UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DE UN SERVOMOTOR MEDIANTE UNA TARJETA STM32 NUCLEO 32

1. **Memoria**
2. **Planos**
3. **Pliego de condiciones**
4. **Presupuesto**

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR: **Jaime Ferre Martínez**

TUTORIZADO POR: **Ranko Zotovic Stanisic**

CURSO ACADÉMICO: 2020/2021

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DE UN SERVOMOTOR MEDIANTE UNA TARJETA STM32 NUCLEO 32

1. Memoria

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR: **Jaime Ferre Martínez**

TUTORIZADO POR: **Ranko Zotovic Stanisic**

CURSO ACADÉMICO: 2020/2021

Contenido

INDICE DE FIGURAS

INDICE DE TABLAS

1. Objeto

Este trabajo de final de grado tiene como objetivo el control de un servomotor. Este proyecto consiste en controlar, mediante una tarjeta STM32 nucleo-32, un servomotor y diseñar su correspondiente circuito de potencia. La finalidad de este proyecto está dirigida a llevar a cabo la construcción de un robot industrial profesional completamente funcional. Pero para ello, antes se debe abordar el control de cada una de sus partes primarias, es decir, los servomotores.

El control del servomotor diseñado deberá de poseer las características necesarias para asemejarse a un control profesional. Para ello, se necesitará que el control de posición sea preciso, además, deberá de poseer las rutinas básicas que cualquier control de servomotor posee en el mercado, como la rutina de homing, paradas controladas y control y medición de la alimentación de la etapa de potencia. Dado que este trabajo de final de grado se ha propuesto durante la pandemia del coronavirus, se ha llevado a cabo desde la vivienda del estudiante, por lo que existen limitaciones a la hora del diseño.

2. Antecedentes

A lo largo de los últimos años, se han desarrollado multitud de métodos diferentes de automatización industrial. Todos ellos poseen en común el mismo elemento, los motores eléctricos. Existen una gran variedad de tipos y modelos, de los cuales se pueden distinguir:

- Motores de corriente continua
    - Imanes permanentes
    - Motor en serie
    - Motor shunt
    - Motor compuesto
- Motor de corriente alterna
    - Motor síncrono
    - Motor asíncrono
    - Motor trifásico
    - Sin escobillas
- Motor paso a paso
- Motor universal

Dado que la intención principal es profundizar en los motores más adecuados para la robótica, los más apropiados para este campo son los motores de corriente continua con imanes permanentes y los motores de corriente alterna sin escobillas. La tendencia actual es utilizar los motores de corriente alterna sin escobillas ya que no requieren mantenimiento, pueden funcionar en cualquier tipo de entornos y tienen mejor movimiento. Aunque tiene el inconveniente que son más complejos que el motor de corriente continua con imanes permanentes.

Cada uno de estos motores eléctricos, poseen el mismo modelo matemático que define sus comportamientos. Este modelo está definido por las siguientes ecuaciones:

$$U = E + I * R + L\frac{di}{dt}$$

$$M = K_m * I$$

$$E = K_N * n$$

Donde:

- U: Voltaje eléctrico aplicado al motor.
- E: Fuerza electromotriz del motor.
- R: Resistencia de la bobina del motor.
- I: Corriente del motor.
- M: Par generado por el motor.
- K_m: Constante de par del motor.
- K_n: Constante de velocidad del motor.
- N: Velocidad del motor en rpm.

Además, hay que tener en cuenta el modelo matemático de la carga del motor.

$$M = J\ddot{q} + b\dot{q}$$

Donde:

- J: Momento de inercia total en la articulación.
- q: Coordenada generalizada.
- b: Fricción viscosa.

Una vez obtenidos ambos modelos, se pueden combinar para realizar el modelo del motor eléctrico.

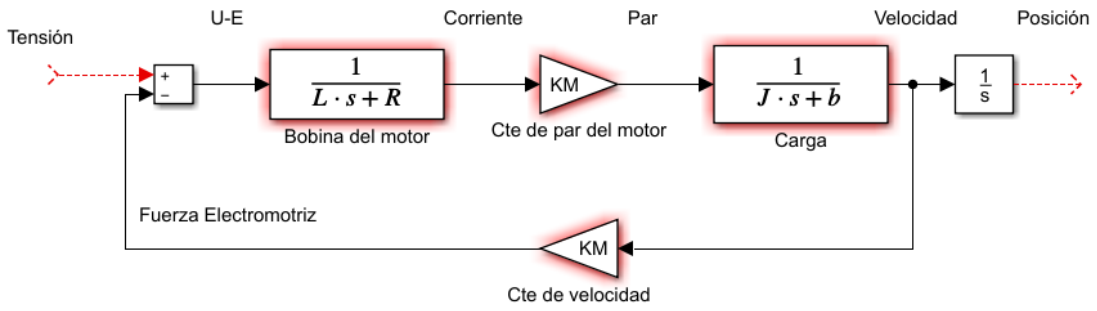


Figura 1. Modelo de funcionamiento del motor eléctrico

Fuente: Propia

3. Estudio de necesidades, limitaciones y condicionantes.

3.1 Requerimientos del consumidor

El trabajo explicado a lo largo del documento es solamente un paso para conseguir el objetivo de construir y controlar un brazo robot compuesto por hasta 7 servomotores. Por esta razón, el control de este servomotor deberá de cumplir varios requerimientos. El primero será la posibilidad de comunicación entre el microcontrolador empleado y el computador u otro microcontrolador funcionando en modo maestro. El segundo, es realizar una rutina de homing encargada de situar el motor en la posición inicial. El tercero será introducir dos finales de carrera encargados de alertar si el motor ha llegado al final de su recorrido completo. El último requerimiento será controlar la alimentación de la etapa de potencia al igual que detectar en qué estado se encuentra.

3.2 Limitaciones del diseñador

Dado que el trabajo se ha realizado en la vivienda del estudiante, existen varias limitaciones que definen en gran medida tanto el resultado como el desarrollo del proyecto. Para empezar, dado que no se iba a realizar el diseño en una placa de circuito impreso, sino en una protoboard, todos los componentes empleados han sido de tipo agujero pasante. Además, dado que el estudiante ha tenido que financiar personalmente el proyecto, se han intentado elegir los componentes más económicos. Otra limitación clave ha sido la falta de un osciloscopio para medir algunas señales características.

3.3 Requerimientos generales

Los requerimientos generales del proyecto hacen referencia a las características que debían de poseer los componentes principales. Estas características son las siguientes:

- El motor deberá de llevar el sensor de posición incorporado de fábrica.
- Se deberá de emplear una señal de onda cuadrada de frecuencia de 1 milisegundo o 100 microsegundos para controlar el circuito de potencia del motor.

4. Descripción de alternativas, criterios de selección y solución adoptada.

Durante la etapa de diseño, varias opciones para cada componente deben ser consideradas ya que no existe solamente una solución para un proyecto. Para hacerlo, se ha realizado una búsqueda exhaustiva de información, analizando los diferentes modelos del mercado.

A continuación, se van a enumerar los componentes, se explicará su función dentro del proyecto y se expondrán las diferentes alternativas al igual que los criterios de selección correspondientes para cada elemento. Sin embargo, el factor económico se tomará en consideración como criterio general. Además, dado que el alumno ha tenido que comprar los materiales por su cuenta, los componentes que ya posean, tendrán más peso en la elección.

La siguiente figura muestra la estructura de desglose de trabajo (EDT) de los componentes necesarios.

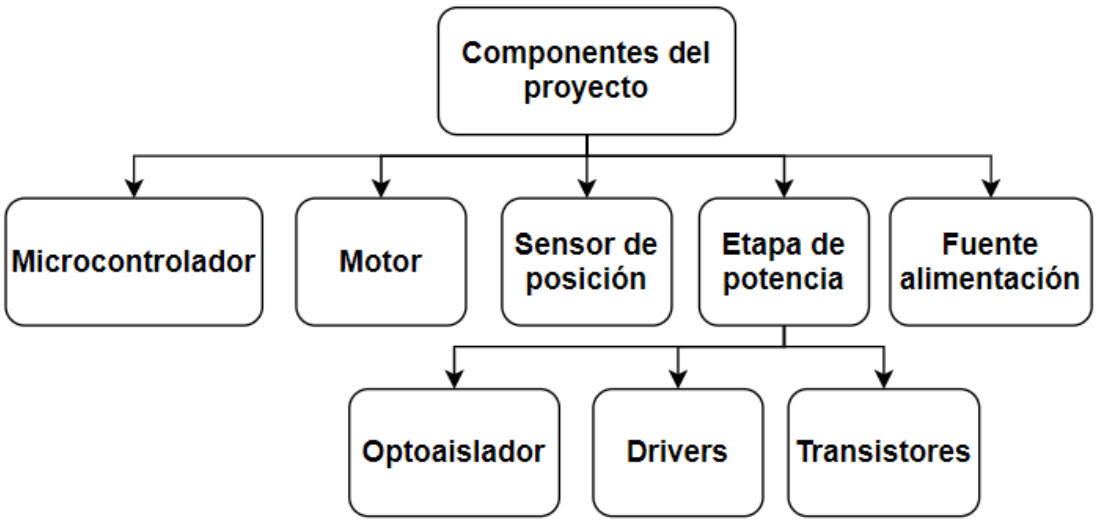


Figura 2. Estructura de desglose de trabajo de los componentes.

Fuente: Propia.

4.1 Microcontrolador

El microcontrolador es la unidad principal de procesamiento de información y el encargado de controlar todos los componentes y realizar los cálculos correspondientes. Para este proyecto se han propuesto varias tarjetas de desarrollo para conseguir el objetivo. Las tarjetas de desarrollo son placas con el microcontrolador deseado, diversos periféricos y con varias funcionalidades para realizar la depuración del código. A continuación, se muestran dos posibles soluciones:

- Tarjeta de desarrollo STM32F4Discovery
- Tarjeta de desarrollo STM32F303K8 Nucleo-32

Para poder elegir que microcontrolador que sea más recomendable para la aplicación descrita anteriormente, se necesita que posea las siguientes características:

- Poder realizar bucles de control de 100 uS controlado por interrupción de timer.
- Poseer más de 10 pines disponibles.
- Poseer al menos una salida PWM, dos entradas para encoder, tres entradas digitales de interrupción, un timer y dos salidas digitales.

Los términos “encoder” (Apartado 4.3), y “timer” (Aparatado 4.1.1) serán explicadas más adelante.

Teniendo en consideración las dos tarjetas propuestas, se ha optado por el modelo STM32F303K8T6 Nucleo-32. Aunque ambas tarjetas cumplían los requisitos, la nucleo-32 está más optimizada para el uso que se le va a dar, ya que, aunque posea menos pines que la Discovery, sigue teniendo las funciones necesarias para el control del servomotor. Además, su precio es inferior, por lo que ha sido la mejor elección.

La tarjeta STM32F303K8T6 posee un controlador Arm Cortex M-4 con la posibilidad de operar hasta una frecuencia de 72 MHz. Además, posee muchas otras funciones diferentes, así que, dado que la información disponible es extensa, se explicarán solamente las funciones que se han empleado en la realización del proyecto.

4.1.1 Timer

Los timers son periféricos hardware que se encargan de realizar retardos de cierta precisión y contar eventos. Estos dispositivos permiten realizar varias actividades. Por ejemplo, se encargan de medir anchos de pulso de señales, generar señales digitales, contar impulsos, provocar acciones periódicas, implementar relojes en tiempo real, generar el ritmo para comunicaciones, generar señales PWM, etc. Existen además varios tipos de temporizadores, como los timers avanzados, los timers de uso genérico y los timers básicos. La diferencia entre ellos es que poseen características extra que les dan más funcionalidades. Aunque todos suelen seguir la misma estructura básica.

La configuración de los temporizadores básicos suele seguir el esquema de la figura 3.

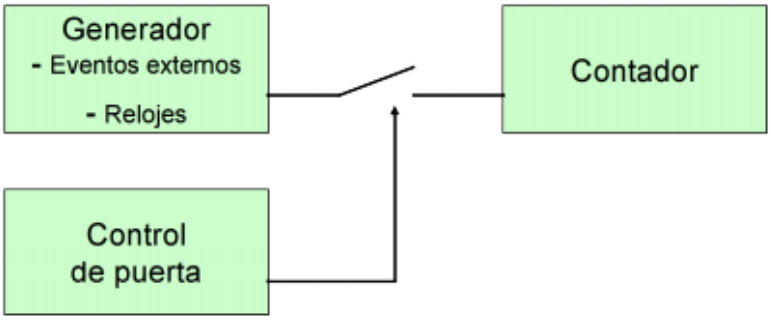


Figura 3. Bloques de un timer.

Fuente: ARM Cortex-M práctico. 1 - Introducción a los microcontroladores STM32 de St

El bloque de "Generador" es el origen de las señales digitales que han de ser contabilizadas. "Control de puerta" hace referencia al componente que habilita o deshabilita la llegada de las señales digitales del "Generador". Se suele emplear el control por software o señales externas para realizar dicho control. Por último, se tiene el "contador", cuyo valor varía en función de los eventos que reciba (incrementándose o decrementándose). Además, se suelen incluir características para comparar el valor de la cuenta con un patrón y para realizar recargas automáticas de la cuenta (por ejemplo, reiniciar el valor del contador a 0 cuando supere un valor determinado).

4.1.2 PWM

El modulador de ancho de pulso o Pulse Width Modulator (**PWM**) consiste en una señal cuadrada con una frecuencia determinada que puede variar su ciclo de trabajo para así poder modificar el valor nominal de la tensión. Para poder producir una señal PWM en la tarjeta nucleo-32, se emplea un timer en modo PWM.

El funcionamiento es el siguiente. El timer cuenta hacia arriba desde 0 hasta el valor del registro de auto recarga (TIM_ARR). Una vez que el contador iguala el valor del registro de captura (TIM_CCRx), el canal en el que ha sido configurado el timer, produce una señal de valor 0. Al llegar de nuevo al valor del registro de auto recarga (TIM_ARR), el contador se reinicia a 0 y la señal de salida se selecciona a 1.

Configurando el valor de TIM_CCR y TIM_ARR, se puede modificar el valor del ciclo de trabajo y la frecuencia de la señal PWM.

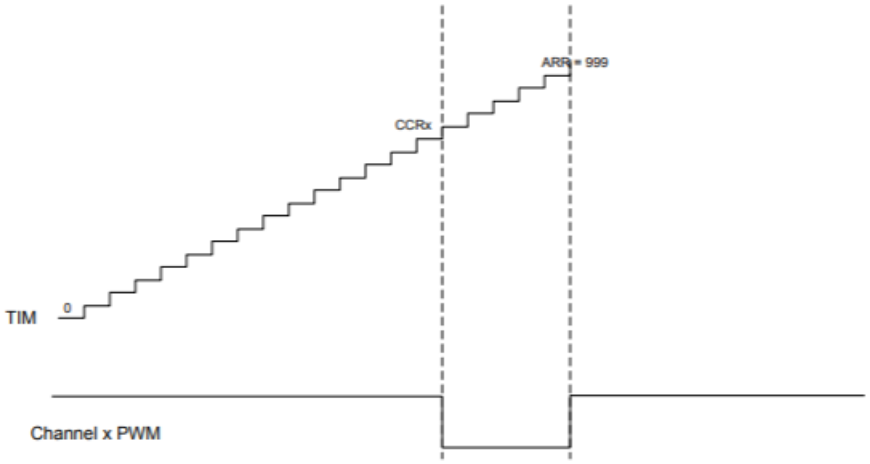


Figura 4. Formación de señal PWM.

Fuente: https://www.st.com/resource/en/application_note/dm00513079-generating-pwm-signals-using-stm8-nucleo64-boards-stmicroelectronics.pdf

Además, un timer puede generar varias señales PWM con el mismo periodo, pero con distinto ciclo de trabajo. Esta funcionalidad no la poseen los timers básicos.

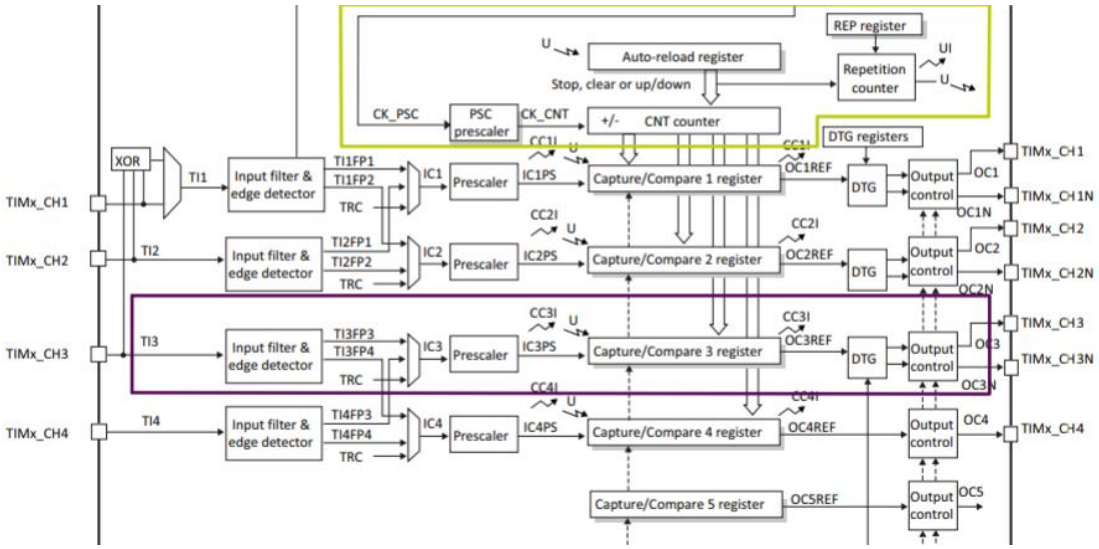


Figura 5. Diagrama canales del timer.

Fuente: https://www.st.com/resource/en/application_note/dm00236305-general-purpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf

4.1.3 USART

El USART es el periférico que se encarga de recibir y enviar información desde el microcontrolador al computador. La tarjeta utilizada posee tres periféricos de este tipo (USART1, USART2 y USART3). Este componente se comunica a velocidades de hasta 9 Mbits/s.

4.2 Motor

Para el motor se han elegido dos modelos de la marca Maxon. Se ha utilizado esta marca porque se ha comprobado que sus productos son fiables y profesionales. Los posibles modelos para tener en cuenta son:

- Motor A-max 16 Ø16 mm, Imán permanente, 1.2W, 12V
- Motor RE 16 Ø16 mm, Imán permanente CLL, 3.2W, 12V

Dado que el objetivo de este proyecto es realizar el control de un servomotor, es irrelevante qué motor se utilice siempre y cuando sea de uso profesional. Por lo tanto, se utilizará el motor que mejor cumpla las siguientes características:

- Motor de 12V.
- Mejor ratio de calidad/precio.

Finalmente se ha elegido el motor **A-max 16 Ø16 mm de imán permanente** La razón principal ha sido el precio del motor. En comparación con el otro motor, este costaba dos tercios menos.

Esta diferencia de precio ha sido debido a que la potencia de este es de 1.2 W mientras que el otro era de 3.2W. Así que se ha optado por este.

Figura 6. Motor A-max 16 Ø16 mm de imán permanente de Maxon.

Fuente: https://www.maxongroup.com/maxon/view/content/Overview-brushed-DC-motors

Este motor posee una velocidad nominal de 4920 rpm, un par nominal de 2,11mNm y una corriente nominal de 0.209A.

4.3 Sensor de posición

El sensor de posición es el encargado de medir la rotación angular del eje del motor. Existen varios tipos de sensores de posición. Los más comunes son los potenciómetros, estos componentes varían su resistencia interna a partir de la posición angular en la que se encuentren. Al ser proporcional la resistencia con la posición, también lo será con la tensión de salida. Los potenciómetros funcionan bien en aplicaciones con ciclos no exigentes y bajo rendimiento, pero cuando las aplicaciones son más exigentes no son una buena idea ya que son susceptibles al desgaste. Por estas razones, no se empleará este tipo de sensor para la detección de la posición angular del motor.

Otros sensores de posición son los denominados sensores ópticos, también llamados encoders. Su funcionamiento consiste en los siguientes pasos. Primero, una luz brilla a través o sobre una rejilla codificada. La luz que pasa por los agujeros es medida por un fotodetector y genera una señal de posición. Esta rejilla posee un número limitado de patrones que delimita la precisión del sensor y por tanto su calidad. Para la aplicación propuesta, se empleará este tipo de sensor dado que tiene una mayor resistencia al desgaste, ya que no tiene contactos mecánicos, y una mayor repetibilidad y precisión.

También existen otros tipos de sensores como los magnéticos, capacitivos o los incoders. Estos tipos no se van a poder utilizar porque el fabricante de los motores seleccionado no emplea dichos sensores. Por esta razón, no se explicará su funcionamiento.

Los posibles encoders elegidos han sido los siguientes:

- Encoder MR, tipo ML, 512 ppv (pulsos por vuelta) y 3 canales.
- Encoder MR, tipo M, 128 ppv y 3 canales.

Estos modelos de encoder son de tipo incremental de cuadratura. Este tipo determina la posición angular a partir de la última posición registrada. Para ello, estos encoders utilizan dos sensores ópticos posicionados con un desfase de 90º entre ellos. Gracias a este desfase, los pulsos que generan ambos sensores también están desfasados y comparándolos se puede obtener la posición y dirección de la rotación del eje. Estas señales reciben el nombre de A y B. Si solamente poseen estas dos señales, se considera que el encoder es de 2 canales. Para el caso de 3 canales, a parte de las señales A y B, también se introduce una señal de referencia llamada I. Esta señal es independiente de A y B y se activa cuando el eje gira hasta llegar a una posición determinada. Esta señal se emplea para tener un punto de referencia para cuando se desconecte el encoder y así poder repetir los mismos movimientos. Gracias a este tercer canal se resuelve el problema de no poder saber en qué posición se encontraba el eje anteriormente, ya que cuando se desconecta y se vuelve a conectar el sensor, la posición actual se convierte en la posición 0.

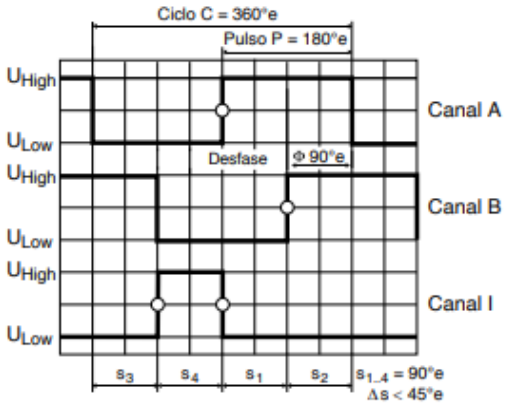


Figura 7. Señales de los canales A, B e I.

Fuente: https://www.maxongroup.com/medias/sys_master/root/8846495678494/20-ES-461-462.pdf

Las principales características que debe poseer el encoder que se va a utilizar son:

- Debe poseer tres canales de control.
- Debe poseer el mejor ratio de precisión/precio.

El encoder elegido ha sido el encoder MR, tipo ML, de 512 ppv y 3 canales. La principal razón de elegir este encoder ha sido que la relación precisión/precio ha sido superior en este sensor.

Figura 8. Encoder MR de 512 ppv y 3 canales.

Fuente: https://www.maxongroup.com/medias/sys_master/root/8846495678494/20-ES-461-462.pdf

Antes de pasar al siguiente apartado, hay que hacer una breve mención sobre una característica que tiene este encoder en concreto. Esta característica consiste en que posee, además de los canales A, B e I, también sus complementarios. Es decir, producen la misma señal, pero de forma invertida. Posee esta característica para poder emplear en compañía del encoder, un componente llamado differential line receiver. Este componente se explicará brevemente a continuación.

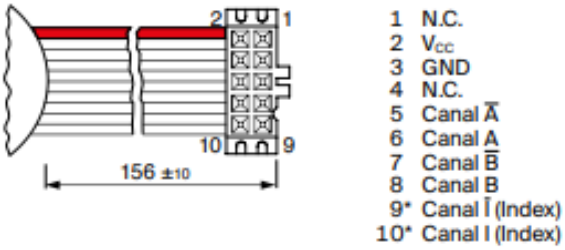


Figura 9. Conexiones y pines del encoder.

Fuente: https://www.maxongroup.com/medias/sys_master/root/8846495678494/20-ES-461-462.pdf

4.3.1 Differential line receiver

El differential line receiver es un componente extra empleado junto con el encoder. Se utiliza para eliminar las interferencias de ruido que pueden afectar a la señal del encoder. Para su correcto funcionamiento se necesita utilizar la señal del encoder y su complementaria. Ambas señales son introducidas en el componente, ahí se resta la señal original con la complementaria. Al realizar la resta, las interferencias que son iguales para ambas líneas se restan, obteniendo la señal sin el ruido. A continuación, se muestra una imagen con la explicación gráfica:

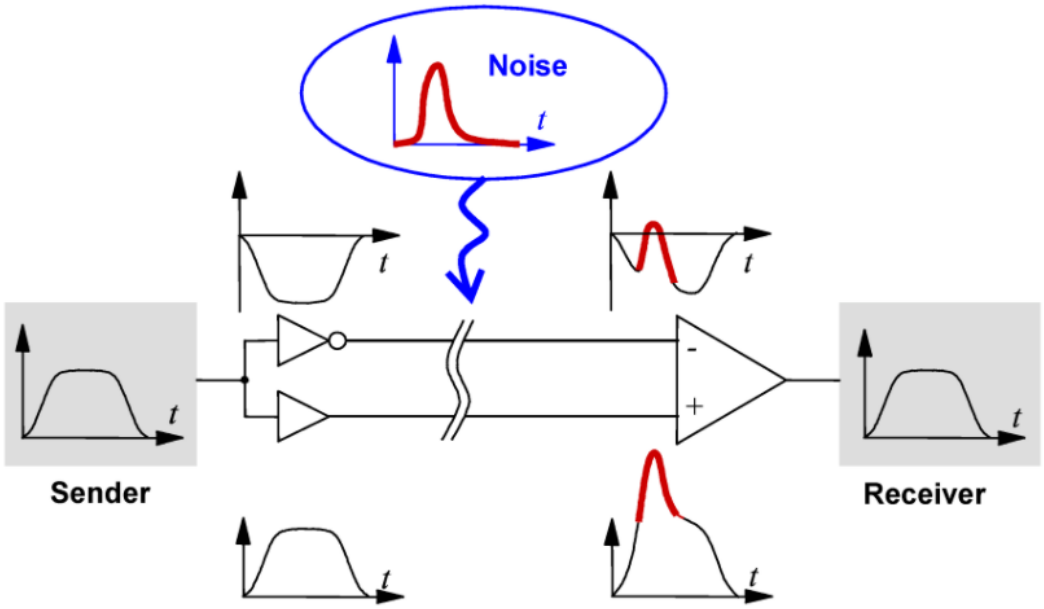


Figura 10. Representación del efecto del differential line receiver.

Fuente: https://resources.altium.com/es/p/principios-de-pares-diferenciales-parte-1

Este componente funciona igual que un comparador con la única diferencia que pueden conmutar valores a velocidades superiores.

El modelo elegido ha sido el SN75175.

4.4 Transistores

Para poder controlar la etapa de potencia del motor, es necesario el uso de transistores. Para este caso, el tipo de transistor que mejor se ajusta para el control de un servomotor, es el tipo MOSFET. Su principal característica es que funcionan por tensión, es decir, son controlados por el nivel de voltaje y no por el nivel de corriente. Además, la velocidad de conmutación es muy alta con respecto a otros transistores. Por esta razón son los más utilizados para este tipo de aplicaciones de potencia. Se han elegido varios modelos de transistores MOSFET:

- IRF540
- IRLZ24
- P55NF06

Para elegir adecuadamente el tipo de transistor que se va a emplear de los propuestos en el apartado anterior, se van a filtrar por las siguientes características:

- Voltaje de drenaje-fuente superior a 24V.
- Velocidad total de conmutación inferior a 1us.
- Corriente máxima superior a 2A.

El transistor que mejor características poseía ha sido el transistor P55NF06. Aunque los tres transistores propuestos cumplían los criterios, este ha sido el que tenía el menor tiempo de conmutación y sería el más apto para la aplicación propuesta. Sin embargo, el transistor que se empleará será el modelo IRF540 debido a que ya se poseían varios transistores de este tipo procedentes de otras aplicaciones. A continuación, se comparan los tiempos de conmutación de las hojas de datos de los dos componentes.

Turn-On Delay Time[c]	$t_{d(on)}$	V_{DD} = 30 V, R_L = 0.6 Ω $I_D \cong$ 50 A, V_{GEN} = 10 V, R_g = 2.5 Ω		10	20	ns
Rise Time[c]	t_r			15	25	
Turn-Off Delay Time[c]	$t_{d(off)}$			35	50	
Fall Time[c]	t_f			20	30	

Figura 11. Tiempos de subida y bajada de conmutación del mosfet P55NF06.

Fuente: https://pdf1.alldatasheet.com/datasheet-pdf/view/1274994/VBSEMI/P55NF06.html

Switching Characteristics (T_C = 25°C, *Figures 1, 2,*)[3]

$t_{d(on)}$	Turn-On Delay Time		30	ns	V_{DD} = 45 V, I_D = 15 A V_{GS} = 10 V, R_{GEN} = 4.7 Ω R_{GS} = 4.7 Ω
t_r	Rise Time		60	ns	
$t_{d(off)}$	Turn-Off Delay Time		80	ns	
t_f	Fall Time		30	ns	
$t_{d(on)}$	Turn-On Delay Time		60	ns	V_{DD} = 25 V, I_D = 15 A V_{GS} = 10 V, R_{GEN} = 50 Ω R_{GS} = 50 Ω
t_r	Rise Time		450	ns	
$t_{d(off)}$	Turn-Off Delay Time		150	ns	
t_f	Fall Time		200	ns	

Figura 12. Tiempos de subida y bajada de conmutación del mosfet IRF540.

Fuente: https://pdf1.alldatasheet.com/datasheet-pdf/view/52961/FAIRCHILD/IRF540.html

4.5 Drivers

Los drivers son componentes encargados de controlar de forma adecuada el componente para el cual son diseñados. Para este caso, se emplean drivers para los mosfets. Su acometido principal es poder controlar la puerta del transistor y mantener el estado de la salida indefinidamente. Se emplea para cualquier control de motor, lampara o calentador y se encarga de que los transistores no se dañen por altos voltajes y augmenta la vida útil de estos.

Se han seleccionado varios componentes. Lo mejor hubiese sido obtener un driver que controlase los cuatro transistores que forman el puente en H. No obstante, dado que la gran mayoría de estos poseían un encapsulado que no era de agujero pasante, se eligieron drivers de un mosfet o de dos (tipo Half-bridge o medio puente).

- TC4431: Mosfet Driver.
- MIC4605: Half-bridge Mosfet Driver.
- MIC5014: Mosfet Driver.

Dado que los drivers son los encargados de controlar los transistores, deberán de poseer las mismas características o superiores que los transistores elegidos.

El driver elegido para controlar el transistor ha sido del MIC5014. Se ha elegido dicho driver porque, aunque los otros drivers también cumplieran los criterios propuestos, este era el que tenía menor precio. Aunque si se compara con el driver MIC4605, este resultaría más barato ya que se emplearían dos en vez de cuatro. Pero debido a que se ha comprado con los fondos personales del alumno, se ha optado por la versatilidad del MIC5014 ya que se puede emplear para más aplicaciones y proyectos que el MIC4605 por ser este un driver de medio puente H.

4.6 Fotoacoplador

Un fotoacoplador es un interruptor activado mediante luz infrarroja emitida por un led. Este tipo de componentes se utilizan principalmente cuando se requiere obtener una señal que se encuentra en un circuito con un potencial diferente. Este componente aporta un gran aislamiento eléctrico entre dos circuitos con potenciales diferentes de alimentación. Para este proyecto se utilizará para detectar si la etapa de potencia está encendida o no.

El modelo de fotoacoplador elegido ha sido el PC817. La razón principal de su uso es su bajo precio y su amplio uso en el mercado.

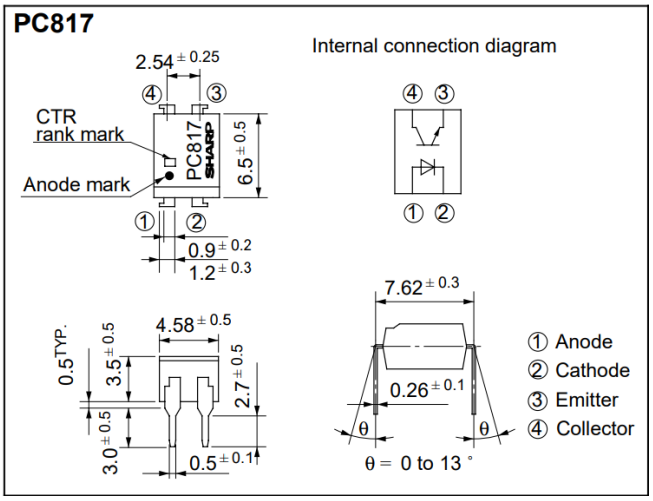


Figura 13. Diagrama y pines del fotoacoplador PC817.

Fuente: https://pdf1.alldatasheet.com/datasheet-pdf/view/43371/SHARP/PC817.html

4.7 Fuente de alimentación

El motor que se va a utilizar debe funcionar a 12 voltios, por lo tanto, la fuente de alimentación deberá de poseer dicho valor. Hay que tener en cuenta que para cada par de driver y transistor existe una pequeña caída de potencial, por lo que el motor recibe un poco menos que 12 voltios. El hecho de recibir menos voltaje reduce la velocidad máxima a la que se mueve el motor, pero dado que se trata de realizar un control de la posición, no es un problema crítico por lo que se elegirá igualmente una fuente de 12 V. No se ha intentado probar a alimentar el motor con una fuente de voltaje superior ya que podría poner en peligro la seguridad del motor.

El hecho de poseer una fuente de alimentación conmutada de 12V y 25A ha evitado tener que comprar este componente. En el caso de no estar en posesión de dicha fuente, se habría optado por un alimentador de 12V y 2A.

Figura 14. Fuente de alimentación conmutada de 12V y 25A.

5. Funcionamiento del sistema empleado

A continuación, se procederá a explicar tanto el montaje como la programación de la solución adoptada para resolver el problema.

5.1 Montaje del circuito

Se han realizado varios subcircuitos para poder conseguir todas las funcionalidades necesarias descritas en el apartado 3.

El primer subcircuito que se muestra es la etapa de potencia. Como se puede observar en la figura 15, esta etapa está formada por el puente en H de los mosfet IRF540 controlados por los drivers MIC5014. El puente H es una disposición de los transistores que permite controlar el giro del motor en dirección horaria y antihoraria. Para poder girar el motor en un sentido hay que enviar la señal PWM a la pareja de transistores que forman una diagonal y desconectar la otra pareja complementaria. En el caso que se activasen los cuatro transistores a la vez, produciría un cortocircuito en el motor. Para esta aplicación, las parejas de motores son los transistores Q1-Q4 y Q2-Q3 respectivamente.

Para realizar el conexionado de los drivers correctamente, hay que conectar el pin 5 a la puerta y el pin número 3 a la fuente del mosfet. Además, para evitar fallos en la alimentación de los drivers, hay que conectar un condensador electrolítico de 100uF al pin número 1 de cada controlador.

Dado que para controlar el servomotor solamente se utilizará una señal PWM, se han empleado dos fotoacopladores PC817 para controlar el sentido de giro del motor. Estos dos componentes son controlados a partir de dos señales digitales provenientes de la tarjeta nucleo-32. El pin número 3 de los fotoacopladores se conecta a cada uno a los drivers U3-U6 y U4-U5 respectivamente.

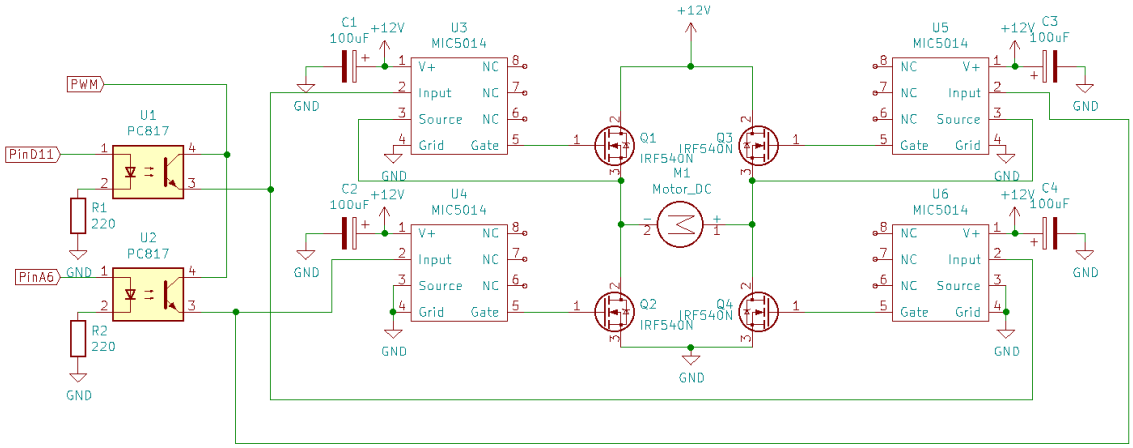


Figura 15. Puente H del control del motor.

Fuente: Propia

Además de la figura 15, también se ha introducido una tabla para representar las conexiones más importantes de forma sencilla.

Pin D0 PWM Nucleo-32	**Pin 4 PC817 (U1 y U2)**
Pin D11 Nucleo-32	**Pin 1 PC817 (U1)**
Pin A6 Nucleo-32	**Pin 1 PC817 (U2)**
Pin 3 PC817 (U1)	**Pin 2 MIC5014 (U3 y U6)**
Pin 3 PC817 (U2)	**Pin 2 MIC5014 (U4 y U5)**
Pin 3 MIC5014 (U3)	**Pin 3 Fuente IRF540 (Q1)**
Pin 3 MIC5014 (U5)	**Pin 3 Fuente IRF540 (Q3)**
Pin 3 MIC5014 (U4)	**GND**
Pin 3 MIC5014 (U6)	**GND**
Salida +12 fuente alimentación	**Pin +12V**

Tabla 1. Conexionado principal subcircuito etapa de potencia.

Fuente: Propia

Una vez explicada la etapa de potencia, se va a explicar el conexionado del encoder con el differential line receiver y la nucleo-32.

Como se ha explicado anteriormente en el apartado 4.3.1, para el correcto funcionamiento del line receiver, se necesita tanto la señal original de cada hilo del encoder como su complementario. A continuación, se muestra en la figura 16, donde se deben conectar los pines y las resistencias de cada hilo de señal.

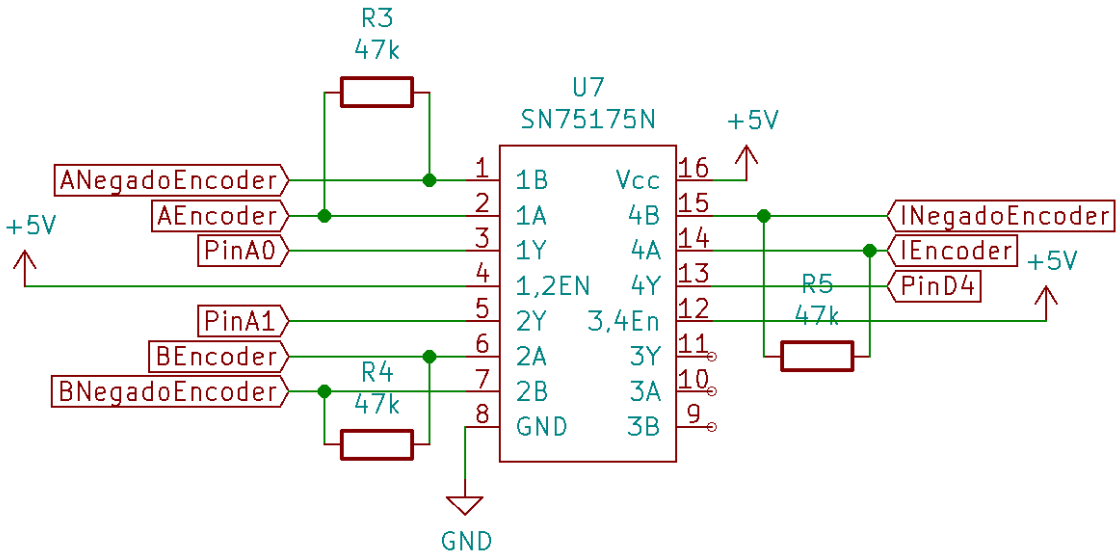


Figura 16. Subcircuito de conexión del encoder con el differential line receiver y la nucleo-32.

Fuente: Propia

Como se puede observar, se han colocado resistencia para cada par de señales. Esta resistencia debe poseer un valor superior a 1KΩ y se emplea para proteger el dispositivo de las corrientes de polarización que se crean entre ambos cables.

Además, se ha realizado una tabla para explicar de forma directa que tipo de pines son y de donde provienen.

Pin A0 Nucleo-32	**Entrada señal del encoder A**
Pin A1 Nucleo-32	**Entrada señal del encoder B**
Pin D4 Nucleo-32	**Interrupción de entrada digital** **Indice del encoder**
Alimentación +5V Nucleo-32	**Pin +5V**

Tabla 2. Conexiones encoder con la tarjeta nucleo-32.

Fuente: Propia.

El tercer subcircuito hace referencia al conexionado de los finales de carrera encargados de limitar la posición del motor y el fotoacoplador encargado de detectar si la alimentación de la etapa de potencia (+12V) está activada o no. Como se puede observar en la figura 17, tanto para los finales de carrera como para el circuito de detección del fotoacoplador, se han colocado las resistencias en modo pull-down. Los pines de entrada digital D12, D5 y D3 de la nucleo-32 han sido configurados como interrupción para así darles prioridad respecto el resto del código.

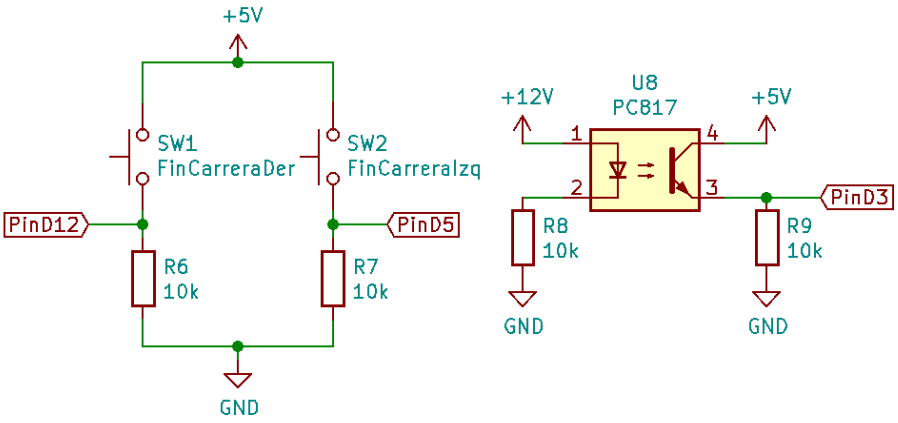


Figura 17. Circuito de los finales de carrera y conexionado del fotoacoplador de la etapa de potencia.

Fuente: Propia.

5.2 Programación

Una vez expuestas las conexiones necesarias de todo el circuito, se procederá a explicar el funcionamiento del código empleado para la programación de la tarjeta de desarrollo nucleo-32.

5.2.1 Entorno de programación

Antes de introducir el código, hay que hacer una breve referencia al entorno de desarrollo empleado para la programación del microcontrolador. Se han utilizado dos programas complementarios entre sí, el programa “STM32CubeMX” y “Keil uVision 5”.

5.2.1.1 STM32CubeMX

El programa STM32Cube es un conjunto de herramientas gratuito que permite la implementación rápida de proyectos en la plataforma STM32. Este programa posee una interfaz gráfica que permite generar el código para inicializar los componentes necesarios. Por ejemplo, se puede seleccionar que quieres habilitar una señal PWM y él te muestra en qué pines estaría disponible su implementación. Además, una vez seleccionados los componentes que se quieren habilitar, te genera el código automático necesario para la configuración elegida.

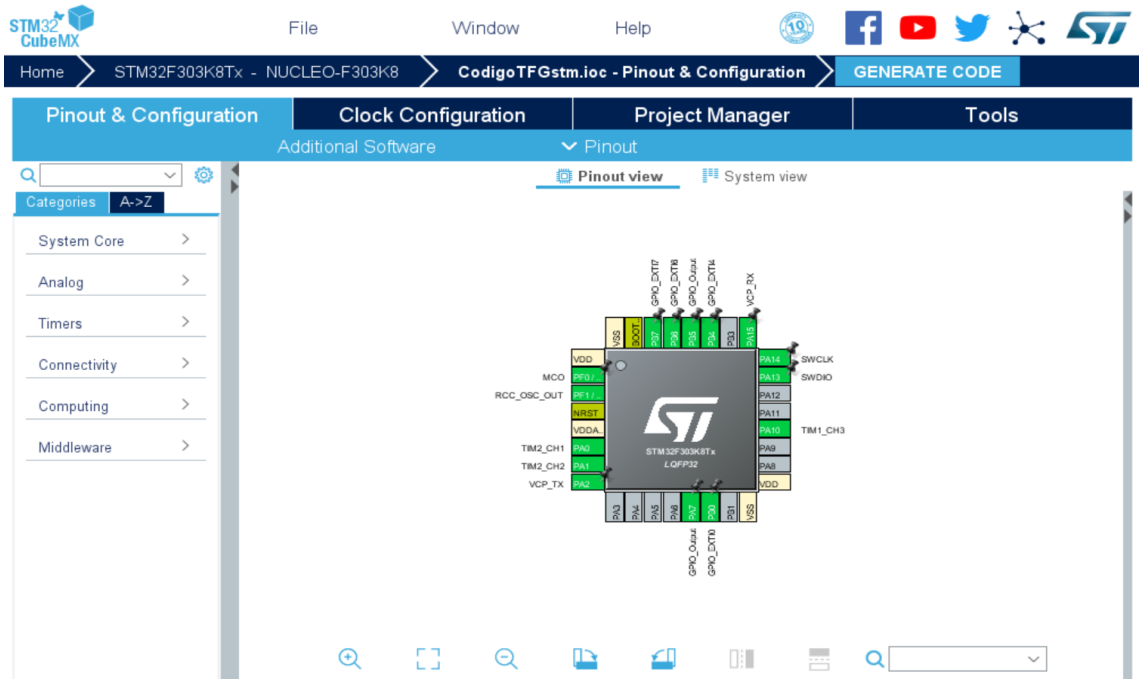


Figura 18. Pantalla de inicio STM32CubeMX con pines inicializados.

Fuente: Propia

Para mostrar de forma gráfica la configuración que se ha realizado para cada pin, se ha construido una tabla donde se señalan las características principales que deben poseer cada uno de los periféricos.

FUNCIÓN	TIPO	PINES	CONFIGURACIÓN			
Señal PWM	TIMER1	PA10	**Channel3** = PWM Generation CH3	**Prescaler** = 32-1	**Counter Period** = 1000-1	
Conexión USART	USART2	PA2 y PA15	**Mode** = Asynchronous	**Baud Rate** = 9600	**Word Length** = 8 bits	
Señal Encoder	TIMER2	PA0 y PA1	**Combined Channels** = Encoder Mode	**Prescaler** = 0	**Counter Period** = 512*4	**Encoder Mode** = Encoder Mode TI1 and TI2
Temporizador de 100uS	TIMER6		**Prescaler** = X	**Counter Period** = X	**NVIC Interrupt Table: TIM6 global and DAC1** = ENABLE	
Interrupción detección señal indice encoder	GPIO_EXTI7	PB7	**EXTI line[9:5] interrupts** = Enabled			
Interrupción final de carrera izquierdo	GPIO_EXTI6	PB6	**EXTI line[9:5] interrupts** = Enabled			
Interrupción final de carrera derecho	GPIO_EXTI4	PB4	**EXTI line 4 interrupt** = Enabled			
Interrupción detección alimentación etapa potencia	GPIO_EXTI0	PB0	**EXTI line 0 interrupt** = Enabled			
Salida digital sentido de giro del motor	GPIO_Output	PA7				
Salida digital sentido de giro del motor	GPIO_Output	PB5				

Tabla 3. Configuración de los pines en STM32CubeMX.

Fuente: Propia

Como se puede observar en la tabla 3, el TIMER6 posee un prescaler = X y un counter period = X. Esto significa que se debe introducir cualquier valor aleatorio. Se elige un valor aleatorio porque en el apartado 5.2.1.2 se configurará correctamente dichos valores a través del programa uKeil.

Para el caso de la señal PWM, se han elegido los valores de prescaler= 32-1 y counter period = 1000-1 porque se ha seleccionado una frecuencia igual a 10 KHz. Estos valores se pueden calcular a partir de la siguiente ecuación:

$$f = \frac{Clock}{(prescaler + 1) * (counter + 1)}$$

Donde "f" es la señal de PWM y "clock" es la frecuencia del reloj del microcontrolador.

Se ha elegido un valor de clock de 32 MHz y un valor de counter period de 1000. Así, al despejar la ecuación, se ha obtenido una señal de prescaler igual a 32.

Una vez configurados los pines, hay que modificar el valor de frecuencia del reloj, para ello, hay que ir al apartado "Clock Configuration". Allí se pueden modificar tanto los valores de la

frecuencia del reloj general en el apartado “HCLK (MHz)” como los específicos para cada periférico. Para este caso, se ha seleccionado una frecuencia de 32 MHz de reloj. Hay que tener en cuenta que, al modificar un parámetro, el programa modifica los valores que dependen de este valor para así encontrar la solución adecuada.

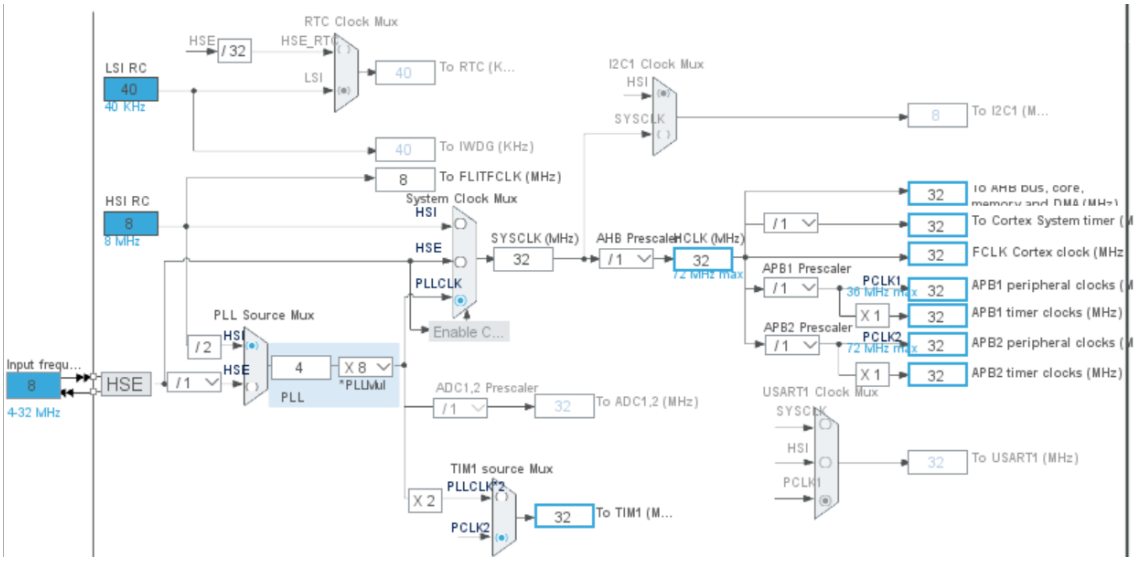


Figura 19. Configuración de la frecuencia del reloj.

Fuente: Propia

Como se ha explicado anteriormente, el programa genera el código necesario para la inicialización. Este código se puede crear para diferentes plataformas de desarrollo. Para este caso, se deberá implementar para el programa Keil uVision 5, por lo tanto, en el apartado “Toolchain/IDE”, hay que introducir “MDK-ARM”.

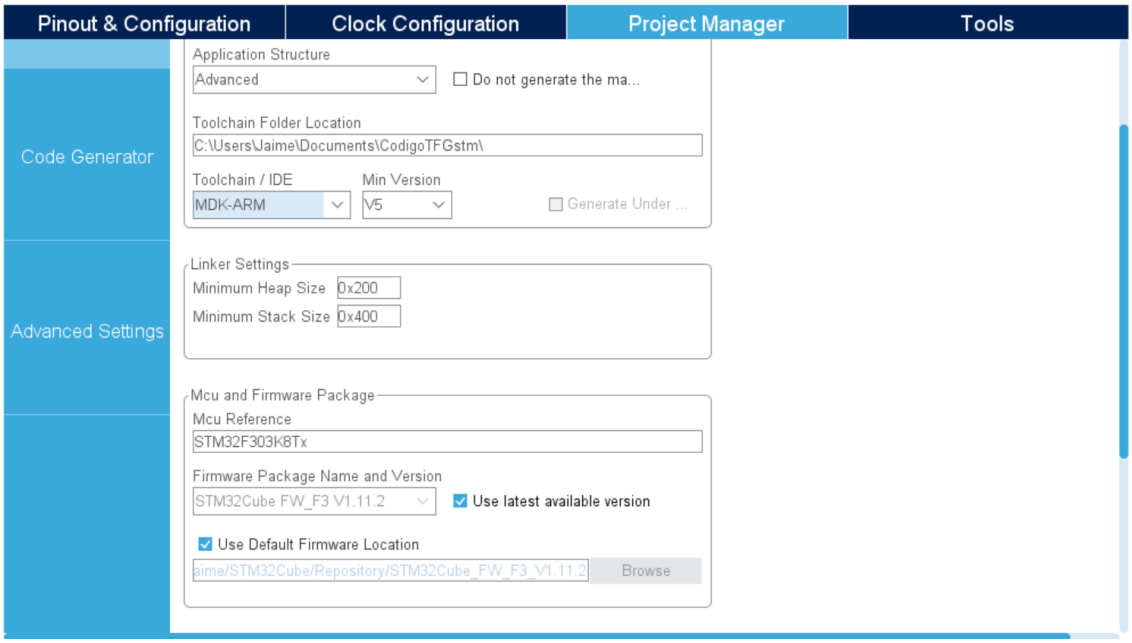


Figura 20. Configuración de la generación de código.

Fuente: Propia

Una vez acabado de realizar las configuraciones pertinentes en el programa STM32CubeMX y generado el programa de Keil uVision 5 se puede empezar a programar el funcionamiento del sistema propuesto.

5.2.1.2 Keil uVision 5

El programa Keil uVision 5 es un entorno de desarrollo con diversas herramientas para la programación, test, verificación y depuración de código para sistemas embebidos.

Una vez generado el archivo de Keil desde el CubeMX se puede observar la pantalla de la figura 21.

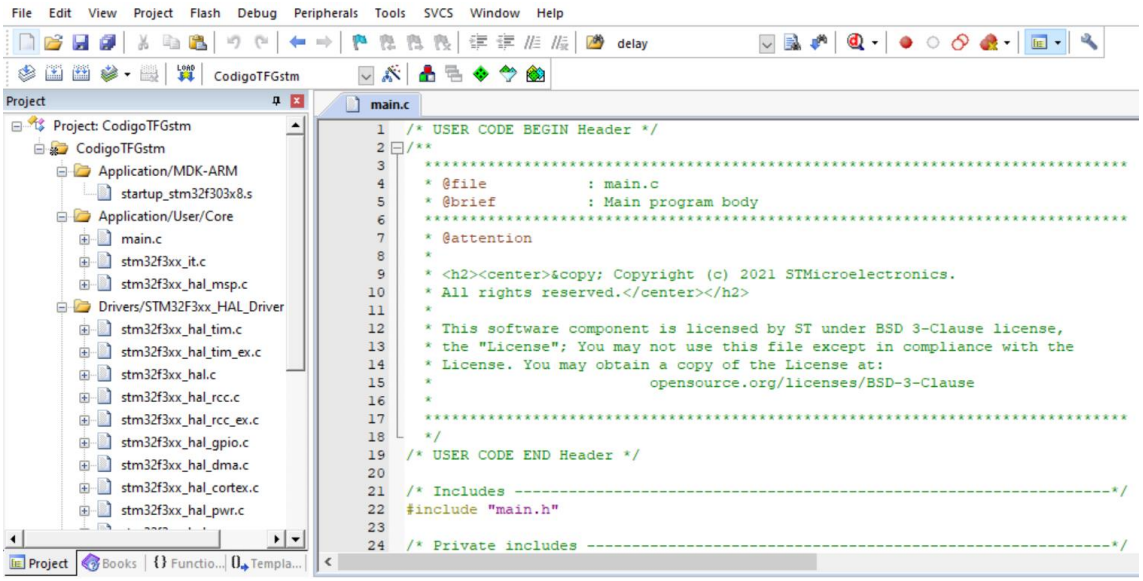


Figura 21. Inicio Keil uVision 5.

Fuente: Propia

Como se ha indicado en el apartado 5.2.1.1, ahora se explicará como configurar correctamente los valores de prescaler y de counter period del TIMER6.

Para el valor del prescaler, se debe emplear el comando “SystemCoreClock”, que obtiene el valor de la frecuencia del reloj. Este valor se debe dividir entre la frecuencia que se quiere obtener. Para este caso, se quiere obtener un periodo de 1us, por lo tanto, se debe dividir entre 10^6. Para poder obtener un timer de 100us, se debe multiplicar este valor por 100. Para ello, solamente se debe introducir este valor como counter period. Hay que tener en consideración que para ambos valores se debe restar una unidad para compensar que ambas variables empiezan por 0. A continuación se muestran los valores del apartado de configuración del TIMER6.

```
htim6.Init.Prescaler = (SystemCoreClock/1000000)-1;
htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
htim6.Init.Period = 100-1;
```

5.2.1.3 Tera Term

Para poder comunicar el microcontrolador con el computador, se necesita un software especial para poder realizar la comunicación en serie. Para ello se ha utilizado Tera Term. Este programa

es un emulador de terminal gratuito compatible con Microsoft Windows. Además de establecer conexiones de puerto serie, también permite realizar conexiones SSH y Telnet.

Una vez abierto el programa, hay que seleccionar la opción “Serial” y elegir el puerto en que se encuentra la tarjeta de desarrollo.

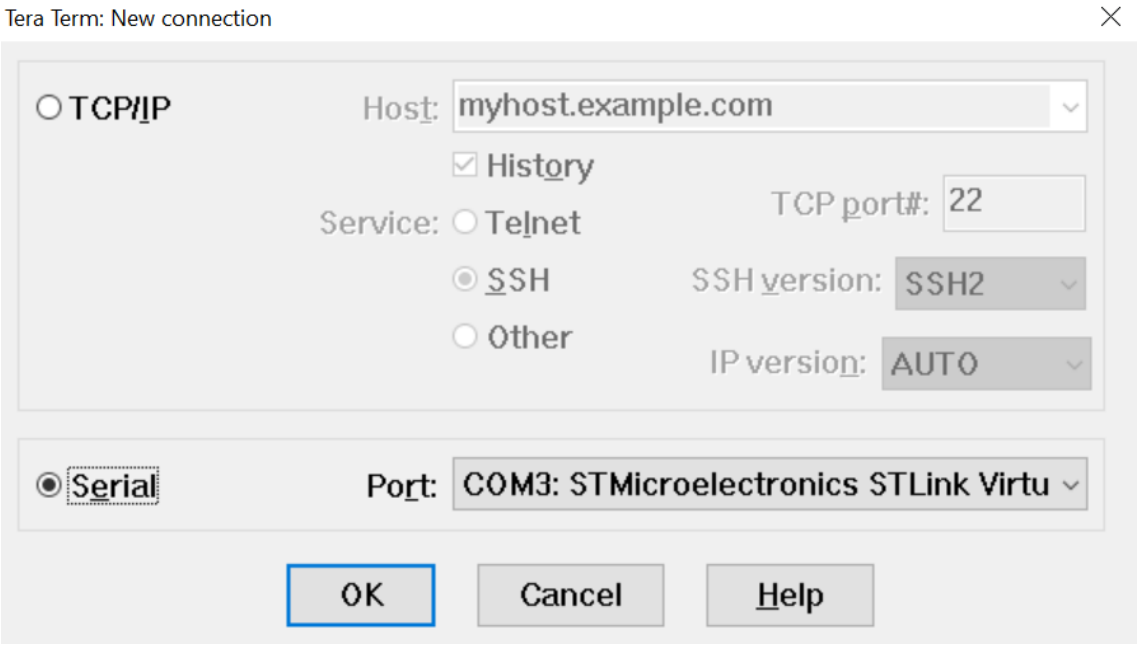


Figura 22. Pantalla inicial Tera Term.

Fuente: Propia

Una vez elegida, aparecerá una pantalla en negro donde se puede configurar la velocidad en baudios de la comunicación y otras características de la consola. Para esta aplicación, no se necesita realizar ningún cambio. La única acción que se debe ejecutar es reiniciar la tarjeta nucleo-32 una vez se encienda esta pantalla.

5.2.2 Código

Antes de generar el código, se debe determinar qué tipo de control se realizará. El tipo de controlador será Proporcional-Derivativo (PD) ya que no se van a tener en cuenta interferencias externas como la gravedad y por tanto no se requiere un controlador Proporcional-Integral-Derivativo (PID), ya que, aparte de hacer el control más lento, también puede llegar a la inestabilidad si se introducen las constantes erróneas.

Para poder generar las funciones necesarias de una forma ordenada y sencilla, se ha dibujado un diagrama que explica brevemente el funcionamiento y las acciones que se deben de realizar. Cada uno de los bloques serán explicados con profundidad más adelante.

Para el código principal, se programarán las siguientes funciones dentro de la función principal “main()”:

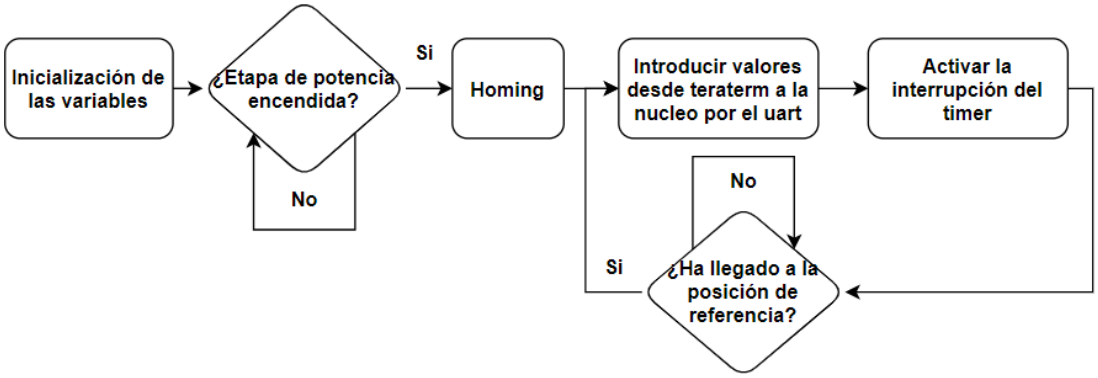


Figura 23. Diagrama de bloques de la función "main()".

Fuente: Propia.

Como se puede observar, el primer paso es inicializar las variables y constantes que harán falta para la administración de la información. Una vez inicializadas, se realiza una parada del sistema hasta que no se detecte que la etapa de potencia está alimentada.

Si está alimentada, se realiza la función "Homing". Esta función emplea la señal índice del encoder y consiste en situar el motor en la posición donde se activa dicha señal. Para ello, primero se gira el motor en sentido horario a velocidad media hasta detectar que la señal índice se ha activado. Al hacerlo, debido a la inercia de giro del motor, este no para exactamente en la posición objetivo. Así que se debe girar en sentido antihorario a velocidad aún más reducida para llegar a dicha posición. Por convenio, una vez realizado el giro en sentido antihorario, se debe volver a girar el motor en sentido horario a la velocidad más reducida posible para asegurarse que se encuentra en la posición exacta.

Una vez situado el motor en la posición inicial correcta, la tarjeta pide al usuario por el programa teraterm la posición objetivo en grados, la velocidad en grados/s, la aceleración en grados/s^2 y la precisión de seguimiento de la trayectoria. La precisión tiene en cuenta si se elige la interrupción del timer que se activa cada 100 uS o 1 mS.

Al introducir todos los datos, el timer se activa. Dentro de este, se deberán realizar una serie de comandos en cada periodo de tiempo. A continuación, se muestran dichos comandos:

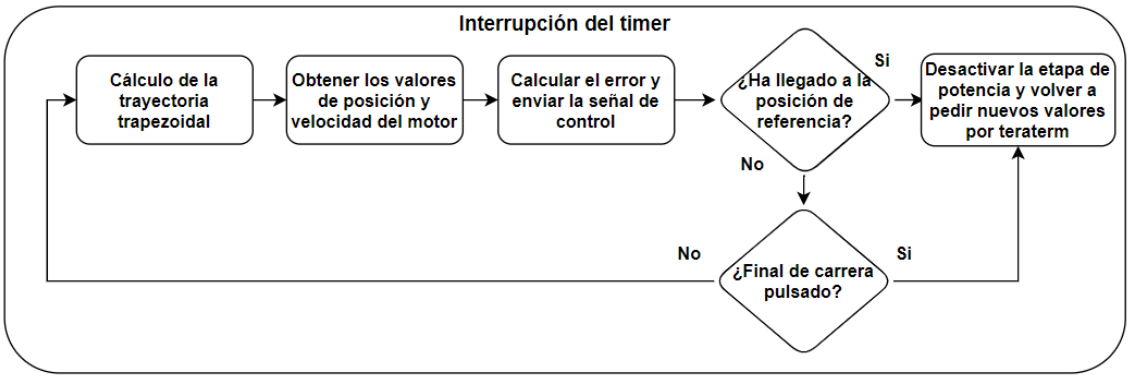


Figura 24. Diagrama de bloques del bucle de interrupción.

Fuente: Propia.

5.2.2.1 Funciones generales

Antes de explicar las funciones descritas en los diagramas, se van a exponer algunas funciones que realizan funciones sencillas que permiten reducir el tamaño del código. Estas son las siguientes:

- **posActualRad():** Obtiene la posición del encoder en pulsos por vuelta y la devuelve en radianes.

```
//Función que lee el valor de la posición del encoder actual y la devuelve
en rad
float posActualRad()
{
   uint16_t posicion;
   float posicionRad;
   posicion= TIM2->CNT; //La resolución del encoder es de 4x512 =
2048  pulsos por vuelta
   posicionRad=(double)posicion*(((double)3.14159*2)/(double)2048);
   //Obtener la posición en radianes a partir del número de pulsos
   return posicionRad;

}
```

- **giroHorario(), giroAntihorario() y paro():** Funciones para determinar el sentido de giro del motor a partir de las dos salidas digitales que controlan los dos fotoacopladores.

```
//Funciones para configurar el giro del motor
void giroHorario()
{
   HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_RESET); //La señal se
vuelve 0
   HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_SET);  //La señal se
vuelve 1
}
void giroAntihorario()
{
   HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET);
   HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_SET);
}
void paro()
{
   HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET);
   HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_RESET);
}
```

5.2.2.2 Función principal main()

Dentro de la función “main()”, se ejecutarán los siguientes comandos para poder realizar el funcionamiento del diagrama de la figura 24. Dado que el código completo se expondrá en el apartado Anexos, solamente se introducirán los comandos más representativos. Para esta parte del código, se han inicializado el encoder, la señal PWM, el timer de 100 uS y las variables que guardan los mensajes y valores de Tera Term. Más adelante, dentro del bucle infinito “while()”, se espera hasta que se detecte la alimentación de la etapa de potencia. Después, cuando se detecta, se inicia la función “Homing”. Una vez realizada se pide por Tera Term al usuario que introduzca los valores pertinentes. Para el código de a continuación, solamente se ha

introducido el bloque referente a la obtención de datos de la posición, ya que, para la velocidad, aceleración y precisión, son similares. Una vez obtenidos los valores correspondientes, se transforman para convertirlos de tipo carácter a tipo entero, para así, poder utilizarlos. Luego se transforman de grados a radianes y finalmente se activa el timer encargado de realizar el control del motor.

Es importante tener en cuenta que para realizar el correcto funcionamiento de cada etapa sin que el programa solape procesos, se han empleado variables globales. Estas variables son: etapaPotenciaOn (Encargada de controlar si está alimentada la etapa de potencia o no), flagHoming (Encargada de controlar si se ha realizado la función Homing), flagIniciarControl (Encargada de seleccionar si se realiza la obtención de información por Tera Term o el bucle de control del movimiento del motor) y las variables finCarreraDer y finCarreraIzq (Detectar si se ha pulsado algún final de carrera).

```
 /* USER CODE BEGIN 2 */
//Inicialización del encoder, señal PWM y timer de 100uS
HAL_TIM_Encoder_Start(&htim2,TIM_CHANNEL_ALL);
HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_3);
HAL_TIM_Base_Start_IT(&htim6);

//Variables para guardar los valores recibidos por teraterm
uint8_t mensFinalGrados[3], mensVrefGrados[3], mensArefGrados[3],
mensPrecision[1];
//Valores recibidos por teraterm de tipo enteros
int intFinalGrados,intVrefGrados,intArefGrados,intPrecision;
float vrefGrados,arefGrados;

//Mensajes para teraterm
uint8_t mensPosFinal [15]=" Pos(360 max): ";
uint8_t mensVel [15]=" Vel(100 max): ";
uint8_t mensAcel [16]=" Acel(150 max): ";
uint8_t mensPrec [30]="Precision(1=1ms o 2=100us): ";
uint8_t espacio[2]=" ";
uint8_t mensError[15]="Valor erroneo";
uint8_t mensFinDer[30]="Interruptor derecho accionado!";
uint8_t mensFinIzq[32]="Interruptor izquierdo accionado!";

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
//Esperar hasta que se detecte la alimentación de la etápa de potencia
if(etapaPotenciaOn==1)
{
	//Homing motor
	if(flagHoming==0)

	{

		HAL_Delay(2000); //Realiza un tiempo de espera en milisegundos
		motorHoming();
		finCarreraDer=0;
		finCarreraIzq=0;
```

```
}

//Obtener la posicion deseada(grados),vel(grados/s),acel(grados/s^2) y
precision por ordenador (teraterm)
if(flagIniciarControl==0)
{
//Mostrar por teraterm si se han accionado los finales de carrera
	if(finCarreraDer==1)
	{

	//Transmitir mensaje por Tera Term

	HAL_UART_Transmit(&huart2,mensFinDer,sizeof(mensFinDer),30);
	finCarreraDer=0;
	}
	if(finCarreraIzq==1)
	{
	HAL_UART_Transmit(&huart2,mensFinIzq,sizeof(mensFinIzq),30);
	finCarreraIzq=0;
	}

	//Obtener posición deseada en grados
	//Transmitir mensaje por Tera Term
	HAL_UART_Transmit(&huart2,mensPosFinal,sizeof(mensPosFinal),20);
	//Recibir mensaje por Tera Term

	HAL_UART_Receive(&huart2,mensFinalGrados,sizeof(mensFinalGrados)
	,1000000);
	HAL_UART_Transmit(&huart2,mensFinalGrados,sizeof(mensFinalGrados
	),20);
	HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
	//Convertir el valor obtenido por Tera Term de tipo carácter a
	tipo entero
	intFinalGrados=atoi((char*)mensFinalGrados);


	//Función para limitar el valor máximo que se puede introducir.
	En el caso que se introduzca un valor incorrecto, se indica al
	usuario y le pide que introduzca uno nuevo.
	while(intFinalGrados>360)
	{
	HAL_UART_Transmit(&huart2,mensError,sizeof(mensError),20);
	HAL_UART_Transmit(&huart2,mensPosFinal,sizeof(mensPosFinal),20);
	HAL_UART_Receive(&huart2,mensFinalGrados,sizeof(mensFinalGrados)
	,1000000);
	HAL_UART_Transmit(&huart2,mensFinalGrados,sizeof(mensFinalGrados
	),20);
	HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
	intFinalGrados=atoi((char*)mensFinalGrados);

	}

}

//Guardar datos en las variables globales
vrefGrados=intVrefGrados;
arefGrados=intArefGrados;
precision=intPrecision;
```

```
        //Convertir grados a rad
        inicio=posActualRad();
        final= intFinalGrados*(3.14159f/180);
        vref=vrefGrados*(3.14159f/180);
        aref=arefGrados*(3.14159f/180);

        //Activar flag para iniciar movimiento (habilita timer 100us)
        flagIniciarControl=1;
        HAL_Delay(1000);
}
```

Una vez realizado el código principal, se procederá a explicar el bucle de interrupción de 1 ms.

5.2.2.3 Función trapezoidal

Dado que el control del motor se realiza a partir del seguimiento de una trayectoria, antes de realizar el control, hay que construir una función que la represente correctamente. Existen varios tipos de trayectorias que se pueden emplear en el campo de la robótica. Estas son, la trayectoria Trapezoidal, Perfil en "S", Perfil en "S" Parcial y Polinomial. Sus principales ventajas son:

- **Trapezoidal:** La más rápida y simple.
- **Perfil en "S":** Suaviza los cambios en la aceleración.
- **Perfil en "S" Parcial:** Suaviza los cambios en la aceleración y velocidad.
- **Polinomial:** Permite una mayor flexibilidad para establecer condiciones de entorno.

Dado que la más sencilla de implementar es la trapezoidal, se ha optado por esta trayectoria.

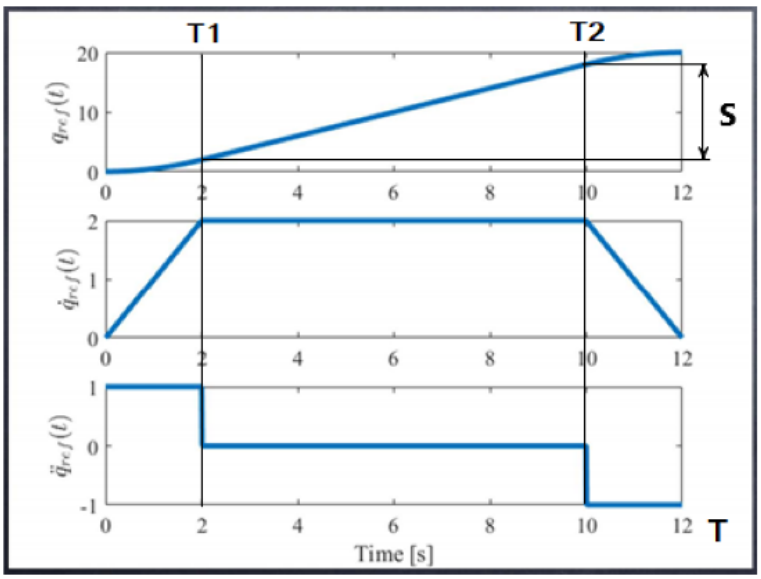


Figura 25. Estructura de trayectoria trapezoidal.

Fuente: https://drive.google.com/file/d/1htDgq-212EZLyZI0YrHEQbfl8bCLGy_L/view?usp=sharing

Donde:

q: Posición.

$\dot{q}$: Velocidad.

$\ddot{q}$: Aceleración.

Una vez elegido la trayectoria objetivo, se procederá a desarrollar la función que calcule la posición adecuada a partir del tiempo. Para poder describir la trayectoria correctamente, se necesita calcular sus valores característicos. Estos valores son T1, T2 y S, y se pueden calcular mediante las siguientes ecuaciones:

$$T1 = \frac{v_{max}}{a_{max}}$$

$$S = q_{final} - q_{inicial} - \frac{v_{max}^2}{a_{max}}$$

$$T2 = \frac{S}{v_{max}} + T1$$

Donde:

v_{max}: Velocidad consigna máxima.

a_{max}: Aceleración consigna máxima.

q_{final}: Posición inicial.

q_{final}: Posición final.

La función desarrollada necesita como valores de entrada, el valor de la posición inicial del motor en radianes, la posición final en radianes, la velocidad en radianes/s, la aceleración en radianes/s2 y el instante de tiempo actual. Como resultado devuelve el valor de la posición y velocidad del motor en el instante actual. La función primero calcula los componentes s, T1 y T2 a partir de las ecuaciones anteriores. Una vez obtenidos, se calcula la posición en la que se encontrará el motor para el final de cada subperiodo T1 y T2. Una vez calculados los límites de la función, si se aplican las ecuaciones de cinemática, se puede simular cada tramo de la trayectoria.

Ecuaciones empleadas para cada tramo:

Para el tramo de tiempo=0 hasta T1, se ha empleado las ecuaciones del movimiento rectilíneo uniformemente acelerado (MRUA).

$$pos_{final} = pos_{inicial} + v_{inicial} * t + \frac{1}{2} * a_{const} * t^2$$

$$v_{final} = v_{inicial} * a_{const} * t$$

Para el tramo de T1 hasta T2, se ha empleado la ecuación del movimiento rectilíneo uniforme (MRU).

$$pos_{final} = pos_{inicial} * v_{const} * t$$

Para el último tramo, de T2 hasta el final T, se emplean también las ecuaciones de MRUA, pero con aceleración negativa.

Un último factor para tener en cuenta es la posibilidad de realizar movimientos en los que la posición final es menor que la inicial. Para estos casos, los valores de a_{const} y v_{const} se vuelven negativos, ya que se invierten los tramos. Además, para calcular el valor de S, se deben modificar los valores de la posición inicial y la final como se muestra a continuación:

$$S = q_{inicial} - q_{final} - \frac{v_{max}^2}{a_{max}}$$

A continuación, se muestra el código descrito de la función trapezoidal completa.

```
//Función para realizar la señal trapezoidal de velocidad que seguirá el
motor
//Se introducen el valor de posición inicial (rad), final (rad), velocidad
máxima de referencia (rad/s), aceleración máxima de referencia (rad/s2) y el
instante de tiempo actual.
//La función devuelve el valor de la posición y de la velocidad de tipo
float.
void trapezoide(float inicio, float final,float vref, float aref, float
tiempo,float *ref,float *vel)
{
	double s;
	float T1,T2;
	float acel;
	float posFinalT1, posFinalT2;

	//Calcular valores característicos de los periodos T1,T2,T y la
posición en cada instante
	T1=vref/aref;
	if(final>=inicio)  //la posición final es superior que la inicial
	{
		s=final-inicio-((vref*vref)/aref);
		T2=(s/vref)+T1;
		T=T1+T2;
		posFinalT1=inicio+0.5f*aref*T1*T1;
		posFinalT2=posFinalT1+vref*(T2-T1);

	}
	else  //La posición inicial es mayor que la final
	{
		s=inicio-final-((vref*vref)/aref);
		T2=(s/vref)+T1;
		T=T1+T2;
		posFinalT1=inicio-0.5f*aref*T1*T1;
		posFinalT2=posFinalT1-vref*(T2-T1);
	}

	//Definir cada tramo de la onda trapezoidal
	if(tiempo<T1)	//Aceleración constante MRUA
	{
		if(final>=inicio)
		{
			acel=aref;
		}
		else
		{
```

```
                acel=-aref;
            }
            *vel=acel*tiempo;   //v=v(inicial)+acel*tiempo
            *ref=inicio+0.5f*acel*tiempo*tiempo;
        //ref=ref(inicial)+v(inicial)*tiempo+0.5*acel*t2
        }
        else if(tiempo>=T1 && tiempo <T2) //Velocidad constante MRU
        {
            acel=0;
            if(final>=inicio)
            {
                *vel=vref;
            }
            else
            {
                *vel=-vref;
            }
            *ref=posFinalT1+*vel*(tiempo-T1);
        }
        else if (tiempo>=T2 && tiempo<T)//Deceleración constante MRUA
        {
            if(final>=inicio)
            {
                acel=-aref;
                *vel=vref+acel*(tiempo-T2);
            }
            else
            {
                acel=aref;
                *vel=-vref+acel*(tiempo-T2);
            }
            //vel=vref+acel*tiempo;
            *ref=posFinalT2+*vel*(tiempo-T2)+0.5f*acel*(tiempo-T2)*(tiempo-
T2);
        }
        else   //Paro del motor
        {
            acel=0;
            *vel=0;
            *ref=final;
        }
}
```

Las variables que llevan el símbolo "*" delante del nombre son las variables que se devuelven como resultado.

5.2.2.4 Interrupción del timer

Una vez construido el código de la generación de la trayectoria a seguir, se debe realizar el bucle de control integrado en el timer para poder calcular el error y enviar la acción de control correspondiente.

El código que se ejecutará se encuentra en una función llamada "Callback", esta es la manera que tiene el programa uKeil de administrar y ejecutar el código de las interrupciones. Dado que esta función se dispara siempre que alguna interrupción de timer se ejecuta, hay que diferenciar

cual es la interrupción de timer accionada. Para ello se ha introducido la condición "htim->Instance == TIM6", que ejecuta el código dentro de ella si la interrupción disparada proviene del timer número 6. Para este caso, dado que solamente se ha empleado un timer, no haría falta emplearlo. Pero ya que en un futuro se puede modificar el programa e introducir nuevos componentes, es adecuado introducirlo.

Una vez dentro de la función, el código solamente se ejecutará si la variable global "flagIniciarControl" está activada. Dentro de dicha función, se han inicializado las variables locales y se ha programado el código tanto para una precisión de 100us como para 1ms. Debido a que ambas funciones son equivalentes, solamente se explicará el bucle de 1ms.

Teniendo en cuenta que 1ms equivale a 10 veces 100us, se puede programar el bucle para que ejecute el código cada 10 ciclos. Dentro del bucle, se calcula el tiempo actual y se obtiene la posición y velocidad de referencia a partir de la función "trapezoide()", explicada anteriormente. Luego, se obtienen la posición y velocidad real del motor y se calculan los errores de posición y velocidad. Una vez obtenidos, se calcula la acción de control a partir de los errores y las constantes proporcional y derivativa, y se delimita entre los valores 85 y 0 por seguridad.

Cuando se llega a la posición objetivo o se han realizado los suficientes ciclos necesarios para llegar a dicha posición, se para el motor y se reconfiguran los valores de las variables globales de control. Al hacerlo, se vuelve al bucle principal donde se vuelven a pedir nuevos datos.

En el caso que alguno de los finales de carrera sea accionado, se realizará una parada de forma escalonada y se saldrá del bucle del timer. A continuación, se muestra el código para la precisión de 1ms solamente.

```
//Callback del timer de control de 100us y 1ms
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	if(htim->Instance == TIM6)
	{
		if(flagIniciarControl==1)
		{
			float u,tiempo;//Accion de control y variable del tiempo
actual
			uint8_t i=0; //Variable auxiliar
			float Kp1=550;	//Constante proporcional bucle 1ms
			float Kv1=45;	//Constante derivativa bucle 1ms
			float Kp2=120;	//Constante proporcional bucle 100us
			float Kv2=50;	//Constante derivativa bucle 100us

			//Bucle precision de 1ms
			//Al ser 1ms = 10*100us, solamente se puede realizar este
			//control cada 10 ciclos
			if(precision==1&&cuenta%10==0)
			{
			//Convertir el tiempo al valor real
			tiempo=tiempoPrec1*0.001f;
			trapezoide(inicio,final,vref, aref,
		tiempo,&posicionCalculada,&velocidadCalculada);
			posicionActual=posActualRad();  //Obtener la posición
		actual en rad

			//Calcular la velocidad a partir la posición
			//anterior y el tiempo transcurrido
```

```
		velocidadActual=(posicionActual-
posicionAnterior)/0.001f;


	//Si la posición final es mayor que la inicial,
	//realiza el giro en sentido horario
	if(inicio<=final){
		giroHorario();

		//Calcular el error de posición y velocidad
		//para giro horario

	errorPos=(float)posicionCalculada-(float)posicionActual;

	errorVel=(float)velocidadCalculada-(float)velocidadActual;
	}
	//Si la final es menor, realiza el giro antihorario
	else if(final<inicio){
		giroAntihorario();
		//Calcular el error de posición y velocidad
		//para giro antihorario
		errorPos=(float)posicionActual-
(float)posicionCalculada;
		errorVel=(float)velocidadActual-
(float)velocidadCalculada;
	}

	//Calcula la acción de control a partir del error
	//de posición y velocidad.

	u=Kp1*(float)errorPos+Kv1*(float)errorVel;
	u=(int)u;
	//Limitación de la acción de control máxima a señal
	//PWM de 85%
	if(u>=85)
	{
		u=85;
	}
	//Limitación de la acción de control para que no
	//acceda a los valores negativos
	else if(u<=0)
	{
		u=0;
	}
	//Envía la señal de la acción de control por PWM
	htim1.Instance-> CCR3 = u;
	//Final del bucle de control si se ha llegado a la
	//posición deseada o el tiempo de espera máximo de
	//espera ha prescindido (1 segundo)
	if(posicionActual==final ||tiempo>=T+1)
	{
	//Acción de control = 0 (Velocidad = 0)
		htim1.Instance-> CCR3 = 0;
		//Reinicio de las variables de control para
		//salir del timer y volver a pedir la nueva
```

```
                //posición por tera term
                flagIniciarControl=0;
                tiempoPrec1=0;
                cuenta=0;
                paro();
        }
        //Salida del bucle cuando se pulse alguno de los
        //finales de carrera y paro del motor
        if(finCarreraIzq==1||finCarreraDer==1)
        {
        //Bucle while para parar el motor de forma
        //escalonada
                while(u>0)
                {
                //Realizar un retraso sin usar la
                //función HAL_Delay()
                        while(i<10)
                        {
                                htim1.Instance-> CCR3 = u;
                                i++;
                        }
                        //Reducir el valor de la acción de
                        //control para reducir velocidad hasta
                        //que se pare
                        i=0;
                        if(u>0)
                        {
                                u--;
                        }
                        else if(u<=0)
                        {
                                u=0;
                        }
                }
                htim1.Instance-> CCR3 = 0;
                //Reiniciar los valores que controlan el
                //bucle y volver al bucle main principal
                flagIniciarControl=0;
                tiempoPrec1=0;
                cuenta=0;
                paro();
        }
        //Actualizar la posición anterior
        posicionAnterior=posicionActual;

        //Incrementar el instante de simulación
        ++tiempoPrec1;
}

…
```

5.3 Resultados

Para poder representar los resultados junto a la señal trapezoidal de referencia, se han guardado todos los valores dentro de dos variables globales y una vez acabado el movimiento del motor, se han enviado por tera term. Una vez enviados, se han copiado en dos archivos de texto tipo .txt y se han abierto en Matlab para poder representarlos. Hay que tener en cuenta que, para no cargar con matrices muy extensas, se ha decidido obtener muestras de la posición real y calculada del motor cada 20 ciclos y solamente para el bucle de control de 1 ms. El código empleado para obtener los datos se ha introducido a posteriori de realizar el código del timer. Así que dicho código quedará reflejado en el apartado "Anexos" junto con el código completo de forma comentada. Asimismo, el código empleado en Matlab ha sido el siguiente:

```
%Abrir el archivo que contiene los resultados reales del motor
%y guardarlos en la matriz A
fileID = fopen('Valoresresultado55045.txt','r');
formatSpec = '%f';
A = fscanf(fileID,formatSpec);
fclose(fileID);
%Construir la matriz del tiempo. Cada muestra se recoge cada 20 ms.
B=1:234;
B=B';
B=B*0.02;

%Abrir el archivo que contiene los resultados de la función
%trapezoidal del motor y guardarlos en la matriz C
fileID = fopen('Valoresresultadoref.txt','r');
formatSpec = '%f';
C = fscanf(fileID,formatSpec);
fclose(fileID);

%Representar las funciones en una misma gráfica
figure;
plot(B,A,'r');
hold on;
plot(B,C,'b');
title('Título');
xlabel('t(s)');
ylabel('rad');
legend('Pos Real','Pos Referencia')
```

Una vez creado el programa, se han realizado varios experimentos para observar el peso que tienen los valores de las constantes Kp y Kv dentro del seguimiento de la trayectoria.

Primero se ha realizado un control proporcional solamente. Como se pude observar, el seguimiento no es constante y existen escalones debido a altas acciones de control por intervalos.

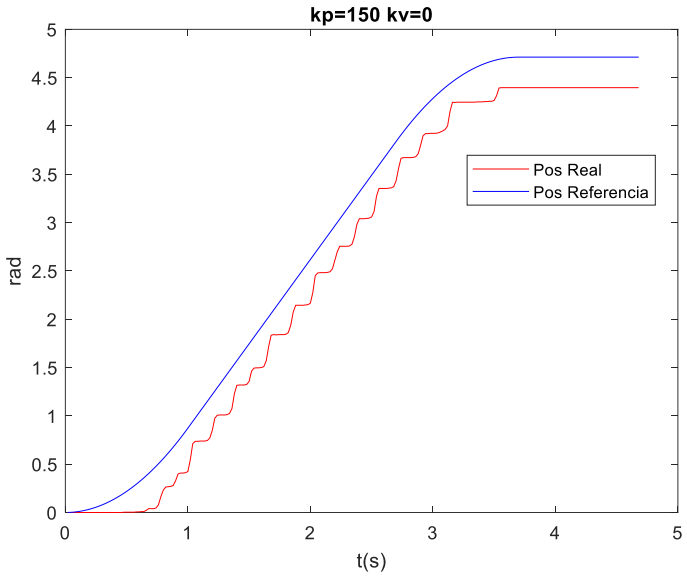


Figura 26. Control proporcional con Kp=150.

Fuente: Propia.

Después, se ha introducido la acción derivativa, haciendo desaparecer los escalones del control anterior.

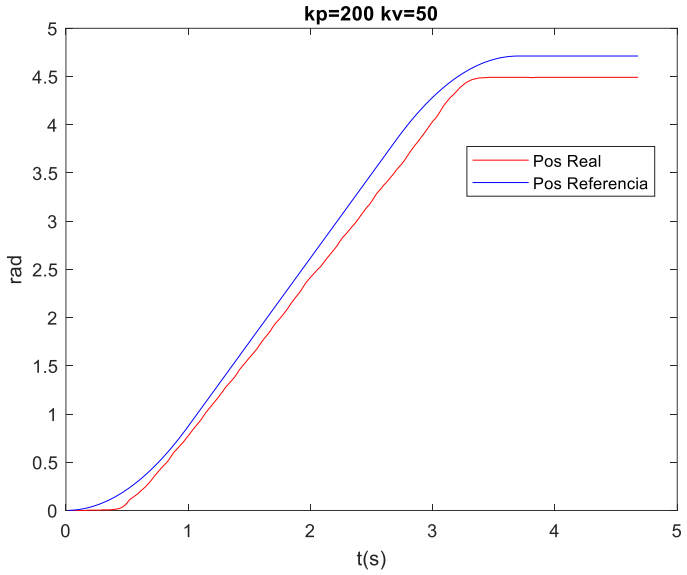


Figura 27. Control proporcional-derivativo con Kp=200 y Kv=50.

Fuente: Propia

En el caso que la acción derivativa supere cierto umbral, el error de posición es igual a 0 pero la posición del motor se adelanta al seguimiento de la trayectoria.

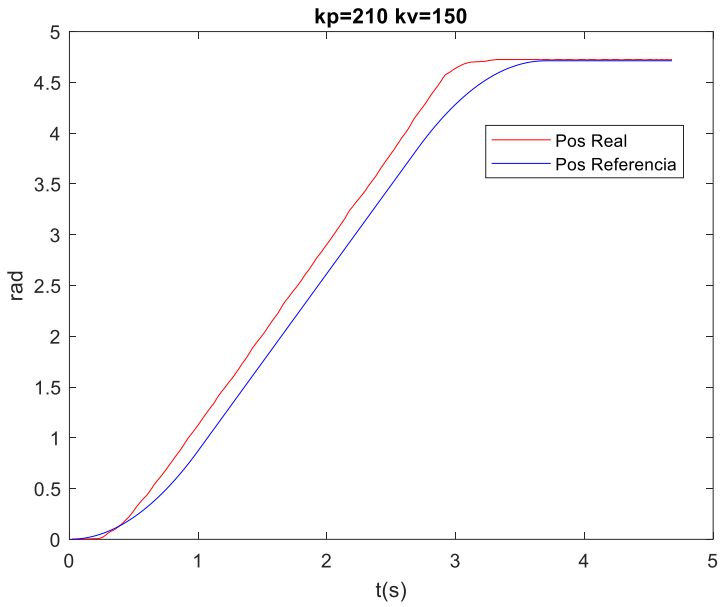


Figura 28. Control proporcional-derivativo con Kp = 210 y Kv=150.

Fuente: Propia.

A través de varios intentos, se ha obtenido la función que más se adecua al seguimiento de la trayectoria y que posee menor error de posición.

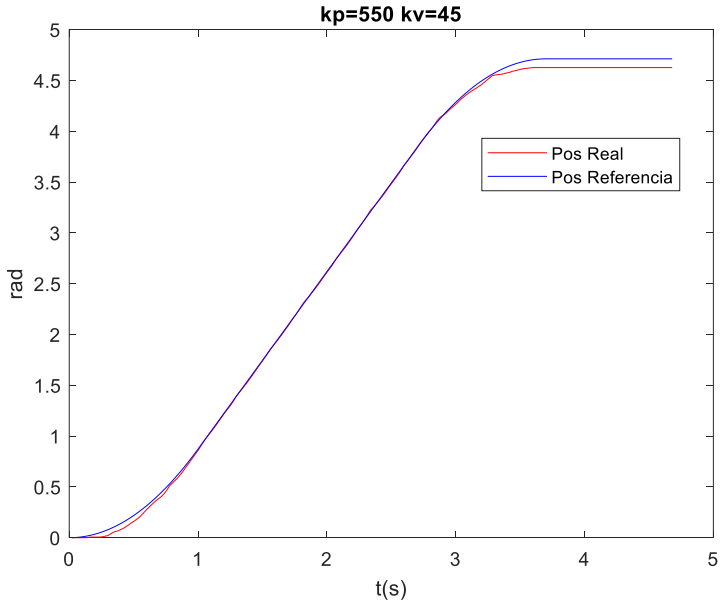


Figura 29. Control proporcional-derivativo con Kp=550 y Kv=45.

Fuente: Propia.

6. Conclusión

A lo largo de la realización del proyecto, se han puesto en práctica varios conocimientos aprendidos a lo largo del grado universitario. Además, se ha aprendido a diseñar y elegir correctamente los componentes electrónicos necesarios para cumplir con los requisitos técnicos del proyecto.

Al haber realizado un estudio previo de todos los componentes necesarios, se han podido conseguir los objetivos sin ningún tipo de contratiempo, por ejemplo, quemar componentes por errores de conexión o que no soportaran los valores de tensión o corriente adecuados.

Otro aspecto a tener en cuenta es la importancia del equipamiento disponible. Ya que se pueden evitar bastantes problemas y reducir el tiempo en implementar la solución si se poseen los materiales necesarios, por ejemplo, la falta de un osciloscopio.

En cuanto al tema del proyecto, se han podido obtener conocimientos más profundos en cuanto al funcionamiento y control de un motor eléctrico, al igual que las problemáticas que existen para controlarlo y ponerlo en funcionamiento. También se han ampliado los conocimientos para programar una tarjeta basada en STM32 y emplear de forma más optimizada sus recursos.

En conclusión, se pude decir que el desarrollo de este proyecto ha salido correctamente. En un futuro, se debería poder realizar un control por corriente al introducir un sensor de corriente y se debería modificar el bucle de control para tener en consideración los pesos de los eslabones que conformarían un brazo robótico.

7. Bibliografía

Formación de una señal PWM en tarjetas STM32:

https://www.st.com/resource/en/application_note/dm00513079-generating-pwm-signals-using-stm8-nucleo64-boards-stmicroelectronics.pdf

Funcionamiento de un timer genérico de tarjeta STM32:

https://www.st.com/resource/en/application_note/dm00236305-general-purpose-timer-cookbook-for-stm32-microcontrollers-stmicroelectronics.pdf

Referencias keil 5 uVision:

https://www.keil.com/pack/doc/CMSIS/DSP/html/modules.html

Práctica 4 de sistemas robotizados:

https://drive.google.com/file/d/1htDgq-212EZLyZI0YrHEQbfl8bCLGy_L/view?usp=sharing

Datasheets

Encoder:

https://www.maxongroup.com/medias/sys_master/root/8846495678494/20-ES-461-462.pdf

Transistor:

https://pdf1.alldatasheet.com/datasheet-pdf/view/52961/FAIRCHILD/IRF540.html

Fotoacoplador:

https://pdf1.alldatasheet.com/datasheet-pdf/view/43371/SHARP/PC817.html

Motor:

https://www.maxongroup.com/maxon/view/content/Overview-brushed-DC-motors

Driver del transistor:

https://pdf1.alldatasheet.com/datasheet-pdf/view/239016/MICREL/MIC5014.html

Differential line driver:

https://www.ti.com/lit/ds/symlink/sn65175.pdf?HQS=TI-null-null-alldatasheets-df-pf-SEP-wwe&ts=1623783479502&ref_url=https%253A%252F%252Fwww.alldatasheet.com%252F

8. Anexos

Código empleado:

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <math.h>
#include "stdlib.h"
#include "stdio.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim1;
```

```
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim6;

UART_HandleTypeDef huart2;
DMA_HandleTypeDef hdma_usart2_rx;

/* USER CODE BEGIN PV */

/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM2_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM6_Init(void);
/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
//Declaración de variables globales
static uint8_t flagIniciarControl = 0; //Flag empleada para activar el inicio de control del motor
static uint8_t flagHoming=0;    //Flag empleada en el main para activar/desactivar el protocolo de homing

//Variables globales del callback del timer de control de 100us y 1ms
static uint16_t cuenta = 0;            //Contador para obtener valor de la iteración actual en control del motor
static uint16_t tiempoPrec1=0;
static uint16_t tiempoPrec2=0;
static float inicio,final,vref,aref,errorPos,errorVel;    //Definir las variables empleadas
static uint8_t precision; //Variable para elegir si activar el control de 100us o 1ms
static float posicionCalculada,posicionActual,posicionAnterior;
static float velocidadCalculada,velocidadActual;
static float T;

//Decomentar estas tres líneas de código para poder obtener datos por Matlab
//static float TotalPosiciones[1000];  //Variable empleada para la obtención de datos
//static float TotalPosicionesRef[1000];     //Variable empleada para la obtención de datos
//static uint8_t j=0;     //Variable empleada para la obtención de datos

//Variables globales del callback de las interrupciones externas
static uint8_t indice = 0;
static uint8_t finCarreraIzq=0;
static uint8_t finCarreraDer=0;
static uint8_t etapaPotenciaOn=0;
static uint8_t mensEncendido[15]="Pot encendida";
static uint8_t mensApagado[15]="Pot apagada";
```

```
//Función que lee el valor de la posición del encoder actual y la devuelve en
rad
float posActualRad()
{
	uint16_t posicion;
	float posicionRad;
	posicion= TIM2->CNT;//La resolución del encoder es de 4x512 = 2048
pulsos por vuelta
	posicionRad=(double)posicion*(((double)3.14159*2)/(double)2048);
	//Obtener la posición en radianes a partir del número de pulsos
	return posicionRad;

}

//Funciones para configurar el giro del motor
void giroHorario()
{
	HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_RESET); //La señal se
vuelve 0
	HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_SET);  //La señal se vuele
1
}
void giroAntihorario()
{
	HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_SET);
}
void paro()
{
	HAL_GPIO_WritePin(GPIOB,GPIO_PIN_5, GPIO_PIN_RESET);
	HAL_GPIO_WritePin(GPIOA,GPIO_PIN_7, GPIO_PIN_RESET);
}

//Función para realizar la señal trapezoidal de velocidad que seguirá el
motor
//Se introducen el valor de posición inicial (rad),final (rad),velocidad
máxima de referencia (rad/s),
//aceleración máxima de referencia (rad/s2) y el instante de tiempo actual.
//La función devuelve el valor de la posición de tipo float.
void trapezoide(float inicio, float final,float vref, float aref, float
tiempo,float *ref,float *vel)
{
	double s;
	float T1,T2;
	float acel;
	float posFinalT1, posFinalT2;

	//Calcular valores característicos de los periodos T1,T2,T y la
posición en cada instante
	T1=vref/aref;
	if(final>=inicio)  //la posición final es superior que la inicial
	{
		s=final-inicio-((vref*vref)/aref);
		T2=(s/vref)+T1;
		T=T1+T2;
		posFinalT1=inicio+0.5f*aref*T1*T1;
		posFinalT2=posFinalT1+vref*(T2-T1);
```

```
	}
	else  //La posición inicial es mayor que la final
	{
		s=inicio-final-((vref*vref)/aref);
		T2=(s/vref)+T1;
		T=T1+T2;
		posFinalT1=inicio-0.5f*aref*T1*T1;
		posFinalT2=posFinalT1-vref*(T2-T1);
	}

	//Definir cada tramo de la onda trapezoidal
	if(tiempo<T1)	//Aceleración constante
	{
		if(final>=inicio)
		{
			acel=aref;
		}
		else
		{
			acel=-aref;
		}
		*vel=acel*tiempo;  //v=v(inicial)+acel*tiempo
		*ref=inicio+0.5f*acel*tiempo*tiempo;
	//ref=ref(inicial)+v(inicial)*tiempo+0.5*acel*t2
	}
	else if(tiempo>=T1 && tiempo <T2) //Velocidad constante
	{
		acel=0;
		if(final>=inicio)
		{
			*vel=vref;
		}
		else
		{
			*vel=-vref;
		}
		*ref=posFinalT1+*vel*(tiempo-T1);
	}
	else if (tiempo>=T2 && tiempo<T)//Deceleración constante
	{
		if(final>=inicio)
		{
			acel=-aref;
			*vel=vref+acel*(tiempo-T2);
			*ref=posFinalT2+vref*(tiempo-T2)+0.5f*acel*(tiempo-
T2)*(tiempo-T2);
		}
		else
		{
			acel=aref;
			*vel=-vref+acel*(tiempo-T2);
			*ref=posFinalT2-vref*(tiempo-T2)+0.5f*acel*(tiempo-
T2)*(tiempo-T2);
		}
		//vel=vref+acel*tiempo;

	}
	else if(tiempo>=T) //Paro del motor
	{
```

```
			acel=0;
			*vel=0;
			*ref=final;
		}
}



//Callback del timer de control de 100us y 1ms
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
	if(htim->Instance == TIM6)
	{
		if(flagIniciarControl==1)
		{
			float u,tiempo;//Accion de control y variable del tiempo
actual
			uint8_t i=0; //Variable auxiliar
			float Kp1=550;	//Constante proporcional bucle 1ms
			float Kv1=45;	//Constante derivativa bucle 1ms
			float Kp2=120;	//Constante proporcional bucle 100us
			float Kv2=50;	//Constante derivativa bucle 100us

			//Bucle precision de 1ms
			//Al ser 1ms = 10*100us, solamente se puede realizar este
control cada 10 ciclos
			if(precision==1&&cuenta%10==0)
			{
				//Convertir el tiempo al valor real
				tiempo=tiempoPrec1*0.001f;
				trapezoide(inicio,final,vref, aref,
tiempo,&posicionCalculada,&velocidadCalculada);
				posicionActual=posActualRad();  //Obtener la
posición actual en rad
				//Calcular la velocidad a partir de la posición
				//anterior y el tiempo transcurrido
				velocidadActual=(posicionActual-
posicionAnterior)/0.001f;

				//Si la posición final es mayor que la inicial,
				//realiza el giro en sentido horario
				if(inicio<=final){
					giroHorario();

					//Calcular el error de posición y velocidad
					//para giro horario
					errorPos=(float)posicionCalculada-
(float)posicionActual;
					errorVel=(float)velocidadCalculada-
(float)velocidadActual;
				}
				//Si la final es menor, realiza el giro antihorario
				else if(final<inicio){
					giroAntihorario();

					//Calcular el error de posición y velocidad
					para giro antihorario
					errorPos=(float)posicionActual-
(float)posicionCalculada;
```

```
                        errorVel=(float)velocidadActual-
(float)velocidadCalculada;
                }

                //Calcula la acción de control a partir del error
                //de posición y velocidad.
                u=Kp1*(float)errorPos+Kv1*(float)errorVel;
                u=(int)u;
                //Limitación de la acción de control máxima a señal
                //PWM de 85%
                if(u>=85)
                {
                        u=85;
                }
                //Limitación de la acción de control para que no
                //acceda a los valores negativos
                else if(u<=0)
                {
                        u=0;
                }
                //Envía la señal de la acción de control por PWM
                htim1.Instance-> CCR3 = u;
                //Final del bucle de control si se ha llegado a la
                //posición deseada o el tiempo de espera máximo de
                //espera ha prescindido (1 segundo)
                if(posicionActual==final ||tiempo>=T+1)
                {
                        //Acción de control = 0 (Velocidad = 0)
                        htim1.Instance-> CCR3 = 0;
                        //Reinicio de las variables de control para
                        //salir del timer y volver a pedir la nueva
                        //posición por tera term
                        flagIniciarControl=0;
                        tiempoPrec1=0;
                        cuenta=0;
                        //j=0; //Decomentar si se requiere obtener
                        //datos por teraterm
                        paro();
                }
                //Salida del bucle cuando se pulse alguno de los
                //finales de carrera y paro del motor
                if(finCarreraIzq==1||finCarreraDer==1)
                {
                        //Bucle while para parar el motor de forma
                        //escalonada
                        while(u>0)
                        {
                                //Realizar un retraso sin usar la
                                //función HAL_Delay()
                                while(i<10)
                                {
                                        htim1.Instance-> CCR3 = u;
                                        i++;
                                }
                                //Reducir el valor de la acción de
                                //control para reducir velocidad hasta
                                //que se pare
                                i=0;
                                if(u>0)
```

```
                                        {
                                        u--;
                                        }
                                        else if(u<=0)
                                        {
                                              u=0;
                                        }
                                  }
                                  htim1.Instance-> CCR3 = 0;
                                  //Reiniciar los valores que controlan el
                                  //bucle y volver al bucle main principal
                                  flagIniciarControl=0;
                                  tiempoPrec1=0;
                                  cuenta=0;
                                  //j=0; //Decomentar si se requiere obtener
                                  //datos por teraterm
                                  paro();
                            }
                            //Actualizar la posición anterior
                            posicionAnterior=posicionActual;
                            //Bloque empleado para obtener las posiciones en
                            //cada instante
                            //Decomentar si se requiere obtener datos por
                            //Teraterm
                            //if (tiempoPrec1%20==0){
                            //    TotalPosiciones[j]=posicionActual;
                            //    TotalPosicionesRef[j]=posicionCalculada;
                            //    j++;
                            //}
                            //Incrementar el instante de simulación
                            ++tiempoPrec1;
                      }

                      else if(precision==2)
                      {
                            tiempo=tiempoPrec2*0.0001f;

                            trapezoide(inicio,final,vref, aref,
tiempo,&posicionCalculada,&velocidadCalculada);
                            posicionActual=posActualRad();

                            velocidadActual=(posicionActual-
posicionAnterior)/0.0001f;
                            if(inicio<=final){
                                  giroHorario();

                                  errorPos=(float)posicionCalculada-
(float)posicionActual;
                                  errorVel=(float)velocidadCalculada-
(float)velocidadActual;
                            }
                            else if(final<inicio){
                                  giroAntihorario();

                                  errorPos=(float)posicionActual-
(float)posicionCalculada;
                                  errorVel=(float)velocidadActual-
(float)velocidadCalculada;
                            }
```

```
                        u=Kp2*errorPos+Kv2*errorVel;
                        if(u>=85)
                        {
                              u=85;
                        }
                        else if(u<=0)
                        {
                              u=0;
                        }

                        htim1.Instance-> CCR3 = u;
                        if(finCarreraIzq==1||finCarreraDer==1)
                        {
                              while(u>0)
                              {
                                    while(i<10)
                                    {
                                          htim1.Instance-> CCR3 = u;
                                          i++;
                                    }
                                    i=0;
                                    if(u>0)
                                    {
                                    u--;
                                    }
                                    else if(u<=0)
                                    {
                                          u=0;
                                    }

                              }
                              htim1.Instance-> CCR3 = 0;
                              flagIniciarControl=0;
                              tiempoPrec2=0;
                              cuenta=0;
                              paro();
                        }
                        if(posicionActual==final ||tiempo>=T+1)
                        {
                              htim1.Instance-> CCR3 = 0;
                              flagIniciarControl=0;
                              tiempoPrec2=0;
                              cuenta=0;
                              paro();
                        }
                        posicionAnterior=posicionActual;
                        ++tiempoPrec2;
                  }
            cuenta++;
            }
      }
}


//Callback de las interrupciones externas
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
      //Interrupción de la señal índice del encoder
      if(GPIO_Pin==GPIO_PIN_7)
```

```
	{
		if(flagHoming==0)
		{
			indice++;
		}
	}
	//Interrupción de la señal de entrada final de carrera izquierdo (PB6)
	else if(GPIO_Pin==GPIO_PIN_6)
	{
		finCarreraIzq=1;
	}
	//Interrupción de la señal de entrada final de carrera derecho (PB4)
	else if(GPIO_Pin==GPIO_PIN_4)
	{
		finCarreraDer=1;
	}
	//Interrupción de la señal que mide si está encendida la etapa de
	potencia
	else if(GPIO_Pin==GPIO_PIN_0)
	{
		int i=0;
		if (etapaPotenciaOn==0)
		{
			etapaPotenciaOn=1;

	HAL_UART_Transmit(&huart2,mensEncendido,sizeof(mensEncendido),20);
			//Delay para evitar que el posible ruido de los
			contactores pueda enviar varias señales
			while(i<=3000000)
			{
				i++;
			}

		}
		else if (etapaPotenciaOn==1)
		{
			etapaPotenciaOn=0;

	HAL_UART_Transmit(&huart2,mensApagado,sizeof(mensApagado),20);
			//Delay para evitar que el posible ruido de los
			contactores pueda enviar varias señales
			while(i<=300000)
			{
				i++;
			}

		}
	}
}

//Función que mueve el motor a velocidad reducida continua para llegar a la
posición índice
void motorHoming()
{
	//Inicializar variables
	uint8_t posicionActualHoming,posicionPreviaHoming=0;
	uint8_t vel=0;

	giroHorario();
```

```
	//Mientras no detecte la señal indice mover el motor en sentido
horario
	while(indice==0)
	{
		//Obtener la posición actual
		posicionActualHoming=TIM2->CNT;
		//Si la posición actual es igual a la anterior, incrementar
velocidad
		//hasta llegar a 65
		if(posicionActualHoming==posicionPreviaHoming)
		{
			if(vel>=65)
			{
				vel=55;
			}
			else
			{
				vel++;
			}
		}
		//Si no es igual, reducir velocidad
		else if(posicionActualHoming!=posicionPreviaHoming)
		{
			if(vel<=0)
			{
				vel=0;
			}
			else
			{
				vel=vel-5;
			}
		}
		//Enviar seña pwm con la velocidad adecuada
		htim1.Instance-> CCR3 = vel;
		//Velocidad anterior es igual a la actual
		posicionPreviaHoming=posicionActualHoming;
	}
	//Parar el motor y esperar
	htim1.Instance-> CCR3 = 0;
	HAL_Delay(1000);

	giroAntihorario();
	//Volver a girar el motor en sentido antihorario para encontrar de
nuevo la posición índice
	while(indice==1)
	{
		posicionActualHoming=TIM2->CNT;
		if(posicionActualHoming==posicionPreviaHoming)
		{
			if(vel>=70)
			{
				vel=55;
			}
			else
			{
				vel++;
			}
		}
		else if(posicionActualHoming!=posicionPreviaHoming)
```

```
		{
			if(vel<=0)
			{
				vel=0;
			}
			else
			{
				vel=vel-5;
			}
		}
		htim1.Instance-> CCR3 = vel;
		posicionPreviaHoming=posicionActualHoming;
	}

	htim1.Instance-> CCR3 = 0;
	giroHorario();
	HAL_Delay(1000);
	//Realizar el último giro horario a velocidad reducida hasta llegar a
la posición índice
	while(indice==2)
	{
		posicionActualHoming=TIM2->CNT;
		if(posicionActualHoming==posicionPreviaHoming)
		{
			if(vel>=65)
			{
				vel=55;
			}
			else
			{
				vel++;
			}
		}
		else if(posicionActualHoming!=posicionPreviaHoming)
		{
			if(vel<=0)
			{
				vel=0;
			}
			else
			{
				vel=vel-5;
			}
		}
		htim1.Instance-> CCR3 = vel;
		posicionPreviaHoming=posicionActualHoming;
	}

	htim1.Instance-> CCR3=0;
	paro();
	HAL_Delay(100);
	//Resetear el valor inicial del encoder para que coincida la posición
inicial con la posición de índice.
	htim2.Instance->CNT=0;
	//Dar el valor de 1 a flagHoming para no volver a realizar el homing
	flagHoming=1;
}
```

```
/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------
-*/

  /* Reset of all peripherals, Initializes the Flash interface and the
Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */

  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */

  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_USART2_UART_Init();
  MX_TIM2_Init();
  MX_TIM1_Init();
  MX_TIM6_Init();
  /* USER CODE BEGIN 2 */
  //Inicialización del encoder, señal PWM y timer de 100uS
  HAL_TIM_Encoder_Start(&htim2,TIM_CHANNEL_ALL);
  HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_3);
  HAL_TIM_Base_Start_IT(&htim6);

//Variables para guardar los valores recibidos por teraterm
uint8_t mensFinalGrados[3], mensVrefGrados[3], mensArefGrados[3],
mensPrecision[1];
//Valores recibidos por teraterm de tipo enteros
int intFinalGrados,intVrefGrados,intArefGrados,intPrecision;
float vrefGrados,arefGrados;

//Mensajes para teraterm
uint8_t mensPosFinal [15]=" Pos(360 max): ";
uint8_t mensVel [15]=" Vel(100 max): ";
uint8_t mensAcel [16]=" Acel(150 max): ";
uint8_t mensPrec [30]="Precision(1=1ms o 2=100us): ";
uint8_t espacio[2]=" ";
uint8_t mensError[15]="Valor erroneo";
uint8_t mensFinDer[30]="Interruptor derecho accionado!";
uint8_t mensFinIzq[32]="Interruptor izquierdo accionado!";
```

```
char buffer[12] = {'\0'};

/* USER CODE END 2 */

/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
//Esperar hasta que se detecte la alimentación de la etápa de potencia
      if(etapaPotenciaOn==1)
      {
      //Homing motor
            if(flagHoming==0){
            HAL_Delay(2000); //Realiza un tiempo de espera en milisegundos
            motorHoming();     //Realiza la función Homing
            finCarreraDer=0;
            finCarreraIzq=0;
            }

            //Obtener la posición deseada(grados), vel(grados/s),
acel(grados/s^2) y precision por ordenador (teraterm)
            if(flagIniciarControl==0)
            {
            //Los dos bucles while se emplean para obtener los valores en
            cada iteración por teraterm y pasarlos a matlab
            //Decomentar el siguiente código para poder obtener los valores
            //Posición real del motor
            //int i=0;
            //while (i<500){
            //     float adios=TotalPosiciones[i];
            //     sprintf(buffer,"%f\r\n",adios);
            //HAL_UART_Transmit(&huart2,(uint8_t*)buffer,sizeof(buffer),20);
            //     i++;
            //}

            //Posición calculada de referencia
            //i=0;
            //while (i<500){
            //     float adios=TotalPosicionesRef[i];
            //     sprintf(buffer,"%f\r\n",adios);
            //HAL_UART_Transmit(&huart2,(uint8_t*)buffer,sizeof(buffer),20);
            //     i++;
            //}

            //Mostrar por teraterm si se han accionado los finales de
carrera
            if(finCarreraDer==1)
            {
            //Transmitir mensaje por Tera Term
            HAL_UART_Transmit(&huart2,mensFinDer,sizeof(mensFinDer),30);
            finCarreraDer=0;
            }
            if(finCarreraIzq==1)
            {
            HAL_UART_Transmit(&huart2,mensFinIzq,sizeof(mensFinIzq),30);
            finCarreraIzq=0;
            }

            //Obten posicion deseada en grados
```

```
      //Transmitir mensaje por Tera Term

      HAL_UART_Transmit(&huart2,mensPosFinal,sizeof(mensPosFinal),20);
      //Recibir mensaje por Tera Term
      HAL_UART_Receive(&huart2,mensFinalGrados,sizeof(mensFinalGrados)
,1000000);

      HAL_UART_Transmit(&huart2,mensFinalGrados,sizeof(mensFinalGrados
),20);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
      //Convertir el valor obtenido por Tera Term de tipo carácter a
tipo entero
      intFinalGrados=atoi((char*)mensFinalGrados);
      //Función para limitar el valor máximo que se puede introducir.
En el caso
      //que se introduzca un valor incorrecto, se indica al usuario y
le pide que
      //introduzca uno nuevo.
      while(intFinalGrados>360)
      {
      HAL_UART_Transmit(&huart2,mensError,sizeof(mensError),20);

      HAL_UART_Transmit(&huart2,mensPosFinal,sizeof(mensPosFinal),20);

      HAL_UART_Receive(&huart2,mensFinalGrados,sizeof(mensFinalGrados)
,1000000);

      HAL_UART_Transmit(&huart2,mensFinalGrados,sizeof(mensFinalGrados
),20);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);

      intFinalGrados=atoi((char*)mensFinalGrados);
      }

      //Obten la velocidad deseada (grados/s)

      HAL_UART_Transmit(&huart2,mensVel,sizeof(mensVel),20);

      HAL_UART_Receive(&huart2,mensVrefGrados,sizeof(mensVrefGrados),1
000000);

      HAL_UART_Transmit(&huart2,mensVrefGrados,sizeof(mensVrefGrados),
20);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
      intVrefGrados=atoi((char*)mensVrefGrados);
      while(intVrefGrados>100)
      {

      HAL_UART_Transmit(&huart2,mensError, sizeof(mensError),20);

      HAL_UART_Transmit(&huart2,mensVel,sizeof(mensVel),20);

      HAL_UART_Receive(&huart2,mensVrefGrados,sizeof(mensVrefGrados),1
000000);

      HAL_UART_Transmit(&huart2,mensVrefGrados,sizeof(mensVrefGrados),
```

```
20);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);

      intVrefGrados=atoi((char*)mensVrefGrados);
      }

      //Obtén la aceleración deseada (grados/s2)

      HAL_UART_Transmit(&huart2,mensAcel,sizeof(mensAcel),20);

      HAL_UART_Receive(&huart2,mensArefGrados,sizeof(mensArefGrados),1
000000);

      HAL_UART_Transmit(&huart2,mensArefGrados,sizeof(mensArefGrados),
20);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
      intArefGrados=atoi((char*)mensArefGrados);
      while(intArefGrados>150)
      {

      HAL_UART_Transmit(&huart2,mensError,sizeof(mensError),20);

      HAL_UART_Transmit(&huart2,mensAcel,sizeof(mensAcel),20);

      HAL_UART_Receive(&huart2,mensArefGrados,sizeof(mensArefGrados),1
000000);

      HAL_UART_Transmit(&huart2,mensArefGrados,sizeof(mensArefGrados),
20);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
      intArefGrados=atoi((char*)mensArefGrados);
      }

      //Elige la precisión de muestreo 1ms(Valor 1) o 100us (Valor 2)

      HAL_UART_Transmit(&huart2,mensPrec,sizeof(mensPrec),20);

      HAL_UART_Receive(&huart2,mensPrecision,sizeof(mensPrecision),100
0000);

      HAL_UART_Transmit(&huart2,mensPrecision,sizeof(mensPrecision),20
);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);
      intPrecision=atoi((char*)mensPrecision);
      while(intPrecision!=1 && intPrecision!=2)
      {

      HAL_UART_Transmit(&huart2,mensError,sizeof(mensError),20);

      HAL_UART_Transmit(&huart2,mensPrec,sizeof(mensPrec),20);

      HAL_UART_Receive(&huart2,mensPrecision,sizeof(mensPrecision),100
0000);

      HAL_UART_Transmit(&huart2,mensPrecision,sizeof(mensPrecision),20
```

```
);

      HAL_UART_Transmit(&huart2,espacio,sizeof(espacio),10);

      intPrecision=atoi((char*)mensPrecision);
      }

      //Guardar datos en las variables globales
      vrefGrados=intVrefGrados;
      arefGrados=intArefGrados;
      precision=intPrecision;

      //Convertir grados a rad
      inicio=posActualRad();
      final= intFinalGrados*(3.14159f/180);
      vref=vrefGrados*(3.14159f/180);
      aref=arefGrados*(3.14159f/180);

      //Activar flag para iniciar movimiento (habilita timer 100us)
      flagIniciarControl=1;
      HAL_Delay(1000);
      }
   }
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
 }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};

  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLMUL = RCC_PLL_MUL8;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2
;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
```

```
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) !=
HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_TIM1;
  PeriphClkInit.Tim1ClockSelection = RCC_TIM1CLK_HCLK;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief TIM1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM1_Init(void)
{

  /* USER CODE BEGIN TIM1_Init 0 */

  /* USER CODE END TIM1_Init 0 */

  TIM_ClockConfigTypeDef sClockSourceConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};
  TIM_OC_InitTypeDef sConfigOC = {0};
  TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};

  /* USER CODE BEGIN TIM1_Init 1 */
      //Obtener frecuencia de pwm igual a 10KHz (clock=32MHz)
      //freq=Clock/((prescaler+1)*(Period+1))
  /* USER CODE END TIM1_Init 1 */
  htim1.Instance = TIM1;
  htim1.Init.Prescaler = 32-1;
  htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim1.Init.Period = 1000-1;
  htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim1.Init.RepetitionCounter = 0;
  htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  if (HAL_TIM_Base_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
  if (HAL_TIM_ConfigClockSource(&htim1, &sClockSourceConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
  {
    Error_Handler();
  }
```

```
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  sConfigOC.OCMode = TIM_OCMODE_PWM1;
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
  sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
  sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
  if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_3) !=
HAL_OK)
  {
    Error_Handler();
  }
  sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
  sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
  sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
  sBreakDeadTimeConfig.DeadTime = 0;
  sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
  sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
  sBreakDeadTimeConfig.BreakFilter = 0;
  sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
  sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
  sBreakDeadTimeConfig.Break2Filter = 0;
  sBreakDeadTimeConfig.AutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
  if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM1_Init 2 */

  /* USER CODE END TIM1_Init 2 */
  HAL_TIM_MspPostInit(&htim1);

}

/**
  * @brief TIM2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM2_Init(void)
{

  /* USER CODE BEGIN TIM2_Init 0 */

  /* USER CODE END TIM2_Init 0 */

  TIM_Encoder_InitTypeDef sConfig = {0};
  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM2_Init 1 */
```

```
  /* USER CODE END TIM2_Init 1 */
  htim2.Instance = TIM2;
  htim2.Init.Prescaler = 0;
  htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim2.Init.Period = 2048;
  htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
  htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_ENABLE;
  sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
  sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
  sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
  sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
  sConfig.IC1Filter = 10;
  sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
  sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
  sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
  sConfig.IC2Filter = 10;
  if (HAL_TIM_Encoder_Init(&htim2, &sConfig) != HAL_OK)
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM2_Init 2 */

  /* USER CODE END TIM2_Init 2 */

}

/**
  * @brief TIM6 Initialization Function
  * @param None
  * @retval None
  */
static void MX_TIM6_Init(void)
{

  /* USER CODE BEGIN TIM6_Init 0 */

  /* USER CODE END TIM6_Init 0 */

  TIM_MasterConfigTypeDef sMasterConfig = {0};

  /* USER CODE BEGIN TIM6_Init 1 */
      //Para interrupcion de 100us
      //htim6.Init.Prescaler = (SystemCoreClock/1000000)-1;
  //htim6.Init.Period = 100-1;
  /* USER CODE END TIM6_Init 1 */
  htim6.Instance = TIM6;
  htim6.Init.Prescaler = (SystemCoreClock/1000000)-1;
  htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
  htim6.Init.Period = 100-1;
  htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
  if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
```

```
  {
    Error_Handler();
  }
  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
  if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) !=
HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN TIM6_Init 2 */

  /* USER CODE END TIM6_Init 2 */

}

/**
  * @brief USART2 Initialization Function
  * @param None
  * @retval None
  */
static void MX_USART2_UART_Init(void)
{

  /* USER CODE BEGIN USART2_Init 0 */

  /* USER CODE END USART2_Init 0 */

  /* USER CODE BEGIN USART2_Init 1 */

  /* USER CODE END USART2_Init 1 */
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 9600;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
  huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN USART2_Init 2 */

  /* USER CODE END USART2_Init 2 */

}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();
```

```
  /* DMA interrupt init */
  /* DMA1_Channel6_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Channel6_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Channel6_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOF_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_5, GPIO_PIN_RESET);

  /*Configure GPIO pin : PA7 */
  GPIO_InitStruct.Pin = GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pin : PB0 */
  GPIO_InitStruct.Pin = GPIO_PIN_0;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pins : PB4 PB6 PB7 */
  GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_6|GPIO_PIN_7;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : PB5 */
  GPIO_InitStruct.Pin = GPIO_PIN_5;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /* EXTI interrupt init*/
  HAL_NVIC_SetPriority(EXTI0_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI0_IRQn);

  HAL_NVIC_SetPriority(EXTI4_IRQn, 0, 0);
```

```
  HAL_NVIC_EnableIRQ(EXTI4_IRQn);

  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);

}

/* USER CODE BEGIN 4 */

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error
return state */

  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line
number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and
line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n",
file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

/************************ (C) COPYRIGHT STMicroelectronics *****END OF
FILE****/
```

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DE UN SERVOMOTOR MEDIANTE UNA TARJETA STM32 NUCLEO 32

2. Planos

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR: **Jaime Ferre Martínez**

TUTORIZADO POR: **Ranko Zotovic Stanisic**

CURSO ACADÉMICO: 2020/2021

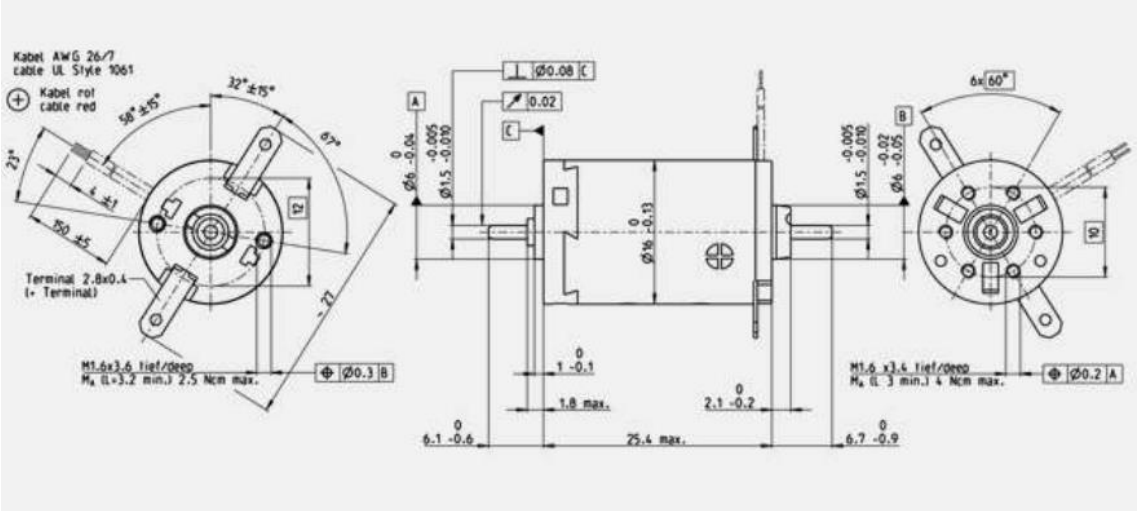
Índice

1. Introducción

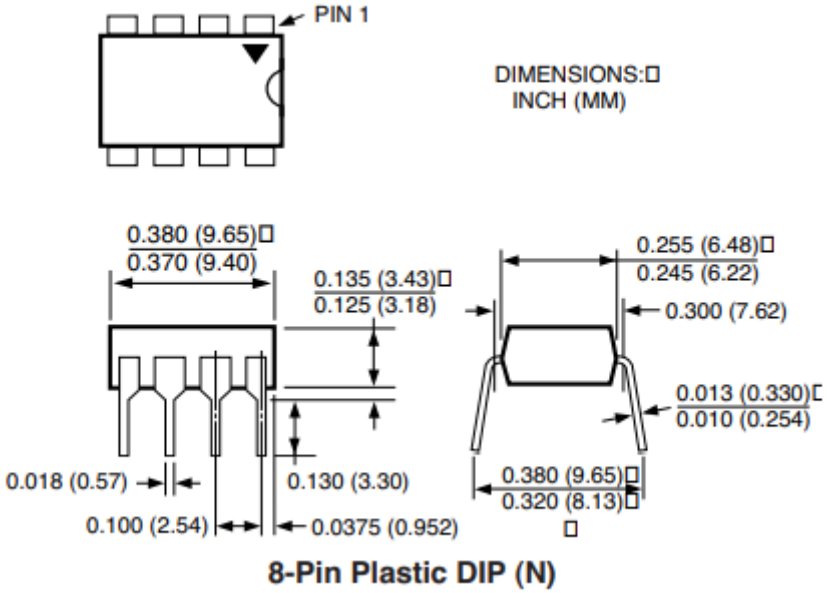
Dado que no se ha fabricado ninguna pieza ni componente, solamente se han adjuntado los planos del circuito construido y las medidas de los componentes.

2. Planos

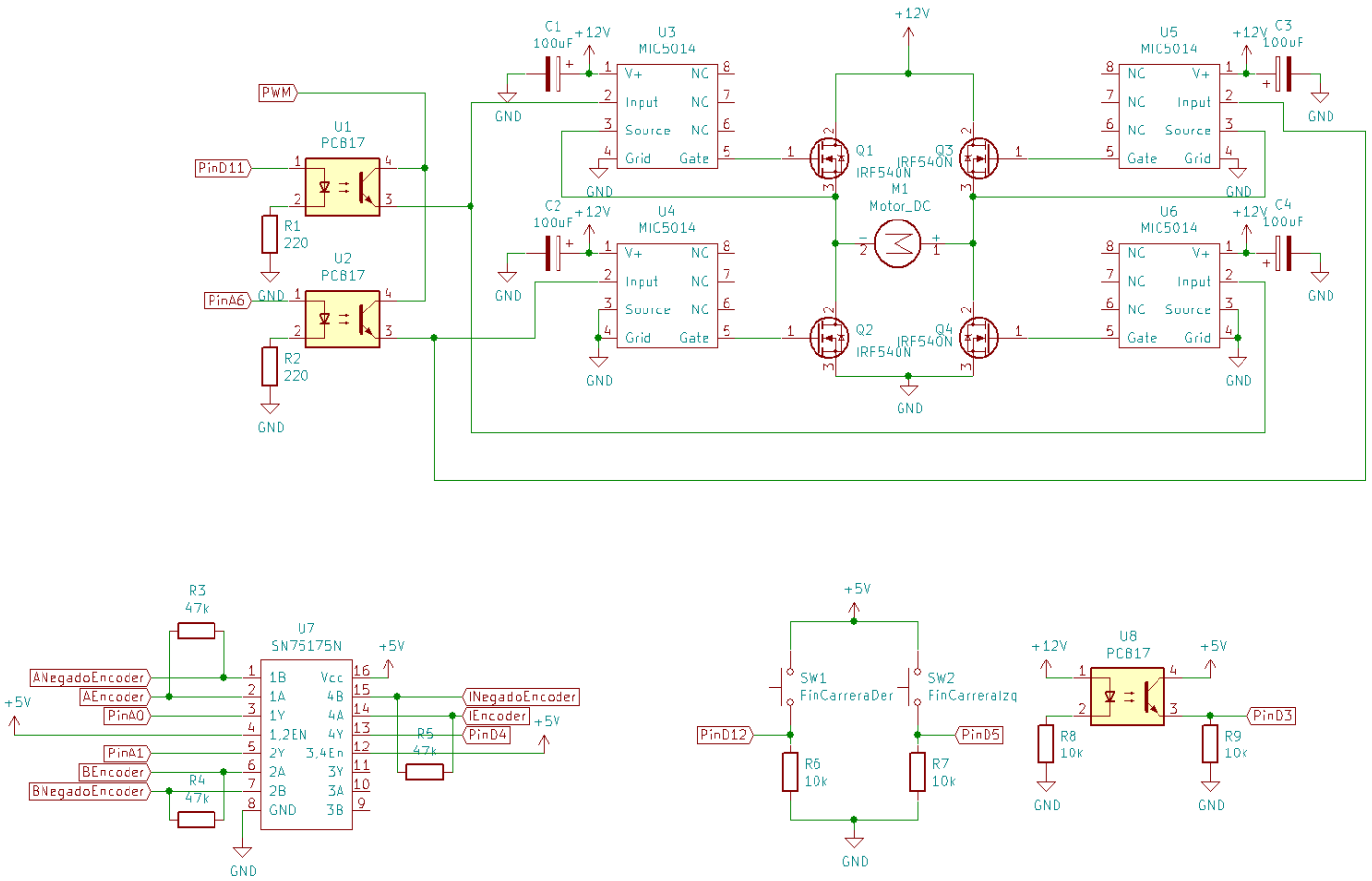
2.1 Motor



2.2 Mosfet driver

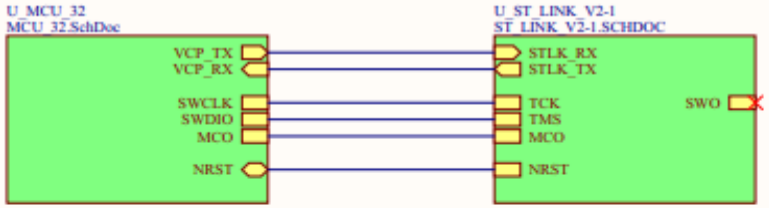


8-Pin Plastic DIP (N)



Sheet: /		
File: tfg.sch		
Title: Circuito control servomotor		
Size: A4	Date: 2021-04-20	**Rev: Jaime Ferre**
KiCad E.D.A. eeschema (5.1.6)-1		Id: 1/1

STM32 Nucleo-32 board (top view)

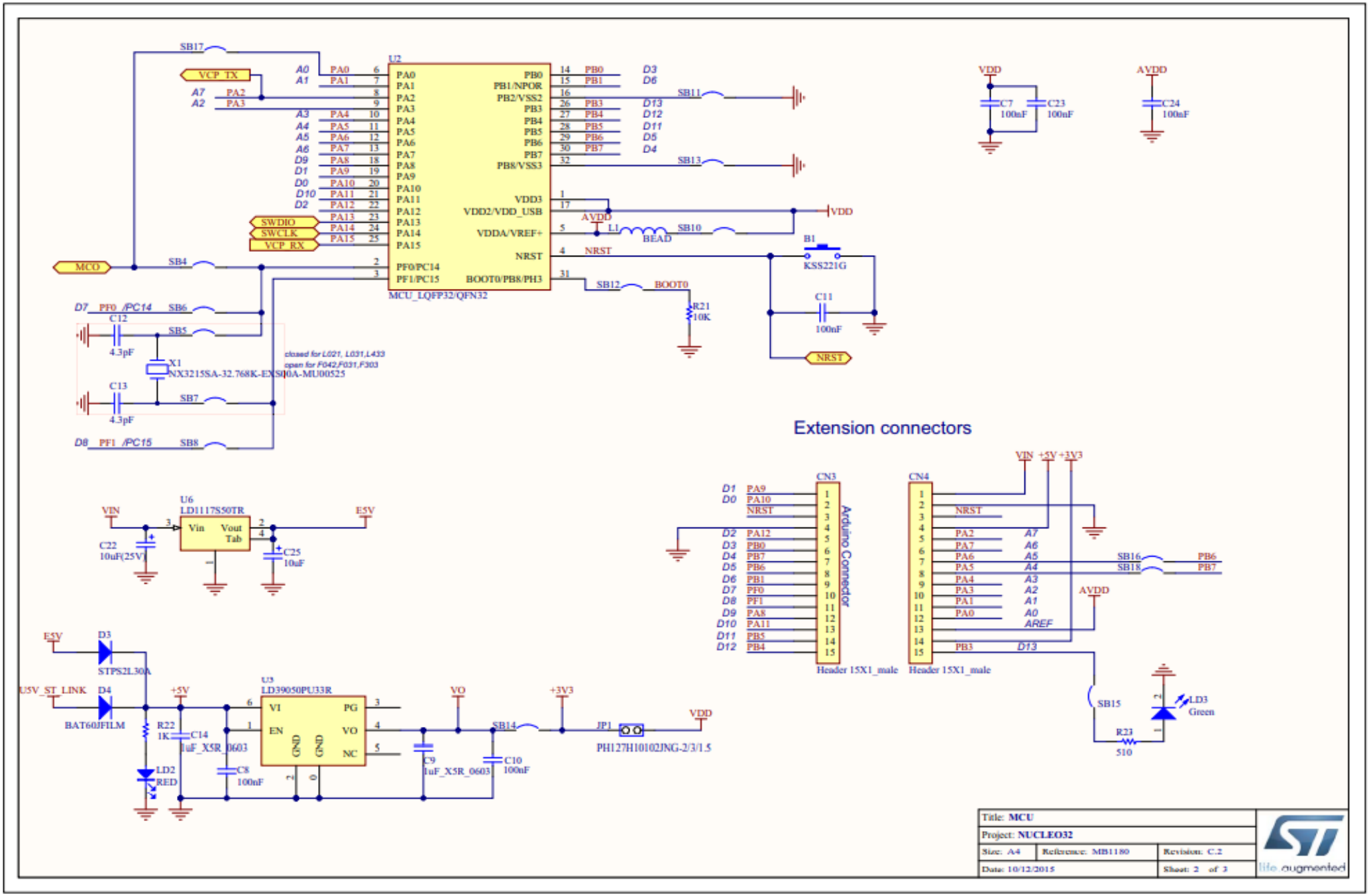


REV B: SB14 changed to JP1 Jumper for easy IDD measurement, and enlarge board length; CN1 USB PN changed to Micro-B for Device.
REV C: Add SB18/SB16 for connecting D4/D5 to A4/A5
REV C.2: correct silkscreen D7/D8 on SB6 and SB8

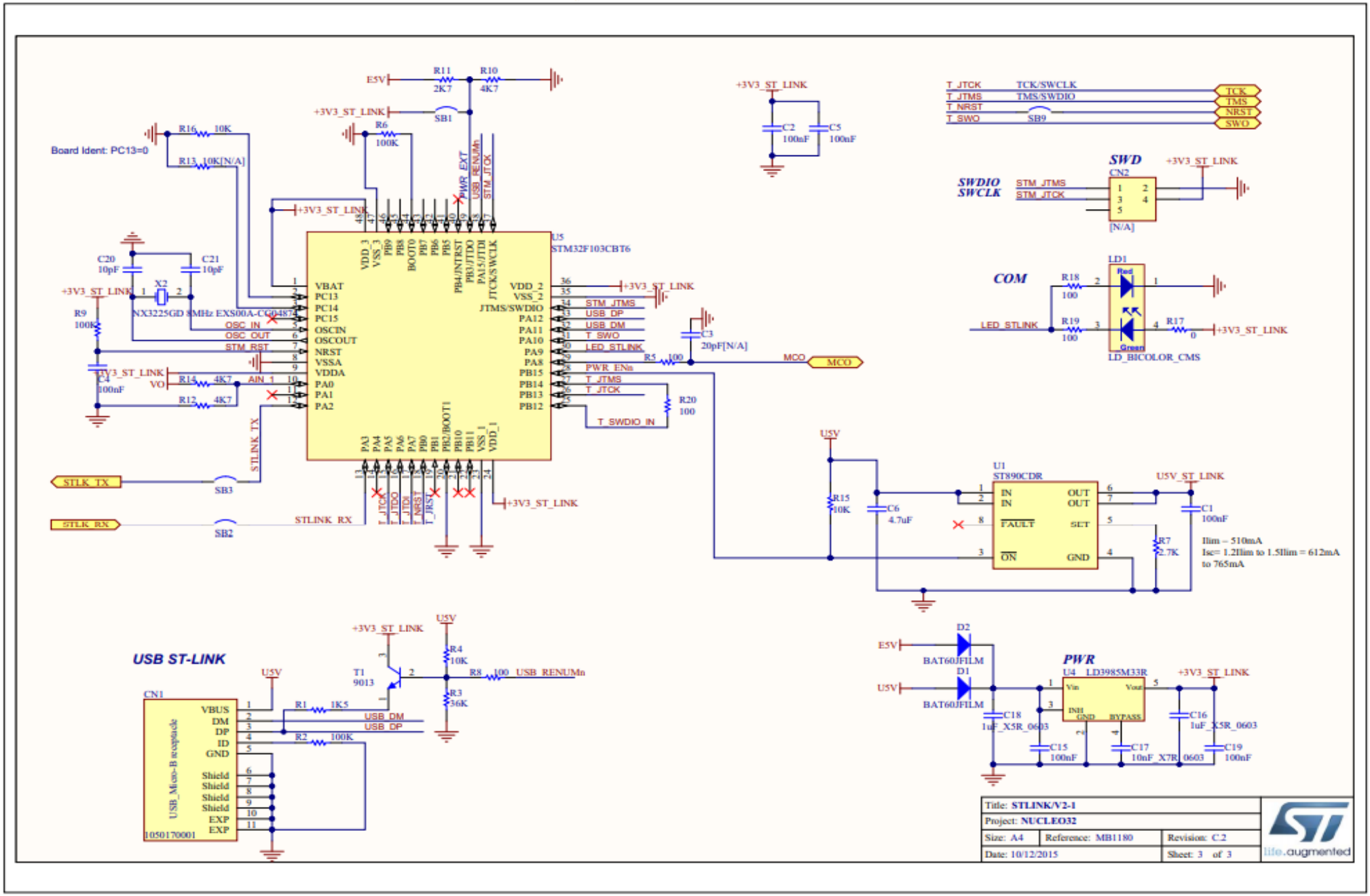
Title: **TOP**		
Project: **NUCLEO32**		
Size: A4	Reference: MB1180	Revision: C.2
Date: 10/12/2015		Sheet: 1 of 3

life.augmented

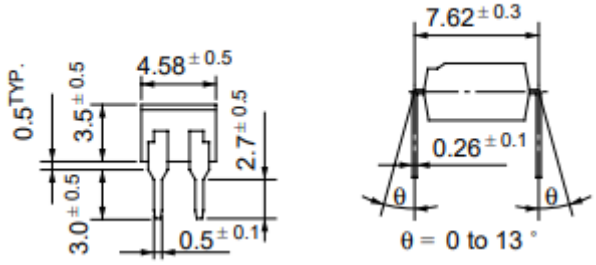
MCU


SB17
VCP_TX
A0 PA0
A1 PA1
A7 PA2
A2 PA3
A3 PA4
A4 PA5
A5 PA6
A6 PA7
D9 PA8
D1 PA9
D0 PA10
D10 PA11
D2 PA12
SWDIO PA13
SWCLK PA14
VCP_RX PA15
U2
PA0 PA1 PA2 PA3 PA4 PA5 PA6 PA7 PA8 PA9 PA10 PA11 PA12 PA13 PA14 PA15
PB0 PB1/NPOR PB2/VSS2 PB3 PB4 PB5 PB6 PB7 PB8/VSS3
VDD3 VDD2/VDD_USB VDDA/VREF+ NRST BOOT0/PB8/PH3 PF0/PC14 PF1/PC15
MCU_LQFP32/QFN32
D3 D6 D13 D12 D11 D5 D4
SB11 SB13
VDD AVDD L1 BEAD SB10
NRST B1 KSS221G
SB12 BOOT0 R21 10K
C11 100nF
MCO SB4
D7 PF0 /PC14 SB6
C12 4.3pF SB5
X1 NX3215SA-32.768K-EXS00A-MU00525
closed for L021, L031,L433
open for F042,F031,F303
C13 4.3pF SB7
D8 PF1 /PC15 SB8
C7 100nF C23 100nF
AVDD C24 100nF
Extension connectors
CN3 Arduino Connector
D1 PA9 D0 PA10 NRST D2 PA12 D3 PB0 D4 PB7 D5 PB6 D6 PB1 D7 PF0 D8 PF1 D9 PA8 D10 PA11 D11 PB5 D12 PB4
Header 15X1_male
CN4
VIN +5V +3V3 NRST PA2 A7 PA7 A6 PA6 A5 PA5 A4 PA4 A3 PA3 A2 PA1 A1 PA0 A0 AREF PB3 D13
SB16 PB6 SB18 PB7 AVDD
Header 15X1_male
U6 LD1117S50TR
VIN Vin Vout Tab E5V
C22 10uF(25V) C25 10uF
E5V D3 STPS2L30A
U5V_ST_LINK D4 BAT60JFILM
+5V R22 1K C14 1uF_X5R_0603 LD2 RED C8 100nF
U3 LD39050PU33R
VI EN GND GND PG VO NC
VO C9 1uF_X5R_0603 SB14 +3V3 JP1 PH127H10102JNG-2/3/1.5 VDD
SB15 R23 510 LD3 Green
Title: MCU
Project: NUCLEO32
Size: A4 Reference: MB1180 Revision: C.2
Date: 10/12/2015 Sheet: 2 of 3
life.augmented

ST-LINK/V2-1


Board Ident: PC13=0
STM32F103CBT6
NX3225GD 8MHz EXS00A-CG04874
USB ST-LINK
SWD
SWDIO
SWCLK
COM
PWR
LD_BICOLOR_CMS
USB_Micro-B receptacle
1050170001
BAT60JFILM
LD_LINK_1
Title: STLINK/V2-1
Project: NUCLEO32
Size: A4
Reference: MB1180
Revision: C.2
Date: 10/12/2015
Sheet: 3 of 3
life.augmented

2.5 Fotoacoplador



UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DE UN SERVOMOTOR MEDIANTE UNA TARJETA STM32 NUCLEO 32

3. Pliego de condiciones

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR: **Jaime Ferre Martínez**

TUTORIZADO POR: **Ranko Zotovic Stanisic**

CURSO ACADÉMICO: 2020/2021

Ínidice

1. Objeto

Esta especificación se refiere a todas las medidas que se deben tomar para el montaje del circuito destinado al control de un servomotor teniendo en cuenta las condiciones de los materiales que lo componen.

Todo el trabajo que no se ha descrito, no está incluido en esta especificación.

2. Condiciones de los materiales

Todos los componentes deben poseer un encapsulado de tipo agujero pasante.

2.1 Resistencias

Las resistencias no deben tener una disipación mayor de ½ W.

La tolerancia de las resistencias debe ser igual o menor que 10%.

2.2 Condensadores electrolíticos

Los condensadores electrolíticos deben tener 20% o menor tolerancia.

2.3 Protoboard

La protoboard debe poseer más de 80 líneas de conexiones.

2.4 Motor

El motor debe soportar una alimentación de 12V.

2.5 Encoder

El encoder debe poseer una resolución de 512 pulsos por vuelta.

El encoder debe poseer 3 canales con sus respectivas salidas complementarias.

2.6 Mosfet driver MIC5014

El driver MIC5014 debe poseer un encapsulado tipo SOIC-8.

2.7 Driver SN75175

El driver SN75175 debe poseer un encapsulado tipo PDIP-16.

2.8 Fuente de alimentación

La fuente de alimentación debe ser de +12V DC con potencia superior a 24W.

La fuente de alimentación debe poseer un interruptor para controlar la etapa de alimentación. Dicho interruptor debe poder controlar 220V y más de 6A.

3. Condiciones de la ejecución

3.1 Instalación

Para el montaje del circuito en la protoboard, todos los componentes deben ser escogidos y clasificados previamente.

Primero se deberá conectar la tarjeta STM32 Nucleo-32 al lado izquierdo de la protoboard, dirigiendo el puerto usb hacia el exterior. A continuación, se conectarán el componente SN75175, los dos fotoacopladores PC817, los cuatro drivers MIC5014 y los cuatro transistores IRF540, de izquierda a derecha siguiendo a la tarjeta STM32 respectivamente.

Luego se introducirán las conexiones entre dichos componentes y se conectarán los finales de carrera y el tercer fotoacoplador PC817 a la derecha de los transistores.

Una vez conectados todos los componentes principales, se conectarán las resistencias y condensadores electrolíticos.

Finalmente, se conectará el puerto usb de la tarjeta STM32 al computador y se conectará la fuente de alimentación de +12V.

3.2 Seguridad

Para la manipulación del sistema se deben emplear guantes aislantes para evitar descargas eléctricas procedentes de la fuente de alimentación de +12V o de la red eléctrica de 220 AC. Al no estar aislada la etapa de potencia, se debe encarecidamente no modificar ningún componente ni conexión mientras la fuente de alimentación esté encendida.

Se requerirá el uso de herramientas de instrumentación que estén previamente homologadas.

3.3 Uso y control

Para utilizar y controlar debidamente el sistema, se deben de realizar varios pasos.

Primero, se debe conectar el usb de la tarjeta al computador y se debe realizar la comunicación entre ambos mediante Tera Term.

Segundo, después de comunicar ambos dispositivos por Tera Term, se debe pulsar el botón reinicio de la tarjeta STM32.

Tercero, se espera unos segundos y se enciende la etapa de alimentación. Al activarla, se mostrará por pantalla un mensaje indicándolo.

Cuarto, se realizará de forma automática la función de homing. El operario no debe realizar ninguna acción.

Quinto, se pedirá al operario que introduzca los valores de posición, velocidad, aceleración y precisión por pantalla. Al hacerlo, el motor llegará a la posición deseada y se volverá a pedir nuevos datos de forma indefinida.

4. Pruebas y ajustes finales o de servicio.

Cada conexión debe ser comprobada para asegurar que el funcionamiento del circuito es el correcto. Para conseguir esta tarea, se debe desconectar la fuente de alimentación y se debe conectar la tarjeta STM32 al computador. Una vez realizada la comunicación entre ambos por Tera Term, se introduce una señal de +3.3V DC mediante una fuente de alimentación variable, en el pin de detección de la etapa de alimentación. Si el circuito funciona correctamente, se mostrará por pantalla el mensaje que indica que se ha activado la etapa de alimentación. A continuación, se realizan tres giros manuales del motor. Si se ha realizado correctamente, al cabo de unos segundos pedirá por pantalla los valores de la posición objetivo indicando que funciona correctamente.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

CONTROL DE UN SERVOMOTOR MEDIANTE UNA TARJETA STM32 NUCLEO 32

4. Presupuesto

Trabajo final del Grado en Ingeniería Electrónica Industrial y Automática

REALIZADO POR: **Jaime Ferre Martínez**

TUTORIZADO POR: **Ranko Zotovic Stanisic**

CURSO ACADÉMICO: 2020/2021

Índice

1. Listado de precios

Costes materiales

Código	Unidad	Descripción	Coste Unitario
m1	u.	Condensador electrolítico 100uF	0,236 €
m2	u.	Resistor 220 Ω	0,004 €
m3	u.	Resistor 10 kΩ	0,004 €
m4	u.	Resistor 47 kΩ	0,004 €
m5	u.	Protoboard	4,990 €
m6	u.	STM32F303K8 Nucleo-32	8,970 €
m7	u.	Mosfet IRF540	0,950 €
m8	u.	Mosfet driver MIC5014	2,500 €
m9	u.	Differential line driver SN75175	3,600 €
m10	u.	Fotoacoplador PC817	0,330 €
m11	u.	Final de carrera	0,650 €
m12	u.	Motor A-max 16 Ø16 mm 12V	40,820 €
m13	u.	Encoder MR de 512 ppv y 3 canales	96,910 €

Costes de mano de obra

Código	Unidad	Descripción	Coste unitario
l1	h.	Técnico	10 €

2. Precios por unidad

Código	Unidad	Descripción	Coste Unitario	Cantidad	Total (€)
d1	u.	Montaje del circuito en la protoboard			
Materiales					
m1	u.	Condensador electrolítico 100uF	0,236 €	4	0,94 €
m2	u.	Resistor 220 Ω	0,004 €	2	0,01 €
m3	u.	Resistor 10 kΩ	0,004 €	4	0,02 €
m4	u.	Resistor 47 kΩ	0,004 €	3	0,01 €
m5	u.	Protoboard	4,990 €	1	4,99 €
m6	u.	STM32F303K8 Nucleo-32	8,970 €	1	8,97 €
m7	u.	Mosfet IRF540	0,950 €	4	3,80 €
m8	u.	Mosfet driver MIC5014	2,500 €	4	10,00 €
m9	u.	Differential line driver SN75175	3,600 €	1	3,60 €
m10	u.	Fotoacoplador PC817	0,330 €	3	0,99 €
m11	u.	Final de carrera	0,650 €	2	1,30 €
m12	u.	Motor A-max 16 Ø16 mm 12V	40,820 €	1	40,82 €
m13	u.	Encoder MR de 512 ppv y 3 canales	96,910 €	1	96,91 €
Mano de obra					
I1	h.	Técnico	10 €	1	10,00 €
Sobrecostes					
	%	Sobrecostes del precio total	172,359 €	20%	34,47 €
				TOTAL (€)	216,83 €

3. Validación final

Código	Unidad	Descripción	Precio(€)	Cantidad	Total (€)
d1	u.	Montaje del circuito en la protoboard	216,83	1	216,83
	%	IVA	216,83	21%	45,53
			Precio Final (€)		**262,36**