ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

DESARROLLO DE UN CONTROLADOR PARA MOTOCICLETA ELÉCTRICA

Autor: Victor Guizien Martin

Directores : José Daniel Muñoz Frías y Aurelio García Cerrada

Madrid

Junio 2018

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo de un controlador para motocicleta eléctrica

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2017/18 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos.

El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido

tomada de otros documentos está debidamente referenciada.

Fdo.: Victor Antonio Guizien Martin Fecha: 17 / 7 / 2018

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: José Daniel Muñoz Frías Fecha: 17 / 7 / 2018

Fdo.: Aurelio García Cerrada Fecha: 17 / 7 / 2018

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor D._____

DECLARA ser el titular de los derechos de propiedad intelectual de la obra: _____, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar "marcas de agua" o cualquier otro sistema de seguridad o de protección.
b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia Creative Commons.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente)*.

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
c) Solicitar la retirada de la obra del repositorio por causa justificada.
d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusive del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a …….. de ……... de …….

ACEPTA

Fdo……

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

GRADO EN INGENIERÍA TELEMÁTICA

DESARROLLO DE UN CONTROLADOR PARA MOTOCICLETA ELÉCTRICA

Autor: Victor Guizien Martin

Directores: José Daniel Muñoz Frías y Aurelio García Cerrada

Madrid

Junio 2018

DESARROLLO DE UN CONTROLADOR PARA MOTOCICLETA ELÉCTRICA

Autor: Guizien Martin, Victor Antonio.
Director: Muñoz Frías, José Daniel.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

En este proyecto se diseña de forma completa un controlador para una motocicleta eléctrica: se eligen los componentes de forma justificada, se explica la técnica de control utilizada, se desarrolla el software de control de forma completa y finalmente se simula el control con la ayuda de Simulink para verificar su correcto funcionamiento.

Palabras clave: PMSM, FOC, SVPWM.

1. Introducción

Debido a la creciente demanda de motores eléctricos, la necesidad de controladores eficientes también se ha disparado y el caso de la automoción no es ninguna excepción.

El mercado de las motocicletas eléctricas es todavía joven, y no ha sido hasta este año que el ICAI Speed Club ha empezado a desarrollar su primera motocicleta eléctrica.

2. Definición del proyecto

Este proyecto tiene como objetivo desarrollar un controlador que sea más económico que un controlador comercial, permitiendo además un control más granular de los parámetros que gestiona el software, puesto que se desarrolla en su totalidad.

Para ello, primero será necesario desarrollar el circuito de potencia, que se compone de inversor, un radiador y un driver. A continuación, es necesario desarrollar el circuito de control, formado por un microcontrolador. El siguiente paso es el de simular el control con la ayuda de Simulink, para verificar que no se sobrepasan las corrientes o tensiones límite. Finalmente se diseña el software de control.

3. Descripción del motor de imanes permanentes y del control

Un motor PMSM (*Permanent Magnet Synchronous Motor*) se compone de una parte móvil (rotor) y una estática (estator). El estator se compone de 3 devanados que se alojan de tal forma que crean un cierto número de pares de polos.

El rotor está formado por un imán permanente y tiene el mismo número de pares de polos que los que forma el estator.

El funcionamiento del motor se basa en la interacción entre dos campos magnéticos. Haciendo circular las corrientes por los devanados, las espiras generan un campo magnético que interactúa con el del rotor, consiguiendo el movimiento (Ilustración 1).

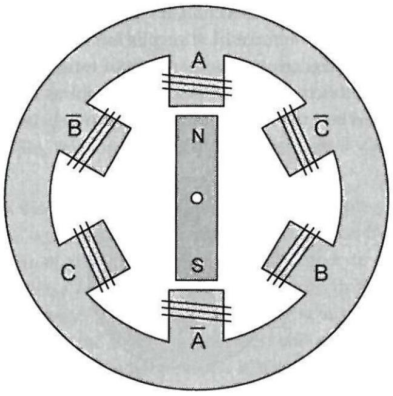


Ilustración 1 – Esquema de un motor PMSM trifásico [1]

La figura a continuación (Ilustración 2) presenta el diagrama del control de un motor PMSM. Esta técnica de control se llama *Field Oriented Control* (FOC).

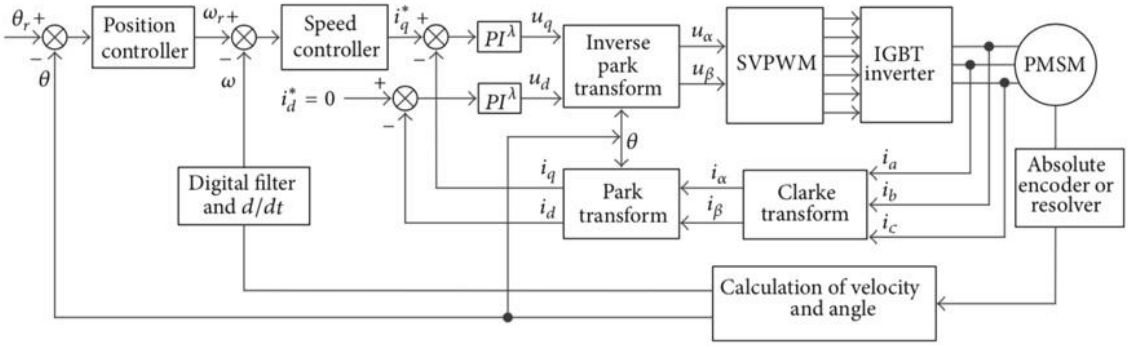


Ilustración 2 – Diagrama de bloques del control de un motor PMSM [2]

El funcionamiento se puede resumir en las siguientes etapas:

- A partir de la posición del rotor, el software diseñado tiene que ser capaz de calcular la velocidad y el ángulo,
- En el mismo instante que se miden esas señales, se miden las corrientes a, b y c del motor (una para cada fase),
- El algoritmo realiza la transformada de Clarke y de Park para obtener las corrientes de referencia i_d e i_q,
- Comparando la demanda de velocidad del piloto (a través del puño acelerador-potenciómetro) con la velocidad medida anteriormente, se calcula con el control de velocidad la demanda de corriente en el eje q,
- Se comparan las corrientes de referencia del eje q (demanda de velocidad del piloto) y d (que vale 0) con las corrientes medidas,
- Se pasan a un control PI para generar las tensiones de referencia en ejes d y q,
- Se calcula la inversa de la transformada de Park,

- Se calculan los tiempos de activación de los transistores del inversor con el algoritmo SVPWM,
- Se disparan las señales PWM que generan las corrientes necesarias para la demanda del piloto mediante el microcontrolador.

4. Resultados

- El software que realiza el funcionamiento descrito en el apartado anterior se ha desarrollado por completo. Aunque no se haya podido comprobar con el motor, las funciones se han evaluado por separado y realizan los cálculos de forma correcta.
- La simulación del control en Simulink muestra los resultados obtenidos en la aceleración máxima con parámetros estimados y parecidos a los del motor de la competición (en azul se muestra la demanda y en amarillo la velocidad actual, ambos en RPM). 8000RPM son 200km/h, por lo que la motocicleta alcanza 100km/h (4000RPM) en a penas 4.2 segundos.

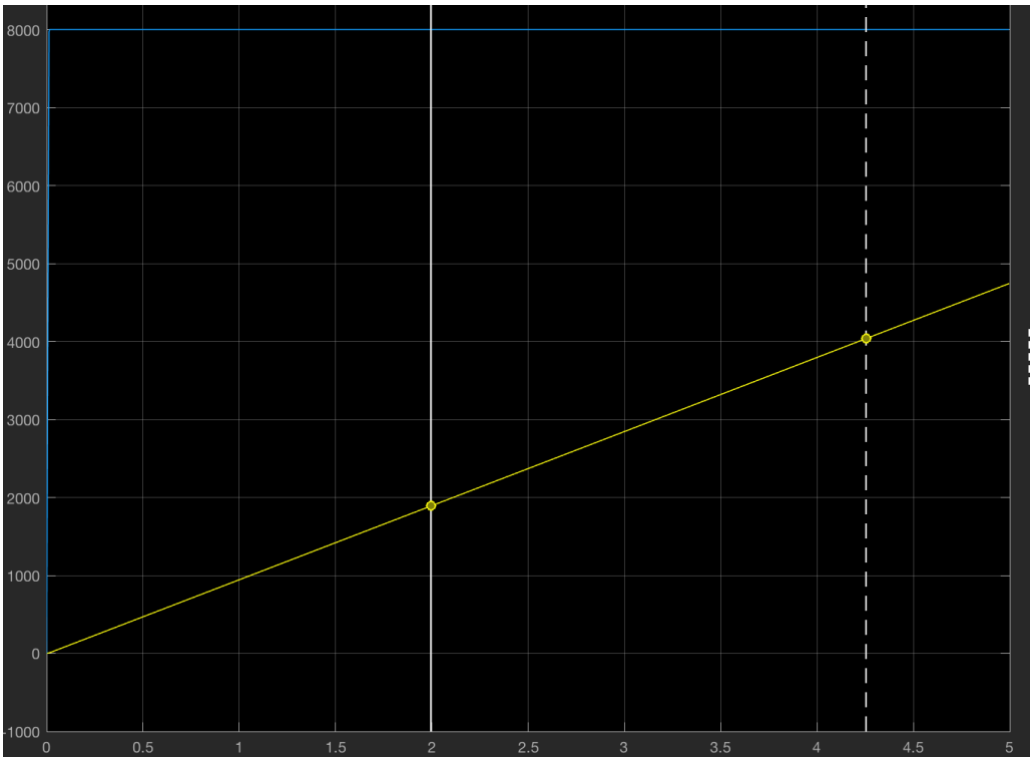


Ilustración 3 – Aceleración máxima de la motocicleta

- Los componentes (inversor, driver, microcontrolador, sensores de corrientes, etc) se han escogido de forma justificada y cumplen con los requisitos de la competición Motostudent. No se ha podido fabricar por problemas logísticos.

5. Conclusiones

Este proyecto ha diseñado un controlador para motocicleta eléctrica (elección del hardware, realización del software, simulación del control, algoritmos de control) aunque no haya podido fabricarse el dispositivo por problemas logísticos.

Ofrece una solución escalable y económica para el desarrollo del controlador, puesto que los componentes son escogidos individualmente se asegura que correspondan exactamente con lo exigido y no se sobrepasen los requisitos, abaratando los costes. El software de control, al ser diseñado, ofrece un control granular sobre el comportamiento de la motocicleta, permitiendo tomar en cuenta cualquier decisión en función de los posibles sensores de los que disponga la motocicleta. En el momento en el que se realizó este trabajo, tan solo disponía de sensores básicos de temperatura para proteger el motor, pero se estaba investigando la opción de incorporar un control *anti-wheelie* (anti caballitos) en la próxima edición o incluso control de tracción.

6. Referencias

[1] **D. C. Hanselman.** "Brushless Permanent Magnet Motor Design, 2ed", Magna Physics Publishing, 2006.

[2] **Zixu Jiang.** "Sliding Mode Disturbance Observer-Based Fractional Second-Order Nonsingular Terminal Sliding Mode Control for PMSM Position Regulation System", Mathematical Problems in Engineering, 2015.

DEVELOPMENT OF A CONTROLLER FOR ELECTRIC MOTORCYCLE

Author: Guizien Martin, Victor Antonio.
Director: Muñoz Frías, José Daniel.
Collaborative entity: ICAI – Universidad Pontificia Comillas.

SUMMARY OF PROJECT

This project explains the steps prior to developing a controller for an electric motorcycle. In the first place, it shortly explains both the electric motorcycle as a whole and the permanent magnet motor it uses. Then, the project focuses on the controller, its operation and its structure, and thus it also details the choice of the components that form it. Finally, the control of a permanent magnet motor is simulated and the code that has been developed is detailed.

Palabras clave: PMSM, FOC, SVPWM.

1. Introduction

Due to the growing demand for electric motors, the need for efficient controllers has also skyrocketed and the automotive case is no exception.

The market for electric motorcycles is still young, and it has not been until this year that the ICAI Speed Club has started to develop its first electric motorcycle.

2. Project definition

The objective of this project is to develop a controller that is more economical than a commercial controller, allowing at the same time a more granular control of the parameters managed by the software, since it is developed in its entirety.

For this, first it will be necessary to design the power circuit, which consists of an inverter, a radiator and a driver. Next, it is necessary to design the control circuit, formed by a microcontroller. The next step is to simulate the control with the help of Simulink, to verify that the currents or limit voltages are not exceeded. Finally, the control software is designed.

3. Description of permanent magnet synchronous motor and its control

A PMSM (Permanent Magnet Synchronous Motor) motor consists of a moving part (rotor) and a static part (stator). The stator is composed of 3 windings that are housed in such a way that they create a certain number of pole pairs.

The rotor is formed by a permanent magnet and has the same number of pairs of poles as those formed by the stator.

The operation of the motor is based on the interaction between two magnetic fields. By circulating the currents through the windings, they generate a magnetic field that interacts with the rotor, achieving motion.

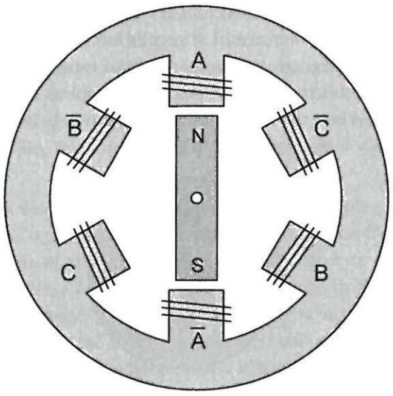


Illustration 1 – Three phased PMSM [1]

The next figure (Illustration 2) depicts the block diagrams of a PMSM control. This control technique is called Field Oriented Control (FOC for short).

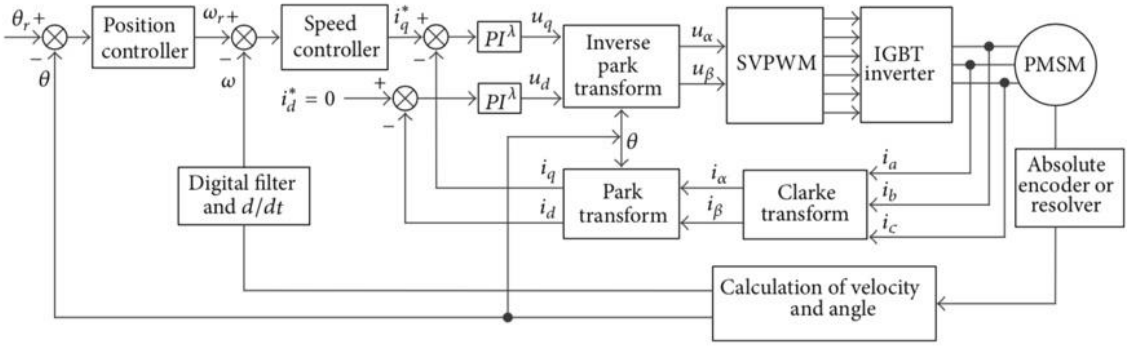


Illustration 2 – Block diagram of a PMSM control [2]

The operation can be summarized in the following steps:

- From the position of the rotor, the software has to be able to calculate the angle and the current speed,
- At the same moment where these signals are measured, the currents a, b and c (one for each phase) are measured,
- The algorithm performs Park's and Clarke's transformation to obtain reference currents i_d and i_q,
- Comparing the speed demanded by the pilot (through the throttle-potentiometer grip) with the previously measured speed, the speed control calculates the resulting demanded q axis current,
- q axis current demanded is compared with reference q axis measured and the same is done with d axis currents, except the reference in normal conditions is set to 0.

- The result of both comparations are sent to individual PI current controllers to obtain reference voltages in d and q axis.
- Park's inverse is calculated
- The activation times of the inverter transistors are calculated with software and SVPWM algorithm
- PWM signals that generate the currents necessary for the pilot's speed demand are triggered by the microcontroller.

4. Results

- The software that performs the operations described in the previous section has been fully developed. Although it was not possible to check its functionality with the motor itself, the functions have been evaluated separately and the results they yield are correct.
- Simulation of the control loop in Simulink shows the results obtained in the case of maximum acceleration with estimated and similar parameters to those of the actual PMSM used (the blue line indicates the pilot's demand and the yellow the current speed, both in RPM). 8000RPM equates to 200km/h, so the motorcycle reaches 100km/h (4000RPM) in just 4.2 seconds.

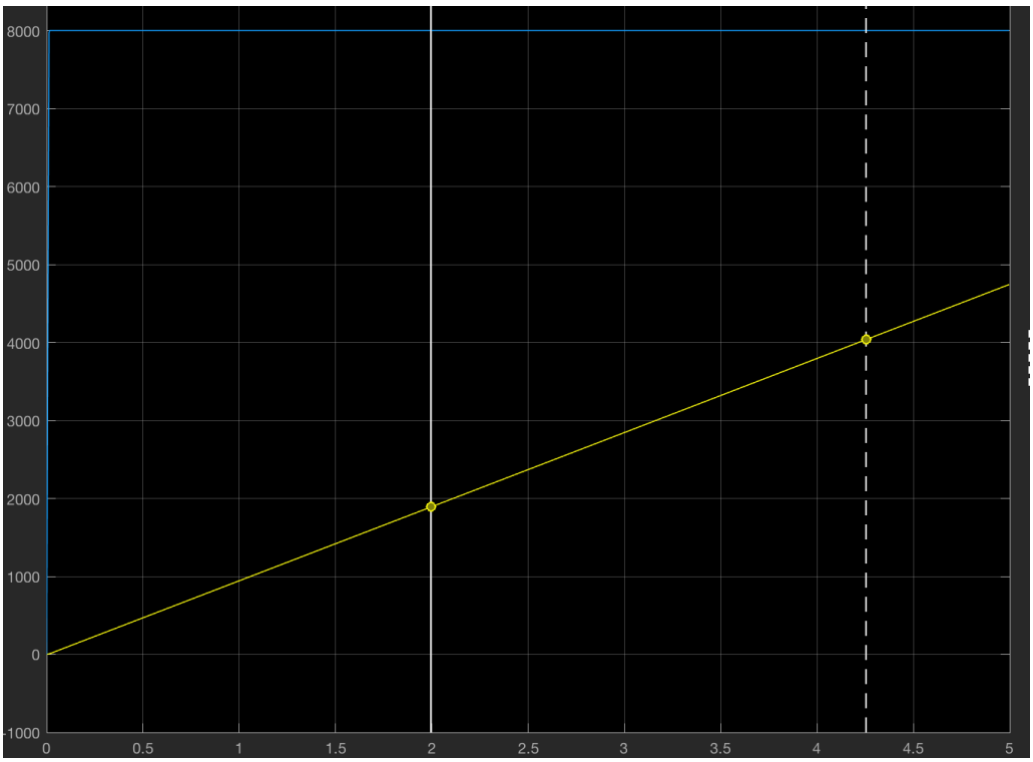


Illustration 3 – Maximum acceleration of the motorcycle

- The components (inverter, driver, microcontroller, current sensors, etc.) have been chosen in a justified manner and meet the requirements of the Motostudent competition. The controller itself could not be manufactured due to logistical problems.

5. Conclusions

This project has designed a controller for an electric motorcycle in its entirety (choice of hardware, software realization, control loop simulation, control algorithms explained, etc), even though the controller itself could not be manufactured due to logistical problems.

It offers a scalable and economical solution for the development of the controller, since the components are chosen individually, ensuring that they correspond exactly to the mandated requirements, and that these requirements are not exceeded, reducing costs. Because the control software has been designed from the ground up, it allows for more granular control over the behavior of the motorcycle, allowing it to take into account any decision based on the sensors incorporated in the motorcycle. At the time this work was done, only basic temperature sensors were set in the motorcycle to protect the engine, but the option of incorporating an anti-wheelie control in the next edition or even traction control was being investigated.

6. References

[1] **D. C. Hanselman.** "Brushless Permanent Magnet Motor Design, 2ed", Magna Physics Publishing, 2006.

[2] **Zixu Jiang.** "Sliding Mode Disturbance Observer-Based Fractional Second-Order Nonsingular Terminal Sliding Mode Control for PMSM Position Regulation System", Mathematical Problems in Engineering, 2015.

Índice de la memoria

Índice de figuras – Memoria

Índice de tablas – Memoria

Índice figuras – Anexo A

Índice figuras – Anexo B

Índice figuras – Anexo C

Capítulo 1. INTRODUCCIÓN

En este capítulo se estudia la situación actual de los controladores para motores eléctricos, en especial de aquellos destinados a vehículos eléctricos, y dentro de estos, a las motocicletas eléctricas. Además, se presentan los distintos tipos de control que existen para un motor de imanes permanentes.

Para entender la necesidad de los controladores, es necesario poner en contexto el auge de los motores eléctricos, explicar muy brevemente ciertas técnicas de control y por lo tanto hablar de motores. Los términos técnicos y relacionados al funcionamiento del controlador se explican en sus respectivos capítulos más adelante.

A partir de este análisis, se exponen las razones que han llevado a la creación de un prototipo nuevo de controlador, así como los objetivos que se persiguen.

1.1 ESTADO DEL ARTE

Los controladores son necesarios para controlar cualquier motor eléctrico. En los últimos años, la demanda de motores eléctricos ha incrementado notablemente, como se destaca en [1] y [2]. Esta subida se explica principalmente por el uso tan variado que tienen los motores eléctricos. Se usan en ascensores, ventiladores, frigoríficos, compresores, vehículos, máquinas industriales, equipos médicos y muchos otros. Se caracterizan por su aguante (a tensiones que puedan fluctuar), su bajo mantenimiento, su bajo consumo eléctrico y una larga vida útil.

La subida del precio del petróleo y la necesidad de reducir la contaminación son otros factores que generan una preferencia considerable a la hora de elegir el tipo de motor que se va a utilizar, sobre todo en el sector del automóvil.

En los últimos años, la venta de coches eléctricos se ha multiplicado [3], incrementando a su vez la demanda de controladores. Conforme sube esta demanda, el precio de la electricidad también lo hace, lo que promueve el desarrollo de motores más eficientes y la mejora y optimización de técnicas de control [2].

1.1.1 TÉCNICAS DE CONTROL PARA MOTORES DE IMANES PERMANENTES

1.1.1.1 Control senoidal

La técnica de control más común para un motor de imanes permanentes (PMSM – *Permanent Magnet Synchronous Motor*) es la del control senoidal [4] (Figura 1). Cuando el motor está funcionando, se resta a la velocidad de referencia (la velocidad deseada) la velocidad medida, y el error resultante se procesa en un control PID para generar la amplitud de la señal del seno.

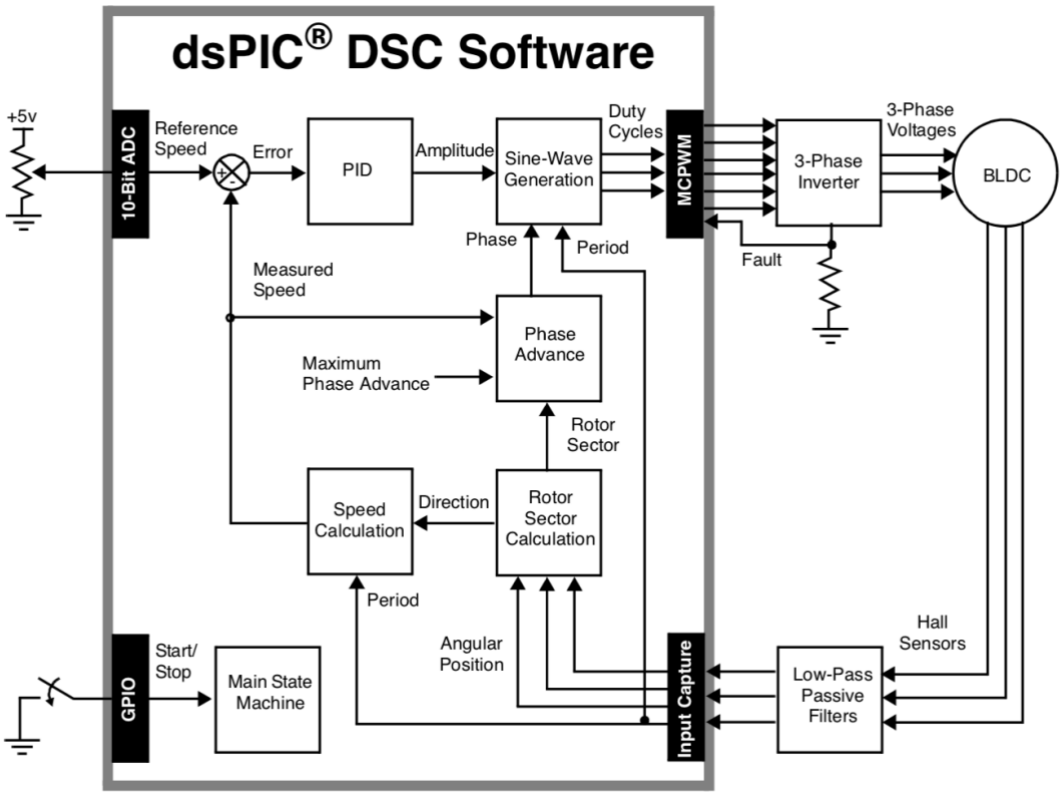


Figura 1 – Esquema de bloques de un control senoidal [5]

Una vez se tenga la amplitud – la variable "Amplitude", hacen falta otros dos parámetros para generar la señal del seno. El primero es la fase, que se calcula en función de la velocidad medida y de la posición del rotor. El otro parámetro necesario es el período, que se toma de uno de los sensores de efecto Hall.

Finalmente se generan las señales PWM para conseguir que el inversor genere los voltajes adecuados en el motor.

1.1.1.2 Control FOC (Field Oriented Control)

Esta técnica se ha popularizado en los últimos años debido a que su coste de implementación ha disminuido de forma drástica. La tecnología disponible hoy en día permite que se implemente esta técnica en una máquina de 16 bits y de coma fija, como son los DSCs (*Digital Signal Controllers*) de la familia de dsPIC, de Microchip.

La idea básica de este algoritmo es descomponer la corriente del estator en dos partes:

- La que genera el campo magnético (corriente d)
- La que genera el par (corriente q)

Una vez descompuesta, dichas componentes se pueden controlar de forma separada, de forma que el control del motor se simplifica considerablemente.

Esta técnica es la que se usa para este trabajo, y por lo tanto será explicada en mucho más detalle más adelante.

1.1.2 CONTROL DE MOTOR BASADO EN SENSORES VS. SIN SENSORES

Cualquier tipo de control de motor necesita saber la posición del rotor para poder aplicar las corrientes adecuadas en el momento adecuado. Existen distintos métodos para obtener la posición del rotor. Algunos se basan en las medidas proporcionadas por sensores tradicionales, mientras que otros lo hacen sin el uso de estos últimos.

1.1.3 TÉCNICAS DE CONTROL BASADAS EN LECTURA POR SENSORES

Las técnicas de control basados en lectura por sensores se apoyan en sensores electromecánicos incorporados en el rotor, que proporcionan información de velocidad y posición (del rotor). Los sensores más comunes para conseguir dicha información son los sensores de efecto Hall, los "resolvers" y los "encoders". Los motores BLDCs (*Brushless DC*) y PMSM suelen operar con uno o más de estos sensores, ya que la excitación eléctrica tiene que ser síncrona con la posición del rotor.

La opción más barata suele ser la de los sensores de efecto Hall. Tres sensores de este tipo, colocados en el rotor, proporcionan información cada vez que el motor rota 60º. Este tipo de sensores ofrece un control adecuado, aunque no ideal.

Para aplicaciones en las que la precisión es vital, la mejor opción es la del resolver, aunque también es la más cara, ya que está internamente montada en el motor. Gracias a su alta resolución (1024 estados o más por revolución) y a su capacidad para conocer la posición absoluta del ángulo, es la opción preferida para servo-aplicaciones industriales.

En el caso de los encoders, la resolución también es bastante alta (unos 512 estados por revolución), aunque no se puede conocer la posición absoluta al encenderse el motor y también suelen ser caros. Debido a su resolución, también se usan en servo-aplicaciones.

La Figura 2 muestra la forma de onda de los sensores mencionados anteriormente.

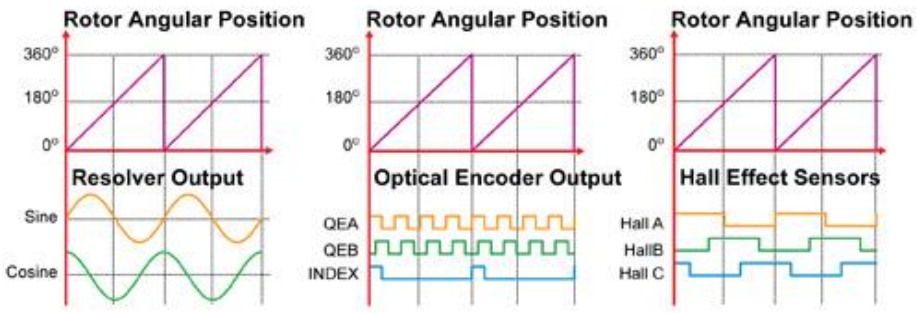


Figura 2 – Forma de onda de resolver, encoder y sensores de efecto Hall [4]

1.1.4 TÉCNICAS DE CONTROL SIN SENSORES

Con el fin de ahorrar costes y espacio, distintas técnicas de control sin sensores de posición han surgido en los últimos años. Entre ellas, destacan sobre todo las siguientes:

- Técnicas basadas en la estimación de la fuerza contra-electromotriz o BEMF (*back EMF, back electromagnetic force*)
- Técnicas basadas en observadores de estado y filtros Kalman extendidos (EKF)
- Otras técnicas basadas en modelar el motor el tiempo real

Este tipo de control permite abaratar los costes, mejorar el consumo eléctrico y mejorar la eficiencia del motor, como se explica en amplio detalle en [6]

1.1.5 CONTROLADORES COMERCIALES

Los controladores comerciales ofrecen una solución sencilla y rápida para controlar un motor PMSM. Además, suelen venir en un conjunto compacto y en general, bien diseñado. Esto implica que dicho controlador ofrece un buen rendimiento y una alta fiabilidad. Asimismo, la mayoría de las compañías que venden estos aparatos ofrecen un software de control que permite alterar ciertos parámetros del controlador, como la corriente máxima permitida o lo rápida que debe ser la respuesta a la entrada.

Sin embargo, los controladores suelen ser muy caros y carecen de ningún tipo de personalización. Además, se venden por categorías de intensidad y voltaje soportados, que no siempre de adecúan a los requisitos del proyecto; es decir, si como requisitos se tiene que el controlador tiene que soportar $220V$ y $300A$, puede que el controlador que valga sea uno que soporte hasta $600A$, ya que el modelo inferior no vale. Esto no resulta óptimo económicamente hablando. Por otro lado, un controlador "hecho en casa" que soporte las corrientes establecidas (más un cierto margen), también permitiría ahorrar espacio, ya que los componentes internos serían más pequeños.

1.2 MOTIVACIÓN

Este trabajo de fin de grado tiene como objetivo diseñar un controlador que sea personalizable y más económico que un controlador comercial.

Al ser diseñado desde cero, este controlador se adapta perfectamente a los requisitos establecidos (detallados más adelante), permite un control granular de las acciones que realiza (aceleración, freno regenerativo, etc) y permite abaratar los costes de forma considerable.

Finalmente, se podrá reutilizar en el futuro, ofreciendo la posibilidad de mejorar tanto el software como el hardware del dispositivo.

1.3 OBJETIVOS

1.3.1 DESARROLLAR EL CIRCUITO DE POTENCIA

La primera etapa consiste en diseñar el circuito físico (hardware) del controlador. Se compone principalmente de un inversor formado por 6 transistores IGBT's, un radiador para disipar el calor y un driver para controlar las señales y disparar los PWM. Más adelante se detallan los parámetros tomados en cuenta, así como la justificación de cada componente.

1.3.2 DESARROLLAR EL CIRCUITO DE CONTROL

Una vez esté diseñado el circuito de potencia y los componentes estén elegidos, es necesario desarrollar el circuito de control electrónico. El circuito de control se compone de un microprocesador encargado de controlar el controlador. En función de ciertos parámetros de entrada (demanda de velocidad del piloto, corriente actual, etc), el microprocesador debe enviar señales PWM adecuadas para conseguir el par de velocidad deseado.

1.3.3 SOFTWARE DE CONTROL

El software es el encargado de asegurar el correcto funcionamiento del controlador. Este código decide la respuesta que se obtiene en función de la entrada, y es crucial tanto en el rendimiento como en la seguridad del piloto. Tendrá que realizar una serie de cálculos en un tiempo determinado para ofrecer en cada instante las señales de control del inversor correspondientes. También ofrecerá distintos modos de operación, ofreciendo distintas pendientes de aceleración en función del modo seleccionado, así como el uso de freno regenerativo para emular el freno motor de las motocicletas tradicionales y permitir una leve regeneración de la batería.

1.3.4 SIMULACIÓN E INVESTIGACIÓN

Por problemas logísticos, el material hardware necesario para el controlador no llegará a tiempo y no sé podrá fabricar. Por este motivo, el proyecto hace hincapié las fases previas al diseño del controlador. Es decir, la elección de componentes, la explicación de las técnicas utilizadas y el desarrollo del software, incluyendo la explicación detallada del microcontrolador y de sus distintos registros.

1.3.5 OBJETIVO FINAL

Este trabajo tiene como objetivo final diseñar un controlador capaz de controlar la motocicleta de competición diseñada por el ICAI Speed Club. Al no poder fabricarse, el presente trabajo hace hincapié en sentar el máximo número de conceptos e ideas para el desarrollo de dicho controlador. Para ello, los distintos capítulos que contiene se han explicado con abundante detalle.

Capítulo 2. LA MOTOCICLETA ELÉCTRICA

En este capítulo se explica brevemente los componentes principales de una motocicleta eléctrica. Es importante conocer el funcionamiento básico de estos componentes ya que son los que se comunican con el controlador.

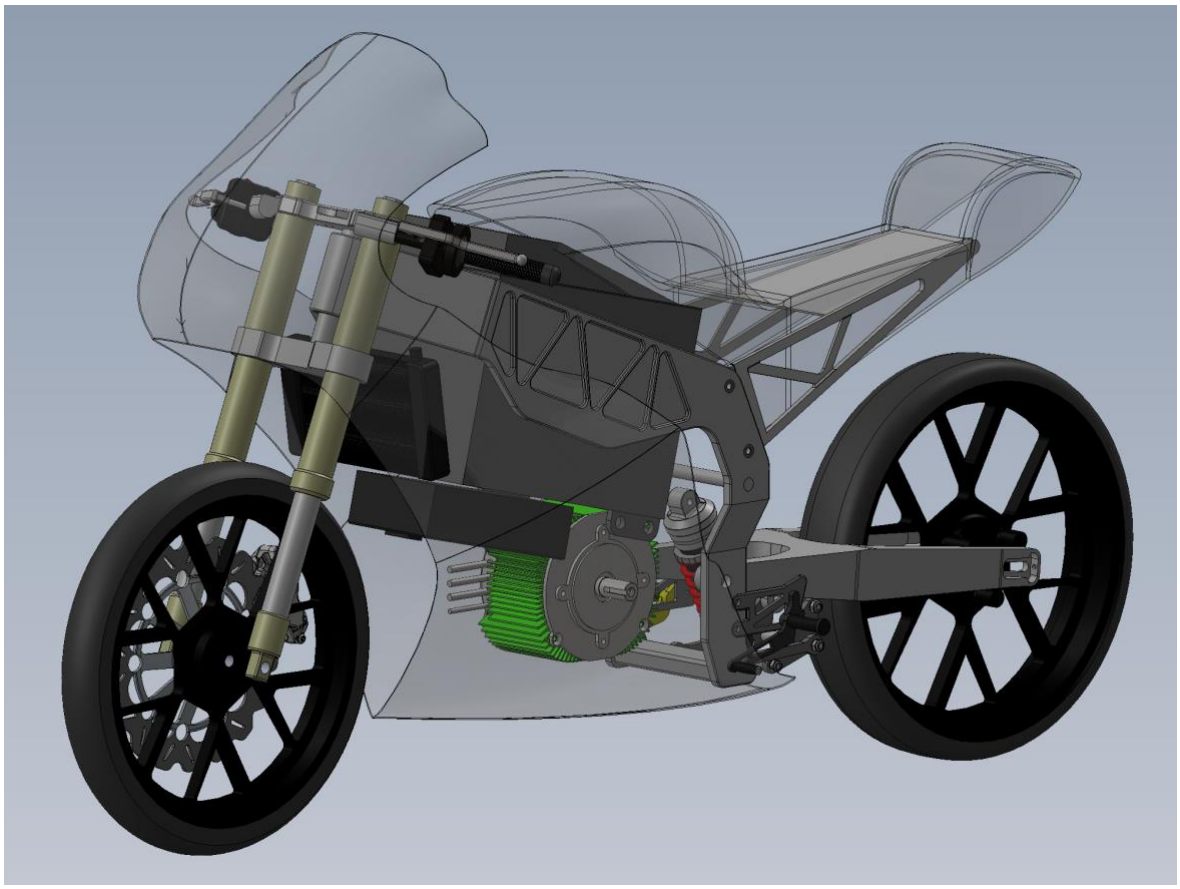


Figura 3 – Modelo 3D – Ensamblaje completo de la motocicleta eléctrica del ISC

2.1 COMPONENTES PRINCIPALES

2.1.1 BATERÍA

El principal componente de la motocicleta eléctrica o de cualquier vehículo eléctrico es la batería. Hoy en día suelen ser de ion de litio, aunque algunos modelos usan baterías híbridas de níquel.

El motor tiene una potencia nominal de $12kW$, y máxima de $20kW$(Figura 4). La batería debe ser suficientemente grande para satisfacer esa potencia durante el tiempo de la carrera.

ME-MS1718 TECHNICAL SPECS

TECHNICAL SPECS

Type	RFPM Electric Motor (Brushless)
Operation Voltage	96-116 VDC
RPM Max.	8.000 rpm
Rated power	12 kW
Peak power	20 kW
Peak torque	65 Nm
Cooling	Air cooling system
Weight	21,4 kg

Figura 4 – Características del motor suministrado por la competición

También se tienen que tener en cuenta consideraciones de diseño y de seguridad. Las consideraciones de diseño tienen que ver con el espacio que ocupan las celdas y el circuito eléctrico que las conecta entre sí. En cuanto a la seguridad, al tratarse de baterías de ion de litio, existe un riesgo de que puedan explotar en el caso de hacerse un mal uso. Por ello hay que atenerse a un diseño correcto y dentro de las normas y recomendaciones de la competición.

Las celdas elegidas son las "XALT Energy 75 Ah (HE)", modelo "F910-0004". Su hoja de características se encuentra en [32].

Al tratarse de celdas de $3.7V$, hacen falta un total de 26 celdas colocadas en serie.

$$\frac{96V}{3.7V} = 26$$

El motor funciona con corriente y tensiones alternas, ya que se trata de un motor trifásico. El voltaje que dan las baterías ($96V$) es una tensión DC, y por lo tanto hay que estimar cuánto sería en AC. Además, la relación entre potencia, intensidad y tensión[1] viene dada por:

$$P_{nom} = 3 \cdot V_{AC_{nom}} \cdot I_{nom}$$

$$I_{nom} = \frac{P_{nom}}{3 \cdot V_{AC_{nom}}} \quad (1)$$

De (1), podemos deducir que cuanto más pequeño V_{AC} más grande será el valor de la corriente.

Típicamente ([36]), el valor de la tensión en AC se puede expresar por:

$$V_{AC_{nom}} = \frac{V_{DC} \cdot 0.61}{\sqrt{3}}$$

Por lo tanto:

$$V_{AC_{nom}} = \frac{96 \cdot 0.61}{\sqrt{3}}$$

$$V_{AC_{nom}} = 33.8V \quad (2)$$

Esta tensión parece un poco baja, y como se verá más adelante en el Capítulo 5. , no es suficiente para alcanzar las vueltas por minuto del motor. Esto se debe probablemente a algún problema con los parámetros del motor dados por la competición.

[1] $V_{AC_{nom}}$ se referirá a la tensión de alterna nominal **simple**.

El valor máximo (valor pico-pico) es:

$$V_{AC_{peak}} = V_{AC_{nom}} \cdot \sqrt{2}$$

$$V_{AC_{max}} = 47.814V \quad (3)$$

Reemplazando en (1) con el valor de (2):

$$I_{nom} = \frac{12000}{3 \cdot 33.8}$$

$$I_{nom} = 118.34A \quad (4)$$

En el caso más desfavorable:

$$I_{max} = \frac{20000}{3 \cdot 33.8}$$

$$I_{max} = 197.23A \quad (5)$$

Las celdas tienen una capacidad de $75Ah$. Para calcular cuanto tiempo aguantarían las baterías hay que calcular lo siguiente:

$$\frac{75Ah}{197.23A} = 0.38h = 22.84min$$

Puesto que la carrera dura aproximadamente 15 minutos y que la motocicleta nunca va a estar en el caso más desfavorable **toda** la carrera, hay tiempo de sobra para acabar la competición.

También hay tiene que tener en cuenta que otros dispositivos como el display, el controlador, los diferentes sensores y otros dispositivos electrónicos consumen energía, que, aunque parezca despreciable individualmente, al final puede resultar una suma importante. De todas formas y como se ha dicho antes, hay energía de sobra para la carrera.

La competición obliga el uso de un BMS ("*Battery Management System*"), que deberá leer la tensión de cada celda y la temperatura de las celdas en su punto más caliente. Por lo tanto, el controlador recibe directamente el valor de la tensión de la batería leída por el BMS. En este trabajo se supondrá que la medida de la batería viene directamente por medio de una tensión adaptada entre $0V$ y $3.3V$ a un pin del módulo ADC del microcontrolador.

Las siguientes figuras (Figura 5 y Figura 6) ilustran las 26 celdas elegidas y su contenedor.

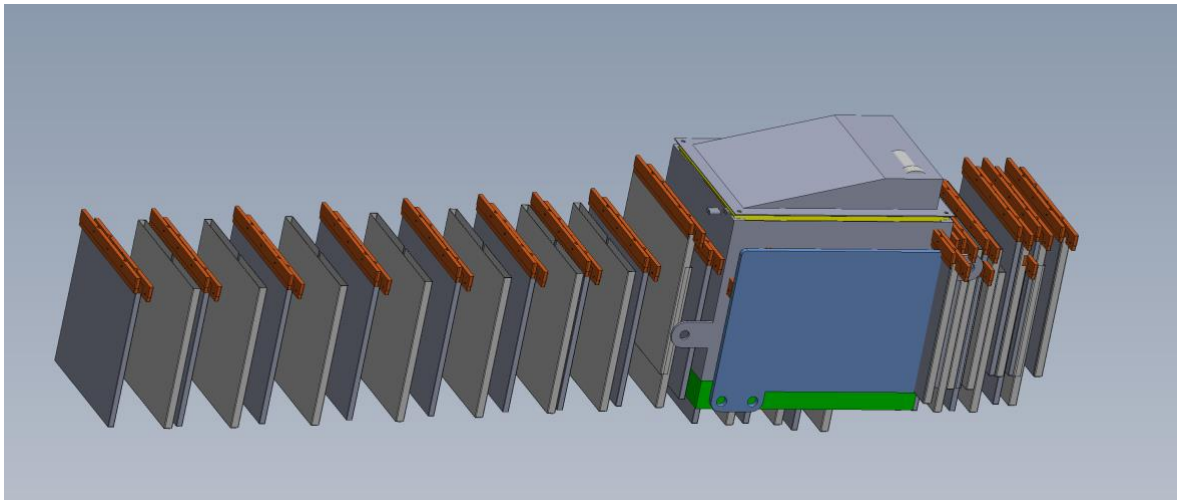


Figura 5 – Modelo 3D – Celdas y contenedor de las celdas

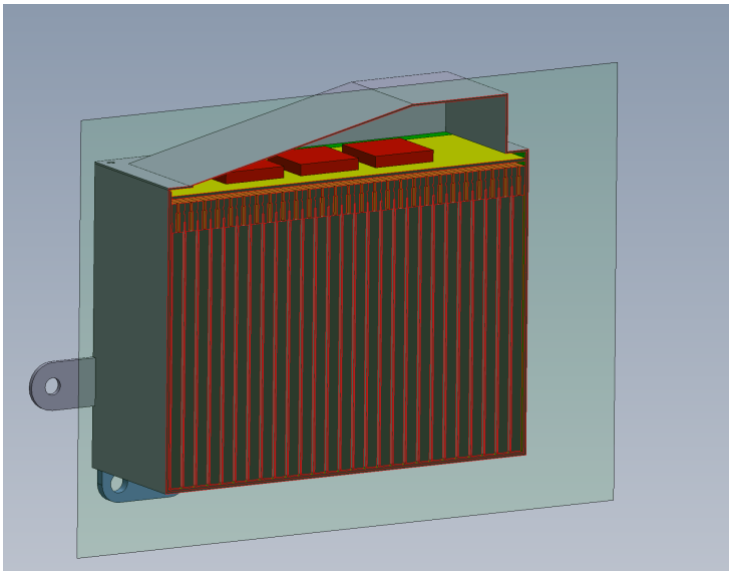


Figura 6 – Modelo 3D – Sección del contenedor de las celdas

2.1.2 FRENOS

2.1.2.1 Freno tradicional

Al frenar, la energía cinética se convierte en calor. Este proceso tiene una perfecta armonía con el principio de conservación de la energía. Así, la moto disipa parte de su energía cinética al frenar en forma de calor gracias a la fricción provocada entre el disco de freno y las pastillas de freno (Figura 7):

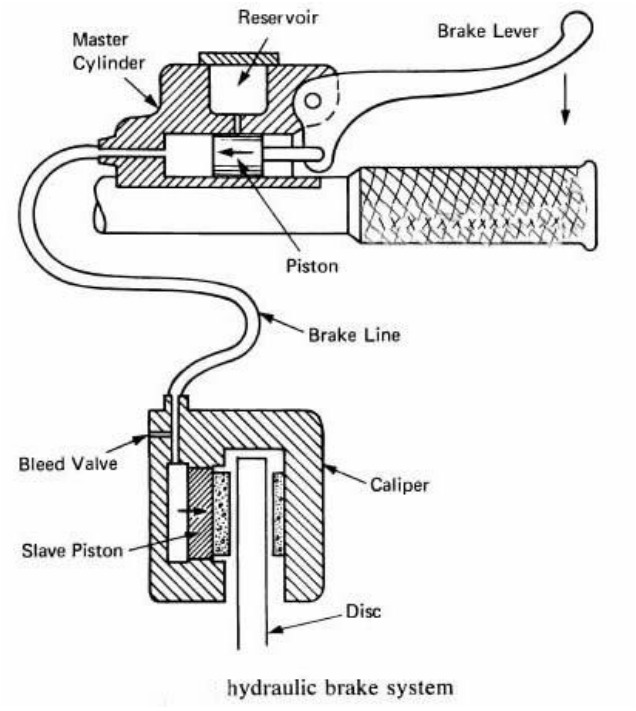


Figura 7 – Freno hidráulico convencional de una motocicleta [10]

Esto tiene varios problemas:

- De alguna forma hay que encargarse del calor disipado en el sistema de frenado,
- Toda la energía de frenado es malgastada, ya que simplemente se disipa el calor en la atmósfera,
- Se desgastan las pastillas de freno.

Hay que destacar también que en una motocicleta existen dos frenos: el freno delantero y el freno trasero. El freno delantero se maneja mediante una maneta, localizada en frente del puño acelerador derecho. El freno trasero se activa con el pie derecho, mediante una palanca localizada en frente de la estribera derecha Esta información es relevante para la incorporación del freno regenerativo, la cual se detalla más adelante.

2.1.2.2 Freno eléctrico

Además de los frenos de disco tradicionales, que actúan independientemente en la rueda delantera (con la maneta derecha) y en la rueda trasera (con el pedal actuado con el pie derecho), las motocicletas eléctricas (y cualquier vehículo eléctrico) tienen la opción de incorporar el freno regenerativo.

Hay distintas formas de incorporar dicha funcionalidad. Hoy en día, distintos coches eléctricos ya lo incorporan, y cada uno lo hace de la forma que mejor le parezca. Por ejemplo, en el BMW i3, nada más levantar el pie del acelerador, el vehículo entra en estado de freno regenerativo. Otros vehículos entran en este estado nada más empezar a pisar el pedal del freno. Finalmente, unos pocos ofrecen la posibilidad de decidir si se quiere que haya freno regenerativo al levantar el acelerador, al pisar el freno o en ambos. Todo este proceso es controlado por el software del controlador.

El funcionamiento del freno regenerativo se explica con mucho más detalle más adelante.

2.1.3 MOTOR

El último componente más importante y que más difiere de una motocicleta tradicional es el motor. El motor es el encargado de transmitir la fuerza eléctrica en fuerza mecánica a través de una cadena.

El motor que utiliza en la competición es explicado en detalle en el Capítulo 3. , por lo que no vale la pena explicar su funcionalidad aquí.

Lo único que hay que destacar es que la motocicleta tiene que llevar una reductora. Una reductora es un sistema que permite reducir las vueltas que da el motor para que las ruedas giren a una velocidad menor.

En la fecha en la que se realizó este trabajo, todavía no se había escogido una reductora, pero a través de los siguientes cálculos se puede estimar.

Supongamos que la velocidad máxima de la motocicleta es de $200\frac{Km}{h}$ (igual que antes, esto no se ha podido comprobar ya que depende de los rozamientos con el aire – y por lo tanto del ensamblaje completo – y de otros factores.).

Esto significaría que cuando el motor alcance sus vueltas por minuto máximas, la motocicleta debería ir a esa velocidad. En la hoja de características del motor (Hoja de características del motor de la competición), se puede ver que el motor puede funcionar a $8000rpm$ máximo.

A $200\frac{Km}{h}$, la motocicleta recorre 3333 metros en 1 minuto. Por lo tanto, la rueda trasera (la que recibe la fuerza mecánica) tendría dar suficientes vueltas en 1 minuto como para recorrer dicha distancia:

$$\frac{3333}{2 \cdot \pi \cdot R_{BackWheel}} \quad (6)$$

$R_{BackWheel}$ vale $\frac{576}{2} mm$ (3.3.1) por lo que:

$$\frac{3333}{2 \cdot \pi \cdot \frac{0.576}{2}} = 1841rpm \quad (7)$$

Por lo tanto, hay que escoger una reductora de un factor de aproximadamente 4.

$$\frac{8000}{1841} \approx 4$$

2.2 OTROS COMPONENTES

Todos los componentes de la motocicleta son esenciales, pero prácticamente cada uno de ellos podría constituir un TFG por sí mismo. Aunque este trabajo se concentre en el controlador, no hay que olvidar que la motocicleta se compone de decenas de partes cada cual igual de importante. Para asegurar un funcionamiento adecuado de la motocicleta y la seguridad del piloto, el conjunto de las piezas tiene que encajar y funcionar en harmonía.

Un resumen de componentes y pasos a seguir para la fabricación de una motocicleta eléctrica puede encontrarse en [11].

Capítulo 3. EL MOTOR PMSM

En este capítulo se detalla el funcionamiento del motor PMSM que se utiliza para la competición. Es necesario conocer dicho funcionamiento para comprender por qué es necesario utilizar un controlador, y qué funciones desempeña este último.

También se presenta el modelo de un motor PMSM en el programa Simulink, que se utilizará para realizar las simulaciones del circuito de control.

3.1 ESTRUCTURA

La forma que suele tener un motor PMSM es la de un cilindro. Tanto esta forma geométrica como otras, se componen de dos partes principales. Una parte fija estacionaria denominada estator y una parte móvil denominada rotor. La forma más común de estructura es tener el rotor en el interior del estator, aunque existen otras configuraciones. Esta estructura – el rotor al interior del estator – tiene distintas ventajas. Para empezar, permite que el motor sea de más fácil colocación ya que se tiene que juntar la parte fija – estator – y, además, tener la parte móvil dentro del estator le ofrece – al rotor – una protección natural [9].

3.1.1 ESTATOR

El estator de un motor PMSM se compone láminas de acero magnético apiladas, que actúan de soporte para los devanados. La mayoría de motores PMSM tienen los devanados del estator conectados en forma de estrella. Un motor PMSM tiene 3 devanados, y por dentro se alojan (en el estator) de tal forma que crean un número de pares de polos determinado.

3.1.2 ROTOR

El rotor está formado por un imán permanente. El número de polos del rotor depende de cómo y para qué se usa y está relacionado con el par y la velocidad máxima. Es decir, el

número de polos es un compromiso entre el par y la velocidad máxima. A más número de polos, más par puede producir el motor, a cambio de sacrificar velocidad punta [9].

Además, el número de pares de polos del rotor tiene que ser el mismo que en número de par de polos formado por los devanados del estator.

3.2 FUNCIONAMIENTO

El funcionamiento de un motor PMSM se basa en la interacción entre dos campos magnéticos, uno del estator y otro del rotor. Para ello, una corriente se aplica en la dirección apropiada en los devanados del estator como se enseña en la Figura 8.

El campo electromagnético generado en una de las espiras (formadas por el devanado) es opuesto al del rotor, lo que genera una fuerza de atracción. Esta fuerza produce un par, que es lo que hace que el motor se mueva.

La Figura 9 muestra el caso contrario: la corriente circula en sentido opuesto, por lo que las fuerzas se repelen y el rotor se mueve en dirección contraria.

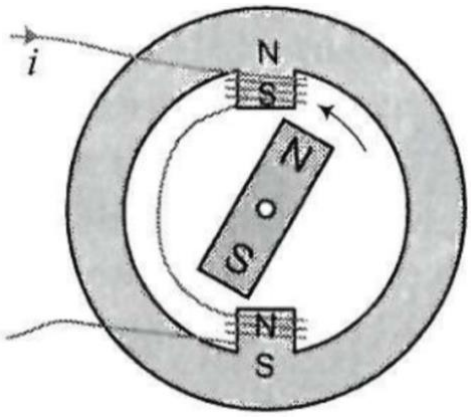


Figura 8 – Corriente aplicada en el devanado para rotación anti-horaria [9]

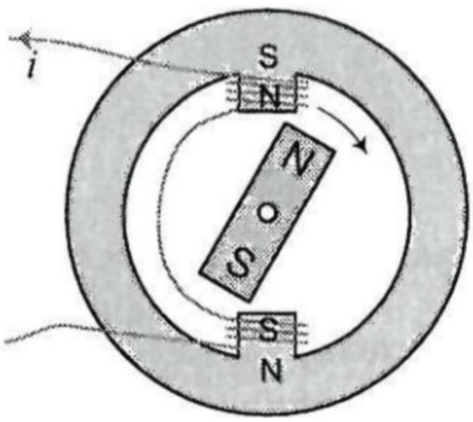


Figura 9 – Corriente aplicada en el devanado para rotación horaria [9]

Para que el rotor permanezca en movimiento y para asegurar que ese movimiento sea suave, es común que un devanado se enrolle en distintos sitios para conseguir más pares de polos, como se muestra en la Figura 10. Para conseguir esa rotación, la corriente que circula por los devanados tiene que formar los campos magnéticos adecuados de forma que atraigan o repulsen a los del rotor.

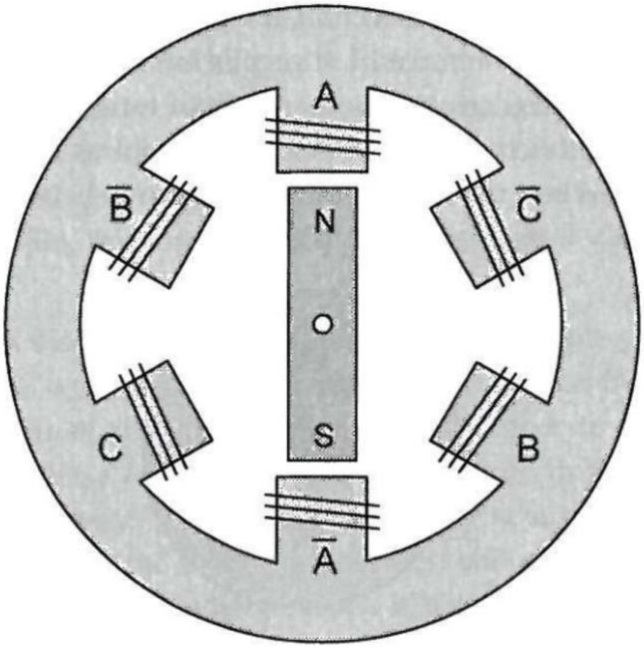


Figura 10 – Esquema de un motor PMSM trifásico [9]

3.3 PARÁMETROS DEL MOTOR

En el Capítulo 5. , se utiliza el software Simulink para simular el control del motor PMSM. Sin embargo, para poder simularlo y como se podrá ver en dicho capítulo, hace falta introducir ciertos parámetros del motor.

Un motor PMSM tiene gran cantidad de parámetros y el estudio, optimización y determinación de estos constituye en sí mismo un trabajo importante [14], [15].

En este caso, interesan aquellos que son necesarios para la simulación, que son los presentados en la siguiente tabla (Tabla 1):

Símbolo	Valor	Nomenclatura en inglés	Nomenclatura en español
Ld	$0.062\ mH$	d-axis inductance	Inductancia del eje d
Lq	$0.011\ mH$	q-axis inductance	Inductancia del eje q
L_0	NA	0-axis inductance	Inductancia del eje 0
Rs	$0.0027\ \Omega$	Stator resistance	Resistencia del estátor
N	5	Number of pole pairs	Número de pares de polos
J	TBD	Moment of inertia	Momento de inercia
PM	TBD	Permanent magnet flux linkage	Enlaces de flujo concatenados

Tabla 1 – Parámetros principales del motor PMSM estudiado

La Figura 11 es un extracto de la ficha técnica del motor proporcionado por la competición. La hoja completa se encuentra en el anexo E (Ficha técnica del motor de la competición).

NOTES:
1.CCW（from shaft）
2.TURNS IS 12 PER PHASE
3.Sensor magnet is to be 2.5mm to 3mm from the sensor mounting face. The minimum amplitude for the sensor is 2 volts AC,peak to peak.
4.Class H insulation system.
5.Motor Phase Resistance is 0.0027Ω
6.Motor inductance at 1000Hz is 0.062~0.110 mH

Figura 11 – Características del motor de la competición

Se observa que la inductancia del motor viene dada en por un rango. El valor más pequeño de este rango representa el valor de L_d y el valor más grande representa el de L_q [35].

La corriente I_0, también llamada corriente homopolar, es igual a cero en sistemas trifásicos equilibrados, como es el caso de este motor, y por lo tanto no influye en el resto de cálculos [22].

A partir de está hoja de características podemos establecer que:

$$L_d = 0.062\ mH$$
$$L_q = 0.011\ mH$$
$$R_s = 0.0027\ \Omega$$

3.3.1 MOMENTO DE INERCIA

El momento de inercia de puede estimar de un motor de prestaciones muy parecidas, de la misma compañía. Sus características se pueden ver en la figura siguiente (Figura 12):

DESCRIPTION

Kelly KLS72601-8080IPS controller at 72V or KLS96601-8080IPS controller at 96V perfectly match with this motor.

Product Description
Output Power of 12 KW Continuous, 30 KW Peak (at 96 volts)
Designed for long life. No brush maintenance. The motor is 92% efficient at voltages between 24 to 96 VDC. Continuous current of 125 amps AC (180 Amps DC into the motor control). This is a 3-phase, Y-connected Permanent Magnet Synchronous Motor with an axial air gap and Sine/Cosine Speed Sensor. It has two stators with a rotor in the center.

1) This is a 4 pole motor (8 magnets).
2) The Phase to Phase winding resistance is 0.013 Ohms.
3) The maximum recommended rotor speed is 5000 RPM.
4) Voltages from 0 to 96 VDC input to the control.
5) Torque constant of 0.15 Nm per Amp
6) The Inductance Phase to Phase is 0.10 Milli-Henry with a 28 turns per phase.
7) Armature Inertia is 45 Kg Cm Squared.
8) Continuous current of 125 Amps AC (180 Amps DC into the motor control).
9) Peak current of 420 Amps AC for 1 minute (600 Amps DC into the motor control).
10) Weight of 35 pounds.
11) Peak Stall Torque if 60 ft lb.
12) This is an Open Frame, Fan Cooled motor.

Figura 12 – Características del motor MOTENERGY ME1115 [16]

Por lo tanto:

$$45\ Kg \cdot Cm^2 = 0.0045\ Kg \cdot m^2$$

Luego:

$$J_{engine} = 0.0045\ Kg \cdot m^2 \quad (8)$$

Sin embargo, lo que se requiere para el simulador es el momento de inercia **total**, y el que se ha estimado es el momento de inercia **del motor**.

La tabla siguiente (Tabla 2) representa las abreviaturas que se van a utilizar en los próximos cálculos:

J_{Eq}	Momento de inercia total o equivalente
J_{Wheels}	Momento de inercia de las ruedas
J_{Engine}	Momento de inercia del motor
ω_R	Velocidad angular de las ruedas
v	Velocidad de la motocicleta
M_{Total}	Masa total del sistema motocicleta + piloto
M_{Bike}	Masa de la motocicleta
M_{Rider}	Masa del piloto
R_{max}	Radio más exterior
J_{Wheels}	Momento de inercia de las ruedas

Tabla 2 – Abreviaturas para ecuaciones (9) a (22)

Se puede relacionar los distintos momentos de inercia por:

$$\frac{1}{2} \cdot J_{Eq} \cdot \omega_R^2 = \frac{1}{2} \cdot M_{Total} \cdot v^2 + \frac{1}{2} \cdot J_{Wheels} \cdot \left(\frac{\omega_R}{4}\right)^2 + \frac{1}{2} \cdot J_{Engine} \cdot \omega_R^2 \quad (9)$$

Por otra parte, se sabe que:

$$v = \omega_R \cdot R$$

De 2.1.3 se sabe que (en teoría) existe una reductora de valor 4. Por lo tanto:

$$v = \frac{\omega_R}{4} \cdot R \quad (10)$$

Por otro lado, R representa el radio más externo posible. La Figura 13 muestra dicho radio.

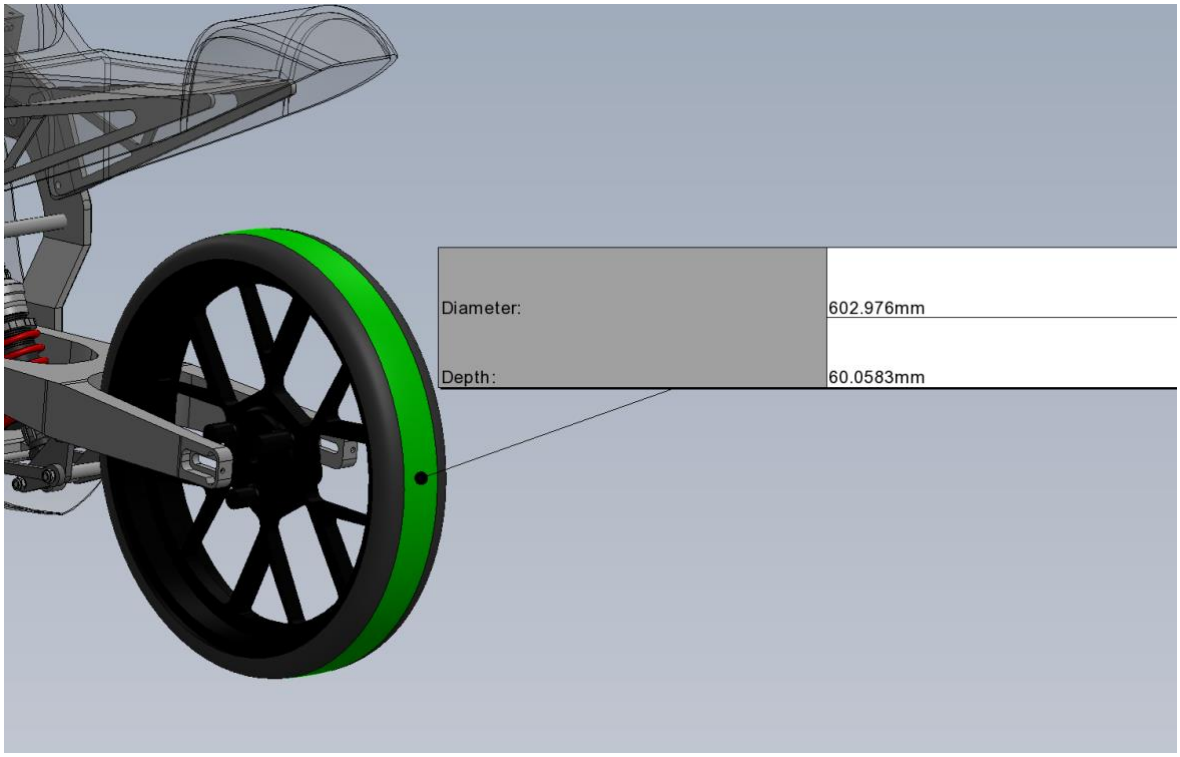


Figura 13 – Diámetro más externo (rueda trasera – 602mm)

Resolviendo (9) y (10) se obtiene:

$$J_{Eq} = (M_{Bike} + M_{Rider}) \cdot \left(\frac{R}{4}\right)^2 + \frac{J_{Wheels}}{4^2} + J_{Engine} \quad (11)$$

Falta por calcular el valor de J_{Wheels}, cuyo valor viene dado por la siguiente expresión:

$$J_{Wheels} = J_{FrontWheel} + J_{BackWheel} \quad (12)$$

$$J_{FrontWheel} = M_{FrontWheel} \cdot R_{FrontWheel}^2 \quad (13)$$

$$J_{BackWheel} = M_{BackWheel} \cdot R_{BackWheel}^2 \quad (14)$$

Para los radios de ambas ruedas se calcula el radio medio entre el radio del neumático y el radio de la llanta. Las siguientes figuras – Figura 14, Figura 15, Figura 16, Figura 17 – indican los radios de llanta y neumático de las ruedas delantera y trasera respectivamente.

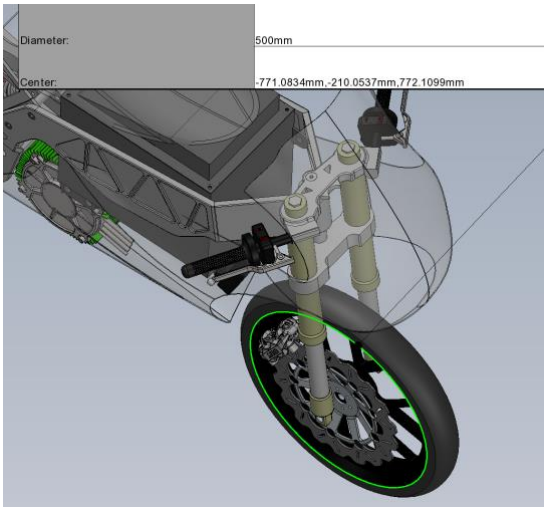


Figura 14 – Radio de la llanta delantera (500mm)

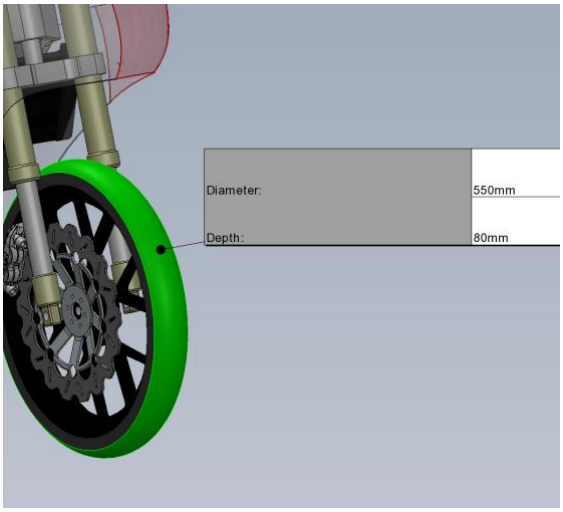


Figura 15 – Radio del neumático delantero(550mm)

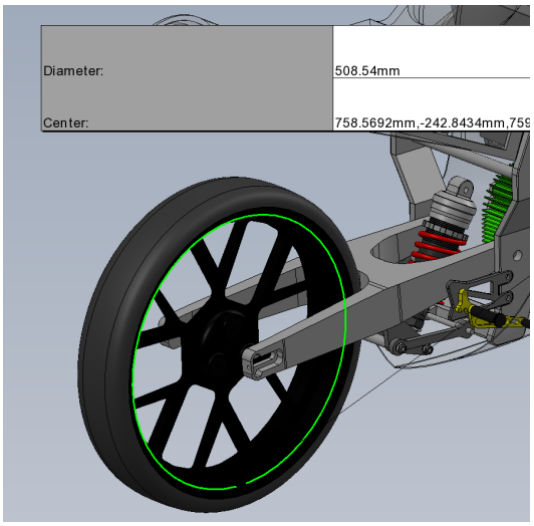


Figura 16 – Radio de llanta trasera (508mm)

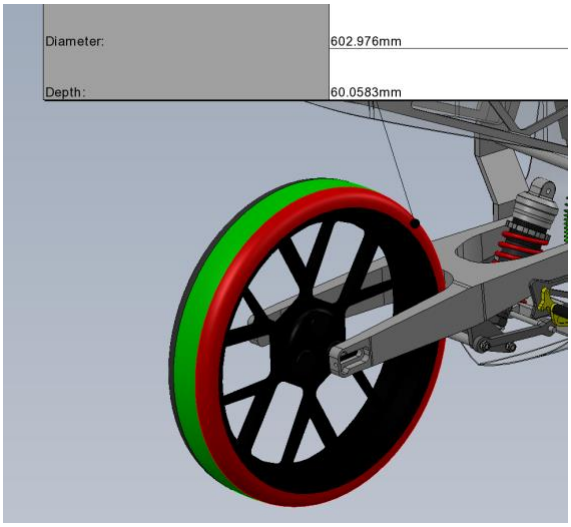


Figura 17 – Radio del neumático trasero(602mm)

Por lo tanto, los radios medios de las ruedas delantera y trasera son, respectivamente:

$$R_{FrontWheel} = \frac{\frac{500}{2} + \frac{508}{2}}{2}$$

$$R_{FrontWheel} = 252\ mm \quad (15)$$

$$R_{BackWheel} = \frac{\frac{550}{2} + \frac{602}{2}}{2}$$

$$R_{BackWheel} = 288\ mm \quad (16)$$

La masa de las ruedas delantera y trasera son a su vez la media entre la masa de la llanta y la masa del neumático:

$$M_{FrontTyre} \quad 3.1Kg$$
$$M_{FrontRim} \quad 6.3Kg$$
$$M_{BackTyre} \quad 3.9Kg$$
$$M_{BackRim} \quad 7.1Kg$$

Por lo tanto, la masa de las ruedas delantera y trasera son:

$$M_{FrontWheel} = \frac{3.1 + 6.3}{2}$$

$$M_{FrontWheel} = 4.7\ Kg \quad (17)$$

$$M_{BackWheel} = \frac{3.9 + 7.1}{2}$$

$$M_{BackWheel} = 5.5\ Kg \quad (18)$$

Resolviendo (13) y (14)

$$J_{FrontWheel} = 0.298g \cdot m^2 \quad (19)$$

$$J_{BackWheel} = 0.456Kg \cdot m^2 \quad (20)$$

Y por lo tanto, de (12) se obtiene:

$$J_{Wheels} = 0.754Kg \cdot m^2 \quad (21)$$

Retomando la ecuación (11) y reemplazando por los valores siguientes:

$$\begin{array}{ll} M_{Bike} & 140Kg \\ M_{Rider} & 75Kg \\ R & 30.1cm \end{array}$$

Se obtiene finalmente la inercia equivalente de la motocicleta:

$$J_{Eq} = 1.261Kg \cdot m^2 \quad (22)$$

3.3.2 POLOS DEL MOTOR PMSM

Como muchos de los otros parámetros, la elección del número de polos del rotor y del estator es objeto de estudio por sí mismo [17], [18]. El número de pares de polos son importantes para la simulación en el Capítulo 5. . Este motor tiene 5 pares de polos.

3.3.3 ENLACES DE FLUJO CONCATENADOS - PM

De la Figura 12 (el punto 5) también puede verse que la constante de par para ese motor es de $0.15 \cdot \frac{N \cdot m}{A}$.

El valor de PM se relaciona con la constante de par de la siguiente forma ([43]):

$$PM = \frac{K_T}{N_p}$$

Donde K_T es la constante de par y N_p es el número de pares de polos.

Por lo tanto,

$$PM = \frac{0.15}{5} = 0.03 \cdot \frac{N \cdot m}{A} \quad (23)$$

Capítulo 4. ARQUITECTURA DEL CONTROLADOR

En este capítulo se explica en detalle la arquitectura del controlador, sus distintos componentes, y por qué se han elegido estos.

En la segunda parte del capítulo se explica de forma detallada el funcionamiento de este.

4.1 COMPONENTES PRINCIPALES

A continuación, se detallan los distintos componentes que constituyen el controlador, así como los distintos cálculos y razonamientos que han llevado a la elección de cada uno de ellos.

4.1.1 MICROCONTROLADOR

El microcontrolador es el cerebro del controlador. Es el encargado de interpretar las entradas y de producir las señales de salida adecuadas.

La Figura 18 presenta el diagrama de bloques de la arquitectura del controlador. Lo primero que hay que destacar es el número de entradas analógicas que tiene que procesar el microcontrolador. Se puede ver como existen **9 entradas** (Tabla 3).

También se puede ver que el microcontrolador tiene que tener **6 salidas PWM** para poder controlar el inversor correctamente.

A partir de estas dos condiciones, sirve cualquier microcontrolador que además de cumplirlas, sea lo suficientemente rápido. Por regla general, un microcontrolador que satisfaga el número de conversores AD necesarios, será más que apto para realizar los cálculos necesarios.

Señal	**Descripción**
$Throttle$	Puño acelerador con potenciómetro
$Front\ regen\ brake$	Señal de freno regenerativo del freno delantero
$Back\ regen\ brake$	Señal de freno regenerativo del freno trasero
V_{DC}	Tensión de la batería
I_a	Fase a de la corriente
I_b	Fase b de la corriente
I_c	Fase c de la corriente
$Resolver\ signals$	Señales seno y coseno

Tabla 3 – Señales de entrada del microcontrolador

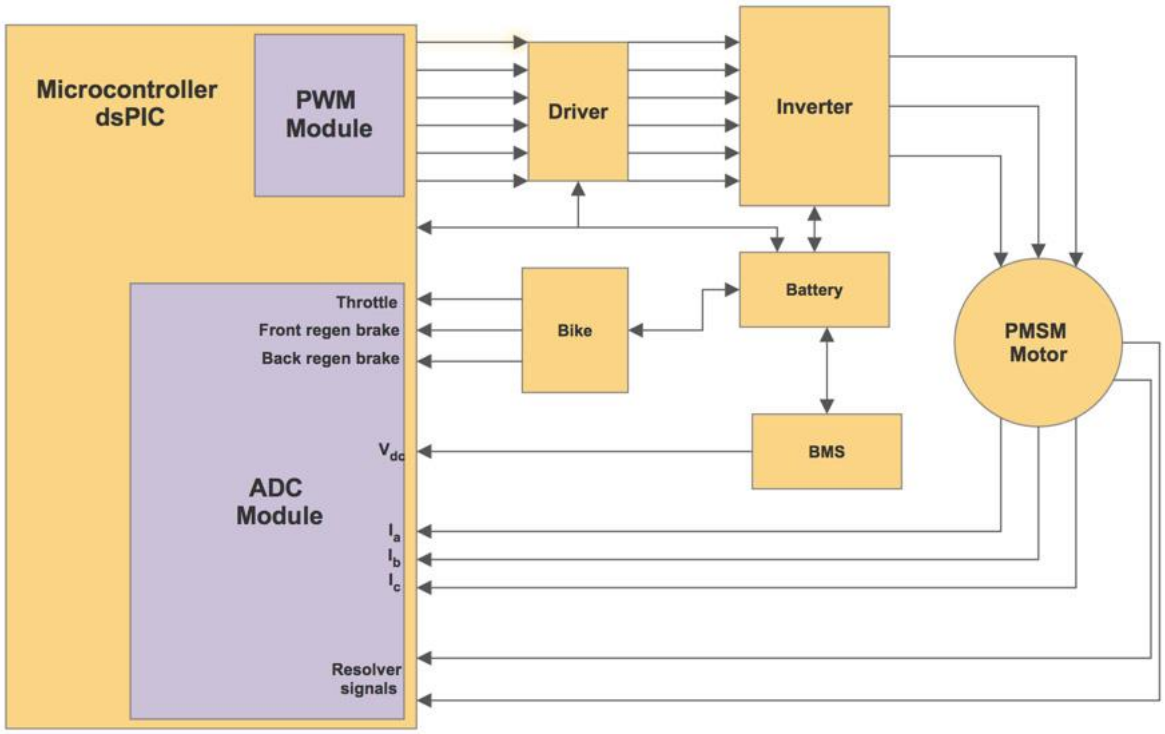


Figura 18 – Diagrama de bloques del controlador

Se ha escogido entonces el microcontrolador dsPIC33FJ32GS606. La Figura 19 representa los pines de este controlador:

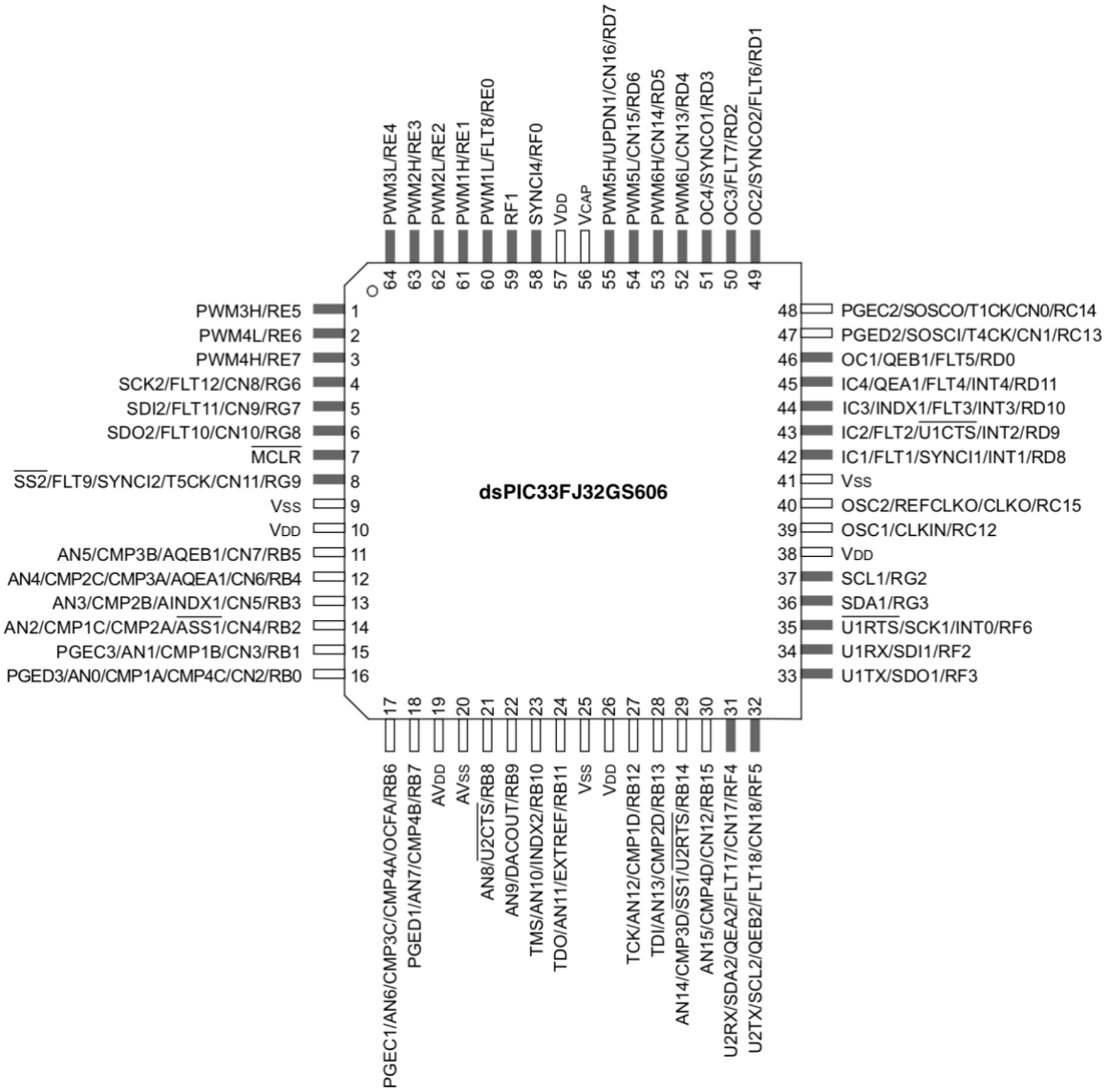


Figura 19 – Pines del dsPIC33FJ32GS606

4.1.1.1 Configuración del oscilador en el dsPIC33FJ32GS606

A continuación, se presenta se presenta un **breve resumen** de la configuración del oscilador en el dsPIC33FJ32GS606.

Para utilizar cualquier función del microcontrolador se tiene que configurar su oscilador primero, ya que es el que determina la referencia para el reloj y por lo tanto los *MIPS* (millones de instrucciones por segundo).

La Figura 20 muestra la relación entre F_{OSC}, F_{CY} y el tiempo de ejecución de las instrucciones.

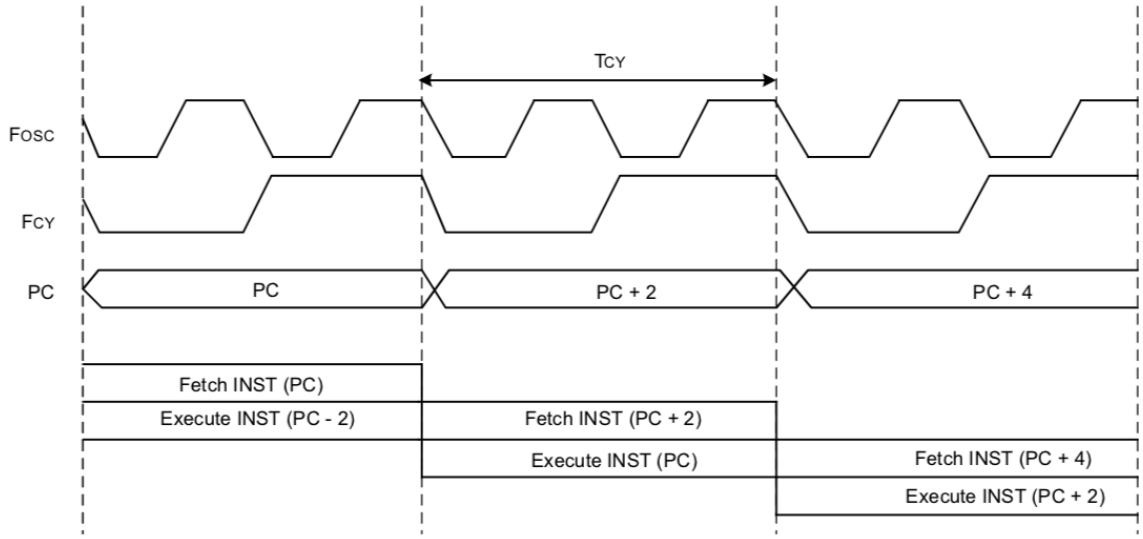


Figura 20 – Relación entre F_{OSC}, F_{CY} y el tiempo de ejecución de instrucciones del dsPIC33FJ32GS606 [28]

Se puede ver como $F_{CY} = \frac{F_{OSC}}{2}$.

La frecuencia máxima que puede tomar F_{OSC} es de $80MHz$, con lo que se consiguen $40MIPS$ [28].

Para calcular F_{OSC}, lo primero es elegir la frecuencia de referencia. Esta puede venir dada por:

- Un oscilador primario – externo – (P_{OSC}), a través de los pines 39 y 40 (*OSCI1* y *OSCI2* respectivamente),
- Un oscilador secundario – externo – (S_{OSC}), a través de los pines 47 y 48 (*SOSCI* y *SOSCO* respectivamente),
- Por el cristal interno (*FRC*).

En este caso se usará el cristal interno.

Además, el microcontrolador dispone de un circuito *PLL* (*Phase Locked-Loop*), que permite obtener velocidad de operaciones mayores. Para conseguir la mayor velocidad posible ($40MIPS$), será necesario utilizar dicho circuito.

El cálculo de F_{OSC} viene dado por:

$$F_{OSC} = F_{IN} \cdot \frac{M}{N_1 \cdot N_2} \quad (24)$$

$$F_{OSC} = F_{IN} \cdot \frac{PLLDIV + 2}{(PLLPRE + 2) \cdot (2 \cdot (PLLPOST + 1))} \quad (25)$$

F_{IN} representa la frecuencia de entrada, $PLLDIV$ el multiplicador del circuito *PLL*, $PLLPRE$ el pre-escalado del *PLL* y $PLLPOST$ el post-escalado del *PLL*.

- La frecuencia nominal del cristal interno es de $7.37MHz$,
- $PLLDIV$ es un valor entre 2 y 513 [28],
- $PLLPRE$ es un valor entre 0 y 31 [28],
- $PLLPOST$ toma el valor 0, 1 o 3 [28].

Tomando $F_{IN} = 7.37KHz$, $F_{OSC} = 80KHz$ y los valores mínimos de N_1 y N_2 (2 en ambos casos), se despeja $PLLDIV$ de (24):

$$M = \frac{F_{OSC}}{F_{IN}} \cdot (N_1 \cdot N_2) \quad (26)$$

$$M = 40$$

Luego $PLLDIV = 43 - 2 = 41$.

La explicación y configuración completa del oscilador se encuentra en el ANEXO A – Configuración del oscilador en el dsPIC33FJ32GS606.

4.1.1.2 Módulo PWM en el dsPIC33FJ32GS606

A continuación, se presenta un **breve resumen** de la configuración del módulo PWM en el dsPIC33FJ32GS606.

El dsPIC33FJ32GS606 dispone de 6 generadores de PWM, cada uno con su respectiva salida *High* y *Low*, por lo que en total el módulo PWM ocupa 12 pines del microcontrolador - solo se necesitan 3 generadores, por lo tanto, se usan 6 pines (Anexo B – Figura 6).

La Figura 21 presenta el esquema del módulo PWM del microcontrolador dsPIC33FJ32GS606. Cada generador PWM tiene dos salidas: PWMxH y PWMxL. Existe un registro de tiempo maestro que permite sincronizar varias salidas PWM (*Master Time Base* o MTB), aunque cada generador PWM puede funcionar de forma independiente (*Independent Time Base* o ITB). El módulo PWM también dispone de pines de entrada que monitorizan el sistema y lo protegen de sobre-corrientes o avisan de posibles fallos.

Para que el reloj del PWM funcione de forma independiente al reloj del sistema, se tiene que utilizar el generador de reloj auxiliar. Para generarlo, se puede usar:

- El oscilador primario (POSCCLK),
- La salida PLL (F_{VCO}),
- El reloj interno FRC (FRCCLK).

El registro de control de reloj auxiliar (ACLKCON) permite elegir la referencia (una de las tres mencionadas antes) y habilitar el circuito PLL auxiliar para obtener la frecuencia deseada. La frecuencia nominal de entrada de PWM tendría que ser de $120MHz$ [29].

Se usará el reloj interno (FRC) como referencia del reloj de PWM.

El siguiente paso es configurar el modo de operación del PWM. Típicamente, para control de motores se hace que las señales PWM se alineen de forma centrada (en inglés *center-aligned mode*). A partir de este instante y con el fin de simplificar la lectura, se usará 'CAM' para referirse a dicho modo.

FIGURE 16-1: HIGH-SPEED PWMx MODULE ARCHITECTURAL DIAGRAM

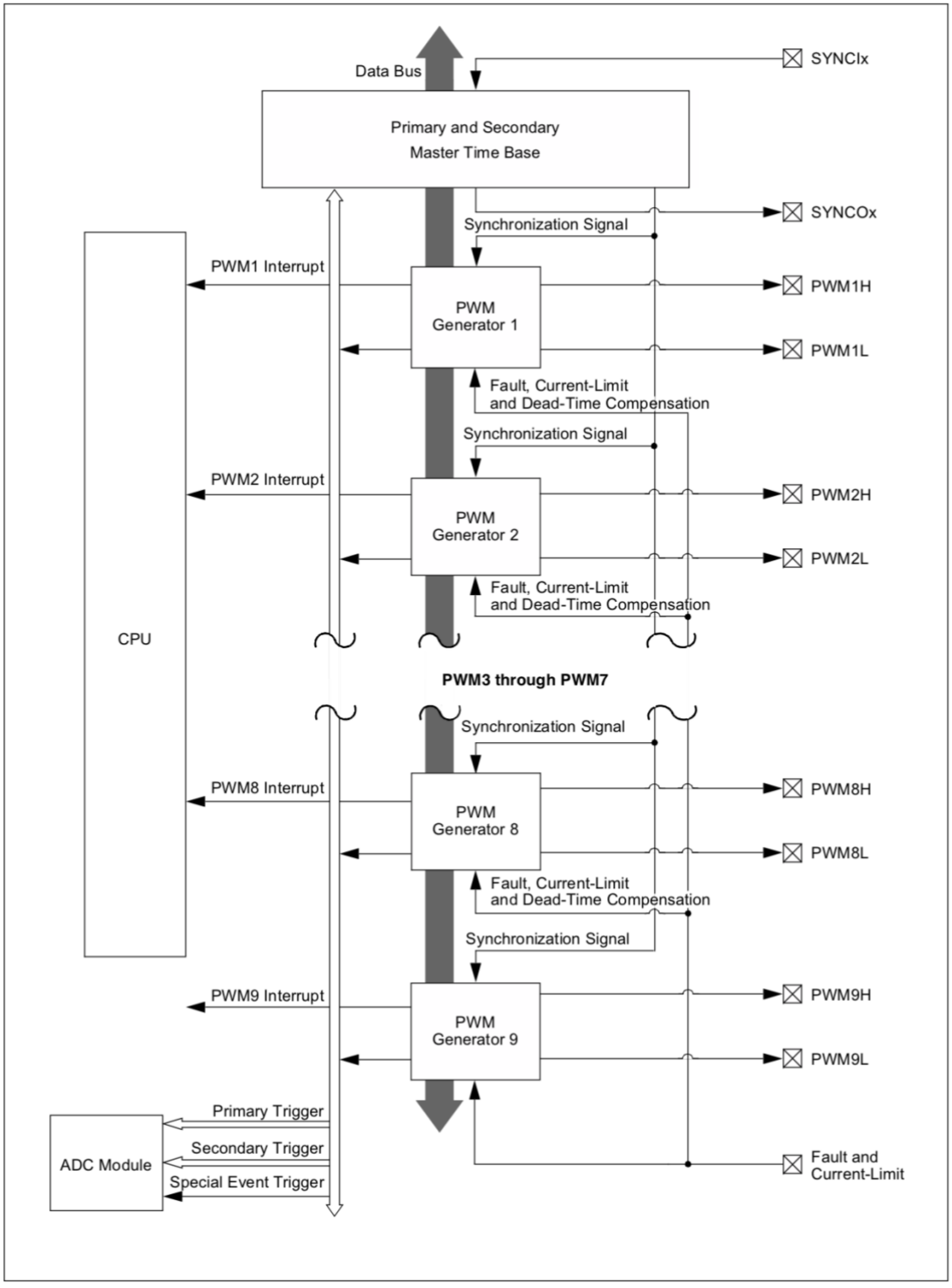


Figura 21 – Diagrama de bloques del funcionamiento del módulo PWM en el dsPIC33FJ32GS606 [29]

Al utilizar el CAM, los generadores PWM tienen que trabajar en modo independiente, y por tanto se tiene que especificar el período de **cada generador PWM**.

A continuación, hay que especificar el período en el que tiene que estar *on* cada generador (llamado factor de servicio o *duty cycle* en inglés), el cual va almacenado en los registros PDCx y que se calcula con el software de control.

El último paso consiste en gestionar el disparo de los conversores analógico-digitales. Como los generadores PWM tienen que trabajar en modo independiente, cada uno de ellos puede generar una señal de aviso al módulo ADC. Por otro lado, se ha comentado anteriormente que los 3 generadores trabajan con el mismo período, por lo que, con programar una señal de aviso en uno de los 3 generadores valdría.

La explicación y configuración completa del módulo PWM se encuentra en el ANEXO B – Configuración del módulo PWM en el dsPIC33FJ32GS606.

4.1.1.3 Módulo ADC en el dsPIC33FJ32GS606

A continuación, se presenta un **breve resumen** de la configuración del módulo ADC en el dsPIC33FJ32GS606.

La funcionalidad del módulo ADC es relativamente sencilla. Este microcontrolador dispone de dos conversores SAR (*Succesive Approximation Register)* y 16 canales para muestrear señales. El funcionamiento es por parejas de pines analógicos. Cada pareja (por ejemplo, Pareja 0 (AN0, AN1), Pareja 1 (AN2, AN3)) recibe una orden de conversión. Como este microcontrolador dispone de dos SAR, se pueden muestrear los pines pares e impares de forma simultánea por turnos, ya que un SAR controla los números pares y otro los números impares. Por ejemplo, se muestrea primero la Pareja 0, luego la Pareja 1, etc.

El módulo ADC recibe la orden de empezar a muestrear a partir del módulo PWM (en concreto del generador PWM nº1).

Una vez muestreados todos los canales necesarios, se genera una interrupción y se dispara el algoritmo de control.

La explicación y configuración completa del módulo PWM se encuentra en el ANEXO C – Configuración del módulo ADC en el dsPIC33FJ32GS606.

4.1.2 INVERSOR

El inversor es el encargado de convertir la tensión DC en tensión AC. Se compone principalmente por transistores IGBT's o MOSFET y un radiador para disipar el calor. La Figura 22 muestra la topología de un inversor trifásico de dos niveles:

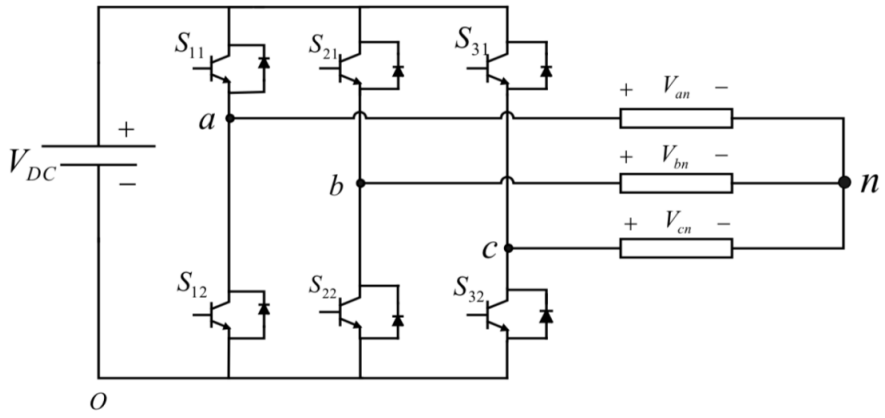


Figura 22 – Esquema de un inversor trifásico

Existen muchos criterios que hay que tener en cuenta a la hora del diseño de un inversor [23]. Si se opta por fabricarlo, la complejidad es muy alta ya que hay que elegir los componentes individualmente, montar el dispositivo y asegurarse de que el funcionamiento y compatibilidad es correcto [24]. Como no es el objetivo de este trabajo, se ha optado por buscar una combinación transistores-radiador que cumpla con los requisitos establecidos.

4.1.2.1 Elección de transistores

En este trabajo se ha escogido utilizar un inversor prefabricado con el fin de asegurar que el montaje está realizado correctamente.

Existen muchas empresas fabricantes de puentes IGBTs o MOSFET, en este caso se ha mirado en el catálogo de Semikron [26].

El parámetro más importante y determinante para la elección del tipo de transistor es la corriente máxima que puede circular por este. Como se ha visto en (5), la corriente máxima que puede dar la batería es de $197{,}12A$.

Sin embargo, ese valor es el valor eficaz de la corriente. Es importante tener en cuenta el valor de pico de la corriente para la elección de los transistores.

$$I_{Peak} = \sqrt{2} \cdot I_{RMS}$$

$$I_{Peak} = 271.69A \qquad (27)$$

Por lo que el transistor debe ser capaz de aguantar esa corriente, además de un margen. Tampoco interesa que dicho transistor aguante corrientes mucho más elevadas porque el coste se dispara.

Semisel tiene una herramienta que muestra, tras introducir una serie de parámetros, una serie de transistores que se adaptan a esos parámetros introducidos [37]. La figura siguiente (Figura 23) muestra la pantalla de introducción de parámetros:

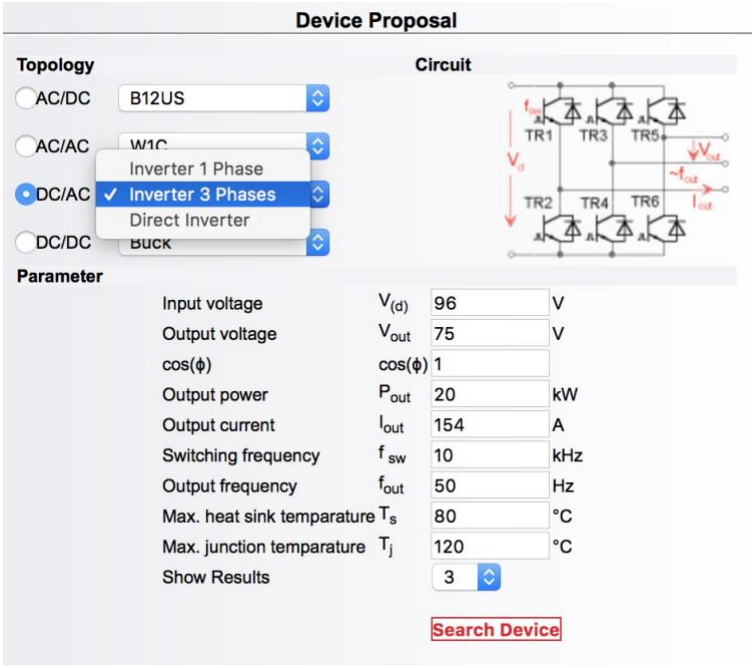


Figura 23 – Inserción de parámetros para propuesta de transistores

Lo primero que hay que cambiar es el tipo de inversor. Al tratarse de un motor PMSM trifásico, hay que utilizar un inversor trifásico.

A continuación, se introduce:

- La tensión de entrada DC: $96V$,
- El coseno del ángulo ϕ: vale 1,
- La potencia del motor, se elige el caso más desfavorable: $20kW$[2],
- La frecuencia del PWM: $10kHz$.

El resto de parámetros se calculan automáticamente.

El resultado se puede resumir en la Figura 24. En ella se pueden ver hasta 3 transistores por cada familia de transistores. De esta lista, se pueden filtrar algunos transistores. Por ejemplo, aquellos en los que su nombre haya '600' o algo similar indican que pueden soportar hasta $600A$, que, aunque son perfectamente aptos, sobrepasan por mucho la corriente máxima calculada en (26). De la misma forma, una lectura a primera vista del datasheet de cada transistor indica la corriente nominal máxima que pueden soportar.

Por lo tanto, tras un previo filtrado, estudiaremos los siguientes transistores:

- SKiiP38GB12E4V1_HpTp
- SKM400GB07E3
- SKiM406GD066HD
- SEMiX303GB12E4p

[2] En la Figura 23 se puede ver que la corriente de salida es de $154A$. Eso se debe a que Semisel asume que la equivalencia entre la tensión DC y AC viene dada por un factor más elevado (pero menos realista) que el que se ha utilizado en el apartado 2.1.1 (0.61). Al aumentar el valor de la tensión de AC, disminuye la corriente.

MiniSKiiP

Device :	Transistor Losses P_{tr}	Diode Losses P_d	Tj = f(t) during overload and fmin
SKiiP38GB12E4V1_HpTp	316.48 W	12 W	118 °C
SKiiP39GB12E4V1_HpTp[1)]	252.45 W	16 W	100 °C

SEMITRANS

Device :	Transistor Losses P_{tr}	Diode Losses P_d	Tj = f(t) during overload and fmin
SKM600GB066D[2)]	197.85 W	18 W	112 °C
SKM400GB07E3[3)]	229.30 W	11 W	113 °C
SKM350MB120SCH15*[4)]	370.33 W	0.00 W	116 °C

SKiM

Device :	Transistor Losses P_{tr}	Diode Losses P_d	Tj = f(t) during overload and fmin
SKiM406GD066HD*[5)]	235.44 W	9.79 W	112 °C
SKiM606GD066HD_HpTp[6)]	205.23 W	15 W	96 °C
SKiM606GD066HD*[5)]	206.17 W	15 W	102 °C

SKiiP

Device :	Transistor Losses P_{tr}	Diode Losses P_d	Tj = f(t) during overload and fmin
SKiiP613GD123-3DUW_V3[7)]	312.92 W	18 W	103 °C
SKiiP613GD123-3DUL_V3[8)]	310.84 W	18 W	97 °C
SKiiP603GD123-3DUW_V3[9)]	310.15 W	18 W	95 °C

SEMiX

Device :	Transistor Losses P_{tr}	Diode Losses P_d	Tj = f(t) during overload and fmin
SEMiX603GB066HD	201.77 W	11 W	115 °C
SEMiX303GB12E4p[10)]	304.27 W	15 W	118 °C
SEMiX404GB12Vs	273.32 W	24 W	118 °C

Figura 24 – Resultado de propuesta de transistores de Semisel

4.1.2.2 Simulación Semisel

La herramienta web Semisel de Semikron [25] permite estudiar el perfil de temperatura de un transistor montado en un radiador determinado. El objetivo de este apartado es encontrar la combinación transistor-radiador apta para el trabajo.

En un primer lugar se introducen una serie de parámetros descritos a continuación y en un segundo lugar se elige el transistor que se desea estudiar.

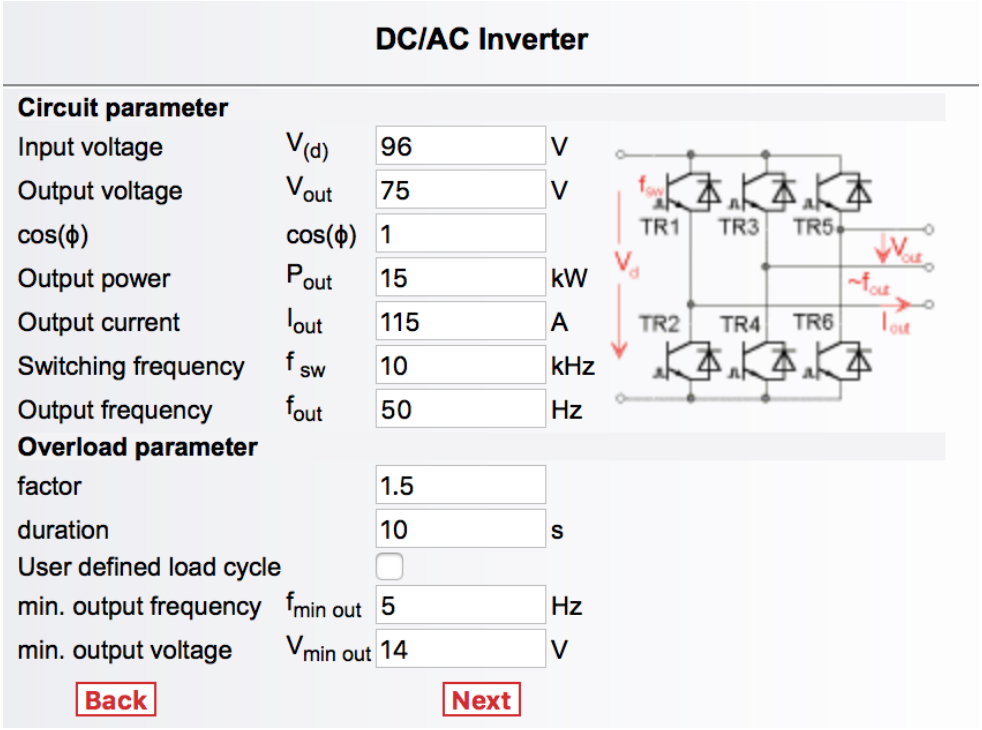


Figura 25 – Parámetros de simulación Semisel para estudio de transistor

La Figura 25 presenta la ventana de introducción de parámetros para la simulación. Se han introducido los mismos parámetros que en el apartado anterior, salvo algunos que se explican a continuación.

Se ha escogido $15KW$ como potencia de entrada con un factor de sobrecarga de 1.5. Esto significa que se pueden dar $22.5KW$ durante 10s, es decir, más del máximo ($20kW$) durante unos segundos.

La frecuencia de PWM será de $10KHz$. Esta frecuencia tiene que ver con lo frecuentemente que se va a actualizar el ancho de pulso de la señal PWM y como todavía no se ha explicado el funcionamiento del controlador, la justificación no se hará en este apartado sino en el apartado 4.3.

La frecuencia mínima es la frecuencia mínima a la que gira el motor en régimen permanente. Es decir, de forma prolongada, a qué frecuencia giraría el motor. Como la motocicleta nunca

va a estar a bajas revoluciones durante tiempos elevados, este parámetro no es de mucha importancia y se ha puesto un valor de $5Hz$.

El resto de parámetros se calculan de forma automática.

A continuación, aparece una ventana donde se presentan los distintos transistores de Semikron disponibles. Hay que escoger uno de los mencionados en 4.1.2.1 (Figura 26).

Finalmente hay que introducir unos parámetros de temperatura y elegir entre los radiadores propuestos para simular (Figura 27).

Como temperatura ambiente se ha escogido $40º$, una temperatura razonable en un circuito en un día de verano. El siguiente parámetro que hay que cambiar es el número de transistores por radiador, que en este caso son 6 (posteriormente se verá que en realidad son 7).

Finalmente hay que introducir el flujo de aire. Con el fin de asegurar el mejor funcionamiento posible, se ha escogido el caso más desfavorable de flujo de aire para cada radiador (este valor cambia en función del radiador).

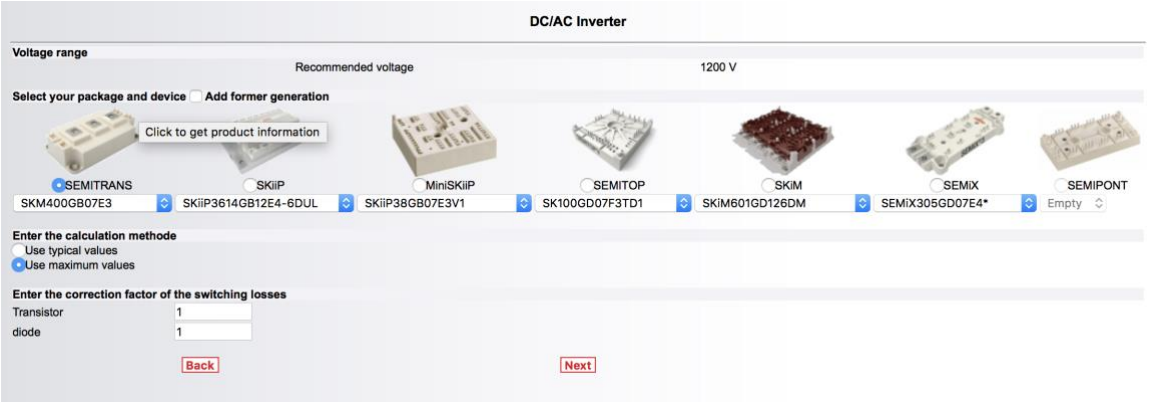


Figura 26 – Elección del transistor que se quiere simular

Como se han escogido 4 transistores previamente (4.1.2.1), y existen 4 radiadores posibles (Figura 27), se tienen 16 posibilidades diferentes, asumiendo que por cada combinación transistor-radiador, se simule con un solo flujo de aire.

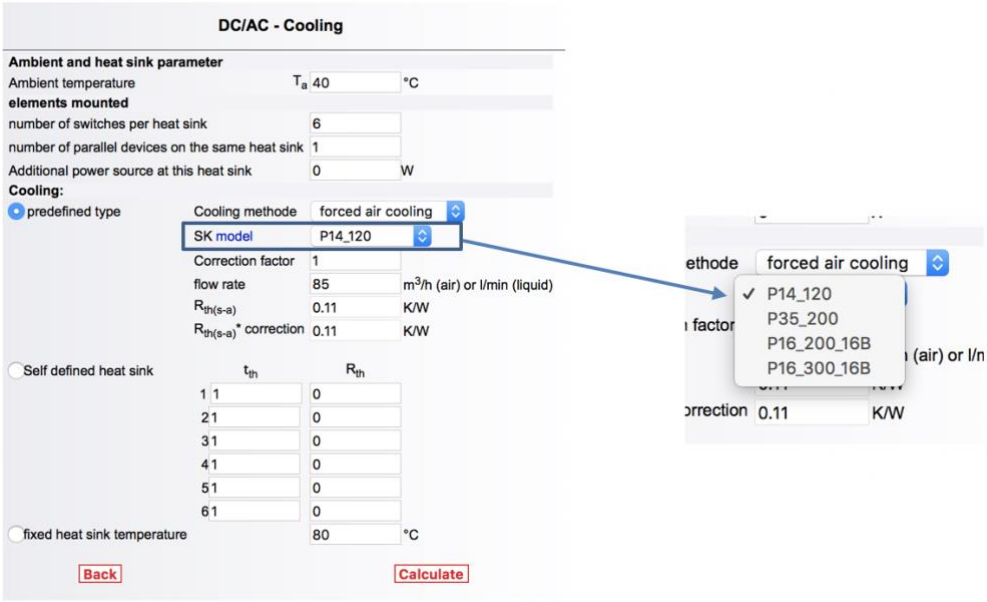


Figura 27 – Parámetros de refrigeración y elección de radiador

El resultado de una simulación cualquiera se puede ver en el ANEXO E (Ejemplo de resultado de simulación de Semisel). En ella se pueden ver los distintos parámetros introducidos previamente, el transistor elegido (en el ejemplo se ha escogido un transistor cualquiera), el radiador escogido y una serie de resultados. Entre estos destaca una gráfica que representa la temperatura de distintos puntos del radiador o transistor. Semisel ofrece la opción de descargar los resultados bajo el formato Excel, lo que permite un estudio conjunto y más profundo de distintas combinaciones (Ejemplo de datos exportados de Semisel a Excel).

También se puede ver un veredicto de la evaluación (en dicho ejemplo, la configuración no funciona).

Además de que obviamente la configuración tiene que funcionar, se ha buscado la combinación que ofrezca la menor temperatura posible en el mayor número de puntos.

La Figura 28 representa el resultado de 18 simulaciones realizadas.

En ella se pueden ver los distintos transistores que se han mencionado en 4.1.2.1, (columna 'IGBT'), así como los radiadores vistos en la Figura 27(columna 'Cooler'). Además, en dos combinaciones transistor-radiador se ha simulado con otro flujo de aire que no fuese el mínimo. Este parámetro se puede ver en la columna 'Flow Rate'.

La columna 'Works' representa el veredicto de la simulación, mientras que la columna 'Min Value to work' indica el mínimo valor de flujo de aire para que funcione la configuración.

En la primera parte del estudio (hasta la columna 'Min Value to work') la leyenda de colores es la siguiente:

- El color verde indica que la configuración funciona para el mínimo valor posible de flujo de aire.
- El color amarillo indica que la configuración funciona, pero no para el valor mínimo del flujo de aire. En este caso se precisa en la columna 'Min Value to work' dicho valor mínimo.
- El color rojo indica que la configuración no funciona para **ningún** valor de flujo de aire, incluido el máximo (afortunadamente no se ha dado este caso).

IGBT	Cooler	Gráfica	Flow Rate	Works	Min Value to work	Max Temp Ts	Max Temp Tc	Max Temp Ttr	Max Temp Td
SKiiP38GB12E4V1_HpTp	P14_120	1.1	63.75	Yes	All	116,281	141,354	N/A	147,985
	P35_200	1.2	60	No	80	156,153	182,621	N/A	189,574
		1.3	80	Yes		147,196	168,272	N/A	174,779
	P16_200_16B	1.4	232.5	Yes	All	72,6973	92,0218	N/A	97,9729
	P16_300_16B	1.5	221.25	Yes	All	62,3973	81,6462	N/A	87,5239
SKM400GB07E3	P14_120	2.1	63.75	Yes	All	100,006	102,971	119,922	122,263
	P35_200	2.2	60	Yes	All	128,779	131,848	149,2	151,961
	P16_200_16B	2.3	232.5	Yes	All	66,1498	69,089	85,7641	87,6161
	P16_300_16B	2.4	221.25	Yes	All	*58,176*	*61,1299*	*77,7896*	*79,5276*
SKiM406GD066HD	P14_120	3.1	63.75	Yes	All	102,183	118,094	N/A	121,903
	P35_200	3.2	60	Yes	All	102,183	118,094	N/A	121,903
	P16_200_16B	3.3	232.5	Yes	All	66,958	82,6231	N/A	85,7574
	P16_300_16B	3.4	221.25	Yes	All	*58,7347*	74,4123	N/A	*77,391*
SEMiX303GB12E4p	P14_120	4.1	63.75	Yes	All	114,737	119,462	135,776	144,484
	P35_200	4.2	60	No	85	154,166	159,174	176,449	185,608
		4.3	85	Yes		143,844	148,802	165,88	174,922
	P16_200_16B	4.4	232.5	Yes	All	72,139	76,7074	92,2886	100,516
	P16_300_16B	4.5	221.25	Yes	All	61,9833	*66,5481*	*82,0351*	90,1493

Figura 28 – Estudio detallado de distintas configuraciones transistor-radiador

En la segunda parte del estudio (las últimas 4 columnas), se hace uso de los resultados aportados por Semisel, que se han exportado en Excel.

Para cada configuración, se ha calculado el valor máximo de las distintas temperaturas. Se busca la configuración que ofrezca el mayor número de valores mínimos de temperaturas máximas.

El gradiente de colores funciona por columna, e indica el valor más alto en rojo oscuro y el valor más pequeño en amarillo claro. Además, vienen encuadrados en negro los 2 valores más pequeños de cada columna.

A partir de esta tabla podemos escoger la combinación que mejor se ajuste al problema. En este caso podemos observar que el transistor SKiM406GD066HD junto con el radiador P16_300_16B tiene las temperaturas más bajas en todas categorías.

4.1.2.3 Conclusión

Al igual que con los parámetros del motor (3.3), las características del transistor sirven para la posterior simulación del control del motor (Capítulo 5.). Como primera opción, se puede elegir el transistor SKiM406GD066HD.

4.1.3 DRIVER

Las señales PWM generadas por el microcontrolador no dan corriente suficiente para activar los transistores del inversor. Por esta razón, se coloca un "*gate driver*" delante de los transistores. Su función es controlar las características de encendido y apagado de los transistores.

La propia página de Semikron propone opciones de drivers para un transistor determinado. Como se ha escogido un transistor previamente, podemos elegir un driver apropiado para este.

Del apartado anterior se sabe que el transistor elegido es el SKiM406GD066HD. La siguiente figura muestra el interfaz de la herramienta de Semikron que propone un driver apto para el transistor introducido.

Driver Select Tool

Preselect	All Transistors	
Product	SKiM (33)	
Device	SKiM406GD066HD*	
Number of IGBT Modules	6	
Switching Frequency f_{sw}	10	kHz
Applied Gate Resistor	3.5	Ohm

Update

Result

Driver Channels	6
Collector Emitter Voltage	600 V
Required average current	192 mA
Gate Charge	19,2 mC(6Modules in parallel)

Driver

Name	$I_{out(av)}$ /mA	$\hat{I}_{out}$ /A	V_{isol} /kV	$V_{ce\ max}$ /V	R_{gmin} / Ohm	Channels
3x SKYPER 52 R*(1	300	50	4.0	1200	0.6	2

Note

1) * not for new design, replaced by SKYPER 42 R or SKYPER 42 LJ R

Figura 29 – Elección de driver con Semisel

Se puede ver como el Semisel propone utilizar el driver SKYPER 42R. (Figura 29)

4.2 ACLARACIONES

Es importante comprender tanto la estructura como el funcionamiento del motor PMSM antes de explicar la funcionalidad del controlador. En el Capítulo 3. se ha explicado en detalle dicho motor, aunque se han dejado ciertos aspectos por explicar para que tuvieran más sentido a la hora de mencionarlos.

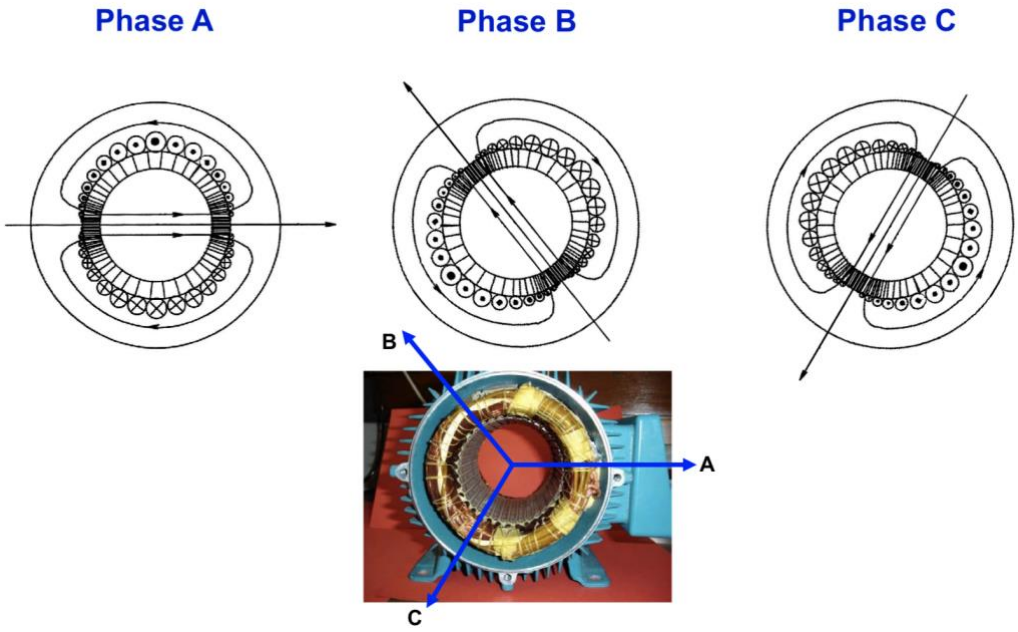


Figura 30 – Esquema de las tres fases de un motor PMSM [30]

El interés de utilizar un motor de imanes permanentes es para poder producir un movimiento suave. Se ha mencionado en el Capítulo 3. que esto se consigue haciendo que los devanados se enrollen más veces en distintos lugares, consiguiendo así más pares de polos al circular la corriente. Independientemente del número de polos del motor, un motor PMSM se compone de tres fases (los devanados tienen que estar enrollados de tal forma que las fases estén separadas geométricamente de 120º). La Figura 30 ilustra lo explicado.

4.2.1 FOC – FIELD ORIENTED CONTROL

La Figura 31 muestra un ejemplo de como interactúan las tres fases entre sí para crear un campo magnético rotacional. Al ser un motor trifásico, las corrientes también están desfasadas 120º entre sí.

El resultado son tres vectores magnéticos que crecen y decrecen de forma síncrona con sus corrientes y en sus ejes magnéticos correspondientes. La suma de estos vectores (como se puede observar en la parte inferior derecha de la Figura 31) produce un vector magnético que rota suavemente, representado en magenta en la parte inferior izquierda de la Figura 31.

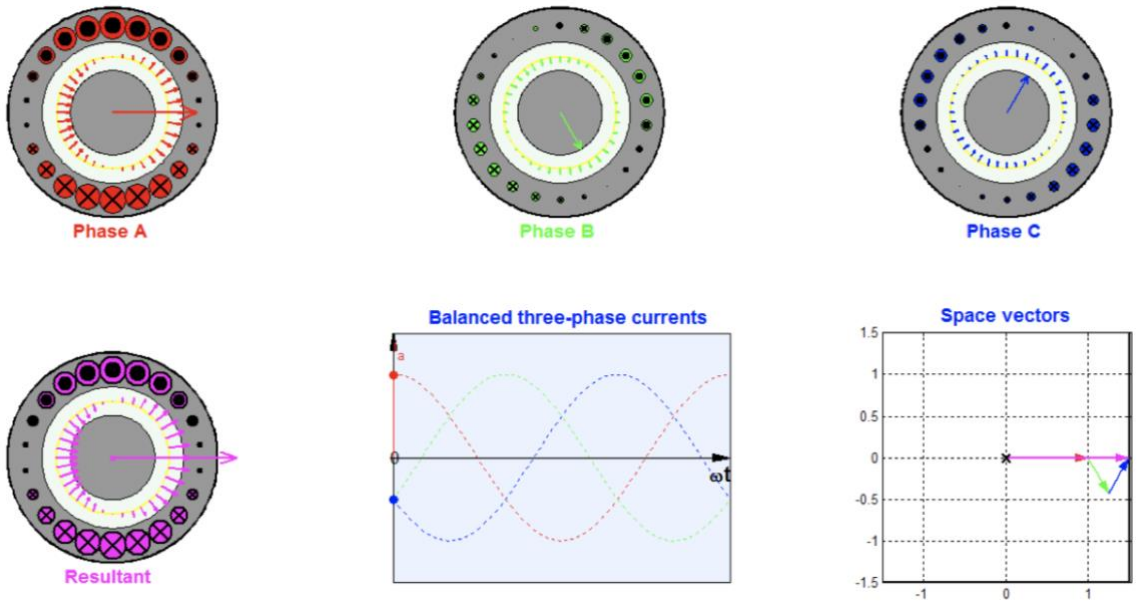


Figura 31 – Resumen gráfico del FOC [30]

El rotor lo único que ve es un estator que 'va girando' a la misma frecuencia que la señal seno de la corriente. Lo más destacable es que se puede ver como se puede crear un vector equivalente en cualquier ángulo si se aplican las corrientes adecuadas en las fases. Esta es la base del *Field Oriented Control* – FOC.

En resumen, lo que se busca es orientar el campo global generado por las fases – o sea, el campo del estator – con el campo del rotor para conseguir el máximo rendimiento en todo momento.

4.2.2 ÁNGULOS EN EL PMSM

Se ha explicado en el apartado anterior que con la técnica del FOC se puede crear un vector magnético del estator en cualquier ángulo aplicando las corrientes adecuadas en las fases del estator. La pregunta que surge ahora es, ¿qué ángulo se quiere?

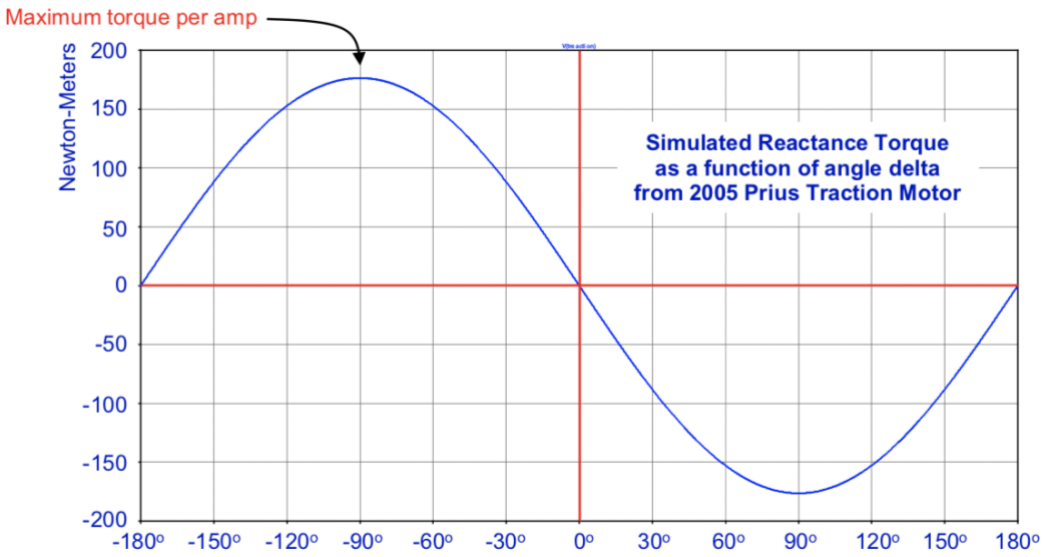


Figura 32 – Par por amperios en función de la posición del rotor en un motor PMSM trifásico [33]

Lo más común es orientar el campo del estator de tal forma que se consiga el mayor par máximo. La Figura 32 representa una gráfica del par por amperios en función del ángulo entre el campo del rotor y el campo del estator.

Como se puede observar y como era de esperar, el par máximo se da cuando hay 90º entre los campos magnéticos. Esto significa que una vez sea conocido el ángulo en el que está el rotor, se pueden aplicar las corrientes correspondientes a las fases del motor para que dichas fases produzcan un campo magnético perpendicular al del rotor. Conforme el rotor gira, el ángulo cambia y hay que recalcular las corrientes necesarias para que el campo formado por el estator siga siendo perpendicular.

En la Figura 33 se pueden ver en verde el vector de flujo creado por las fases del estator, y en rojo el vector de flujo creado por el rotor. Como el flujo del rotor está fijo al rotor (el rotor está formado por imanes permanentes), el rotor gira a la misma velocidad que el flujo del estator y por eso se dice que es **síncrono** con el flujo del estator.

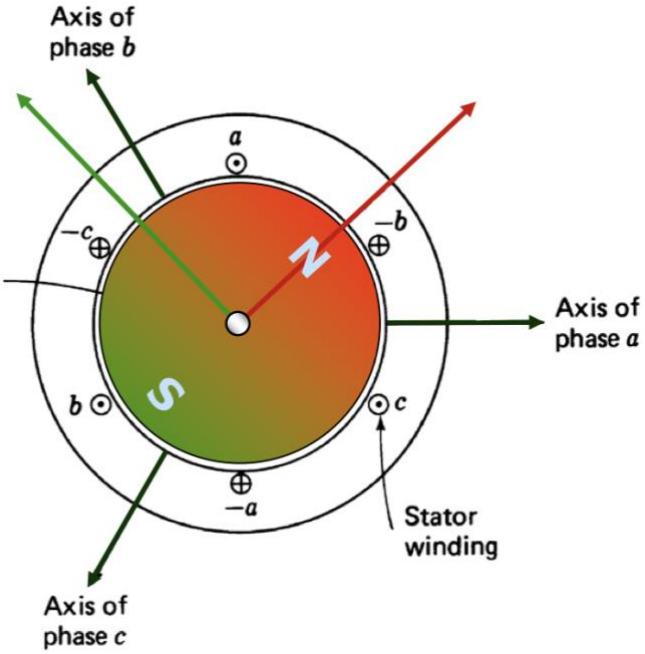


Figura 33 – Vectores de flujo del estator y del rotor [34]

4.2.3 EJES 'D' Y 'Q'

La última aclaración antes de explicar el funcionamiento del controlador es ilustrar qué son los ejes d y q.

El **eje d** no es más que el eje que forma el flujo creado por los imanes del rotor (Figura 34).

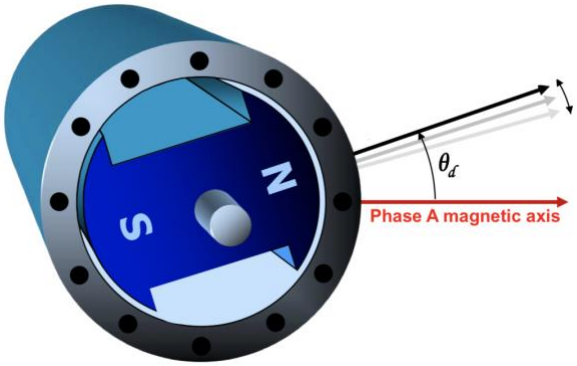


Figura 34 – Eje d de un PMSM [30]

El **eje q** es el eje que situado a 90º del eje d.

4.3 FUNCIONAMIENTO

La Figura 35 presenta el esquema de bloques del control de un motor de imanes permanentes.

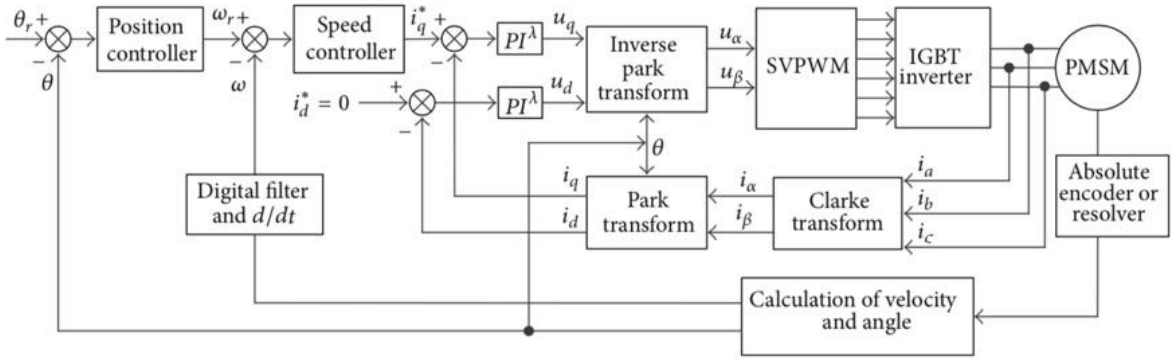


Figura 35 – Esquema de bloques del control de un motor PMSM utilizando un encoder como medida de posición

Figura original en [12]

De la ficha técnica del motor (Ficha técnica del motor de la competición) se puede ver que el motor dispone de un resolver como medida de posición. Por otro lado, se recuerda de 1.1.3 que un resolver permite dar la posición absoluta del motor a través de dos señales (seno y coseno).

El funcionamiento es el siguiente:

- A partir de estas señales (seno y coseno), el software diseñado tiene que ser capaz de calcular la velocidad y el ángulo,
- En el mismo instante que se miden esas señales, se miden las corrientes a, b y c del motor (una para cada fase),
- El algoritmo realiza la transformada de Clarke y de Park (se explica en detalle más adelante) para obtener las corrientes de referencia i_d e i_q,
- Comparando la demanda de velocidad del piloto (a través del puño acelerador-potenciómetro) con la velocidad medida anteriormente, se calcula con el control de velocidad la demanda de corriente en el eje q,

- Se comparan las corrientes de referencia del eje q (demanda de velocidad del piloto) y d (que vale 0) con las corrientes medidas,
- Se pasan a un control PI para generar las tensiones de referencia en ejes d y q,
- Se calcula la inversa de la transformada de Park,
- Se calculan los tiempos de activación de los transistores,
- Se disparan las señales PWM que generan las corrientes necesarias para la demanda del piloto.

En los próximos apartados se explica de forma detallada las etapas mencionadas anteriormente.

4.3.1 LECTURA Y CÁLCULO DE POSICIÓN Y VELOCIDAD

Las señales de seno y coseno dadas por el resolver del motor son de voltaje superior al que puede soportar el microcontrolador ($3.3V$). Además, al ser señales senoidales, tienen valores negativos. Por lo tanto, hay que acondicionar el sistema para la lectura correcta de estas señales. La figura siguiente (Figura 36) presenta el esquema de un resolver:

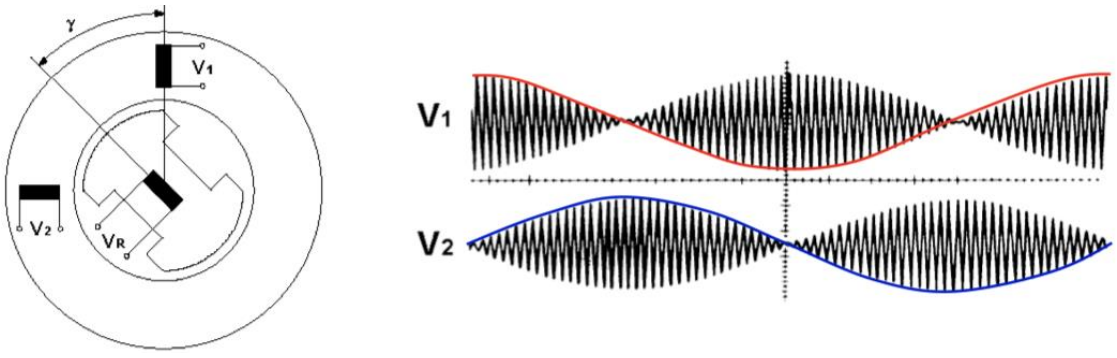


Figura 36 – Esquema de un resolver [30]

Cuando el ángulo es 0, el valor de V_1 es máximo y el de V_2 es 0, y cuando el ángulo es de 90º, el valor de V_2 es máximo y el de V_1 igual a 0.

Las señales del resolver del motor PMSM están centradas en $\pm 5V$, y tienen una tensión pico-pico de $10V$. Lo primero que hay que hacer es centrar ambas señales de tal forma que el

valor mínimo sea 0. Como la señal coseno ya cumple este requisito, solo hay que añadir un offset a la señal del seno.

Después hay que utilizar un divisor de tensión para llevarse la tensión máxima dada por el resolver, a la tensión máxima admitida por el microcontrolador.

La figura siguiente (Figura 37) presenta el diagrama de bloques para la lectura de las señales del resolver:

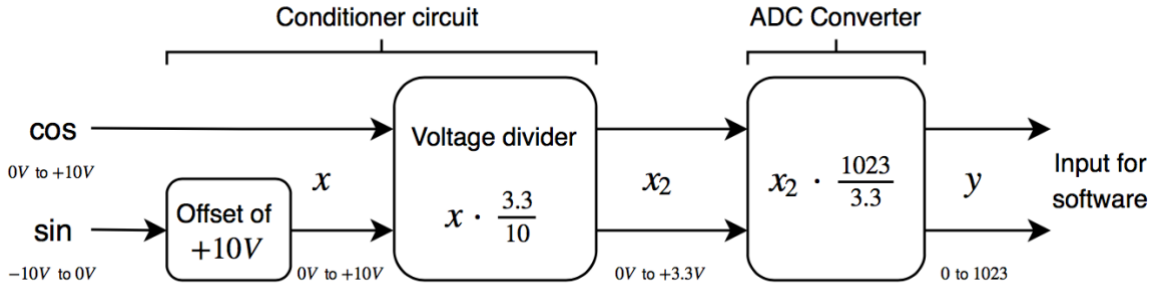


Figura 37 – Diagrama de bloques para la lectura de las señales del resolver

Cuanto el microcontrolador haya muestreado ambas señales, simplemente se normaliza el valor de entrada por 1023 para tener un número entre 0 y 1:

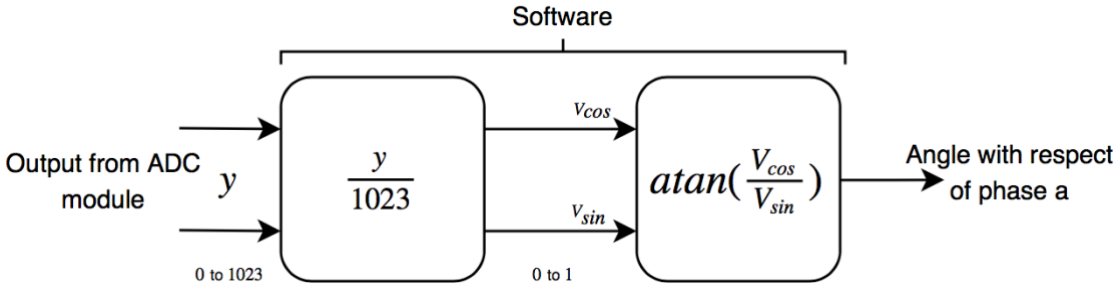


Figura 38 – Cálculo del ángulo entre el eje d y la fase a de un motor PMSM con un resolver una vez muestreadas las señales

El circuito de acondicionamiento hardware para conseguir el offset y el divisor de tensión no se diseñará.

Para calcular la velocidad actual basta con calcular la diferencia entre el ángulo en un instante t_1 y t_2:

$$\omega = \frac{\phi_1 - \phi_0}{t_1 - t_0} = \frac{\Delta\phi}{\Delta t}$$

Esta velocidad (en radianes por segundos) es la que se compara con la referencia (la velocidad demandada por el piloto), de la siguiente forma:

Se sabe que la motocicleta puede trabajar a $8000RPM$ máximo. Puesto que la señal del puño acelerador también es muestreada por el módulo ADC, el valor máximo es 1023. Con una simple regla de 3 se puede pasar del valor digital entre 0 y 1023 al valor en RPM realmente demandado. Este valor se multiplica por 2π y se divide por 60 para convertirlo a radianes por segundo.

Las figuras a continuación (Figura 39 y Figura 40) presentan el diagrama de bloques para la lectura del potenciómetro en el puño acelerador y el cálculo de la velocidad en un instante t. El cálculo de la velocidad

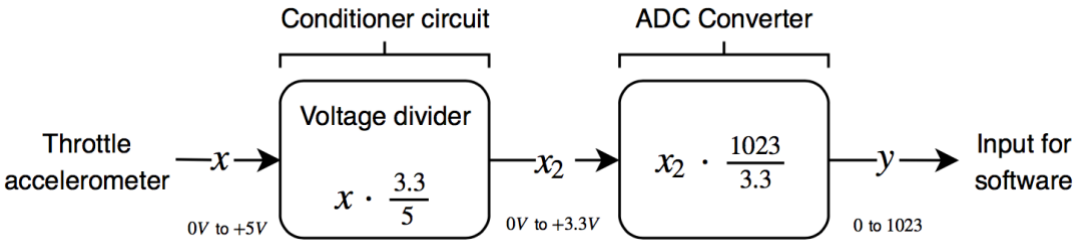


Figura 39 - Diagrama de bloques para la lectura de puño acelerador

El circuito de acondicionamiento hardware para conseguir el offset y el divisor de tensión no se diseñará.

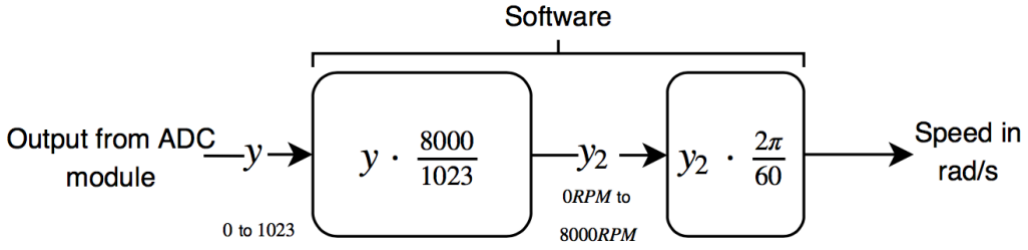


Figura 40 – Cálculo de la velocidad de referencia a partir de la señal muestreada

4.3.2 LECTURAS DE LAS CORRIENTES

Para leer las corrientes hay que utilizar sensores de corriente. Típicamente, para circuitos en lazo cerrado se utilizan sensores de efecto hall compensados. La Figura 41 presenta un esquema de dicho sensor.

Del mismo modo que para el resolver, se tiene que hacer un circuito de acondicionamiento para poder leer correctamente las corrientes, ya que el micro no acepta voltajes superiores a $3.3V$.

Estos sensores están clasificados según la corriente máxima que pueden leer. En este caso, se espera una corriente máxima de $271A$ (2.1.1), por lo que para tener un margen bastante amplio se escogerá el modelo que puede medir hasta $500A$.

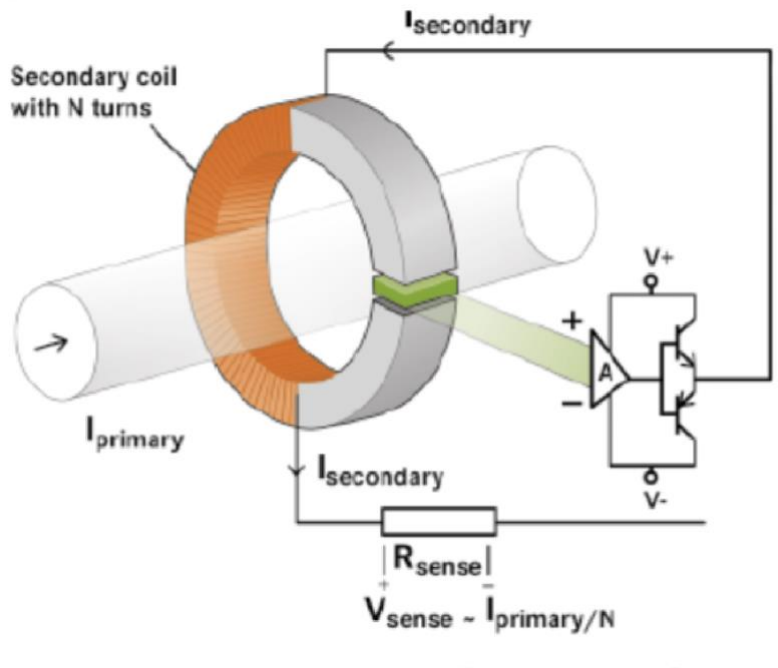


Figura 41 – Sensor de efecto hall para medir corrientes [27]

La Figura 42 siguiente muestra las características técnicas del sensor de corriente de efecto Hall HCS 300A.

I_{PN}	Nominal primary current	300 A			
I_P	Measuring range	0 ... ±500 A			
R_M	Burden resistance				
	with ±12 V	at ±300 A max	R_M min 0	R_M max	53 Ohm
		at ±500 A max	R_M min 0	R_M max	7 Ohm
	with ±15 V	at ±300 A max	R_M min 5	R_M max	90 Ohm
		at ±500 A max	R_M min 5	R_M max	40 Ohm
I_{SN}	Nominal secondary current	150 mA			
K_N	Turns ratio	1 : 2000			
V_C	Nominal power supply (±5 %)	±12 ... 24 V			
I_C	Supply current @ VC = 15 V	25+ I_S mA			
X	Overall accuracy at I_{PN} T_A = 25°C	±0.5 %			
E_L	Linearity	< 0.1 %			
I_O	Offset current at I_P = 0, T = 25 °C	max ±0.3 mA			
I_{OT}	Zero offset/temperatur, I_O, -40 °C ... 85 °C	max ±0.7 mA			
t_r	Delay time of I_{PN}	<1 µs			
Di/dt	di/dt correctly following	>100 A/µs			
f	Bandwidth	DC ... 100 kHz			
T_A	Operating temperature range	-40 ... +85 °C			
T_S	Storage temperature range	-45 ... +90 °C			
m	Weight	~ 0.25 kg			
RS	Coil resistance at T_A= 85 °C	35 Ohm			
V_D	Proof stress voltage, effective, 50 Hz, 1 minute	3 kV			
V_{st}	Rated impulse voltage 1.2/50 µs	10 kV			
V_B	Rated voltage 1)	600 V			

Figura 42 – Características del sensor de corriente HCS 300A [27]

Este sensor tiene una ratio de conversión de 2000, es decir, que para una corriente nominal de entrada de 300A, la salida nominal es de $\frac{300}{2000} = 150mA$.

Puesto que el microcontrolador solo puede leer hasta $3.3V$, se debe escoger la resistencia adaptada para que, al darse la corriente máxima, se de el voltaje máximo permitido.

Sin embargo, al tratarse de corrientes y al haber corrientes negativas, se puede elegir una resistencia tal que la tensión que salga sea fácil de manejar, como por ejemplo $\pm 5V$, para posteriormente añadir un offset y divisor de tensión.

Por lo tanto:

$$V = I \cdot R$$

$$R = \frac{5}{150 \cdot 10^{-3}} = 33.33\Omega$$

La figura siguiente (Figura 43) presenta el diagrama de bloques para la lectura de las corrientes:

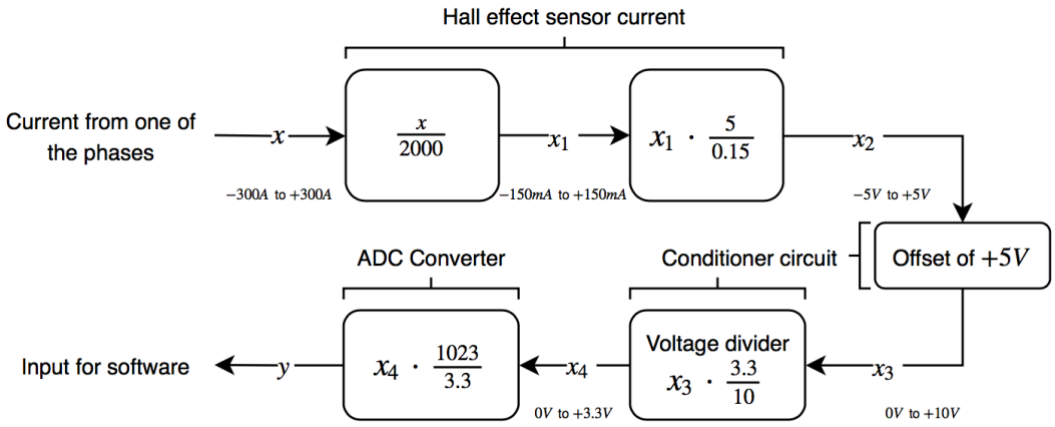


Figura 43 – Diagrama de bloques para la lectura de las corrientes

El circuito de acondicionamiento electrónico para conseguir el offset y el divisor de tensión no se diseñará.

Cuanto el microcontrolador haya muestreado las señales, se tiene que realizar el proceso inverso por software, es decir:

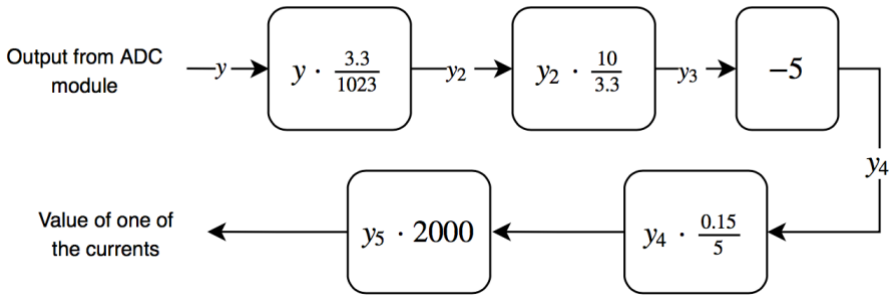


Figura 44 – Cálculo de las corrientes una vez muestreadas

4.3.3 TRANSFORMADAS CLARKE Y PARK

Se recuerda que las corrientes medidas son tres; una para cada fase. La suma vectorial de los flujos creados por cada fase es el vector de flujo resultante que se crea en el estator. Este sistema de referencia (las referencias son tres corrientes) se puede simplificar pasando a un sistema de dos referencias. Se llama la transformada de Clarke.

En la Figura 46, se puede ver como al vector de flujo total en el estator (i_s), es la suma de los flujos creado por corrientes de las fases a, b y c (i_a, i_b e i_c respectivamente). Se puede ver que el vector i_s se puede expresar mediante dos coordenadas, y por lo tanto se puede cambiar el sistema de referencia. Las dos referencias son ahora i_α e i_β.

La Figura 45 ilustra la transformada de Clarke en el tiempo.

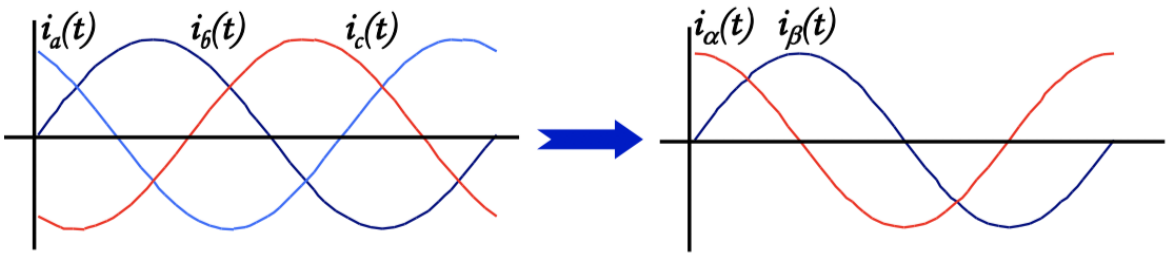


Figura 45 – Transformada de Clarke – Ejemplo el tiempo

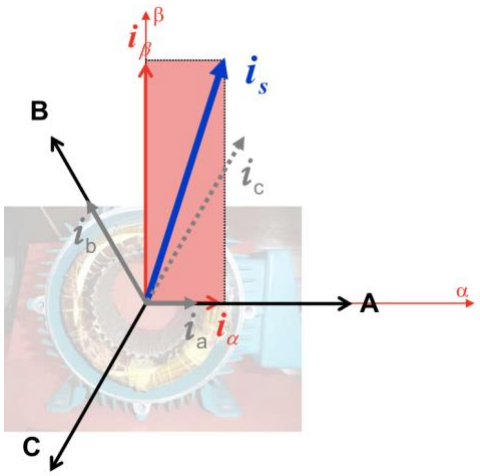


Figura 46 – Transformada de Clarke – Ejemplo vectorial

Los valores de i_α e i_β son:

$$i_\alpha = i_a$$

$$i_\beta = \frac{1}{\sqrt{3}} \cdot i_a + \frac{2}{\sqrt{3}} \cdot i_b$$

Aunque esto simplifica mucho los cálculos, queda todavía un problema. Conforme gira el rotor (y por consecuente el vector del estator también), los valores de i_α e i_β van a variar de forma senoidal, lo cual es difícil de corregir.

La transformada de Park permite expresar el vector de flujo del estator i_s en función de dos referencias **que se mueven con el rotor**, el eje d y el eje q. Esto hace que los valores ya no sean senoidales, sino que sean constantes (DC) y mucho más fáciles de manejar.

Las figuras siguientes (Figura 47 y Figura 48) representan este cambio de referencia de forma vectorial y temporal respectivamente.

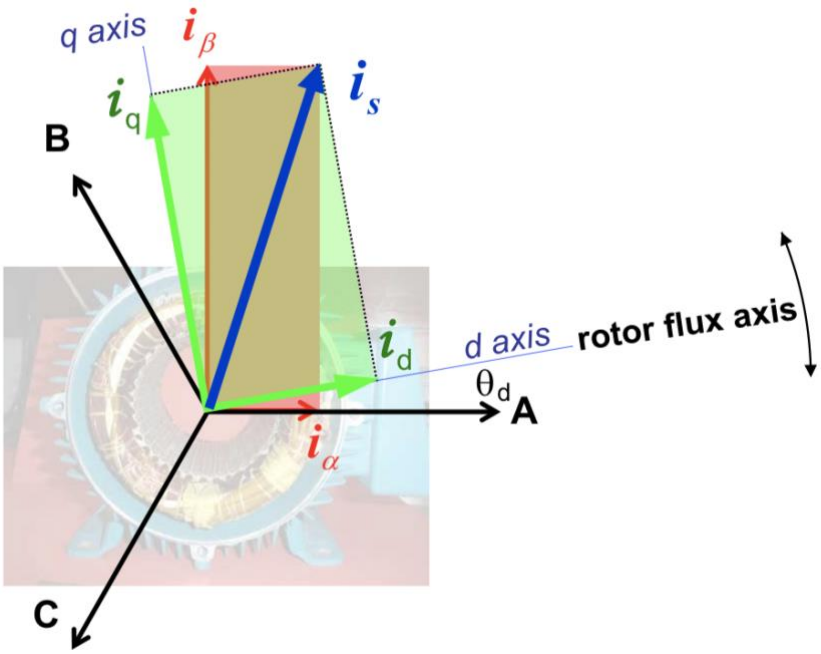


Figura 47 – Transformada de Park - Ejemplo vectorial

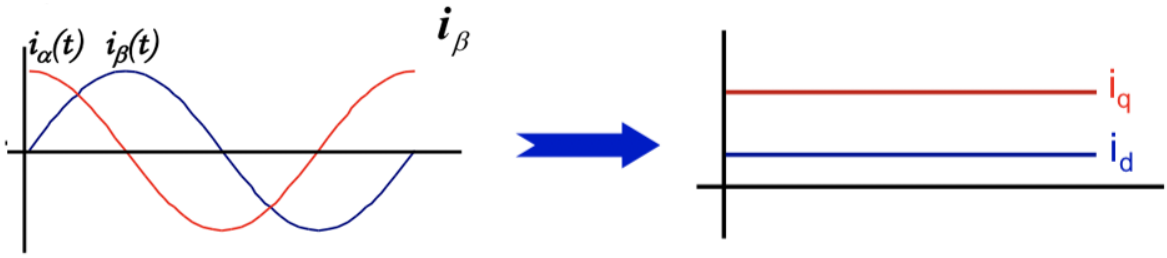


Figura 48 – Transformada de Park - Ejemplo en el tiempo

Los valores de i_d e i_q son:

$$i_d = i_\alpha \cdot \cos(\theta_d) + i_\beta \cdot \sin(\theta_d)$$

$$i_q = -i_\alpha \cdot \sin(\theta_d) + i_\beta \cdot \cos(\theta_d)$$

4.3.4 CONTROLES PID

Un controlador PID es un mecanismo de control por realimentación ampliamente usado en sistemas de control. Su función es anular el error entre el valor actual y el valor deseado.

Tiene tres parámetros distintos: el proporcional, el integral y el derivativo. Se necesitan dos controles, uno para la velocidad y otro para la corriente. Para ambos es suficiente un control PI [30], para los cuales se verán más en detalle los valores P e I en el próximo capítulo (Capítulo 5.).

Los valores obtenidos con la transformada de Park son los que se comparan con las referencias respectivas y se introducen en el regulador PI.

En el caso del eje q, la referencia es la demanda del piloto que, mediante el puño acelerador, indica cuanto quiere acelerar.

En cuanto al eje d, en condiciones normales todo el flujo necesario para el eje d lo suministran los imanes permanentes del rotor, por lo tanto, la referencia se pone a 0.

4.3.5 ALGORITMO SVPWM

Se sabe que el inversor se compone de 6 transistores:

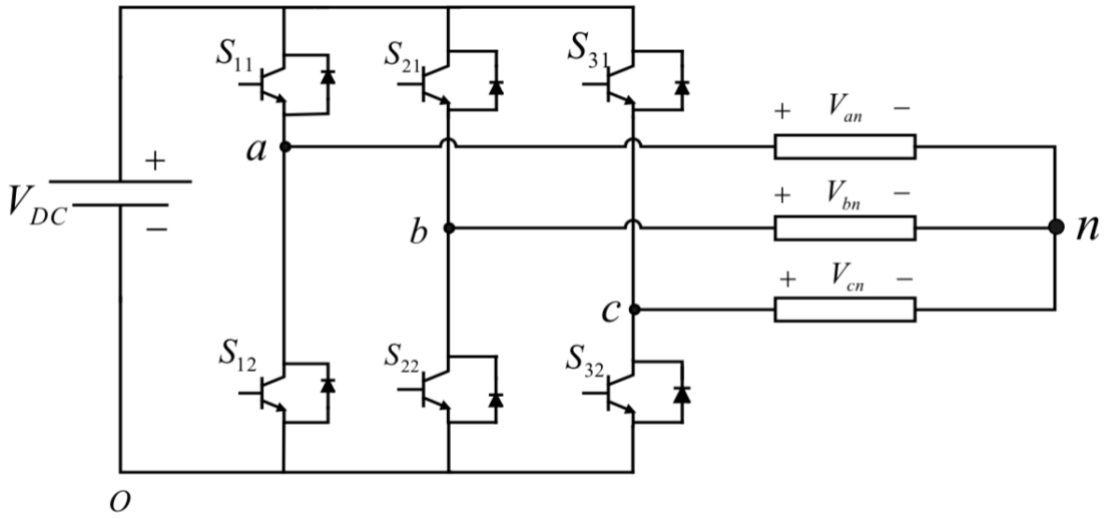


Figura 49 – Esquema de un inversor trifásico

En dicho inversor, dos transistores en serie en una rama no pueden estar **nunca** cerrados (ON) de forma simultánea por razones obvias. Tampoco es de mucha ayuda que estén ambos abiertos (OFF), ya que en ese caso el voltaje en la fase correspondiente no está definido. Es decir, en cada momento solo puede haber un transistor de cada rama cerrado (ON).

Por estas razones el inversor tiene un total de 8 combinaciones ON y OFF posibles. En la Tabla 4 se han elegido los transistores superiores (S_{11}, S_{21} y S_{31}) para representar los 8 estados posibles en los que puede estar el inversor, ya que, si el transistor superior está en un estado cualquiera, el inferior tiene que estar en el estado opuesto. Esto hace que el sistema sea idéntico a otro en el que cada rama se represente por un solo switch con dos estados posibles, por lo que se ve claramente que hay $2^3 = 8$ combinaciones posibles, llamadas de $\vec{V}_0$ a $\vec{V}_7$.

Seis de los estados posibles corresponden a diferentes tensiones aplicadas al motor y se llaman *vectores básicos*. Los otros dos se llaman *vectores nulos* ya que representan voltaje

nulo en los bornes. En binario, estas combinaciones de vectores se pueden escribir como se ve en la última fila de la Tabla 4.

Estados / Switch	$\vec{V}_0$	$\vec{V}_1$	$\vec{V}_2$	$\vec{V}_3$	$\vec{V}_4$	$\vec{V}_5$	$\vec{V}_6$	$\vec{V}_7$
11	ON	ON	ON	ON	OFF	OFF	OFF	OFF
21	ON	ON	OFF	OFF	ON	ON	OFF	OFF
31	ON	OFF	ON	OFF	ON	OFF	ON	OFF
Binario	000	001	010	011	100	101	110	111

Tabla 4 – Estados posibles de los transistores en el inversor

Se recuerda que las transformadas de Clarke y Park (4.3.3) permiten pasar de un sistema de referencia trifásico a un sistema de referencia de dos ejes (d y q) que se mueve con el rotor (rectángulo azul inferior de la Figura 50). Por otro lado, la demanda de velocidad del piloto se traduce a demanda de par, es decir, una corriente según el eje q (rectángulo azul superior de la Figura 50). Finalmente, también se recuerda que, en condiciones normales, $i_{d_{ref}} = 0$.

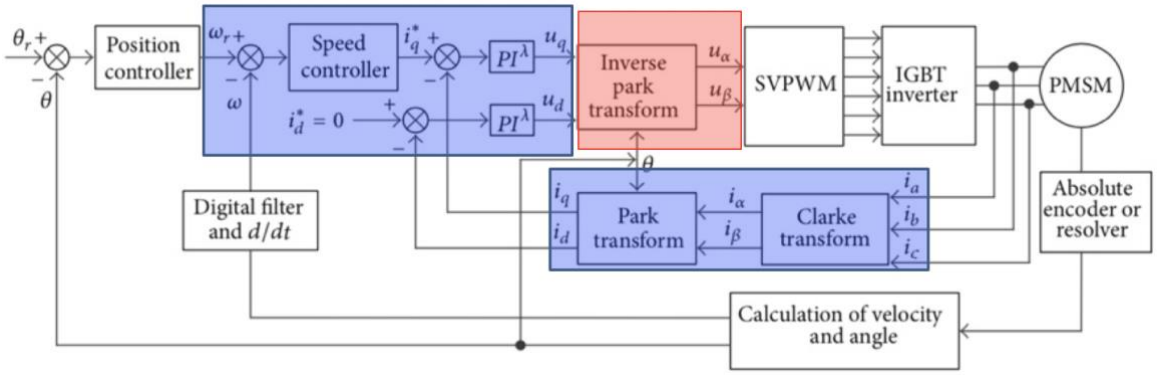


Figura 50 – Esquema de bloques del control de un motor PMSM con ciertos bloques indicados

Primero se comparan las corrientes de eje d y de eje q de referencia con las corrientes medidas. Los controles PI colocados a continuación permiten obtener **las tensiones de referencia** (u_q y u_d) que se pasan a una referencia fija de dos ejes (α y β) con la transformada inversa de Park (u_α y u_β, rectángulo rojo de la Figura 50).

En esta referencia (α y β) también se han representado los 8 posibles estados del inversor de la Tabla 4. Este se puede ver en la Figura 51.

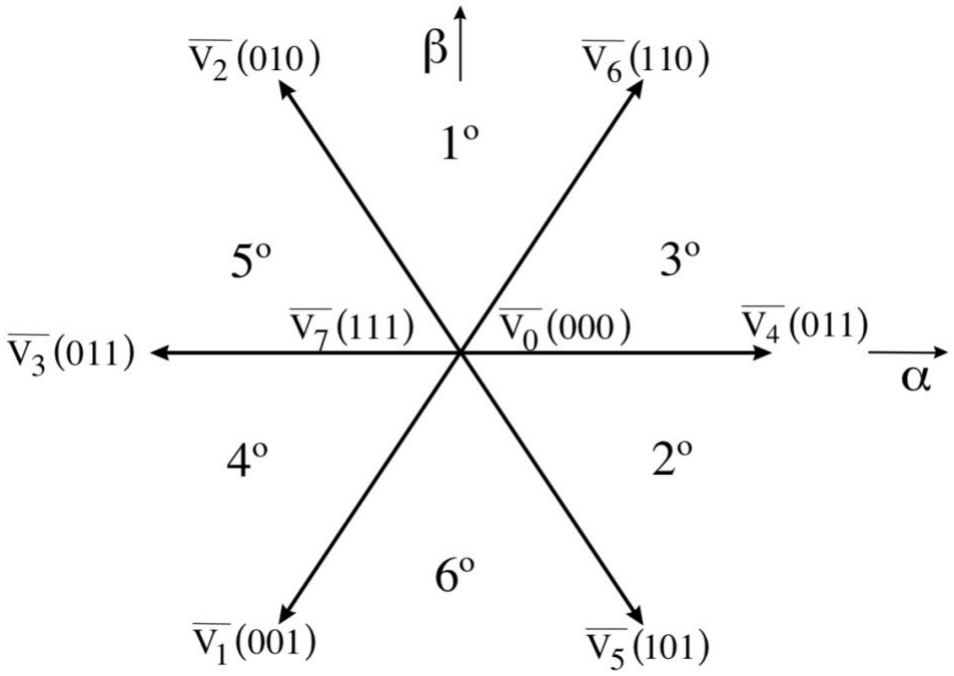


Figura 51 – Representación espacial de los posibles estados del inversor

El origen de los ángulos es la estructura geométrica de los devanados en el interior del motor, que, como se ha visto en el apartado 4.2, tienen que formar tres fases separadas de 120º. Como cada devanado puede tener una tensión positiva y negativa, hay una separación de 180º entre estados opuestos.

Se ha explicado en el apartado 4.2.2 que el campo creado por el estator debe colocarse a 90º del campo creado por el rotor, que se representa según el eje d. La demanda de velocidad del piloto influye principalmente en la **magnitud** del vector creado por el estator. Aunque a mayor magnitud, más par, y si aumenta el par, la velocidad rotacional aumenta y la distancia angular recorrida también. Es decir que cada vez hay que mover el campo creado por el estator **más rápido**, y **más lejos** con respecto a la posición anterior, aunque se mantenga la separación de 90º.

Hasta este punto, se sabe que se obtienen dos vectores u_α y u_β que forman la tensión de referencia – la demandada – para conseguir el par deseado por el piloto. Esta tensión puede estar en cualquier lugar de la Figura 51. En dicha figura se observa que se han denominado distintas regiones. Dos vectores pertenecientes a una misma región se dicen *adyacentes*.

Como raramente coincide el vector de la tensión de referencia (se llamará $\vec{V}_{s_{ref}}$ a partir de ahora para aligerar la lectura) con alguno de los vectores básicos, hay que alternar entre dos vectores adyacentes para conseguir la tensión deseada. Cada vector básico se deberá multiplicar por una constante que permita alcanzar la tensión de referencia. Esa constante es simplemente el tiempo en el que tienen que estar activos los transistores correspondientes a ese vector básico.

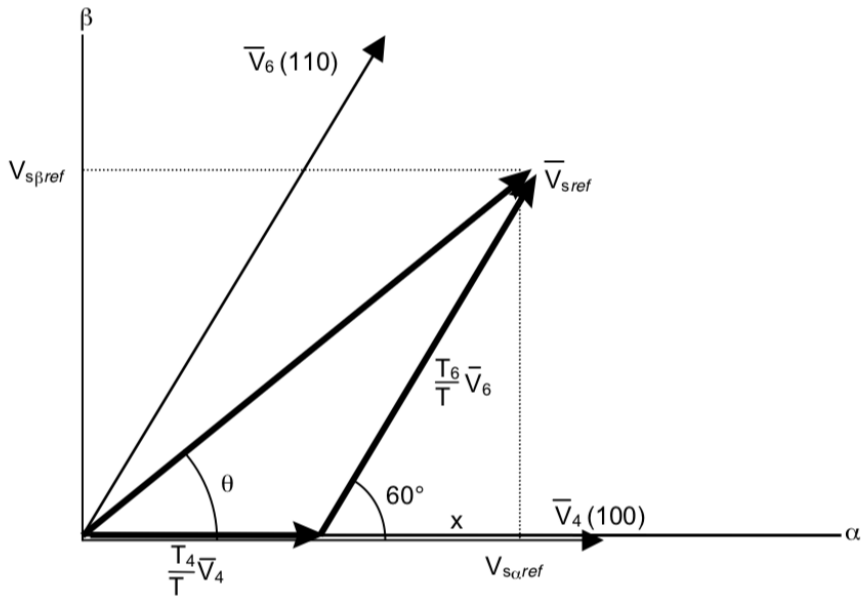


Figura 52 – Tensión de referencia expresada por la combinación de vectores básicos

La Figura 52 muestra un ejemplo de tensión de referencia. Se encuentra en la región 3, por lo que los vectores adyacentes son $\vec{V}_6$ y $\vec{V}_4$. Como sugiere la figura, $\vec{V}_{s_{ref}}$ se puede descomponer en:

$$\frac{T_4}{T}\vec{V}_4 + \frac{T_6}{T}\vec{V}_6$$

Donde T_4 y T_6 son los tiempos en los que estarán activos los transistores correspondientes. Por ejemplo, en T_4, que se relaciona con $\vec{V}_4$ (100), estará activa la primera rama y las dos siguientes abiertas (Tabla 4).

Por lo tanto, queda generalizar los siguientes pasos:

- Calcular el valor de los vectores básicos (estos valores no cambian) en función del sistema de referencias de los ejes α y β,
- Calcular los distintos tiempos en los que hay que aplicar los vectores básicos.
- Identificar en qué región está $\vec{V}_{s_{ref}}$ para determinar qué vectores básicos utilizar.

Recordando la figura del inversor trifásico (Figura 49), se puede expresar la tensión de entrada V_{DC} de siguiente forma:

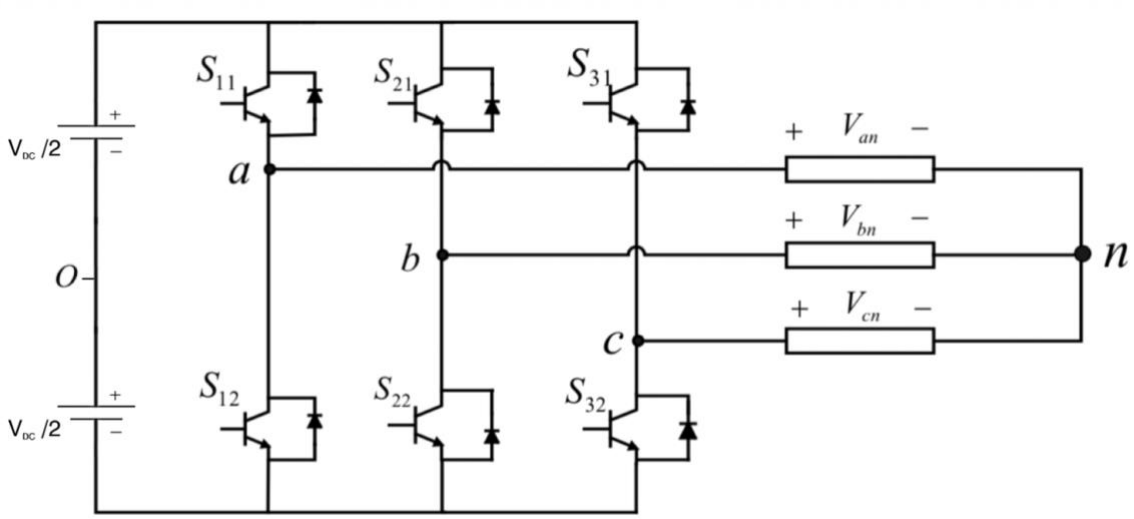


Figura 53 – Esquema de un inversor trifásico

Se pueden escribir las tensiones de salida V_{Oa}, V_{Ob} y V_{Oc} de la siguiente forma (Tabla 5):

a	b	c	V_{Oa}	V_{Ob}	V_{Oc}
0	0	0	$-\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$
0	0	1	$-\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$
0	1	0	$-\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$
0	1	1	$-\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$
1	0	0	$+\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$
1	0	1	$+\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$
1	1	0	$+\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{2}$
1	1	1	$+\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$	$+\frac{V_{DC}}{2}$

Tabla 5 – Tensiones de salida del inversor trifásico

De [42] se sabe que las tensiones V_{an}, V_{bn} y V_{cn} se pueden escribir de la siguiente forma:

$$V_{an} = \frac{1}{3}(2V_{Oa} - V_{Ob} - V_{Oc})$$

$$V_{bn} = \frac{1}{3}(2V_{Ob} - V_{Oa} + V_{Oc})$$

$$V_{cn} = \frac{1}{3}(2V_{Oc} - V_{Oa} - V_{Ob})$$

Por lo que las tensiones resultantes en cada estado posible se pueden ver en la Tabla 6.

Como las tensiones de referencia u_α y u_β están, como ellas mismas lo indican, en dos referencias fijas (ejes α y β), hay que escribir las tensiones de la Tabla 6 según estas referencias.

Dichas tensiones se pueden ver en la Tabla 7, Puesto que:

$$\begin{bmatrix} V_{s\alpha} \\ V_{s\beta} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_{an} \\ V_{bn} \\ V_{cn} \end{bmatrix}$$

a	b	c	V_{an}	V_{bn}	V_{cn}
0	0	0	0	0	0
0	0	1	$-\frac{V_{DC}}{3}$	$-\frac{V_{DC}}{3}$	$+\frac{2V_{DC}}{3}$
0	1	0	$-\frac{V_{DC}}{3}$	$+\frac{2V_{DC}}{3}$	$-\frac{V_{DC}}{3}$
0	1	1	$-\frac{2V_{DC}}{3}$	$+\frac{V_{DC}}{3}$	$+\frac{V_{DC}}{3}$
1	0	0	$+\frac{V_{DC}}{2}$	$-\frac{V_{DC}}{3}$	$-\frac{V_{DC}}{3}$
1	0	1	$+\frac{V_{DC}}{3}$	$-\frac{2V_{DC}}{3}$	$+\frac{V_{DC}}{3}$
1	1	0	$+\frac{V_{DC}}{3}$	$+\frac{V_{DC}}{3}$	$-\frac{2V_{DC}}{3}$
1	1	1	0	0	0

Tabla 6 – Tensiones neutras de salida del inversor

a	b	c	V_{α}	V_{β}	
0	0	0	0	0	$\vec{V}_0$
0	0	1	$-\frac{V_{DC}}{3}$	$-\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_1$
0	1	0	$-\frac{V_{DC}}{3}$	$+\frac{2V_{DC}}{\sqrt{3}}$	$\vec{V}_2$
0	1	1	$-\frac{2V_{DC}}{3}$	0	$\vec{V}_3$
1	0	0	$+\frac{2V_{DC}}{3}$	0	$\vec{V}_4$
1	0	1	$+\frac{V_{DC}}{3}$	$-\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_5$
1	1	0	$+\frac{V_{DC}}{3}$	$+\frac{V_{DC}}{\sqrt{3}}$	$\vec{V}_6$
1	1	1	0	0	$\vec{V}_7$

Tabla 7 – Tensiones de salida del inversor expresadas en ejes α y β

Retomando la Figura 52, en la que se podía ver que $\vec{V}_{s_{ref}}$ estaba en el tercer sector, se establece que:

$$T = T_4 + T_6 + T_0$$

$$\vec{V}_{s_{ref}} = \frac{T_4}{T}\vec{V}_4 + \frac{T_6}{T}\vec{V}_6$$

Se puede ver que la determinación de los tiempos T_4 y T_6 viene dada por simples proyecciones:

$$V_{s\beta_{ref}} = \frac{T_6}{T}\left\|\vec{V}_6\right\| \cdot \cos(30º)$$

$$V_{s\alpha_{ref}} = \frac{T_4}{T}\left\|\vec{V}_4\right\| + x$$

$$x = \frac{V_{s\beta_{ref}}}{tg(60)}$$

Finalmente, puesto que las componentes (α, β) de cada vector básico ya se han dado en la Tabla 7, reemplazando $\vec{V}_6$ y $\vec{V}_4$:

$$T_4 = \frac{T}{2V_{DC}}\left(3V_{s\alpha_{ref}} - \sqrt{3}V_{s\beta_{ref}}\right)$$

$$T_6 = \sqrt{3}\frac{T}{V_{DC}}V_{s\beta_{ref}}$$

Para cada sector, un tiempo de conmutación se calcula. Esos tiempos de conmutación se pueden escribir y relacionar todos a través de las siguientes variables:

$$X = \sqrt{3} \cdot \frac{T}{V_{DC}} \cdot V_{s\beta_{ref}}$$

$$Y = \frac{\sqrt{3}}{2} \cdot \frac{T}{V_{DC}} \cdot V_{s\beta_{ref}} + \frac{3}{2} \cdot \frac{T}{V_{DC}} \cdot V_{s\alpha_{ref}}$$

$$Z = \frac{\sqrt{3}}{2} \cdot \frac{T}{V_{DC}} \cdot V_{s\beta_{ref}} - \frac{3}{2} \cdot \frac{T}{V_{DC}} \cdot V_{s\alpha_{ref}}$$

En el ejemplo anterior, la región 3 corresponde a $T_4 = -Z$ y $T_6 = X$.

El último paso es identificar en qué región se encuentra $\vec{V}_{s_{ref}}$. Para determinar el sector, se pueden calcular las proyecciones V_a, V_b y V_c de la tensión de referencia y compararlas con 0.

Estas proyecciones son dadas por la transformada inversa de Park:

$$V_a = V_{s\beta_{ref}}$$

$$V_b = \frac{1}{2}(\sqrt{3}V_{s\alpha_{ref}} - V_{s\beta_{ref}})$$

$$V_c = \frac{1}{2}(-\sqrt{3}V_{s\alpha_{ref}} - V_{s\beta_{ref}})$$

Un simple algoritmo se puede implementar entonces para determinar el sector (el código y su contexto se explica en detalle en el Capítulo 6.):

```
// Locate the sector
            if (v_a_ref > 0){
                temp_a = 1;
            }else{
                temp_a = 0;
            }

            if (v_b_ref > 0){
                temp_b = 1;
            }else{
                temp_b = 0;
            }

            if (v_c_ref > 0){
                temp_c = 1;
            }else{
                temp_c = 0;
            }

            sector = temp_a + 2*temp_b + 4*temp_c;
```

Finalmente, solo hay que asignar el factor de servicio (t_x_on) a la fase del motor que corresponda, es decir a los registros PDCx correctos:

Sector / Registro	1	2	3	4	5	6
PDC1	t_b_on	t_a_on	t_a_on	t_c_on	t_c_on	t_b_on
PDC2	t_a_on	t_c_on	t_b_on	t_b_on	t_a_on	t_c_on
PDC3	t_c_on	t_b_on	t_c_on	t_a_on	t_b_on	t_a_on

Tabla 8 – Asignación de los factores de servicio a los registros del módulo PWM

4.3.6 FRENO REGENERATIVO

Para poder controlar correctamente cuándo y cómo de fuerte frena el piloto, se pueden instalar dos sensores de efecto hall tanto en la maneta de freno como en el pedal de freno. Estos sensores se componen de dos elementos:

- El propio sensor que se instala en la parte fija del mecanismo de freno,
- Un pequeño imán que se instala en la parte móvil del mecanismo de freno.

Cuando el imán se acerca al sensor, la tensión de salida aumenta hasta un máximo de $5V$ (por lo tanto, también hay que acondicionar este circuito).

A continuación, se muestra cómo se realiza el muestro de una señal de frenado (Figura 54) y el cálculo de la demanda de frenado (Figura 55).

Para el resto de este trabajo se ha asumido que en la posición más alejada, el sensor da una tensión de $1V$. Esto implica que el valor mínimo de x_2 es:

$$1 \cdot \frac{3.3}{5} = 0.66$$

Por lo que el valor mínimo que puede dar el conversor AD es:

$$0.66 \cdot \frac{1023}{3.3} = 204.6$$

Por este motivo existe un offset de 205 en el cálculo de la demanda (Figura 55). El resultado se normaliza y se obtiene un valor entre 0 (no se frena) y 1 (frenada máxima).

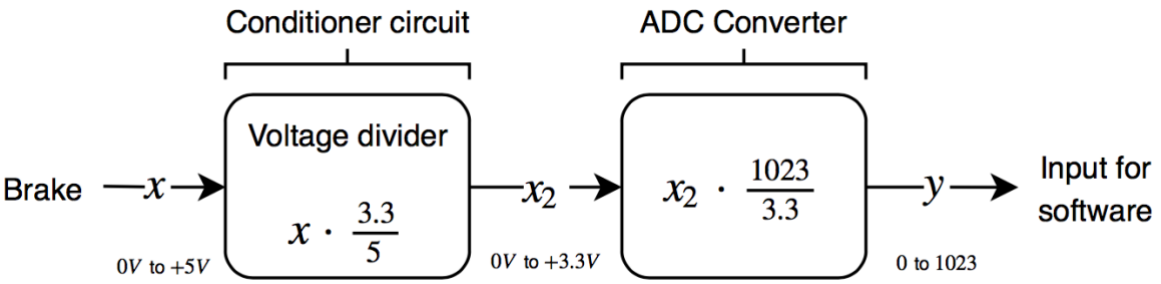


Figura 54 – Diagrama de bloques para la lectura de las señales de frenado

El circuito de acondicionamiento hardware para conseguir el divisor de tensión no se diseñará.

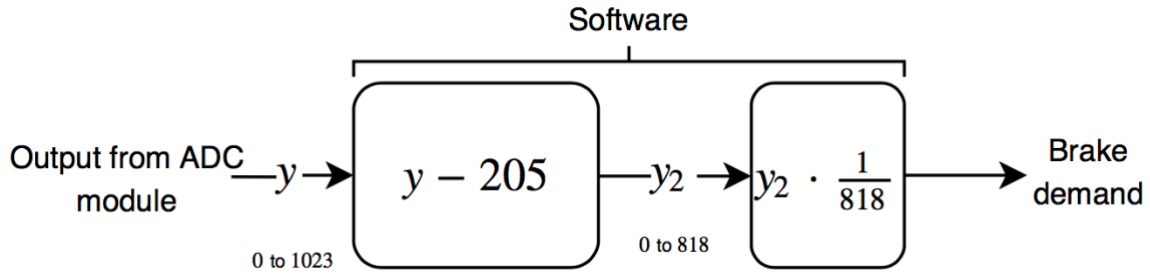


Figura 55 – Cálculo de la demanda de frenado

Como se ha comentado anteriormente, las corrientes del estator se miden y se pasan a un sistema de referencia que se mueve con el rotor (ejes d y q). La componente d indica la cantidad de flujo y la componente q se utiliza para controlar la cantidad de par que se requiere. También se ha comentado que en condiciones normales y para conseguir el par máximo, $i_d = 0$.

El freno eléctrico basado en FOC se puede realizar de distintas formas. En este trabajo se ha elegido utilizar el control de velocidad.

Cuando se detecta que se quiere frenar, la referencia de velocidad pasa a ser inferior a la actual, por lo que provoca una pendiente de velocidad negativa. El resto del algoritmo de control se encarga de calcular las corrientes necesarias para esa velocidad.

Por defecto, en cuanto el piloto deje de acelerar (es decir, que el puño acelerador vuelve a su posición inicial), se entra en estado de freno regenerativo. La máxima frenada eléctrica se programará para que se consiga tan solo si se frena al máximo con uno de los dos frenos.

Esto se explica en el Capítulo 6. .

Capítulo 5. SIMULACIÓN DEL CONTROL DE UN MOTOR PMSM

En el apartado 3.3, se han comentado los distintos parámetros que hacen falta para empezar a simular el motor PMSM con Simulink.

Este capítulo tiene como objetivo calcular los parámetros de los distintos controles PI que contiene el sistema de control, de forma que la respuesta al impulso sea adecuada. Además, se estudiarán las gráficas de corrientes, velocidad y par para comprobar que todo funciona correctamente.

5.1 PARÁMETROS DE LA SIMULACIÓN

Se ha utilizado un modelo de Simulink que permite controlar el motor directamente con las tensiones de referencia u_d y u_q. Dichas tensiones se calculan de la misma manera que se ha visto en los capítulos anteriores.

Se han encontrado numerosos problemas al simular, ya que se alcanzaban tensiones muy elevadas muy rápido, la corriente no era todo lo alta que debería ser y existían muchas inconsistencias con respecto a la hoja de características proporcionada por la competición.

Retomando la gráfica proporcionada por la competición (Figura 53), se pueden escoger dos o tres puntos para intentar relacionar los distintos parámetros. Como la gráfica es de calidad bastante pobre, se han dibujado los puntos para facilitar la lectura.

Torque [Nm]	Amps	Watts	Volts	RPM
20	100	6200	71	3000
12	60	3800	71	3000
6	30	2000	71	3000

Tabla 9 – Puntos correspondientes a la Figura 56

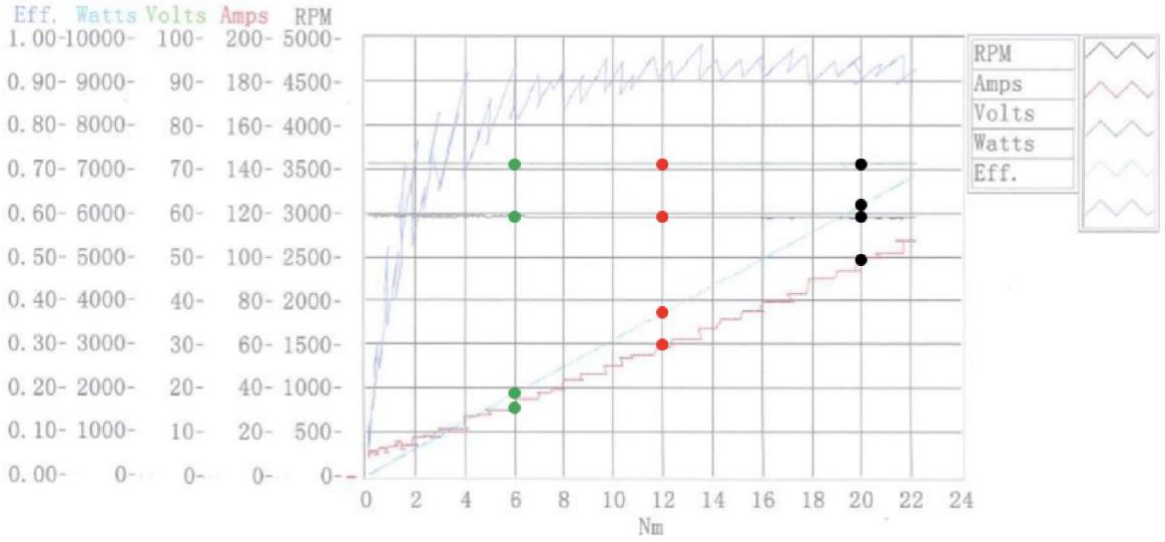


Figura 56 – Gráfica que relaciona potencia, tensión, corriente y RPM del motor de la competición en función del par

Las ecuaciones del motor PMSM que relacionan tensiones, corrientes y par son las siguientes:

$$V_d + I_q \omega_e L_q - R_s I_d = L_d \cdot \frac{d_{i_d}}{d_t} \quad (28)$$

$$V_q - \omega_e PM - \omega_e L_d I_d - R_s i_q = L_q \cdot \frac{d_{i_q}}{d_t} \quad (29)$$

Donde:

- V_q es la tensión de eje q
- V_d es la tensión de eje d
- I_q es la corriente de eje q
- ω_e son los RPM del motor en rad/s
- L_q es la inductancia del eje q del motor
- PM es en enlace de flujo concatenado (calculado en (23))

Despreciando los términos casi nulos, se obtiene:

$$V_d + I_q \omega_e L_q = 0$$

$$V_q - \omega_e PM - R_s i_q = 0$$

Y finalmente:

$$V_q^2 + V_d^2 = \left(I_q \cdot \omega_e \cdot L_q\right)^2 + (\omega_e \cdot PM)^2 \quad (30)$$

Se sabe que $PM = 0.03$, y que $\sqrt{V_q^2 + V_d^2} = V_{AC_{nom}} \cdot \sqrt{2}$, calculado en (2) ($33.8V$).

Si se despeja la corriente $I_{q_{max}}$ de (30), tomando $\omega_{e_{max}} = \frac{8000 \cdot 2\pi \cdot N}{60}$, se obtiene una corriente compleja, lo que no tiene mucho sentido.

Si se despeja PM de (30) tomando I_q del resultado de (27) (271.69), también se obtiene un resultado complejo.

Por otro lado, retomando la Tabla 9, tampoco se específica que tipo de corriente se está dando. Asumiendo que se trata de la corriente eficaz, hay que multiplicar por la raíz de 2. De la misma forma, asumiendo que la tensión que se da es la tensión DC (ya que se ha visto de (3) que la máxima tensión en AC es de $\sim 47V$), hay que multiplicar ese valor por 0.61 y dividirlo por la raíz de 3. De todas formas, tampoco tendría mucho sentido que fuese la tensión DC ya que el rango de tensiones para el funcionamiento del motor, según su ficha técnica es entre 96 y 116VDC.

Si se despeja PM de (30), tomando $\omega_{e_{max}} = \frac{3000 \cdot 2\pi \cdot N}{60}$ y utilizando los valores de dicha tabla se obtiene:

Torque [Nm]	Amps	Watts	Volts	RPM	PM
20	100	6200	71	3000	0.0163
12	60	3800	71	3000	0.0205
6	30	2000	71	3000	0.0220

Tabla 10 – Cálculo de PM en función de la gráfica del motor de la competición

Se puede ver que el valor de PM no queda constante y por lo tanto dicha gráfica no parece tener mucho sentido.

Por otro lado, la potencia viene dada por la siguiente expresión:

$$P = \omega_e \cdot K$$

K siendo el par, y $\omega_e = \frac{RPM \cdot 2\pi \cdot N}{60}$

Si se calcula la potencia a partir de los datos de la tabla anterior, se obtiene:

Torque [Nm]	RPM	Watts
20	3000	3141
12	3000	1885
6	3000	942

Tabla 11 – Cálculo de par en función de RPM y vatios

Se observa como existe un factor de 2, de nuevo mostrando algún tipo de incoherencia entre los datos proporcionados.

Se ha decidido simular el control de todas formas, asumiendo que se parecen lo más posibles a los del motor. Los parámetros elegidos son los siguientes:

Símbolo	Valor	Unidades	Nomenclatura
K_T	0.15	$\frac{N \cdot m}{A}$	Constante de par
N	5	-	Número de pares de polos
PM	0.03	$\frac{N \cdot m}{A}$	Enlace de flujos concatenados
L_d	$0.062 \cdot 10^{-3}$	H	Inductancia del eje d
L_q	$0.110 \cdot 10^{-3}$	H	Inductancia del eje q
R_s	0.0027	Ω	Resistencia entre fases

V_{DC}	96	V	Tensión DC
RPM_{max}	8000	-	Máximas RPM del motor
J	1.26	$kg \cdot m^2$	Inercia equivalente
$V_{AC_{simple_{RMS}}}$	33.8	V	Tensión nominal AC
$V_{AC_{simple_{RMS_{peak}}}}$	47.8	V	Tensión de pico nominal AC
P_{nom}	12000	W	Potencia nominal
P_{max}	20000	W	Potencia máxima
I_{nom}	118.3	A	Corriente nominal
I_{max}	197	A	Corriente máxima
$I_{nom_{peak}}$	167.31	A	Corriente de pico nominal
$I_{max_{peak}}$	278.85	A	Corriente de pico máxima
$\omega_{e_{max}}$	4188	rad/s	Revoluciones máximas del motor
T_{nom}	75.29	Nm	Par nominal
T_{max}	125.48	Nm	Par máximo

Tabla 12 – Parámetros utilizados para la simulación

5.2 SIMULINK

La Figura 57 muestra la vista desde más alto nivel del modelo de Simulink que se ha diseñado y utilizado para simular el motor. El rectángulo de color violeta es el modelo del motor de imanes permanentes, que contiene las ecuaciones diferenciales descritas en (28) y (29). Por debajo de esta 'caja', se encuentra lo que se ve en la Figura 58.

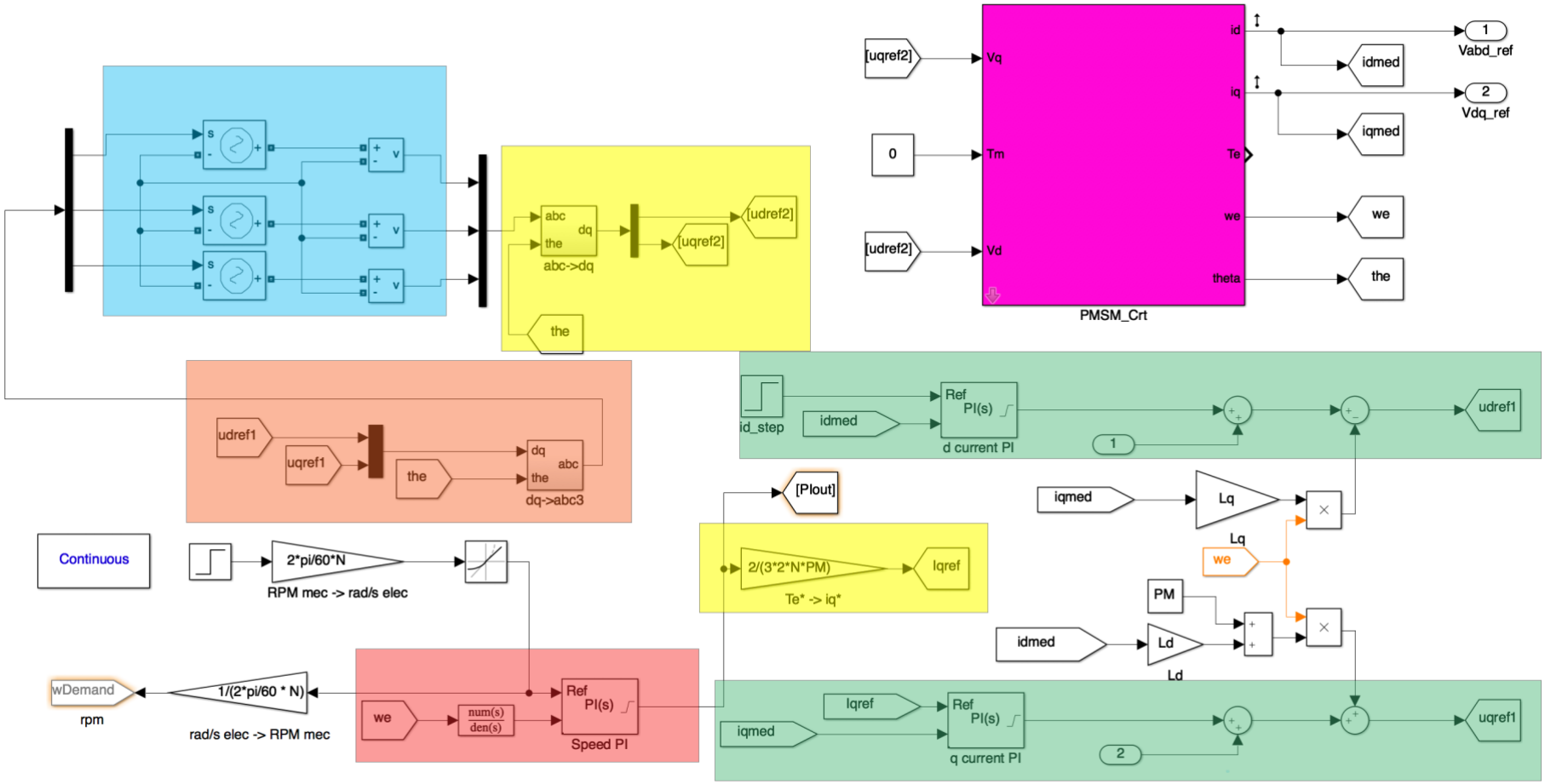


Figura 57 – Vista más alta del modelo utilizado para simular el control de un motor PMSM

Se ha escogido trabajar con un modelo de motor que reciba directamente las tensiones de referencia v_d y v_q. En la realidad, un motor recibe las tensiones trifásicas (una para cada fase). Para compensar esto, las tensiones v_d y v_q que salen del control no son enchufadas directamente al motor, sino que se convierten a tensiones trifásicas para simular el papel del inversor y a continuación se vuelven a pasar en tensiones v_d y v_q que van al motor.

El control empieza con un el bloque rojo en la parte inferior de la figura. Ese es el control de velocidad. Se recibe un comando en RPM, que se transforma a radianes por segundo y se limita a la máxima aceleración física del motor.

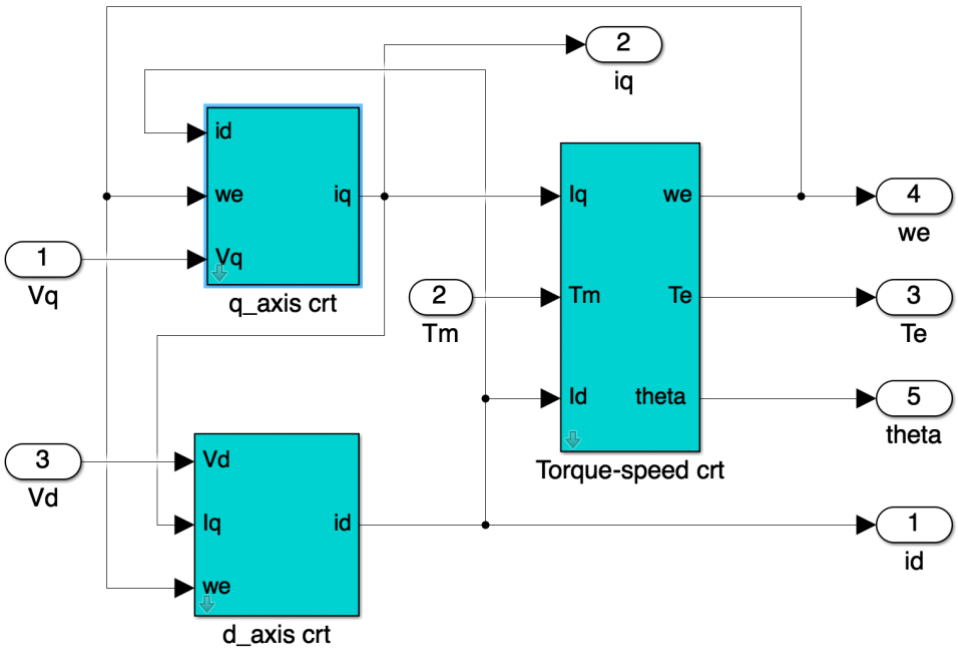


Figura 58 – Modelo de Simulink del motor PMSM

Se compara con la velocidad actual en RPM del motor con lo que se obtiene el par deseado. Se transforma el par en corriente i_q (rectángulo amarillo inferior).

A continuación, están los controles de corriente (rectángulos verdes). El superior es el de la corriente en eje d y el inferior para las corrientes en eje q. Se puede ver que para el control de la corriente d existe un "step". Esto es simplemente para poner la referencia a 0.

El bloque entre medias es el desacoplo de las dinámicas de los ejes d y q. Con esas operaciones, la tensión u_d solo influye en la corriente i_d, mientras que la tensión u_q sólo influye en la corriente i_q. Este cálculo se hace con las tensiones sugeridas por los reguladores, para compensar la influencia cruzada entre los ejes, que es conocida.

Con las tensiones de referencia a la salida de los controles PI de corriente, se puede ver que en el bloque de color naranja se realiza la transformada de Park para pasar a tensiones trifásicas, las cuales se generan en el bloque azul. Finalmente, en el bloque amarillo superior se vuelven a pasar a tensiones en ejes d y q y se envían al motor.

Se ha simulado el arranque de la motocicleta.

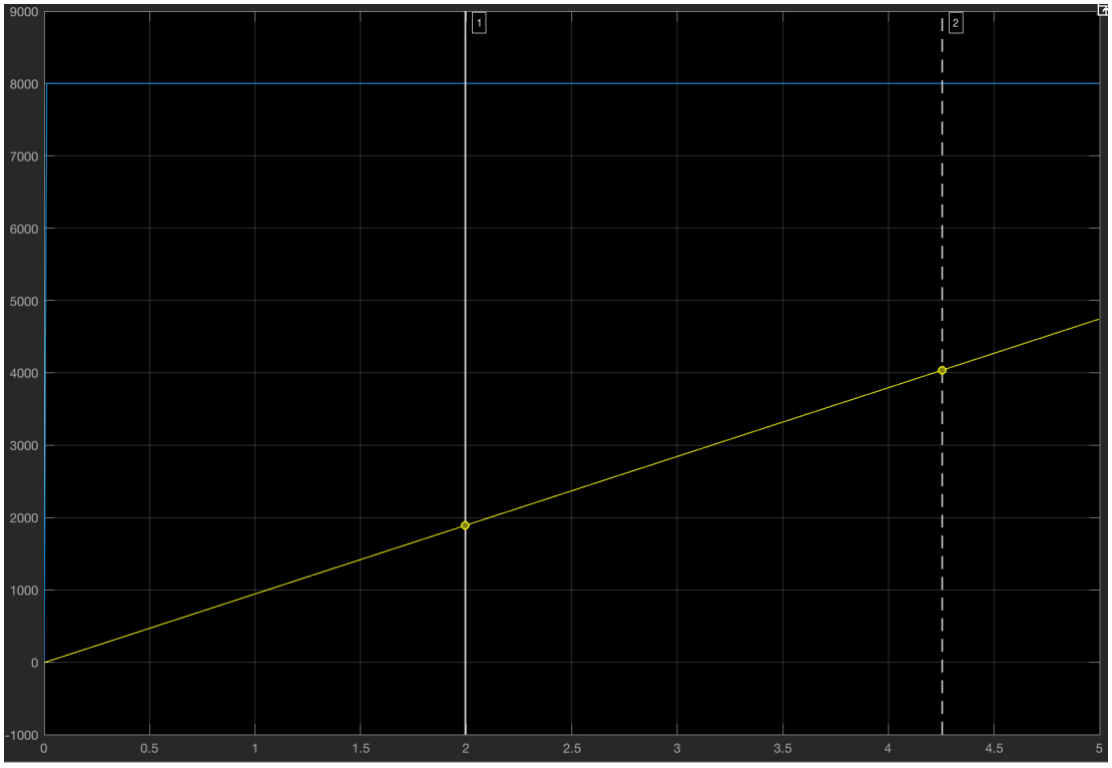


Figura 59 – Simulación del arranque de la motocicleta

En la Figura 59 se puede ver que la motocicleta alcanza los 4000RPM (100 km/h) en 4.2s.

La Figura 60 muestra las corrientes trifásicas en el momento de dicha aceleración. Se puede ver como se alcanza esa corriente máxima calculada previamente (278A).

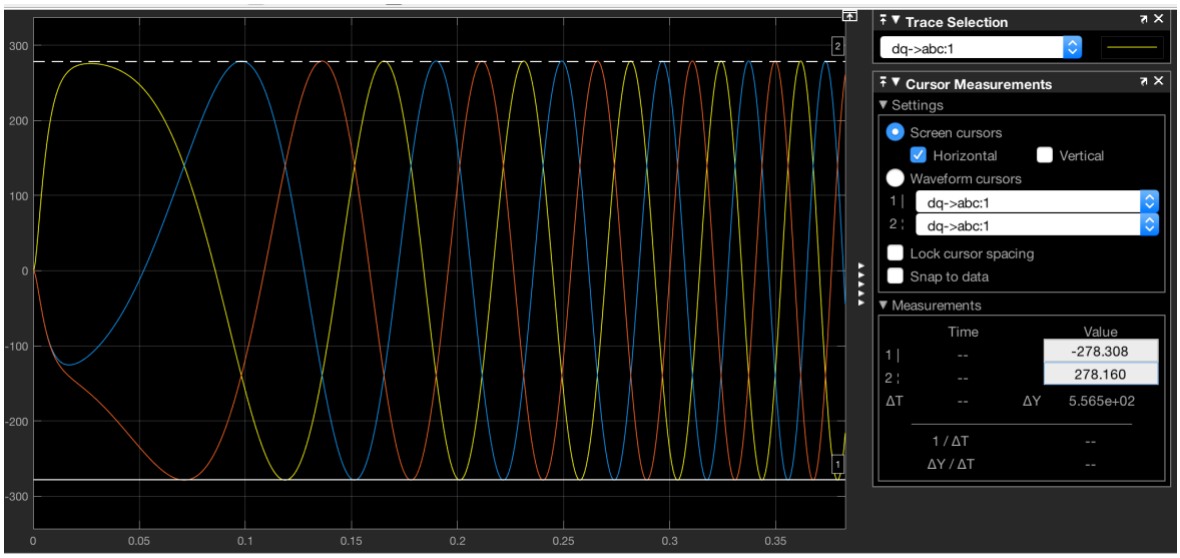


Figura 60 – Corrientes trifásicas en el arranque de la motocicleta

También se comprueba que la corriente del eje d es casi nula (nótese que la escala vertical está $\cdot 10^{-12}$)

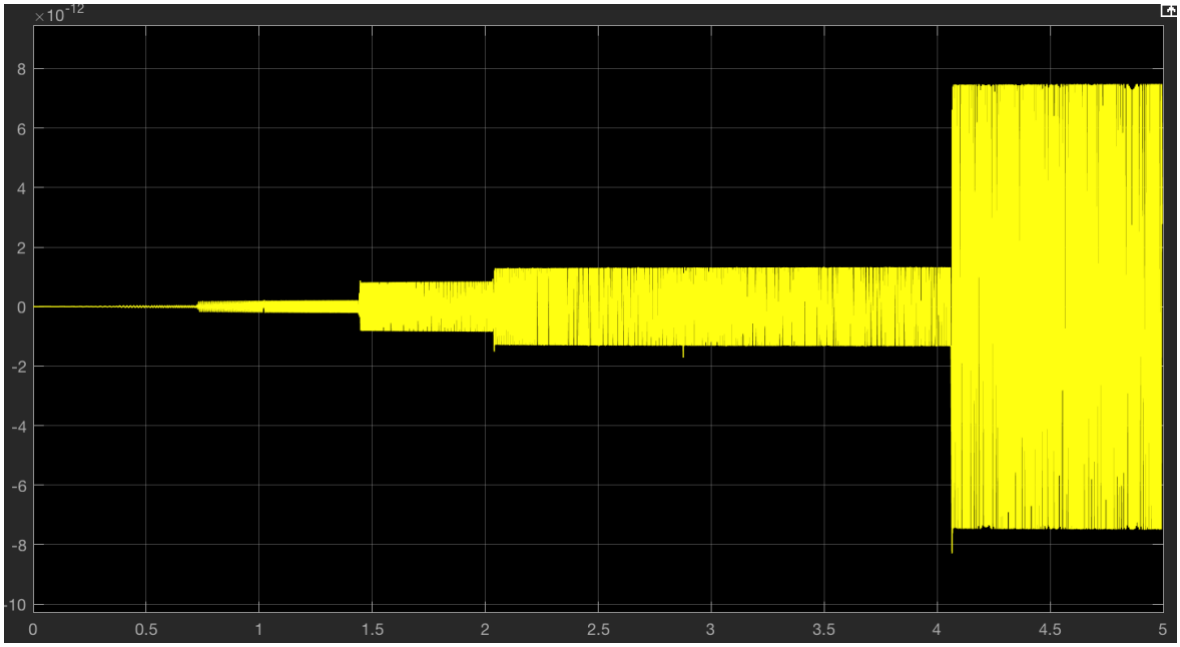


Figura 61 – Corriente en eje d en el arranque de la motocicleta

También se puede comprobar que el par dado es el máximo (125) en la aceleración:

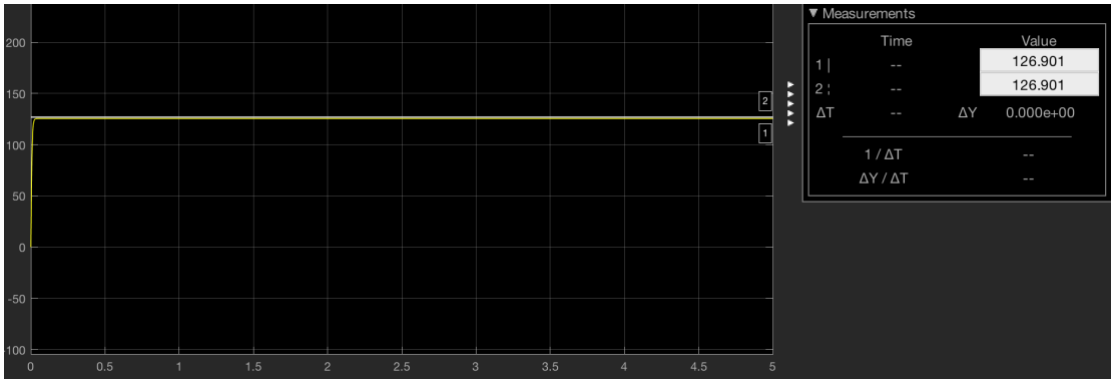


Figura 62 – Par dado en el arranque de la motocicleta

El único problema es que con estos parámetros se alcanza una tensión trifásica demasiado alta:

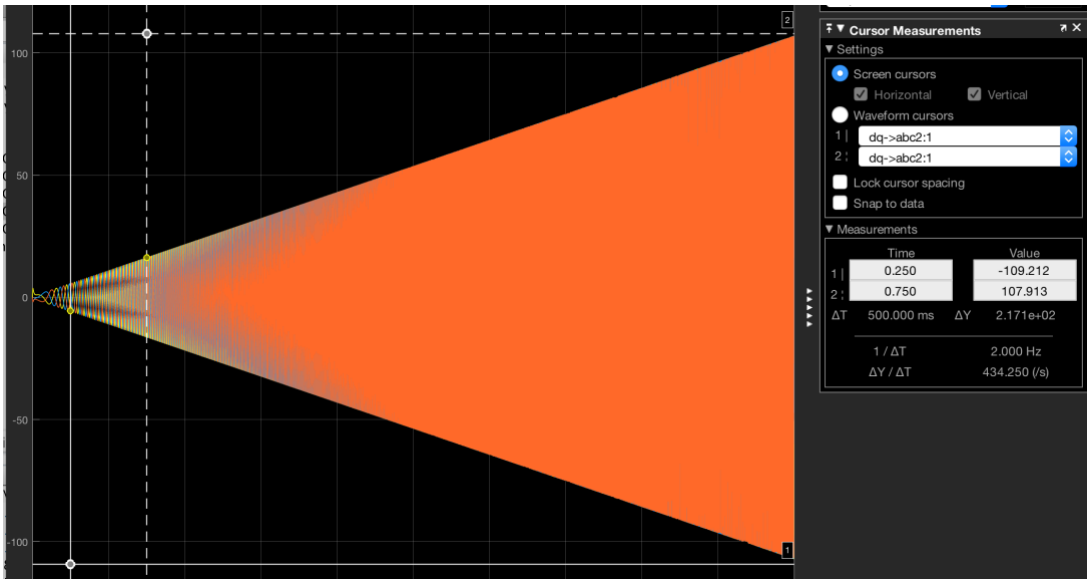


Figura 63 – Tensiones trifásicas en el arranque de la motocicleta

Se puede ver que se alcanzan valores de 110V en alterna, cuando el máximo al que se puede llegar en teoría era de 47V. Además, la tensión tiene que ver con las vueltas a las que gira el motor y no con la aceleración. Es decir, si se quiere que el motor gire tan deprisa, se tiene que aumentar la tensión de DC (con los parámetros de la Tabla 12).

También existe la posibilidad de debilitar el flujo, aunque no se ha contemplado en este trabajo, o simplemente aumentar la tensión de la batería

5.3 CONTROLES PI

Simulink dispone de una herramienta para diseñar controles PI de forma sencilla.

Para los controles de corriente, se eligen las tensiones u_d y u_q como entrada y las corrientes i_d e i_q como salidas. Se supone que la planta es un sistema RL.

Se calcula la función de transferencia entre la entrada y la salida y se diseña el PI a partir de esa planta.

Se hace lo mismo para el control de velocidad, salvo que esta vez la entrada es la corriente i_q y la salida la velocidad angular del motor.

Las figuras a continuación muestran la respuesta[3] al escalón de dichos controles:

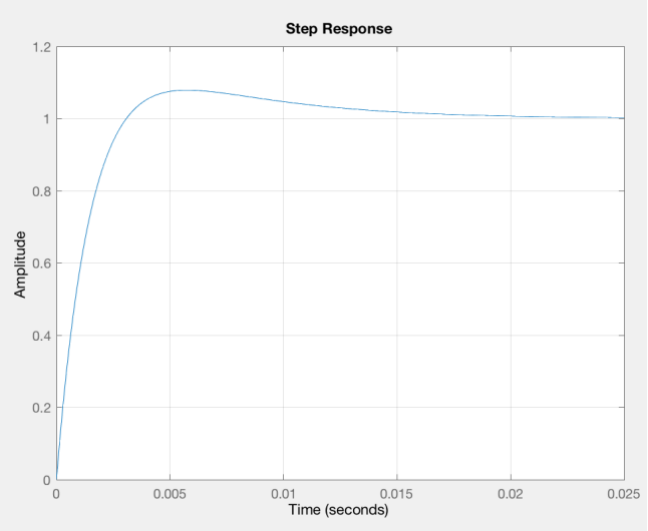


Figura 64 – Respuesta al escalón del control PI de corriente de eje q

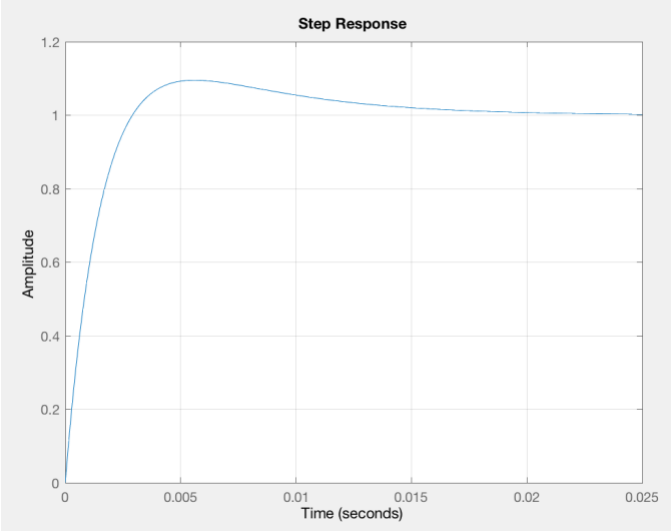


Figura 65 – Respuesta al escalón del control PI de corriente de eje d

[3] Esta respuesta, así como los parámetros proporcional e integral de los controles PI utilizados, cambian en función de los parámetros presentados en la Tabla 12, por lo que no se mostrarán los valores en sí sino que se mostrarán las figuras de la respuesta al escalón.

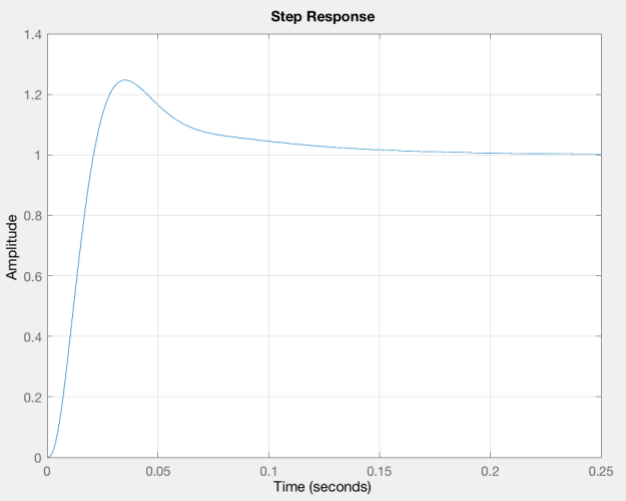


Figura 66 – Respuesta al escalón del control de PI de velocidad

Se puede observar que la respuesta al impulso de los controles de corriente es mucho más rápida que la del control de velocidad. Esto es porque desde un punto de vista mecánico, la velocidad tarda más en actualizarse. Además, como se han considerado los lazos de dentro cerrados (los dos controles de corriente), este tiene que ser más lento que aquellos para asegurar que no interfieran entre sí.

5.4 CONCLUSIÓN

En este capítulo se ha explicado cómo realizar la simulación del control de un motor de imanes permanentes.

A pesar de los problemas que se han tenido con los parámetros del motor de la competición, se han realizado unas pruebas que permiten hacerse una idea del rendimiento de la motocicleta.

Se puede ver que la motocicleta acelera en un tiempo muy razonable y que las corrientes y el par máximo al que se llega son los calculados y los estimados. Sin embargo, las tensiones trifásicas se disparan y duplican la estimada. Como se ha comentado anteriormente, este problema se podría solventar rediseñando la reductora o intentando debilitar el flujo en el eje d, o simplemente aumentando la tensión de la batería.

Capítulo 6. SOFTWARE

En este capítulo se explica el software que se ha desarrollado para el control de un motor PMSM. Los distintos parámetros de control (frecuencia de PWM, parámetros de controles PI, parámetros de control de velocidad) se han intentado dejar lo más genérico posible con el fin de simplificar la explicación.

6.1 PRINCIPIO DE FUNCIONAMIENTO

En software se compone de los siguientes archivos:

- Adc.c
- Adc.h
- Config.c
- Config.h
- Functions.c
- Functions.h
- Main.c
- Pwm.c
- Pwm.h

Los archivos de configuración de registros no valen la pena ser explicados en detalle porque ya se explican en sus correspondientes anexos. De todas formas, el código entero se encuentra en el ANEXO D – Código.

Lo destacable está en el main.c y functions.c, y de todas formas, solo se comentarán las partes relevantes del código.

Lo primero es definir los distintos parámetros que se van a utilizar y que en un futuro tendrán que ser configurables a través de un software y no tocando el código.

```
/*********************************************************/
/* DEFINES */

#define LQ 0.110e-3     // q-axis inductance
#define LD 0.062e-3     // d-axis inductance
#define PM 0.03         // Permanent Magnet Flux Linkage
#define N 5             // Number of pair poles
#define PWMPRD 10000    // PWM period is 10kHz so 10000 Hz
#define CTRLPWRD 2000   // Control loop period is 2kHz so 2000 Hz
#define MAX_RPM 8000    // Maximum RPM of motor is 8000
#define MAX_CURRENT 250 // Maximum current allowed is 250A for safety
```

Se puede ver que los parámetros coinciden con los calculados en los capítulos anteriores.

Los siguientes parámetros tienen que ver con los distintos modos de operación que tiene el controlador. En un futuro, el modo de conducción se elegirá a través de un display y unos controles básicos en las manetas del piloto.

El controlador de momento incluye 3 modos:

- (1) Modo sport (por defecto), máximo par en todo momento
- (2) Modo normal, par reducido en velocidades bajas
- (3) Modo confort, par reducido en cualquier velocidad

```
- #define MAX_SPEED_DEC_THROTTLE_100 0.3      // When throttle is
  lifted, decelerate as max rate of 10km/h
- #define MAX_SPEED_DEC_THROTTLE_50_100 0.7
- #define MAX_SPEED_DEC_THROTTLE_20_50 1.5
- #define MAX_SPEED_DEC_THROTTLE_20 3
- #define MAX_SPEED_INC_MODE_2_0_20 1/1000
- #define MAX_SPEED_INC_MODE_2_20_60 1/1500
- #define MAX_SPEED_INC_MODE_2_60_100 1/2000
- #define MAX_SPEED_INC_MODE_3 1/200
```

Se puede ver que existen 3 variables para el modo 2. Esto se hace para conseguir distintas pendientes en función de la velocidad. Por ejemplo, si la velocidad actual se encuentra entre 0 y $20 km/h$, se limita la pendiente máxima de aceleración (la explicación del cálculo se explica más adelante).

La figura a continuación representa las distintas pendientes de los modos de conducción en función del tiempo (arbitrario).

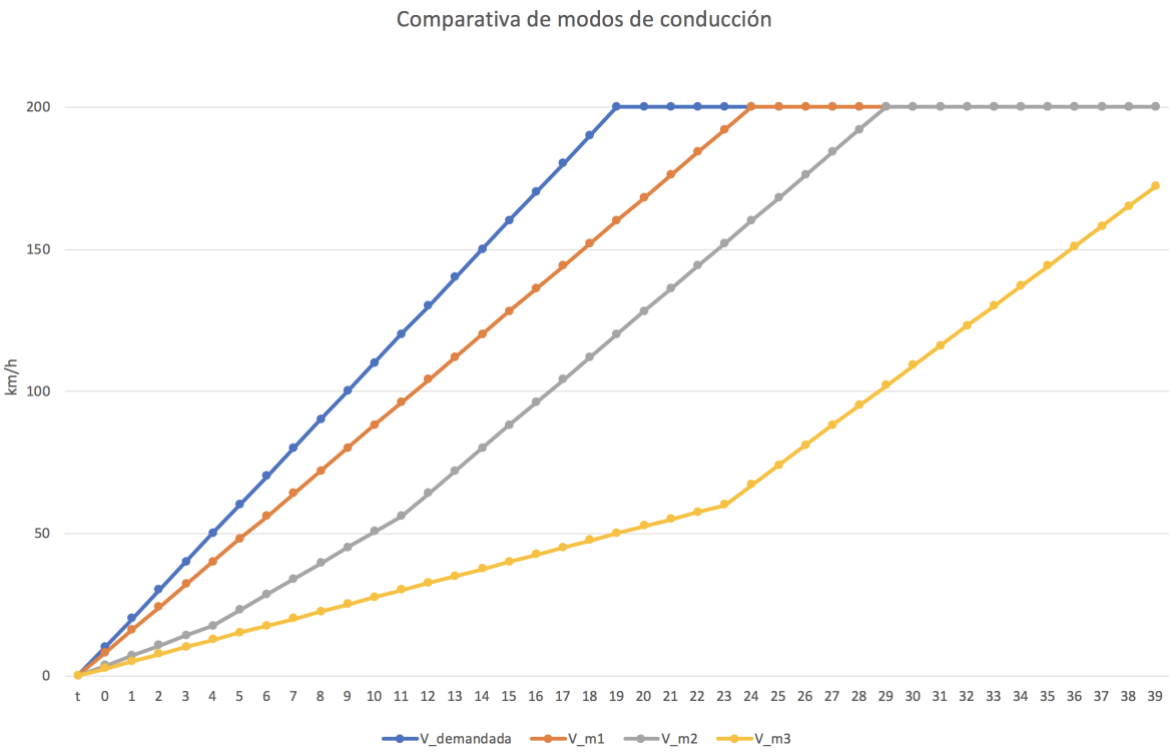


Figura 67 – Comparativa de los modos de conducción

A continuación, se inicializan las variables globales necesarias para las señales provenientes del módulo ADC, así como la variable global de control del algoritmo:

```
/*****************************************************/
/* GLOBAL VARIABLES */

int an0 = 0;     // Phase a
int an1 = 0;     // Phase b
int an2 = 0;     // Phase c
int an3 = 0;     // Battery Voltaje
int an4 = 0;     // Cosine
int an5 = 0;     // Sine
int an6 = 0;     // Front brake
int an7 = 0;     // Rear brake
int an8 = 0;     // Throttle
```

```
int trigger_control_algorithm = 0;
```

El siguiente paso es inicializar los distintos módulos e inicializar los registros PORT y TRIS para configurar los pines PWM y ADC en sus correspondientes estados de lectura o escritura:

```
int main(void) {
    /***************** Initialize functions ********************/

    iniClock();
    iniPWM();
    iniADC();

    /*
     * TRIS: registro de control asociado a cada puerto
     * del micro que permite configurar sus pines como
     * entradas (1) o salidas (0). Son registros de lectura y escritura.
     *
     * PORT: registro de datos asociado a cada puerto del
     * micro que permite leer un valor (0/1) de los pines
     * configurados como INPUTs, y escribir un valor (0/1)
     * de los pines configurados como OUTPUTs.
     */

    /*************** Initialize PORT and TRIS ******************/

    TRISB = 0x01FF; // AN0-AN8 as inputs
    TRISE = 0x0000; // PWM1-PWM3 as outputs

    PORTB = 0x01FF; // Read values
    PORTE = 0x01FF; // Write values
```

Acto seguido se inicializan todas las variables que se van a utilizar para el algoritmo de control. Esta parte no es interesante por lo que se omitirá insertar el código correspondiente y que ya se encuentra en el ANEXO D – Código.

Antes de describir el algoritmo de control, veamos cuándo se activa este.

Cuando el módulo ADC interrumpe, se llama a la siguiente función:

```
void __attribute__((interrupt, no_auto_psv)) _ADCP4Interrupt (void)
```

En ella se puede ver que se leen los distintos valores de las señales analógicas y se almacenan en las variables globales.

```
void __attribute__((interrupt, no_auto_psv)) _ADCP4Interrupt (void){
    an0 = ADCBUF0;
    an1 = ADCBUF1;
    an2 = ADCBUF2;
    an3 = ADCBUF3;
    an4 = ADCBUF4;
    an5 = ADCBUF5;
    an6 = ADCBUF6;
    an7 = ADCBUF7;
    an8 = ADCBUF8;
    trigger_control_algorithm = 1;
    IFS7bits.ADCP4IF = 0;    // Clear ADC pair 4 Interrupt flag
}
```

También se puede ver que al final se pone a uno la variable *trigger_control_algorithm* y se limpia la bandera de interrupción.

El programa espera en un while(1) hasta que dicha variable sea 1. Una vez que eso pasa, se lanza el algoritmo de control y se inicializan las variables:

```
    while(1){
        if (trigger_control_algorithm == 1){
            // Initialize variables
            int i_a = an0;          // Phase a
            int i_b = an1;          // Phase b
            int i_c = an2;          // Phase c
            int v_bat = an3;        // Battery Voltage
            int cos_resolver = an4; // Cosine
            int sin_resolver = an5; // Sine
            int f_bra = an6;        // Front brake
            int r_bra = an7;        // Rear brake
            int throttle = an8;     // Throttle
```

A continuación, se calcula la tensión de la batería, el ángulo actual y la velocidad actual:

```
            // Calculate battery voltage
            v_dc = calculate_battery_voltage(v_bat);
```

```
            // Calculate current angle and speed (rad/s)
            angle_t1 = angle_t0;
            angle_t0 = calculate_angle(sin_resolver, cos_resolver);
            speed_meas = calculate_speed(angle_t0, angle_t1, delta_time);
// measured speed in rad/s
```

La demanda de frenado se calcula como un valor entre 0 y 1 para cada freno (delantero y trasero). Esto quiere decir que la suma de ambos es superior a 1, por lo que se satura el frenado total en 1. En este caso se ha elegido que, si uno de ellos está frenando al máximo, se activa el estado de freno regenerativo máximo.

```
            // Calculate braking status/demand
            // This will yield a value between 0 and 1
            // 1 is max braking, 0 is not braking
            calculate_braking_demand(&f_bra_demand, &b_bra_demand);
            // Summed up they give a value between 0 and 2.
            // Max regen brake is achieved when either of them is 1 or if
            // combination of both is 1
            total_brake_demand = (f_bra_demand + b_bra_demand);
            if (total_brake_demand > 1){
                total_brake_demand = 1;
            }
```

Además, también se quiere que se entre en estado de frenado regenerativo si el piloto deja de acelerar, por lo que se calcula la demanda de velocidad del piloto y se transforma a km/h para el control de velocidad de los modos de conducción.

```
            // Throttle value also indicates regen braking
            // Calculate demanded speed (rad/s) with modes
            // First calculate real demanded speed
            speed_demand = throttle*(MAX_RPM/1023.0)*(2*PI/60.0);

            // Check if accelerating or braking
            speed_demand_before_kmh = speed_demand_kmh;
            speed_demand_kmh = speed_demand*6/(8*PI); // rad/s -> rpm ->
kmh
            speed_meas_kmh = speed_meas*6/(8*PI);

            speed_demand_kmh_diff = speed_demand_kmh - speed_meas_kmh;
```

Por defecto, la velocidad anterior es la actual, lo que permite comprobar en cada momento si el piloto está demandando un cambio de velocidad positivo o negativo.

También se puede ver que el paso de radianes por segundo a kilómetros por hora se ha simplificado, puesto que:

$$1\frac{rad}{s} = \frac{60}{2\pi} \cdot \frac{r}{min}$$

Y, a 8000rpm se dan 200km/h, luego:

$$1\frac{rad}{s} \cdot \frac{60}{2\pi} \cdot \frac{200}{8000} = \frac{6}{8\pi} km/h$$

A continuación, se filtra la demanda de velocidad del piloto por el control de demanda de velocidad que toma en cuenta los modos de conducción y si se está frenando o acelerando.

Cuando la diferencia entre la velocidad demandada y la velocidad medida es negativa, significa que el piloto quiere decelerar, por lo que se entraría en estado de freno regenerativo. Esto se da cuando *speed_demand_kmh* es negativa.

Es decir, si por ejemplo se va a una velocidad de 50km/h de forma constante y se suelta de forma brusca el acelerador, es como si se demanda un cambio de -50km/h en lo que tarda el puño en volver a su posición inicial.

Supongamos que el puño acelerador tarda 0.5s en volver a su posición inicial (obviamente esto varía en función de la posición actual, pero se depreciará en este caso).

El algoritmo de control salta cada 0.5ms, es decir, que en el tiempo que tarda el puño acelerador en volver a su posición inicial, se ha realizado el algoritmo de control unas 1000 veces.

En ese número de veces se tiene que poder regular la velocidad de forma que no se alcance la velocidad demandada tan rápido. Es decir, si se circula a 50km/h y el piloto suelta el puño acelerador de golpe, la velocidad demandada pasa a ser 0, y el error entre la actual y la

demandada es muy grande, por lo que las corrientes del motor serán más grandes y se sentirá un tirón fuerte por entrar en estado de freno regenerativo. Por ello, si salta el algoritmo 1000 veces en el tiempo que tarda el puño en volver a su posición inicial, se desea que **en al menos** ese número de veces no se llegue la máxima frenada regenerativa. Es decir que hay que restar un valor muy pequeño a la velocidad actual para regular de forma suave el freno regenerativo.

Se ha escogido que, en lo que tarda el puño acelerador en volver a su posición inicial, se calcule la mitad de la demanda real del piloto. Retomando el ejemplo anterior, si el piloto circula a 50km/h y suelta el puño acelerador (velocidad demandada 0), el algoritmo de control tiene que tener una velocidad demanda de 25km/h y no de 0. Por lo tanto, en 1000 iteraciones hay que disminuir la velocidad de 25km/h.

Un simple programa en Python permite ilustrar este proceso (Figura 68).

El problema que plantea un algoritmo tan lineal es que, si el piloto circula a 10km/h y realiza la misma acción, el algoritmo de control sacaría una velocidad de referencia de 5mk/h, lo cual no tiene mucho sentido ya que a velocidades tan bajas sí interesa que se active el freno regenerativo al máximo.

```
speed = 100

for i in range(1000):
    decel_rate = (speed/1000)*(0.7)
    speed = speed-decel_rate

print ("Final speed:", speed)
```

```
Final speed: 49.646359850237104
```

Figura 68 – Ejemplo del cálculo de velocidad de referencia al soltar el acelerador bruscamente

Por ello se ha dividido el cálculo en 4 secciones de velocidad:

- Mayor que 100km/h

- Entre 50 y 100km/h
- Entre 20 y 50km/h
- Menor que 20km/h

Para cada franja existe un coeficiente que se ha parametrizado con la ayuda de Python. Los siguientes ejemplos han permitido calcular dichos parámetros:

```
speed = 160

for i in range(1000):
    decel_rate = (speed/1000)*(0.3)
    speed = speed-decel_rate

print ("Final speed:", speed)
```

```
Final speed: 118.52558047092587
```

```
speed = 80

for i in range(1000):
    decel_rate = (speed/1000)*(0.7)
    speed = speed-decel_rate

print ("Final speed:", speed)
```

```
Final speed: 39.7170878801897
```

```
speed = 50

for i in range(1000):
    decel_rate = (speed/1000)*(1.5)
    speed = speed-decel_rate

print ("Final speed:", speed)
```

```
Final speed: 11.14395144217067
```

```
speed = 15

for i in range(1000):
    decel_rate = (speed/1000)*(3)
    speed = speed-decel_rate

print ("Final speed:", speed)
```

```
Final speed: 0.7434462423473335
```

Figura 69 – Ejemplos de cálculo de velocidad de referencia al soltar el puño acelerador en función de la velocidad actual

Esos ejemplos indican la nueva referencia de velocidad en el caso en el que el piloto suelte el acelerador de golpe.

Por otro lado, si el piloto está frenando, se aplica un coeficiente proporcional a la demanda de frenado para incrementar la frenada regenerativa. La figura siguiente retoma uno de los ejemplos anteriores (velocidad de 80) y le añade un factor de frenada de 0.8:

```
speed = 80
total_brake_demand = 0.8

for i in range(1000):
    decel_rate = (speed/1000)*(0.7)
    speed = speed-decel_rate
    decel_rate = (speed/1000)*total_brake_demand
    speed = speed-decel_rate

print ("Final speed:", speed)
```

```
Final speed: 17.840325089855305
```

Figura 70 – Ejemplo de cálculo de velocidad de referencia al utililzar los frenos

Se puede ver que la velocidad de referencia final es menor que en el caso anterior (17.84km/h frente a 39.7km/h).

El código por lo tanto quedaría así:

```
            // Check if pilot is decelerating to apply regen braking
            // Deceleration rate is controlled.
            if (speed_demand_kmh_diff < 0){
                // Calculate decel rate
                if (speed_meas_kmh >= 100){
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_100;
                }else if(50 =< speed_meas_kmh < 100){
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_50_100;
                }else if(20 =< speed_meas_kmh < 50){
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_20_50;
                }else{
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_20;
                }
                speed_demand_kmh = speed_demand_kmh -
decel_rate_throttle;
                // If pilot touches brakes, apply a deceleration rate
proportional to
                // braking applied
                // This is only possible when pilot is not accelerating
                if (total_brake_demand > 0.1){
```

```
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*total_brake_demand;
                    speed_demand_kmh = speed_demand_kmh -
decel_rate_throttle
                }
```

Nótese también que se ha dado un leve margen de error para la detección de frenado (*total_brake_demand > 0.1*).

Si el piloto está acelerando, se controla la aceleración en función del modo y de la velocidad actual. En el modo dos, solo se controlará el incremento de velocidad en el caso de que dicho incremento sea mayor de 30km/h con respecto a la velocidad actual. Es decir, si se circula a 10km/h y el piloto mueve bruscamente el puño acelerador, solo se le permitirá una cierta aceleración hasta 20km/h, otra hasta 60km/h y finalmente la máxima a partir de 60.

Los parámetros de incremento de velocidad se han calculado de la misma forma que antes, con un simple programa en Python que permite esta vez en cuantas iteraciones se alcanza la velocidad pedida. Retomando el ejemplo anterior, si se quiere limitar la aceleración entre 0 y 20km/h, se ha de conseguir (y de forma análoga al caso de la frenada) que la velocidad demandada final no sea la real, sino una que se controle por software.

Por ejemplo, para la franja de 0-20km/h:

```
speed = 0
MAX_SPEED_INC_MODE_2_0_20 = 1/1000
i = 0

while (speed < 20):
    accel_rate = (speed/1000)+MAX_SPEED_INC_MODE_2_0_20
    speed = speed + accel_rate
    i += 1

print (i)
print ("Final speed:", speed)
```

```
3047
Final speed: 20.020066201684596
```

Figura 71 – Ejemplo I de estimación de parámetro de incremento de velocidad para control de aceleración

Optando por un parámetro de aceleración de $1/1000$ se obtiene que alcanzarán los 20km/h en 3047 iteraciones, que traducido a segundos serían 1.5s.

Se realiza lo mismo para las otras dos franjas, solo que en este caso se espera que sea más rápido:

```
speed = 20
MAX_SPEED_INC_MODE_2_20_60 = 1/1500
i = 0

while (speed < 60):
    accel_rate = (speed/1500)+MAX_SPEED_INC_MODE_2_20_60
    speed = speed + accel_rate
    i += 1

print (i)
print ("Final speed:", speed)
```

```
1601
Final speed: 60.03821548183994
```

Figura 72 – Ejemplo II de estimación de parámetro de incremento de velocidad para control de aceleración

En este caso vemos que se tardarían aproximadamente 0.75s en obtener la velocidad de referencia demandada.

En la última franja este valor debería ser 1000 o menos para que cualquier movimiento del piloto de la velocidad demandada real.

```
speed = 60
MAX_SPEED_INC_MODE_2_60_100 = 1/2000
i = 0

while (speed < 100):
    accel_rate = (speed/2000)+MAX_SPEED_INC_MODE_2_60_100
    speed = speed + accel_rate
    i += 1

print (i)
print ("Final speed:", speed)
```

```
1009
Final speed: 100.01285451809774
```

Figura 73 – Ejemplo III de estimación de parámetro de incremento de velocidad para control de aceleración

Se hace lo mismo para el modo 3. Los valores están definidos todos al inicio del archivo main.c.

El control de aceleración queda por lo tanto:

```
            // If difference isn't negative it means pilot is
accelerating
            }else{
                // Check if not in sport mode
                if (mode != 1){
                    switch(mode){
                        case 2:
                            if (speed_demand_kmh_diff > 30){
                                if (0 < speed_meas_kmh < 20){
                                    accel_rate = (speed_meas_kmh/1000.0)
+ MAX_SPEED_INC_MODE_2_0_20;
                                }else if (20 < speed_meas_kmh < 60){
                                    accel_rate = (speed_meas_kmh/1500.0)
+ MAX_SPEED_INC_MODE_2_20_60;
                                }else if (60 < speed_meas_kmh < 100){
                                    accel_rate = (speed_meas_kmh/2000.0)
+ MAX_SPEED_INC_MODE_2_60_100;
                                }
                                speed_demand_kmh = speed_demand_kmh +
accel_rate;
                            }
                            break;
                        case 3:
                            if (speed_demand_kmh_diff > 15){
                                if (speed_meas_kmh < 60){
                                    accel_rate = (speed_meas_kmh/2000.0)
+ MAX_SPEED_INC_MODE_3;
                                }
                            }
                            break;
                    }
                }
            }
```

A continuación, se vuelve a pasar la velocidad a radianes por segundo y se ejecuta el algoritmo de control normal.

```
            // Convert back to rad/s
            // When accelerating and in mode 1 this line is useless
            speed_demand = speed_demand_kmh*(8*PI)/6;

            // PI Speed control, output is torque
            torque_demand = pi_speed(speed_meas, speed_demand);
```

```
            // Transform torque to q current
            i_q_demand = torque_demand*(2/(3*N*PM));

            // Saturate if maxes are reached
            if (i_q_demand > 1.0*MAX_CURRENT){
                i_q_demand = 1.0*MAX_CURRENT;
            }
            if (i_q_demand < -1.0*MAX_CURRENT){
                i_q_demand = -1.0*MAX_CURRENT;
            }
```

Lo único destacable de lo anterior es que, en el caso de demandar demasiada corriente, se satura a la máxima (tanto positiva como negativa). Esto implica que realmente el control de velocidad no tiene que estar del todo ajustado ya que, aunque se le pida una velocidad muy alta, el motor no podrá alcanzar esa aceleración. Sin embargo en el otro extremo si que es importante entender lo que está pasando (como se ha visto anteriormente, para poder ajustar las pendientes).

El siguiente paso consiste en preparar las medidas de las corrientes leídas, realizar la transformada de Clarke y de Park y de calcular las tensiones de referencias en ejes d y q:

```
           // Ready the current lectures
            i_a_meas = (i_a*(10/1023.0) - 5)*(0.15/5.0)*2000.0;
            i_b_meas = (i_b*(10/1023.0) - 5)*(0.15/5.0)*2000.0;
            i_c_meas = (i_c*(10/1023.0) - 5)*(0.15/5.0)*2000.0;

            // Transform measured currents to fixed alpha beta system
            // using clarke's transform
            clarke_transform(i_a, i_b, i_c, &i_alpha_meas, &i_beta_meas);

            // Transform fixed alpha beta system to rotating system in d
q componentes
            // using park's transform
            park_transform(i_alpha_meas, i_beta_meas, angle_t0,
&i_d_meas, &i_q_meas);

            // Current PI Control individual for each component
            v_d_ref = pi_current_d(i_d_meas, i_d_demand);
            v_q_ref = pi_current_q(i_q_meas, i_q_demand);
```

Finalmente, se vuelven a pasar esas tensiones al sistema de referencia trifásico para determinar el sector, se calculan los posibles tiempos existentes X, Y y Z, y se aplican los tiempos en función del sector:

```
            // Transform voltages of rotating reference back into fixed
            // alpha and beta system using park's inverse transform
            inverse_park_transform(v_d_ref, v_q_ref, angle_t0,
&v_alpha_ref, &v_beta_ref);

            // Transform alpha beta voltages back to a b c reference
system
            // to start using SVPWM using clarke's inverse transform
            inverse_park_transform(v_alpha_ref, v_beta_ref, &v_alpha_ref,
%v_b_ref %v_c_ref);

            // Locate the sector
            if (v_a_ref > 0){
                temp_a = 1;
            }else{
                temp_a = 0;
            }

            if (v_b_ref > 0){
                temp_b = 1;
            }else{
                temp_b = 0;
            }

            if (v_c_ref > 0){
                temp_c = 1;
            }else{
                temp_c = 0;
            }

            sector = temp_a + 2*temp_b + 4*temp_c;

            // Calculate X, Y and Z
            X = sqrt(3)*v_dc*v_beta_ref;
            Y = (sqrt(3)/2)*v_dc*v_beta_ref + (3/2)*v_dc*v_alpha_ref;
            Z = (sqrt(3)/2)*v_dc*v_beta_ref - (3/2)*v_dc*v_alpha_ref;

            switch(sector){
                case 1:
```

```
                t_1 = Z;
                t_2 = Y;
                break;
            case 2:
                t_1 = Y;
                t_2 = -X;
            case 3:
                t_1 = -Z;
                t_2 = X;
            case 4:
                t_1 = -X;
                t_2 = Z;
            case 5:
                t_1 = X;
                t_2 = -Y;
            case 6:
                t_1 = -Y;
                t_2 = -Z;
    }
```

El último paso es traducir esos tiempos (2) a tiempos de registros del micro (3), y asignarlos en función del sector con la función *setPDC*:

```
    t_a_on = (PWMPRD - t_1 - t_2)/2;
    t_b_on = t_a_on + t_1;
    t_c_on = t_b_on + t_2;

    // Assign duty cycles to PWM module
    switch(sector){
        case 1:
            setPDC(t_b_on, t_a_on, t_c_on);
            break;
        case 2:
            setPDC(t_a_on, t_c_on, t_b_on);
            break;
        case 3:
            setPDC(t_a_on, t_b_on, t_c_on);
            break;
        case 4:
            setPDC(t_c_on, t_b_on, t_a_on);
            break;
        case 5:
            setPDC(t_c_on, t_a_on, t_b_on);
            break;
```

```
                case 6:
                    setPDC(t_b_on, t_c_on, t_a_on);
                    break;

            }
        trigger_control_algorithm = 0;
        } // End if - end of algorithm
    } // End of while
    return 0;
} // End of main function
```

Al final del código se puede ver que se resetea el flag de inicio de control *trigger_control_algorithm* para poder repetir el algoritmo.

Lo último que se necesita comentar es que en el archivo functions.c se encuentran las variables de los distintos controles PI necesarios. Se encuentran al inicio del archivo y en este caso se han utilizado valores genéricos (los valores reales dependen de cada aplicación y en este trabajo se calculan a partir del diseño de los controles PI visto en el apartado 5.3).

```
/*
 * File:   functions.c
 * Author: Victor
 *
 * Created on 27 January 2018, 15:06
 */

/*****************************************************/
/* INCLUDES */

#include <xc.h> // Como se usa el compilador
                // automрcamente las defini
#include <dsp.h>

#include "config.h"
#include "math.h"

/*****************************************************/
/* DEFINES */

#define MAX_RPM 8000    // Max motor RPM
#define N 5             // Number of pair poles
#define PM 0.03         // Permanent magnet flux linkage
```

```
#define p_speed 10      // Proportional speed constant
#define i_speed 50      // Integral speed constant

#define p_d_current 10  // Proportional current constant for d axis
#define i_d_current 50  // Integral current constant for d axis

#define p_q_current 10  // Proportional current constant for q axis
#define i_q_current 50  // Integral current constant for q axis

/*******************************************************/
```

Capítulo 7. CONCLUSIONES Y TRABAJOS FUTUROS

El objetivo de este proyecto era el de desarrollar un controlador para la motocicleta eléctrica del ICAI Speed Club. Dicho proyecto ha conseguido cumplir muchas metas, mientras que ciertos objetivos no han sido alcanzables. Primero se comentará lo que no ha sido posible conseguir.

Se recuerda que el objetivo inicial era desarrollar el controlador **completo**. Esto ha sido imposible desde un principio por problemas logísticos, pero de todas formas hubiese constituido un trabajo demasiado difícil y de mucha envergadura, por lo que se redefinieron los objetivos.

El software ha sido comprobado a parte, y aunque las funciones cumplen con su propósito y los cálculos se realizan correctamente, no se ha verificado su funcionamiento con el motor, por lo que no se ha visto si se generan o no correctamente las señales PWM, el frenado regenerativo o el muestro de las señales.

Por otro lado, sí se han escogido los componentes del controlador de forma justificada y se considera que son aptos para los requisitos establecidos. También se ha simulado el control del motor en el momento de arranque (aceleración máxima) a pesar de haber encontrado muchas inconsistencias con los documentos proporcionados por la competición acerca del motor. Por esas razones se han escogido parámetros lo más parecidos y realistas posibles, aunque a pesar de ello se obtiene tensiones que sobrepasan el máximo calculado.

En un futuro, sería conveniente cerciorarse de los parámetros reales del motor, aunque para ello se necesitan conocimientos más profundos de máquinas eléctricas. También convendría finalizar la simulación del control introduciendo un inversor formado por transistores cuyas características sean las de los transistores elegidos. Finalmente, 'solo' quedaría montar el dispositivo físico.

También se podría adaptar el software para el uso de comandos físicos para la elección del modo de conducción, por ejemplo, o el uso de un display que permita modificar otros parámetros.

En el momento en el que se realizó este proyecto, la motocicleta tan solo contaba con sensores básicos de temperatura con el fin de proteger el motor. En un futuro se instalarán sensores *anti-wheelie* (anti-caballito) o controles de tracción, que limitan la aceleración del piloto para asegurar su protección. Todo esto se controla con el software.

Este proyecto constituye entonces el punto de comienzo de una persona que desee perseguir el objetivo de desarrollar un controlador completo para una motocicleta eléctrica cualquiera. La mayoría de componentes ya se han escogido y la explicación de cómo se implementan y cómo se eligen también. Asimismo, se ha desarrollado el algoritmo de control completo que, aunque esté sujeto a cambios, contiene el código para el funcionamiento esencial, además de alguna funcionalidad extra. Finalmente, también se ha explicado cómo simular dicho control, los distintos parámetros que influyen y las consideraciones a tomar en cuenta.

Capítulo 8. BIBLIOGRAFÍA

[1] **Grand View Research.** "Electric Motor Sales Market Analysis By Motor Type (AC Motor, DC Motor, Hermetic Motor), By Power Output (Integral HP Output, Fractional Output), By Application (Industrial, Motor vehicles, HVAC), And Segment Forecasts, 2018-2025" – Report Summary, Grand View Research (GVR), 2017.

[2] **Don Laksay.** "BLDC Motor Controllers…for maximum Performance & Efficiency", Data Device Corporation, 2015. https://www.jdtechsales.com/wp-content/uploads/2017/08/BLDC_Motor_Controllers_Perf_Efficiency_wp.pdf

[3] **International Energy Agency.** "Global EV Outlook 2017", International Energy Agency, 2017. https://www.iea.org/publications/freepublications/publication/GlobalEVOutlook2017.pdf

[4] **Daniel Torres.** "Comparing motor-control techniques", Microchip, 2009. https://www.ecnmag.com/article/2009/10/comparing-motor-control-techniques

[5] **Jorge Zambada.** "Sinusoidal Control of PMSM Motors with dsPIC30F DSC", Microchip Technology Inc., 2005. http://ww1.microchip.com/downloads/en/AppNotes/01017A.pdf

[6] **Anders Norlin Frederiksen.** "Sensorless Vector Control Techniques For Efficient Motor Control Continues", Analog Devices, 2013. http://www.analog.com/media/en/technical-documentation/tech-articles/Sensorless-Vector-Control-Techniques-for-Efficient-Motor-Control-Continues-MS-2549.pdf

[7] **Microchip Technology Inc.** dsPIC33FJ32MC204 Device Overview https://www.microchip.com/wwwproducts/en/en530334

[8] **Amr Tolba.** "Development of a simulation environment to investigate the control behaviour of a permanent magnet synchronous motor", Thesis, 2015.

[9] **D. C. Hanselman.** "Brushless Permanent Magnet Motor Design, 2ed", Magna Physics Publishing, 2006.

[10] **Sri Lanka Motorcycle.** "Motorcycle Brake System". https://www.srilankamotorcycle.com/motorcycle-Brake-System.php

[11] **Haruo Sakamoto.** "Designing and Manufacturing of Hand-made Electric Motorcycle", Department of Intelligent Mechanical Systems Engineering, Kochi University of Technology, 2004.

[12] **Zixu Jiang.** "Sliding Mode Disturbance Observer-Based Fractional Second-Order Nonsingular Terminal Sliding Mode Control for PMSM Position Regulation System", Mathematical Problems in Engineering, 2015.

[13] **Microchip Technology Inc.** dsPIC33FJ32GS606 Device Overview https://www.microchip.com/wwwproducts/en/dsPIC33FJ32GS606

[14] **S.F. Gorman, C. Chen, J. J. Cathey.** "Determination of permanent magnet synchronous motor parameters for use in brushless DC motor drive analysis", IEEE Transactions on Energy Conversion, Vol. 3, No.3, 1988.

[15] **Viktor Bobek.** "PMSM Electrical Parameters Measurement", NXP, 2013.

[16] **Kelly Inc.** "Motoenergy ME1115 Description". http://kellycontroller.com/motenergy-me1115-p-1388.html?osCsid=j50lmodbaftv6h0lf8274vvng4

[17] **Haixing Wang.** "Modeling Analysys and Parameters Calculation of Permanent Magnet Linear Synchronous Motor", School of Electrical Engineering and automation, 2013.

[18] **F. Libert, J. Soulard.** "Investigation on Pole-Slot Combinations for Permanent Magnet Machines witch concentrated Windings", Department of Electrical Machines and Power Electronics.

[19] **MathWorks.** "Three phase PMSM Drive". https://de.mathworks.com/help/physmod/sps/examples/three-phase-pmsm-drive.html

[20] **Microchip Technology Inc.** dsPIC33FJ32MC202 Device Overview. https://www.microchip.com/wwwproducts/en/dsPIC33FJ32MC202

[21] **Microchip Technology Inc.** dsPIC33FJ32GS406/606/608/610 and dsPIC33FJ64GS406/606/608/610 Datasheet. http://ww1.microchip.com/downloads/en/DeviceDoc/70000591f.pdf

[22] **Samuel J. Underwood.** "On-line parameter estimation and adaptive control of permanent magnet synchronous machines", 2006.

[23] **Que Wang.** "Investigation and implementation of MOSFET losses equations in a three-phase inverter", Chalmers University of Technology, 2015.

[24] **Fredrik Fürst.** "Design of a 48V three-phase inverter", Chalmers University of Technology, 2015.

[25] **Semikron.** Semisel – Simulation Software. http://semisel.semikron.com/DCAC_IxC_Circuitparam.asp

[26] **Semikron.** "Short form catalog 2018", 2017-2018.

[27] **Harting.** "Hall effect current sensors". https://www.mouser.es/datasheet/2/179/98_42_938_0201_harting_hall_effect_current_sensors-372002.pdf

[28] **Microchip Technology Inc.** "Section 42. Oscillator (Part IV)". http://ww1.microchip.com/downloads/en/DeviceDoc/70307C.pdf

[29] **Microchip Technology Inc.** "Section 43. High Speed PWM". http://ww1.microchip.com/downloads/en/DeviceDoc/70323D.pdf

[30] **Dave Wilson.** "Motor Control Compendium", 2011.

[31] **Semikron.** Semisel – Simulation Software. http://semisel.semikron.com/DriverSelectTool.asp

[32] **XALT Energy.** XALT 75Ah High Energy Lithium Ion Cell Datasheet. https://www.xaltenergy.com/wp-content/uploads/2017/10/XE_Data_75HE.pdf

[33] **Laura D. Marlino. "**Report on Toyota Prius Motor Therman Management", 2005.

[34] **Ned Mohan.** "Electric Drives, and Integrative Approach", University of Minn, 2000.

[35] **Mathworks.** "Permanent Magnet Synchronous Motor", Permanent magnet synchronous motor with sinusoidal flux distribution. https://de.mathworks.com/help/physmod/sps/ref/permanentmagnetsynchronousmotor.html

[36] **Ned Mohan.** "Power Electronics", University of Minnesota, 1995.

[37] **Semikron**. Semisel – Simulation Software, Device Proposal. http://semisel.semikron.com/proposal.asp

[38] **Microchip Technology Inc.** "Section 44. High-Speed 10-bit ADC". http://ww1.microchip.com/downloads/en/DeviceDoc/70321D.pdf

[39] **Semikron.** SkiM406GD066HD Datasheet. https://www.semikron.com/dl/service-support/downloads/download/semikron-datasheet-skim406gd066hd-23630030/

[40] **Semikron.** Skyper 42 R Datasheet. https://www.semikron.com/dl/service-support/downloads/download/semikron-datasheet-skyper-42-r-l5054301/

[41] **Yngve Solbakken.** "Space Vector PWM Intro", Switchcraft, 2017.

[42] **Erwan Simon.** "Implementation of a Speed Field Oriented Control of 3-phase PMSM Motor using TMS320F240", Texas Instruments, 1999.

[43] **D. Lin, P. Zhou.** "In-Depth Study of the Torque Constant for Permanent-Magnet Machines".

ANEXO A – CONFIGURACIÓN DEL OSCILADOR EN EL DSPIC33FJ32GS606

En este anexo, se explica en detalle la configuración de pines, registros y la programación del oscilador del microcontrolador que se usa para el controlador. En el apartado 4.1.1.1, se han explicado brevemente la configuración y funcionalidad del oscilador, pero la explicación completa se encuentra a continuación.

OSCILADOR

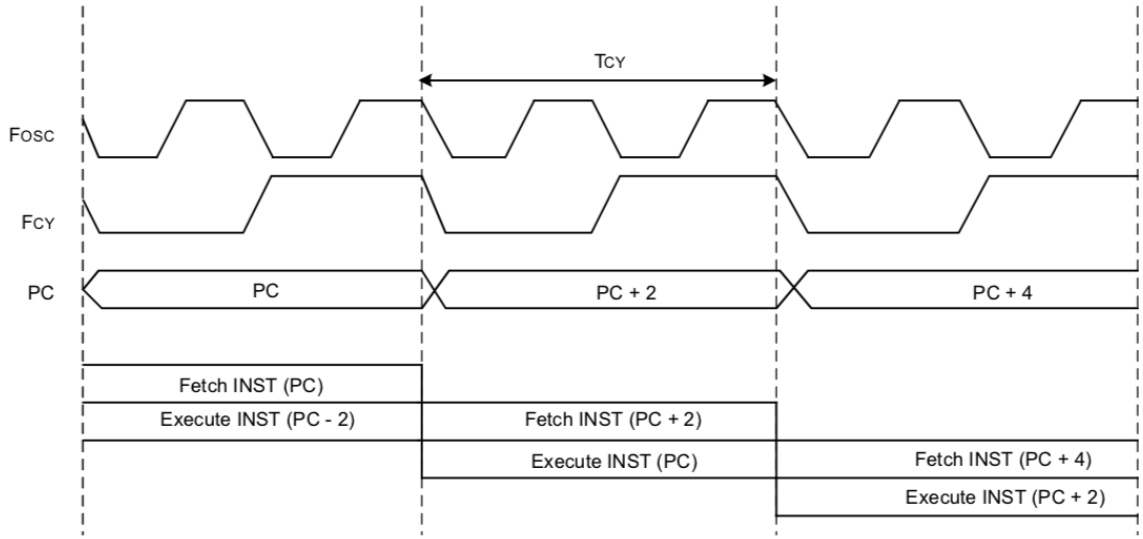
El oscilador es esencial para que un microcontrolador funcione correctamente. Para poder determinar el tiempo que tarda una instrucción en completarse, hay que elegir una referencia de tiempo para el reloj del microcontrolador. Esta referencia es un oscilador: un cristal que vibra al aplicarle una corriente.

Para utilizar cualquier función del microcontrolador se tiene que configurar su oscilador primero, ya que es la referencia para el reloj y por lo tanto está directamente relacionado con los *MIPS* (millones de instrucciones por segundo).

La Anexo A – Figura 1 – Relación entre F_{OSC}, F_{CY} y el tiempo de ejecución de instrucciones del dsPIC33FJ32GS606 [28] muestra la relación entre F_{OSC}, F_{CY} y el tiempo de ejecución de las instrucciones.

Se puede ver como $F_{CY} = \frac{Fosc}{2}$.

La frecuencia máxima que puede tomar F_{OSC} es de $80MHz$, con lo que se consiguen $40MIPS$ [28].



Anexo A – Figura 1 – Relación entre F_{OSC}, F_{CY} y el tiempo de ejecución de instrucciones del dsPIC33FJ32GS606 [28]

Para calcular F_{OSC}, lo primero es elegir la frecuencia de referencia. Esta puede venir dada por:

- Un oscilador primario – externo – (P_{OSC}), a través de los pines 39 y 40 (*OSCI1* y *OSCI2* respectivamente),
- Un oscilador secundario – externo – (S_{OSC}), a través de los pines 47 y 48 (*SOSCI* y *SOSCO* respectivamente),
- Por el cristal interno (*FRC*).

En este caso se usará el cristal interno.

Además, el microcontrolador dispone de un circuito *PLL* (*Phase Locked-Loop*), que permite obtener velocidad de operaciones mayores. Para conseguir la mayor velocidad posible ($40 MIPS$), será necesario utilizar dicho circuito.

El cálculo de F_{OSC} viene dado por:

$$F_{OSC} = F_{IN} \cdot \frac{M}{N_1 \cdot N_2} \quad (31)$$

$$F_{OSC} = F_{IN} \cdot \frac{PLLDIV + 2}{(PLLPRE + 2) \cdot (2 \cdot (PLLPOST + 1))} \quad (32)$$

F_{IN} representa la frecuencia de entrada, $PLLDIV$ el multiplicador del circuito *PLL*, $PLLPRE$ el pre-escalado del *PLL* y $PLLPOST$ el post-escalado del *PLL*.

- La frecuencia nominal del cristal interno es de $7.37MHz$,
- $PLLDIV$ es un valor entre 2 y 513 [28],
- $PLLPRE$ es un valor entre 0 y 31 [28],
- $PLLPOST$ toma el valor 0, 1 o 3 [28].

Tomando $F_{IN} = 7.37KHz$, $F_{OSC} = 80KHz$ y los valores mínimos de N_1 y N_2 (2 en ambos casos), se despeja $PLLDIV$ de (32):

$$M = \frac{F_{OSC}}{F_{IN}} \cdot (N_1 \cdot N_2) \quad (33)$$

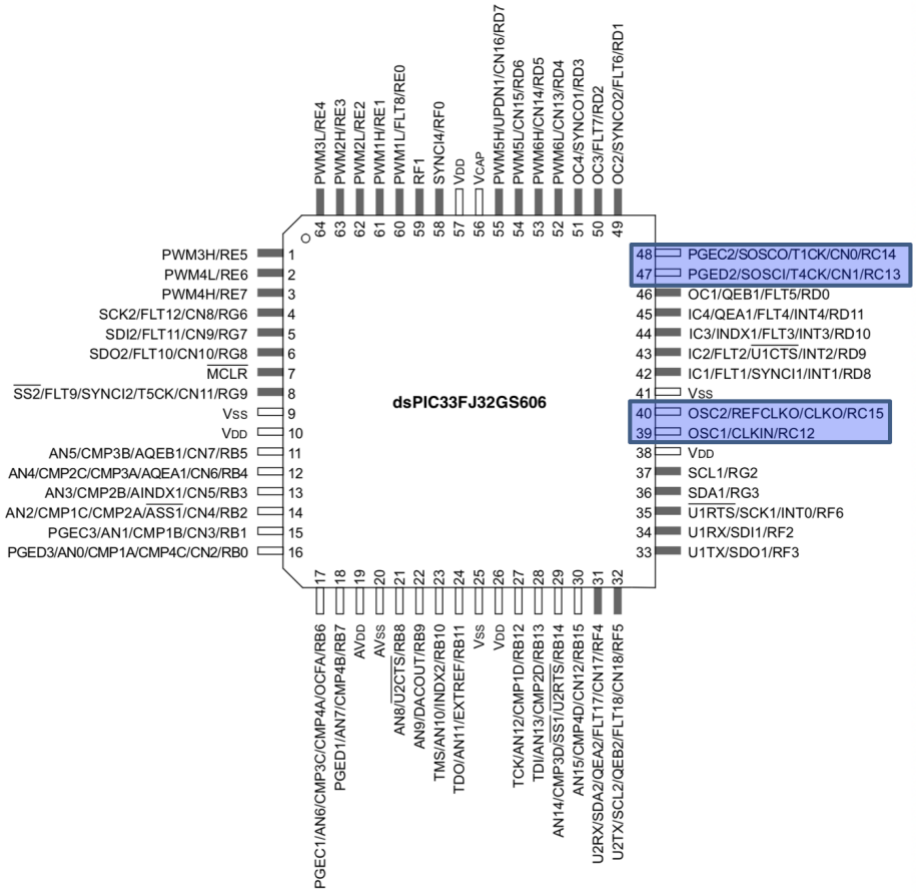
$$M = 43$$

Luego $PLLDIV = 43 - 2 = 41$.

En resumen, tan solo hay que configurar el reloj del microcontrolador para que use su cristal interno como oscilador de referencia y configurar los registros para conseguir la máxima velocidad posible.

PINES

Los únicos pines relacionados con el reloj en el microcontrolador son las parejas de pines 39-40 y 47-48, que sirven para utilizar un oscilador externo. Como no se van a utilizar, serán irrelevantes (Anexo A – Figura 2).



Anexo A – Figura 2 – Pines del oscilador en el dsPIC33FJ32GS606 [21]

REGISTROS

1. FOSCSEL

Este registro es que el que decide la referencia de tiempo para el reloj del microcontrolador. Contiene dos sub-registros, IESO y FNOSC.

- **IESO** puede tomar 0 o 1. Si se pone a 1, el microcontrolador se enciende utilizando el oscilador interno FRC, y después cambia al seleccionado por el usuario. Si se pone a 0, empieza directamente con la referencia seleccionada por el usuario.

- **FNOSC** es el sub-registro que realmente elije la referencia. Puede tomar 8 valores distintos (tamaño de 3 bits). Las opciones vienen dadas por la siguiente figura (Anexo A – Figura 3):

FNOSC<2:0>: Initial Oscillator Source Selection bits
111 = Fast RC Oscillator with Divide-by-N (FRCDIVN)
110 = Fast RC Oscillator with Divide-by-16 (FRCDIV16)
101 = Low-Power RC Oscillator (LPRC)
100 = Secondary Oscillator (SOSC)(1)
011 = Primary Oscillator with PLL (XTPLL, HSPLL, ECPLL)
010 = Primary Oscillator (XT, HS, EC)
001 = Fast RC Oscillator with PLL (FRCPLL)
000 = Fast RC Oscillator (FRC)

Anexo A – Figura 3 – Opciones para la referencia de oscilador en el dsPIC33FJ32GS606 [28]

2. FOSC

Este registro contiene distintos sub-registros de configuración de oscilador, como permitir que se reconfigure el reloj una vez iniciado el dispositivo y el número de veces que se permite ese cambio. No es relevante.

3. OSCON

Otro registro de configuración. Destacan:

- **NOSC**, un sub-registro de 3 bits que permite, si se ha habilitado en FOSC, indicar cuál es la nueva referencia del reloj.
- **CLKLOCK**, un bit que permite el cambio de configuración del reloj del sistema (este registro solo está habilitado si el bit de control **FOSC<FCKSM>** vale 1, bloqueando la reconfiguración del reloj).

4. CLKDIV

Registro divisor de reloj. Contiene, los registros esenciales de configuración del circuito PLL (entre otros):

- **FRCDIV**, un sub-registro de 3 bits que permite dividir la frecuencia del oscilador entre un máximo de 256.
- **PLLPOST**, un sub-registro de 2 bits que permite dividir la salida del circuito PLL entre un máximo de 8.
- **PLLPRE**, un sub-registro de 5 bits que permite dividir la entrada del circuito PLL entre un máximo de 33.

5. PLLFBD

Otro registro de configuración del circuito de PLL que contiene el último parámetro importante para su funcionamiento:

- **PLLDIV**, un sub-registro de 9 bits que permite multiplicar la frecuencia entrante al PLL por diferentes valores (desde 2 hasta 513, con un paso de 1).

6. OSCTUN

Registro que contiene tan solo un sub-registro:

- **TUN**, un sub-registro de 6 bits que permite alterar la frecuencia nominal del FRC ($7.37MHz$) hasta un mínimo de $6.49MHz$ y un máximo de $8.23MHz$.

7. ACLKCON

Registro de control para el reloj auxiliar. Este registro es relevante para la configuración del módulo PWM (ANEXO B – Configuración del módulo PWM en el dsPIC33FJ32GS606). Contiene los sub-registros siguientes:

- **ENAPLL**, un bit que indica si el circuito PLL auxiliar está activo o no.
- **APLLCLK**, un bit de lectura que indica si el circuito PLL auxiliar está en modo bloqueo o no.
- **SELACLK**, un bit que indica la referencia para el divisor del reloj auxiliar.
- **APSTSCLR**, sub-registro de 3 bits que divida la salida del reloj auxiliar entre un máximo de 256.

- **ASRCSEL**, un bit que indica si la referencia para el reloj auxiliar es el oscilador primario (externo) o no.
- **FRCSEL**, un bit que indica si la referencia para el reloj auxiliar es el FRC o lo determina el sub-registro ASRCSEL.

8. REFOCON

Este registro permite sacar el osciloscopio por un pin. No se utiliza.

PROGRAMACIÓN DEL DSPIC33FJ32GS606

La programación del oscilador se encuentra en el ANEXO D – Código.s

ANEXO B – CONFIGURACIÓN DEL MÓDULO PWM EN EL DSPIC33FJ32GS606

En este anexo, se explica en detalle la configuración de pines, registros y la programación del módulo PWM del microcontrolador que se usa para el controlador. En el apartado 4.1.1.2, se han explicado brevemente la configuración y funcionalidad del módulo PWM, pero la explicación completa se encuentra a continuación.

MÓDULO PWM

FUNCIONAMIENTO

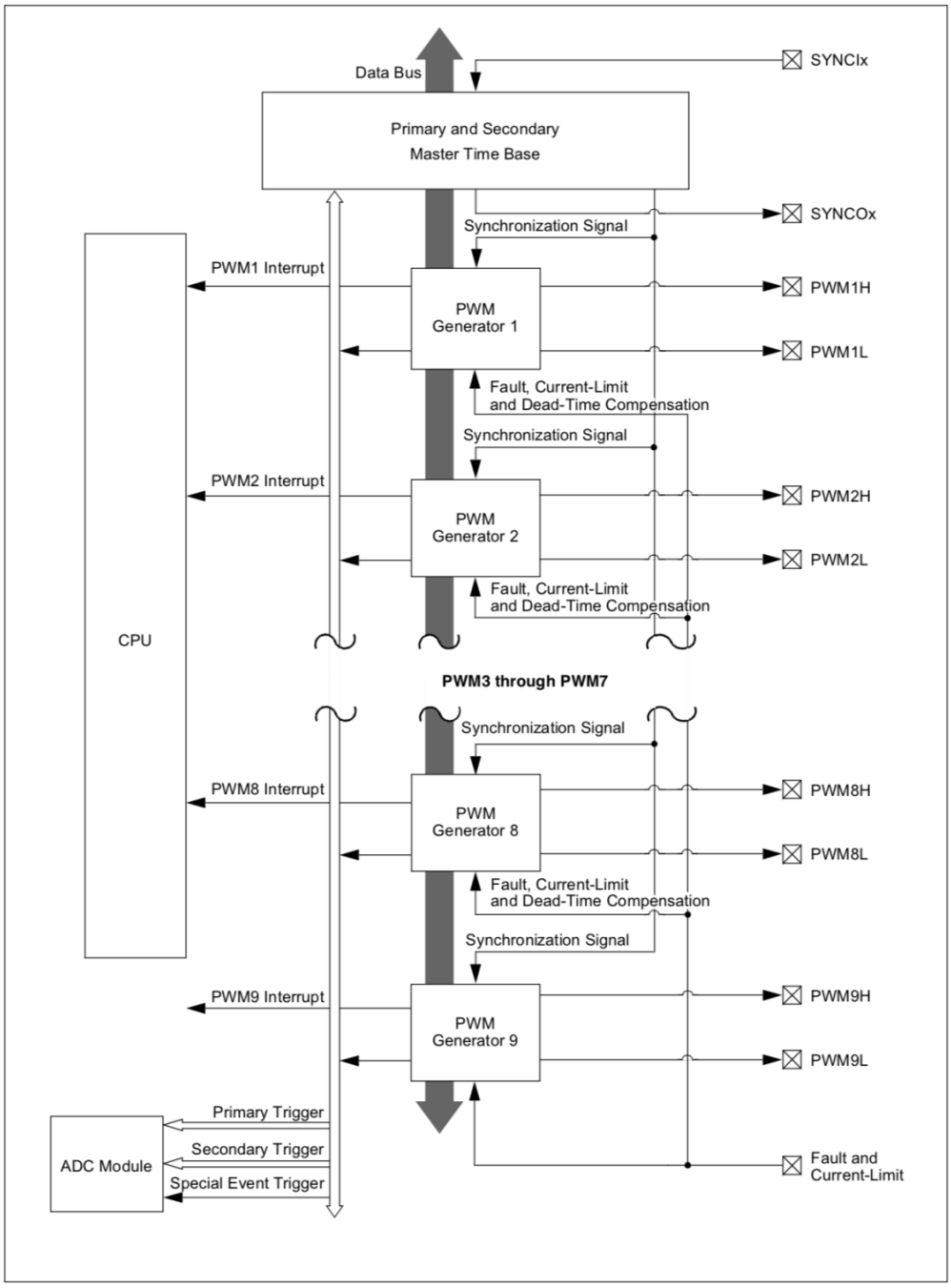
La Anexo B – Figura 1 presenta el esquema del módulo PWM del microcontrolador dsPIC33FJ32GS606. Cada generador PWM tiene dos salidas: PWMxH y PWMxL. Existe un registro de tiempo maestro que permite sincronizar varias salidas PWM (*Master Time Base* o MTB), aunque cada generador PWM puede funcionar de forma independiente (*Independent Time Base* o ITB). El módulo PWM también dispone de pines de entrada que monitorizan el sistema y lo protegen de sobre-corrientes o avisan de posibles fallos.

Además, cada generador PWM puede generar una señal de disparo para el conversor AD para que este empiece a muestrear en ese instante dado (también se puede generar esa señal de disparo a partir del registro de tiempo maestro, a través de un evento especial, ver Anexo B – Figura 1).

Para que el reloj del PWM funcione de forma independiente al reloj del sistema, y sobre todo, para que funcione a su resolución máxima, se tiene que utilizar el generador de reloj auxiliar. Para generarlo, se puede usar:

- El oscilador primario (POSCCLK),
- La salida PLL (F_{VCO}),
- El reloj interno FRC (FRCCLK).

FIGURE 16-1: HIGH-SPEED PWMx MODULE ARCHITECTURAL DIAGRAM

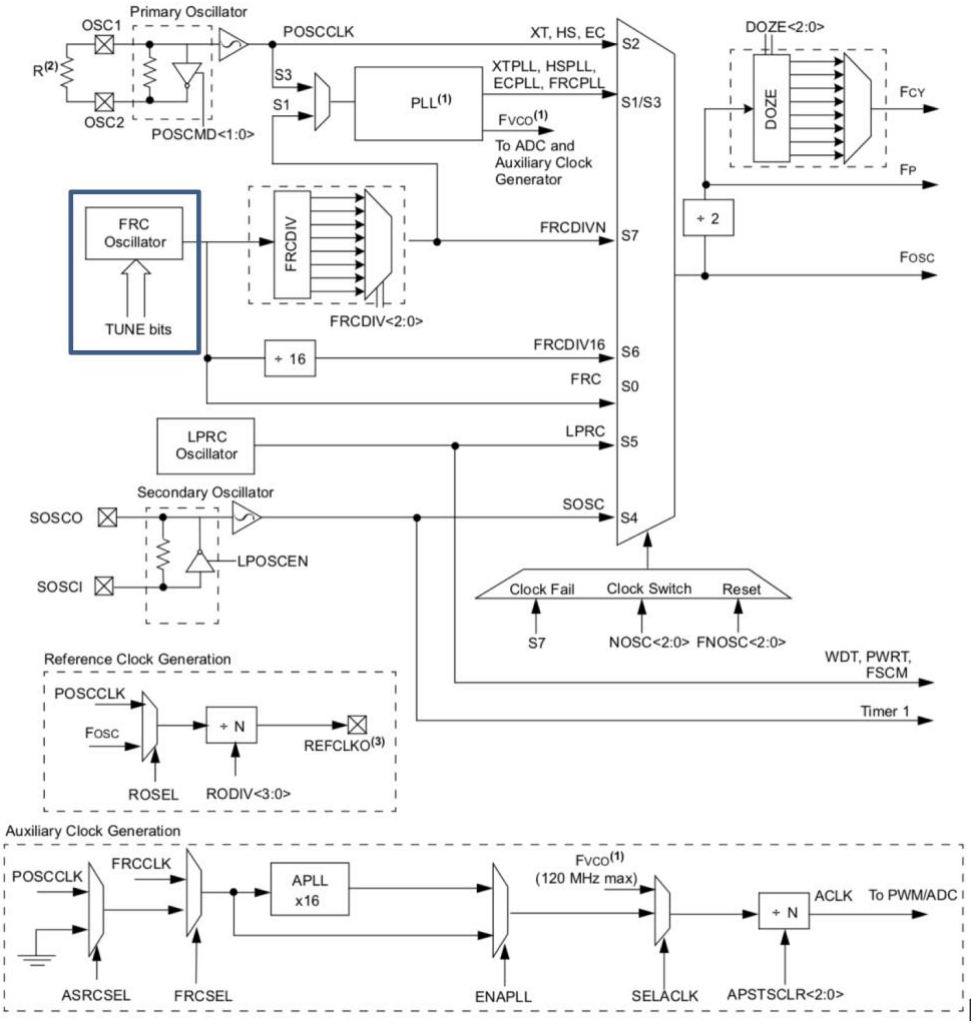


Anexo B – Figura 1 – Esquema del módulo PWM del dsPIC33FJ32GS606 [21]

El registro de control de reloj auxiliar (ACLKCON) permite elegir la referencia (una de las tres mencionadas antes) y habilitar el circuito PLL auxiliar para obtener la frecuencia deseada. La frecuencia nominal de entrada de PWM tendría que ser de $120MHz$ [29].

Se usará el reloj interno (FRC) como referencia del reloj de PWM.

La siguiente figura presenta el sistema del oscilador, ilustrando las distintas referencias para el reloj PWM. También se puede ver (cuadrado azul) como, al utilizar el FRC como referencia, no se utiliza la frecuencia nominal de $7.37MHz$, sino que se utiliza la frecuencia que se haya modificado con el registro TUN.



Anexo B – Figura 2 – Sistema del oscilador del dsPIC33FJ32GS606 [29]

La frecuencia del reloj auxiliar se calcula de la siguiente forma:

$$ACLK = \frac{REFCLK \cdot M_1}{N} \quad (34)$$

Donde:

- $REFCLK$ es la frecuencia del reloj FRC interno (si se elije como tal).
- M_1 es el multiplicador del PLL auxiliar (si se activa vale 16).
- N es la ratio de post-escalado elegido en el registro de post-escalado auxiliar.

La frecuencia de referencia $REFCLK$vale $7.37MHz$. Para obtener la máxima resolución del PWM – $1.04ns$ – la frecuencia ha de ser de $120Mhz$. Para ello hay que utilizar el multiplicador del PLL auxiliar y dividir por el mínimo valor posible.

Por lo tanto:

$$ACLK = \frac{7.37 \cdot 16}{1}$$

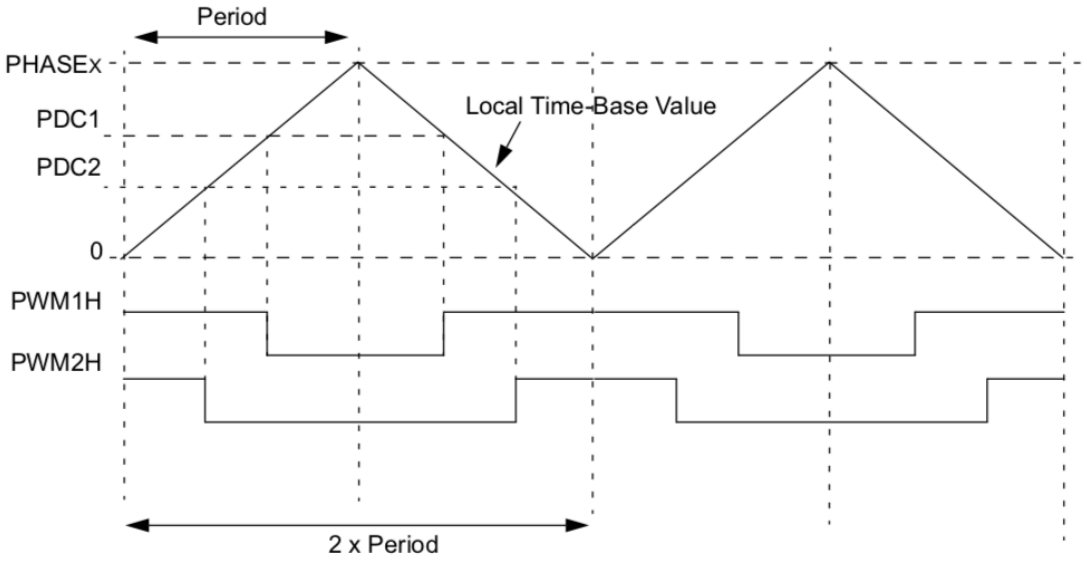
$$ACLK = 117.92MHz \quad (35)$$

A continuación, hay que configurar el modo de operación del PWM. La forma en la que se alinean las señales PWM es muy importante, y típicamente se alinean de forma centrada (en inglés *center-aligned mode*). En este caso, todas las señales PWM se alinean en su centro a partir de un período, como se muestra en Anexo B – Figura 3. A partir de este instante y con el fin de simplificar la lectura, se usará 'CAM' para referirse a dicho modo.

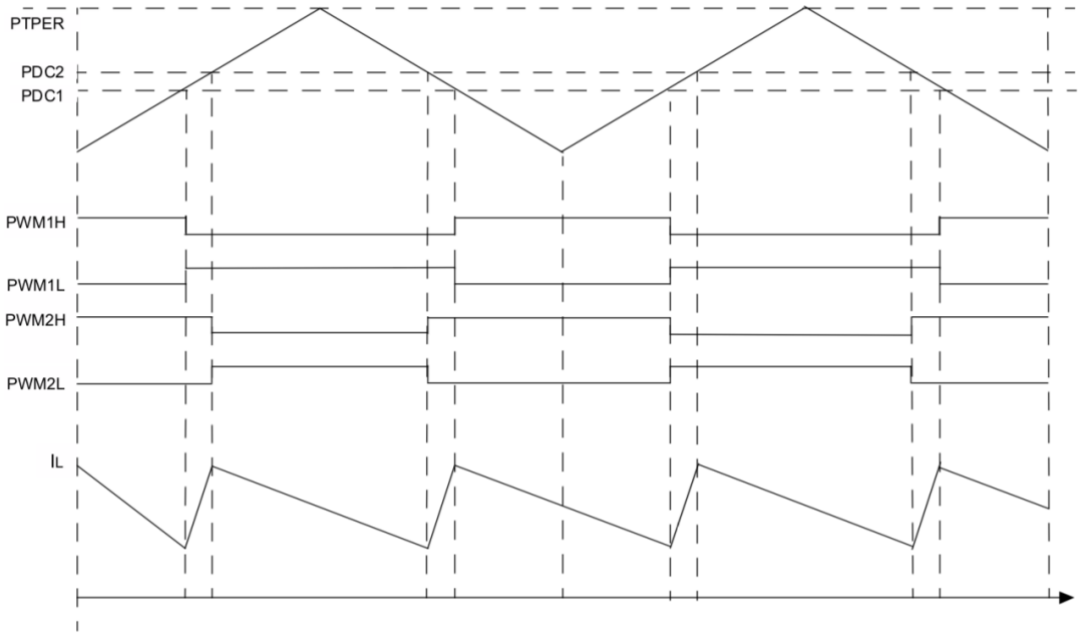
Con esta configuración (CAM), la frecuencia del rizado de la corriente y la frecuencia del ruido son el doble de la frecuencia de muestreo (Anexo B – Figura 4). Esto permite obtener una distribución más uniforme y permite que los harmónicos de la portadora, al ser de mayor frecuencia, sean más fácilmente filtrados por la inductancia del motor.

El único problema es que, al activar el CAM, los generadores PWM tienen que trabajar en modo independiente, y por lo tanto ya no pueden utilizar el tiempo maestro como su

referencia de tiempo, sino que hay que especificar en **cada generador PWM** el valor del período.



Anexo B – Figura 3 – Funcionamiento del modo 'centrado' del generador PWM [29]



Anexo B – Figura 4 – Rizado de la corriente en una configuración PWM 'centrada' [29]

Como hay que utilizar los generadores PWM en modo independiente, hay que utilizar el registro PHASEx (donde la 'x' representa el número del módulo PWM) para establecer el período de dicho módulo PWM.

El cálculo de PTPER, STPER, PHASEx y SPHASEx viene dado por:

$$\frac{ACLK \cdot 8 \cdot período\_deseado}{Pre-escalado\ del\ PWM < PCLKDIV >} - 8 \qquad (36)$$

Al trabajar en CAM, el período calculado es realmente el semiperíodo de la señal que se desea. Por lo tanto, para conseguir un período de $10kHz$, hay que hacer los cálculos con $20kHz$.

Reemplazando en (36), tomando $ACLK$ de (35), un período deseado de $\frac{1}{20 \cdot 10^3} s$ y suponiendo un pre-escalado inicial de 1:

$$\frac{117.92 \cdot 10^6 \cdot 8 \cdot \frac{1}{20 \cdot 10^3}}{1} - 8 = 47168 \qquad (37)$$

Puesto que se trabaja con registros de 16 bits, el máximo valor que pueden almacenar es de $2^{16} = 65536$, con lo cual no hay que utilizar ningún pre-escalado ya que cabe sin problemas.

Sin embargo, en CAM, para conseguir un ancho de pulso igual que el período de la señal PWM, el registro que almacena el factor de servicio (PDCx) tiene que ser igual a dos veces el valor de PHASEx (Anexo C – Figura 3).

$2 \cdot 47168 > 65536$ por lo que hay que utilizar un pre-escalado de 2.

Por lo que finalmente quedaría:

$$\frac{117.92 \cdot 10^6 \cdot 8 \cdot \frac{1}{20 \cdot 10^3}}{2} - 8 = 23584 \qquad (38)$$

Otro aspecto a tener en cuenta es el registro TRIGx. Este registro, aunque también es de 16 bits, tiene sus últimos 3 bits a 0 por defecto. Esto implica que, si el número anterior acaba en 1, 2 o 4, hay que sesgarlo. En este caso acabo en 4 por lo que se usará 23580, que multiplicado por 2 da 47160.

El sesgo es tan insignificante que la frecuencia no cambia.

Hasta este punto, lo único que queda por especificar es el período en el que tiene que estar *on* cada generador PWM (llamado factor de servicio o *duty cycle* en inglés), el cual va almacenado en los registros PDCx y que se calcula con el software de control.

El último paso consiste en gestionar el disparo de los conversores analógico-digitales. Como los generadores PWM tienen que trabajar en modo independiente (cada uno puede tener su propio período), cada uno de ellos puede generar una señal de aviso al módulo ADC. Por otro lado, se ha comentado anteriormente que los 3 generadores trabajan con el mismo período, por lo que, con programar una señal de aviso en uno de los 3 generadores valdría (Anexo B – Figura 1). En modo independiente, PHASEx almacena el valor del período y TRIGx almacena el valor que se compara con PHASEx. Si los valores coinciden, se dispara una señal de aviso. El funcionamiento es **parecido**, aunque no idéntico, si se usara el evento especial: PTPER almacenaría el valor del período y SEVTCMP el valor de comparación.

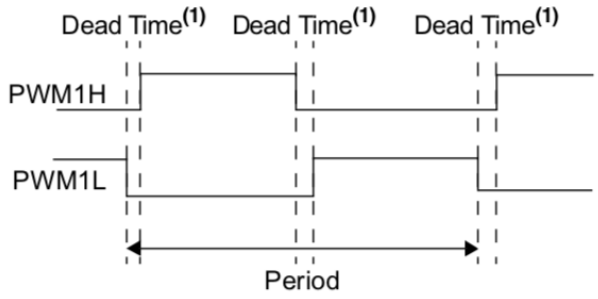
Cuando coinciden PHASEx y TRIGx se genera una señal. Si el registro TRGCONx<TRGDIV> se ha puesto a 0, se genera, a cada **coincidencia**, un *trigger*, lo que pone PWCONx<TRGSTAT> a 1. Si se pone otra cosa en TRGCONx<TRGDIV>, por ejemplo, un 1, PWCONx<TRGSTAT> se pondrá a 1 cada **dos coincidencias**.

Cuando coinciden PTPER y SEVTCMP se genera una señal. Si el registro PTCON<SEVTPS> se ha puesto a 0, cada **período** se comprueba si coinciden PTPER y SEVTCMP, y cuando coincidan se pone PTCON<SESTAT> A 1. Si se pone otra cosa en PTCON<SEVTPS>, por ejemplo, un 1, cada **dos períodos** se comprueban si coinciden PTPER y SEVTCMP.

Como se requiere una frecuencia de control de $2kHz$ y el PWM tiene una frecuencia de $10kHz$, **en teoría** habría que configurar TRGCON1<TRGDIV> para que generara el *trigger* cada 5 comparaciones. Sin embargo, al funcionar en CAM, PHASEx cuenta primero hacia arriba y después hacia abajo, por lo que en un ciclo completo TRIGx es igual a PHASEx **dos veces**. Por lo tanto, para conseguir que genere un *trigger* cada $2kHz$, hay que hacer que TRGCON1<TRGDIV> genere un *trigger* cada **10 comparaciones**.

Es mejor muestrear las señales al final de la cuenta del PWM, por lo que el valor de TRIG1 será igual 0. Esto quiere decir que, al cabo de 10 ceros en la cuenta, se disparará la señal al módulo ADC.

Lo último por mencionar es relativo a los tiempos muertos. Se sabe de 4.3.5 que dos transistores de una rama del inversor no pueden estar activos al mismo tiempo. Esto implica que hay que introducir un pequeño período de tiempo muerto entre las señales *High* y *Low* para evitar este estado, como se puede ver en Anexo B – Figura 5.



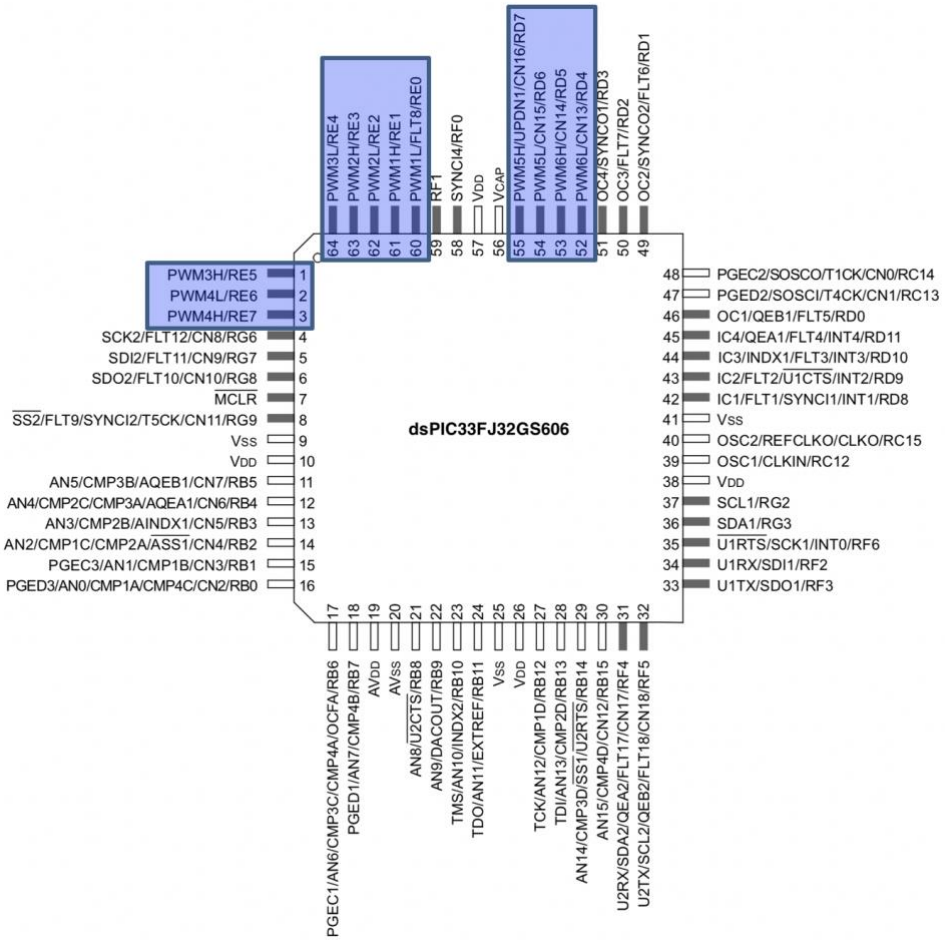
Anexo B – Figura 5 – Ejemplo de inserción de tiempo muerto para una señal PWM [29]

El transistor elegido tiene un tiempo de encendido $t_{on} = 180ns$ y $t_{off} = 950ns$ [39].

Afortunadamente, el driver elegido (4.1.3) inserta un tiempo muerto de $2\mu s$ [40], lo cual es suficiente para evitar que mientras un transistor se apaga el otro se encienda.

PINES

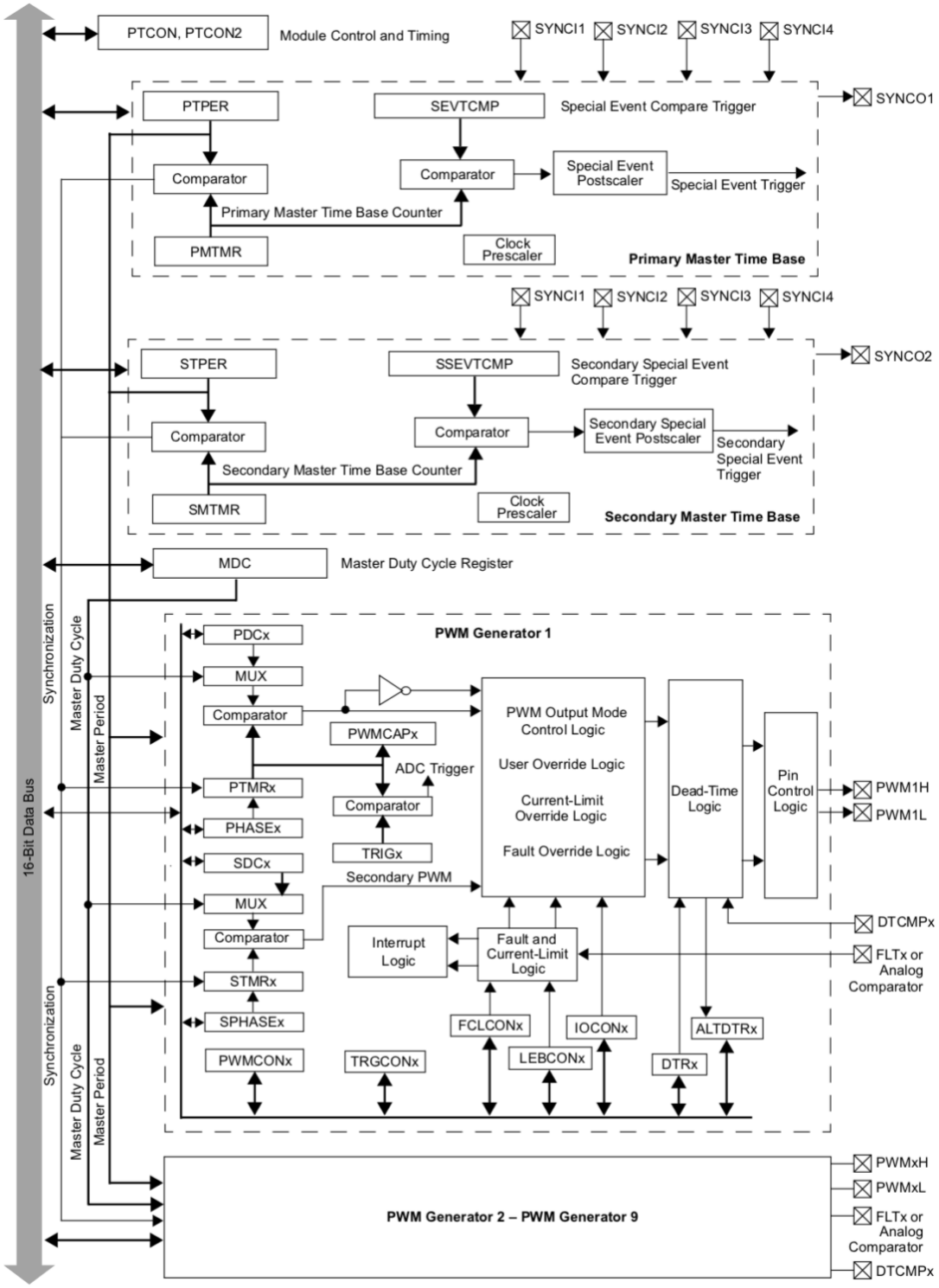
El módulo PWM utiliza 12 pines del microcontrolador, ya que tiene 6 generadores PWM independientes y cada cual tiene un pin para el *high* y otro para el *low*. Los pines PWM que tiene el dsPIC33FJ32GS606 se pueden ver en Anexo B – Figura 6.



Anexo B – Figura 6 – Pines PWM del dsPIC33FJ32GS606 [21]

REGISTROS

El módulo PWM tiene 10 registros comunes, y 16 registros para cada generador de PWM. La vista global de los distintos registros y del funcionamiento de estos se puede ver en Anexo B – Figura 7.



Anexo B – Figura 7 – Vista global de los registros y del funcionamiento del módulo PWM del dsPIC33FJ32GS606 [29]

1. Registros comunes

1.1. PTCON

PTCON es el registro de control de tiempo principal del módulo PWM. Los sub-registros relevantes son:

- **PTEN**, un bit que habilita (1) o deshabilita (0) el módulo PWM.
- **PTSIDL**, un bit que habilita (0) o deshabilita (1) el funcionamiento del módulo PWM en modo suspensión.
- **SESTAT**, un bit de lectura que indica si la señal del evento especial ha ocurrido (1) o sigue pendiente (0), es decir, si SEVTCMP = PTPER.
- **SEIEN**, un bit que habilita (1) o deshabilita (0) la interrupción del evento especial.
- **EIPU**, un bit que habilita (1) o deshabilita (0) la actualización inmediata del período, sin esperar a la sincronización de la señal.
- **SEVTPS**, un sub-registro de 4 bits que indica el post-escalado para el envío de la señal del evento especial. Por ejemplo (en MTB), si SEVTPS = 0b0001, se comprueba si PTPER = SEVTCMP cada dos períodos del PWM, y si coinciden se pone SESTAT a 1.

1.2. PTCON2

Registro de división de reloj. Contiene solo un sub-registro de 3 bits, **PCLKDIV**, que indica el pre-escalado de la señal de entrada del PWM. Se puede dividir dicha señal por un mínimo de 1 (000) o un máximo de 64 (110).

1.3. PTPER

Registro maestro primario de tiempo. Almacena el número que resulta del cálculo de (36). En el caso de trabajar en ITB, no se utiliza.

1.4. SEVTCMP

Registro que almacena el valor con el que se compara a PTPER. Si coinciden, PTCON<SESTAT> se pone a 1.

1.5. MDC

Registro maestro primario para el factor de servicio. Puesto que se quieren generar 3 señales PWM distintas, no se utiliza.

1.6. STCON

Igual que PTCON, pero para el canal secundario. **No se utiliza en este caso.**

1.7. STCON2

Igual que STCON, pero para el canal secundario. **No se utiliza en este caso.**

1.8. STPER

Igual que PTER, pero para el canal secundario. **No se utiliza en este caso.**

1.9. SSEVTCMP

Igual que SEVTCMP, pero para el canal secundario. **No se utiliza en este caso.**

1.10. CHOP

No se utiliza en este caso.

2. Registros de cada generador PWM

Los siguientes registros son particulares para módulo generador de señal PWM, por lo que los nombres acaban en 'x', indicando que los registros del módulo PWM1 serían *PWMCON1, IOCON1,* etc., los del módulo PWM2 serían *PWMCON2*, *IOCON2,* y así seguidamente.

2.1. PWMCONx

Registro de control del módulo PWMx. Contiene muchos sub-registros, de los cuales destacan:

- **TRGSTAT**, un bit de lectura que indica si la señal de '*trigger*' ha ocurrido (1) o sigue pendiente (0), es decir, si TRIGx = PHASEx. Dependiendo de TRGCONx<TRGDIV> se puede dar más o menos a menudo.
- **TRGIEN**, un bit que habilita (1) o deshabilita (0) la interrupción de la señal de '*trigger*' del generador PWM en cuestión.
- **ITB**, un bit que elije la referencia de tiempo para el módulo PWM. En 1, el generador PWM utiliza el tiempo que hay en el registro PHASEx, es decir, que trabaja en modo independiente. En 0, el generador PWM utiliza el tiempo que hay en PTPER.
- **MDCS**, un bit que indica la referencia del factor de servicio (*duty cycle*). En 1, el generador PWM utiliza el factor de servicio que hay en el registro maestro MDC. En 0, el generador PWM utiliza el factor de servicio que hay en PDCx y SDCx.
- **DTC**, sub-registro de 2 bits que habilita o deshabilita la función del tiempo muerto más algún otro parámetro.
- **MTBS**, un bit que elije la referencia de tiempo para la sincronización del módulo PWM en el caso de usarse en MTB. En 1, el módulo PWM utiliza el tiempo maestro secundario (STPER) para la sincronización y en 0 utiliza el tiempo maestro primario (PTPER).
- **CAM**, un bit que indica si el generador PWM funciona en modo *center-aligned* (1) o modo *edge-aligned* (0). Para que funcione en modo *center-aligned*, el generador PWM tiene que estar en modo independiente (ITB = 1).
- **IUE**, un bit que habilita o deshabilita la actualización del factor de servicio de forma inmediata.

2.2. IOCONx

Registro de configuración I/O. Destacan:

- **PENH**, un bit que indica si el control del pin lo tiene el módulo PWM (1) o el módulo GPIO (0).
- **PENL**, un bit que indica si el control del pin lo tiene el módulo PWM (1) o el módulo GPIO (0).
- **PMOD**, sub-registro de 2 bits que indica en qué modo funciona cada par de pines (*high* y *low*). La siguiente figura indica las distintas posibilidades (figura x):

PMOD<1:0>: PWM # I/O Pin Mode bits(1)
11 = PWM I/O pin pair is in the True Independent PWM Output mode
10 = PWM I/O pin pair is in the Push-Pull PWM Output mode
01 = PWM I/O pin pair is in the Redundant PWM Output mode
00 = PWM I/O pin pair is in the Complementary PWM Output mode

Anexo B – Figura 8 – Distintos modos de operación de un par de pines PWM

2.3. PDCx

Registro que de 16 bits contiene el factor de servicio de la señal PWMx primaria.

2.4. PHASEx

- En modo independiente, este registro indica el período para las salidas PWMxH y PWMxL.
- En modo complementario, este registro indica el desplazamiento con respecto al tiempo maestro, y por lo tanto no se usará.

2.5. TRIGx

Registro que almacena el valor con el que se compara a PHASEx. Si coinciden, PWMCONx<TRGSTAT> se pone a 1.

2.6. DTRx

Registro que almacena el valor de tiempo muerto de generador PWM.

2.7. TRGCONx

Registro de control de *triggers* primario. Destacan:

- **TRGDIV**, un sub-registro de 4 bits que indica el post-escalado para el envío de la señal de *trigger*. Por ejemplo, si TRGDIV = 0b0010, se genera una señal *trigger* cada 3 veces que la TRIGx = PHASEx.
- **TRGSTRT**, sub-registro de 5 bits que indica los ciclos PWM que tienen que pasar antes de generar el primer *trigger*.

2.8. ALTDTRx

Registro de control de *triggers* secundario. **No se utiliza en este caso.**

2.9. FCLCONx

Registro de control de corriente límite y otros parámetros de seguridad. **No se utiliza en este caso.**

2.10. LEBCONx

Registro de control de flancos de subida y bajada (versión 1). **No se utiliza en este caso.**

2.11. LEBDLYx

Registro de control de flancos de subida y bajada (versión 2). **No se utiliza en este caso.**

2.12. AUXCONx

Registro de control auxiliar del PWM. **No se utiliza en este caso.**

2.13. PWMCAPx

Registro que almacena el valor del PWM cuando se detecta la corriente máxima. **No se utiliza en este caso.**

2.14. SDCx

Registro que contiene el factor de servicio secundario de la señal PWMx. **No se utiliza en este caso.**

2.15. SPHASEx

Igual que PHASEx, pero para el canal secundario. **No se utiliza en este caso.**

2.16. STRIGx

Igual que TRIGx, pero para el canal secundario. **No se utiliza en este caso.**

PROGRAMACIÓN DEL DSPIC33FJ32GS606

La programación del módulo PWM se encuentra en el ANEXO D – Código.

ANEXO C – CONFIGURACIÓN DEL MÓDULO ADC EN EL DSPIC33FJ32GS606

En este anexo, se explica en detalle la configuración de pines, registros y la programación del módulo ADC del microcontrolador que se usa para el controlador. En el apartado 4.1.1.3, se han explicado brevemente los registros principales, pero la explicación completa se encuentra a continuación.

MÓDULO ADC

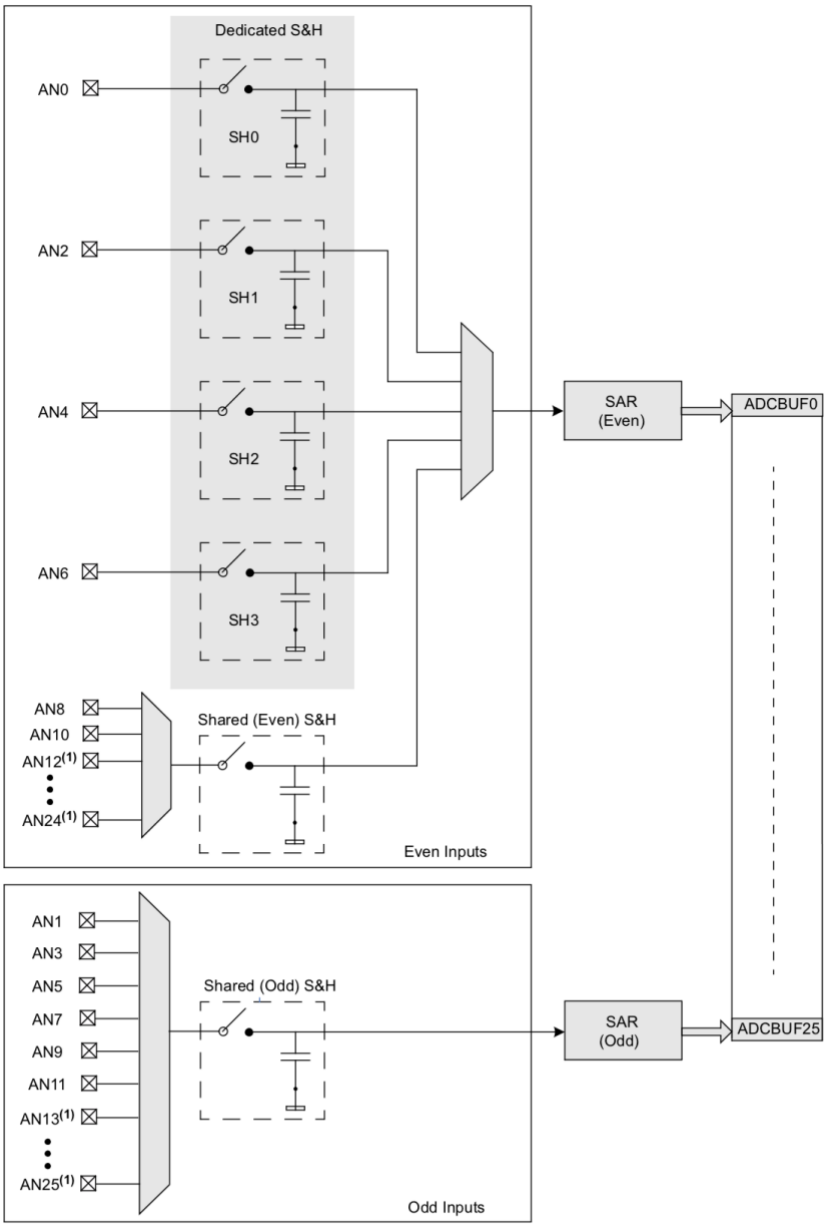
El dsPIC33FJ32GS606 dispone de un módulo analógico-digital de alta velocidad de 10 bit. Tiene dos SARs (*Successive Approximation Register*) y 16 canales para muestrear señales. Como se ha visto en el apartado 4.1.1, se necesitan muestrear 9 señales, por lo que se usarán 9 pines del módulo ADC.

FUNCIONAMIENTO

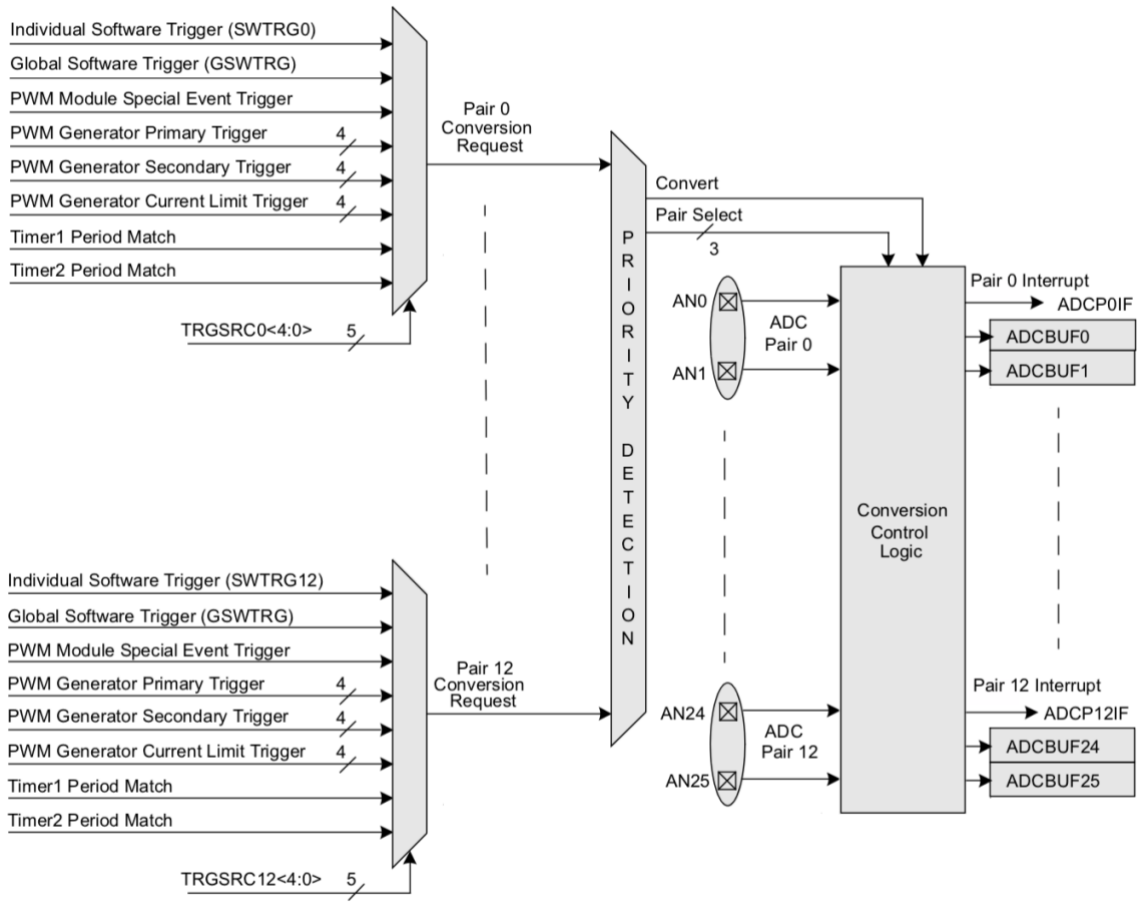
Este microcontrolador consta de un módulo ADC de dos conversores SAR. El funcionamiento es por parejas de pines analógicos. Cada pareja (por ejemplo, Pareja 0 (AN0, AN1), Pareja 1 (AN2, AN3)) recibe una orden de conversión por separado. Esa orden de inicio de conversión puede ser enviada de distintas formas (Anexo B – Figura 2). En este caso, es el generado PWM nº1 quien envía la señal de inicio de conversión.

Si múltiples parejas analógicas reciben una orden de inicio de conversión, las conversiones se priorizan. La Pareja 0 tiene la prioridad más alta mientras que la Pareja 12 tiene la prioridad más baja.

La figura siguiente (Anexo B – Figura 1) presenta el diagrama de bloques del funcionamiento del módulo ADC del microcontrolador. En este módulo, los números pares e impares son muestreados en paralelo. Los números pares son controlados por un SAR y los impares por otro. Esto permite muestrear ambos pines de cada pareja de forma simultánea y en paralelo.



Anexo C – Figura 1 – Módulo ADC de dos SAR [38]



Anexo C – Figura 2 – Control de cada pareja analógica [38]

Lo primero que hay que configurar es la referencia temporal del módulo ADC. De la misma forma que para el módulo PWM, se utilizará el reloj auxiliar (ACLK – *Auxiliary Clock*), que vale:

$$ACLK = \frac{REFCLK \cdot M}{N}$$

Donde $REFCLK$ representa la frecuencia interna del reloj ($7.37MHz$), $M = 16$ si el PLL auxiliar está habilitado y 1 si no lo está (ver ANEXO A – Configuración del oscilador en el dsPIC33FJ32GS606) y N es el post-escalado auxiliar (ver ANEXO A – Configuración del oscilador en el dsPIC33FJ32GS606).

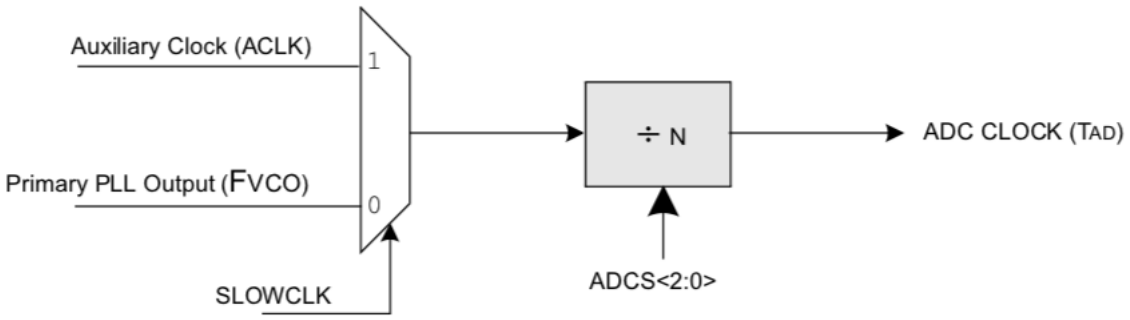
En este caso, $M = 16$ y $N = 1$, por lo que:

$$ACLK = 117.92MHz$$

De [21] (tabla 27-41) se sabe que la velocidad máxima del módulo ADC T_{AD} es de $35.8ns$, es decir una frecuencia de $27.933MHz$. Con lo cual hay que escoger un divisor de reloj:

$$\frac{117.92}{27.933} = 4.22$$

Puesto que el resultado no es entero y se sobrepasa la velocidad máxima al escoger 4, hay que escoger el divisor inmediatamente superior, o sea 5. La siguiente figura (Anexo C – Figura 3) presenta el esquema de bloques de la configuración del reloj para el módulo ADC.



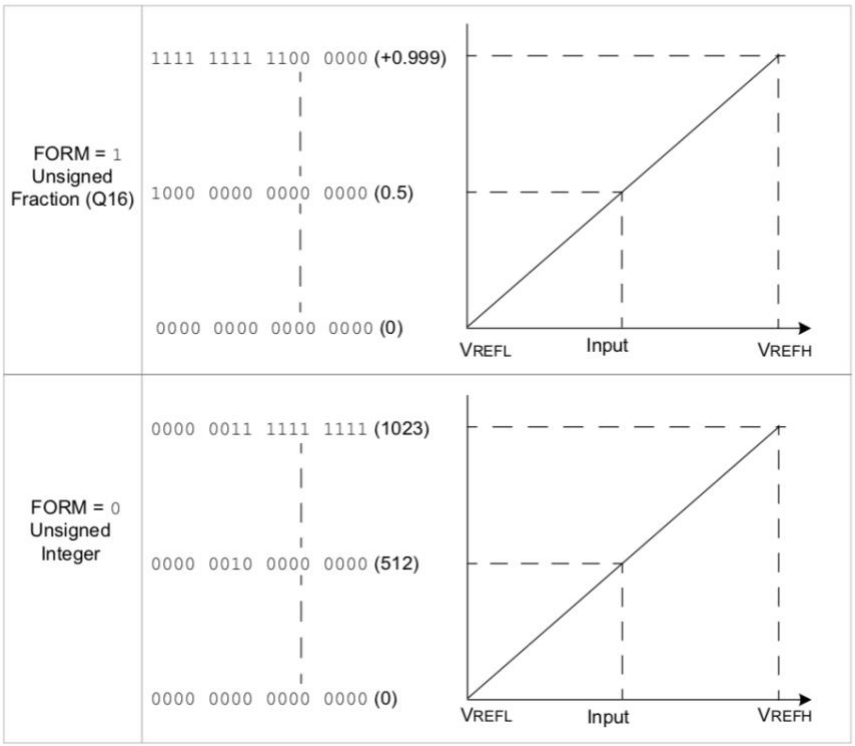
Anexo C – Figura 3 – Configuración del reloj para el módulo ADC [38]

El siguiente paso es escoger el formato en el que muestre el módulo ADC. La figura 4 (Anexo C – Figura 4) presenta las dos posibilidades que existen. Se escogerá *Unsigned Integer*.

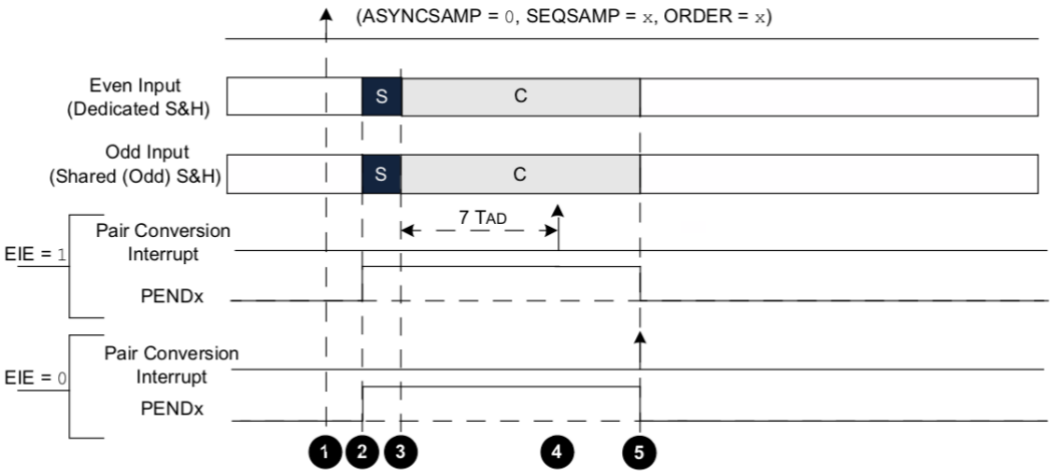
Finalmente, solo queda por configurar el modo de funcionamiento del módulo. Se puede escoger entre una conversión asíncrona o síncrona.

En la conversión síncrona, cuando una pareja recibe la orden de inicio de muestreo, se esperan 2-3 ciclos T_{AD} y a continuación se empiezan a muestrear ambos canales (el muestro dura aproximadamente 2 ciclos T_{AD}. Una vez muestreados se convierten y, si se ha

configurado, al final de la conversión se lanza una interrupción. Esta interrupción se puede disparar 7 ciclos T_{AD} después del comienzo de la conversión o al finalizar esta. La figura 5 (Anexo C – Figura 5) resume lo descrito en este párrafo.

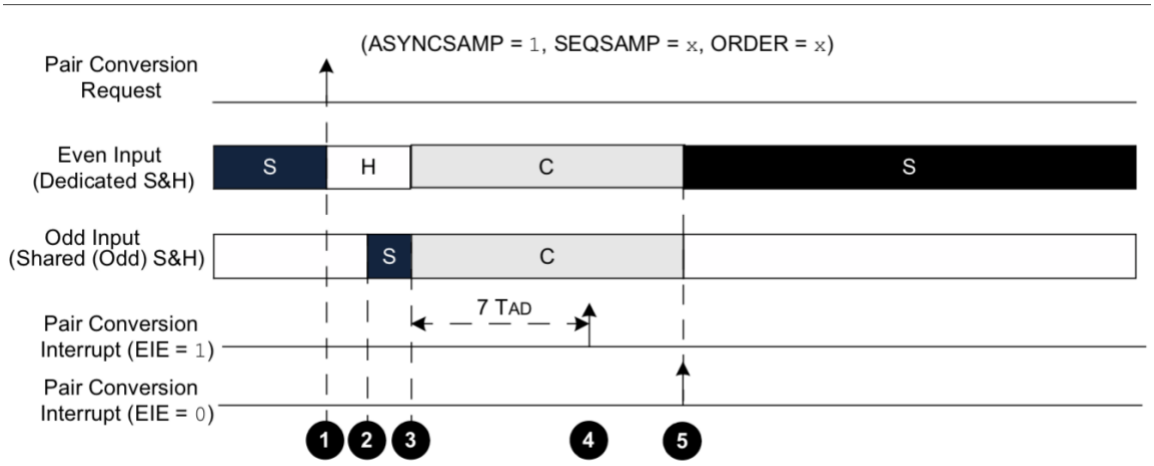


Anexo C – Figura 4 - Formato de muestro del módulo ADC [38]



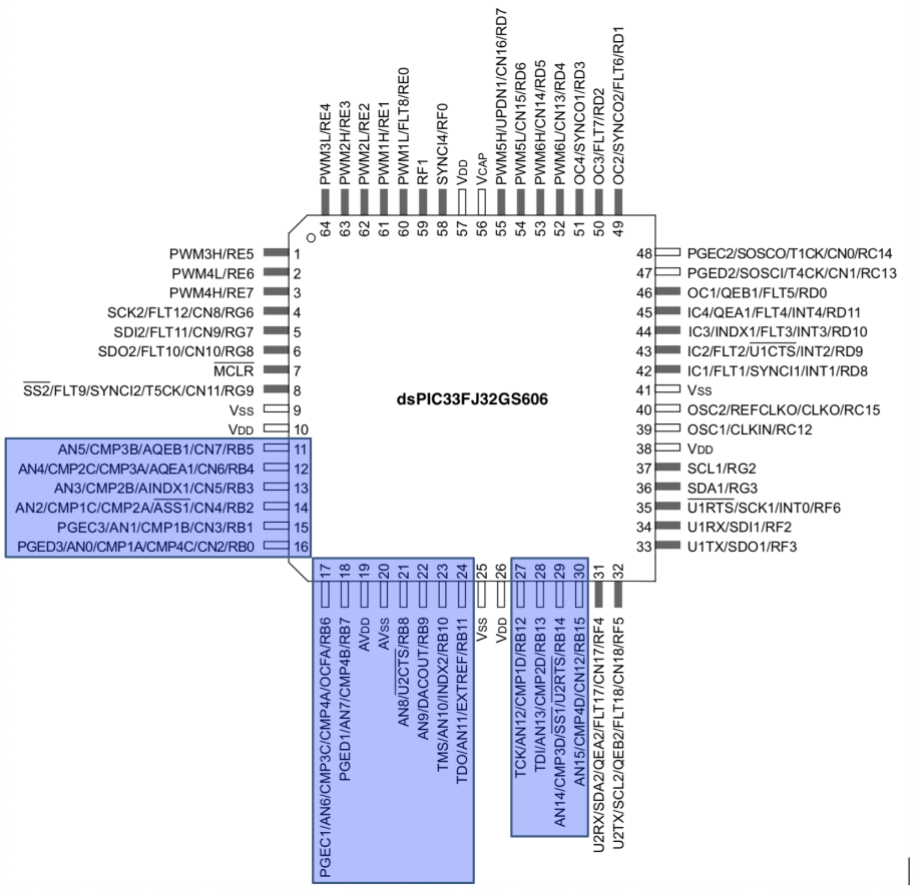
Anexo C – Figura 5 – Modo síncrono del módulo ADC [38]

En el modo asíncrono, el canal que la pareja que tenga el número par está muestreando de forma continua. Cuando se recibe la orden de inicio de muestro, el proceso de muestro termina inmediatamente y se procese a muestrear el otro canal de la pareja. Al finalizar el segundo muestro es cuando se convierten los valores. Las interrupciones funcionan de la misma forma que en el modo síncrono. La siguiente figura (Anexo C – Figura 6) resume lo descrito en este párrafo.



Anexo C – Figura 6 – Modo asíncrono del módulo ADC [38]

PINES



Anexo C – Figura 7 – Pines del módulo ADC en el dsPIC33FJ32GS606 [21]

REGISTROS

1. ADCON

Control principal de configuración del módulo ADC. Contiene los siguientes sub-registros:

- **ADON**, un bit que habilita (1) o deshabilita (0) el módulo ADC.
- **ADSIDL**, un bit que habilita (0) o deshabilita (1) el funcionamiento del módulo ADC en modo suspensión.
- **SLOWCLK**, un bit que elije la referencia del reloj del módulo ADC. En 1, el módulo utiliza el PLL auxiliar (ACLK) y en 0 utilizar el PLL primario (F_{VCO}).

- **GSWTRG**, un bit que se pone a 1 por software para disparar el inicio de conversión global, si así se elige en el sub-registro TRGSRC<4:0> del registro ADCPCx.
- **FORM**, un bit que elije el modo de conversión del muestre. En 1 los números serán fraccionarios y en 0 serán enteros.
- **EIE**, un bit que habilita (1) o deshabilita (0) la interrupción de finalización de conversión de forma rápida.
- **ORDER**, **no habilitado en el dsPIC33FJ32GS606.**
- **SEQSAMP, no habilitado en el dsPIC33FJ32GS606.**
- **ASYNCSAMP**, bit que elije si el modo de funcionamiento es síncrono (0) o asíncrono (1).
- **ADCS**, sub-registro de 3 bits que divide el reloj de referencia por 1 (000) como mínimo y por 8 (111) como máximo.

2. ADSTAT

Registro que contiene 12 sub-registros de un bit (uno para cada pareja) indicando que los datos están listos en el buffer correspondiente.

3. ADBASE

No se utiliza en este caso.

4. ADPCFG

Registro que contiene 16 sub-registros de un bit indicando si el pin está en modo analógico (0) o digital (1).

5. ADPCFG2

Para aquellos microcontroladores que tengan hasta 24 canales ADC, igual que ADPCFG.

6. ADCPCx

La 'x' representa un número de 0 a 6. Cada registro contiene 8 sub-registros: 4 para una pareja. Por ejemplo, ADCPC0 se compone de:

- **IRQEN1**
- **PEND1**
- **SWTRG1**
- **TRGSRC1**
- **IRQEN0**
- **PEND0**
- **SWTRG0**
- **TRGSRC0**

Es decir, los registros para las parejas AN3-AN2 y AN1-AN0 respectivamente.

- **IRQENx**, bit que habilita (1) o deshabilita (0) la generación de interrupción al finalizar la conversión de la pareja en cuestión.
- **PENDx**, bit que indica si la conversión está pendiente (1) o se ha completado (0).
- **SWTRGx**, bit que al ponerse a 1 por software dispara el comienzo del muestro.
- **TRGSRCx**, sub-registro de 5 bits que indica la referencia del comienzo del muestro para la pareja en cuestión. La siguiente figura (Anexo C – Figura 8) ilustra las distintas posibilidades.

11111 = Timer2 period match
11110 = PWM Generator 8 current-limit ADC trigger
11101 = PWM Generator 7 current-limit ADC trigger
11100 = PWM Generator 6 current-limit ADC trigger
11011 = PWM Generator 5 current-limit ADC trigger
11010 = PWM Generator 4 current-limit ADC trigger
11001 = PWM Generator 3 current-limit ADC trigger
11000 = PWM Generator 2 current-limit ADC trigger
10111 = PWM Generator 1 current-limit ADC trigger
10110 = PWM Generator 9 secondary trigger selected
10101 = PWM Generator 8 secondary trigger selected
10100 = PWM Generator 7 secondary trigger selected
10011 = PWM Generator 6 secondary trigger selected
10010 = PWM Generator 5 secondary trigger selected
10001 = PWM Generator 4 secondary trigger selected
10000 = PWM Generator 3 secondary trigger selected
01111 = PWM Generator 2 secondary trigger selected
01110 = PWM Generator 1 secondary trigger selected
01101 = Reserved
01100 = Timer1 period match
01011 = PWM Generator 8 primary trigger selected
01010 = PWM Generator 7 primary trigger selected
01001 = PWM Generator 6 primary trigger selected
01000 = PWM Generator 5 primary trigger selected
00111 = PWM Generator 4 primary trigger selected
00110 = PWM Generator 3 primary trigger selected
00101 = PWM Generator 2 primary trigger selected
00100 = PWM Generator 1 primary trigger selected
00011 = PWM Special Event Trigger selected
00010 = Global software trigger selected
00001 = Individual software trigger selected
00000 = No conversion enabled

Anexo C – Figura 8 – Referencia para el comienzo de muestro de una pareja de pines del módulo ADC [38]

PROGRAMACIÓN DEL DSPIC33FJ32GS606

La programación del módulo ADC se encuentra en el ANEXO D – Código.

ANEXO D – CÓDIGO

CONFIG.C

```
/*
 * File:   config.c
 * Author: Victor
 *
 * Created on 11 March 2018, 17:59
 */

#include <xc.h>
#include "config.h"

/*
 *
 */
void iniClock(){
        // FOSCEL
        // ?FOSCSELbits.IESO = 1;      // Device starts up using internal
FRC, then uses user-seletected reference
        // ?FOSCSELbits.FNOSC = 1;     // Fast RC Oscillator with PLL
        // Select Internal FRC at POR
        _FOSCSEL(FNOSC_FRC);

        // FOSC - NOT USED

        // OSCCON
        // Initiate Clock Switch to Internal FRC with PLL (NOSC = 0b001)
        __builtin_write_OSCCONH(0x01);
        __builtin_write_OSCCONL(0x01);

        // CLKDIV
        // CLKDIVbits.ROI - NOT USED
        // ?CLKDIVbits.DOZE
        // ?CLKDIVbits.DOZEN
        CLKDIVbits.FRCDIV = 0;         // Internal FRC Oscillator
Postscaler bits (/1)
        CLKDIVbits.PLLPOST = 0; // PLL Output Divider 'N2' (/2)
        CLKDIVbits.PLLPRE = 0; // PLL Input Divider 'N1' (/2)
```

```
        // PLLFBD
        PLLFBDbits.PLLDIV = 41; // PLL Multiplier 'M-2' (41)

        // OSCTUN - NOT USED

        // ACLKCON
        ACLKCONbits.SELACLK = 1;       // FRC is source clock for
auxiliary clock divider
        ACLKCONbits.APSTSCLR = 7;      // Auxiliary Clock Divided by 1
        ACLKCONbits.ASRCSEL = 0;       // No clock input
        ACLKCONbits.FRCSEL = 1;                // FRC clock for Clock
Source
        ACLKCONbits.ENAPLL = 1;                // Auxiliary PLL is enabled

        // REFOCON - NOT USED

        // Wait for Clock Switch to occur
        while (OSCCONbits.COSC != 1);

        // Wait for PLL to lock
        while (OSCCONbits.LOCK != 1);

        // Wait for Auxiliary PLL to lock
        while (ACLKCONbits.APLLCK != 1);
}
```

CONFIG.H

```
/*
 * File:   config.h
 * Author: Victor
 *
 * Created on 11 March 2018, 17:59
 */

#ifndef _CONFIG_H
#define _CONFIG_H

// ------------------------------------------------------------------------
-------
```

```
// ------------------------------- PARMETROS ---------------------------
------
// ---------------------------------------------------------------------
-------

/// Frecuencia de operacion del microprocesador (Hz)
#define FCY 39613750

// ---------------------------------------------------------------------
-------
// ------------------- PROTOTIPOS DE LAS FUNCIONES PUBLICAS -------------
-------
// ---------------------------------------------------------------------
-------

/**
 * Initialze FRC clock with PLL
 *
 * Configure oscillator frequency: nominal frequency (Fin) is 7.37MHz
 * To achieve 40 MIPS, FOSC = 80KHz:
 * FOSC = Fin * M/(N1*N2) and  FCY = FOSC/2
 * FOSC = 79.225MHz            so              FCY = 39.61375MHz
 */
void iniClock(void);

#endif
```

PWM.C

```
/*
 * File:   pwm.c
 * Author: Victor
 *
 * Created on 11 March 2018, 17:59
 */

#include <xc.h>
#include "pwm.h"
#include "config.h" // For FCY

/*
 *
 */

void iniPWM(void){

// ------------------------------------------------------------------------
-------
// --------------------------- Primary registers ----------------------------
-------
// ------------------------------------------------------------------------
-------
        // PTCON
        // PTCONbits.PTEN = 1;         // PWM Module is enabled - at the
end
        PTCONbits.PTSIDL = 1;  // PWM runs in CPU idle mode
        // PTCONbits.SESTAT    - Read only
        PTCONbits.SEIEN = 0;   // Special Event Interrupt Enable bit
disabled
        PTCONbits.EIPU = 0;            // Active Period register updates
occur on PWM cycle boundaries
        // PTCONbits.SYNCPOL   - NOT USED
        // PTCONbits.SYNCEN            - NOT USED
        // PTCONbits.SYNCSRC   - NOT USED
        // PTCONbits.SEVTPS            - NOT USED

        // PTCON2
                // Even if PTPER is not used, we need to calculate the
PHASEx times for each
```

```
                // pwm module, which are all equal. We will simply
calculate these times as
                // PTPER for simplifications purposes
                //
                // PTPER = [ (ACLK * 8 * desired_pwm_period) / (PCLKDIV)]
- 8
                // The desired pwm period is 10kHz or 0.0001s, but in
center-aligned mode
                // this is equivalent to a desired period of half that,
so 20kHz
                // ACLK is 117.92MHz
                // PCLKDIV is TBD (assumed 1 in first instance)
                //                                    This yields 47168
or 0xB840
                // MAX PTPER VALUE IS 0xFFFB or 65531 in decimal
                // MIN PTPER VALUE is 8 (This value corresponds to a
desired pwm period of 8.48ns with 117.92MHz)
                // So in theory don't have to use a prescaler, but in
CAM, in order to have
                // a duty cycle that is full width, it has to be equal to
2xPTPER.
                // In this case, 2xPTPER > 65536, so we have to use a
prescaler of /2
                // This will result in a PTPER of 23584
                // As it finishes in 4 and TRIGx's last 3 bits have to be
0, we will use 23580 instead
                // 23580 in hexadecimal is 0x5C1C
                // 47160 in hexadecimal is 0xB838

        PTCON2bits.PCLKDIV = 1; // PWM Input Clock Prescaler Divider (/2)
                                                        // It affects ALL
timing parameters

        // PTPER - NOT USED

        // SEVTCMP - NOT USED

        // MDC - NOT USED

        // STCON - NOT USED

        // STCON2 - NOT USED

        // STPER - NOT USED
```

```
        // SSEVTCMP - NOT USED

        // CHOP - NOT USED

// ------------------------------------------------------------------------
-------
// ------------------------ Secondary registers ------------------------
-------
// ---------------------------- PWM MODULE 1  ---------------------------
-------
// ------------------------------------------------------------------------
-------
        // PWMCON1
        // PWCON1bits.FLSTAT   - Read only
        // PWCON1bits.CLSTAT   - Read only
        // PWCON1bits.TRGSTAT  - Read only
        // PWCON1bits.FLTIEN   - NOT USED
        // PWCON1bits.CLIEN            - NOT USED
        // PWCON1bits.TRGIEN   - NOT USED
        PWMCON1bits.ITB = 1;   // ITB = 1 must be used to use Center-
Aligned mode
        PWMCON1bits.MDCS = 0;  // PDC1 register provide duty cycle
information
        PWMCON1bits.DTC = 1;           // Dead time function is disabled
        // PWCON1bits.DTCP             - NOT USED
        PWMCON1bits.MTBS = 0;  // PWM Generator uses primary master time
base for sync
        PWMCON1bits.CAM = 1;           // Center-aligned mode enabled
        // PWCON1bits.XPRES            - NOT USED
        PWMCON1bits.IUE = 0;           // Updates to PDC registers are
synchronized to the local PWM time base

        // IOCON1
        IOCON1bits.PENH = 1;   // PWM Module controls PWM1H pin
        IOCON1bits.PENL = 1;   // PWM Module controls PWM1L pin
        IOCON1bits.POLH = 0;   // PWM1H is active-high
        IOCON1bits.POLL = 0;   // PWM1L is active-high
        IOCON1bits.PMOD = 0;   // PWM pin pair is in complementary output
mode
        // IOCON1bits.OVERNH   - NOT USED
        // IOCON1bits.OVERNL   - NOT USED
        // IOCON1bits.OVRDAT   - NOT USED
        // IOCON1bits.FLTDAT   - NOT USED
```

```
        // IOCON1bits.CLDAT            - NOT USED
        // IOCON1bits.SWAP             - NOT USED
        // IOCON1bits.OSYNC            - NOT USED

        // PDC1 - CONTROLLED BY FUNCTION

        // PHASE1
        PHASE1 = 0x5C1C;       // Calculated above

        // TRIG1
        TRIG1 = 0;                     // One PWM module triggering the
ADC module is enough

        // DTR1 - NOT USED

        // TRGCON1
        TRGCON1bits.TRGDIV = 9;        // Trigger ADC every 10 match
events
        TRGCON1bits.TRGSTRT = 0;       // First ADC trigger event occurs
after first match event

        // ALTDTR1 - NOT USED

        // FCLCON1 - NOT USED

        // LEBCON1 - NOT USED

        // LEBDLY1 - NOT USED

        // AUXCON1
        AUXCON1bits.HRPDIS = 0;        // High-resolution PWM period is
enabled
        AUXCON1bits.HRDDIS = 0; // High-resolution duty cycle is enabled
        //AUXCON1bits.CHOPSEL  - NOT USED
        //AUXCON1bits.CHOPHEN  - NOT USED
        //AUXCON1bits.CHOPLEN  - NOT USED

        // PWMCAP1 - NOT USED

        // SDC1 - NOT USED

        // SPHASE1 - NOT USED

        // STRIG1 - NOT USED
```

```
// -------------------------------------------------------------------
-------
// ------------------------ Secondary registers ----------------------
-------
// ---------------------------- PWM MODULE 2  --------------------------
-------
// -------------------------------------------------------------------
-------

        // PWMCON2
        PWMCON2bits.ITB = 1;    // ITB = 1 must be used to use Center-
Aligned mode
        PWMCON2bits.MDCS = 0;  // PDC1 register provide duty cycle
information
        PWMCON2bits.DTC = 1;           // Dead time function is disabled
        PWMCON2bits.MTBS = 0;  // PWM Generator uses primary master time
base for sync
        PWMCON2bits.CAM = 1;           // Center-aligned mode enabled
        PWMCON2bits.IUE = 0;           // Updates to PDC registers are
synchronized to the local PWM time base

        // IOCON2
        IOCON2bits.PENH = 1;    // PWM Module controls PWM1H pin
        IOCON2bits.PENL = 1;    // PWM Module controls PWM1L pin
        IOCON2bits.POLH = 0;    // PWM1H is active-high
        IOCON2bits.POLL = 0;    // PWM1L is active-high
        IOCON2bits.PMOD = 0;    // PWM pin pair is in complementary output
mode

        // PDC2 - CONTROLLED BY FUNCTION

        // PHASE2
        PHASE2 = 0x5C1C;        // Calculated above

        // AUXCON2
        AUXCON2bits.HRPDIS = 0;        // High-resolution PWM period is
enabled
        AUXCON2bits.HRDDIS = 0; // High-resolution duty cycle is enabled

// -------------------------------------------------------------------
-------
// ------------------------ Secondary registers ----------------------
-------
```

```
// ----------------------------- PWM MODULE 3  -------------------------
-------
// ---------------------------------------------------------------------
-------

        // PWMCON3
        PWMCON3bits.ITB = 1;    // ITB = 1 must be used to use Center-
Aligned mode
        PWMCON3bits.MDCS = 0;   // PDC1 register provide duty cycle
information
        PWMCON3bits.DTC = 1;    // Dead time function is disabled
        PWMCON3bits.MTBS = 0;   // PWM Generator uses primary master time
base for sync
        PWMCON3bits.CAM = 1;    // Center-aligned mode enabled
        PWMCON3bits.IUE = 0;    // Updates to PDC registers are
synchronized to the local PWM time base

        // IOCON3
        IOCON3bits.PENH = 1;    // PWM Module controls PWM1H pin
        IOCON3bits.PENL = 1;    // PWM Module controls PWM1L pin
        IOCON3bits.POLH = 0;    // PWM1H is active-high
        IOCON3bits.POLL = 0;    // PWM1L is active-high
        IOCON3bits.PMOD = 0;    // PWM pin pair is in complementary output
mode

        // PDC3 - CONTROLLED BY FUNCTION

        // PHASE3
        PHASE3 = 0x5C1C;        // Calculated above

        // AUXCON3
        AUXCON3bits.HRPDIS = 0;         // High-resolution PWM period is
enabled
        AUXCON3bits.HRDDIS = 0; // High-resolution duty cycle is enabled

// ---------------------------------------------------------------------
-------
// --------------------------- Primary registers -----------------------
-------
// ---------------------------------------------------------------------
-------

        PTCONbits.PTEN = 1;     // PWM Module is enabled
}
```

```
void setPDC(unsigned int pdc1, unsigned int pdc2, unsigned int pdc3){
        // PDC = PHASEx corresponds to half the width (in CAM)
        // So PDC max is 47160
        PDC1 = pdc1;
        PDC2 = pdc2;
        PDC3 = pdc3;
}
```

PWM.H

```
/*
 * File:   pwmT.h
 * Author: Victor
 *
 * Created on 11 March 2018, 17:59
 */

#ifndef _PWM_H
#define _PWM_H

// Public Functions

void iniPWM(void);

void setPDC(unsigned int pdc1, unsigned int pdc2, unsigned int pdc3);

#endif
```

ADC.C

```
/*
 * File:   adc.c
 * Author: Victor
 *
 * Created on 11 March 2018, 17:59
 */

#include <xc.h>
#include "pwm.h"
#include "config.h" // For FCY

/*
 *
 */

void iniADC(void){
        // Fastest Tad is 35.8ns or 27.933MHz

        // ADCON
        //ADCONbits = 1;                // At the end
        ADCONbits.ADSIDL = 0;  // Continue working in idle mode
        ADCONbits.SLOWCLK = 1; // ADC is clocked by the auxiliary PLL
(ACLK)
        //ADCONbits.GSWTRG              - NOT USED
        ADCONbits.FORM = 0;             // Sample in integer mode
        ADCONbits.EIE =0;               // ADC pair conversion interrupt is
generated after completing the conversion
        //ADCONbits.ORDER               - NO USE IN GS606
        //ADCONbits.SEQSAMP             - NO USE IN GS606
        ADCONbits.ASYNCSAMP = 0;// Starts sampling when trigger event is
detected, synchronous sampling
        // ACLK is 117.92 so 117.92/4 is higher than max Tad and 117.92/5
is ok
        ADCONbits.ADCS = 4;             // ADC Conversion Clock Divider -
Divide by 5

        // ADSTAT - READ ONLY

        // ADBASE - NOT USED

        // ADPCFG
```

```
        ADPCFGbits.PCFG0 = 0;  // AN0 is configured as analog input
        ADPCFGbits.PCFG1 = 0;  // AN1 is configured as analog input
        ADPCFGbits.PCFG2 = 0;  // AN2 is configured as analog input
        ADPCFGbits.PCFG3 = 0;  // AN3 is configured as analog input
        ADPCFGbits.PCFG4 = 0;  // AN4 is configured as analog input
        ADPCFGbits.PCFG5 = 0;  // AN5 is configured as analog input
        ADPCFGbits.PCFG6 = 0;  // AN6 is configured as analog input
        ADPCFGbits.PCFG7 = 0;  // AN7 is configured as analog input
        ADPCFGbits.PCFG8 = 0;  // AN8 is configured as analog input

        // ADCPC0: AN3&AN2, AN1&AN0
        ADCPC0bits.IRQEN1 = 0; // Interrupt request is not enabled for
AN3 and AN2 pair
        //ADCPC0bits.PEND1             - NOT USED
        //ADCPC0bits.SWTRG1            - NOT USED
        ADCPC0bits.TRGSRC1 = 4;        // 0b00100 PWM Generator 1 primary
trigger selected

        ADCPC0bits.IRQEN0 = 0; // Interrupt request is not enabled for
AN1 and AN0 pair
        //ADCPC0bits.PEND0             - NOT USED
        //ADCPC0bits.SWTRG0            - NOT USED
        ADCPC0bits.TRGSRC0 = 4;        // 0b00100 PWM Generator 1 primary
trigger selected

        // ADCPC1: AN7&AN6, AN5&AN4
        ADCPC1bits.IRQEN3 = 0; // Interrupt request is not enabled for
AN3 and AN2 pair
        ADCPC1bits.TRGSRC3 = 4;        // 0b00100 PWM Generator 1 primary
trigger selected

        ADCPC1bits.IRQEN2 = 0; // Interrupt request is not enabled for
AN3 and AN2 pair
        ADCPC1bits.TRGSRC3 = 4;        // 0b00100 PWM Generator 1 primary
trigger selected

        // ADCPC2: AN11&AN10, AN9&AN8
        ADCPC2bits.IRQEN4 = 1; // Interrupt request is enabled for AN9
and AN8 (pair 4)
        ADCPC2bits.TRGSRC4 = 4;// 0b00100 PWM Generator 1 primary trigger
selected

        // INTERRUPT
    IFS7bits.ADCP4IF = 0;    // Clear ADC pair 4 Interrupt flag
```

```
    IEC7bits.ADCP4IE = 1;    // Enable ADC Pair 4 interrupt

    ADCONbits = 1;           // Enable ADC module
}
```

ADC.H

```
/*
 * File:   adcT.h
 * Author: Victor
 *
 * Created on 11 March 2018, 17:59
 */

#ifndef _ADC_H
#define _ADC_H

// Public Functions

void iniADC(void);

#endif
```

FUNCTIONS.C

```
/*
 * File:   functions.c
 * Author: Victor
 *
 * Created on 27 January 2018, 15:06
 */

/*****************************************************/
/* INCLUDES */

#include <xc.h> // Como se usa el compilador
                // autompcamente las defini
#include <dsp.h>

#include "config.h"
#include "math.h"

/*****************************************************/
/* DEFINES */

#define MAX_RPM 8000     // Max motor RPM
#define N 5              // Number of pair poles
#define PM 0.03          // Permanent magnet flux linkage

#define p_speed 10       // Proportional speed constant
#define i_speed 50       // Integral speed constant

#define p_d_current 10   // Proportional current constant for d axis
#define i_d_current 50   // Integral current constant for d axis

#define p_q_current 10   // Proportional current constant for q axis
#define i_q_current 50   // Integral current constant for q axis

/*****************************************************/

float long integral_speed = 0;
float long integral_d = 0;
float long integral_q = 0;

void calculate_braking_demand(int f_bra, int r_bra, float* f_bra_demand,
float* b_bra_demand){
```

```
    *f_bra_demand = (f_bra - 205)/818.0;
    *b_bra_demand = (b_bra - 205)/818.0;
}

float calculate_angle(int sin, int cos){
    float v_sin = sin/1023.0;
    float v_cos = cos/1023.0;
    float theta = atan(v_sin/v_cos);
    return theta;
}

float calculate_speed(float angle_t0, float angle_t1, float delta_time){
    float delta_angle = angle_t1 - angle_t0;
    float speed = delta_angle/delta_time;
    return speed;
}

float calculate_battery_voltage(int v){
    //1023 -> 96
    // v -> v_dc
    float v_dc = v*96/1023.0;
    return v_dc;
}

float pi_speed(float speed_current, float speed_target){
    float error = speed_target - speed_current;
    integral_speed = integral_speed + error;

    float torque = (p_speed * error) + (i_speed * integral_speed);
    return torque;
    // pi control for speed
    // returns needed torque
}

float pi_current_d(float d_current, float d_target){
    // pi control for current
    float error = d_target - d_current;
    integral_d = integral_d + error;

    float v_d_ref = (p_d_current * error) + (i_d_current * integral_d);
    return v_d_ref;
}

float pi_current_q(float q_current, float q_target){
```

```
    float error = q_target - q_current;
    integral_q = integral_q + error;

    float v_q_ref = (p_q_current * error) + (i_q_current * integral_q);
    return v_q_ref;
}

void clarke_transform(float i_a, float i_b, float i_c, float* i_alpha,
float* i_beta){
    *i_alpha = i_a;
    *i_beta = (1/sqrt(3))*i_a + (2/sqrt(3))*i_b;
}

void inverse_clarke_transform(float v_alpha, float v_beta, float* v_a,
float* v_b, float* v_c){
    *v_a = v_beta;
    *v_b = 0.5*(sqrt(3)*v_alpha - v_beta);
    *v_c = 0.5*(-sqrt(3)*v_alpha - v_beta);
}

void park_transform(float i_alpha, float i_beta, float angle, float* i_d,
float* i_q){
    *i_d = i_alpha*cos(angle) + i_beta*sin(angle);
    *i_q = -i_alpha*sin(angle) + i_beta*cos(angle);
}

void inverse_park_transform(float v_d, float v_q, float angle, float*
v_alpha, float *v_beta){
    *v_alpha = v_d*cos(angle) - v_q*sin(angle);
    *v_beta = v_d*sin(angle) - v_q*cos(angle);
}
```

FUNCTIONS.H

```
/*
 * File:   functions.h
 * Author: Victor
 *
 * Created on 27 January 2018, 15:43
 */

#ifndef FUNCTIONS_H
#define FUNCTIONS_H

void calculate_braking_demand(int f_bra, int r_bra, float* f_bra_demand,
float* b_bra_demand);

float calculate_angle(int sin, int cos);

float calculate_speed(float angle_t0, float angle_t1, float delta_time);

float calculate_battery_voltage(int v);

float pi_speed(float speed_current, float speed_target);

float pi_current_d(float d_current, float d_target);

float pi_current_q(float q_current, float q_target);

float clarke_transform(float i_a, float i_b, float i_c, float* i_alpha,
float* i_beta);

float inverse_clarke_transform(float v_alpha, float v_beta, float* v_a,
float* v_b, float* v_c);

float park_transform(float i_alpha, float i_beta, float angle, float*
i_d, float* i_q);

float inverse_park_transform(float v_d, float v_q, float angle, float*
v_alpha, float *v_beta);

#endif  /* FUNCTIONS_H */
```

MAIN.C

```
/*
 * File:   main.c
 * Author: Victor
 *
 * Created on 27 January 2018, 15:06
 */

/*****************************************************/
/* INCLUDES */

#include <xc.h>
#include <stdio.h>
#include <stdlib.h>
#include <float.h>
#include <math.h>

#include "config.h"
#include "functions.h"
#include "pwm.h"
#include "dsp.h"

/*****************************************************/
/* DEFINES */

#define LQ 0.110e-3     // q-axis inductance
#define LD 0.062e-3     // d-axis inductance
#define PM 0.03         // Permanent Magnet Flux Linkage
#define N 5             // Number of pair poles
#define PWMPRD 10000    // PWM period is 10kHz so 10000 Hz
#define CTRLPWRD 2000   // Control loop period is 2kHz so 2000 Hz
#define MAX_RPM 8000    // Maximum RPM of motor is 8000
#define MAX_CURRENT 250 // Maximum current allowed is 250A for safety

#define MAX_SPEED_DEC_THROTTLE_100 0.3      // When throttle is lifted,
decelerate as max rate of 10km/h
#define MAX_SPEED_DEC_THROTTLE_50_100 0.7
#define MAX_SPEED_DEC_THROTTLE_20_50 1.5
#define MAX_SPEED_DEC_THROTTLE_20 3
#define MAX_SPEED_INC_MODE_2_0_20 1/1000
#define MAX_SPEED_INC_MODE_2_20_60 1/1500
#define MAX_SPEED_INC_MODE_2_60_100 1/2000
```

```
#define MAX_SPEED_INC_MODE_3 1/200

/*********************************************************/
/* GLOBAL VARIABLES */

int an0 = 0;     // Phase a
int an1 = 0;     // Phase b
int an2 = 0;     // Phase c
int an3 = 0;     // Battery Voltaje
int an4 = 0;     // Cosine
int an5 = 0;     // Sine
int an6 = 0;     // Front brake
int an7 = 0;     // Rear brake
int an8 = 0;     // Throttle

int trigger_control_algorithm = 0;

/*********************************************************/
/************************ MAIN ***************************/
/*********************************************************/

int main(void) {
    /***************** Initialize functions ********************/

    iniClock();
    iniPWM();
    iniADC();

    /*
     * TRIS: registro de control asociado a cada puerto
     * del micro que permite configurar sus pines como
     * entradas (1) o salidas (0). Son registros de lectura y escritura.
     *
     * PORT: registro de datos asociado a cada puerto del
     * micro que permite leer un valor (0/1) de los pines
     * configurados como INPUTs, y escribir un valor (0/1)
     * de los pines configurados como OUTPUTs.
     */

    /*************** Initialize PORT and TRIS ******************/

    TRISB = 0x01FF; // AN0-AN8 as inputs
    TRISE = 0x0000; // PWM1-PWM3 as outputs
```

```
    PORTB = 0x01FF; // Read values
    PORTE = 0x01FF; // Write values

    /***************** Initialize variables ********************/

    // Braking
    float f_bra_demand = 0;
    float b_bra_demand = 0;
    float total_brake_demand = 0;

    // Speed and angles
    float angle_t0 = 0;
    float angle_t1 = 0;
    float delta_time = 1/CTRLPWRD;
    float speed_meas = 0;
    float speed_meas_kmh = 0;
    float speed_demand = 0;
    float speed_demand_kmh = 0;
    float speed_demand_before_kmh = 0;
    float speed_demand_kmh_diff = 0;

    // Battery
    float v_dc = 0;

    // Torque
    float torque_demand = 0;

    // Currents
    float i_a_meas = 0;
    float i_b_meas = 0;
    float i_c_meas = 0;
    float i_alpha_meas = 0;
    float i_beta_meas = 0;
    float i_d_meas = 0;
    float i_q_meas = 0;
    float i_d_demand = 0;
    float i_q_demand = 0;

    // Voltages
    float v_d_ref = 0;
    float v_q_ref = 0;
    float v_alpha_ref = 0;
    float v_beta_ref = 0;
    float v_a_ref = 0;
```

```
    float v_b_ref = 0;
    float v_c_ref = 0;

    // Sector determination
    int temp_a = 0;
    int temp_b = 0;
    int temp_c = 0;
    int sector = 0;

    // SVPWM
    float X = 0;
    float Y = 0;
    float Z = 0;
    float t_1 = 0;
    float t_2 = 0;
    int t_a_on = 0;
    int t_b_on = 0;
    int t_c_on = 0;
    float t_1_sat = 0;
    float t_2_sat = 0;

    // Conduction modes
    // Mode should be selected by an infotainment system that does
    // not exist just yet.
    // 1. Sport mode. Max torque available at any moment
    // 2. Normal mode. Torque reduced at low speeds
    // 3. Confort mode. Reduced torque at all speeds
    int mode = 1; // Default mode is default mode
    float decel_rate_throttle = 0;
    float accel_rate = 0;

    /*********************************************************/
    /***************** START OF ALGORITHM ********************/
    /*********************************************************/

    while(1){
        if (trigger_control_algorithm == 1){
            // Initialize variables
            int i_a = an0;          // Phase a
            int i_b = an1;          // Phase b
            int i_c = an2;          // Phase c
            int v_bat = an3;        // Battery Voltage
            int cos_resolver = an4; // Cosine
            int sin_resolver = an5; // Sine
```

```
            int f_bra = an6;         // Front brake
            int r_bra = an7;         // Rear brake
            int throttle = an8;      // Throttle

            // Calculate battery voltage
            v_dc = calculate_battery_voltage(v_bat);

            // Calculate current angle and speed (rad/s)
            angle_t1 = angle_t0;
            angle_t0 = calculate_angle(sin_resolver, cos_resolver);
            speed_meas = calculate_speed(angle_t0, angle_t1, delta_time);
// measured speed in rad/s

            // Calculate braking status/demand
            // This will yield a value between 0 and 1
            // 1 is max braking, 0 is not braking
            calculate_braking_demand(&f_bra_demand, &b_bra_demand);
            // Summed up they give a value between 0 and 2.
            // Max regen brake is achieved when either of them is 1 or if
            // combination of both is 1
            total_brake_demand = (f_bra_demand + b_bra_demand);
            if (total_brake_demand > 1){
                total_brake_demand = 1;
            }

            // Throttle value also indicates regen braking
            // Calculate demanded speed (rad/s) with modes
            // First calculate real demanded speed
            speed_demand = throttle*(MAX_RPM/1023.0)*(2*PI/60.0);

            // Check if accelerating or braking
            speed_demand_before_kmh = speed_demand_kmh;
            speed_demand_kmh = speed_demand*6/(8*PI); // rad/s -> rpm ->
kmh
            speed_meas_kmh = speed_meas*6/(8*PI);

            speed_demand_kmh_diff = speed_demand_kmh - speed_meas_kmh;

            // Check if pilot is decelerating to apply regen braking
            // Deceleration rate is controlled.
            if (speed_demand_kmh_diff < 0){
                // Calculate decel rate
                if (speed_meas_kmh >= 100){
```

```
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_100;
                }else if(50 =< speed_meas_kmh < 100){
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_50_100;
                }else if(20 =< speed_meas_kmh < 50){
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_20_50;
                }else{
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*MAX_SPEED_DEC_THROTTLE_20;
                }
                speed_demand_kmh = speed_demand_kmh -
decel_rate_throttle;
                // If pilot touches brakes, apply a deceleration rate
proportional to
                // braking applied
                // This is only possible when pilot is not accelerating
                if (total_brake_demand > 0.1){
                    decel_rate_throttle =
(speed_meas_kmh/1000.0)*total_brake_demand;
                    speed_demand_kmh = speed_demand_kmh -
decel_rate_throttle;
                }
            // If difference isn't negative it means pilot is
accelerating
            }else{
                // Check if not in sport mode
                if (mode != 1){
                    switch(mode){
                        case 2:
                            if (speed_demand_kmh_diff > 30){
                                if (0 < speed_meas_kmh < 20){
                                    accel_rate = (speed_meas_kmh/1000.0)
+ MAX_SPEED_INC_MODE_2_0_20;
                                }else if (20 < speed_meas_kmh < 60){
                                    accel_rate = (speed_meas_kmh/1500.0)
+ MAX_SPEED_INC_MODE_2_20_60;
                                }else if (60 < speed_meas_kmh < 100){
                                    accel_rate = (speed_meas_kmh/2000.0)
+ MAX_SPEED_INC_MODE_2_60_100;
                                }
                                speed_demand_kmh = speed_demand_kmh +
accel_rate;
```

```
                            }
                            break;
                        case 3:
                            if (speed_demand_kmh_diff > 15){
                                if (speed_meas_kmh < 60){
                                    accel_rate = (speed_meas_kmh/2000.0)
+ MAX_SPEED_INC_MODE_3;
                                }
                            }
                            break;
                    }
                }
            }
            // Convert back to rad/s
            // When accelerating and in mode 1 this line is useless
            speed_demand = speed_demand_kmh*(8*PI)/6;

            // PI Speed control, output is torque
            torque_demand = pi_speed(speed_meas, speed_demand);

            // Transform torque to q current
            i_q_demand = torque_demand*(2/(3*N*PM));

            // Saturate if maxes are reached
            if (i_q_demand > 1.0*MAX_CURRENT){
                i_q_demand = 1.0*MAX_CURRENT;
            }
            if (i_q_demand < -1.0*MAX_CURRENT){
                i_q_demand = -1.0*MAX_CURRENT;
            }

            // Ready the current lectures
            i_a_meas = (i_a*(10/1023.0) - 5)*(0.15/5.0)*2000.0;
            i_b_meas = (i_b*(10/1023.0) - 5)*(0.15/5.0)*2000.0;
            i_c_meas = (i_c*(10/1023.0) - 5)*(0.15/5.0)*2000.0;

            // Transform measured currents to fixed alpha beta system
            // using clarke's transform
            clarke_transform(i_a, i_b, i_c, &i_alpha_meas, &i_beta_meas);

            // Transform fixed alpha beta system to rotating system in d
q componentes
            // using park's transform
```

```
            park_transform(i_alpha_meas, i_beta_meas, angle_t0,
&i_d_meas, &i_q_meas);

            // Current PI Control individual for each component
            v_d_ref = pi_current_d(i_d_meas, i_d_demand);
            v_q_ref = pi_current_q(i_q_meas, i_q_demand);

            // Transform voltages of rotating reference back into fixed
            // alpha and beta system using park's inverse transform
            inverse_park_transform(v_d_ref, v_q_ref, angle_t0,
&v_alpha_ref, &v_beta_ref);

            // Transform alpha beta voltages back to a b c reference
system
            // to start using SVPWM using clarke's inverse transform
            inverse_park_transform(v_alpha_ref, v_beta_ref, &v_alpha_ref,
%v_b_ref %v_c_ref);

            // Locate the sector
            if (v_a_ref > 0){
                temp_a = 1;
            }else{
                temp_a = 0;
            }

            if (v_b_ref > 0){
                temp_b = 1;
            }else{
                temp_b = 0;
            }

            if (v_c_ref > 0){
                temp_c = 1;
            }else{
                temp_c = 0;
            }

            sector = temp_a + 2*temp_b + 4*temp_c;

            // Calculate X, Y and Z
            X = sqrt(3)*v_dc*v_beta_ref;
            Y = (sqrt(3)/2)*v_dc*v_beta_ref + (3/2)*v_dc*v_alpha_ref;
            Z = (sqrt(3)/2)*v_dc*v_beta_ref - (3/2)*v_dc*v_alpha_ref;
```

```
            switch(sector){
                case 1:
                    t_1 = Z;
                    t_2 = Y;
                    break;
                case 2:
                    t_1 = Y;
                    t_2 = -X;
                case 3:
                    t_1 = -Z;
                    t_2 = X;
                case 4:
                    t_1 = -X;
                    t_2 = Z;
                case 5:
                    t_1 = X;
                    t_2 = -Y;
                case 6:
                    t_1 = -Y;
                    t_2 = -Z;
            }

            t_a_on = (PWMPRD - t_1 - t_2)/2;
            t_b_on = t_a_on + t_1;
            t_c_on = t_b_on + t_2;

            // Assign duty cycles to PWM module
            switch(sector){
                case 1:
                    setPDC(t_b_on, t_a_on, t_c_on);
                    break;
                case 2:
                    setPDC(t_a_on, t_c_on, t_b_on);
                    break;
                case 3:
                    setPDC(t_a_on, t_b_on, t_c_on);
                    break;
                case 4:
                    setPDC(t_c_on, t_b_on, t_a_on);
                    break;
                case 5:
                    setPDC(t_c_on, t_a_on, t_b_on);
                    break;
                case 6:
```

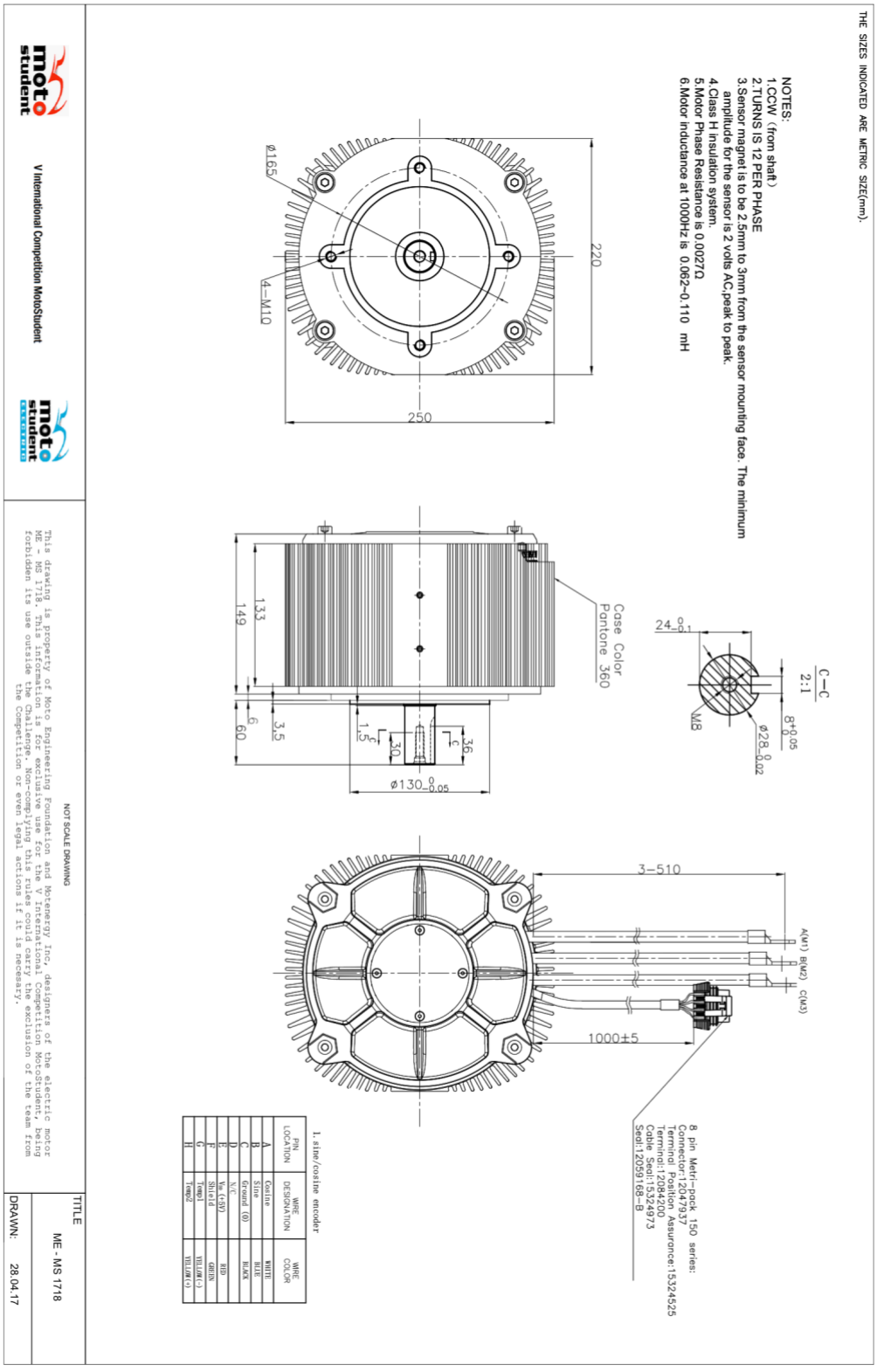
```
                    setPDC(t_b_on, t_c_on, t_a_on);
                    break;

            }
        trigger_control_algorithm = 0;
        } // End if - end of algorithm
    } // End of while
    return 0;
} // End of main function

void __attribute__((interrupt, no_auto_psv)) _ADCP4Interrupt (void){
    an0 = ADCBUF0;
    an1 = ADCBUF1;
    an2 = ADCBUF2;
    an3 = ADCBUF3;
    an4 = ADCBUF4;
    an5 = ADCBUF5;
    an6 = ADCBUF6;
    an7 = ADCBUF7;
    an8 = ADCBUF8;
    trigger_control_algorithm = 1;
    IFS7bits.ADCP4IF = 0;    // Clear ADC pair 4 Interrupt flag
}
```

ANEXO E – HOJAS DE CARACTERÍSTICAS Y EJEMPLOS

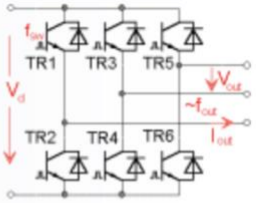
1. Ficha técnica del motor de la competición



2. Ejemplo de resultado de simulación de Semisel

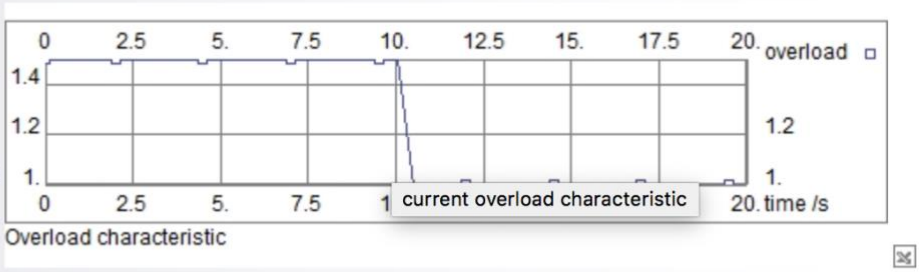
Project:

Topology	DC/AC
Circuit	Inverter 3 Phases
E-Mail	victorguizien@hotmail.com



Circuit:

V_d	96 V
V_{out}	75 V
I_{out}	115 A
P_{out}	15 kW
f_{out}	50 Hz
cos(ϕ)	1
f_{sw}	19 kHz
$V_{min\ out}$	14 V
$f_{min\ out}$	5 Hz
Overload factor	1.5
Overload duration	10 sec



Overload characteristic

Device :

Product line	**SEMITRANS**
Device	**SKM111AR**
Use maximum values	Yes
Max. junction temparature	150°C

Transistor

E_{tr}	1.3 mJ (@50V)
$V_{CE0.125}$	0 V
$r_{C.125}$	14.9 mOhm
$V_{CE.sat}$	1.94 V
I_c	130.00 A
$R_{th(j-c)}$	0.18 K/W
$R_{th(c-s)}$	0.05 K/W

Data set from 2002/02/19

Diode

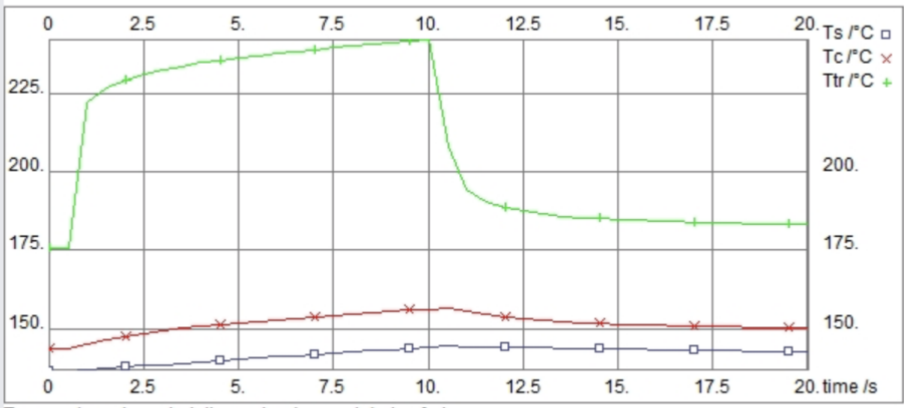
E_d	0.2 mJ
$V_{T0.125}$	0.51 V
$r_{T.125}$	3.45 mOhm
V_f	0.96 V
I_f	130.00 A
$R_{th(j-c)}$	0.18 K/W

Cooling:

Ambient temperature	40°C
Number of switches per heat sink	6
Number of parallel devices on the same heat sink	1
Additional power source at this heat sink	0 W
Predefined SK-Heat Sink	P14_120
Correction factor	1
Forced Air Cooling, Flow Rate:	85 m^3/h
$R_{th(s-a)}$	0.110 K/W

Calculated losses and temperatures with rated current, at overload and at $f_{min\ out}$:

	Rated current	Overload	f_{min} and Overload
$P_{cond\ tr}$	123 W	346 W	177 W
$P_{sw\ tr}$	21 W	38 W	35 W
P_{tr}	147 W	387 W	254 W
$P_{cond\ d}$	0.00 W	0.00 W	36 W
$P_{sw\ d}$	3.21 W	4.76 W	4.40 W
P_d			
P_{tot}	880 W	2323 W	1524 W
	Average Values	Average Values	Maximum Values
T_s	137 °C	154 °C	145 °C
T_c	144 °C	172 °C	157 °C
T_{tr}	170 °C	241 °C	242 °C
T_d			



Temperature characteristic overload current during fmin

Evaluation:

This configuration does not work!

Device driver suggestion

Name	Iout(av) /mA	Îout /A	Visol /kV	Vce max /V	Rgmin / Ohm	Channels
3x SKYPER 32 R or SKYPER 32PRO R	50	15	4.0	1200	1.5	2
3x SKHI22A R or SKHI22B R(1	40	8	2.5	1200	3.0	2
3x SKHI23/12 R	50	8	2.5	1200	2.7	2

Additional Characteristics at given nominal operation conditions with one free parameter - X:

None selected

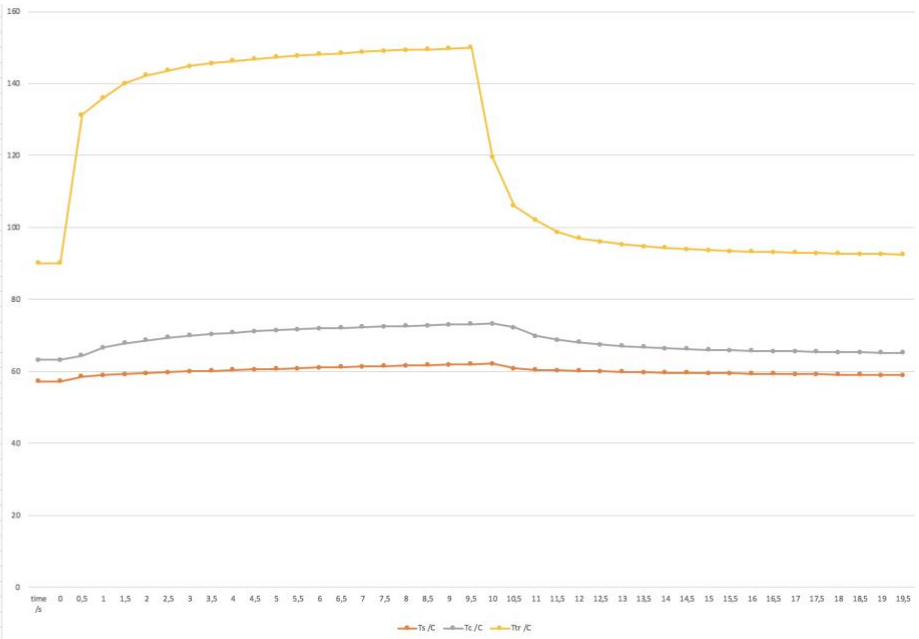
Note

1) A: 15V - Vin; B: 5V - Vin

Change Circuit	Circuit parameter	Device Parameter	Heat Sink Parameter
Printer friendly format	Load File	Save	Project information

3. Ejemplo de datos exportados de Semisel a Excel

time /s	Ts /C	Tc /C	Ttr /C
0	57,2311	63,297	90,1801
0,5	57,2311	63,297	90,1801
1	58,553	64,3828	131,312
1,5	59,0205	66,646	136,136
2	59,3065	67,846	140,066
2,5	59,5642	68,704	142,259
3	59,7937	69,3989	143,692
3,5	60,0055	69,9595	144,769
4	60,205	70,4222	145,614
4,5	60,3942	70,812	146,298
5	60,5744	71,1461	146,868
5,5	60,7463	71,4372	147,353
6	60,9108	71,6945	147,773
6,5	61,0684	71,9252	148,143
7	61,2196	72,1347	148,473
7,5	61,3649	72,327	148,771
8	61,5046	72,5053	149,044
8,5	61,6392	72,6718	149,296
9	61,7691	72,8286	149,531
9,5	61,8944	72,9771	149,752
10	62,0156	73,1183	149,961
10,5	62,1329	73,2533	119,595
11	60,8085	72,2015	106,029
11,5	60,4636	69,9082	102,03
12	60,2942	68,8274	98,702
12,5	60,1479	68,0964	97,0446
13	60,0242	67,5297	96,0264
13,5	59,9137	67,0934	95,2914
14	59,812	66,7506	94,7403
14,5	59,7178	66,4762	94,3134
15	59,63	66,2535	93,9748
15,5	59,5481	66,0699	93,7013
16	59,4715	65,9164	93,4769
16,5	59,3999	65,7861	93,29
17	59,3327	65,6739	93,1318
17,5	59,2697	65,5761	92,9961
18	59,2104	65,4897	92,878
18,5	59,1548	65,4125	92,774
19	59,1024	65,3429	92,6814
19,5	59,053	65,2796	92,5981
20	59,0065	65,2217	92,5224



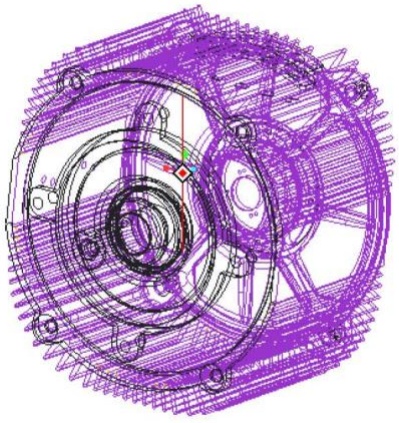
4. Hoja de características del motor de la competición

moto student

V International Competition MotoStudent

2017 - 2018

ME-MS1718 OFFICIAL ELECTRIC MOTOR TECHNICAL SPECS



MEF Moto Engineering Foundation

rfme real federación motociclista española

FIM

TECHNOPARK MOTORLAND

V INTERNATIONAL COMPETITION MOTOSTUDENT

2017-2018

ME-MS1718 OFFICIAL ELECTRIC MOTOR

Motenergy and MEF Technologies have developed the ME-MS1718, the Official Electric Motor for the V International Competition MotoStudent 2017-2018. All teams registered in the Category MotoStudent Electric will receive a unit of this motor within their MotoStudent Kit, compulsory to install in their prototypes.

GENERAL CONSIDERATIONS

The ME-MS1718 Official Electric Motor supplied will be sealed by the MotoStudent Organization to avoid internal manipulations, as reflects Art. D.2.1 of the Technical Regulations.

These seals will avoid the opening of the crankcases and covers, and must be intact at the moment of participation at the Final Event in Autumn 2018. These seals will strictly checked at the Static Scrutineering.

ANY BROKEN OR DAMAGED SEAL WILL BE REASON FOR TECHNICAL NON-CONFORMITY

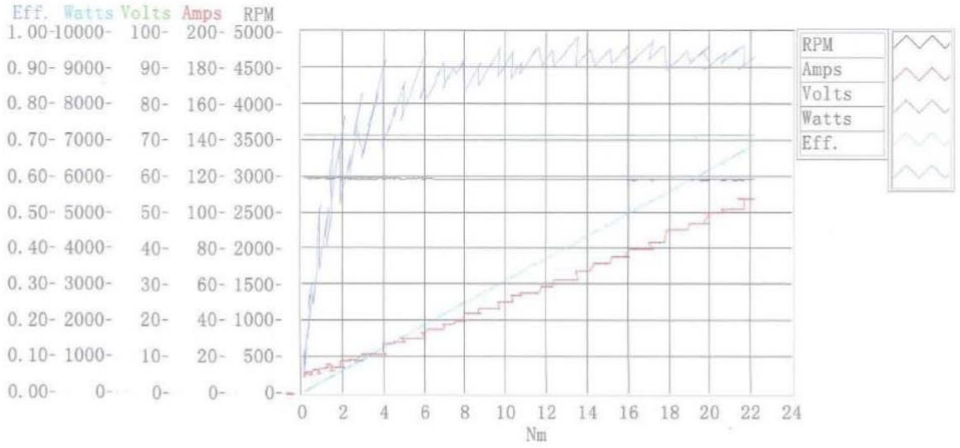
In case of breakdown or malfunction of any internal part, please contact the Organization to take the appropriate solution.

ME-MS1718 TECHNICAL SPECS

TECHNICAL SPECS

Type	RFPM Electric Motor (Brushless)
Operation Voltage	96-116 VDC
RPM Max.	8.000 rpm
Rated power	12 kW
Peak power	20 kW
Peak torque	65 Nm
Cooling	Air cooling system
Weight	21,4 kg

ME-MS1718 QUALITY CHECK GRAPHIC (<5000 RPM)



CONTACT

All technical questions and requests about the ME-MS1718 Official Electric Motor must be directed to the MotoStudent Technical Department:

MotoStudent Technical Department - faq@motostudent.com

TechnoPark MotorLand 44600 - Alcañiz (Teruel) - Spain Tel. +34 978 877 935

www.motostudent.com

FOR MOTOSTUDENT USE ONLY

All the information reflected in this document is confidential and must not be distributed to third parties.

OFFICIAL MOTOR SUPPLIER