Universidad de Cádiz
Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE ESTRUCTURAS DE CONTROL TRADICIONAL PARA EL CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE MODULACIÓN POR ANCHO DE PULSO Y FRECUENCIA DE PULSOS

AUTOR: DAVID MORALES MILLÁN

Cádiz, 29 de junio de 2020

Universidad de Cádiz

Escuela Superior de Ingeniería

TRABAJO DE FIN DE GRADO

GRADO EN INGENIERÍA ELECTRÓNICA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE ESTRUCTURAS DE CONTROL TRADICIONAL PARA EL CONTROL DE UN MOTOR DE CORRIENTE CONTINUA MEDIANTE MODULACIÓN POR ANCHO DE PULSO Y FRECUENCIA DE PULSOS

DIRECTOR: FERNANDO PÉREZ PEÑA

CODIRECTOR: RAFAEL GUARDEÑO RAMÍREZ

AUTOR: DAVID MORALES MILLÁN

Cádiz, 29 de junio de 2020

Índice general

1. Introducción **1**

2. Modelado matemático **5**
2.1. Principio de funcionamiento 5
2.2. Obtención de la función de transferencia 8
2.3. Parámetros del motor 10

3. Diseño de los controladores. Simulación en tiempo continuo **13**
3.1. Controlador I-PD. Método analítico 13
3.2. Controlador PID. Curva de reacción 17
3.3. Simulación del controlador I-PD 20
3.4. Simulación del controlador PID 24

4. Discretización. Simulación en tiempo discreto **29**
4.1. Selección del periodo de muestreo. Discretización de los controladores 30
4.2. Simulación PWM 31
4.3. Simulación PFM 38
4.4. Simulación Encoder 45
4.4.1. Simulación de la determinación del sentido de giro del motor 54
4.5. Simulación Frecuencímetro 57
4.6. Simulación del controlador I-PD en tiempo discreto 67
4.6.1. Simulación sin el bloque frecuencímetro en la realimentación 68
4.6.2. Simulación con el bloque frecuencímetro en la realimentación 73
4.7. Simulación del controlador PID en tiempo discreto 80
4.7.1. Simulación del controlador PID sin frecuencímetro en la realimentación 81
4.7.2. Simulación del controlador PID con el frecuencímetro en la realimentación 84
4.8. Controladores en recursiva 87

4.8.1. Simulación de ambos controladores en recursiva 90
4.9. Consideraciones previas a la implementación en *hardware* . . 91
4.9.1. Obtención de los nuevos valores del controlador PID con la reductora 95
4.9.2. Ajuste fino del controlador I-PD con la reductora . . . 98

5. Implementación en hardware. Programación. 101
5.1. Hardware 101
5.2. Programación 103
5.2.1. Programación y funcionamiento de los *Timers* 106
5.2.2. Programación y funcionamiento del periférico PWM . 110
5.2.3. Programación y funcionamiento del periférico QEI . . 121
5.2.4. Envío de datos a través de la UART a *Matlab* 130

6. Resultados 141
6.1. Pruebas iniciales 143
6.1.1. Controlador PID 144
6.1.2. Controlador I-PD 146
6.1.3. Comparativa entre controladores 149
6.1.4. Prueba con una cadena de *Set-Points* 150
6.2. Ajuste fino con curvas de velocidad reales 152
6.2.1. Controlador PID 153
6.2.2. Comparativa entre los controladores ajustados 156
6.2.3. Prueba con una cadena de *Set-Points* 157

7. Anexo: Códigos de los bloques en *Simulink* 161
7.1. Bloque PWM 162
7.2. Bloque PFM 163
7.3. Bloques de los canales del encoder 164
7.3.1. Bloque inicial del canal A 164
7.3.2. Bloque final del canal A 164
7.3.3. Bloque inicial del canal B 165
7.4. Bloque final del canal B 165
7.5. Bloque encargado de determinar el sentido de giro del motor 166
7.6. Bloque frecuencímetro 167
7.7. Bloques de las acciones de control en recursiva 168
7.7.1. Acción integral 168
7.7.2. Acción derivada 168
7.7.3. Filtro a la acción derivada 169

Índice de figuras

1.1. Esquema del procedimiento llevado a cabo en la realización de este trabajo. 3

2.1. Máquina de corriente continua funcionando como motor.[1] . 6
2.2. Circuito equivalente de un motor de corriente continua con excitación independiente 8

3.1. Esquema del controlador I-PD. 14
3.2. Curva de reacción de M(s) ante una entrada en escalón unitario. 18
3.3. Planta y controlador I-PD en tiempo continuo en *Simulink*. . 20
3.4. Respuesta del sistema con controlador I-PD(s) sin limitador. 22
3.5. Respuesta del sistema con controlador I-PD(s) con limitador. 22
3.6. Ajuste a los valores de presintonía del controlador I-PD en tiempo continuo. 23
3.7. Planta y controlador PID en tiempo continuo en *Simulink* . . 24
3.8. Respuestas del sistema. Controlador PID (s). Variaciones de Z-N. 25
3.9. Proceso de ajuste de los valores de presintonía del PID(s) sin limitador. 27
3.10. Ajuste de presintonía completo. Controlador PID(s) y su señal de control. 27
3.11. Respuesta del sistema con PID(s) y limitador junto a los valores de la señal de control. 28

4.1. Respuesta del bloque PWM ante una entrada sinusoidal de frecuencia igual a 8333.333Hz 35
4.2. Respuesta del bloque PWM ante una entrada en rampa con pendiente igual a 200000. 35
4.3. Respuesta del bloque PWM ante una entrada en rampa con pendiente igual a 20000 37
4.4. Ejemplo de una señal PFM. 40
4.5. Ejemplos de respuestas del bloque PFM inicial 43
4.6. Respuesta del bloque PFM final con una señal de control u=4.2. 44

4.7. Respuesta del bloque PFM final ante una señal sinusoidal f=8333.333Hz. 45
4.8. Representación de los aspectos constructivos de un encoder incremental óptico. Las zonas grises son transparentes.[4] . . 46
4.9. Señales de salida de un encoder incremental óptico.[4] 47
4.10. Respuesta de los bloques de los canales A y B del encoder (inicial) ante una entrada de velocidad en rampa. 50
4.11. Aumento a la respuesta de los bloques de los canales A y B del encoder (inicial) ante una entrada de velocidad en rampa. 51
4.12. Respuesta de los canales A y B del encoder (final) ante una entrada en rampa con pendiente positiva. 53
4.13. Respuesta de los canales A y B del encoder (final) ante una entrada en rampa con pendiente negativa. 54
4.14. Salida del bloque detector del sentido de giro. 56
4.15. Ejemplo de funcionamiento del frecuencímetro. 58
4.16. Diagrama de flujo del bloque frecuencímetro. 62
4.17. Respuesta del bloque frecuencímetro ante una entrada de velocidad en rampa con pendiente positiva. 63
4.18. Respuesta del bloque frecuencímetro ante una entrada de velocidad en rampa con pendiente negativa. 64
4.19. Respuestas del bloque frecuencímetro ante entradas sinusoidales 66
4.20. Conjunto completo con el controlador I-PD en tiempo discreto en *Simulink*. 67
4.21. Respuesta del sistema con controlador I-PD en tiempo discreto. PWM frente a PFM. *Set-Point* en escalón de amplitud 100[rad/s]. 68
4.22. Respuesta de las acciones de control frente a la señal del controlador I-PD en tiempo discreto. Entrada en escalón de amplitud 100[rad/s]. 69
4.23. Velocidad obtenida por el frecuencímetro frente a la respuesta del sistema en lazo cerrado, con controlador I-PD(z) y entrada en escalón. 70
4.24. Respuesta del sistema en lazo cerrado. Controlador I-PD(z). PWM vs PFM. *Set-Point* en escalón de amplitud 600 [rad/s]. 71
4.25. Respuesta de las acciones de control frente a la señal del controlador I-PD en tiempo discreto. Entrada en escalón de amplitud 600[rad/s]. 72
4.26. Respuesta del sistema frente a la velocidad obtenida por el frecuencímetro. Ambas modulaciones. 73
4.27. Respuesta del sistema con controlador I-PD en tiempo discreto con frecuencímetro en la realimentación. PWM frente a PFM. *Set-Point* en escalón de amplitud 100[rad/s]. 74

4.28. Respuesta del controlador IPD(z) con frecuencímetro. Comparativa de tiempos de ejecución. *Set-Point* en escalón de 100 rad/s. 75
4.29. Respuesta del controlador IPD(z) con frecuencímetro. Comparativa de tiempos de ejecución. *Set-Point* en escalón de 600 rad/s. 76
4.30. Respuesta del sistema con IPD(z) y frecuencímetro con el tiempo de ejecución ajustado *Set-Point* de 100 [rad/s]. 77
4.31. Respuesta del sistema con IPD(z) y frecuencímetro con el tiempo de ejecución ajustado. *Set-Point* de 600 [rad/s]. . . . 78
4.32. Respuesta del sistema. Controlador IPD(z). Conjunto frecuencímetro-encoder con tiempos de ejecución ajustados. *Set-Point* sinusoidal(f=33.3Hz). 79
4.33. Conjunto formado por la planta, el controlador PID en tiempo discreto y el resto de bloques en *Simulink*. 80
4.34. Respuesta del sistema con controlador PID. Modulación PWM frente a modulación PFM. *Set-Point* en escalón. 81
4.35. Respuesta del sistema con los nuevos valores del controlador PID. Modulación PWM frente a modulación PFM. *Set-Point* en escalón. 82
4.36. Respuesta de las acciones de control del controlador PID en tiempo discreto, frente a la señal de control. *Set-Point* en escalón de amplitud igual a 100. 84
4.37. Respuesta del sistema. Controlador PID en tiempo discreto. Frecuencímetro en la realimentación. PWM frente a PFM. *Set-Point* en escalón de amplitud igual a 100. 85
4.38. Respuesta del sistema. Controlador PID(z) con frecuencímetro ajustado. PWM vs PFM. 85
4.39. Respuesta del sistema. Controlador PID(z). Frecuencímetro ajustado. PWM vs PFM. 86
4.40. Respuesta del sistema. Controlador PID(z). Frecuencímetro ajustado. PWM vs PFM. *Set-Point* sinusoidal de amplitud 600. f=33Hz. 86
4.41. Diagrama de Bode del motor. 92
4.42. Prueba en lazo abierto empleando modulación PWM con periodos de 500 microsegundos y 5 microsegundos. 93
4.43. Respuesta del sistema teniendo en cuenta la reductora acoplada al motor. PID frente a I-PD en recursiva. 95
4.44. Respuesta del sistema en lazo abierto ante una entrada en escalón unitario con reductora. 96
4.45. Respuesta del sistema con controlador PID en recursiva y reductora ante un *Set-Point* en escalón. PWM frente a PFM. 97
4.46. Respuesta del sistema. Controlador I-PD(k) con frecuencímetro. PWM vs PFM. 99

4.47. Respuesta del sistema. Controlador I-PD(k) con frecuencímetro. 99

5.1. Características de la tarjeta LPC4078FBD80 102
5.2. Esquema de conexiones. 103
5.3. Diagrama de flujo general del funcionamiento del programa. . 104
5.4. Registros de funcionamiento de los *Timers*[6]. 106
5.5. Ejemplo de funcionamiento del *Timer*. PR=2,MRx=6, interrupción y reinicio activados.[6] 108
5.6. Ejemplo de funcionamiento del *Timer*. PR=2,MRx=6, interrupción y parada activados.[6] 108
5.7. Diagrama de flujo de la función: Programar interrupción del timer0 en milisegundos. 109
5.8. Diagrama de flujo de la función: Comprobar interrupción del timer0. 109
5.9. Registros de control de flancos del periférico PWM.[6] 111
5.10. Ejemplo de funcionamiento del periférico PWM.[6] 112
5.11. Diagrama de flujo de la función: Iniciar PWM. 114
5.12. Diagrama de flujo de la función: Ajustar velocidad mediante PWM. 116
5.13. Diagrama de flujo de la función: Iniciar PFM. 117
5.14. Diagrama de flujo de la función: Ajustar velocidad mediante PFM. 119
5.15. Captura de la pantalla de un osciloscopio de la señal PFM con una entrada de 5 (voltios). 120
5.16. Captura de la pantalla de un osciloscopio de la señal PFM con una entrada de 4 (voltios). 120
5.17. Tabla de estados del periférico QEI proporcionada en el *User Manual*.[6] 121
5.18. Tabla de transición de estados del periférico QEI proporcionada en el *User Manual*.[6] 122
5.19. Tabla de dirección del periférico QEI proporcionada en el *User Manual*.[6] 123
5.20. Ejemplo de la detección de la dirección de giro del QEI proporcionado en el *User Manual*.[6] 123
5.21. Diagrama de flujo de la función: Iniciar QEI. 128
5.22. Diagrama de flujo de la función: Obtener velocidad QEI. . . . 129
5.23. Algoritmo proporcionado por el fabricante para calcular la velocidad en baudios. 131
5.24. Diagrama de flujo de la función: Inicializar UART0. Transmisión de 8 bits, 1 bit de *stop* y sin paridad. 133
5.25. Diagrama de flujo de la función: Transmitir dato por la UART0. 8 bits. 134
5.26. Diagrama de flujo de la función: Recibir dato por la UART0. 8 bits. 134

5.27. Diagrama de flujo de la función: Inicializar Uart0. Transmitir cadena de caracteres por la UART0. 135

6.1. Velocidad real frente a la velocidad simulada mediante control PID y modulación PWM. *Set-Point* en escalón de 3 radianes por segundo. 144
6.2. Velocidad real frente a la velocidad simulada empleando control PID y modulación PFM. Entrada en escalón igual a 3 radianes por segundo. 145
6.3. Velocidad real frente a la velocidad simulada empleando control PID y modulación PFM. *Set-Point*=3. Sin frecuencímetro en la realimentación en la simulación. 146
6.4. Velocidad real frente a la velocidad simulada. Control I-PD. Modulación PWM. *Set-Point*=3. 147
6.5. Velocidad real frente a la velocidad simulada. Control I-PD. Modulación PFM. *Set-Point*=3. 148
6.6. Velocidad real con modulación PWM. Control PID frente a control I-PD. *Set-Point*=3. 149
6.7. Velocidad real con modulación PFM. Control PID frente a control I-PD. *Set-Point*=3. 150
6.8. Velocidad real empleando control PID. Modulación PWM frente a PFM. Entrada en forma de cadena de *Set-Points*. . . 151
6.9. Velocidad real empleando control IPD. PWM frente a PFM. Entrada en forma de cadena de *Set-Points*. 151
6.10. Velocidad real frente a la velocidad simulada. Control PID ajustado en la práctica. Modulación PWM. *Set-Point* en escalón igual a 3 radianes por segundo. 154
6.11. Velocidad real frente a la velocidad simulada. Control PID ajustado en la práctica. Modulación PFM. *Set-Point* en escalón igual a 3 radianes por segundo. 155
6.12. Velocidad real. Controladores PID e I-PD ajustados en la práctica. Modulación PWM. *Set-Point* en escalón igual a 3 radianes por segundo. 156
6.13. Velocidad real con controladores PID e I-PD ajustados en la práctica, empleando modulación PFM. *Set-Point* en escalón igual a 3 radianes por segundo. 157
6.14. Velocidad real. Controlador PID ajustado en la práctica. Modulación PFM frente a modulación PWM. Entrada igual a una cadena de *Set-Points* en escalón. 158
6.15. Velocidad real. Controlador I-PD ajustado en la práctica. Modulación PFM frente a modulación PWM. Entrada igual a una cadena de *Set-Points* en escalón. 159

Índice de tablas

2.1. Parámetros del *datasheet* del motor 10

4.1. Parámetros de la simulación y valores de partida del controlador I-PD en tiempo discreto. 68
4.2. Parámetros de la simulación y valores del controlador I-PD en tiempo discreto ajustados 79
4.3. Parámetros de la simulación y valores del controlador PID de partida 81
4.4. Parámetros de la simulación y valores del controlador PID en tiempo discreto ajustados. 87
4.5. Parámetros de la simulación y valores de las acciones de control del PID con la reductora. Valores finales. 98
4.6. Parámetros de la simulación y valores de las acciones de control del I-PD con la reductora. Valores finales. 100

Índice de códigos de los bloques en *Simulink*

7.1. Código del bloque PWM 162
7.2. Código del bloque PFM 163
7.3. Código inicial del bloque encoder. Canal A 164
7.4. Código final del bloque encoder. Canal A 164
7.5. Código inicial del bloque encoder. Canal B 165
7.6. Código final del bloque encoder. Canal B 165
7.7. Código bloque detector del sentido de giro 166
7.8. Código del bloque frecuencímetro. 167
7.9. Código de la acción integral en recursiva 168
7.10. Código de la acción derivada en recursiva 168
7.11. Código del filtro a la acción derivada en recursiva 169

Capítulo 1

Introducción

El objetivo de este trabajo es proponer dos métodos de control de la velocidad angular de un motor de corriente continua comercial: el control Proporcional-Integral-Derivativo (PID) y su variación I-PD. Además, se propone una modulación por ancho de pulsos (PWM) y por frecuencia de pulsos (PFM) para actuar sobre el motor. La cual es de interés debido a que puede mejorar la eficiencia con respecto a la modulación por ancho de pulsos con cargas bajas[1].

Como requisitos de partida para realizar el diseño de ambos controladores se establece:

- Un tiempo de subida, t_s, de 1 milisegundo.
- Un coeficiente de amortiguamiento, $\xi = 1$.
- Una ganancia estacionaria, $K = 1$.

El objetivo de estos requisitos iniciales es, al final, encontrar un buen compromiso entre la eliminación de la sobreoscilación en la respuesta y un bajo tiempo de subida.

A partir de los parámetros obtenidos del *datasheet* proporcionado por el fabricante, se realizará un modelado lineal convencional para obtener la función de transferencia del motor. A continuación, se realizará el diseño del controlador I-PD de forma analítica y el diseño del controlador PID mediante los métodos de Ziegler-Nichols basados en la curva de reacción. Para este último, será necesario el empleo del software *Matlab* y de su entorno de programación visual *Simulink*, que además, serán los programas a emplear para realizar la simulación por ordenador del comportamiento de ambos métodos de control antes de ser implementados en la práctica.

[1] https://web.archive.org/web/20160304065846/http://powerelectronics.com/site-files/powerelectronics.com/files/archive/powerelectronics.com/mag/709PET22.pdf

Se comenzará la realización de la simulación con ambos controladores en tiempo continuo, y por etapas, se irán añadiendo elementos que la acercarán a la materialización final de sus componentes, comenzando por la discretización de los controladores, seguido de la adición de ambos métodos de modulación, la transformación de la velocidad de giro del eje del motor a un tren de pulsos mediante un encoder y la traducción de dichos pulsos en valores de velocidad en radianes por segundo.

Una vez finalizada la simulación, se implementarán los controladores y ambos métodos de modulación en un microcontrolador. Dicho microcontrolador es el *LPC4078FBD80* de la marca *NXP*, basado en ARM. La programación se realizará en lenguaje "C" mediante el software *μVision*, que además será el software necesario para realizar la configuración del control en la práctica. Debido a que el giro del motor por sí mismo no proporciona suficiente información para verificar el éxito de la simulación, será necesario realizar una comparación representando gráficamente los datos de velocidad obtenidos de un encoder acoplado al eje del motor. Dichos datos de velocidad se transmitirán a un ordenador con el software *Matlab* y *Simulink* mediante el empleo de la comunicación UART. Para ello, será necesario el empleo de un adaptador que transforme uno de los puertos USB del ordenador en un puerto virtual RS232 habilitado a los niveles de voltaje TTL, necesarios para el empleo de este tipo de comunicación.

En función de los resultados obtenidos se establecerá cuáles son los métodos de control y modulación que proporcionan mejores resultados, y cuánto se ha aproximado la simulación a los resultados reales. Finalmente se realizará un ajuste fino a los valores de las acciones de control empleando el motor y sus resultados reales.

Por tanto, los distintos capítulos del documento formarán la siguiente estructura:

- Un primer bloque dedicado dedicado al modelado matemático del motor, la obtención de su función de transferencia y a partir de ella realizar el diseño de los controladores PID e I-PD.

- Un segundo bloque dedicado a la simulación. Comenzando por la simulación de los controladores en tiempo continuo y añadiendo, por etapas: los controladores discretizados, la modulación por ancho de pulsos (PWM) y por frecuencia de pulsos (PFM), la inclusión del encoder y la interpretación de sus señales como valores de velocidad.

- Un tercer bloque dedicado a la descripción a nivel interno de los periféricos que son absolutamente necesarios para el funcionamiento del motor y del control en la práctica, junto a los diagramas de flujo de las distintas funciones programadas en lenguaje "C". Además, se describirá la comunicación con un ordenador con *Matlab*.

- Un cuarto bloque final dedicado a la obtención de resultados y comparación con la simulación.

A modo de resumen del procedimiento llevado a cabo en la realización de este trabajo, se proporciona el diagrama de flujo mostrado en la figura 1.1.

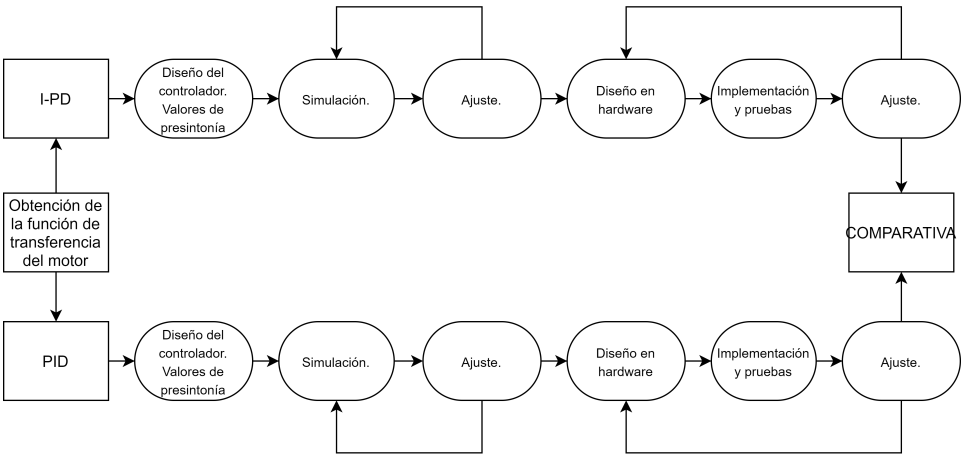


Figura 1.1: Esquema del procedimiento llevado a cabo en la realización de este trabajo.

Capítulo 2

Modelado matemático

2.1. Principio de funcionamiento

El motor de corriente continua es un dispositivo cuyo comportamiento puede ser descrito como un modo de funcionamiento de la máquina de corriente continua en el que la energía eléctrica de entrada es transformada en energía mecánica. La descripción teórica de su funcionamiento, sus aspectos constructivos y ecuaciones, se describirán a continuación. Dicha descripción está basada en el libro "Máquinas Eléctricas"'de Jesús Fraile Mora [1], excepto donde se indique lo contrario.

El motor está constituido por una parte fija denominada estátor y una parte móvil llamada rotor. El estátor está formado por el circuito magnético inductor y además ejerce la función de soporte mecánico del conjunto. El rotor está formado por el circuito inducido y el colector de delgas o conmutador. El colector de delgas es el encargado de la conversión mecánica de la corriente alterna inducida en las bobinas en corriente continua de salida, es por tanto el elemento que caracteriza a la máquina de corriente continua. Está formado por láminas de cobre, las denominadas delgas, cuya sección transversal tiene forma de cola de milano.

El suministro de corriente al colector se realiza mediante el uso de escobillas. Éstas permanecen inmóviles en los portaescobillas de forma que, mientras gira el rotor, las escobillas conservan una posición invariable con respecto a los polos. Existen motores que no emplean escobillas, siendo reemplazadas por un inversor de frecuencia constante o un conmutador electrónico[1].

Cuando la máquina de corriente continua funciona como motor, se debe aplicar una tensión continua de alimentación al inducido que provoca que

[1]Moczala, Helmut. "Small electric motors", 1998. Pg 165. https://books.google.es/books?id=EBteVXBlF-sC&lpg=PA165&dq=small%20electric%20motors%20moczala%20brushless&hl=es&pg=PA165#v=onepage&q=small%20electric%20motors%20moczala%20brushless&f=false

circule la corriente por éste. La interacción de esta corriente con el flujo magnético del inductor origina un par de rotación que obliga a girar a la máquina. La dirección de rotación dependerá del sentido de las corrientes inducidas. Esto provoca la aparición de una fuerza electromotriz en el inducido que se opone a la corriente que circula por los conductores y por ello recibe el nombre de fuerza contraelectromotriz (f.e.m.). El movimiento del motor se mantendrá en tanto en cuanto el par electromagnético de rotación producido sea superior al par resistente ejercido por la carga conectada al motor. Un ejemplo del par de giro electromagnético generado en función del sentido de las corrientes es el mostrado en la figura 2.1.

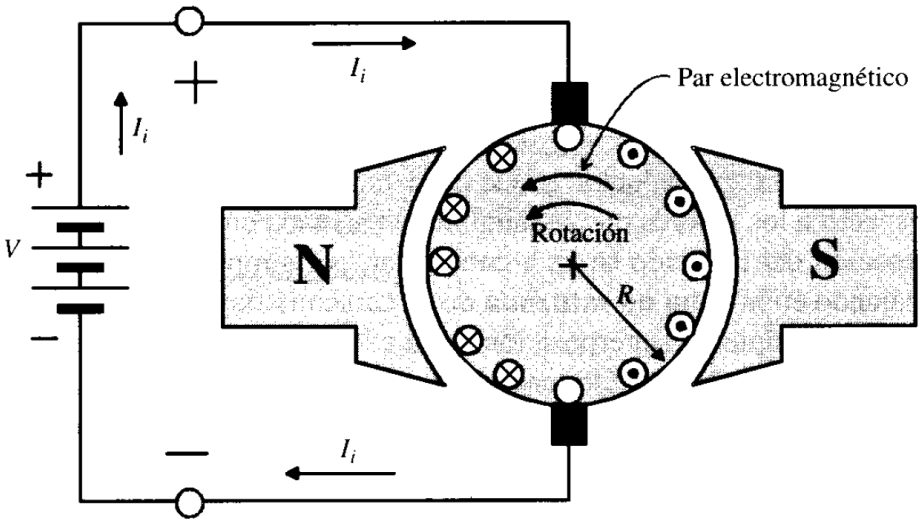


Figura 2.1: Máquina de corriente continua funcionando como motor.[1]

Para calcular la f.e.m. producida, hay que tener en cuenta que en cada bobina del arrollamiento se obtiene una f.e.m. alterna, de tal forma que en un semiperiodo de la misma el flujo varía entre los límites $+\Phi$ y $-\Phi$ (flujo de los polos), en consecuencia, el valor medio de la f.e.m. será:

$$E_{med} = -\frac{2}{T}\int_{-\Phi}^{+\Phi} d\Phi = \frac{4\Phi}{T} \tag{2.1}$$

Donde T indica el período de la corriente. Como la frecuencia de la tensión alterna generada va ligada al número de polos, $2p$, y a la velocidad de rotación w en r.p.m. por la siguiente expresión:

$$f = \frac{wp}{60} \tag{2.2}$$

La f.e.m. media en una espira será:

$$E_{med} = 4\Phi\frac{wp}{60} \tag{2.3}$$

Las escobillas recogen las f.e.m. inducidas en las distintas bobinas durante un semiperíodo, por tanto, la f.e.m total en el inducido será la suma de las f.e.m. medias de las distintas bobinas que componen cada rama del devanado. Si dicho devanado consta de Z conductores que forman $\frac{Z}{2}$ bobinas y está dividido por las escobillas en $2c$ circuitos se tienen en cada rama $\frac{Z}{4c}$ bobinas. Sustituyendo se obtiene una f.e.m. resultante de:

$$E = \frac{Zp}{60c} w\Phi = K_e w\Phi \tag{2.4}$$

Que para un flujo variable será:

$$e(t) = K_e w(t) \frac{d\Phi}{dt} \tag{2.5}$$

Donde $K_e$es una constante determinada para cada máquina.

Tal y como se ha comentado anteriormente, el paso de la corriente continua por el inducido provoca en el rotor un par electromagnético. Para calcular la magnitud de este par debe tenerse en cuenta que si I es la corriente total del inducido, la corriente que circula por los conductores del rotor en una máquina con $2c$ circuitos será $\frac{I}{2c}$. Si se denomina B_{med} al valor medio del campo magnético inducido en los conductores y L la longitud de los mismos, se obtiene aplicando la ley de Laplace:

$$F_{med} = B_{med} L \frac{I}{2c} \tag{2.6}$$

Que tiene sentido tangencial al rotor. Si se denomina R al radio del rotor y Z al número de conductores del inducido, se obtiene un par resultante:

$$T = RF_{med}Z = RB_{med}L\frac{I}{2c}Z \tag{2.7}$$

Para relacionar el par con el flujo magnético es necesario en primer lugar hallar la superficie del inducido que está comprendida entre el paso polar de la máquina:

$$S = \frac{2\pi RL}{2p} \tag{2.8}$$

El valor del flujo magnético será entonces:

$$\Phi = B_{med}\frac{2\pi RL}{2p} \tag{2.9}$$

Que sustituyendo en la expresión del par se obtiene:

$$T = \frac{Zp}{2\pi c}\Phi I = K_p I \Phi \tag{2.10}$$

Y finalmente en flujo variable resulta:

$$T = K_p I \frac{d\Phi}{dt} \tag{2.11}$$

Donde K_p es una constante determinada para cada máquina.

2.2. Obtención de la función de transferencia

Se parte del circuito equivalente de un motor de corriente continua con excitación independiente [5], mostrado en la figura 2.2, donde R representa la resistencia de armadura, L la inductancia de la armadura $v_{motor}(t)$ el voltaje de entrada del inductor, y $e(t)$ la fuerza contraelectromotriz generada por el motor.

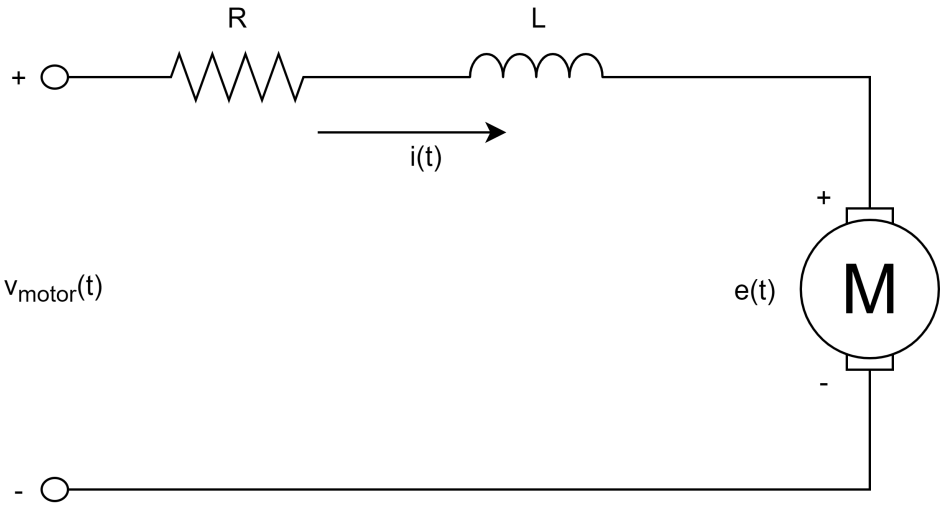


Figura 2.2: Circuito equivalente de un motor de corriente continua con excitación independiente

Se obtiene aplicando la Ley de las Tensiones de Kirchhoff:

$$v_{motor}(t) = Ri(t) + L\frac{di(t)}{dt} + e(t) \tag{2.12}$$

Que sustituyendo $e(t)$ y considerando el flujo constante se obtiene:

$$v_{motor}(t) = Ri(t) + L\frac{di(t)}{dt} + K_e w(t) \tag{2.13}$$

A continuación se realiza un equilibrio de pares (de energía) y se obtiene:

$$T_{elec} = T_{acel} + T_{roz} + T_L \tag{2.14}$$

Donde:

- T_{elec}: Es el par electromecánico del motor, (2.11), que considerando el flujo magnético, $\Phi(t)$, constante, se obtiene:

$$T_{elec} = K_p i(t) \tag{2.15}$$

- T_{acel}: Es el par debido a la aceleración rotacional del rotor, que se define como [5, Pgs. 175-178][8, Pgs. 95-97]:

$$T_{acel} = J\frac{d^2\theta(t)}{dt^2} \tag{2.16}$$

  Donde J es el coeficiente de inercia del motor.

- T_{roz}: Es el par debido a la fricción y velocidad angular del motor [5, Pgs. 175-178][8, Pgs. 95-97]:

$$T_{roz} = B\frac{d\theta(t)}{dt} \tag{2.17}$$

  Donde B es el coeficiente de fricción del motor.

- T_L: Es el par de carga, que para este análisis se considerará nulo.

Sustituyendo se obtiene la siguiente ecuación de equilibrio:

$$K_p i(t) = J\frac{d^2\theta(t)}{dt^2} + B\frac{d\theta(t)}{dt} \tag{2.18}$$

Debido a que la variable controlada es la velocidad angular, se sustituye $\frac{d\theta(t)}{dt}$ por $w(t)$, quedando las ecuaciones de la Ley de las Tensiones de Kirchhoff y la ecuación de equilibrio como:

$$v_{motor}(t) = Ri(t) + L\frac{di(t)}{dt} + K_e w(t) \tag{2.19}$$

$$K_p i(t) = J\frac{dw(t)}{dt} + Bw(t) \tag{2.20}$$

A continuación se aplica la Transformada de Laplace a ambas ecuaciones, resultando:

$$V_{motor}(s) = RI(s) + sI(s)L + K_e W(s) \tag{2.21}$$

$$K_p I(s) = sW(s)J(s) + BW(s) \tag{2.22}$$

Para obtener la función de transferencia que defina el comportamiento del motor, para el caso en el que la velocidad angular es la variable controlada, las ecuaciones deben de ser reordenadas de tal forma que se cumpla que: $M(s) = \frac{salida}{entrada}$, es decir $M(s) = \frac{W(s)}{V(s)}$. Por tanto es necesario eliminar la intensidad de corriente $I(s)$ del sistema de ecuaciones. Despejando de la ecuación de equilibrio resulta:

$$I(s) = \frac{W(s)(sJ(s) + B)}{K_p} \tag{2.23}$$

Sustituyendo y reordenando se obtiene:

$$V_{motor}(s) = R\frac{W(s)(sJ+B)}{K_p} + s\frac{W(s)(sJ+B)}{K_p}L + K_eW(s) \tag{2.24}$$

$$V_{motor}(s) = \frac{W(s)R(Js+B) + W(s)Ls(Js+B)}{K_p} + W(s)K_e \tag{2.25}$$

$$V_{motor}(s) = \frac{W(s)R(Js+B) + W(s)Ls(Js+B) + W(s)K_eK_p}{K_p} \tag{2.26}$$

$$V_{motor}(s) = \frac{W(s)\left(R(Js+B) + Ls(Js+B) + K_eK_p\right)}{K_p} \tag{2.27}$$

$$\frac{W(s)}{V_{motor}(s)} = \frac{K_p}{R(Js+B) + Ls(Js+B) + K_eK_p} \tag{2.28}$$

$$\tag{2.29}$$

Operando en el denominador se obtiene finalmente la función de transferencia del motor, siendo la velocidad angular la variable controlada, $M(s)$:

$$\frac{W(s)}{V_{motor}(s)} = M(s) = \frac{K_p}{LJs^2 + (RJ+LB)s + (RB+K_eK_p)} \tag{2.30}$$

2.3. Parámetros del motor

Voltaje nominal (U_N)	6	V	6	V
Resistencia en bornes (R)	1.94	Ω	1.94	Ω
Velocidad sin carga (η_0)	8200	min^-1	858.70	rad/s
Torque de fricción (M_R)	0.2	mNm	$0{,}2 \cdot 10^-3$	Nm
Constante de f.e.m. (k_E)	0.725	mV/min^-1	$6{,}92 \cdot 10^-3$	$V/rad/s$
Constante de par (k_M)	6.92	mNm/A	$6{,}92 \cdot 10^-3$	Nm/A
Inductancia en el rotor (L)	45	μH	$45 \cdot 10^-6$	H
Inercia del rotor (J)	2.7	gcm^2	$2{,}7 \cdot 10^-7$	Kgm^2
Velocidad nominal(n_N)	6390	min^{-1}	669.159	rad/s

Cuadro 2.1: Parámetros del *datasheet* del motor

El motor de corriente continua que se va a emplear es un motor del fabricante *Faulhaber* con código de serie *2224 U 006 SR*[2]. En la tabla 2.1 se muestran los parámetros del motor con la nomenclatura proporcionada por el fabricante y sus respectivas conversiones al sistema internacional.

Como se puede observar en la tabla de datos proporcionados por el fabricante, el coeficiente de fricción B no se proporciona directamente. Es necesario obtenerlo a partir de la ecuación del par de rozamiento del motor,

[2] https://www.faulhaber.com/fileadmin/Import/Media/EN_2224_SR_DFF.pdf

(2.17). Dicho torque es conocido y la velocidad angular a emplear será la del caso más desfavorable posible, cuando el motor gire a velocidad máxima, es decir, su velocidad sin carga. Por tanto se obtiene:

$$B = \frac{T_{roz}}{w} = \frac{M_R}{\eta_0} = 2{,}329 \cdot 10^{-}7 \tag{2.31}$$

Sustituyendo estos datos en la función de transferencia del motor, (2.30), resulta:

$$M(s) = \frac{6{,}92 \cdot 10^{-}3}{1{,}215 \cdot 10^{-}11s^2 + 5{,}238 \cdot 10^{-}7s + 4{,}834 \cdot 10^{-}5} \tag{2.32}$$

Capítulo 3

Diseño de los controladores. Simulación en tiempo continuo

Se proponen dos métodos de control partiendo de la función de transferencia del motor $M(s)$: El control Proporcional-Integral-Derivativo (PID)[5, Pgs. 708-711], para el que se reservan los métodos de diseño basados en la curva de reacción, y su variación I-PD[8, Pgs. 591-592], el cual se diseñará de forma analítica. En ambos métodos de control, la variable controlada es la velocidad angular,w, y la variable manipulada el voltaje, V. Como requisitos de partida para el diseño de ambos controladores se establece:

- Un tiempo de subida, t_s, de 1 milisegundo.
- Un coeficiente de amortiguamiento, $\xi = 1$.
- Una ganancia estacionaria, $K = 1$.

Con el objetivo de, al final, tras realizar los ajustes necesarios, encontrar un buen compromiso entre la eliminación de la sobreoscilación en la respuesta y un bajo tiempo de subida.

3.1. Controlador I-PD. Método analítico

En primer lugar, para facilitar la posterior realización de operaciones, se simplifica la expresión (2.32) a:

$$M(s) = \frac{K}{As^2 + Bs + C} \tag{3.1}$$

Donde K representa la ganancia estacionaria del sistema, y, A, B, y C los coeficientes de s^2, s, y el término independiente, respectivamente.

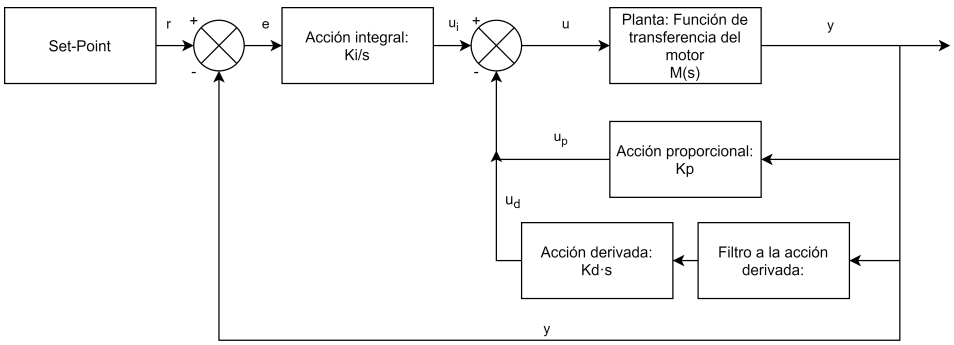


Figura 3.1: Esquema del controlador I-PD.

El diagrama de bloques de un controlador I-PD es el mostrado en la imagen 3.1. Se pretende obtener, en primer lugar, la función de transferencia del sistema en lazo cerrado con realimentación unitaria, es decir:

$$G_{yr}(s) = \frac{Y(s)}{R(s)} \tag{3.2}$$

Observando la estructura del controlador se deduce:

$$U(s) = \frac{K_i}{s}E(s) - (K_d s + K_p)Y(s) \tag{3.3}$$

$$E(s) = R(s) - Y(s) \tag{3.4}$$

$$Y(s) = M(s)U(s) \tag{3.5}$$

Sustituyendo $E(s)$ en $U(s)$ resulta:

$$U(s) = \frac{K_i}{s}\left(R(s) - Y(s)\right) - (K_d s + K_p)Y(s) \tag{3.6}$$

$$U(s) = \frac{K_i}{s}R(s) - \frac{K_i}{s}Y(s) - Y(s)K_d s - Y(s)K_p \tag{3.7}$$

$$U(s) = \frac{K_i}{s}R(s) - Y(s)(K_p + \frac{K_i}{s} + K_d s) \tag{3.8}$$

A continuación, se sustituye $U(s)$ en $Y(s)$ y se obtiene:

$$Y(s) = M(s)\left(\frac{K_i}{s}R(s) - Y(s)(K_p + \frac{K_i}{s} + K_d s)\right) \tag{3.9}$$

$$Y(s) = M(s)R(s)\frac{K_i}{s} - M(s)Y(s)(K_p + \frac{K_i}{s} + K_d s) \tag{3.10}$$

$$0 = M(s)R(s)\frac{K_i}{s} - M(s)Y(s)(K_p + \frac{K_i}{s} + K_d s) - Y(s) \tag{3.11}$$

$$0 = M(s)R(s)\frac{K_i}{s} - Y(s)\left(M(s)(K_p + \frac{K_i}{s} + K_d s) + 1\right) \tag{3.12}$$

$$Y(s)\left(M(s)(K_p + \frac{K_i}{s} + K_d s) + 1\right) = M(s)R(s)\frac{K_i}{s} \tag{3.13}$$

Reordenando para tener $\frac{Y(s)}{R(s)}$:

$$\frac{Y(s)}{R(s)} = \frac{M(s)\frac{K_i}{s}}{M(s)(K_p + \frac{K_i}{s} + K_d s) + 1} \tag{3.14}$$

$$\frac{Y(s)}{R(s)} = \frac{M(s)K_i}{sM(s)(K_p + \frac{K_i}{s} + K_d s) + s} \tag{3.15}$$

Dividiendo todo por $M(s)K_i$:

$$\frac{Y(s)}{R(s)} = \frac{1}{\frac{s}{K_i}(K_p + \frac{K_i}{s} + K_d s) + \frac{s}{M(s)K_i}} \tag{3.16}$$

$$\frac{Y(s)}{R(s)} = \frac{1}{\frac{K_p}{K_i}s + 1 + \frac{K_d}{K_i}s^2 + \frac{s}{M(s)K_i}} \tag{3.17}$$

$$\tag{3.18}$$

Sustituyendo $M(s)$ (3.1):

$$\frac{Y(s)}{R(s)} = \frac{1}{\frac{K_p}{K_i}s + 1 + \frac{K_d}{K_i}s^2 + \frac{As^3+Bs^2+Cs}{K_i K}} \tag{3.19}$$

Finalmente:

$$G_{yr}(s) = \frac{1}{\frac{A}{K_i K}s^3 + (\frac{B}{K_i K} + \frac{K_d}{K_i})s^2 + (\frac{C}{KiK} + \frac{K_p}{K_i}) + 1} \tag{3.20}$$

El resultado es una función de transferencia de tercer orden por lo que no es posible aproximarla directamente a ninguno de los dos modelos (de primer y segundo orden) convencionales. Es necesario por tanto, añadir a la función de transferencia modelo de segundo orden, un tercer polo lo suficientemente alejado de los dos polos deseados [8, Pgs. 308-310]. Como criterio se va a emplear un tercer polo que sea diez veces menor que éstos.

La función de transferencia modelo de segundo orden se define como:

$$G_{modelo}(s) = \frac{K}{\tau^2 s^2 + 2\xi\tau s + 1} \tag{3.21}$$

Donde K es la ganancia estacionaria, ξ el coeficiente de amortiguamiento y τ la constante de tiempo.

Teniendo en cuenta las condiciones de diseño, la función de transferencia modelo resulta:

Para $\xi = 1$

$$\tau = \frac{0{,}001}{4} = 2{,}5 \cdot 10^{-4} \tag{3.22}$$

$$G_{modelo}(s) = \frac{1}{6{,}25 \cdot 10^{-8}s^2 + 5 \cdot 10^{-4}s + 1} \tag{3.23}$$

Siendo el denominador de G_{modelo} la ecuación característica, los polos deseados obtenidos son:

$$s_1 = -4000 \tag{3.24}$$
$$s_2 = -4000 \tag{3.25}$$

Por lo que según el criterio establecido:

$$s_3 = -40000 \tag{3.26}$$

Que resulta en la siguiente función de transferencia modelo de segundo orden con un tercer polo añadido:

$$G_{modelo}(s) = \frac{1}{6{,}25 \cdot 10^{-}8s^2 + 5 \cdot 10^{-}4s + 1} \cdot \frac{1}{s + 40000} \tag{3.27}$$

Operando se tiene:

$$G_{modelo}(s) = \frac{1}{6{,}25 \cdot 10^{-}8s^3 + 3 \cdot 10^{-}3s^2 + 21s + 40000} \tag{3.28}$$

A continuación se divide numerador y denominador por 40000 para hacer que el término independiente del denominador de $G_{modelo}(s)$ sea igual a 1:

$$G_{modelo}(s) = \frac{\frac{1}{40000}}{\frac{6{,}25 \cdot 10^{-8}}{40000}s^3 + \frac{3 \cdot 10^{-3}}{40000}s^2 + \frac{21}{40000}s + 1} \tag{3.29}$$

Igualando los denominadores de (3.20) y (3.29)

$$\text{Coeficientes de } s^3;\ K_i = \frac{A}{\frac{6{,}25 \cdot 10^{-8}}{40000}K} \tag{3.30}$$
$$\text{Coeficientes de } s^2;\ K_d = \frac{3 \cdot 10^{-}3}{40000}K_i - \frac{B}{K} \tag{3.31}$$
$$\text{Coeficientes de } s;\ K_p = \frac{21}{40000}K_i - \frac{C}{K} \tag{3.32}$$
$$\text{Término independiente; } 1 = 1 \tag{3.33}$$

Teniendo en cuenta los coeficientes de (3.1) finalmente se obtiene:

$$\begin{aligned} K_i &= 1123{,}699 \\ K_d &= 8{,}5838 \cdot 10^{-}6 \\ K_p &= 0{,}58296 \end{aligned} \tag{3.34}$$

Que son los valores de presintonía del controlador I-PD. A los cuales se les realizará en la simulación un primer ajuste en función de la curva de reacción obtenida. Finalmente el ajuste fino tendrá lugar antes de la implementación en hardware del controlador, cuando estén incluidos todos los elementos en la

simulación. Es importante anotar que en el momento en el que se obtuvieron los valores de presintonía, no se incluyó un filtro en la acción derivada debido a que introduciría un orden más en la ecuación característica de la función de transferencia del sistema en lazo cerrado con realimentación unitaria, dando como resultado una ecuación de cuarto orden. Dificultando así la identificación con una función de transferencia modelo de segundo orden. Por ello se reserva la introducción del filtro a la acción derivada al apartado dedicado a la simulación con el software *Simulink*. Éste es necesario debido a que la acción derivada amplifica los efectos de la señal que reciba a su entrada a altas frecuencias. Sirva de ejemplo de señal de entrada una señal de error sinusoidal:

$$e(t) = Asen(wt) \tag{3.35}$$

Derivando se obtiene:

$$\frac{de(t)}{dt} = Awcos(wt) \tag{3.36}$$

Cuanto mayor sea la frecuencia, reflejada en w, mayor será su amplitud. Por ello, es necesaria la incorporación de un filtro paso-bajas, para compensar sus efectos.

3.2. Controlador PID. Curva de reacción

El diseño del controlador PID se realizará mediante el método basado en la curva de reacción de Ziegler-Nichols. Dicho método es una forma experimental de obtener los valores de presintonía de un controlador PID a partir de la respuesta del sistema en lazo abierto. Además, genera tiempos de subida cortos con sobreoscilación, lo que puede no ser adecuado para todas las aplicaciones. Por ello, existen dos variaciones del método que pretenden minimizar estos dos efectos. Las ecuaciones tradicionales de Ziegler-Nichols y sus variaciones son las siguientes:

- Z-N clásico

$$K_p = \frac{1{,}2\tau}{K\tau_d}, T_i = 2\tau_d, T_d = \tau_d/2 \tag{3.37}$$

- Z-N modificado (sobreoscilación disminuida)

$$K_p = \frac{0{,}33}{0{,}66}K_p, T_i = 2\tau_d, T_d = \frac{8}{3}T_d \tag{3.38}$$

- Z-N modificado bis (sin sobreoscilación)

$$K_p = \frac{0{,}2}{0{,}6}K_p, T_i = 2\tau_d, T_d = \frac{8}{3}T_d \tag{3.39}$$

Donde:

- τ: Es la constante de tiempo.
- τ_d: Es el *dead time* o retardo
- T_i: Es la constante de tiempo integral o *Reset Time*, que se traduce a ganancia integral mediante: $K_i = \frac{K_p}{T_i}$
- T_d: Es la constante de tiempo derivativa *Rate Time* o pendiente, que se traduce a ganancia derivativa mediante: $K_d = K_p T_d$
- K: Es la ganancia estacionaria del sistema, que viene dada por:

$$K = \frac{y_{final} - y_{inicial}}{u_{final} - u_{inicial}} \tag{3.40}$$

Para obtener dichos parámetros partiendo de la función de transferencia del sistema, se le somete a una entrada del tipo escalón unitario y se identifica que clase de respuesta se obtiene a partir de la forma de la curva obtenida.

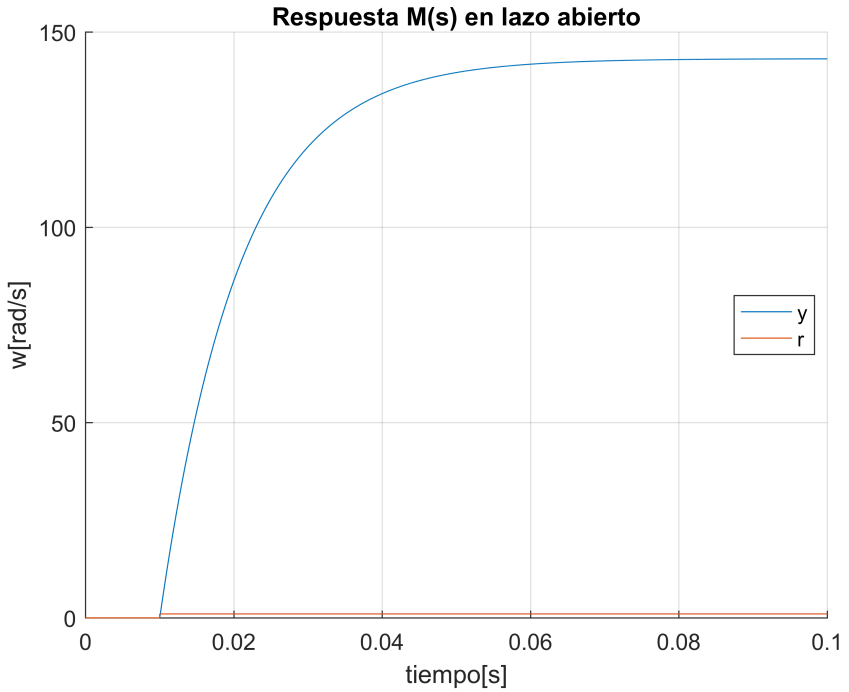


Figura 3.2: Curva de reacción de M(s) ante una entrada en escalón unitario.

Como se puede observar en 3.2, la respuesta se puede aproximar a un sistema de primer orden con retardo, que se define como:

$$G_{modelo} = \frac{K}{\tau s + 1} \cdot e^{-\tau_d s} \tag{3.41}$$

Cuyas constantes de tiempo y retardo se pueden obtener de la siguiente forma:

$$\begin{aligned} \tau &= \frac{3}{2}(t_{63} - t_{28}) \\ \tau_d &= t_{63} - \tau \end{aligned} \tag{3.42}$$

Donde t_{63} y t_{28} son los instantes de tiempo en los que la respuesta alcanza el 63 % y el 28 % de su amplitud, respectivamente.

Por tanto, teniendo en cuenta que el escalón unitario actúa al segundo:

$$K = \frac{143{,}2}{1} = 143{,}2 \tag{3.43}$$

$$\tau = \frac{3}{2}(1{,}011 - 1) - (1{,}004 - 1) = 0{,}0105 \tag{3.44}$$

$$\tau_d = (1{,}011 - 1) - \tau = 5 \cdot 10^{-}4 \tag{3.45}$$

Se obtiene la siguiente función de transferencia modelo de primer orden con retardo:

$$G_{modelo} = \frac{143{,}2}{0{,}0105s + 1} \cdot e^{-5 \cdot 10^{-}4s} \tag{3.46}$$

Una vez obtenidos los distintos parámetros, se procede a obtener los valores del controlador PID mediante Ziegler-Nichols.

Teniendo en cuenta los requisitos de partida, en principio la segunda modificación del método de Z-N es la más adecuada para los objetivos que se han marcado. De todas formas, se harán pruebas en la simulación con todas las distintas variaciones del método de Z-N y se seleccionará el más adecuado en función de sus resultados. Usando la ecuación clásica de Z-N (3.37) se obtienen unos valores de presintonía para el control PID de:

$$\begin{aligned} K_p &= \frac{1{,}2 \cdot 0{,}0105}{143{,}2 \cdot 5 \cdot 10^{-}4} = 0{,}1759 \\ T_i &= 2 \cdot 5 \cdot 10^{-}4 = 1 \cdot 10^{-}3 \\ K_i &= \frac{K_p}{T_i} = 175{,}9776 \\ T_d &= 2{,}5 \cdot 10^{-}4 \\ K_d &= K_p T_d = 4{,}3975 \cdot 10^{-}5 \end{aligned} \tag{3.47}$$

Con la primera modificación de las ecuaciones (3.38) resulta:

$$\begin{aligned} K_p &= 0{,}096745 \\ K_i &= \frac{K_p}{T_i} = \frac{0{,}05863}{1 \cdot 10^{-}3} = 96{,}745 \\ T_d &= \frac{8}{3} T_d = 6{,}666 \cdot 10^{-}4 \\ K_d &= K_p T_d = 6{,}449 \cdot 10^{-}5 \end{aligned} \tag{3.48}$$

Finalmente, usando las segundas ecuaciones de Z-N modificadas (3.39) se obtiene:

$$\begin{aligned} K_p &= 0{,}05863 \\ K_i &= \frac{K_p}{T_i} = \frac{0{,}05863}{1 \cdot 10^{-}3} = 58{,}63 \\ T_d &= \frac{8}{3} T_d = 6{,}666 \cdot 10^{-}4 \\ K_d &= K_p T_d = 3{,}9086 \cdot 10^{-}5 \end{aligned} \tag{3.49}$$

Éstos son los valores de presintonía del controlador PID. De igual forma al controlador I-PD, se les realizará en la simulación un primer ajuste en función de la curva de reacción y un ajuste fino definitivo anterior a implementar el controlador en hardware. También, en el diseño del controlador mediante Z-N no se incluye el filtrado a la acción derivada que se realizará a continuación en el apartado dedicado a la simulación.

3.3. Simulación del controlador I-PD

La simulación se realizará por etapas, realizando cambios y añadiendo elementos en cada una de ellas que acercarán progresivamente el resultado a su implementación real en hardware. Por lo cual se comenzará realizando la simulación en tiempo continuo. El software empleado es el entorno de programación visual *Simulink* que funciona sobre el entorno de programación *Matlab*. El conjunto completo formado por el controlador I-PD en tiempo continuo y la planta, en *Simulink*, se muestra en la figura 3.3.

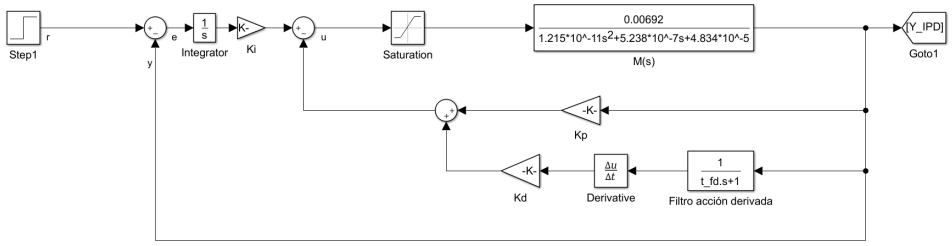


Figura 3.3: Planta y controlador I-PD en tiempo continuo en *Simulink*.

En primer lugar, siguiendo lo ya mencionado en el apartado dedicado al diseño del controlador, se introducirá el filtro a la acción derivada, cuya función de transferencia es la siguiente:

$$F(s) = \frac{1}{\tau_{fd} s + 1} = \frac{w_{fd}}{s + w_{fd}} \tag{3.50}$$

Donde t_{fd} es la constante de tiempo del filtro y w_{fd} la frecuencia de corte.

Como criterio para la selección del valor del filtro se empleará:

$$\tau_{fd} = 0{,}1T_d = 0{,}1\frac{K_d}{K_p} \tag{3.51}$$

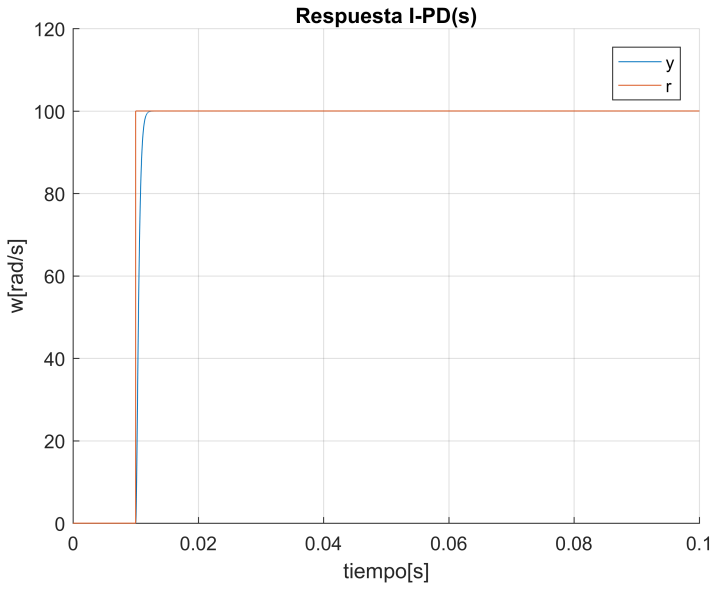
Siguiendo las recomendación de la *Instrumentation, Systems and Automation Society* de que debe ser inferior a 1/3 del *dead time* [1].

Sustituyendo los valores de presintonía obtenidos para el controlador I-PD (3.34) se obtiene una constante del filtro igual a:

$$\tau_{fd} = 0{,}1T_d = 0{,}1\frac{8{,}5838 \cdot 10^{-}6}{0{,}58296} = 1{,}47245 \cdot 10^{-}06 \tag{3.52}$$

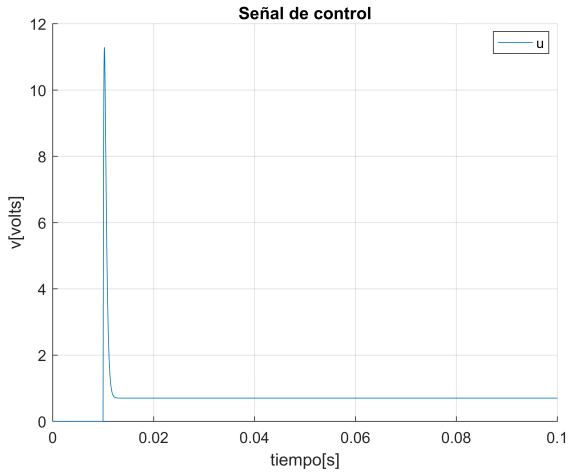
A continuación se configuran los parámetros de la simulación. Se empleará el método numérico Runge-Kutta para la resolución interna de ecuaciones diferenciales. Teniendo en cuenta que la constante de tiempo efectiva del sistema, en su aproximación como modelo de primer orden con retardo, (3.46), es de 10,5 milisegundos, un paso de integración fijo de $1\mu s$ es suficiente como para garantizar una buena precisión en sus cálculos en las simulaciones que se van a realizar en tiempo continuo.

Finalmente, incluyendo el filtro en el controlador I-PD la respuesta del sistema ante un *SetPoint* de tipo escalón con una amplitud de 100, es la reflejada en la figura 3.4a.



(a) Respuesta del sistema con los valores de presintonía.

[1] https://www.isa.org/standards-and-publications/isa-publications/intech-magazine/documents/pid-tuning-rules-appendices/



(b) Señal de control sin limitador.

Figura 3.4: Respuesta del sistema con controlador I-PD(s) sin limitador.

La respuesta cumple con los requisitos de partida, es decir, un coeficiente de amortiguamiento igual a 1 y un tiempo de subida de 994.095μs. Sin embargo, prestando atención a la respuesta de la señal de control, u, representada en la figura 3.4b, se observa que alcanza un valor de pico de 11.3 voltios. Teniendo en cuenta que el voltaje nominal del motor a emplear es de 6 voltios, es necesario añadir un limitador a la simulación, que alterará la respuesta del sistema.

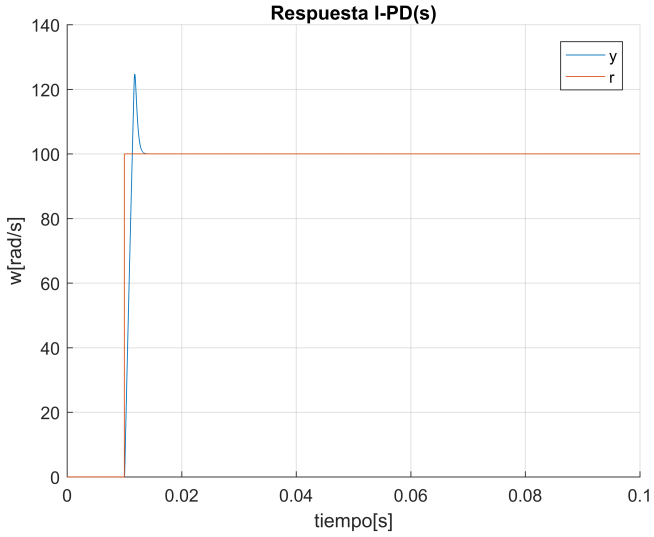
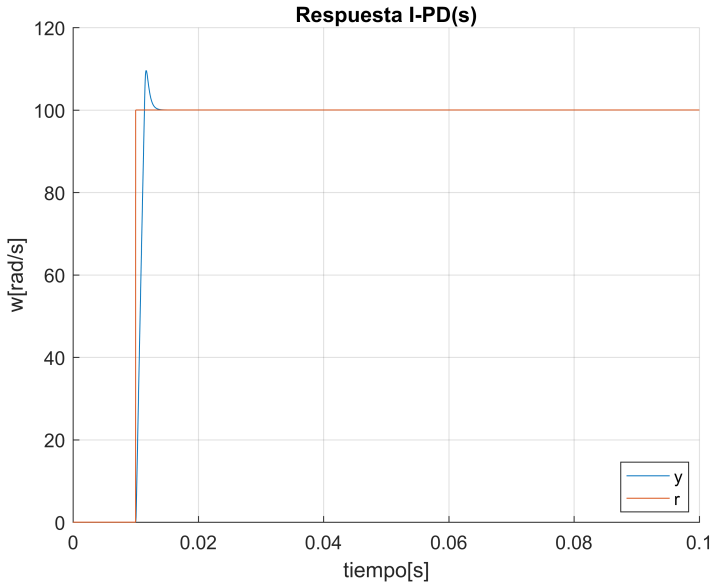
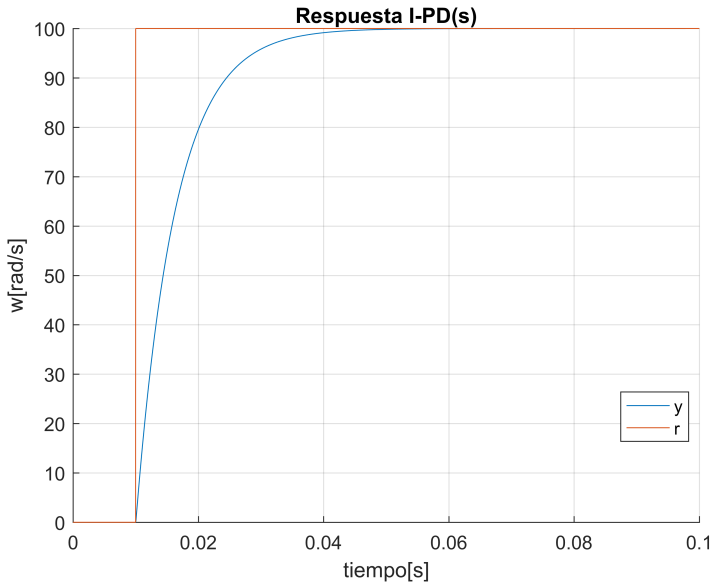


Figura 3.5: Respuesta del sistema con controlador I-PD(s) con limitador.

Una vez incorporado a la simulación el limitador de ± 6 voltios se genera una sobreoscilación en la respuesta del sistema, tal y como se refleja en la figura 3.5.



(a) Respuesta del sistema con Ki=923.699.



(b) Respuesta del sistema con controlador I-PD ajustado.

Figura 3.6: Ajuste a los valores de presintonía del controlador I-PD en tiempo continuo.

Es necesario reajustar los valores del controlador I-PD para eliminar la sobreoscilación. Debido a la estructura del controlador I-PD la mayor parte del peso del control recae sobre la acción integral en primer lugar se disminuirá su valor cien unidades. El efecto de este cambio se representa en la figura 3.6a.Se ha conseguido disminuir la sobreoscilación pero no ha desaparecido por completo, para ello es necesario compensar aumentando los valores de K_d y K_p. Ambos han sido multiplicados por diez. El resultado se muestra en la figura 3.6b.

Como se puede observar se ha conseguido eliminar por completo la sobreoscilación a costa de aumentar el tiempo de subida a 14.515 milisegundos. A pesar de ello, se considera un resultado aceptable debido a que se busca encontrar el mejor compromiso entre la ausencia de sobreoscilación y un bajo tiempo de subida, como se estableció en los requisitos de partida.

Finalmente, los valores finales del controlador I-PD en tiempo continuo son:

$$\begin{aligned} K_i &= 923{,}699 \\ K_p &= 5{,}8296 \\ K_d &= 8{,}5838 \cdot 10^{-}5 \\ t_{fd} &= 1{,}4725 \cdot 10^{-}6 \end{aligned} \tag{3.53}$$

3.4. Simulación del controlador PID

En primer lugar se añadirá un filtro a la acción derivada de la misma forma que se realizó con el controlador I-PD. También se establecen los mismos parámetros de simulación. El conjunto completo formado por la planta y el controlador PID en tiempo continuo se muestra en la figura 3.7.

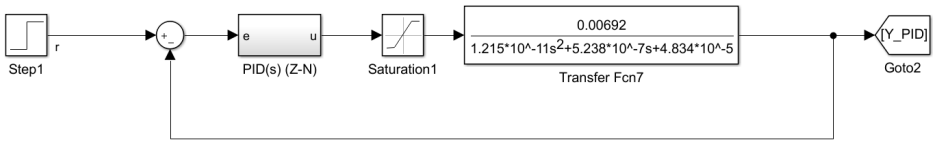


Figura 3.7: Planta y controlador PID en tiempo continuo en *Simulink*

Siguiendo el criterio establecido en la ecuación (3.50) se obtiene sustituyendo los valores de presintonía del controlador PID para cada una de las formas del método de Z-N:

- Z-N Original

$$t_{fd} = 2{,}5 \cdot 10^{-}05 \tag{3.54}$$

- Z-N Modificado

$$t_{fd} = 6{,}666 \cdot 10^{-}05 \tag{3.55}$$

- Z-N Modificado Bis

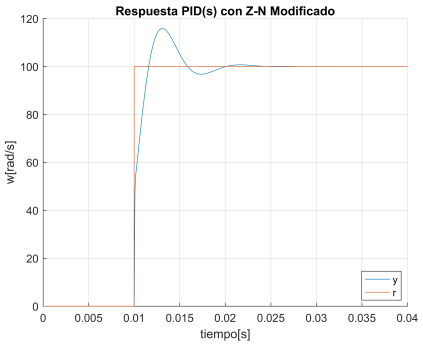
$$t_{fd} = 6{,}666 \cdot 10^{-}05 \tag{3.56}$$

Incluyendo el filtro a la acción derivada y usando los valores del controlador PID usando las ecuaciones de Z-N originales (3.47), la respuesta del sistema en lazo cerrado con realimentación unitaria es la representada en 3.8a. El tiempo de subida de la respuesta es de 1.572 milisegundos y el valor máximo que se alcanza en la sobreoscilación es de 113.7 rad/s.

Para el caso de las ecuaciones modificadas, la respuesta del sistema se representa en la gráfica 3.8b. En estas condiciones se obtiene un tiempo de subida de 1.210 milisegundos y un valor de pico en la sobreoscilación de 115.8 rad/s.

Y finalmente, usando la segunda modificación de las ecuaciones de Z-N la respuesta del sistema es muestra en la gráfica 3.8c. En este caso, el tiempo de subida es de 1.572 milisegundos y el valor máximo alcanzado en la sobreoscilación es de 121.5 rad/s.

(a) Ecuaciones de Z-N originales.



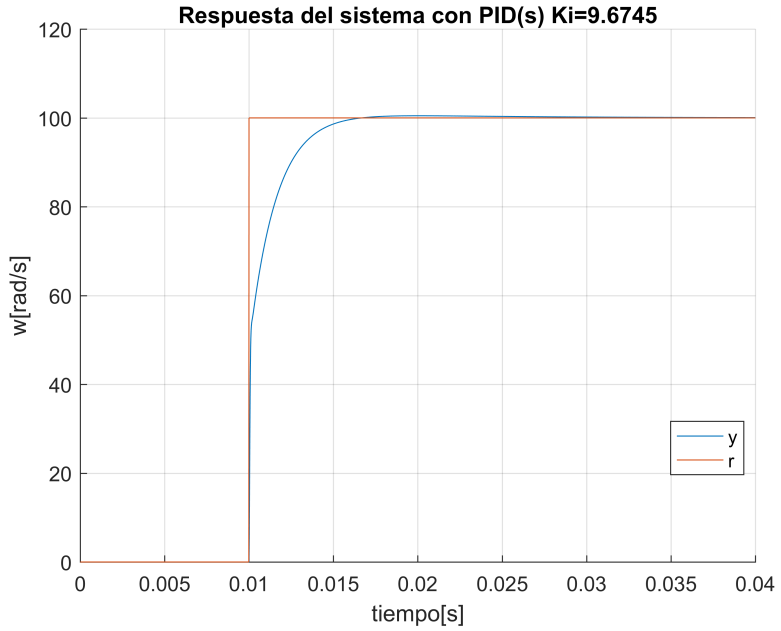
(b) Ecuaciones de Z-N modificadas.

(c) Ecuaciones de Z-N modificadas bis.

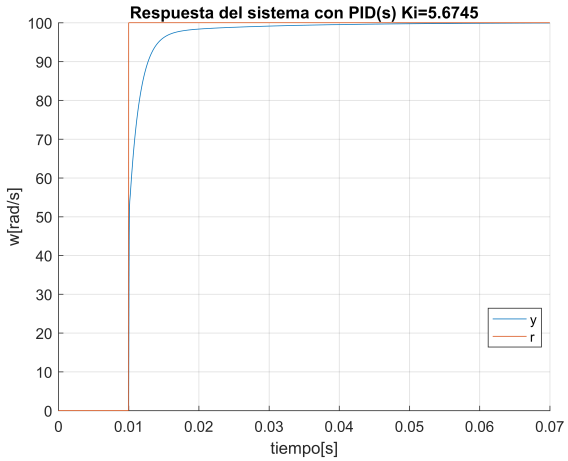
Figura 3.8: Respuestas del sistema. Controlador PID (s). Variaciones de Z-N.

Analizando los resultados, es mejor usar como valores de presintonía los obtenidos de las ecuaciones de Z-N modificadas, ya que presentan el tiempo de subida más cercano a los requisitos de partida y la sobreoscilación a

ajustar es muy similar a la obtenida con las ecuaciones de Z-N originales. Para eliminarla por completo se comenzará disminuyendo el valor de la acción integral, ya que al tener un valor tan elevado con respecto a los de la acción proporcional y derivada, tiene mayor peso en el control, y provoca que el sistema no tenga error en régimen permanente a costa de incrementar su oscilación. Esto se debe a que la acción integral es un sumatorio de la señal de error hasta el instante de cálculo, lo que provoca un efecto de carga-descarga. La acción integral va sumando el error (se carga) y no corrige (se descarga) hasta que el error cambia de signo. Cuando esto ocurre ya se ha sobrepasado el *Set-Point*. Por tanto, se comienza dividiendo el valor de K_i por diez, obteniendo la respuesta representada en la gráfica 3.9a. El resultado es una reducción considerable de la sobreoscilación pero sin eliminarse por completo, por lo que se continua disminuyendo el valor de K_i de uno en uno hasta que finalmente se consigue lo reflejado en la figura 3.9b.



(a) Respuesta del sistema con PID(s) y Ki=9.6745.



(b) Respuesta del sistema con PID(s) y Ki=5.6745.

Figura 3.9: Proceso de ajuste de los valores de presintonía del PID(s) sin limitador.

Como se puede observar se ha conseguido eliminar la sobreoscilación a costa de aumentar el tiempo de establecimiento. Aumentar K_p incrementa la velocidad de respuesta pero también el tiempo de establecimiento. Aumentar K_d hace que la respuesta pase a estar subamortiguada. Por ello, se disminuye K_p centésima a centésima, de 0.09 hasta 0.06, y se disminuye el valor de K_d diviéndolo primero por diez y finalmente por cien hasta obtener $6{,}449 \cdot 10^{-}7$. El resultado se representa en la figura 3.10.

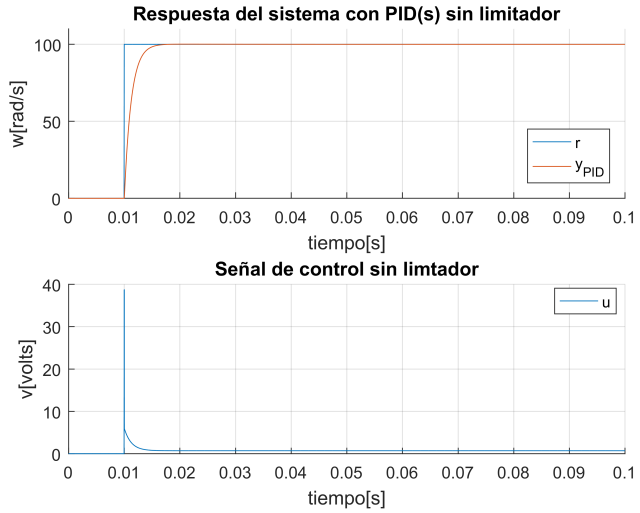


Figura 3.10: Ajuste de presintonía completo. Controlador PID(s) y su señal de control.

Se obtiene una respuesta sin amortiguamiento con un tiempo de subida de 2.866 milisegundos el cual es inferior al del controlador I-PD por lo que se considera un resultado aceptable. Observando la señal de control del controlador en la figura 3.10 se observa que alcanza un valor de 40 voltios, por lo que es necesario implementar en la simulación un limitador de ±6 voltios, que es el voltaje nominal del motor.

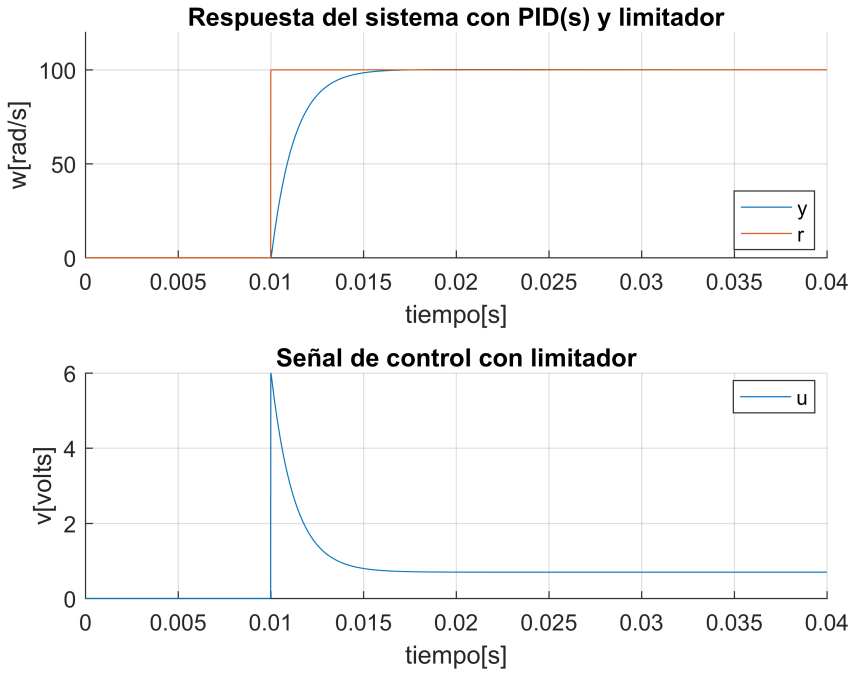


Figura 3.11: Respuesta del sistema con PID(s) y limitador junto a los valores de la señal de control.

Sin embargo como se puede observar en la figura 3.11 la respuesta no se ve alterada. Por lo que los valores finales del controlador PID en tiempo continuo son los mostrados a continuación:

$$\begin{aligned} K_p &= 0{,}06 \\ K_i &= 5{,}6745 \\ K_d &= 6{,}449 \cdot 10^{-}7 \\ t_{fd} &= 1{,}07483 \cdot 10^{-}6 \end{aligned} \tag{3.57}$$

Capítulo 4

Discretización. Simulación en tiempo discreto

La discretización es el proceso que consiste en convertir funciones, ecuaciones, modelos y variables continuas en sus contrapartes discretas, es decir, pasan a estar definidas en unos instantes determinados, generalmente periódicos[1], con el objetivo de hacerlos adecuados para su evaluación numérica y por tanto implementables en controladores digitales[7, Pg. 4]. Dichos instantes periódicos, en el caso de señales continuas, definen el *periodo o frecuencia de muestreo* o número de muestras que se toman para producir una señal discreta.

Hay varias formas de representar este proceso de muestreo matemáticamente. Una de ellas es representar inmediatamente la señal muestreada por una serie $y[k]$ donde k es el número de muestras tomadas. De tal forma que la primera muestra será $y[0]$, la siguiente $y[1]$, etcétera. Un ejemplo de esta representación es la siguiente señal, o función, en la que se muestrea con un periodo de muestreo T_m:

$$y(t) = \sum_{k=0}^{\infty} y(t - kT_m) \tag{4.1}$$

Debido a que se trata de una función en el tiempo, siguiendo el proceso de análisis y diseño de sistemas de control[8, Pgs. 15-16], el siguiente paso será transformarla en una función del dominio de la transformada de Laplace:

$$Y(s) = \int_0^{\infty} y(t)e^{-st}dt \tag{4.2}$$

[1] http://diccionario.raing.es/es/lema/discretizar-0

Si $t = kT_m$

$$Y(s) = T_m \sum_{k=0}^{\infty} y(kT_m) e^{(-sKT_m)} \tag{4.3}$$

$$Y(s) = T_m \sum_{k=0}^{\infty} y(k) e^{(sKT_m)^{-k}} \tag{4.4}$$

Haciendo el cambio de variable $z = e^{sT_m}$

$$Y(z) = T_m \sum_{k=0}^{\infty} y(k) z^{-k} \tag{4.5}$$

Obteniendo la transformada z de $y(t)$[7, Cp. 2]. La cual, transforma una secuencia de números complejos en una representación en el dominio de la frecuencia.

Una de las formas más habituales de relacionar la transformada z con la transformada de Laplace es la transformación bilineal, que es una aproximación de s a z partiendo de la base que se quiere obtener una función de transferencia del tipo $G(z) = \frac{N(z)}{D(z)}$. Haciendo el desarrollo en serie de Taylor del logaritmo natural de $z = e^{sT_m}$ se obtiene:

$$s = \frac{2}{T_m} \frac{z-1}{z+1} \tag{4.6}$$

Consiguiendo en definitiva la discretización de un sistema continuo representado en el dominio de la transformada de Laplace.

4.1. Selección del periodo de muestreo. Discretización de los controladores

Como criterio para la elección del periodo de muestreo se sigue la recomendación indicada por las conclusiones de un estudio[2] de la *International Society of Automation*, $T_m = \frac{t_{efectiva}}{10}$. Teniendo en cuenta la función de transferencia del motor (2.32) el periodo de muestreo recomendado es $T_m = \frac{5{,}238 \cdot 10^{-7}}{10} = 5{,}238 \cdot 10^{-8}$ segundos. Debido a que el paso de integración que se va a emplear en la posterior simulación debe ser menor que el periodo de muestreo, debería ser del orden de unos $5 \cdot 10^{-9}$ segundos, haciendo el proceso de simulación extremadamente lento debido a la densidad de cálculos que se realizan, por lo que, al final se toma la decisión de establecer el periodo de muestreo en:

$$T_m = 5{,}238 \cdot 10^{-6} \text{ segundos.} \tag{4.7}$$

[2] https://www.isa.org/store/sample-time-is-a-fundamental-parameter-for-controller-design-and-tuning-isa-expo-2007-technical-c/120991

La discretización de los controladores, tanto I-PD como PID, se realizará a través de las distintas acciones de control por separado. De tal forma que, usando la transformación bilineal para la acción integral se tiene:

$$G_{K_i}(s) = \frac{K_i}{s} \tag{4.8}$$

$$G_{K_i}(z) = K_i \frac{T_m(z+1)}{2(z-1)} \tag{4.9}$$

Para el caso de la acción derivada se obtiene:

$$G_{K_d}(s) = K_d s \tag{4.10}$$

$$G_{K_d}(z) = K_d \frac{2(z-1)}{T_m(z+1)} \tag{4.11}$$

Y su filtro:

$$F(s) = \frac{1}{\tau_{fd} s + 1} \tag{4.12}$$

$$F(z) = \frac{1}{\tau_{fd}(\frac{2z-2}{T_m z + T_m}) + 1} \tag{4.13}$$

$$F(z) = \frac{1}{\frac{\tau_{fd}(2z-2)+T_m z+T_m}{T_m z+T_m}} \tag{4.14}$$

$$F(z) = \frac{T_m z + T_m}{z(2\tau_{fd} + T_m) - 2\tau_{fd} + T_m} \tag{4.15}$$

La acción proporcional permanece inalterada con la discretización pero, a la hora de ser implementada en la simulación, se acompañará de un *Zero-Order-Hold* (Z0H) funcionando a un periodo igual al periodo de muestreo.

4.2. Simulación PWM

Siguiendo con el proceso de aproximación a la implementación en hardware a través de la simulación, a continuación se procede a añadir los actuadores, es decir, aquellos elementos que transforman la corriente continua proporcionada por el controlador en una señal que pueda ser usada por la planta. En este caso, se trata del modulador por ancho de pulso (*Pulse-Width-Modulation o PWM* y del modulador por frecuencia de pulso (*Pulse-Frecuency-Modulation o PFM*).

La modulación por ancho de pulsos es un método para reducir el voltaje medio, y por tanto la potencia media entregada por una señal, dividiéndola en partes discretas, formando una señal cuadrada, mediante el control del ciclo de trabajo de ésta. De tal forma que mediante el apagado y encendido de un interruptor a una frecuencia determinada, se consigue aumentar el

voltaje medio aplicado a la carga cuanto más tiempo permanezca encendido dicho interruptor[3]. Es decir, se establece la siguiente relación:

$$V_{sal} = D \cdot V_{max} \tag{4.16}$$

Donde V_{sal} es el voltaje medio que se quiere obtener, D es el ciclo de trabajo de la señal cuadrada PWM, y V_{max} es la amplitud, en voltios, de la señal PWM a obtener. El ciclo de trabajo D se define como:

$$D = \frac{t_{on}}{t_{on} + t_{off}} = \frac{t_{on}}{T_{PWM}} \tag{4.17}$$

Donde T_{PWM} es el periodo de la señal PWM.

Una vez descrito el funcionamiento y modelo matemático de la modulación por ancho de pulsos se procede a su simulación en *Simulink*. Para ello, se empleará un bloque del tipo *Matlab Function* que permite introducir funciones de *Matlab* en *Simulink*, siendo una función local dentro de la simulación. En primer lugar es necesario establecer el *Sample Time* del bloque PWM, el cual, se puede definir como el tiempo de ejecución particular del bloque. Es decir indica cuando, dentro de la simulación, el bloque debe producir salidas y actualizar su estado interno. Este bloque PWM va a recibir una señal de entrada por parte del controlador que funciona al periodo de muestreo establecido en (4.7) por lo que necesita tener un tiempo de ejecución menor que éste para poder realizar las operaciones internas y entregar una salida. La relación entre el periodo de muestreo y el tiempo de ejecución del bloque PWM va a determinar la resolución de los datos producidos por el bloque. Si se denomina al periodo de ejecución del bloque T_res (de resolución) se determina:

$$T_res = 5 \cdot 10^{-7} \text{ segundos.} \tag{4.18}$$

De tal forma que la resolución de los datos de salida del bloque PWM será de 0.1 voltios. Por tanto es necesario hacer el paso de integración de simulación más pequeño, es decir a $5 \cdot 10^{-8}$ segundos. A continuación se realizará la programación del bloque.

Teniendo en cuenta que el controlador proporciona el valor de voltaje medio que se quiere transmitir al motor, a través de la señal de control u, y que el voltaje máximo de la señal PWM es conocido, ya que viene determinado por el voltaje nominal del motor de ± 6 voltios, la incógnita es el ciclo de trabajo de la señal PWM, que será uno distinto para cada valor de entrada proporcionado por el controlador. Por lo que se tendría según (4.16)

$$u = D \cdot V_{max} \tag{4.19}$$

$$D = \frac{u}{V_{max}} \tag{4.20}$$

El siguiente paso es traducir dicho ciclo de trabajo a una señal cuadrada con una amplitud de ±6 voltios. En esencia, desde el punto de vista digital, una señal PWM es una señal cuadrada que como tal es una sucesión determinada de unos y ceros, por lo que puede ser descrita como un *array* o vector inicialmente lleno de ceros, en el caso de la programación en *Matlab*, de un número de posiciones concreto con unos y ceros en unas posiciones determinadas. El número de posiciones de dicho vector vendrá dado por la resolución que se pretende que tenga la señal PWM, es decir por la relación entre el periodo de muestreo T_m y el periodo de ejecución del bloque T_res. Como se pretende enviar una señal PWM distinta, con un ciclo de trabajo distinto, cada vez que el controlador envíe un dato, el periodo de la señal PWM debe ser igual al periodo de muestreo. Teniendo en cuenta que se estableció el periodo de ejecución del bloque como $T_res = 5 \cdot 10^{-7}$ segundos, es decir, el intervalo de tiempo en el que se envía un solo dato de la señal PWM, serán necesarios diez datos para obtener una señal de un periodo igual al periodo de muestreo. Por lo que, en definitiva, diez es el número de posiciones que tendrá el vector que va a contener los datos de la señal PWM. Se define entonces el tamaño del vector como:

$$tam = \frac{T_m}{T_res} = \frac{5 \cdot 10^{-6}}{5 \cdot 10^{-7}} = 10 \tag{4.21}$$

Es importante apuntar que, debido al propio funcionamiento del software *Simulink*, el tiempo de ejecución del bloque no podrá ser menor que el paso de integración de cálculo de la simulación. Tampoco podrá ser mayor que el periodo de muestreo del controlador, ya que, debe realizar las operaciones necesarias antes de concluir un periodo de muestreo. Es decir, desde el punto de vista del funcionamiento de *Simulink*, en el supuesto caso de establecer un tiempo de ejecución diez veces mayor que el periodo de muestreo ($T_m = 5 \cdot 10^{-6}$ y $T_res \cdot 10^{-5}$) se tendría que el bloque PWM solo actuaría una vez hayan transcurrido diez periodos de muestreo, perdiendo así, diez valores provenientes del controlador. Además, implicaría un resultado del cálculo del tamaño del vector contenedor de la señal PWM, decimal. No es posible, desde el punto de vista de la programación, declarar vectores con un número de posiciones decimal. Desde el punto de vista físico, implicaría establecer un periodo de la señal PWM mayor que el periodo de muestreo del controlador. Por tanto se tiene como limitación del periodo de ejecución del bloque:

$$T_sim < T_res < T_m \tag{4.22}$$

Teniendo en cuenta la definición del ciclo de trabajo de la ecuación (4.17) y el tamaño del vector a emplear, se puede obtener el número de unos de la señal PWM como:

$$D = \frac{t_{on}}{T_{PWM}} \tag{4.23}$$

$$t_{on} = D \cdot T_{PWM} \tag{4.24}$$

Es importante señalar que como en un vector no puede existir un número de posición decimal, será necesario redondear el resultado de la operación anterior. Lo cual significa una pérdida de precisión, que deberá ser tenida en cuenta a la hora de realizar la comparación de los resultados reales con los de la simulación. Más adelante se ofrecerán dos soluciones a esta pérdida de precisión.

Para rellenar el vector con el número de unos determinado, se recorrerá cada una de las posiciones mediante un bucle *for* comprobando en cada iteración si el número de posiciones recorridas es inferior o igual al número de unos a introducir. En caso afirmativo, se introducirá 1 en el vector que va a almacenar la señal PWM (o un -1 en el caso de ser una señal de entrada negativa), y en caso contrario un 0. Una vez terminado el bucle *for* se enviarán los datos contenidos en el vector uno a uno, cada T_res, multiplicados por 6 que es el valor de voltaje nominal del motor y por tanto la amplitud de la señal PWM generada.

Debido a que cada vez que se produce una ejecución del bloque las variables locales empleadas se resetean, es necesario establecer que la variable tipo vector que va a contener la señal PWM sea de tipo *persistent* para que no sea borrada cada vez que se envíe un dato.

Con el objetivo de evitar que se realice el recorrido y rellenado del vector una y otra vez cada vez que se envíe un dato, se empleará otra variable de tipo *persistent* llamada j que se usará para asegurar que solo se entre una vez en el bucle, y a su vez, será usada como índice que recorrerá las posiciones del vector a enviar, aumentando en uno tras cada ejecución del bloque. Este proceso se muestra en el código 7.1, donde *tam* es un parámetro de entrada externo de la función.

Para mostrar el funcionamiento, en general, del bloque PWM se le introduce una señal sinusoidal de amplitud ± 6 voltios, que es el valor del voltaje nominal del motor. El resultado se presenta en la figura 4.1. El valor de la señal sinusoidal que se encuentra en la intersección entre ésta y la señal PWM es el valor de entrada del bloque PWM. Como se puede observar cuanto mayor sea el valor de la señal de entrada mayor es el tiempo en alto de la señal PWM de salida. Lo mismo ocurre cuando el valor de entrada es negativo.

Para mostrar el funcionamiento con mayor claridad, se mostrará únicamente la reacción del bloque PWM ante una entrada positiva, ya que presenta una mayor claridad visual, de tipo rampa. Dicha rampa tendrá una pendiente igual a 200000, consiguiendo que, cada vez que el bloque lea un nuevo valor de entrada de la señal en rampa, es decir cada periodo de muestreo (5 microsegundos), coincida con un número natural, aumentando de uno en uno. El resultado se representa en la gráfica 4.2.

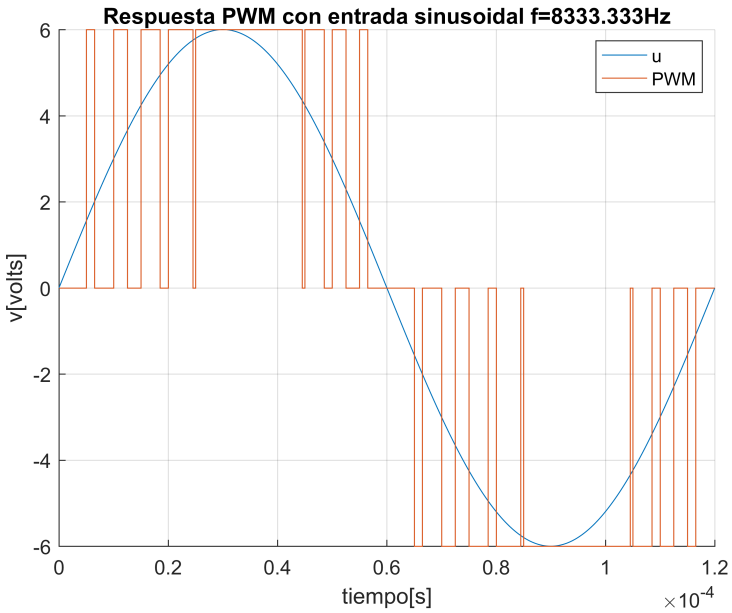


Figura 4.1: Respuesta del bloque PWM ante una entrada sinusoidal de frecuencia igual a 8333.333Hz

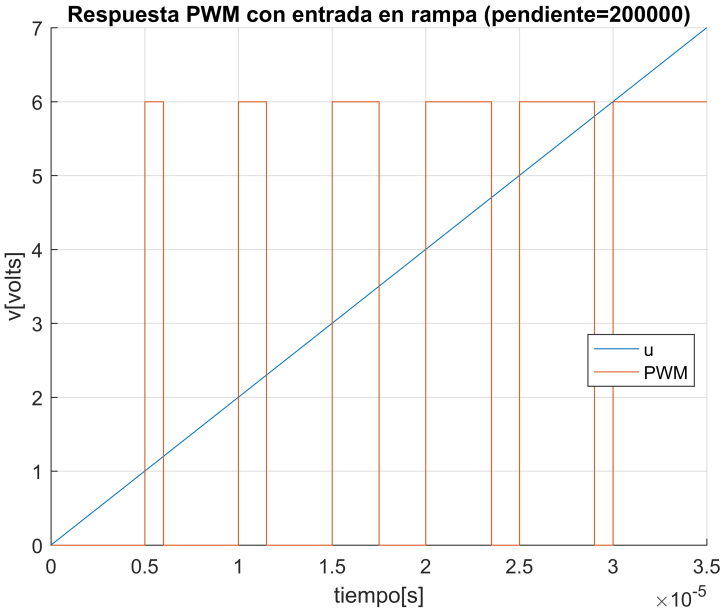


Figura 4.2: Respuesta del bloque PWM ante una entrada en rampa con pendiente igual a 200000.

En este caso se observa con claridad que cuando el valor muestreado, que es el que se encuentra en la intersección entre la señal de entrada rampa y la salida PWM, aumenta, también lo hace su ciclo de trabajo hasta ser una señal continua cuando se llega al valor máximo de 6 voltios.

Para llevar el bloque PWM a su resolución limite de 0.1 definida por el tamaño del vector que almacena la señal PWM (10 posiciones para un valor) se cambia la pendiente de la señal en rampa de entrada a 20000. Con esto se consigue que cada valor muestreado sea un incremento de 0.1. El resultado se muestra en la figura 4.3a, en la que se muestra como aumenta progresivamente el ciclo de trabajo hasta ser una señal continua.

Sin embargo, en el inicio se observa una pérdida de datos, ya que si se realiza un muestreo cada 5 microsegundos en el intervalo inicial de 50 microsegundos debería haber 9 pulsos distintos, uno para cada nuevo valor de entrada, ignorando el inicial nulo, y solo hay 6. Esta pérdida de datos se muestra con mayor claridad en el *zoom* a la zona inicial representado en la gráfica 4.3b.

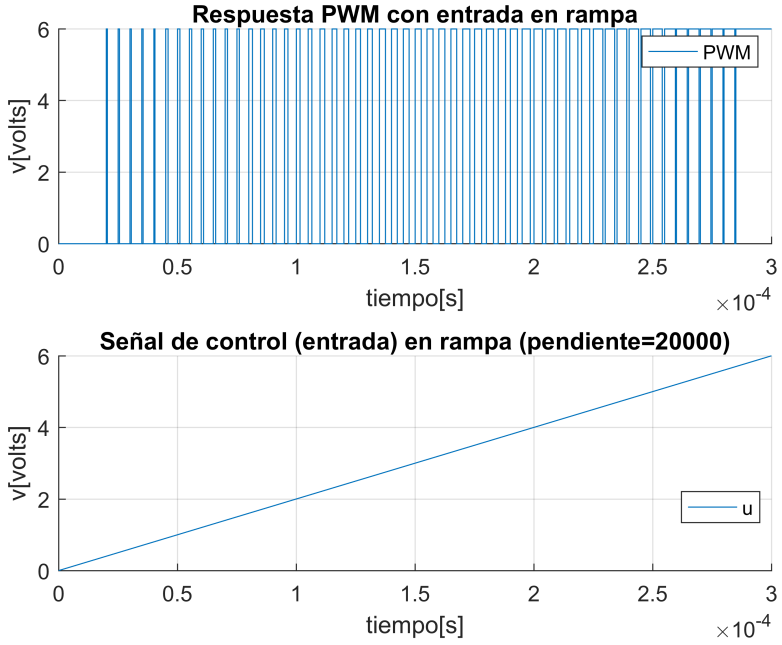
Esta pérdida de precisión, como se comentó anteriormente, se debe al redondeo introducido en la programación en la operación $th = round(tam * D)$ debido a que no se existen posiciones decimales de un vector. De tal forma que:

$$tam \cdot D \geq 0{,}51 \tag{4.25}$$

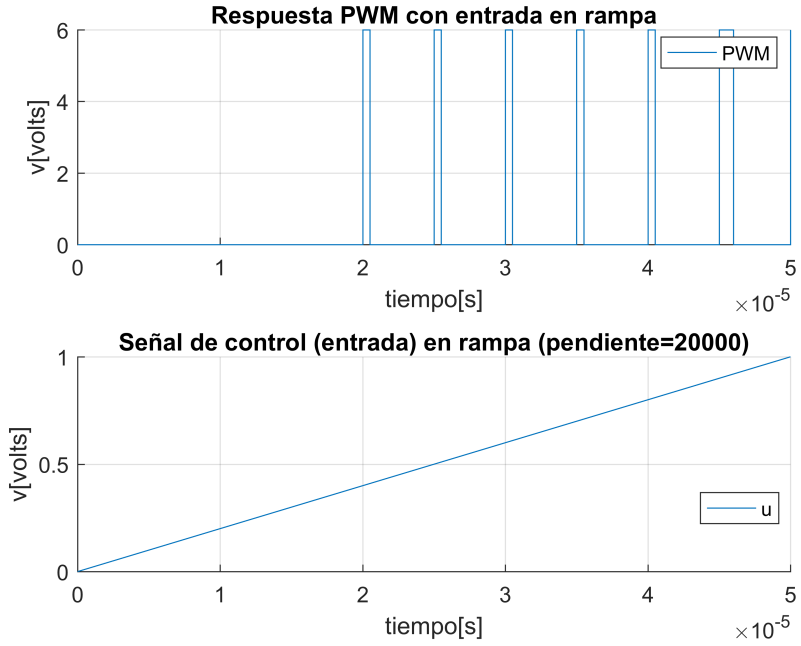
$$10 \cdot \frac{u}{6} \geq 0{,}51 \tag{4.26}$$

$$u \geq 0{,}306 \tag{4.27}$$

Los valores la señal de control de entrada al bloque PWM menores que 0.306 serán ignorados. La solución a este problema de diseño sería aumentar el tamaño del vector para así disponer de un mayor número de posiciones y por tanto de una mayor precisión. Es decir, aumentar la relación $\frac{T_m}{T_res}$. Una opción es disminuir el periodo de ejecución del bloque T_res. Sería necesario entonces un menor paso de integración, ralentizando la simulación y haciendo su finalización extremadamente lenta para tiempos de simulación pequeños como 1 segundo. Otra opción es aumentar el periodo de muestreo del controlador.



(a) Respuesta completa.



(b) Sección inicial de la respuesta.

Figura 4.3: Respuesta del bloque PWM ante una entrada en rampa con pendiente igual a 20000

4.3. Simulación PFM

La modulación por frecuencia de pulsos, *Pulse-Frequency-Modulation - PFM-*, es un método análogo a la modulación por ancho de pulsos con la diferencia de que es la frecuencia del tren de pulsos la que es variada mientras el ciclo de trabajo permanece fijo[3]. Por lo que en este caso, a diferencia del desarrollo realizado en la simulación del bloque PWM, la incógnita es el periodo de la señal de salida. Partiendo de la ecuación de la modulación PWM (4.16) y la definición del ciclo de trabajo (4.17) se obtiene el periodo de la señal PFM en función de la señal de control de entrada como:

$$V_{sal} = D \cdot V_{max} \tag{4.28}$$

$$u = \frac{t_{on}}{T_{PFM}} V_{max} \tag{4.29}$$

$$T_{PFM} = \frac{t_{on} V_{max}}{u} \tag{4.30}$$

Donde t_{on} es el tiempo en alto, que al tratarse de modulación por frecuencia de pulsos es un parámetro fijo, u es la señal de control de entrada y V_{max} el voltaje nominal del motor de ± 6 voltios. El tiempo en alto estará determinado por el tiempo de ejecución del bloque PFM. El cual será, al igual que en el caso de la modulación por ancho de pulsos, un bloque del tipo *Matlab Function*.

El motivo por el que el tiempo de ejecución del bloque determina el tiempo en alto de la señal PFM es que se trata del tiempo que transcurre entre operación y operación del bloque en la simulación, entregando el mismo valor hasta que las operaciones internas realizadas en dicho periodo de tiempo se ejecuten. Dicho periodo se denominará t_on_PFM (ya que determina su tiempo en alto) y será de $5 \cdot 10^{-7}$ segundos, igual que en el caso del bloque PWM. Por lo que se tiene:

$$T_{PFM} = \frac{t_on_PFM \cdot V_{max}}{u} \tag{4.31}$$

La simulación de la modulación por frecuencia de pulsos consistirá entonces en traducir el periodo obtenido de la señal PFM en una señal cuadrada con un ciclo de trabajo fijo, cada periodo de muestreo. Siguiendo de manera análoga el procedimiento empleado en la modulación por ancho de pulsos, se guardará en un vector de un tamaño determinado los unos y ceros que componen la señal PFM de salida del bloque para después ser enviados uno a uno. Dicho vector, deberá ser una variable de tipo *persistent* para preservar su valor entre la emisión de sus valores cada ejecución. El tamaño de éste vendrá dado por la relación entre el periodo de muestreo T_m y el periodo

[3] https://www.digikey.com/en/articles/the-advantages-of-pulse-frequency-modulation-for-dc-dc-switching-voltage-converters

de ejecución del bloque t_on_PFM; la cual determinará la resolución que tendrá la señal de salida. Es decir, 0,5 microsegundos (t_on_PFM) entre cada emisión de valores binarios de la señal PFM, en un intervalo de 5 microsegundos (T_m), resulta en una resolución de 0,1. Como se pretende enviar una señal PFM distinta cada vez que la señal de control envie un nuevo dato, es decir cada periodo de muestreo de 5 microsegundos; y se ha establecido que el bloque haga cálculos cada 0,5 microsegundos, se podrán enviar 10 datos como máximo. Por tanto, 10 es el tamaño del vector contenedor de la señal PFM. Se define entonces el tamaño del vector como:

$$tam_PFM = \frac{T_m}{t_on_PFM} = \frac{5 \cdot 10^{-6}}{5 \cdot 10^{-7}} = 10 \tag{4.32}$$

De forma análoga al bloque PWM, es importante apuntar que, debido al propio funcionamiento del software *Simulink*, el tiempo de ejecución del bloque no podrá ser menor que el paso de integración de cálculo de la simulación. Tampoco podrá ser mayor que el periodo de muestreo del controlador, ya que, debe realizar las operaciones necesarias antes de concluir un periodo de muestreo. Es decir, empleando el mismo ejemplo que en el caso del bloque PWM, desde el punto de vista del funcionamiento de *Simulink*, en el supuesto caso de establecer un tiempo de ejecución diez veces mayor que el periodo de muestreo ($T_m = 5 \cdot 10^{-6}$ y $t_on_PFM = 5 \cdot 10^{-5}$) se tendría que el bloque PFM solo actuaría una vez hayan transcurrido diez periodos de muestreo, perdiendo así, diez valores provenientes del controlador. Además, implicaría un resultado del cálculo del tamaño del vector contenedor de la señal PFM (4.32) decimal. No es posible, desde el punto de vista de la programación, declarar vectores con un número de posiciones decimal. Desde el punto de vista físico, implicaría establecer un tiempo en alto de la señal PFM mayor que el periodo de muestreo del controlador. Por tanto se tiene como limitación del periodo de ejecución del bloque:

$$T_sim < t_on_PFM < T_m \tag{4.33}$$

Por otro lado, a diferencia del bloque PWM, el parámetro clave no es el número de unos consecutivos a escribir en el vector, si no el número de ceros. Esto se debe a que el tiempo en alto es fijo, y será equivalente a colocar un "1" acompañado de un número determinado de ceros, repitiendo la secuencia hasta rellenar el vector. Cada dato se mantendrá como salida un tiempo igual al tiempo de ejecución del bloque, haciendo que el envio consecutivo de ceros establezca el tiempo en bajo de la señal PFM para cada dato proveniente de la señal de control, recibido cada periodo de muestreo. La ejecución posterior al envio del último dato contenido en el vector coincidirá con un nuevo periodo de muestreo, recibiendo por tanto un nuevo valor a su entrada, generando para el cual una secuencia distinta.

Se propone como ejemplo de señal PFM la representada en la figura 4.4.

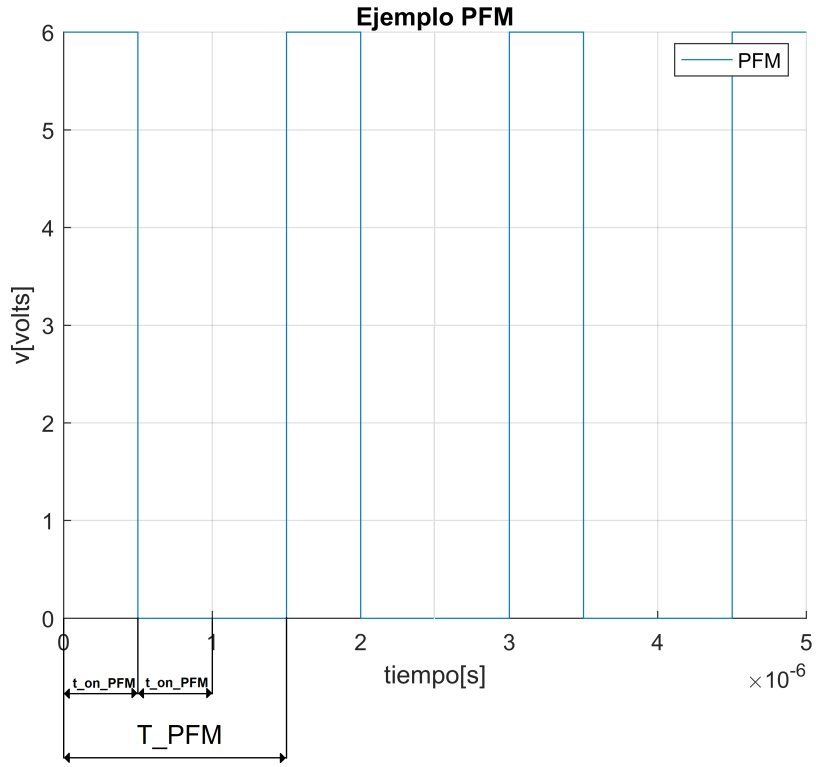


Figura 4.4: Ejemplo de una señal PFM.

En ella se muestra una señal PFM con un periodo calculado para un valor concreto de la señal de control que se enviará durante un periodo de muestreo.

Si se define el número de segmentos de tamaño t_on_PFM en un periodo de la señal PFM como:

$$N = \frac{T_{PFM}}{t_on_PFM} \tag{4.34}$$

Estableciendo como condición sine qua non de funcionamiento que el periodo de la señal PFM empiece en alto, se obtiene finalmente el número de ceros como:

$$t_off = N - 1 \tag{4.35}$$

Para rellenar el vector con el número de ceros determinado, primero se declarará inicialmente lleno de unos y se recorrerán las posiciones mediante un bucle *for*. Para preservar que la señal siempre empiece en alto, se empezará a recorrer el vector desde la segunda posición. En el caso en el que la señal de entrada sea negativa será necesario corregir el signo del primer valor. A continuación, se comprobará en cada iteración del bucle si el número de ceros introducido es inferior o igual al número de ceros objetivo mediante

una variable contador. En caso afirmativo, se introducirá un "0" en el vector que va a almacenar la señal PFM, y en caso contrario se dejará el "1" que contenía (o se multiplicará por -1 en el caso de ser una señal de entrada negativa). Una vez terminado el bucle *for* se enviarán los datos contenidos en el vector uno a uno, cada t_on_PFM, multiplicados por 6 que es el valor de voltaje nominal del motor y por tanto la amplitud de la señal PFM generada.

Debido a que no existen posiciones del vector decimales, será necesario redondear el valor de N de la ecuación (4.34), lo que introducirá una pérdida de precisión significativa que debe de ser solucionada. Para ello, en primer lugar, se realizará un análisis de los distintos valores posibles de la señal de control:

- Para el caso en el que la señal de control sea igual a cero, debido a que se encuentra en el denominador de la ecuación del periodo de la señal PFM (4.30), será necesario en la programación añadir una excepción en la cual el resultado sea directamente cero. En caso contrario, se producirá una indeterminación en el cálculo.

- Para el caso en el que la señal de control sea igual a seis, el resultado tras calcular el periodo de la señal PFM (4.30) y a continuación el número de segmentos con la ecuación (4.34), es de uno. Por lo que, el número de ceros será $t_off = N - 1 = 0$. Es decir, el resultado será una señal continua. Esto representa una imprecisión con respecto al funcionamiento real de una modulación PFM, en la cual en esta situación, los pulsos estarían muy cercanos pero con una pequeña distancia entre ellos debido al tiempo necesario que lleva realizar la conmutación del interruptor.

- Para los casos restantes en los que la señal de control es mayor que cero y menor que seis, el resultado tras calcular el número de segmentos con (4.34) será siempre mayor que uno y teniendo en cuenta que este valor debe ser redondeado, debido a que no existen posiciones del vector decimales, se tiene:

$$N = \frac{T_{PFM}}{t_on_PFM} \tag{4.36}$$

$$\frac{T_{PFM}}{t_on_PFM} \geq 1{,}51 \tag{4.37}$$

$$\frac{T_{PFM}}{5 \cdot 10^{-7}} \geq 1{,}51 \tag{4.38}$$

$$T_{PFM} \geq 7{,}55 \cdot 10^{-7} \tag{4.39}$$

Que se corresponde con unos valores de señal de control de:

$$T_{PFM} = \frac{V_{max} \cdot t_on_PFM}{u} \tag{4.40}$$

$$\frac{6 \cdot 5 \cdot 10^{-7}}{u} \geq 7{,}55 \cdot 10^{-7} \tag{4.41}$$

$$3{,}97 \geq u \tag{4.42}$$

Lo que implica que todos los valores de u mayores de 3,97 y menores de 6 serán redondeados, es decir, la señal PFM tendrá un error de precisión para todo ese rango de valores.

Este problema se ilustra gráficamente en las figuras 4.5a y 4.5b. Cuando la entrada es igual a tres, operando con las ecuaciones (4.30), (4.34) y (4.35) se obtiene:

$$T_{PFM} = \frac{5 \cdot 10^{-7} \cdot 6}{3} = 1 \cdot 10^{-6} \tag{4.43}$$

$$N = \frac{1 \cdot 10^{-6}}{5 \cdot 10^{-7}} = 2 \tag{4.44}$$

$$t_off = 1 \tag{4.45}$$

Es decir, en cada periodo de la señal PFM, cada "1" estará seguido por un único "0", obteniendo la gráfica 4.5a. Sin embargo, si la entrada es de $u = 4{,}2$ resulta:

$$T_{PFM} = \frac{5 \cdot 10^{-7} \cdot 6}{4{,}2} = 7{,}143 \cdot 10^{-7} \tag{4.46}$$

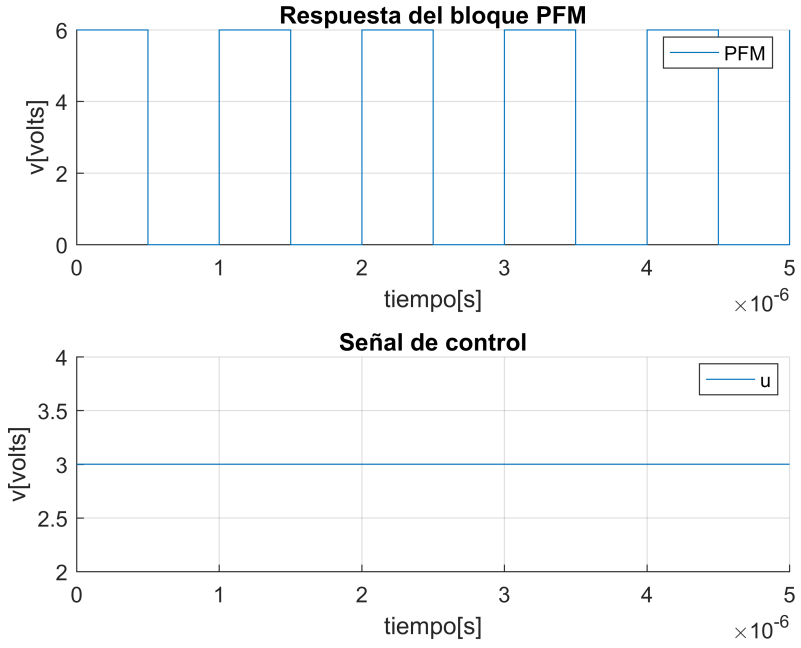
$$N = \frac{7{,}143 \cdot 10^{-7}}{5 \cdot 10^{-7}} = 1{,}4285 \tag{4.47}$$

$$\text{Redondeando; } N = 1 \tag{4.48}$$

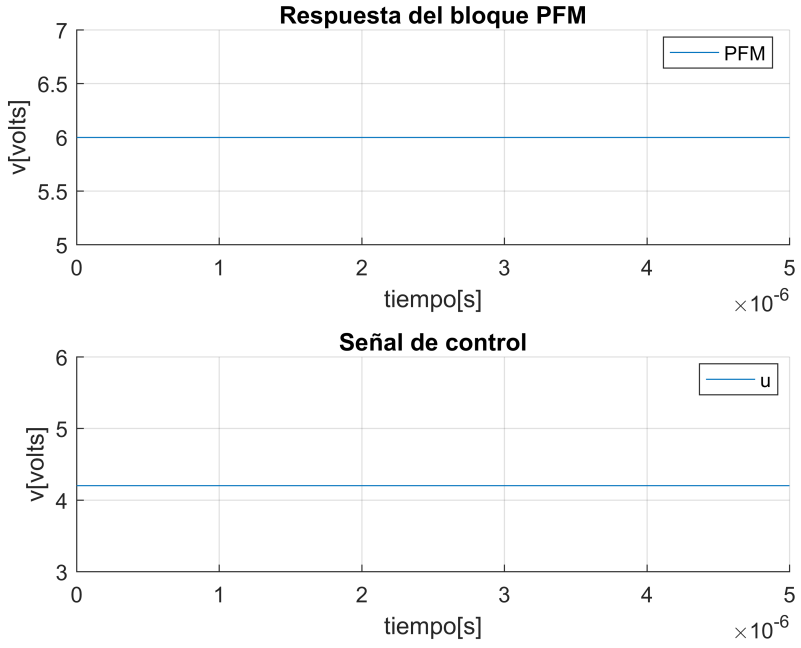
$$t_off = 0 \tag{4.49}$$

Lo que dará como resultado una señal continua. Dicho resultado es el que se debe obtener si la señal de control es igual al valor máximo, es decir, 6 voltios. Se muestra en la gráfica 4.5b.

Para solucionar esta pérdida de precisión se propone una segunda modulación en la que los valores de la señal de control en el rango $4 \leq u < 6$ sean tratados como valores del rango $0 < u < 4$. Para ello se deja la señal PFM en alto el tiempo correspondiente a una señal de entrada igual a tres. Es decir, si una señal de control de entrada del máximo valor posible, 6, se corresponde con una señal en alto todo el periodo de muestreo, una de valor igual a tres se correspondería con la mitad del periodo. A continuación se colocaría la modulación PFM correspondiente a un valor del rango $0 < u < 4$. La suma de ambas modulaciones daría como resultado una modulación equivalente de un valor de entrada del rango $4 \leq u < 6$.



(a) Señal de control u=3.



(b) Señal de control u=4.2.

Figura 4.5: Ejemplos de respuestas del bloque PFM inicial

Para implementar dicho procedimiento en la programación, en primer lugar se restará tres, el valor equivalente de la mitad del periodo en alto, al valor absoluto de la señal de control comprendido en el rango de $4 \leq u < 6$ y se realizarán las mismas operaciones que se han descrito anteriormente para cualquier valor del rango $0 < u < 4$. A continuación se recorrerá el vector inicialmente lleno de unos, mediante un bucle *for* desde la mitad más uno de sus posiciones. En el caso en el que la señal de entrada sea negativa será necesario corregir el signo de la primera mitad del vector. Independientemente del signo, de esta forma se consigue dejar la mitad del periodo en alto. De nuevo, si el número de ceros introducido es menor que el número de ceros calculado se introducirá un "0" en el vector y en caso contrario un "1" o "-1" en función del signo de la señal de control.

Finalmente, se entregarán los valores almacenados uno a uno, multiplicados por la amplitud de la señal (correspondiente al voltaje nominal del motor), recorriendo el vector usando como índice la variable de tipo *persistent*, j. Que además será usada para asegurar que las operaciones solo se hagan la primera vez que se recibe un valor y el resto de ejecuciones únicamente se envíen. El procedimiento completo se muestra en el código 7.2

Para mostrar los efectos de la segunda modulación se somete al bloque a la misma entrada de $u = 4{,}2$ usada en la figura 4.5b. Obteniendo en este caso, la señal PFM mostrada en la gráfica 4.6.

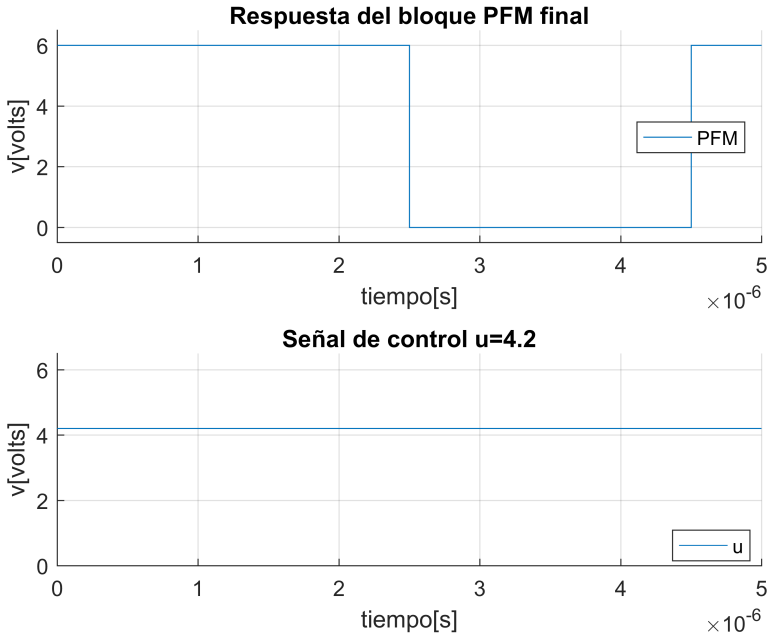


Figura 4.6: Respuesta del bloque PFM final con una señal de control u=4.2.

En ella se observa como se mantiene la señal en alto hasta la mitad del periodo de muestreo y a continuación emo la modulación PFM normal para la señal de entrada $u = 4{,}2 - 3 = 1{,}2$.

Finalmente, ante una entrada de tipo sinusoidal de amplitud igual a seis, se obtiene la respuesta del bloque PFM para el rango de valores posibles del controlador. El resultado se muestra en la figura 4.7

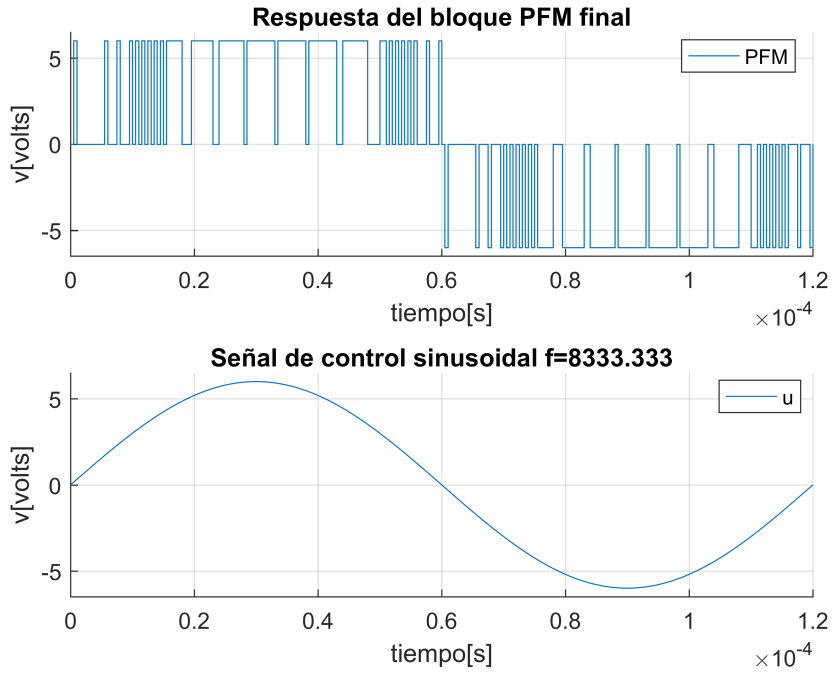


Figura 4.7: Respuesta del bloque PFM final ante una señal sinusoidal f=8333.333Hz.

Las zonas donde los pulsos tienen un tiempo en alto mayor que el resto son aquellas en las que está actuando la segunda modulación. Debido a que el siguiente periodo de la señal debe comenzar en alto, se une con el final del periodo anterior, dando esa impresión visual de que algunos pulsos son más anchos que el resto en las zonas donde actúa segunda modulación.

4.4. Simulación Encoder

Para realizar la lectura de la velocidad de giro del motor se empleará un encoder. El encoder es un dispositivo que genera pulsos eléctricos cuando su eje está en rotación. El número de pulsos generado por segundo es proporcional a la velocidad angular del eje. Existen dos grandes tipos de encoder: lineales y rotatorios. En cada grupo, a su vez, existen distinciones en función

de su tipo de codificación (absolutos e incrementales) y en función de sus principios electromecánicos de funcionamiento. En este caso, el encoder a emplear es un encoder lineal magnético.

En los encoders magnéticos, una rueda magnetizada gira sobre una placa de sensores magneto-resistivos. A medida que el disco gira sobre dicha placa, permite el paso de la luz en patrones, que causan respuestas predecibles en el sensor, basadas en la fuerza del campo mágnetico[4]. El principio de funcionamiento de este tipo de encoders es el mismo que el de los encoders ópticos, que debido a la mayor familiaridad del uso de luz que de campos magnéticos, se van a emplear para describir el funcionamiento general y modelado matemático de los encoders. La descripción teórica del funcionamiento de éstos, junto a sus ecuaciones, están basadas en la publicación "Incremental Encoder in Electrical Drives: Modeling and Simulation"[4].

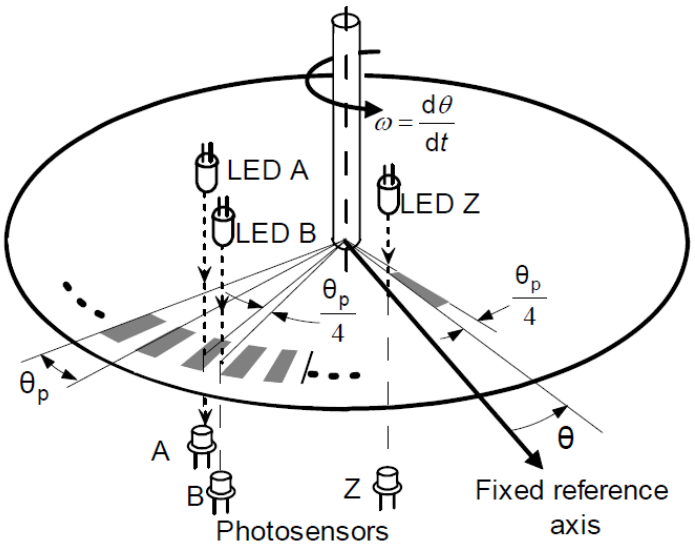


Figura 4.8: Representación de los aspectos constructivos de un encoder incremental óptico. Las zonas grises son transparentes.[4]

Los encoders ópticos tienen junto al eje un disco rotatorio transparente (normalmente de vidrio) que contiene una pista circular graduada, realizada de tal forma que en ella queda impresa una secuencia periódica de zonas transparentes y opacas, las cuales modulan el rayo de luz emitido por una fuente de luz lumínica situada en la parte fija del encoder. En el lado opuesto, se encuentran sensores ópticos que reciben dicha luz modulada, y circuitos electrónicos que la procesan. El encoder generará a su salida un pulso eléctrico cuando el eje rote un ángulo igual al paso angular de graduación, es decir, el ángulo correspondiente a una zona transparente y opaca

[4] https://www.dynapar.com/technology/encoder_basics/magnetic_encoder/

sucesivamente.

Debido al hecho de que la secuencia de pulsos generada no ofrece ninguna información sobre la dirección de rotación del eje, se provee al encoder con un segundo grupo de emisores de luz y fotorreceptores, los cuales se encuentran desplazados un cuarto del paso angular de graduación en comparación con el primer grupo de sensores. Si se invierte la rotación, el desfase entre las dos salidas de estos dos grupos de sensores también se encontrará invertido.

En algunos casos, se añade al disco una segunda pista, colocada debajo de la pista circular graduada, con una única graduación, con su respectivo grupo emisor-fotorreceptor. Esta disposición, produce un tercer canal de salida con un solo pulso en el curso de una rotación completa del disco. La posición del eje correspondiente con este pulso se usa como referencia.

El conjunto completo se muestra en la figura 4.8 donde el grupo led-fotorreceptor A es el canal principal, el grupo B es el que se encuentra desplazado un cuarto del paso angular de graduación y el grupo led-fotorreceptor Z es el usado como referencia.

Las señales de los tres canales salida están representadas en la gráfica 4.9.

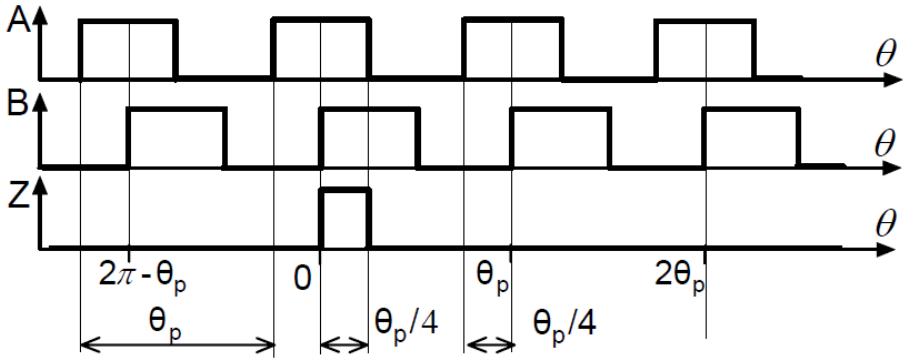


Figura 4.9: Señales de salida de un encoder incremental óptico.[4]

Por lo que, denominando a θ como el ángulo actual del motor y θ_p como el paso angular de graduación, el modelo matemático de un encoder es el siguiente[4]:

$$A(\theta) = \begin{cases} 1 \text{ si } 0 \leq mod\left(\frac{\theta}{\theta_p}\right) \leq \frac{\theta_p}{2} \\ 0 \text{ si } \frac{\theta_p}{2} < mod\left(\frac{\theta}{\theta_p}\right) \leq \theta_p \end{cases} \tag{4.50}$$

$$B(\theta) = \begin{cases} 1 \text{ si } 0 \leq mod\left(\frac{\theta - \frac{\theta_p}{4}}{\theta_p}\right) \leq \frac{\theta_p}{2} \\ 0 \text{ si } \frac{\theta_p}{2} < mod\left(\frac{\theta - \frac{\theta_p}{4}}{\theta_p}\right) \leq \theta_p \end{cases} \tag{4.51}$$

$$Z(\theta) = \begin{cases} 1 \text{ si } mod(\frac{\theta}{2\pi}) = 0 \\ 0 \text{ si } mod(\frac{\theta}{2\pi}) \neq 0 \end{cases} \tag{4.52}$$

Debido a la operación módulo, es decir, obtener el resto de la división de θ y θ_p, es posible acotar el valor de la posición angular del motor entre 0 y θ_p. De tal forma que, si el resto se encuentra entre 0 y $\frac{\theta_p}{2}$ el eje del motor se encuentra en una zona transparente del encoder, en el caso opuesto, el motor se encontrará en una zona opaca del encoder. En el canal B, ocurre de la misma forma excepto que hay que considerar el desplazamiento de los sensores de un cuarto de θ_p.

Dividiendo los 360 grados que componen una vuelta completa del disco del encoder entre el paso angular de graduación, se obtiene el número de pulsos generado por el encoder por rotación[4]:

$$N_r = \frac{2\pi}{\theta_p} \tag{4.53}$$

El encoder a emplear es de la serie *IE2-512* de la marca *Faulhaber*. Según los datos proporcionados por el fabricante en su *datasheet*[5], se trata de un encoder que funciona en el rango de 4,5 a 5,5 voltios, además presenta un desfase entre canales de 90 grados, 512 líneas por revolución y funciona a una frecuencia de 160 Khz. Emplea dos canales de salida, uno para el canal A y otro para el canal B. No dispone de un canal para la señal de referencia.

Partiendo del número de líneas por revolución y despejando de la ecuación (4.53) se obtiene un paso angular de graduación de:

$$512 = \frac{360}{\theta_p} \tag{4.54}$$

$$\theta_p = 0{,}7 \text{ grados} \tag{4.55}$$

$$\frac{\theta_p}{2} = 0{,}35 \text{ grados} \tag{4.56}$$

Por tanto, cada vez que se recorran 0.35 grados el eje se encontrará en una zona transparente del encoder. Para traducir este comportamiento a la simulación en *Simulink* se usan dos bloques del tipo *Matlab Function*, uno para cada uno de los canales disponibles del encoder. Debido a que los datos de salida proporcionados por la planta, es decir, la función de transferencia del motor, son datos de velocidad en radianes por segundo, y las ecuaciones del modelo matemático del encoder trabajan con grados; será necesario en primer lugar integrar la velocidad, obteniendo así datos de posición en radianes. A continuación se aplica una ganancia igual al factor de conversión de radianes a grados, es decir, 1 [rad] $= \frac{180}{\pi}$ [grados] y se conecta a las entradas de los bloques *Matlab Function* correspondientes a los canales A y B.

[5] https://www.faulhaber.com/fileadmin/Import/Media/EN_IE2-1024_DFF.pdf

Posteriormente será necesario establecer el periodo de ejecución de dichos bloques. Según los datos proporcionados por el fabricante el encoder trabaja a una frecuencia máxima de 160KHz, que se corresponde con un periodo de 6,25 microsegundos. Es decir, en ese periodo, el encoder ha producido un pulso de salida, que se corresponde con un "1" y un "0" digitales. Por tanto, el periodo de ejecución del bloque debe ser lo suficientemente rápido como para que tenga tiempo suficiente para enviar un "1" en primer lugar y un "0" a continuación en 6,25 microsegundos. Por lo cual, dicho periodo de ejecución, debe ser igual a la mitad del periodo de trabajo (de refresco) del encoder, es decir, 3,125 microsegundos. Debido a que en *Simulink* únicamente se pueden colocar tiempos de ejecución que sean múltiplos enteros del paso de integración (que como recordatorio, se estableció en $5 \cdot 10^{-8}$ segundos, es decir, 50 nanosegundos) se establecerá el periodo de ejecución de los bloques encoder como:

$$T_enc = 3 \cdot 10^{-6} \text{ [s]} \tag{4.57}$$

En el caso del bloque dedicado al canal A, se aplican las ecuaciones (4.50) directamente. De tal forma que, la operación módulo extraerá el resto de la división entre el ángulo recibido del eje del motor en ese instante y θ_p. A continuación se comprobará si el resultado se encuentra entre 0 y $theta_p/2$, es decir, 0.35 grados. En caso afirmativo el eje del motor se encontrará en una zona transparente del disco de graduación del encoder y se deberá entregar un "1" a la salida. En caso negativo, siguiendo las ecuaciones (4.50), se deberá comprobar si el resto de la división se encuentra entre $theta_p/2$ y $theta$ es decir, 0,35 y 0,7. En el caso en que sea cierto, el eje del motor se encontrará en una zona opaca y por tanto se entregará un "0" a la salida. Matemáticamente, el caso contrario a esta última situación es imposible, ya que el resto de la operación $\frac{\theta}{0,7}$ no puede ser mayor que 0,7, pero desde el punto de vista de la simulación, si no se han establecido todos los resultados posibles de la variable de salida ante la condición anterior la simulación no terminará, ya que el programa determina que habrá situaciones en la que no podrá entregar una salida del bloque. Por tanto, se establece como resultado al caso negativo de la condición anterior, una respuesta fuera de rango como que la salida deberá ser 2, que podrá ser utilizado para depurar errores en la simulación.

Este proceso se muestra en la programación del bloque dedicado al canal A, en el código 7.3.

Para el caso del bloque dedicado al canal B, es necesario establecer en primer lugar el desfase de 90º existente entre los canales A y B del encoder. Según lo detallado en los aspectos constructivos del encoder, se conoce que el conjunto B de emisor-fotorreceptor se encuentra desplazado un cuarto del paso de graduación con respecto al conjunto dedicado al canal A. Por tanto, restando al ángulo recibido del motor este desplazamiento, ya se tiene en consideración el desfase entre los canales y no será necesario aplicar

las ecuaciones del canal B (4.4) sino las ecuaciones del canal A, ya que el desplazamiento se ha hecho implicito en el ángulo con el que se opera.

Por lo que, procediendo de igual modo que lo descrito en la programación del bloque dedicado al canal A, se obtiene el código 7.5.

La respuesta de ambos bloques ante una entrada de velocidad de tipo rampa, que a continuación es integrada y a la que se le aplica una ganancia con el factor de conversión de radianes a grados, es la que se muestra en la gráfica 4.10

Sin embargo, esta programación a partir de la descripción matemática del funcionamiento de un encoder tiene el problema de representar un comportamiento ideal, es decir, bajo esta descripción cualquier pequeño movimiento del motor es registrado por el encoder teniendo así una resolución tan alta como *Matlab* pueda calcular. Entonces, en el arranque del motor habrá instantes en los que su movimiento sea muy lento y su eje se desplace apenas unos grados, dando como resultado un pulso inicial cuyo ancho es mayor que los pulsos que vienen a continuación tal y como se muestra en la figura 4.10. La figura 4.11 representa la sección inicial de la respuesta mostrada en la gráfica 4.10 ante la anterior entrada de posición (integral de la velocidad).

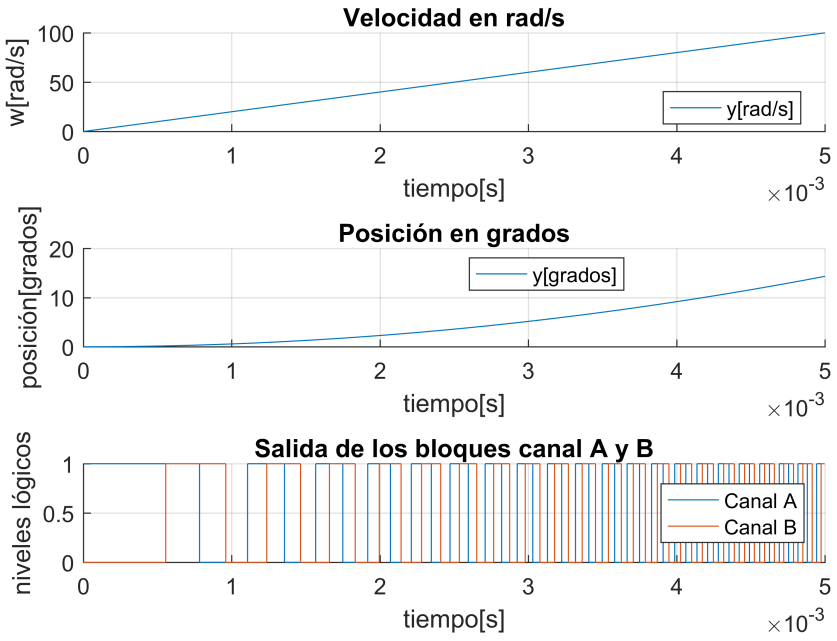


Figura 4.10: Respuesta de los bloques de los canales A y B del encoder (inicial) ante una entrada de velocidad en rampa.

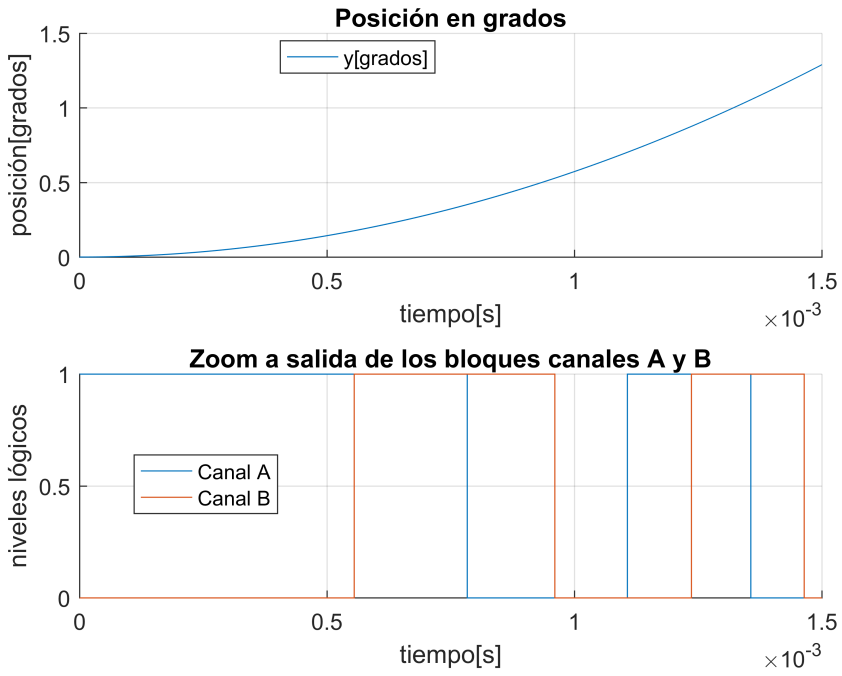


Figura 4.11: Aumento a la respuesta de los bloques de los canales A y B del encoder (inicial) ante una entrada de velocidad en rampa.

Como se puede observar en ella, el canal A comienza en alto, es decir, se encuentra en una zona transparente, siguiendo lo descrito en las ecuaciones (4.50). Mientras el motor acelera, los pequeños cambios en el ángulo del eje se van registrando dando como resultado un pulso inicial más ancho que el resto hasta que alcanza el valor de 0,35 grados, es decir, $\theta_p/2$. Por tanto a partir de entonces se encuentra en una zona opaca del disco del encoder y el nivel lógico de la señal cambia a "0". Como recordatorio, indicar que las "zonas transparentes" y "zonas opaca" son solo menciones para facilitar la explicación del comportamiento del encoder. En la práctica el encoder usado (el *IE2-512* de la marca *Faulhaber*) es magnético.

En el canal B ocurrirá lo mismo con la diferencia de que el ángulo que hace cambiar el nivel lógico de la señal se encontraría desplazado un cuarto del paso angular de graduación con respecto al de 0,35 grados del canal A. Es decir, $0{,}35 - \frac{0{,}7}{4} = 0{,}175$. Sin embargo, esto no ocurre. El canal B empieza en un nivel bajo, como si el ángulo recorrido por el encoder en el inicio fuese mayor de 0,35 grados y se encontrase en una zona opaca del disco de graduación. Esto se debe a la resta inicial realizada para implementar el desfase entre los canales A y B; y la propia definición de la función *mod* del software *Matlab*.

Según la documentación proporcionada por *Mathworks*[6] la función *mod* realiza internamente la siguiente operación matemática:

$$b = mod(a, m) \tag{4.58}$$

$$b = a - m.\ \textbf{floor}\ (\frac{a.}{m}) \tag{4.59}$$

Donde a es el dividendo y m es el divisor.

Es importante prestar atención a la introducción de la función *floor* en la realización de la división entre a y m. Dicha función según la definición proporcionada por *Mathworks* "redondea hacia el infinito negativo" es decir, devuelve el número introducido truncado hacia abajo. Para el caso de números positivos esto no representa un problema como se muestra a continuación usando como ejemplo la división de 4 entre 3:

$$resto = mod(4, 3) \tag{4.60}$$

$$resto = 4 - 3\ \textbf{floor}\ (\frac{4}{3}) \tag{4.61}$$

$$\tag{4.62}$$

Truncando hacia abajo el resultado de la división:

$$resto = 4 - 3 \cdot 1 \tag{4.63}$$

$$resto = 1 \tag{4.64}$$

Sin embargo, con un número negativo en el dividendo, se obtiene dividiendo -4 entre 3:

$$resto = mod(-4, 3) \tag{4.65}$$

$$resto = -4 - 3\ \textbf{floor}\ (\frac{-4}{3}) \tag{4.66}$$

$$\tag{4.67}$$

Truncando hacia abajo (hacia el infinito negativo) el resultado de la división:

$$resto = -4 - 3 \cdot -2 \tag{4.68}$$

$$resto = 2 \tag{4.69}$$

Obteniendo como resultado un resto incorrecto. Estas operaciones pueden ser comprobadas en uno de los ejemplos proporcionados en la documentación en la web de *Mathworks*.

Por tanto, al hacer en la programación del canal B del encoder una resta inicial, se producirá un error cuando el resultado de dicha resta sea un

[6] https://es.mathworks.com/help/releases/R2016b/matlab/ref/mod.html?searchHighlight=mod&s_tid=doc_srchtitle

número negativo. También se producirán errores en general, si los valores de entrada son números menores que cero. Para solventar este problema es necesario realizar la operación módulo con el valor absoluto del valor de entrada, y en el caso del canal B, el valor absoluto del valor de entrada tras ser restado.

La programación de los bloques de los canales A y B con este problema solventado se muestra en 7.4 y 7.6, respectivamente.

Ante una entrada de velocidad en rampa con pendiente positiva, tras ser integrada y el factor de conversión de radianes a grados haber sido aplicado se obtiene la figura 4.12.

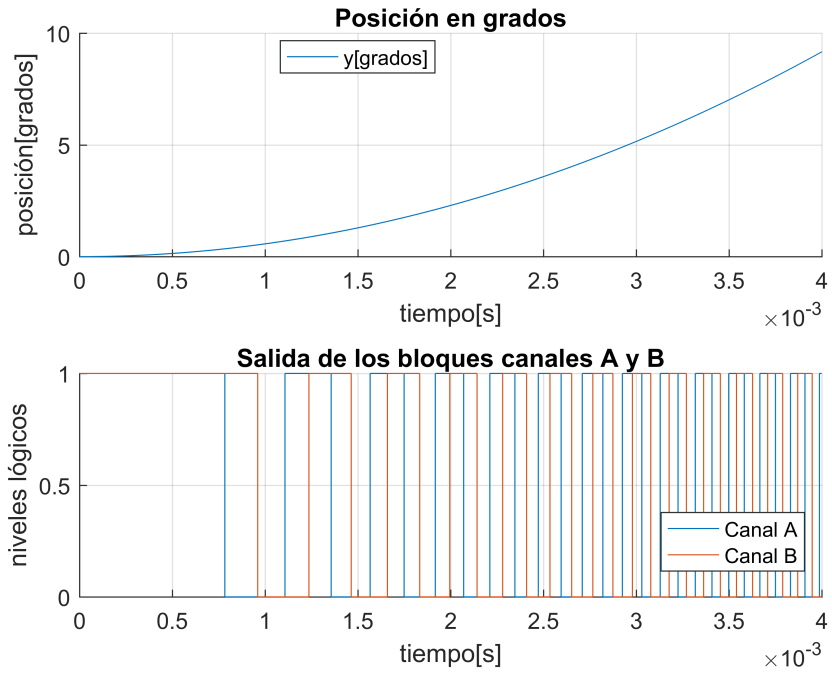


Figura 4.12: Respuesta de los canales A y B del encoder (final) ante una entrada en rampa con pendiente positiva.

En ella se observa como se ha solucionado el error de la respuesta del canal B en el inicio y como cambia de nivel lógico cuando el ángulo es de $0{,}35 - \frac{0{,}7}{4} = 0{,}175$. El resto de pulsos se encuentran desfasados 90 grados.

Y finalmente, ante una entrada de velocidad en rampa con pendiente negativa, se obtiene la figura 4.13.

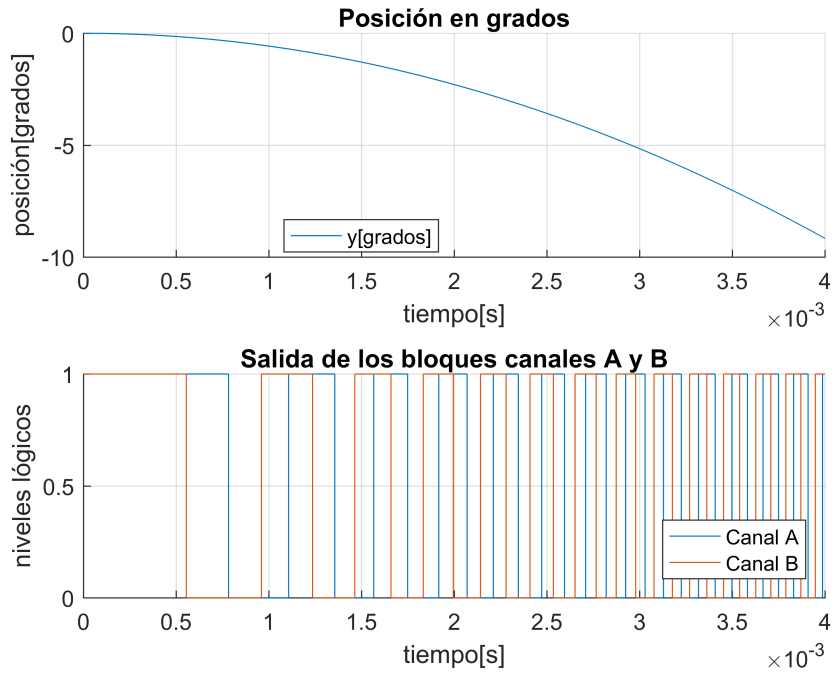


Figura 4.13: Respuesta de los canales A y B del encoder (final) ante una entrada en rampa con pendiente negativa.

En la cual se muestra que al cambiar el sentido de rotación del motor, en este caso es el canal B el que cambia de nivel lógico antes que el canal A, al contrario de lo que ocurría en la gráfica 4.12 donde el sentido de rotación era el opuesto. También se observa una consecuencia importante de la implementación matemática directa del funcionamiento de los encoder (ecuaciones (4.50) y (4.4)), que es el efecto de la aceleración. Cuanto más tiempo pase el motor en una "zona transparente" debido a que acaba de iniciar su aceleración, mayor será el ancho de pulso resultante en la simulación. Lo opuesto ocurrirá cuanto menos tiempo pase en dichas zonas.

4.4.1. Simulación de la determinación del sentido de giro del motor

A partir de la posición de los distintos pulsos proporcionados por los canales A y B del encoder, es posible determinar la dirección de rotación del eje del motor. Se propone como solución muestrear los valores del canal A cuando se produzca un flanco de subida del canal B, de tal forma que, cuando el valor del canal A sea "1" el motor girará en sentido horario, según establece el fabricante en su datasheet; y en el caso contrario, el motor girará en sentido antihorario[4].

Para implementar este comportamiento en la simulación se empleará en *Simulink* un bloque tipo *Matlab Function*, que se colocará a continuación de los bloques dedicados a los canales A y B del encoder. Dicho bloque tendrá un tiempo de ejecución igual al establecido para los bloques del encoder anteriores, 4.57, y entregará en su salida "1" si el sentido de giro es horario, es decir, si en el instante en el que se produzca un flanco de subida del canal B, el canal A también está en alto; y "-1" si el motor gira en sentido antihorario.

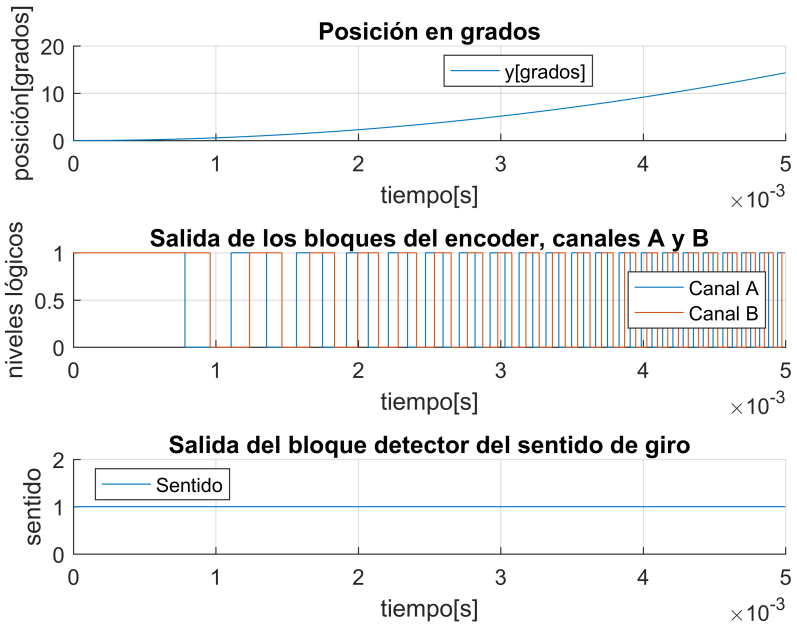
Para ello será necesario emplear dos variables del tipo *persistent*, aquellas que mantienen su valor ejecución tras ejecución. Una de ellas se usará para almacenar los valores de entrada provenientes del canal B del encoder y la otra para almacenar el valor de salida que deberá ser enviado ejecución tras ejecución. Serán llamadas *buffer_B* y *buffer_sentido*, respectivamente.

La variable encargada de almacenar los valores provenientes del canal B será un vector *buffer* de una fila y dos columnas. La primera columna se reservará para el valor leído anterior y la segunda para el valor actual. De tal forma que, en primer lugar se leerá el valor actual proveniente del canal B del encoder y se almacenará en la segunda columna del *buffer*. Luego, se comprobará si el valor anterior proveniente del canal B del encoder es un "0" y al mismo tiempo si el valor actual es un "1". Es decir, se comprueba si en la primera columna del vector el valor es un "0" y además si en la segunda columna el valor es un "1". Lo cual implica, en caso de ser cierto, que se ha detectado un flanco de subida en el canal B del encoder.

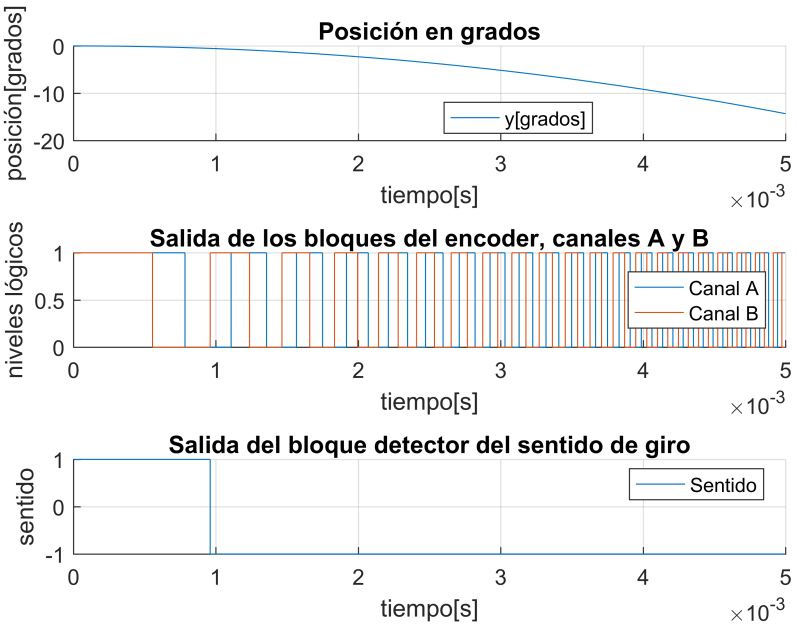
En caso afirmativo, se comprobará si en ese instante el canal A se encuentra en un nivel lógico alto. En caso de ser cierto, es decir se ha recibido un "1" por la entrada dedicada al canal A; según lo establecido en el párrafo anterior, el motor se encuentra girando en sentido horario y por tanto se tendrá que almacenar un "1" en la variable encargada de guardar el valor de salida. En el caso opuesto, el motor se encuentra girando en sentido antihorario y por tanto se almacenará un "-1" en *buffer_sentido*.

A continuación, el valor recibido del canal B del encoder pasa a ser almacenado en la primera columna del *buffer* de los valores de B, pasando a ser el valor "anterior". Finalmente se enviará el valor almacenado en la variable que almacena los valores de salida, *buffer_sentido*. El proceso al completo se muestra en el código 7.7.

Ante una entrada de velocidad en rampa, con pendiente positiva, tras ser integrada y haber sido aplicado el factor de conversión a grados, se obtiene las respuestas de los canales del encoder y el sentido de giro mostrados en la figura 4.14a. En ella se observa como los flancos de subida del canal B se encuentran con momentos en los que la señal del canal A se encuentra en alto. Interpretando un sentido de giro horario, representado como un "1" a la salida del bloque.



(a) Entrada positiva de posición en grados.



(b) Entrada negativa de posición en grados.

Figura 4.14: Salida del bloque detector del sentido de giro.

En el caso de una entrada en rampa con pendiente negativa y bajo las mismas condiciones se obtiene la figura 4.14b. En este caso, los flancos de subida del canal B coinciden con instantes en los que la señal del canal A se encuentra en nivel bajo. Interpretando un sentido de giro antihorario, representado como un "-1" a la salida del bloque.

Como se indicó anteriormente, la programación realizada del encoder representa un comportamiento ideal del encoder donde cualquier pequeño movimiento del motor es registrado por éste. Teniendo una resolución tan alta como *Matlab* pueda calcular. Por tanto en los primeros instantes en los cuales se produce el arranque del motor, tanto el canal A como el canal B se encontrarán en una zona transparente del disco de graduación hasta que se alcancen los 0.35 grados, en el caso del canal A, y los 0.175 grados, en el caso del canal B. Por ello, en el arranque, siempre coincidirá el flanco de subida del canal B con un instante en el que el canal A se encuentra en un nivel alto interpretando un sentido de giro horario. Ésto se muestra en la figura 4.14b. En el próximo flanco de subida del canal B esta imprecisión es corregida.

4.5. Simulación Frecuencímetro

Para traducir las señales proporcionadas por los canales del encoder a un dato de velocidad de giro del motor es necesario obtener la frecuencia o el periodo de éstas. Ambas magnitudes son válidas debido a la reciprocidad entre ellas, pero el término frecuencímetro es mucho más común. Los frecuencímetros son instrumentos analógicos o digitales capaces de determinar la frecuencia de una señal. Son de interés para la simulación que se está llevando a cabo los frecuencímetros digitales. Los frecuencímetros digitales son aquellos capaces de determinar la frecuencia por conteo de los pulsos de una señal en periodos patrones determinados[2].

Existen dos principios básicos de funcionamiento de este tipo de instrumentos: uno de ellos parte de contar pulsos durante un tiempo constante y conocido. El otro consiste en usar el periodo de la señal incógnita como el que permite la cuenta de pulsos de una señal de frecuencia mucho mayor a la estimada para la señal incógnita[2]. Este segundo método es el más adecuado para el caso de señales de entrada como las que proporciona el bloque del encoder, ya que, será el periodo de éstas el encargado de determinar la velocidad de giro. Un periodo de una de las señales proporcionadas por uno de los canales del encoder, equivale a un giro de 0,7 grados según el paso angular de graduación del encoder empleado, (4.55). El tiempo empleado en completar dicho giro vendrá determinado por la duración del periodo. Siendo el incremento de posición y el tiempo conocidos es posible calcular la velocidad. Por tanto, es el método que se va a implementar en la simulación.

Con este método, es necesario una conversión matemática que relacione

el número de pulsos contados de la señal patrón con el periodo de la señal incógnita. Dicha relación matemática es la siguiente[2]:

$$N_{pulsos} = \frac{T_x}{T_{reloj}} \tag{4.70}$$

Donde N_{pulsos} es el número de pulsos contados, T_x es el periodo de la señal incógnita y T_{reloj} es el periodo de la señal de reloj patrón de la cual se cuentan los pulsos. Despejando se obtiene el periodo de la señal incógnita como:

$$T_x = N_{pulsos} T_{reloj} \tag{4.71}$$

Para ilustrar gráficamente su funcionamiento, se toma como señal de entrada incógnita una señal cuadrada de frecuencia igual a 1Hz. La frecuencia usada para la señal de reloj patrón será de 10 Hz. No son suficientes órdenes de magnitud de diferencia como para garantizar una buena resolución pero son suficientes para facilitar una claridad visual. Se realizará el conteo de pulsos durante la duración de cada periodo de la señal incógnita. Se muestra únicamente un periodo para facilitar la comprensión del proceso. El resultado se muestra en la figura 4.15.

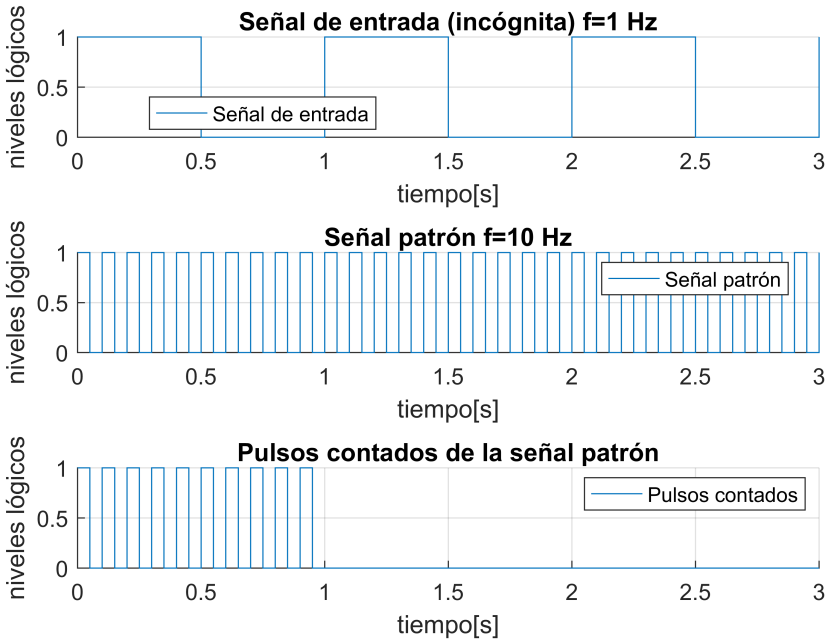


Figura 4.15: Ejemplo de funcionamiento del frecuencímetro.

Como se puede observar se han contado diez pulsos de la señal de reloj patrón en un periodo de la señal incógnita. Se obtiene:

$$T_x = N_{pulsos} T_{reloj} \tag{4.72}$$

$$T_x = 10 \cdot \frac{1}{10} \tag{4.73}$$

$$T_x = 1 \text{ [segundo]} \tag{4.74}$$

$$f_x = 1 \text{ [Hz]} \tag{4.75}$$

Por tanto, para el caso de las señales provenientes de los canales del encoder, se deberá tomar una de ellas como señal de entrada de periodo incógnito. Cuanto mayor sea la velocidad del motor mayor será la frecuencia de los pulsos y por lo cual menor será su periodo. Sabiendo que debido al paso angular de graduación del encoder, (4.55), el eje del motor se habrá movido 0,7 grados cada periodo de la señal de entrada. Se obtiene la velocidad de giro del eje del motor a partir de los pulsos de entrada de uno de los canales del encoder como:

$$w = \frac{0{,}7}{T_x} \text{ [grados/segundo]} \tag{4.76}$$

En grados por segundo. Debido a que la variable controlada, la velocidad angular, se requiere que esté expresada en radianes por segundo, será necesario aplicar un factor de conversión. Resultando:

$$w = \frac{0{,}7}{T_x} \cdot \frac{\pi}{180} \text{ [radianes/segundo]} \tag{4.77}$$

Para implementar este procedimiento en la simulación se usará un bloque del tipo *Matlab Function*. Se colocará a continuación de los bloques dedicados a los canales del encoder. Deberá tener un periodo de ejecución menor que el empleado en los bloques mencionados, (4.57). Teniendo en cuenta que se deberán realizar operaciones por cada valor binario proveniente de uno de los canales del encoder, el periodo de ejecución del bloque frecuencímetro deberá ser la mitad del periodo de ejecución de éstos. Debido a que únicamente se pueden establecer tiempos de ejecución que sean multiplos del paso integral de simulación, se tiene:

$$T_frec = 1 \cdot 10^{-6} \text{ segundos.} \tag{4.78}$$

Este bloque dedicado a la simulación del frecuencímetro tendrá tres entradas:

- En una de ellas irán los datos de entrada provenientes de uno de los canales del encoder.
- Otra, estará dedicada a recibir los datos de dirección enviados por el bloque que determina la dirección de giro. Éstos se usarán para establecer el signo de la velocidad calculada.

- La última entrada recibirá un escalón unitario que actuará al mismo tiempo que el *Set-Point* de velocidad empleado en la simulación. El propósito de esta entrada es empezar el conteo de pulsos al mismo tiempo que se aplique el *Set-Point*, sin importar la forma que tenga éste (escalón, rampa o seno). Esto es necesario debido a que, según se estableció en el apartado del encoder, cuando el disco de graduación no haya girado, es decir, se encuentre en una posición igual a cero grados; se interpreta como si se encontrase en una zona transparente y por lo tanto, mientras esté parado, dará como resultado un pulso inicial más ancho de lo debido, introduciendo un error en los cálculos. Esta entrada se denominará "Temporizador".

Además, recibirá como parámetro el periodo de ejecución del bloque frecuencímetro, (4.78).

No es necesario dedicar una entrada para la señal de reloj patrón, ya que, se usará el propio tiempo de ejecución del bloque como patrón. Es decir, cuando se haya detectado un flanco de subida de la señal de entrada, se aumentará en uno el valor de una variable contador. Ésta seguirá aumentando su valor de uno en uno, ejecución tras ejecución, hasta que se reciba de nuevo un flanco de subida en la señal de entrada. Lo cual indicará que un periodo completo ha sido enviado. Se tiene por tanto el mismo efecto que contando los pulsos de una señal de reloj patrón. El valor almacenado en la variable contador será equivalente a contar pulsos de una señal patrón.

Por lo tanto, el periodo T_{reloj} de la ecuación (4.71) será igual a T_{frec}, (4.78).

Serán necesarias cinco variables del tipo *persistent*, aquellas que mantienen su valor ejecución tras ejecución. Se definen a continuación:

- Una de ellas será un vector *buffer*, inicialmente lleno de ceros, de una fila y dos columnas. En él se almacenará, en la segunda columna, el valor actual recibido del canal del encoder usado como entrada. En la primera columna se almacenará el valor anterior. Se denominará *buffer_entrada*.

- Otra será la ya mencionada variable contador de pulsos. Aumentará su valor de uno en uno ejecución tras ejecución. Se llamará *N*.

- La siguiente, será otra variable contador encargada de contar el número de flancos de subida detectados. Se empleará para comprobar si ha terminado de recibir un periodo completo de la señal de entrada. Se llamará *contador_flancos*.

- Para comenzar la cuenta de pulsos, se establecerá como *true* una variable *booleana*, denominada *flanco_detectado* cuando se detecte un flanco de subida. De tal forma que si al mismo tiempo se recibe un "1" por

la entrada en escalón que actúa al mismo tiempo que el *Set-Point*, se comenzará a aumentar el tamaño de *N*.

- Finalmente, será necesario una variable que almacene los valores de velocidad calculados ejecución tras ejecución. Dicha variable se denominará *w*.

El proceso realizado en la programación es el siguiente:

- En primer lugar se lee el valor recibido por uno de los canales del encoder. Dicho valor se almacena en la segunda columna del vector *buffer* de los valores de entrada. La segunda columna del vector se reserva para los valores actuales.

- A continuación se comprueba si el valor recibido en la ejecución anterior, es decir, el almacenado en la primera columna del vector, es un "0". Y al mismo tiempo, se comprueba si el valor actual es un "1". En caso afirmativo se habrá detectado un flanco de subida de la señal de entrada, y por tanto se aumenta en uno, el valor de la variable contador de flancos, *contador_flancos*. También se establecerá como *true* el valor de la variable *flanco_detectado*.

- Luego, se comprueba si el número de flancos de subida contados es igual a dos. En caso afirmativo, se habrá terminado de recibir un periodo completo de la señal de uno de los canales del encoder. Por tanto, se resolverán a continuación las ecuaciones (4.71) y (4.77). Para salvar una posible división por cero en la ecuación (4.71), será necesario establecer que si efectivamente el valor de T_x es cero, no se realicen operaciones y el resultado de la velocidad angular sea directamente cero. A continuación, se reinician tanto los valores de N como de *contador_flancos* a "1", para que estén listos para el siguiente periodo de la señal de entrada. El motivo por el que se reinicia su valor a "1" en vez de a "0", es porque este proceso se realiza a continuación de detectar un flanco de subida. Es necesario contar con éste.

- Independientemente del resultado de los procedimientos anteriores, se comprobará si el valor de la entrada "Temporizador", que, como recordatorio, se activa al mismo tiempo que el establecido en el *Set-Point*; es "1". Si al mismo tiempo, el valor de la variable *flanco_detectado* es *true*, se aumentará el valor de *N*.

- El valor actual se almacena en la primera columna del vector *buffer* pasando a ser el valor anterior. Finalmente, se envía el valor de velocidad multiplicado por su sentido, según se reciba por la entrada correspondiente.

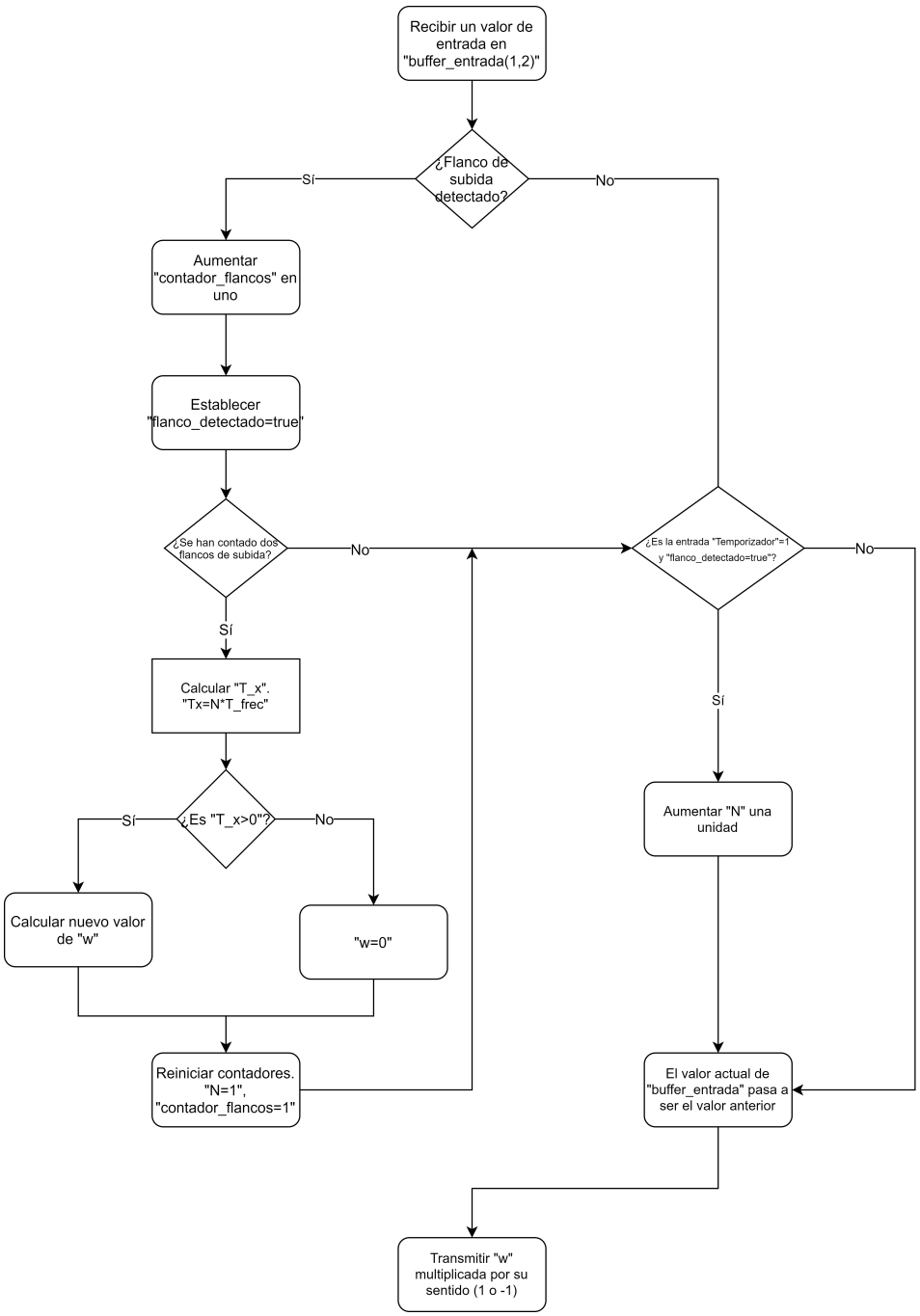


Figura 4.16: Diagrama de flujo del bloque frecuencímetro.

El diagrama de flujo de este procedimiento se representa en la figura 4.16. El código resultante se muestra en 7.8

Para mostrar el funcionamiento de este bloque en la simulación, se coloca a continuación de los bloques dedicados a los canales del encoder y del bloque encargado de determinar la dirección de rotación. Para hallar el sentido de giro son necesarios ambos canales del encoder, pero para determinar la velocidad solo es necesario uno de ellos. Se usará el canal "A" como entrada para determinar la velocidad. Será éste el único canal que se muestre en las gráficas de resultados con el objetivo de garantizar una mayor claridad visual. También es importante señalar que, al contrario de representaciones gráficas anteriores, se omite mostrar la señal de entrada una vez integrada y haber sido aplicado el factor de conversión a grados, procedimiento que es requerido para el correcto funcionamiento del encoder. Esto se debe a que para mostrar la velocidad calculada por el encoder en comparación con la velocidad de entrada, dicha gráfica no es relevante. También se omite mostrar el resultado del bloque encargado de determinar el sentido de rotación. Ya que únicamente entrega información de cuando su salida es "1" o "-1", disminuyendo considerablemente el espacio disponible para mostrar el resto de gráficas que tienen mayor relevancia.

Una vez tomadas las anteriores consideraciones, se obtiene para una entrada de velocidad en rampa con pendiente positiva la figura 4.17.

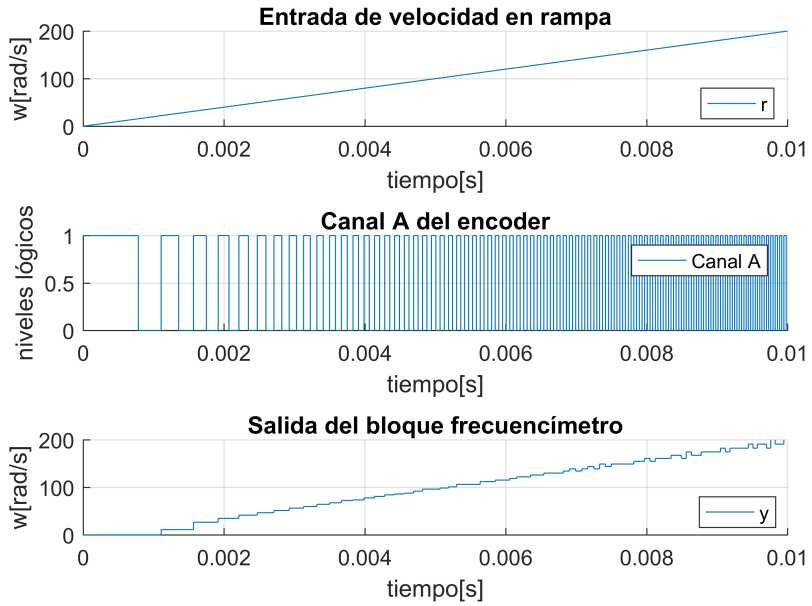


Figura 4.17: Respuesta del bloque frecuencímetro ante una entrada de velocidad en rampa con pendiente positiva.

Como se puede observar en ella, a medida que aumenta la frecuencia de los pulsos de entrada del canal "A" del encoder, mayor es la velocidad de salida del bloque frecuencímetro. Como se comentó en el apartado dedicado al funcionamiento de los canales del encoder, a velocidades bajas se producirá un error de precisión debido a la mayor amplitud que toma el pulso, como se observa al inicio de la respuesta. Hasta que el siguiente flanco de subida es detectado, el cálculo de la velocidad no es realizado. Siendo ésta la fuente del error de precisión, que es corregido a medida que aumenta la velocidad.

Además del error introducido por el encoder, también se introduce una imprecisión en el cálculo de la velocidad angular. Cuanto mayor sea la velocidad, menor será el valor de la variable contador N. Debido a que su valor únicamente puede ser un número entero, a velocidades altas se requerirá una mayor resolución para determinar con exactitud el periodo de la señal de entrada del encoder. Es decir, N podrá tener valores iguales a "1" o "2" pero no "1,1".

Para el caso de una entrada en rampa con pendiente negativa, se obtiene la figura 4.18. En la cual se ha tenido en consideración el cambio en el sentido de giro.

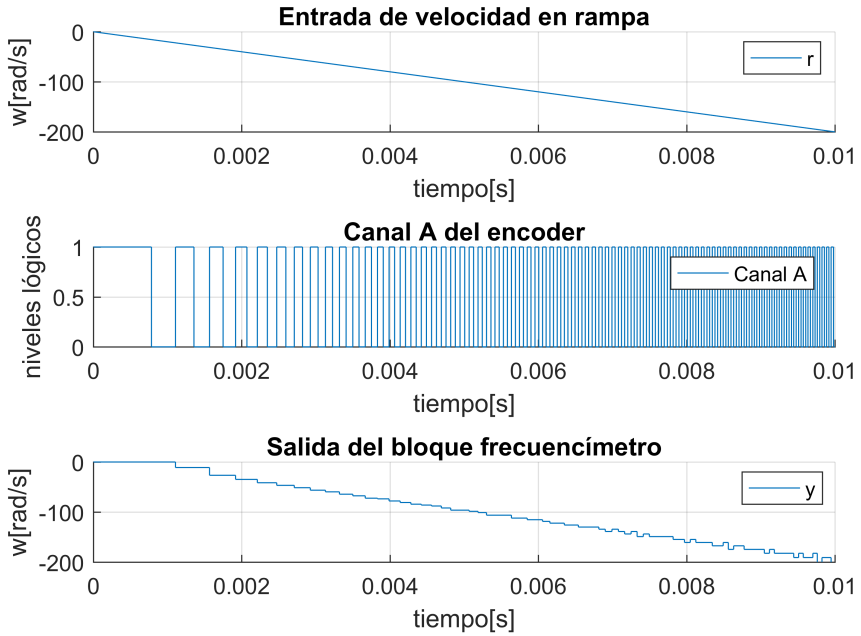
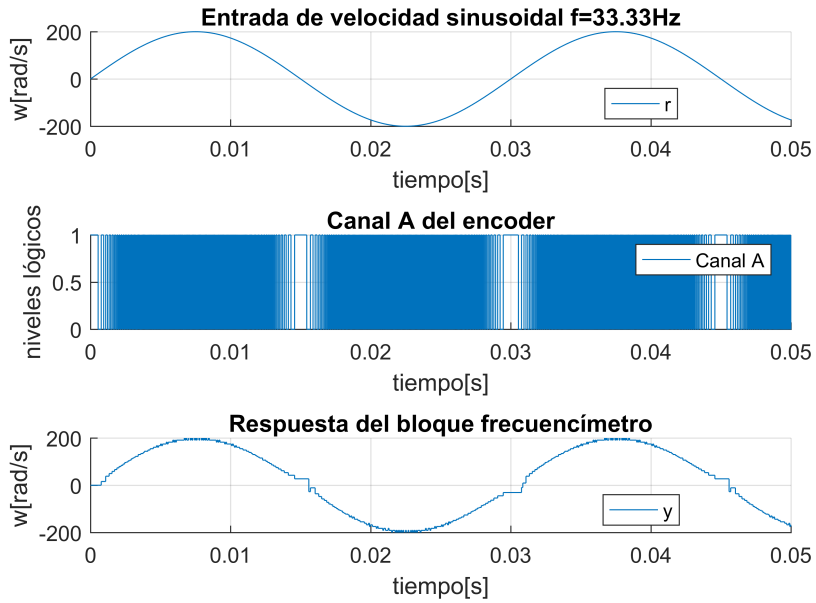


Figura 4.18: Respuesta del bloque frecuencímetro ante una entrada de velocidad en rampa con pendiente negativa.

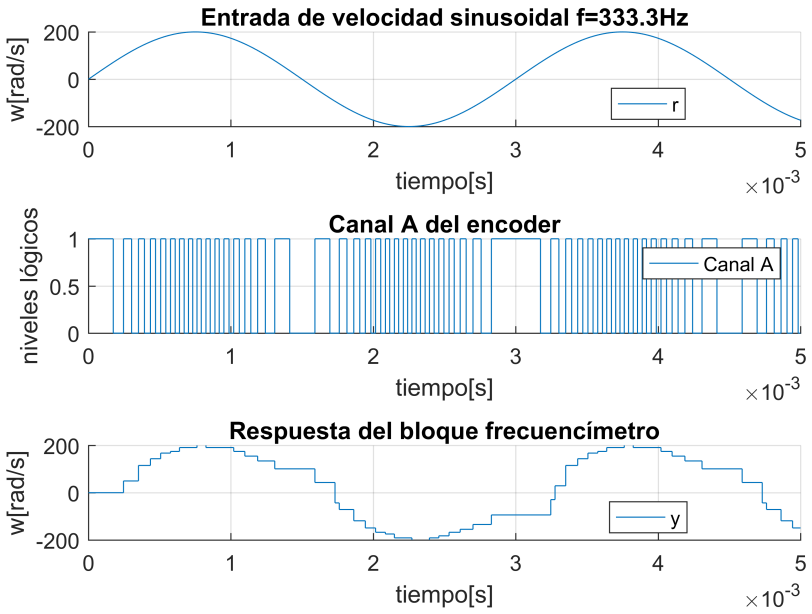
Es importante apuntar que el funcionamiento de este bloque frecuencímetro estará limitado por la frecuencia a la que se ha establecido el funcionamiento de los bloques de los canales del encoder. Cuanto más se acerque la frecuencia de la señal de entrada a la frecuencia de ejecución de los bloques del encoder, mayor cantidad de datos serán perdidos y por tanto peor precisión tendrá su respuesta. Para ilustrar este comportamiento, en primer lugar se somete el conjunto encoder-detector de sentido de giro-frecuencímetro a una entrada de velocidad sinusoidal de frecuencia igual a 33,333Hz. Teniendo en cuenta que la frecuencia a la que se ejecuta los bloques dedicados a los canales del encoder es de $1/T_enc = 1/3 \cdot 10^{-}6 = 333{,}333$Khz, se trata de una frecuencia diez mil veces menor. El resultado se muestra en la gráfica 4.19a, en la que hay muy poca pérdida de precisión a excepción de las zonas con velocidades bajas.

Si se aumenta la frecuencia a una mil veces menor que la de ejecución de los bloques del encoder, es decir, 333,333Hz se obtiene la figura 4.19b, en la que la pérdida de precisión es significativa.

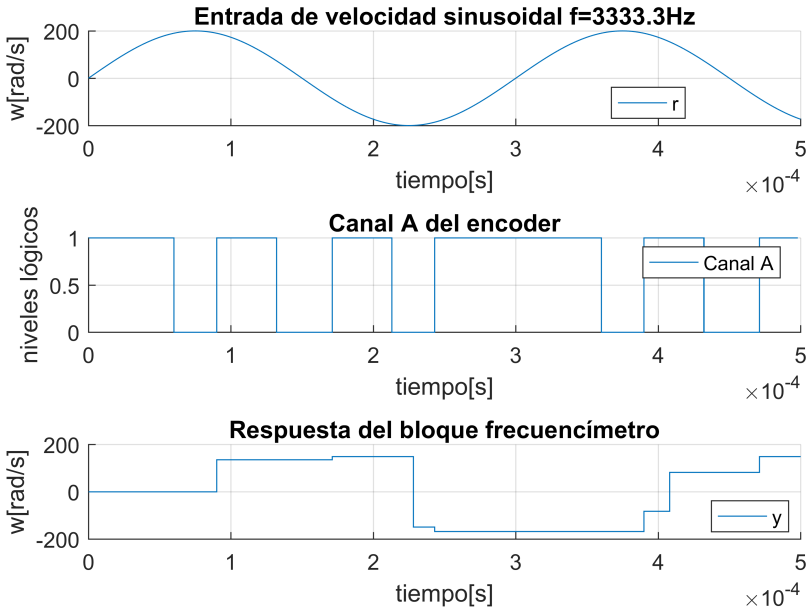
Finalmente ante una entrada con una frecuencia cien veces menor a la original, es decir 3,33KHz resulta la gráfica 4.19c, en la cual la pérdida de precisión es importante. Por lo que, es un aspecto a tener en cuenta cuando se realice la simulación de los controladores con todos los bloques extras (PWM, PFM, encoder y frecuencímetro) integrados. Deberá ser ajustado el periodo de ejecución de los bloques encoder y frecuencímetro si es necesario.



(a) Entrada de velocidad sinusoidal de f=33.333Hz.



(b) Entrada de velocidad sinusoidal de f=333.333Hz.



(c) Entrada de velocidad sinusoidal de f=3333.33Hz.

Figura 4.19: Respuestas del bloque frecuencímetro ante entradas sinusoidales

4.6. Simulación del controlador I-PD en tiempo discreto

De manera análoga a lo realizado en la simulación de los controladores en tiempo continuo, se simularán los controladores en tiempo discreto por etapas añadiendo bloques que acercarán la simulación a su implementación en hardware. En primer lugar, se comenzará con el controlador I-PD con su salida limitada al voltaje nominal del motor de ±6 voltios y usando la modulación por ancho de pulsos (PWM), a continuación, para el mismo *Set-Point* se obtendrá la respuesta empleando la modulación por la frecuencia de los pulsos (PFM) y se presentarán gráficas comparativas de ambas respuestas. Luego, se mostrará la respuesta obtenida por el bloque frecuencímetro y finalmente será ésta la que se introduzca en la realimentación del bucle en lazo cerrado. De nuevo, se ofrecerán gráficas comparativas entre modulación PWM y PFM. Los parámetros de la simulación iniciales junto con los valores de las acciones de control del controlador I-PD son los mostrados en la tabla 4.1.

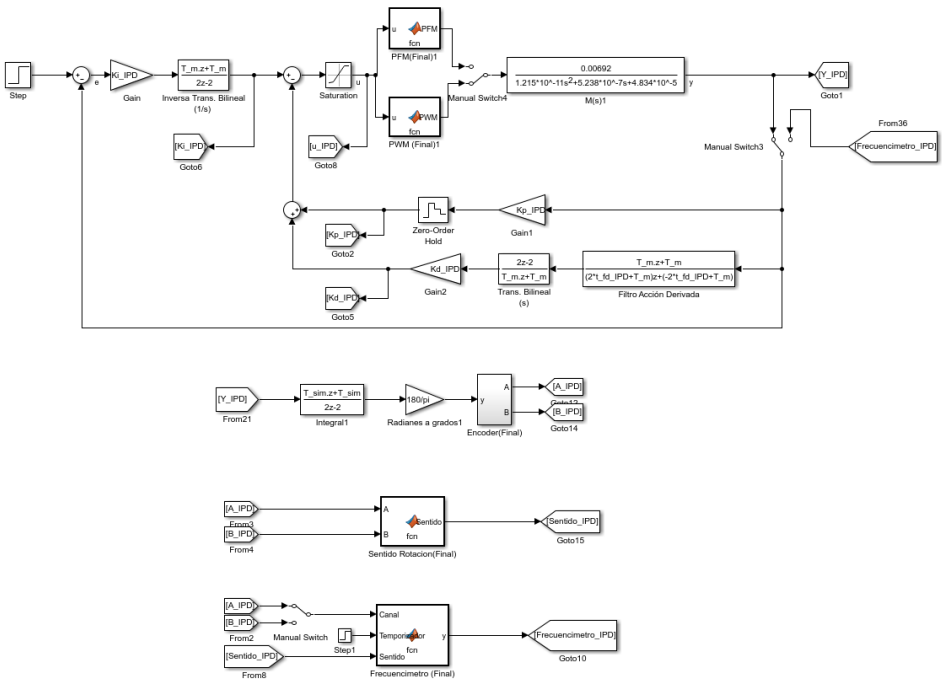


Figura 4.20: Conjunto completo con el controlador I-PD en tiempo discreto en *Simulink*.

El conjunto completo compuesto por la planta, el controlador y el resto de los bloques de tipo *Matlab function* realizados se muestran en la figura 4.20.

Parámetros de la simulación	Acciones de control
$T_sim = 5 \cdot 10^{-8}[s]$	$K_i = 923{,}699$
$T_m = 5 \cdot 10^{-6}[s]$	$K_p = 5{,}8296$
$T_res = 5 \cdot 10^{-7}[s]$	$K_d = 8{,}5838 \cdot 10^{-5}$
$t_on_PFM = 5 \cdot 10^{-7}[s]$	$t_{fd} = 1{,}4725 \cdot 10^{-6}$
$T_enc = 3 \cdot 10^{-6}[s]$	
$T_frecc = 1 \cdot 10^{-6}[s]$	

Cuadro 4.1: Parámetros de la simulación y valores de partida del controlador I-PD en tiempo discreto.

4.6.1. Simulación sin el bloque frecuencímetro en la realimentación

La respuesta del sistema ante una entrada en escalón de velocidad en radianes por segundo, con una amplitud de cien, para ambas modulaciones y con los valores de las acciones de control obtenidos en 3.53, es la representada en la figura 4.21.

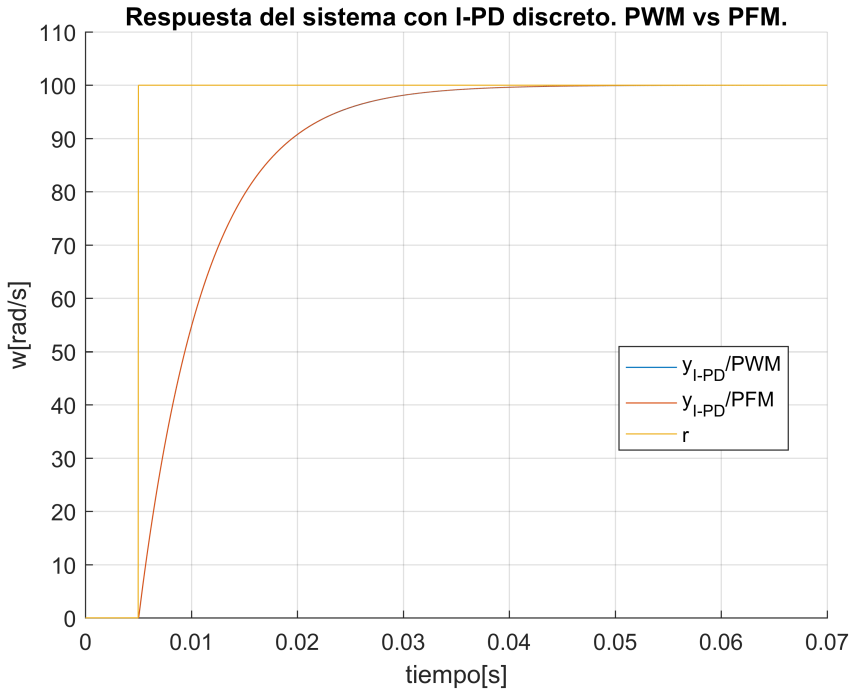
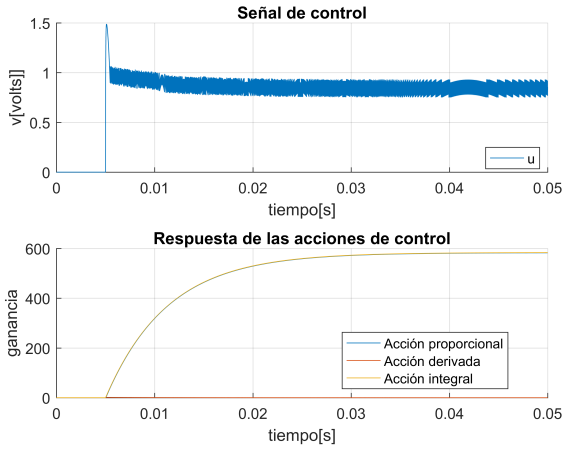


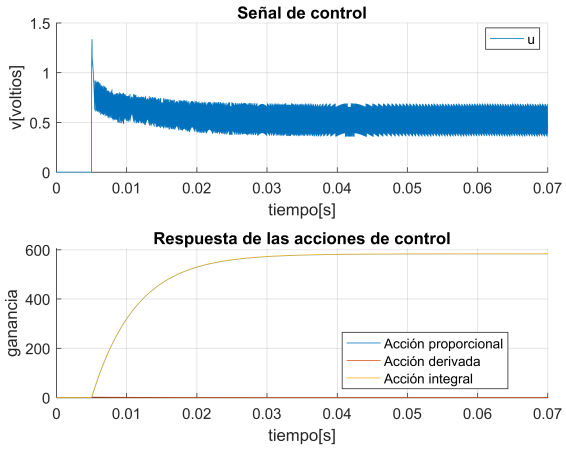
Figura 4.21: Respuesta del sistema con controlador I-PD en tiempo discreto. PWM frente a PFM. *Set-Point* en escalón de amplitud 100[rad/s].

En ella se observa una respuesta del sistema sobreamortiguada muy similar a la obtenida con el controlador I-PD en tiempo continuo, 3.6b, para ambos métodos de modulación, entre los cuales, existen tan pocas diferencias que sólo es posible ver un único trazo en la gráfica. Con respecto a la respuesta en tiempo continuo, presenta una pequeña diferencia en el tiempo de subida. En el caso del controlador I-PD en tiempo continuo el tiempo de subida era de 14,515 milisegundos y en el caso del controlador en tiempo discreto, para ambas modulaciones, es de 14,521 milisegundos.

En la figura 4.22a y 4.22b se refleja la respuesta de las acciones integral, proporcional y derivativa ante la señal de control para el caso de la modulación PWM y PFM, respectivamente.



(a) Modulación PWM.



(b) Modulación PFM.

Figura 4.22: Respuesta de las acciones de control frente a la señal del controlador I-PD en tiempo discreto. Entrada en escalón de amplitud 100[rad/s].

En ellas se muestra como la mayor parte del peso del control cae sobre las acciones proporcional e integral.

En la figura 4.23 se representa la respuesta del sistema en lazo cerrado frente a la velocidad obtenida por el bloque frecuencímetro para ambos métodos de modulación.

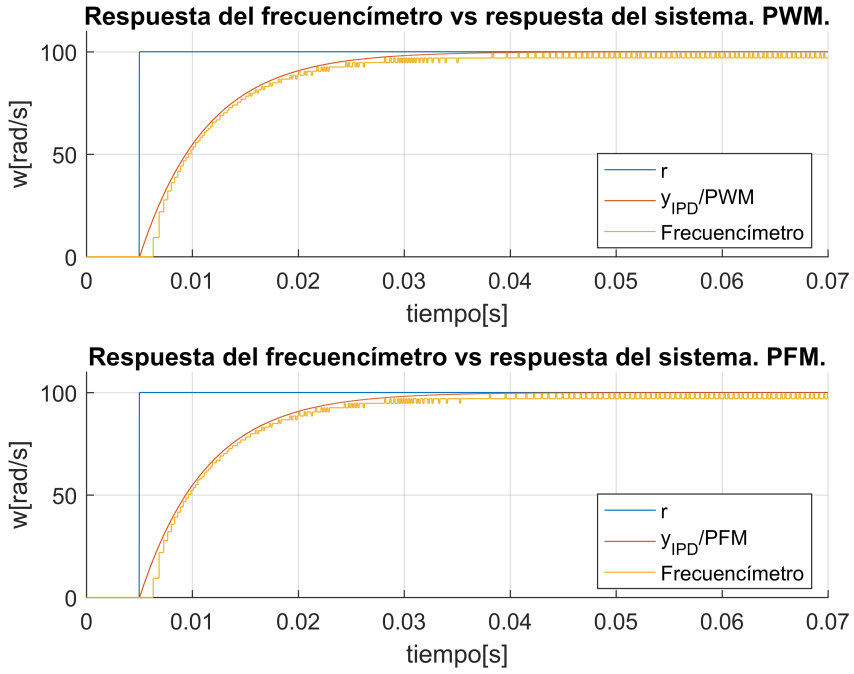


Figura 4.23: Velocidad obtenida por el frecuencímetro frente a la respuesta del sistema en lazo cerrado, con controlador I-PD(z) y entrada en escalón.

En ella se observa que los mayores errores de precisión ocurren en las zonas donde la velocidad es inferior (de 0 a 40 radianes por segundo). Para valores superiores de velocidad *Set-Point*, los errores de precisión del cálculo de la velocidad en el frecuencímetro debido a, tal y como se comentó en su apartado, la imposibilidad de la variable contador de tiempos de ser un número decimal; se hacen notar en las zonas donde la velocidad es más alta (en este caso, en la sección de la respuesta comprendida entre 80 a 100 radianes por segundo). A pesar de ello, son menos importantes que los errores en las zonas donde la velocidad es más baja. Por tanto, es posible preveer mayores imprecisiones en la respuesta en lazo cerrado en las zonas donde la velocidad sea más baja cuando se incluya el frecuencímetro en la realimentación, y en general, cuando se introduzcan *Set-Points* de velocidad baja.

Debido a que el *Set-Point* en escalón de 100 radianes por segundo no es

suficiente para hacer alcanzar al controlador su valor máximo de 6 voltios como se muestra en la figuras 4.22a y 4.22b, es de interés realizar de nuevo las gráficas anteriores para un valor más alto del *Set-Point* que sí lleve al controlador a su valor máximo. Por lo que, ante un *Set-Point* en escalón de 600 radianes por segundo se obtiene la gráfica representada en la imágen 4.24.

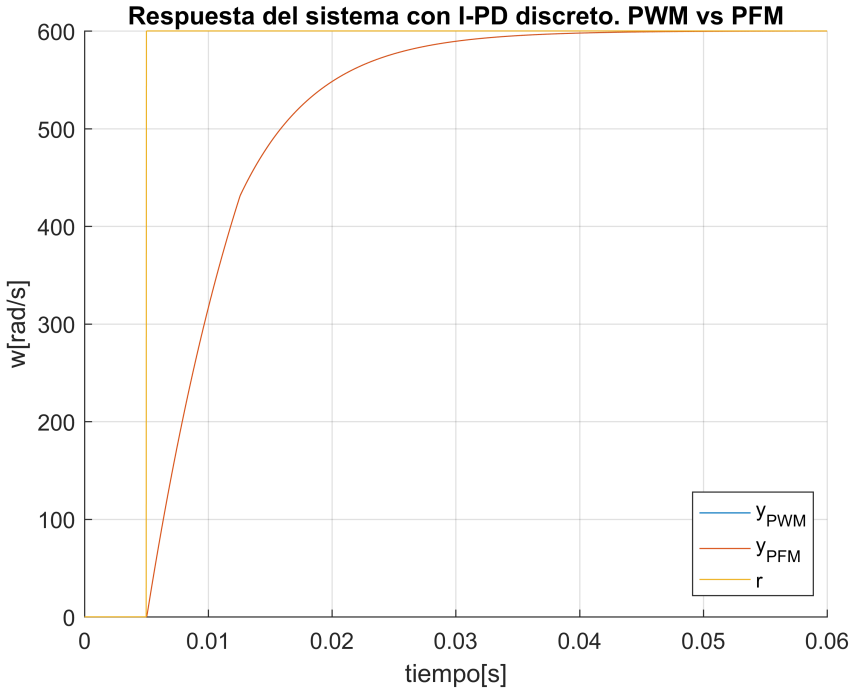
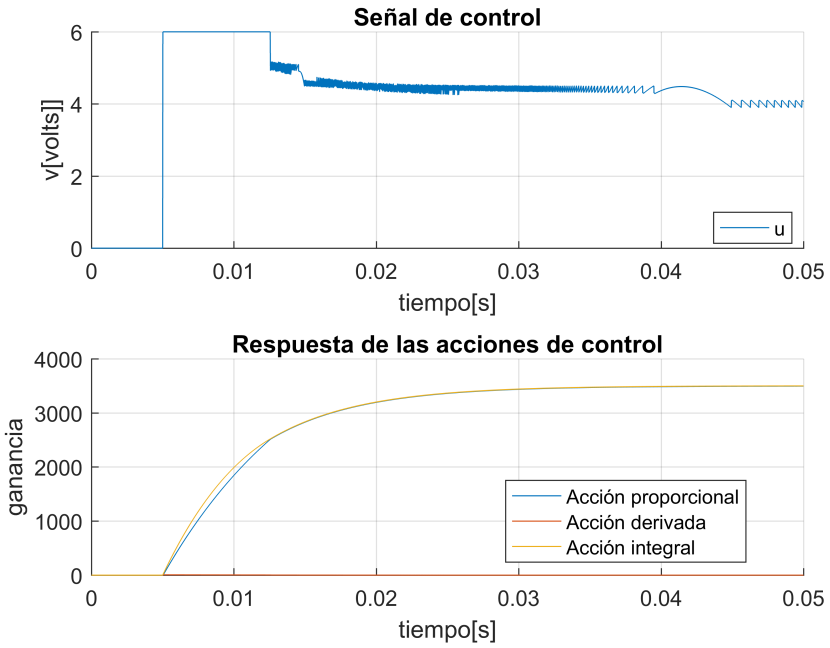


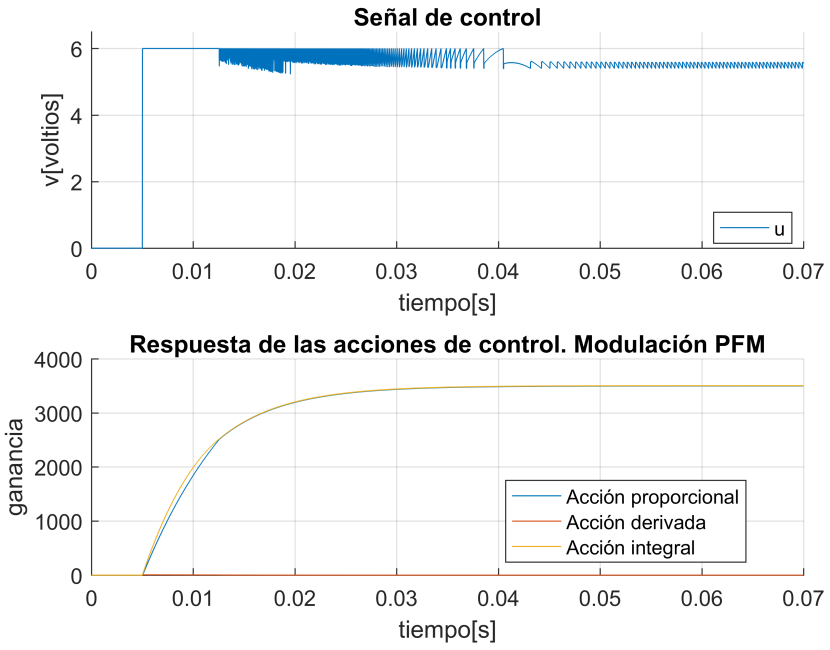
Figura 4.24: Respuesta del sistema en lazo cerrado. Controlador I-PD(z). PWM vs PFM. *Set-Point* en escalón de amplitud 600 [rad/s].

Obteniendo una respuesta sobreamortiguada con un tiempo de subida de 18,425 milisegundos

La respuesta de las acciones proporcional, integral y derivativa frente a la señal del controlador se representa para ambas modulaciones en las figuras 4.25a y 4.25b. En ellas se muestra como en este caso se lleva la señal de control al límite máximo de tensión. También se observa como en el caso de la modulación PFM el valor límite se mantiene más tiempo que en el caso PWM. Además, el valor de establecimiento de la señal de control en el caso de emplear modulación PFM es mayor que en el caso PWM, requiriendo por tanto, un mayor esfuerzo de control.



(a) Modulación PWM.



(b) Modulación PFM.

Figura 4.25: Respuesta de las acciones de control frente a la señal del controlador I-PD en tiempo discreto. Entrada en escalón de amplitud 600[rad/s].

La velocidad del sistema frente a la calculada por el bloque frecuencímetro se representa en la gráfica 4.26.

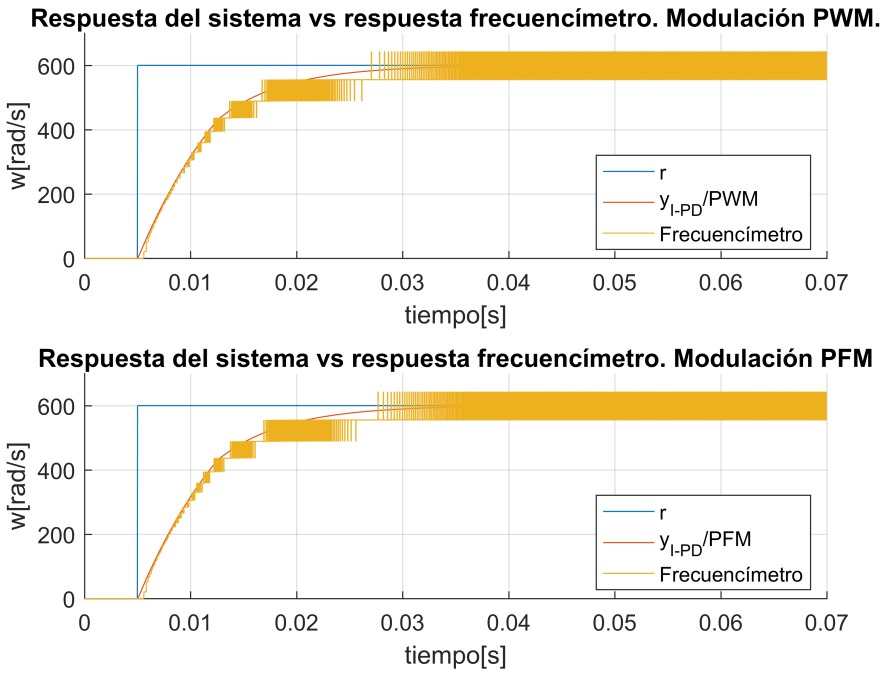


Figura 4.26: Respuesta del sistema frente a la velocidad obtenida por el frecuencímetro. Ambas modulaciones.

En ésta se observa como en las zonas donde la velocidad es inferior y en aquellas donde es mayor, se producen imprecisiones tal y como se expuso anteriormente: en las zonas de velocidad inferior (de 0 a 40 radianes por segundo, en concreto) se producen imprecisiones debido al mayor ancho de pulso de las señales del encoder y en las zonas donde la velocidad es mayor (de 350 a 600 radianes por segundo) se producen debido a la falta de resolución de la variable contador en el frecuencímetro. No ocurre lo mismo en velocidades medias. En este caso, al contrario que empleando un *Set-Point* de 100 radianes por segundo, las imprecisiones en las secciones donde la velocidad es mayor son importantes.

En el caso de emplear los mismos valores de *Set-Point* pero con signo negativo, se obtienen las mismas respuestas con amplitud negativa.

4.6.2. Simulación con el bloque frecuencímetro en la realimentación

A continuación, se introducirá el bloque del frecuencímetro en la realimentación del lazo cerrado en la simulación (ver imagen 4.20) y se le

introducirán al sistema los mismos valores de *Set-Point* que en los casos en los que no se empleó el bloque del frecuencímetro. De nuevo, se ofrecerán gráficas comparativas entre ambos métodos de modulación.

Comenzando con un *Set-Point* de 100 radianes por segundo, se produce la respuesta mostrada en la gráfica 4.27.

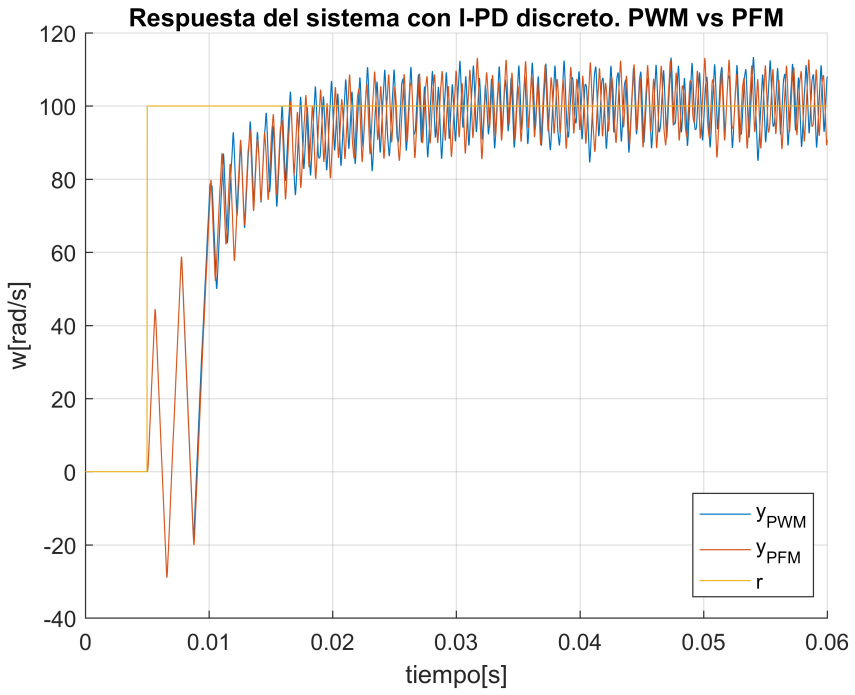


Figura 4.27: Respuesta del sistema con controlador I-PD en tiempo discreto con frecuencímetro en la realimentación. PWM frente a PFM. *Set-Point* en escalón de amplitud 100[rad/s].

En ella se observa como las incertidumbres introducidas al sistema por el conjunto encoder-frecuencímetro, provocan discontinuidades en la respuesta para ambos métodos de simulación en forma de dientes de sierra. Llegados a este punto, es necesario comprobar si se trata únicamente una falta de precisión del conjunto encoder-frecuencímetro o de si es necesario un ajuste fino de las acciones de control que tengan en cuenta las no linealidades introducidas al sistema. Para intentar compensar las imprecisiones, se disminuye el tiempo de ejecución de los bloques encoder y frecuencímetro, llevándolos a ejecutarse una frecuencia más alta, es decir, realizan más operaciones para un único valor proveniente de la señal de control. Se seleccionan unos tiempos de ejecución de:

$$\begin{aligned} T_enc &= 3 \cdot 10^{-8} \\ T_frec &= 1 \cdot 10^{-8} \end{aligned} \tag{4.79}$$

Intentado mantener , siendo los originales los hallados en (4.57) y (4.78), respectivamente. Al disminuir el tiempo de ejecución de los bloques es necesario también disminuir el paso de integración de la simulación, que pasa a ser de:

$$T_sim = 5 \cdot 10^{-9} \tag{4.80}$$

Que provoca la desventaja de necesitar un tiempo más elevado para realizar simulaciones de tiempos muy pequeños, como $0,1$ segundos.

En la gráfica 4.28 se muestra una comparativa de la respuesta del sistema con el conjunto encoder-frecuencímetro funcionando a sus tiempos de ejecución originales ($T_enc = 3 \cdot 10^{-6}$, $T_frec = 1 \cdot 10^{-6}$ y $T_sim = 5 \cdot 10^{-8}$) y con los reajustados, ante un *Set-Point* de 100 radianes por segundo, con ambas formas de modulación. No muestra ninguna mejoría.

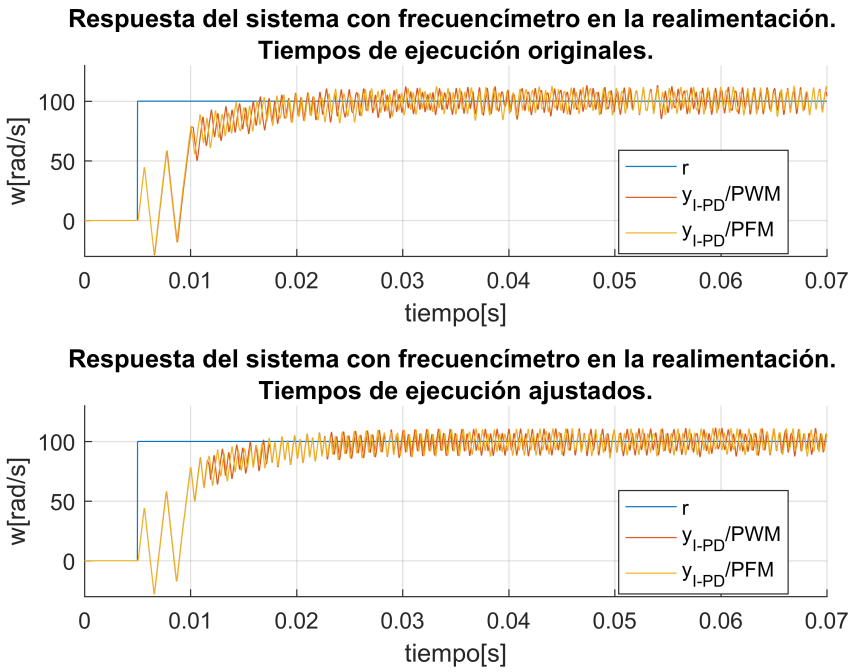


Figura 4.28: Respuesta del controlador IPD(z) con frecuencímetro. Comparativa de tiempos de ejecución. *Set-Point* en escalón de 100 rad/s.

Sin embargo, en el caso de un *Set-Point* de 600 radianes por segundo, se obtiene la figura 4.29, en la que la mejoría es notable. Con los tiempos de ejecución originales, las imprecisiones introducen discontinuidades en la respuesta además de un error en régimen permanente. Por tanto, son los valores de las acciones de control los responsables no poder hacer frente a las incertidumbres introducidas por el conjunto encoder-frecuencímetro a velocidades bajas como 100 radianes por segundo. Es necesario un ajuste

fino de las acciones de control válido para ambos *Set-Points*.

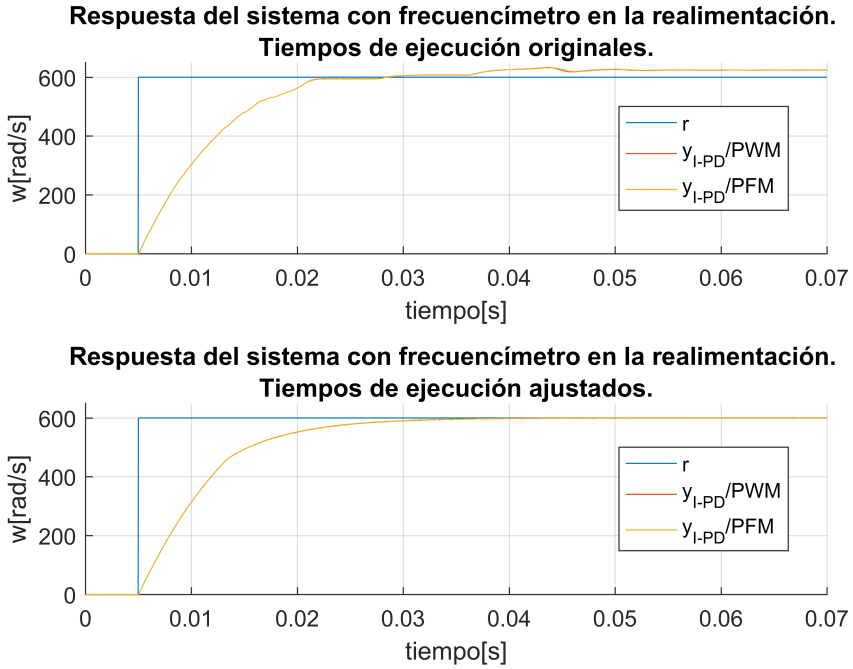


Figura 4.29: Respuesta del controlador IPD(z) con frecuencímetro. Comparativa de tiempos de ejecución. *Set-Point* en escalón de 600 rad/s.

4.6.2.1. Ajuste fino del controlador I-PD con frecuencímetro

El procedimiento llevado a cabo para el ajuste fino del controlador es el siguiente: Partiendo de un *Set-Point* de 100 radianes por segundo se ajustarán los valores de las acciones de control ya obtenidos, (3.53), con el bloque frecuencímetro en la realimentación empleando los tiempos de ejecución ajustados (4.79) con el objetivo de eliminar las discontinuidades. Cuando se encuentre un conjunto de valores válidos para dicho *Set-Point*, se ajustarán de nuevo para una entrada de 600 radianes por segundo. El proceso se repetirá hasta llegar a un compromiso entre eliminar las discontinuidades con un *Set-Point* de 100 radianes por segundo y una respuesta adecuada (sin sobreoscilación y el menor tiempo de subida posible) con uno de 600 radianes por segundo.

Se comenzará disminuyendo la acción proporcional, al contrario que en el ajuste a los valores de presintonía anterior, ya que en la respuesta con *Set-Point* de 100 radianes por segundo, a pesar de las discontinuidades, no existe un error en régimen permanente como tal. A medida que el valor de la acción proporcional disminuye, desaparecen las discontinuidades pero

aumenta la sobreoscilación, por lo que se compensa disminuyendo el valor de la acción integral hasta encontrar unos valores válidos.

Una vez se hayan obtenido unos valores adecuados, se comprueba la respuesta para un *Set-Point* de 600 radianes por segundo. Se obtiene una respuesta con sobreoscilación, por lo que es necesario disminuir el valor de la acción integral y reajustar de nuevo para el caso de un *Set-Point* de 100 radianes por segundo.

Tras varias iteraciones, se obtienen unos valores de ajuste fino de:

$$\begin{aligned} K_i &= 53{,}699 \text{ (Original } 923{,}699) \\ K_p &= 0{,}3296 \text{ (Original } 5{,}8296) \\ K_d &= 8{,}5838 \cdot 10^{-5} \text{ (Original } 8{,}5838 \cdot 10^{-5}) \\ t_{fd} &= (Kd/Kp)/10 = 2{,}6043 \cdot 10^{-5} (\text{ Original } 1{,}4725 \cdot 10^{-6}) \end{aligned} \tag{4.81}$$

Cada vez que fue necesario compensar la acción integral, su valor se redujo de cien en cien hasta llegar a 103,699. A partir de entonces se compensó disminuyendo de diez en diez. En el caso de la acción proporcional, se compensó disminuyendo de uno en uno hasta llegar a 0,8296, a partir del cual, se restó 0,1.

Se obtiene para una entrada en escalón de 100 radianes por segundo, la respuesta mostrada en la figura 4.30.

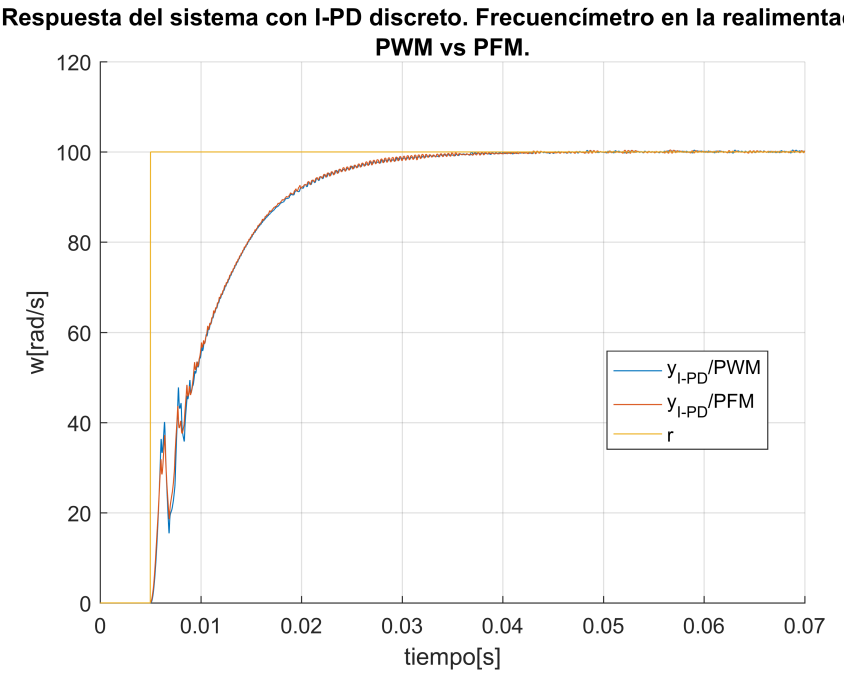


Figura 4.30: Respuesta del sistema con IPD(z) y frecuencímetro con el tiempo de ejecución ajustado *Set-Point* de 100 [rad/s].

En ella, se observa una notable mejoría en las zonas donde la velocidad es mayor con respecto a la figura 4.27. Sin embargo, las discontinuidades en zonas donde la velocidad es inferior no han podido ser eliminadas por completo.

Para un *Set-Point* en escalón de 600 radianes por segundo se obtiene la gráfica 4.31. Donde se muestra que a pesar de algunas discontinuidades, se ha logrado una respuesta dentro de los requisitos iniciales de diseño.

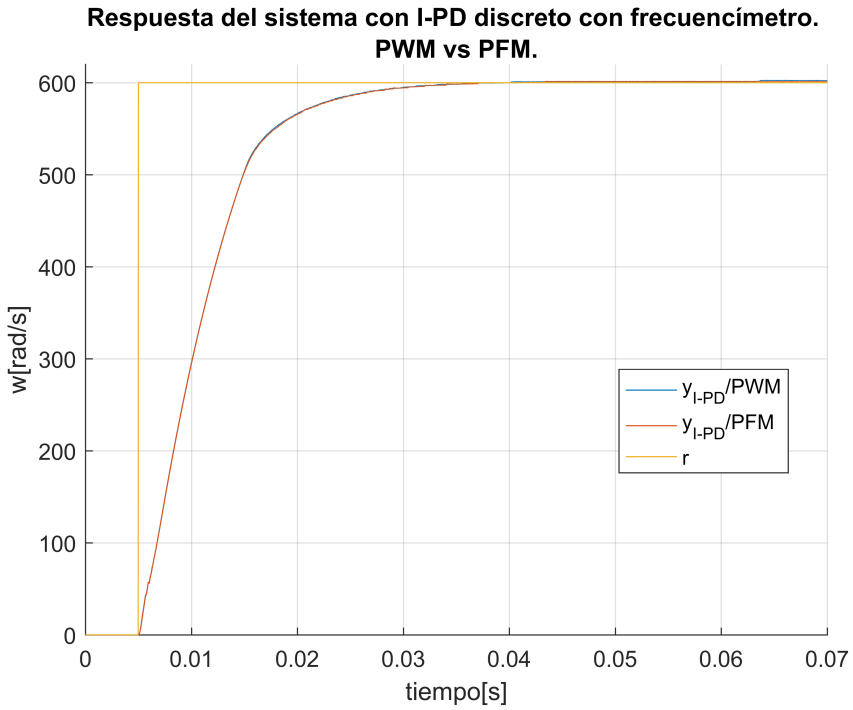


Figura 4.31: Respuesta del sistema con IPD(z) y frecuencímetro con el tiempo de ejecución ajustado. *Set-Point* de 600 [rad/s].

Debido a que el bloque frecuencímetro depende a su vez del bloque encargado determinar la dirección de giro del motor para proporcionar velocidades negativas, es de interés mostrar qué ocurre a velocidades negativas. Para ello, se introduce un *Set-Point* de velocidad angular de tipo sinusoidal al sistema, con una amplitud de 600 radianes por segundo y una frecuencia de 33.3Hz, que como se apuntó en el apartado dedicado al bloque del frecuencímetro, garantiza mejores resultados. La respuesta del sistema en lazo cerrado se muestra en la figura 4.32.

En la tabla 4.2 se resumen los parámetros de la simulación y los valores de las acciones de control del controlador I-PD ajustados.

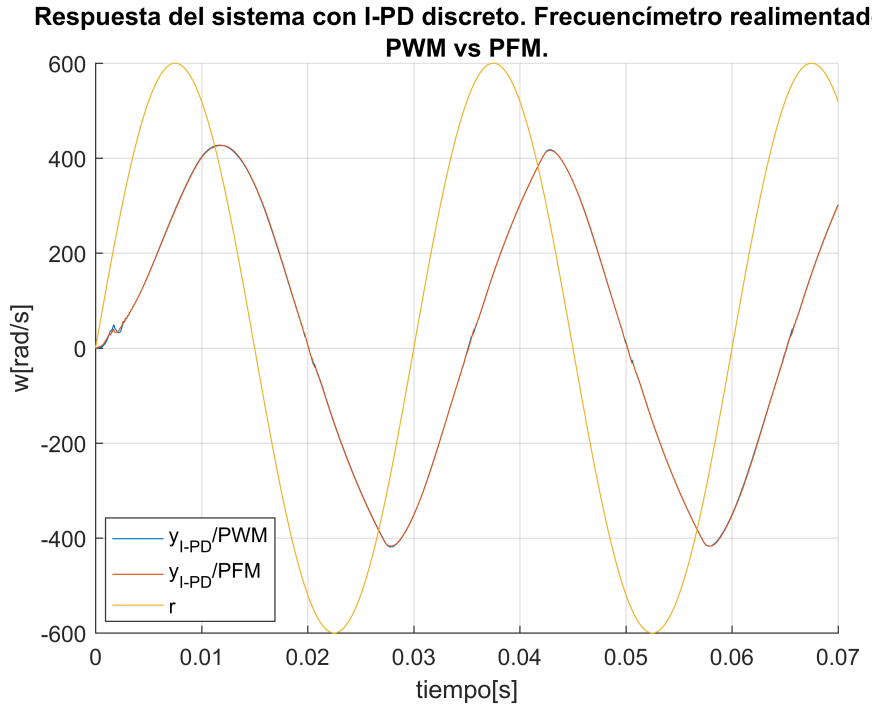


Figura 4.32: Respuesta del sistema. Controlador IPD(z). Conjunto frecuencímetro-encoder con tiempos de ejecución ajustados. *Set-Point* sinusoidal(f=33.3Hz).

Parámetros de la simulación	Acciones de control
$T_sim = 5 \cdot 10^{-9}[s]$	$K_i = 53{,}699$
$T_m = 5 \cdot 10^{-6}[s]$	$K_p = 0{,}3296$
$T_res = 5 \cdot 10^{-7}[s]$	$K_d = 8{,}5838 \cdot 10^{-5}$
$t_on_PFM = 5 \cdot 10^{-7}[s]$	$t_{fd} = 2{,}6043 \cdot 10^{-5}$
$T_enc = 3 \cdot 10^{-8}[s]$	
$T_frec = 1 \cdot 10^{-8}[s]$	

Cuadro 4.2: Parámetros de la simulación y valores del controlador I-PD en tiempo discreto ajustados

4.7. Simulación del controlador PID en tiempo discreto

Para realizar la simulación del controlador PID en tiempo discreto, se realizará por etapas añadiendo bloques que acercarán la simulación a su implementación en hardware. En la figura 4.33 se presenta el conjunto formado por la planta, el controlador PID en tiempo discreto, y el resto de bloques desarrolados del tipo *Matlab function*. Se comenzará la simulación con el controlador PID con su salida limitada al voltaje nominal del motor de ±6 voltios y usando la modulación por ancho de pulsos (PWM) y a continuación empleando modulación por frecuencia de pulsos (PFM). Se presentarán gráficas comparativas con ambas modulaciones. Finalmente, se introducirá el conjunto encoder-frecuencímetro en la realimentación. Los valores de partida de las acciones de control y los parámetros de la simulación se indican en la tabla 4.3.

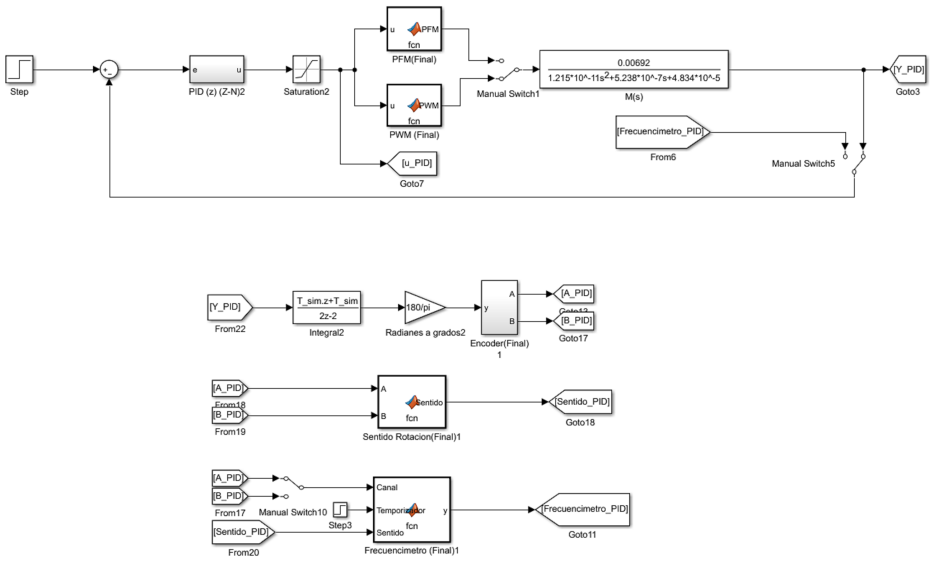


Figura 4.33: Conjunto formado por la planta, el controlador PID en tiempo discreto y el resto de bloques en *Simulink*.

Parámetros de la simulación	Acciones de control
$T_sim = 5 \cdot 10^{-8}[s]$	$K_i = 5{,}6745$
$T_m = 5 \cdot 10^{-6}[s]$	$K_p = 0{,}06$
$T_res = 5 \cdot 10^{-7}[s]$	$K_d = 6{,}449 \cdot 10^{-7}$
$t_on_PFM = 5 \cdot 10^{-7}[s]$	$t_{fd} = 1{,}0748 \cdot 10^{-6}$
$T_enc = 3 \cdot 10^{-6}[s]$	
$T_frec = 1 \cdot 10^{-6}[s]$	

Cuadro 4.3: Parámetros de la simulación y valores del controlador PID de partida

4.7.1. Simulación del controlador PID sin frecuencímetro en la realimentación

Siguiendo un procedimiento análogo al realizado con el controlador I-PD, La respuesta del sistema ante una entrada en escalón de velocidad en radianes por segundo, con una amplitud de cien y con los valores de las acciones de control obtenidos en 3.57, es la representada en la figura 4.34.

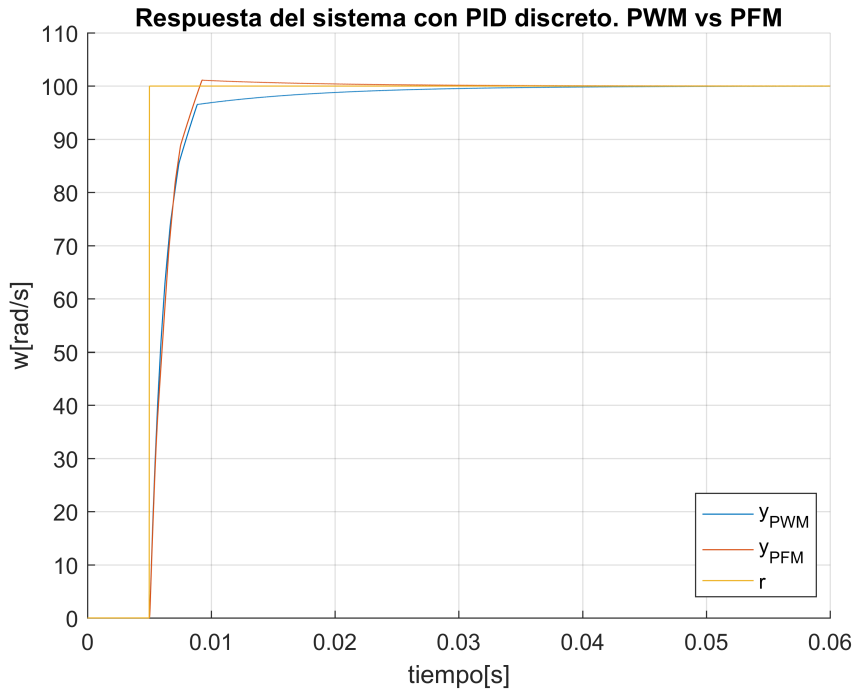


Figura 4.34: Respuesta del sistema con controlador PID. Modulación PWM frente a modulación PFM. *Set-Point* en escalón.

En ella se observa que los bloques PWM y PFM afectan a la dinámica del sistema introduciendo discontinuidades en la respuesta. También se advierte como en el caso de modulación PFM se introduce una sobreoscilación. Por lo cual, será necesario reajustar los valores de ganancia de las acciones de control para eliminar en la medida de lo posible estos efectos.

4.7.1.1. Ajuste fino del controlador PID sin frecuencímetro

El procedimiento usado para hallar los nuevos valores de las acciones derivativa, proporcional e integral, consiste en mantener fijo el valor de la acción derivada y disminuir a la vez los valores de las acciones proporcional e integral. El objetivo es no alterar el tipo de respuesta obtenida, es decir, no obtener una respuesta subamortigada. Tras varias iteraciones, se decide definitivamente usar los siguientes valores del controlador PID:

$$\begin{aligned} K_i &= 0{,}81 \\ K_p &= 0{,}009 \\ K_d &= 6{,}449 \cdot 10^{-6} \\ t_{fd} &= (K_d/K_p)/10 = 7{,}165 \cdot 10^{-5} \end{aligned} \tag{4.82}$$

La respuesta del sistema ante un *Set-Point* en escalón de amplitud de 100 radianes por segundo, es para ambas modulaciones es la mostrada en la figura 4.35.

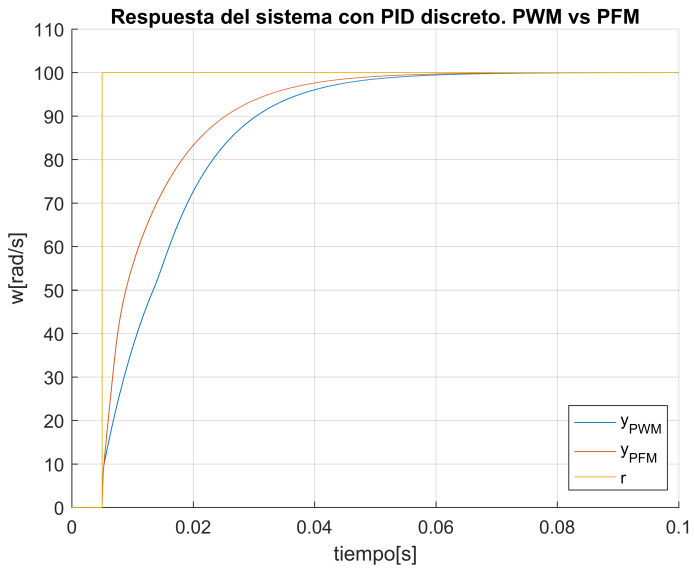
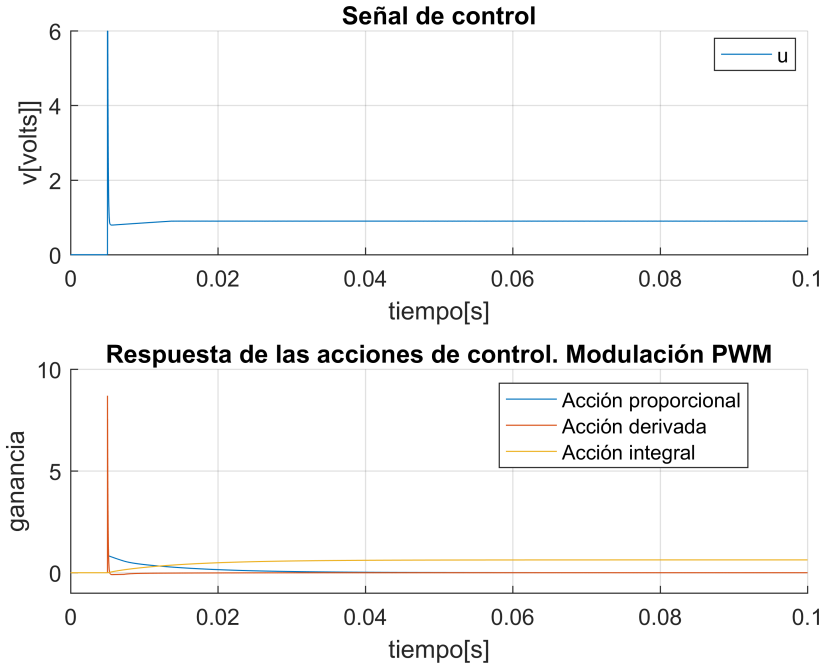


Figura 4.35: Respuesta del sistema con los nuevos valores del controlador PID. Modulación PWM frente a modulación PFM. *Set-Point* en escalón.

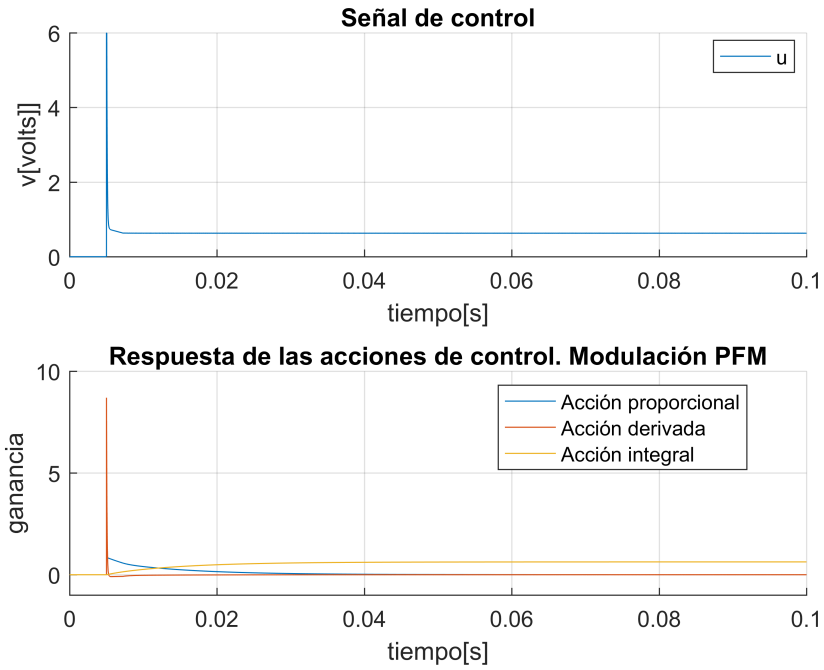
Se obtiene un tiempo de subida para el caso con modulación por ancho de pulsos de: 25,423 milisegundos. En el caso de la modulación por frecuencia de pulsos se obtiene: 20,345 milisegundos. Ambos más lentos que con el controlador I-PD cuyo tiempo de subida era de 14,521 milisegundos.

El comportamiento de las acciones de control en función de la señal del controlador es para el caso de la modulación PWM, la representada en la figura 4.36a. Como se muestra en la imagen, en el caso del controlador PID un *Set-Point* en escalón de amplitud igual a 100 radianes por segundo, es suficiente para hacer llegar al controlador a su valor máximo, al contrario de lo ocurrido en el controlador I-PD.

En el caso de modulación PFM, el comportamiento de las acciones de control frente a la señal del controlador se muestra en la imagen 4.36b. En la que se observa que, tras estabilizarse, el valor de la señal de control es inferior en el caso de modulación por frecuencia de pulsos.



(a) Modulación PWM.



(b) Modulación PFM.

Figura 4.36: Respuesta de las acciones de control del controlador PID en tiempo discreto, frente a la señal de control. *Set-Point* en escalón de amplitud igual a 100.

4.7.2. Simulación del controlador PID con el frecuencímetro en la realimentación

A continuación se introduce el frecuencímetro en la realimentación del bucle en lazo cerrado. Se someterá al sistema a la misma entrada en escalón anterior de 100 radianes por segundo, obteniéndose para ambos métodos de modulación la figura 4.37. En la cual se muestra que, como era de esperar según lo sucedido con el controlador I-PD, es necesario aumentar el tiempo de ejecución del conjunto encoder-frecuencímetro para hacer frente a las discontinuidades de la respuesta. A pesar de ello, las discontinuidades que presenta la respuesta son notablemente inferiores al caso I-PD ante el mismo *Set-Point*, 4.27, por lo que se puede preveer una mejor respuesta del controlador PID frente a las incertidumbres introducidas por el conjunto encoder-frecuencímetro. También se observa, una mayor discrepancia entre las respuestas empleando modulación PWM y PFM que en el caso del controlador I-PD reajustado.

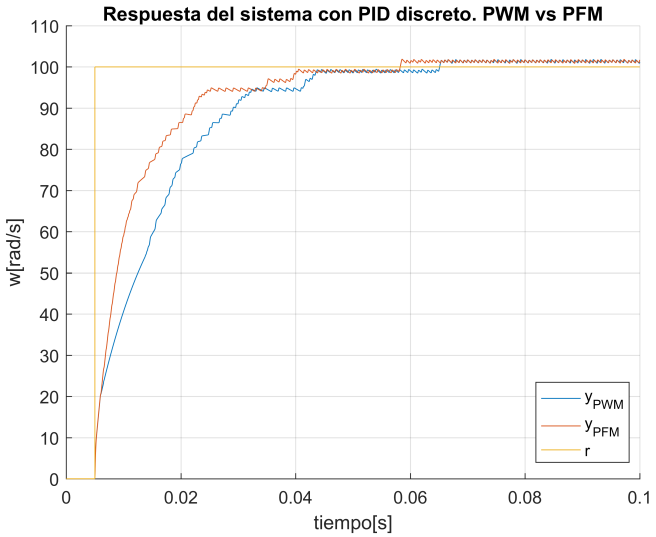


Figura 4.37: Respuesta del sistema. Controlador PID en tiempo discreto. Frecuencímetro en la realimentación. PWM frente a PFM. *Set-Point* en escalón de amplitud igual a 100.

Por tanto, seleccionando los mismos valores que para el caso I-PD, (4.79), se obtiene la figura 4.38, en la que las discontinuidades han sido en gran parte eliminadas.

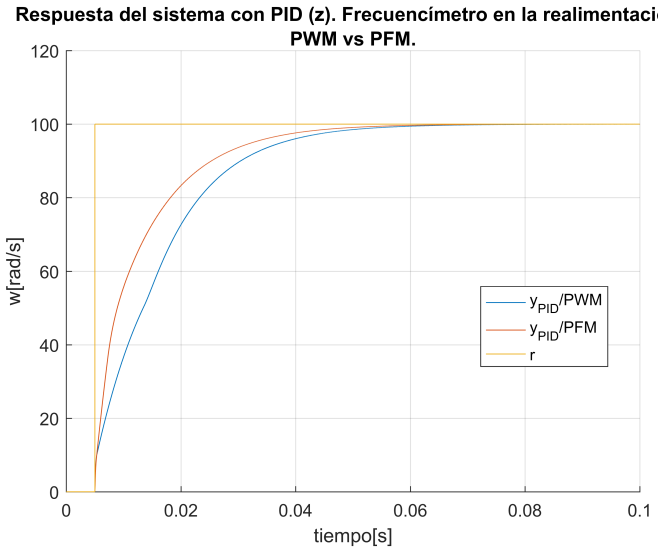


Figura 4.38: Respuesta del sistema. Controlador PID(z) con frecuencímetro ajustado. PWM vs PFM.

En el caso del controlador I-PD fue necesario emplear un *Set-Point* de

amplitud igual a 600 radianes por segundo para hacer llegar al controlador a su valor máximo de tensión continua. Es de interés por tanto, mostrar la respuesta del controlador PID ante el mismo *Set-Point* con el frecuencímetro en la realimentación. El resultado se muestra en la gráfica 4.39.

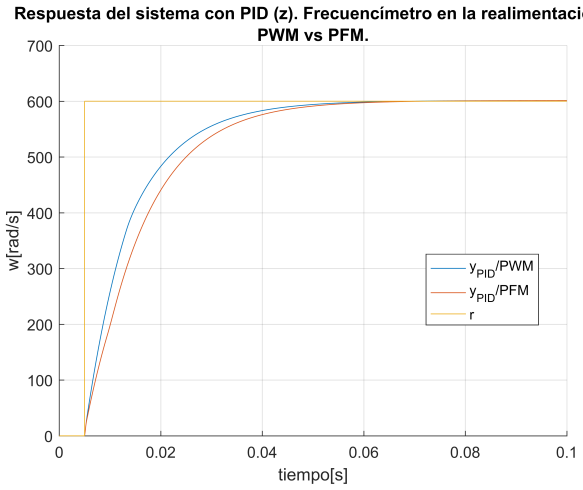


Figura 4.39: Respuesta del sistema. Controlador PID(z). Frecuencímetro ajustado. PWM vs PFM.

Comparando con la respuesta obtenida con una entrada en escalón de 100 radianes por segundo, se observa que la modulación PFM es más rápida cuando el *Set-Point* de velocidad es inferior.

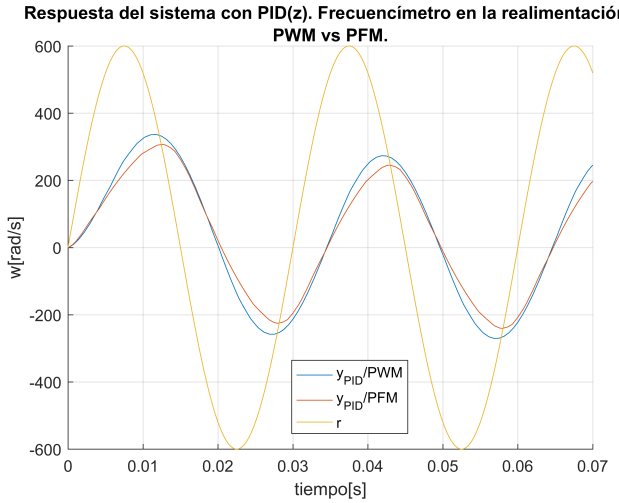


Figura 4.40: Respuesta del sistema. Controlador PID(z). Frecuencímetro ajustado. PWM vs PFM. *Set-Point* sinusoidal de amplitud 600. f=33Hz.

Finalmente, para mostrar que ocurre ante *Set-Points* negativos, se emplea una entrada de velocidad angular sinusoidal de amplitud igual a 600 radianes por segundo y una frecuencia de 33 herzios; se representa en la figura 4.40.

En la tabla ref se presenta un resumen de los parámetros de la simulación junto a las acciones de control del controlador PID ajustados, en tiempo discreto:

Parámetros de la simulación	Acciones de control
$T_sim = 5 \cdot 10^{-9}[s]$	$K_i = 0{,}81$
$T_m = 5 \cdot 10^{-6}[s]$	$K_p = 0{,}009$
$T_res = 5 \cdot 10^{-7}[s]$	$K_d = 6{,}449 \cdot 10^{-6}$
$t_on_PFM = 5 \cdot 10^{-7}[s]$	$t_{fd} = 7{,}1656 \cdot 10^{-5}$
$T_enc = 3 \cdot 10^{-8}[s]$	
$T_frec = 1 \cdot 10^{-8}[s]$	

Cuadro 4.4: Parámetros de la simulación y valores del controlador PID en tiempo discreto ajustados.

4.8. Controladores en recursiva

Implementar los controladores diseñados en hardware, como puede ser el caso de un microcontrolador, requiere su transformación a ecuaciones en diferencia o recursiva. Éstas consisten en representar la señal muestreada por una serie $y[k]$ donde k es el número de muestras tomadas. De tal forma que la primera muestra será $y[0]$, la siguiente $y[1]$, etcétera. Un método común para hallar estas ecuaciones consiste en aplicar la inversa de la transformada z. De tal forma que, partiendo de la función de transferencia de las acciones de control discretizadas, se obtendrán sus ecuaciones en recursiva.

A la acción proporcional no será necesario aplicarle la inversa de la transformada z pero, a la hora de ser implementada en la simulación, se acompañará de un *Zero-Order-Hold* (Z0H) funcionando a un periodo igual al periodo de muestreo. Esto se debe a que, por defecto, *Simulink* entregará cada dato obtenido de la acción proporcional cada vez que realice un cálculo, es decir, cada paso de integración. El *Zero-Order-Hold* (Z0H) funcionando a un periodo igual al periodo de muestreo, se encargará de enviar los datos de la acción proporcional al mismo tiempo que el resto de las acciones de control.

Partiendo de la función de transferencia de la acción integral en tiempo

discreto, (4.9), se tiene para ambos controladores:

$$G_{K_i}(z) = K_i \frac{T_m(z+1)}{2(z-1)} \tag{4.83}$$

$$\frac{u(z)}{e(z)} = K_i \frac{T_m(z+1)}{2(z-1)} \tag{4.84}$$

$$u(z)\big(2(z-1)\big) = e(z)\big(K_i T_m(z+1)\big) \tag{4.85}$$

Donde $u_i(z)$ es la señal proveniente de la acción integral del controlador en tiempo discreto, $e(z)$ es la señal de error y T_m es el periodo de muestreo. A continuación, se divide la ecuación por z, obteniendo:

$$u_i(z)\big(2(1-z^{-1})\big) = e(z)\big(K_i T_m(1+z^{-1})\big) \tag{4.86}$$

$$u_i(z)(2-2z^{-1}) = e(z)\big(K_i T_m + K_i T_m z^{-1}\big) \tag{4.87}$$

$$2u_i(z) - 2u_i(z)z^{-1} = K_i T_m e(z) + K_i T_m e(z) z^{-1} \tag{4.88}$$

Aplicando la inversa de la transformada z con condiciones iniciales nulas resulta:

$$2u_i(k) - 2u_i(k-1) = K_i T_m e(k) + K_i T_m e(k-1) \tag{4.89}$$

$$u_i(k) - u_i(k-1) = K_i T_m \frac{1}{2}\big(e(k) + K_i T_m e(k-1)\big) \tag{4.90}$$

$$u_i(k) = K_i T_m \frac{1}{2}\big(e(k) + K_i T_m e(k-1)\big) + u_i(k-1) \tag{4.91}$$

Donde $u_i(k)$ es la señal de control proveniente de la acción integral en el instante actual k, $e(k)$ es la señal de error en el instante actual k y $e(k-1)$ es la señal de error en el instante k inmediatamente anterior al actual.

Partiendo de la función de transferencia de la acción derivada en tiempo discreto, (4.11), se tiene para el controlador I-PD:

$$G_{K_d}(z) = K_d \frac{2(z-1)}{T_m(z+1)} \tag{4.92}$$

$$\frac{u_d(z)}{y(z)} = K_d \frac{2(z-1)}{T_m(z+1)} \tag{4.93}$$

$$u_d(z)\big(T_m(z+1)\big) = y(z)\big(2K_d(z-1)\big) \tag{4.94}$$

Donde $u_d(z)$ es la señal de control proveniente de la acción derivada e $y(z)$ es la respuesta del sistema proveniente de la planta, en este caso la función de transferencia del motor, $M(s)$ (2.32), en lazo cerrado. Para el caso del controlador PID se tiene:

$$G_{K_d}(z) = K_d \frac{2(z-1)}{T_m(z+1)} \tag{4.95}$$

$$\frac{u_d(z)}{e(z)} = K_d \frac{2(z-1)}{T_m(z+1)} \tag{4.96}$$

$$u_d(z)\big(T_m(z+1)\big) = e(z)\big(2K_d(z-1)\big) \tag{4.97}$$

Donde $e(z)$ es la señal de error. Continuando con el desarrollo para el caso del controlador I-PD, se divide la última ecuación por z. Obteniendo:

$$u_d(z)\big(T_m(1+z^{-1})\big) = y(z)\big(2K_d(1-z^{-1})\big) \tag{4.98}$$

$$u_d(z)T_m + u_d(z)T_m z^{-1} = 2K_d y(z) - 2K_d y(z)z^{-1} \tag{4.99}$$

Aplicando la inversa de la transformada z con condiciones iniciales nulas resulta:

$$u_d(k)T_m + u_d(k-1)T_m = 2K_d y(k) - 2K_d y(k-1) \tag{4.100}$$

$$u_d(k) + u_d(k-1) = 2K_d\frac{1}{T_m}y(k) - 2K_d\frac{1}{T_m}y(k-1) \tag{4.101}$$

$$u_d(k) = 2K_d\frac{1}{T_m}\big(y(k) - y(k-1)\big) - u(k-1) \tag{4.102}$$

Donde $u_d(k)$ es la señal de control proveniente de la acción derivada, $y(k)$ es la respuesta del sistema proveniente de la planta en el instante actual k e $y(k-1)$ es la respuesta del sistema en el instante inmediatamente anterior al actual.

El procedimiento es análogo para el caso del controlador PID, en el cual se obtiene:

$$u_d(k) = 2K_d\frac{1}{T_m}\big(e(k) - e(k-1)\big) - u(k-1) \tag{4.103}$$

La única variación con respecto al resultado obtenido con respecto al controlador I-PD es la señal de entrada, $e(k)$.

Vontinuando con el desarrollo para el controlador I-PD, se tiene para el filtro a la acción derivada, (4.15):

$$F(z) = \frac{T_m z + T_m}{z(2\tau_{fd} + T_m) - 2\tau_{fd} + T_m} \tag{4.104}$$

$$\frac{y_f(z)}{y(z)} = \frac{T_m z + T_m}{z(2\tau_{fd} + T_m) - 2\tau_{fd} + T_m} \tag{4.105}$$

$$y_f(z)\big(z(2\tau_{fd} + T_m) - 2\tau_{fd} + T_m\big) = y(z)(T_m z + T_m) \tag{4.106}$$

Donde $y_f(k)$ es la salida del sistema proveniente de la planta, filtrada y τ_{fd} es la constante de tiempo del filtro a la acción derivada. Dividiendo la ecuación anterior por z resulta:

$$y_f(z)(2\tau_{fd} + T_m - 2\tau_{fd}z^{-1} + T_m z^{-1}) = y(z)(T_m + T_m z^{-1}) \tag{4.107}$$

$$2y_f(z)\tau_{fd} + y_f(z)T_m - 2y_f(z)\tau_{fd}z^{-1} + y_f(z)T_m z^{-1} = y(z)T_m + y(z)T_m z^{-1} \tag{4.108}$$

Aplicando la inversa de la transformada z se obtiene:

$$2y_f(k)\tau_{fd} + y_f(k)T_m - 2y_f(k-1)\tau_{fd} + y_f(k-1)T_m = y(k)T_m + y(k-1)T_m \tag{4.109}$$

$$y_f(k)(2\tau_{fd} + T_m) + y_f(k-1)(T_m - 2\tau_{fd}) = T_m\big(y(k) + y(k-1)\big) \tag{4.110}$$

$$y_f(k) = \frac{T_m\big(y(k) + y(k-1)\big) - y_f(k-1)(T_m - 2\tau_{fd})}{2\tau_{fd} + T_m} \tag{4.111}$$

$$y_f(k) = \Big(\frac{T_m}{2\tau_{fd} + T_m}\Big)\big(y(k) + y(k-1)\big) - \Big(\frac{1}{2\tau_{fd} + T_m}\Big)\big(y_f(k-1)(T_m - 2\tau_{fd})\big) \tag{4.112}$$

$$y_f(k) = \Big(\frac{T_m}{2\tau_{fd} + T_m}\Big)\big(y(k) + y(k-1)\big) - y_f(k-1)\Big(\frac{T_m - 2\tau_{fd}}{2\tau_{fd} + T_m}\Big) \tag{4.113}$$

Donde $y_f(k)$ es la respuesta de la planta filtrada en el instante actual k, $y(k)$ es la respuesta de la planta sin filtrar en el mismo instante e $y(k-1)$, en el instante inmediatamente anterior al actual.

Para el caso del filtro a la acción derivada del controlador PID, basta con cambiar la señal de entrada $y(z)$ por la señal de error $e(z)$, obteniendo:

$$y_f(k) = \Big(\frac{T_m}{2\tau_{fd} + T_m}\Big)\big(e(k) + e(k-1)\big) - y_f(k-1)\Big(\frac{T_m - 2\tau_{fd}}{2\tau_{fd} + T_m}\Big) \tag{4.114}$$

4.8.1. Simulación de ambos controladores en recursiva

Para implementar las acciones de control en recursiva en *Simulink* se emplearán bloques del tipo *Matlab Function* en el que se introducirán las ecuaciones en recursiva de cada una de las acciones de control. Éstos tendrán como entradas los valores actuales de las señales que requieran. En el caso del controlador PID será la señal de error la entrada para cada una de las acciones de control. En el caso del controlador I-PD será la señal de error para la acción integral, y la respuesta de la planta (el motor) para las acciones derivada y proporcional. Además se les pasará como parámetro el valor del periodo de muestreo, T_m y el valor de ganancia para cada acción de control concreta. El periodo de ejecución de estos bloques será el periodo de muestreo T_m, (4.7).

El procedimiento empleado en la programación es el mismo para cada uno de estos bloques: Se emplearán tantas variables del tipo *persistent*, aquellas que mantienen su valor ejecución tras ejecución, como señales que requieran usar su valor inmediatamente anterior al instante actual en la ecuación. Es decir, en el caso de la acción integral, se requieren los valores en el instante anterior de la señal de error, $e(k-1)$, y de la señal de control $u_i(k-1)$, por tanto serán necesarias dos variables del tipo *persistent*. Éstas se denominarán *contenedor_* y a continuación el nombre de la señal a

almacenar. Una vez declaradas las variables se resolverá la ecuación para la acción de control concreta. Finalmente, se almacenarán los valores actuales en las variables *contenedor_*. Es decir, los valores actuales se almacenan como valores del instante inmediatamente anterior, para la siguiente ejecución. Como ejemplo de la programación de estos bloques, se muestra el código de la acción integral en 7.9.

Las respuestas son exactamente las mismas que las mostradas en el apartado de los controladores en tiempo discreto. Esto se debe a que las ecuaciones en diferencias o recursiva son una forma alternativa a la transformada z de representar el proceso de discretización. De esta forma se verifica que el paso de los controladores en tiempo discreto a ecuaciones en recursiva se ha realizado de forma satisfactoria.

4.9. Consideraciones previas a la implementación en *hardware*

Una vez obtenidas e implementadas en la simulación las acciones de control en recursiva, ya están listos los controladores para ser llevados a la práctica. Sin embargo, hasta ahora el motor de la marca *Faulhaber* de la serie *2224 006SR* cuya velocidad se quiere controlar, era tratado como una función de transferencia. Una vez adquirido el motor, es necesario tener en cuenta ciertas consideraciones y limitaciones mecánicas inherentes a él. La primera de ellas es que el motor tiene incorporada una reductora de velocidad en su eje, que no era conocida hasta el momento. Dicha reductora tiene un coeficiente de reducción de 112:1, es decir, la velocidad del eje del motor, en revoluciones por minuto, será reducida mecánicamente un factor de 112. Lo cual implica que el eje con la reductora acoplada no se moverá si se introducen *Set-Points* de velocidad inferiores a 112 revoluciones por minuto, es decir, $11,728$ radianes por segundo. Para solventar este problema, se tendrá en cuenta la reductora a la hora de introducir el *Set-Point* de velocidad en radianes por segundo. Es decir, la velocidad mínima de 112 rpm (sin la reductora) se corresponde a 1 revolución por minuto, que equivalen a 0,105 radianes por segundo y la velocidad nominal del motor de 6390 rpm, 2.1, se reduce a $57,0536$ rpm, que equivalen a $5,975$ radianes por segundo. Por tanto el rango de *Set-Points* válido a introducir será de 0,105 a unos 6 radianes por segundo.

Desde el punto de vista de la simulación y teórico, esta reductora se traduce en introducir una ganancia de $1/112$ a continuación de la planta del sistema, es decir, la función de transferencia del motor que no había sido tenida en cuenta. Ésto implica alterar la ganancia estacionaria del sistema, afectando a su respuesta en lazo cerrado. Es necesario por tanto introducir dicha ganancia proveniente de la reductora en la simulación para comprobar sus efectos.

La segunda de las limitaciones inherentes al motor está relacionada con las frecuencias a las que es capaz de trabajar. Una vez recibido el motor, se realizaron pruebas básicas en lazo abierto para comprobar el funcionamiento de una de las primeras funciones que se programaron, que es la correspondiente a la modulación por ancho de pulsos, cuyo funcionamiento se detallará en el capítulo dedicado a la programación. Distintos voltajes fueron introducidos directamente al motor, en lazo abierto, a través de la modulación PWM funcionando a un periodo igual al periodo de muestreo del controlador, (4.7). Sin embargo, independientemente del voltaje introducido, el motor no respondía. Tras varias pruebas, se llegó a la conclusión de que el motor filtraba la frecuencia correspondiente al periodo de muestreo empleado. Esto se debe a que debido a la inercia del motor, su respuesta en frecuencia se puede aproximar a un filtro paso-bajas de primer orden. Por lo cual es preciso comprobar a qué frecuencia se empieza a atenuar la respuesta del motor. Dicha frecuencia se hallará en el diagrama de Bode del motor, que se representará a partir de su función de transferencia, (2.32). El diagrama se muestra en la figura 4.41.

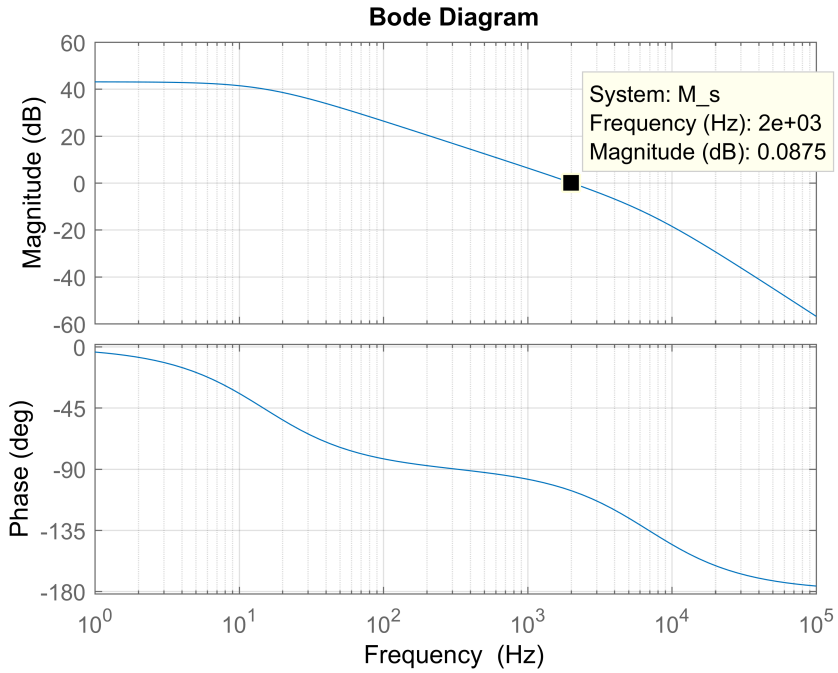


Figura 4.41: Diagrama de Bode del motor.

En ella se puede observar que la respuesta del motor se empieza a atenuar alrededor de los 2 KHz, que se corresponden con un periodo de $5 \cdot 10^{-4}$ segundos. Comparando esta frecuencia con la frecuencia de muestreo selec-

cionada, obtenida a partir de la inversa del periodo de muestreo (4.7), de $5 \cdot 10^{-6}$ segundos, se obtiene una frecuencia de 200 KHz. En la cual, según el diagrama de Bode, la respuesta del motor estará muy atenuada y por tanto no girará. Es imprescindible entonces cambiar la frecuencia de muestreo de los controladores. Se selecciona precisamente la frecuencia a la que la respuesta no es atenuada: 2 KHz, es decir, un periodo de muestreo de $5 \cdot 10^{-4}$ segundos.

$$T_m = 5 \cdot 10^{-4} \text{ segundos} \tag{4.115}$$

Para comprobarlo de forma experimental, se realiza una prueba en lazo abierto introduciendo en la modulación PWM un voltaje que sea la mitad del voltaje nominal del motor, es decir 3 voltios, con distintos periodos de la señal PWM: uno igual al periodo de muestreo anterior de 5 microsegundos y otro con el periodo de muestreo ajustado de 500 microsegundos. Con esto se establece una velocidad para comprobar el funcionamiento. Si la velocidad nominal, con la reductora, es de 6 radianes por segundo que es aquella que se consigue cuando el motor recibe 6 voltios, la mitad de ésta, deberá ser aproximadamente la velocidad que se consiga en lazo abierto si el motor no la filtra. El resultado se muestra en la figura 4.42.

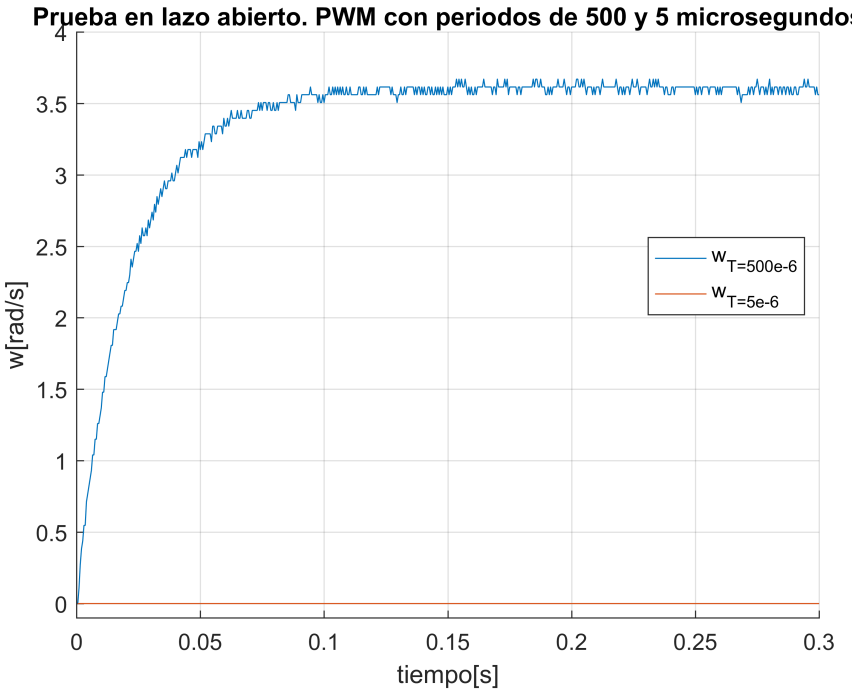


Figura 4.42: Prueba en lazo abierto empleando modulación PWM con periodos de 500 microsegundos y 5 microsegundos.

Por otro lado, con el periodo de muestreo anterior, existía poco márgen para establecer la resolución del bloque PWM, determinada por la relación entre el periodo de muestreo y el periodo de ejecución de éste, estableciéndose en una resolución de 0,1. Ahora, sin embargo, es posible establecer resoluciones de 0,1 con un periodo de ejecución de $5 \cdot 10^{-5}$ segundos, una de 0,01 con un periodo de ejecución de $5 \cdot 10^{-6}$ segundos, una de $0,001$ con un periodo de ejecución de $5 \cdot 10^{-7}$ segundos y finalmente una de 0,0001 con un tiempo de ejecución de $5 \cdot 10^{-8}$. No se puede establecer una mayor resolución ya que los tiempos de ejecución están limitados por el paso de integración de la simulación establecido anteriormente en $5 \cdot 10^{-9}$ segundos, en la sección del ajuste fino con el frecuencímetro (4.80). Esto se debe al modelar un tiempo continuo en la simulación mediante el paso de integración y el uso de un tiempo discreto en los sistemas digitales.

En el caso del bloque PFM en la simulación, su periodo de ejecución determinaba el ancho de pulso de la señal generada. Teniendo en cuenta que según el diagrama de Bode del motor las frecuencias correspondientes al periodo de muestreo anterior del motor, $5 \cdot 10^{-6}$ segundos, eran filtradas, deberá seleccionarse un tiempo mayor a éste como ancho de pulso de la señal PFM. No puede ser demasiado mayor ya que afectaría a la resolución del bloque. Por tanto se selecciona un periodo de ejecución del bloque de $5 \cdot 10^{-5}$ segundos. Esto limita la resolución del bloque PFM a valores de 0,1.

En resumen, se tiene para garantizar la máxima resolución:

$$T_res = 5 \cdot 10^{-8} \text{ segundos} \tag{4.116}$$

y:

$$t_on_PFM = 5 \cdot 10^{-5} \text{ segundos} \tag{4.117}$$

Incluyendo estos parámetros, la reductora incorporada como ganancia colocada a continuación de la función de transferencia del motor y con el frecuencímetro en la realimentación; se obtiene para los controladores PID e I-PD en recursiva la respuesta del sistema representada en la figura 4.43. En ella se observa que, ambos controladores, se ven afectados por la introducción de la reductora, especialmente el controlador PID, para el cual será necesario obtener de nuevo los valores de control empleando métodos de la curva de reacción, mientras que para el controlador I-PD se realizará un ajuste fino. También se observa en el caso del controlador I-PD, como las incertidumbres producidas por la falta de precisión del bloque del PFM, más aquellas introducidas por el conjunto encoder-frecuencímetro (especialmente con *Set-Points* bajos como los que se van a introducir con la reductora) crean discontinuidades en la respuesta.

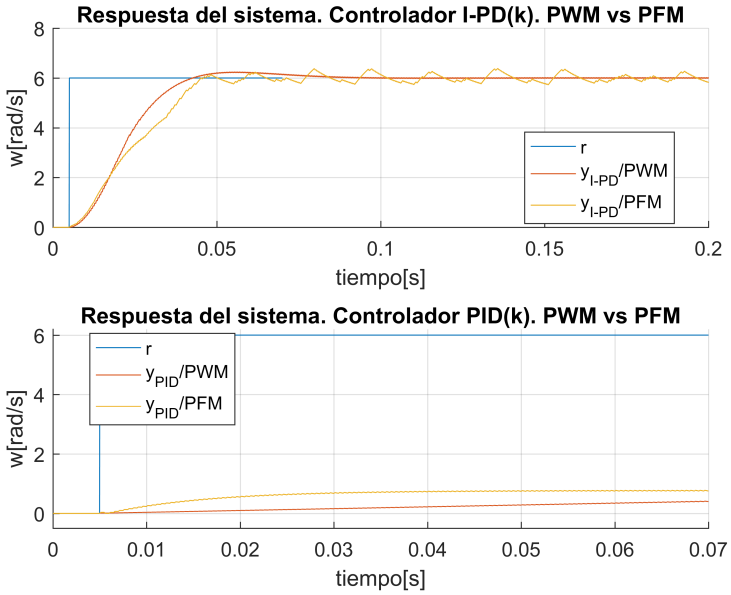


Figura 4.43: Respuesta del sistema teniendo en cuenta la reductora acoplada al motor. PID frente a I-PD en recursiva.

4.9.1. Obtención de los nuevos valores del controlador PID con la reductora

Se comenzará con la simulación en tiempo continuo. De nuevo, se empleará el método basado en la curva de reacción de Ziegler-Nichols. En este caso, se parte de la función de transferencia del sistema del motor, con la ganancia de la reductora colocada a continuación. Siguiendo el método, a continuación se les somete a una entrada del tipo escalón unitario para identificar que clase de respuesta se obtiene a partir de la forma de la curva obtenida. El resultado se muestra en la figura 4.44, cuya respuesta se puede aproximar a un sistema de primer orden con retardo.

Aplicando las ecuaciones de dichos sistemas, (3.42), se obtiene:

$$\begin{aligned} K &= 1{,}278 \\ \tau &= \frac{3}{2}(0{,}011 - 0{,}004) = 0{,}0105 \\ \tau_d &= 0{,}011 - 0{,}0105 = 0{,}0005 \end{aligned} \tag{4.118}$$

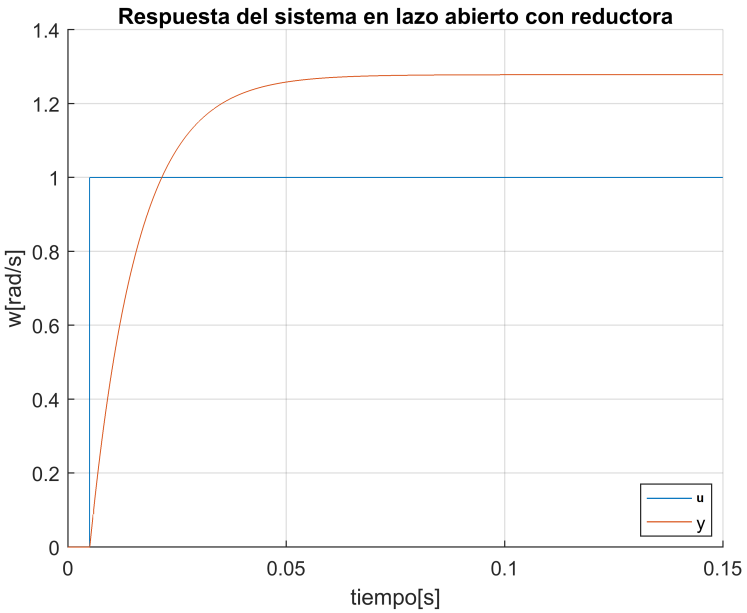


Figura 4.44: Respuesta del sistema en lazo abierto ante una entrada en escalón unitario con reductora.

Aplicando las 3 formas de las ecuaciones de Z-N se obtiene:

- Para el caso de las ecuaciones clásicas, (3.37):

$$\begin{aligned} K_p &= 19{,}718 \\ K_i &= 19718 \\ K_d &= 4{,}9295 \cdot 10^{-3} \\ t_{fd} &= (K_d/K_p)/10 = 2{,}5 \cdot 10^{-5} \end{aligned} \tag{4.119}$$

- Para el caso de las ecuaciones modificadas, (3.38):

$$\begin{aligned} K_p &= 9{,}859 \\ K_i &= 9859 \\ K_d &= 6{,}573 \cdot 10^{-3} \\ t_{fd} &= (K_d/K_p)/10 = 6{,}666 \cdot 10^{-5} \end{aligned} \tag{4.120}$$

- Finalmente para la segunda modificación de las ecuaciones (3.39):

$$\begin{aligned} K_p &= 6{,}573 \\ K_i &= 6573 \\ K_d &= 4{,}382 \cdot 10^{-3} \\ t_{fd} &= (K_d/K_p)/10 = 6{,}666 \cdot 10^{-5} \end{aligned} \tag{4.121}$$

En primer lugar se prueba la respuesta en lazo cerrado de cada una de las opciones con el controlador en tiempo continuo, sin limitación de los valores de tensión continua que puede alcanzar el controlador. Finalmente se decide usar los valores obtenidos aplicando la segunda modificación de las ecuaciones, que dan como resultado una respuesta subamortiguada ante un *Set-Point* en escalón de 6 radianes por segundo (el valor nominal al que podría girar con la reductora).

Para transformar la respuesta en una sin sobreoscilación, en primer lugar se reduce en un factor de cien el valor de la acción integral, que es la que parece tener mayor peso en el control, pasando a ser 65,73. Con esto se consigue una respuesta sin sobreoscilación, muy rápida pero también muy amortiguada. Para compensar, se reducen los valores de la acción proporcional y derivada, en un factor de diez y mil, respectivamente.

El resultado es una respuesta sobreamortiguada con un tiempo de subida de 27,022 milisegundos. A continuación se prueban los nuevos valores del controlador PID con el controlador en recursiva, con la limitación de tensión de éste igual al voltaje de funcionamiento del motor, con el frecuencímetro en la realimentación y con ambos métodos de modulación. La respuesta se muestra en la figura 4.45.

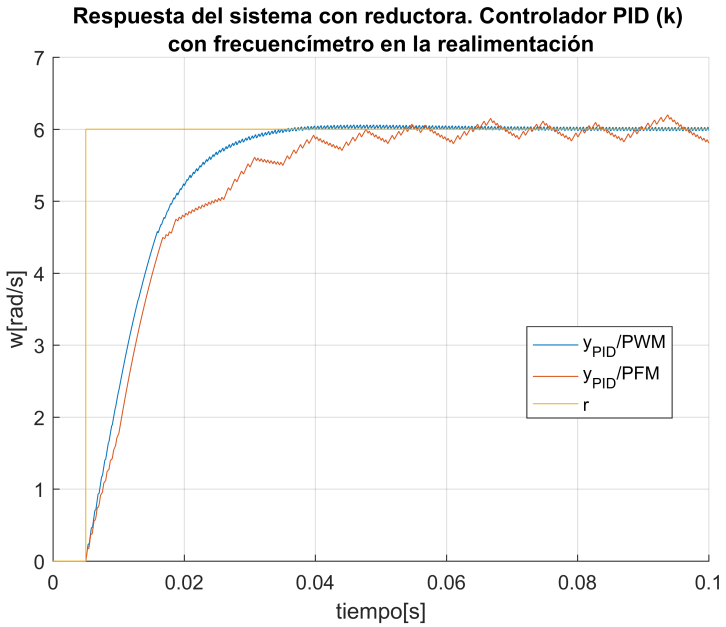


Figura 4.45: Respuesta del sistema con controlador PID en recursiva y reductora ante un *Set-Point* en escalón. PWM frente a PFM.

En la que se observa como las incertidumbres producidas por la falta de precisión del bloque PFM y el conjunto encoder-frecuencímetro son mucho más notables que en el caso de la modulación PWM. Cuanto menor sea el

Set-Point de velocidad, mayores serán las discontinuidades. A pesar de ello, se consideran los valores de las acciones de control adecuados. Por tanto, se obtiene:

$$\begin{aligned} K_p &= 0{,}6573 \\ K_i &= 65{,}73 \\ K_d &= 4{,}382 \cdot 10^{-6} \\ t_{fd} &= (K_d/K_p)/10 = 6{,}666 \cdot 10^{-7} \end{aligned} \tag{4.122}$$

Un resumen de las acciones de control obtenidas y los parámetros de simulación empleados se muestra en la tabla 4.5.

Parámetros de la simulación	Acciones de control
$T_sim = 5 \cdot 10^{-9}[s]$	$K_i = 65{,}73$
$T_m = 5 \cdot 10^{-4}[s]$	$K_p = 0{,}6573$
$T_res = 5 \cdot 10^{-8}[s]$	$K_d = 4{,}382 \cdot 10^{-6}$
$t_on_PFM = 5 \cdot 10^{-5}[s]$	$t_{fd} = 6{,}666 \cdot 10^{-7}$
$T_enc = 3 \cdot 10^{-8}[s]$	
$T_frec = 1 \cdot 10^{-8}[s]$	

Cuadro 4.5: Parámetros de la simulación y valores de las acciones de control del PID con la reductora. Valores finales.

El ajuste final de estos valores se realizará de forma experimental en la práctica.

4.9.2. Ajuste fino del controlador I-PD con la reductora

Como se pudo comprobar en la primera prueba tras introducir el frecuencímetro en la figura 4.45, la reductora introduce una sobreoscilación en la respuesta con un *Set-Point* igual a la velocidad nominal del motor, es decir, 6 radianes por segundo, más visible en el caso de la modulación PWM. Por lo cual es necesario un ajuste fino de las acciones de control del controlador I-PD para solventar dicha sobreoscilación. Debido a que las discontinuidades introducidas por el bloque dedicado a la modulación por la frecuencia de pulsos complican la identificación de la sobreoscilación, únicamente se observará el cambio en la respuesta con modulación PWM mientras se realiza el ajuste.

Partiendo del controlador en tiempo discreto, y de los valores obtenidos en (4.81), se comienza disminuyendo la acción integral de diez en diez. Cuando ésta alcanza el valor de $K_i = 43{,}699$ la sobreoscilación ha sido notablemente disminuida. Se continua reduciendo su valor de uno en uno hasta llegar a 39,699, eliminando por completo la sobreoscilación. La respuesta se muestra en la figura 4.46.

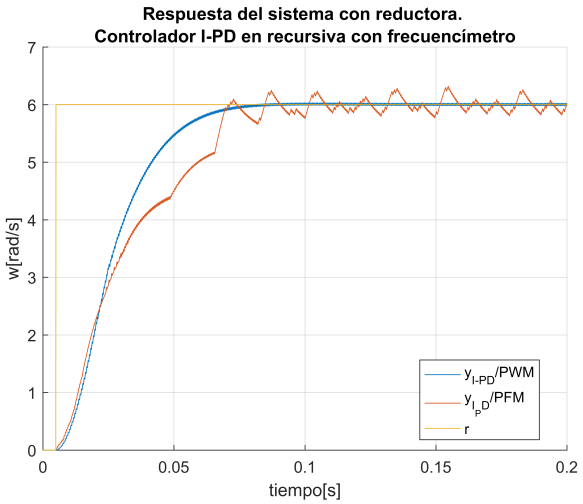


Figura 4.46: Respuesta del sistema. Controlador I-PD(k) con frecuencímetro. PWM vs PFM.

Es de interés intentar solventar las discontinuidades de la respuesta con modulación PFM disminuyendo el valor de la ejecución del bloque, lo que implica un tiempo en alto de la señal PFM menor y una mayor resolución del bloque. Se seleccionará un tiempo de ejecución del bloque igual al usado con la modulación PWM (4.116), es decir, $5 \cdot 10^{-8}$ segundos. No es un tiempo en alto de la señal PFM que vaya a ser implementado en la práctica, se emplea para encontrar la raíz de los errores en el caso de la modulación PFM.

Ante el mismo *Set-Point* se obtiene la gráfica 4.47.

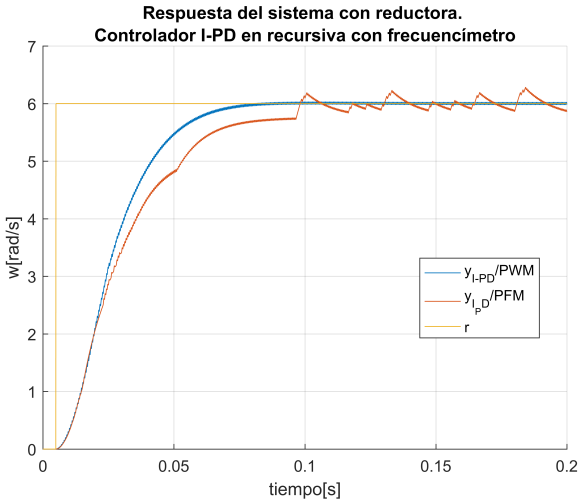


Figura 4.47: Respuesta del sistema. Controlador I-PD(k) con frecuencímetro.

En la que se observa una mejoría excepto cuando la velocidad alcanza valores mayores.

Viendo los resultados obtenidos tanto en las simulaciones sin reductora como en aquellas con ésta incorporada, se deduce que las discontinuidades están en su mayoría relacionadas con el bloque dedicado a la modulación PFM. Sería necesario mejorar la implementación de ésta en la simulación y comprobar si mejora la respuesta. Se deja como una propuesta de mejora.

Como resumen de los parámetros de la simulación empleados y de los valores de las acciones de control del I-PD en recursiva con la reductora, se presenta la tabla 4.6.

Parámetros de la simulación	Acciones de control
$T_sim = 5 \cdot 10^{-9}[s]$	$Ki = 39{,}699;$
$T_m = 5 \cdot 10^{-4}[s]$	$Kp = 0{,}3296$
$T_res = 5 \cdot 10^{-8}[s]$	$K_d = 8{,}5838 \cdot 10^{-5}$
$t_on_PFM = 5 \cdot 10^{-5}[s]$	$t_{fd} = 2{,}6043 \cdot 10^{-5}$
$T_enc = 3 \cdot 10^{-8}[s]$	
$T_frec = 1 \cdot 10^{-8}[s]$	

Cuadro 4.6: Parámetros de la simulación y valores de las acciones de control del I-PD con la reductora. Valores finales.

Capítulo 5

Implementación en hardware. Programación.

5.1. Hardware

La tarjeta microcontroladora basada en el *LPC4078FBD80*, microcontrolador de la serie *LPC408x/7x* de la marca *NXP*[6], será la empleada para la implementación en hardware de los controladores PID e I-PD. Dicho microcontrolador está basado en *ARM Cortex-M4*, funcionando a una frecuencia de 120 Mhz. Los perífericos incluidos en la serie *LPC4078FBD80* son los siguientes:

- 512 kB de memoria flash de programa.
- 96 kB de memoria SRAM de datos.
- Hasta 4032 bytes de memoria EEPROM de datos.
- Controlador de memoria externo, *External Memory controller (EMC)*.
- LCD.
- Ethernet.
- USB Device/Host/OTG.
- Interfaz SPI, *SPI Flash Interface (SPIFI)*.
- Un controlador de propósito general DMA *Direct Memory Access*.
- Cinco UARTs.
- Tres controladores SSP (puertos serie síncronos).
- Tres interfaces I2C.

- Una interfaz de encoder cuadrático *Quadrature Encoder Interface (QEI)*.
- Cuatro *timers* de propósito general.
- Dos PWMs de propósito general con seis salidas cada uno y un *Motor Control PWM*.
- Un RTC (*Real Time Clock* de baja potencia con alimentación por batería separada y *event recorder*.
- Un *windowed watchdog timer*.
- Un periférico de cálculo de verificación por redundancia cíclica (CRC).
- Hasta 165 pines de propósito general.
- Periféricos analógicos que incluyen un convertidor analógico-digital de ocho canales y 12 bits, dos comparadores y un convertidor digital-analógico.

De los cuales, los incluidos en la tarjeta *LPC4078FBD80* se muestran en la imagen 5.1. De todos ellos, serán de especial importancia los siguientes:

Type number	Flash (kB)	SRAM (kB)	EEPROM (B)	EMC bus width (bit)	LCD	Ethernet	USB	CAN	UART	QEI	SD/MMC	Comparator	FPU	Package
LPC4078FBD144	512	96	4032	8	no	yes	H/O/D	2	5	yes	yes	yes	yes	LQFP144
LPC4078FBD100	512	96	4032	-	no	yes	H/O/D	2	5	yes	yes	yes	yes	LQFP100
LPC4078FBD80	512	96	4032	-	no	yes	H/O/D	2	5	yes	no	yes	yes	LQFP80

Figura 5.1: Características de la tarjeta LPC4078FBD80

- Un *Timer* de propósito general, en concreto el *Timer0*, que será usado para controlar el tiempo que deberá girar el motor antes de transmitir los datos almacenados a través de la UART.
- Una de las interfaces PWM, la PWM1, que usará dos de sus canales para enviar las señales PWM y PFM al motor. En concreto los canales PWM1[1] y PWM1[2]. Cuando se quiera un sentido de giro horario se enviará la señal por el canal PWM1[1] mientras el canal PWM1[2] se mantiene en bajo. Para el sentido de giro opuesto, se invertirán las funciones de los canales.
- La interfaz QEI, empleada para leer y muestrear los datos provenientes del encoder. En ella se establecerá el tiempo de muestreo de los controladores.

- Los datos de velocidad de giro del motor serán enviados a través de una de las UARTs, la UART0.

Para programar los parámetros necesarios en el microcontrolador, será necesario emplear un *JTAG*. En este caso se empleará un *JTAG* basado en el *JLINK* de *Segger*.

Además, se usará como elemento actuador un puente en "H" empleando el driver *L298N*. También será necesario un adaptador USB-Uart basado en el chip *MAX 232*, el cual simulará un puerto RS232 virtual que funciona a niveles TTL, que irá conectado a un ordenador. Éste recibirá datos de velocidad provenientes del encoder del motor, los cuales serán tratados y representados gráficamente a través del software *Matlab*.

5.2. Programación

El lenguaje empleado en la programación será el lenguaje "C" usando el estándar C90, que es el seleccionado por defecto por el compilador del entorno IDE de programación *µVision*. Toda la información al respecto del funcionamiento de los periféricos contenida en este capítulo, se encuentra en el *User Manual*[6] proporcionado por el fabricante.

Los pines 2 y 3 del puerto 0 del microcontrolador, serán usados como salida "TX" y entrada "RX" de la UART0, respectivamente. Los pines 18, 20 y 23 del puerto 1 serán usados como salida del canal 1 del primer módulo PWM disponible, entrada del canal "A" del encoder y entrada del canal "B", respectivamente. El pin 1 del puerto 2, será usado como la salida del segundo canal del módulo PWM. La conexión de todos estos elementos se muestra en el esquema representado en la figura 5.2.

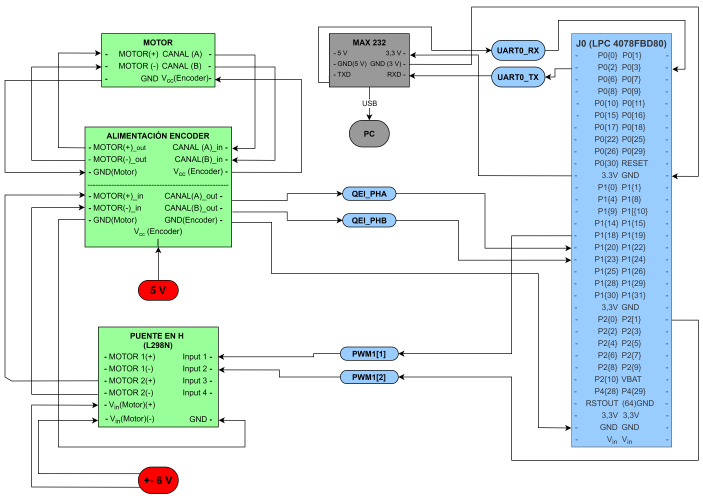


Figura 5.2: Esquema de conexiones.

El funcionamiento general del programa es el siguiente: En primer lugar se establecerán el *Set-Point* de velocidad, el método de control y de modulación y el número de datos de velocidad que se van a almacenar, en el microcontrolador. A continuación se programará un tiempo de funcionamiento. Mientras no se haya llegado al tiempo establecido, se efectuará el control seleccionado para el *Set-Point* elegido empleando como actuador el puente en "H" con el chip *L298N*. Una vez finalizado el tiempo, el motor se parará y a continuación los datos serán enviados a través de la UART a un ordenador, mediante el adaptador USB-Uart. Finalmente, estos datos serán representados gráficamente en *Matlab*. El diagrama de flujo de este procedimiento se muestra en la figura 5.3.

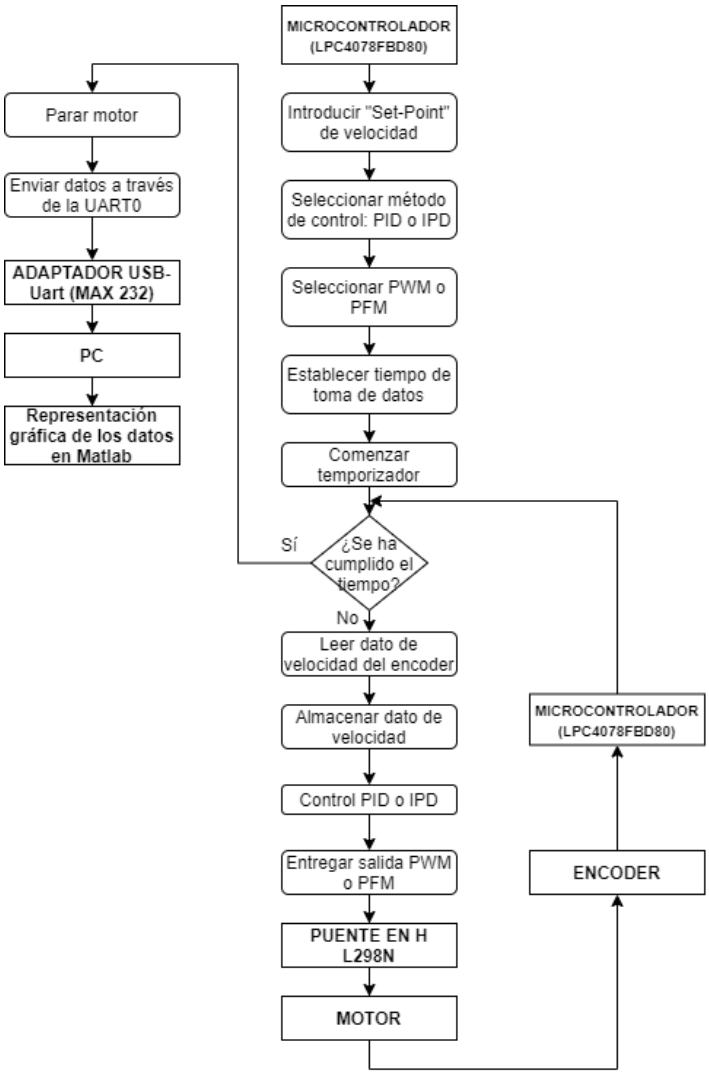


Figura 5.3: Diagrama de flujo general del funcionamiento del programa.

El software empleado para programar el microcontrolador es el entorno de desarrollo integrado (IDE) *µVision* de *Keil*. El lenguaje de programación usado es "C". El proyecto estará compuesto de los siguientes archivos:

- constantes.h: Archivo de cabecera en el que el usuario establecerá distintas constantes necesarias para el funcionamiento del programa, como por ejemplo los valores de ganancia de las acciones de control, el tiempo de muestreo, etc. Es necesario modificar este archivo y compilarlo antes de cada ejecución.

- main.c: Archivo en el que se encuentra el punto de entrada del programa. En él estará definido el comportamiento de los controladores PID e I-PD.

- pwm.c y pwm.h: Archivos en los que se encuentran las funciones necesarias para el funcionamiento del periférico PWM. En"pwm.c" se hallan las funciones que realizan el ajuste de la velocidad del motor en función de si se emplea modulación PWM ó PFM, y para parar el motor. En el archivo de cabezera "pwm.h"se encuentran las definiciones de las funciones que se encuentran en "pwm.c".

- qei.c y qei.h: Archivos en los que se encuentran las funciones necesarias para el funcionamiento del periférico *Quadratic-Encoder-Interface (QEI)*. En "qei.c" se hallan las funciones necesarias para la obtención de datos de velocidad del motor a partir de las señales proporcionadas por los canales del encoder acoplado en el motor. En "qei.h" se encuentran las definiciones de las funciones de "qei.c".

- timers.c y timers.h: Archivos en los que se encuentran las funciones necesarias para el funcionamiento del *timer*. Se empleará un *timer* para establecer el tiempo que el motor funcionará, almacenando datos de velocidad, para luego ser enviados a través de la UART. En el archivo de cabecera "timers.h" se encuentran las definiciones de las funciones.

- uart0.c y uart0.h: Archivos que contienen las funciones necesarias para el funcionamiento de la UART0. En "uart0.c" se encuentran aquellas empleadas para enviar y recibir cadenas de caracteres a través de la UART0.c. Comunicando el microcontrolador con un ordenador con Matlab a través del adaptador USB-Uart basado en el chip MAX232. En "uart0.h" se localizan las definiciones de las funciones de "uart0.c".

- control.c y control.h: En ellos se encuentran las funciones que integran las acciones de control en recursiva, descritas en las ecuaciones (4.91), (4.102) y (4.113). En "control.h" se ubican las definiciones de dichas funciones.

- extra_math_C90.c y extra_math_C90.h: Son archivos en los que se encuentran funciones matemáticas de redondeo no incluidas en "math.h" para compiladores funcionando en el estándar ANSI C90, el cual es el estándar por defecto de funcionamiento del compilador de *µVision*. Dichas funciones son dos: una para realizar redondeos (sustituye a la función *round()*) y otra para determinar el signo del número introducido. En el archivo de cabecera se encuentra la definición de dicha función.

5.2.1. Programación y funcionamiento de los *Timers*

Según el manual[6] proporcionado por el fabricante, el periférico *Timer/Counter* está diseñado para contar ciclos de la señal del reloj de periféricos (PCLK) o de un reloj externo. Puede, opcionalmente, generar interrupciones o realizar otras acciones a valores de tiempo específicos, designados en cuatro registros de *match*. También incluye cuatro entradas dedicadas a atrapar un valor de tiempo de una señal, opcionalmente generando una interrupción. Hay disponibles hasta 4 *timers*. Los registros que emplea para su funcionamiento son los mostrados en la imagen 5.4.

Name	Access	Address offset	Description	Reset value[1]	Section
IR	R/W	0x000	Interrupt Register. The IR can be written to clear interrupts. The IR can be read to identify which of eight possible interrupt sources are pending.	0	Table 540
TCR	R/W	0x004	Timer Control Register. The TCR is used to control the Timer Counter functions. The Timer Counter can be disabled or reset through the TCR.	0	Table 541
TC	R/W	0x008	Timer Counter. The 32 bit TC is incremented every PR+1 cycles of PCLK. The TC is controlled through the TCR.	0	Table 542
PR	R/W	0x00C	Prescale Register. When the Prescale Counter (PC) is equal to this value, the next clock increments the TC and clears the PC.	0	Table 543
PC	R/W	0x010	Prescale Counter. The 32 bit PC is a counter which is incremented to the value stored in PR. When the value in PR is reached, the TC is incremented and the PC is cleared. The PC is observable and controllable through the bus interface.	0	Table 544
MCR	R/W	0x014	Match Control Register. The MCR is used to control if an interrupt is generated and if the TC is reset when a Match occurs.	0	Table 545
MR0	R/W	0x018	Match Register 0. MR0 can be enabled through the MCR to reset the TC, stop both the TC and PC, and/or generate an interrupt every time MR0 matches the TC.	0	Table 546
MR1	R/W	0x01C	Match Register 1. See MR0 description.	0	Table 546
MR2	R/W	0x020	Match Register 2. See MR0 description.	0	Table 546
MR3	R/W	0x024	Match Register 3. See MR0 description.	0	Table 546
CCR	R/W	0x028	Capture Control Register. The CCR controls which edges of the capture inputs are used to load the Capture Registers and whether or not an interrupt is generated when a capture takes place.	0	Table 547
CR0	RO	0x02C	Capture Register 0. CR0 is loaded with the value of TC when there is an event on the CAPn.0 input.	0	Table 548
CR1	RO	0x030	Capture Register 1. See CR0 description.	0	Table 548
EMR	R/W	0x03C	External Match Register. The EMR controls the external match pins.	0	Table 549
CTCR	R/W	0x070	Count Control Register. The CTCR selects between Timer and Counter mode, and in Counter mode selects the signal and edge(s) for counting.	0	Table 550

[1] Reset Value reflects the data stored in used bits only. It does not include reserved bits content.

Figura 5.4: Registros de funcionamiento de los *Timers*[6].

Es de interés para este trabajo su funcionamiento como *Timer*. No se capturará ningún valor de tiempo de una señal externa pues se contarán únicamente los ciclos de la señal del reloj de periféricos. Su funcionamiento es el siguiente: El valor del registro PC (*Prescale Counter*) aumentará cada flanco de subida de la señal del reloj de periféricos (PCLK). Dicho reloj funciona a una fracción de la frecuencia de trabajo del microcontrolador, 120 MHz. Por defecto, esta fracción es dos, es decir 60 MHz, lo cual implica que el valor de PC se incrementará cada $\frac{1}{60 \cdot 10^6} = 16{,}67$ nanosegundos. Este valor por si mismo es poca referencia, por ello se proporciona un registro de preescala, PR, en el cual se establece el número máximo de ciclos contados que podrá alcanzar el registro PC. Una vez alcanzado este valor máximo se aumentará el valor del registro denominado TC (*Timer Counter*. Con la combinación de ambos se consigue dividir la frecuencia del reloj de periféricos. Por ejemplo: Se establece como valor máximo en el registro PR, $60 \cdot 10^3$ ciclos. Sabiendo que en un segundo hay $60 \cdot 10^6$ ciclos y que el registro TC aumenta cada vez que se cuenten $60 \cdot 10^3$ ciclos, se tiene que se habrán contado $\frac{60 \cdot 10^6}{60 \cdot 10^3} = 1000$ ciclos en un segundo, es decir, a efectos prácticos se ha dividido la frecuencia del reloj de periféricos a 1000 Hz. Estableciendo un aumento del valor del registro TC cada $\frac{1}{1000} = 1$ milisegundos. Es necesario tener en cuenta que el registro PC comienza su cuenta en cero, por lo cual, al número de ciclos máximos que se quieren contar (PR) es necesario restarle uno. Todos los eventos anteriores se llevarán a cabo cuando se establezca en alto el bit correspondiente en el registro de control del *Timer* (TCR).

El valor del registro TC aumentará hasta alcanzar el número objetivo introducido en uno de los cuatro registros MATCH proporcionados. Cuando ambos coincidan se pueden establecer hasta tres escenarios distintos activando los bits correspondientes del registro de control de MATCH, MCR:

- Generar una interrupción: Poniendo en alto el bit correspondiente del registro de interrupciones, IR.
- Resetar el valor de TC a cero.
- Parar de aumentar los valores de TC y PC; y desactivar el registro de control, TCR, estableciendo el valor de su bit correspondiente en cero.

En el manual[6] proporcionado por el fabricante se adjuntan dos ejemplos de funcionamiento: En el primero de ellos, mostrado en la imagen 5.5, se ha establecido en el registro de preescala, PR, dos ciclos y seis como objetivo en uno de los registros de MATCH. Además se ha programado el registro de control, MCR, para que haga cero el valor de TC cuando ocurra una coincidencia y se genere una interrupción. Se observa como el valor del registro TC aumenta cada tres ciclos de la señal de reloj de periféricos, ya que comienza a contar en cero. Cuando llega a seis, se reinicia su valor en el siguiente ciclo de TC. Un ciclo de la señal de reloj después de ocurrir la coincidencia, es cuando se genera la interrupción.

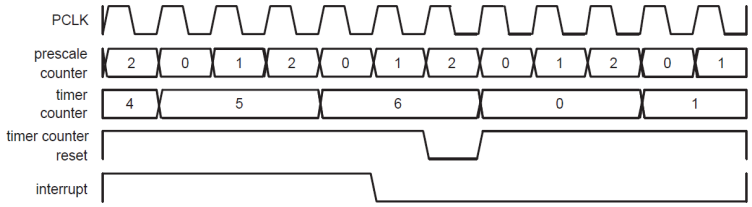


Figura 5.5: Ejemplo de funcionamiento del *Timer*. PR=2,MRx=6, interrupción y reinicio activados.[6]

En el segundo de ellos, mostrado en la figura 5.6, se establecen los mismos parámetros que en el ejemplo anterior a excepción de los eventos que se producen cuando ocurre una coincidencia. En este caso, se para el aumento de TC y se desactiva el *Timer* mediante el registro TCR. Se llevan a cabo en el ciclo de la señal de reloj posterior a la coincidencia con el valor programado en uno de los registros de MATCH.

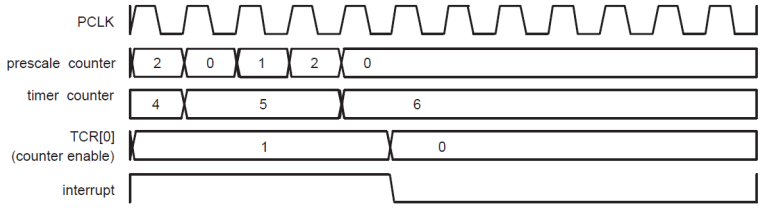


Figura 5.6: Ejemplo de funcionamiento del *Timer*. PR=2,MRx=6, interrupción y parada activados.[6]

Teniendo en cuenta el funcionamiento del periférico, se desarrollan dos funciones que se ejecutarán conjuntamente: La primera será una función de tipo *void*, que estará dedicada a establecer que ocurra una interrupción cuando pase un tiempo determinado a introducir. El diagrama de flujo de esta función se presenta en la imagen 5.7.

Una vez se llame a esta función se comenzará el conteo en milisegundos. Para comprobar si esta cuenta ha terminado será necesario emplear la segunda función anteriormente mencionada. Ésta será una de tipo *unsigned int* y retornará "1" si ha transcurrido el tiempo establecido y "0" en caso contrario. El diagrama de flujo de ésta se representa en la imagen 5.8.

Ambas se usarán conjuntamente para establecer el tiempo que el motor girará antes de enviar los datos de velocidad almacenados a través de la UART.

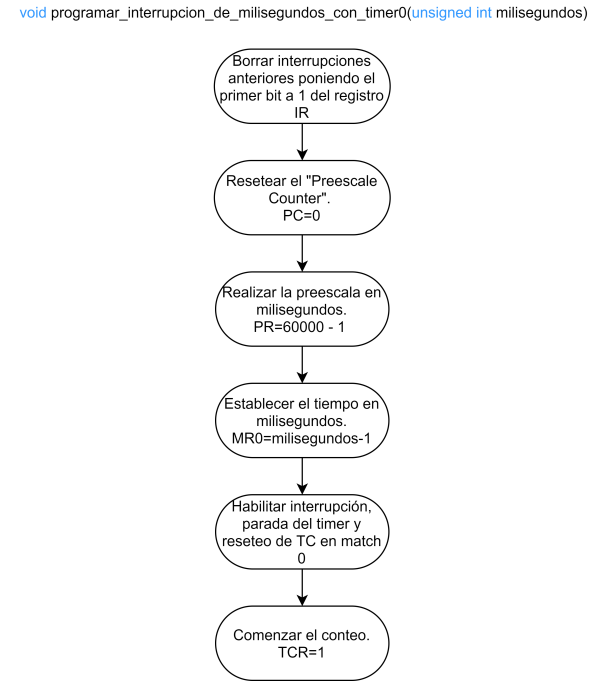


Figura 5.7: Diagrama de flujo de la función: Programar interrupción del timer0 en milisegundos.

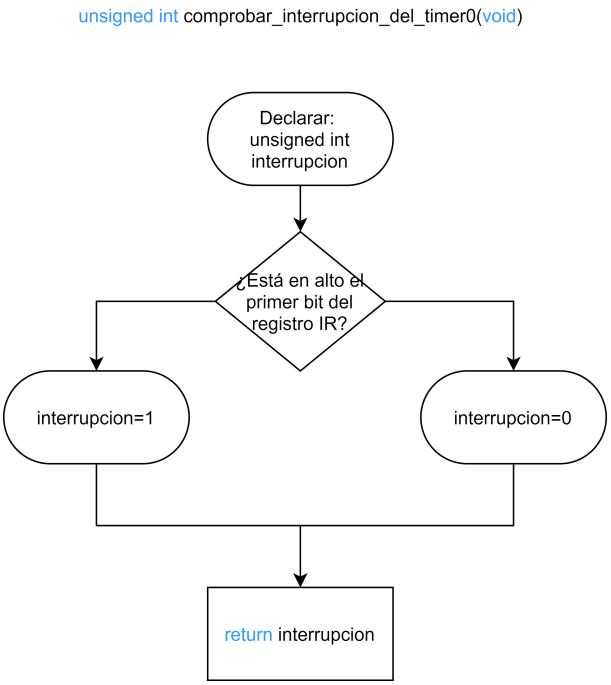


Figura 5.8: Diagrama de flujo de la función: Comprobar interrupción del timer0.

5.2.2. Programación y funcionamiento del periférico PWM

El periférico PWM está basado en el bloque de los *Timers* y por lo tanto hereda todas sus características. Realmente, la función PWM del microcontrolador es una adición extra a las funciones del *Timer* que actúa cuando ocurren coincidencias con los registros MATCH. Emplea un *Timer* específico que funciona por separado de los cuatro ya incluidos y comentados en el apartado anterior. Puede ser controlado por un único flanco o por dos. El periférico debe ser encendido aplicando alimentación mediante el bit 6 del registro PCONP. Para seleccionar los pines de entrada y salida se emplea el registro IOCON.

Dos registros MATCH deben ser usados para controlar una salida PWM por un único flanco: El registro MATCH0 (MR0) controla el periodo de la señal PWM, reseteando el contador del *Timer* (TC) cuando ocurre una coincidencia con el valor de tiempo almacenado en él. El segundo registro MATCH controla la posición del flanco de bajada de la señal PWM, ya que, en el control de un único flanco, la señal comienza el periodo en alto. Es decir, este registro controla el ciclo de trabajo de la señal PWM de salida. Comienza en alto automáticamente al inicio de cada periodo de la señal de salida y se establece a nivel bajo cuando ocurre una coincidencia con el valor almacenado en el segundo registro. Si por algún motivo no ocurriera ninguna coincidencia con el registro MATCH encargado de controlar el flanco de bajada, la señal permanecería en alto todo el periodo.

Cada salida adicional por uno de los seis canales disponibles, controladas por un único flanco, tendrán un segundo registro de control de flanco de bajada específico. Para el canal 1, éste será el registro MATCH1 (MR1), para el segundo canal será el registro MATCH2 (MR2), etc. El periodo de todas ellas será controlado por el registro MATCH0 (MR0).

Para controlar el PWM por dos flancos se emplean tres registros: De nuevo, el MR0 controla el periodo de la señal PWM. Los otros dos registros controlan las dos posiciones de los flancos de la señal PWM, uno para el flanco de subida y otro para el de bajada. Cuando se emplea el control por dos flancos, es posible generar salidas PWM de amplitud negativa. Para ello, es necesario establecer que la primera coincidencia con el valor del contador del *Timer* (TC), sea con el valor almacenado en el registro MATCH encargado de controlar el flanco de bajada. La siguiente deberá ser con el registro encargado de controlar el flanco de subida.

De nuevo, para salidas adicionales de la señal PWM controladas por ambos flancos, se requieren dos registros de control de flancos específicos. El MR0 controla el periodo de todos los canales de salida.

Los registros empleados para cada canal de salida en función de su método de control se muestran en la imagen proveniente del manual del fabricante[6] 5.9.

También, en el manual se muestran tres ejemplos de funcionamiento. En

PWM Channel	Single Edge PWM (PWMSELn = 0)		Double Edge PWM (PWMSELn = 1)	
	Set by	Reset by	Set by	Reset by
1	Match 0	Match 1	Match 0[1]	Match 1[1]
2	Match 0	Match 2	Match 1	Match 2
3	Match 0	Match 3	Match 2[2]	Match 3[2]
4	Match 0	Match 4	Match 3	Match 4
5	Match 0	Match 5	Match 4[2]	Match 5[2]
6	Match 0	Match 6	Match 5	Match 6

[1] Identical to single edge mode in this case since Match 0 is the neighboring match register. Essentially, PWM1 cannot be a double edged output.

[2] It is generally not advantageous to use PWM channels 3 and 5 for double edge PWM outputs because it would reduce the number of double edge PWM outputs that are possible. Using PWM[2], PWM[4], and PWM[6] for double edge PWM outputs provides the most pairings.

Figura 5.9: Registros de control de flancos del periférico PWM.[6]

ellos, se establecen tres canales de salida para el periférico PWM: 2, 4 y 5. El canal 5, estará controlado por un único flanco, y los canales 2 y 4, serán controlados por dos flancos. Según la tabla representada en la imagen 5.9, el registro MATCH5 (MR5), será el encargado de controlar el flanco de bajada del canal 5. Los registros MR1 y MR2 controlarán los flancos de subida y bajada, respectivamente, del canal de salida 2. Los registros MR3 y MR4 controlarán los flancos de subida y bajada, respectivamente, del canal de salida 4. El registro MATCH0 (MR0) controlará el periodo de las señales PWM de los tres canales. El valor del registro *Timer Counter* (TC) se reiniciará cuando ocurra una coincidencia con el registro MR0.

Se asigna al registro MR0 el valor de 100, que podrán ser segundos, milisegundos o microsegundos, en función del valor de preescala del *Timer*. Se supone para este ejemplo que son milisegundos, por tanto, las señales tendran un periodo de 100 milisegundos y una frecuencia de 1000 Hz.

El registro MR5, que es el encargado de controlar el flanco de bajada del canal 5, tendrá un valor de 65, por lo que, la señal de salida comenzará en alto y a los 65 milisegundos se establecerá a nivel bajo.

Los registros MR1 y MR2, encargados de controlar los flancos de subida y bajada del canal 2, tendrán unos valores de 41 y 78 milisegundos, respectivamente. Por tanto, la señal de salida estará en bajo hasta llegar a los 41 milisegundos, instante en el que se establecerá en alto. Se mantendrá hasta llegar a los 78 milisegundos.

Los registros MR3 y MR4, encargados de controlar los flancos de subida y bajada del canal 4, tendrán unos valores de 53 y 27 milisegundos respectivamente. En este caso, la primera coincidencia será del *Timer* será con el registro encargado de controlar el flanco de bajada, por lo que la señal de salida tendrá una amplitud negativa.

Estos tres casos se muestran en la imagen 5.10.

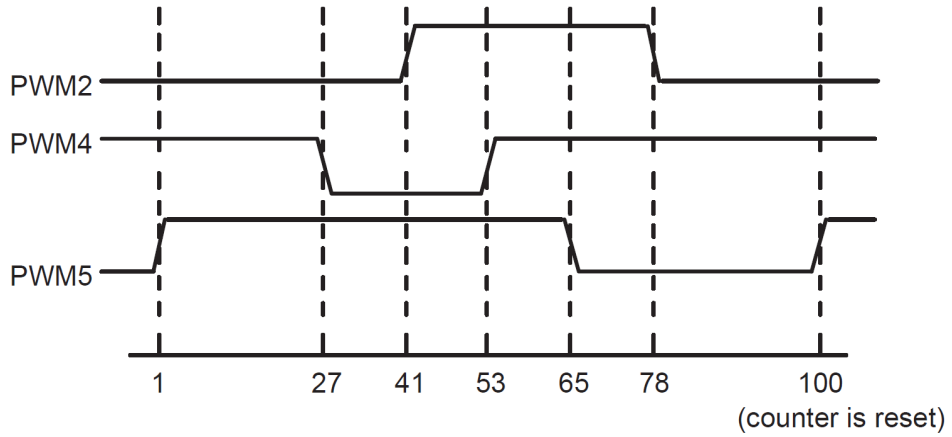


Figura 5.10: Ejemplo de funcionamiento del periférico PWM.[6]

Hasta ahora todos los registros nombrados son heredados del periférico *Timer/Counter*. A parte de ellos, el periférico PWM emplea dos registros exclusivos, que se muestran a continuación:

- *PWM Control Register*(PCR): Este registro se usa para habilitar y seleccionar la función de cada canal de salida.

- *Load Enable Register*(LER): Este es el registro más importante, ya que se emplea para controlar la actualización de los registros MATCH. Cuando se escribe en software un valor en un registro MATCH y está activado el modo PWM en el registro de control del *Timer PWM* (TCR), el valor se almacena en otro registro en vez de ser usado directamente.

  El funcionamiento es el siguiente: Los valores permanecerán en estos registros intermedios de almacenamiento hasta la siguiente coincidencia del *Timer Counter* con el valor establecido en el MR0. En ese momento se comprobará si se han habilitado los bits correspondientes a los registros de MATCH a modificar en LER. En caso afirmativo, los valores almacenados se tendrán en consideración en el siguiente periodo de la señal PWM. Con esto se consigue aplicar cambios en los distintos registros de control de flancos al mismo tiempo.

Teniendo en cuenta el funcionamiento del periférico, se desarrollarán dos funciones para inicializar la modulación por PWM y PFM empleando uno de los módulos PWM disponibles, el PWM1. Se usarán dos canales de salida, uno para hacer girar al motor en sentido horario (canal 1) y otro para invertir la polaridad, haciendo girar al motor en sentido anti-horario (canal 2). Ambas funciones serán muy similares siendo la principal diferencia el uso

de los registros MATCH. Para el caso de la modulación por ancho de pulsos, el registro MR0 (el encargado de establecer el periodo de la señal de salida) será fijo y los registros MR1 y MR2 (encargados de establecer el tiempo en alto para los canales 1 y 2, respectivamente) variarán en función del valor de la señal de control. En el caso de la modulación PFM el funcionamiento es el opuesto, es el registro MR0 el que variará y cambiará el periodo de la señal de salida, y por tanto su frecuencia, en función del valor de la señal de control, mientras MR1 y MR2 permanecen constantes.

Además, se desarrollarán otras dos funciones que se encargarán de ajustar la velocidad del motor en función del valor de control. Una para el caso de la modulación PWM y otra para la modulación PFM.

Finalmente, se desarrollará una función dedicada a parar el motor a su llamada. Ésta únicamente tendrá el objetivo de hacer que los registros MR1 y MR2 sean nulos. A continuación activará los bits correspondientes del registro LER.

5.2.2.1. Programación de la modulación PWM

Siguiendo lo descrito anteriormente, en primer lugar se desarrollará la función encargada de inicializar la modulación PWM. Debido a que el objetivo de ésta será hacer los ajustes previos necesarios para preparar el periférico, será importante asegurar que el motor no gire tras su ejecución. Para ello, se introducirá en los registros MR1 y MR2 un cero, ya que, según el manual del microcontrolador: "Todas las salidas PWM controladas por un solo flanco comenzarán con un flanco de subida al principio de cada ciclo de la señal PWM, a menos que su valor en MATCH sea igual a 0". El valor del registro MR0 será igual al periodo de muestreo del controlador, (4.115), 500 microsegundos. El cual implica que será necesario realizar la preescala del *Timer* del periférico PWM en microsegundos. El diagrama de flujo de la función encargada de inicializar la modulación PWM se muestra en la figura 5.11.

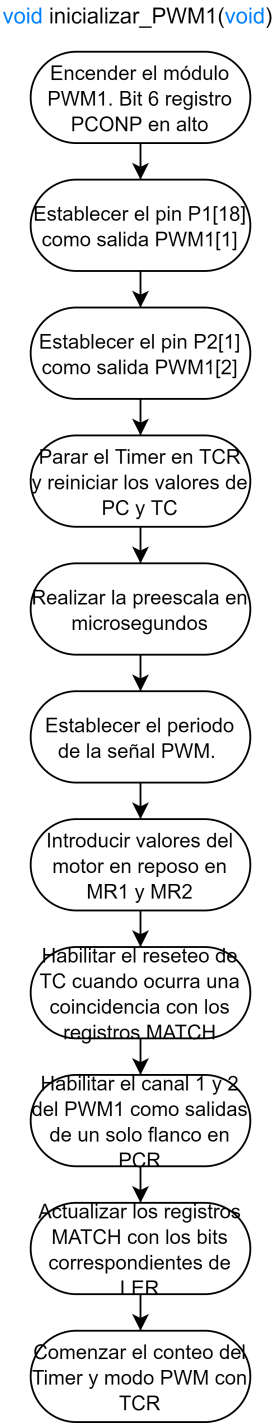


Figura 5.11: Diagrama de flujo de la función: Iniciar PWM.

A continuación, se realizará la función encargada de ajustar la velocidad del motor mediante modulación por ancho de pulsos. Para ello, se calculará el tiempo en alto de la señal de salida aplicando las mismas ecuaciones que las empleadas en la simulación del bloque PWM. Es decir, partiendo de (4.16) y sustituyendo (4.17), se obtuvo:

$$u = D \cdot 6 \tag{5.1}$$

$$D = \frac{u}{6} \tag{5.2}$$

$$\frac{t_{on}}{T_{PWM}} = \frac{u}{6} \tag{5.3}$$

$$t_{on} = \frac{u \cdot T_{PWM}}{6} \tag{5.4}$$

$$t_{on} = \frac{u \cdot T_m}{6} \tag{5.5}$$

El procedimiento llevado a cabo en la función es el siguiente: En primer lugar, debido a que matemáticamente es posible que el valor de la señal de control sea mayor que el voltaje nominal del motor, se limitará su valor máximo a ±6 voltios. A continuación se realizará el desarrollo matemático anterior. El resultado para cada valor de control se introducirá en MR1 si el valor de control es positivo y en MR2 si el valor de control es negativo. En cada caso, el registro opuesto será nulo. Debido a que únicamente es posible introducir en dichos registros números enteros positivos, será necesario redondear en primer lugar el resultado. También, se tendrá en cuenta que si se obtiene un tiempo en alto de 0 segundos, es obligatorio establecer que su valor sea de 1, ya que, como se describió en el apartado del *Timer*, se le restará siempre 1 al valor a introducir en los registros MATCH. Con esto se evita insertar un tiempo negativo. Finalmente se activarán los bits correspondientes a los registros MR1 y MR2 en el registro de actualización LER. El diagrama de flujo de esta función se muestra en la imagen 5.12.

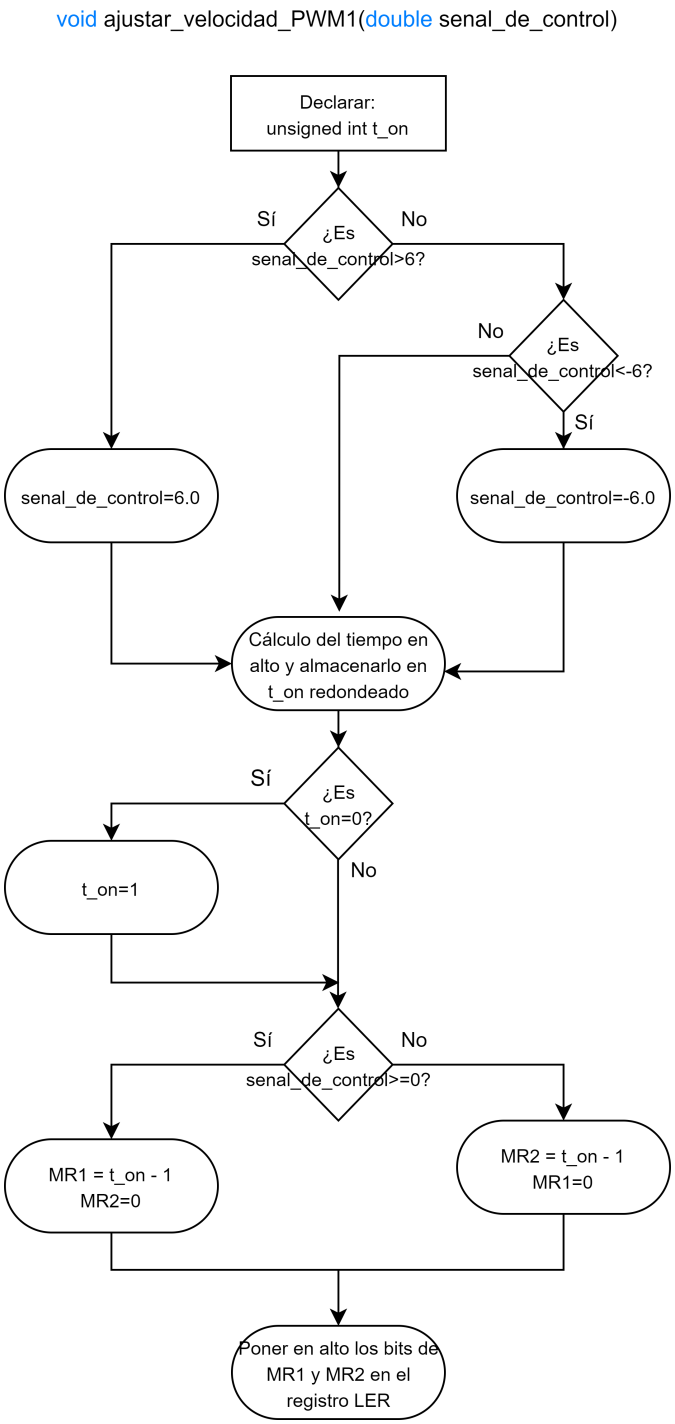


Figura 5.12: Diagrama de flujo de la función: Ajustar velocidad mediante PWM.

5.2.2.2. Programación de la modulación PFM

La programación de la modulación PFM comenzará con la función encargada de realizar las preparaciones necesarias para llevarla a cabo, asegurando que el motor no gire tras su ejecución. Para ello, se establecerá en el registro MR0 una frecuencia que el motor filtrará por completo mientras los valores introducidos en MR1 y MR2 permanecen constantes. El valor introducido en estos registros deberá ser el establecido como *t_on_PFM* en la simulación (4.116), 50 microsegundos. Para el caso de MR0 será necesario tener en cuenta el diagrama de Bode del motor. De acuerdo con dicho diagrama, 4.41, una frecuencia de 200 KHz, que se corresponde con un periodo de 5 microsegundos, será suficiente. De nuevo, para el caso de la modulación PFM, será necesario hacer la preescala del *Timer* del periférico PWM en microsegundos. El diagrama de flujo de este procedimiento se muestra en la figura 5.13.

Figura 5.13: Diagrama de flujo de la función: Iniciar PFM.

A continuación, se programará la función encargada de ajustar la velocidad del motor mediante modulación por frecuencia de pulsos. Con ese fin, se comenzará calculando el periodo de la señal de salida aplicando las mismas ecuaciones que las empleadas en la simulación del bloque PFM. Se obtuvo la ecuación (4.30):

$$T_{PFM} = \frac{t_{on} \cdot V_{max}}{u} \tag{5.6}$$

Sustituyendo:

$$T_{PFM} = \frac{t_on_PFM \cdot 6}{u} \tag{5.7}$$

El procedimiento llevado a cabo por la función es el siguiente: En primer lugar, debido a que matemáticamente es posible que el valor de la señal de control sea mayor que el voltaje nominal del motor, se limitará su valor máximo a ±6 voltios. A continuación se realizará el desarrollo matemático anterior. En este caso, es necesario evitar la división por cero. Para ello, se establecerá que T_{PFM} sea directamente 1 en tal caso, ya que posteriormente será restado. Para el resto de valores, el resultado se introducirá en MR0. El valor constante establecido como tiempo en alto de la señal PFM, denominado t_on_PFM, se insertará en MR1 cuando el valor de control sea positivo y en MR2 si es negativo. En cada caso, el registro opuesto será nulo. Debido a que únicamente es posible introducir en dichos registros números enteros positivos, es necesario redondear previamente el resultado obtenido. También se deberá tener en cuenta que si se obtiene un periodo de 0 segundos es obligatorio establecer que su valor sea 1, ya que, como se describió en el apartado del *Timer*, se le restará siempre 1 al valor a introducir en los registros MATCH. Con esto se evita insertar un tiempo negativo. Finalmente se activarán los bits correspondientes a los registros MR0, MR1 y MR2, en el registro de actualización LER, para que puedan ser usados en el siguiente periodo de la señal PFM.

El diagrama de flujo de esta función se muestra en la imagen 5.14.

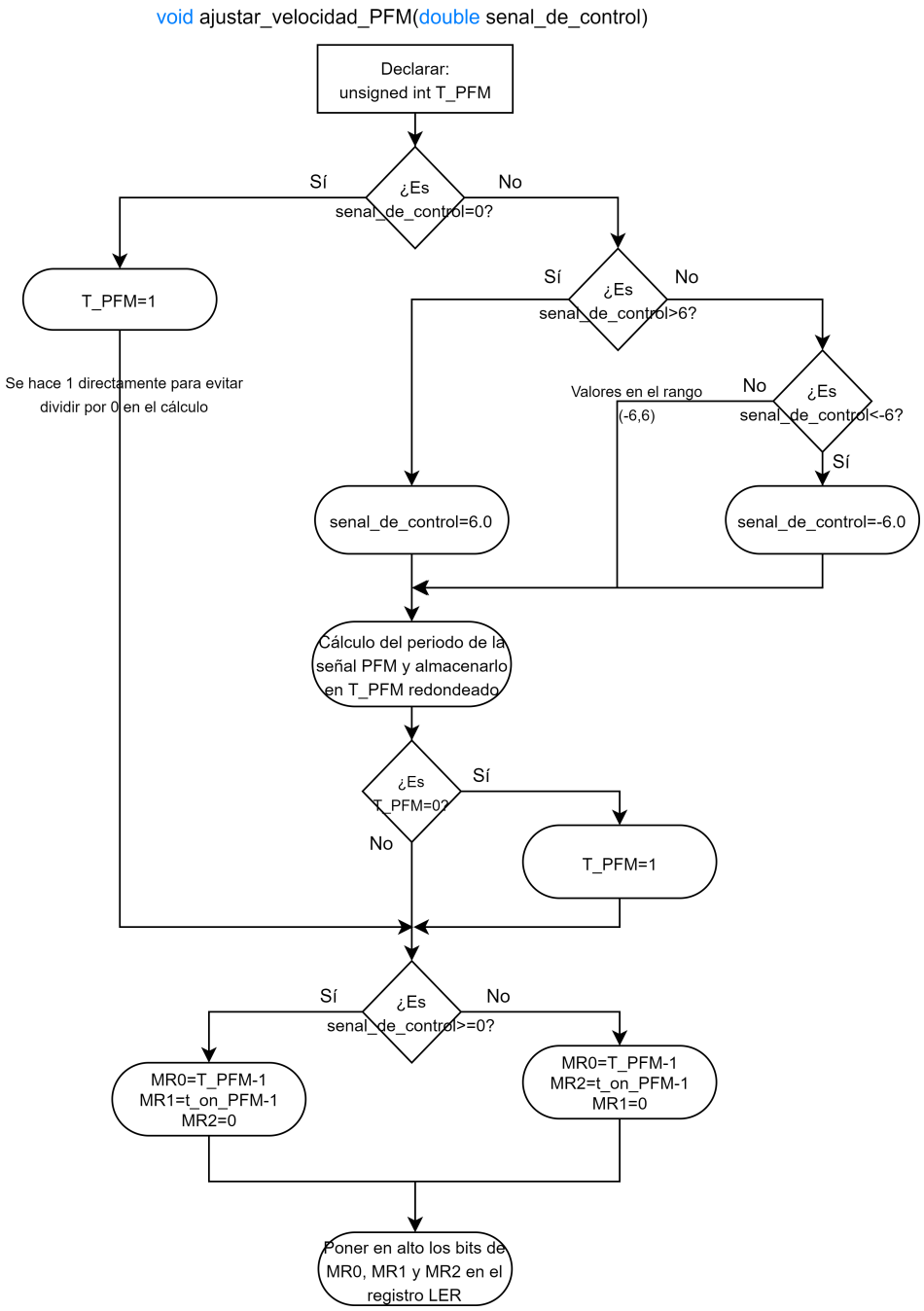


Figura 5.14: Diagrama de flujo de la función: Ajustar velocidad mediante PFM.

Para mostrar el funcionamiento de la modulación PFM, se le introduce manualmente como entrada un valor de señal de control fijo de 5 (voltios). Observando en el osciloscopio su resultado, se observa que se produce una señal con un ancho de pulso fijo de 50 microsegundos y una frecuencia de 16,67KHz. Se muestra dicha señal en la imagen 5.15.

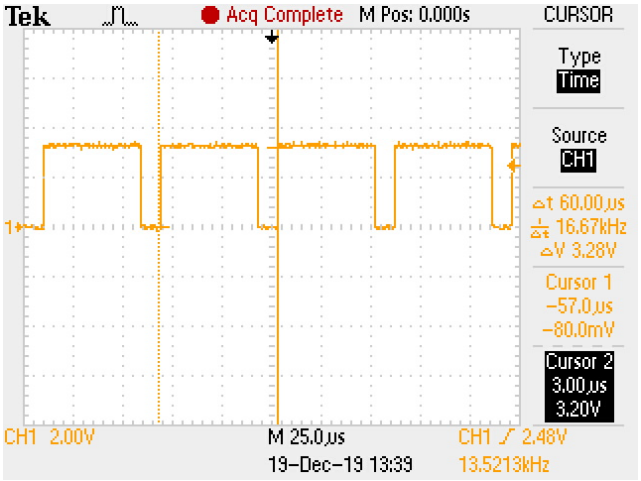


Figura 5.15: Captura de la pantalla de un osciloscopio de la señal PFM con una entrada de 5 (voltios).

En el caso de una entrada fija de 4 (voltios), se observa que, en este caso, la frecuencia de la señal es de 13.513 KHz, mientras el ancho de pulso es el mismo que en el caso anterior, 50 microsegundos. Se muestra dicha señal en la imagen 5.16.

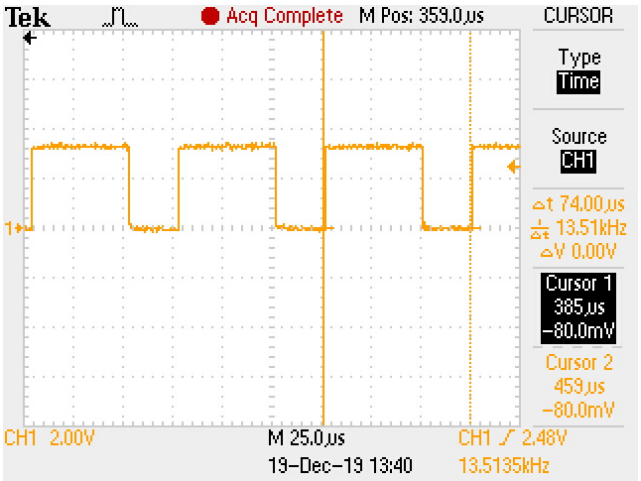


Figura 5.16: Captura de la pantalla de un osciloscopio de la señal PFM con una entrada de 4 (voltios).

5.2.3. Programación y funcionamiento del periférico QEI

La *Quadratic-Encoder-Interface* (QEI) decodifica los pulsos digitales provenientes de un encoder cuadrático para integrar la posición a través del tiempo y determinar la dirección de rotación. Además, puede capturar la velocidad de giro de la rueda del encoder y compararla con otro valor de velocidad de referencia[6]. Para el modo de captura de la velocidad incorpora un *timer* interno exclusivo. El periférico debe ser encendido aplicando alimentación mediante el bit 18 del registro PCONP. Para seleccionar los pines de entrada y salida se emplea el registro IOCON.

El periférico soporta dos modos de distintos de tratamiento de las señales de entrada: El modo fase en cuadratura y el modo reloj/dirección. En el modo fase en cuadratura, se asume que el encoder produce dos señales de reloj que se encuentran desfasadas 90 grados, una para el canal A y otra para el canal B. Estas entradas serán denominadas "PhA" y "PhB", respectivamente. La relación entre los flancos de ambas es usada para determinar la dirección de rotación. En el modo reloj/dirección se asume que el encoder produce una señal de reloj que indica los pasos de graduación y una señal de dirección que indica el sentido de rotación. En este caso se seleccionará el modo de fase en cuadratura asegurando que el bit 1 del registro de configuración, CONF, se encuentra a nivel bajo. El procedimiento empleado para detectar la dirección de giro del motor es el descrito a continuación:

Cuando un flanco de subida o de bajada, de la entrada que se reciba primero ("PhA" o "PhB"), haya sido detectado, se leerá también el estado de la otra entrada. En función de las distintas combinaciones posibles de ambas, se define la tabla de estados mostrada en la imagen 5.17.

Phase A	Phase B	state
1	0	1
1	1	2
0	1	3
0	0	4

Figura 5.17: Tabla de estados del periférico QEI proporcionada en el *User Manual*.[6]

Si la transición entre los distintos estados es ascendente, se determinará que la dirección de giro es positiva y se aumentará el valor de posición si dicho

modo ha sido activado. En caso contrario, si la transición entre los distintos estados es descendente, se establecerá que la dirección de giro es negativa y se disminuirá el valor de posición. Una regla para detectar de forma más sencilla qué tipo de transición entre estados se está realizando es la siguiente: cuando los flancos de subida de la entrada "PhA" sean detectados antes que los de la entrada "PhB" la transición será ascendente, en caso contrario, será descendente. Este procedimiento se muestra en la tabla de la imagen 5.18.

from state	to state	Direction
1	2	positive
2	3	
3	4	
4	1	
4	3	negative
3	2	
2	1	
1	4	

[1] All other state transitions are illegal and should set the ERR bit.

Figura 5.18: Tabla de transición de estados del periférico QEI proporcionada en el *User Manual.*[6]

En el caso de que un flanco de subida y bajada (o al contrario), consecutivos, sean detectados en una de las entradas, mientras en la otra entrada no se haya detectado ningún flanco, la dirección de rotación habrá cambiado. En este momento, cambiará el valor del bit de dirección DIR contenido en el registro de estado, STAT. Se hará "1" si la dirección de giro es negativa y "0" si es positiva, en función de la tabla de transición de los estados 5.18. En el caso de que se quiera invertir los canales de entrada, (el canal B del encoder conectado a la entrada "PhA") existe la posibilidad de activar el bit inverso de dirección, DIRINV, en el registro CONF. Éste bit complementará el valor del registro de dirección, de tal forma que, leyendo el valor de ambos, se determinará la dirección de rotación según la tabla de la imagen 5.19.

DIR bit	DIRINV bit	direction
0	0	forward
1	0	reverse
0	1	reverse
1	1	forward

Figura 5.19: Tabla de dirección del periférico QEI proporcionada en el *User Manual*.[6]

Un ejemplo de funcionamiento proporcionado por el fabricante en el *User Manual*, es el mostrado en la imagen 5.20.

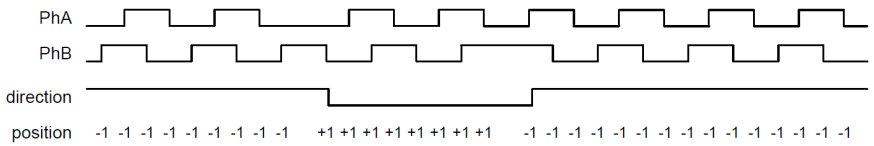


Figura 5.20: Ejemplo de la detección de la dirección de giro del QEI proporcionado en el *User Manual*.[6]

En la que se observa que, al inicio, los flancos de subida de la entrada "PhB" llegan antes que los de la entrada "PhA", por tanto, según la regla descrita anteriormente, se sabe que la transición de estados es descendente, es decir, comienza en el estado 4 ($PhA = 0$ y $PhB = 0$), a continuación pasa al estado 3 ($PhA = 0$ y $PhA = 1$), etc. Según la tabla de transiciones entre estados, 5.18, se determina que la dirección es negativa y por tanto el valor del bit de dirección será "1". Cuando los flancos de subida y bajada de la entrada "PhB" se encuentren con valores invariables de la entrada "PhA" se producirá un cambio de dirección. Lo mismo ocurrirá más adelante con una cadena de flancos de bajada y subida de la entrada "PhA".

Debido a que lo que se quiere obtener es la velocidad de giro del motor, el modo de funcionamiento adecuado de este periférico es el de captura de velocidad. Los registros necesarios para el funcionamiento de este modo se presentan a continuación[6]:

- Registro del *timer* de velocidad (TIME): Este registro contiene el valor actual del *timer* de velocidad.

- Registro contador de velocidad (VEL): Contiene la cuenta en curso de pulsos de velocidad para el periodo actual.
- Registro de recarga del *timer* de velocidad (LOAD): Este registro contiene el valor de recarga del timer de velocidad.
- Registro de captura de velocidad (CAP): Contiene el valor más reciente de velocidad. Se corresponde con el número de pulsos de velocidad contados en el periodo anterior del *timer* de velocidad (VEL).
- Registro de control de la QEI (CON): Este registro contiene los bits que controlan la operación de posición y velocidad del periférico QEI. Para este caso será de importancia el bit 2, que al ser establecido en alto, se encargará de reiniciar el contador de velocidad (VEL), recargar el *timer* de velocidad (TIME) con el valor almacenado en el registro de recarga (LOAD) y reajustar el valor de comparación, si dicha función es habilitada.
- Registro de configuración de la QEI (CONF): Este registro contiene la configuración del periférico QEI. En él se podrá seleccionar la activación del bit inverso de dirección DIRINV, estableciendo en "1" el bit 0. También, se seleccionará el modo de tratamiento de las señales de entrada a través del bit 1 y cuál de los dos canales de las dos señales de entrada del encoder deberá ser tenido en cuenta para el cálculo de la velocidad. Es posible elegir, mediante el bit 2, que únicamente se tenga en cuenta la señal proveniente del canal A o ambos canales simultáneamente, asegurando una mayor resolución a costa de un menor rango del contador de velocidad. Asegurando que el bit 2 sea un "0", se tendrán en cuenta ambos canales para la obtención de la velocidad.
- Registro de interrupciones habilitadas (IE): Este registro de solo lectura, muestra que interrupciones han sido activadas si sus bits correspondientes se encuentran en alto.
- Registro de activación de interrupciones (IES): Permite que se generen interrupciones ante determinados eventos. Escribe un "1" en el bit correspondiente del registro IE.
- Registro de borrado de interrupciones (IEC): Escribir un "1" en los bits correspondientes de este registro borrará interrupciones establecidas. Escribe un "0" en el bit correspondiente del registro IE.
- Registro de estado de las interrupciones (INTSTAT): Este registro indica si se han producido determinados eventos durante la ejecución y al mismo tiempo si alguna interrupción ha sido activada, en el caso de que se hayan sido habilitadas. Los bits en '1" muestran que dicho evento ha ocurrido.

- Registro de borrado (CLR): Escribir "1" en alguno de sus bits borra los estados detectados del registro INTSTAT.
- Registro de estado general (STAT): El primer bit de este registro muestra el valor del bit de dirección DIR. El resto de bits no se encuentran disponibles.
- Registros de filtrado digital de las señales de entrada (FILTERPHA Y FILTERPHB): El filtrado funciona estableciendo en estos registros los ciclos de la señal de reloj de periféricos en los que un valor de los canales de entrada deberá permanecer invariable. El valor mínimo es 1 y el máximo 4.294.967.295 (correspondiente a 0xFFFF FFFF).

Es necesario comentar que el filtrado de las señales de entrada es imprescindible, ya que, si no se filtran las señales de entrada es posible que no se lea correctamente el sentido de giro del motor. Por ejemplo, en el caso de introducir un *Set-Point* negativo, indicando que se quiere un sentido de giro opuesto (anti-horario), si se interpreta de forma incorrecta el sentido de giro del motor se producirá una realimentación positiva en los controladores, haciendo que el motor gire a velocidad máxima. Es decir, en la obtención del error entre el valor asignado como *Set-Point* y el valor de velocidad del motor muestreado, los valores no serán restados, debido a que interpretará el valor de velocidad como positivo: $error = (-SetPoint) - Velocidad$. Para que se produzca un correcto cálculo del error es necesario que la velocidad muestreada con un *Set-Point* negativo también se interprete como tal. Para asegurar la menor pérdida de precisión el número de ciclos a introducir en los registros de filtrado debe ser el menor posible.

El procedimiento llevado a cabo para la captura de datos de velocidad provenientes del encoder descrito en el manual[6] proporcionado por el fabricante es el siguiente: El *timer* de velocidad contará el número de ciclos de la señal de reloj de periféricos (PCLK) en un periodo de tiempo dado. Esta cuenta se almacenará en el registro contador del *timer*, TIME. Al mismo tiempo, se contará el número de flancos, tanto de subida y de bajada, de las señales provenientes de los canales del encoder. En función de lo configurado en el registro CONF, se tendrá en cuenta únicamente el canal A o ambos. El número flancos contados se almacenará en el registro contador de velocidad, VEL. Cuando el *timer* de velocidad desborde, el contenido del registro VEL se transferirá al registro de captura de velocidad (CAP). A continuación se limpiará el contenido de VEL, y se recargará el registro TIME con el contenido del registro de recarga (LOAD). Finalmente se establecerá en alto el bit TIM_Int del registro INTSTAT.

Para traducir los datos del contador de velocidad en revoluciones por minuto el fabricante proporciona en su manual la siguiente fórmula:

$$RPM = \frac{PCLK \cdot CAP \cdot 60}{LOAD \cdot PPR \cdot Flancos} \tag{5.8}$$

Donde "PCLK" es el número de ciclos de la señal de reloj del periféricos en un segundo, "PPR" es el número de pulsos por revolución del encoder a emplear y "Flancos" es 2 o 4 en función de si se cuentan los flancos de subida y bajada únicamente del canal A o de ambos.

Además, en el manual[6], se incluye un ejemplo del cálculo de la velocidad que será descrito a continuación:

Se supone que un motor gira a 600 revoluciones por minuto y que tiene acoplado un encoder cuadrático de 2048 pulsos por revolución. También, se supone que la frecuencia del reloj de periféricos es de 10000 Hz y que se quiere un muestreo del encoder cada cuarto de segundo. Finalmente, se considera que se quieren contar los flancos tanto del canal A como del canal B.

En primer lugar en el registro CONF, se establecerá que se tendrán en cuenta tanto los flancos de ambos canales. Es decir, cada periodo de las señales de reloj provenientes de los canales A y B tendrán en total cuatro flancos (dos de subida y bajada del canal A y otros dos del canal B), por tanto el encoder proporcionará:

$$2048\ [\text{pulsos por revolución}] \cdot 4\ [\text{flancos}] = 8192\ [\text{flancos por revolución}] \tag{5.9}$$

600 revoluciones por minuto equivalen a 10 revoluciones por segundo, por tanto se tendrán 81920 pulsos por segundo. Como se quiere muestrear cada cuarto de segundo se contarán:

$$CAP = 81920\ [\text{pulsos por segundo}] \cdot \frac{1}{4}\ [\text{segundo}] = 20480\ [\text{pulsos por muestreo}] \tag{5.10}$$

El valor del registro de recarga LOAD será un cuarto de segundo en ciclos de la señal de reloj de periféricos, es decir:

$$\begin{aligned} 10000\ [\text{ciclos}] &-> 1\ [\text{segundo}] \\ LOAD\ [\text{ciclos}] &-> 1/4\ [\text{segundo}] \\ LOAD &= \frac{10000}{4} = 2500\ [\text{ciclos}] \end{aligned} \tag{5.11}$$

Finalmente sustituyendo en la ecuación proporcionada por el fabricante 5.8 se obtiene:

$$RPM = \frac{10000 \cdot 20480 \cdot 60}{2500 \cdot 2048 \cdot 4} = 600[rpm] \tag{5.12}$$

En este ejemplo se muestra cierta ambigüedad en el funcionamiento del *timer* incluido en el periférico QEI en el manual proporcionado por el fabricante. Según el funcionamiento teórico descrito anteriormente, el contenido del registro del contador de velocidad (VEL) será transmitido al registro de captura (CAP) cuando el *timer* desborde. Sin embargo, en el ejemplo, se

interpreta que el dato de velocidad será capturado cuando se hayan contado el número de ciclos establecido en el registro de recarga LOAD. No se proporciona mayor información al respecto del funcionamiento del *timer* interno de este periférico. En definitiva, es el número de ciclos a contar establecido en el registro LOAD, el que determinará el periodo de muestreo de los controladores PID e I-PD.

Por tanto, para el caso concreto de este trabajo en el que se usa un encoder con 512 pulsos por revolución, un periodo de muestreo de $5 \cdot 10^{-4}$ segundos (500 microsegundos) y una frecuencia del reloj de periféricos establecida por defecto en 60 MHz; se tiene, suponiendo una velocidad de giro de 600 revoluciones por minuto:

$$512\ [\text{PPR}] \cdot 4\ [\text{flancos}] = 2048\ [\text{flancos por revolución}] \tag{5.13}$$

$$600\ [\text{rpm}] = 10\ [\text{rps}] \tag{5.14}$$

$$2048\ [\text{FPR}] \cdot 10\ [\text{rps}] = 20480\ [\text{pulsos por segundo}] \tag{5.15}$$

$$CAP = 20480\ [\text{PPS}] \cdot 5 \cdot 10^{-4}\ [\text{s}] = 10{,}24\ [\text{pulsos por muestreo}] \tag{5.16}$$

$$LOAD = 60 \cdot 10^{6} \cdot 5 \cdot 10^{-4} = 30000\ [\text{ciclos}] \tag{5.17}$$

$$RPM = \frac{60000000 \cdot 10{,}24 \cdot 60}{30000 \cdot 512 \cdot 4} = 600[rpm] \tag{5.18}$$

Como no es posible contar 10.24 pulsos habrá una pérdida de precisión, en concreto con 10 pulsos se obtiene: $585,9375$[rpm]. Se emplea una velocidad de 600 rpm para comparar con el ejemplo proporcionado por el fabricante.

Teniendo en cuenta el funcionamiento del periférico, se desarrollarán dos funciones: una para inicializarlo y otra para obtener datos de velocidad a partir de las señales del encoder.

El procedimiento realizado en la función dedicada a inicializar el periférico es el siguiente: En primer lugar se aplicará alimentación al periférico mediante el registro PCONP. Después, se desactivará la generación de interrupciones mediante el registro IEC, ya que, no se emplearán interrupciones para el manejo del periférico; por el contrario, se detectarán los eventos directamente, leyendo los valores de los bits de velocidad (TIM_Int) y de dirección (DIR_Int) del registro de estado, INTSTAT. A continuación, se limpará el contenido de INTSTAT mediante el uso del registro CLR, con el objetivo de asegurar que no haya un estado activo de una posible ejecución anterior. Luego, se seleccionará el modo de tratamiento de las señales de entrada como señales de fase en cuadratura, es decir, las entrada del periférico "PhA" estará dedicada al canal A del encoder y la entrada "PhB" al canal B, por tanto no será necesario tener en cuenta el inverso del bit de dirección. No se realizará un filtrado digital a ambas señales. También, se establecerá que se deberán tener en cuenta los flancos de subida y bajada de ambos canales de entrada. El tiempo de recarga del timer del registro LOAD, será igual al periodo de muestreo seleccionado para los controladores en ciclos de

la señal de reloj de periféricos. La regla es la siguiente:

$$\begin{aligned} LOAD &= \text{ciclos de PCLK en un segundo} \cdot \text{periodo de muestreo} \\ LOAD &= 60 \cdot 10^6 \cdot 500 \cdot 10^{-6} = 30000 \text{ ciclos} \end{aligned} \tag{5.19}$$

El diagrama de flujo de la función inicializar QEI se muestra en la imagen 5.21.

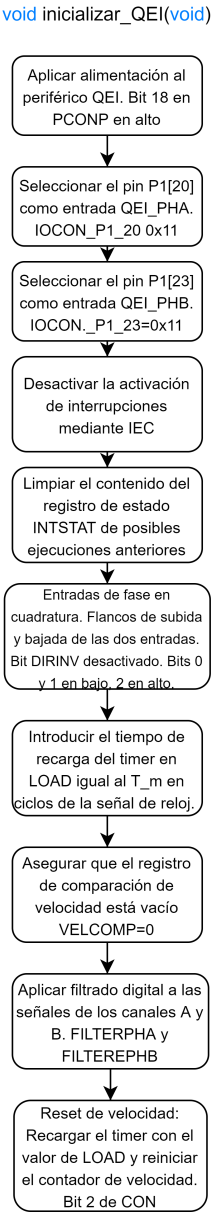


Figura 5.21: Diagrama de flujo de la función: Iniciar QEI.

El funcionamiento de la función dedicada a obtener la velocidad del encoder es el siguiente: Comenzará esperando que el bit del *timer* de velocidad del registro INTSTAT esté activo. Ésto se producirá cuando el timer cuente un número de ciclos igual al establecido en el registro LOAD. A continuación se resolverá la ecuación proporcionada por el fabricante para obtener la velocidad en revoluciones por minuto (5.8). Debido a que el encoder está colocado en una posición anterior a la reductora, es necesario aplicar el factor de reducción de ésta, igual a 112:1, para obtener la velocidad real de giro. A continuación, se comprobará si se ha detectado un cambio de dirección leyendo el estado del bit de dirección del registro INTSTAT. En caso afirmativo, se comprobará si el bit de dirección, DIR, del registro de estado general, STAT, está activo. En caso afirmativo, se hará que la velocidad calculada tenga signo negativo, indicando que el motor se encuentra girando en el sentido opuesto (anti-horario), según la tabla ref. Finalmente, se convertirá la velocidad obtenida a radianes por segundo.

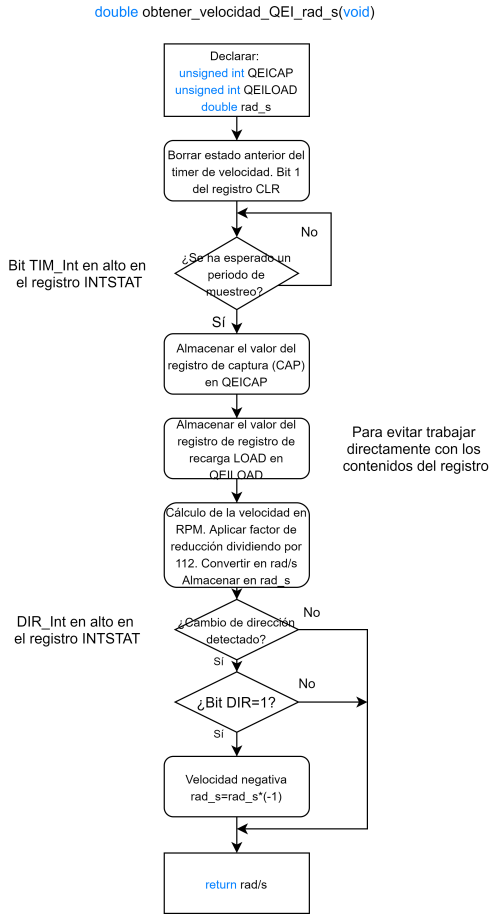


Figura 5.22: Diagrama de flujo de la función: Obtener velocidad QEI.

El diagrama de flujo de la función dedicada a obtener datos de velocidad del encoder se muestra en la imagen 5.22.

5.2.4. Envío de datos a través de la UART a *Matlab*

El funcionamiento de la comunicación Transmisor-Receptor Asíncrono Universal (UART) se describirá de forma más breve que los apartados anteriores, ya que, no es estrictamente necesario para el control de la velocidad del motor. Sin embargo, hay aspectos que son imprescindibles para el funcionamiento del proyecto completo.

Un adaptador USB-UART basado en el chip *MAX-232* será el encargado de enviar los datos a un ordenador con *Matlab* mediante la UART0 disponible en el microcontrolador. Al introducirlo en el puerto USB de un ordenador, creará un puerto serie virtual en él habilitado a niveles de voltaje TTL (*Transistor-Transistor Logic)*, que son aquellos empleados por la comunicación UART. En el caso de emplear un sistema operativo *Windows*, es imprescindible entrar en el *Administrador de Dispositivos* disponible en el *Panel De Control*, para conocer el número asignado al puerto serie virtual. En este caso, será denominado "COMx" siendo "x" el número asignado al puerto. Este número es importante, ya que, se usará para indicar en *Matlab* cuál es el puerto usado para la comunicación. Haciendo click derecho en éste, y a continuación yendo a *Propiedades*, se podrá comprobar la velocidad de transmisión en baudios, junto con el número de bits a enviar y los bits de *stop*. Por defecto, en el puerto serie virtual, la velocidad será de 9600 baudios, se realizarán transmisiones de 8 bits, sin paridad, y con un bit de stop. Por tanto se desarrollará una función para inicializar el periférico UART, empleando la UART0, con las mismas características de transmisión de datos creadas por defecto en el puerto virtual del ordenador. Para encender el periférico es necesario activar el bit 3 del registro PCONP, además se seleccionará el pin P0[2] como salida "Tx" y el pin P0[3] como entrada "Rx" (ver esquema de conexiones 5.2).

El periférico cuenta con un bloque generador de velocidad en baudios. Para ello, divide la frecuencia del reloj de periféricos (PCLK) por un divisor que deberá ser configurado. Esta frecuencia de reloj fraccionada, será sobremuestrada 16 veces, según el protocolo establecido para la comunicación asíncrona.

Para realizar el cálculo de la velocidad en baudios, el fabricante proporciona en el *User Manual*[6] la siguiente ecuación:

$$UARTn_{baudrate} = \frac{PCLK}{16 \cdot (256 \cdot UnDLM + UnDLL) \cdot (1 + \frac{DivAddVal}{MulVal})} \quad (5.20)$$

Donde:

- n es el número de la UART a emplear. Puede ser 0,2 o 3.

- $PCLK$ es la frecuencia del reloj de periféricos, establecida por defecto en 60 MHz.
- $UnDLM$ y $UnDLL$ son los registros que contienen el byte más significativo y el menos significativo, respectivamente, del divisor de la señal de reloj de periféricos para la UART usada.
- La fracción $\frac{DivAddVal}{MulVal}$ introduce una parte fraccionaria a la velocidad calculada, en el caso de que sea necesario, introduciendo ambos valores en el Registro Divisor Fraccional (UnFDR). Por defecto, dicha fracción se encuentra anulada teniendo unos valores de 0 en el numerador y 1 en el denominador.

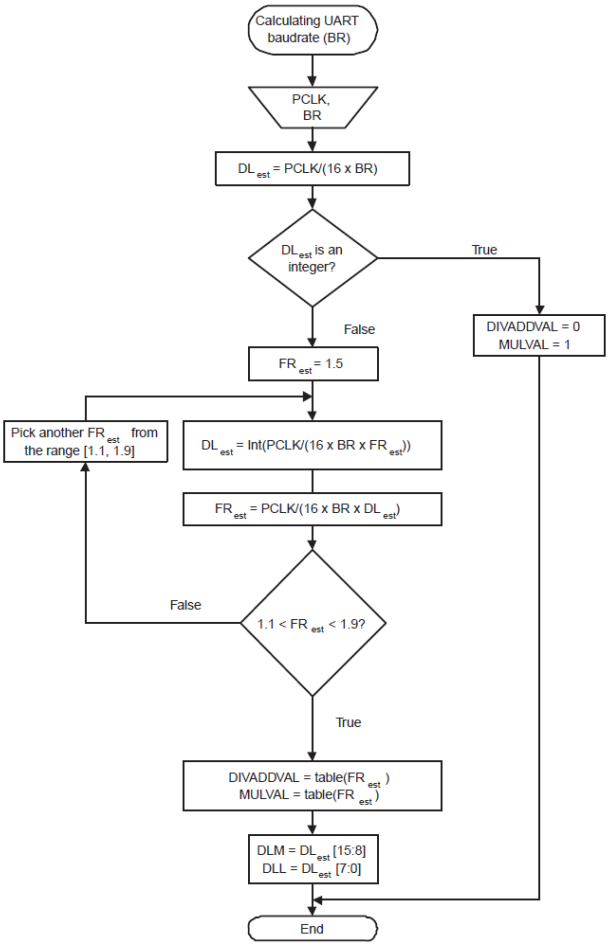


Figura 5.23: Algoritmo proporcionado por el fabricante para calcular la velocidad en baudios.

Teniendo en cuenta la fórmula, la incógnita serán los valores del divisor de la señal de reloj de periféricos ya que el resto de valores deberán ser

introducidos por el usuario. Para hallar dicho divisor, el fabricante propone un algoritmo para el cálculo de éste y de la parte fraccionaria si es necesaria. El algoritmo completo se muestra en la imagen 5.23. Donde DL es el divisor y FR el resultado de la fracción $\frac{DivAddVal}{MulVal}$.

Con el objetivo de simplificar el cálculo del divisor de la velocidad en baudios, se obligará a qué la velocidad introducida sea un número entero. En tal caso, no será necesario emplear el divisor fraccional, ya que se dejará en sus valores por defecto $DivAddVal = 0$ y $MulVal$=1. Por lo cual, según el algoritmo, la fórmula del cálculo del divisor sería:

$$DL = \frac{PCLK}{16 \cdot Baudrate} \tag{5.21}$$

A cuyo resultado es necesario aplicar una máscara para obtener el byte más significativo y menos significativo para ser almacenados en los registros DLM y DLL, respectivamente.

Adicionalmente, se requerirán los registros siguientes:

- Registro de control de transmisión ó *Line Control Register* (LCR): Registro en el que se establecerá el número de bits a enviar, los bits de *stop* y la paridad. Además, activa o desactiva la introducción de los valores en los registros DLM y DLL.

- Registro de control de los *buffer* FIFO (FCR): Registro que habilita o deshabilita los *buffer* FIFO de entrada y salida.

El diagrama de flujo de la función dedicada a inicializar la UART0 es la representada en la imagen 5.24.

Además, se desarrollarán dos funciones encargadas de recibir y transmitir caracteres (8 bits):

La función encargada de transmitir un carácter a través de la UART0 funcionará de la siguiente manera: Recibirá el carácter a transmitir como parámetro. A continuación, esperará a que el registro encargado de la transmisión, el *Transmit-Holding-Register* (THR) esté vacío. Esto se realizará leyendo el bit 5, del registro de estado LSR (*Line-Status-Register*). Si dicho bit es "1", se podrá realizar la transmisión almacenando el carácter recibido como parámetro en el THR. El diagrama de flujo de la función se muestra en la imagen 5.25.

void inicializar_uart0_con_ocho_bits_sin_paridad_un_stop_bit(unsigned int velocidad)

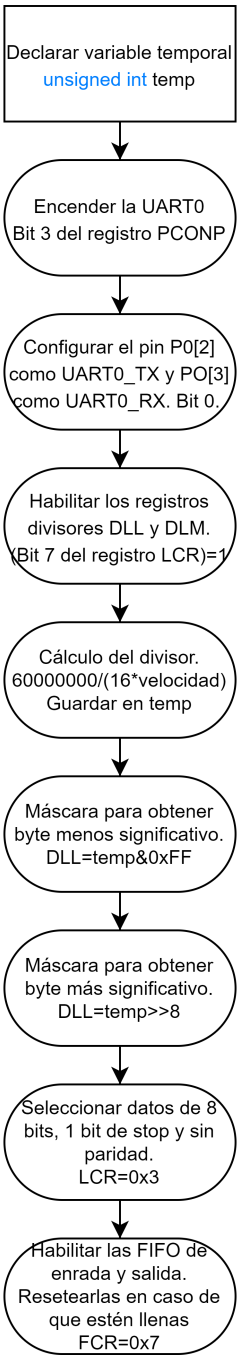


Figura 5.24: Diagrama de flujo de la función: Inicializar UART0. Transmisión de 8 bits, 1 bit de *stop* y sin paridad.

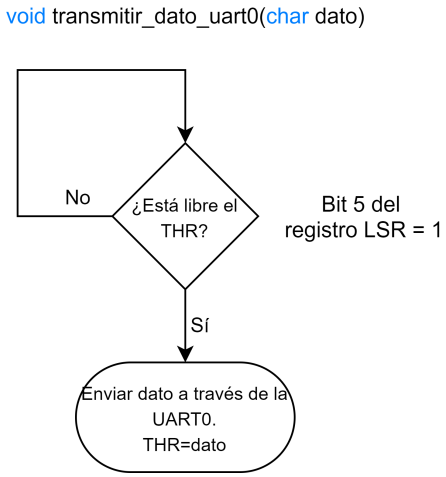


Figura 5.25: Diagrama de flujo de la función: Transmitir dato por la UART0. 8 bits.

La función encargada de recibir un carácter a través de la UART0, funcionará de forma similar: Esperará a que el *buffer FIFO (First In, First Out)* de entrada de datos esté lleno, lo cual significará que los datos están disponibles para su lectura en el registro *buffer* de entrada (RBR). Para ello, es necesario comprobar que el primer bit del registro de estado, LSR, sea "1". Cuando ocurra, se retornará el carácter almacenado en el registro RBR. El diagrama de flujo de la función se muestra en la imagen 5.26.

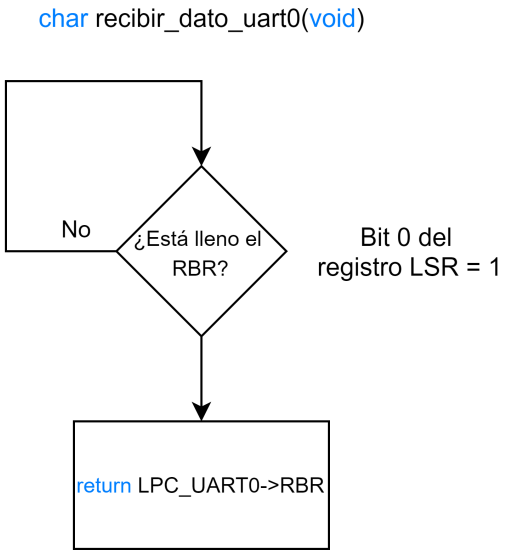


Figura 5.26: Diagrama de flujo de la función: Recibir dato por la UART0. 8 bits.

Una vez desarrolladas las funciones necesarias para transmitir y recibir caracteres, se hará una función encargada de recibir una cadena de caracteres.

El funcionamiento de dicha función es el siguiente: Recibirá como parámetro un puntero a la dirección de memoria de la primera posición de la cadena de caracteres. A continuación, leerá el carácter. Mientras éste no sea el carácter fin de cadena, "/0", o el valor 0 en decimal según la tabla del código ASCII, se transmitirá el carácter a través del registro LSR empleando la función anterior. Finalmente, se desplazará una posición la dirección de memoria del puntero a la cadena de caracteres. El diagrama de flujo de esta función se presenta en la imagen 5.27.

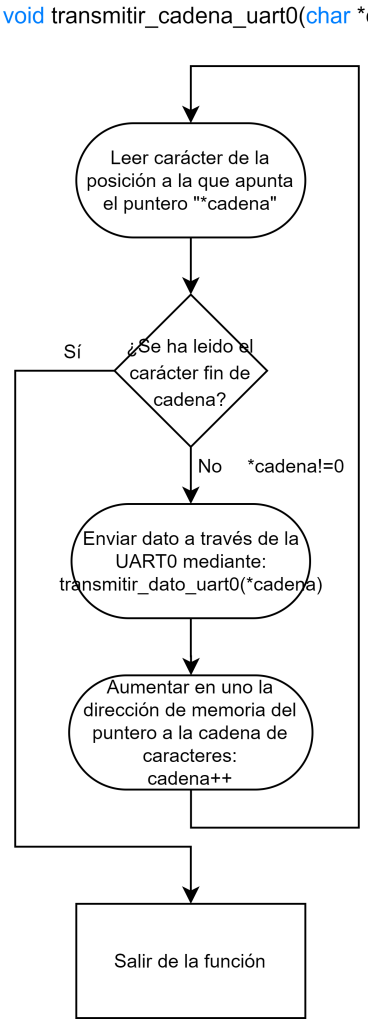


Figura 5.27: Diagrama de flujo de la función: Inicializar Uart0. Transmitir cadena de caracteres por la UART0.

Con ella, se enviarán los datos de velocidad del motor a un ordenador con *Matlab* como una cadena de caracteres a través de la UART0. El procedimiento llevado a cabo es el descrito a continuación:

- Una vez se lea un dato de velocidad del encoder, ya sea mediante el control PID o I-PD, se almacenará en un *array* de datos denominado *buffer*.
- Cuando éste se haya llenado, y el tiempo de giro establecido del motor se haya cumplido, se esperará a recibir el carácter "S" a través de la UART0.
- Cuando se emita dicho carácter empleando *Matlab* a través de la UART0, se recorrerá cada una de las posiciones del *buffer*. En cada una de ellas se aplicará la función *snprintf()* que formateará como cadena de caracteres el número de cada posición, y lo almacenará en un *array* concreto.
- Este array se denominará "cadena" y será de 7 posiciones de tipo "char". La penúltima posición de éste se sobreescribirá con el carácter nueva línea, "/n" , que indicará a *Matlab* el final de la cadena, seguido del carácter fin de cadena "/0".

El mismo procedimiento se llevará a cabo para la emisión a *Matlab* del *Set-Point* establecido en el microcontrolador, a través de la UART0. En este caso se empleará otro *array buffer* distinto, denominado $buffer_setpoints$.

Para calcular el tamaño de ambos *arrays* que van a contener tanto los datos de velocidad como el *Set-Point* a lo largo de cada periodo de muestreo, se empleará la siguiente relación:

$$NUMDATOS = \frac{t_sim}{T_m_seg} \tag{5.22}$$

Donde:

- t_sim es el tiempo de almacenamiento de datos. Deberá coincidir con el tiempo que se va a emplear en la simulación, para en el caso en el que se quiera realizar la comparación con la respuesta de la simulación, tengan el mismo eje de tiempos. Es una constante que deberá ser fijada antes de la ejecución en el fichero "constantes.h".
- T_m_seg es el periodo de muestreo de los controladores en segundos. Es una constante que deberá ser fijada antes de la ejecución en el fichero "constantes.h".

En *Matlab* será necesario elaborar un *script* que se encargará de enviar el carácter "S" para comenzar la emisión de datos de velocidad del microcontrolador a través de la UART0, y las variables necesarias para almacenarlos.

Dicho *script* se denominará "Comunicacion_Uart_COM.m" y desarrollará el siguiente procedimiento tras su ejecución:

- En primer lugar, creará la conexión a través del puerto serie empleando la variable "s". En ella se almacenarán los datos necesarios para la comunicación a través del puerto serie empleando la función *serial()*. En ella, es obligatorio establecer el puerto COMx a usar y la velocidad en baudios.

- A continuación se declaran las variables t_sim y T_m. Ambas tendrán la misma función que las empleadas en la ecuación del número de datos, (5.22). En ellas se deberá introducir el mismo valor que el establecido en las mismas variables en el microcontrolador. Si no, no será posible almacenar los datos de velocidad provenientes del microcontrolador y hacer una representación gráfica con ellos.

- Luego, se declarará la variable "Simulink", que se empleará para determinar si se va a acompañar a la representación gráfica de los datos de velocidad reales con los datos de velocidad de la simulación. Estableciendo el valor "0" en ella se desactiva el empleo de la simulación. Con "1" se activa.

- Despúes, se calculará el tamaño de los *buffer* que contendrán los datos de entrada de velocidad y el *Set-Point*, provenientes del microcontrolador, *buffer* y *buffer_velocidad*, respectivamente. Se empleará la ecuación (5.22).

- A continuación, se declará un vector denominado "eje_x" que contendrá los valores de tiempo correspondientes a cada valor de velocidad recibido. Cada valor se recibirá cada periodo de muestreo, establecido anteriormente como T_m, hasta llegar al valor establecido en t_sim.

- Luego, se abrirá la comunicación a través del puerto serie empleando la función *fopen()*, donde el argumento será la variable "s".

- Después, se emitirá el carácter "S" a través de la UART0, mediante la función *fwrite()*.

- A continuación se recibirán los datos de velocidad provenientes del microcontrolador. Mediante un bucle *for* serán almacenados en cada una de las posiciones del vector de almacenamiento de datos de velocidad, *buffer*. Antes de almacenarlos, será necesario reformatearlos de cadena de caracteres a datos tipo *double* mediante la función *str2double()*.

- Acto seguido, se enviará de nuevo el carácter "S" a través de la UART0, mediante la función *fwrite()*. Esto comenzará la emisión del *Set-Point*,

establecido en el microcontrolador, cada periodo de muestreo. Se almacenarán en el vector *buffer_velocidad*, empleando un bucle *for* de la misma forma que en el caso de los datos de velocidad.

- Una vez recibidos todos los datos del microcontrolador, se finalizará la comunicación a través del puerto serie empleando la función *fclose()*, usando "s" como argumento.

- Finalmente se transformarán los datos de velocidad y de *Set-Point* recibidos en datos de tipo serie de tiempos. Se empleará la función *timeseries()*.

- Los datos serán representados gráficamente de manera automática. En el caso de que se haya activado la comparación con los datos de *Simulink* mediante la variable del mismo nombre, se simulará el archivo de *Simulink* "Control_Recursivo" empleando como *Set-Point* los datos en serie de tiempos recibidos del microcontrolador. Una vez finalizada la simulación se representará automáticamente la gráfica de comparación de resultados.

En el caso de querer realizar la comparación con los datos de la simulación, es necesario realizar unos ajustes previos en el *script*: "Ajuste_Control_Recursivo.m" antes de iniciar el giro del motor y la comunicación con *Matlab*. Dichos ajustes se describen a continuación:

- Asegurar que el tiempo de simulación, t_sim, es el mismo que el introducido en el microcontrolador en "constantes.h" y en el *script* de comunicación "Comunicacion_Uart_COM.m".

- Mediante las variables interruptor denominadas "$sw_Variable$" seleccionar si se quiere control PID o I-PD, modulación PWM o PFM, y si se quiere usar un *Set-Point* de *Simulink* o uno proveniente del microcontrolador. Éste último es el necesario para realizar la comparación con los datos de velocidad reales.

- Asegurar que el valor del periodo de muestreo T_m es el mismo que el introducido en el microcontrolador en T_m_seg en "constantes.h" y en el *script* de comunicación "Comunicacion_Uart_COM.m".

- Asegurar que los valores de las acciones de control de los controladores PID e I-PD son los mismos que los introducidos en el microcontrolador en el fichero "constantes.h".

Finalmente, en el archivo de comunicación se seleccionará que se quiere tener en cuenta los valores de la simulación haciendo la variable "Simulink=1". Para ver el resultado solo es necesario ejecutar el archivo de comunicación, los ajustes del fichero de simulación se aplicarán automáticamente.

En resumen, el procedimiento para realizar la comunicación con *Matlab* es el siguiente:

En el caso de no querer realizar una comparación con la simulación:

- En "Comunicacion_Uart_COM.m", establecer el puerto serie y el mismo valor para las variables t_sim y T_m que en el microcontrolador. Solo es necesario establecer el puerto serie a emplear una vez. Únicamente cambiará el puerto si se emplea otro ordenador. Hacer "Simulink=0".
- Introducir en el microcontrolador los parámetros de control en "constantes.h".
- Compilar y ejecutar mediante *µVision*.
- Ejecutar "Comunicacion_Uart_COM.m" en *Matlab*.

En el caso de querer realizar una comparación con la simulación:

- En "Ajuste_Control_Recursivo.m", establecer los parámetros de la simulación y del control en *Matlab*.
- En "Comunicacion_Uart_COM.m", establecer el puerto serie y el mismo valor para las variables t_sim y T_m que en el microcontrolador. Solo es necesario establecer el puerto serie a emplear una vez. Únicamente cambiará el puerto si se emplea otro ordenador. Hacer "Simulink=1".
- Introducir en el microcontrolador los mismos parámetros de control que en la simulación en "constantes.h".
- Compilar y ejecutar mediante *µVision*.
- Ejecutar "Comunicacion_Uart_COM.m" en *Matlab*.

IMPORTANTE: El archivo de simulación "Control_Recursivo.mld", el archivo de ajuste del mismo "Ajuste_Control_Recursivo.m" y el archivo de comunicación mediante la UART0 "Comunicacion_Uart_COM.m", deben estar en la misma carpeta.

Capítulo 6

Resultados

La velocidad nominal del motor, según su *datasheet* es de 6390 rpm, sin tener en cuenta la reductora acoplada a su eje. Teniendo en cuenta el factor de reducción de 112:1, se obtiene una velocidad nominal de 57,0536 rpm, que se corresponden con 5.975 radianes por segundo. Aproximadamente 6 radianes por segundo. La velocidad máxima sin carga, dada en el *datasheet* del motor como 8200 rpm, será de 68,333 rpm con la reductora, que equivalen a 7,156 radianes por segundo. Para una primera toma de resultados, se elige como *Set-Point* una velocidad intermedia a la velocidad nominal del motor, es decir 3 radianes por segundo.

En primer lugar es necesario conectar el conector JTAG a la placa microcontroladora y a un ordenador; y el adaptador USB-UART a un puerto USB del ordenador. A continuación, se abre el programa *Matlab* donde se seleccionará la carpeta donde se encuentren contenidos los archivos, y el programa μVision, donde se abrirá el proyecto que contiene el archivo de configuración denominado "constantes.h".

Para comprobar cómo se ajusta la respuesta real a la obtenida en la simulación, se realizarán los ajustes comentados en el apartado dedicado a la comunicación entre el microcontrolador y un ordenador con *Matlab*, que son los siguientes:

En el caso de querer realizar una comparación con la simulación:

- En "Ajuste_Control_Recursivo.m", establecer los parámetros de la simulación y del control en *Matlab*.
- En "Comunicacion_Uart_COM.m", establecer el puerto serie y el mismo valor para las variables t_sim y T_m que en el microcontrolador. Solo es necesario establecer el puerto serie a emplear una vez. Únicamente cambiará el puerto si se emplea otro ordenador. Hacer "Simulink=1".
- Introducir en el microcontrolador los mismos parámetros de control que en la simulación en "constantes.h".

- Compilar y ejecutar mediante *µVision*.
- Ejecutar "Comunicacion_Uart_COM.m" en *Matlab*.

Se comenzará estableciendo los parámetros de la simulación del sistema en recursiva en el archivo "Ajuste_Control_Recursivo.m". Los valores de las acciones de control para el controlador PID son los obtenidos en (4.122), que se exponen a continuación:

$$\begin{aligned} Kp_{PID} &= 0{,}6573 \\ Ki_{PID} &= 65{,}73 \\ Kd_{PID} &= 4{,}382 \cdot 10^{-6} \\ t_{fd/PID} &= (K_d/K_p)/10 = 6{,}666 \cdot 10^{-7} \end{aligned} \tag{6.1}$$

Para el caso del controlador I-PD:

$$\begin{aligned} Ki_{I-PD} &= 39{,}699 \\ Kp_{I-PD} &= 0{,}3296 \\ Kd_{I-PD} &= 8{,}5838 \cdot 10^{-5} \\ t_{fd/I-PD} &= 2{,}6043 \cdot 10^{-5} \end{aligned} \tag{6.2}$$

El periodo de muestreo de ambos controladores es el obtenido en (4.115):

$$T_m = 5 \cdot 10^{-4} \text{ segundos } = 500 \text{ microsegundos} \tag{6.3}$$

El periodo de ejecución del bloque PWM en la simulación es el obtenido en (4.116):

$$T_res = 5 \cdot 10^{-8} \text{ segundos } = 0{,}05 \text{ microsegundos } = 50 \text{ nanosegundos} \tag{6.4}$$

Que se corresponde con una resolución de:

$$\frac{T_m}{T_res} = 10000 \tag{6.5}$$

$$\frac{1}{1000} = 1 \cdot 10^{-4} \tag{6.6}$$

El periodo de ejecución del bloque PFM en la simulación debe ser igual al tiempo en alto de la señal PFM. Se obtuvo (4.117):

$$t_on_PFM = 5 \cdot 10^{-5} \text{ segundos } = 50 \text{ microsegundos} \tag{6.7}$$

Por último, se seleccionan los valores del periodo de ejecución de los bloques encoder y frecuencímetro. Éstos estaban determinados por la frecuencia de refresco del encoder proporcionada en su *datasheet*. Para compensar

las discontinuidades introducidas en las respuesta por el conjunto encoder-frecuencímetro en la simulación, se decidió disminuirlos. Se obtuvo (4.57) y (4.78):

$$T_enc = 3 \cdot 10^{-8} \text{ segundos } = 30 \text{ nanosegundos} \tag{6.8}$$

$$T_frec = 1 \cdot 10^{-8} \text{ segundos } = 10 \text{ nanosegundos} \tag{6.9}$$

Finalmente se establecerá un tiempo de simulación, t_sim de 0,3 segundos, y se seleccionará en las variables interruptores del archivo "Ajuste_Control_Recursivo" cuál es la respuesta que se quiere obtener.

Siguiendo con el segundo paso, en el archivo de comunicación, se hará el tiempo de simulación igual a 0.3 segundos y la variable "Simulink" igual a uno, indicando que se quiere realizar una comparación con los datos de la simulación. La primera vez es necesario asegurar que el puerto serie virtual por el que se realizará la comunicación por la UART0 del microcontrolador es el correcto. Según el panel de control de *Windows*, en este caso se trata del puerto "COM4", que funciona por defecto a una velocidad de 9600 baudios, transmitiendo datos de 8 bits, con un bit de stop y sin paridad. Esta configuración por defecto ya se encuentra en el microcontrolador.

A continuación, en el proyecto de μVision, en el archivo de cabecera denominado "constantes.h" se establecerán los mismos valores de ganancia de las acciones de control para ambos controladores, el periodo de muestreo y el tiempo de simulación. Finalmente, se selecciona el control que se quiere realizar y el método de modulación. Se compila y ejecuta el programa.

Cuando el motor pare de girar se ejecuta el *script* del archivo de comunicación con el PC "Comunicacion_Uart_COM.m". Automáticamente se realizará la simulación y se representarán gráficamente los resultados.

6.1. Pruebas iniciales

En primer lugar se realizarán pruebas ante el mismo *Set-Point* para cada controlador, empleando ambos métodos de modulación, y se comparará la respuesta con la obtenida en la simulación con los controladores en recursiva con todos los bloques incorporados. A continuación se ofrecerá una comparativa entre los resultados obtenidos en la práctica con los distintos métodos de modulación para controlador en concreto. Después, se presentará una comparativa entre la respuesta real de los controladores. Finalmente, para obtener unas gráficas de velocidad real que definan de forma más amplia el comportamiento del motor con ambos métodos de control y modulación, se modificará de forma provisional el código para introducir una serie de *Set-Points* de tipo escalón consecutivos. Entre ellos habrá una distancia de 0,3 segundos e irán aumentando de uno en uno, partiendo de -6 radianes por segundo a 6 radianes por segundo.

6.1.1. Controlador PID

6.1.1.1. Modulación PWM. Comparativa con la simulación

Para el caso del controlador PID empleando modulación PWM se obtiene la gráfica de la figura 6.1.

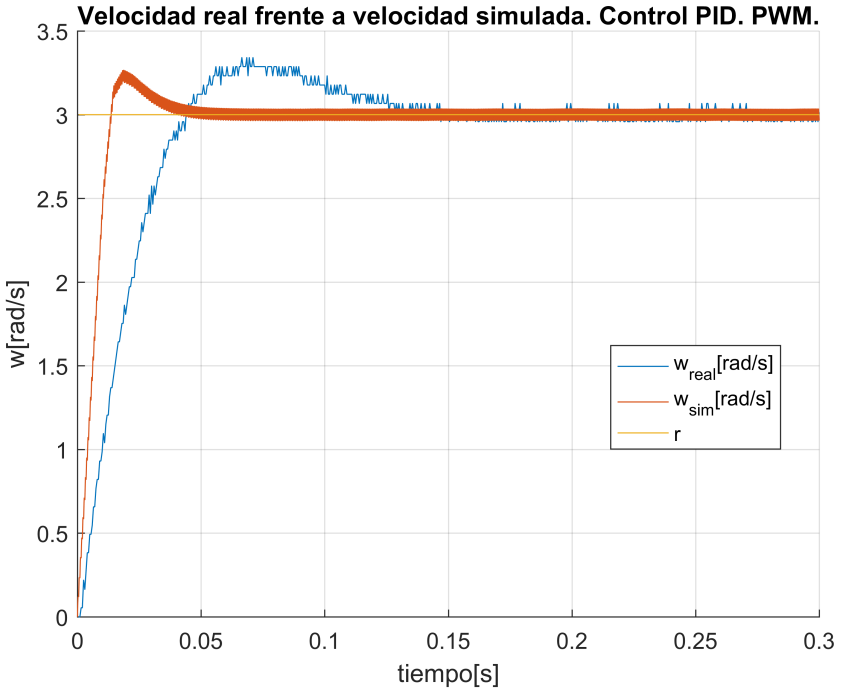


Figura 6.1: Velocidad real frente a la velocidad simulada mediante control PID y modulación PWM. *Set-Point* en escalón de 3 radianes por segundo.

En ella se observa que en la simulación se obtenía una respuesta más rápida que la que realmente puede dar el motor, sin embargo, si que era posible prever que bajo las condiciones establecidas y a una velocidad de 3 radianes por segundo, se iba a obtener una sobreoscilación.

6.1.1.2. Modulación PFM. Comparativa con la simulación

A continuación, se alterna el método de modulación. Es necesario cambiar la variable interruptor de modulación en el archivo de configuración de la simulación, y asegurar que el tiempo en alto de la señal PFM es de 50 microsegundos. Finalmente, se selecciona modulación PFM en el microcontrolador mediante el archivo "constantes.h". Tras compilar y ejecutar se obtiene la gráfica de la imagen 6.2.

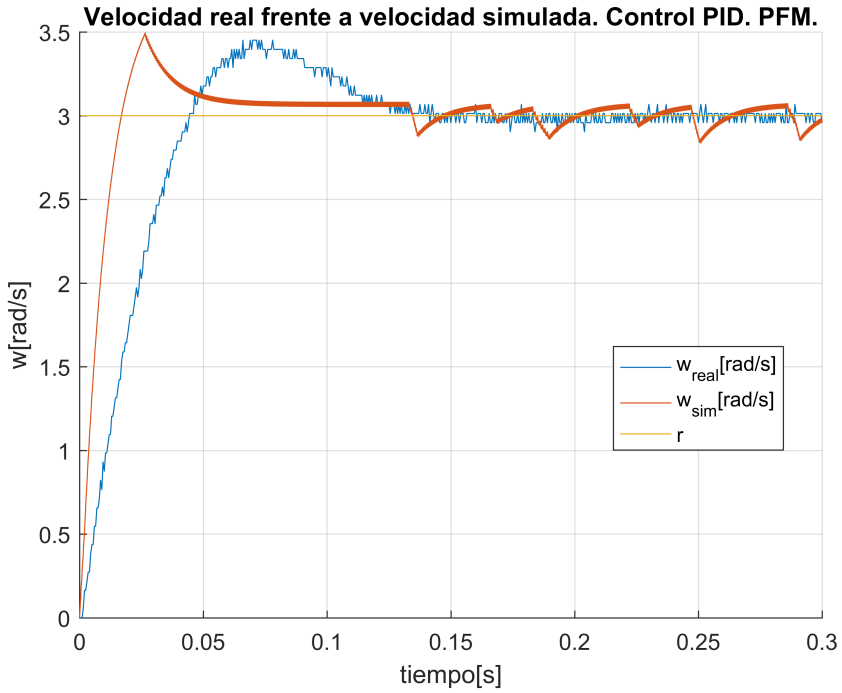


Figura 6.2: Velocidad real frente a la velocidad simulada empleando control PID y modulación PFM. Entrada en escalón igual a 3 radianes por segundo.

En ella se observa que, la falta de resolución en el bloque PFM en la simulación, acentúa las discontinuidades de la respuesta con respecto a las obtenidas empleando modulación PWM. A pesar de ello, si que era posible prever una sobreoscilación de la respuesta ante un *Set-Point* en escalón de 3 radianes por segundo. De nuevo, la respuesta real es más lenta que la simulada, la modulación PFM produce una sobreoscilación ligeramente mayor que en el caso de la modulación PWM.

Con el objetivo de disminuir las discontinuidades de la respuesta simulada, se eliminará el bloque frecuencímetro de la realimentación de la simulación, para realizar una prueba momentánea, cambiando el valor de la variable interruptor correspondiente en el archivo "Ajuste_Control_Recursivo.m". El resultado se muestra en la gráfica 6.3, en la cual se observa como en la simulación se pudo prever una sobreoscilación, aunque notablemente menor que la obtenida en la práctica.

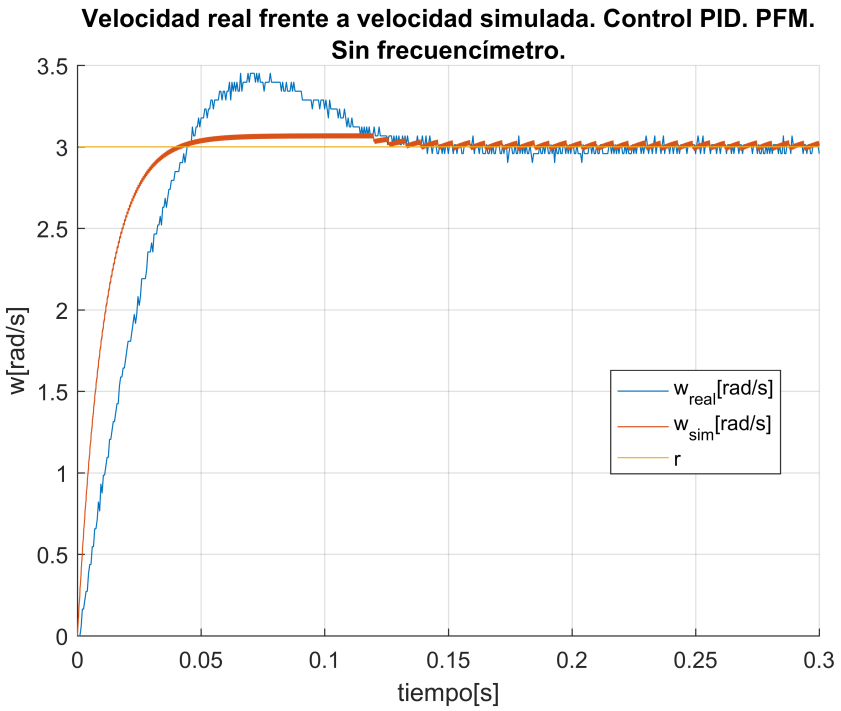


Figura 6.3: Velocidad real frente a la velocidad simulada empleando control PID y modulación PFM. *Set-Point*=3. Sin frecuencímetro en la realimentación en la simulación.

6.1.2. Controlador I-PD

6.1.2.1. Modulación PWM. Comparativa con la simulación

Seleccionando control I-PD y modulación PWM tanto en la simulación como en el archivo de configuración "constantes.h", se obtiene para un *Set-Point* de 3 radianes por segundo la respuesta mostrada en la figura 6.4. En ésta se refleja como las no linealidades propias del funcionamiento interno del motor introducen una sobreoscilación que no se encuentra presente en la simulación.

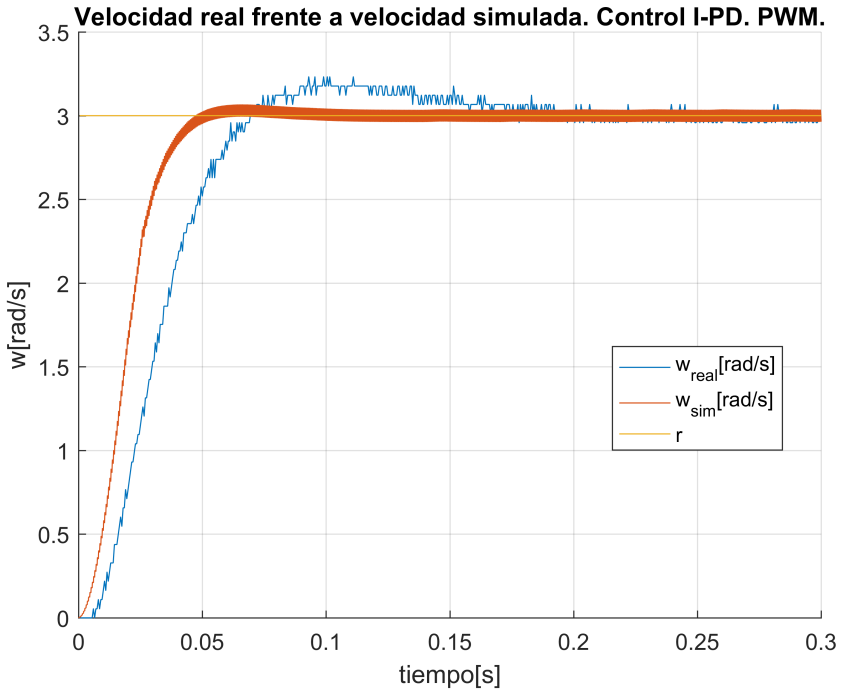
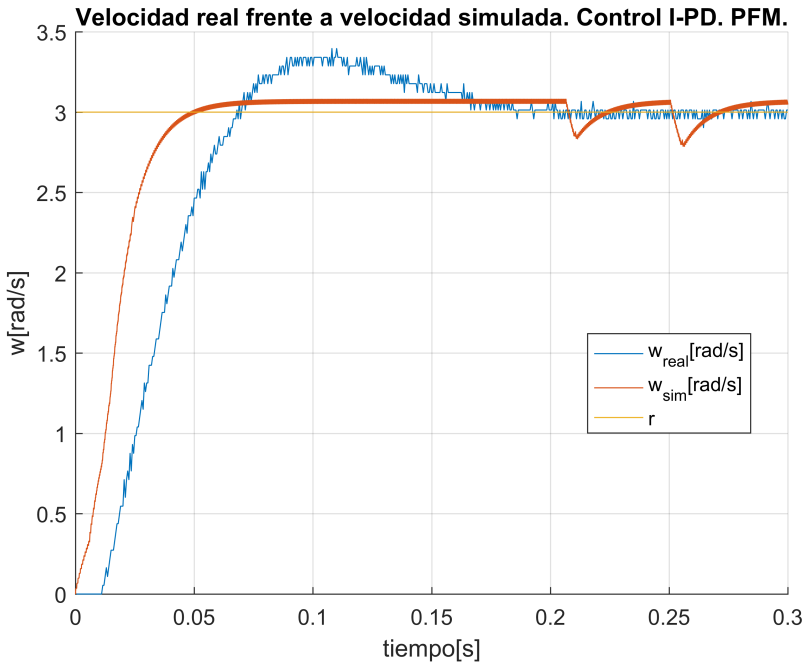


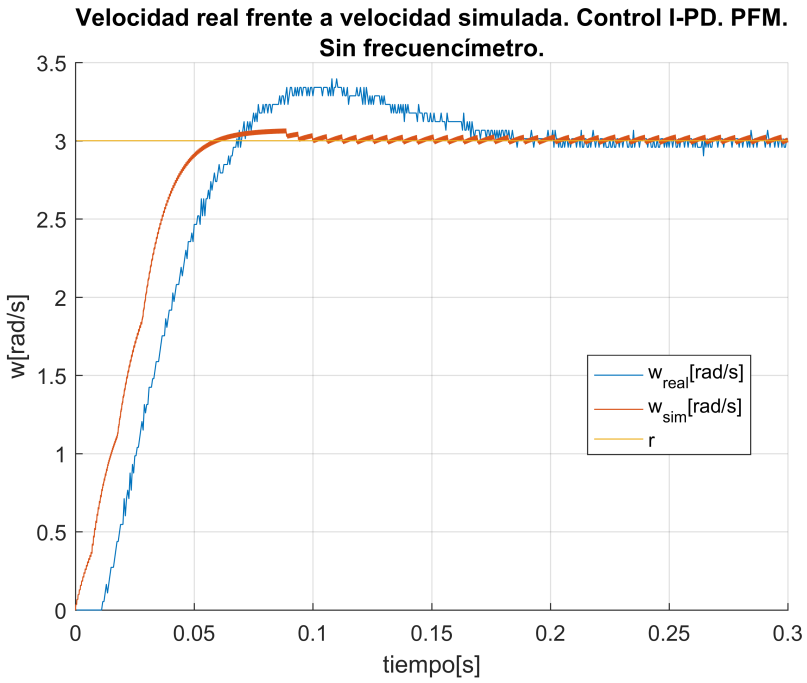
Figura 6.4: Velocidad real frente a la velocidad simulada. Control I-PD. Modulación PWM. *Set-Point*=3.

6.1.2.2. Modulación PFM. Comparativa con la simulación

Cambiando el método de modulación a PFM, tanto en el archivo de configuración de la simulación como en la práctica, se obtiene ante el mismo *Set-Point* la figura 6.5a. En la cual se refleja como la modulación PFM en la práctica presenta mayor sobreoscilación que la modulación PWM. La curva obtenida de la simulación presenta discontinuidades que dificultan su comparación con los resultados en la práctica. De manera análoga a lo realizado en el caso del controlador PID, se eliminará de forma momentánea el bloque del frecuencímetro de la realimentación de la simulación, con el objetivo de minimizar las discontinuidades. Haciendo el cambio correspondiente en la variable interruptor del fichero "Ajuste_Control_Recursivo", se obtiene la gráfica 6.5b. En ella se muestra como la respuesta del motor en la práctica presenta una mayor sobreoscilación y un mayor tiempo de subida que la obtenida en la simulación. En ésta, se observa como si que se pudo prever la sobreoscilación del motor empleando un *Set-Point* de 3 radianes por segundo.



(a) Frecuencímetro en la realimentación en la simulación.



(b) Sin frecuencímetro en la realimentación en la simulación.

Figura 6.5: Velocidad real frente a la velocidad simulada. Control I-PD. Modulación PFM. *Set-Point*=3.

6.1.3. Comparativa entre controladores

Para tener una comparativa de ambos métodos de control, se representan gráficamente las respuestas de uno frente al otro empleando el mismo método de modulación. En este caso, no se mostrará la curva de la velocidad obtenida en la simulación, por tanto, en primer lugar, se hace la variable "Simulink" del archivo de comunicación, "Comunicacion_Uart_COM.m", igual a cero. No será necesario cambiar nada más con respecto al procedimiento.

6.1.3.1. Modulación PWM

En el caso de emplear modulación PWM el resultado se muestra en la figura 6.6, en la que se observa una mayor velocidad de la respuesta en el control PID y ambas con sobreoscilación. También se muestra que el controlador I-PD presenta un retardo al inicio de su respuesta que no se encuentra presente en el caso del controlador PID.

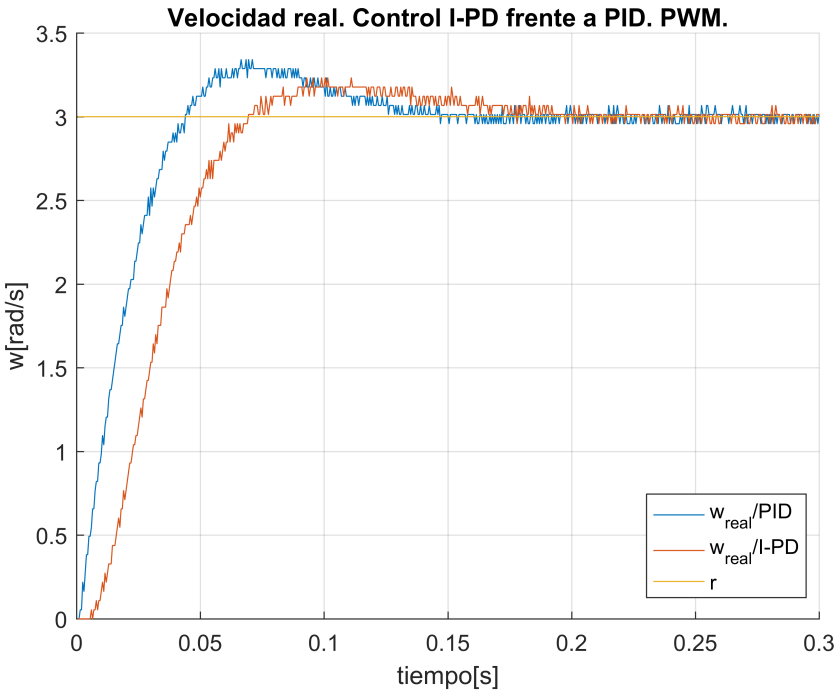


Figura 6.6: Velocidad real con modulación PWM. Control PID frente a control I-PD. *Set-Point*=3.

6.1.3.2. Modulación PFM

Empleando modulación PFM se obtiene la gráfica 6.7, en la que, de nuevo, el control PID es más rápido que el control I-PD. La sobreoscilación

de ambas es mayor que en el caso de la modulación por ancho de pulsos.

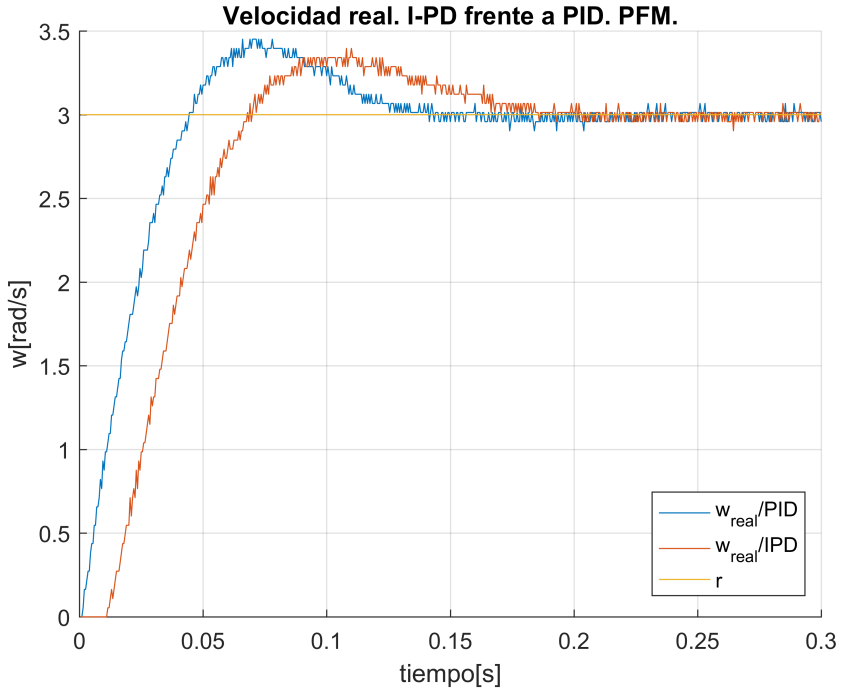


Figura 6.7: Velocidad real con modulación PFM. Control PID frente a control I-PD. *Set-Point*=3.

6.1.4. Prueba con una cadena de *Set-Points*

A continuación, como se mencionó anteriomente, se modificará de forma provisional el código para introducir una serie de *Set-Points* de tipo escalón consecutivos. Entre ellos habrá una distancia de 0,3 segundos e irán aumentando de uno en uno, partiendo de -6 radianes por segundo a 6 radianes por segundo.

6.1.4.1. Control PID

En el caso del control PID se obtiene para ambos métodos de modulación la respuesta mostrada en la gráfica 6.8. En ella se muestra que a velocidades intermedias, el control PID se aprovecha de la inercia del motor para dar respuestas con poca sobreoscilación, excepto a velocidades bajas como en $\pm$ 1 radianes por segundo, donde es la modulación PFM la que se ve más perjudicada; dando como resultado mayores sobreoscilaciones.

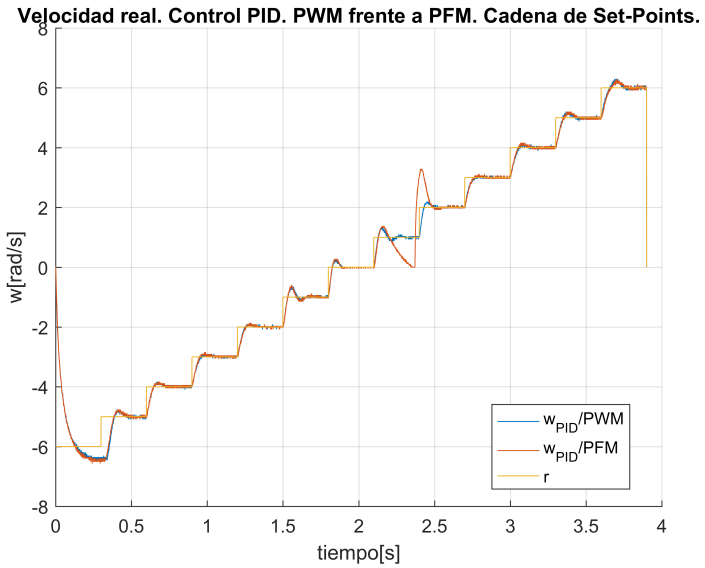


Figura 6.8: Velocidad real empleando control PID. Modulación PWM frente a PFM. Entrada en forma de cadena de *Set-Points*.

6.1.4.2. Control I-PD

En el caso del control I-PD para las mismas condiciones se obtiene la figura 6.9.

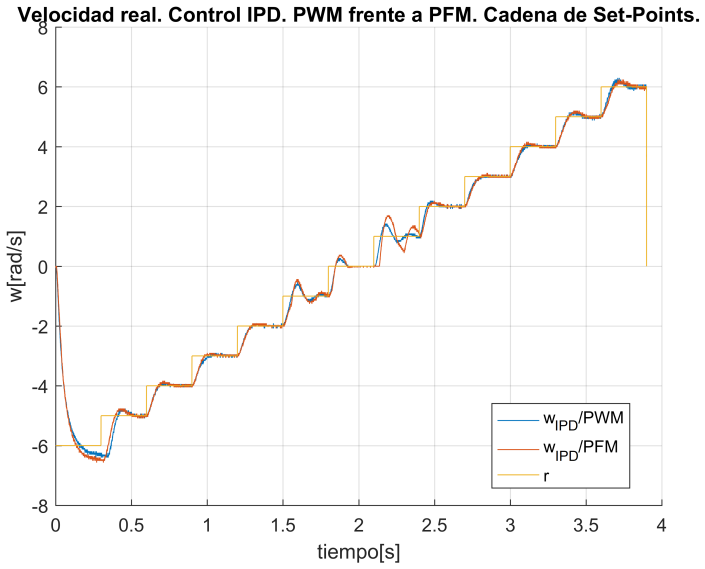


Figura 6.9: Velocidad real empleando control IPD. PWM frente a PFM. Entrada en forma de cadena de *Set-Points*.

En este caso, se observa que el control I-PD presenta mayores dificultades para frenar, esto se debe a que todo el esfuerzo de seguimiento del *Set-Point* lo realiza la acción integral. A velocidades intermedias no presenta tantos problemas y ocurre algo parecido al control PID en el que la inercia le lleva a dar respuestas con bajo sobreoscilamiento. Sin embargo, a velocidades bajas de ± 1 radianes por segundo, en el caso de la modulación PFM, presenta más problemas para hacer frente a la inercia, dando respuestas con mayor sobreoscilación. Sobretodo en el caso de la modulación PFM, ya que, a velocidades más bajas, el motor recibe pulsos de muy corta duración (50 microsegundos) más distanciados entre sí que a velocidades más altas.

6.2. Ajuste fino con curvas de velocidad reales

Con el objetivo de mejorar la respuesta se realizará un ajuste fino experimental, es decir, empleando las curvas de velocidad obtenidas en la práctica; a ambos controladores. Se comenzará con el controlador PID empleando modulación PWM. Se deduce de los valores de las acciones de control para dicho controlador, (4.82), que es necesario comenzar reduciendo la influencia de la acción integral, que puede estar provocando la sobreoscilación debido a la carga-descarga del error, es decir, la acción integral no compensará hasta que el error cambie de signo, cuando esto ocurra ya se habrá sobrepasado el *Set-Point*.Por tanto, se reduce su valor de diez en diez, siendo su valor inicial $Ki_{PID} = 65{,}73$, mientras se mantienen inalterados los valores de la acción derivada y proporcional. Haciendo pruebas con cada uno de los nuevos valores de la acción integral y empleando el mismo valor de *Set-Point* intermedio que en las pruebas interiores (3 radianes por segundo), se observa como se va reduciendo la sobreoscilación de la respuesta. Finalmente, se llega al valor de la acción integral $Ki_{PID} = 35{,}73$ en el cual la sobreoscilación ha sido completamente eliminada para el caso de la modulación PWM. A continuación, se realiza una prueba con este nuevo valor de la acción integral y el mismo *Set-Point*, con modulación PFM. Se observa, que para el caso de la modulación por frecuencia de pulsos, la respuesta tiene una ligera sobreoscilación, por lo que, es necesario llegar a un compromiso frente a una menor velocidad de respuesta empleando modulación PWM y la desaparición de la sobreoscilación empleando la modulación PFM. Finalmente, bajando el valor de la acción integral de uno en uno (siendo el último valor válido $Ki_{PID} = 35{,}73$), se establece como: $Ki_{PID} = 29{,}73$. Tras el ajuste fino, los valores del controlador PID se establecen como:

$$\begin{aligned} K_p &= 0{,}6573 \\ K_i &= 29{,}73 \\ K_d &= 4{,}382 \cdot 10^{-6} \\ t_{fd} &= (K_d/K_p)/10 = 6{,}666 \cdot 10^{-7} \end{aligned} \tag{6.10}$$

Continuando con el ajuste para el controlador I-PD, se procede de forma análoga. Partiendo de la modulación por ancho de pulsos, es necesario reducir en primer lugar la influencia de la acción integral. Sin embargo, una vez se ha eliminado la sobreoscilación disminuyendo la acción integral, el tiempo de subida de la respuesta ha aumentado, y al comprobar los resultados empleando modulación PFM, existe una sobreoscilación. Se decide entonces emplear otra estrategia que consiste en reducir la influencia de la acción derivada dividiéndola por diez y aumentar el valor de la acción proporcional. Finalmente se reduce ligeramente el valor de la acción integral para compensar, obteniendo:

$$\begin{aligned} Ki_{I-PD} &= 38{,}699 \\ Kp_{I-PD} &= 0{,}8296 \\ Kd_{I-PD} &= 8{,}5838 \cdot 10^{-}6 \\ t_{fd/I-PD} &= 1{,}0346 \cdot 10^{-}6 \end{aligned} \tag{6.11}$$

A continuación se mostrarán gráficas de resultados siguiendo el mismo orden que en la sección dedicada a las pruebas iniciales. En la simulación se introducirán los valores obtenidos en la práctica, que a pesar de que en la simulación no se tienen en cuenta las no linealidades propias de los componentes internos del motor, es de interés observar la respuesta.

6.2.1. Controlador PID

6.2.1.1. Modulación PWM. Comparativa con la simulación

La respuesta del motor ante un *Set-Point* de 3 radianes por segundo empleando los nuevos valores de ganancia para el controlador PID, frente a la obtenida en la simulación, con modulación por ancho de pulsos es la mostrada en la figura 6.10. En la cual se observa como se ha eliminado la sobreoscilación en la curva de la velocidad del motor. La respuesta simulada presenta una discontinuidad al inicio que puede ser disminuida eliminado el conjunto encoder-frecuencímetro de la realimentación en la simulación. A pesar de ello, ambas respuestas son similares.

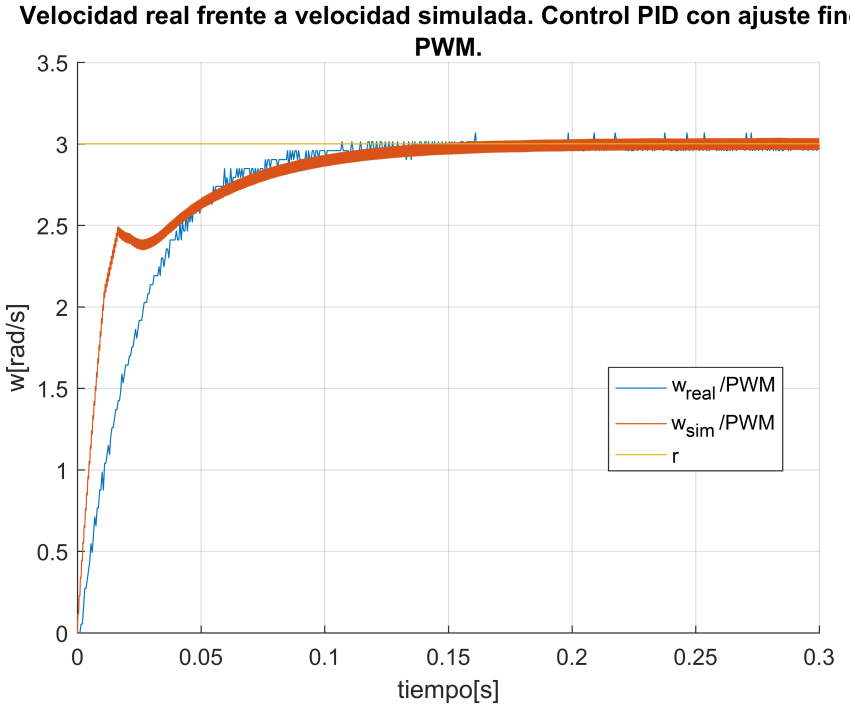
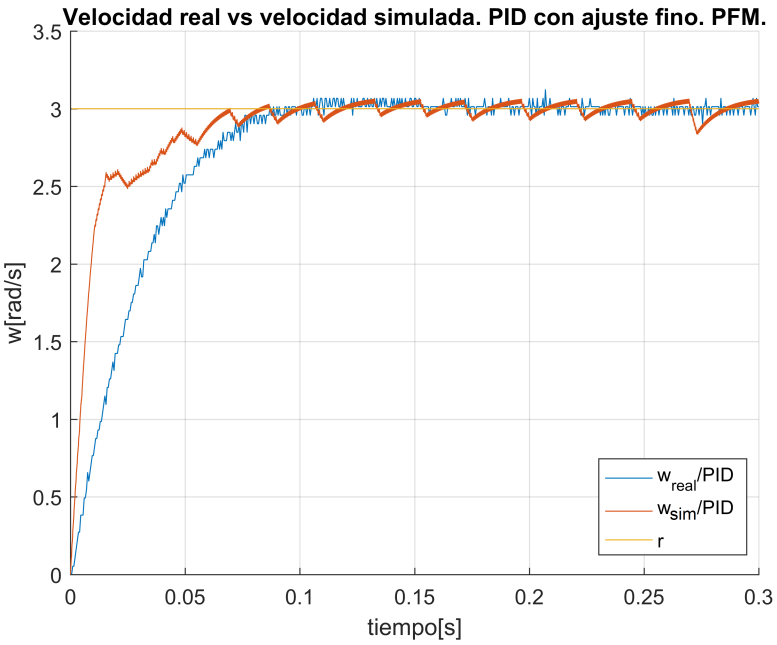


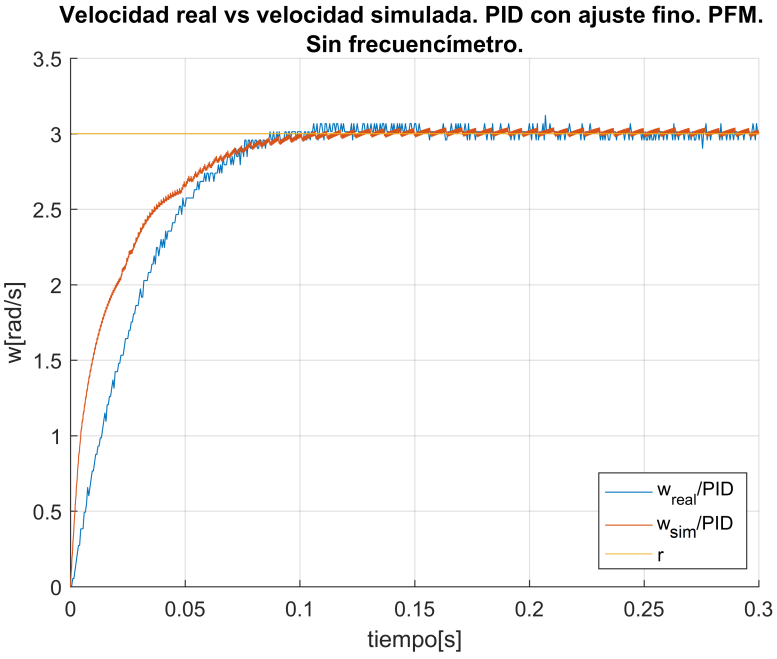
Figura 6.10: Velocidad real frente a la velocidad simulada. Control PID ajustado en la práctica. Modulación PWM. *Set-Point* en escalón igual a 3 radianes por segundo.

6.2.1.2. Modulación PFM. Comparativa con la simulación.

Tras realizar los cambios pertinentes para emplear modulación PFM, se obtiene la gráfica presentada en la imagen 6.11a. En la que se muestra que la sobreoscilación, en la curva de velocidad obtenida del motor, ha sido en gran parte eliminada. Por otro lado, en la respuesta obtenida en la simulación, a pesar de las discontinuidades, tampoco presenta sobreoscilación como tal, y es más rápida que la curva real obtenida. Para disminuir las discontinuidades, de forma momentánea se eliminará el conjunto encoder-frecuencíemtro de la realimentación. Tras hacer los cambios correspondientes en las variables interruptor del archivo de configuración de la simulación se obtiene la figura 6.11b. En la cual se refleja que ambas respuestas son muy similares, siendo la curva obtenida en la simulación más rápida.



(a) Con frecuencímetro en la realimentación de la simulación.



(b) Sin frecuencímetro en la realimentación de la simulación.

Figura 6.11: Velocidad real frente a la velocidad simulada. Control PID ajustado en la práctica. Modulación PFM. *Set-Point* en escalón igual a 3 radianes por segundo.

6.2.2. Comparativa entre los controladores ajustados

De la misma forma que en lo realizado en la sección dedicada a las pruebas iniciales, se presentarán gráficas comparativas entre ambos controladores, en la práctica, empleando las mismas formas de modulación.

6.2.2.1. Modulación PWM

Empleando modulación por ancho de pulsos se obtiene la figura 6.12. Se observa como, ante la misma entrada en escalón y la misma modulación, el controlador PID es más rápido que el I-PD, presentando un retardo al inicio de la respuesta.

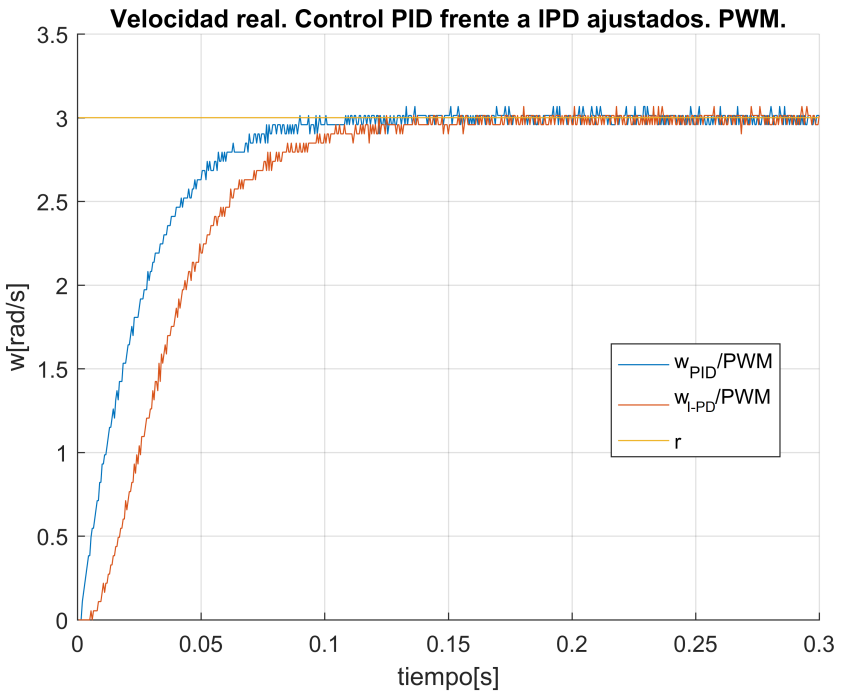


Figura 6.12: Velocidad real. Controladores PID e I-PD ajustados en la práctica. Modulación PWM. *Set-Point* en escalón igual a 3 radianes por segundo.

6.2.2.2. Modulación PFM

En el caso de la modulación por frecuencia de pulsos se obtienen resultados similares, con la diferencia de que las respuestas son menos amortiguadas que en el caso de la modulación por ancho de pulsos. De nuevo, el controlador PID es más rápido que el controlador I-PD, presentando un retardo al inicio de la respuesta. La comparativa se presenta en la figura 6.13.

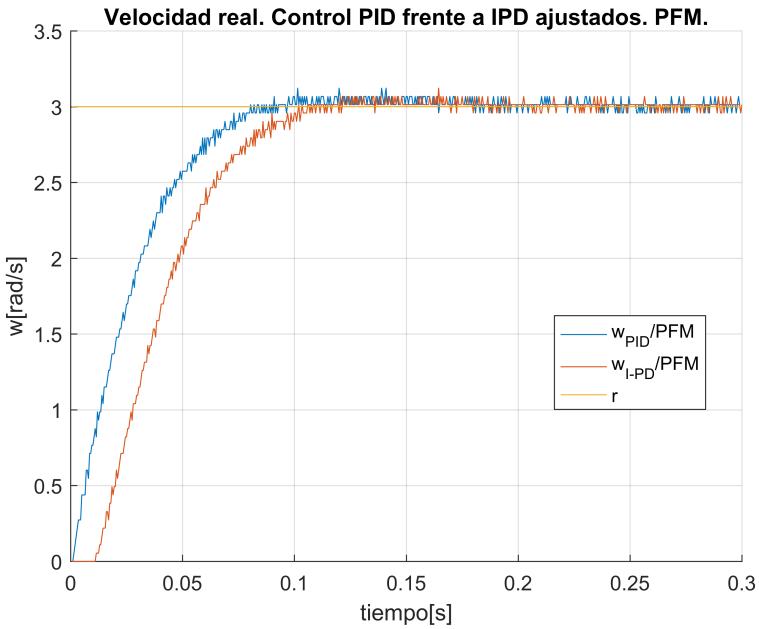


Figura 6.13: Velocidad real con controladores PID e I-PD ajustados en la práctica, empleando modulación PFM. *Set-Point* en escalón igual a 3 radianes por segundo.

6.2.3. Prueba con una cadena de *Set-Points*

Siguiendo lo realizado en la sección anterior, se modificará de forma momentánea el código del microcontrolador para introducir una cadena de *Set-Points* en escalón para definir de forma más amplia el funcionamiento de los controladores con el ajuste fino realizado en la práctica.

6.2.3.1. Control PID

Empleando el controlador PID se obtiene la figura 6.14. En la que se observa una mejoría general en ambas formas de modulación con respecto a la respuesta obtenida antes del ajuste, 6.8, especialmente en la zona con valores de *Set-Point* bajos, como 1 y -1 radianes por segundo, siendo ligeramente superiores en el caso de la modulación PFM. Con un nuevo ajuste fino, se podrían eliminar estas sobreoscilaciones.

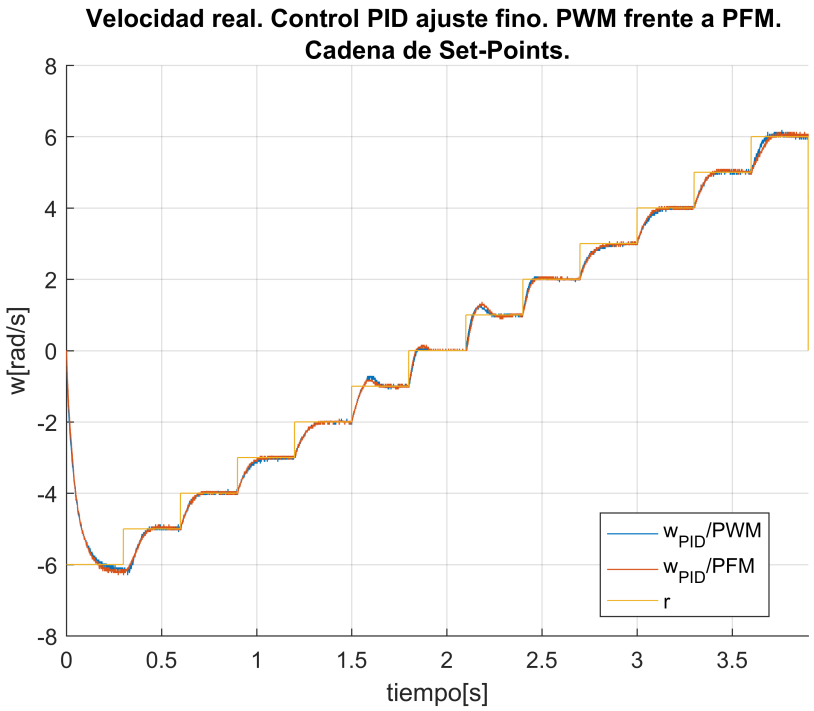


Figura 6.14: Velocidad real. Controlador PID ajustado en la práctica. Modulación PFM frente a modulación PWM. Entrada igual a una cadena de *Set-Points* en escalón.

6.2.3.2. Control I-PD

Finalmente, empleando el controlador I-PD se obtiene ante una cadena de *Set-Points* en escalón, la figura 6.15. En la cual se observa como se ha mejorado la respuesta especialmente en la transición entre un 0 y 1 radianes por segundo con respecto a la respuesta obtenida antes del ajuste (figura 6.9). Sin embargo, la respuesta empleando modulación PFM sigue mostrando mayores dificultades para frenar en la zona de transición entre -1 y 0 radianes por segundo, generando una sobreoscilación. Esto se debe a la propia acción de la modulación PFM, como se comentó con anterioridad. Ésta, a velocidades bajas, proporciona al motor pulsos de un tiempo en alto muy pequeño muy distanciados entre sí, además, en el caso del control I-PD, todo el esfuerzo de seguimiento del *Set-Point* recae en la acción integral.

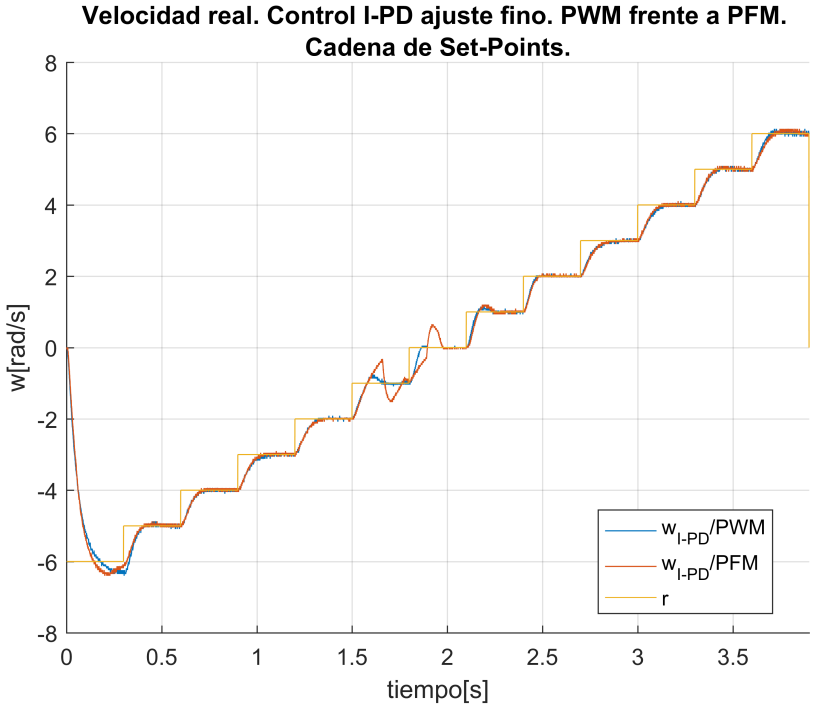


Figura 6.15: Velocidad real. Controlador I-PD ajustado en la práctica. Modulación PFM frente a modulación PWM. Entrada igual a una cadena de *Set-Points* en escalón.

Por lo que viendo los resultados obtenidos ante un mayor rango de *Set-Points* de entrada, se concluye que, con el ajuste realizado, el mejor método de control para este motor se consigue empleando un controlador PID, con escasas diferencias entre el método de modulación empleado. Los valores de las acciones de control finalmente obtenidos son:

$$\begin{aligned} K_p &= 0{,}6573 \\ K_i &= 29{,}73 \\ K_d &= 4{,}382 \cdot 10^{-6} \\ t_{fd} &= (K_d/K_p)/10 = 6{,}666 \cdot 10^{-7} \end{aligned} \tag{6.12}$$

Capítulo 7

Anexo: Códigos de los bloques en *Simulink*

7.1. Bloque PWM

Código 7.1: Código del bloque PWM

```
function PWM = fcn(u,tam)
%Declarar variables.
    %Indice para recorrer el vector.
    %Debe conservar su valor.
    persistent j;
    %Vector contenedor. Debe conservar su valor.
    persistent contenedor_PWM;
    %Necesario establecer valor inicial.
    if isempty(j)
        j = 1;
    end
    %Necesario establecer valor inicial.
    if isempty(contenedor_PWM)
        contenedor_PWM=zeros(1,tam);
    end
%Calculo tiempo en alto
    D=u/6;                  %Ciclo de trabajo.
    D=abs(D);
    th=round(tam*D);        %Numero de unos.

    if(j==1)
%Generar periodo PWM
        for i=1:1:tam
            if(i<=th)
                contenedor_PWM(1,i)=sign(u)*1;
            else
                contenedor_PWM(1,i)=0;
            end
        end
    end
    PWM=6*contenedor_PWM(1,j);
    if(j<tam)
        j=j+1;
    else
        j=1;
    end
end
```

7.2. Bloque PFM

Código 7.2: Código del bloque PFM

```
function PFM = fcn(u, tam_PFM, t_on_PFM)
    % Declarar variables.
    persistent j;    % Indice para recorrer el vector. Debe conservar su valor.
    persistent contenedor_PFM;   % Vector contenedor. Debe conservar su valor.
    contador_ceros=0; % Variable contador de ceros introducidos.
        if isempty(j) % Necesario establecer valor inicial del indice.
            j = 1;
        end

        if isempty(contenedor_PFM) % Necesario establecer valores iniciales.
            contenedor_PFM=ones(1,tam_PFM);
        end

        if u~=0 % Salvar division por cero.
    % Primera modulacion.
            if abs(u)<4 || abs(u)==6
                T_PFM=(6*t_on_PFM)/abs(u); % Calculo del periodo PFM.
                N=T_PFM/t_on_PFM; % Numero de divisiones de ancho t_on_PFM.
                N=round(N);
                t_off=(N-1); % Numero de ceros.
                    if (j==1) % Corregir la primera entrada si u<0.
                        if u<0
                          contenedor_PFM(1,1)=sign(u)*1;
                        end
                        for i=2:1:tam_PFM % Introducir ceros.
                            if contador_ceros<t_off
                                contenedor_PFM(1,i)=0;
                                contador_ceros=contador_ceros+1;
                            else
                                contenedor_PFM(1,i)=sign(u)*1;
                                contador_ceros=0;
                            end
                        end
                    end
            else
      % Segunda modulacion
        % Hacer que el valor de entrada se encuentre en el rango 0<u<4.
                mod2=abs(u)-3;
                T_PFM=(6*t_on_PFM)/mod2;
                N=T_PFM/t_on_PFM; % Calculo del numero de divisiones.
                N=round(N);
                t_off=(N-1); % Calculo del numero de ceros.
                    if (j==1)
                        if u<0 % Corregir la mitad del vector si u<0.
                            for i=1:1:(tam_PFM/2)
                                contenedor_PFM(1,i)=sign(u)*1;
                            end
                        end
                        % Rellenar desde la posicion media + 1.
                        for i=(tam_PFM/2)+1:1:tam_PFM
                            if contador_ceros<t_off      % Introducir ceros.
                                contenedor_PFM(1,i)=0;
                                contador_ceros=contador_ceros+1;
                            else
                                contenedor_PFM(1,i)=sign(u)*1;
                                contador_ceros=0;
                            end
                        end
                    end
            end
    % Enviar datos cada t_on_PFM.
            PFM=6*contenedor_PFM(1,j);
                if j<tam_PFM
                    j=j+1;
                else
                    j=1;
                    contenedor_PFM=ones(1,tam_PFM);
                end
        else
            PFM=0; % Si u=0, PFM =0.
        end
end
```

7.3. Bloques de los canales del encoder

7.3.1. Bloque inicial del canal A

Código 7.3: Código inicial del bloque encoder. Canal A

```
function A = fcn(y)
%Angulo de giro en el que se genera un pulso = 0,7
if (mod(y,0.7)>=0) && (mod(y,0.7)<=0.35)
    A=1;
elseif(mod(y,0.7)>0.35) && (mod(y,0.7)<=0.7)
    A=0;
else
    A=2; %Error
end
```

7.3.2. Bloque final del canal A

Código 7.4: Código final del bloque encoder. Canal A

```
function A = fcn(y)
%Angulo de giro en el que se genera un pulso = 0,7
%Desfase entre canales A y B = 90 grados
if (mod(abs(y),0.7)>=0) && (mod(abs(y),0.7)<=0.35)
    A=1;
elseif(mod(abs(y),0.7)>0.35) && (mod(abs(y),0.7)<=0.7)
    A=0;
else
    A=2; %Error
end
```

7.3.3. Bloque inicial del canal B

Código 7.5: Código inicial del bloque encoder. Canal B

```
    function B = fcn(y)
%Angulo de giro en el que se genera un pulso = 0,7
%Desfase entre canales A y B = 90 grados
y=y-(0.7/4);
if (mod(y,0.7)>=0) && (mod(y,0.7)<=0.35)
    B=1;
elseif (mod(y,0.7)>0.35) && (mod(y,0.7)<=0.7)
    B=0;
else
    B=2; %Error
end
```

7.4. Bloque final del canal B

Código 7.6: Código final del bloque encoder. Canal B

```
function B = fcn(y)
%Angulo de giro en el que se genera un pulso = 0,7
%Desfase entre canales A y B = 90 grados
y=y-(0.7/4);
if (mod(abs(y),0.7)>=0) && (mod(abs(y),0.7)<=0.35)
    B=1;
elseif(mod(abs(y),0.7)>0.35)&&(mod(abs(y),0.7)<=0.7)
    B=0;
else
    B=2;
end
```

7.5. Bloque encargado de determinar el sentido de giro del motor

Código 7.7: Código bloque detector del sentido de giro

```
function Sentido = fcn(A,B)

persistent buffer_B
persistent buffer_sentido

if isempty(buffer_B)
    buffer_B=zeros(1,2);
end
if isempty(buffer_sentido)
    buffer_sentido=0;
end
buffer_B(1,2)=B;
%buffer_in(1,1) es el valor anterior
%buffer_in(1,2) es el valor actual
if buffer_B(1,1)==0 && buffer_B(1,2)==1
   if A==1
       buffer_sentido=1; %Un 1 es sentido horario.
   else
       buffer_sentido=-1; %Un -1 es sentido horario.
   end
end
buffer_B(1,1)=buffer_B(1,2);
Sentido=buffer_sentido;
```

7.6. Bloque frecuencímetro

Código 7.8: Código del bloque frecuencímetro.

```
function y = fcn(Canal, T_frec, Temporizador, Sentido)

persistent buffer_entrada
persistent w
persistent flanco_detectado
persistent N
persistent contador_flancos

if isempty(buffer_entrada)
    buffer_entrada=zeros(1,2);
end
if isempty(w)
    w=0;
end
if isempty(flanco_detectado)
    flanco_detectado=false;
end
if isempty(N)
    N=0;
end
if isempty(contador_flancos)
    contador_flancos=0;
end
buffer_entrada(1,2)=Canal;
if buffer_entrada(1,1)==0 && buffer_entrada(1,2)==1
% buffer_entrada(1,1) es el valor anterior
% buffer_in(1,2) es el actual
  flanco_detectado=true;
  contador_flancos=contador_flancos+1;
    if contador_flancos==2
        T_x=N*T_frec;
        if T_x>0
            w=(0.7/T_x)*(pi/180);
        else
            w=0;
        end
        N=1; %Se reinicia la cuenta.
        contador_flancos=1;
% Se hace 1 porque se encuentra en un flanco de nuevo.
    end
end
if flanco_detectado==true && Temporizador>0
% En el caso de que el encoder empiece en alto,
% empieza a contar cuando se haya producido el escalon
%if flanco==true
    N=N+1;
end
buffer_entrada(1,1)=buffer_entrada(1,2);
y=w*Sentido;
```

7.7. Bloques de las acciones de control en recursiva

7.7.1. Acción integral

Código 7.9: Código de la acción integral en recursiva

```
function u_i = fcn(e, Ki_IPD, T_m)
persistent contenedor_u;
persistent contenedor_e;
if isempty(contenedor_u)
    contenedor_u=zeros(1,1);
end
if isempty(contenedor_e)
    contenedor_e=zeros(1,1);
end
%Calcula u
u_i=(Ki_IPD*T_m*0.5)*(e+contenedor_e(1,1))+
contenedor_u(1,1);
contenedor_e(1,1)=e;
contenedor_u(1,1)=u_i;
end
```

7.7.2. Acción derivada

Código 7.10: Código de la acción derivada en recursiva

```
function u_d = fcn(y_fd, Kd_IPD, T_m)
    persistent contenedor_u;
    persistent contenedor_y_fd;
        if isempty(contenedor_u)
            contenedor_u=zeros(1,1);
        end
        if isempty(contenedor_y_fd)
            contenedor_y_fd=zeros(1,1);
        end
    %Calcula u
    u_d=(Kd_IPD*(2/T_m))*(y_fd-contenedor_y_fd(1,1))-
    contenedor_u(1,1);
    contenedor_y_fd(1,1)=y_fd;
    contenedor_u(1,1)=u_d;
end
```

7.7.3. Filtro a la acción derivada

Código 7.11: Código del filtro a la acción derivada en recursiva

```
function y_fd = fcn(y,T_m, t_fd_IPD)
    persistent contenedor_y_fd;
    persistent contenedor_y;
        if isempty(contenedor_y_fd)
            contenedor_y_fd=zeros(1,1);
        end
        if isempty(contenedor_y)
            contenedor_y=zeros(1,1);
        end
%Calcula u
y_fd=(T_m/(2*t_fd_IPD+T_m))*(y+contenedor_y(1,1))-
contenedor_y_fd(1,1)*((T_m-2*t_fd_IPD)/(2*t_fd_IPD+T_m));
contenedor_y(1,1)=y;
contenedor_y_fd(1,1)=y_fd;
end
```

Bibliografía

[1] Fraile Mora, Jesús: *Máquinas eléctricas*, capítulo 6, páginas 483–488. McGraw-Hill, 5ª edición, 2003.

[2] Guerrero Rodríguez, José María: *Instrumentación básica del laboratorio electrónico*, capítulo 6, páginas 177–180. Departamento de Ingeniería de Sistemas y Automática, Tecnología Electronica y Electrónica. Universidad de Cádiz, 2000.

[3] Hart, D.W.: *Electrónica de potencia*, páginas 200–203. Pearson Educación, 2001.

[4] Incze, Ioan, Csaba Szabo y Mária Imecs: *Incremental Encoder in Electrical Drives: Modeling and Simulation*, volumen 313, páginas 287–300. Technical University of Cluj-Napoca, Octubre 2010.

[5] Kuo, Benjamin C: *Sistemas de control automático*. Pearson Educación, 7ª edición, 1996.

[6] NXP: *LPC408x/407x User manual*, 2004 (accedido Junio,2020). https://www.nxp.com/docs/en/user-guide/UM10562.pdf.

[7] Ogata, Katsuhiko: *Sistemas de control en tiempo discreto*. Pearson Educación, 2ª edición, 1996.

[8] Ogata, Katsuhiko: *Ingeniería de control moderna*. Prentice-Hall, 5ª edición, 2010.