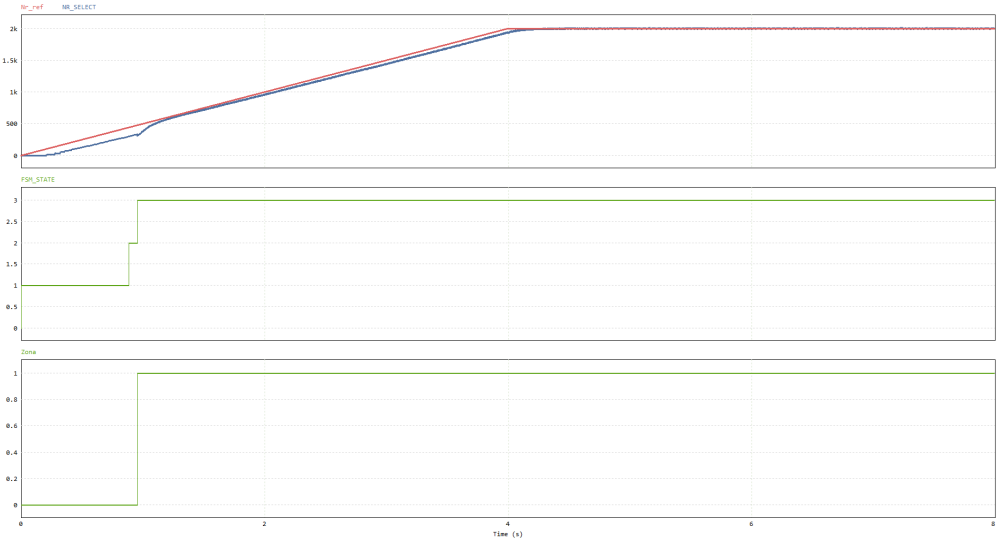


**Figura 3.56: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Estados de FSM y zona de funcionamiento. Modelo discreto**

Los ciclos de trabajo que se pueden observar en la figura 3.57 tienen una forma trapezoidal típica, y presentan para una velocidad de en torno a 1500RPM un valor de pico de en torno

a 0.75. Al no haber presencia de saturación y ser su forma de onda continua, se deduce que el control FOC está correctamente diseñado.

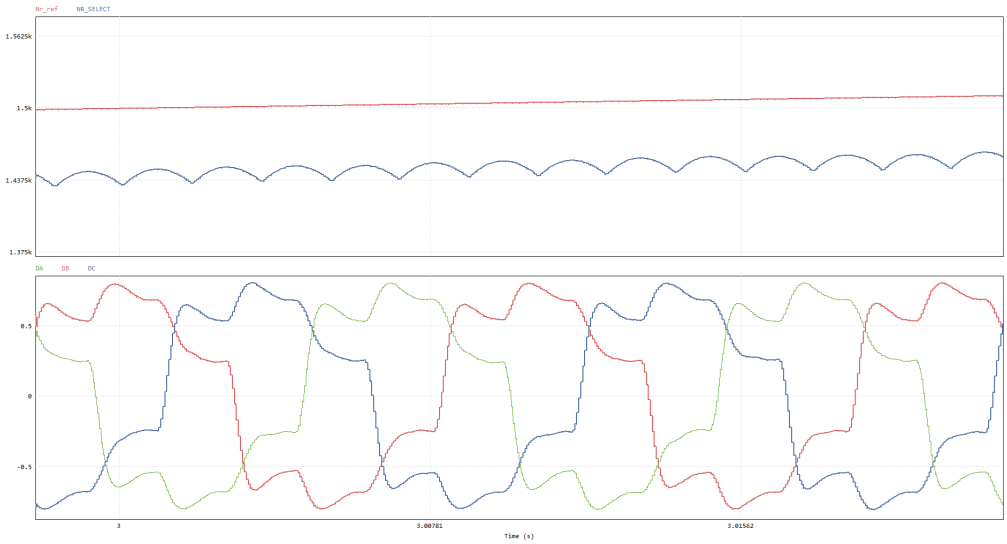


**Figura 3.57: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Ciclos de trabajo ABC. Modelo discreto**

En la figura 3.58 puede observarse que la forma de onda de las tensiones de línea y la corriente de línea son muy similares a las representadas en la figura 3.43. La tensión está formada por un tren de pulsos con forma trapezoidal, y la corriente presenta una forma típica para el control FOC de máquinas BLDC.

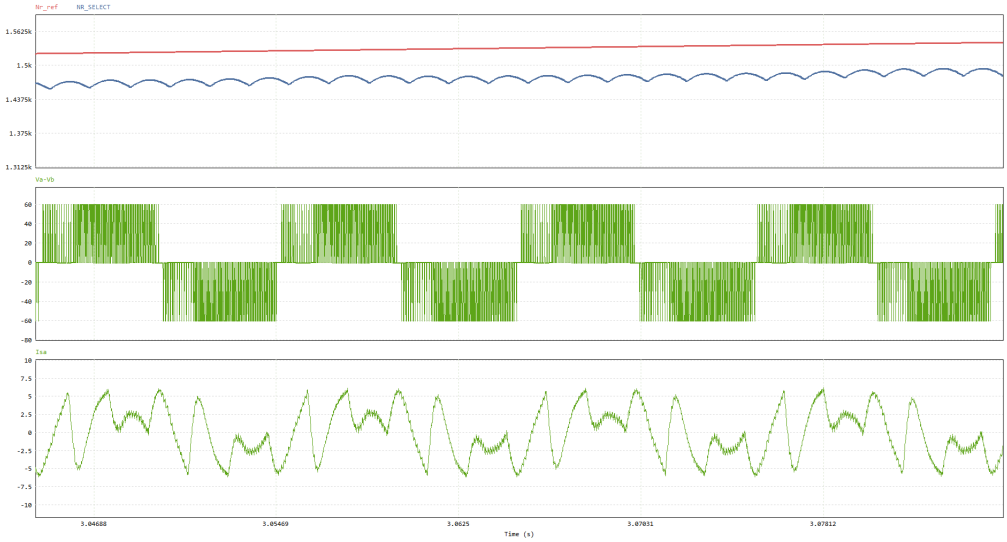


**Figura 3.58: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Tensión y corriente de línea. Modelo discreto**

Una observación de gran importancia para evaluar el rendimiento del SOGI-FLL en esta simulación será en qué manera la medida de fase es correcta y está en fase con la entrada de

sensores de efecto Hall. Esto se muestra en la figura 3.59, donde se observa una medida de fase sin distorsión, y perfectamente sincronizada con los sensores.

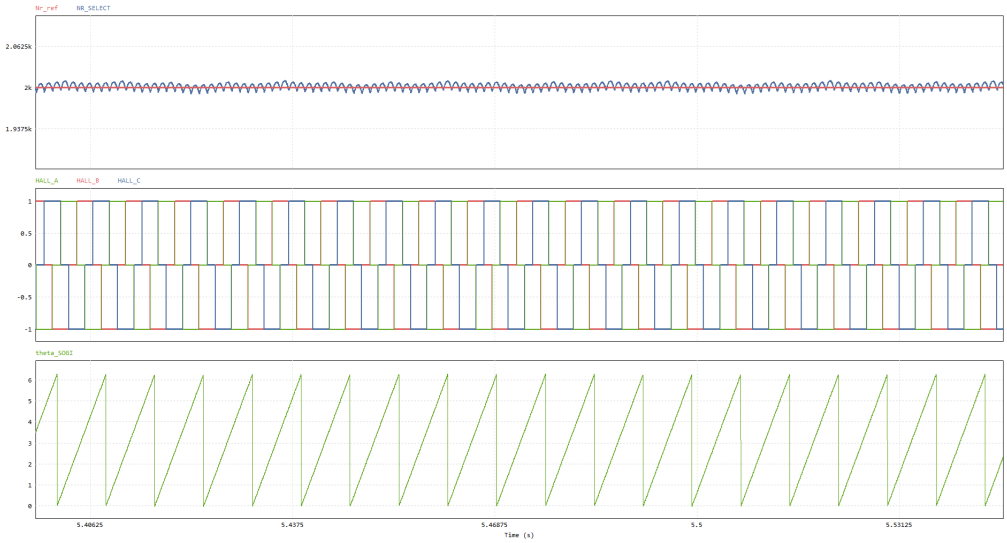


**Figura 3.59: Rampa de velocidad desde 0 RPM hasta 2000 RPM. Fase del sistema respecto a sensores Hall para velocidad constante. Modelo discreto**

- **Simulación con paso de 2000 RPM a -2000 RPM**

  En esta serie de simulaciones, se creará una rampa desde 0RPM hasta 2000RPM, y después se transitará de forma rápida hasta -2000RPM.

  En la figura 3.60 se observa cómo la velocidad de salida del multiplexor de velocidad se adapta de manera perfecta en casi todo el recorrido a la referencia, y cómo la velocidad mecánica real del motor es casi idéntica a la velocidad estimada digitalmente. Además, se observa que las transiciones en el cambio de control son muy suaves.

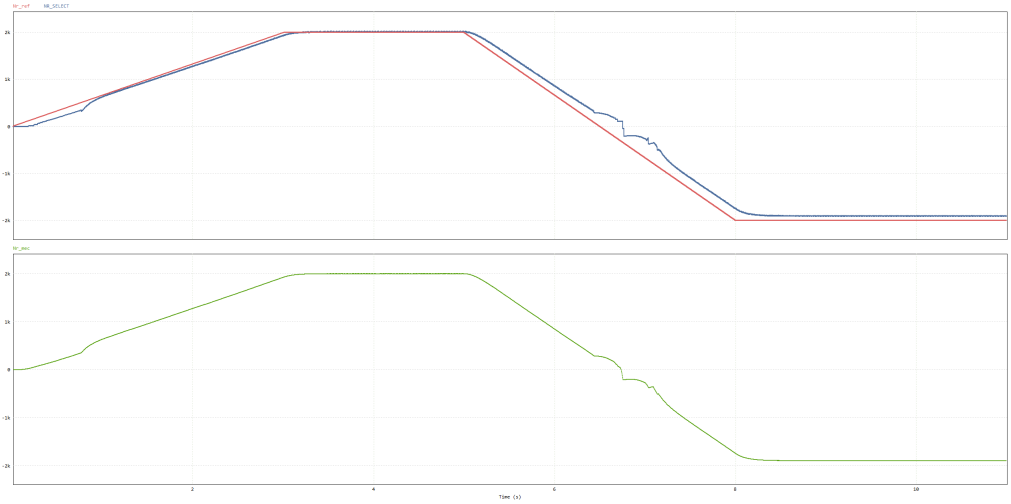


**Figura 3.60: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Medida de velocidad mecánica. Modelo discreto**

Por otro lado, de nuevo los estados de la FSM suceden de manera suave y sin errores, pero en la bajada hacia $-Nr_{max}$ se producen conmutaciones de estados entre el estado 2 (SOGI en sincronización, FOC desactivado) y el estado 3 (SOGI sincronizado, FOC activado). Esto se debe en parte a que en el proceso de sincronización, el SOGI-FLL presenta pequeñas oscilaciones, que varían el error de manera instantánea.

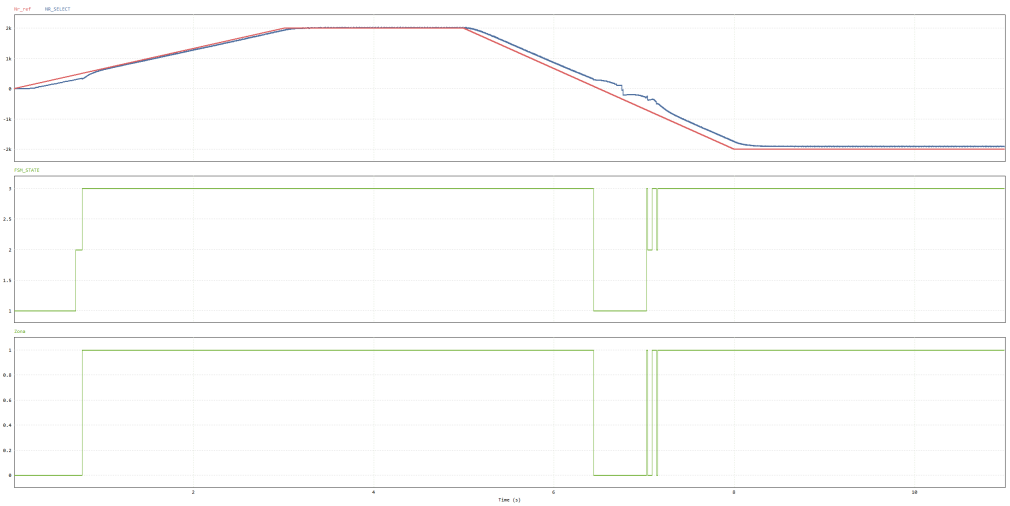

**Figura 3.61: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Estados de FSM y zona de funcionamiento. Modelo discreto**

El fenómeno anterior se ve representado en la figura 3.62, donde se aprecian las velocidades 'NR_DIGITAL' y 'NR_SOGI' en valor absoluto, y se aprecia cómo en la sincronización, el FLL crea un pequeño *undershoot*. No obstante, el control es capaz de resolver esta situación.

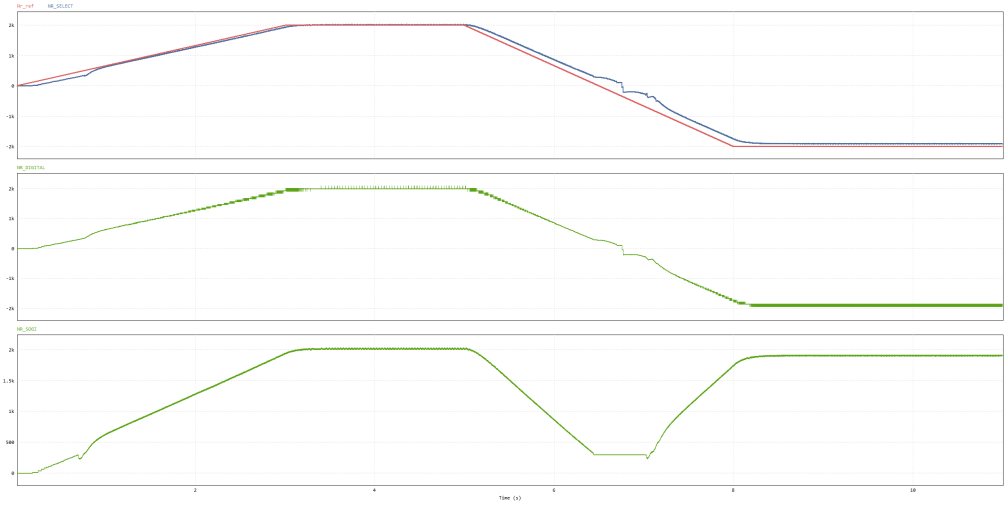

**Figura 3.62: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Medidas de velocidad del sistema. Modelo discreto**

Para examinar el sincronismo del SOGI-FLL y observar qué sucede con el cálculo de fase y velocidad , se ha creado la figura 3.63. En ella se observa el contador de zona de funcio-

namiento, que se corresponde con la activación y desactivación del control FOC, y la fase calculada por el SOGI. Se puede apreciar que para velocidades positivas la fase es ascendente, y para velocidades negativas, es descendente.

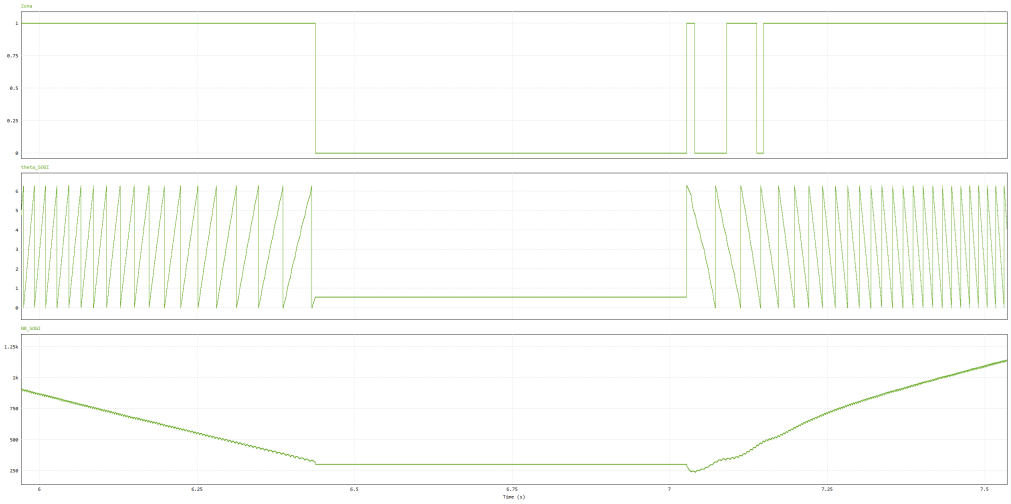

**Figura 3.63: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Sincronización del SOGI-FLL en paso por 0 RPM. Modelo discreto**

Por último, se puede evaluar el error proporcional del sistema en su periodo de sincronización. Este es pequeño durante la primera sincronización en la rampa ascendente, pero es muy grande cuando se produce el paso por 0. Esto se debe a que ante control trapezoidal, la velocidad del FLL será fija al valor de su *feed-forward*. Esto hace que el error aumente para velocidades próximas a 0, tenga su pico máximo en $Nr = 0RPM$, y disminuya de nuevo al transitar hacia velocidad negativa.

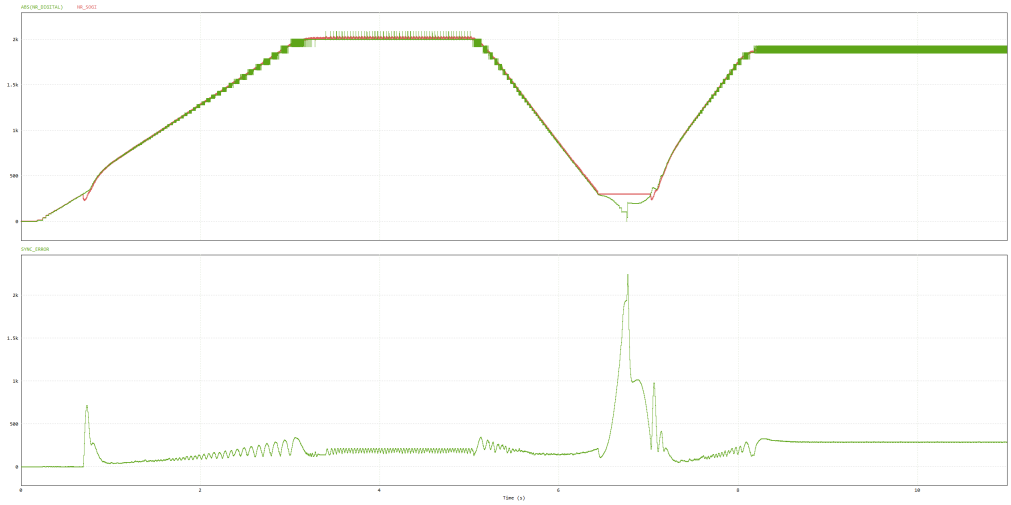

**Figura 3.64: Rampa de velocidad desde 2000 RPM hasta -2000 RPM. Error de sincronización para arbitraje del control. Modelo discreto**

- **Simulación para estabilidad para distintos valores de $J_{mot}$**

  Al igual que en el apartado anterior, se ha querido evaluar la estabilidad del sistema para distintos valores de momento de inercia J. De nuevo, se ha realizado un barrido paramétrico desde $100 \cdot J_{mot}$ hasta $950 \cdot J_{mot}$, para una rampa de velocidad constante de 1500RPM. Observando la figura 3.51 se determina que el sistema es estable para todo el rango de J propuesto.

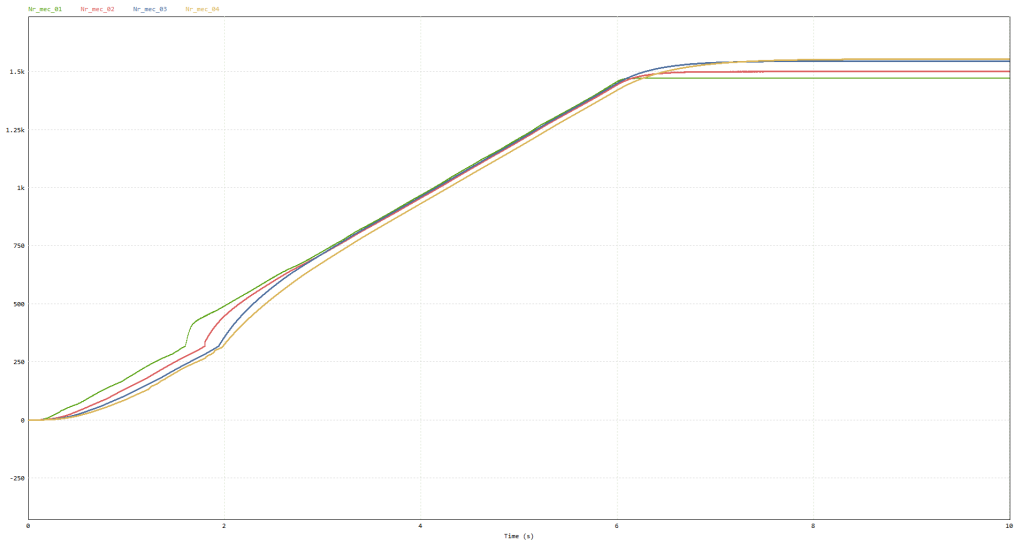


**Figura 3.65: Barrido paramétrico de momento de inercia J para evaluación de estabilidad del control. Modelo discreto**

# Capítulo 4

# IMPLEMENTACIÓN DEL SISTEMA DE CONTROL HÍBRIDO EN DSP

La última fase del diseño del sistema de control híbrido FOC-Trapezoidal con uso de SOGI-FLL, tras la creación de los modelos de simulación en PSIM, será la implementación de este sobre un soporte físico real. Para ello se tendrán que seleccionar componentes como un motor BLDC, un inversor adaptado a los requerimientos eléctricos del motor, y una plataforma DSP sobre la que implementar todos los algoritmos de control.

El objetivo último será probar el funcionamiento de varias de las características que ya se han probado en el entorno de simulación de PSIM, y medir valores reales con uso de instrumentación de laboratorio.

## 4.1. DESCRIPCIÓN DEL ENTORNO HARDWARE UTILIZADO

Para el diseño se van a utilizar componentes electrónicos que conformarán una versión reducida del diseño que se hizo en entorno de simulación. El objetivo no será representar fielmente todo el hardware del modelo, sino crear un prototipo a escala que permita implementar varios de los algoritmos utilizados en simulación.

### 4.1.1. ELECCIÓN DE UN MOTOR BLDC

El motor BLDC elegido es un motor de 24V y 8 polos del fabricante RS 4.1 [21]. Este motor tiene una potencia nominal de 45W, y dadas sus dimensiones reducidas y bajo coste es un candidato perfecto para la realización de pruebas en el laboratorio. Las conexiones que presenta este motor son las tres entradas y/o salidas de tensión trifásica, los terminales de alimentación para los sensores de efecto Hall (entre +5V y +24V), y las salidas de las lecturas de los tres sensores $Hall_a$, $Hall_b$ y $Hall_c$.

Los parámetros extraídos del datasheet de la figura 4.1, son similares a los del motor utilizado para la implementación en PSIM, y de nuevo llama la atención la ausencia del parámetro de enlace de flujo.

**SPECIFICATION**

| | Model | | 2163790 | 2163791 |
|---|---|---|---|---|
| 1 | N° of pole | | 8 | 8 |
| 2 | N° of phase | | 3 | 3 |
| 3 | Rated Voltage | V | 24 | 24 |
| 4 | Rated Speed | rpm | 4500 | 4800 |
| 5 | Rated Torque | Nm | 0,07 | 0,08 |
| 6 | Max Peak Torque | Nm | 0,21 | 0,24 |
| 7 | Torque Constant | Nm/A | 0,037 | 0,035 |
| 8 | Line to Line Resistance | Ω | 1,1 | 0,9 |
| 9 | Line to Line inductance | mH | 1,2 | 1 |
| 10 | No-Load Current | mA | 270 | 220 |
| 11 | Max Peak Current | A | 5,3 | 6,5 |
| 12 | Rated Current | A | 1,89 | 2,29 |
| 13 | Length | mm | 57 | 60 |
| 14 | Rotor Inertia | g-cm$^2$ | 27 | 30 |
| 15 | Weight | Kg | 0,25 | 0,3 |

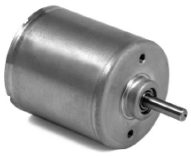

**Figura 4.1: Datos del motor RS-2163790 utilizado para el montaje [21]**

En las imágenes 4.2 y 4.3 se puede observar el motor, junto con una base impresa en 3D que se ha diseñado para dotarlo de mayor estabilidad durante el desarrollo en el laboratorio.

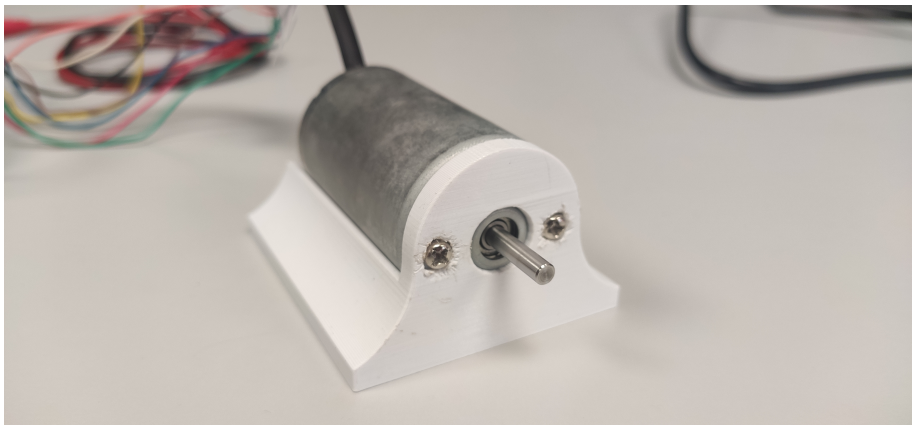

**Figura 4.2: Motor BLDC en perspectiva, con base impresa en 3D**

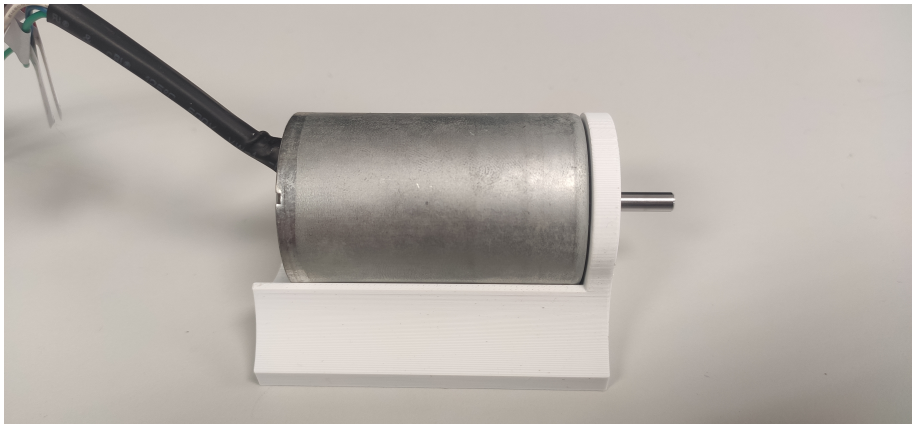

**Figura 4.3: Motor BLDC de perfil, con base impresa en 3D**

### 4.1.2. ELECCIÓN DE UN INVERSOR COMERCIAL DE ACUERDO A ESPECIFICACIONES

Habiendo planteado cuál es el motor que se va a utilizar, hará falta elegir un inversor que se adapte a los requisitos eléctricos de este. Para esta aplicación específica, se ha optado por elegir el inversor DRV8329AEVM del fabricante Texas Instruments [22]. En las figuras 4.4 y 4.5 se muestran dos vistas de la PCB del inversor.

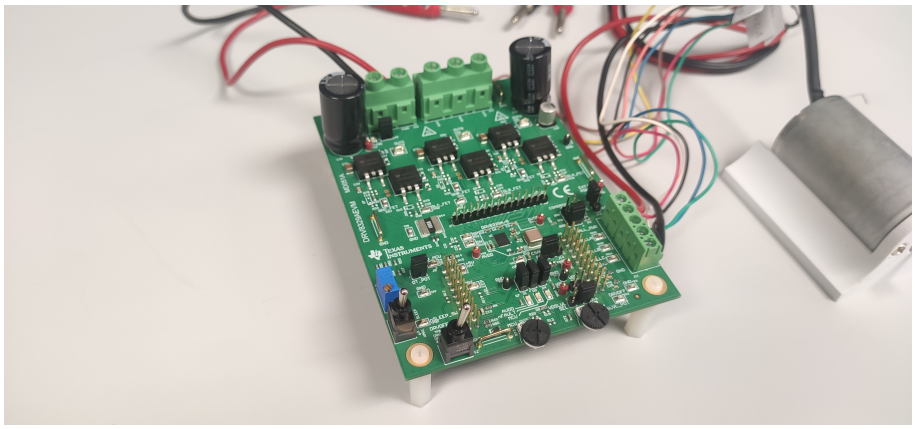

**Figura 4.4: Inversor DRV8329AEVM de perfil**

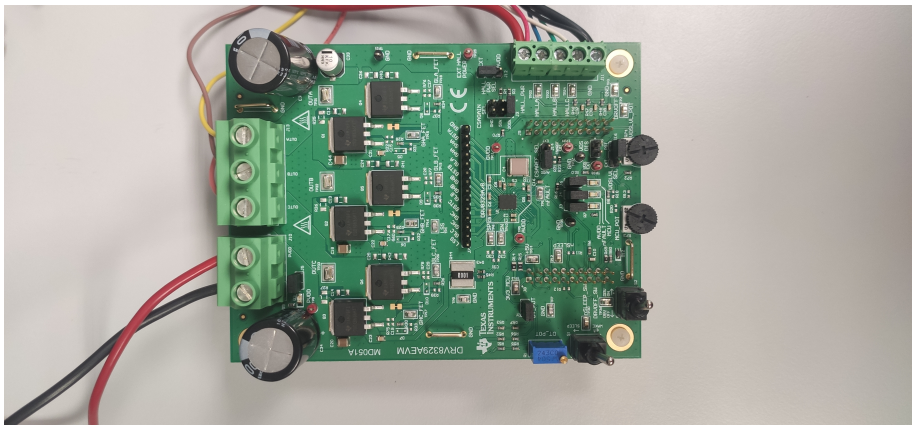

**Figura 4.5: Inversor DRV8329AEVM de planta**

Algunas de las características más relevantes sobre este inversor son las siguientes:

- Entrada de tensión DC variable, desde 4.5V hasta 60V.
- Salida de tensión trifásica.
- Indicadores LED para tensión.
- Entradas para lectura de sensores de efecto Hall, $Hall_A$, $Hall_B$ y $Hall_C$.

Añadido a todo lo anterior, utilizar un ecosistema de Texas Instruments facilita mucho el desarrollo de cualquier implementación, no solo por contar con su propio IDE[I], sino también por contar con toda una gama de DSP del mismo fabricante, que agilizan mucho todas las labores de diseño de software. Además, también hay gran cantidad de bibliotecas y soluciones software proporcionadas directamente por Texas Instruments, o por la extensa comunidad de internet.

Un punto a tener en cuenta a la hora de utilizar el inversor para el diseño del control sobre la plataforma DSP será qué lecturas de parámetros estarán disponibles. En este caso, las lecturas disponibles mediante ADC son las siguientes:

- Tensión continua de entrada PVDD.
- Tensiones de fase Va, Vb y Vc.
- Corriente en el bus de continua.
- Valores de los sensores de efecto Hall (en formato unipolar).

En la figura 4.6 se puede ver el puerto J13 sobre el que se realiza la conexión de las tres fases del motor BLDC. Por otro lado, en la figura 4.7 se muestra el puerto J11 del inversor, donde se encuentran conectados los sensores de efecto Hall. En este caso, también ha sido necesario añadir una alimentación externa de 5V dado que la tensión de entrada por defecto en los sensores es de 3.3V, y se determinó que esta no era suficiente para asegurar que funcionasen correctamente.

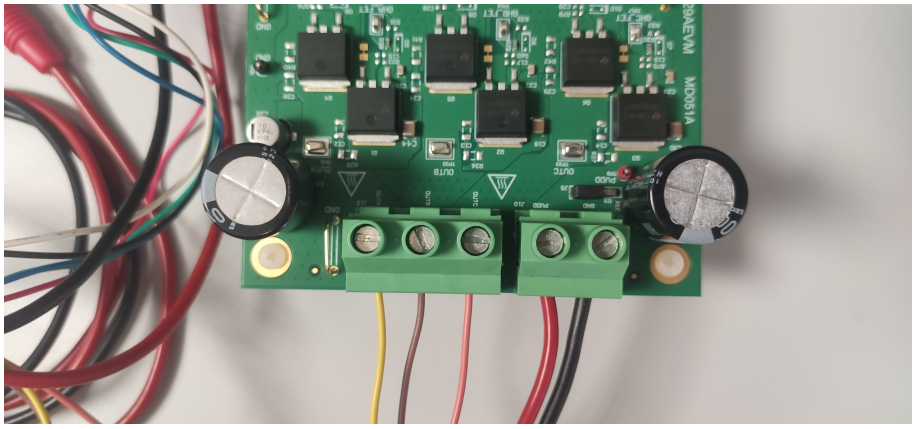

**Figura 4.6: Salida de tensión trifásica en DRV8329AEVM**

Una vez se han planteado las características principales del inversor, surge un inconveniente, y este es la ausencia de lecturas incorporadas de las corrientes de línea. Esto será un claro inconveniente para la implementación del lazo de control FOC, ya que este consta de un lazo de corriente y otro de velocidad. Más adelante se estudiarán las medidas y simplificaciones que se han tenido que asumir por este motivo.

[I]Entorno de Desarrollo Integrado

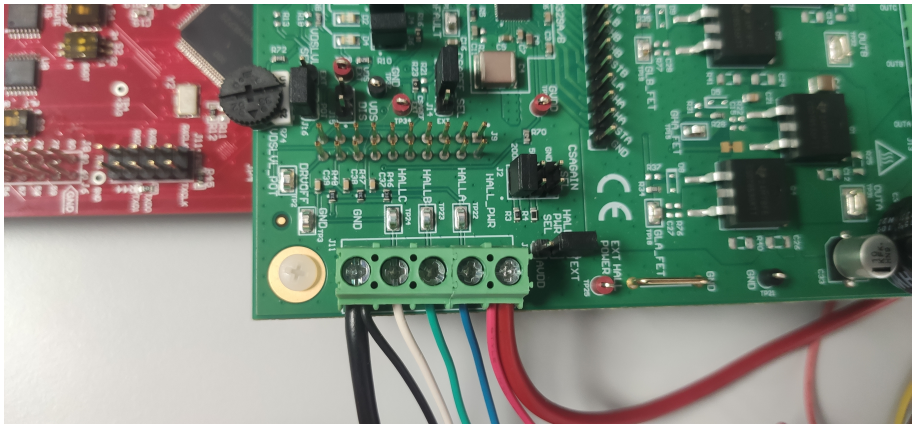

**Figura 4.7: Conexiones para sensores Hall en DRV8329AEVM**

### 4.1.3. USO DE UNA PLATAFORMA DSP ESPECÍFICA PARA LA IMPLEMENTACIÓN DEL CONTROL

La plataforma de desarrollo DSP integrada que se ha seleccionado es la C2000 LAUNCHXL-F280049C [23]. Esta es una placa de desarrollo que se construye en torno al MCU[II] TMS320F280049C, y cuenta con múltiples periféricos como interfaces USB, depuradores XDS110 on-board, bus CAN, bus FSI, ADCs y DACs.

En las figuras 4.8 y 4.9 se muestran diferentes vistas del LAUNCHXL-F280049C.

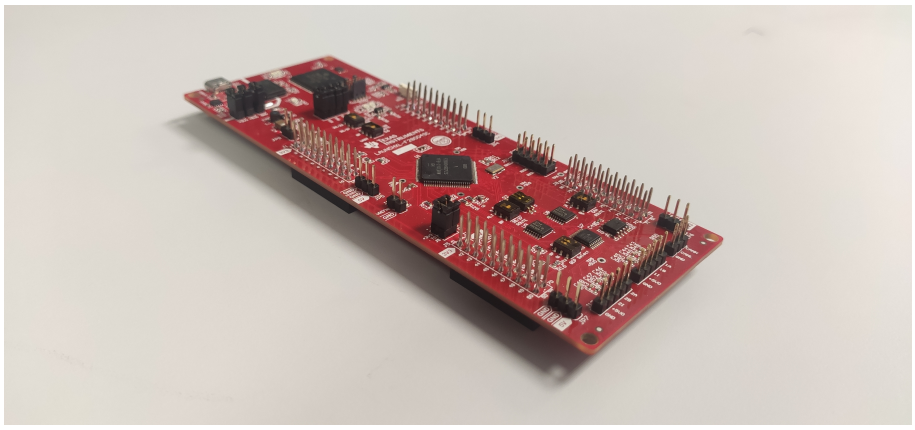

**Figura 4.8: LAUNCHXL- F280049C de perfil**

La conexión entre la placa de desarrollo LAUNCHXL-F280049C y el inversor DRV8329AEVM es muy sencilla, y se hace a través de los conectores integrados en ambas PCB. El resultado se muestra en la figura 4.10 y es una conexión sólida que agiliza mucho el proceso de captura de datos, escritura de ciclos de trabajo, y manejo de los pines GPIO desde el inversor.

[II] Micro Controller Unit

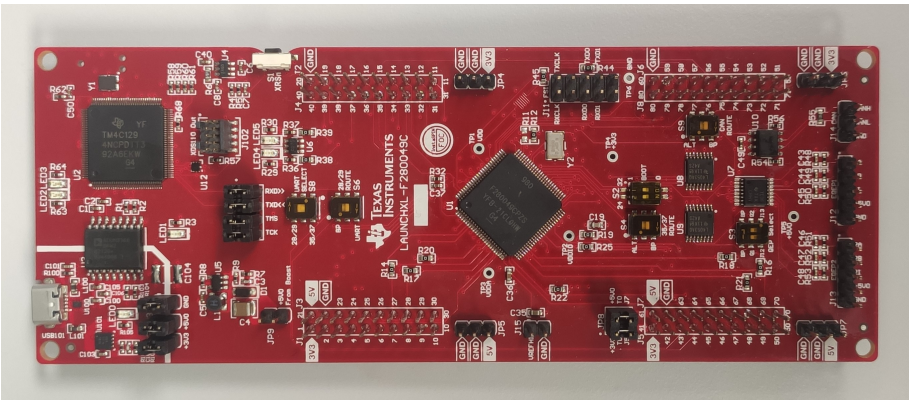


**Figura 4.9: LAUNCHXL- F280049C de planta**

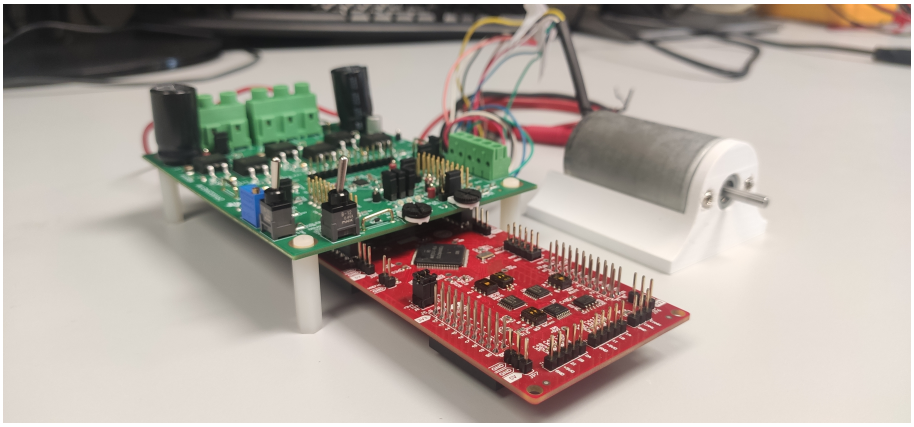

**Figura 4.10: LAUNCHPAD y DRV8329AEVM conectados**

### 4.1.4. DESARROLLO DE UN MODELO PARA LA IMPLEMENTACIÓN

Para la puesta en marcha del prototipo, se ha creado un modelo a escala del sistema planteado en simulación, aunque con algunas limitaciones que provienen del hardware disponible. Si bien el objetivo inicial era la implementación sobre hardware del mismo sistema del modelo de simulación, la ausencia de lecturas de las corrientes de fase ha dificultado este proceso.

Debido a lo anterior, se ha creado un esquema de control alternativo puramente trapezoidal y en lazo abierto, que permitirá la variación de velocidad del motor, en ambos sentidos.

El foco del desarrollo será la correcta implementación del algoritmo SOGI-FLL, que se utilizará del mismo modo que en el modelo, para estimar fase y frecuencia angular del motor.

En la figura 4.11 se puede encontrar el diagrama de bloques completo del diseño que se implementará en el DSP. En él, se muestra un primer bloque de modulación trapezoidal, que servirá para generar las tensiones de disparo de los transistores del inversor. En este caso, del motor BLDC se extraen los datos de los sensores de efecto Hall, que servirán tanto para calcular la velocidad en base al número de conmutaciones de estos por unidad de tiempo, como para sincronizar el conjunto del SOGI-FLL. El multiplexor para la selección de la velocidad se encarga de ofrecer a su salida una medida de velocidad, que será la aproximación digital para velocidades bajas, y cambiará a

la del FLL cuando este esté sincronizado. Por último, se observa la incorporación de dos DAC que servirán para observar mediante un osciloscopio los valores de fase y velocidad del motor, y facilitarán la obtención de conclusiones sobre los resultados experimentales.

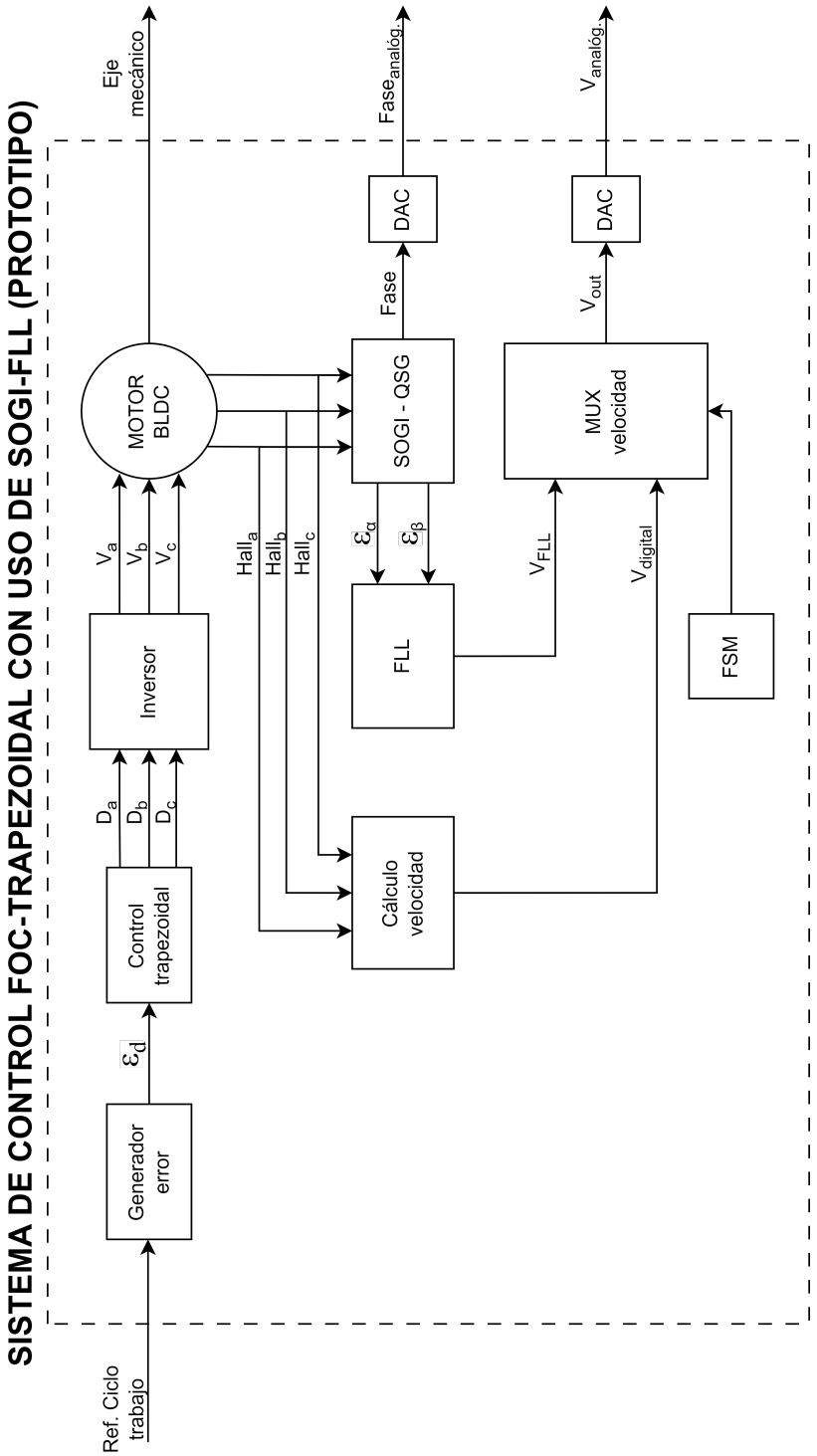


**Figura 4.11: Diagrama de bloques del modelo desarrollado para la implementación (original)**

## 4.2. DESARROLLO DEL CÓDIGO

Para el desarrollo del código, ha sido necesario instalar los plugins apropiados dentro del software **CCS**[III]. El primero de estos es 'C2000Ware_Digital_Power', que contiene soluciones integradas para el desarrollo de convertidores de potencia aplicados a automoción, generación solar, o telecomunicaciones, o la implementación de algoritmos de control. El otro complemento que se ha utilizado es 'C2000Ware_Motor_Control', que es un conjunto de infraestructuras, herramientas y documentación orientadas a minimizar el esfuerzo de diseño de algoritmos de control para motores. Estos dos paquetes de software serán especialmente útiles cuando se haga el desarrollo sobre placas de evaluación de Texas Instruments.

Se han creado dos archivos principales:

- **DRV8329A_BLDC_CONTROL.c**: Programa principal del motor, incluído en las bibliotecas del add-on 'C2000Ware_MotorControl'. Este programa se encarga de controlar las lecturas de los ADC, crear un ciclo de trabajo para los transistores de cada rama, y ejecutar el algoritmo SOFI-FLL. El código completo de este archivo se puede encontrar en el anexo B.1.
- **MOTOR_CONTROL.c**: Biblioteca elaborada para recoger la implementación del cálculo de velocidad a partir de los sensores Hall, del algoritmo SOGI-FLL y la máquina de estados general del prototipo. Además se incluyen otras funcionalidades como transformadas de Clarke directas e inversas, Este código se puede encontrar en el anexo B.2.

### 4.2.1. FUNCIONES DESARROLLADAS PARA CONTROL DEL MOTOR

Con el fin de implementar el algoritmo SOGI-FLL y el cálculo de velocidad mediante conmutaciones de los sensores Hall, de manera similar a como se hizo en PSIM, se ha creado la biblioteca **MOTOR_CONTROL.c** con numerosas funciones similares a las que se utilizaron en el modelo del motor discretizado en PSIM. Se podría decir que a efectos prácticos esta biblioteca actúa de manera similar al bloque de control central de los esquemas de PSIM.

El archivo está estructurado en tres secciones principales. La primera contiene la definición de estructuras, que agilizará todo el proceso de realización de transformadas de Clarke directas e inversas, y la generación de estados en la máquina de estados. La segunda parte es la declaración de todas las variables y constantes necesarias para el funcionamiento del motor, y la tercera son las funciones y prototipos que se explicarán a continuación.

- **motorFSM**: La función 'motorFSM' es la que crea la sucesión de estados durante el funcionamiento del programa. Esta es esencial para arbitrar elementos como el método de cálculo de velocidad más adecuado, el método de control más apropiado según la región de funcionamiento, o el encendido y apagado del SOGI-FLL. En la figura 4.12 se observa cómo para sincronizar el SOGI-FLL harán falta más de 1000RPM, pero una vez este está sincronizado, se podrá mantener funcionando correctamente hasta por debajo de las 800RPM.

[III] Code Composer Studio, IDE de Texas Instruments

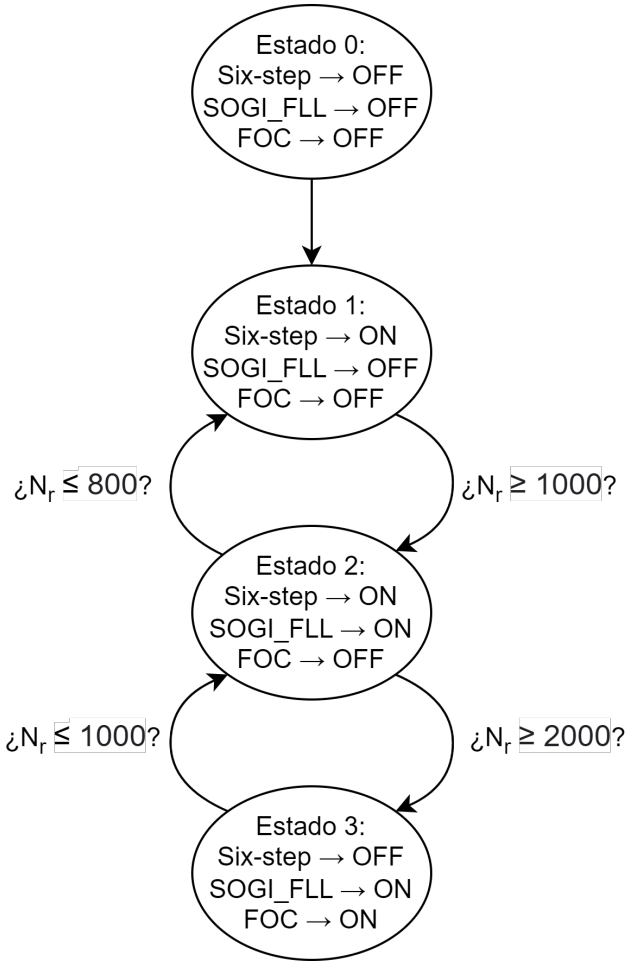


**Figura 4.12: FSM del arbitraje en el control del prototipo (original)**

En este caso, aunque el método de control es puramente trapezoidal, se ha introducido un esquema de variables que representan la conmutación entre modulación trapezoidal y FOC, igual que se hizo con el modelo de simulación.

- **sogi_ff_calc**: El cálculo del feed-forward para el FLL será de vital importancia para asegurar que este se sincroniza con las señales de entrada lo más rápido posible. Existen dos posibilidades a la hora de diseñar el feed-forward para el FLL. La primera es fijar un valor estático de velocidad, lo que hará que el SOGI-FLL siempre comience a sincronizarse con la entrada desde la misma velocidad, y la segunda es hacer que este valor de feed-forward dependa de la velocidad angular estimada mediante conmutaciones.

  Durante la experimentación, se ha observado que este segundo método es el que mejor resultado ofrece, ya que esto permite modificar libremente los umbrales de conmutación entre estados en la FSM, sin alterar el comportamiento del SOGI-FLL. En el código de la función, si el SOGI-FLL está en estado de reset, se asigna a la variable '$w_{ff}$' el valor de la velocidad medida por conmutaciones, y cuando se sale del reset, el valor de '$w_{ff}$' será fijado al último valor de velocidad antes del reset.

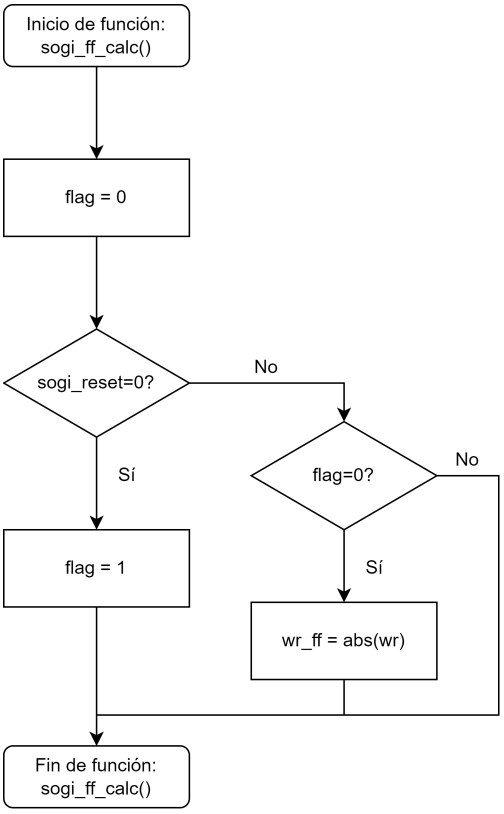


**Figura 4.13: Diagrama de flujo de función para cálculo de feed-forward en SOGI-FLL (original)**

- **sogi_fll_step**: Para implementar el SOGI-FLL discreto, primero habrá que recurrir a las ecuaciones 3.48 y 3.55, donde se muestra la caracterización de las funciones de transferencia de las ramas Alfa y Beta, en dominio de Z. Si se transforman estas funciones para estar representadas por potencias negativas de Z, se puede hacer una asociación directa entre un cierto orden negativo en el exponente, y la dimensión del buffer que tendrán los búferes de los vectores de entrada y salida en lenguaje C. Por ejemplo, para las funciones de transferencia del SOGI, se tiene un orden de exponente -3 en el numerador y denominador, lo que quiere decir que la entrada y salida estarán compuestos por dos vectores de tres dimensiones, que almacenarán los valores actuales, y los valores de hasta dos ciclos de ejecución de antigüedad. Como resultado, se han creado un total de cuatro búferes de tres dimensiones para las entradas y salidas de las componentes Alfa y Beta del SOGI, y dos búferes de dos dimensiones para la entrada y salida del FLL.

  Tras crear estas variables, el primer paso es inicializarlas a 0, y esto se hará en los primeros instantes de ejecución, cuando el conjunto SOGI-FLL está en estado de reset. Tras salir de este estado, se asignarán los valores de $xd\_alfa[0]$, $xq\_alfa[0]$, $xd\_beta[0]$, $xq\_beta[0]$, y $x\_fll[0]$, y tras calcular las salidas para el instante actual, habrá que actualizar todos los búferes retrasándolos una muestra. Esto es, para los vectores del FLL, asignar $y\_fll[1] = y\_fll[0]$, y $x\_fll[1] = x\_fll[0]$.

  Tras estos cálculos, se puede obtener la fase del sistema mediante operaciones sencillas de las salidas de los SOGI, y la velocidad angular en rad/s será directamente la salida $y\_fll[0]$.

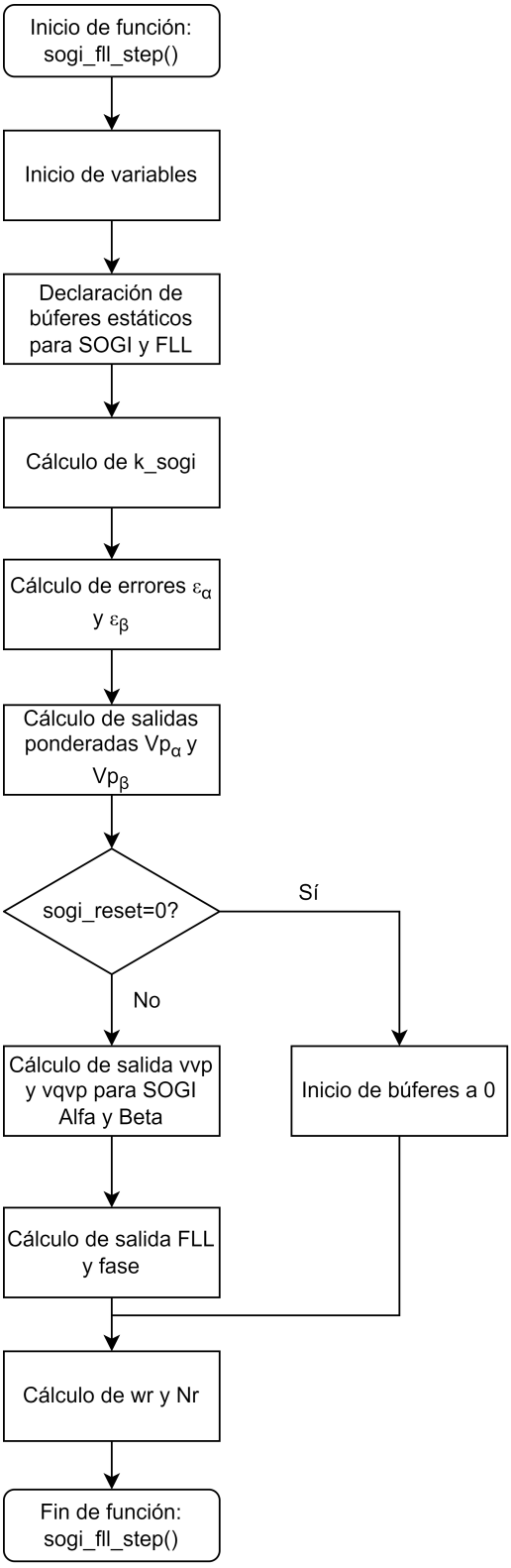


**Figura 4.14: Diagrama de flujo de función para cálculo de salidas del SOGI-FLL (original)**

- **func_speed_calc**: El cálculo de velocidad a partir de las conmutaciones de los sensores de efecto Hall es sencillo, y consiste en calcular la cantidad de veces que se ha incrementado el contador 'nsamples' antes de detectar una conmutación en cualquiera de los tres sensores. Como se ha programado una frecuencia de 20kHz para el temporizador 'timer0', se puede estimar que el periodo entre ejecuciones sucesivas de esta función es de $5 \cdot 10^{-5}$. La ecuación 4.1 muestra cómo realizar el cálculo de velocidad angular en RPM, y estará multiplicada por 1/6 dado que hay un total de seis posibilidades de conmutación de los sensores en el motor.

$$\begin{aligned} T_s &= \frac{1}{f_s} = 5 \cdot 10^{-5} \\ Nr &= \frac{60}{T_s \cdot nsamples \cdot 6 \cdot PP} \end{aligned} \tag{4.1}$$

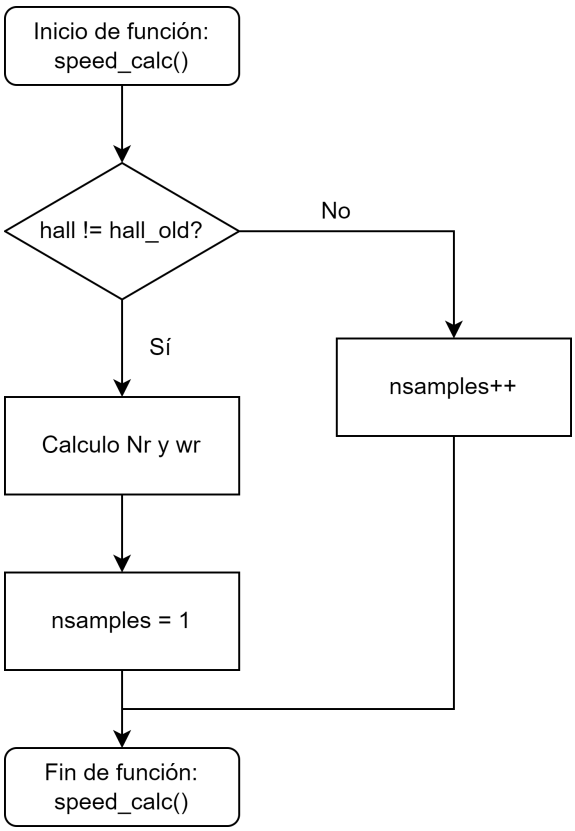


**Figura 4.15: Diagrama de flujo de función para cálculo de velocidad a partir de conmutaciones Hall (original)**

### 4.2.2. FLUJO DEL PROGRAMA PRINCIPAL

En el archivo principal del control, se encuentra todo lo relativo a la configuración de los ciclos de trabajo en el control de los transistores del inversor, protecciones contra sobrecorriente y sobretensión, indicadores de fallos y las rutinas de transformadas, cálculo digital de velocidad, y SOGI-FLL.

En el diagrama de flujo de la figura 4.16 se puede observar cómo transcurre un ciclo completo del programa, desde la declaración de variables hasta el bucle principal de control. En primer lugar, se declaran todas las variables necesarias, se realiza la configuración de registros de periféricos como los ADC, DAC y Timer 0, y se inician sus correspondientes rutinas de interrupción. Tras esto, si se ha habilitado la salida del inversor, se procede a leer los conversores analógico-digital correspondientes a las medidas de tensiones de fase, corriente del bus de continua, y ciclo de trabajo, que en este caso se programará utilizando el potenciómetro 'R73' incorporado en el DRV8329AEVM. Si alguno de los parámetros de tensión estuviese fuera del rango definido, la salida del inversor se deshabilitaría, para no producir daños a los componentes electrónicos de este.

Llegados a este punto deberemos hacer un paréntesis y preguntarnos si en algún momento ha saltado ya la interrupción que genera el Timer 0 cuando llega al valor final de su cuenta. Si este fuese el caso, se entraría a su rutina de interrupción, donde se invocan la función '**func_speed_calc**' para el cálculo de velocidad a partir de conmutaciones, y '**motorFSM**' para el cálculo de la máquina de estados en base a la velocidad del sistema. El motivo de llamar a estas dos funciones dentro de la rutina de interrupción es la importancia de estas (dado que la FSM es el principal coordinador en la activación del SOGI-FLL y los algoritmos de control), y su simplicidad en cuanto a cálculo, que no produce grandes sobrecargas sobre la interrupción. Por último, en la rutina de interrupción se escribe un '1' en la variable 'motor_step', que hará que en la siguiente iteración del bucle del programa, se pueda acceder al segmento de control del SOGI-FLL.

Volviendo al diagrama de flujo, una vez que 'motor_step' está a 1, se puede acceder a toda una gama de funciones que intervendrán en el control del BLDC. Al ser estas funciones más complejas, o requerir llamadas recursivas a otras funciones, se ha decidido colocarlas en el bucle principal del programa arbitradas por la interrupción, de modo que se eviten sobrecargas en el controlador. Dentro de este segmento de código, se comienza con la lectura de los sensores de efecto Hall, que se almacenan con tipo de datos 'ThreePhase' (Tres variables float que simulan una señal trifásica), para posteriormente convertirlos a coordenadas Alfa y Beta con la transformada de Clarke. Esta nueva señal se utiliza como entrada para el SOGI-FLL, y se ejecuta un ciclo de cálculo de este, obteniendo una velocidad angular en el FLL y una fase a la salida de los SOGI-QSG, siempre que se esté en los estados 2 o 3 de la FSM. Tras esto, se escriben los valores de fase y velocidad angular en los DAC, y se borra el flag 'motor_step' para evitar realizar un nuevo paso de cálculo hasta que la interrupción del Timer 0 vuelva a suceder.

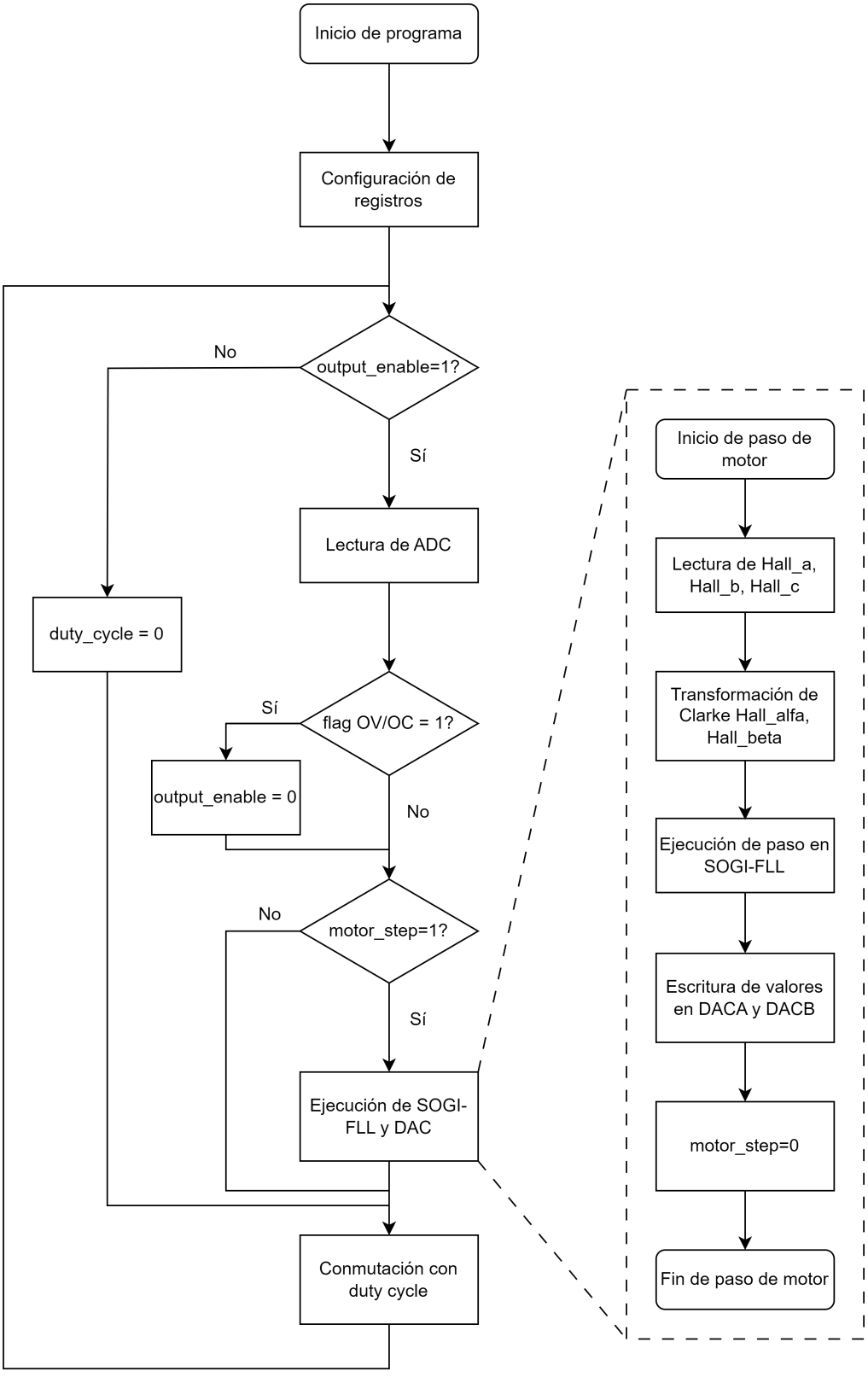


**Figura 4.16: Diagrama de flujo del programa principal DRV8329A_BLDC_CONTROL.c (original)**

## 4.3. OBTENCIÓN DE RESULTADOS EXPERIMENTALES

Este apartado tendrá como objetivo mostrar los resultados experimentales que se han podido obtener tras la implementación del código en el DSP TMS320F280049C, y la correcta configuración de los DAC para la visualización de datos en un osciloscopio. Se observarán diferentes señales disponibles para la medida como las tensiones de fase, sensores de efecto Hall, y medidas de fase y velocidad angular, simulando además condiciones de funcionamiento con rampas de velocidad y escalones de carga. Para todos los resultados obtenidos se tendrá como núcleo principal el SOGI-FLL.

### 4.3.1. CREACIÓN DE RAMPAS DE VELOCIDAD

Una de las pruebas principales que han servido para determinar la estabilidad del control implementado ha sido la creación de rampas de velocidad, mediante la variación del ciclo de trabajo en control trapezoidal. Esta es una de las pruebas principales para determinar la estabilidad del algoritmo SOGI-FLL que se ha implementado.

En primer lugar, en la figura 4.17 se observa una rampa completa de velocidad, que comienza con un ciclo de trabajo de 0.1, con el SOGI-FLL completamente desincronizado (se observa cómo la medida de la fase es nula). Además, se observa cómo la velocidad, aunque es muy pequeña, está siendo estimada por el cálculo de velocidad por conmutaciones (función 'speed_calc'). Una vez que el ciclo de trabajo aumenta a 0.9, la velocidad angular es suficiente para sincronizar el SOGI-FLL, y se muestra cómo la fase del sistema comienza a oscilar. Es importante observar también cómo la frecuencia del sensor Hall A decrece junto con la frecuencia de la medida de fase, cuando se crean las ranpas de velocidad descendentes.

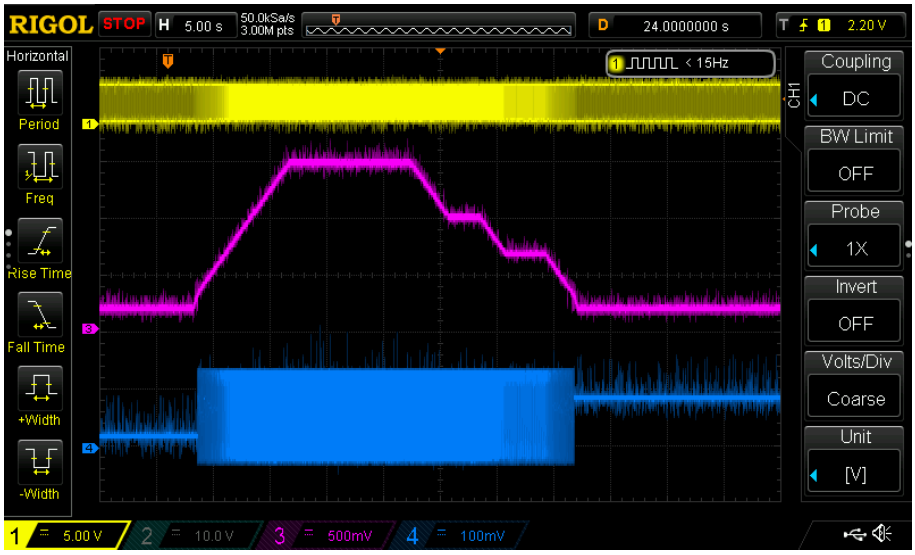


**Figura 4.17: Sensor Hall A, rampa de velocidad de secuencia d=0.1, d=0.9, d=0.6, d=0.5 y fase del SOGI-FLL trifásico**

El efecto de sincronización del SOGI-FLL y de cambio en la selección de uno u otro método de cálculo de velocidad está presente en la figura 4.18. En este caso, se ha creado una pequeña rampa

de velocidad desde un *duty cycle* de 0, hasta un *duty cycle* de 0.2. Se observa cómo el sistema es capaz de calcular la velocidad previa a la sincronización del SOGI-FLL de manera muy precisa, y cómo se produce una pequeña oscilación cuando la FSM conmuta de estado 1 (SOGI-FLL apagado) a estado 2 (SOGI-FLL sincronizado).

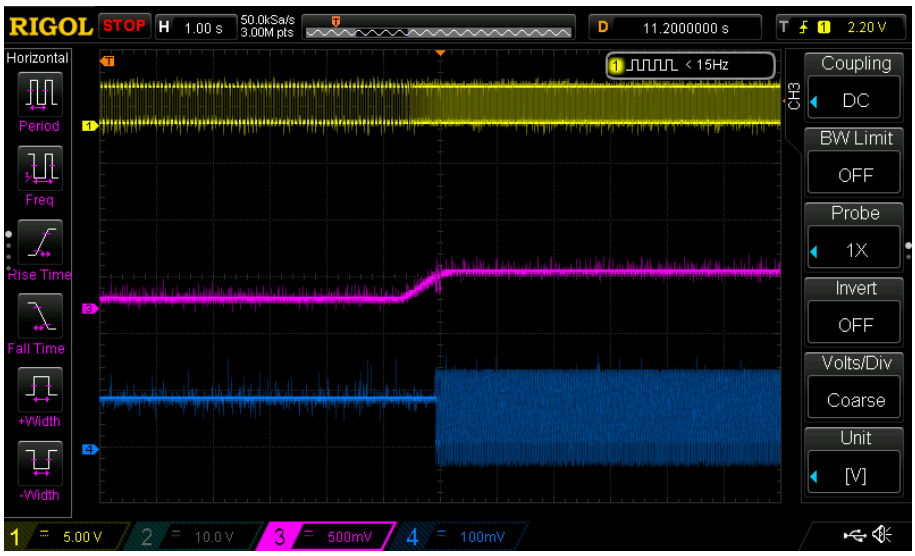


**Figura 4.18: Sensor Hall A, rampa de velocidad ascendente de d=0 a d=0.2 y sincronización de SOGI-FLL trifásico**

En la figura 4.19 se tiene algo parecido a los casos anteriores, y esta vez es una rampa de velocidad descendente desde un *duty cycle* de 0.9 hasta un *duty cycle* de 0.2, que hará al sistema conmutar entre los estados 2 y 3 de la FSM, pero que permitirá mantener el SOGI-FLL sincronizado en todo momento.

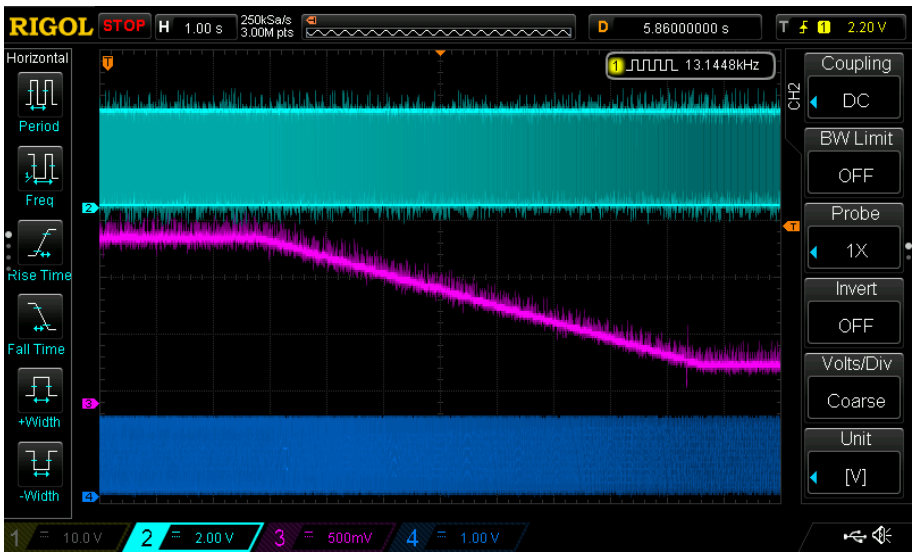


**Figura 4.19: Sensor Hall A, rampa de velocidad descendente de d=0.9 a d=0.2 y fase del SOGI-FLL trifásico**

Se puede observar cómo el sensor Hall A en la parte superior y la fase del SOGI-FLL en la parte inferior van disminuyendo en frecuencia conforme la velocidad disminuye.

### 4.3.2. FASE DEL SISTEMA ESTIMADA CON SOGI-FLL TRIFÁSICO

Si ahora se presta una mayor atención a la fase calculada por los SOGI-FLL, se podrá observar más en detalle cómo esta varía en frecuencia cuando se varía el ciclo de trabajo. En primer lugar, en la figura 4.20 se cuenta con una velocidad constante dada por un ciclo de trabajo d=0.2. Si tomamos la frecuencia de la fase obtenida con este ciclo de trabajo como referencia, se podrá observar una notable diferencia cuando este se aumenta hasta 0.9 en la figura 4.21.

Este efecto es interesante, porque se observa que los valores límites de la fase no han sido afectados, y solamente ha habido una variación en la frecuencia de la medida de la fase, como resultado de la variación en la frecuencia de los sensores de efecto Hall.

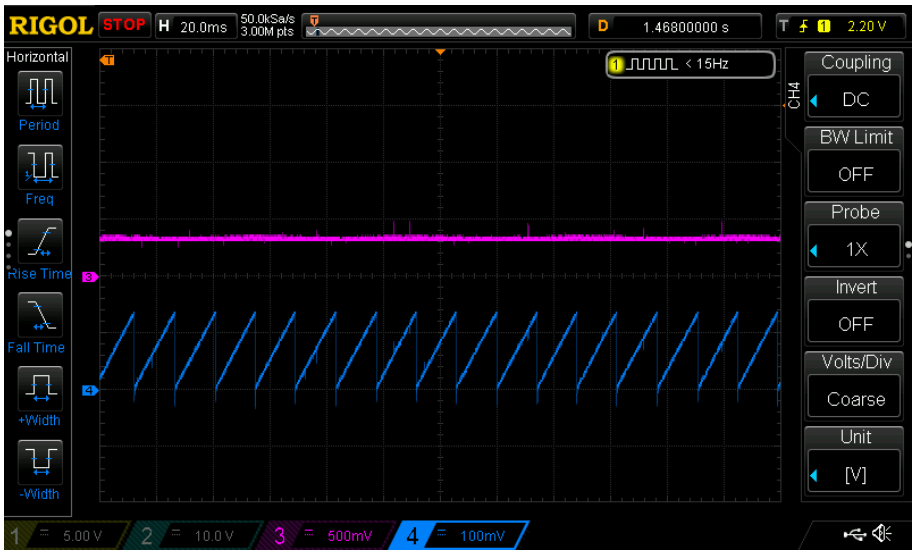


**Figura 4.20: Fase del SOGI-FLL trifásico para velocidad constante con d=0.2**

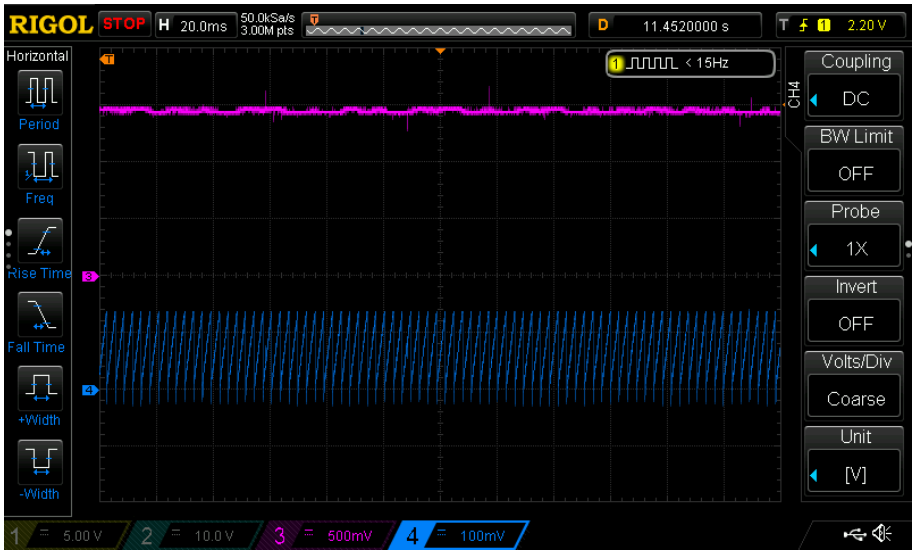


**Figura 4.21: Fase del SOGI-FLL trifásico para velocidad constante con d=0.9**

Por otro lado, se puede observar de manera más general en la figura 4.22 cómo la frecuencia de la estimación de la fase varía en los dos sentidos, cuando se crea una rampa ascendente de velocidad desde un *duty cycle* de 0.2 hasta 0.9, y se vuelve a bajar hasta 0.2.

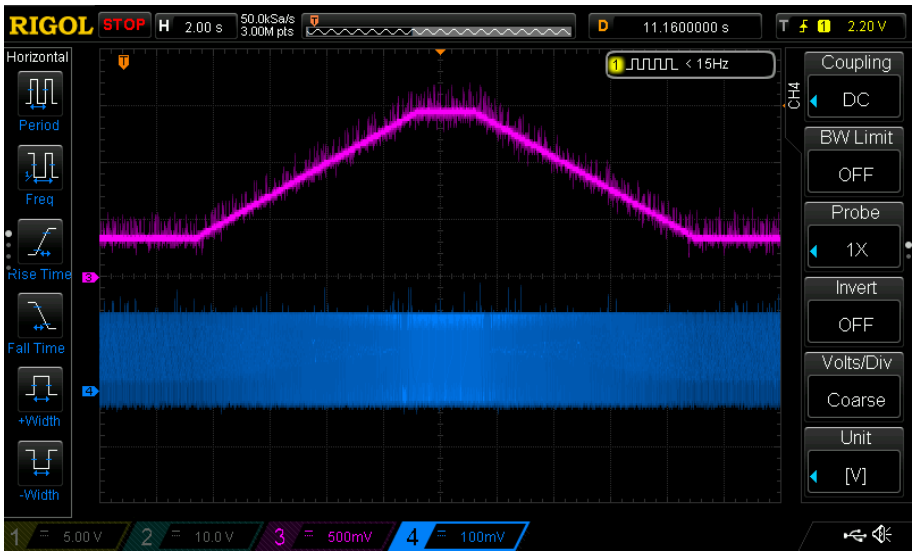


**Figura 4.22: Fase del SOGI-FLL trifásico para rampa ascendente-descendente de velocidad de d=0.2 a d=0.9**

Para comprobar qué robustez tiene el SOGI-FLL frente a escalones en la carga, se ha provocado una deceleración repentina del motor BLDC con una velocidad constante. El resultado de la figura 4.23 indica que el sistema ha sido capaz de recuperar su velocidad original, y el sincronismo del SOGI-FLL no se ha visto comprometido en ningún momento. Además, se puede observar cómo la frecuencia en la fase estimada por el SOGI-FLL decrece cuando se alica el escalón de carga y del mismo modo sucede con la lectura del sensor Hall A. Tras el escalón, la fase del SOGI-FLL

vuelve a ser idéntica a la que se tenía al comienzo del mismo.

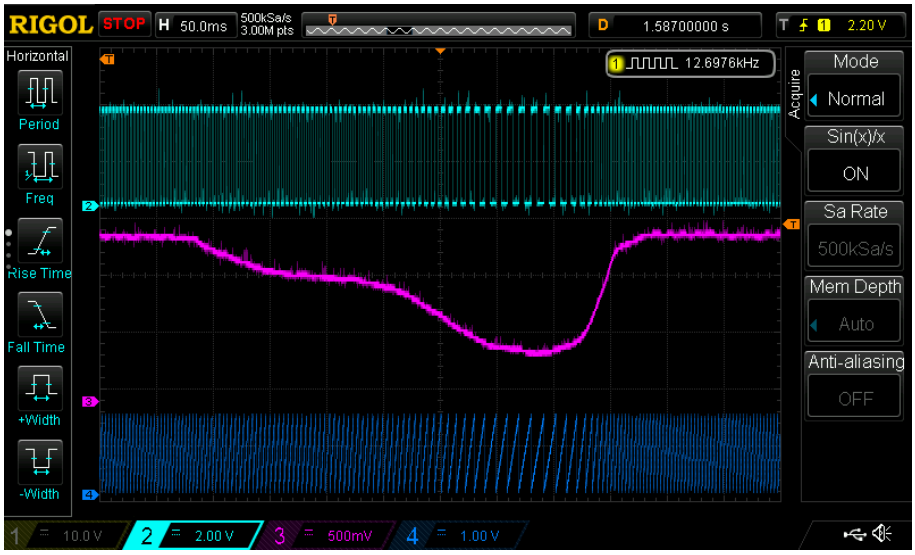


**Figura 4.23: Sensor Hall A, velocidad constante con escalón de carga y fase con SOGI-FLL sincronizado**

### 4.3.3. TENSIONES DE LÍNEA FRENTE A OTROS PARÁMETROS

Un parámetro muy interesante que se puede observar midiendo sobre el inversor, es la medida de las tensiones de fase Va, Vb y Vc. Estas tensiones estarán formadas por un tren de pulsos a causa de la modulación trapezoidal, y deberán asemejarse a un trapecio cuando la frecuencia del tren de pulsos aumenta (para ciclos de trabajo mayores).

Si se toma la figura 4.24 como referencia, puede observarse cómo para un ciclo de trabajo de 0.2, la fase, el sensor de efecto Hall A y el tren de pulsos tendrán una frecuencia relativamente baja. Se puede comparar esta figura con la de 4.25, donde en este caso el ciclo de trabajo ha aumentado hasta 0.9, y se observa cómo la frecuencia en la tensión, sensor Hall A y fase del SOGI-FLL ha aumentado (con una escala temporal idéntica en ambos casos). También se observa cómo la tensió´n Va tiene una forma cada vez más trapezoidal.

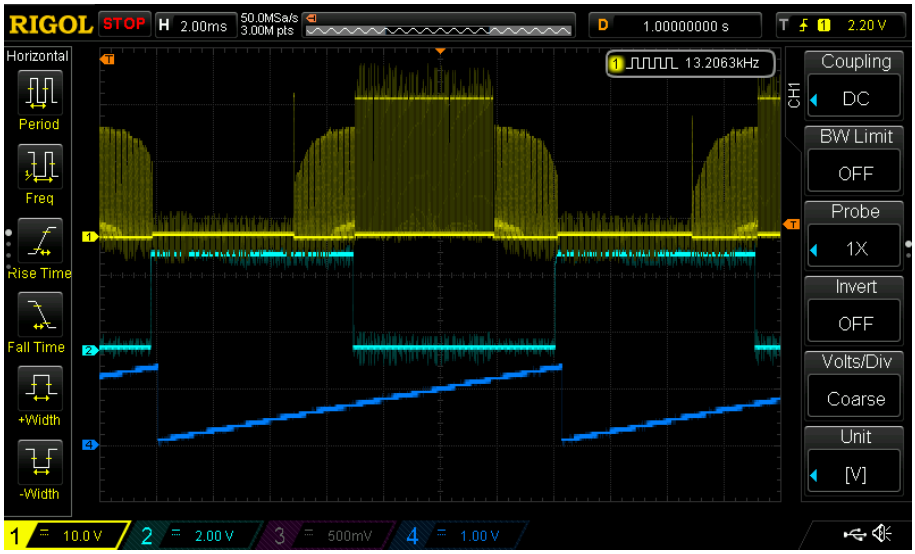


**Figura 4.24: Tensión de fase Va, sensor Hall A y fase del sistema con d=0.2**

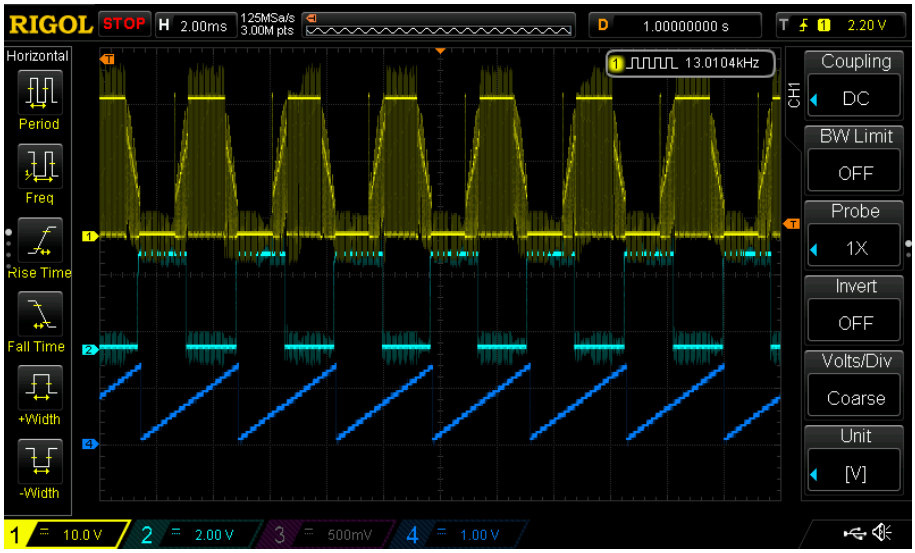


**Figura 4.25: Tensión de fase Va, sensor Hall A y fase del sistema con d=0.9**

Si se comparan los resultados anteriores con la figura 4.26, con una escala temporal que está ampliada, se puede observar cómo en este caso, para un ciclo de trabajo máximo de 1, la señal PWM está la mayor parte del tiempo a nivel alto, y se distingue claramente la forma trapezoidal de la tensión de fase Va. Por otro lado, se puede destacar que no hay presencia de ningún tipo de distorsión en la fase producida por los cambios de velocidad.

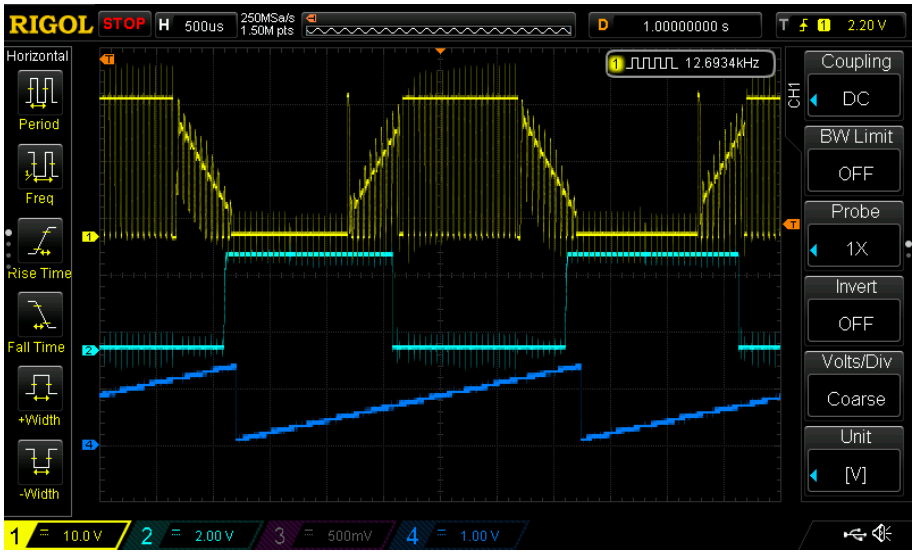


**Figura 4.26: Tensión de fase Va, sensor Hall A y fase del sistema con d=1.0**

Por último, se pueden poner en contexto las tres tensiones de fase Va, Vb y Vc, con un ciclo de trabajo de 1 y una forma trapezoidal casi perfecta. Estas están desfasadas 120 grados exactamente entre sí, y muestran que el entorno del motor es un sistema trifásico equilibrado.

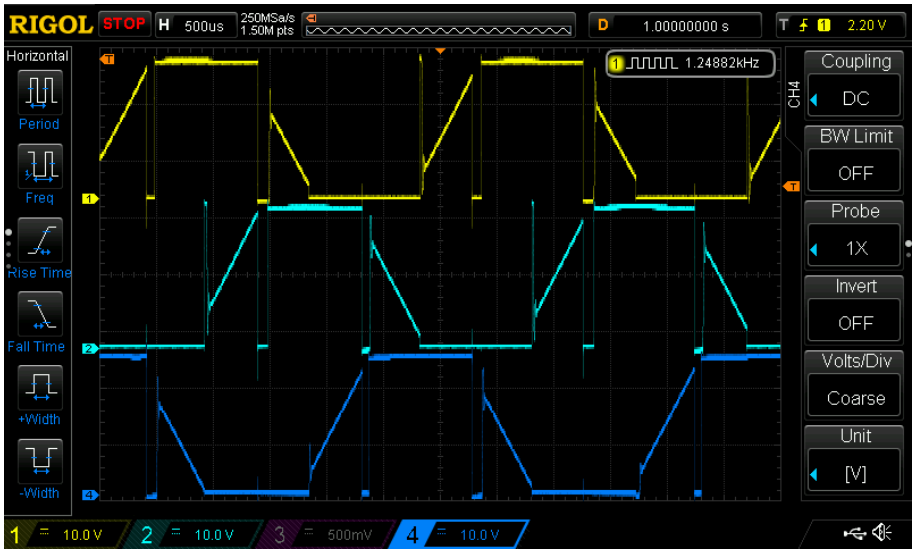


**Figura 4.27: Tensiones de fase Va, Vb, Vc don d=1.0**

# Capítulo 5

# CONCLUSIONES FINALES ACERCA DEL DESARROLLO

En este Trabajo de Fin de Máster se ha desarrollado un sistema de control completo y eficiente para el manejo de un motor BLDC, utilizando de manera original el algoritmo SOGI-FLL para el cálculo de parámetros que suelen estimarse mediante componentes en la interfaz mecánica como medidores de revoluciones, o encoders para la medida de la posición angular. En los siguientes apartados se desglosarán los puntos fuertes del diseño que se ha propuesto, y se darán ideas nuevas para posibles investigaciones que utilicen lo aquí propuesto como base.

## 5.1. RESUMEN SOBRE EL DESARROLLO TÉCNICO

En esta memoria, se ha afrontado el diseño completo de un esquema de control híbrido Trapezoidal-FOC orientado a motores BLDC. Como resumen de las actividades que se han llevado a cabo es el siguiente:

- **Desarrollo conceptual acerca de motores BLDC**: Se han definido algunas características principales de los motores BLDC, como sus métodos de control, o los distintos tipos que existen.
- **Diseño de lazos de control**: Se han diseñado los lazos de control a partir de los modelos del inversor y motor BLDC. Se han obtenido las ecuaciones de lazo para el control trapezoidal y para el control de corriente y velocidad del FOC, y también se ha hecho todo el desarrollo teórico necesario para el dimensionamiento del DC-LINK y SOGI-FLL. Este último ha sido el mecanismo que ha cohesionado el esquema de control híbrido, y es sin duda el núcleo principal del control.
- **Obtención de resultados mediante simulación**: El diseño de un banco de pruebas con componentes reales y la simulación con PSIM permiten evaluar el comportamiento del control. Se han realizado simulaciones donde se demuestra la capacidad de sincronización del SOGI-FLL, el seguimiento de las referencias en los lazos de control, y se observan otros parámetros como ciclos de trabajo, corrientes y tensiones.

- **Obtención de resultados experimentales**: Para demostrar el funcionamiento real de algunas características en un entorno real con DSP, se han adquirido componentes para la evaluación, como una placa de desarrololllo y un inversor de Texas Instruments, y un motor BLDC que se adapte al inversor. En este contexto, se han podido probar características como el SOGI-FLL, el cálculo de velocidad por conmutaciones, el cálculo de feed-forwards, la FSM para el control general, y las transformadas de Clarke directas e inversas.

Todo lo anterior se ha validado mediante resultados obtenidos por simulación en PSIM con dos modelos diferentes, y mediante la implementación de algunas características en un DSP real.

## 5.2. OPCIONES PARA INVESTIGACIONES FUTURAS

Al tratarse este documento de un trabajo de fin de máster, se ha visto limitado un desarrollo mucho más profundo de la arquitectura de control que en este se propone, o un enfoque hacia aplicaciones específicas en situaciones reales de funcionamiento. No obstante, con este documento como premisa, se facilita el desarrollo de posteriores investigaciones donde se realicen desarrollos similares.

Para avanzar en el desarrollo sistema de control híbrido con SOGI-FLL aquí propuesto, se proponen varias ideas:

- **Rediseño de control FOC para debilitamiento de flujo**: Para aplicaciones donde se requiera una gran velocidad sin gran necesidad de par, podrá hacerse debilitamiento de flujo para así aumentar la velocidad del rotor por encima de $Nr_{nom}$.
- **Implementación del diseño en FPGA**: En una plataforma FPGA, podrá implementarse gran parte de la lógica de control utilizando los CLB, y utilizar procesadores integrados para tareas específicas programadas en lenguaje C.
- **Diseño de un inversor con DSP integrado**: Se podrá trasladar la propuesta de diseño de este documento a una PCB donde se incluyan tanto la etapa de conversión de potencia como las distintas etapas de control, de forma aislada.

# Bibliografía


[1] Ajay Sharma. *Construction of DC Motor*. 2023. URL: https://www.yourelectricalguide.com/2017/09/construction-of-dc-motor-machine.html.

[2] MOTION CONTROL TIPS: A Design World Resource. *What's the difference between an EC motor and a BLDC motor?* 2023. URL: https://www.motioncontroltips.com/whats-the-difference-between-an-ec-motor-and-a-bldc-motor.

[3] R. Krishnan y P. Pillay. «Application Characteristics of Permanent Magnet Synchronous and Brushless dc Motors for Servo Drives». En: *IEEE TRANSACTIONS ON INDUSTRY APPLICATIONS* (1991). ISSN: 0093-9994/91/0900-0986$01.00.

[4] R. Krishnan. *Permanent Magnet Synchronous and Brushless DC Motor Drives*. California Technical Pub, 2009, pág. 612. ISBN: 0824753844.

[5] BLAUBERG Motoren. *PMSM vs. BLDC*. 2023. URL: https://blauberg-motoren.com/news/article/pmsm-vs-bldc.

[6] TYTO ROBOTICS. *How to Calculate Motor Kv & Motor Poles*. 2023. URL: https://www.tytorobotics.com/blogs/articles/how-to-calculate-motor-poles-and-brushless-motor-kv.

[7] DigiKey. *Why and How to Sinusoidally Control Three-Phase Brushless DC Motors*. 2017. URL: https://www.digikey.es/es/articles/why-and-how-to-sinusoidally-control-three-phase-brushless-dc-motors.

[8] TEXAS INSTRUMENTS INC. *slyp711_TI_BLDC_CONTROL*. TEXAS INSTRUMENTS INC. Texas Instruments, Post Office Box 655303, Dallas, Texas 75265. URL: www.ti.com.

[9] Chang-Liang Xia. «PERMANENT MAGNET BRUSHLESS DC MOTOR DRIVES AND CONTROLS». En: *ResearchGate* (2012). ISSN: 10.1002/9781118188347.fmatter.

[10] Jordan Yates. *What Are DC Link Supporting Filters?* 2023. URL: https://blog.knowlescapacitors.com/blog/what-are-dc-link-supporting-filters.

[11] Ren Xuhu et al. «Synchronization signal extraction method based on enhanced DSSOGI-FLL in power grid distortion». En: *SYSTEMS SCIENCE & CONTROL ENGINEERING: AN OPEN ACCESS JOURNAL* (2018). ISSN: 10.1080/21642583.2018.1554843.

[12] Pedro Rodríguez et al. «Grid Synchronization of Power Converters using Multiple Second Order Generalized Integrators». En: *IEEE Xplore* (2008). ISSN: 978-1-4244-1766-7/08/$25.00.

[13] Pedro Rodríguez et al. «Multiresonant Frequency-Locked Loop for Grid Synchronization of Power Converters Under Distorted Grid Conditions». En: *IEEE Xplore* (2011). ISSN: 0278-0046/$26.00.

[14] Saeed Golestan et al. «A Study on Three-Phase FLLs». En: *IEEE TRANSACTIONS ON POWER ELECTRONICS* (2019). ISSN: 0885-8993.

[15] Raúl González Medina. «Estudio y evaluación de prestaciones estáticas y dinámicas de los integradores generalizados de segundo orden en las estructuras de control de inversores fotovoltaicos de dos etapas con conexión a red». Tesis doct. Universitat Politècnica de València, 2015.

[16] SHENZEN FLIER TECH CO. LTD. *D80BLD350-48V-10S Product Data sheet*. SHENZEN FLIER TECH CO. LTD. Nov. de 2023.

[17] Bioenno Tech LLC. *The Pros and Cons of Lithium Ion Batteries: A Deep Dive*. 2023. URL: `https://www.bioennopower.com/blogs/news/the-pros-and-cons-of-lithium-ion-batteries-a-deep-dive`.

[18] Pacto Power CO. *60V 40Ah Portable Lithium-Ion BATTERY Pack Product Data Sheet*. PPCB-6040. Pacto Power CO. Second Floor, 145, C – Block, Sector-10, Noida, 201301, feb. de 2024.

[19] BATTERY UNIVERSITY. *BU-205: Types of Lithium-ion*. 2022. URL: `https://batteryuniversity.com/article/bu-205-types-of-lithium-ion`.

[20] KEMET Electronics Corporation. *KEM_A4024_PEH534-3316397*. YAGEO. Fort Lauderdale, FL 33301 USA, ago. de 2023. URL: `www.kemet.com`.

[21] RS PRO. *RS n° 2163790-2163791*. RS GROUP PLC. URL: `rspro.com`.

[22] TEXAS INSTRUMENTS INC. *DRV8329AEVM User's Guide*. SLVUCF5A. TEXAS INSTRUMENTS INC. Texas Instruments, Post Office Box 655303, Dallas, Texas 75265, ago. de 2022. URL: `www.ti.com`.

[23] TEXAS INSTRUMENTS INC. *C2000™ Piccolo™ F28004x Series LaunchPad™ Development Kit*. SPRUII7B. TEXAS INSTRUMENTS INC. Texas Instruments, Post Office Box 655303, Dallas, Texas 75265, abr. de 2020. URL: `www.ti.com`.

# Parte II

# Anexos

# Apéndice A

# CÓDIGO MATLAB DESARROLLADO

## A.1. CÓDIGO MATLAB PARA CÁLCULO DE CONTROL Y SOGI-FLL

**Listing A.1: Código de Matlab para cálculo de control y SOGI-FLL**

```
%%
%=========================================================================
%Definición de parámetros del motor BLDC
%=========================================================================

    Num_phases = 3;
    Num_poles = 8;
    PP=Num_poles/2;

    Vdc=60;
    P_nom = 345;
    T_nom = 3.3;
    I_nom = 11;

    nr_nom=1000;
    f_nom=nr_nom*PP/60;
    w_nom = 2*pi*f_nom;

    nr_max=2000;
    f_max=nr_max*PP/60;
    w_max = 2*pi*f_max;

    Rpp = 0.596;
    Lpp = 0.96E-3;

    Rs = Rpp/2;
    Ls = Lpp/2;
    M = -0.4*Ls;

    Jmot=1.68E-5;
```

```
31
32      K_t = 0.33;
33      K_v = 41.7;          %Constante de velocidad en rpm/v
34      K_e = 60/(K_v*2*pi);    %Constante contraelectromotriz en Wb/rad.
35      K_m = 1/(K_v*PP);    %Constante contraelectromotriz en Wb/revolución.
36      %K_m = K_t/PP;
37
38      nr_noload = 2000;
39      I_noload = 1.1;
40
41      s=tf('s');
42
43  %%
44  %=========================================================================
45  %Cálculo de coeficientes para SOGI-FLL
46  %=========================================================================
47
48  sec=1;  %Coeficiente de seguridad
49  f_SOGI=f_max*sec;
50  w_SOGI=2*pi*f_SOGI;
51  Ts_qsg=5/f_SOGI;     %Tiempo de establecimiento de n periodos
52
53  K_qsg=10/(Ts_qsg*w_SOGI);
54  BW_qsg=K_qsg*w_SOGI/(2*pi);
55
56  Ts_fll=5/f_SOGI;
57  Vamp=1; %Amplitud de la entrada
58  Gamma_fll=5/Ts_fll; %aproximación
59  Gamma_fll_norm=K_qsg*w_SOGI*Gamma_fll/(Vamp^2);
60
61  D=(K_qsg*w_SOGI*s)/((s^2)+(K_qsg*w_SOGI*s)+(w_SOGI^2));
62  figure;
63  bode(D);
64  %plot(step(D))
65
66  Q=(K_qsg*w_SOGI^2)/((s^2)+(K_qsg*w_SOGI*s)+(w_SOGI^2));
67  figure;
68  bode(Q);
69
70  FLL=Gamma_fll/(s+Gamma_fll);
71  bode(FLL);
72
73  %%
74  %Definición de ganancia del inversor
75      Kinv = 1/sqrt(3);
76      Kinv=1
77      Zs = Rs + Ls*s;
78
79      Gi = (Kinv*Vdc)/Zs;
80
81  %%
82  %=========================================================================
83  %Lazos de regulación de corriente
84  %=========================================================================
85
86  fsw=10000;
87  fci=fsw/20;
88  wci=2*pi*fci;
```

```
89
90  Gi = (Kinv*Vdc)/Zs;
91  Ki_i=wci*Rs/(Vdc*Kinv);
92  Kp_i=Ki_i*Ls/Rs;
93
94  PI_i = Kp_i + Ki_i/s;
95
96  pidTuner(Gi,PI_i);
97
98  %Representación de curvas de Bode para estabilidad
99  w=1:0.1:2*pi*100000;
100
101 figure
102 %margin(Gi,w)
103
104 figure
105 margin(PI_i,w)
106
107 figure
108 margin(Gi*PI_i,w)
109
110 %%
111 %=========================================================================
112 %Lazos de regulación de velocidad FOC
113 %=========================================================================
114
115 fcn = fci/10;
116 wcn = 2*pi*fcn;
117
118 Jmin = Jmot;
119 Jmax = 100*Jmot;
120
121 Gn_max=(60/(2*pi))*(3/2)*K_m*PP*(1/(Jmin*s));
122 Gn_min=(60/(2*pi))*(3/2)*K_m*PP*(1/(Jmax*s));
123
124 Kp_N = (Jmin*wcn)/(K_t);
125
126 wcn_min = Kp_N*K_t*(1/(Jmax));
127
128 Ki_N = wcn_min*Kp_N/5;
129
130 PI_N = Kp_N + Ki_N/s;
131
132 T1_N = Gn_min*PI_N;
133 T2_N = Gn_max*PI_N;
134
135 T1 = 1/(1+T1_N);
136 T2 = 1/(1+T2_N);
137
138 %Representación de curvas de Bode para estabilidad
139
140 w=0.001:0.005:2*pi*8000;
141
142 figure
143 margin(Gn_max,w)
144 hold on
145 margin(Gn_min,w)
146 hold off
```

```
147
148 figure
149 margin(T1_N,w)
150 hold on
151 margin(T2_N,w)
152 hold off
153
154 figure
155 margin(T1,w)
156 hold on
157 margin(T2,w)
158 hold off
159
160 %%
161 %=========================================================================
162 %Lazos de regulación de velocidad SIX-STEP
163 %=========================================================================
164
165 fcss = fcn*4;
166 wcss = 2*pi*fcss;
167
168 Jmin = Jmot;
169 Jmax = 1000*Jmot;
170
171 mod_zs = sqrt(pow2(Rs)+pow2(Ls*wcss));
172
173 Gn_SS_min=(Vdc/(2*mod_zs))*(60/(2*pi))*(3/2)*K_m*PP*(1/(Jmax*s));
174 Gn_SS_max=(Vdc/(2*mod_zs))*(60/(2*pi))*(3/2)*K_m*PP*(1/(Jmin*s));
175
176 Kp_SS = (Jmin*wcss)/(((Vdc/(2*mod_zs))*(60/(2*pi))*(3/2)*K_m*PP));
177
178 wcn_min_SS = Kp_SS*(Vdc/(2*mod_zs))*(60/(2*pi))*(3/2)*K_m*PP*(1/(Jmax));
179
180 Ki_SS = wcn_min_SS*Kp_SS/10;
181
182 PI_SS = Kp_SS + Ki_SS/s;
183
184 T1_SS = Gn_SS_min*PI_SS;
185 T2_SS = Gn_SS_max*PI_SS;
186
187 w=0:0.001:2*pi*1000;
188
189 %Representación de curvas de Bode para estabilidad
190
191 figure
192 margin(T1_SS,w)
193 hold on
194 margin(T2_SS,w)
195 hold off
```

## A.2. CÓDIGO MATLAB PARA CÁLCULO DE DC-LINK

**Listing A.2: Código de Matlab para cálculo de DC-LINK**

```
%%
%=========================================================================
%Aproximación teórica en el cálculo de DC-LINK
%=========================================================================
fripp = 10000;
s = tf('s');


%Se supone Rc = Rb en primera aproximación
Rc = 30E-3;
Rb = 30E-3;

fc_ideal = 1000;

Req = sqrt((Rb^2)+(2*Rb*Rc)-Rc^2);

C_teorico = 1/(2*pi*fc_ideal*Req);

Zc = Rc + (1/(s*C_teorico));
H = Zc/(Rb + Zc);

figure
bode(H);
title("Diagrama de Bode teórico")

%%
%=========================================================================
%Diseño de DC-LINK con componentes reales
%=========================================================================
%%SELECCIÓN DE CONDENSADOR%%

%Aproximación logarítmica de la resistencia ESR

ESR_slope = (36E-3-46E-3)/(5-2);

ESR_1k=(3-2)*ESR_slope+46E-3;

C_real = 2*(4.27E-3);

Rc_real = (ESR_1k)/2;

Req_real = sqrt(((Rb^2)+(Rb*Rc_real))-Rc_real^2);

fc_real = 1/(2*pi*C_real*Req_real);

Zc_real = Rc_real + (1/(s*C_real));

H_real = Zc_real/(Rb + Zc_real);

figure
w=1:1:1000000;
bode(H_real,w);
title("Diagrama de Bode ajustado")
```

## A.3. CÓDIGO MATLAB PARA DISCRETIZACIÓN DE LAZOS DE CONTROL

**Listing A.3: Código de Matlab para discretización de lazos de control**

```
clear;
close all;

%%
%=========================================================================
%Discretización de lazos de corriente
%=========================================================================

fs = 20000;
ts = 1/fs;

td_i = c2d(PI_i, ts, 'tustin')

%%
%=========================================================================
%Discretización de lazo de velocidad Six-Step
%=========================================================================

fs = 20000;
ts = 1/fs;

td_n = c2d(PI_SS, ts, 'tustin')

%%
%=========================================================================
%Discretización de lazo de velocidad FOC
%=========================================================================

fs = 20000;
ts = 1/fs;

td_n = c2d(PI_N, ts, 'tustin')
```

# Apéndice B

# CÓDIGO C DESARROLLADO

## B.1. CÓDIGO DEL PROGRAMA PRINCIPAL PARA CONTROL DEL MOTOR BLDC

**Listing B.1: Código C del programa principal**

```
1
2  //INCLUSIÓN DE ARCHIVOS
3  #include "F28x_Project.h"
4  #include "SFO_V8.h"
5  #include "driverlib.h"
6  #include "device.h"
7  #include "motor_control.h"
8
9  //
10 //VARIABLE PARA DAC
11 //
12 volatile struct DAC_REGS* DAC_PTR[3] = {0x0,&DacaRegs,&DacbRegs};
13 uint16_t dacval = 2048;
14
15 //
16 //DEFINICIONES
17 //
18 #define nFAULT 25
19 #define nSLEEP 30
20 #define HALLA 58
21 #define HALLB 23
22 #define HALLC 59
23 #define MCU_LED 57
24 #define DRVOFF 39
25
26 #define nFAULT_flag GpioDataRegs.GPADAT.bit.GPIO25
27
28 #define REFERENCE_VDAC      0
29 #define REFERENCE_VREF      1
30 #define DACA          1
31 #define DACB          2
32
33 #define REFERENCE            REFERENCE_VDAC
```

```
34
35 //
36 //ESTRUCTURA PARA ALMACENAMIENTO DE VALORES PWM
37 //
38 typedef struct
39 {
40     volatile struct EPWM_REGS *EPwmRegHandle;
41 } EPWM_INFO;
42
43 //
44 //DEFINICIÓN DE VARIABLES GLOBALES
45 //
46 int cpuTimer0IntCount = 0;
47
48 int nsamples = 0;   //Cuenta de ciclos de interrupción para cálculo de
       velocidad
49
50 Uint16 PWM_FREQ = 10000, PWM_FREQ_old, PWM_PERIOD;  //Frecuencia general del
       sistema de 10kHz
51 Uint16 DEAD_TIME_ns = 0, DEAD_TIME_old;
52 float pwmDutyCycle;
53 int pwmTrip;
54 int current_duty_cycle;
55 int rampCounter;
56 int accelDelay_ms_dc, accelDelay_old, accelDelay = 0;
57 int hall_state;
58 int hall_state_old;
59 int hall_read;
60 EPWM_INFO epwm6_info;
61 EPWM_INFO epwm5_info;
62 EPWM_INFO epwm3_info;
63 bool direction_CCW = false, direction_old = false;
64 Uint16 StopMotor = false;
65 Uint16 motorBrakeType = 0;
66
67 float SenseResistorValue;
68 int CSAgain;
69 float OCLIM;
70 Uint16 FaultLED;
71 Uint16 PVDD_OV_Fault;
72 Uint16 PVDD_UV_Fault;
73 Uint16 OC_Fault;
74 Uint16 DRV8329_Fault;
75 Uint16 FaultRESET;
76 Uint16 FaultCounter;
77 Uint16 FaultLimit;
78
79 float PVDD_OVLO_LIM;
80 float PVDD_UVLO_LIM;
81 Uint16 VSENPVDD_ADCREAD;
82 float VSENPVDD_V;
83 Uint16 VSENA_ADCREAD;
84 float VSENA_V;
85 Uint16 VSENB_ADCREAD;
86 float VSENB_V;
87 Uint16 VSENC_ADCREAD;
88 float VSENC_V;
89 Uint16 ISEN_ADCREAD;
```

```
90  float ISEN_CURRENT;
91
92  int POT_ADCREAD;
93
94  Uint16  OutputEnable;
95  bool PotEnable;
96  bool DRVOFF_signal = 0, DRVOFF_signal_old;
97  bool nSLEEP_signal = 0, nSLEEP_signal_old;
98
99  int temp, temp1 = 0;
100
101 Uint16 motor_step=0;    //Flag para la realización de un paso de cálculo en el
        motor.
102
103 ThreePhase hall_tri;
104 ClarkeTrans hall_clarke;
105 FSMtypes fsm;
106
107 float wr=0;
108 float nr=0;
109 float wr_sogi=0;
110 float nr_sogi=0;
111 float nr_select=0;
112 float theta_sogi=0;
113 float wr_ff_sogi=420;
114 int direction=1;
115
116 int speed_iteration=0;
117
118 uint16_t dacValueA = 2048; // Valor de inicio del DACA
119 uint16_t dacValueB = 2048; // Valor de inicio del DACB
120
121 float potMult=1.0;
122
123 //
124 //FUNCIONES PROTOTIPO
125 //
126 void resetDRV(void);
127 void DRV_InitGpio(void);
128 void DRV_InitGpioInput(void);
129 void DRV_InitGpioOutput(void);
130 void EPWM_Init(void);
131 void error (void);
132 void Set_HS0_LS1(EPWM_INFO *epwm_info);
133 void Set_HS0_LS0(EPWM_INFO *epwm_info);
134 void Set_PWM(EPWM_INFO *epwm_info, int duty_cycle);
135
136 void InitADCs(void);
137 void SetVREF(int module, int mode, int ref);
138 int16_t sampleADC_A(int channel);
139 int16_t sampleADC_B(int channel);
140 int16_t sampleADC_C(int channel);
141
142 void Config_evm_spi(void);
143
144 void configureDAC(uint16_t dac_num);
145
146 void initCPUTimers(void);
```

```
147 void configCPUTimer(uint32_t, float, float);
148
149 //
150 //FUNCIÓN PROTOTIPO PARA INTERRUPCIÓN DEL TIMER 0
151 //
152 __interrupt void cpuTimer0ISR(void);
153
154 void main(void)
155 {
156     //
157     // INICIO DE PLL, WATCHDOG Y PERIFÉRICOS PRINCIPALES
158     //
159     InitSysCtrl();
160
161     //
162     // INICIO DE GPIO
163     //
164     InitGpio();
165     InitADCs();
166
167     DRV_InitGpioInput();
168     DRV_InitGpioOutput();
169
170     //
171     // Initializes PIE and clears PIE registers. Disables CPU interrupts.
172     //
173     Interrupt_initModule();
174
175     //
176     // Initializes the PIE vector table with pointers to the shell Interrupt
177     // Service Routines (ISR).
178     //
179     Interrupt_initVectorTable();
180
181     //
182     // ISR PARA CADA CPU TIMER
183     //
184     Interrupt_register(INT_TIMER0, &cpuTimer0ISR);
185
186     //
187     // iNICIO DE MÓDULO EPWM
188     //
189
190     InitCpuTimers(); // Inicializa los temporizadores del CPU
191
192     configCPUTimer(CPUTIMER0_BASE, DEVICE_SYSCLK_FREQ, 50);
193
194     EPWM_Init();
195
196     CPUTimer_enableInterrupt(CPUTIMER0_BASE);
197
198     Interrupt_enable(INT_TIMER0);
199
200     CPUTimer_startTimer(CPUTIMER0_BASE);
201
202
203     EINT;  // Habilita las interrupciones globales
204
```

```
205     // Inicializar el DAC A
206     DAC_setShadowValue(DACA_BASE, 0); // Valor inicial
207     DAC_enableOutput(DACA_BASE);
208     DAC_setReferenceVoltage(DACA_BASE, DAC_REF_ADC_VREFHI);
209
210     // Inicializar el DAC B
211     DAC_setShadowValue(DACB_BASE, 0); // Valor inicial
212     DAC_enableOutput(DACB_BASE);
213     DAC_setReferenceVoltage(DACB_BASE, DAC_REF_ADC_VREFHI);
214
215
216     //ACK inicial para interrupciones
217     PieCtrlRegs.PIEACK.all = PIEACK_GROUP1;
218
219     //Initialize and Setup DRV
220     resetDRV();
221     GPIO_WritePin(MCU_LED,0);     //LED del MCU encendido
222
223     //Código para configuración inicial del inversor
224     pwmDutyCycle = 0.0;
225     rampCounter = 0;
226     pwmTrip = 0;
227     accelDelay_ms_dc = 50;
228
229     SenseResistorValue = 0.001;       //Rshunt de 1mOhm
230     CSAgain = 20;                     //Ganancia CSA de 20 V/V
231
232     PVDD_OVLO_LIM = 80;               //PVDD rango entrada 4.5V - 60V
233     PVDD_UVLO_LIM = 2;
234
235     OCLIM = 30;                       //Protección contra sobrecorriente de 30A
236
237     FaultLED = 0;
238     PVDD_OV_Fault = 0;
239     PVDD_UV_Fault = 0;
240     OC_Fault = 0;
241     FaultRESET = 0;
242
243     FaultCounter = 0;
244     FaultLimit = 10;
245
246     OutputEnable = 1;
247     PotEnable = 1;
248
249     while(1)
250     {
251
252         //
253         //CONTROL BÁSICO DE ENCENDIDO/APAGADO DEL MOTOR
254         //
255         if(!OutputEnable)
256         {
257             pwmDutyCycle = 0;
258             PotEnable = 0;
259         }
260         else if(OutputEnable == 1)
261         {
262             if(FaultLED)
```

```
263             {
264                 OutputEnable = 0;
265             }
266         }
267 
268         //
269         // LECTURA DE PARÁMETROS DE ADC
270         //
271         VSENA_ADCREAD = sampleADC_A(5);         //ADCINA5
272         VSENB_ADCREAD = sampleADC_B(0);         //ADCINB0
273         VSENC_ADCREAD = sampleADC_C(2);         //ADCINC2
274         VSENPVDD_ADCREAD = sampleADC_B(1);      //ADCINB1
275         ISEN_ADCREAD = sampleADC_C(0);          //ADCINC0
276 
277         VSENPVDD_V = VSENPVDD_ADCREAD * 0.01857174495; //VSENPVDD
278         VSENA_V = VSENA_ADCREAD * 0.01857174495;       //VSENA
279         VSENB_V = VSENB_ADCREAD * 0.01857174495;       //VSENB
280         VSENC_V = VSENC_ADCREAD * 0.01857174495;       //VSENC
281         ISEN_CURRENT = (((float)(ISEN_ADCREAD) / 4095 * 3.3) - (3.29/7.92)) /
                SenseResistorValue / CSAgain;
282 
283         //
284         // PROTECCIONES CONTRA SOBRETENSIONES/SOBRECORRIENTE
285         //
286         if(nFAULT_flag == 0 || VSENPVDD_V > PVDD_OVLO_LIM || VSENPVDD_V <
                PVDD_UVLO_LIM || (Uint16)(ISEN_CURRENT) > OCLIM)
287         {
288 
289             FaultCounter++;
290 
291             if(FaultCounter >= FaultLimit)
292             {
293                 FaultLED = 1;
294                 if(VSENPVDD_V > PVDD_OVLO_LIM)              //PVDD OV/UV
295                 {
296                     PVDD_OV_Fault = 1;
297                 }
298 
299                 else if(VSENPVDD_V < PVDD_UVLO_LIM)         //PVDD OV/UV
300                 {
301                     PVDD_UV_Fault = 1;
302                 }
303 
304                 else if((Uint16)(ISEN_CURRENT) > OCLIM)     //Sobrecorriente
305                 {
306                     OC_Fault = 1;
307                 }
308 
309                 else if(nFAULT_flag == 0)
310                 {
311                     DRV8329_Fault = 1;
312                 }
313 
314                 OutputEnable = 0;
315                 FaultCounter = 0;
316             }
317         }
318         else
```

```
319         {
320             FaultCounter = 0;                                   //Si no hay fallos,
                    resetea el contador
321         }
322
323         if(FaultRESET)                                          //Borra todos los
                bits de fallos y LEDs
324         {
325             FaultLED = 0;
326             PVDD_OV_Fault = 0;
327             PVDD_UV_Fault = 0;
328             DRV8329_Fault = 0;
329             OC_Fault = 0;
330             FaultRESET = 0;
331             FaultCounter = 0;
332             GPIO_WritePin(nSLEEP,0);
333             for (temp=0;temp<500;temp++);
334             GPIO_WritePin(nSLEEP,1);
335         }
336
337         if(nSLEEP_signal != nSLEEP_signal_old)
338         {
339             nSLEEP_signal_old = nSLEEP_signal;
340             if (nSLEEP_signal)
341             {
342                 GPIO_WritePin(nSLEEP,0);
343             }
344             else
345             {
346                 GPIO_WritePin(nSLEEP,1);
347             }
348         }
349
350         if(DRVOFF_signal != DRVOFF_signal_old)
351         {
352             DRVOFF_signal_old = DRVOFF_signal;
353             if (DRVOFF_signal)
354             {
355                 GPIO_WritePin(DRVOFF,1);
356             }
357             else
358             {
359                 GPIO_WritePin(DRVOFF,0);
360             }
361         }
362
363         //
364         // ITERACIÓN DE CONTROL DEL MOTOR. SUCEDE CUANDO SE EJECUTA LA RUTINA
                DE INTERRUPCIÓN Y MOTOR_STEP=1.
365         //
366         if(motor_step==1){
367
368             //Almacenamiento de lectura de sensores Hall.
369             hall_tri.a = GpioDataRegs.GPBDAT.bit.GPIO58;
370             hall_tri.b = GpioDataRegs.GPADAT.bit.GPIO23;
371             hall_tri.c = GpioDataRegs.GPBDAT.bit.GPIO59;
372             hall_read = (((int)(hall_tri.a) << 2) | ((int)(hall_tri.b) << 1) |
                    ((int)(hall_tri.c) << 0));
```

```
373             hall_state = (hall_read^0xF)&0x7;
374
375             //Se realiza la transformada de Clarke para obtener Hall_alfa y
                    Hall_beta.
376             clarke_dir_transform(&hall_tri, &hall_clarke);
377
378             //Realiza una iteración del SOGI-FLL.
379             sogi_fll_step(&hall_clarke,&wr_ff_sogi, &fsm.motor_sogi, &direction
                    , &theta_sogi, &wr_sogi, &nr_sogi);
380
381             //ALMACENAMIENTO DE NR Y WR EN LOS DACS A Y B.
382             //FONDO ESCALA NR: 6000RPM.
383             //FONDO ESCALA THETA: 2.5*pi.
384             dacValueA = (uint16_t)((nr_select/(6000.0))*4096);
385             dacValueB = (uint16_t)((theta_sogi/(2.5*pi))*4096);
386
387             DAC_setShadowValue(DACA_BASE, dacValueA);
388             DAC_setShadowValue(DACB_BASE, dacValueB);
389
390             //Se sitúa el flag 'motor_step' en 0 para no volver a ejecutar
                    hasta próxima interrupción.
391             motor_step=0;
392         }
393
394         if (PWM_FREQ != PWM_FREQ_old)
395         {
396             PWM_FREQ_old = PWM_FREQ;
397             PWM_PERIOD = 48300000/PWM_FREQ;
398             EPwm6Regs.TBPRD = PWM_PERIOD;
399             EPwm5Regs.TBPRD = PWM_PERIOD;
400             EPwm3Regs.TBPRD = PWM_PERIOD;
401         }
402
403         if (DEAD_TIME_ns != DEAD_TIME_old)
404         {
405             DEAD_TIME_old = DEAD_TIME_ns;
406             EPwm6Regs.DBFED.bit.DBFED = DEAD_TIME_ns * 0.1;
407             EPwm6Regs.DBRED.bit.DBRED = DEAD_TIME_ns * 0.1;
408             EPwm5Regs.DBFED.bit.DBFED = DEAD_TIME_ns * 0.1;
409             EPwm5Regs.DBRED.bit.DBRED = DEAD_TIME_ns * 0.1;
410             EPwm3Regs.DBFED.bit.DBFED = DEAD_TIME_ns * 0.1;
411             EPwm3Regs.DBRED.bit.DBRED = DEAD_TIME_ns * 0.1;
412         }
413
414         if (PotEnable)
415         {
416             POT_ADCREAD = sampleADC_B(2);       //ADCINB2
417             pwmDutyCycle = (float)(POT_ADCREAD*potMult)/40.95;
418         }
419
420         while (StopMotor)
421         {
422             if (motorBrakeType == 0) //FRENADA CON LOW-SIDE A MASA
423             {
424                 Set_HS0_LS1(&epwm6_info);
425                 Set_HS0_LS1(&epwm5_info);
426                 Set_HS0_LS1(&epwm3_info);
427             }
```

```
428             if (motorBrakeType == 1) //fRENADA EN CIRCUITO ABIERTO
429             {
430                 Set_HS0_LS0(&epwm6_info);
431                 Set_HS0_LS0(&epwm5_info);
432                 Set_HS0_LS0(&epwm3_info);
433             }
434             current_duty_cycle = PWM_PERIOD;
435             pwmTrip = 0;
436         }
437 
438         if (direction_CCW != direction_old)
439         {
440             direction_old = direction_CCW;
441             //Frena en circuito abierto y después genera rampa en sentido
                    opuesto
442             Set_HS0_LS0(&epwm6_info);
443             Set_HS0_LS0(&epwm5_info);
444             Set_HS0_LS0(&epwm3_info);
445             for (temp=0;temp<3000;temp++){
446                 for (temp1=0;temp1<3000;temp1++);
447             }
448             current_duty_cycle = PWM_PERIOD;
449             pwmTrip = 0;
450         }
451 
452         if (accelDelay_ms_dc != accelDelay_old)
453         {
454             accelDelay_old = accelDelay_ms_dc;
455             accelDelay = (Uint16)((float)(accelDelay_ms_dc)*2.381);
456         }
457 
458         rampCounter++;
459 
460         if(rampCounter == accelDelay)
461          {
462            if(pwmTrip < (PWM_PERIOD * (pwmDutyCycle/100)))
463            {
464                pwmTrip = pwmTrip + 1;
465            }
466            else if(pwmTrip > (PWM_PERIOD * (pwmDutyCycle/100)))
467            {
468                pwmTrip = pwmTrip - 1;
469            }
470            rampCounter = 0;
471          }
472 
473         current_duty_cycle = PWM_PERIOD - pwmTrip;
474 
475         if (direction_CCW)
476         {
477             if (hall_state)
478             {
479                 switch(hall_state)
480                 {
481                 case 1:
482                     //B-C
483                     Set_HS0_LS0(&epwm6_info);
484                     Set_PWM(&epwm5_info, current_duty_cycle);
```

```
485                     Set_HS0_LS1(&epwm3_info);
486                     break;
487                 case 2:
488                     //A-C
489                     Set_PWM(&epwm6_info, current_duty_cycle);
490                     Set_HS0_LS1(&epwm5_info);
491                     Set_HS0_LS0(&epwm3_info);
492                     break;
493                 case 3:
494                     //A-B
495                     Set_PWM(&epwm6_info, current_duty_cycle);
496                     Set_HS0_LS0(&epwm5_info);
497                     Set_HS0_LS1(&epwm3_info);
498                     break;
499                 case 4:
500                     //C-B
501                     Set_HS0_LS1(&epwm6_info);
502                     Set_HS0_LS0(&epwm5_info);
503                     Set_PWM(&epwm3_info, current_duty_cycle);
504                     break;
505                 case 5:
506                     //C-A
507                     Set_HS0_LS1(&epwm6_info);
508                     Set_PWM(&epwm5_info, current_duty_cycle);
509                     Set_HS0_LS0(&epwm3_info);
510                     break;
511                 case 6:
512                     //B-A
513                     Set_HS0_LS0(&epwm6_info);
514                     Set_HS0_LS1(&epwm5_info);
515                     Set_PWM(&epwm3_info, current_duty_cycle);
516                     break;
517                 }
518             }
519         }
520         else
521         {
522             if (hall_state)
523             {
524                 switch(hall_state)
525                 {
526                 case 6:
527                     //B-C
528                     Set_HS0_LS0(&epwm6_info);
529                     Set_PWM(&epwm5_info, current_duty_cycle);
530                     Set_HS0_LS1(&epwm3_info);
531                     break;
532                 case 5:
533                     //A-C
534                     Set_PWM(&epwm6_info, current_duty_cycle);
535                     Set_HS0_LS1(&epwm5_info);
536                     Set_HS0_LS0(&epwm3_info);
537                     break;
538                 case 4:
539                     //A-B
540                     Set_PWM(&epwm6_info, current_duty_cycle);
541                     Set_HS0_LS0(&epwm5_info);
542                     Set_HS0_LS1(&epwm3_info);
```

```
543                     break;
544                 case 3:
545                     //C-B
546                     Set_HS0_LS1(&epwm6_info);
547                     Set_HS0_LS0(&epwm5_info);
548                     Set_PWM(&epwm3_info, current_duty_cycle);
549                     break;
550                 case 2:
551                     //C-A
552                     Set_HS0_LS1(&epwm6_info);
553                     Set_PWM(&epwm5_info, current_duty_cycle);
554                     Set_HS0_LS0(&epwm3_info);
555                     break;
556                 case 1:
557                     //B-A
558                     Set_HS0_LS0(&epwm6_info);
559                     Set_HS0_LS1(&epwm5_info);
560                     Set_PWM(&epwm3_info, current_duty_cycle);
561                     break;
562                 }
563             }
564         }
565     }
566 }
567
568 //
569 // DRV_InitGpio - Inicializa GPIO en Launchpad
570 //
571 void DRV_InitGpioInput()
572 {
573     EALLOW; // EALLOW Es usado para escribir en registros protegidos
574     // GPIO DRV Hall A
575     GPIO_SetupPinMux(HALLA, GPIO_MUX_CPU1, 0);
576     GPIO_SetupPinOptions(HALLA, GPIO_INPUT, GPIO_PULLUP);
577
578     // GPIO DRV Hall B
579     GPIO_SetupPinMux(HALLB, GPIO_MUX_CPU1, 0);
580     GPIO_SetupPinOptions(HALLB, GPIO_INPUT, GPIO_PULLUP);
581
582     // GPIO DRV Hall C
583     GPIO_SetupPinMux(HALLC, GPIO_MUX_CPU1, 0);
584     GPIO_SetupPinOptions(HALLC, GPIO_INPUT, GPIO_PULLUP);
585
586     // nFAULT
587     GPIO_SetupPinMux(nFAULT, GPIO_MUX_CPU1, 0);
588     GPIO_SetupPinOptions(nFAULT, GPIO_INPUT, GPIO_PULLUP);
589
590     EDIS; // Edis para deshabilitar acceso a registros protegidos
591 }
592
593 void DRV_InitGpioOutput()
594 {
595     EALLOW;
596
597     //INHA DRV
598     GpioCtrlRegs.GPADIR.bit.GPIO10 = 1;
599     GpioCtrlRegs.GPAPUD.bit.GPIO10 = 1;
600     GpioCtrlRegs.GPAMUX1.bit.GPIO10 = 1;
```

```
601     //INLA DRV
602     GpioCtrlRegs.GPADIR.bit.GPIO11 = 1;
603     GpioCtrlRegs.GPAPUD.bit.GPIO11 = 1;
604     GpioCtrlRegs.GPAMUX1.bit.GPIO11 = 1;
605     //INHB DRV
606     GpioCtrlRegs.GPADIR.bit.GPIO8 = 1;
607     GpioCtrlRegs.GPAPUD.bit.GPIO8 = 1;
608     GpioCtrlRegs.GPAMUX1.bit.GPIO8 = 1;
609     //INLB DRV
610     GpioCtrlRegs.GPADIR.bit.GPIO9 = 1;
611     GpioCtrlRegs.GPAPUD.bit.GPIO9 = 1;
612     GpioCtrlRegs.GPAMUX1.bit.GPIO9 = 1;
613     //INHC DRV
614     GpioCtrlRegs.GPADIR.bit.GPIO4 = 1;
615     GpioCtrlRegs.GPAPUD.bit.GPIO4 = 1;
616     GpioCtrlRegs.GPAMUX1.bit.GPIO4 = 1;
617     //INLC DRV
618     GpioCtrlRegs.GPADIR.bit.GPIO5 = 1;
619     GpioCtrlRegs.GPAPUD.bit.GPIO5 = 1;
620     GpioCtrlRegs.GPAMUX1.bit.GPIO5 = 1;
621
622     //MCU LED
623     GPIO_SetupPinMux(MCU_LED, GPIO_MUX_CPU1, 0);
624     GPIO_SetupPinOptions(MCU_LED, GPIO_OUTPUT, GPIO_PUSHPULL);
625
626     //nSLEEP
627     GPIO_SetupPinMux(nSLEEP, GPIO_MUX_CPU1, 0);
628     GPIO_SetupPinOptions(nSLEEP, GPIO_OUTPUT, GPIO_PUSHPULL);
629     GPIO_WritePin(nSLEEP,1);
630
631     EDIS;
632 }
633
634 //
635 // EPWM_Init - Inicia configuración de EPWM
636 //
637 void EPWM_Init()
638 {
639     //
640     // enable PWM6, PWM5 and PWM3
641     //
642     CpuSysRegs.PCLKCR2.bit.EPWM6=1;
643     CpuSysRegs.PCLKCR2.bit.EPWM5=1;
644     CpuSysRegs.PCLKCR2.bit.EPWM3=1;
645     //
646     // Setup TBCLK
647     //
648     EPwm6Regs.TBPRD = PWM_PERIOD;               // Set timer period 16000
            TBCLKs
649     EPwm6Regs.TBPHS.bit.TBPHS = 0x0000;         // Phase is 0
650     EPwm6Regs.TBCTR = 0x0000;                   // Clear counter
651     EPwm5Regs.TBPRD = PWM_PERIOD;               // Set timer period 16000
            TBCLKs
652     EPwm5Regs.TBPHS.bit.TBPHS = 0x0000;         // Phase is 0
653     EPwm5Regs.TBCTR = 0x0000;                   // Clear counter
654     EPwm3Regs.TBPRD = PWM_PERIOD;               // Set timer period 16000
            TBCLKs
655     EPwm3Regs.TBPHS.bit.TBPHS = 0x0000;         // Phase is 0
```

```
656     EPwm3Regs.TBCTR = 0x0000;                       // Clear counter
657
658     //
659     // Setup counter mode
660     //
661     EPwm6Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;     // Count up/down
662     EPwm6Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
663     EPwm6Regs.TBCTL.bit.CLKDIV = TB_DIV1;
664     EPwm5Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;     // Count up/down
665     EPwm5Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
666     EPwm5Regs.TBCTL.bit.CLKDIV = TB_DIV1;
667     EPwm3Regs.TBCTL.bit.CTRMODE = TB_COUNT_UPDOWN;     // Count up/down
668     EPwm3Regs.TBCTL.bit.HSPCLKDIV = TB_DIV1;       // Clock ratio to SYSCLKOUT
669     EPwm3Regs.TBCTL.bit.CLKDIV = TB_DIV1;
670
671     //
672     // Setup shadowing
673     //
674     EPwm6Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
675     EPwm6Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
676     EPwm6Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
677     EPwm6Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
678     EPwm5Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
679     EPwm5Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
680     EPwm5Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
681     EPwm5Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
682     EPwm3Regs.CMPCTL.bit.SHDWAMODE = CC_SHADOW;
683     EPwm3Regs.CMPCTL.bit.SHDWBMODE = CC_SHADOW;
684     EPwm3Regs.CMPCTL.bit.LOADAMODE = CC_CTR_ZERO; // Load on Zero
685     EPwm3Regs.CMPCTL.bit.LOADBMODE = CC_CTR_ZERO;
686
687     EPwm6Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM1A
688     EPwm6Regs.AQCTLA.bit.CAD = AQ_CLEAR;
689     EPwm6Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
690     EPwm6Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
691     EPwm6Regs.DBFED.bit.DBFED = DEAD_TIME_ns * 0.1;
692     EPwm6Regs.DBRED.bit.DBRED = DEAD_TIME_ns * 0.1;
693     EPwm5Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM1A
694     EPwm5Regs.AQCTLA.bit.CAD = AQ_CLEAR;
695     EPwm5Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
696     EPwm5Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
697     EPwm5Regs.DBFED.bit.DBFED = DEAD_TIME_ns * 0.1;
698     EPwm5Regs.DBRED.bit.DBRED = DEAD_TIME_ns * 0.1;
699     EPwm3Regs.AQCTLA.bit.CAU = AQ_SET; // set actions for EPWM1A
700     EPwm3Regs.AQCTLA.bit.CAD = AQ_CLEAR;
701     EPwm3Regs.DBCTL.bit.OUT_MODE = DB_FULL_ENABLE; // enable Dead-band module
702     EPwm3Regs.DBCTL.bit.POLSEL = DB_ACTV_HIC; // Active Hi complementary
703     EPwm3Regs.DBFED.bit.DBFED = DEAD_TIME_ns * 0.1;
704     EPwm3Regs.DBRED.bit.DBRED = DEAD_TIME_ns * 0.1;
705
706     //
707     //Disable Interrupt
708     //
709     EPwm6Regs.ETSEL.bit.INTEN = 0;                  // disable INT
710     EPwm5Regs.ETSEL.bit.INTEN = 0;                  // disable INT
711     EPwm3Regs.ETSEL.bit.INTEN = 0;                  // disable INT
712
713
```

```
714     //Set Handles
715     epwm6_info.EPwmRegHandle = &EPwm6Regs;
716     epwm5_info.EPwmRegHandle = &EPwm5Regs;
717     epwm3_info.EPwmRegHandle = &EPwm3Regs;
718
719     //Set Initial Compare values to be counter + 1 so that low side gates turn
            on and stay on to charge HS bootstrap
720     EPwm6Regs.CMPA.bit.CMPA = PWM_PERIOD + 1;
721     EPwm5Regs.CMPA.bit.CMPA = PWM_PERIOD + 1;
722     EPwm3Regs.CMPA.bit.CMPA = PWM_PERIOD + 1;
723 }
724
725 //Función EPWM High-Side 0, Low-Side 1
726 void Set_HS0_LS1(EPWM_INFO *epwm_info)
727 {
728     epwm_info->EPwmRegHandle->DBCTL.bit.POLSEL = DB_ACTV_HIC;     //set to keep
            low side on
729     epwm_info->EPwmRegHandle->CMPA.bit.CMPA = (PWM_PERIOD + 1); // Set so PWM
            counter never trips, keeping high side off
730 }
731
732 //Función EPWM High-Side 0, Low-Side 0
733 void Set_HS0_LS0(EPWM_INFO *epwm_info)
734 {
735     epwm_info->EPwmRegHandle->DBCTL.bit.POLSEL = DB_ACTV_HI;      //Set to keep
            low side off
736     epwm_info->EPwmRegHandle->CMPA.bit.CMPA = (PWM_PERIOD + 1); // Set so PWM
            counter never trips, keeping high side off
737 }
738
739
740 void Set_PWM(EPWM_INFO *epwm_info, int duty_cycle)
741 {
742     epwm_info->EPwmRegHandle->DBCTL.bit.POLSEL = DB_ACTV_HIC;
743     epwm_info->EPwmRegHandle->CMPA.bit.CMPA = duty_cycle;
744 }
745
746 void InitADCs()
747 {
748             //
749             // Set the VREF to internal
750             //
751             SetVREF(ADC_ADCA, ADC_INTERNAL, ADC_VREF3P3);
752             SetVREF(ADC_ADCB, ADC_INTERNAL, ADC_VREF3P3);
753             SetVREF(ADC_ADCC, ADC_INTERNAL, ADC_VREF3P3);
754
755             //
756             // Configure the ADC: Initialize the ADC
757             //
758             EALLOW;
759
760             //
761             // write configurations
762             //
763             AdcaRegs.ADCCTL2.bit.PRESCALE = 6;      // set ADCCLK divider to /4
764             AdcbRegs.ADCCTL2.bit.PRESCALE = 6;
765             AdccRegs.ADCCTL2.bit.PRESCALE = 6;
766             //
```

```
767             // Set pulse positions to late
768             //
769             AdcaRegs.ADCCTL1.bit.INTPULSEPOS = 1;
770             AdcbRegs.ADCCTL1.bit.INTPULSEPOS = 1;
771             AdccRegs.ADCCTL1.bit.INTPULSEPOS = 1;
772
773             //
774             // power up the ADCs
775             //
776             AdcaRegs.ADCCTL1.bit.ADCPWDNZ = 1;
777             AdcbRegs.ADCCTL1.bit.ADCPWDNZ = 1;
778             AdccRegs.ADCCTL1.bit.ADCPWDNZ = 1;
779
780             //
781             // delay for 1ms to allow ADC time to power up
782             //
783             DELAY_US(1000);
784
785             EALLOW;
786
787 }
788
789 //
790 // configureDAC - Configura la salida especificada del DAC
791 //
792 void configureDAC(uint16_t dac_num)
793 {
794     EALLOW;
795     DAC_PTR[dac_num]->DACCTL.bit.DACREFSEL = REFERENCE;
796     DAC_PTR[dac_num]->DACOUTEN.bit.DACOUTEN = 1;
797     DAC_PTR[dac_num]->DACVALS.all = 0;
798     DELAY_US(10); // Delay for buffered DAC to power up
799     EDIS;
800 }
801
802 void SetVREF(int module, int mode, int ref)
803 {
804     Uint16 *offset, offval;
805
806     //
807     // Define offset locations from OTP
808     //
809     offset = (Uint16 *)(0x70594 + (module * 6));
810
811     if((mode == ADC_INTERNAL) && (ref == ADC_VREF3P3))
812     {
813         offval = (*offset) >> 8;    // Internal / 1.65v mode offset
814     }
815     else
816     {
817         offval = (*offset) & 0xFF;  // All other modes
818     }
819
820     //
821     // Write offset trim values and configure reference modes
822     //
823     EALLOW;
824     switch(module)
```

```
825     {
826         case 0:
827             AdcaRegs.ADCOFFTRIM.bit.OFFTRIM = offval;
828             AnalogSubsysRegs.ANAREFCTL.bit.ANAREFASEL = mode;
829             AnalogSubsysRegs.ANAREFCTL.bit.ANAREFA2P5SEL = ref;
830             break;
831         case 1:
832             AdcbRegs.ADCOFFTRIM.bit.OFFTRIM = offval;
833             AnalogSubsysRegs.ANAREFCTL.bit.ANAREFBSEL = mode;
834             AnalogSubsysRegs.ANAREFCTL.bit.ANAREFB2P5SEL = ref;
835             break;
836         case 2:
837             AdccRegs.ADCOFFTRIM.bit.OFFTRIM = offval;
838             AnalogSubsysRegs.ANAREFCTL.bit.ANAREFCSEL = mode;
839             AnalogSubsysRegs.ANAREFCTL.bit.ANAREFC2P5SEL = ref;
840             break;
841         default:
842             break;
843     }
844     EDIS;
845 }
846
847 int16_t sampleADC_A(int channel)
848 {
849     int16_t sample;
850
851     EALLOW;
852
853     AdcaRegs.ADCSOC0CTL.bit.CHSEL = channel;
854
855     AdcaRegs.ADCSOC0CTL.bit.ACQPS = 25;
856     AdcaRegs.ADCINTSEL1N2.bit.INT1SEL = 1;          //end of SOC1 will set INT1
          flag
857     AdcaRegs.ADCINTSEL1N2.bit.INT1E = 1;            //enable INT1 flag
858     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;          //make sure INT1 flag is
          cleared
859
860     AdcaRegs.ADCSOCFRC1.all = 0x03;                 // Force start of
          conversion on SOC0
861
862     while(AdcaRegs.ADCINTFLG.bit.ADCINT1 == 0)      // Wait for end of
          conversion.
863     {
864          // Wait for ADCINT1
865     }
866     AdcaRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;          // Clear ADCINT1
867
868     sample = AdcaResultRegs.ADCRESULT0;             // Get ADC sample result
          from SOC0
869
870     return(sample);
871 }
872
873 int16_t sampleADC_B(int channel)
874 {
875     int16_t sample;
876
877     EALLOW;
```

```
878
879     AdcbRegs.ADCSOC0CTL.bit.CHSEL = channel;
880
881     AdcbRegs.ADCSOC0CTL.bit.ACQPS = 25;
882     AdcbRegs.ADCINTSEL1N2.bit.INT1SEL = 1;          //end of SOC1 will set INT1
            flag
883     AdcbRegs.ADCINTSEL1N2.bit.INT1E = 1;            //enable INT1 flag
884     AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;          //make sure INT1 flag is
            cleared
885
886     AdcbRegs.ADCSOCFRC1.all = 0x03;                 // Force start of
            conversion on SOC0
887
888     while(AdcbRegs.ADCINTFLG.bit.ADCINT1 == 0)      // Wait for end of
            conversion.
889     {
890          // Wait for ADCINT1
891     }
892     AdcbRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;          // Clear ADCINT1
893
894     sample = AdcbResultRegs.ADCRESULT0;             // Get ADC sample result
            from SOC0
895
896     return(sample);
897 }
898
899 int16_t sampleADC_C(int channel)
900 {
901     int16_t sample;
902
903     EALLOW;
904
905     AdccRegs.ADCSOC0CTL.bit.CHSEL = channel;
906     AdccRegs.ADCSOC0CTL.bit.ACQPS = 25;
907     AdccRegs.ADCINTSEL1N2.bit.INT1SEL = 1;  //end of SOC1 will set INT1 flag
908     AdccRegs.ADCINTSEL1N2.bit.INT1E = 1;    //enable INT1 flag
909     AdccRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;  //make sure INT1 flag is cleared
910
911     AdccRegs.ADCSOCFRC1.all = 0x03;                 // Force start of
            conversion on SOC0
912
913     while(AdccRegs.ADCINTFLG.bit.ADCINT1 == 0)       // Wait for end of
            conversion.
914     {
915         // Wait for ADCINT1
916     }
917     AdccRegs.ADCINTFLGCLR.bit.ADCINT1 = 1;          // Clear ADCINT1
918
919     sample = AdccResultRegs.ADCRESULT0;             // Get ADC sample result
            from SOC0
920
921     return(sample);
922 }
923
924 void resetDRV(void){    //set nSLEEP low for 100us
925     GPIO_WritePin(nSLEEP,0);
926     int i;
927     for(i=0;i<10000;i++);
```

```
928     GPIO_WritePin(nSLEEP,1);
929 }
930
931 //
932 // initCPUTimers - This function initializes all three CPU timers
933 // to a known state.
934 //
935 void
936 initCPUTimers(void)
937 {
938     //
939     // Initialize timer period to maximum
940     //
941     CPUTimer_setPeriod(CPUTIMER0_BASE, 0xFFFFFFFF);
942
943     //
944     // Initialize pre-scale counter to divide by 1 (SYSCLKOUT)
945     //
946     CPUTimer_setPreScaler(CPUTIMER0_BASE, 0);
947
948     //
949     // Make sure timer is stopped
950     //
951     CPUTimer_stopTimer(CPUTIMER0_BASE);
952
953     //
954     // Reload all counter register with period value
955     //
956     CPUTimer_reloadTimerCounter(CPUTIMER0_BASE);
957
958     //
959     // Reset interrupt counter
960     //
961     cpuTimer0IntCount = 0;
962 }
963
964 //
965 // configCPUTimer - This function initializes the selected timer to the
966 // period specified by the "freq" and "period" parameters. The "freq" is
967 // entered as Hz and the period in uSeconds. The timer is held in the stopped
968 // state after configuration.
969 //
970 void
971 configCPUTimer(uint32_t cpuTimer, float freq, float period)
972 {
973     uint32_t temp;
974
975     //
976     // Initialize timer period:
977     //
978     temp = (uint32_t)((freq / 1000000) * period);
979     CPUTimer_setPeriod(cpuTimer, temp - 1);
980
981     //
982     // Set pre-scale counter to divide by 1 (SYSCLKOUT):
983     //
984     CPUTimer_setPreScaler(cpuTimer, 0);
985
```

```
986     //
987     // Initializes timer control register. The timer is stopped, reloaded,
988     // free run disabled, and interrupt enabled.
989     // Additionally, the free and soft bits are set
990     //
991     CPUTimer_stopTimer(cpuTimer);
992     CPUTimer_reloadTimerCounter(cpuTimer);
993     CPUTimer_setEmulationMode(cpuTimer,
994                               CPUTIMER_EMULATIONMODE_STOPAFTERNEXTDECREMENT);
995     CPUTimer_enableInterrupt(cpuTimer);
996
997     //
998     // Resets interrupt counters for the three cpuTimers
999     //
1000    if (cpuTimer == CPUTIMER0_BASE)
1001    {
1002        cpuTimer0IntCount = 0;
1003    }
1004
1005 }
1006
1007 //
1008 // cpuTimer0ISR - Rutina de interrupción para Timer0
1009 //
1010 __interrupt void
1011 cpuTimer0ISR(void)
1012 {
1013     cpuTimer0IntCount++;
1014     nsamples++; //Se aumenta en 1 el número de cuentas para cálculo de la
            velocidad
1015     motor_step=1; //Se activa otra iteración en el bucle principal.
1016
1017     //Se activa cálculo de velocidad
1018     speed_calc(&hall_state_old, &hall_state, &nsamples, &wr, &nr);
1019
1020     //Ejecuta paso de la FSM
1021     motorFSM(&fsm, &nr, &nr_sogi, &nr_select);
1022
1023     //
1024     // Acknowledge para recibir más interrupciones futuras
1025     //
1026     Interrupt_clearACKGroup(INTERRUPT_ACK_GROUP1);
1027 }
1028
1029
1030 //
1031 // error - Halt debugger when called
1032 //
1033 void error(void)
1034 {
1035     ESTOP0;          // Stop here and handle error
1036 }
1037
1038
1039 //
1040 // Fin de archivo
1041 //
```

## B.2. CÓDIGO DE LA BIBLIOTECA PARA CONTROL DEL BLDC

(CAMBIAR NOMBRE DE SECCIÓN)

**Listing B.2: Código C de biblioteca de funciones para BLDC**

```
#include <math.h>

//DEFINICIÓN DE ESTRUCTURAS
typedef struct {
    float a;
    float b;
    float c;
} ThreePhase;

typedef struct {
    float alpha;
    float beta;
    float gamma;
} ClarkeTrans;

typedef struct {
    int zone;
    int motor_ss;
    int motor_foc;
    int motor_sogi;
} FSMtypes;

//DEFINICIÓN DE CONSTANTES
const float sqrt2=1.4142113;
const float sqrt3=1.7320508;
const float pi=3.14159265;
const float pi2d3=2.09439510;

const float ts_qsg = 0.0375;
const float ts = 0.00005;
const float tsd2 = 0.000025;
const float g_fll = 133.3333;
const float pp = 4;

//DEFINICIÓN DE VARIABLES GLOBALES
Uint16 zone = 0;
Uint16 motor_ss = 0;
Uint16 motor_foc = 0;

float k_qsg = 0.3183;

//DEFINICIÓN DE FUNCIONES PROTOTIPO
void clarke_dir_transform(ThreePhase*,ClarkeTrans*);
void clarke_inv_transform(ClarkeTrans*,ThreePhase*);

void sogi_ff_calc(float*, float*, float*);
void constCalcD(float*,float*,float*, float*, float*, float*, float*, float*);
void constCalcQ(float*,float*,float*, float*, float*, float*, float*, float*);
void sogi_fll_step(ClarkeTrans*, float*, int*, int*, float*, float*, float*);
void motorFSM(FSMtypes*, float*, float*, float*);

```

```
52 void speed_calc(int*, int*, int*, float*, float*);
53
54
55 //DEFINICIÓN DE FUNCIONES
56 void clarke_dir_transform(ThreePhase *input, ClarkeTrans *output){
57
58     // Función para la realización de la transformada directa de Clarke.
59     // Entradas:
60     // ThreePhase *input: Puntero de entrada trifásica. Coordenadas A,B,C.
61     // Salidas:
62     // ClarkeTrans *output: Puntero de salida bifásica. Coordenadas Alfa, Beta.
63
64     output->alpha = (2.0/3.0)*((input->a)-0.5*(input->b)-0.5*(input->c));
65     output->beta = (1.0/sqrt3)*((input->b)-(input->c));
66     output->gamma = (sqrt2/3.0)*((input->a)+(input->b)+(input->c));
67
68 }
69
70 void clarke_inv_transform(ClarkeTrans *input, ThreePhase *output){
71
72     // Función para la realización de la transformada inversa de Clarke.
73     // Entradas:
74     // ClarkeTrans *output: Puntero de salida bifásica. Coordenadas Alfa, Beta.
75     // Salidas:
76     // ThreePhase *input: Puntero de entrada trifásica. Coordenadas A,B,C.
77
78     output->a = input->alpha+(input->gamma)/sqrt2;
79     output->b = (1.0/2.0)*(-(input->alpha)+sqrt3*(input->beta)+sqrt2*(input->
           gamma));
80     output->c = (1.0/2.0)*(-(input->alpha)-sqrt3*(input->beta)+sqrt2*(input->
           gamma));
81
82 }
83
84 void constCalcD(float *wr,float *k,float *c1, float *c2, float *c3, float *c4,
       float *c5, float *c6){
85
86     // Función para cálculo de constantes de función VVP en SOGI-FLL.
87     // Entradas:
88     // float *wr: Frecuencia angular del FLL.
89     // float *k: Constante K_SOGI.
90     // Salidas:
91     // float *c1, *c2, *c3, *c4, *c5, *c6: Constantes de la función VVP en SOGI
           -FLL.
92
93     *c1 = 2.0*ts**k**wr;
94     *c2 = 0;
95     *c3 =  -2.0*ts**k**wr;
96     *c4 = pow(ts,2)*pow(*wr,2)+2.0**k*ts**wr+4.0;
97     *c5 = 2*pow(ts,2)*pow(*wr,2)-8.0;
98     *c6 = pow(ts,2)*pow(*wr,2)-2.0**k*ts**wr+4.0;
99
100 }
101
102 void constCalcQ(float *wr,float *k,float *c1, float *c2, float *c3, float *c4,
       float *c5, float *c6){
103
104     // Función para cálculo de constantes de función VQVP en SOGI-FLL.
```

```
105     // Entradas:
106     // float *wr: Frecuencia angular del FLL.
107     // float *k: Constante K_SOGI
108     // Salidas:
109     // float *c1, *c2, *c3, *c4, *c5, *c6: Constantes de la función VQVP en
            SOGI-FLL
110
111     *c1 = pow(ts,2)**k*pow(*wr,2);
112     *c2 = 2.0*pow(ts,2)**k*pow(*wr,2);
113     *c3 = pow(ts,2)**k*pow(*wr,2);
114     *c4 = pow(ts,2)*pow((*wr),2)+2.0**k*ts**wr+4.0;
115     *c5 = 2.0*pow(ts,2)*pow(*wr,2)-8.0;
116     *c6 = pow(ts,2)*pow(*wr,2)-2.0**k*ts**wr+4.0;
117
118 }
119
120 void sogi_ff_calc(float *wr, float *wr_ff, float *sogi_reset){
121     // Función para cálculo de feed-forward en el FLL.
122     // Entradas:
123     // float *wr: Frecuencia angular del sistema.
124     // float *sogi_reset: Flag de reset del SOGI-FLL
125     // Salidas:
126     // float *wr_ff: Frecuencia de feed-forward del FLL
127     static int flag=0;
128
129     if((*sogi_reset == 1) && (flag == 0)){
130         if(*wr>=0){
131             *wr_ff = *wr;
132         }else *wr_ff = -*wr;
133     }else if(*sogi_reset == 0){
134         flag = 1;
135     }
136
137 }
138
139 void motorFSM(FSMtypes* controlVars, float* nr, float* nr_sogi, float*
        nr_select){
140
141     // Función de FSM para arbitraje en el control
142     // Entradas:
143     // float *nr: Velocidad calculada a través de conmutaciones de sensores
            Hall.
144     // float *nr_sogi: Velocidad calculada a ravés del algoritmo FLL.
145     // Salidas:
146     // float *nr_select: Velocidad tras multiplexor de velocidad.
147     // FSMtypes *controlVars: Estados actuales de FSM y zona de funcionamiento.
148
149     static int state=0;
150
151     //Nrmax del motor: 4500RPM
152     //FSM simplificada con tres estados: 0 - arranque; 1 - Six-step; 2 - Six-
            step + SOGI-FLL; 3 - FOC + SOGI-FLL
153
154     //Logica de estado siguiente
155     //state = next_state;
156
157     if(state==0){
158         state=1;
```

```
159     }else if((state==1)&&(*nr>=1000)){
160         state=2;
161     }else if((state==2)&&(*nr>=2000)){
162         state=3;
163     }else if((state==3)&&(*nr<=1000)){
164         state=1;
165     }else if((state==2)&&(*nr<=800)){
166         state=1;
167     }
168 
169     //Lgica de salida
170     if(state==0){
171         controlVars->zone=0;
172         controlVars->motor_ss=0;
173         controlVars->motor_sogi=0;
174         controlVars->motor_foc=0;
175         *nr_select = *nr;
176     }else if(state==1){
177         controlVars->zone=0;
178         controlVars->motor_ss=1;
179         controlVars->motor_sogi=0;
180         controlVars->motor_foc=0;
181         *nr_select = *nr;
182     }else if(state==2){
183         controlVars->zone=0;
184         controlVars->motor_ss=1;
185         controlVars->motor_sogi=1;
186         controlVars->motor_foc=0;
187         *nr_select = *nr_sogi;
188     }else if(state==3){
189         controlVars->zone=1;
190         controlVars->motor_ss=0;
191         controlVars->motor_sogi=1;
192         controlVars->motor_foc=1;
193         *nr_select = *nr_sogi;
194     }
195 }
196 
197 void sogi_fll_step(ClarkeTrans* hall_sens, float* wr_ff, int*sogi_reset, int*
        freq_sign, float* theta, float* wr_sogi_fll, float* nr_sogi_fll){
198 
199     // Función para cálculo de algoritmo SOGI-FLL completo
200     // Entradas:
201     // ClarkeTrans *hall_sens: Lectura actual de sensores Hall en coordenadas
            Alfa/Beta.
202     // float *wr_ff: Velocidad angular de feed-forward en FLL.
203     // int *sogi_reset: Flag de reset en SOGI-FLL completo.
204     // int *freq_sign: Dirección de giro del rotor.
205     // Salidas:
206     // float *theta: Fase del rotor en radianes. Rango de 0 a 2*pi.
207     // float *wr_sogi_fll: Velocidad angular calculada con FLL en rad/s.
208     // float *nr_sogi_fll: Velocidad angular calculada con FLL en rpm.
209 
210 
211     float sogi_prod=0;
212     float err_alfa=0;
213     float err_beta=0;
214 
```

```
215     float vp_alpha_pos;
216     float vp_beta_pos;
217 
218     //CONSTANTES PARA COMPONENTE DIRECTA ALFA
219     float cxd_0 = 0.00661963;
220     float cxd_1 = 0;
221     float cxd_2 = -0.00661963;
222 
223     float cyd_0 = 1;
224     float cyd_1 = -1.9850185;
225     float cyd_2 = 0.98676074;
226 
227     //CONSTANTES PARA COMPONENTE CUADRATURA ALFA
228     float cxq_0 = 0.00013864;
229     float cxq_1 = 0;
230     float cxq_2 = 0.00013864;
231 
232     float cyq_0 = 1;
233     float cyq_1 = -1.9850185;
234     float cyq_2 = 0.98676074;
235 
236     // DECLARACIn de memorias
237     static float yd_alfa[3];
238     static float xd_alfa[3];
239     static float yd_beta[3];
240     static float xd_beta[3];
241 
242     static float yq_alfa[3];
243     static float xq_alfa[3];
244     static float yq_beta[3];
245     static float xq_beta[3];
246 
247     static float x_fll[2];
248     static float y_fll[2];
249 
250     if((*wr_sogi_fll*ts_qsg) != 0){
251         sogi_prod = 10.0/(*wr_sogi_fll*ts_qsg);
252         if(sogi_prod >= 100.0) sogi_prod = 100.0;
253         else if(sogi_prod <= -100.0) sogi_prod = -100.0;
254         k_qsg = sogi_prod;
255     }else
256 
257     err_alfa = hall_sens->alpha - yd_alfa[0];
258     err_beta = hall_sens->beta - yd_beta[0];
259 
260     vp_alpha_pos = 0.5*yd_alfa[0] - 0.5*yq_beta[0];
261     vp_beta_pos = 0.5*yq_alfa[0] + 0.5*yd_beta[0];
262 
263     if(*sogi_reset==0){
264         yd_alfa[0] = 0;
265         yd_alfa[1] = 0;
266         yd_alfa[2] = 0;
267         xd_alfa[0] = 0;
268         xd_alfa[1] = 0;
269         xd_alfa[2] = 0;
270         yq_alfa[0] = 0;
271         yq_alfa[1] = 0;
272         yq_alfa[2] = 0;
```

```
273         xq_alfa[0] = 0;
274         xq_alfa[1] = 0;
275         xq_alfa[2] = 0;
276         yd_beta[0] = 0;
277         yd_beta[1] = 0;
278         yd_beta[2] = 0;
279         xd_beta[0] = 0;
280         xd_beta[1] = 0;
281         xd_beta[2] = 0;
282         yq_beta[0] = 0;
283         yq_beta[1] = 0;
284         yq_beta[2] = 0;
285         xq_beta[0] = 0;
286         xq_beta[1] = 0;
287         xq_beta[2] = 0;
288
289         x_fll[0] = 0;
290         x_fll[1] = 0;
291         y_fll[0] = 0;
292         y_fll[1] = 0;
293     }else {
294
295         //Calculo de constantes para SOGI
296
297         constCalcD(wr_sogi_fll,&k_qsg,&cxd_0,&cxd_1,&cxd_2,&cyd_0,&cyd_1,&cyd_2
                );
298         constCalcQ(wr_sogi_fll,&k_qsg,&cxq_0,&cxq_1,&cxq_2,&cyq_0,&cyq_1,&cyq_2
                );
299
300         xd_alfa[0] = hall_sens->alpha;
301         xq_alfa[0] = hall_sens->alpha;
302
303         xd_beta[0] = hall_sens->beta;
304         xq_beta[0] = hall_sens->beta;
305
306         //Calculo VD ALFA
307
308         yd_alfa[0] = (cxd_0*xd_alfa[0]+cxd_1*xd_alfa[1]+cxd_2*xd_alfa[2]-cyd_1*
                yd_alfa[1]-cyd_2*yd_alfa[2])/cyd_0;
309
310         xd_alfa[2] = xd_alfa[1];
311         xd_alfa[1] = xd_alfa[0];
312         yd_alfa[2] = yd_alfa[1];
313         yd_alfa[1] = yd_alfa[0];
314
315         //Calculo VQ ALFA
316
317         yq_alfa[0] = (cxq_0*xq_alfa[0]+cxq_1*xq_alfa[1]+cxq_2*xq_alfa[2]-cyq_1*
                yq_alfa[1]-cyq_2*yq_alfa[2])/cyq_0;
318
319         xq_alfa[2] = xq_alfa[1];
320         xq_alfa[1] = xq_alfa[0];
321         yq_alfa[2] = yq_alfa[1];
322         yq_alfa[1] = yq_alfa[0];
323
324         //Calculo VD BETA
325
```

```
326         yd_beta[0] = (cxd_0*xd_beta[0]+cxd_1*xd_beta[1]+cxd_2*xd_beta[2]-cyd_1*
                yd_beta[1]-cyd_2*yd_beta[2])/cyd_0;
327
328         xd_beta[2] = xd_beta[1];
329         xd_beta[1] = xd_beta[0];
330         yd_beta[2] = yd_beta[1];
331         yd_beta[1] = yd_beta[0];
332
333         //Calculo VQ BETA
334
335         yq_beta[0] = (cxq_0*xq_beta[0]+cxq_1*xq_beta[1]+cxq_2*xq_beta[2]-cyq_1*
                yq_beta[1]-cyq_2*yq_beta[2])/cyq_0;
336
337         xq_beta[2] = xq_beta[1];
338         xq_beta[1] = xq_beta[0];
339         yq_beta[2] = yq_beta[1];
340         yq_beta[1] = yq_beta[0];
341
342         //FLL
343
344         if((vp_alpha_pos != 0) || (vp_beta_pos != 0)){
345             x_fll[0] = (-g_fll*((err_alfa*yq_alfa[0])+(err_beta*yq_beta[0])))
                    *(*wr_sogi_fll*0.3/(pow(vp_alpha_pos,2)+pow(vp_beta_pos,2)));
346             *theta = atan2(vp_alpha_pos,(-1.0**freq_sign*vp_beta_pos));
347             if(*theta < 0) *theta += 2.0*pi;
348         }
349
350         y_fll[0] = tsd2*(x_fll[0]+x_fll[1])+y_fll[1];
351
352         x_fll[1] = x_fll[0];
353         y_fll[1] = y_fll[0];
354
355     }
356
357     // Finalmente, la velocidad se transforma
358     *wr_sogi_fll = y_fll[0] + *wr_ff;
359
360     *nr_sogi_fll = *wr_sogi_fll * 60.0 / (2.0*pi*pp);
361
362 }
363
364 void speed_calc(int *hall_read_old, int *hall_read, int *nsamples, float *wr,
        float *nr){
365
366     // Función para cálculo de velocidad por conmutaciones de Hall
367     // Entradas:
368     // int *hall_read_old: Lectura anterior de los sensores Hall.
369     // int *hall_read: Lectura actual de los sensores Hall.
370     // int *nsamples: Número de muestras transcurridas entre capturas de los
            sensores Hall.
371     // Salidas:
372     // float *wr: Velocidad angular del rotor en rad/s.
373     // float *nr: Velocidad angular del rotor en rpm.
374
375     //Hall sensor format0: {HALLA, HALLB, HALLC} 3-bit
376     float sign = 1;
377
378     if(*hall_read != *hall_read_old){
```

```
379             *nr = sign*60/(0.00005*(*nsamples)*6*pp);
380             *wr = *nr*2*pi*pp/60;
381             *nsamples=1;
382         }else *nsamples++;
383 
384         *hall_read_old = *hall_read;
385 
386 }
```