ESCOLA TÈCNICA SUPERIOR ENGINYERIA
UNIVERSITAT ROVIRA I VIRGILI
Departament d'Enginyeria Electrònica Elèctrica i Automàtica

Accionamiento con motor *brushless* para ventilador de refrigeración en un automóvil

TITULACIÓ: Enginyeria Automàtica i Electrònica Industrial

AUTOR: Laura Albiol Tendillo
DIRECTOR: Javier Maixé Altès

DATA: 12 / 2009

Índice

Símbolos y abreviaturas 5
1 Introducción 7
2 Objetivos 9
3 Estado del arte 11
4 El motor *brushless* DC 13
4.1 Características constructivas 13
4.1.1 Estator 13
4.1.2 Rotor 14
4.2 Funcionamiento 14
5 Características del motor utilizado y su carga 17
5.1 Determinación del punto de trabajo 18
6 Modelado mediante Ansoft-Simulink 23
6.1 Extracción de características 23
6.2 Paso de información a Simulink 23
6.3 Modelado matemático 24
6.4 Modelado mediante *simulink* 26
6.4.1 Descripción de los bloques 28
6.5 Simulación del accionamiento 30
7 Control *sensorless* 33
7.1 Introducción 33
7.2 Ventajas e inconvenientes 33
7.3 Recopilación de técnicas de control sin sensores 34
7.3.1 Técnicas que usan ecuaciones y mediciones 34
7.3.2 Técnicas que usan observadores 34
7.3.3 Técnicas que usan la fuerza contraelectromotriz inducida 34
7.4 Principio de funcionamiento 35
7.4.1 Muestreo y retención de las BEMF 36
7.4.2 Filtrado de las BEMF 41
7.4.3 Baja velocidad 49
7.4.4 Alta velocidad 52
7.5 Implementación 55
7.5.1 Algoritmo de arranque 55
7.6 Ensayo 58

8 Introducción a la previsión de fallos 61
8.1 Posibles causas de fallo 61
8.2 Métodos para evitar fallos o minimizar sus efectos 62
9 Conclusiones 63
10 Futuras vías de trabajo 65
Referencias 67
11 Anexo I: Características del motor usado 69
12 Anexo II: Transferencia de datos del modelo 73
12.1 corrents.m 73
12.2 motor.m 74
12.3 pi.m 74
12.4 trapez.m 74
12.5 inici.m 75
13 Anexo III: Obtención del filtro digitalizado 77
14 Anexo IV: Adaptación de señales para el conversor A/D 79
15 Anexo V: Código 83
15.1 filtro_digital.s 83
15.2 IIRT_filter.s 84
15.3 main.c 88
15.4 comm.h 94
15.5 defs.h 94
15.6 filtro_digital.h 95
15.7 funcs_AD.h 96
15.8 IIR_filter.h 97
15.9 in_PWM.h 97
15.10 ini_tmrs.h 98
15.11 med_ev.h 98
15.12 vars.h 100

Símbolos y abreviaturas

BEMF	Back ElectroMotive Force (fuerza contraelectromotriz)
BLDC	BrushLess Direct Current (motor sin escobillas de corriente continua)
DC	Direct Current (Corriente Continua)
FEM	ElectroMotive Force (fuerza electromotriz)
ISR	Interrupt Service Routine (rutina de servicio a la interrupción)
PWM	Pulse Width Modulation (modulación por anchura de pulsos)

1 Introducción

En este proyecto se estudia y desarrolla una alternativa viable para el accionamiento del ventilador del radiador de un automóvil mediante un motor tipo *brushless*. Para ello, se establecen una serie de objetivos, descritos de forma resumida en el capítulo 2.

A continuación se procede a realizar un repaso sobre el estado del arte de este tipo de motor en los motores auxiliares de los turismos que se encuentran en el mercado. Seguidamente, en el punto 4 del documento se describe el funcionamiento del motor *brushless*. Además, se introducen algunos parámetros constructivos que influyen en el comportamiento eléctrico y mecánico del accionamiento.

En el 5, se presenta el motor y la carga utilizados para realizar los ensayos. Esta información es estrictamente necesaria para realizar el modelo del accionamiento mediante *Matlab-Simulink*. Este procedimiento se relata en el capítulo 6.

En el apartado número 7 se presenta la necesidad de prescindir de los sensores del motor *brushless*, y se presenta el método seguido para hacerlo. Además, se describen los algoritmos implementados. También se presentan los resultados del ensayo realizado para comprobar el correcto funcionamiento del accionamiento. Para completar el sistema, en el capítulo 8 se perfilan algunos fallos del motor fácilmente detectables, y como identificarlos usando el sistema actual o añadiendo algunas características.

En los puntos 9 y 10 se recogen las conclusiones del presente estudio y las posibles futuras vías de trabajo que se proponen, respectivamente.

Se han añadido varios anexos. El primero es una recopilación de las características del motor usado en el laboratorio. El segundo es el código que realiza el procedimiento de extracción de datos del modelo del motor y su introducción en el entorno *Simulink*. El tercero es la transcripción del código usado para la evaluación del filtro utilizado. En el cuarto se recogen el esquema de la placa de adaptación de señales, juntamente con el trazado y la imagen del circuito una vez soldados todos los componentes. Finalmente, en el quinto anexo se presenta el código desarrollado para realizar el control sin sensores del accionamiento de un ventilador mediante un motor *brushless*.

2 Objetivos

Los objetivos planteados para el correcto desarrollo del proyecto son los siguientes:

- Comprender el funcionamiento del motor *brushless*.
- Obtener un conjunto de motor y carga de características comparables a la aplicación real en el automóvil.
- Realizar el modelo del motor elegido mediante un programa de caracterización de motores.
- Extraer la información del modelo del motor y crear una simulación de éste en un entorno *Matlab-Simulink*.
- Elegir un algoritmo de control sin sensores y aplicarlo, mediante un dsPIC, al conjunto motor-ventilador.
- Describir las características de funcionamiento del accionamiento realizado, y compararlas con la aplicación original.
- Introducir la predicción de fallos en motores y describir cómo se podría aplicar en el sistema desarrollado.

3 Estado del arte

Hasta mediados de los años 90, los motores usados en las aplicaciones auxiliares del automóvil, tales como ventiladores, limpiaparabrisas, elevador de ventanillas, etc. han sido los motores de corriente continua. Sólo el 5 % de los accionamientos en este campo usan mototes de corriente continua sin escobillas, o *brushless*, según *The World Market for Electrical Motors in Automotive Applications* [18].

Esta diferencia abismal se debe principalmente a la sencillez del accionamiento del motor de continua. Se realiza a partir de la batería existente en el automóvil, sin ningún requerimiento de control. El único dispositivo necesario para encender y parar el motor es un relé. Este factor hace que el coste sea bajo, requerimiento imprescindible para que la industria automovilística decida consumirlo en grandes cantidades.

Sin embargo, este tipo de motores tienen características bastante pobres por lo que a mantenimiento y esperanza de vida se refiere. Es natural, pues, que se estén barajando varias alternativas para sustituir los motores de corriente continua, y entre ellas los motores *brushless*.

Las ventajas que el uso del motor *brushless* supone son varias. Para empezar, tal como su nombre indica, no dispone de escobillas. Éstas están sometidas a un considerable desgaste debido al rozamiento. Además, como resultado de la fricción, el grafito que se desprende de las escobillas puede entrar en contacto con los devanados. Esto provoca un cortocircuito entre varias espiras, reduciendo el número de espiras efectivas, o entre una espira y tierra, causando el fallo el motor.

Otros factores que convierten el motor *brushless* en una opción altamente atractiva son su alta eficiencia, una buena densidad de potencia, baja inercia por la ligereza del rotor y bajo ruido acústico.

Por otra parte, este motor presenta dos inconvenientes. El primero es la complejidad del control a usar. El segundo, y más importante, es el coste, que excede el del motor de continua por el uso de materiales como los imanes permanentes de tierras raras y una inferior escala de producción.

Sin embargo, las expectativas de mercado son muy buenas, ya que los vehículos híbridos y eléctricos, con una creciente presencia en el parque automovilístico internacional, suponen una demanda en constante aumento de motores altamente eficientes, como son los motores *brushless*.

4 El motor *brushless* DC

El motor *brushless* es ampliamente usado en una gran variedad de aplicaciones y entornos, debido, entre otros motivos, a su alta densidad de potencia, la fiabilidad y ausencia de escobillas. La alta densidad de potencia permite obtener accionamientos de bajo volumen y peso.

El motor de corriente continua sin escobillas de imanes permanentes consiste de un estator con una serie de slots devanados y un rotor con varios polos, creados con imanes permanentes.

4.1 Características constructivas

Las características constructivas de cualquier motor, y por lo tanto también de los *brushless*, determinan el comportamiento tanto eléctrico como mecánico. Es por ello que, para modelar un motor ya existente, es necesario conocer con el máximo detalle todas las medidas y materiales usados en su construcción.

4.1.1 Estator

El estator del motor se construye a partir de una serie de láminas de acero apiladas. En ellas se realizan una serie de aperturas, llamadas ranuras, que es donde se ubican los devanados. Estas ranuras pueden adoptar diferentes formas, algunas de las cuales se representan en la figura 4.1.

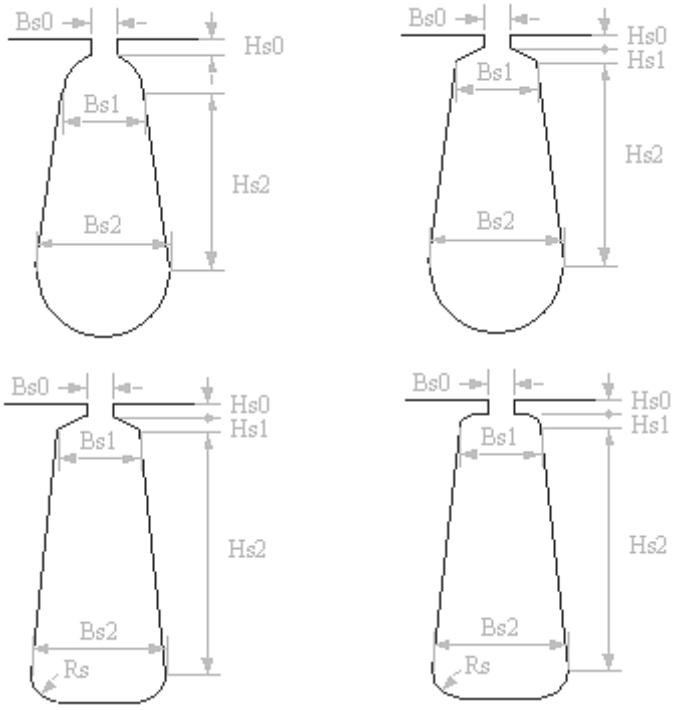


Figura 4.1. Posibles formas de las ranuras y medidas relevantes.

El devanado del estator determina en gran medida el funcionamiento del motor. Son características de diseño el número de fases, su distribución, el número de espiras, el diámetro del conductor, ...

4.1.2 Rotor

Por otra parte, el rotor dispone de varios imanes permanentes. El uso extendido de imanes permanentes empezó alrededor del año 1940, con la aparición del Alnico [17]. Poco después se descubrieron los imanes de ferrita. En 1970 se introdujeron los imanes de samario-cobalto (SmCo), con altas densidades de energía magnética. Otra generación de imanes de tierras raras se sumaría al SmCo el 1983, NdFeB. Estos imanes de neodimio, hierro y boro superan al SmCo en la densidad de energía, además de ser más económicos, ya que el neodimio es una de las tierras raras más comunes.

4.2 Funcionamiento

Alimentando los devanados presentes en el estator se crea un campo electromagnético. Los imanes permanentes se alinearan de acuerdo con este campo, provocando el giro del rotor. El inversor usado para excitar un motor *brushless DC* de tres fases conectadas en estrella es el siguiente:

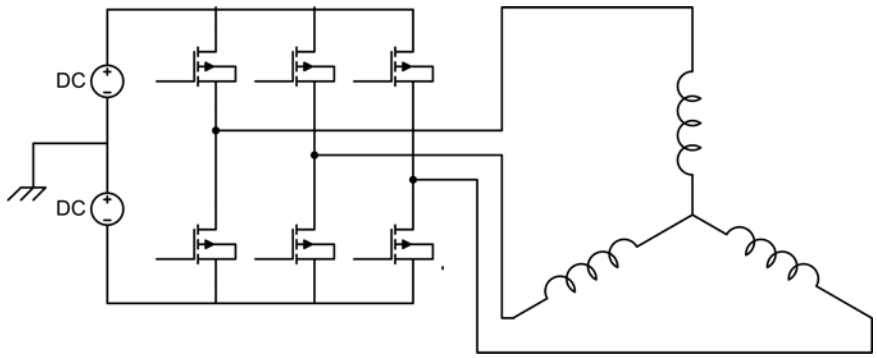


Figura 4.2. Inversor y devanados de un motor de tres fases.

La estrategia de excitación de las fases del motor que se va a usar es la conmutación de seis pasos. Básicamente, esta estrategia de conmutación consiste en aplicar tensión positiva en un terminal una fase, negativa en otro y el último se deja en circuito abierto. El par máximo se consigue cuando el campo del devanado del estator y el de los imanes permanentes del rotor están desfasados 90º. Para ello, la estrategia adoptada consiste en dividir una revolución en 6 partes, de 60º cada una, considerando el devanado de la figura 4.3.(a), distribuido como se muestra en la figura 4.3.(b).

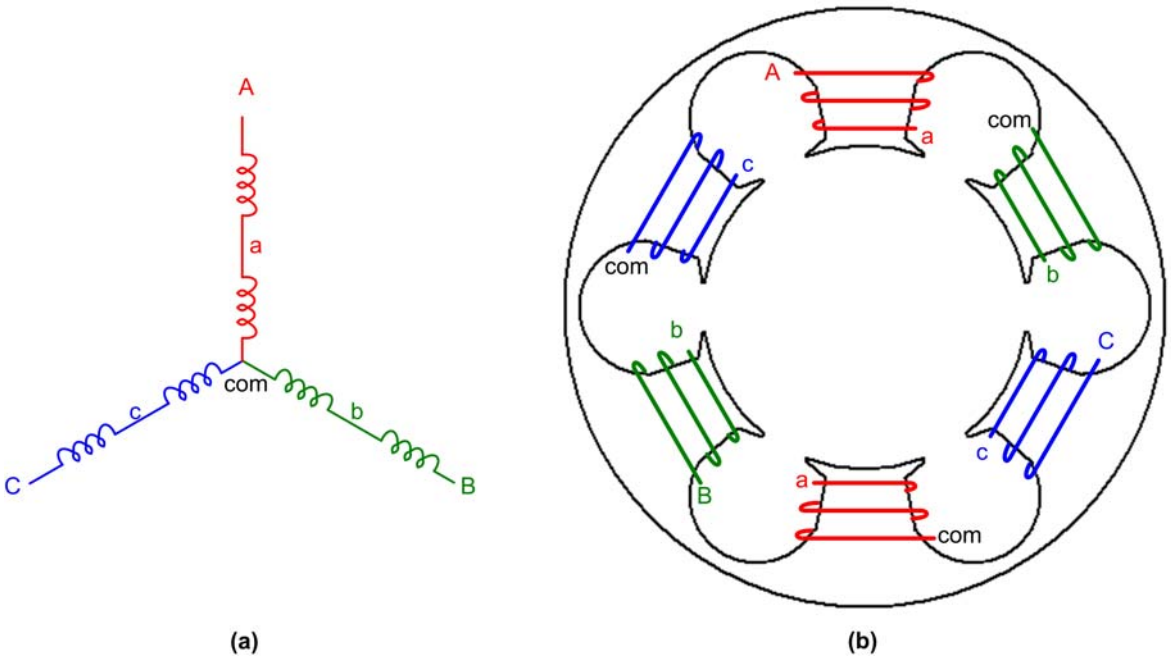


Figura 4.3. (a) Partes de los devanados. (b) Distribución física de estas partes.

La secuencia de excitación que conseguiría un giro del rotor en sentido contrario a las agujas del reloj es la representada en la figura 4.4.

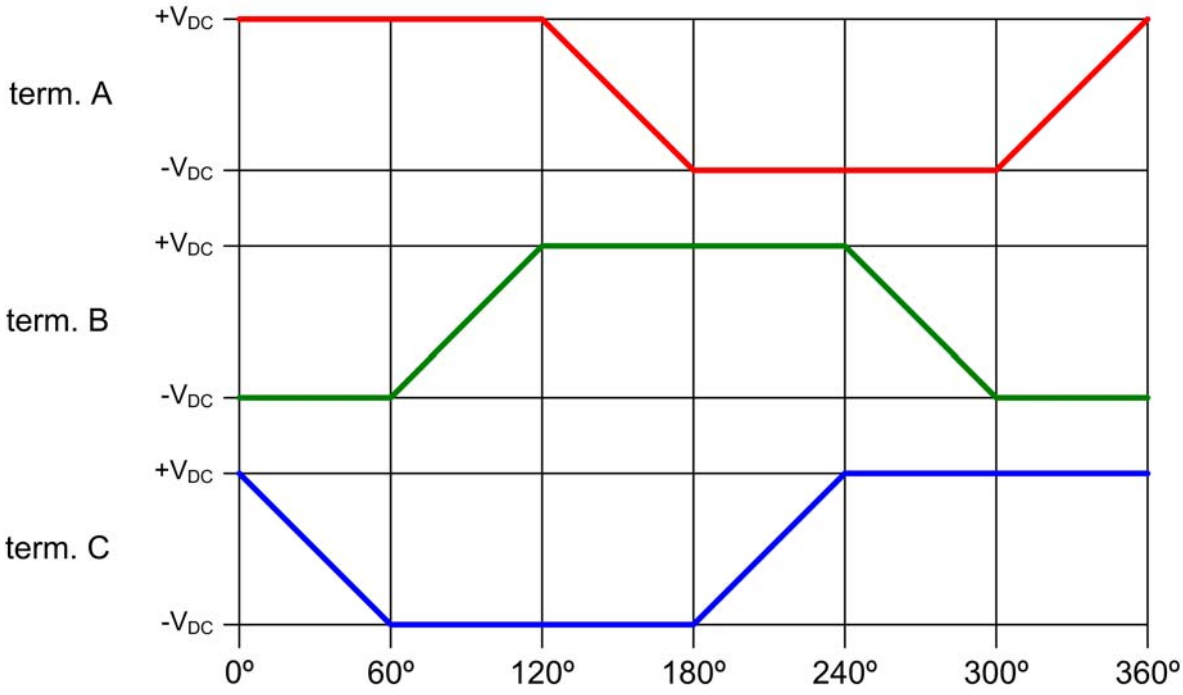


Figura 4.4. Secuencia de excitación de los terminales A, B y C.

A continuación se muestra la evolución de la posición del rotor de acuerdo con la secuencia de excitación anterior.

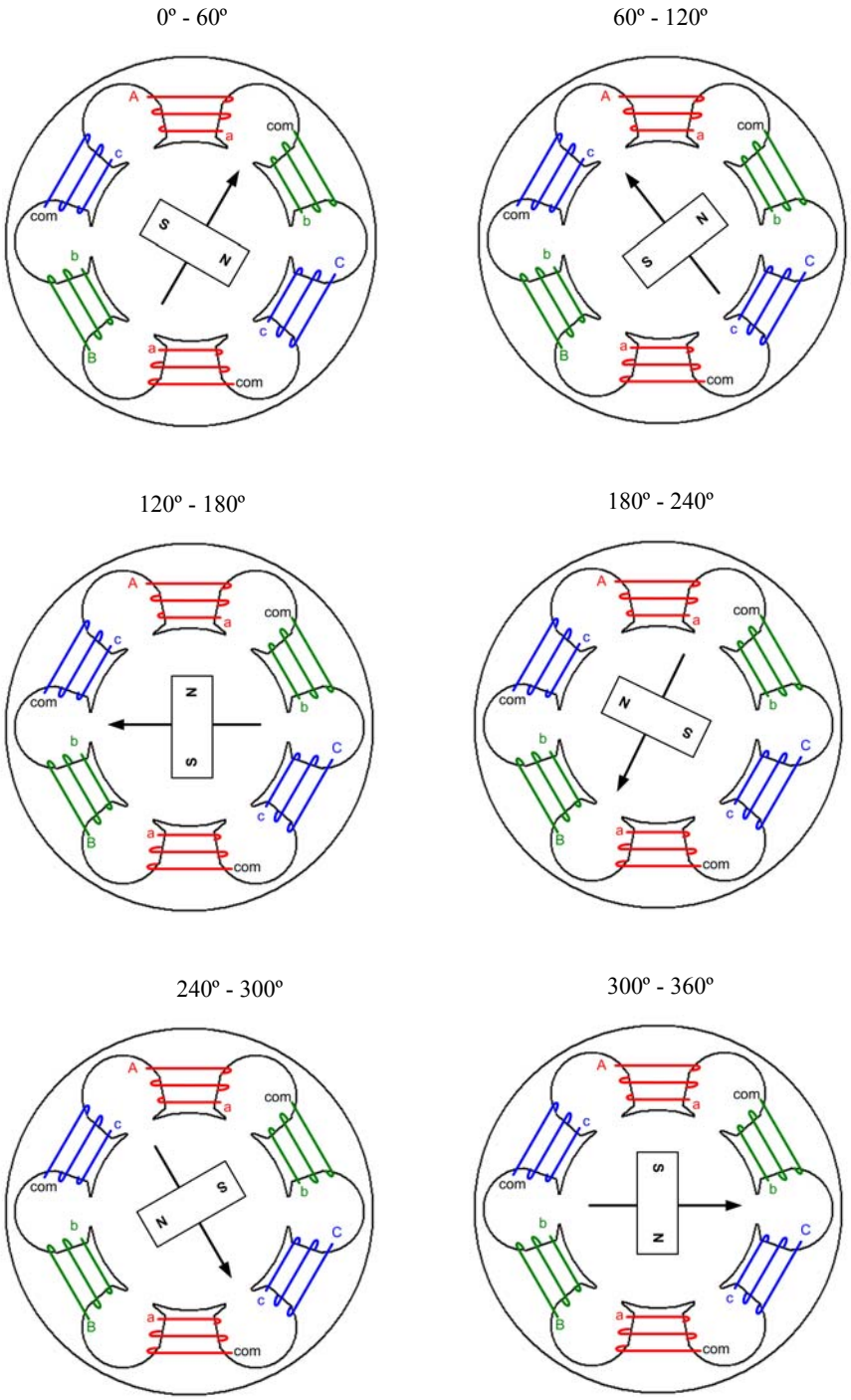


Figura 4.5. Evolución de la posición del rotor mediante la conmutación de seis pasos.

De forma simplificada, se representa el campo creado por los devanados con una flecha, y el rotor se simplifica a un solo dipolo.

5 Características del motor utilizado y su carga

La aplicación estudiada es el accionamiento del ventilador de un radiador de automóvil mediante un motor brushless dc. Para esta aplicación se ha utilizado un ventilador de un Renault Clio, que en la realidad está accionado por un motor DC alimentado a 12 V, cuyo conjunto se muestra en la figura 5.1.

Figura 5.1. Imagen anterior y posterior del ventilador usado en la parte experimental.

En nuestro caso el ventilador es accionado por un motor brushless dc con referencia DMB0224C10002. El motor es comercializado por *Microchip* como parte de un hardware de desarrollo de control de motores. La potencia desarrollada por este motor no es comparable al de la aplicación original. A pesar de ello, se ha usado este motor porque ya se disponía de él en el laboratorio. Su hoja de características se ha incluido en el Anexo I: Características del motor usado. Para poder acoplar motor y ventilador fue necesario fabricar un pie para elevar el motor y permitir así el giro al ventilador. Este se puede observar en la figura 5.2.

Figura 5.2. Motor brushless con su soporte.

Sin embargo, este motor no ofrecía suficiente par para hacer girar el ventilador a una velocidad en que este fuera realmente eficaz. Por lo tanto se ha optado por utilizar un ventilador de dimensiones inferiores. Con ello, el conjunto motor-ventilador obtenido para implementar el algoritmo de control y realizar los ensayos pertinentes es una versión a escala reducida de lo que sería la aplicación real.

El resto del hardware empleado es un *dsPICDEM MC1Motor Control Development Board* y un *dsPICDEM MC1L 3-Phase Low Voltage Power Module*. El primer elemento se trata del propio microprocedor, junto con una serie de entradas fácilmente accesibles, interruptores, LEDs, una pantalla LCD, un conector para programar el chip y una interfície con el módulo de potencia. Éste consta básicamente del inversor que va a alimentar cada una de las fases del motor de forma independiente.

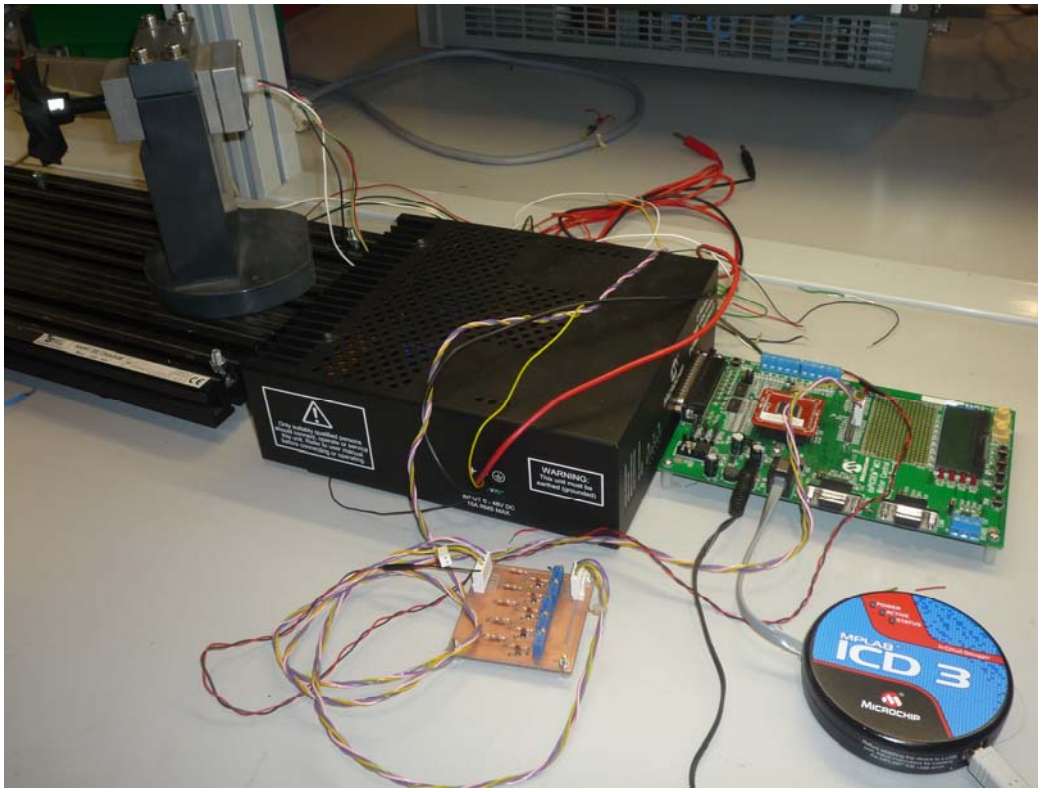


Figura 5.3. Montaje para ensayo y validación del control.

5.1 Determinación del punto de trabajo

Originalmente, como ya se ha indicado, el ventilador de la figura 5.1 era accionado por un motor de corriente continua de 12 V. Es por esta razón que en principio nos interesa determinar el punto de trabajo del conjunto motor-ventilador en su aplicación primigenia, el cual se tomará como referencia central para definir el margen de regulación del ventilador accionado por el nuevo motor brushless.

Tal como ya se ha mencionado, la aplicación original consiste en un motor de continua que acciona el ventilador del radiador de un automóvil. El motor es alimentado por una batería de 12 V nominales, pero en realidad la tensión puede oscilar entre los 9 y los 14 V. Por lo tanto, el ensayo realizado contempla este margen de tensión.

v [V]	i [A]	P [W]	n [rpm]
9	5,4	48,6	2332
9,5	5,74	54,53	2456
10	6,06	60,6	2585
10,5	6,45	67,725	2716
11	6,85	75,35	2850
11,5	7,27	83,605	2963
12	7,75	93	3079
12,5	8,19	102,375	3204
13	8,65	112,45	3307
13,5	9,15	123,525	3416
14	9,6	134,4	3526

Tabla 5.1. Resultados del ensayo para el accionamiento con motor de continua.

Es decir, el motor girará a una velocidad comprendida entre los 2332 y los 3526 rpm, dependiendo únicamente de la tensión que tenga la batería. Para este rango de funcionamiento, el consumo varía entre los 48,4 W y los 134,4 W. La curva completa se representa en la figura 5.4.

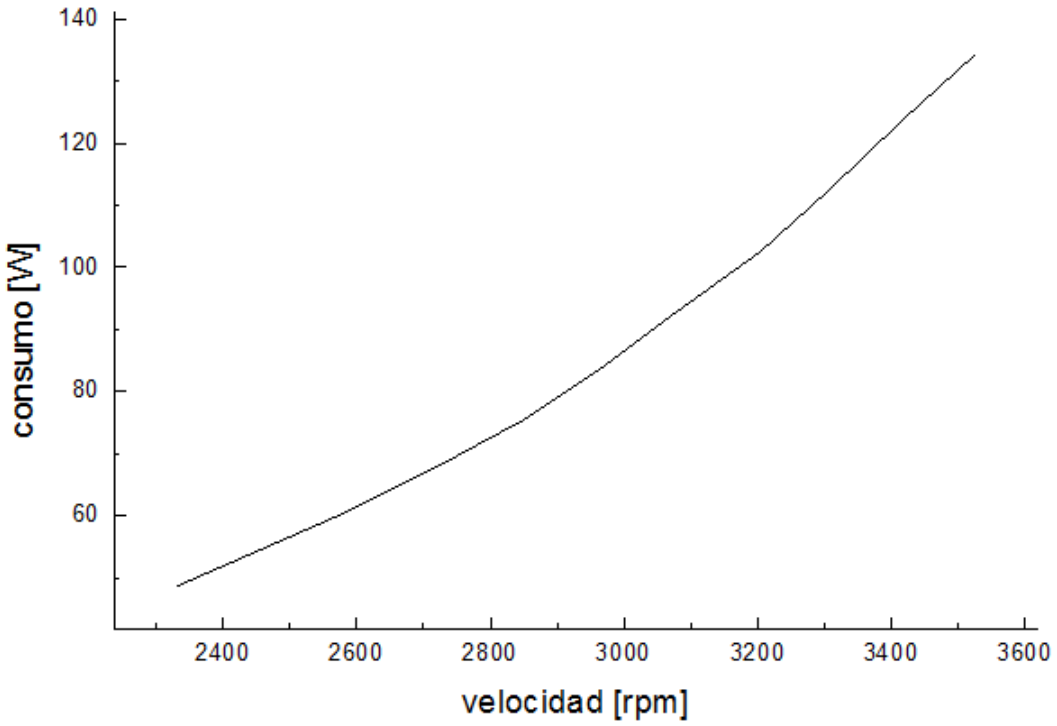


Figura 5.4. Consumo en función de la velocidad para el accionamiento con el motor de continua.

A partir de los datos obtenidos anteriormente, es posible calcular el par electromagnético que genera el motor. Se parte del modelo básico de un motor de corriente continua, representado en la figura 5.5.

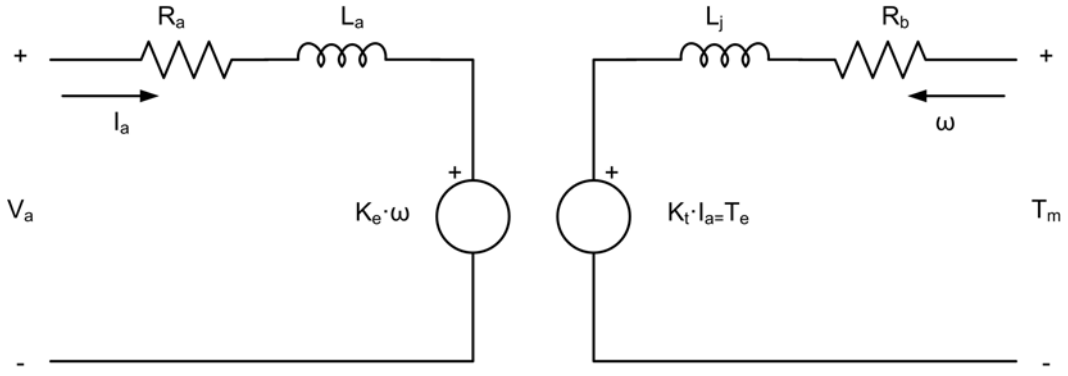


Figura 5.5. Modelo de un motor de corriente contínua.

Analizando la armadura se obtiene la expresión siguiente

$$V_a = R_a \cdot I_a + L \cdot \frac{dI_a}{dt} + K_e \cdot \omega \qquad (1)$$

En estado estacionario, tal como se ha realizado el ensayo, la derivada de la corriente es nula, y en este caso

$$V_a = R_a \cdot I_a + K_e \cdot \omega \qquad (2)$$

Tomando dos resultados del ensayo del motor de corriente continua con el ventilador utilizado para refrigerar el radiador del motor se plantea el siguiente sistema de ecuaciones

$$\left.\begin{aligned} 10\ \text{V} &= R_a \cdot 6{,}06\ \text{A} + K_e \cdot 271\ \text{rad/s} \\ 12\ \text{V} &= R_a \cdot 7{,}75\ \text{A} + K_e \cdot 322\ \text{rad/s} \end{aligned}\right\} \qquad (3)$$

Resolviendo, se obtienen los valores de la resistencia de armadura y la constante eléctrica del motor

$$K_e = 0{,}03386\ \frac{\text{V} \cdot \text{s}}{\text{rad}}, \quad R_a = 141{,}7\ \text{m}\Omega \qquad (4)$$

Una vez conocidos estos valores, es posible calcular el par que genera el motor en estado estacionario a partir del consumo de corriente, mediante la expresión

$$T_e = K_t \cdot I_a \qquad (5)$$

Utilizando la expresión anterior se ha completado el ensayo

i [A]	n [rpm]	T_e [N·m]
5,4	2332	0,18283043
5,74	2456	0,19434197
6,06	2585	0,20517637
6,45	2716	0,21838079
6,85	2850	0,23192379
7,27	2963	0,24614393
7,75	3079	0,26239552
8,19	3204	0,27729282
8,65	3307	0,29286726
9,15	3416	0,30979601
9,6	3526	0,32503187

Tabla 5.2. Resultados del ensayo con el par generado por el motor.

En la figura 5.6 se representa la relación entre la velocidad de giro del motor y el par que éste desarrolla para el accionamiento estudiado. Se ha realizado una regresión polinómica para comprobar que la característica es parabólica, tal como corresponde a una carga de ventilador.

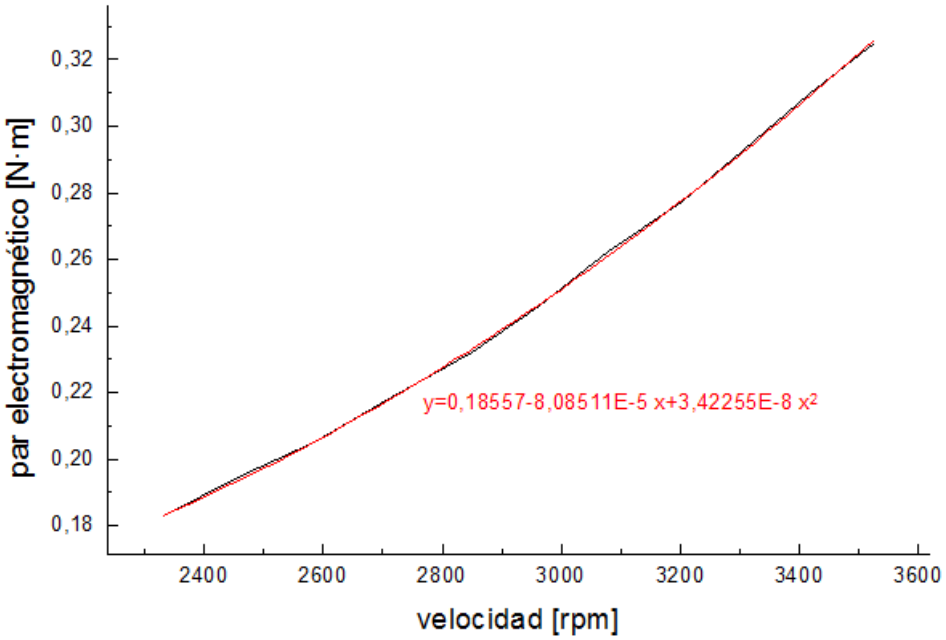


Figura 5.6. Relación par-velocidad del accionamiento con el motor de continua.

6 Modelado mediante Ansoft-Simulink

Se ha desarrollado un procedimiento de caracterización y simulación de motores de corriente continua sin escobillas. Concretamente, se usan dos herramientas software: RMXprt de Ansys, útil para definir los parámetros de la mayoría de configuraciones de máquinas rotativas, y Simulink de Matlab. Mediante RMXprt se han extraído las características funcionales del motor, para ser posteriormente introducidas en el modelo de bloques de Simulink. Más adelante se describe el procedimiento seguido para el motor de refrigeración estudiado.

6.1 Extracción de características

La extracción de características del motor real es un proceso iterativo. El software de *Ansoft* específico para máquinas rotativas es *RMxprt*, y éste permite introducir infinidad de datos del motor, materiales, dimensiones, distribución de imanes y devanados, ... El inconveniente es que la hoja de carcaterísticas del motor no facilita todos los datos necesarios para realizar un modelo ajustado al motor real.

El método seguiido ha sido introducir los datos disponibles, e ir ajustando el resto de forma que las características del motor (par, eficiencia y consumo respecto velocidad) se ajusten a las descritas por el fabricante.

6.2 Paso de información a Simulink

El modelado del motor se ha realizado mediante la aplicación *RMxprt* de *Ansoft*. Sin embargo, este programa no permite realizar simulaciones con carga. Para ello se ha decidido usar el entorno *Matlab Simulink*, ya que es una herramienta muy visual y que permite introducir gran cantidad de datos en forma de tabla.

El procedimiento de paso de información se realiza mediante la ejecución de una serie de instrucciones. Para organizarlas mejor se han creado los cinco ficheros de código siguientes:

- corrents.m: extrae la forma de la corriente que circula por cada devanado en función de la posición. También obtiene la tabla de eficiencia. Todos los datos se importan automáticamente a partir de un fichero *.csv.
- motor.m: define las constantes del motor que vienen determinadas por el diseño. Son introducidas manualmente.
- PI.m: constantes del control, se sintonizan de forma manual.
- trapez.m: crea una forma de fuerza contraelectromotriz a partir de los parámetros dados, que se extraen de forma manual del software *RMxprt*.
- inici.m: llama las funciones anteriores antes de empezar a simular el comportamiento del motor.

Los códigos implementados para cumplir las funciones descritas han sido incluidos en el Anexo II: Transferencia de datos del modelo.

6.3 Modelado matemático

Para realizar el modelo del motor *brushless* la solución tradicional consiste en partir del modelo matemático. A la hora de obtener la expresión matemática que describe el funcionamiento del motor se han ignorado las corrientes inducidas al rotor, ya que el acero y los imanes permanentes presentan una alta resistividad.

De esta forma, las ecuaciones de los tres devanados son

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & R \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} L_a & L_{ba} & L_{ca} \\ L_{ba} & L_b & L_{cb} \\ L_{ca} & L_{cb} & L_c \end{bmatrix} \begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \quad (6)$$

En caso que la reluctancia del rotor no se vea afectada por su posición, se pueden realizar las simplificaciones

$$\begin{aligned} L_a &= L_b = L_c = L \\ L_{ab} &= L_{ca} = L_{cb} = M \end{aligned} \quad (7)$$

Por lo tanto:

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & R \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} L & M & M \\ M & L & M \\ M & M & L \end{bmatrix} \begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \quad (8)$$

Las tres fases del estator están conectadas en estrella, por lo que se sabe que:

$$i_a + i_b + i_c = 0 \quad (9)$$

entonces se puede afirmar que

$$M \cdot i_b + M \cdot i_c = -M \cdot i_a \quad (10)$$

Sustituyendo (10) en (8):

$$\begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} = \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & R \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} L-M & 0 & 0 \\ 0 & L-M & 0 \\ 0 & 0 & L-M \end{bmatrix} \begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} + \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \quad (11)$$

En forma de ecuación de variables de estado

$$\begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} = \begin{bmatrix} \frac{1}{L-M} & 0 & 0 \\ 0 & \frac{1}{L-M} & 0 \\ 0 & 0 & \frac{1}{L-M} \end{bmatrix} \cdot \left\{ \begin{bmatrix} v_a \\ v_b \\ v_c \end{bmatrix} - \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & R \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} - \begin{bmatrix} e_a \\ e_b \\ e_c \end{bmatrix} \right\} \quad (12)$$

Entonces:

$$\begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} = \begin{bmatrix} \frac{1}{L-M}\left(v_a - R \cdot i_a - e_a\right) \\ \frac{1}{L-M}\left(v_b - R \cdot i_b - e_b\right) \\ \frac{1}{L-M}\left(v_c - R \cdot i_c - e_c\right) \end{bmatrix} \quad (13)$$

$$\begin{bmatrix} \dot{i}_a \\ \dot{i}_b \\ \dot{i}_c \end{bmatrix} = \begin{bmatrix} \frac{-R}{L-M} & 0 & 0 \\ 0 & \frac{-R}{L-M} & 0 \\ 0 & 0 & \frac{-R}{L-M} \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix} + \begin{bmatrix} \frac{1}{L-M}\left(v_a - e_a\right) \\ \frac{1}{L-M}\left(v_b - e_b\right) \\ \frac{1}{L-M}\left(v_c - e_c\right) \end{bmatrix} \quad (14)$$

Por otra parte, la ecuación del movimiento es

$$\frac{d\omega_r}{dt} = \frac{1}{J}\left(T_e - T_L - B \cdot \omega_r\right) \quad (15)$$

Y el par electromagnético producido se define como

$$T_e = \frac{e_a \cdot i_a + e_b \cdot i_b + e_c \cdot i_c}{\omega_r} \quad (16)$$

Esto significa que para determinar el par producido es necesario conocer tanto la corriente que circula por cada una de las fases como la fuerza contraelectromotriz que se induce en cada una de ellas. Por lo tanto, se conoce la BEMF (*Back ElectroMotive Force* o fuerza contra electro motriz) mediante la ecuación (14) partiendo de unos valores conocidos de tensión y corriente aplicados en cada fase.

6.4 Modelado mediante *simulink*

La tensión aplicada a la fase va a ser conocida por el usuario, lo que falta por conocer, pues, es la corriente. Para ello se usan los resultados del software *Maxwell* de *Ansoft*, concretamente los que relacionan las corrientes de cada fase con la posición en la que se encuentra el motor. Para ello, la simulación debe realizarse con el mismo valor de tensión de fase que se utilizará posteriormente.

Para la fase 'a':

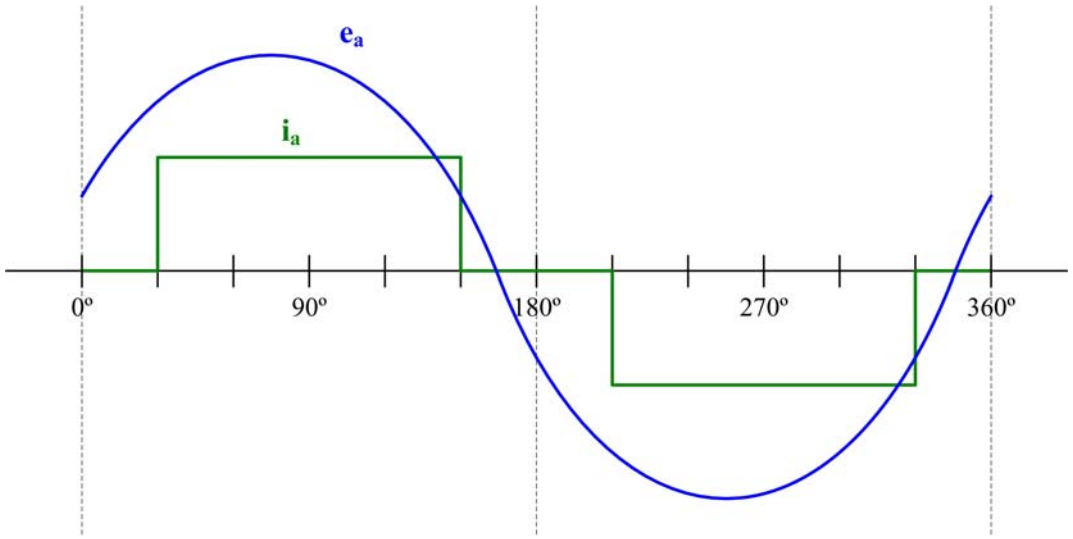


Figura 6.1. Corriente y fuerza contraelectromotriz en una fase.

Las fases 'b' y 'c' estarán desfasadas 120 y 240º, respectivamente. La forma de la corriente evita pulsaciones de par, aunque causa un claro inconveniente de cara a la realización del modelo: la BEMF no será puramente senoidal.

Conviene obtener 'ea', 'eb' y 'ec'. De acuerdo con el modelo matemático, para ello es necesario conocer 'L' y 'M'. El programa de análisis nos da estos datos respecto los ejes de referencia d y q. Por lo tanto, el modelo debería realizarse en el mismo sistema de referencia. La expresión resultante es:

$$\begin{bmatrix} v_d \\ v_q \\ v_o \end{bmatrix} - \begin{bmatrix} e_d \\ e_q \\ e_o \end{bmatrix} = \begin{bmatrix} R_1 + \frac{dL_d}{dt} & -L_q \cdot \omega_e & 0 \\ -L_d \cdot \omega_e & R_1 + \frac{dL_q}{dt} & 0 \\ 0 & 0 & R_1 + \frac{dL_o}{dt} \end{bmatrix} \qquad (17)$$

Sin embargo, el software de análisis de elementos finitos muestra la forma de la fuerza contraelectromotriz inducida en una fase para una velocidad determinada.

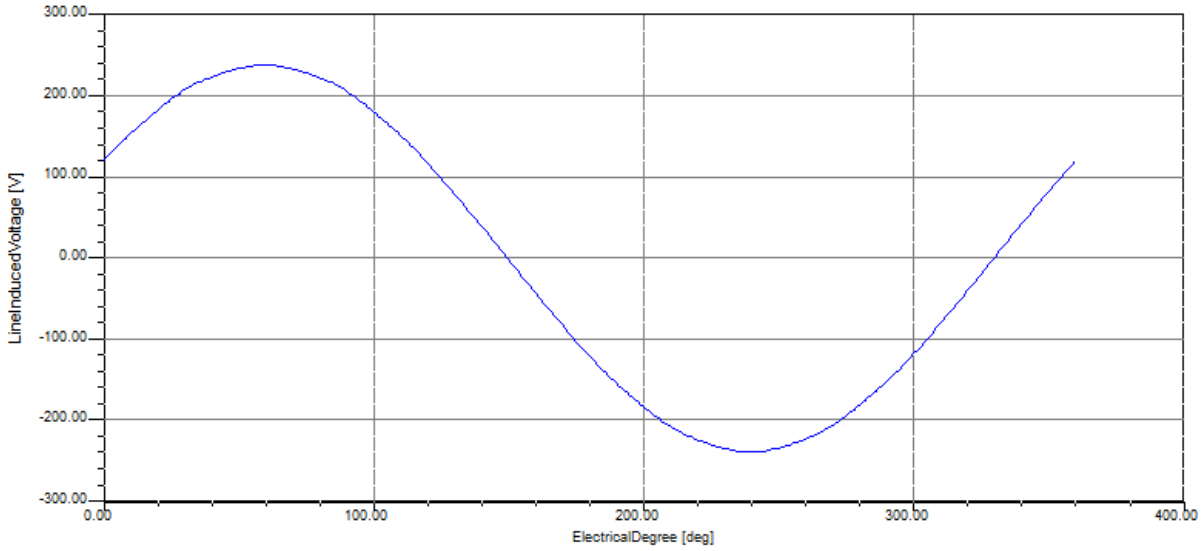


Figura 6.2. Forma de la fuerza contraelectromotriz.

Se conoce que el valor de la amplitud de la BEMF responde a la siguiente expresión:

$$\Delta BEMF = K_e \cdot \omega_e \qquad (18)$$

Por simplicidad, se obtendrá la fuerza contraelectromotriz de las fases de la forma obtenida mediante el software RMxprt. El primer paso para conseguirlo es normalizar la forma obtenida. Posteriormente se determina su amplitud multiplicando la señal resultante por la constante eléctrica y por la velocidad del motor.

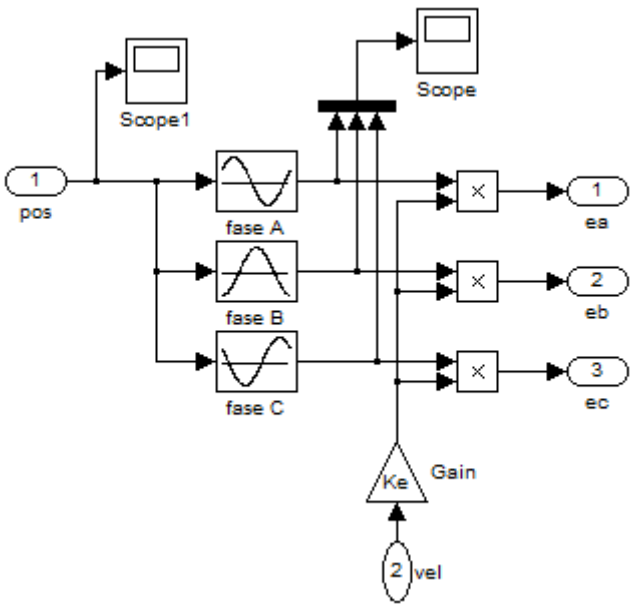


Figura 6.3. Cálculo de las BEMF.

6.4.1 Descripción de los bloques

El modelo se agrupa, por simplicidad, en un único bloque, mostrado a continuación:

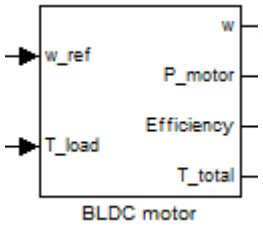


Figura 6.4. Bloque del motor completo.

El bloque anterior se compone de los elementos mostrados a continuación:

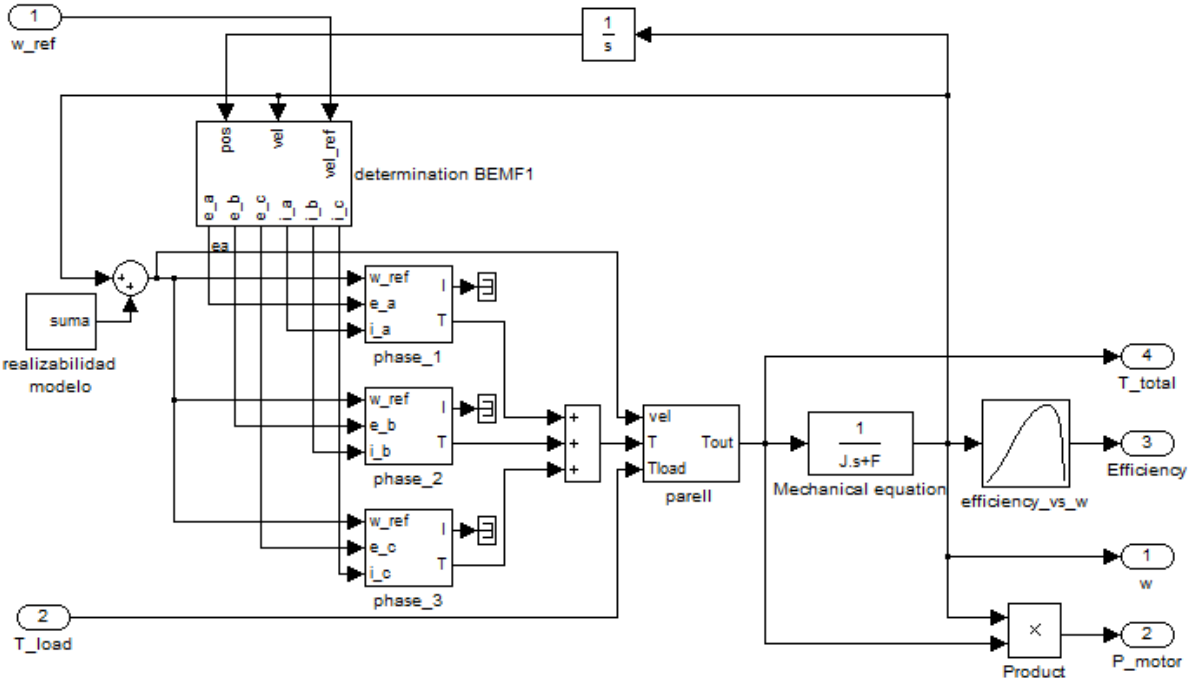


Figura 6.5. Bloques que conforman el modelo.

En la figura anterior se aprecian las tres fases de las que el motor dispone. Cada una de ellas calcula el par realizado a partir de la velocidad, la corriente en cada fase y su fuerza contraelectromotriz inducida, según (16). Los pares producidos por cada una de las fases se suman, logrando así el par desarrollado por el motor.

El problema recae en el momento de restar el par de carga al producido. Si en todos los supuestos se produce esta resta, en el momento del arranque el par es negativo. Si no se añade ninguna limitación, esto provocará a su vez una velocidad negativa y, al seguir ofreciendo par negativo, la velocidad todavía se hará más negativa.

Para sortear este problema se ha adoptado la solución propuesta en la figura anterior. El interior del bloque *parell* se muestra a continuación.

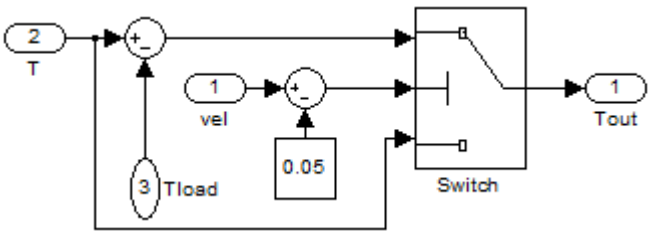


Figura 6.6. Interior del bloque *parell.*

Lo que se consigue añadiendo este bloque es que para velocidades muy bajas (0,05 rad/s e inferiores) el par aplicado al motor no pueda ser negativo.

Además, se ha añadido otra etapa de saturación para limitar el par de arranque de acuerdo con el valor obtenido en el modelado del motor mediante *Rmxprt*.

Tras el ajuste del valor del par total, éste se aplica a la ecuación mecánica del motor. La inercia del motor y la constante de fricción se importan de los parámetros de *Ansoft*. El resultado es la velocidad angular del motor. La posición se obtiene integrando la velocidad, y con esta información se puede determinar la fuerza contraelectromotriz para cada una de las fases. Este proceso se realiza en el bloque de la figura 6.7.

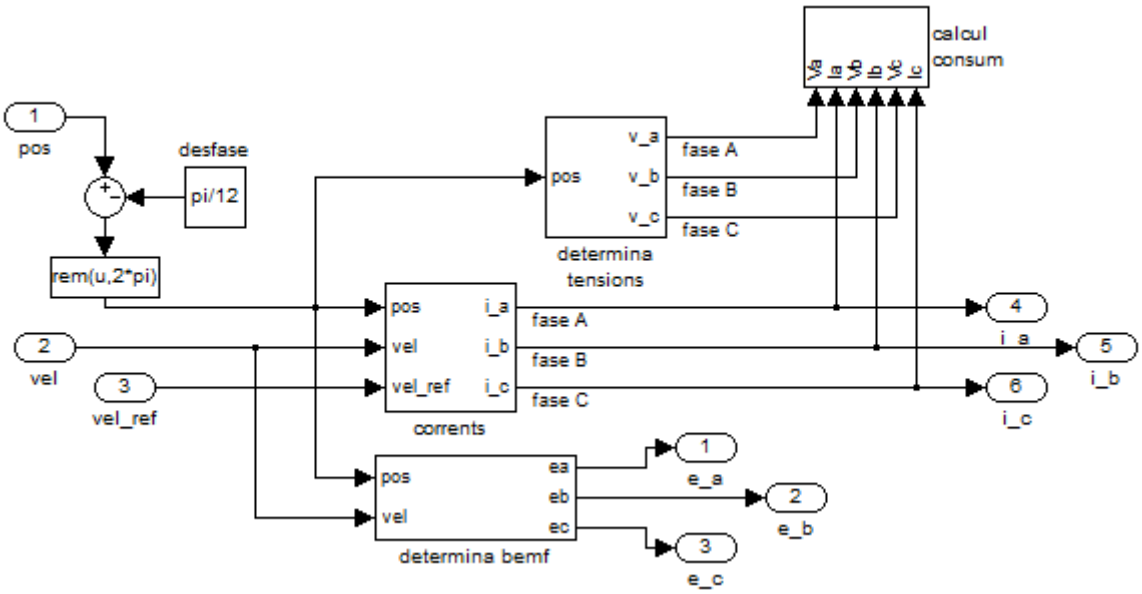


Figura 6.7. Interior del bloque *determination BEMF*.

La posición absoluta se traduce a un ángulo de entre 0 y 2π rad. Previamente, pero, se le puede aplicar un desfase, cuyos efectos se procederá a analizar más adelante. A partir de la posición se determina la tensión a aplicar, la corriente que este hecho genera y la fuerza contraelectromotriz que se induce en cada uno de los devanados. El bloque *calcul*

consum realiza el cálculo instantáneo de consumo de potencia, y después de la simulación permite hacer un cálculo promediado.

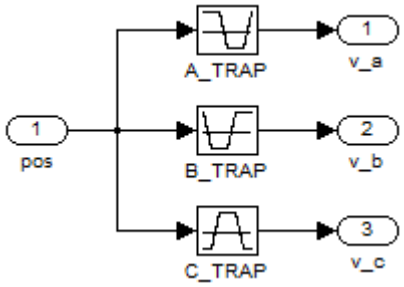


Figura 6.8. Interior del bloque *determina tensions.*

De forma prácticamente idéntica, se determinan las corrientes aplicadas a cada una de las fases mediante *corrents*, aunque en este caso la amplitud de las corrientes es modulada por un control PI, que consigue que el motor siga la velocidad de referencia.

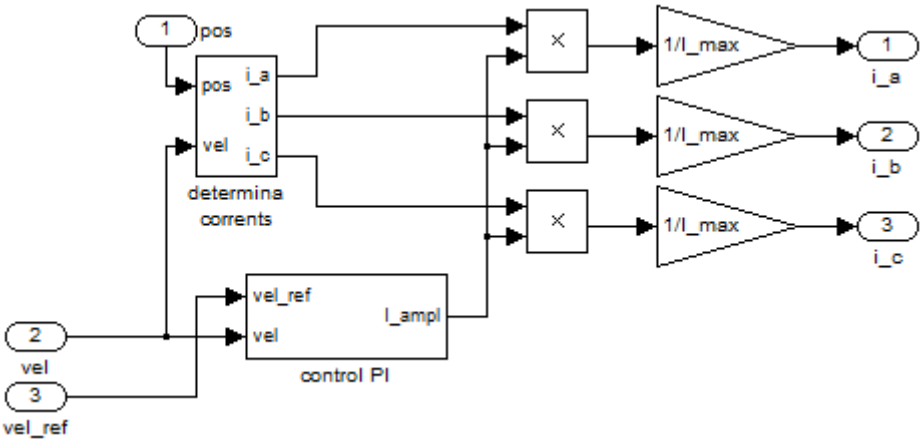


Figura 6.9. Interior del bloque *corrents.*

6.5 Simulación del accionamiento

El accionamiento simulando es el conjunto de motor y control, con un modelo aproximado del ventilador, pues no se disponían de los datos suficientes para modelarlo correctamente. Se ha sometido el motor a una rampa de velocidad hasta los 200 rad/s (1920 rpm), y se ha mantenido durante un periodo de tiempo a esta velocidad.

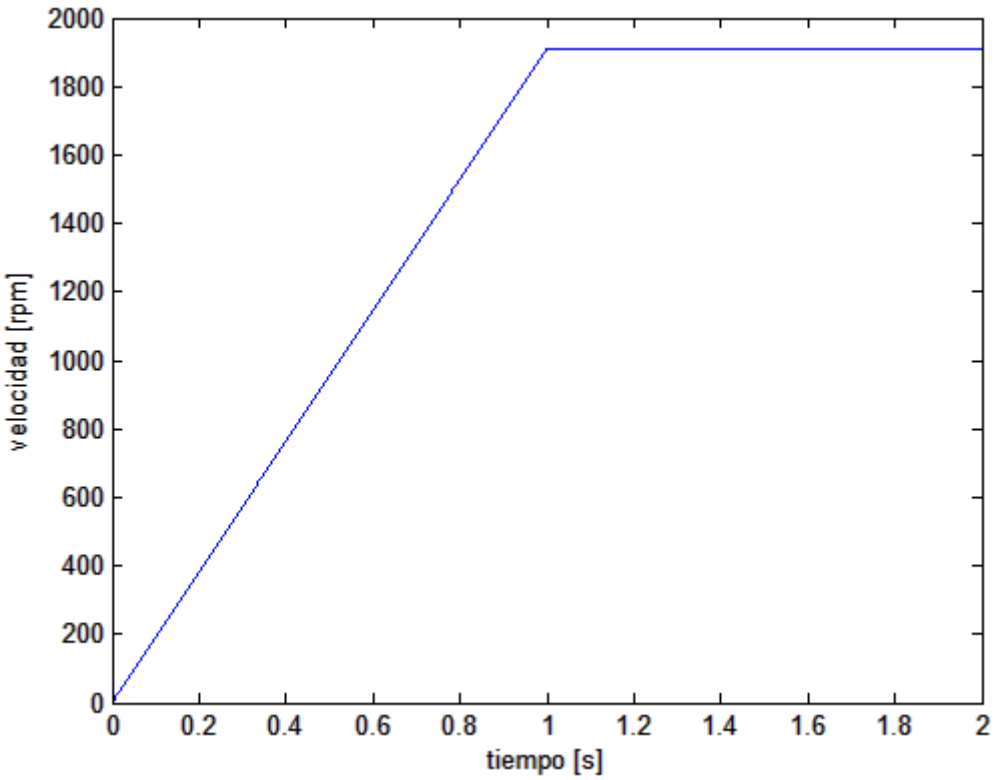


Figura 6.10. Velocidad del motor durante la simulación.

La forma de las corrientes para cada una de las fases es la que se muestra a continuación.

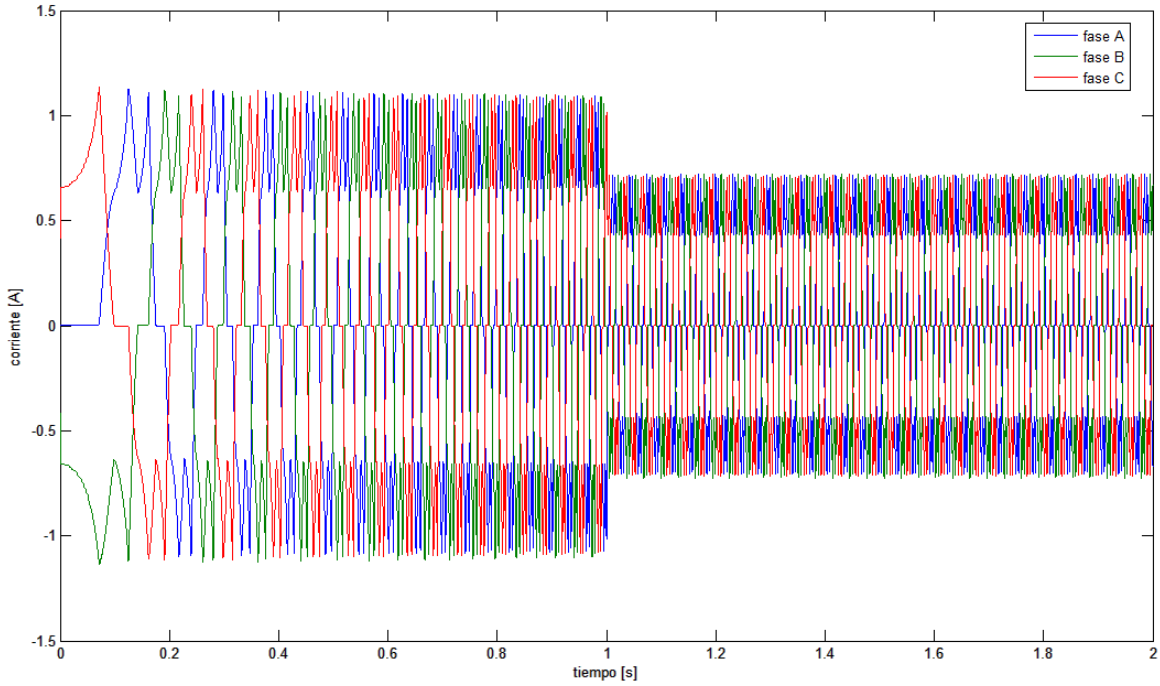


Figura 6.11. Corrientes para cada una de las fases.

Finalmente, el consumo del motor para la situación descrita se representa en la figura 6.12.

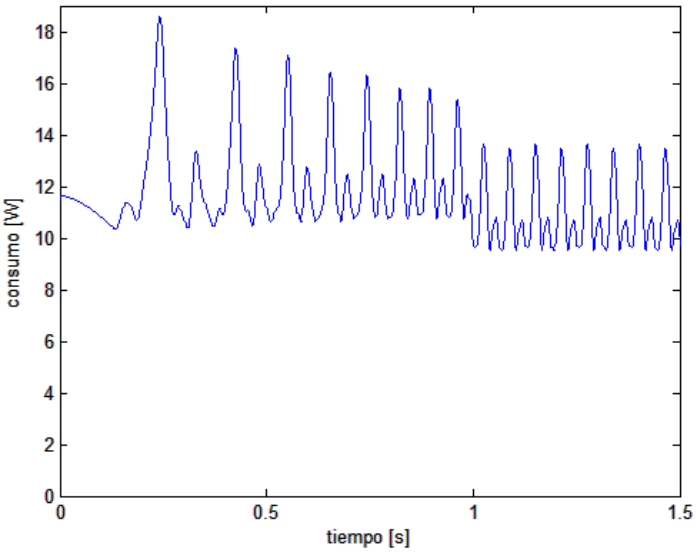


Figura 6.12. Consumo del motor.

Cuando el motor gira a una velocidad de 480 rpm, el consumo medio, de acuerdo con la figura 6.12, es de unos 12 W. Según la hoja de características del motor *brushless* utilizado, para la velocidad a la que se ha realizado el ensayo el consumo es ligeramente superior a los 10 W. Por lo tanto, se comprueba la utilidad del modelo para representar el accionamiento real.

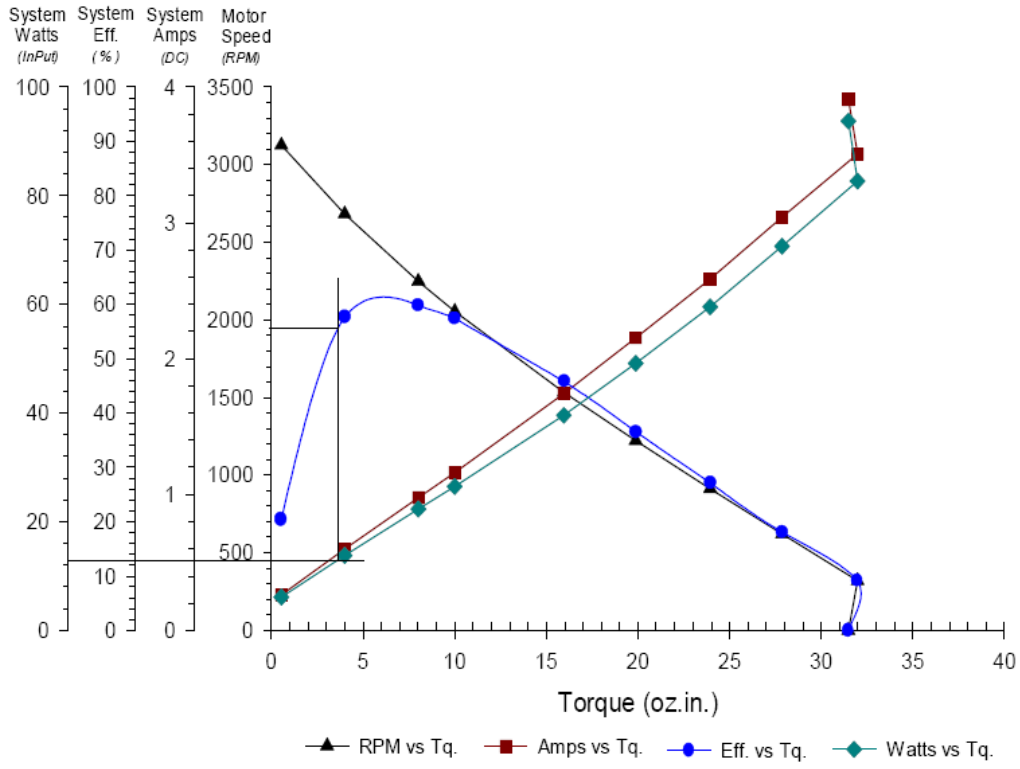


Figura 6.13. Consumo a 1920 rpm según la hoja de características del motor.

7 Control *sensorless*

7.1 Introducción

El principal inconveniente de los motores brushless es, como ya se ha mencionado, su coste, debido a la presencia de imanes permanentes. La adición de encoders o sensores de efecto Hall representa un gasto que encarece todavía más el precio final del producto. Así, el consumo masivo de este tipo de motores pasa por una reducción de costes, empezando por la supresión de los sensores de posición.

7.2 Ventajas e inconvenientes

El motor *brushless DC* es un tipo de motor en que es necesario sincronizar la conmutación de las fases con la posición del rotor. Existen varios motivos por los que resulta deseable evitar el uso de sensores en estos tipos de motores:

- **Coste**: la inclusión de sensores en un accionamiento supone un incremento del precio del conjunto. El cableado de los sensores añade otro coste más.
- **Fiabilidad**: los sensores sufren desgaste y averías. Un motor con sensores necesita mayor mantenimiento, la cual cosa acaba repercutiendo también sobre el coste.
- **Montaje**: a parte de añadir dificultad al montaje del motor, los sensores aumentan su volumen y peso. Puede darse el caso que un motor sea inviable para una determinada aplicación debido a uno de estos factores.
- **Inmersión**: si el motor está inmerso en un fluido, prescindir de los sensores es la opción más recomendable, ya que la probabilidad de que el sensor falle aumenta considerablemente.

Sin embargo, existen escenarios en los cuales el uso de esta técnica de posicionamiento del rotor está desaconsejado:

- **Bajas velocidades**: la amplitud de la fuerza contraelectromotriz es directamente proporcional a la velocidad del rotor. Por lo tanto, la detección del cruce por 0 V será más complicada cuanto más baja sea la velocidad ya que la pendiente de la señal será inferior. Esta detección se complica todavía más debido al ruido acoplado de la alimentación PWM de las otras fases.
- **Cambios de carga rápidos**: en el evento de una rápida variación de la carga puede ocurrir que las estimaciones difieran notablemente de la posición real, dando lugar a una mala conmutación.

En el caso del accionamiento para el ventilador de refrigeración de un vehículo, el control sin sensores es muy interesante para evitar posibles averías de los sensores. Es más, en un entorno tan competitivo como es el mundo del automóvil, la reducción del coste de esta pieza supone una gran ventaja competitiva que debe ser tomada en cuenta.

7.3 Recopilación de técnicas de control sin sensores

Se pueden clasificar las técnicas de control sin sensores existentes en cuatro categorías [15], en función de la información y las manipulaciones matemáticas usadas para estimar la posición:

- las que usan las ecuaciones y mediciones,
- las que usan observadores.
- las que usan la fuerza contraelectromotriz inducida
- las no incluidas en las tres categorías anteriores.

7.3.1 *Técnicas que usan ecuaciones y mediciones*

Dentro de esta categoría se encuentran, básicamente, tres métodos distintos:

- **Los que miden tensión y corriente.** Con la información de tensión y corriente se puede calcular el acoplamiento inductivo (ψ) a partir de la ecuación (19). Entonces, para estimar la posición del rotor es necesario conocer su posición inicial, algunos parámetros del motor y la relación entre el acoplamiento inductivo y la posición.

$$v = R \cdot i + \frac{d}{dt}\psi \qquad (19)$$

- **Los que usan un modelo para predecir la posición.** Se usa un modelo de la máquina en los ejes de referencia d-q, que trabajará en la posición inmediatamente anterior del motor, ya conocida. Se comparan las salidas de tensión del modelo y del motor real, y se extrae la posición que se ha recorrido.
- **Los que usan parámetros, ecuaciones y manipulaciones algebraicas.** En este caso, se usa la transformación del motor a un sistema de referencia vectorial para calcular la posición y la velocidad.

7.3.2 Técnicas que usan observadores

El observador utiliza como entradas tanto las entradas al sistema en cuestión como las salidas. De esta forma es capaz de estimar variables no accesibles, como es el caso de la posición en la aplicación estudiada.

7.3.3 Técnicas que usan la fuerza contraelectromotriz inducida

El sensado de la fuerza contraelectromotriz inducida proporciona una información que se puede interpretar de varias formas para extraer la posición:

- **Sensado de la tensión de fase.** Muestreando la fuerza contraelectromotriz de la fase por donde no se está conduciendo corriente y comparando su valor con la mitad de la tensión del bus de continua se detectan una serie de puntos. La conmutación de las fases se realizará de forma relativa al instante en el que se produce el cruce.

- **Sensado del tercer armónico de la fuerza contraelectromotriz.** En un motor trapezoidal conectado en estrella, el resultado de la suma de las tres tensiones de fase está dominado por el tercer armónico. Tras procesar este armónico puede estimarse la posición del rotor.
- **Sensado de la corriente de los diodos volantes.** En la conmutación de seis pasos siempre hay una fase que se encuentra en circuito abierto. Después de dejar la fase en circuito abierto, la corriente de la fase circula durante un tiempo por el diodo volante. Esta corriente llega a cero en la mitad del intervalo de conmutación, es decir, en el mismo instante en el que se detecta un cruce mediante el método de sensado de la tensión de fase.
- **Integración de la fuerza contraelectromotriz.** Se integra la fuerza contraelectromotriz de la fase sin alimentar, y cuando el resultado llega a un determinado valor es cuando debe producirse la siguiente conmutación.

7.4 Principio de funcionamiento

Para realizar la conmutación de seis pasos, explicada de forma detallada en el capítulo 4, es necesario conocer la posición en la que se encuentra el rotor. En este caso, solamente se requiere distinguir entre seis posiciones distintas (de 0 a 60º, de 60 a 120º, y así sucesivamente hasta los 360º). Por lo tanto, para operar un motor *brushless dc* con esta estrategia no hace falta un encoder, basta con tres sensores de efecto Hall.

Sin embargo, es posible prescindir de ellos, ahorrando así mantenimiento y coste del motor. La técnica más extendida por su simplicidad y parecido con la conmutación mediante sensores de efecto Hall es el filtraje de las fuerzas contraelectromotrices inducidas (BEMF) [4]. Básicamente, la forma de las BEMF para cada una de las fases tiene la siguiente apariencia

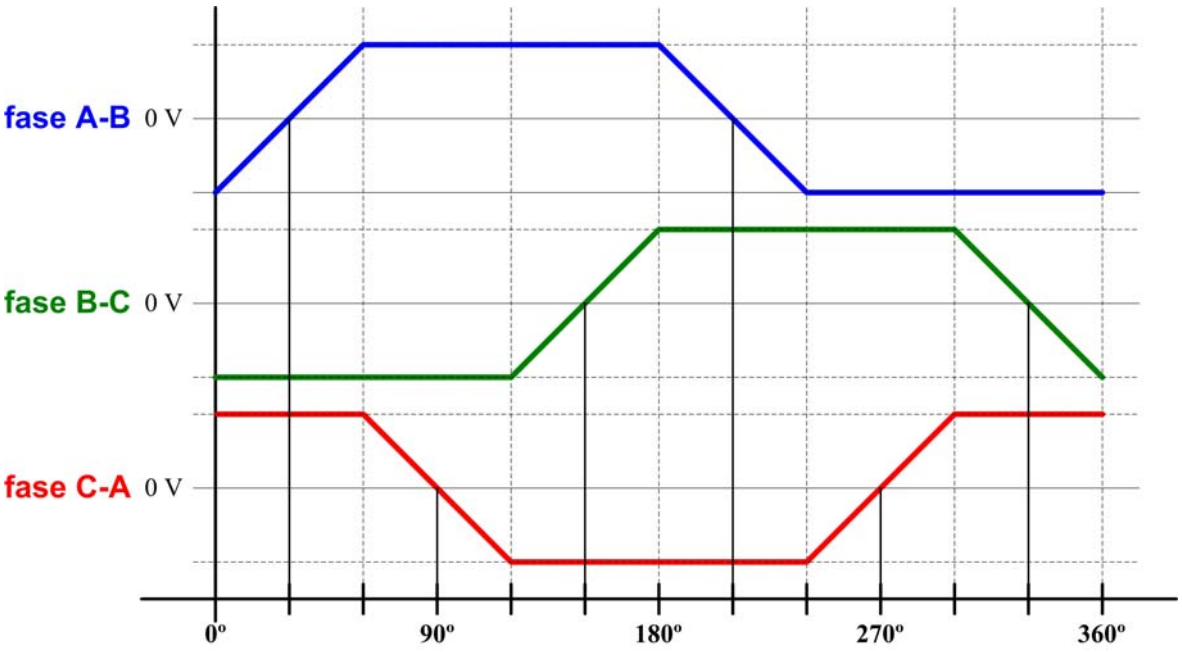


Figura 7.1. Forma de las BEMF.

Tal como se aprecia en la figura, las señales de las fuerzas contraelectromotrices son simétricas respecto los 0 V. Este hecho supone una gran ventaja porque permite conocer con seguridad los cruces con la tensión de 0V.

Se producen seis pasos por 0 V de las BEMF: para 30, 90, 150, 210, 270 y 330º (eléctricos). Estos cruces definen seis sectores diferentes, que pese a no coincidir con los detectados por los sensores de efecto Hall, pueden servir para desempeñar la misma función. La conmutación debe producirse a 60, 120, 180, 240, 300 y 360º, es decir, 30º después de que haya un cruce por 0 V de la fase no alimentada. Básicamente, pues, el algoritmo de conmutación sin sensores consiste en detectar con exactitud el punto de cruce por cero, estimar el tiempo que tardará el rotor en recorrer 30º y realizar la acción de conmutación que corresponda a ese instante.

El ruido PWM complica la detección del evento de cruce por 0 V, por lo que será necesario realizar un filtrado de la fuerza contraelectromotriz. Ya que se desea realizar el control del motor *brushless DC* mediante el microprocesador dsPIC30F6010A, éste también se utilizará para filtrar las BEMF.

7.4.1 Muestreo y retención de las BEMF

Las tensiones en cada fase del motor están comprendidas entre los 0 y los 24 V. Estas señales deben ser sensadas contínuamente para deducir la posición del rotor y realizar las pertinentes conmutaciones. Sin embargo, el conversor ananógico digital de 10 bits presente en el dsPIC30f6010A tiene la característica mostrada en la figura 7.2 [5]. Las tensiones de referencia son 0 y 5 V, y el rango comprendido entre ellas será el que corresponde al tramo lineal de la función de transferencia.

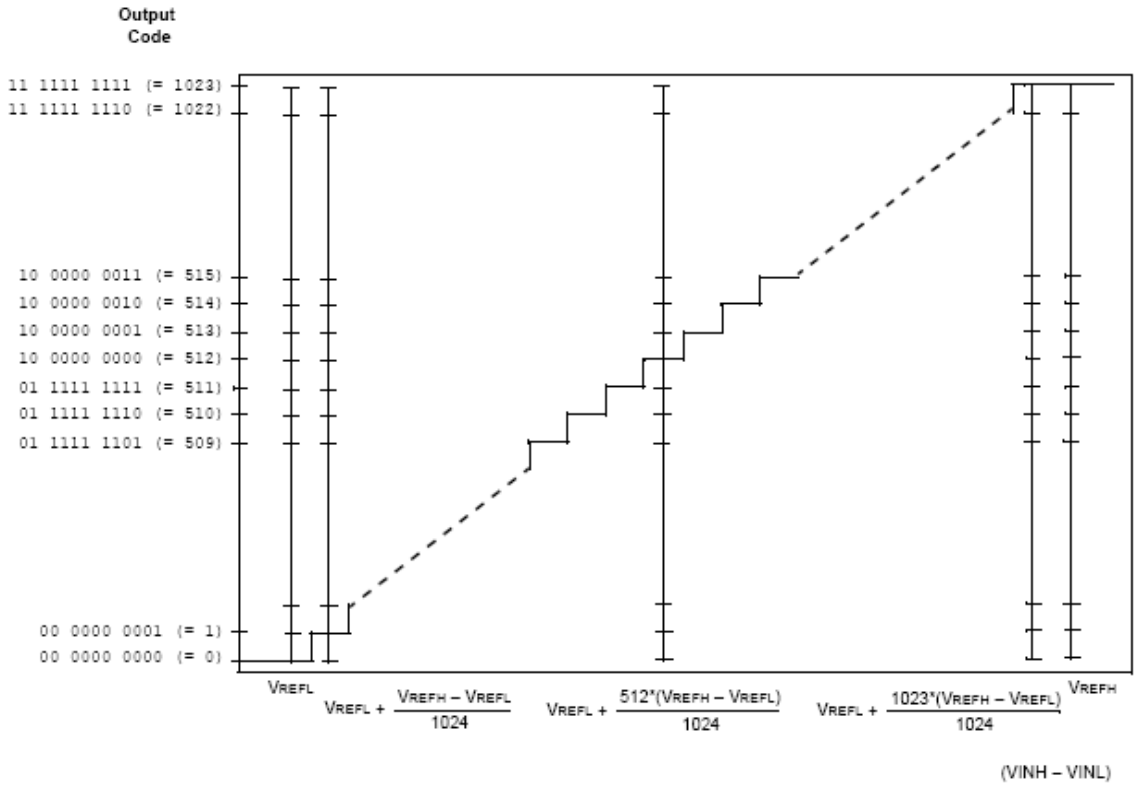


Figura 7.2. Función de transferencia del conversor analógico-digital.

Resulta necesario, por lo tanto, realizar un escalado de las tensiones a sensar. Ya que la impedancia de entrada de los pines analógicos es muy elevada, la solución más sencilla e indicada es un divisor de tensión

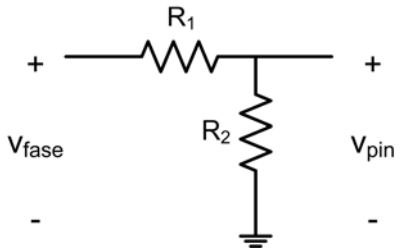


Figura 7.3. Divisor de tensión para el escalado de las tensiones a muestrear.

La tensión que será introducida en el pin analógico del microprocesador será

$$v_{pin} = \frac{R_2}{R_1 + R_2} v_{fase} \qquad (20)$$

Las tensiones en las fases pueden llegar a los 24 V, que es la tensión introducida mediante la fuente de continua. Por si se superan ligeramente estos límites, se ha decidido que el código máximo de salida (0x3FF) corresponda a los 25 V en la fase, y con los 5 V en el pin analógico. Por consiguiente:

$$\frac{R_2}{R_1 + R_2} = \frac{1}{5} \qquad (21)$$

Además, el margen de tensiones de entrada para las cuales el fabricante del microprocesador garantiza su seguridad son de -0,3 a 5,3V. Se han observado picos de hasta 35 V al conectar y desconectar la fuente, por lo que es preciso añadir algún tipo de limitación al divisor propuesto anteriormente.

La solución adoptada consiste en utilizar un zener para recortar las tensiones de salida del divisor superiores a 5,1 V mediante un diodo zener. Para las tensiones negativas se ha usado un diodo Schottky por tener una tensión inversa muy baja, en el caso del componente elegido de 0,8 V.

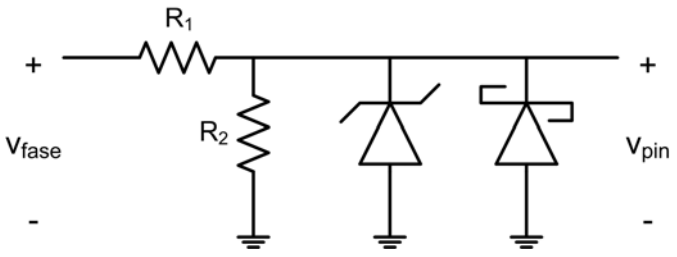


Figura 7.4. Divisor con limitación de tensión positiva y negativa.

En el capítulo 14 se presenta el esquema del circuito propuesto, así como la PCB y imágenes de la placa ya montada.

Se ha realizado un ensayo para comprobar el correcto funcionamiento del circuito y evaluar sus características.

vfase [V]	**vfase/5 [V]**	**vpin [V]**
-15,00	-3	-0,305
-2,50	-0,5	-0,215
-2,00	-0,4	-0,205
-1,50	-0,3	-0,19
-1,00	-0,2	-0,162
-0,50	-0,1	-0,097
0,00	0	0
2,50	0,5	0,498
5,00	1	0,997
7,50	1,5	1,497
10,00	2	1,998
12,50	2,5	2,499
15,00	3	3,002
17,50	3,5	3,502
20,00	4	3,999
22,50	4,5	4,464
25,00	5	4,832
25,50	5,1	4,884
26,00	5,2	4,931
26,50	5,3	4,972
27,00	5,4	5,006
27,50	5,5	5,037
40,00	8	5,272

Tabla 7.1. Resultados del ensayo del circuito de adaptación de señales.

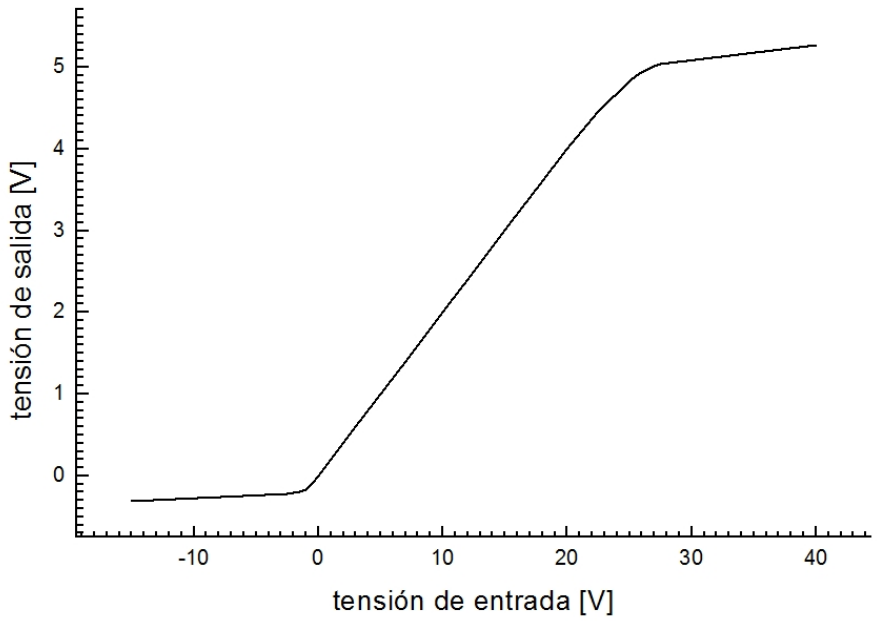


Figura 7.5. Función de transferencia del circuito de protección de las entradas analógicas.

Una vez implementado el circuito descrito, debe procederse a configurar el módulo de conversión analógico-digital del microprocesador usado. La frecuencia máxima a la que el dsPIC30f6010A puede trabajar es de 120 MHz.

$$f = f_{osc} \cdot PLLx \tag{22}$$

El reloj del que dispone la placa de control dsPICDEM MC1 es de 7,3728 MHz. Por lo tanto,

$$f = 7{,}3728\ \text{MHz} \cdot 16 = 117{,}96\ \text{MHz} \tag{23}$$

Teniendo en cuenta que son necesarios 4 ciclos para ejecutar una instrucción

$$\text{MIPS} = \frac{117.960.000\ \text{ciclos}}{\text{s}} \cdot \frac{1\ \text{instrucción}}{4\ \text{ciclos}} = 29.491.000\ \text{instrucciones/s} \tag{24}$$

y entonces el tiempo de ciclo de instrucción es

$$T_{cy} = \frac{1}{\text{MIPS}} = \frac{4}{F_{osc} \cdot PLLx} = 33{,}9\ \text{ns} \tag{25}$$

El periodo de muestreo se compone de dos tiempos distintos: el tiempo de muestreo (T_{samp}) y el tiempo de conversión (T_{conv}). Ambos toman como referencia el periodo del reloj del conversor (T_{AD}), tal como se muestra en las siguientes expresiones.

$$T_{AD} = m \cdot T_{cy} \tag{26}$$

$$T_{samp} = n \cdot T_{AD} \tag{27}$$

$$T_{conv} = 12 \cdot T_{AD} \tag{28}$$

Los valores del periodo del reloj del conversor y el periodo de muestreo se configuran en los registros del módulo A/D mediante los conjuntos de bits ADCS y SAMC, respectivamente. De esta forma el periodo de muestreo es

$$T_m = c \cdot \left(T_{samp} + T_{conv}\right) = c \cdot \left(n + 12\right) \cdot T_{AD} \qquad (29)$$

donde c es el número de canales a muestrear. En este caso se desea muestrear las tres tensiones de fase y el bus de continua, por lo que se hacen necesarios cuatro canales

$$T_m = c \cdot \left(T_{samp} + T_{conv}\right) = 4 \cdot \left(n + 12\right) \cdot T_{AD} \qquad (30)$$

La frecuencia de muestreo debe ser, como mínimo, el doble de la frecuencia del PWM. Se ha establecido que sea de unos 50 kHz, aproximadamente.

Iteración 1

El periodo del reloj del conversor analógico-digital debe ser, como mínimo, de 83,3 ns. Por lo tanto se debe configurar correctamente el valor de ADCS:

$$ADCS = \frac{2 \cdot T_{AD}}{T_{cy}} - 1 = \frac{2 \cdot 83{,}3 \cdot 10^{-9}}{33{,}9 \cdot 10^{-9}} - 1 = 3{,}91 \qquad (31)$$

Redondeando por exceso, se elige ADCS=4. Finalmente, pues, el periodo del reloj del conversor es de

$$T_{AD} = \frac{T_{cy} \cdot \left(ADCS + 1\right)}{2} = \frac{33{,}9 \cdot 10^{-9} \cdot 5}{2} = 84{,}8 \text{ ns} \qquad (32)$$

De acuerdo con la frecuencia de muestreo deseada

$$f_m = \frac{1}{T_m} = \frac{1}{4 \cdot \left(n + 12\right) \cdot T_{AD}} = 50.000 \text{ Hz} \qquad (33)$$

$$n = \frac{1}{4 \cdot T_{AD} \cdot f_m} - 12 = 46{,}98 \qquad (34)$$

Por lo tanto n, que se especifica mediante SAMC, debe ser 47. Sin embargo, el máximo valor configurable es 31, por lo que se debe aumentar el periodo del reloj de conversión y volver a realizar los cálculos.

Iteración 2

Se elige ADCS=5, y en consecuencia

$$T_{AD} = \frac{T_{cy} \cdot \left(ADCS + 1\right)}{2} = \frac{33{,}9 \cdot 10^{-9} \cdot 6}{2} = 102 \text{ ns} \qquad (35)$$

El nuevo valor de SAMC debería ser de

$$n = \frac{1}{4 \cdot T_{AD} \cdot f_m} - 12 = 37{,}15 \qquad (36)$$

aunque continua siendo demasiado elevado.

Iteración 3

Eligiendo ADCS=6 se obtiene un periodo del reloj del conversor de

$$T_{AD} = \frac{T_{cy} \cdot (ADCS + 1)}{2} = \frac{33{,}9 \cdot 10^{-9} \cdot 7}{2} = 119 \text{ ns} \qquad (37)$$

Para una frecuencia de muestreo de aproximadamente 50 KHz, el valor de SAMC es de

$$n = \frac{1}{4 \cdot T_{AD} \cdot f_m} - 12 = 30{,}13 \qquad (38)$$

Este valor es válido. Finalmente, configurando ADCS=6 y SAMC=30 se consigue una frecuencia de muestreo de

$$f_m = \frac{1}{4 \cdot (30 + 12) \cdot 119 \text{ ns}} = 50.155 \text{ Hz} \qquad (39)$$

7.4.2 Filtrado de las BEMF

Aunque para simplificar se está hablando continuamente de sensado y filtrado de la fuerza contraelectromotriz, en realidad se muestrea toda la tensión de fase. Tomando como referencia la fase A-B de la figura 7.1, por ejemplo, se observa que durante 120º eléctricos la fase se alimenta con una tensión positiva y durante otros 120º con una tensión negativa. Los 120º eléctricos restantes se dividen en dos sectores de 60º, durante los cuales la fase no es alimentada por el módulo PWM, y se induce en ella una fuerza contraelectromotriz.

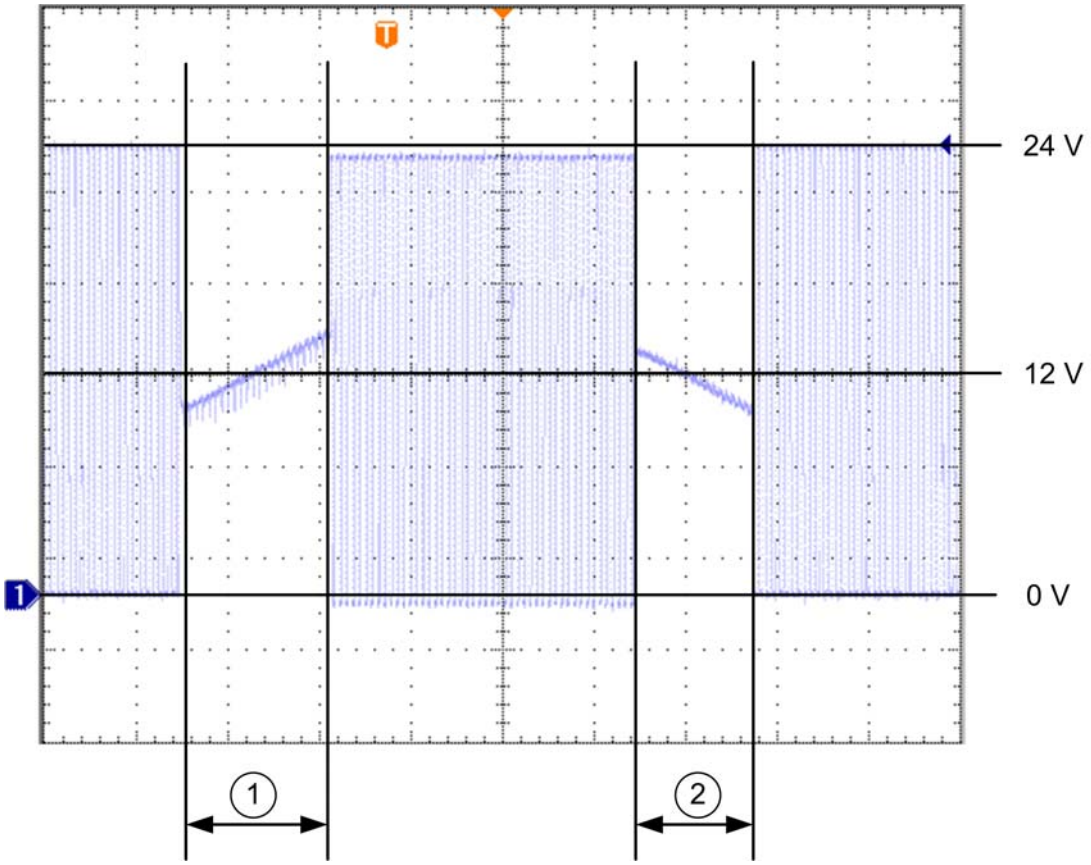


Figura 7.6. Zonas de la tensión sensada que corresponden a la fuerza contraelectromotriz.

En la figura 7.6 se distinguen claramente los sectores durante los cuales la fase es alimentada por el módulo PWM de los que no. Estos segundos han sido señalados en la imagen anterior como los sectores 1 y 2. Dado que en esta aplicación las tensiones de fase van de los 0 a los 24 V, el valor medio de tensión que servirá como referencia para los cruces de la fuerza contraelectromotriz será 12 V. La fuerza contraelectromotriz contiene ruido de alta frecuencia, provinente de la conmutación PWM, que puede provocar errores en la detección del cruce.

Existen varias opciones para filtrar el ruido PWM acoplado a la fuerza contraelectromotriz. El procedimiento seguido ha sido diseñar filtros pasa bajos sencillos con distintas topologías, para poder compararlos y evaluarlos posteriormente. La figura 7.7 muestra la FFT de la tensión en uno de los devanados del motor. El primer armónico aparece a 7,3 kHz, por lo que se ha trabajado con una frecuencia de corte del filtro de 700 Hz para todas las propuestas estudiadas.

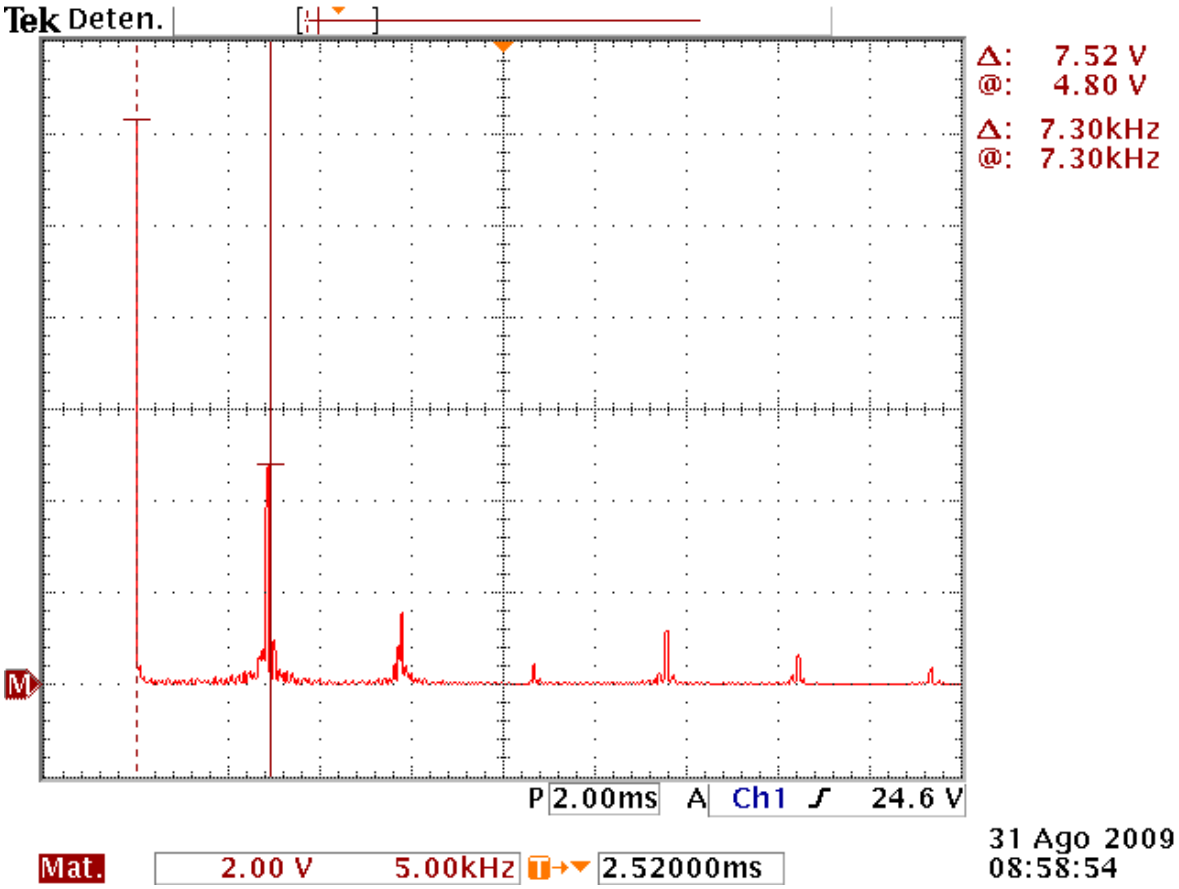


Figura 7.7. Transformada rápida de Fourier de la fuerza contraelectromotriz.

7.4.2.1 Filtro analógico RC

La primera de las opciones que se han considerado es la implementación de un filtro analógico RC, que presenta la topología mostrada en la figura 7.8.

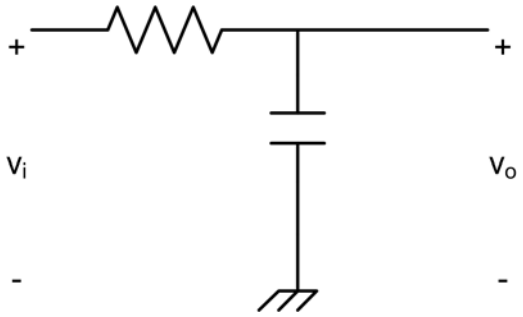


Figura 7.8. Filtro RC.

La función de transferencia de este filtro es

$$H(s)\frac{V_o(s)}{V_i(s)} = \frac{1}{1+RCs} \qquad (40)$$

Y la respuesta en frecuencia

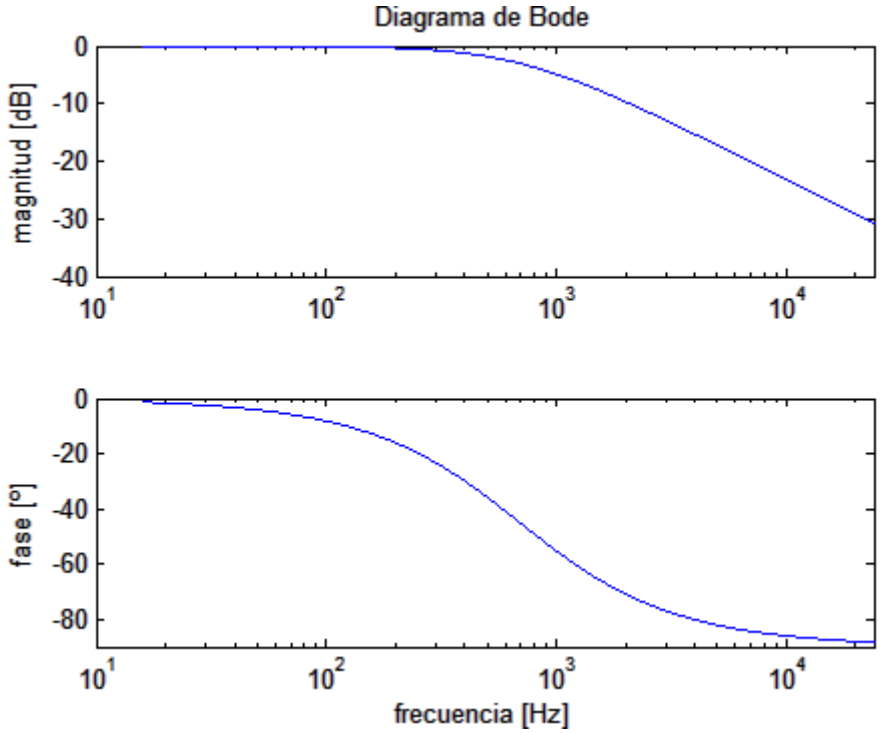


Figura 7.9. Respuesta en frecuencia del filtro RC.

La frecuencia de corte se define como la frecuencia para la cual el módulo de la función de transferencia se ha visto reducido en 3 dB respecto su valor máximo:

$$\left|H(j\omega_c)\right| = \frac{1}{\sqrt{2}}H_{max} \qquad (41)$$

El valor máximo de la función de transferencia para el filtro estudiado es

$$H_{max} = H(j\omega)\big|_{\omega=0} = 1 \qquad (42)$$

A partir de las expresiones anteriores se relaciona la frecuencia de corte con los valores de los componentes que forman el filtro:

$$|H(j\omega_c)| = \frac{1}{\sqrt{1^2 + R^2C^2\omega_c^2}} = \frac{1}{\sqrt{2}} \qquad (43)$$

Finalmente, simplificando la relación anterior se obtiene la expresión de la frecuencia de corte.

$$\omega_c = \frac{1}{RC}; \quad f_c = \frac{1}{2\pi RC} \qquad (44)$$

La frecuencia de corte debe situarse, por lo mínimo, una década por debajo de la frecuencia que se desea filtrar. Para simplificar, la frecuencia de corte será de 700 Hz. De forma arbitraria se ha elegido un valor de 1 µF para el condensador.

La resistencia para obtener la frecuencia de corte deseada debería ser

$$\omega_c = \frac{1}{RC}; R = \frac{1}{2\pi f_c C} = 227{,}4\ \Omega \qquad (45)$$

El valor comercial más próximo es 227 Ω, y los efectos de esta modificación en la frecuencia de corte son irrelevantes.

A continuación se muestra el resultado del filtrado de la señal. En el canal 1 se muestra la tensión que cae en bornes de uno de los devanados antes del filtrado, y en el 2 después de él.

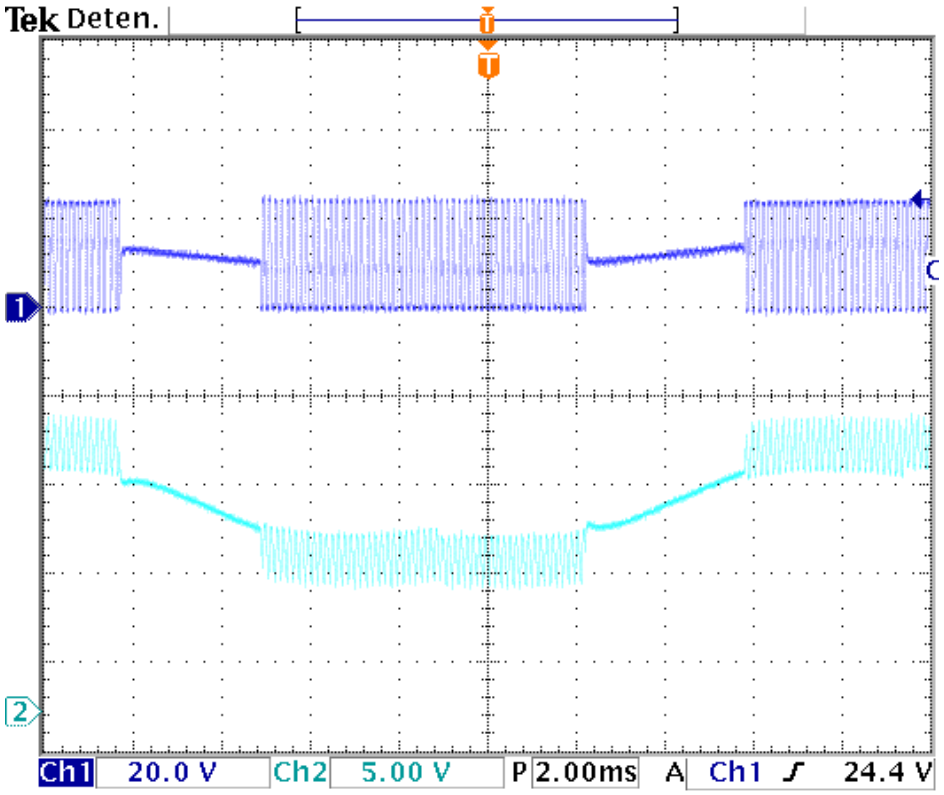


Figura 7.10. Señal original y filtrada mediante RC.

Si se aumenta la velocidad de giro del motor el efecto del inductive kickback se hace evidente, tal como se muestra en la figura 7.11. Puede incluso darse el caso que este efecto produzca una detección de paso por cero falsa. Este fenómeno se describe más adelante, en el apartado 7.4.4.1.

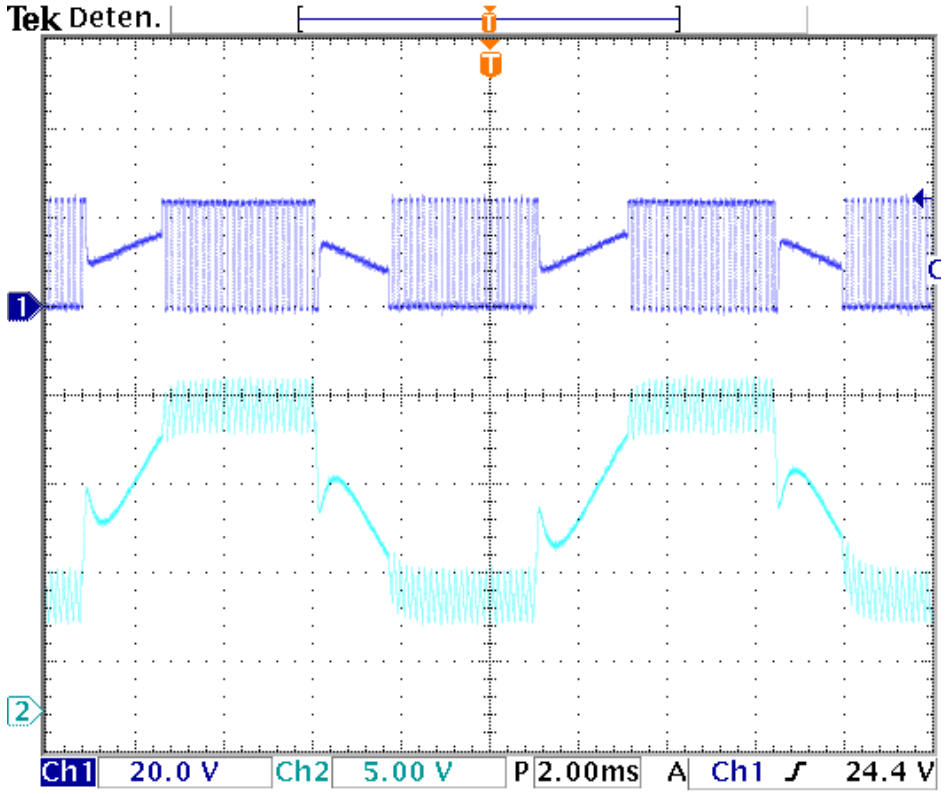


Figura 7.11. Señal original y filtrada mediante RC.para una velocidad superior.

El ruido acoplado a la señal procedente de la alimentación PWM no se ha atenuado lo suficiente. Una solución podría ser aumentar el orden del filtro, aunque se ha optado por analizar otras topologías.

7.4.2.2 Filtro analógico Sallen-Key

El filtro Sallen-Key es un filtro pasa bajos de segundo orden. Se pueden encontrar dos topologías básicas: con ganancia unitaria y ganancia diferente a la unidad. Se ha optado por el filtro con ganancia unitaria para realizar este estudio previo. El circuito se muestra a continuación

Mediante la ley de corrientes de Kirchhoff se obtienen las dos ecuaciones siguientes

$$\frac{v_i - v_x}{R_1} = \frac{v_x - v_o}{\frac{1}{C_1 s}} + \frac{v_x - v_o}{R_2} \qquad (46)$$

$$\frac{v_x - v_o}{R_2} = \frac{v_o}{\frac{1}{C_2 s}} \qquad (47)$$

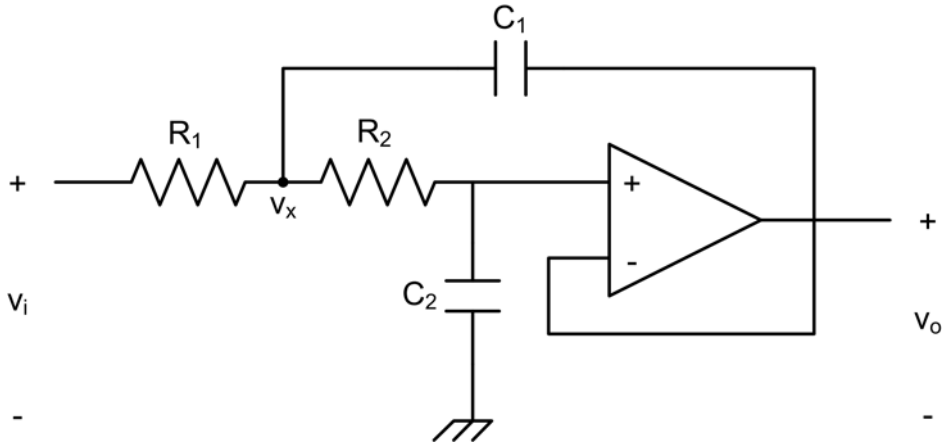


Figura 7.12. Topología Sallen-Key de ganancia unitaria.

Aislando v_x de la ecuación (47) se obtiene la expresión mostrada a continuación

$$v_x = v_o(1 + R_2C_2s) \tag{48}$$

Sustituyendo (48) en (46)

$$\frac{v_i - v_o(1 + R_2C_2s)}{R_1} = \frac{v_o(1 + R_2C_2s) - v_o}{\frac{1}{C_1s}} + \frac{v_o(1 + R_2C_2s) - v_o}{R_2} \tag{49}$$

Operando y realizando las simplificaciones pertinentes, la función de transferencia que se obtiene es

$$H(s)\frac{V_o(s)}{V_i(s)} = \frac{1}{1 + C_2(R_1 + R_2)s + C_1C_2R_1R_2s^2} \tag{50}$$

Siguiendo el mismo procedimiento que para el caso del filtro RC, se conoce que la frecuencia de corte del filtro viene definida por

$$f_c = \frac{1}{2\pi\sqrt{R_1R_2C_1C_2}} \tag{51}$$

Para simplificar los cálculos, se decide $R_1 = R_2 = R$ y $C_1 = C_2 = C$. Hecha esta consideración, la frecuencia de corte toma la siguiente expresión

$$f_c = \frac{1}{2\pi RC} \tag{52}$$

Esta expresión es la misma que la obtenida para el filtro RC. Por lo tanto, la elección de los componentes se realiza mediante los mismos criterios, y finalmente los valores conseguidos son $R_1 = R_2 = 227\ \Omega$ y $C_1 = C_2 = 1\ \mu F$.

La respuesta en frecuencia para estos valores es la siguiente

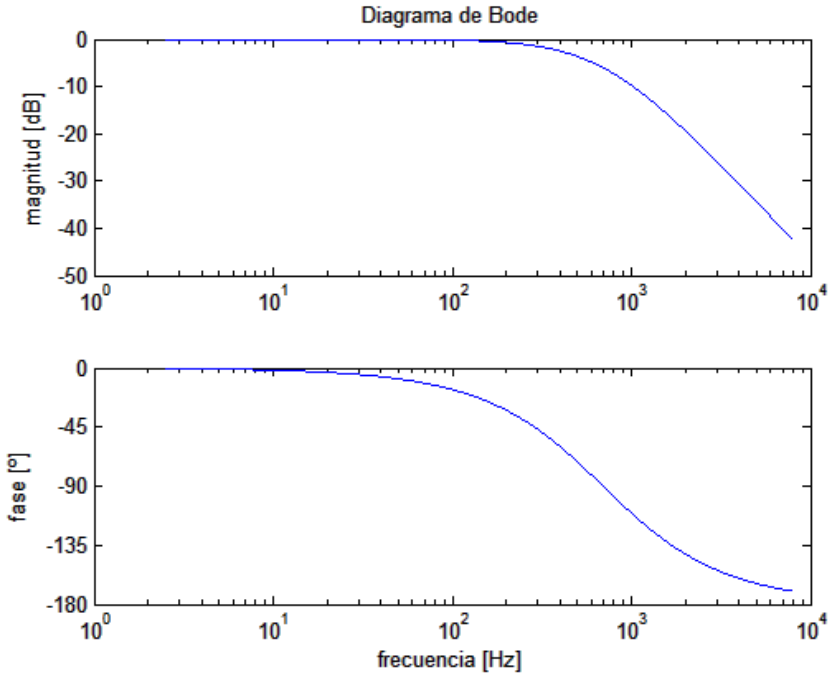


Figura 7.13. Respuesta en frecuencia del filtro Sallen Key diseñado.

El filtro descrito se ha construido y a continuación se muestran los resultados.

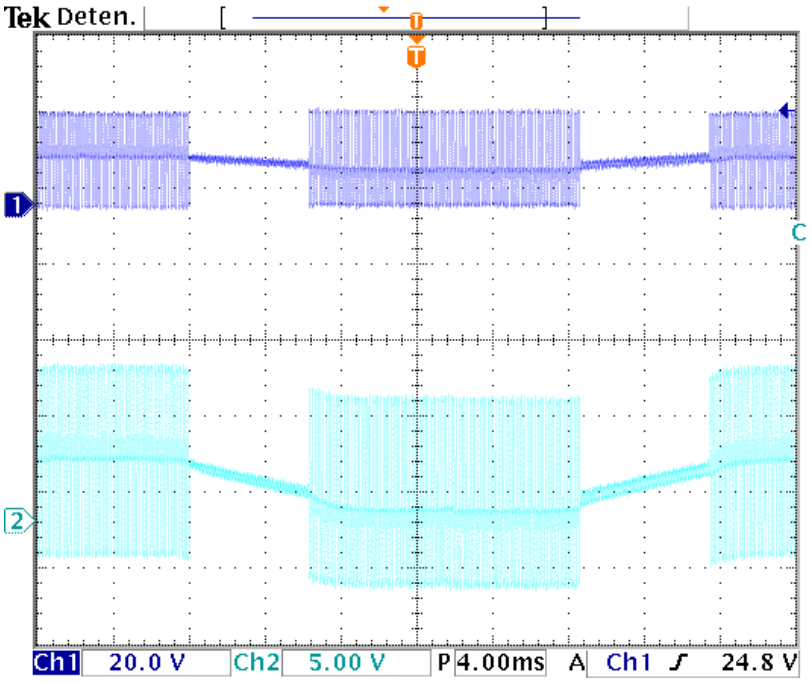


Figura 7.14. Resultados del filtrado a baja velocidad.

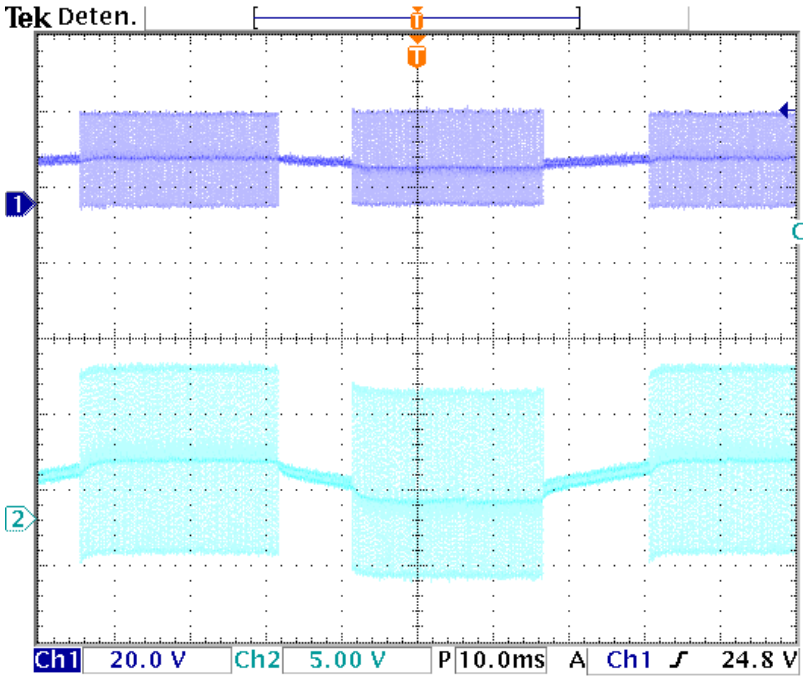


Figura 7.15. Resultados del filtrado a alta velocidad.

Tal como se ve en la figura 7.14 y en la figura 7.15, la señal presenta demasiado ruido para realizar una detección clara del cruce de la fuerza contraelectromotriz por el punto medio. Uno de los motivos por lo que esto sucede es por la reducción de la calidad del filtro al asignar el mismo valor a ambos pares de resistencia-condensador. Sin embargo, si se mejorara este aspecto el resultado del filtrado no sería considerablemente mejor.

7.4.2.3 Filtro RC digitalizado

La tercera opción considerada ha sido la digitalización del filtro RC estudiado en el apartado 7.4.2.1. Para ello se ha tomado la función de transferencia descrita en (40), y evaluado con los valores de resistencia y condensador anteriores (227 Ω y 1 μF, respectivamente)

$$H(s) = \frac{1}{0{,}000227s + 1} \qquad (53)$$

Para discretizar la función de transferencia anterior primero se debe conocer la frecuencia de muestreo que se va a usar. En este caso se ha usado la misma que en [4], que es de 49.152 Hz. La función de transferencia obtenida en el dominio discreto es la siguiente

$$H(z) = \frac{0{,}08573}{z - 0{,}9143} \qquad (54)$$

Finalmente, pues, el filtro que se estudia aplicar a la señal muestreada es

$$y(n) = 0{,}08573 \cdot x(n-1) + 0{,}9143 \cdot y(n-1) \qquad (55)$$

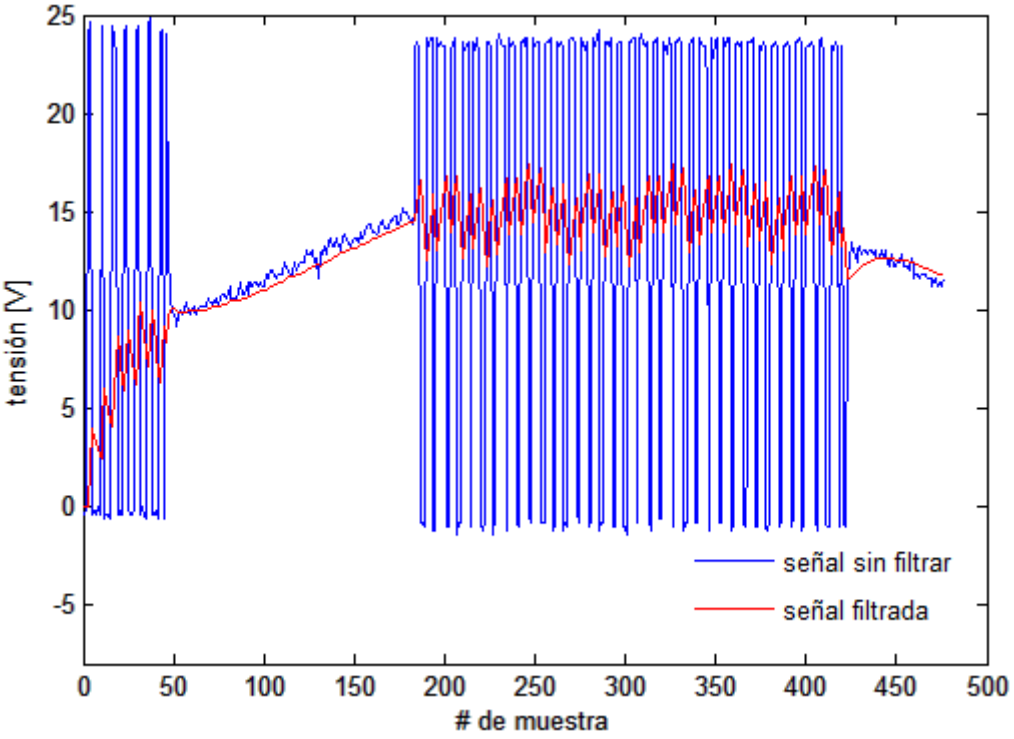


Figura 7.16. Resultado de aplicar el filtro RC digitalizado.

Tal como se aprecia en la figura 7.16, la señal de la fuerza contraelectromotriz presenta un filtrado que permite una fácil detección del cruce de dicha señal por el punto medio de la tensión de alimentación. El código utilizado para realizar la transformada y l imagen anterior se ha trascrito en el capítulo 13, Anexo III: Obtención del filtro digitalizado.

7.4.3 Baja velocidad

El presente algoritmo empieza por realizar un sensado de las fuerzas contraelectromotrices de cada uno de los devanados. La frecuencia de este muestreo deberá ser superior al doble de la frecuencia del PWM usado para alimentar las fases.

Además, es necesaria la utilización de dos *timers* del microprocesador. El primero, timer 1, se usará para medir el tiempo transcurrido entre un paso por 0 V de cualquiera de las fases y el siguiente. Este procedimiento se ilustra en la figura 7.17.

El tiempo que se obtiene en esta medición es el que se ha necesitado para realizar un desplazamiento de 60º eléctricos.

Por lo tanto, dividiendo el valor del *timer* 1 por dos se obtiene, aproximadamente, el tiempo transcurrido para recorrer los últimos 30º. Si la velocidad del motor no varía muy bruscamente, el tiempo a emplear para girar los siguientes 30º será muy parecido al calculado. De esta forma se tiene una referencia temporal para modificar la alimentación de los devanados.

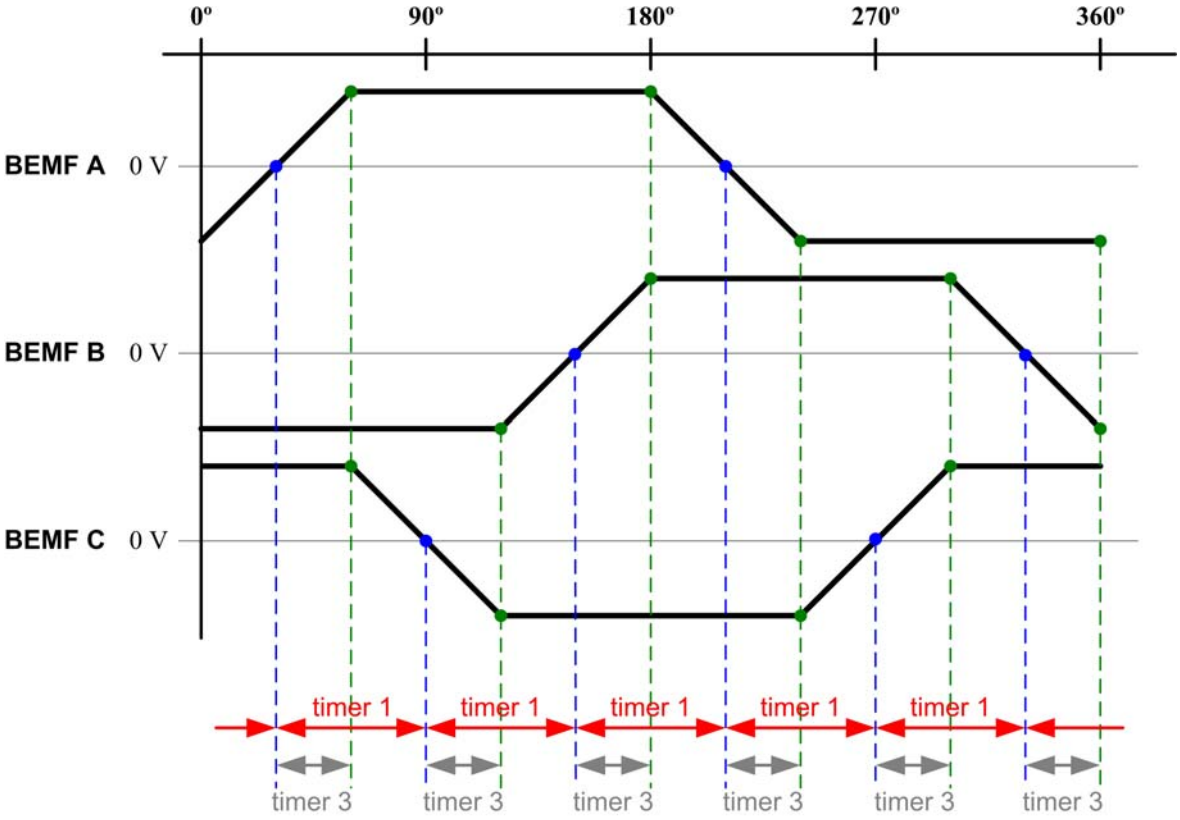


Figura 7.17. Contadores usados en el algoritmo de baja velocidad.

Se sabe que 30º eléctricos tras el cruce de la fuerza contraelectromotriz se debe realizar la conmutación, y se ha calculado de forma aproximada el tiempo que tardarán en recorrerse los 30º. Así pues, el *timer* 3, de tipo decremental, se carga con el valor del *timer* 1 dividido por dos y se inicia después de cada paso por 0 V. El instante en que llegue a cero será el momento en el cual conmutar las fases, tal como se aprecia en la figura 7.17. Para ello, cuando el contador llega a cero genera una interrupción, en cuya rutina de servicio a la interrupción se conmutan las fases.

El diagrama de flujo del algoritmo se presenta en la figura 7.18.

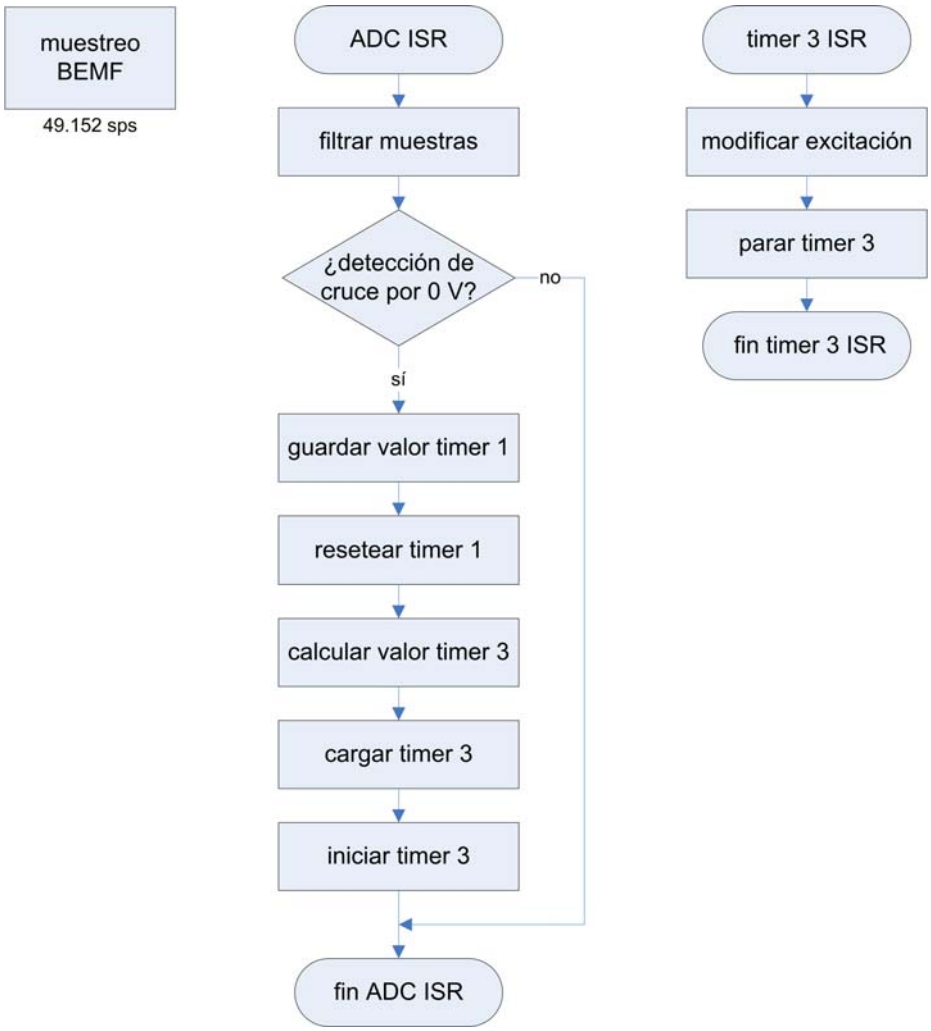


Figura 7.18. Diagramas de flujo del algoritmo de baja velocidad.

7.4.3.1 Retrasos debidos al microprocesador

El algoritmo expuesto se ve afectado por varias fuentes de retraso:

- **Filtro**: la presencia del filtro digital IIR ocasiona un retraso, que se debe cuantificar y restar del valor a cargar en el contador *timer* 3.
- **Conversor analógico-digital**: la rutina de servicio a la interrupción del conversor analógico digital supone también un retraso, ya que ejecuta tres veces el filtro IIR antes de determinar si se ha producido un paso por 0V. Este retraso también debe ser tomado en cuenta para el *timer* 3.

El valor del tiempo a precargar en el *timer* 3 también se ve reducido si se desea aplicar un **avance de fase**. Éste se utiliza para hacer girar el motor a una velocidad superior a la nominal. Para ello se conmuta antes de haberse desplazado 30º desde detección del cruce por cero. Sin embargo esta técnica supone un exceso de corriente a través de los devanados, lo que produce sobrecalentamiento y, finalmente, una reducción de la vida del motor.

Finalmente, el periodo del timer 3 viene determinado por la expresión siguiente [4]

$$PR3 = T_{30} - D_{FILT} - D_{PROC} - D_{PA} \qquad (56)$$

Donde:

PR3 es el valor del registro de periodo del *timer* 3

T_{30} es el tiempo transcurrido para recorrer los últimos 30º eléctricos

D_{FILT} es el retraso provocado por el filtro

D_{PROC} es el retraso por el procesamiento de la interrupción del conversor A/D

D_{PA} es el tiempo correspondiente al avance de fase, en caso que se implemente

7.4.4 Alta velocidad

El algoritmo de alta velocidad es necesario cuando los retardos del filtro y del procesador son más largos que lo que se tarda en recorrer 30º eléctricos. Por lo tanto, el cruce por cero es detectado después de que la conmutación debiera ser llevada a cabo, y la implementación anterior deja de ser válida.

El muestreo se realiza únicamente para una de las fases, pero su frecuencia de muestreo aumenta a 81.940 Hz. Es decir, la resolución será cinco veces mayor que en el caso de baja velocidad. Cabe recordar que la amplitud de la fuerza contraelectromotriz en una fase es directamente proporcional a la velocidad de giro del motor. Por lo tanto, cuanto mayor sea la velocidad mayor será también la pendiente de la BEMF durante la cual debe producirse la detección del paso por 0 V. Por este motivo el aumento de resolución resulta necesario, y asegurar una correcta detección del cruce por cero.

En este caso, se usará el *timer* 3 para contabilizar 90º eléctricos, en lugar de los 30º para el algoritmo de baja velocidad. De esta forma se consigue determinar no el instante de conmutación inmediatamente posterior, sino el siguiente.

En la figura 7.19 se muestra el uso dado a cada uno de los tres contadores. El *timer* 3 no se inicia hasta haber calculado el periodo de tiempo para 90º, dividiendo el valor del *timer* 1 entre 2. El resto de conmutaciones vendrán determinadas por el timer 2, que contará el periodo necesario para recorrer 60º eléctricos. Este contador funciona continuamente, y se sincroniza con el *timer* 3 al producirse su rutina de servicio a la interrupción. Esta sincronización es necesaria, ya que el tiempo estimado para recorrer 60º no será exactamente el invertido a tal efecto, y surgirán pequeños retrasos o adelantes de fase en la conmutación. Mediante la sincronización, cada 180º eléctricos se eliminará el error acumulado.

En la figura 7.20 se representan los diagramas de flujo para implementar el algoritmo de alta velocidad.

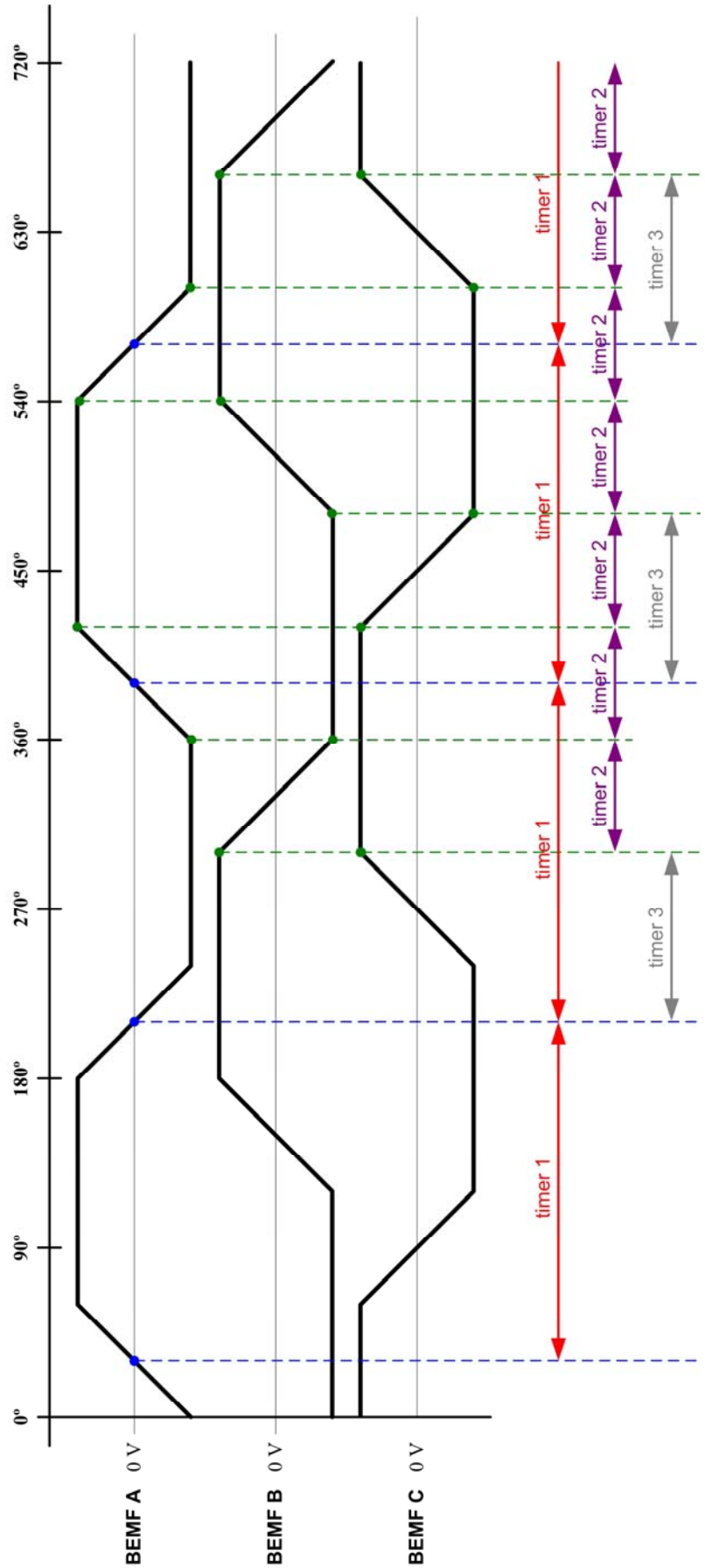


Figura 7.19. Contadores usados en la implementación de alta velocidad.

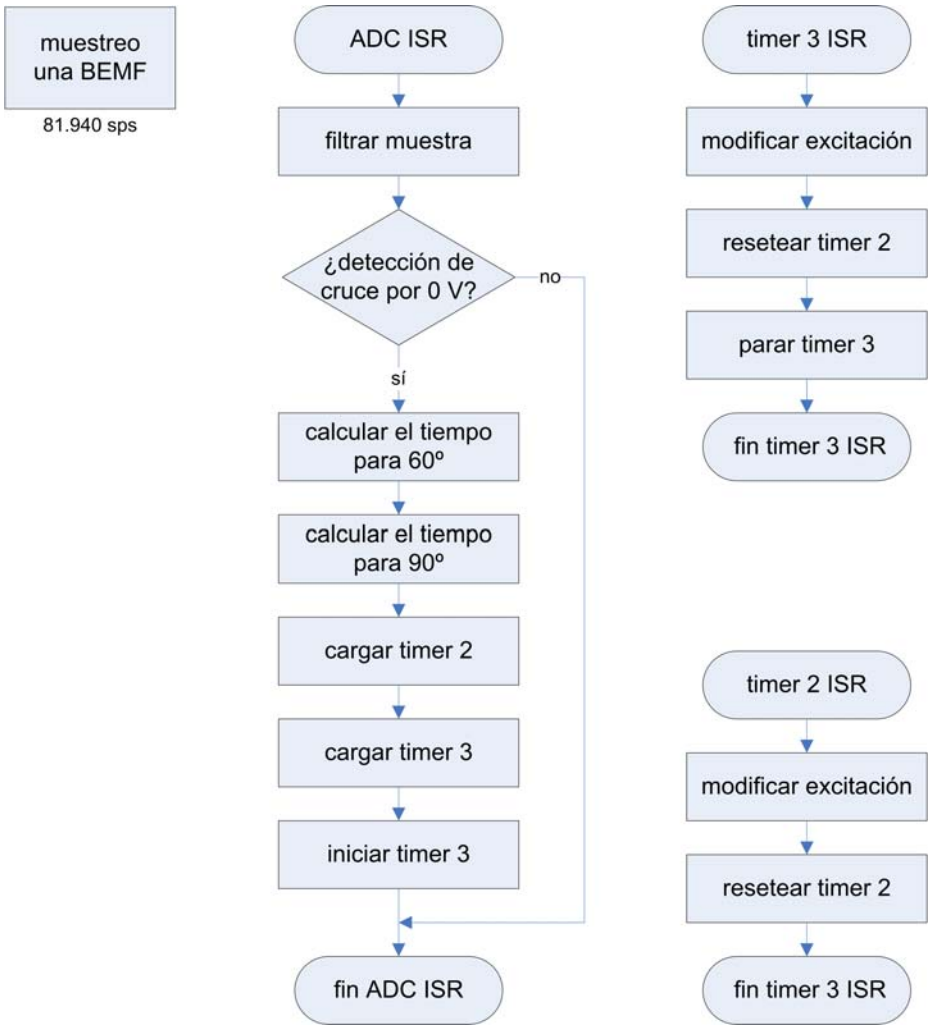


Figura 7.20. Algoritmo para la implementación de alta velocidad.

7.4.4.1 Inductive kickback

El *inductive kickback* produce picos en la tensión de las fases. Cuando una fase del motor deja de estar alimentada, el devanado de esta fase intenta mantener durante un tiempo limitado la circulación de corriente, produciendo picos de tensión. Este fenómeno se produce cuando el motor gira a velocidades elevadas, ya que la corriente que circula por los devanados es mayor cuanto más alta es la velocidad.

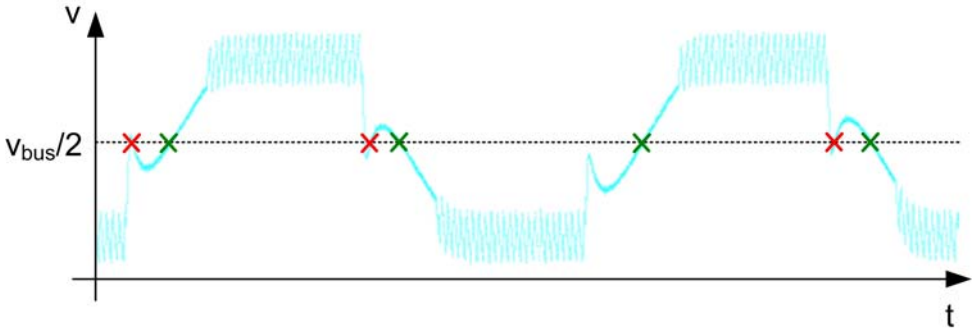


Figura 7.21. Efecto del *kickback* inductivo en la detección de los cruces.

En la figura 7.21 se representa la tensión de una fase en la cual el fenómeno *kickback* es bien visible. Se puede observar que algunos de los picos de tensión son suficientemente grandes como para introducir errores en la detección del cruce de la fuerza electromotriz. En la imagen anterior los cruces a detectar se han señalado con una cruz verde, mientras que las falsas detecciones se simbolizan con una roja.

A la vista de este hecho, resulta necesario que el algoritmo a implementar ignore los cruces precoces. Para ello, tras cada conmutación hay un periodo de tiempo durante el cual la muestra proporcionada al filtro digital no es la de la fuerza contralectromotriz, sino la última muestra filtrada. Este procedimiento se realiza durante unas determinadas cuentas de reloj, periodo denominado como cuentas en blanco. Es preciso estimar el tiempo necesario para obviar los falsos cruces por cero, pero de forma que no se dejen de detectar los cruces reales.

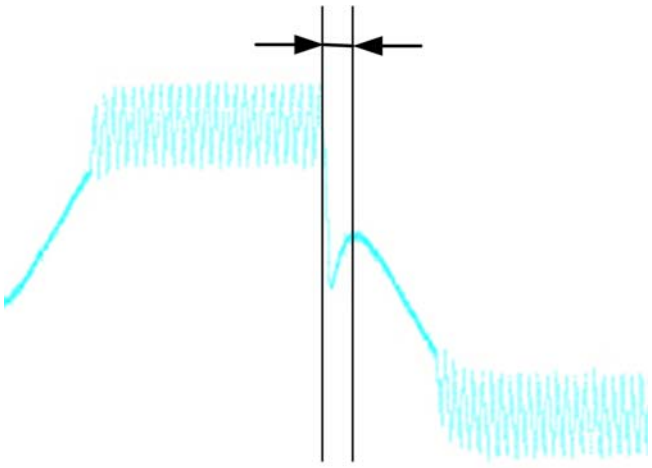


Figura 7.22. Cuentas en blanco para evitar la detección de falsos cruces.

7.5 Implementación

7.5.1 Algoritmo de arranque

El conocimiento de la posición del rotor es imprescindible para accionar el motor. Sin embargo, cuando éste se encuentra reposo la posición se desconoce. El arranque, pues, se conformará básicamente de dos partes: llevar el motor a una posición conocida y hacerlo girar de forma síncrona de acuerdo con una rampa de velocidad, hasta que la fuerza contraelectromotriz pueda ser sensada.

7.5.1.1 Preposicionamiento

El procedimiento consiste en alimentar dos de las tres fases, durante suficiente tiempo para que el rotor se alinee con el campo del estator. Si la corriente aplicada es demasiado elevada, el rotor oscilará alrededor de la posición final. Con el objetivo de

evitar este inconveniente, se aplican unos valores no constantes de corriente. Para ello, se aplican varios pulsos de tensión a las fases, con un ciclo de trabajo que se decrementará de forma progresiva. Tras realizar varios ensayos, la rampa adoptada para el motor y la carga usados son los representados a continuación.

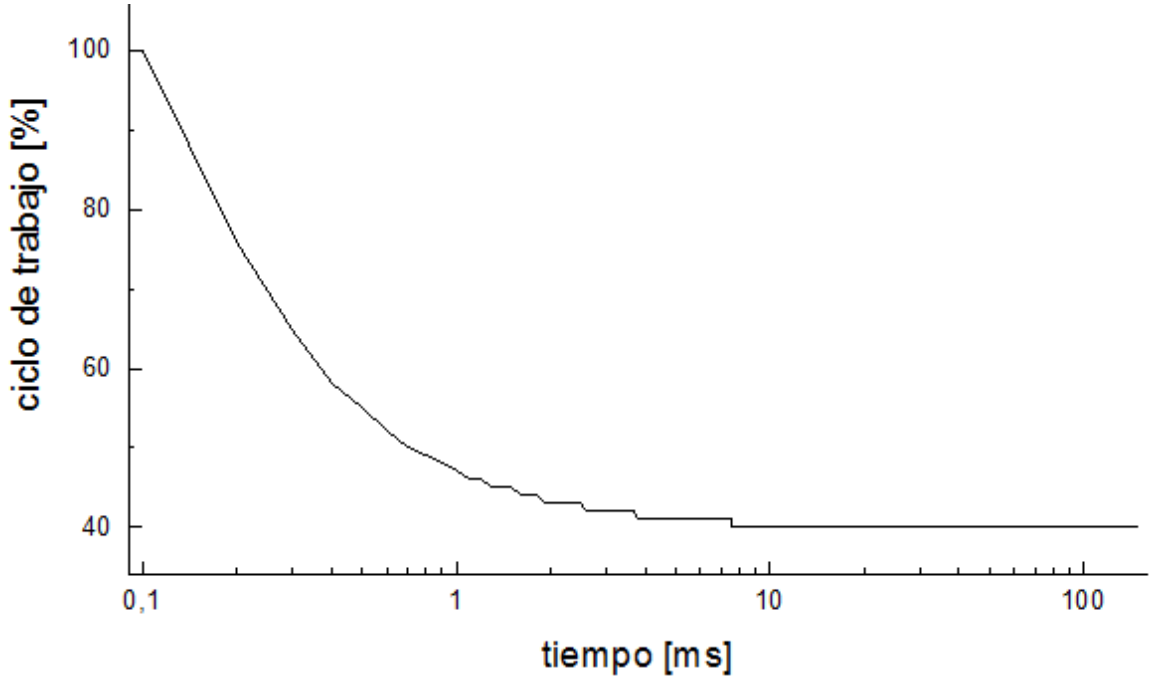


Figura 7.23. Ciclo de trabajo para la primera alineación del rotor.

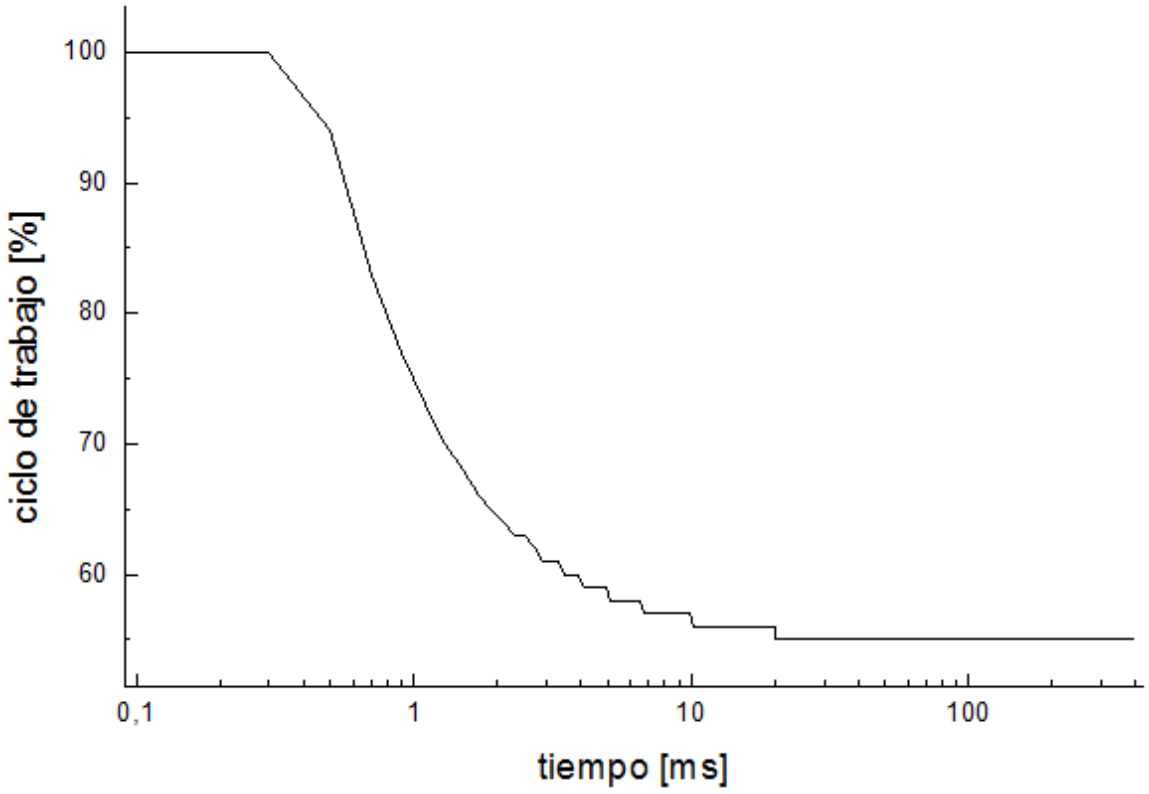


Figura 7.24. Ciclo de trabajo para la segunda alineación.

Con un valor inicial de corriente más elevado, se consigue poner el motor en movimiento. Una vez éste ya se está desplazando, la corriente necesaria para realizar la alineación es menor. Se realizan dos alineaciones a diferentes posiciones para estar seguros de que esta alineación se ha producido. Si este procedimiento se llevara a cabo una sola vez y el rotor estuviera colocado por azar en la que debería ser su posición final, no se podría discernir si la corriente aplicada es demasiado baja o si el motor ya se encuentra en su posición final.

7.5.1.2 Rampa de velocidad

Tal como se ha comentado anteriormente, la fuerza contraelectromotriz es directamente proporcional a la velocidad de giro. La posición del rotor se extrae del momento en que la fuerza contraelectromotriz coincide con la mitad de la tensión del bus.

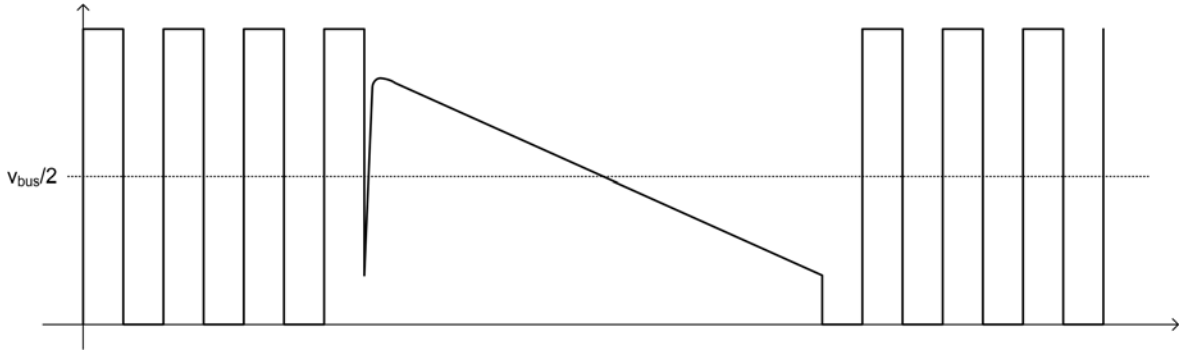


Figura 7.25. Cruce por la mitad de la tensión del bus a velocidad media-alta.

Figura 7.26. Cruce por la mitad de la tensión del bus a baja velocidad.

De la comparación de la figura 7.25 y la figura 7.26 se extrae que la detección del instante en el que se produce el cruce con exactitud es más sencilla cuanto mayor sea la velocidad. Esto se debe al incremento de la pendiente de la fuerza contraelecromotriz inducida a medida que la velocidad aumenta.

7.6 Ensayo

Tal como se ha comentado anteriormente, el accionamiento estudiado y utilizado en el laboratorio se trata de una versión a escala de la aplicación real. Por lo tanto, el objetivo del ensayo es comparar el funcionamiento de ambas aplicaciones, sin olvidar las diferencias que hay entre los dos conjuntos motor-ventilador.

El procedimiento que se ha llevado a cabo ha sido llevar el motor a distintas velocidades, y comprobar cuál es su consumo. Los resultados se presentan en la siguiente tabla.

velocidad [rpm]	corriente [A]	tensión [V]	potencia [W]
628	0,1	24,2	2,42
902	0,105	24,2	2,541
1166	0,115	24,2	2,783
1359	0,125	24,2	3,025
1444	0,13	24,2	3,146
1543	0,14	24,2	3,388
1687	0,15	24,2	3,63
1774	0,16	24,2	3,872

Tabla 7.2. Resultados del ensayo con el motor *brushless*.

Por lo tanto, el rango de funcionamiento del accionamiento con motor *brushless* con el algoritmo desarrollado es de aproximadamente 600 a 1800 rpm. Para velocidades inferiores a las reproducidas en la tabla, el cálculo de la velocidad presenta algunos errores, debido a las malas detecciones de los cruces. Además, la cantidad de aire desplazado es muy baja. Este hecho no supone ningún inconveniente, ya que el ventilador del radiador de un coche deberá girar siempre a velocidades superiores a ésta.

Para velocidades por encima de la máxima en la tabla, en cambio, el par que el motor proporciona no es suficiente para arrastrar el ventilador. Es decir, si se quisieran conseguir velocidades superiores a los 1800 rpm se debería elegir un motor que proporcionara más par que el usado en este proyecto. Sin embargo, este hecho no descarta la validez del algoritmo elegido e implementado a lo largo de este proyecto.

La característica de consumo con respecto a la velocidad que presenta el accionamiento estudiado es la representada en la figura 7.27.

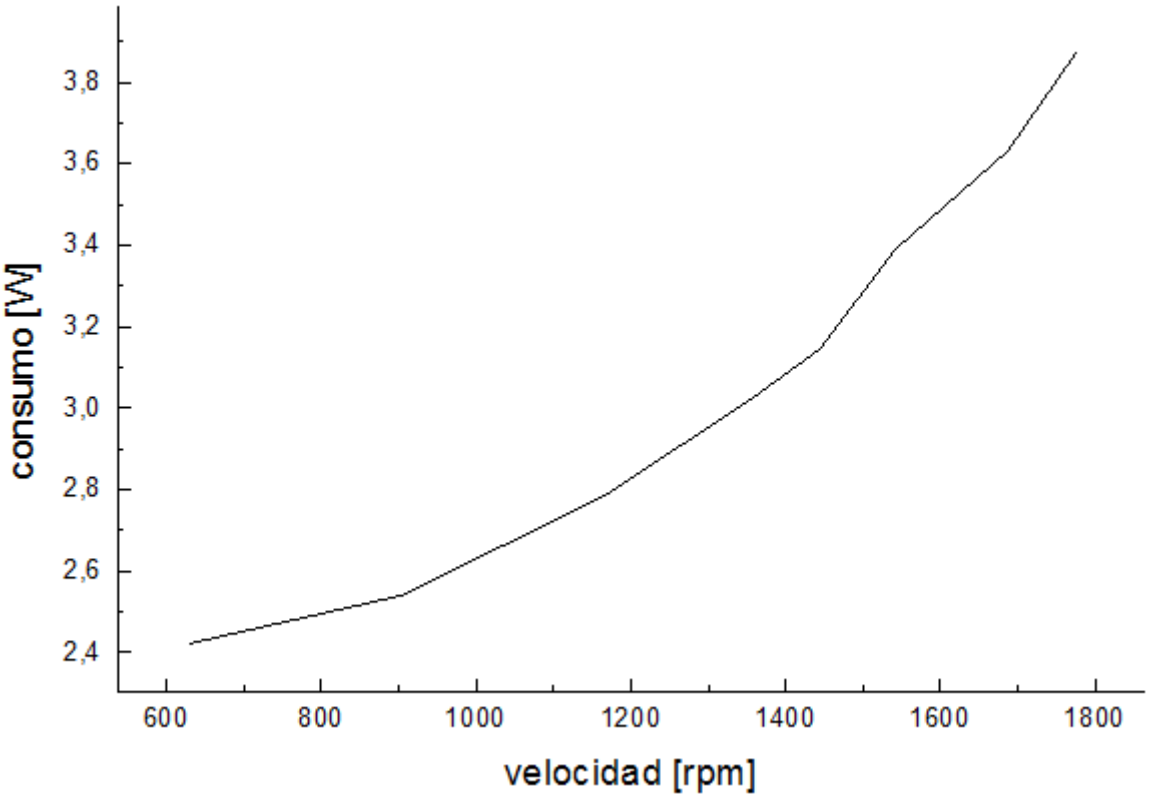


Figura 7.27. Curva consumo-velocidad del accionamiento.

8 Introducción a la previsión de fallos

Las ventajas de la predicción de fallos son muchas. La más inmediata es mejorar el diseño y sus protecciones una vez identificados los puntos débiles. Pero la predicción de fallos permite, además, monitorizar el estado del sistema y conseguir prever el fallo que se va a producir.

8.1 Posibles causas de fallo

Algunos fallos son causa del desgaste del propio motor, y otras de la parte de control y alimentación del accionamiento. Los más habituales se enumeran a continuación [14]:

- **fase en circuito abierto**: se puede producir bien por el fallo de los dos transistores en una rama del convertidor o bien por la desconexión del conductor que une el terminal del devanado con la salida del inversor si este se queda al aire.
- **fase cortocircuitada**: uno de los transistores de la rama no puede desconectarse. Otra posibilidad es que el conductor que une el terminal y el inversor se desconecte y toque a la carcasa del motor.
- **cortocircuito de todas las fases**: existen varias posibilidades. La primera es que la fuente de tensión continua que alimenta el inversor sea cortocircuitada. Otra es que conduzcan a la vez todos los transistores de arriba o de abajo del inversor. La última, menos probable, es que los tres terminales de los devanados estén conectados a masa.
- **fallo de conducción de uno de los transistores**: ocurre cuando el interruptor está cortado de forma permanente porque no le llega una señal de conmutación a la puerta.
- **generación no controlada**: se produce cuando el motor gira a una velocidad superior a la nominal, realizando un control con debilitamiento de flujo y este control desaparece repentinamente. Esto sucede si se daña el controlador o el sensor de posición, en caso de usarlo.
- **fallo del aislamiento de los devanados**: habitualmente se produce por un exceso de *stress* eléctrico, térmico o mecánico. También puede darse por impurezas del propio material de aislamiento.
- **cortocircuito de algunas espiras del devanado**: en caso de un desgaste del aislamiento avanzado, se pueden producir arcos eléctricos entre una y otra espira.

8.2 Métodos para evitar fallos o minimizar sus efectos

Cada uno de los fallos descritos en el apartado anterior debe tratarse de una forma determinada. A continuación se analiza la detección a emprender para cada uno de ellos y su realizabilidad sobre el algoritmo ya implementado.

Detectar una fase en circuito abierto es relativamente sencillo, ya que se está sensando constantemente la tensión de cada una de las fases. Si se obserba que transcurrido cierto tiempo después de la conmutación las fases, en las que se debería aplicar una alimentación PWM, la tensión no coincide con lo previsto, existe un fallo de este tipo.

De forma similar, es posible detectar cuando por una fase que se debería encontrar en circuito abierto está circulando corriente. Este fallo corresponde a un transistor que no puede desconectarse. El otro caso de fallo por fase cortocircuitada es el de fase cortocircuitada a masa. La detección se realiza de forma idéntica al método para las fases en circuito abierto.

Los fallos por cortocircuito de todas las fases o bien por fallo de conducción de uno de los transistores se podrían detectar también por comparación de la alimentación PWM ordenada por el microprocesador y la que realmente se produce. Este sistema es efectivo con el algoritmo de control *sensorless* que se ha utilizado en este proyecto. Sin embargo, existe un inconveniente: aunque se detecta el fallo, éste no es diagnosticado. Para ello deberían usarse otras técnicas, que entrarían en conflicto con la capacidad del dsPIC30f6010a.

Cuando se trata de evitar la generación incontrolada, el accionamiento utilizado goza de una ventaja: la ausencia de sensores de posición, lo que impide que estos se averíen y den una lectura errónea de velocidad. Por lo que respecta a evitar la conmutación que haga girar el motor a velocidades superiores a la nominal, es tan sencillo como establecer un valor mínimo de tiempo entre conmutación y conmutación al contador que determina los cambios en la excitación de las fases.

Los fallos de aislamiento y cortocircuito de algunas espiras del devanado provocan una degradación de las características del motor. Estos son más complejos de detectar, ya que el proceso requiere transformaciones algebraicas y un cálculo continuo de datos que, juntamente con la alimentación PWM, el sensado y filtrado de la tensión de fase y el cálculo de los tiempos, superarían la capacidad del microprocesador.

Por lo tanto, con el accionamiento y el algoritmo de control de los que se dispone, es posible detectar varios casos de fallo del inversor y de las conexiones, mejorando aún más las características, y sobre todo la fiabilidad, del accionamiento con motor *brushless*.

9 Conclusiones

En este proyecto se ha estudiado la forma de accionar un ventilador para el radiador de un automóvil mediante un motor *brushless DC*, y posteriormente se ha llevado a cabo sobre una aplicación a escala de la mencionada.

Para ello se ha estudiado con detalle la conveniencia y el funcionamiento de este tipo de motores, así como el inversor usado. Posteriormente, se ha realizado un ensayo con el accionamiento presente actualmente en los vehículos, es decir, con el motor de corriente continua. Tras ello, con el motor brushless del que se disponía se ha realizado un montaje similar, pero con un ventilador adecuado a las características del nuevo motor.

Partiendo de la información recopilada y de los datos disponibles acerca del motor usado en la aplicación a escala, se ha realizado un modelo en el entorno *Matlab-Simulink*. Para tal efecto se ha hecho uso también de un programa de caracterización de motores. De esta forma, se ha dispuesto de dos referencias distintas para comparar los resultados del accionamiento final: el ensayo con el motor de corriente continua y la simulacción del accionamiento con el *brushless*.

Tras constatar la necesidad de implementar un algoritmo de control sin usar sensores de posición y evaluar las distintas alternativas propuestas hasta el momento, se ha implementado dicho algoritmo. Además, se ha introducido la teoría de control de fallos en este tipo de motores, ya que además de viable resulta muy interesante en vista a protección del accionamiento y fiabilidad.

En definitiva, se ha desarrollado un algoritmo válido de control de un motor sis sensores de posición. Con la simple modificación de unos parámetros, el mismo procedimiento puede ser aplicado para una aplicación de refrigeración de un radiador de autómóvil, en que la demanda de par y potencia es superior. Además, este sistema viene acompañado de un modelo mediante el cual evaluar los efectos que se producirían realizando pequeños cambios en el accionamiento.

Finalmente, se ha presentado la posibilidad de introducir la detección de ciertos fallos del inversor en el algoritmo existente, de forma sencilla y factible. El resultado final es un accionamiento de altas prestaciones, con una gran fiabilidad y unos costes bastante competitivos.

10 Futuras vías de trabajo

Este proyecto deja abiertas varias posibles vías de trabajo que serían interesantes de desarrollar. La más inmediata es el uso de un motor *brushless* de mayor potencia para accionar el ventilador realmente usado en la refrigeración del radiador de un automóvil. En esta nueva situación sería posible implementar y validar el algoritmo para altas velocidades. Además, las características de par, eficiencia y potencia serían directamente comparables a las obtenidas con el motor de corriente contínua.

Por otra parte, resultaría interesante desarrollar la detección de fallos en el inversor, mediante los procedimientos introducidos en el capítulo 8. El procedimiento a seguir consistiría en la introducción y simulación de los fallos en el modelo *Matlab-Simulink*. De esta forma sería posible conocer las desviaciones mediante el accionamiento normal y el defectuoso, y usar los datos obtenidos para implementar el algoritmo de detección con el microprocesador ya utilizado para el algoritmo de control.

Referencias

[1] Skvarenina, T.L., “The Power Electronics Handbook”, CRC Press, 2002

[2] J.M. Angulo, B. García, I. Angulo y J. Vicente. “Microcontroladores Avanzados dsPIC. Controladores Digitales de Señales. Arquitectura, programación y aplicaciones.”, Ed. Thomson-Paraninfo, 2006

[3] Yedamale, P., “AN885: Brushless DC (BLDC) Motor Control Fundamentals”, Microchip Technology Inc., 2003

[4] Condit, R., “AN1083: Sensorless BLDC Control with Back-EMF Filtering”, Microchip Technology Inc., 2007

[5] Microchip, “dsPIC30F6010 data sheet”

[6] Iizuka, Kenichi; Uzuhashi, Hideo; Kano, Minoru; Endo, Tsunehiro; Mohri, Katsuo, "Microcomputer Control for Sensorless Brushless Motor," *Industry Applications, IEEE Transactions on* , vol.IA-21, no.3, pp.595-601, May 1985

[7] Rani, B.I.; Tom, A.M., "Dynamic simulation of brushless DC drive considering phase commutation and backemf waveform for electromechanical actuator," *TENCON 2008 - 2008, TENCON 2008. IEEE Region 10 Conference* , vol., no., pp.1-6, 19-21 Nov. 2008

[8] Tao Sun; Suk-Hee Lee; Jung-Pyo Hong, "Faults analysis and simulation for interior permanent magnet synchronous motor using Simulink@MATLAB," *Electrical Machines and Systems, 2007. ICEMS. International Conference on* , vol., no., pp.900-905, 8-11 Oct. 2007

[9] Uk-Youl Huh; Je-Hie Lee; Tae-Gyoo Lee, "A torque control strategy of brushless DC motor with low resolution encoder," *Power Electronics and Drive Systems, 1995., Proceedings of 1995 International Conference on* , vol., no., pp.496-501 vol.1, 21-24 Feb 1995

[10] Ji Hua; Li Zhiyong, "Simulation of sensorless permanent magnetic brushless DC motor control system," *Automation and Logistics, 2008. ICAL 2008. IEEE International Conference on* , vol., no., pp.2847-2851, 1-3 Sept. 2008

[11] Pillay, P.; Krishnan, R., "Modeling, simulation, and analysis of permanent-magnet motor drives. I. The permanent-magnet synchronous motor drive," *Industry Applications, IEEE Transactions on* , vol.25, no.2, pp.265-273, Mar/Apr 1989

[12] Pillay, P.; Krishnan, R., "Modeling, simulation, and analysis of permanent-magnet motor drives. II. The brushless DC motor drive," *Industry Applications, IEEE Transactions on* , vol.25, no.2, pp.274-279, Mar/Apr 1989

[13] Byoung-Gun Park; Tae-Sung Kim; Ji-Su Ryu; Dong-Seok Hyun, "Fault Tolerant Strategies for BLDC Motor Drives under Switch Faults," *Industry Applications Conference, 2006. 41st IAS Annual Meeting. Conference Record of the 2006 IEEE* , vol.4, no., pp.1637-1641, 8-12 Oct. 2006

[14] M.A. Awadallah and M.M. Morcos, “Automatic fault diagnosis of electric machinery: a case study in PM brushless DC motors", Electric Power Components and Systems, Vol. 33, No. 6, pp. 597-610, 2005.

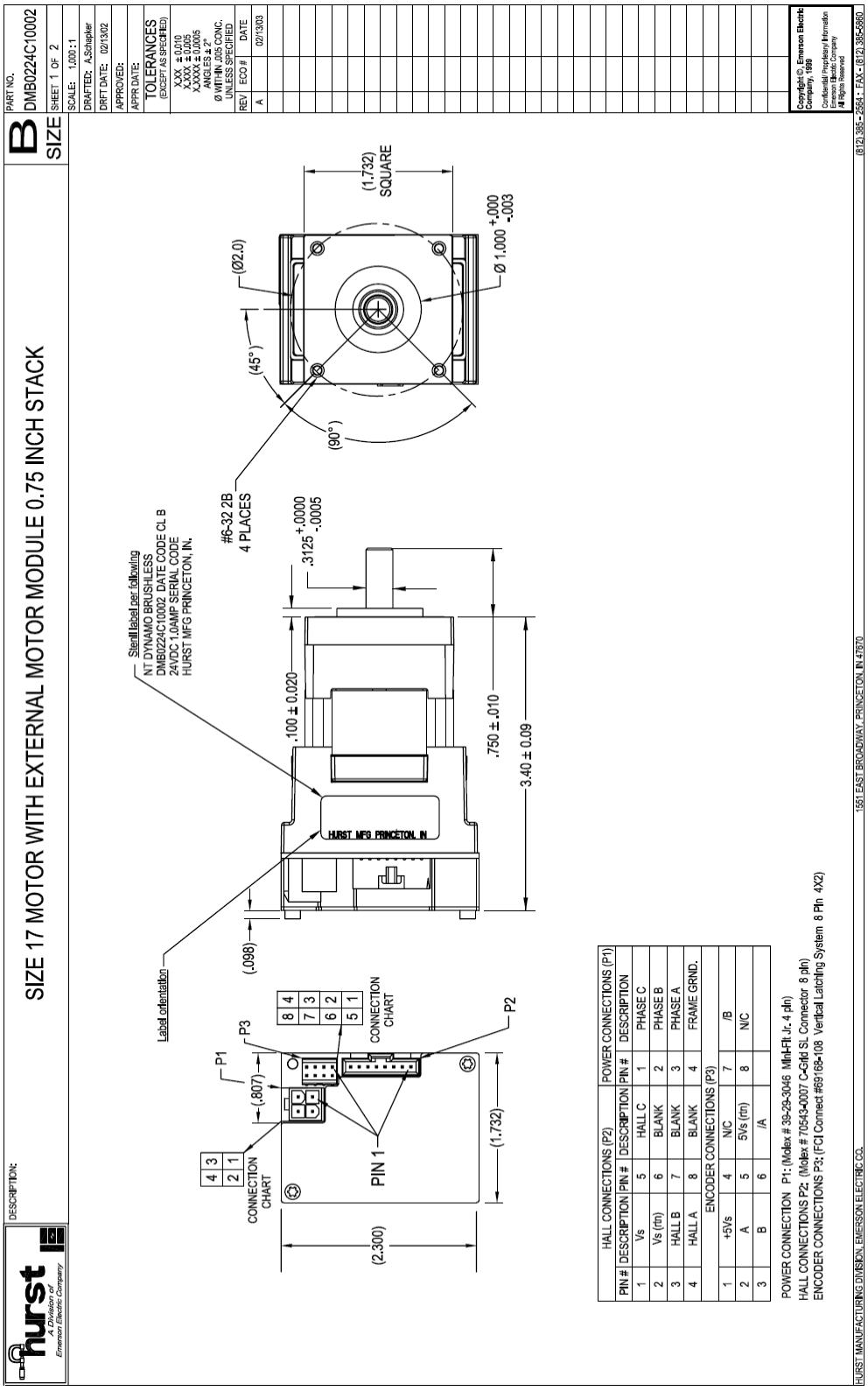
[15] Johnson, J.P.; Ehsani, M.; Guzelgunler, Y., "Review of sensorless methods for brushless DC," *Industry Applications Conference, 1999. Thirty-Fourth IAS Annual Meeting. Conference Record of the 1999 IEEE* , vol.1, no., pp.143-150 vol.1, 1999

[16] Agile Systems, “Advances in Sensorless Control of Brushless DC Motor”, E-Drive magazine, abril 2005

[17] http://www.magneticsolutions.com

[18] http://www.eetasia.com/ART_8800589208_1034362_NT_458b97fd.HTM

11 Anexo I: Características del motor usado



hurst

Sample Motor Data Sheet

Date: 2/14/02

Customer	Microchip	Model Number	DMB0224C10002
		Serial #	12482

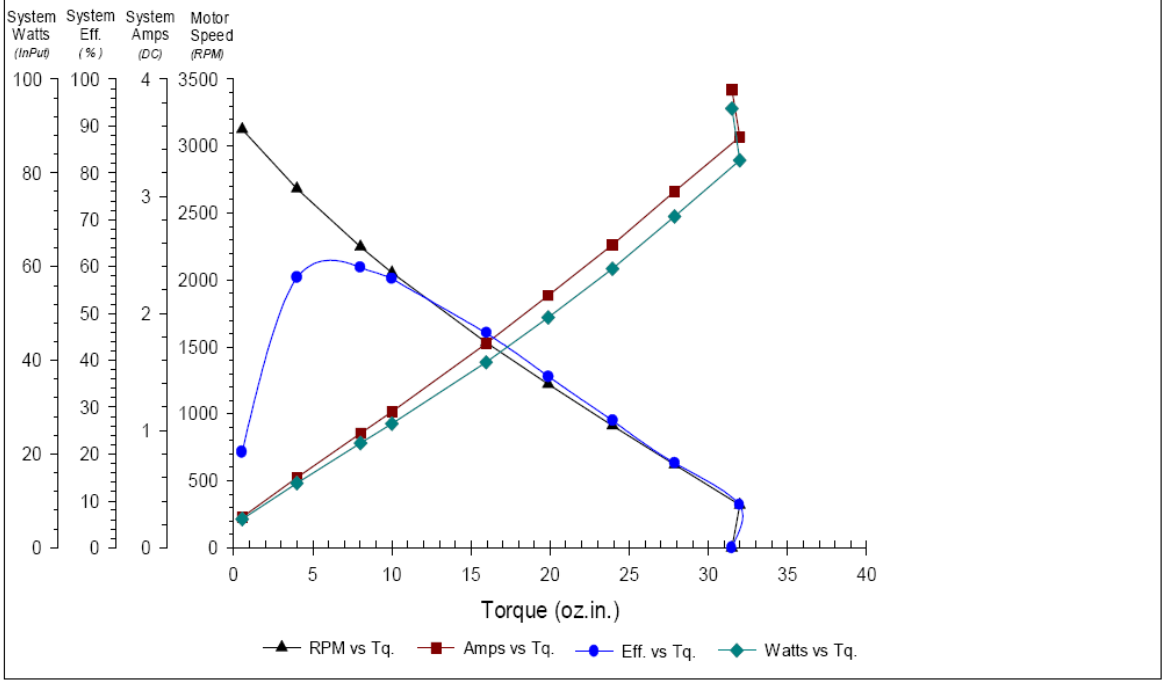
L-L Resistance (R_{tm}) Ohms :	4.03	Electrical Time Constant (t_e) mSec. :	1.14
L-L Inductance (L_{tm}) mH at 1Khz :	4.60	Mechanical Time Constant (t_m) mSec. :	3.74
Torque Constant (K_t) oz.in./Amp :	9.79	Thermal Resistance (R_{th}) °C/watt	4.78
Voltage Constant (K_e) V_{peak}/K_{RPM} :	7.24	Thermal Time Constant (t_{th}) min. :	16
Amb. Temp. (°C) :	22.7	Rotor Inertia (Jr) oz-in-s² :	0.000628
		Stack Length:	0.75

Notes:

Speed / Torque Test Data -Control Input set at 100% duty cycle.

Load	Volts (DC)	Amps (DC)	Watts (DC)	Speed (RPM)	Torque (oz.in.)	Output (watts)	Output (HP)	Eff. (%)	
1	24.01	0.26	6.11	3125	0.54	1.25	0.002	20.4	
2	24.01	0.60	13.76	2683	4.00	7.94	0.011	57.7	
3	24.01	0.98	22.29	2248	8.02	13.34	0.018	59.9	
4	24.01	1.16	26.46	2054	10.00	15.20	0.020	57.4	Max Continuous Rating
5	24.02	1.74	39.56	1534	15.96	18.12	0.024	45.8	
6	24.02	2.15	49.16	1222	19.88	17.98	0.024	36.6	
7	24.03	2.59	59.55	913	23.94	16.18	0.022	27.2	
8	24.03	3.04	70.70	621	27.86	12.80	0.017	18.1	
9	24.04	3.51	82.69	319	31.98	7.55	0.010	9.1	
10	24.04	3.91	93.72	0	31.48	0.00	0.000	0.0	
	Special Load Points								
1									
2									

Sample Motor Test Data



This motor is intended for sampling and customer approval only. No application fitness approval is implied, as that can only be determined by the customer. These data represent performance of a single sample motor. These values are not to be construed as guaranteed values.

EXTERNAL CONTROL MODULE DATA SHEET

Description: The External Control Module simplifies the connection of an external motor drive to the Dynamo motor by providing the user with a standard set of hall signals, numerous encoder options, and a high current connector for the motor phase windings. The module is compatible with external motor drives using a 10 to 48Vdc power supply. The External Control Module provides a standard system for rotor position sensing required by many brushless motor drives. Three hall sensors spaced 120 electrical degrees apart, sense a magnetic disk, which is synchronized to the rotor of the motor. The hall signals can be used to provide inexpensive speed feedback to the motor drive, or for more precise control a wide array of integral two channel quadrature encoder options are available. The quadrature nature of an encoder allows the user to determine the direction of motor rotation as well as speed.

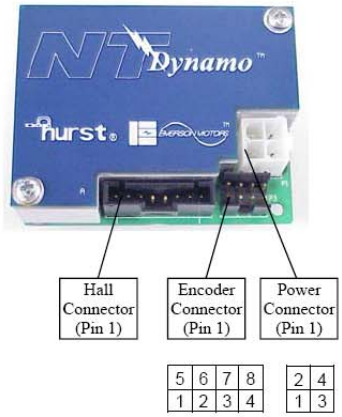
Environment: The NT Dynamo uses a TENV(totally enclosed non-ventilated) non-gasket construction. Installation and operating conditions should not exceed the recommended values for humidity and temperature. Contact the Hurst engineering department regarding any special installation issues you may have regarding vapors, oils or dust.

Storage Temp.: 32-158°F (0-70°C) **Humidity: 90% Max. Non-condensing** **Operating Temp.: 32-104°F (0-40°C)**

Power: Power to the motor windings is via the four pin connector. A regulated DC supply must be provided for the encoder and hall devices. Observe the correct polarity when making these connections. For maximum flexibility and noise immunity, the hall and encoder power supplies are separated. Excessive amounts of voltage ripple can cause shortened product life.

Motor Windings:	**Minimum DC Voltage: 10Vdc**	**Maximum DC Voltage: 48Vdc**
Encoder:	**Minimum DC Voltage: 4.75Vdc**	**Maximum DC Voltage: 5.25Vdc**
Halls:	**Minimum DC Voltage: 4.2Vdc**	**Maximum DC Voltage: 24Vdc**

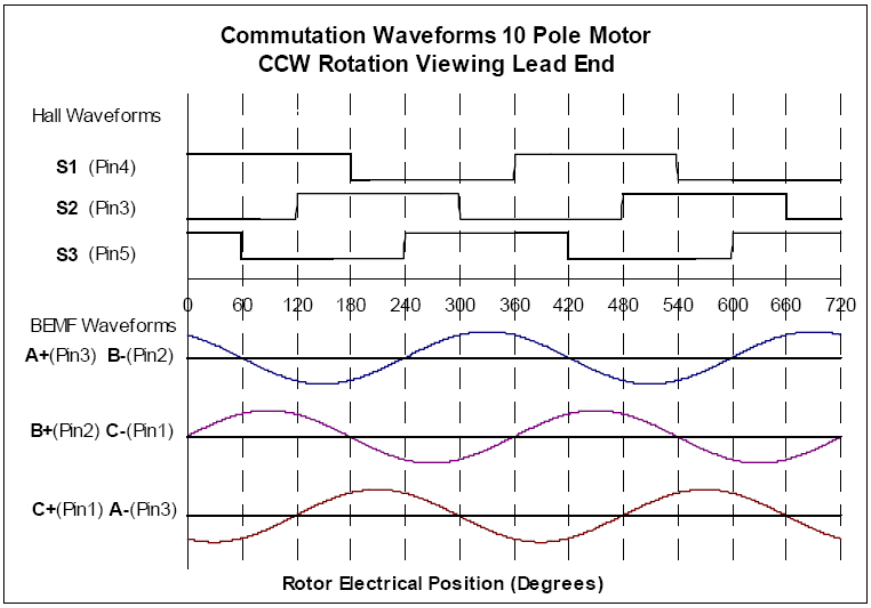
Connector	Pin #	Function	Mating Connector	Mating Terminal	Recommended Wire Size	Cable Length
Power	1	Phase C	Molex 39-01-2040	Molex 39-00-0039	22 AWG	30 ft. Max[1]
	2	Phase B				
	3	Phase A				
	4	Gnd				
Hall	1	V_s	Molex 50-57-9408	Molex 16-02-0103	22 AWG	30 ft. Max[1]
	2	$V_{s(RTN)}$				
	3	Hall S2				
	4	Hall S1				
	5	Hall S3				
	6	N/A				
	7	N/A				
	8	N/A				
Encoder	1	$+5V_s$	FCI 65846-016	FCI 48236-000	22 AWG	30 ft. Max[1]
	2	Encoder A				
	3	Encoder B				
	4	Encoder I				
	5	$+5V_{s(RTN)}$				
	6	Encoder /A				
	7	Encoder /B				
	8	Encoder /I				



Notes

1) Longer cable runs may require a larger wire size to maintain the correct input voltage level and a signal amplifier / conditioner to avoid erroneous signal values. For cable runs longer than 3 ft, shielded wire is recommended.

Commutation:



Encoder:
The drive may contain a optional shaft mounted optical encoder. The encoder outputs two or four quadrature signals from which direction and speed can be determined. These outputs can be used by an external drive to close the speed loop.

12 Anexo II: Transferencia de datos del modelo

12.1 corrents.m

```
% Carrega de dades de corrent
clear data taula
data=textread('corrents3f.csv' ,'%s' ,'headerlines',1,'delimiter',',' );
for i=1:length(data)
taula(floor((i-1)/6)+1,rem(i-1,6)+1)=data(i,1);
end
taula=str2double(taula);
angle=taula(:,1);
faseA=taula(:,2);
faseB=taula(:,3);
faseC=taula(:,4);
% Carrega de dades d'eficiencia
clear data taula
data=textread('efficiency.csv' ,'%s' ,'headerlines',1,'delimiter',',' );
for i=1:length(data)
taula(floor((i-1)/2)+1,rem(i-1,2)+1)=data(i,1);
end
taula=str2double(taula);
speed=taula(:,1);
efficiency=taula(:,2);
% Look-Up funcio sinus
fsin(1:181,1)=sin(pi/180*angle(1:181,1));
fsin(182:361,1)=-fsin(2:181,1);
angle_rad=angle*pi/180;

%%%%%%%%%%%%%%%% CREAR PERFILS CORRENT %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
close all
%--- Fase A CW-------------------------------------------------------------
clear tros A_CW
tros=faseA(1:find(angle==180)-1)
A_CW(1:find(angle==180)-1,1)=tros
A_CW(find(angle==180):length(angle)-1,1)=-tros(1:length(tros))
A_CW(length(angle),1)=tros(1,1)
%--- Fase C CW-------------------------------------------------------------
clear C_CW
desfase=(find(angle==180)-1)*2/3;
C_CW(desfase*2+1:length(angle),1)=tros(1:desfase+1)
C_CW(1:find(angle==desfase/2)-1,1)=tros(length(tros)-
desfase/2+1:length(tros))
C_CW(find(angle==desfase/2):desfase*2,1)=-tros(1:length(tros))
%--- Fase B CW-------------------------------------------------------------
clear B_CW
desfase=(find(angle==180)-1)*2/3;
B_CW(desfase+1:desfase*5/2,1)=tros(:,1)
B_CW(desfase*5/2+1:length(angle),1)=-tros(1:desfase/2+1)
B_CW(1:desfase,1)=-tros(desfase/2+1:length(tros))
%--- Fase A CCW-----------------------------------------------------------
A_CCW=-B_CW
A_CCW=flipud(A_CCW)
%--- Fase B CCW-----------------------------------------------------------
B_CCW=-A_CW
B_CCW=flipud(B_CCW)
%--- Fase C CCW-----------------------------------------------------------
C_CCW=-C_CW
C_CCW=flipud(C_CCW)
%--------------------------------------------------------------------------
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Código 12.1. Código que da forma a la corriente respecto la posición.

12.2 motor.m

```
J=0.00133764;
F=0.0000677;
R1=4.35084;
Ld=0.0341197;
Lq=0.0638516;
Lo=0;

Ke=0.790022;

global VDC
VDC=12;
```

Código 12.2. Características mecánicas y eléctricas del motor.

12.3 pi.m

```
 1 Ki=0.1;
 2 Kp=1;
 3 I_max=5;
```

Código 12.3. Parámetros del control PI.

12.4 trapez.m

```
 1 clear TRAP_A TRAP_B TRAP_C
 2 pendent=15; % en graus
 3 x_pend=find(angle==pendent);
 4
 5 %--- CÀLCUL DE LA FASE A ------------------------------------------------
 6 TRAP_A=zeros(length(angle)-1,1);
 7 % zona positiva
 8 TRAP_A(1:x_pend-1,1)=angle(2:x_pend)*VDC/pendent;
 9 TRAP_A((x_pend):find(angle==(120-pendent))-1,1)=VDC;
10 TRAP_A((find(angle==(120-pendent))):find(angle==120)-1,1)=(pendent-
angle(1:x_pend-
1))*VDC/pendent;
11 % zona negativa
12 TRAP_A(find(angle==180):find(angle==178+x_pend),1)=-
angle(2:x_pend)*VDC/pendent;
13 TRAP_A(find(angle==178+x_pend)+1:find(angle==(300-pendent))-1,1)=-VDC;
14 TRAP_A((find(angle==(300-pendent))):find(angle==300)-1,1)=(angle(1:x_pend-
1)-
pendent)*VDC/pendent;
15
16 %--- Càlcul de la fase B ------------------------------------------------
17 TRAP_B=zeros(length(angle)-1,1);
18 % zona positiva
19 TRAP_B(find(angle==240):find(angle==240+pendent)-
1,1)=angle(2:x_pend)*VDC/pendent;
20 TRAP_B(find(angle==240+pendent):find(angle==(360-pendent))-1,1)=VDC;
21 TRAP_B((find(angle==(360-pendent))):find(angle==360)-1,1)=(pendent-
angle(1:x_pend-
```

```
1))*VDC/pendent;
22 % zona negativa
23 TRAP_B(find(angle==60):find(angle==58+x_pend),1)=-
angle(2:x_pend)*VDC/pendent;
24 TRAP_B(find(angle==58+x_pend)+1:find(angle==(180-pendent))-1,1)=-VDC;
25 TRAP_B((find(angle==(180-pendent))):find(angle==180)-1,1)=(angle(1:x_pend-
1)-
pendent)*VDC/pendent;
26
27 %--- Càlcul de la fase C ---------------------------------------------
28 TRAP_C=zeros(length(angle)-1,1);
29 % zona positiva
30 TRAP_C(find(angle==120):find(angle==120+pendent)-
1,1)=angle(2:x_pend)*VDC/pendent;
31 TRAP_C(find(angle==120+pendent):find(angle==(240-pendent))-1,1)=VDC;
32 TRAP_C((find(angle==(240-pendent))):find(angle==240)-1,1)=(pendent-
angle(1:x_pend-
1))*VDC/pendent;
33 % zona negativa 1
34 TRAP_C(1:find(angle==(60-pendent))-1,1)=-VDC;
35 TRAP_C((find(angle==(60-pendent))):find(angle==60)-1,1)=(angle(1:x_pend-
1)-pendent)
*VDC/pendent;
36 % zona negativa 2
37 TRAP_C(find(angle==300):find(angle==298+x_pend),1)=-
angle(2:x_pend)*VDC/pendent;
38 TRAP_C(find(angle==298+x_pend)+1:find(angle==360)-1,1)=-VDC;
39
40
41 angle_rad_trap=angle_rad(1:360,1);
42
43
44 %------ PERFIL TENSIÓ RECTANGULAR ---------------------------------------
45 RECT_A=zeros(length(angle)-1,1);
46 RECT_A(1:find(angle==120)-1,1)=VDC;
47 RECT_A(find(angle==180):find(angle==300)-1)=-VDC;
48
49 RECT_B=zeros(length(angle)-1,1);
50 RECT_B(find(angle==240):find(angle==360)-1,1)=VDC;
51 RECT_B(find(angle==60):find(angle==180)-1)=-VDC;
52
53 RECT_C=zeros(length(angle)-1,1);
54 RECT_C(find(angle==120):find(angle==240)-1,1)=VDC;
55 RECT_C(1:find(angle==60)-1)=-VDC;
56 RECT_C(find(angle==300):find(angle==360)-1)=-VDC;
```

Código 12.4. Crea las formas trapezoidales para las BEMF.

12.5 inici.m

```
1 directori_act=cd;
2 cd c:\MATLAB7\work\BLDC\BLDC\laura\dades;
3
4 run corrents
5 run motor
6 run PI
7 run trapez
```

```
 8
 9 %eval('c:\MATLAB7\work\BLDC\BLDC\laura\dades\corrents');
10 % eval('c:\MATLAB7\work\BLDC\BLDC\laura\dades\motor');
11 % eval('c:\MATLAB7\work\BLDC\BLDC\laura\dades\PI');
12 % eval('c:\MATLAB7\work\BLDC\BLDC\laura\dades\trapez');
13
14 cd(directori_act);
15
16
17 msgbox('Les característiques del model del motor BLDC han estat carregades
correctament','Avís inicialització','help')
```

Código 12.5. Ejecuta los códigos anteriores.

13 Anexo III: Obtención del filtro digitalizado

El filtro se ha digitalizado y evaluado mediante un breve código *.m para *Matlab*. Este código se detalla a continuación.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% FILTRO RC DISCRETIZADO                                                       %
%                                                                             %
% archivo:     filtro.m                                                       %
% descripción: Este código obtiene la función de transferencia del            %
%              filtro digitalizado. Además realiza un muestreo a partir       %
%              de los datos obtenidos con el osciloscopio y realiza el        %
%              filtrado de estas muestras                                     %
%                                                                             %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

H_RC=tf([1],[0.000227 1])       % filtro RC en dominio de Laplace
H_RC_z=c2d(H_RC,1/49152,'zoh' ) % transformada z del filtro RC

[num,den]=tfdata(H_RC_z);       % numerador y denominador H(z)
num=num{1,1}
den=den{1,1}
datos=load('mostreig pel filtre.txt');
                                % carga de datos del osciloscopio

%~ Emulación del muestreo eligiendo datos del osciloscopio ~~~~~~~~~~~~~~~~
diferencias=diff(datos(:,1));
tiempo_entre_muestras=(round(diferencias(1,1)*1e10))/1e10
ts_filtro=1/49152
ciclo_muestras=ceil(ts_filtro/tiempo_entre_muestras)
longitud=floor(length(datos)/21)
entrada=zeros(longitud,1);
tiempo=zeros(longitud,1);
for i=1:longitud
entrada(i,1)=datos((i-1)*ciclo_muestras+1,2);
tiempo(i,1)=datos((i-1)*ciclo_muestras+1,2)-datos(1,1);
end
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

[salida,zf]=filter(num,den,entrada);
                                % filtrado de las muestras
figure(1)                       % representación de los resultados
plot(entrada)
hold on
plot(salida,'r')
legend('señal sin filtrar','señal filtrada')
hold off
xlabel('# de muestra' )
ylabel('tensión [V]' )
```

Código 13.1. Código para la evaluación del filtro RC discretizado.

14 Anexo IV: Adaptación de señales para el conversor A/D

El circuito usado para la adaptación de señales para el conversor analógico-digital es el siguiente

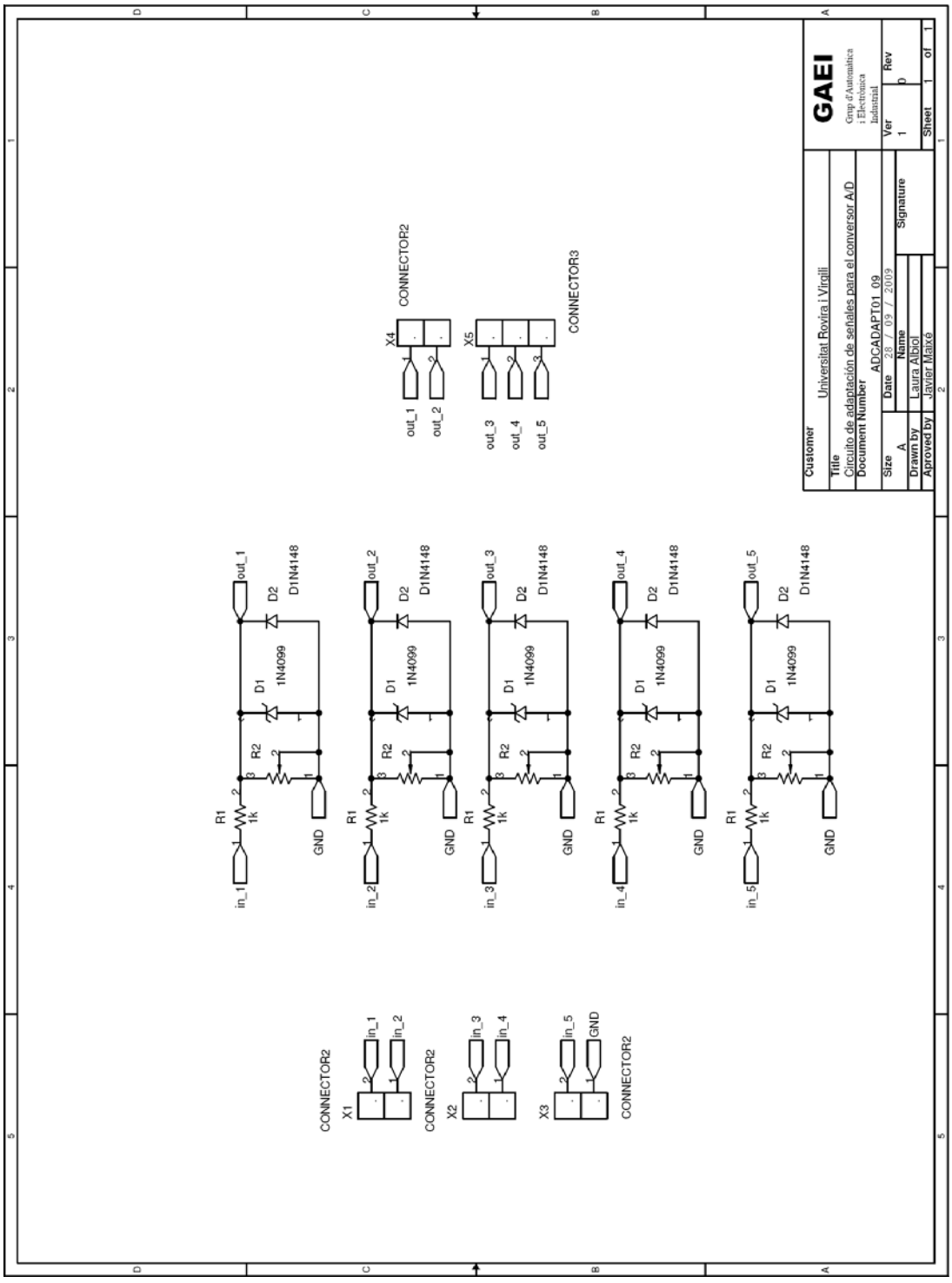


Figura 14.1. Circuito de adaptación para el conversor A/D.

A partir del diseño anterior se ha realizado el trazado la placa de circuito impreso.

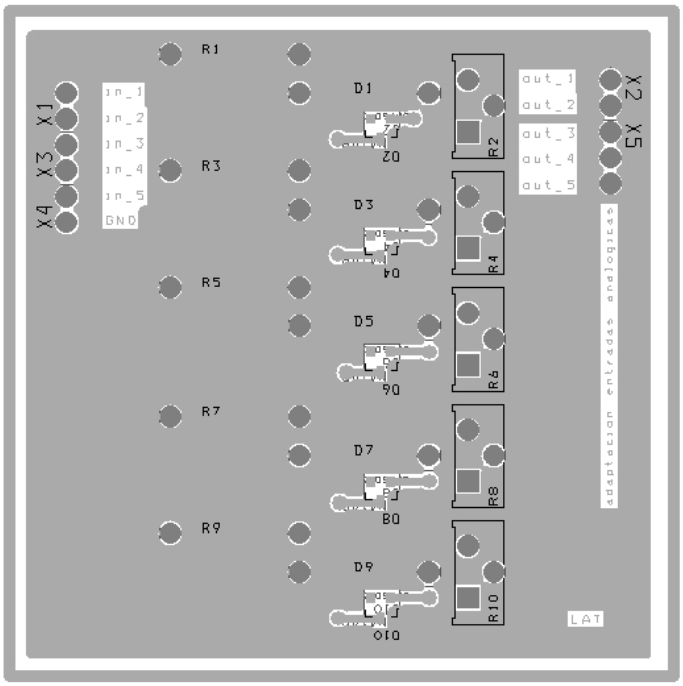


Figura 14.2. Cara anterior de la placa de circuito impreso.

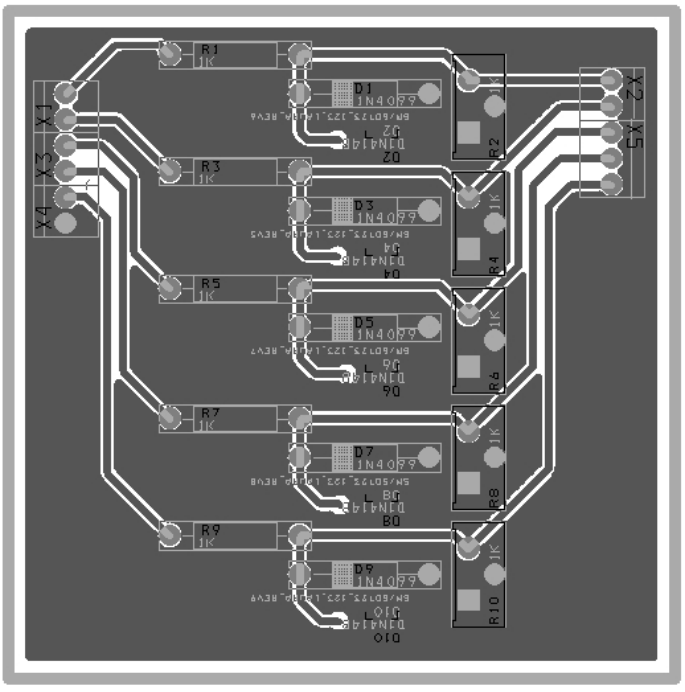


Figura 14.3. Cara posterior de la placa de circuito impreso.

Finalmente, una vez soldados todos los componentes, el circuito obtenido es el mostrado a continuación.

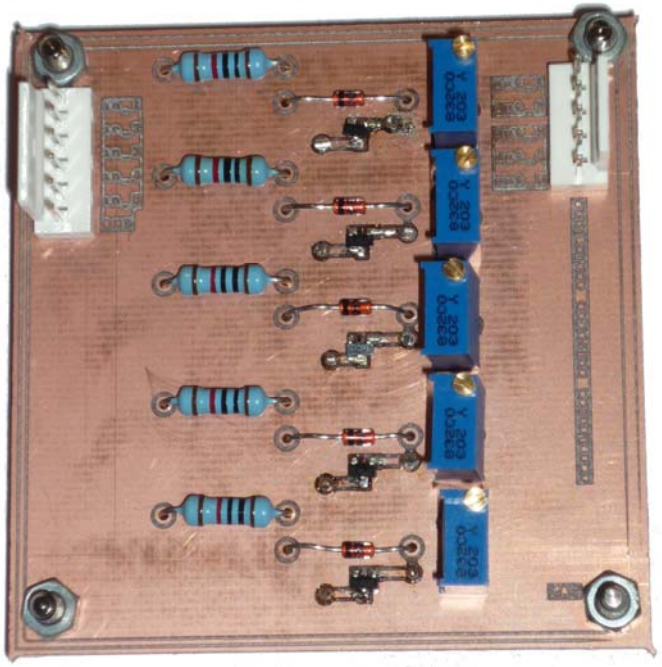


Figura 14.4. Montaje finalizado de la etapa de adaptación de señales.

15 Anexo V: Código

15.1 filtro_digital.s

```
; ..............................................................................
;    File   filtro_digital.s
; ..............................................................................

          .equ filtro_digitalNumSections, 3

; ..............................................................................
;
; Allocate and initialize filter coefficients
;
; These coefficients have been designed for use in the Transpose filter only

          .section data, xmemory
          .global _filtro_digitalCoefs

_filtro_digitalCoefs:
.hword    0x0544      ; b( 1,0)/2
.hword    0x0A89      ; b( 1,1)/2
.hword    0x3E25      ; a( 1,1)/2
.hword    0x0544      ; b( 1,2)/2
.hword    0xECC8      ; a( 1,2)/2
.hword    0x0584      ; b( 2,0)/2
.hword    0x0B08      ; b( 2,1)/2
.hword    0x4F4A      ; a( 2,1)/2
.hword    0x0584      ; b( 2,2)/2
.hword    0xD5D3      ; a( 2,2)/2
.hword    0x1581      ; b( 3,0)/2
.hword    0x1581      ; b( 3,1)/2
.hword    0x1CB4      ; a( 3,1)/2
.hword    0x0000      ; b( 3,2)/2
.hword    0x0000      ; a( 3,2)/2

; ..............................................................................
; Allocate states buffers in (uninitialized) Y data space

          .section b,bss,ymemory

BEMF_PhaseA_States1:
          .space filtro_digitalNumSections*2

BEMF_PhaseA_States2:
          .space filtro_digitalNumSections*2

BEMF_PhaseB_States1:
          .space filtro_digitalNumSections*2

BEMF_PhaseB_States2:
          .space filtro_digitalNumSections*2
```

```
BEMF_PhaseC_States1:
          .space filtro_digitalNumSections*2

BEMF_PhaseC_States2:
          .space filtro_digitalNumSections*2

;; .....................................................................
;; Allocate and intialize filter structure
;
;         .section .data
;         .global _filtro_digitalFilter
;
; .....................................................................
; Allocate and intialize filter structure

          .section .data
          .global _BEMF_phaseA_Filter
          .global _BEMF_phaseB_Filter
          .global _BEMF_phaseC_Filter

_BEMF_phaseA_Filter:
.hword filtro_digitalNumSections-1
.hword _filtro_digitalCoefs
.hword 0xFF00
.hword BEMF_PhaseA_States1
.hword BEMF_PhaseA_States2
.hword 0x0000

_BEMF_phaseB_Filter:
.hword filtro_digitalNumSections-1
.hword _filtro_digitalCoefs
.hword 0xFF00
.hword BEMF_PhaseB_States1
.hword BEMF_PhaseB_States2
.hword 0x0000

_BEMF_phaseC_Filter:
.hword filtro_digitalNumSections-1
.hword _filtro_digitalCoefs
.hword 0xFF00
.hword BEMF_PhaseC_States1
.hword BEMF_PhaseC_States2
.hword 0x0000
```

15.2 IIRT_filter.s

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;
; FILE:    IIRT_Filter.s
;
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

```
; offsets into IIRTransposeFilter structure
; (entries are word wide -- hence +=2)

                .equ oNumSectionsLess1, 0       ; number of second order sections
- 1
                .equ oCoefs,            2       ; pointer to coefficients
                                                ;    number of coefficients is
                                                ;     5 * number of second order
sections
                .equ oPSVpage,           4       ; page number in program memory
if
                                                ;     coefficients are in program
memory
                                                ;    0xFF00 if not
                .equ oStates1,          6       ; pointer to state variable 1
                                                ;    one word for every section
                .equ oStates2,          8       ; pointer to state variable 1
                                                ;    one word for every section
                .equ oFinalShift,      10       ; final left shift count
                                                ;    restores filter gain to 0 dB
                                                ;    shift count may be zero
                                                ;       if  not  zero,  it  is  the
number of bits
                                                ;     to shift the output to the
left

; ......................................................................

; ......................................................................



; ----------------------------------------------------------------------------

; ----------------------------------------------------------------------------
; Block Cascaded Transpose IIR Implementation (cascade form I)
; This file should be assembled and linked against filter coefficients
; generated by dsPicFD -- filter design software by Momentum Data Systems.
;
; Module Re-entrancy:
;     Module re-entrancy is not supported
;
; Input:  for routine      _BlockIIRTransposeFilter
;
;        w0 = pointer to filter structure
;        w1 = pointer to input  sample buffer
;        w2 = pointer to output sample buffer
;        w3 = number of output samples to generate
;
;
; System Resource usage
;       W0                      used but value on exit is same as entry
;       W1, W2, W3              used not restored
;       W4, W5, W6, W7          used not restored
;       w8, W9, w10,W11         used, saved and restored
;       accumulators a,b        used not restored
;       CORCON                  used, saved and restored
;       PSVPAG                  used, saved and restored
```

```
;
;
; Input: for _IIRTransposeFilterInit
;        w0 = pointer to filter structure
;
; System Resource usage
;       w0, w1, W2              used not restored
; ----------------------------------------------------------------------------
; ----------------------------------------------------------------------------
;
; DO and REPEAT instruction usage
;     2 level DO instruction
;     REPEAT instruction is not used
;
;
; Module Program Memory Size
;     _BlockIIRTransposeFilter:  49
;     _IIRTransposeFilterInit:   8
;
;
; Module Cycle Count
;     _BlockIIRTransposeFilter:  27 + N*(11 + S*11)      (+2 if PSV)
;     _IIRTransposeFilterInit:   8 + S*2
; with N: number of samples per block, S: number of sections in filter.
;
; ----------------------------------------------------------------------------
; ----------------------------------------------------------------------------

         .text
         .global _BlockIIRTransposeFilter

_BlockIIRTransposeFilter:
         PUSH  w8                        ; save context of w8
         PUSH  w9                        ; save context of w9
         PUSH  w10                       ; save context of W10
         PUSH  w11                       ; save context of w11
         PUSH  CORCON                    ; save context of CORCON
         PUSH  PSVPAG                    ; save context of PSVPAG

; Check if filter coefficients are stored in Program Space or Data Space and
enable
; PSV and set up PSVPAG accordingly

         MOV   [w0+oPSVpage], W10
         MOV   #0xFF00, W8
         CP     W8, W10             ; perform w8-w10 (if w10 = FF00 then do not
enable PSV)
         BRA    Z, no_psv           ; branch if compare true (coefficients are in
data space)
         MOV   #0x00F4,W8          ; Enable enable Program Space Visibility,
                                   ; Accumulator A Saturation and Data Space write
Saturation
                                   ;    as bits 2, 5 and 7 are set in CORCON
                                   ; Also note bits 2, 5 and 7 are in low order
byte which is
                                   ;   the first byte of a 2 byte word in a little
endian
                                   ;   processor such as the dsPIC30
```

```
                                ; also, enable unbiased (convergent) rounding
mode,
                                ; 1.39 saturation enabled, fractional mult.
        MOV    w10, PSVPAG       ; PSVPAG = Program Space page containing filter
taps
        BRA    SetupPointers

no_psv:
        MOV    #0x00F0,w8        ; Enable Accumulator A Saturation and
                                 ; Data Space write Saturation
                                 ;    as bits 5 and 7 are set in CORCON
; ....................................................................;
; Setup pointers

SetupPointers:
        MOV    W8, CORCON                ; set PSV and saturation options

        MOV [w0+oNumSectionsLess1], w4   ; w4 = number of sections - 1
        MOV [w0+oFinalShift], w9         ; w9 = final shift count
        DEC w3, w3                       ; w3 = number of output samples -1

        DO w3, transposeBlockLoop        ; loop on the number of input samples

        MOV [w0+oCoefs], w8              ; w8 = base address of coefs
        MOV [w1++], w6                   ; w6 = next input sample

        MOV [w0+oStates1], w10           ; w10 = base address of states1 buffer
        MOV [w0+oStates2], w11           ; w11 = base address of states2 buffer
        MOV [w8++], w5                   ; fetch first coefficient
        LAC [w10], #1, a                 ; fetch filter state

        DO   w4, transposeSectionLoop     ; loop on the number of second order
sections

        MAC w5*w6, a, [w8]+=2, w5
        LAC [w11], #1, b
        SAC.R a, #-1, w7
        MAC w5*w6, b, [w8]+=2, w5
        MAC w5*w7, b, [w8]+=2, w5
        SAC.R b, #-1, [w10++]
        MPY w5*w6, b, [w8]+=2, w5
        SAC.R a, #-1, w6
        LAC [w10], #1, a
        MAC w5*w7, b, [w8]+=2, w5
transposeSectionLoop:
        SAC.R b, #-1, [w11++]

        LAC w6, a
        SFTAC a, w9                      ; perform arithmetic shift
transposeBlockLoop:
        SAC.R a, [w2++]                      ; round and store result in output
buffer

        POP  PSVPAG                      ; restore context of PSVPAG
        POP  CORCON                      ; restore context of CORCON
        POP  w11                         ; restore context of w11
        POP  w10                         ; restore context of W10
        POP  w9                          ; restore context of w9
        POP  w8                          ; restore context of w8
```

```
        RETURN                          ; exit from _BlockIIRTransposeFilter:

; ------------------------------------------------------------------------

; Input:
;        w0 = pointer to filter structure

        .text
        .global _IIRTransposeFilterInit

_IIRTransposeFilterInit:
        MOV [w0+oStates1], w1           ; w1 = base address of states1 buffer
        MOV [w0+oStates2], w2           ; w2 = base address of states2 buffer
        MOV [w0+oNumSectionsLess1], w0  ; w0 = number of sections - 1

; initialize state buffers (i.e. fill with zeros)
        DO w0, transposeInitLoop
        CLR [w1++]
transposeInitLoop:
        CLR [w2++]

        RETURN                          ; exit from _IIRTransposeFilterInit:

; ------------------------------------------------------------------------
```

15.3 main.c

```
#include "p30f6010a.h"
#include "defs.h"
#include "vars.h"
#include "comm.h"
#include "med_ev.h"
#include "in_PWM.h"
#include "IIR_filter.h"
#include "filtro_digital.h"
#include "funcs_AD.h"
#include "ini_tmrs.h"

// Setup Configuration bits
_FOSC(CSW_FSCM_OFF & XT_PLL16);
_FWDT(WDT_OFF);
_FBORPOR(PBOR_ON & BORV_45 & PWRT_64 & MCLR_EN);
_FBS(NO_BOOT_CODE);
_FSS(NO_SEC_CODE);
_FGS (GEN_PROT);  // disable all code protection

void cruce (void);

//=============================================================================
void control_PID (void)
{
    if (etapa_arranque==3)
    {
          PID_error[2]=PID_error[1];
          PID_error[1]=PID_error[0];
```

```
        PID_error[0]=vel_pond-t_consigna;


    inc_dc+=K1*(float)(PID_error[0])+K2*(float)(PID_error[1])+K3*(float)(PID_erro
r[2]);
        diag=0;
        inc_dc_aux=((int)(inc_dc));
        if (inc_dc_aux>294)
        {
              inc_dc_aux=294;
        }
        if (inc_dc_aux<-294)
        {
              inc_dc_aux=-294;
        }

        if (inc_dc>0.0)
        {
              PID_dc+=inc_dc_aux;
              if (PID_dc>FULL_DUTY)
              {
                    PID_dc=FULL_DUTY;
              }
        }
        else if (inc_dc<0.0)
        {
              PID_dc+=inc_dc_aux;
              if (PID_dc<0x0520)
              {
                    PID_dc=0x0520;
              }
        }
        PDC1=PID_dc;
        PDC2=PID_dc;
        PDC3=PID_dc;
        inc_dc-=(int)(inc_dc);
    }
}
//=============================================================================
void __attribute__((__interrupt__)) _T3Interrupt( void )
{
    T3CONbits.TON=0;
    TMR3=0;
    IFS0bits.T3IF=0;
    IEC0bits.T3IE=0;

    conmutar(Sector_sig);
    blanking_count=BLANK_CT;
    esp_cruce++;
    if(esp_cruce > 5)
    {
          esp_cruce=0;
    }
}
//=============================================================================
void __attribute__((__interrupt__)) _T1Interrupt( void )
{
    LED9=1;
```

```
    IFS0bits.T1IF=0;

}
//=============================================================================
void __attribute__((__interrupt__)) _T2Interrupt( void )
{
    LED9=1;
    IFS0bits.T2IF=0;
}

//=============================================================================
// Occurs every 50us or at a rate of 20kHz
void __attribute__((__interrupt__)) _PWMInterrupt( void )
{
    IFS2bits.PWMIF=0;
    if (++MediumEventCounter >= med_ev_period)  // Fire Medium event every 100 us
    {
          MediumEventCounter = 0;
          ControlFlags.MediumEventFlag = 1;
    }
}
//=============================================================================
void __attribute__((__interrupt__)) _ADCInterrupt( void )
{
    TMR1=0;
    v1=ADCBUF0;
    v2=ADCBUF1;
    v3=ADCBUF2;
    v4=ADCBUF3;
    filtre();
    blanking_count--;
    if (blanking_count<=0)
    {
          v1_filtered=v1;
          v2_filtered=v2;
          v3_filtered=v3;
    }

    if ((etapa_arranque==3)&&(TMR2>20))
    {
          cruce();
    }
    IFS0bits.ADIF=0;
}
//=============================================================================
int main ( void )
{
    int i;

    TRISAbits.TRISA9 = 0;    // enable control of the D6 LED
    TRISAbits.TRISA10 = 0;   // enable control of the D7 LED
    TRISAbits.TRISA14 = 0;   // enable control of the D8 LED
    TRISAbits.TRISA15 = 0;   // enable control of the D9 LED

    LATD  = 0;
    TRISD = 0xFFFF;
    // RD7 output for D5 LED drive
    TRISDbits.TRISD7 = 0;
```

```
    // RD11 output for PWM buffer enable.
    TRISDbits.TRISD11 = 0;
    // LCD enable pin
    TRISDbits.TRISD13 = 0;
    TRISDbits.TRISD15 = 0;

    TRISF = 0xFFFF;

    TRISEbits.TRISE9=0;
    LATEbits.LATE9=1;
    LATEbits.LATE9=1;
    for(i=0;i<10;i++)
    {
          LATEbits.LATE9=0;
    }

    TRISFbits.TRISF4=0;
    PORTFbits.RF4=0;

    acc=0;
    dc_rampa=SEC2_DC;

    cnt=VOLTA1_COUNT;
    j=1;
    k=1;
    dc=1914+73*1;
    trobat=0;
    med_ev_period=2;
    ini_AD();
    initPWM();                          // initialize the PWM module
    ini_timers();
    T3CONbits.TON=0;
    IEC0bits.T3IE=0;
    IFS0bits.T3IF=0;
    tiempo_60_elec[0]=0x0020;
    tiempo_60_elec[1]=0x0020;
    tiempo_60_elec[2]=0x0020;
    tiempo_60_elec[3]=0x0020;
    tiempo_60_elec[4]=0x0020;
    tiempo_60_elec[5]=0x0020;
    vel_pond_aux[0]=35;
    vel_pond_aux[1]=35;
    vel_pond_aux[2]=35;
    vel_pond_aux[3]=35;
    while(1)
    {
          if(ControlFlags.MediumEventFlag)
          {
                MediumEvent();                      //  execute  start-up  ramp  or
speed control loop
          }
          if(control_pendent)
          {
                control_pendent=0;

    suma_vels=(vel_pond_aux[1]+vel_pond_aux[2]+vel_pond_aux[3]+vel_pond_aux[0]+ve
l_pond_aux[4]+vel_pond_aux[5]);
                //vel_pond=suma_vels>>2;
```

```
        //
    suma_vels=(vel_pond_aux[1]>>2+vel_pond_aux[2]>>2+vel_pond_aux[3]>>2+vel_pond_
aux[0]>>2);
                vel_pond_aux_med=__builtin_divud(suma_vels,6);
                for (ind_pond_aux=0;i>6;i++)
                {
                      if ((vel_pond_aux[0]+100)<vel_pond_aux_med)
                      {
                            vel_pond_aux[0]=vel_pond_aux_med;
                      }
                }

    suma_vels=(vel_pond_aux[1]+vel_pond_aux[2]+vel_pond_aux[3]+vel_pond_aux[0]+ve
l_pond_aux[4]+vel_pond_aux[5]);
                vel_pond=suma_vels;//__builtin_divud(suma_vels,6);
                control_PID();
          }

    }
}
//=============================================================================
void cruce (void)
{
    v_media=v4>>1;
    n_cruces++;
    if (esp_cruce==4)
    {
          if(v2_filtered<v_media)
          {
                Sector_sig=5;
                tiempo_60_elec[4]=TMR1;
                TMR1=0;
                t_prova[4]=TMR2;
                TMR2=0;


    t_total=t_prova[0]+t_prova[1]+t_prova[2]+t_prova[3]+t_prova[4]+t_prova[5];

    t_total2=(t_prova[0]<<1)+(t_prova[1]<<1)+(t_prova[2]<<1)+(t_prova[3]<<1)+(t_p
rova[4]<<1)+(t_prova[5]<<1);

                carrega=__builtin_divud(t_total2,3);
                PR3=carrega-1;
                if (PR3<0x0002)
                {
                      PR3=0x0002;
                }
                T3CONbits.TON=1;
                IEC0bits.T3IE=1;


                vel_pond_aux[ind_pond]=__builtin_divud(t_total,6);
                ind_pond++;
                if (ind_pond>5)
                {
                      ind_pond=0;
                      control_pendent=1;
                }
                TMR2=0;
```

```
      }
}
else if (esp_cruce==5)
{
      if(v3_filtered>v_media)
      {
            Sector_sig=0;
            tiempo_60_elec[5]=TMR1;
            TMR1=0;
            t_prova[5]=TMR2;
            TMR2=0;

            PR3=carrega-1;
            if (PR3<0x0002)
            {
                  PR3=0x0002;
            }
            T3CONbits.TON=1;
            IEC0bits.T3IE=1;
      }
}
else if (esp_cruce==0)
{
      if(v1_filtered<v_media)
      {
            Sector_sig=1;
            tiempo_60_elec[0]=TMR1;
            TMR1=0;
            t_prova[0]=TMR2;
            TMR2=0;

            PR3=carrega-1;
            if (PR3<0x0002)
            {
                  PR3=0x0002;
            }
            T3CONbits.TON=1;
            IEC0bits.T3IE=1;
      }
}
else if (esp_cruce==1)
{
      if(v2_filtered>v_media)
      {
            Sector_sig=2;
            tiempo_60_elec[1]=TMR1;
            TMR1=0;
            t_prova[1]=TMR2;
            TMR2=0;

            PR3=carrega-1;
            if (PR3<0x0002)
            {
                  PR3=0x0002;
            }
            T3CONbits.TON=1;
            IEC0bits.T3IE=1;
      }
```

```
    }
    else if (esp_cruce==2)
    {
          if(v3_filtered<v_media)
          {
                Sector_sig=3;
                tiempo_60_elec[2]=TMR1;
                TMR1=0;
                t_prova[2]=TMR2;
                TMR2=0;

                PR3=carrega-1;
                if (PR3<0x0002)
                {
                      PR3=0x0002;
                }
                T3CONbits.TON=1;
                IEC0bits.T3IE=1;
          }
    }
    else if (esp_cruce==3)
    {
          if(v1_filtered>v_media)
          {
                Sector_sig=4;
                tiempo_60_elec[3]=TMR1;
                TMR1=0;
                t_prova[3]=TMR2;
                TMR2=0;

                PR3=carrega-1;
                if (PR3<0x0002)
                {
                      PR3=0x0002;
                }
                T3CONbits.TON=1;
                IEC0bits.T3IE=1;
          }
    }
}
```

15.4 comm.h

```
void conmutar(unsigned int sector)
{
    OVDCON = SectortoState[sector];                 // Change PWM phase
}
```

15.5 defs.h

```
// LED assignments
```

```
#define LED5    LATDbits.LATD7  //D5 on PCB
#define LED6    LATAbits.LATA9    //D6 on PCB
#define LED7    LATAbits.LATA10   //D7 on PCB
#define LED8    LATAbits.LATA14   //D8 on PCB
#define LED9    LATAbits.LATA15   //D9 on PCB

#define         FCY         29490000UL
#define         FPWM  20000
#define   PERIOD      FCY/FPWM
#define         DEAD_TIME   .000002          // in seconds
#define         DT_COUNT    DEAD_TIME*FCY    // in PWM counts

#define   FULL_DUTY (2*FCY/FPWM)

#define SEC1_ARR 0
#define SEC2_ARR 1
#define RAMP_1 2


#define SEC1_DC_100 40 //(en %)
#define SEC2_DC_100 55 // en
#define SEC1_DC (unsigned int)((unsigned long)SEC1_DC_100)*FULL_DUTY/100
#define SEC2_DC (unsigned int)((unsigned long)SEC2_DC_100)*FULL_DUTY/100
#define PASOS_SEC1 1500 //6000 pasos de 100 us -> 600 ms
#define PASOS_SEC2 4000 //6000 pasos de 100 us -> 600 ms
#define PASOS_X_INCR_1     PASOS_SEC1/20
#define PASOS_X_INCR_2     PASOS_SEC2/20

// PWM Fault protection options
// Load FLTACON below with one of thes options
#define CYCLE_BY_CYCLE_PROTECTION 0x0087;          //  PWM1L/H,  PWM2L/H,  PWM3L/H
have cycle by cycle current limiting enabled
#define FAULT_CAUSES_PWM_SHUTDOWN 0x0007;         // An over-current fault shuts
down PWM1L/H, PWM2L/H, PWM3L/H
#define NO_FAULT_PROTECTION       0x0000;       // No over-current protection

#define         STATE0      0x0600            // A top, B bottom, C BEMF
#define         STATE1      0x1200            // A top, C bottom, B BEMF
#define         STATE2      0x1800            // B top, C bottom, A BEMF
#define         STATE3      0x0900            // B top, A bottom, C BEMF
#define         STATE4      0x2100            // C top, A bottom, B BEMF
#define   STATE5  0x2400          // C top, B bottom, A BEMF

#define VOLTA1_COUNT 500

#define KP      0.007
#define KI      0.0012
#define KD      0.0

#define BLANK_CT 10
```

15.6 filtro_digital.h

```
#ifndef FILTRO_DIGITAL_H
```

```
#define FILTRO_DIGITAL_H

extern IIRTransposeFilter BEMF_phaseA_Filter;
extern IIRTransposeFilter BEMF_phaseB_Filter;
extern IIRTransposeFilter BEMF_phaseC_Filter;
extern unsigned int filtro_digitalCoefs;

#endif /* FILTRO_DIGITAL_H */

/* The following C-code fragment demonstrates how to call the filter routine
#include "IIR_Filter.h"
#include "filtro_digital.h"

// NUM_SAMPLES defines the number of samples in one block of input data.
// This value should be changed as needed for the application
#define NUM_SAMPLES 100

{
   // Declare input and output sample arrays.
   int  inSamples[NUM_SAMPLES], outSamples[NUM_SAMPLES];


   // Call the IIRTransposeFilterInit routine to zero out the state variables
   IIRTransposeFilterInit( &filtro_digitalFilter );

   // Call BlockIIRTransposeFilter for each block of input samples
   // This routine would normally be called inside a FOR or a DO-WHILE loop
   // Only one instance has been shown
   BlockIIRTransposeFilter( &filtro_digitalFilter, &inSamples[0], &outSamples[0],
NUM_SAMPLES );
}
*/
```

15.7 funcs_AD.h

```
void ini_AD (void)
{
          ADCON1bits.ADON=0;

   TRISBbits.TRISB11=1;
   TRISBbits.TRISB12=1;
   TRISBbits.TRISB13=1;
   TRISBbits.TRISB14=1;

   ADPCFG=0x87FF; // modificat
   ADCON1=0x00E0; // no cal modificar
   ADCHS=0x0000;  // modificat
   ADCSSL=0x7800; // modificat
   ADCON2=0x040C; // modificat
   ADCON3=0x1E06; // modificat

   IPC2bits.ADIP=0b111;
   IEC0bits.ADIE=1;
   ADCON1bits.ADON=1;
   ADCON1bits.ASAM=1;
```

```
    IIRTransposeFilterInit(&BEMF_phaseA_Filter);
    IIRTransposeFilterInit(&BEMF_phaseB_Filter);
    IIRTransposeFilterInit(&BEMF_phaseC_Filter);
}

void filtre(void)
{
    BlockIIRTransposeFilter(&BEMF_phaseA_Filter,&v1,&v1_filtered,1);
    BlockIIRTransposeFilter(&BEMF_phaseB_Filter,&v2,&v2_filtered,1);
    BlockIIRTransposeFilter(&BEMF_phaseC_Filter,&v3,&v3_filtered,1);
}
```

15.8 IIR_filter.h

```
#ifndef MDS_IIR_H
#define MDS_IIR_H

/* ..................................................................... */

typedef struct
{
  int  numSectionsLess1;
  int *pCoefs;
  int  psvpage;
  int *pStates;
  int  initialGain;
  int  finalShift;
} IIRCanonicFilter;

typedef struct
{
  int  numSectionsLess1;
  int *pCoefs;
  int  psvpage;
  int *pStates1;
  int *pStates2;
  int  finalShift;
} IIRTransposeFilter;

extern void BlockIIRCanonicFilter( IIRCanonicFilter *, int *, int *, int );
extern void IIRCanonicFilterInit( IIRCanonicFilter *pFilter );

extern void BlockIIRTransposeFilter( IIRTransposeFilter *, int *, int *, int );
extern void IIRTransposeFilterInit( IIRTransposeFilter *);

/* ..................................................................... */

#endif /* MDS_IIR_H */
```

15.9 in_PWM.h

```
void initPWM(void)
```

```
{
    OVDCON = 0;
    PTPER = PERIOD;         // set PWM period
    PWMCON1 = 0x0777; // Enable PWM channels
    DTCON1 = DT_COUNT;      // 2 us deadtime
    FLTACON = NO_FAULT_PROTECTION;      //  PWM1L/H,  PWM2L/H,  PWM3L/H  have  no
current limiting enabled -- it's taken care of in hardware
    PDC1 = 0;               // Set all PWM duty cycles initially to zero
    PDC2 = 0;
    PDC3 = 0;
    SEVTCMP = 0;            // no special event trigger
    PTCON = 0x8000;         //
    IFS2bits.PWMIF = 0;     // enable the PWM interrupt
    IEC2bits.PWMIE = 1;
}
```

15.10 ini_tmrs.h

```
void ini_timers (void)
{
    T1CON=0x0010;

    T3CON=0x0010; // prescaler 1:8
    TMR3=0;
    IFS0bits.T3IF=0;  // deshabilitar la interrupción
    IEC0bits.T3IE=0;  //


    T2CON=0x0020;
}
```

15.11 med_ev.h

```
void MediumEvent(void)
{
    {
          PWMCON2bits.UDIS = 1;              // disable the duty cycle update
          switch (etapa_arranque)
          {
                case SEC1_ARR:
                      paso_arranque++;
                      PDC1 = SEC1_DC+0x001D*(int)(PASOS_X_INCR_1/paso_arranque);
                //    PORTAbits.RA14=!PORTAbits.RA14;
                      PDC2 = PDC1;
                      PDC3 = PDC1;
                      if (--cont_sec1 == 0)
                      {
                            Sector--;                                  //  Increment
Sector (there are 6 total)
                            if(Sector < 0) Sector = 5;
                            conmutar(Sector);             // Change the PWM output
for next sector
                            etapa_arranque++;
```

```
                        paso_arranque=0;
                        LED6=1;
                  }
                  break;
            case SEC2_ARR:
                  paso_arranque++;
                  PDC1 = SEC2_DC+0x001D*(int)(PASOS_X_INCR_2/paso_arranque);
                  //LED8=!LED8;
                  PDC2 = PDC1;
                  PDC3 = PDC1;
                  if (--cont_sec2 == 0)
                  {
                        Sector--;                                    //  Increment
Sector (there are 6 total)
                        if(Sector <0) Sector = 5;
                        conmutar(Sector);             // Change the PWM output
for next sector
                        etapa_arranque++;
                  //    dc_rampa=SEC2_DC;
                        PDC1=dc_rampa;
                  //    LED8=!LED8;
                        PDC2=dc_rampa;
                        PDC3=dc_rampa;

                        paso_arranque=0;
                        LED7=1;
//                      T1CON=T1CON|0x8000;

                  }
                  break;
            case RAMP_1:
                  if (--cnt==0)
                  {
                        k++;
                        Sector++;
                        if (Sector>5)
                        {
                              Sector=0;
                        }
                        conmutar(Sector);
                        dc=dc-73;
                        if (k>6)
                        {
                              k=1;
                              dc=dc+438;
                              j++;
                              if(j>25)
                              {
                                    j=25;
                                    en_trobat=0;
                              //    etapa_arranque++;
                                    med_ev_period=20;
                                    T3CONbits.TON=0;
                                    IEC0bits.T3IE=0;
                                    IFS0bits.T1IF=0;
                                    IEC0bits.T1IE=1;
                                    IEC0bits.T2IE=1;
                                    T1CONbits.TON=1;
```

```
                                    T2CONbits.TON=1;
                                    TMR1=0;
                                    TMR2=0;
                                    etapa_arranque++;
                        //          dc=0x04E8;//bo
                              dc=0x0530;
                                    //PORTFbits.RF4=1;
                        //    dc=0x0650;
                        //    dc=0x0520;
                        //          dc=0x0620;
                        //    dc=0x06E8;
                              PID_dc=dc;
                                    }
                              }
                              cnt=VOLTA1_COUNT/j;
                              PDC1=dc;
                              PDC2=dc;
                              PDC3=dc;
    //                        PDC4=dc;
                        //    PID_dc=dc;
                        //    conmutar(Sector);
                        }

                              break;
                        case 3:
                        //    PDC1=0;
                        //    PDC2=0;
                        //    PDC3=0;
                              break;
                  }

                  PWMCON2bits.UDIS = 0;               // enable the duty cycle update
                  ControlFlags.MediumEventFlag = 0;
    }
}
```

15.12 vars.h

```
unsigned int paso_arranque=0;
unsigned int etapa_arranque = 0;
unsigned int MediumEventCounter = 0;

unsigned int SectortoState[6] = {STATE0,STATE1,STATE2,STATE3,STATE4,STATE5};

int Sector = 0;                   // Sector (0 - 5)


int en_trobat=1;
int count_trobat=0;


int v1;
int v1_filtered;
int v2;
```

```
int v2_filtered;
int v3;
int v3_filtered;
int v4;

int j,k;
int cnt;
int dc;

int trobat=0;


struct ControlFlags{
        unsigned     RunMotor:1;
        unsigned     HighSpeedMode:1;
        unsigned     TakeSnapshot:1;
          unsigned    MediumEventFlag:1;
          unsigned    SlowEventFlag:1;
          unsigned    SpeedControlEnable:1;
          unsigned    EnablePotentiometer:1;
        unsigned     :9;
};

extern volatile struct ControlFlags ControlFlags;
volatile struct ControlFlags ControlFlags;

unsigned int cont_sec1=PASOS_SEC1;           // hold time for first lock
unsigned int cont_sec2=PASOS_SEC2;           // hold time for second lock

unsigned int acc=0;
unsigned int dc_rampa=0;


unsigned int med_ev_period;

unsigned int esp_cruce=4;
unsigned int v_media=0;
unsigned int tiempo_60_elec[6];
unsigned int Sector_sig;
unsigned int tiempo_360_elec;

unsigned int n_cruces=0;

unsigned int tombs0=0;
unsigned int tombs1=0;
unsigned int tombs2=0;
unsigned int tombs3=0;
unsigned int tombs4=0;
unsigned int tombs5=0;

unsigned int t_prova[6];
unsigned int t_total;
unsigned long t_total2;
unsigned int carrega;

unsigned int t_consigna=10000;//864;//2500;//3456;
int PID_error[3];
unsigned int PID_dc;
```

```
float inc_dc;
float K1=KP+KI+KD;
float K2=-(KP+2*KD);
float K3=KD;

unsigned int vel_pond;
unsigned int suma_vels;
unsigned int vel_pond_aux[6];
unsigned int ind_pond=0;
int inc_dc_aux;
unsigned int diag=0;
unsigned int sat_TMR1=0;
unsigned int sat_TMR2=0;
unsigned int control_pendent=0;

unsigned int blanking_count=BLANK_CT;

unsigned int ind_pond_aux=0;
unsigned int vel_pond_aux_med;
```