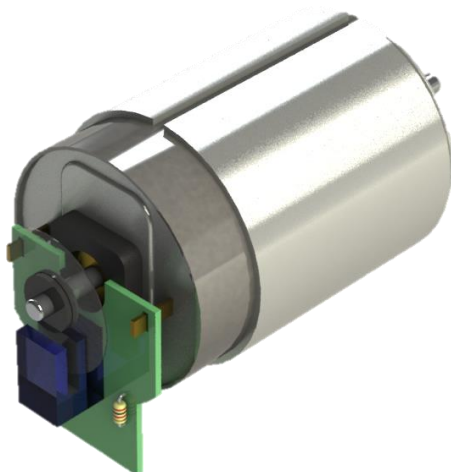


Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

1. Memoria
2. Planos
3. Pliego de condiciones
4. Presupuesto
5. Anexos



Autor:
D. Adrià Álvarez Donet
Tutor:
D. Ángel Perles Ivars
Cotutor:
D. Miguel Sánchez López
Valencia, julio de 2017

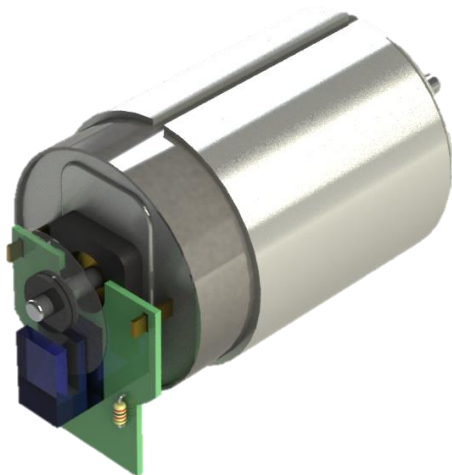
Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Resumen:

El proyecto consiste en el diseño e implementación de un sistema de 32 bits que controlara el movimiento sincronizado de varios motores en bucle cerrado para aplicación en impresión 3D, maquinas CNC, robótica y otros campos que requieran esta tecnología. El objetivo es reemplazar el extendido uso de motores paso a paso por motores de corriente continua con control de posición. Esta tecnología nos permite conocer con precisión la posición real de los motores, además, el controlador será capaz de leer un código de coordenadas (gcode) para ejecutar el movimiento de dichos motores de forma coordinada. Para el controlador se utilizará una placa de desarrollo de la familia stm32 Nucleo a la que se le conectaran los puentes en H para el control de los motores, y los encoders de los motores, que nos proporcionarán la posición de estos.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

1.MEMORIA



Autor:

D. Adrià Álvarez Donet

Tutor:

D. Ángel Perles Ivars

Cotutor:

D. Miguel Sánchez López

Valencia, julio de 2017

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Indice

1. Objeto.....	3
2. Antecedentes.....	3
3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes.....	4
3.1. Versatilidad.....	4
3.2. Muestreo.....	5
3.3. Precisión.....	6
3.4. Restricciones del sistema.....	7
4. Planteamiento de soluciones alternativas y justificación de la solución adoptada.....	9
4.1. Elección de los motores.....	9
4.1.1. Motores Paso a Paso.....	9
4.1.2. Motores sin escobillas.....	10
4.1.3. Motores con corriente continua.....	11
4.2. Sensores de posición Angular.....	13
4.2.1. Sensores de efecto Hall.....	13
4.2.2. Resolvers.....	13
4.2.3. Sensores angulares potenciométricos	
4.2.4. Encoders	
4.3. Microcontrolador.....	15
4.3.1. Arduino.....	15
4.3.2. STM32F4 Discovery.....	16

1. MEMORIA

4.3.3. STM32 NUCLEO.....	16
4.4. Drivers motores.....	17
4.4.1. L293.....	17
4.4.2. L298.....	17
4.4.3. VNH2SP30.....	18
4.5. Plataforma de programación.....	18
4.5.1. Keil μ vision.....	18
4.5.2. ARM mbed.....	18
4.6. Trayectoria.	19
4.6.1. No actuación.....	19
4.6.2. Trapezoidal	19
4.6.3. Perfil en “s”.....	22
4.6.4. Senoidal.....	23
5. Descripción de la solución adoptada	24
5.1. Montaje de pruebas.....	24
5.2. Ajuste de la respuesta.....	26
5.3. Montaje final.....	34
5.4. Programación.....	38
6. Conclusiones.....	42
6.1. Sobre el trabajo realizado.....	42
6.2. Futuras mejoras.....	42
7. Bibliografía.....	44

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

1. Objeto

El objeto de este proyecto es el diseño e implementación de un controlador para motores de corriente continua en bucle cerrado para aplicación en robótica, máquinas de control numérico o impresión 3D. Este controlador será capaz de mover varios motores coordinadamente según las ordenes que reciba.

Para desarrollar este proyecto se decidirán los materiales a utilizar, como los motores, sus controladores y el microcontrolador, basándose en la facilidad de uso y su precio. A continuación, se conectarán los motores, los controladores y la placa del microcontrolador y se programará un código en C++ que posteriormente se cargará a la placa de desarrollo. Posteriormente se decidirá un tiempo de muestreo y se adquirirán los datos de posición mediante el puerto serie para ajustar los parámetros del regulador PID.

Una vez ajustados los PIDs se realizará un montaje más profesional del conjunto y se hará un prototipo que simulará uno de los ejes de una maquina real.

2. Antecedentes

Hoy en día, los motores más usados para la robótica industrial, máquinas de control numérico (CNC) o las impresoras 3D, son los motores paso a paso. Estas máquinas suelen funcionar con un código de coordenadas (gcode) que determina la posición del robot.

Uno de los inconvenientes de estos motores es que funcionan en bucle abierto, es decir no se conoce la posición real del motor. Algunos factores, como la pérdida de pasos debido a fallos eléctricos o mecánicos, puede provocar que la posición deseada de los motores paso a paso no sea la posición real, teniendo así que reiniciar la maquina a un punto de origen. Esto provoca que algunos procesos se tengan que desechar debido a estos fallos pudiendo provocar grandes pérdidas económicas.

1. MEMORIA

Otro gran inconveniente que tienen estos motores es la elevada corriente de mantenimiento, que es necesaria para que la máquina permanezca en la posición deseada. Existen soluciones a estos problemas, como *Mechaduino*, desarrollado por *Tropical Labs* [MEC 14], que consiste en un controlador para motores paso a paso, añadiendo a estos un sensor para el control en bucle cerrado. Esta solución supone un gran desembolso económico, debido a que además del elevado precio de los motores hay que sumar el coste de los controladores, ya que se necesita uno por motor.

Otra posible solución es el uso de motores de corriente continua. Existen en el mercado una gran variedad de este tipo de motores que llevan incorporados encoders para poder controlarlos en bucle cerrado. En las mismas condiciones de par, tensión o intensidad, los motores de corriente continua ofrecen un precio más competitivo que los motores paso a paso.

Los factores mencionados anteriormente motivan el desarrollo de un controlador para motores de corriente continua en bucle cerrado de bajo coste.

3. Estudio de necesidades, factores a considerar: limitaciones y condicionantes

Cada vez está más extendido el uso de máquinas de control numérico (CNC) para la automatización de ciertos procesos de fabricación, ya que son más rápidas, precisas y seguras que los métodos de fabricación convencional. Debido a estos factores se requiere un aumento de precisión y fiabilidad debido al auge de la impresión 3D en el ámbito doméstico donde también se requiere una reducción de los costes de este tipo de controladores.

Entre los factores y necesidades a tener en cuenta en la implementación del proyecto destacan las siguientes:

3.1. Versatilidad

Un factor muy importante en estos controladores es su versatilidad, es decir, su capacidad de adaptarse a la configuración de la máquina. Para ello

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

se debe implementar mediante un controlador programable, ya que es la opción más barata y se pueden modificar fácilmente los parámetros del sistema a controlar, como la longitud máxima de los ejes, la velocidad máxima o la aceleración. Otras ventajas del uso de un microcontrolador son la posibilidad de implementar algoritmos más complejos, la capacidad de comunicarse con otros dispositivos o sistemas.

Algo a tener en cuenta a la hora de desarrollar el código de programación, es que este debe ser fácil de entender y lo más simplificado posible, para que cualquier usuario que necesite modificar el código sea capaz de comprender su funcionamiento y editar las partes que sean necesarias.

3.2. Muestreo

Definimos el muestreo como la medición de las señales que recibe el microcontrolador para convertirla en una variable que pueda interpretar.

El tiempo entre dos medidas realizadas por el procesador se denomina tiempo de muestreo. Existe otras opciones como el muestreo por interrupción, que será el que utilizaremos para medir la posición de los motores mediante los encoders. Este tipo de muestreo consiste en la detección de una variación de la señal en un pin de entrada del microcontrolador para ejecutar un evento. Usar las interrupciones para medir la posición del encoder nos evita perder la medición de algún pulso por lo que el sensado de la posición será más preciso.

Otro factor relativo al muestreo que deberemos tener en cuenta es el aliasing. Este fenómeno sucede cuando la frecuencia de muestreo es inferior a la necesaria y debido a esto, la señal medida no se corresponde con la real.

Para evitar este problema debemos seguir el teorema de Shannon-Nyquist, que dice que la frecuencia de muestreo debe ser mayor que el doble de la frecuencia de la señal a medir:

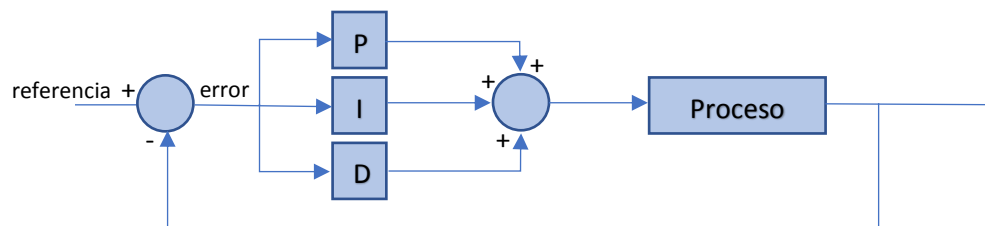
$$f_{muestreo} > 2 \cdot f_{señal}$$

1. MEMORIA

3.3. Precisión

Como ya hemos comentado anteriormente, un factor muy importante en este tipo de máquinas es la precisión. Esto depende de muchos factores, pero principalmente dependerá de la resolución del sensor de posición, y del ajuste de los reguladores.

El regulador más utilizado en el mundo de la automática es el PID (Proporcional Integral Derivativo). El regulador PID responde al siguiente esquema:



La acción proporcional, como indica su nombre, es proporcional a la diferencia entre la señal de referencia y la señal medida a la salida del proceso. A esta diferencia se le denomina error.

$$u(t) = k_p \cdot e(t)$$

Donde $U(t)$ será la salida del PID y K_p será la ganancia o constante proporcional.

La acción de control derivativa será proporcional a la derivada de la señal de error. Basándose en la tendencia de la señal de error, la acción proporcional permite adelantarse a este error, debido a esto la respuesta del sistema será más rápida.

$$u(t) = k_d \cdot \frac{d e(t)}{d t}$$

También disponemos de una acción de control integral que, por consiguiente, será proporcional a la integral de la señal de error. Esta acción tiene la función de eliminar el error de posición en régimen permanente.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

$$u(t) = k_i \cdot \int_0^t e(t) \cdot dt$$

La salida del regulador PID será por tanto la suma de todas estas acciones o la suma de la acción proporcional más la acción que necesitemos. De esta manera podremos implementar controladores de tipo P, PD, PI o PID. Estos controladores tienen las siguientes ecuaciones:

Control PD (Proporcional-Derivativo):

$$u(t) = k_{PD} \cdot \left(e(t) + T_d \cdot \frac{d e(t)}{d t} \right)$$

Control PI (Proporcional-Integral):

$$u(t) = k_{PI} \cdot \left(e(t) + \frac{1}{T_i} \cdot \int_0^t e(t) \cdot dt \right)$$

Control PID (Proporcional-Integral-Derivativo):

$$u(t) = k_{PID} \cdot \left(e(t) + \frac{1}{T_i} \cdot \int_0^t e(t) \cdot dt + T_d \cdot \frac{d e(t)}{d t} \right)$$

Además, como hemos mencionado al principio, los sensores de posición también jugaran un papel importante a la hora de lograr una gran precisión, por lo tanto, cuando mayor sea la resolución de los sensores que utilicemos, mayor será la precisión en el desplazamiento de nuestro mecanismo.

3.4. Restricciones del sistema

Modelar el sistema a controlar y conocer todos sus elementos nos permite desarrollar el código de programación de una manera más sencilla y eficaz.

En nuestro caso, un factor a tener en cuenta es la discretización de los reguladores y las acciones de control. Al usar un microcontrolador como base de nuestro sistema, debemos saber que este no funciona en el dominio del tiempo, si no en el discreto, es decir, nuestro microcontrolador procesará

1. MEMORIA

los datos secuencialmente mediante una frecuencia de reloj programada, y no de forma continua como lo haría un control analógico. Esto nos puede afectar a la hora de trabajar con señales de alta frecuencia, aunque con los microcontroladores actuales se logra una velocidad más que suficiente para medir los pulsos de nuestros encoders y generar las acciones de control para los motores.

Al trabajar en el dominio discreto, debemos tener en cuenta que no podemos trabajar con las ecuaciones de automática en el dominio de la "S" ya que estas están basadas en el dominio continuo. Cuando se trabaja con ecuaciones discretas se denomina dominio de la "Z". Por este motivo, debemos tener en cuenta que para poder trabajar con las ecuaciones del PID en nuestro microcontrolador, será necesario discretizarlas primero. Para discretizar una función, hay que tener en cuenta el tiempo de muestreo con el que trabajara esta.

Otros factores que limitaran el diseño de nuestro proyecto, son los de tipo mecánico. Nuestros ejes no son infinitos, por lo que, a la hora de programar el microcontrolador, deberemos tener en cuenta la posición máxima en la que se podrán situar nuestros ejes y limitar ésta mediante una función, para evitar roturas o posibles accidentes. Esta función se puede implementar fácilmente con un if, convirtiendo la posición objetivo en la posición máxima del eje, en caso de que esta la supere.

Otro factor mecánico a tener en cuenta son las velocidades y las aceleraciones. Unas aceleraciones muy bruscas pueden causar deslizamientos en las correas de la máquina, oscilaciones en el cabezal, o incluso la rotura de componentes de ésta. Para ello, es importante la creación de patrones de desplazamiento que conseguirán un movimiento más suave y un control absoluto sobre la velocidad y la aceleración máxima de los ejes. Las características y tipos de estos perfiles de movimiento se explicarán detalladamente más adelante.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

4. Planteamiento de soluciones alternativas y justificación de la solución adoptada

A continuación, se detallan las alternativas que se estudiaron durante la implementación del proyecto y la justificación de la solución que se ha llevado a cabo.

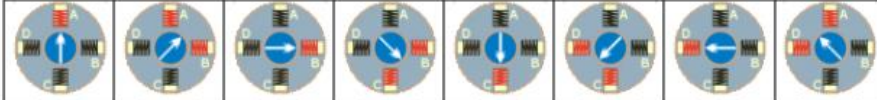
4.1. Elección de los motores

Los motores son una de las partes más importantes en el proyecto, ya que son los encargados de transformar los impulsos eléctricos producidos por el microcontrolador en un movimiento angular.

El primer paso fue elegir el tipo de motores que se iban a controlar. Los más populares son los siguientes:

4.1.1. Motores Paso a Paso

Los motores paso a paso, también conocidos por el anglicismo “steppers” están formados por varias bobinas de excitación en el estator y un imán permanente en el rotor. Al excitar las bobinas no se logra un movimiento de rotación continua como en el caso de los motores de rotación continua convencionales. En estos motores al excitar una de sus bobinas se logra que el eje se sitúe en una posición angular determinada. Para conseguir una rotación continua del eje hay que excitar las bobinas de forma secuencial. Según la bobina, o combinación de ellas, que reciban corriente, se logra una posición diferente denominadas pasos, que se detalla en la siguiente tabla.



A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D	A	B	C	D										
1	0	0	0	1	1	0	0	0	1	0	0	0	1	1	0	0	1	0	0	0	1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1

Ilustración 1. Posiciones Motor Paso a Paso

Estos motores tienen una serie de ventajas:

1. MEMORIA

- No necesitan realimentación, ya la posición deseada se consigue mediante la excitación de sus bobinas.
- No tienen escobillas. Debido a esto, su vida útil es muy elevada porque no hay componentes que se desgasten por fricción.
- Fáciles de controlar. Para mover el motor sólo es necesario conocer los pasos del motor por revolución para programar una sencilla secuencia.

Aunque estos motores también conllevan una serie de inconvenientes:

- Discontinuidad de movimiento a baja velocidad. Este fenómeno se reduce cuando mayores sean los pasos por revolución.
- Pérdida de pasos. Debido a factores mecánicos, como el rozamiento entre partes de la máquina u obstrucción y a factores eléctricos, como la falta de corriente, puede suceder que la posición deseada no sea la posición real del motor. A esto se le denomina pérdida de pasos.
- Alta corriente de mantenimiento. Para fijar el motor en una posición concreta, los motores necesitan una determinada corriente. Debido a esto, las máquinas con este tipo de motores tienen un consumo elevado, aunque no se estén moviendo.

4.1.2. Motores Sin escobillas

Estos motores, también conocidos como “Brushless” tienen las bobinas de excitación en el estator, debido a esto no requieren ningún tipo de elemento rozante para transmitir la corriente entre sus partes. El rotor de este tipo de motores está formado por imanes permanentes. A diferencia de los motores paso a paso, estos motores se alimentan con corriente alterna que permite un movimiento continuo del eje.

Un parámetro importante en este tipo de motores es el K_v que expresa la velocidad angular en rpm por cada voltio de alimentación.

Aunque algunos de estos motores cuentan con un sensor hall para conocer su posición a la hora de generar las señales de control, este no

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

es muy preciso, por lo que requieren otro sensor de posición para más exactitud.

Las ventajas de estos motores son:

- No tiene escobillas, por lo que su vida útil es, muy elevada.
- Velocidad. Tiene un rango de velocidad muy elevado debido a la ausencia de limitaciones mecánicas.
- La relación par/velocidad es prácticamente constante.

Pero también presentan una serie de desventajas a tener en cuenta:

- Son motores caros de fabricar.
- Necesitan un controlador que generalmente tiene un precio elevado.
- Requieren un control complejo.

4.1.3. Motores de corriente continua

El motor de corriente continua con escobillas es el tipo de motor más extendido en el mercado.

Este tipo de motor está formado por el estator, que alberga unos imanes permanentes, y un rotor bobinado. Al circular corriente por las bobinas del rotor, se genera un campo magnético perpendicular al campo generado por los imanes permanentes. La interacción entre estos dos campos, genera un par de fuerzas que hace girar el eje del motor.

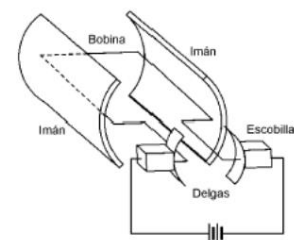


Ilustración 2. Esquema simplificado del motor

Estos motores son sencillos de controlar, ya que solo necesitan una corriente continua para funcionar. También se puede regular su velocidad mediante una señal PWM (Señal de ancho de pulso variable). Al igual que los motores Brushless, estos requieren de un sensado externo para

1. MEMORIA

conocer su posición. Actualmente hay una gran variedad de estos motores con sensor de posición incorporado.

Estos motores poseen una serie de ventajas respecto a las opciones anteriores:

- Control Sencillo. Como se ha dicho anteriormente son sencillos de controlar.
- Tienen un precio muy asequible.

Aunque también tienen una serie de desventajas:

- Debido a las escobillas se producen chispas y a largo plazo se genera un desgaste que impide el correcto funcionamiento.
- Relación par/velocidad no lineal

Debido a todas estas razones, y centrándose principalmente en su facilidad de control y su precio asequible, se ha optado por estos últimos motores para la implementación del sistema. El motor elegido es un Mitsumi M36 de la serie 4E.

4.2. Sensores de posición angular

Para medir la posición de los motores se estudiaron varias alternativas de sensores. Entre ellas destacan las siguientes:

4.2.1. Sensores de efecto Hall

Los sensores de posición angular de efecto Hall son sensores magnéticos, éstos están pensados para la medida de posición angular mediante tecnología magnética. Se cuenta con dos elementos un detector y un emisor, el emisor es principalmente un imán que, al variar su posición, también lo harán sus polos. Por este principio de funcionamiento, contamos con un sensor de ángulo absoluto. El detector interpreta la posición del emisor y calcula el ángulo de forma



Ilustración 3. Sensor Hall

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

absoluta. Estos sensores son inmunes al polvo, la humedad, o las vibraciones, además de tener una gran vida útil. Por el contrario, es muy sensible a interferencias de tipo electromagnético y además, su precio es muy elevado, por lo que se descarta el uso de este sensor en nuestros motores.

4.2.2. Resolvers

Los resolvers son sensores de posición angular inductivos. Están diseñados de tal modo que su coeficiente de acoplamiento entre el rotor y el estator varía según sea la posición angular del eje.

El sensor consta de dos bobinados en el estator, que detectan una señal senoidal de inducción, emitida por un tercer bobinado desde el rotor; una de las bobinas detectoras corresponde al seno y la otra al coseno (están ubicadas en posiciones separadas, por un ángulo de 90°). La ventaja de estos sensores es que al no tener rozamiento entre sus partes su tiempo de vida es elevado. También tiene la ventaja de ser inmune a agentes externos como el polvo, la humedad o la luz, pero por el contrario es muy susceptible a las interferencias electromagnéticas además de ser muy caros

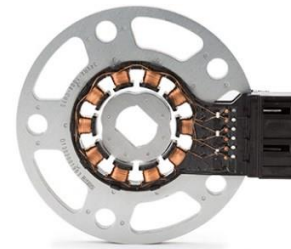


Ilustración 4. Detalle Resolver

4.2.3. Sensores angulares potenciométricos

Los sensores de posición angular potenciométricos se basan en la variación de la resistencia entre los terminales del sensor. Este tipo de sensores dispone normalmente de tres terminales, dos de ellos se corresponden a una resistencia fija y el tercer terminal se corresponde al cursor. El eje del sensor se acopla al eje del dispositivo del cual queremos medir su posición angular. En función de la posición del eje se produce una variación de la resistencia entre los terminales fijos del potenciómetro

1. MEMORIA

y el terminal del cursor. Alimentando los dos terminales de la resistencia fija del potenciómetro con una tensión continua y constante obtendremos una tensión en el terminal del cursor directamente proporcional a la posición angular. Este tipo de sensores, por lo general, son muy baratos, pero tienen poca vida útil y no son capaces de medir vueltas completas. Además, al igual que los sensores de efecto hall y los resolvers, para estos sensores necesitaríamos un conversor analógico-digital para trabajar con sus señales. Por lo que no es viable su utilización.

4.2.4. Encoders

Los encoders son sensores de posición angular ópticos que están formados por un disco con ranuras radiales ubicadas por lo general muy juntas en toda su circunferencia, o con líneas alternadas en color claro y oscuro, que giran frente a un fotosensor (o un conjunto de éstos, para más precisión), generando un pulso por cada ranura o cambio de color. Estos encoders son de tipo incremental. También pueden ser

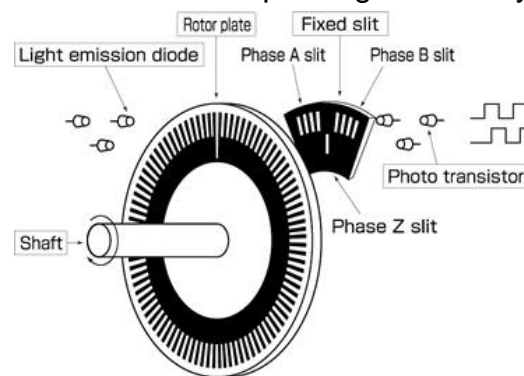


Ilustración 5. Esquema Encoder

absolutos, formados por un disco con un dibujo complejo, distribuido en anillos concéntricos que representan los bits de una palabra binaria. Deben tener un detector óptico por cada uno de estos anillos. Un disco con más anillos concéntricos ofrecerá más bits de resolución y dará un dato de posición angular más preciso. Los encoders ópticos son unos de los más utilizados en el mundo de la automatización y la robótica debido a su gran precisión, su bajo coste y su inmunidad a interferencias electromagnéticas y su alta estabilidad térmica, aunque este sensor es bastante sensible a vibraciones.

Debido a que podemos conectar directamente este sensor a las entradas digitales y su gran disponibilidad en el mercado, se buscaran

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

motores de corriente continua que lleven incorporado este sensor para desarrollar el proyecto.

4.3. Microcontrolador

Otro elemento a elegir es el microcontrolador, que es el elemento principal de nuestro sistema. Para ello se han estudiado varias placas de desarrollo, ya que son más sencillas de programar y de ensamblar que los microcontroladores sueltos.

Entre las opciones estudiadas destacan las siguientes:

4.3.1. Arduino

Arduino es el fabricante de placas de desarrollo más popular de los últimos años. Esto es debido a su precio asequible, la gran cantidad de información, tutoriales y documentación que hay al respecto, pero sobre todo por su entorno de programación sencillo y cercano. Además, debido a la gran cantidad de usuarios de esta placa, existen una infinita cantidad de librerías y muchas plataformas de ayuda.



Ilustración 6. Arduino Uno

Las placas más populares de Arduino están basadas en un microcontrolador de 8 bits, y suelen tener un total de 20 entradas/salidas (14 digitales y 6 analógicas). Hay modelos más avanzados, con más entradas y salidas o con un microcontrolador de 32 bits, pero el precio de estos es elevado.

1. MEMORIA

4.3.2. STM32F4 Discovery

Esta placa de desarrollo de fabricada por ST incorpora un potente integrado de 32 bits y arquitectura ARM Cortex-M4. Esta plataforma cuenta con un gran número de entradas y salidas, además de un procesador de audio y algunos sensores, como un acelerómetro.

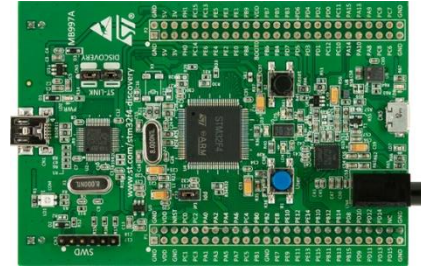


Ilustración 7. STM32F4 Discovery

Aunque en menor medida que Arduino, esta plataforma también cuenta con una gran variedad de documentación y códigos de ejemplo. Uno de los principales problemas de esta placa es que existen pocos entornos de desarrollo compatibles.

4.3.3. STM32 Núcleo

Siguiendo con el fabricante ST, nos centramos en las nuevas placas STM32 Nucleo. Estas placas son similares a Arduino, de hecho, las Shields, que son circuitos modulares que se acoplan sobre él para añadir nuevas funciones, son compatibles entre las dos marcas. Además, las placas de ST disponen de más salidas.

La gran ventaja de estas placas es su precio, ya que son muy asequibles. Además, incorporan un microcontrolador de 32 bits de bajo consumo perfecto para aplicaciones de gran rendimiento. Además, esta placa esta soportada por una gran variedad de plataformas de desarrollo.

De todos los modelos que ofrece nos decantamos por el modelo Nucleo-F411RE, de la familia Nucleo-64. Esta decisión se basa en las altas prestaciones que ofrece esta placa por su bajo coste y por la variedad de opciones a la hora de programarla.



Ilustración 8. Núcleo F411RE

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

4.4. Drivers motores

Para controlar los motores necesitaremos un puente en H. Un puente en H es un circuito electrónico que permite que nuestro motor sea capaz de girar en ambos sentidos.

Algunas características a tener en cuenta a la hora de elegir un buen integrado es el voltaje de alimentación, ya que nuestra placa funciona a 3,3V, y la corriente de salida máxima. Entre las opciones posibles, destacan las siguientes:

4.4.1. L293

El L293 es un integrado muy popular para controlar motores de corriente continua de poco tamaño. Es capaz de controlar dos motores por cada circuito integrado. Además, detecta un nivel lógico alto a partir de 2.3V, así que se podría controlar con nuestra placa.

Los motores que vamos a utilizar, consumen una corriente de 350mA en vacío, pero se disparan hasta los 9 A cuando aumenta su carga. Este integrado es capaz de entregar 1 A de corriente máxima, por lo que debemos descartar su uso.

4.4.2. L298

Este integrado también es muy conocido en el mundo "Do It Yourself" debido a su facilidad de conexión y la gran cantidad de información al respecto. Al igual que el integrado anterior, este también es capaz de controlar dos motores. También se puede controlar con los 3.3V, aunque su corriente de salida también es algo limitada (2 A) y necesita un disipador de calor.

La corriente de salida limitada y el alto consumo de corriente de este integrado, además de su elevado precio, hacen que descartemos esta opción.

1. MEMORIA

4.4.3. VNH2SP30

Esta opción no es tan conocida como las anteriores, aunque se popularizó con la shield de Arduino “Monster Moto Shield”.

Los VHN2SP30 son capaces de entregar hasta 30 A de salida y 41V, por lo que es más que suficiente para controlar nuestros motores. Su baja impedancia interna hace que tenga un consumo de corriente muy bajo y que no necesite disipador. Además, se pueden alimentar y controlar desde 3.25V hasta 7.5 V y su precio es muy competitivo. Por todo esto motivos, estos controladores son los elegidos.



Ilustración 9. VNH2SP30

4.5. Plataforma de programación

Una vez elegido todo el hardware, es momento de decidir con que plataforma vamos a programar nuestra placa. El fabricante recomienda utilizar los entornos de desarrollo IAR, ARM Keil o mbed, aunque los más populares son estos dos últimos.

4.5.1. Keil μ vision

Keil μ vision, desarrollado por ARM, es un software de programación que incorpora herramientas de depuración, desarrollo de código y comunicación para microcontroladores.

Aunque en su web hay bastantes tutoriales y documentación para iniciarse en la plataforma, su lenguaje de programación, pese a ser C++, es más complejo y se requiere un gran conocimiento de microcontroladores y programación.

4.5.2. ARM mbed

ARM mbed es una plataforma on-line que permite programar y compartir código en “la nube”, y tiene la ventaja de poder ser utilizado con cualquier dispositivo con conexión a internet. Esta plataforma contiene una gran cantidad de documentación y códigos de ejemplo para iniciarse

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

en la programación de microcontroladores. Su programación es muy similar a Arduino así que es muy sencillo y accesible.

Por todos estos motivos, se ha decidido trabajar sobre esta plataforma.

4.6. Trayectoria

Cuando se trabaja con máquinas de control numérico, brazos robóticos o cualquier máquina que requiera un desplazamiento es importante generar una trayectoria con el objetivo que se produzca un desplazamiento suave, sin acelerones ni frenazos bruscos o velocidades excesivas.

4.6.1. No actuación

La primera opción que se propone es no crear ninguna trayectoria, es decir, la señal de error del PID sería la posición deseada menos la posición actual. Esto provocaría que cuando la distancia a recorrer sea muy grande, la señal de salida del PID saturaría y el motor giraría a máxima velocidad. Debido a esto se produciría una aceleración muy elevada al inicio del movimiento y una brusca detención al final del recorrido. Estas aceleraciones tan pronunciadas pueden causar que el sistema se inestabilice o, en los casos en los que la carga sea muy elevada, puede provocar la rotura de correas, poleas u otros elementos mecánicos. Otro problema ligado a la ausencia de programación de trayectorias es la falta de coordinación de los ejes, es decir, si los ejes de nuestra máquina van a recorrer diferentes distancias, no llegarán al mismo tiempo a su objetivo, además, no podremos ajustar parámetros como su velocidad.

Para intentar evitar todos estos problemas, se descartará esta idea.

4.6.2. Trapezoidal

El perfil trapezoidal cuenta con tres fases: la primera es una fase de aceleración, en la que aumenta su velocidad progresivamente hasta

1. MEMORIA

alcanzar la velocidad máxima establecida; a continuación, una etapa en la que el motor avanza a velocidad constante y finalmente, se reduce la velocidad hasta llegar al objetivo.

Las gráficas de posición, velocidad y aceleración para este patrón de movimiento son las siguientes:

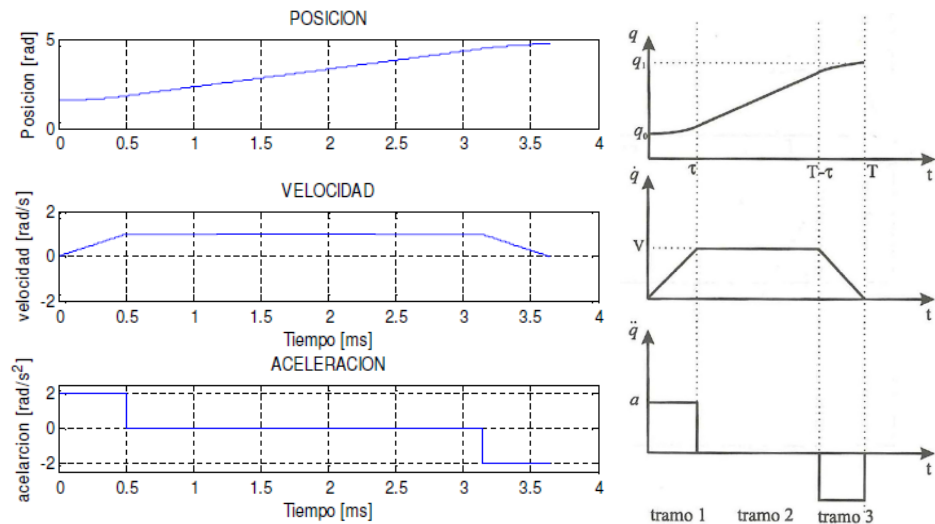


Ilustración 10. Gráfica trayectoria trapezoidal

Lo que haremos será generar puntos de referencia más cercanos para el PID, de manera que cada determinado tiempo se generará un nuevo objetivo y de esta manera podremos controlar la velocidad y la posición de nuestros motores.

Los cálculos para generar este tipo de trayectoria son sencillos:

En primer lugar, hay que determinar si nuestro sistema alcanzara la velocidad máxima establecida. Para ello seguimos la siguiente ecuación:

$$Si L \leq \frac{v^2}{a}$$

Donde “L” es la distancia a recorrer, “V” la velocidad máxima establecida y “a” la aceleración, no se alcanzará la velocidad máxima, por lo tanto, en este caso, solo tendrá los tramos 1 y 3, es decir, no habrá un periodo de velocidad constante. Ahora hay que calcular “t”, que corresponde al tiempo donde nuestro sistema deja de ganar velocidad y empieza a decelerar.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

$$t = 2 \cdot \sqrt{\frac{L}{a}}$$

En el caso de que si se alcance la velocidad máxima:

$$L > \frac{V^2}{a}$$

Habrá que calcular dos tiempos, “t1” que corresponde al tiempo en que nuestro sistema deja de acelerar y se mantiene a velocidad constante, y “t2” que corresponde al tiempo en que nuestro sistema deja de ir a velocidad constante para empezar a decelerar.

$$t1 = \frac{V}{a} \quad t2 = \frac{L}{V}$$

Una vez obtenidos los tiempos, obtendremos las referencias de posición para nuestro sistema:

Para el tramo 1:

$$referencia = 0.5 \cdot a \cdot t^2$$

Donde t es el tiempo transcurrido desde que se ha iniciado el movimiento.

Para el tramo 2:

$$referencia = e_{t1} + V \cdot (t - t1)$$

Donde e_{t1} es la posición al finalizar el primer tramo.

Para el tramo 3:

$$referencia = e_{t2} + V \cdot (t - t2) - 0.5 \cdot a \cdot (t - t2)^2$$

Donde e_{t2} es la posición al finalizar el segundo tramo.

Como ya hemos comentado anteriormente, si la trayectoria no llega a la velocidad máxima establecida, no se ejecutarán los cálculos del tramo 2, por lo tanto, solo serán necesarios los cálculos de los tramos 1 y 3.

Cuando controlamos varios ejes simultáneamente es importante que todos ellos lleguen al objetivo a la vez. Para lograr esto es necesario que

1. MEMORIA

el tiempo total del desplazamiento sea el mismo para todos ellos, por lo tanto, t_1 y t_2 serán iguales. T_1 y t_2 se deben calcular con la distancia del eje que tenga mayor recorrido. Por lo tanto, lo que será diferente para cada eje serán las aceleraciones y las velocidades. Estas se calcularán de la siguiente manera:

$$a_{eje} = a_{max} \cdot \frac{L_{eje}}{L_{max}}$$

$$V_{eje} = a_{eje} * t_1$$

Estos valores se sustituirán en las ecuaciones anteriores para generar las referencias en cada eje.

4.6.3. Perfil en "s"

El perfil en S es más complejo que el anterior, pero también se logra una mayor suavidad en el desplazamiento. Esta trayectoria se divide en cinco fases:

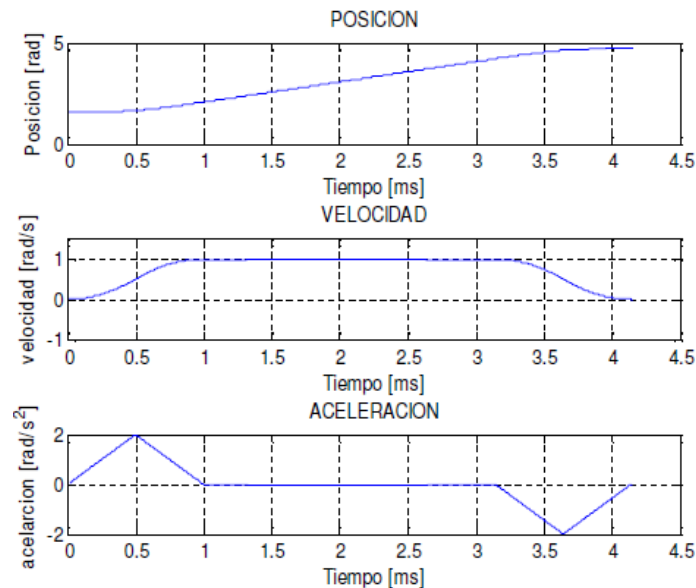


Ilustración 11. Gráficas perfil en "s"

Como observamos en la imagen anterior, la fase de aceleración constante que veíamos en la trayectoria trapezoidal, se sustituye por dos etapas: una de aceleración progresiva hasta t_1 , y una deceleración

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

progresiva hasta t_2 , que es donde empieza el tramo a velocidad constante. Una vez se ha finalizado este tramo (t_3), empieza una deceleración progresiva hasta t_4 y finalmente aumenta la aceleración gradualmente hasta llegar al objetivo establecido. Llegados a este punto debemos mencionar la importancia del Jerk, que es la variación de la aceleración respecto al tiempo, y el responsable de la suavidad del movimiento. En el caso de esta trayectoria se producen Jerks de un valor progresivo y finito cuando la aceleración cambia, esto provoca una mayor suavidad que en el caso de la trayectoria trapezoidal, que provoca unos Jerks de valor infinito en los cambios de aceleración.

Aunque es interesante este perfil debido a su suavidad de movimiento, descartamos esta opción por su complejidad de cálculo.

4.6.4. Senoidal

Esta curva esta relaciona con la anterior, pero a diferencia de esta, la aceleración aumenta de forma senoidal. Esta trayectoria es una de las más suaves, ya que los Jerks asociados a este perfil son muy pequeños. Las gráficas de posición, velocidad y aceleración son las siguientes:

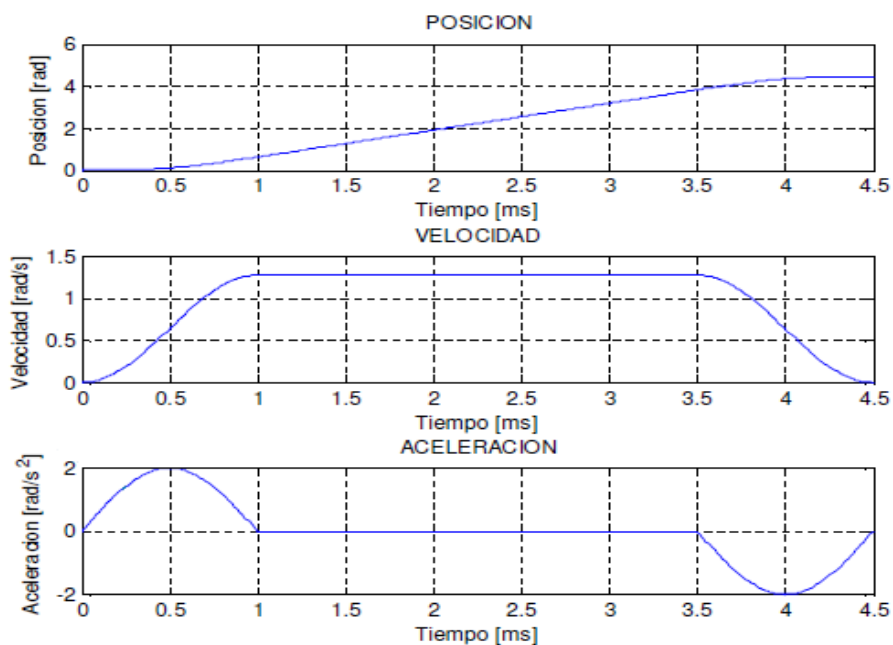


Ilustración 12. Gráfica trayectoria senoidal

1. MEMORIA

Debido a la complejidad de cálculo de las dos últimas trayectorias, nos quedaremos con la trapezoidal, ya que para implementar nuestro proyecto será más que suficiente.

5. Descripción de la solución adoptada.

5.1. Montaje de pruebas.

El primer paso en el desarrollo de nuestro proyecto, es el montaje del sistema con un solo motor, para tratar de reconocer el comportamiento de este y lograr así el comportamiento deseado.

Para ello deberemos conectar las salidas del encoder a la placa Nucleo-f411RE y los bornes del motor a la salida del driver VHN2SP30 respetando la polaridad. Como podemos ver en la siguiente imagen, en el circuito impreso del motor, están nombrados todos los pines del conector JST.

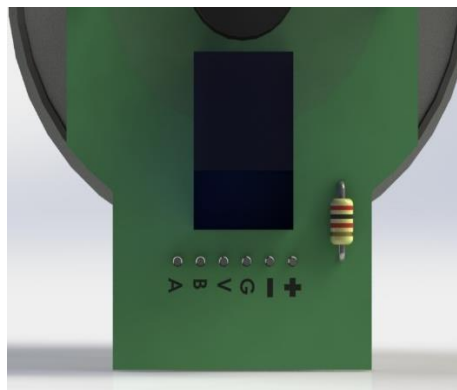


Ilustración 13. Detalle PCB motor

Para realizar las conexiones del motor, primero introduciremos el conector macho del cable en la clavija del motor y a continuación se unirán los cables siguiendo esta tabla:

A	Pin D6
B	Pin D7
V	A los 3.3V de la placa Nucleo
G	A cualquier conector GND de la placa Nucleo
-	Al conector "Out B" del driver
+	Al conector "Out A" del driver

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Una vez conectados todos los pines del motor, procederemos a conectar el driver de los motores a la Nucleo. Al igual que con el motor, el circuito impreso del driver también lleva rotulado el nombre de cada pin por la parte inferior. Este driver se conectará de la siguiente forma:

+5V	A los 3.3V de la placa Nucleo
GND	A cualquier conector GND de la placa Nucleo
EN	A los 3.3V de la placa Nucleo
CS	No conectado
INA	Pin D4
INB	Pin D5
PWM	Pin D10
+	Al positivo de la fuente de alimentación
-	A la masa de la fuente de alimentación

Una vez realizadas estas conexiones, encenderemos la fuente de alimentación, configurada para entregar 12 V en corriente continua. También conectaremos la placa al puerto USB del ordenador.

Para poder observar la respuesta del motor se diseña con el software SolidWorks, un soporte para anclar el motor a una base fija y un puntero para observar su posición. Posteriormente, con una impresora 3D, se imprimen estas piezas, cuyos planos se encuentran en su respectivo documento, en plástico PLA. Posteriormente se ensambla el conjunto, que debería quedar de la siguiente manera:

1. MEMORIA



Ilustración 14. Modelo montaje provisional

5.2. Ajuste de la respuesta

Una vez realizado el ensamblaje, el siguiente objetivo es implementar el código capaz de reconocer el comportamiento de los motores.

Para empezar a familiarizarse con la plataforma de desarrollo mbed, con la que vamos a implementar el código, es importante descargar los códigos de ejemplo y trabajar sobre ellos. El compilador de mbed presenta la siguiente interfaz:

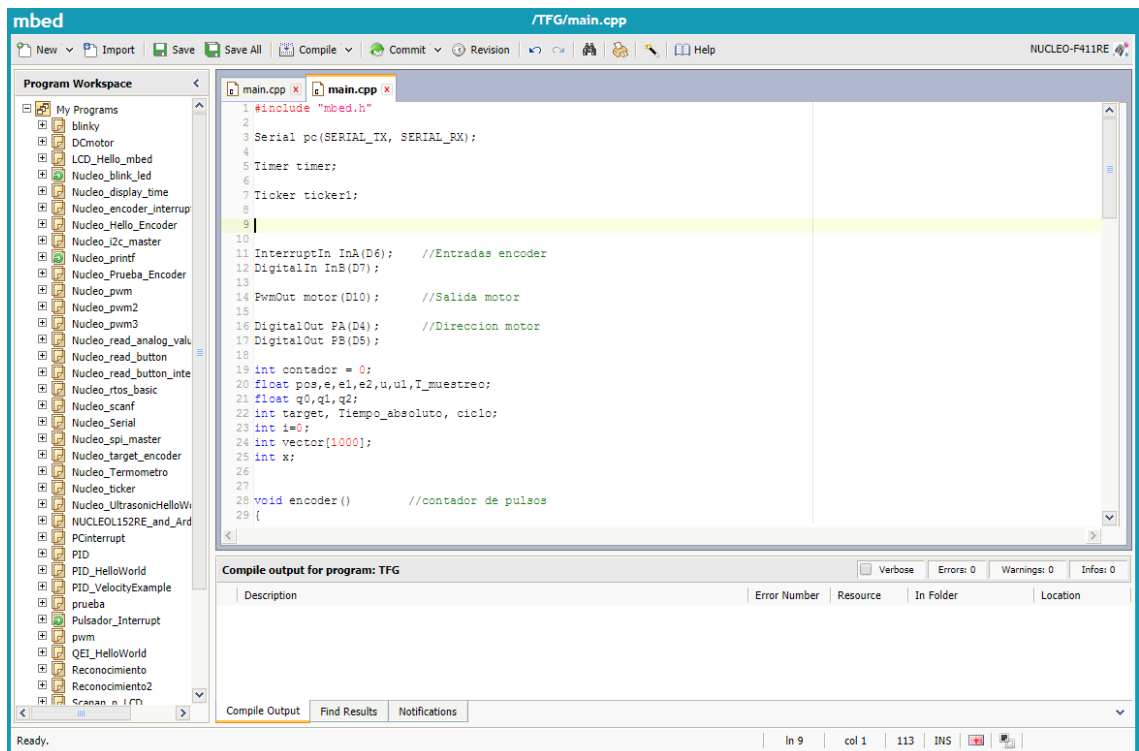


Ilustración 15. Ventana principal del compilador

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Como se observa en la imagen, en la columna de la izquierda tenemos los códigos que hemos creado o descargado. Desde la pestaña “Import”, en la parte superior, podremos descargar un gran número de ejemplos específicamente creados para nuestra placa.

En la parte central de la ventana tenemos la zona de programación donde leer los códigos y trabajar sobre ellos. Como se ha dicho anteriormente, esta plataforma trabaja con el lenguaje de programación C++. Una vez realizados varios proyectos simples relacionados con materias interesantes en nuestro controlador (comunicación serie, PWM, interrupciones), procederemos a programar el código que interpretara la respuesta de los motores.

La idea principal de este código es mandar una referencia de posición, y que el motor llegue hasta ella mediante la implementación de un PID. Cada determinado instante, guardaremos la posición en un vector, y al llegar a las 1000 muestras, la placa transmitirá este vector a nuestro ordenador mediante el puerto serie. Después se realizarán las gráficas asociadas a estos datos para estudiar su respuesta. Cada vez asignaremos unos valores diferentes a las constantes del PID y al tiempo de muestreo, hasta que logremos obtener una respuesta que se adecue a nuestras necesidades.

Para crear el programa dividiremos el código en diferentes partes:

Cabecera: como todo código destinado a microcontroladores, en primer lugar, añadiremos las librerías a utilizar, configuraremos los pines de entrada y salida, los recursos que vayamos a utilizar, como los “Timers”, “Tickers” o la comunicación serie y por último también declararemos las variables que vayamos a utilizar.

El pin A del encoder se configurará como una interrupción. Para ello se utiliza el comando *InterruptIn*. Mas adelante, en el main, deberemos configurar el tipo de interrupción. *InA.rise(&encoder)* Con este comando se ejecutará la función “encoder” cada vez que se detecte un flanco de subida en el pin al que esté conectado el canal A del encoder. El resto de entradas se configurarán como entradas digitales con el comando *DigitalIn*. Las

1. MEMORIA

salidas de control de dirección del driver de los motores se configurarán como salidas digitales tecleando *DigitalOut*. La única salida diferente será la señal PWM del motor, que se configurará como *PwmOut*. La señal PWM nos permite generar una señal cuadrada con el ancho de pulso que deseemos, de esta manera, el valor eficaz de la tensión será directamente proporcional a el porcentaje que ocupe el nivel alto en cada periodo de la salida.

A continuación, también se declaran las variables, como las constantes del PID o el vector que almacenara los valores de posición.

Contador de pulsos: La siguiente función que se va a desarrollar es el contador de pulsos. Esta función, como hemos comentado, se ejecutará cada vez que se detecte un flanco de subida en el correspondiente pin de entrada. El objetivo de esta función será almacenar la posición del motor mediante la medición de los pulsos del encoder. Al ejecutarse esta función, se leerá el estado de los pines A y B del encoder, y siguiendo la máquina de estados que se muestra a continuación, se incrementará o decrementará la variable de posición.

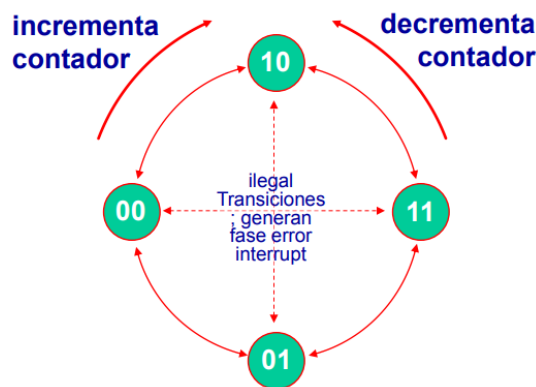


Ilustración 16. Máquina de estados del encoder

PID: La función del PID se llama cada tiempo deseado mediante la función ticker. Esta herramienta se declara así: `ticker1.attach(&funcion, T_muestreo)`, de esta manera se ejecutara “funcion” cada T_muestreo (en segundos). “función” está formada por una llamada a la función PID y el vector de posición.

Para implementar el PID primero se calculará la señal de error, que será la resta de la posición del motor al objetivo de posición. A continuación,

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

calcularemos la salida del PID con la ecuación detallada en el apartado de estudio de necesidades, pero para utilizar esta ecuación, hay que discretizar esta ecuación y convertirla a ecuación en diferencias. La ecuación discretizada del regulador PID es la siguiente:

$$G_{PID} = \frac{q_0 \cdot z^2 + q_1 \cdot z + q_2}{z \cdot (z - 1)}$$

Por lo tanto, su ecuación en diferencias tendrá la siguiente forma:

$$u = u_1 + q_0 \cdot e + q_1 \cdot e_1 + e_2 \cdot q_2$$

En la anterior ecuación u_1 representa la salida del PID en el instante anterior, e es el error, e_1 es el error en el instante anterior, y e_2 , por lo tanto, será la señal de error en el instante anterior a este. q_0 , q_1 y q_2 serán los valores que deberemos modificar, y de estos depende la acción proporcional, la integral y la derivativa.

$$q_0 = k_p \cdot \left[1 + \frac{T_D}{T_M} + \frac{T_M}{2 \cdot T_i} \right]$$

$$q_1 = k_p \cdot \left[-1 - \frac{2 \cdot T_D}{T_M} + \frac{T_M}{2 \cdot T_i} \right]$$

$$q_2 = k_p \cdot \left[\frac{T_D}{T_M} \right]$$

Donde k_p es la ganancia proporcional, T_D el tiempo derivativo y T_i el tiempo integral. T_M será el tiempo de muestreo, que debe ser el mismo que el asociado al Ticker.

Después de la ecuación en diferencias del PID debemos ejecutar un control de saturación. Ya que en la señal PWM el 0 representa un ciclo de trabajo del 0% y el 1 del 100%, debemos evitar que la salida del PID se

1. MEMORIA

mayor que 1 o inferior a -1. Esto se soluciona con un if, que convertirá la salida del PID a 1 cuando esta sea mayor.

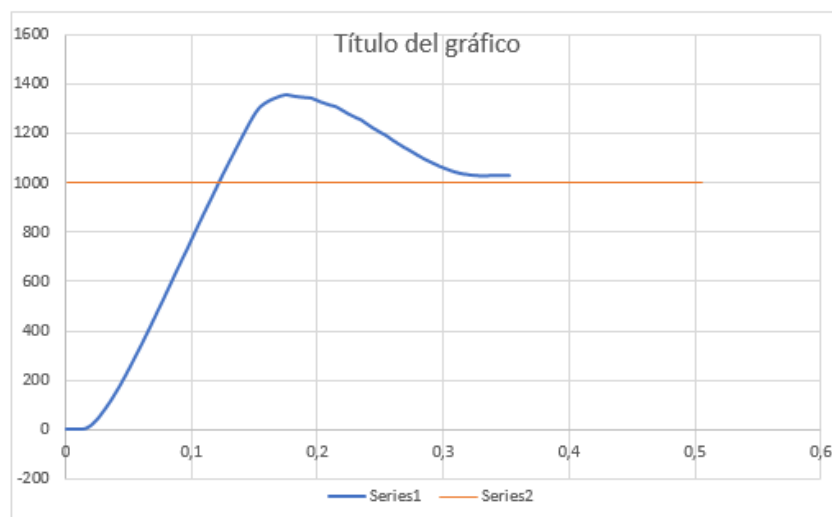
También debemos implementar el control de giro del motor. Cuando la salida del PID sea positiva, el motor deberá girar en sentido horario. Para ello el pin A del driver debe estar a un nivel lógico bajo y el pin B a un nivel lógico alto. Si la señal es negativa el pin A estará a nivel alto y el pin B a nivel bajo y la salida del motor será el módulo de la salida del PID. Otra vez implementamos esta función con un if.

Main: en el main de nuestro código implementaremos la comunicación con nuestro ordenador. Una vez que hemos almacenado 1000 muestras en el vector de posición, el programa, mediante un printf, transmitirá este vector por el puerto USB.

Una vez finalizado el programa procedemos a analizar las respuestas. Para ello calculamos las constantes del PID y representamos su respuesta. Variando K_p , T_d , y T_i trataremos de mejorar la respuesta.

Partimos de unas constantes aleatorias:

$$q_0=0.001 \quad q_1=-0.001 \quad q_2=0.0001$$

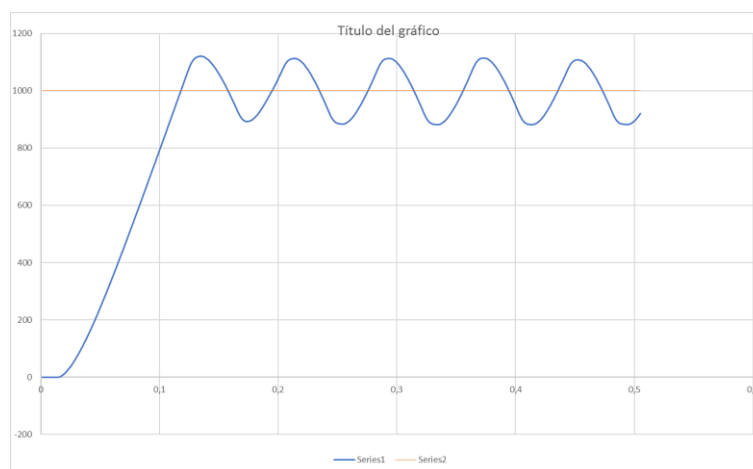


Como podemos observar la respuesta de nuestro sistema ante un escalón de amplitud 1000, presenta una gran sobreoscilación.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Mediante el uso de una hoja de Excel calcularemos las constantes q_0 , q_1 y q_2 con la ganancia proporcional, el tiempo integral y el tiempo derivativo que introduzcamos. Probamos con nuevos valores y analizamos la mejor respuesta.

Tm	K	Ti	Td	q0	q1	q2
0,001	0,008	0,01	0,0001	0,0092	-0,0092	0,0008



Como podemos observar se ha reducido la sobreoscilación, pero la respuesta se vuelve inestable en régimen permanente.

Tm	K	Ti	Td	q0	q1	q2
0,001	0,006	0,08	0,002	0,0180375	-0,0299625	0,012

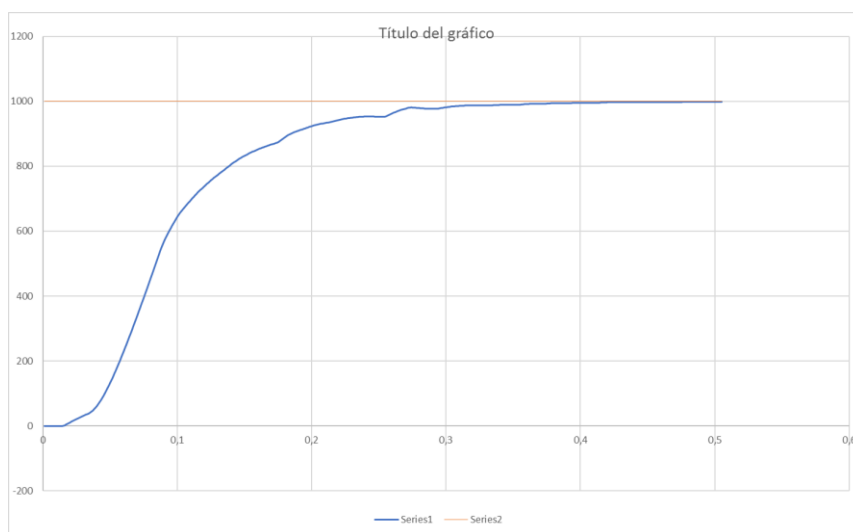


1. MEMORIA

Tm	K	Ti	Td	q0	q1	q2
0,001	0,006	0,08	0,004	0,0300375	-0,0539625	0,024

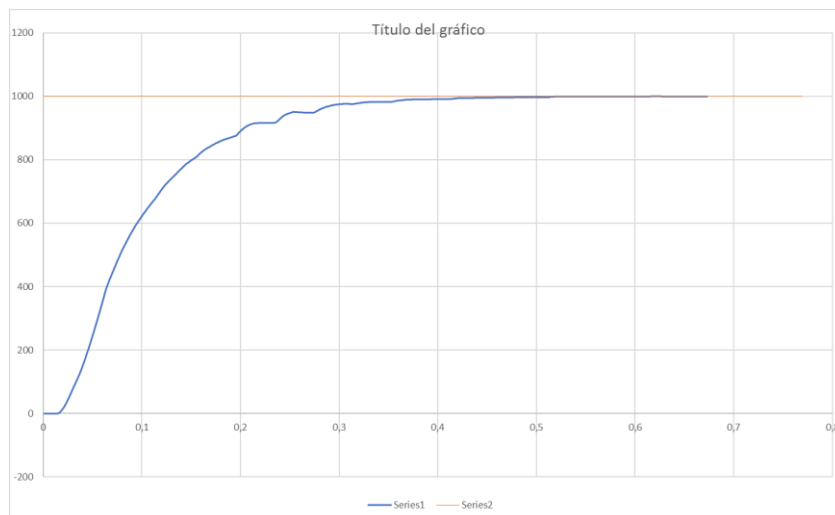


Tm	K	Ti	Td	q0	q1	q2
0,001	0,009	0,08	0,008	0,08105625	-0,15294375	0,072

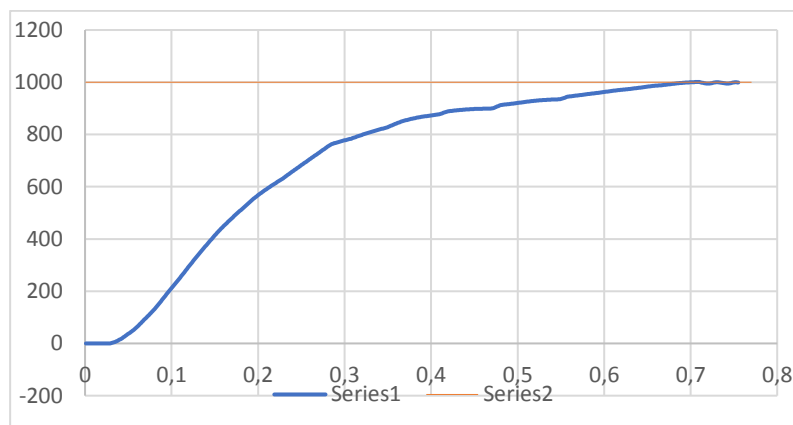


Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Tm	K	Ti	Td	q0	q1	q2
0,001	0,015	0,09	0,008	0,135083333	-0,254916667	0,12



Tm	K	Ti	Td	q0	q1	q2
0,001	0,025	0,09	0,02	0,525138889	-1,024861111	0,5



Después de analizar las respuestas nos quedaremos con los valores

Tm	K	Ti	Td	q0	q1	q2
0,001	0,015	0,09	0,008	0,135083333	-0,254916667	0,12

1. MEMORIA

Con estos valores se logra la respuesta más rápida y con menos sobreoscilaciones, por lo tanto, estos valores serán los que utilizemos en nuestro PID.

5.3. Montaje final

Una vez obtenidos los parámetros para lograr una respuesta correcta de los PIDs, es momento de realizar el montaje final con los tres motores, que representaran los ejes X, Y, Z de una máquina. Para el eje X, se ha representado el eje de una impresora 3D Prusa i3. Basándonos en el modelo original se han creado las nuevas piezas con sus respectivas modificaciones para albergar este tipo de motor. Los ejes Y y Z se han implementado con el montaje anterior, añadiendo unas etiquetas correspondientes al eje. Nuevamente estas piezas se han diseñado con el software SolidWorks y se han impreso en plástico PLA con una impresora 3D modelo Prusa i3.



Ilustración 17. Modelo Eje X

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados



Ilustración 18. Modelo Eje Y



Ilustración 19. Modelo Eje Z

Para el montaje del eje X, se han impreso las piezas creadas con SolidWorks, y los tensores y el carro que han sido obtenidos del repositorio de modelos 3D “Thingiverse” [THI 14]. Se han cortado dos varillas de acero inoxidable de 8mm de diámetro a 370mm de longitud. Se han insertado las dos varillas a la pieza que albergará el motor y se insertan dos rodamientos lineales tipo LM8UU en la varilla superior y otro en la inferior. A continuación, se inserta la pieza que supone el final del eje y se monta el carro sobre los tres rodamientos lineales. Después, se presenta el motor en su alojamiento y se fija atornillándolo con dos tornillos de métrica 3 y 5 mm de longitud que se introducirán por los agujeros de la cara superior. Se colocará la polea en el eje del motor, y se pasará la correa desde esta polea hasta la situada al final del eje y se insertarán a presión ambos terminales en el carro móvil.

Para montar los ejes Y y Z, se introduce el motor en su soporte (tal como se ve en la imagen superior) y se introduce una tuerca de métrica 3 en la ranura que hay en el soporte para este fin. Se atornilla con un tornillo M3 de

1. MEMORIA

20mm y se aprieta hasta que el motor no gire sobre el soporte. Finalmente se inserta el collarín con la etiqueta por la parte frontal del motor y se introduce el cursor en el eje del motor a presión.

Una vez montados todos los ensamblajes, deberemos conectar los motores y los drivers a la tarjeta Nucleo siguiendo estas tablas:

Motor Eje X	
A	Pin D6
B	Pin D5
V	A los 3.3V de la placa Nucleo
G	A cualquier conector GND de la placa Nucleo
-	Al conector "Out B" del driver
+	Al conector "Out A" del driver

Driver Motor Eje X	
+5V	A los 3.3V de la placa Nucleo
GND	A cualquier conector GND de la placa Nucleo
EN	A los 3.3V de la placa Nucleo
CS	No conectado
INA	Pin D4
INB	Pin D2
PWM	Pin D3
+	Al positivo de la fuente de alimentación
-	A la masa de la fuente de alimentación

Motor Eje Y	
A	Pin D7
B	Pin D13
V	A los 3.3V de la placa Nucleo
G	A cualquier conector GND de la placa Nucleo
-	Al conector "Out B" del driver
+	Al conector "Out A" del driver

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Driver Motor Eje Y	
+5V	A los 3.3V de la placa Nucleo
GND	A cualquier conector GND de la placa Nucleo
EN	A los 3.3V de la placa Nucleo
CS	No conectado
INA	Pin D14
INB	Pin D15
PWM	Pin D9
+	Al positivo de la fuente de alimentación
-	A la masa de la fuente de alimentación

Motor Eje Z	
A	Pin A5
B	Pin A4
V	A los 3.3V de la placa Nucleo
G	A cualquier conector GND de la placa Nucleo
-	Al conector "Out B" del driver
+	Al conector "Out A" del driver

Driver Motor Eje Z	
+5V	A los 3.3V de la placa Nucleo
GND	A cualquier conector GND de la placa Nucleo
EN	A los 3.3V de la placa Nucleo
CS	No conectado
INA	Pin A2
INB	Pin A1
PWM	Pin A3
+	Al positivo de la fuente de alimentación
-	A la masa de la fuente de alimentación

1. MEMORIA

A la hora de conectar el pin A de cada encoder es importante tener en cuenta que no compartan el mismo número de puerto, ya que la placa Nucleo utiliza el mismo vector de interrupción para todos los puertos con el mismo número. De esta manera, si conectamos el pin A de un encoder al pin D11 de la placa, que corresponde con el pin PA_7 del microcontrolador, no podremos conectar otro pin A de otro motor al pin D9 de la placa, ya que este se corresponde con el pin PC_7 del microcontrolador.

Una vez conectado todo nos aseguraremos que la fuente de alimentación este configurada para entregar 12 V.

5.4. Programación

Finalmente, el último paso en el desarrollo del proyecto es la programación del controlador con los tres motores y el generador de trayectorias. Al igual que en el código de reconocimiento de los motores, también dividiremos el código en varios apartados.

Cabecera: Como en el caso anterior, lo primero será incluir la librería de mbed, declarar los Tickers y configurar las entradas y salidas teniendo en cuenta las tablas del apartado anterior. Las nuevas variables se nombrarán con la coletilla _X, _Y o _Z en función del eje en las que vayan a tratarse.

Contadores de pulsos de encoder: esta función es la misma que para el caso del reconocimiento del motor, pero ahora cada variable tendrá un nombre dependiendo del eje, por lo tanto, esta función nos devolverá la posición del motor mediante las variables X_pos, Y_pos y Z_pos.

Main: el programa principal de nuestro código empezara asignando los valores de las constantes de los PIDs y el tiempo de muestreo. A continuación, en el bucle principal, lo primero que se hará es leer, mediante un “scanf,” el objetivo de cada eje, es decir, hasta donde se tienen que desplazar. La instrucción que se mandará desde el ordenador es de tipo X0 Y0 Z0, donde 0 es el objetivo de cada eje. A continuación, tendremos que implementar un control de límite de posición para evitar que el cabezal de nuestra maquina se desplace mas allá de la longitud del eje. De nuevo, esta función se implementa con un if.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Después, se calculará la distancia a recorrer máxima. Con esta función “ $max_target=(dY>dZ)?dY:dZ$ ” se almacenara en la variable max_target, la mayor distancia a recorrer. Una vez conocida la máxima distancia, en primer lugar, se averiguará si el desplazamiento alcanzará la velocidad máxima establecida, mediante las ecuaciones adjuntas en la sección de estudio de alternativas, perfil trapezoidal. Una vez sepamos si alcanzará la velocidad máxima, se obtendrán t1 y t2 para el perfil de movimiento. Este trozo de código se ha implementado del siguiente modo:

```

if(max_target<=((v^2)/a)){ // calculo t1 t2
    triang=max_target/a;
    t1=t2=2*sqrt(triang);
}
else if(max_target>((v^2)/a)){
    t1=V/A;
    t2=-V/A+max_target/V+t1;
}

```

El siguiente paso es calcular las aceleraciones y velocidades correspondientes a cada eje. Al eje que mayor desplazamiento tenga que realizar se le asignara la velocidad y la aceleración máxima. Un ejemplo de estos cálculos, en el caso en el que el desplazamiento máximo se produzca en el eje X, es el siguiente:

```

if(dX == max_target){
    a_X=a;
    v_X=v;
    a_Y=a*(d_Y/max_dist);
    v_Y=a_Y*t1;
    a_Z=a*(d_Z/max_dist);
    v_Z=a_Z*t1;
}

```

Con ayuda del comando “printf”, se ha implementado una herramienta para mostrar por el puerto serie algunos parámetros como t1 y t2 o las aceleraciones de cada eje. También se han implementado unas herramientas de depuración que mostraran los puntos de la trayectoria y la posición de los motores. Estas últimas funciones se comentarán en la versión definitiva del programa para que no sean compiladas.

1. MEMORIA

El final del código principal consistirá en poner a 0 el contador de tiempo de las trayectorias y cambiar el estado de una variable booleana a verdadero. Esta variable activará o desactivará la generación de trayectorias.

PID: Las funciones de los PID en este caso, se dividen en dos partes. Al igual que en el código anterior, esta función será llamada por un Ticker. En el inicio del código del PID tenemos el generador de trayectorias.

```
if(move_ave){
  if(cs<t1){
    X_point=0.5*a_X*cs*cs;
    et1_X=X_point;
  }
  if(cs>=t1 && cs<t2){
    X_point=et1_X+v_X*(cs-t1);
    et2_X=X_point;
  }
  if(cs>=t2 && X_point<=X_Target){
    X_point=et2_X+v_X*(cs-t2)-0.5*a_X*(cs-t2)*(cs-t2);
  }
}
if (X_dirPos == false) {
  X_point = -X_point;
}
X_point=X_init+X_point;
```

Como se observa en la captura anterior, el generador de trayectorias solo se activa si la variable booleana está activa. A continuación, el código realiza los cálculos asociados al perfil trapezoidal, que dependerán de la aceleración y la velocidad que hemos calculado en la sección “main” y dependerán también de “cs” que es una variable que almacena el tiempo transcurrido en segundos. La salida de estas ecuaciones es un punto que tiene como referencia el punto de inicio del movimiento. El punto que debe recibir el PID debe estar referenciado al origen. Para ello convertimos este punto a negativo, en el caso de que la dirección sea negativa, y sumamos el punto de partida del movimiento, de esta manera el error del PID estará referenciado sobre el origen de nuestra máquina.

A continuación de este código se encuentran las ecuaciones del PID, que se encargaran de que los motores vayan siguiendo las referencias que cree la función anterior:

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

```
//PID
e_X=X_point-X_pos; //Calculo error

u_X=u1_X+q0_X*e_X+q1_X*e1_X+e2_X*q2_X; //PID

if(u_X>1) {
    u_X=1; //Proteccion saturacion
}
if(u_X<-1) {
    u_X=-1;
}
if(u_X>0) { //Control PWM
    PA_X=0;
    PB_X=1;
    motorX=u_X;
}
if(u_X<0) {
    PA_X=1;
    PB_X=0;
    motorX=0-u_X;
}
e2_X=e1_X; //Actualizacion de variables
e1_X=e_X;
u1_X=u_X;
```

Como hemos comentado, este código se encarga de que la salida del PID no sea mayor de 1 o inferior a -1 y controla la dirección del motor mediante los pines del driver. Además, al final de este código se actualizan las variables del error y de la posición anterior del PID. Una vez realizado el código se cargará a la tarjeta Nucleo y se testeará su correcto funcionamiento.

Finalmente se hacen pruebas variando su velocidad máxima y su aceleración, y se comprueba que efectivamente varíe la velocidad de los desplazamientos. Para asegurarse que funcione correctamente se prueba con desplazamientos muy pequeños (2, 3mm...) y desplazamientos hasta el final del eje. Cuando trabajamos con velocidades y aceleraciones pequeñas se debería apreciar el cambio de velocidad durante el desplazamiento.

6. Conclusiones

6.1. Conclusiones sobre el trabajo realizado

En este trabajo se desarrollan las ideas principales para desarrollar un controlador de motores de corriente continua en bucle cerrado para la ejecución de movimientos coordinados mediante el uso de un microcontrolador ARM Cortex-M4. Partiendo de este punto se puede desarrollar esta idea para la implementación de máquinas complejas, como plotters, impresoras 3D, brazos robots o máquinas de control numérico.

En este trabajo multidisciplinar se han aplicado la mayor parte de los conocimientos adquiridos en el grado de ingeniería, tales como control y automática, electrónica, sistemas robotizados, electrónica digital, programación y diseño y fabricación asistida por computador. También se ha desarrollado un gran trabajo de investigación para adquirir nuevos conocimientos que se han puesto en práctica durante el desarrollo

Aunque quedan muchos aspectos del proyecto a mejorar por falta de tiempo y recursos, en vista de los resultados podemos concluir que el objetivo inicial se ha alcanzado con éxito, logrando así la implementación de un controlador de 32 bits capaz de interpretar las coordenadas recibidas por el puerto serie para mover los tres motores de los que dispone nuestro prototipo.

6.2. Futuras mejoras

Una vez finalizado este proyecto, se abre un mundo de posibilidades para la creación de nuevos productos basados en esta tecnología. Mas allá de los objetivos establecidos al inicio de este proyecto, se sugieren una serie de aspectos a mejorar e ideas para complementar el controlador.

Un aspecto a mejorar es el uso de una placa de desarrollo con un mayor número de entradas y salidas. Esto nos permitiría controlar más motores, si fuera necesario, u otras salidas, como un mosfet para controlar elementos que requieran más potencia, como una electroválvula, un calentador o algún

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

tipo de herramienta eléctrico. Una posible opción sería la serie Nucleo-144, que es la mayor placa de la familia Nucleo. Esto supone una gran ventaja, ya que el sistema de programación, y la mayor parte del código desarrollado, será completamente compatible.

Otro aspecto a mejorar es la resolución del encoder. Aunque nuestros encoders cuentan con 334 pulsos por revolución, esta cantidad se puede duplicar incrementando el contador midiendo los pulsos del canal A y B de nuestro sensor. Otra posible solución sería medir también los flancos de subida y de bajada. En ambos casos se aumentaría notablemente la resolución del sensor.

A parte de los contadores de pulsos, también hay varios aspectos a mejorar en el código por tal de hacerlo más sencillo y entendible. Estas mejoras serían, por ejemplo, la creación de una librería para las funciones de los contadores de pulsos de los encoders, y otra para las funciones de los reguladores PID, de esta manera, eliminaríamos una gran cantidad de líneas de código en el programa principal.

Respecto a los patrones de movimiento, se podrían utilizar otros, como el senoidal, que se ha comentado en el apartado de alternativas, que reduciría los Jerks y, por lo tanto, la suavidad en los desplazamientos mejoraría notablemente. Esta solución no se implementó debido a la cantidad de tiempo que requieren los cálculos asociados a este perfil de movimiento y a que las mejoras no serían lo suficientemente significativas para que valiese la pena el tiempo invertido.

Finalmente, una de las mejoras más importantes sería la creación de una shield, similar a la RAMPS, que es una placa comúnmente utilizada en Arduino para controlar motores paso a paso. Para nuestro proyecto, nuestra placa albergaría los tres drivers de los motores, más las entradas de los encoders más un par de mosfets, que podrían controlar desde los calentadores de una impresora 3D hasta herramientas como un taladro para una máquina de control numérico.

7. Bibliografía

[MEC 14] Tropical Labs. Mechaduino. Disponible en: (<http://tropical-labs.com>)

[THI 14] Thingiverse. I3 rework upgrades and improvements
(<https://www.thingiverse.com/thing:247992>)

[CA 2014] Julián José Salt Llobregat, y otros. Control Automático Tomos I y II
(Tiempo Continuo y Tiempo Discreto). Ed. UPV, 2014.

[RAU 14] Raúl Simarro Fernández. Control de sistemas mecatrónicos (cód.
12175). Escuela Técnica Superior de Ingeniería del Diseño. UPV Ed. 2014.

[SR 15] Zotovic Stanisic, Ranko y otros. Sistemas robotizados. Escuela Técnica
Superior de Ingeniería del Diseño. UPV Ed. 2015

[STM 17] STM32F411 Datasheet. Edición de enero de 2017. Disponible en:
STMicroelectronics web site (<http://www.st.com>).

[NUC 17] NUCLEO-F411RE Datasheet. Disponible en: ARM mbed web site
(<http://www.mbed.org>).

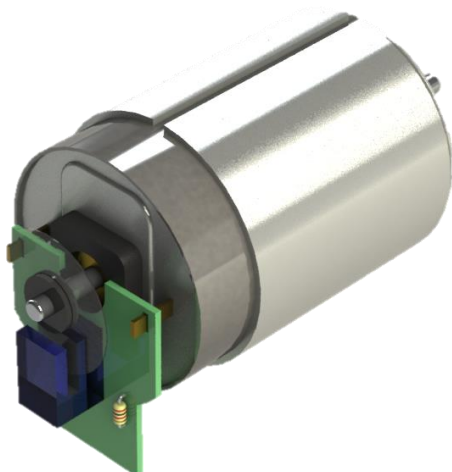
[HB 17] mbed Handbook. Disponible en: ARM mbed web site
(<http://www.mbed.org>).

[LM 17] How to calculate triangular and trapezoidal move profiles. Edición
agosto 2016. Disponible en: linear motion tips
(<http://www.linearmotiontips.com>).

[FGS] Apuntes de Clase del C2000 (Poliformat) (Fco J. Gimeno Sales). Año
2017.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

2.Planos



Autor:
D. Adrià Álvarez Donet
Tutor:
D. Ángel Perles Ivars
Cotutor:
D. Miguel Sánchez López
Valencia, julio de 2017

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Índice

1. Esquemas electrónicos.

1.1. Circuito Electrónico

2. Planos mecánicos.

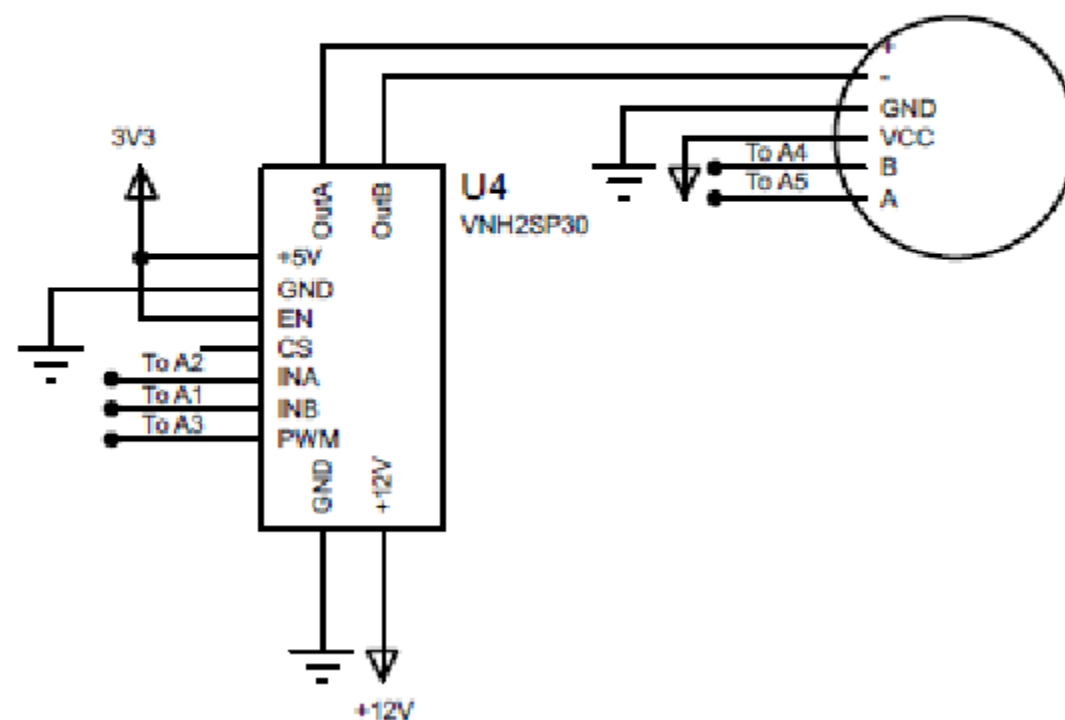
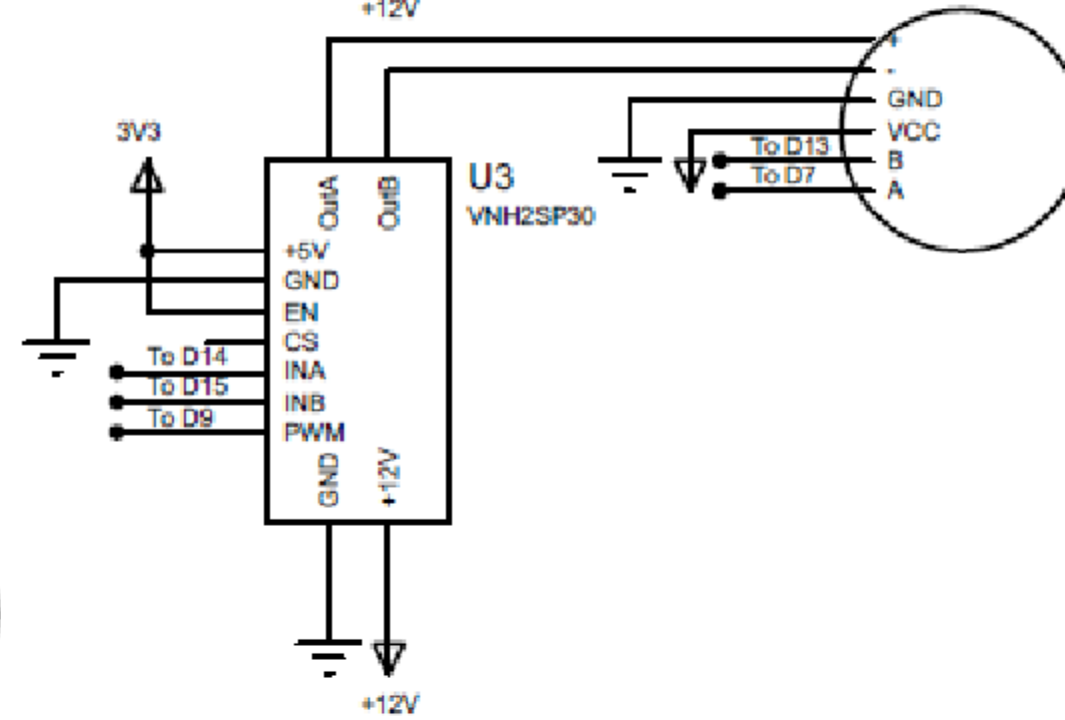
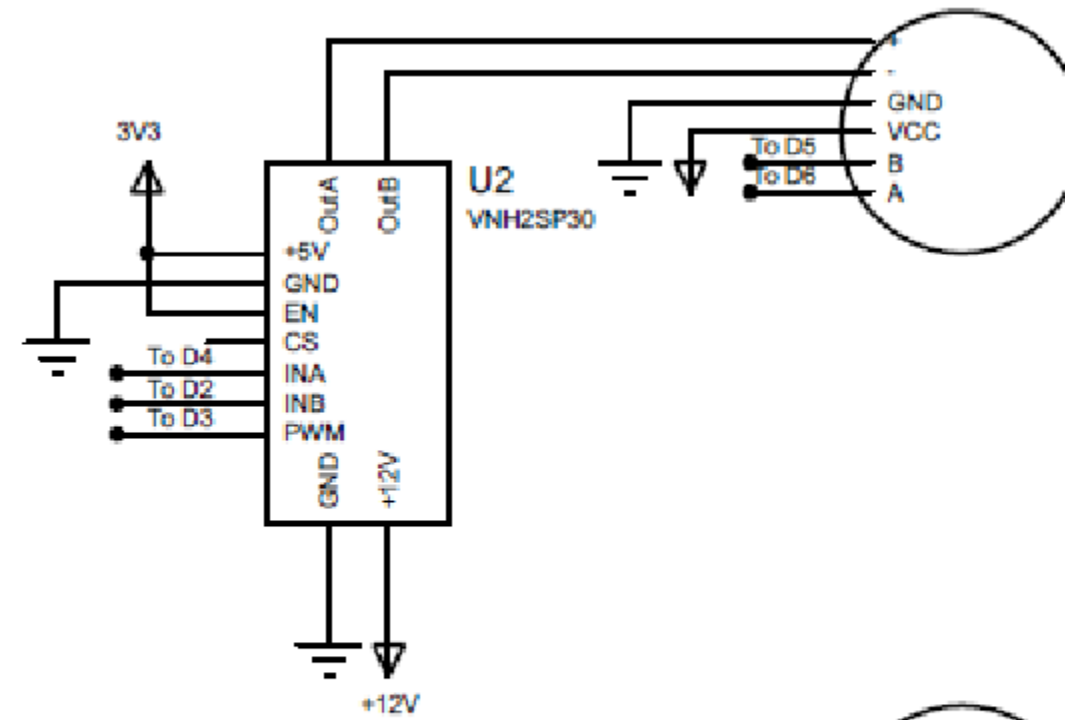
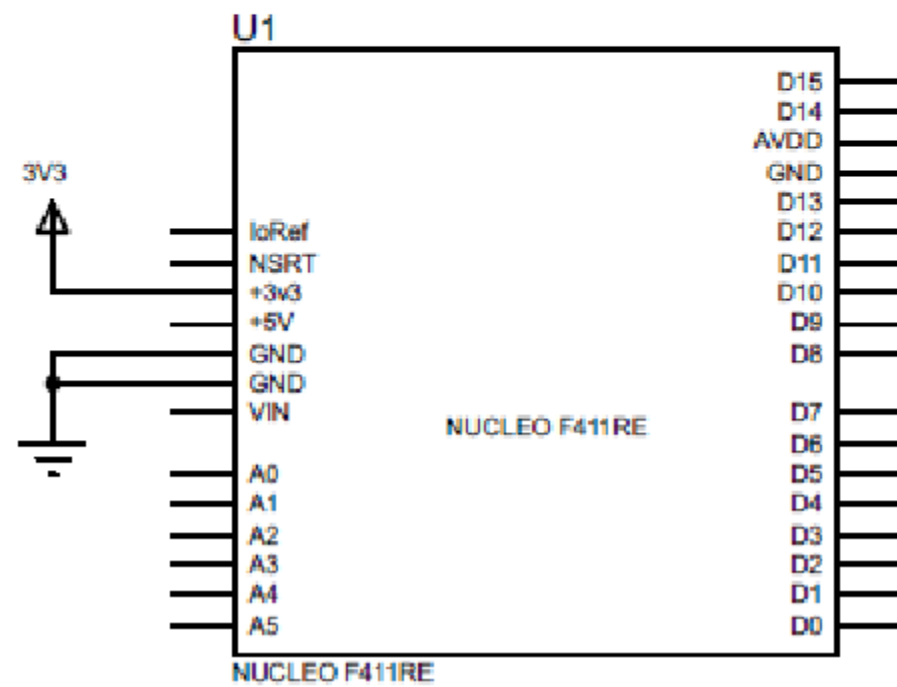
2.1. Varilla de acero

2.2. Soporte motor eje X

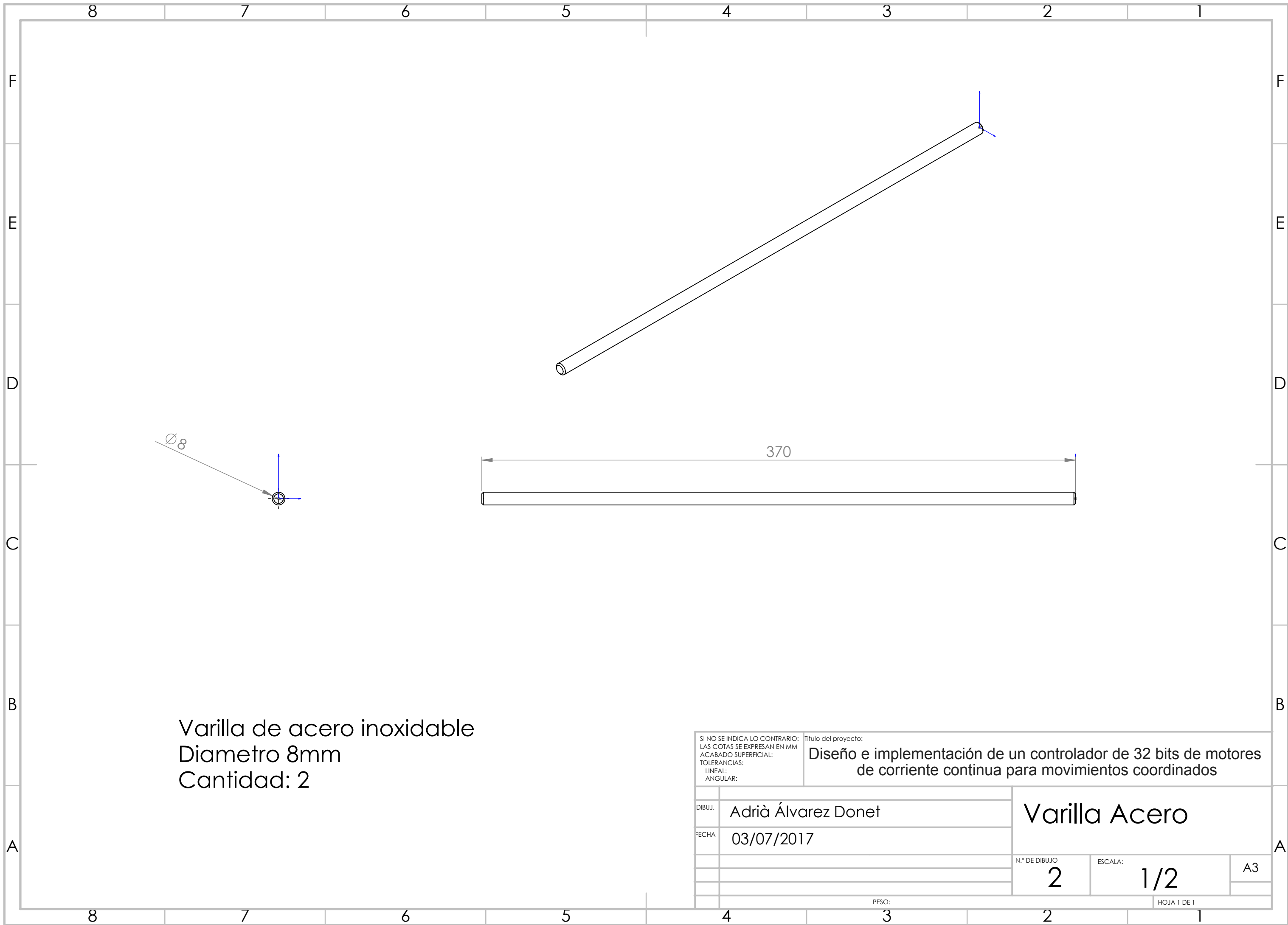
2.3. Ensamblaje eje X

2.4. Soporte motor y cursor

2.5. Etiquetas motores

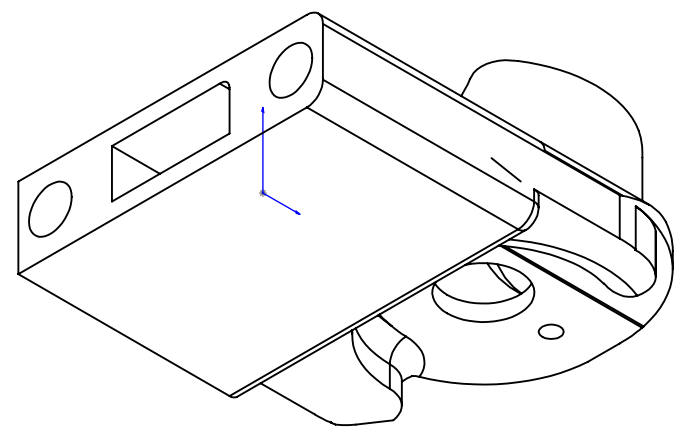
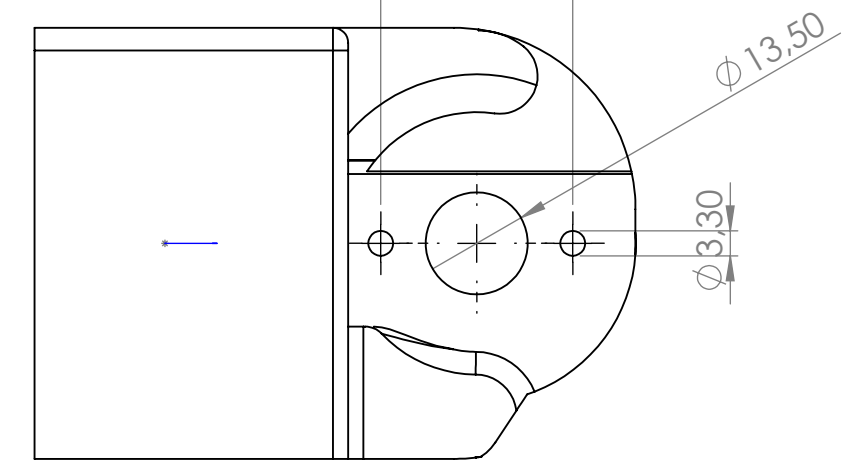
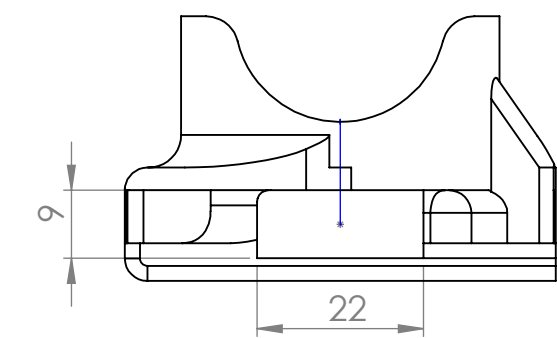
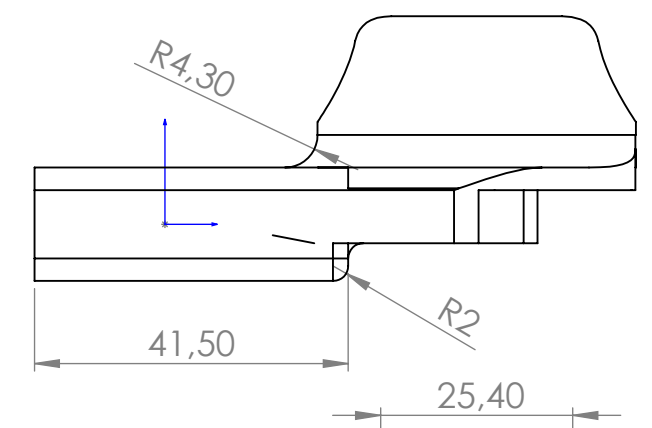
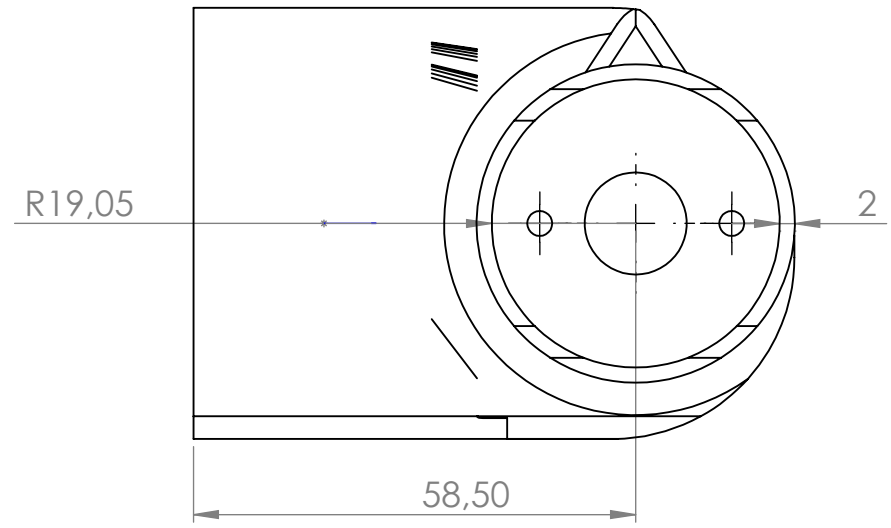
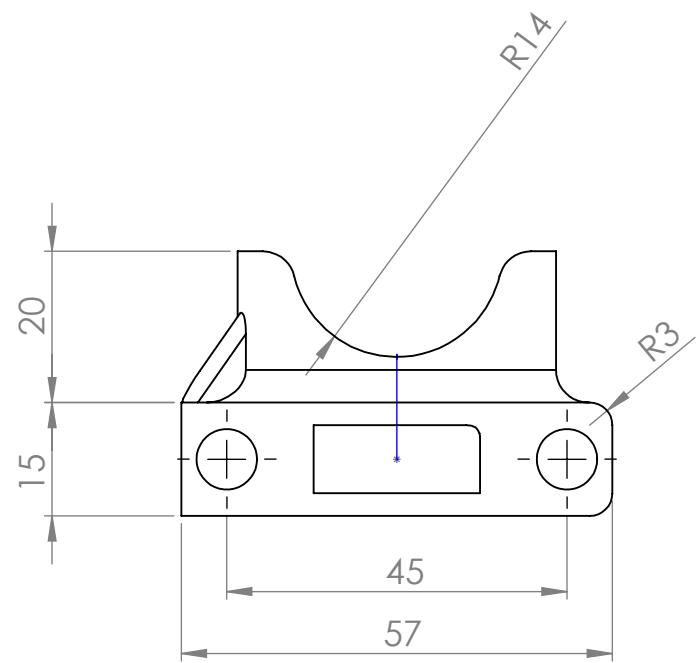


Controlador 32 bits de motores de corriente continua		
DIBUJ: Adrià Álvarez Donet		
EMPLAZAMIENTO: Valencia		
DESCRIPCION: Detalle de la conexion de los componentes		
NOBRE: Circuito electrónico	FIRMA:	

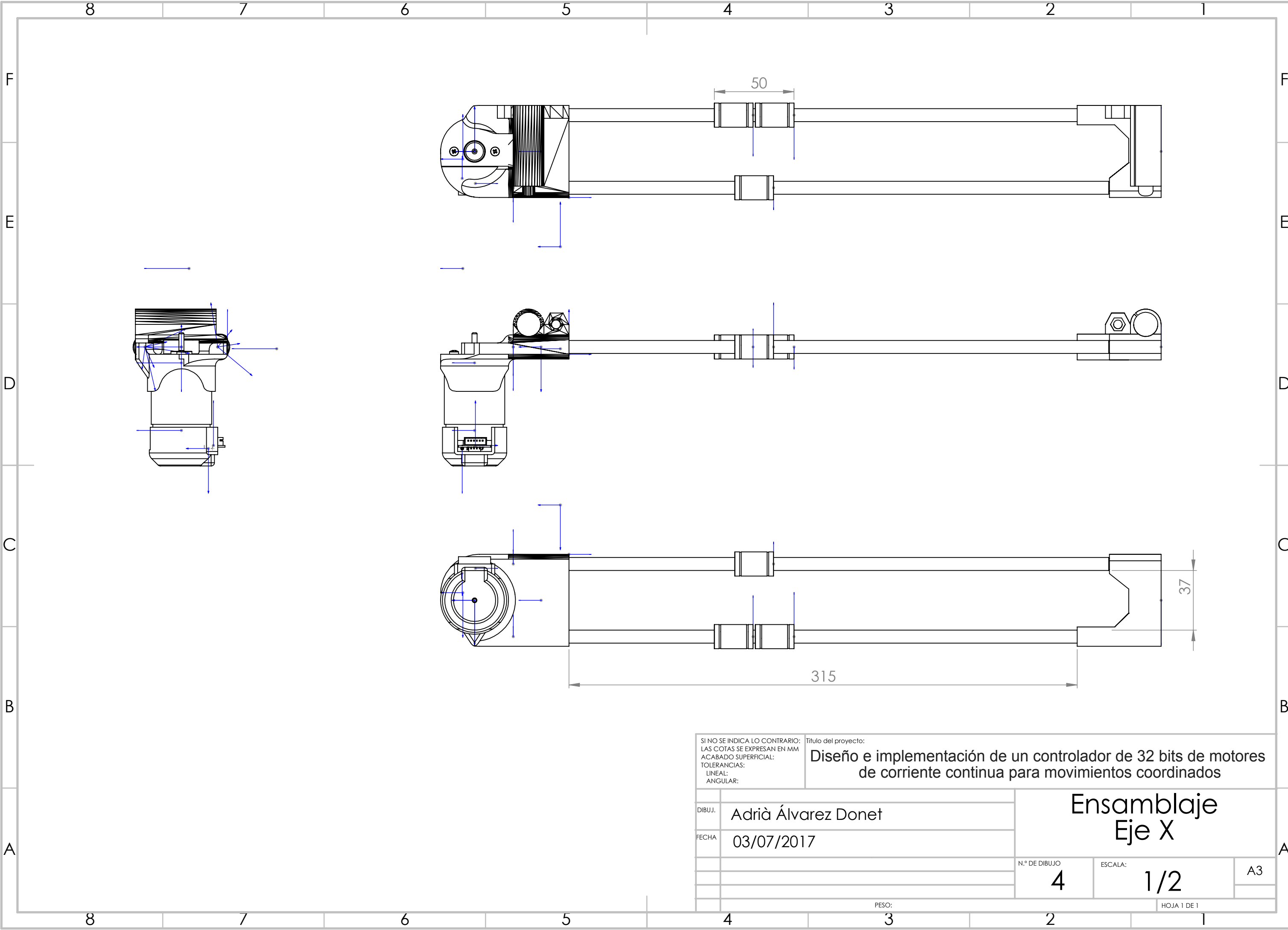


Varilla de acero inoxidable
 Diametro 8mm
 Cantidad: 2

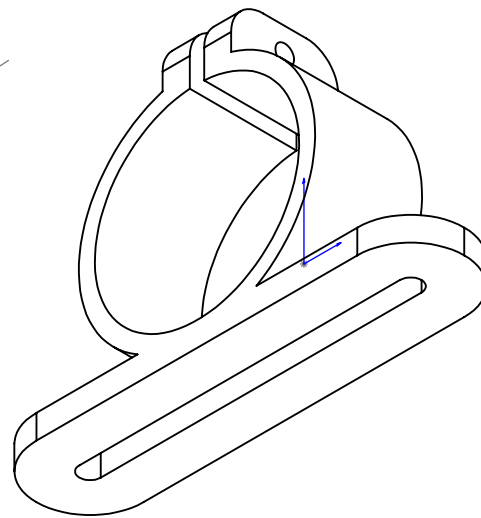
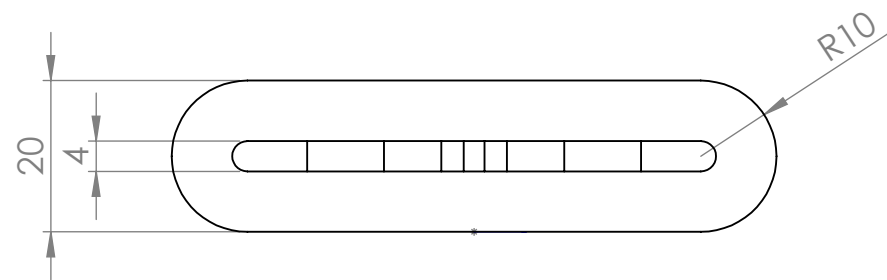
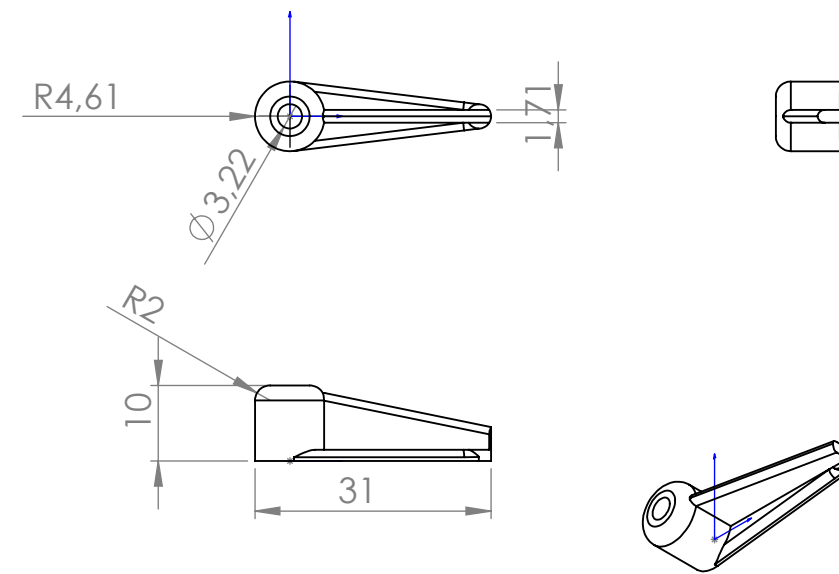
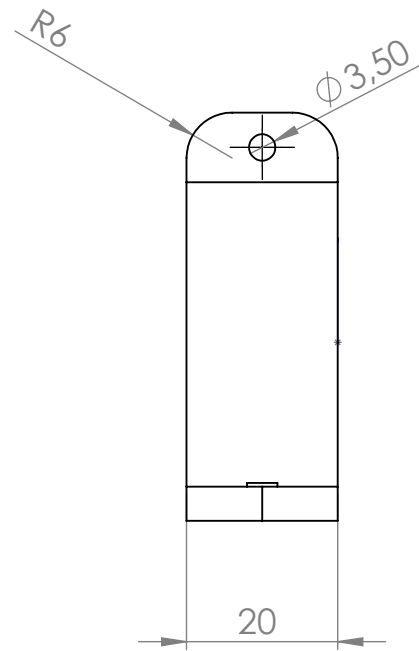
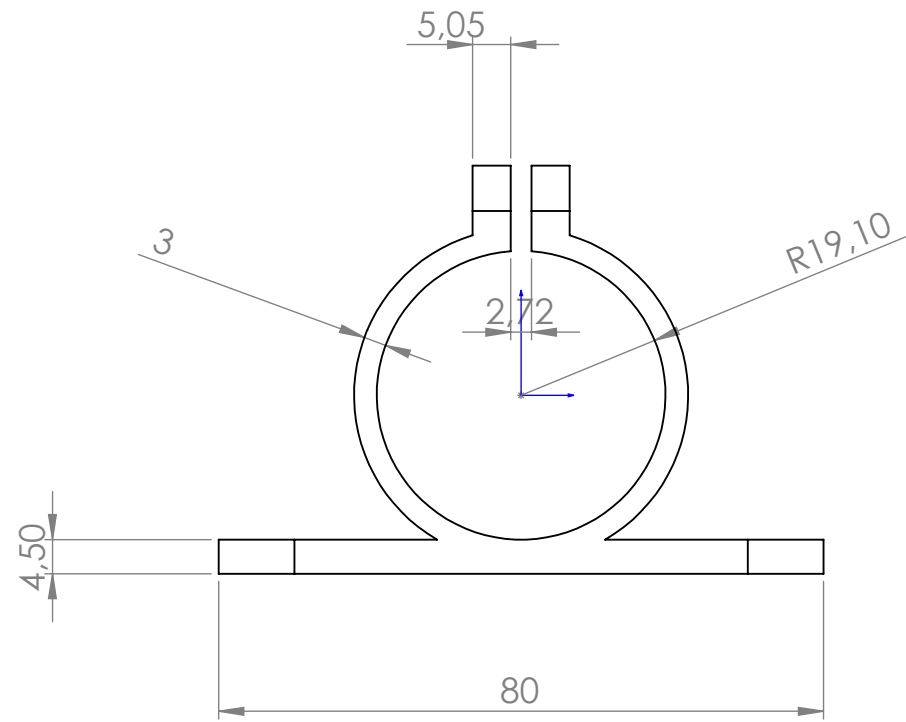
<small>SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:</small>		<small>Título del proyecto:</small> Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados		
<small>DIBUJ.</small>	Adrià Álvarez Donet	<h1>Varilla Acero</h1>		
<small>FECHA</small>	03/07/2017			
		<small>N.º DE DIBUJO</small>	<small>ESCALA:</small>	A3
		2	1/2	
			<small>PESO:</small>	<small>HOJA 1 DE 1</small>



SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		Título del proyecto: Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados		
DIBUJ.	Adrià Álvarez Donet		Soporte Motor Eje X	
FECHA	03/07/2017			
		N.º DE DIBUJO	ESCALA:	A3
		3	1/1	
		HOJA 1 DE 1		

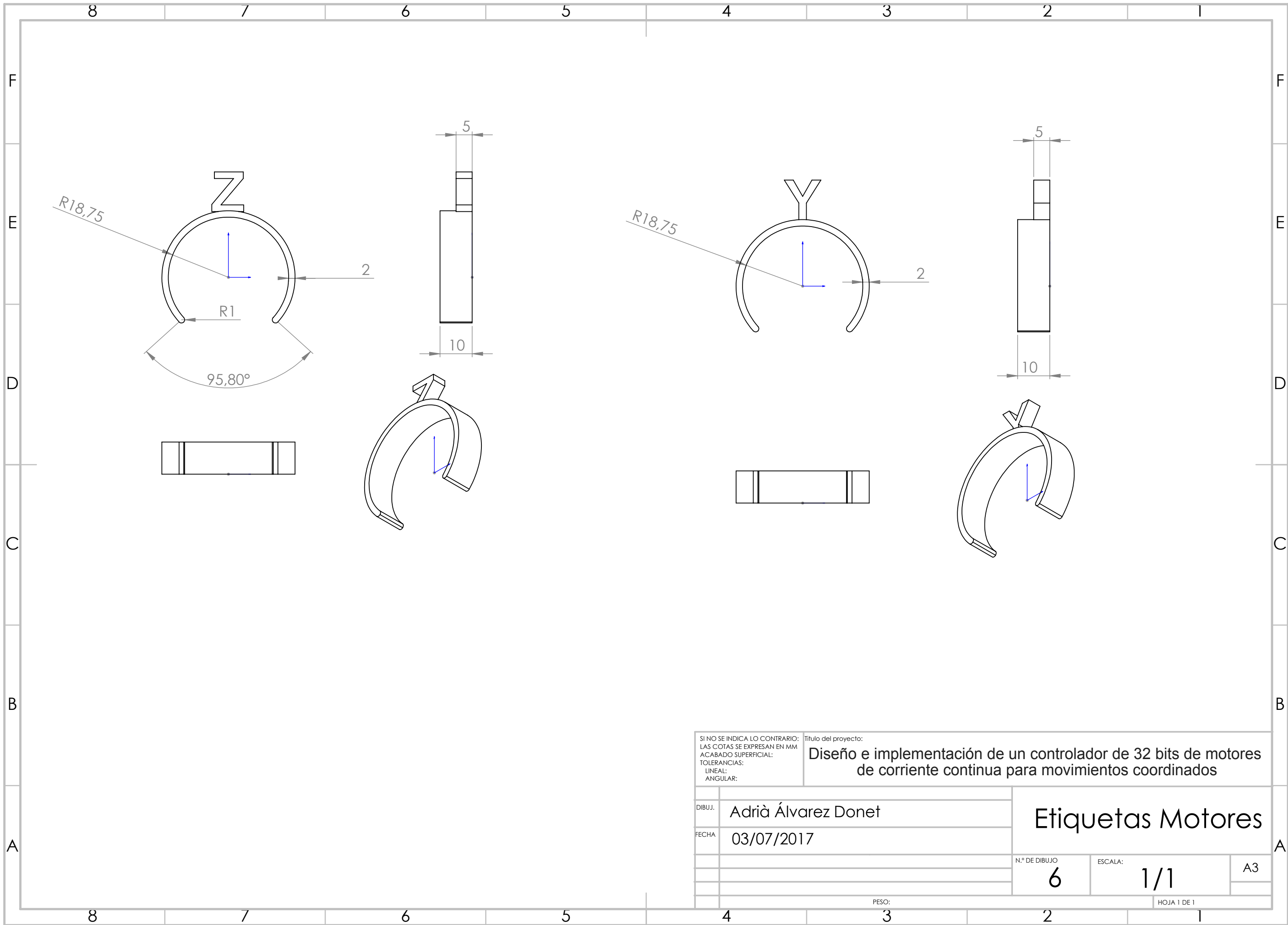


SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		Título del proyecto: Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados		
DIBUJ.	Adrià Álvarez Donet		Ensamblaje Eje X	
FECHA	03/07/2017			
		N.º DE DIBUJO	ESCALA:	A3
		4	1/2	
		PESO:		HOJA 1 DE 1



Se requieren dos piezas de cada

SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		Título del proyecto: Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados		
DIBUJ.	Adrià Álvarez Donet		Soporte motor y cursores	
FECHA	03/07/2017			
		N.º DE DIBUJO	ESCALA:	A3
		5	1/1	
		PESO:	HOJA 1 DE 1	



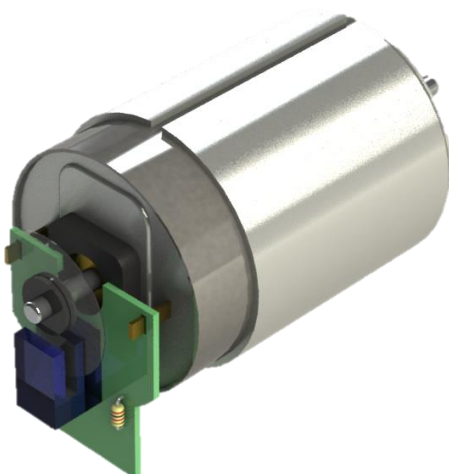
SI NO SE INDICA LO CONTRARIO:
 LAS COTAS SE EXPRESAN EN MM
 ACABADO SUPERFICIAL:
 TOLERANCIAS:
 LINEAL:
 ANGULAR:

Título del proyecto:
Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

DIBUJ.	Adrià Álvarez Donet	<h1>Etiquetas Motores</h1>		
FECHA	03/07/2017			
		N.º DE DIBUJO	ESCALA:	A3
		6	1/1	
		PESO:		HOJA 1 DE 1

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

3. Pliego de condiciones



Autor:
D. Adrià Álvarez Donet
Tutor:
D. Ángel Perles Ivars
Cotutor:
D. Miguel Sánchez López
Valencia, julio de 2017

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Índice

1. Definición y alcance del pliego	2
2. Condiciones y normas de carácter general.....	3
2.1. Instalación.....	3
2.2. Seguridad.....	3
2.3. Utilización	4
2.4. Mantenimiento	4
3. Condiciones particulares	5
4. Coste del proyecto	4

3. Pliego de condiciones

1. Definición y alcance del pliego

El presente documento trata el diseño e implementación de un controlador de 32 bits para la realización de movimientos coordinados mediante el uso de motores de corriente continua en bucle cerrado. Aunque el fondo del proyecto tiene un carácter educativo, se ha desarrollado teniendo en cuenta las necesidades reales de la industria, teniendo así en consideración los campos de la robótica, las herramientas de control numérico y el mundo de la impresión 3D.

El microcontrolador usado para el proyecto está basado en la tecnología ARM Cortex-M4 y se programará mediante la plataforma de programación online ARM mbed.

Para el desarrollo del código, se tendrán en cuenta tanto las características de la placa de desarrollo Nucleo F411RE como las características de los periféricos que se conecten a este. Estos elementos periféricos serán: motores Mitsumi M36N 4E y etapas de control de motores VNH2SP30.

A pesar de la compatibilidad para la que se han diseñado estos productos, el correcto funcionamiento del proyecto solo se ha testado con la configuración y montaje representados en los documentos anteriores.

El objeto del presente documento es la fijación de los criterios y características a contemplar en la implementación o la reproducción de este proyecto.

El ámbito de aplicación de este proyecto abarca todos los elementos mecánicos y electrónicos que componen el sistema. En algunos casos se podrán utilizar componentes alternativos para la implementación del proyecto, que tengan características similares.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

2. Condiciones y normas de carácter general

2.1. Instalación

En primer lugar, se debe comprobar que se cumplan todos los requisitos de software y hardware necesarios.

Para el desarrollo del siguiente proyecto se deben tener conocimientos previos de programación, electrónica, control y robótica.

2.2. Seguridad

Se deberá comprobar que todos los elementos, componentes y herramientas que se usen durante la realización de este trabajo, cumplan con las normas de seguridad vigentes.

La fuente de alimentación utilizada para el proyecto, deberá contar con los dispositivos de protección para sobre-corrientes, y estos no deberán de haber sido manipulados o alterados. Además, se deberá comprobar que estos dispositivos funcionen adecuadamente.

No conectar correctamente los dispositivos puede suponer un gran peligro para los componentes o las personas que los manipulen.

3. Pliego de condiciones

2.3. Utilización

Cualquier uso diferente para el que fue diseñado este proyecto puede ser peligroso. Cualquier alteración que se realice sobre la idea original debe ser estudiada y previamente simulada antes de su implementación para evitar posibles resultados inesperados o inadecuados.

2.4. Mantenimiento

Antes de utilizar esta máquina se deben realizar una serie de comprobaciones rutinarias.

En primer lugar, se comprobará la correcta conexión de los cables y el buen estado de estos. A continuación, con la alimentación desconectada, se comprobará que la salida de esta sea la especificada en el proyecto.

Se deberá comprobar que los discos de los encoders estén limpios y en un lugar apartado de la luz directa o del polvo.

Sera necesaria una lubricación periódica de las varillas y los rodamientos, con un aceite de grado SAE 10 o inferior.

3. Condiciones particulares

3.1. Condiciones técnicas

Tanto las características técnicas del microcontrolador, de los drivers y de los motores se encuentran en el anexo.

Respecto al microcontrolador, será necesario instalar la última versión de los drivers de la placa, y su última versión de firmware.

Esta placa requiere:

- Windows® OS (XP, 7, 8) o Linux 64-bit o Mac OS® X
- Cable USB tipo A a Mini-B.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Además, para programar la placa mediante la plataforma ARM mbed, se requiere un ordenador con conexión a internet.

Respecto a los motores, se requiere comprobar antes de su uso que su eje gire libremente de una forma suave. Se comprobará también que el disco del encoder no esté sucio o doblado.

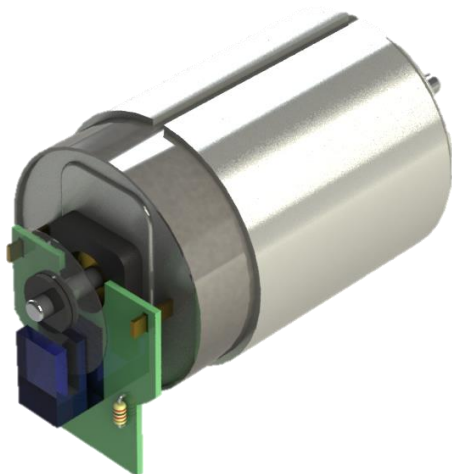
Finalmente, respecto a los drivers de los motores, se comprobará que la conexión de sus terminales sea la correcta, y que no se exceda el voltaje de alimentación especificado en la hoja de características.

3.2. Condiciones legales

Este proyecto se publica bajo una licencia publica general (GNU GPLv3), por lo que se permite el uso, estudio y modificación de este proyecto siempre que se tenga en cuenta el nombre del desarrollador original del presente proyecto. Además, el autor del presente no se hace responsable de los posibles problemas causados por un uso diferente al cual ha sido diseñado.

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

4.Presupuesto



Autor:

D. Adrià Álvarez Donet

Tutor:

D. Ángel Perles Ivars

Cotutor:

D. Miguel Sánchez López

Valencia, julio de 2017

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Índice

1. Sobre el presupuesto	2
2. Materiales	2
2.1. Materias primas	2
2.2. Piezas	3
3. Mano de obra	4
4. Coste del proyecto	4

4. Presupuesto

1. Sobre el presupuesto

Los precios del siguiente presupuesto se han obtenido de la factura y de los tickets de compra de los materiales adquiridos para la realización de este proyecto.

2. Materiales**2.1. Materias primas**

Uds.	Nombre	Cantidad	Precio (€) / U	Total
Kg	Soporte motor eje X	0,037	18	0,67
Kg	Tensor eje X	0,029	18	0,52
Kg	Soporte Motor	0,023	18	0,41
Kg	Etiquetas Motor	0,005	18	0,09
Kg	Cursores	0,002	18	0,04
m	Varillas Acero Ø8	0,7	4,96	3,47
Subtotal: 5.20€				

2.2. Piezas

Uds.	Nombre	Cantidad	Precio (€) / U	Total
U	Motor Mitsumi M36-4E	3	5,65	16,96
U	Monster motor shield VHN2SP30	3	2	6
U	Cable JST	3	0,41	1,24
U	Nucleo F411RE	1	10,29	10,29
U	Rodamiento LM8UU	3	1,24	3,72
U	Polea GT2	1	1,49	1,49
U	Material Fungible	1	2	2
m	Correa GT2 6mm	0,9	3,88	3,5
Subtotal: 45.20€				

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

3. Mano de obra

Las horas de trabajo se estimarán teniendo en cuenta que el proyecto de final de grado tiene un peso de 12 créditos ECTS y teniendo en cuenta que cada crédito ECTS supone 25 horas de trabajo. Las horas estimadas serán un total de 300.

El precio por hora de un graduado en ingeniería industrial es de un precio medio de 50€/h. Teniendo en cuenta esta información se elaborará el presupuesto.

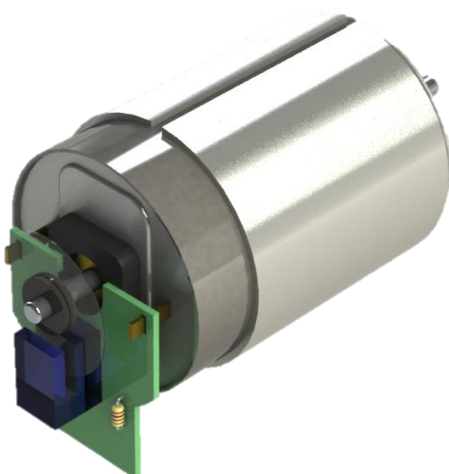
Uds.	Nombre	Cantidad	Precio (€) / U	Total
H	Diseño, montaje y programación.	300	50	15000
Subtotal: 15000€				

4. Coste del proyecto

Coste del proyecto	
Materia prima	5,20€
Piezas	45,20€
Mano de obra	15000€
Total:	15050,20€
IVA (21%)	3160,54€
Gastos adicionales (10%)	1505.02€
Costes totales: 19715,76€	

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

5.Anexos



Autor:

D. Adrià Álvarez Donet

Tutor:

D. Ángel Perles Ivars

Cotutor:

D. Miguel Sánchez López

Valencia, julio de 2017

Diseño e implementación de un controlador de 32 bits de motores de corriente continua para movimientos coordinados

Índice

1. Códigos	2
1.1. Reconocimiento motor	2
1.2. Controlador 3 motores.....	4
2. Características técnicas	11
2.1. Nucleo F411RE	11
2.2. Mitsumi M36N-4E.....	16
2.2. VNH2SP30	17

```

1  #include "mbed.h"
2
3  Serial pc(SERIAL_TX, SERIAL_RX);
4
5  Timer timer;
6
7  Ticker ticker1;
8
9
10
11  InterruptIn InA(D6);    //Entradas encoder
12  DigitalIn InB(D7);
13
14  PwmOut motor(D10);    //Salida motor
15
16  DigitalOut PA(D4);    //Direccion motor
17  DigitalOut PB(D5);
18
19  int contador = 0;
20  float pos,e,e1,e2,u,u1,T_muestreo;
21  float q0,q1,q2;
22  int target, Tiempo_absoluto, ciclo;
23  int i=0;
24  int vector[1000];
25  int x;
26
27
28  void encoder()        //contador de pulsos
29  {
30      if (InA==1) {
31          if(InB==0) {
32              contador--;
33          } else {
34              contador++;
35          }
36      } else {
37          if(InB==0) {
38              contador++;
39          } else {
40              contador--;
41          }
42      }
43      pos=contador;
44  }
45
46
47  void PID()            //contador de pulsos
48  {
49      e=target-pos;    //Calculo error
50
51      //u=q0*e+q1*e1;    //PD
52      u=u1+q0*e+q1*e1+e2*q2;    //PID
53
54      if(u>1) {
55          u=1;    //Proteccion saturacion
56      }
57      if(u<-1) {
58          u=-1;
59      }
60      if(u>0) {        //Control PWM
61          PA=0;
62          PB=1;
63          motor=u;
64      }
65      if(u<0) {
66          PA=1;
67          PB=0;

```

3/7/2017

```
68         motor=0-u;
69     }
70     e2=e1;           //Actualizacion de variables
71     e1=e;
72     u1=u;
73     ciclo++;
74 }
75
76 void funcion()      //contador de pulsos
77 {
78     PID();
79     if(i<1000){
80         vector[i]=pos;
81         i++;
82     }
83     //Mostrar objetivo, posicion, salida PID y error
84
85 }
86
87 int main()
88 {
89
90     pc.printf("INICIO\n\r");
91     q0=0.135083333;   //Constantes PID
92     q1=-0.254916667;
93     q2=0.12;
94     T_muestreo=0.001; //T de muestreo en s
95     InA.rise(&encoder); //Interrupcion Encoder
96     ticker1.attach(&funcion, T_muestreo);
97     target=0;
98
99     while(1) {
100         pc.printf("%d", &target);
101         if(target>2500) target=2500;
102         pc.printf("Target %d\n\r", target);
103         pc.printf("%d",i);
104         if(i>=1000){
105             pc.printf("Transmision\n\r");
106             for (int j=0; j<1000; j++) {
107                 pc.printf("%d\n\r",vector[j]);
108                 wait(0.05);
109             }
110         }
111     }
112 }
113
```

File "/TFG/main.cpp" printed from mbed.org on 3/7/2017

```

1  /*Adrià Álvarez 2017
2
3
4  Trayectoria trapezoidal:
5
6  /-----\
7  /         \
8  /           \
9  /             \
10 /               \
11 /                 \
12 /                   \
13 /                     \
14 /                       \
15 /                         \
16 /                           \
17 /                             \
18 /                               \
19 /                                 \
20 /                                   \
21 /                                     \
22 /                                       \
23 /                                         \
24 /                                           \
25 /                                             \
26 /                                               \
27 /                                                 \
28 /                                                   \
29 /                                                     \
30 /                                                       \
31 /                                                         \
32 /                                                           \
33 /                                                             \
34 /                                                               \
35 /                                                                 \
36 /                                                                   \
37 /                                                                     \
38 /                                                                       \
39 /                                                                         \
40 /                                                                           \
41 /                                                                             \
42 /                                                                               \
43 /                                                                                 \
44 /                                                                                   \
45 /                                                                                       \
46 /                                                                                           \
47 /                                                                                               \
48 /                                                                                                   \
49 /                                                                                                       \
50 /                                                                                       \
51 /                                                                                           \
52 /                                                                                               \
53 /                                                                                                   \
54 /                                                                                                       \
55 /                                                                                       \
56 /                                                                                           \
57 /                                                                                               \
58 /                                                                                                   \
59 /                                                                                                       \
60 /                                                                                       \
61 /                                                                                           \
62 /                                                                                               \
63 /                                                                                                   \
64 /                                                                                                       \
65 /                                                                                       \
66 /                                                                                           \
67 /                                                                                               \
68 /                                                                                                   \
69 /                                                                                                       \
70 /                                                                                       \

```

```

71 float a_X, a_Y, a_Z;
72 float v_X, v_Y, v_Z;
73
74 float cs=0; // s
75 float X_point, Y_point, Z_point;
76 float t1, t2;
77 float et1_X, et2_X, et1_Y, et2_Y, et1_Z, et2_Z;
78
79 float triang;
80
81 int max_target;
82 int dX, dY, dZ;
83 float d_X, d_Y, d_Z, max_dist; //Variables para calculos
84 float X_init, Y_init, Z_init;
85
86 bool move_axe = false;
87 bool X_dirPos = true;
88 bool Y_dirPos = true;
89 bool Z_dirPos = true;
90
91 void encoderX() //contador de pulsos X
92 {
93     if (InAX==1) {
94         if(InBX==0) {
95             X_count--;
96         } else {
97             X_count++;
98         }
99     } else {
100         if(InBX==0) {
101             X_count++;
102         } else {
103             X_count--;
104         }
105     }
106     X_pos=X_count;
107 }
108
109 void encoderY() //contador de pulsos Y
110 {
111     if (InAY==1) {
112         if(InBY==0) {
113             Y_count--;
114         } else {
115             Y_count++;
116         }
117     } else {
118         if(InBY==0) {
119             Y_count++;
120         } else {
121             Y_count--;
122         }
123     }
124     Y_pos=Y_count;
125 }
126
127 void encoderZ() //contador de pulsos Z
128 {
129     if (InAZ==1) {
130         if(InBZ==0) {
131             Z_count--;
132         } else {
133             Z_count++;
134         }
135     } else {
136         if(InBZ==0) {
137             Z_count++;
138         } else {
139             Z_count--;
140         }
141     }
142     Z_pos=Z_count;

```

```

143 }
144
145 void X_PID()          //PID Eje X
146 {                    //Trayectoria
147     cs=cms/1000;
148
149     if(move_axe){
150         if(cs<t1){
151             X_point=0.5*a_X*cs*cs;
152             et1_X=X_point;
153         }
154         if(cs>=t1 && cs<t2){
155             X_point=et1_X+v_X*(cs-t1);
156             et2_X=X_point;
157         }
158         if(cs>=t2 && X_point<=X_Target){
159             X_point=et2_X+v_X*(cs-t2)-0.5*a_X*(cs-t2)*(cs-t2);
160         }
161     }
162
163     if (X_dirPos == false) {
164         X_point = -X_point;
165     }
166     X_point=X_init+X_point;
167
168     //PID
169     e_X=X_point-X_pos;    //Calculo error
170
171     u_X=u1_X+q0_X*e_X+q1_X*e1_X+e2_X*q2_X;    //PID
172
173     if(u_X>1) {
174         u_X=1;    //Proteccion saturacion
175     }
176     if(u_X<-1) {
177         u_X=-1;
178     }
179     if(u_X>0) {    //Control PWM
180         PA_X=0;
181         PB_X=1;
182         motorX=u_X;
183     }
184     if(u_X<0) {
185         PA_X=1;
186         PB_X=0;
187         motorX=0-u_X;
188     }
189     e2_X=e1_X;    //Actualizacion de variables
190     e1_X=e_X;
191     u1_X=u_X;
192 }
193
194 void Y_PID()          //PID Eje Y
195 {
196     //Trayectoria
197     if(move_axe){
198         if(cs<t1){
199             Y_point=0.5*a_Y*cs*cs;
200             et1_Y=Y_point;
201         }
202         if(cs>=t1 && cs<t2){
203             Y_point=et1_Y+v_Y*(cs-t1);
204             et2_Y=Y_point;
205         }
206         if(cs>=t2 && Y_point<=Y_Target){
207             Y_point=et2_Y+v_Y*(cs-t2)-0.5*a_Y*(cs-t2)*(cs-t2);
208         }
209     }
210     //Sentido direccion
211
212     if (Y_dirPos == false) {
213         Y_point = -Y_point;
214     }

```



```

215
216 //Posicion absoluta
217 Y_point=Y_init+Y_point;
218
219 e_Y=Y_point-Y_pos; //Calculo error
220
221 u_Y=u1_Y+q0_Y*e_Y+q1_Y*e1_Y+e2_Y*q2_Y; //PID
222
223 if(u_Y>1) {
224     u_Y=1; //Proteccion saturacion
225 }
226 if(u_Y<-1) {
227     u_Y=-1;
228 }
229 if(u_Y>0) { //Control PWM
230     PA_Y=0;
231     PB_Y=1;
232     motorY=u_Y;
233 }
234 if(u_Y<0) {
235     PA_Y=1;
236     PB_Y=0;
237     motorY=0-u_Y;
238 }
239 e2_Y=e1_Y; //Actualizacion de variables
240 e1_Y=e_Y;
241 u1_Y=u_Y;
242 }
243
244 void Z_PID() //PID Eje Z
245 {
246     //Trayectoria
247
248     if(move_axe){
249         if(cs<t1){
250             Z_point=0.5*a_Z*cs*cs;
251             et1_Z=Z_point;
252         }
253         if(cs>=t1 && cs<t2){
254             Z_point=et1_Z+v_Z*(cs-t1);
255             et2_Z=Z_point;
256         }
257         if(cs>=t2 && Z_point<=Z_Target){
258             Z_point=et2_Z+v_Z*(cs-t2)-0.5*a_Z*(cs-t2)*(cs-t2);
259         }
260     }
261     //Sentido direccion
262
263     if (Z_dirPos == false) {
264         Z_point = -Z_point;
265     }
266
267     //Posicion absoluta
268     Z_point=Z_init+Z_point;
269
270     cms++; //Incrementar contador ms para trayectoria
271
272     if(cs>=(t1+t2)){ //Desactivar Calculo trayectoria
273         move_axe=false;
274     }
275
276     e_Z=Z_point-Z_pos; //Calculo error
277
278     //u=q0_Z*e+q1_Z*e1; //PD
279     u_Z=u1_Z+q0_Z*e_Z+q1_Z*e1_Z+e2_Z*q2_Z; //PID
280
281     if(u_Z>1) {
282         u_Z=1; //Proteccion saturacion
283     }
284     if(u_Z<-1) {
285         u_Z=-1;
286     }

```

```

287     if(u_Z>0) {           //Control PWM
288         PA_Z=0;
289         PB_Z=1;
290         motorZ=u_Z;
291     }
292     if(u_Z<0) {
293         PA_Z=1;
294         PB_Z=0;
295         motorZ=0-u_Z;
296     }
297     e2_Z=e1_Z;           //Actualizacion de variables
298     e1_Z=e_Z;
299     u1_Z=u_Z;
300 }
301
302
303 void funcionX(){X_PID();}
304 void funcionY(){Y_PID();}
305 void funcionZ(){Z_PID();}
306
307 int main()
308 {
309
310     pc.printf("INICIO\n\r");
311
312     q0_X=0.135083333;     //Constantes PID
313     q1_X=-0.254916667;
314     q2_X=0.12;
315
316     q0_Y=0.135083333;     //Constantes PID
317     q1_Y=-0.254916667;
318     q2_Y=0.12;
319
320     q0_Z=0.135083333;     //Constantes PID
321     q1_Z=-0.254916667;
322     q2_Z=0.12;
323
324     A=a;
325     V=v;
326
327     T_muestreo=0.001;     //T de muestreo en s
328
329     InAX.rise(&encoderX); //Interrupcion Encoder
330     InAY.rise(&encoderY);
331     InAZ.rise(&encoderZ);
332
333     tickerX.attach(&funcionX, T_muestreo); //Ticker Funciones
334     tickerY.attach(&funcionY, T_muestreo);
335     tickerZ.attach(&funcionZ, T_muestreo);
336
337
338     while(1) {
339         pc.scanf("%Xd %Yd %Zd", &X_Target, &Y_Target, &Z_Target);
340
341         if(X_Target>2500){X_Target=2500;}
342         if(Y_Target>2500){Y_Target=2500;}
343         if(Z_Target>2500){Z_Target=2500;}
344
345         X_dirPos = true;
346         Y_dirPos = true;
347         Z_dirPos = true;
348
349         //Posicion inicial trayectoria
350         X_init=X_count;
351         Y_init=Y_count;
352         Z_init=Z_count;
353
354         //distancia a recorrer
355         dX=X_Target-X_init;
356         dY=Y_Target-Y_init;
357         dZ=Z_Target-Z_init;
358

```

```

359     if (dX < 0) {           //direccion trayectoria
360         dX = -dX;
361         X_dirPos = false;
362     }
363     if (dY < 0) {
364         dY = -dY;
365         Y_dirPos = false;
366     }
367     if (dZ < 0) {
368         dZ = -dZ;
369         Z_dirPos = false;
370     }
371
372
373     max_target=(dY>dZ)?dY:dZ;           //maxima trayectoria
374     max_target=(max_target>dX)?max_target:dX;
375
376
377     if(max_target<=((v^2)/a)){         // calculo t1 t2
378         triang=max_target/a;
379         t1=t2=2*sqrt(triang);
380     }
381     else if(max_target>((v^2)/a)){
382         t1=V/A;
383         t2=-V/A+max_target/V+t1;
384     }
385
386     //Calculo aceleraciones y velocidades
387     d_X=dX;
388     d_Y=dY;
389     d_Z=dZ;
390     max_dist=max_target;
391
392     if(dX == max_target){
393         a_X=a;
394         v_X=v;
395         a_Y=a*(d_Y/max_dist);
396         v_Y=a_Y*t1;
397         a_Z=a*(d_Z/max_dist);
398         v_Z=a_Z*t1;
399     }
400
401     if(dY == max_target){
402         a_Y=a;
403         v_Y=v;
404         a_X=a*(d_X/max_dist);
405         v_X=a_X*t1;
406         a_Z=a*(d_Z/max_dist);
407         v_Z=a_Z*t1;
408     }
409
410     if(dZ == max_target){
411         a_Z=a;
412         v_Z=v;
413         a_X=a*(d_X/max_dist);
414         v_X=a_X*t1;
415         a_Y=a*(d_Y/max_dist);
416         v_Y=a_Y*t1;
417     }
418
419     cms=0;
420
421     move_axe=true;
422
423     pc.printf("Objetivo X %d Objetivo Y %d Objetivo Z %d\n\r", X_Target, Y_Target, Z_Target);
424     wait(0.1);
425     pc.printf("t1: %f t2: %f\n\r", t1, t2);
426     pc.printf("a_x: %f a_y: %f a_z: %f\n\r", a_X, a_Y, a_Z);
427     /*
428     wait(1);
429     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
430     wait(2);

```

4/7/2017

```
431     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
432     wait(2);
433     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
434     wait(2);
435     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
436     wait(2);
437     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
438     wait(2);
439     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
440     wait(2);
441     pc.printf("PUNTO: X: %f Y: %f Z: %f CmS: %f Pos: %d\n\r", X_point, Y_point, Z_point, cms, X_count);
442     wait(9);
443     pc.printf("X%d Y%d Z%d\n\r", X_count, Y_count, Z_count);
444     */
445
446     //pc.printf("X_pos %d Y_pos %d Z_pos %d\n\r", X_count, Y_count, Z_count);
447 }
448 }
449
```

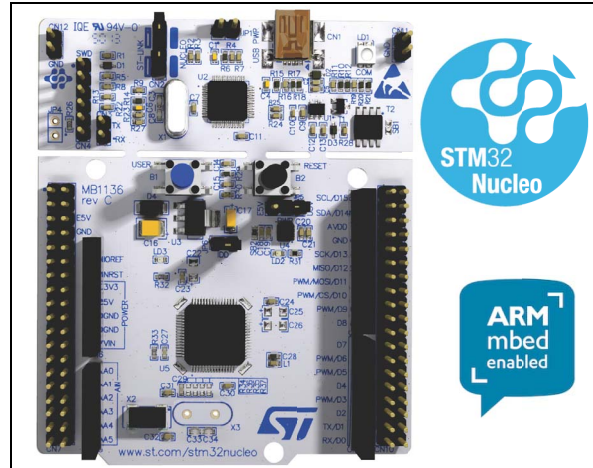
File "/TFG_3_Motores/main.cpp" printed from mbed.org on 4/7/2017

Features

- STM32 microcontroller in QFP64 package
- Two types of extension resources:
 - Arduino™ Uno V3 connectivity
 - ST morpho extension pin headers for full access to all STM32 I/Os
- ARM® mbed™ (see <http://mbed.org>)
- On-board ST-LINK/V2-1 debugger/programmer with SWD connector:
 - Selection-mode switch to use the kit as a standalone ST-LINK/V2-1
- Flexible board power supply:
 - USB VBUS or external source (3.3V, 5V, 7 - 12V)
 - Power management access point
- Three LEDs:
 - USB communication (LD1), user LED (LD2), power LED (LD3)
- Two push-buttons: USER and RESET
- USB re-enumeration capability. Three different interfaces supported on USB:
 - Virtual COM port
 - Mass storage
 - Debug port
- Support of wide choice of Integrated Development Environments (IDEs) including IAR™, ARM® Keil®, GCC-based IDEs

Description

The STM32 Nucleo board provides an affordable and flexible way for users to try out new concepts and build prototypes with the STM32 microcontroller, choosing from the various combinations of performance, power consumption and features. The Arduino™ Uno V3 connectivity support and the ST morpho headers allow to expand easily the functionality of the STM32 Nucleo open development platform



1. Picture is not contractual

with a wide choice of specialized shields. The STM32 Nucleo board does not require any separate probe as it integrates the ST-LINK/V2-1 debugger and programmer. The STM32 Nucleo board comes with the STM32 comprehensive software HAL library together with various packaged software examples, as well as direct access to the ARM® mbed™ online resources at <http://mbed.org>.

Table 1. Device summary

Reference	Part number
NUCLEO-XXXXRX	NUCLEO-F030R8, NUCLEO-F070RB, NUCLEO-F072RB, NUCLEO-F091RC, NUCLEO-F103RB, NUCLEO-F302R8, NUCLEO-F303RE, NUCLEO-F334R8, NUCLEO-F401RE, NUCLEO-F410RB, NUCLEO-F411RE, NUCLEO-F446RE, NUCLEO-L053R8, NUCLEO-L073RZ, NUCLEO-L152RE, NUCLEO-L452RE, NUCLEO-L476RG.

System requirement

- Windows® OS (XP, 7, 8) or Linux 64-bit or Mac OS® X
- USB Type-A to Mini-B cable

Development toolchains

- ARM® Keil®: MDK-ARM^(a)
- IAR™: EWARM^(a)
- GCC-based IDEs (free AC6: SW4STM32, Atollic TrueSTUDIO®^(a) and others)
- ARM® mbed™ online

Demonstration software

Demonstration software is preloaded in the STM32 Flash memory for easy demonstration of the device peripherals in standalone mode. For more information and to download the latest version, refer to the demonstration software for the STM32 Nucleo board at the www.st.com/stm32nucleo website.

Ordering information

[Table 2](#) lists the order codes and the respective targeted STM32.

Table 2. Ordering information

Order code	Targeted STM32
NUCLEO-F030R8	STM32F030R8T6
NUCLEO-F070RB	STM32F070RBT6
NUCLEO-F072RB	STM32F072RBT6
NUCLEO-F091RC	STM32F091RCT6
NUCLEO-F103RB	STM32F103RBT6
NUCLEO-F302R8	STM32F302R8T6
NUCLEO-F303RE	STM32F303RET6

a. On Windows® only.

Table 2. Ordering information (continued)

Order code	Targeted STM32
NUCLEO-F334R8	STM32F334R8T6
NUCLEO-F401RE	STM32F401RET6
NUCLEO-F410RB	STM32F410RBT6
NUCLEO-F411RE	STM32F411RET6
NUCLEO-F446RE	STM32F446RET6
NUCLEO-L053R8	STM32L053R8T6
NUCLEO-L073RZ	STM32L073RZT6
NUCLEO-L152RE	STM32L152RET6
NUCLEO-L452RE	STM32L452RET6
NUCLEO-L476RG	STM32L476RGT6

The meaning of the NUCLEO-TXXXRY codification is explained in [Table 3](#) with an example:

Table 3. Codification explanation

NUCLEO-TXXXRY	Description	Example: NUCLEO-L452RE
TXXX	STM32 product line	STM32L452
R	STM32 package pin count	64 pins
Y	STM32 Flash memory size (8 for 64 Kbytes, B for 128 Kbytes, C for 256 Kbytes, E for 512 Kbytes, G for 1 Mbyte, Z for 192 Kbytes)	512 Kbytes

The order code is printed on a sticker placed at the top or bottom side of the board.

Revision history

Table 4. Document revision history

Date	Revision	Changes
10-Feb-2014	1	Initial release.
13-Feb-2014	2	Added Table 1: Device summary and updated Table 2: Ordering information .
11-Apr-2014	3	Extended the applicability to NUCLEO-F302R8. Updated Table 1: Device summary and Table 2: Ordering information .
26-May-2014	4	Extended the applicability to NUCLEO-L053R8, NUCLEO-F072RB, NUCLEO-F334R8 and NUCLEO-F411RE Updated Table 1 and Table 2 .
09-Sep-2014	5	Extended the applicability to NUCLEO-F091RC and NUCLEO-F303RE. Updated Features . Updated Table 1: Device summary and Table 2: Ordering information .
16-Dec-2014	6	Extended the applicability to NUCLEO-F070RB, NUCLEO-L073RZ and NUCLEO-L476RG. Updated Table 1: Device summary and Table 2: Ordering information .
08-Jul-2015	7	Extended the applicability to NUCLEO-F410RB, NUCLEO-F446RE. Updated Table 1: Device summary and Table 2: Ordering information .
29-Nov-2016	8	Extended the applicability to NUCLEO-L452RE. Updated Table 1: Device summary and Table 2: Ordering information . Added Table 3: Codification explanation .

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2016 STMicroelectronics – All rights reserved



DC Mini-Motors M36N-4E Series

APPLICATIONS

Printer
Sewing Machine

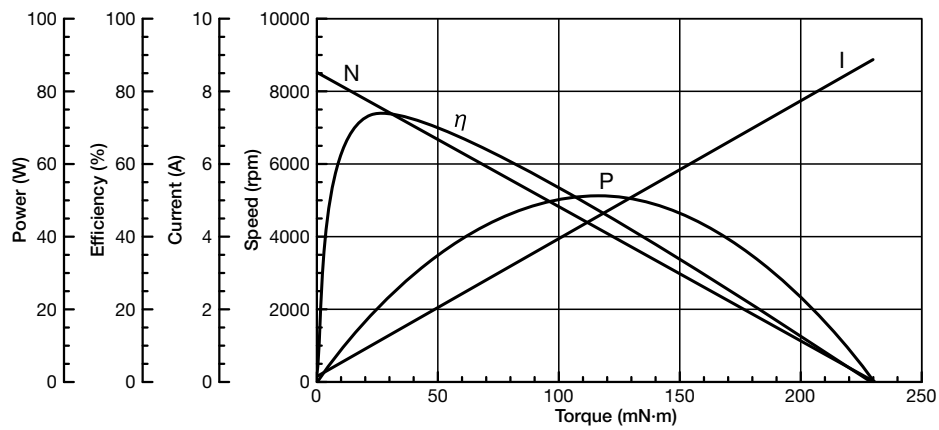


SPECIFICATIONS

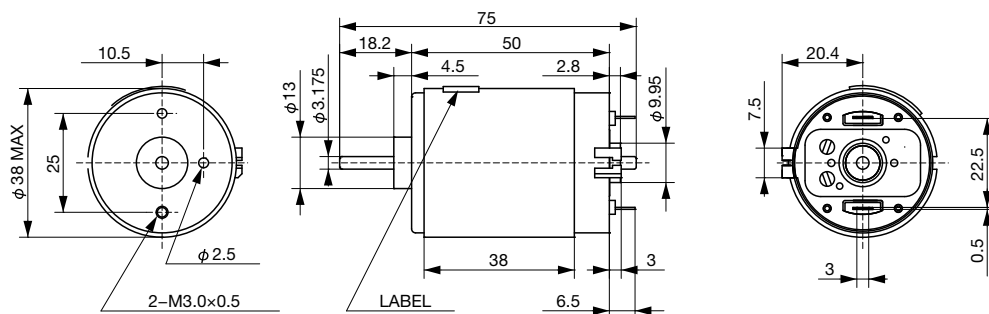
Items	Specifications
Rated Voltage	24.0V
Voltage Range	8.0~27.0V
Rated Load	33.3mN·m
No Load Speed	8,500rpm
No Load Current	350mA or less
Starting Torque	230mN·m
Rotation	CW/CCW

*Characteristics and the shaft length can be customized.

CHARACTERISTICS



DIMENSIONS

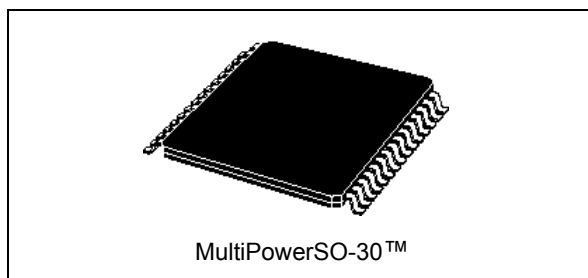


Unit : mm

• Any products mentioned in this catalog are subject to any modification in their appearance and others for improvements without prior notification.
• The details listed here are not a guarantee of the individual products at the time of ordering. When using the products, you will be asked to check their specifications.

Automotive fully integrated H-bridge motor driver

Datasheet - production data



Features

Type	$R_{DS(on)}$	I_{out}	V_{CCmax}
VNH2SP30-E	19 mΩ max (per leg)	30 A	41 V

- AEC-Q100 qualified
- 5 V logic level compatible inputs
- Undervoltage and overvoltage shutdown
- Overvoltage clamp
- Thermal shutdown
- Cross-conduction protection
- Linear current limiter
- Very low standby power consumption
- PWM operation up to 20 kHz
- Protection against loss of ground and loss of V_{CC}
- Current sense output proportional to motor current
- Package: ECOPACK®



Description

The VNH2SP30-E is a full bridge motor driver intended for a wide range of automotive applications. The device incorporates a dual monolithic high side driver and two low side switches. The high side driver switch is designed

using STMicroelectronics well known and proven proprietary VIPower™ M0 technology which permits efficient integration on the same die of a true power MOSFET with intelligent signal/protection circuitry.

The low side switches are vertical MOSFETs manufactured using STMicroelectronics proprietary EHD (STripFET™) process. The three die are assembled in a MultiPowerSO-30 package on electrically isolated leadframes. This package, specifically designed for the harsh automotive environments, offers improved thermal performance thanks to exposed die pads. Moreover, its fully symmetrical mechanical design allows superior manufacturability at board level. The input signals IN_A and IN_B can directly interface with the microcontroller to select the motor direction and brake condition. The $DIAG_A/EN_A$ or $DIAG_B/EN_B$, when connected to an external pull-up resistor, enable one leg of the bridge. They also provide a feedback digital diagnostic signal. The normal operating condition is explained in the truth table. The motor current can be monitored with the CS pin by delivering a current proportional to its value. The speed of the motor can be controlled in all possible conditions by the PWM up to 20 kHz. In all cases, a low level state on the PWM pin will turn off both the LS_A and LS_B switches. When PWM rises to a high level, LS_A or LS_B turn on again depending on the input pin state.

Table 1. Device summary

Package	Order code
	Tape and reel
MultiPowerSO-30	VNH2SP30TR-E

Contents

- 1 Block diagram and pin description 5**

- 2 Electrical specifications 8**
 - 2.1 Absolute maximum ratings 8
 - 2.2 Electrical characteristics 9
 - 2.3 Electrical characteristics curves 17

- 3 Application information 21**
 - 3.1 Reverse battery protection 22

- 4 Package and PCB thermal data 26**
 - 4.1 PowerSSO-30 thermal data 26
 - 4.1.1 Thermal calculation in clockwise and anti-clockwise operation in steady-state mode 27
 - 4.1.2 Thermal resistance definitions (values according to the PCB heatsink area) 27
 - 4.1.3 Thermal calculation in transient mode 27
 - 4.1.4 Single pulse thermal impedance definition (values according to the PCB heatsink area) 27

- 5 Package information 31**
 - 5.1 MultiPowerSO-30 package information 31
 - 5.2 Packing information 33

- 6 Revision history 34**

List of tables

Table 1.	Device summary	1
Table 2.	Block description	5
Table 3.	Pin definitions and functions	6
Table 4.	Pin functions description	7
Table 5.	Absolute maximum ratings	8
Table 6.	Power section	9
Table 7.	Logic inputs (INA, INB, ENA, ENB)	9
Table 8.	PWM	10
Table 9.	Switching ($V_{CC} = 13\text{ V}$, $R_{LOAD} = 0.87\text{ W}$, unless otherwise specified)	10
Table 10.	Protection and diagnostic	11
Table 11.	Current sense ($9\text{ V} < V_{CC} < 16\text{ V}$)	12
Table 12.	Truth table in normal operating conditions	15
Table 13.	Truth table in fault conditions (detected on OUTA).	15
Table 14.	Electrical transient requirements	16
Table 15.	Thermal calculation in clockwise and anti-clockwise operation in steady-state mode	27
Table 16.	Thermal parameters	29
Table 17.	MultiPowerSO-30 mechanical data	31
Table 18.	Document revision history	34

List of figures

Figure 1.	Block diagram	5
Figure 2.	Configuration diagram (top view)	6
Figure 3.	Current and voltage conventions	8
Figure 4.	Definition of the delay times measurement	12
Figure 5.	Definition of the low side switching times	13
Figure 6.	Definition of the high side switching times	13
Figure 7.	Definition of dynamic cross conduction current during a PWM operation.	14
Figure 8.	On state supply current.	17
Figure 9.	Off state supply current.	17
Figure 10.	High level input current.	17
Figure 11.	Input clamp voltage.	17
Figure 12.	Input high level voltage	17
Figure 13.	Input low level voltage	17
Figure 14.	Input hysteresis voltage	18
Figure 15.	High level enable pin current	18
Figure 16.	Delay time during change of operation mode	18
Figure 17.	Enable clamp voltage	18
Figure 18.	High level enable voltage	18
Figure 19.	Low level enable voltage	18
Figure 20.	PWM high level voltage	19
Figure 21.	PWM low level voltage	19
Figure 22.	PWM high level current.	19
Figure 23.	Overshoot shutdown	19
Figure 24.	Undervoltage shutdown	19
Figure 25.	Current limitation.	19
Figure 26.	On state high side resistance vs Tcase	20
Figure 27.	On state low side resistance vs Tcase	20
Figure 28.	Turn-on delay time	20
Figure 29.	Turn-off delay time	20
Figure 30.	Output voltage rise time	20
Figure 31.	Output voltage fall time	20
Figure 32.	Typical application circuit for DC to 20 kHz PWM operation short-circuit protection	21
Figure 33.	Behavior in fault condition (how a fault can be cleared)	22
Figure 34.	Half-bridge configuration.	23
Figure 35.	Multi-motor configuration	23
Figure 36.	Waveforms in full bridge operation	24
Figure 37.	Waveforms in full bridge operation (continued)	25
Figure 38.	MultiPowerSO-30™ PC board	26
Figure 39.	Chipset configuration	26
Figure 40.	Auto and mutual Rthj-amb vs PCB copper area in open box free air condition	26
Figure 41.	MultiPowerSO-30 HSD thermal impedance junction ambient single pulse	28
Figure 42.	MultiPowerSO-30 LSD thermal impedance junction ambient single pulse	28
Figure 43.	Thermal fitting model of an H-bridge in MultiPowerSO-30	29
Figure 44.	MultiPowerSO-30 package outline	31
Figure 45.	MultiPowerSO-30 suggested pad layout	32
Figure 46.	MultiPowerSO-30 tape and reel shipment (suffix "TR")	33

1 Block diagram and pin description

Figure 1. Block diagram

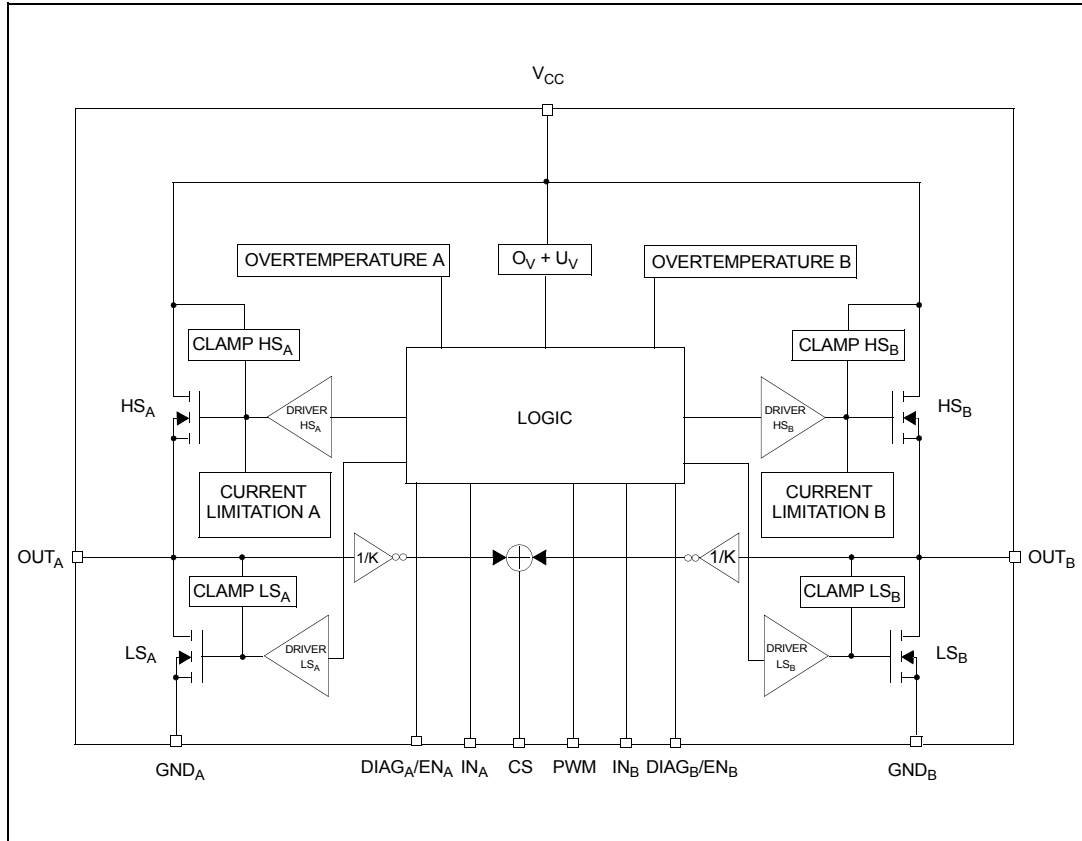


Table 2. Block description

Name	Description
Logic control	Allows the turn-on and the turn-off of the high side and the low side switches according to the truth table
Overvoltage + undervoltage	Shuts down the device outside the range [5.5V..16V] for the battery voltage
High side and low side clamp voltage	Protects the high side and the low side switches from the high voltage on the battery line in all configurations for the motor
High side and low side driver	Drives the gate of the concerned switch to allow a proper $R_{DS(on)}$ for the leg of the bridge
Linear current limiter	Limits the motor current by reducing the high side switch gate-source voltage when short-circuit to ground occurs
Overtemperature protection	In case of short-circuit with the increase of the junction's temperature, shuts down the concerned high side to prevent its degradation and to protect the die
Fault detection	Signals an abnormal behavior of the switches in the half-bridge A or B by pulling low the concerned EN_x /DIAG $_x$ pin

Figure 2. Configuration diagram (top view)

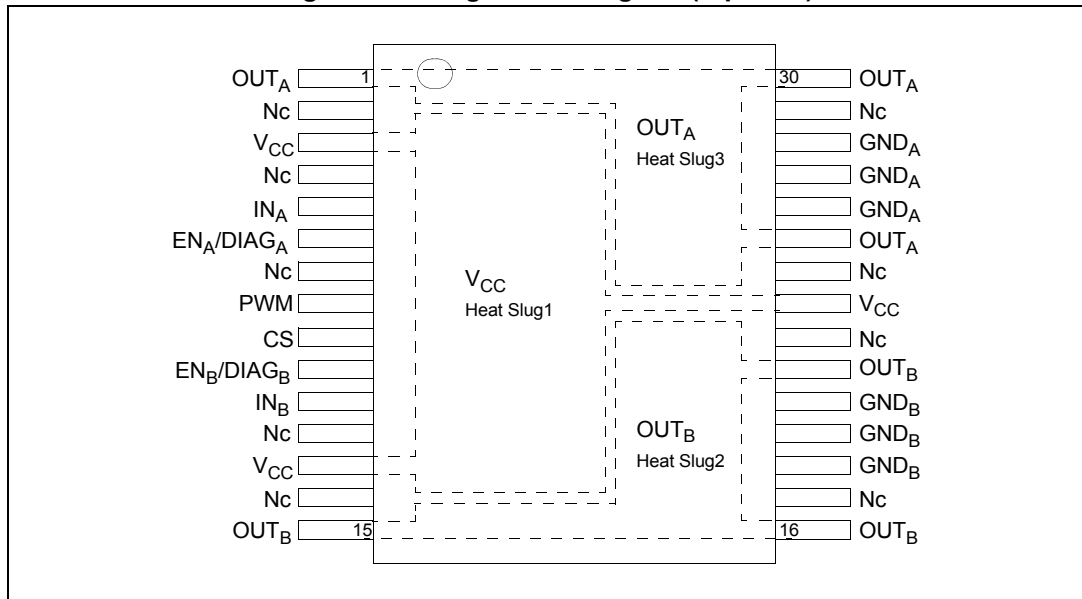


Table 3. Pin definitions and functions

Pin no.	Symbol	Function
1, 25, 30	OUT _A , Heat Slug3	Source of high side switch A / Drain of low side switch A
2, 4, 7, 12, 14, 17, 22, 24, 29	NC	Not connected
3, 13, 23	V _{CC} , Heat Slug1	Drain of high side switches and power supply voltage
6	EN _A /DIAG _A	Status of high side and low side switches A; open drain output
5	IN _A	Clockwise input
8	PWM	PWM input
9	CS	Output of current sense
11	IN _B	Counter clockwise input
10	EN _B /DIAG _B	Status of high side and low side switches B; open drain output
15, 16, 21	OUT _B , Heat Slug2	Source of high side switch B / Drain of low side switch B
26, 27, 28	GND _A	Source of low side switch A ⁽¹⁾
18, 19, 20	GND _B	Source of low side switch B ⁽¹⁾

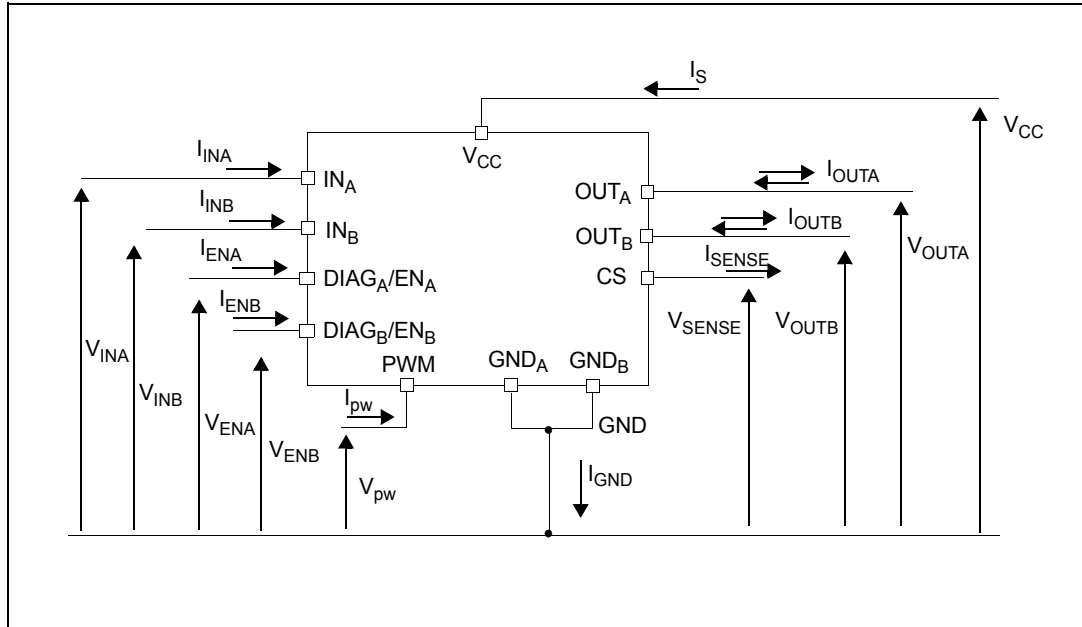
1. GND_A and GND_B must be externally connected together.

Table 4. Pin functions description

Name	Description
V_{CC}	Battery connection
GND_A, GND_B	Power grounds; must always be externally connected together
OUT_A, OUT_B	Power connections to the motor
IN_A, IN_B	Voltage controlled input pins with hysteresis, CMOS compatible. These two pins control the state of the bridge in normal operation according to the truth table (brake to V_{CC} , brake to GND, clockwise and counterclockwise).
PWM	Voltage controlled input pin with hysteresis, CMOS compatible. Gates of low side FETs are modulated by the PWM signal during their ON phase allowing speed control of the motor.
$EN_A/DIAG_A, EN_B/DIAG_B$	Open drain bidirectional logic pins. These pins must be connected to an external pull up resistor. When externally pulled low, they disable half-bridge A or B. In case of fault detection (thermal shutdown of a high side FET or excessive ON state voltage drop across a low side FET), these pins are pulled low by the device (see truth table in fault condition).
CS	Analog current sense output. This output sources a current proportional to the motor current. The information can be read back as an analog voltage across an external resistor.

2 Electrical specifications

Figure 3. Current and voltage conventions



2.1 Absolute maximum ratings

Table 5. Absolute maximum ratings

Symbol	Parameter	Value	Unit
V_{CC}	Supply voltage	+41	V
I_{max}	Maximum output current (continuous)	30	A
I_R	Reverse output current (continuous)	-30	
I_{IN}	Input current (IN _A and IN _B pins)	±10	mA
I_{EN}	Enable input current (DIAG _A /EN _A and DIAG _B /EN _B pins)	±10	
I_{pw}	PWM input current	±10	
V_{CS}	Current sense maximum voltage	-3/+15	V
V_{ESD}	Electrostatic discharge (R = 1.5kΩ, C = 100pF)		
	– CS pin	2	kV
	– logic pins	4	kV
	– output pins: OUT _A , OUT _B , V _{CC}	5	kV
T_j	Junction operating temperature	Internally limited	°C
T_c	Case operating temperature	-40 to 150	
T_{STG}	Storage temperature	-55 to 150	

2.2 Electrical characteristics

$V_{CC} = 9V$ up to $16V$; $-40^{\circ}C < T_J < 150^{\circ}C$, unless otherwise specified.

Table 6. Power section

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
V_{CC}	Operating supply voltage		5.5		16	V
I_S	Supply current	Off state with all Fault Cleared & $EN_x=0$ $I_{N_A} = I_{N_B} = PWM = 0$; $T_J = 25^{\circ}C$; $V_{CC} = 13V$ $I_{N_A} = I_{N_B} = PWM = 0$ Off state: $I_{N_A} = I_{N_B} = PWM = 0$		12	30 60	μA μA mA
		On state: I_{N_A} or $I_{N_B} = 5V$, no PWM			10	mA
R_{ONHS}	Static high side resistance	$I_{OUT} = 15A$; $T_J = 25^{\circ}C$			14	m Ω
		$I_{OUT} = 15A$; $T_J = -40$ to $150^{\circ}C$			28	
R_{ONLS}	Static low side resistance	$I_{OUT} = 15A$; $T_J = 25^{\circ}C$			5	
		$I_{OUT} = 15A$; $T_J = -40$ to $150^{\circ}C$			10	
V_f	High side free-wheeling diode forward voltage	$I_f = 15A$		0.8	1.1	V
$I_{L(off)}$	High side off state output current (per channel)	$T_J = 25^{\circ}C$; $V_{OUTX} = EN_x = 0V$; $V_{CC} = 13V$			3	μA
		$T_J = 125^{\circ}C$; $V_{OUTX} = EN_x = 0V$; $V_{CC} = 13V$			5	
I_{RM}	Dynamic cross-conduction current	$I_{OUT} = 15A$ (see Figure 7)		0.7		A

Table 7. Logic inputs (I_{N_A} , I_{N_B} , EN_A , EN_B)

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
V_{IL}	Input low level voltage	Normal operation ($DIAG_x/EN_x$ pin acts as an input pin)			1.25	V
V_{IH}	Input high level voltage		3.25			
V_{IHYST}	Input hysteresis voltage		0.5			
V_{ICL}	Input clamp voltage	$I_{IN} = 1mA$	5.5	6.3	7.5	
		$I_{IN} = -1mA$	-1.0	-0.7	-0.3	
I_{INL}	Input low current	$V_{IN} = 1.25V$	1			μA
I_{INH}	Input high current	$V_{IN} = 3.25V$			10	
V_{DIAG}	Enable output low level voltage	Fault operation ($DIAG_x/EN_x$ pin acts as an output pin); $I_{EN} = 1mA$			0.4	V

Table 8. PWM

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
V_{pwl}	PWM low level voltage				1.25	V
I_{pwl}	PWM pin current	$V_{pw} = 1.25V$	1			μA
V_{pwh}	PWM high level voltage		3.25			V
I_{pwh}	PWM pin current	$V_{pw} = 3.25V$			10	μA
$V_{pwhhyst}$	PWM hysteresis voltage		0.5			V
V_{pwc}	PWM clamp voltage	$I_{pw} = 1mA$	$V_{CC} + 0.3$	$V_{CC} + 0.7$	$V_{CC} + 1.0$	
		$I_{pw} = -1mA$	-6.0	-4.5	-3.0	
C_{INPWM}	PWM pin input capacitance	$V_{IN} = 2.5V$			25	pF

Table 9. Switching ($V_{CC} = 13V$, $R_{LOAD} = 0.87\ \Omega$, unless otherwise specified)

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
f	PWM frequency		0		20	kHz
$t_{d(on)}$	Turn-on delay time	Input rise time < 1 μs (see Figure 6)			250	μs
$t_{d(off)}$	Turn-off delay time	Input rise time < 1 μs (see Figure 6)			250	
t_r	Rise time	(see Figure 5)		1	1.6	
t_f	Fall time	(see Figure 5)		1.2	2.4	
t_{DEL}	Delay time during change of operating mode	(see Figure 4)	300	600	1800	
t_{rr}	High side free wheeling diode reverse recovery time	(see Figure 7)		110		ns
$t_{off(min)}^{(1)}$	PWM minimum off time	$9V < V_{CC} < 16V$; $T_j = 25^\circ C$; $L = 250\mu H$; $I_{OUT} = 15A$			6	μs

1. To avoid false short to battery detection during PWM operation, the PWM signal must be low for a time longer than 6 μs .

Table 10. Protection and diagnostic

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
V _{USD}	Undervoltage shutdown				5.5	V
	Undervoltage reset			4.7		
V _{OV}	Overvoltage shutdown		16	19	22	
I _{LIM}	High side current limitation		30	50	70	A
V _{CLP}	Total clamp voltage (V _{CC} to GND)	I _{OUT} = 15A	43	48	54	V
T _{TSD}	Thermal shutdown temperature	V _{IN} = 3.25V	150	175	200	°C
T _{TR}	Thermal reset temperature		135			
T _{HYST}	Thermal hysteresis		7	15		

Table 11. Current sense (9 V < V_{CC} < 16 V)

Symbol	Parameter	Test conditions	Min	Typ	Max	Unit
K ₁	I _{OUT} /I _{SENSE}	I _{OUT} = 30A; R _{SENSE} = 1.5kΩ; T _j = -40 to 150°C	9665	11370	13075	
K ₂	I _{OUT} /I _{SENSE}	I _{OUT} = 8A; R _{SENSE} = 1.5kΩ; T _j = -40 to 150°C	9096	11370	13644	
dK ₁ / K ₁ ⁽¹⁾	Analog sense current drift	I _{OUT} = 30A; R _{SENSE} = 1.5kΩ; T _j = -40 to 150°C	-8		+8	%
dK ₂ / K ₂ ⁽¹⁾	Analog sense current drift	I _{OUT} > 8A; R _{SENSE} = 1.5kΩ; T _j = -40 to 150°C	-10		+10	
I _{SENSE0}	Analog sense leakage current	I _{OUT} = 0A; V _{SENSE} = 0V; T _j = -40 to 150°C	0		65	μA

1. Analog sense current drift is deviation of factor K for a given device over (-40 °C to 150 °C and 9 V < V_{CC} < 16 V) with respect to its value measured at T_j = 25°C, V_{CC} = 13 V.

Figure 4. Definition of the delay times measurement

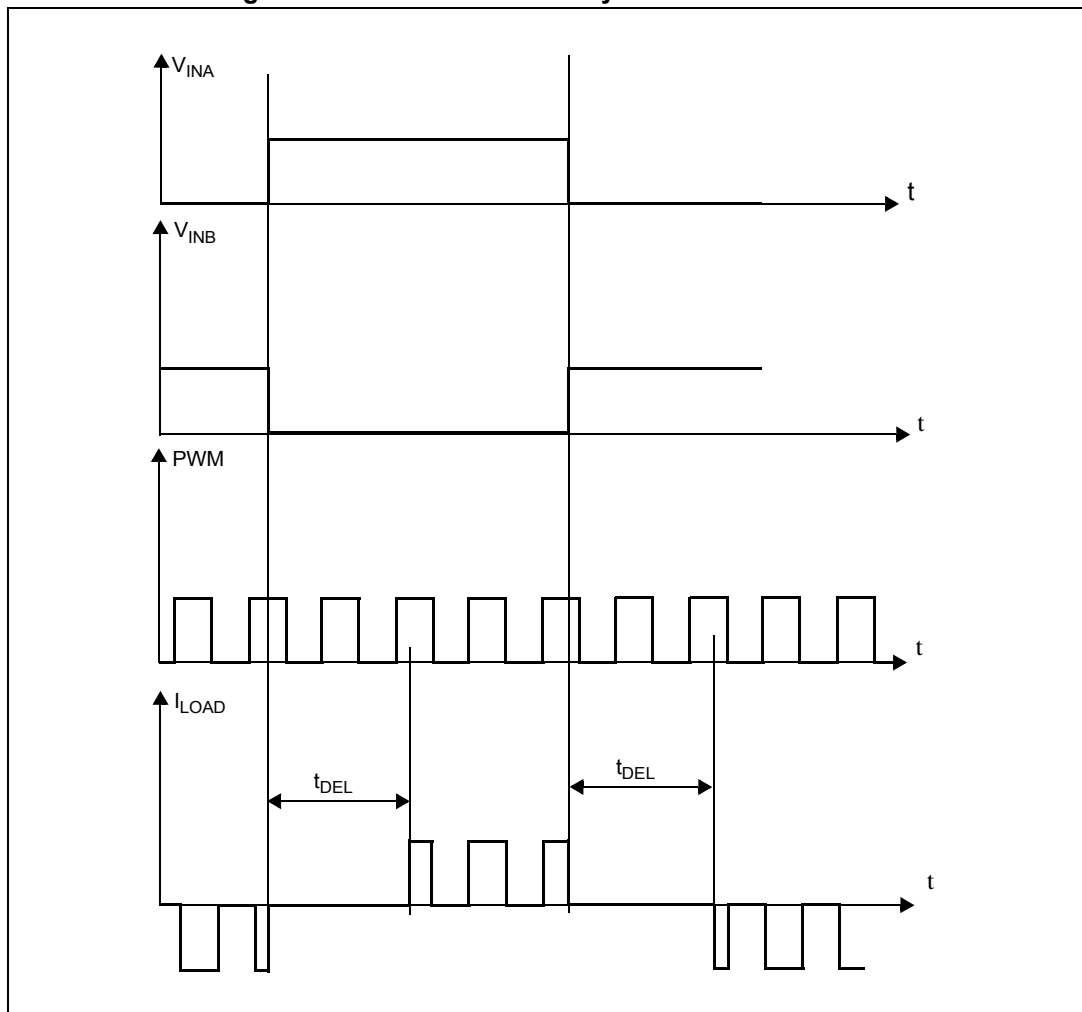


Figure 5. Definition of the low side switching times

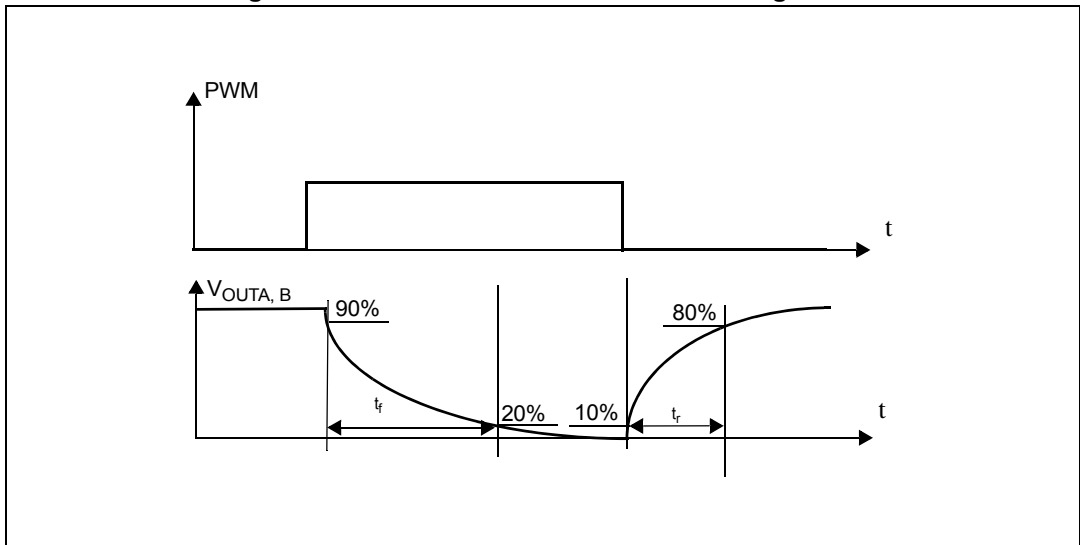


Figure 6. Definition of the high side switching times

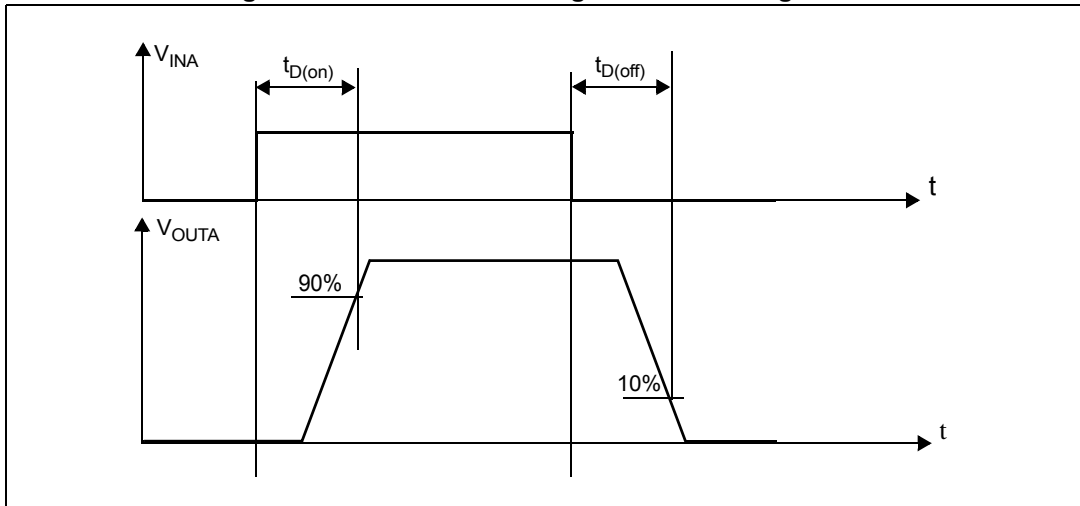


Figure 7. Definition of dynamic cross conduction current during a PWM operation

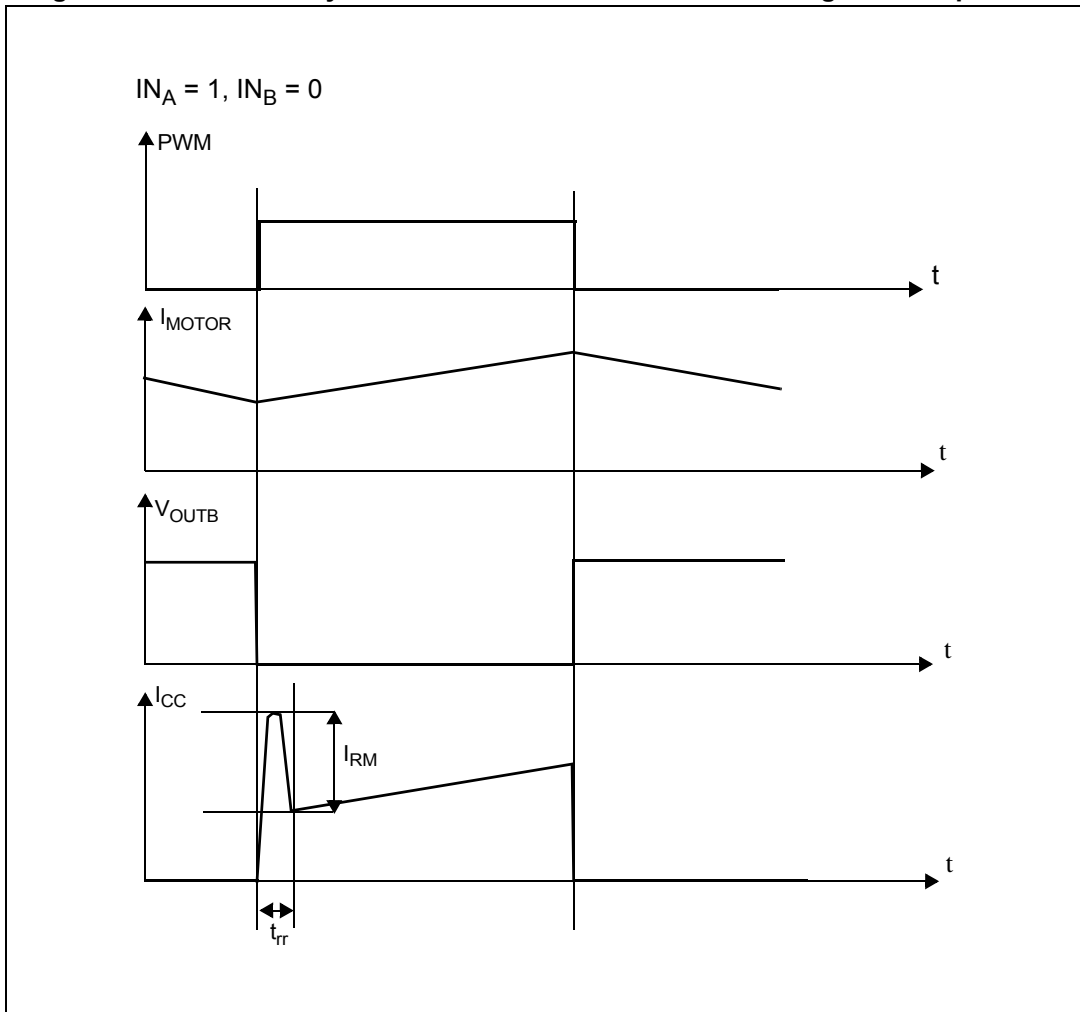


Table 12. Truth table in normal operating conditions

IN _A	IN _B	DIAG _A /EN _A	DIAG _B /EN _B	OUT _A	OUT _B	CS	Operating mode
1	1	1	1	H	H	High Imp.	Brake to V _{CC}
	L				I _{SENSE} = I _{OUT} /K	Clockwise (CW)	
0	1			L		H	High imp.
	0			L	L	High imp.	Brake to GND

Table 13. Truth table in fault conditions (detected on OUT_A)

IN _A	IN _B	DIAG _A /EN _A	DIAG _B /EN _B	OUT _A	OUT _B	CS
1	1	0	1	OPEN	H	High Imp.
	0				L	
0	1				H	I _{OUTB} /K
	0					L
X	X				OPEN	
	1				H	I _{OUTB} /K
	0	L	High Imp.			

↑
Fault Information
↑
Protection Action

Note: The saturation detection on the low side power MOSFET is possible only if the impedance of the short-circuit from the output to the battery is less than 100 mΩ when the device is supplied with a battery voltage of 13.5 V.

Table 14. Electrical transient requirements

ISO T/R - 7637/1 test pulse	Test level I	Test level II	Test level III	Test level IV	Test levels delays and impedance
1	-25V	-50V	-75V	-100V	2ms, 10Ω
2	+25V	+50V	+75V	+100V	0.2ms, 10Ω
3a	-25V	-50V	-100V	-150V	0.1μs, 50Ω
3b	+25V	+50V	+75V	+100V	
4	-4V	-5V	-6V	-7V	100ms, 0.01Ω
5	+26.5V	+46.5V	+66.5V	+86.5V	400ms, 2Ω

ISO T/R - 7637/1 test pulse	Test levels result I	Test levels result II	Test levels result III	Test levels result IV
1	C	C	C	C
2				
3a				
3b				
4				
5 ⁽¹⁾		E	E	E

1. For load dump exceeding the above value a centralized suppressor must be adopted.

Class	Contents
C	All functions of the device are performed as designed after exposure to disturbance.
E	One or more functions of the device are not performed as designed after exposure to disturbance and cannot be returned to proper operation without replacing the device.

2.3 Electrical characteristics curves

Figure 8. On state supply current

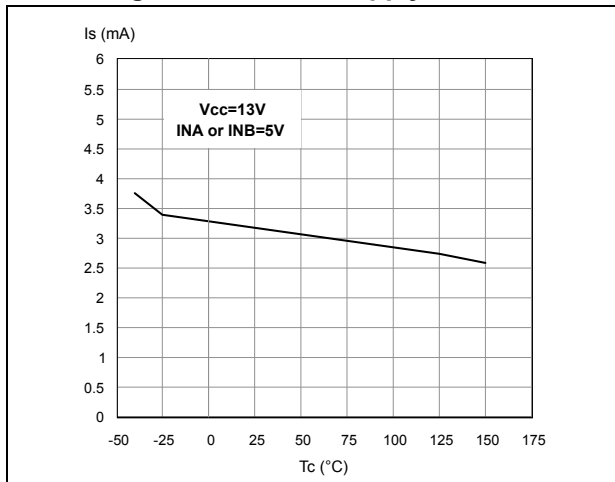


Figure 9. Off state supply current

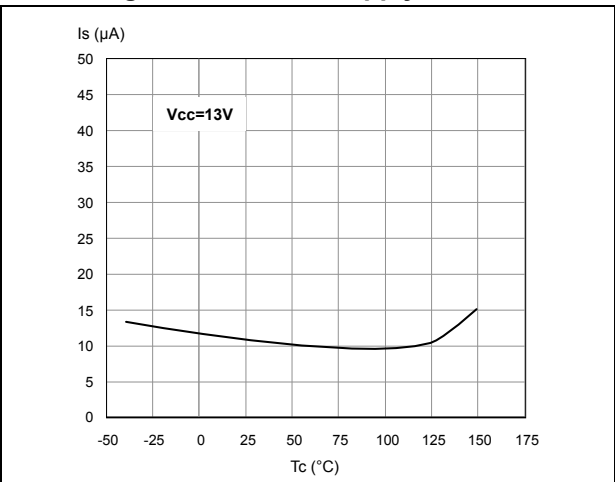


Figure 10. High level input current

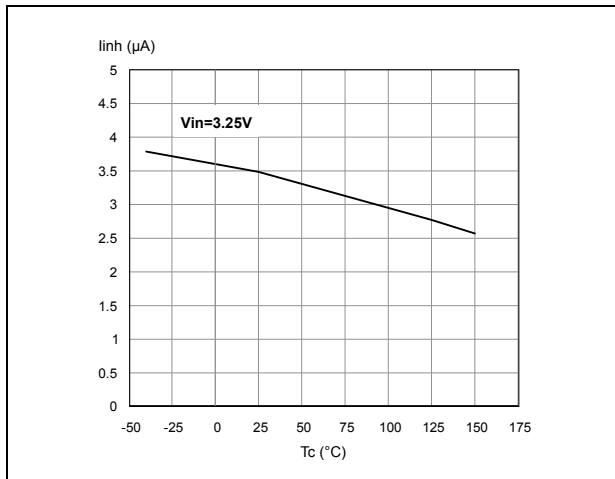


Figure 11. Input clamp voltage

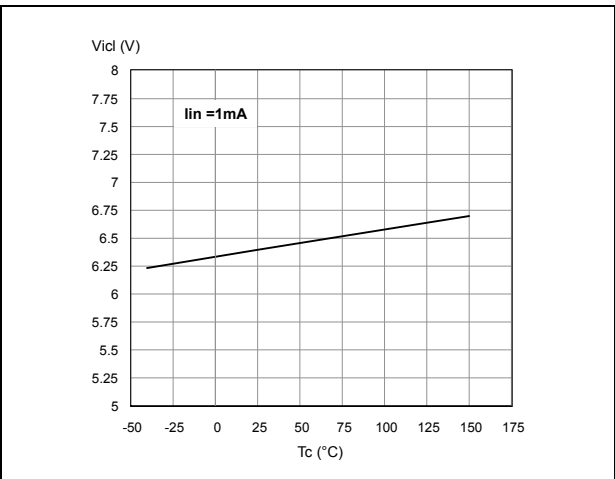


Figure 12. Input high level voltage

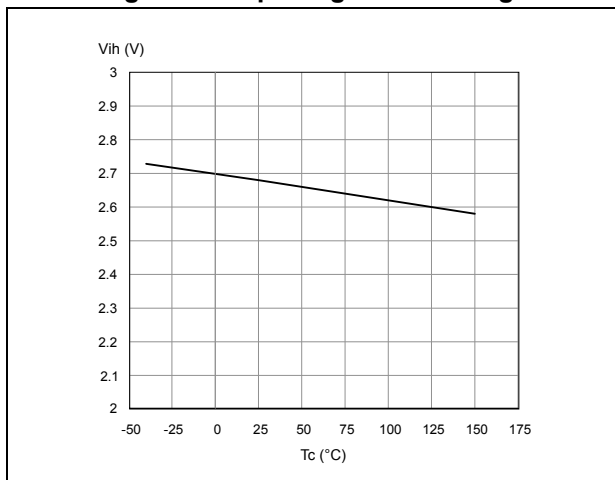


Figure 13. Input low level voltage

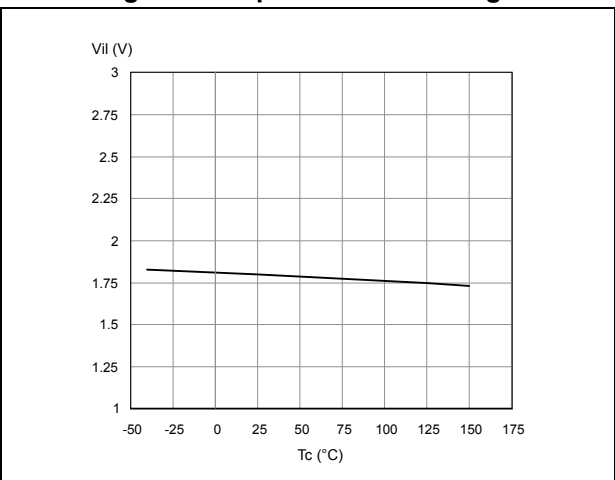


Figure 14. Input hysteresis voltage

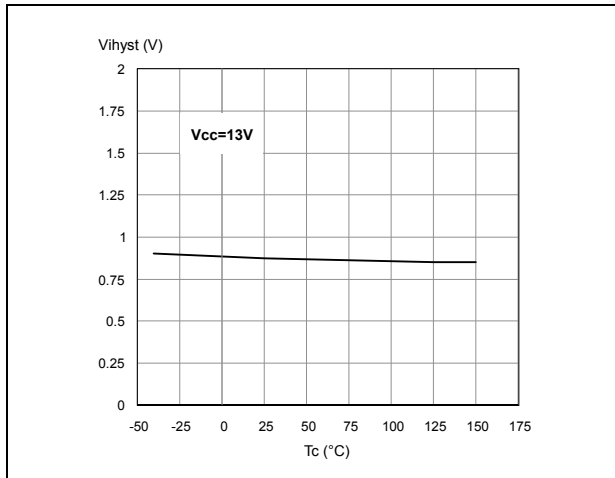


Figure 15. High level enable pin current

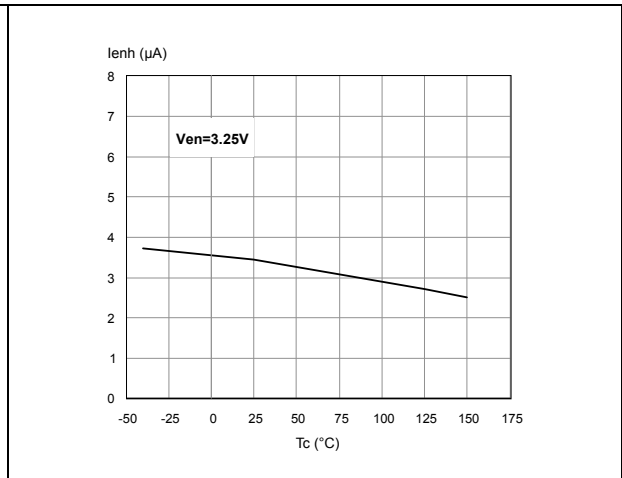


Figure 16. Delay time during change of operation mode

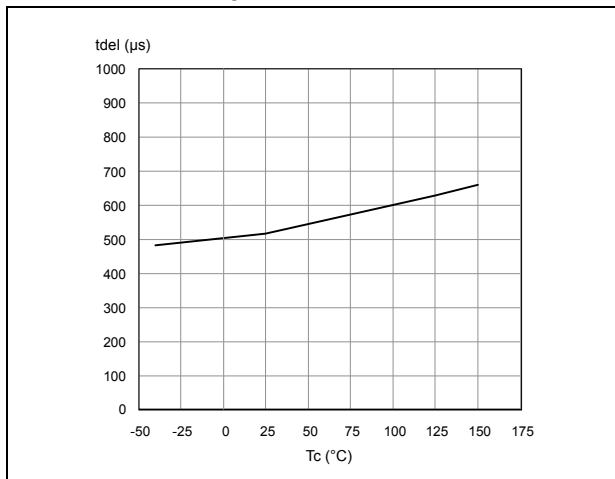


Figure 17. Enable clamp voltage

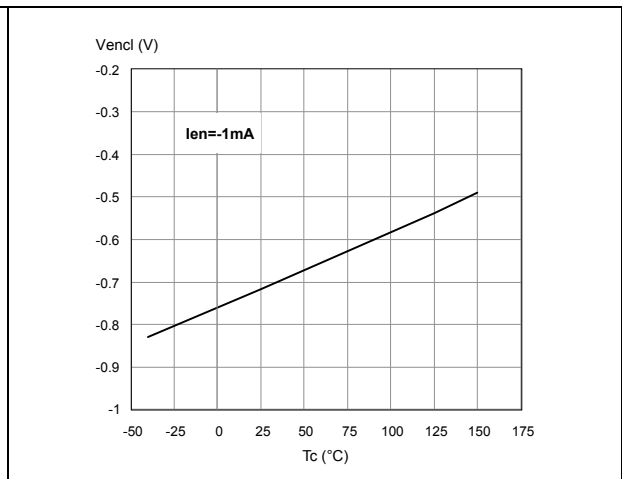


Figure 18. High level enable voltage

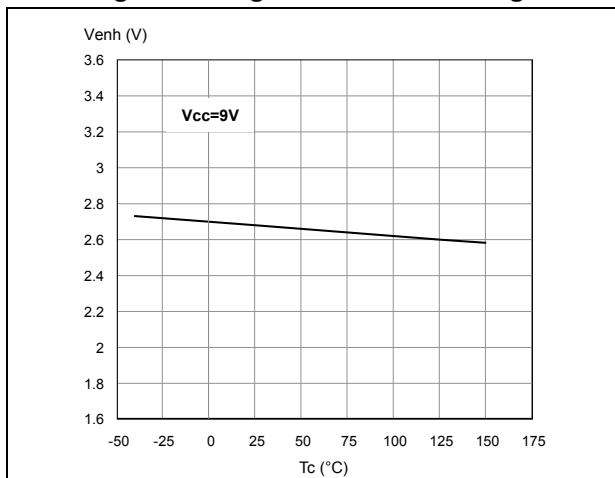


Figure 19. Low level enable voltage

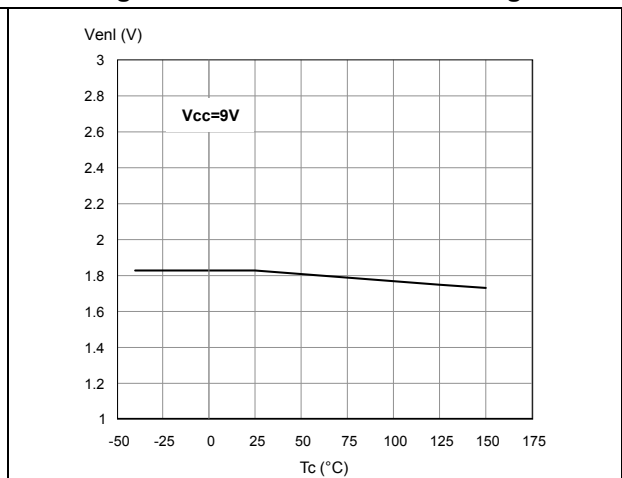


Figure 20. PWM high level voltage

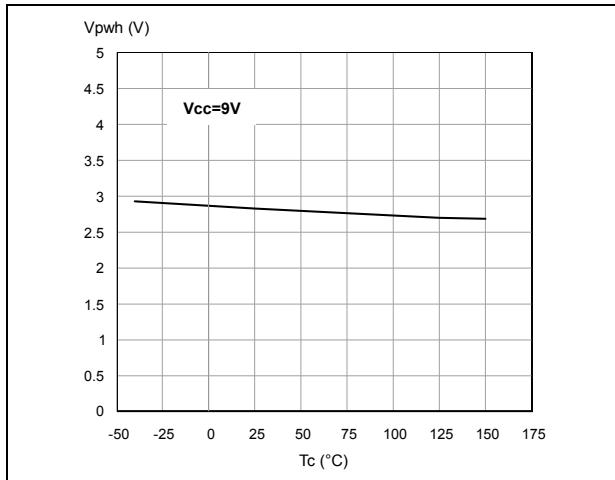


Figure 21. PWM low level voltage

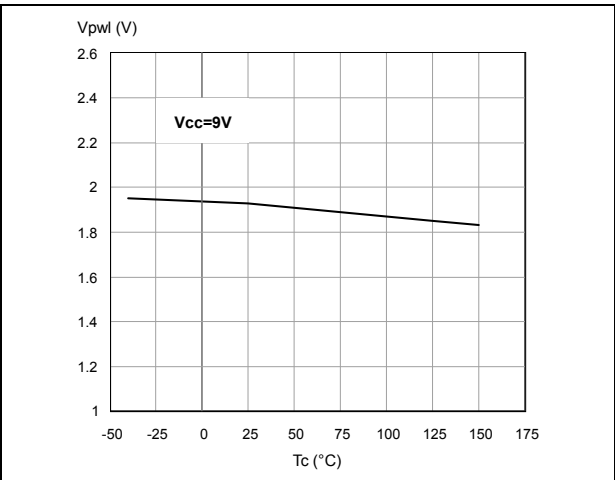


Figure 22. PWM high level current

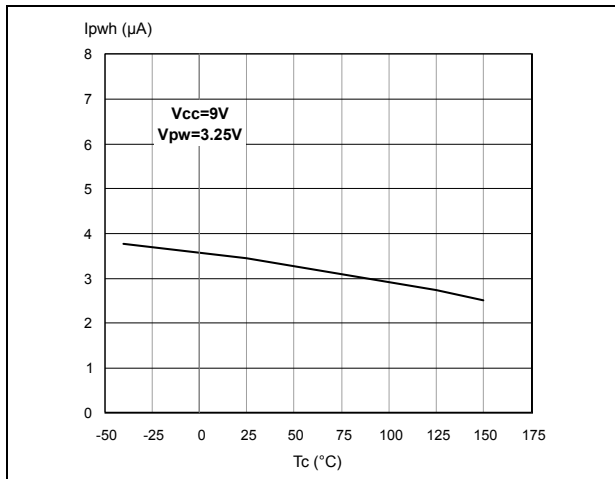


Figure 23. Overvoltage shutdown

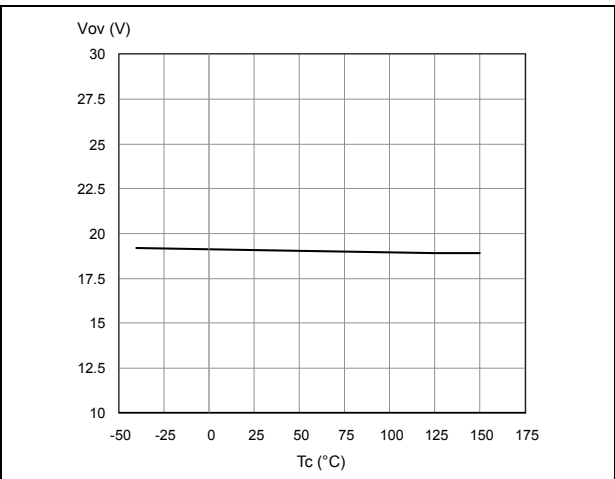


Figure 24. Undervoltage shutdown

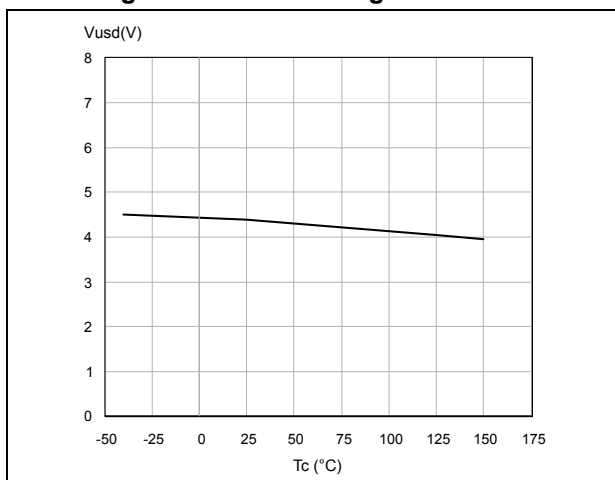


Figure 25. Current limitation

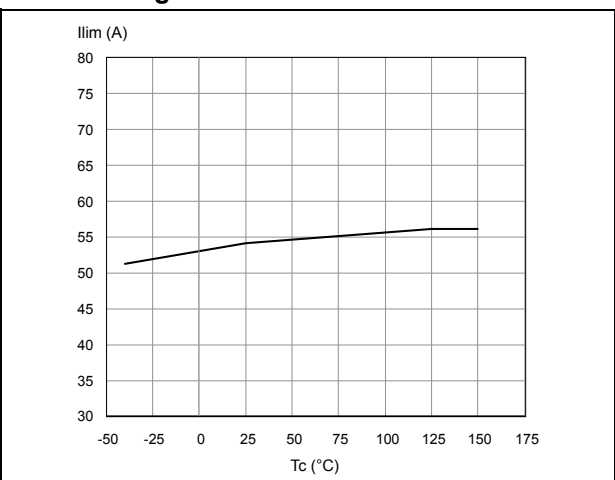


Figure 26. On state high side resistance vs T_{case} Figure 27. On state low side resistance vs T_{case}

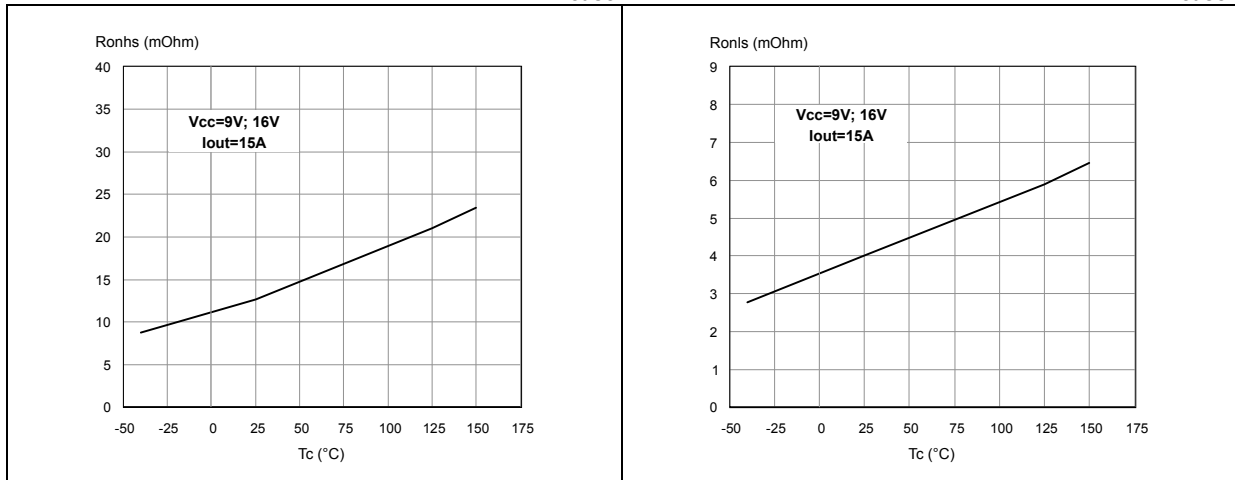


Figure 28. Turn-on delay time

Figure 29. Turn-off delay time

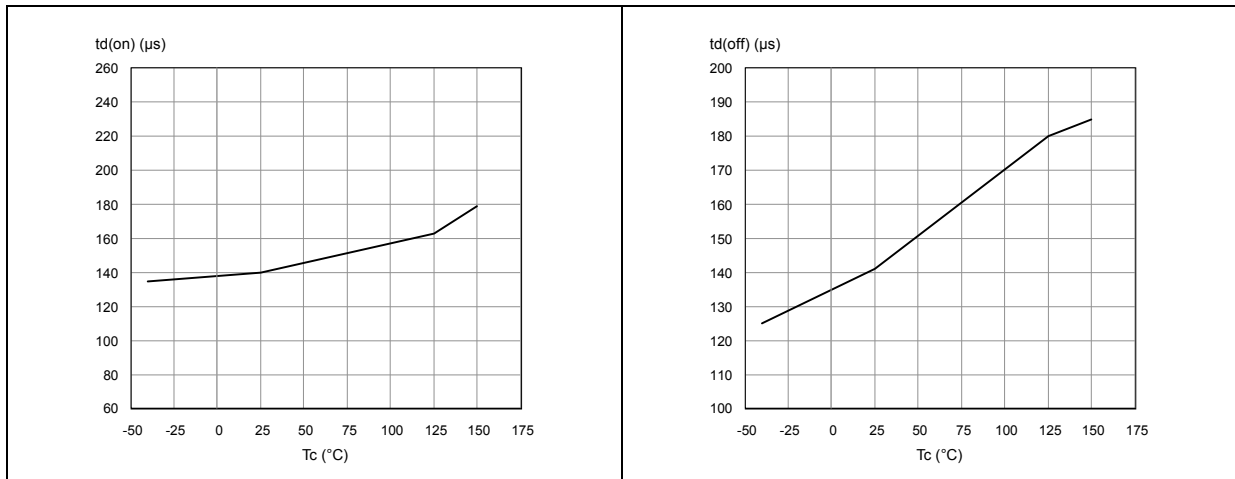
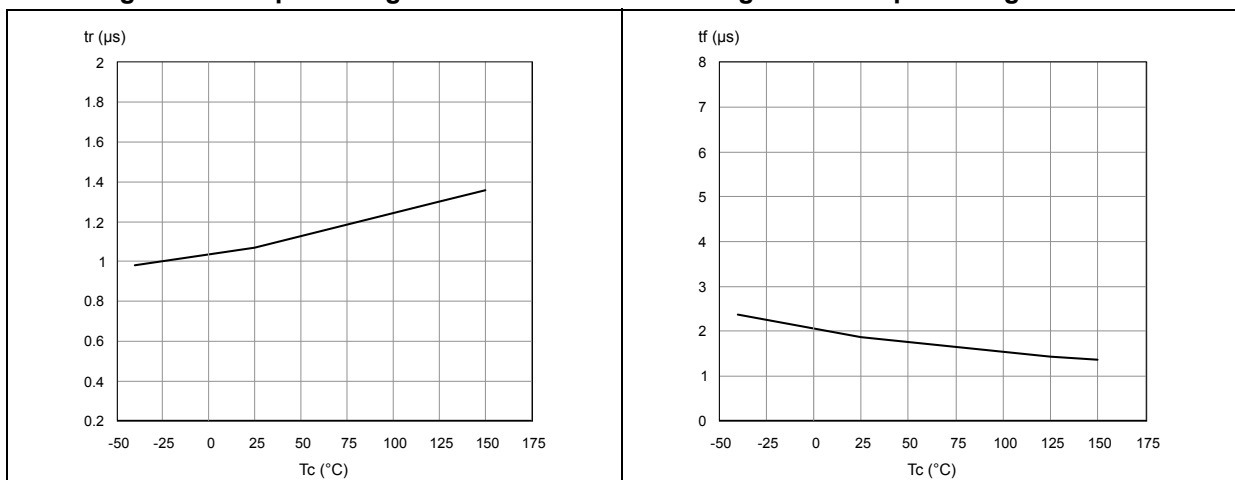


Figure 30. Output voltage rise time

Figure 31. Output voltage fall time

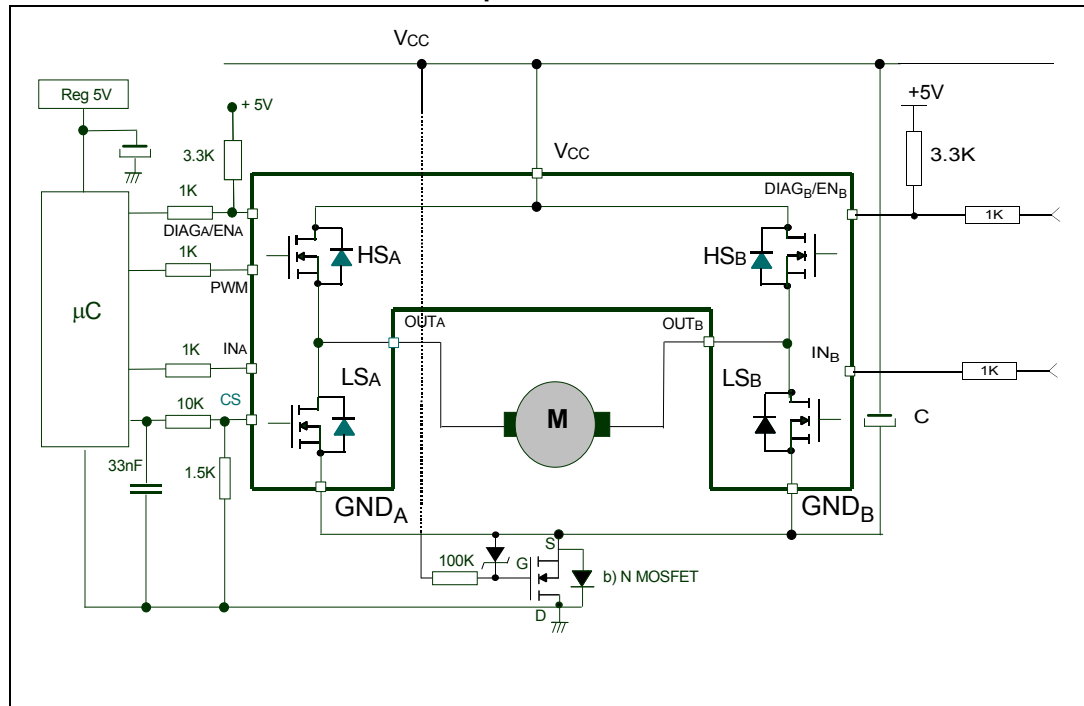


3 Application information

In normal operating conditions the DIAG_X/EN_X pin is considered as an input pin by the device. This pin must be externally pulled high.

PWM pin usage: in all cases, a “0” on the PWM pin will turn off both LS_A and LS_B switches. When PWM rises back to “1”, LS_A or LS_B turn on again depending on the input pin state.

Figure 32. Typical application circuit for DC to 20 kHz PWM operation short-circuit protection



Note: The value of the blocking capacitor (C) depends on the application conditions and defines the voltage and current ripple on the supply line at PWM operation. Stored energy from the motor inductance may fly back into the blocking capacitor if the bridge driver goes into tri-state. This causes a hazardous overvoltage if the capacitor is not large enough. As a basic guideline, 500 µF per 10 A load current is recommended.

In case of a fault condition, the DIAG_X/EN_X pin is considered an output pin by the device. The fault conditions are:

- overtemperature on one or both high sides
- short to battery condition on the output (saturation detection on the low side power MOSFET)

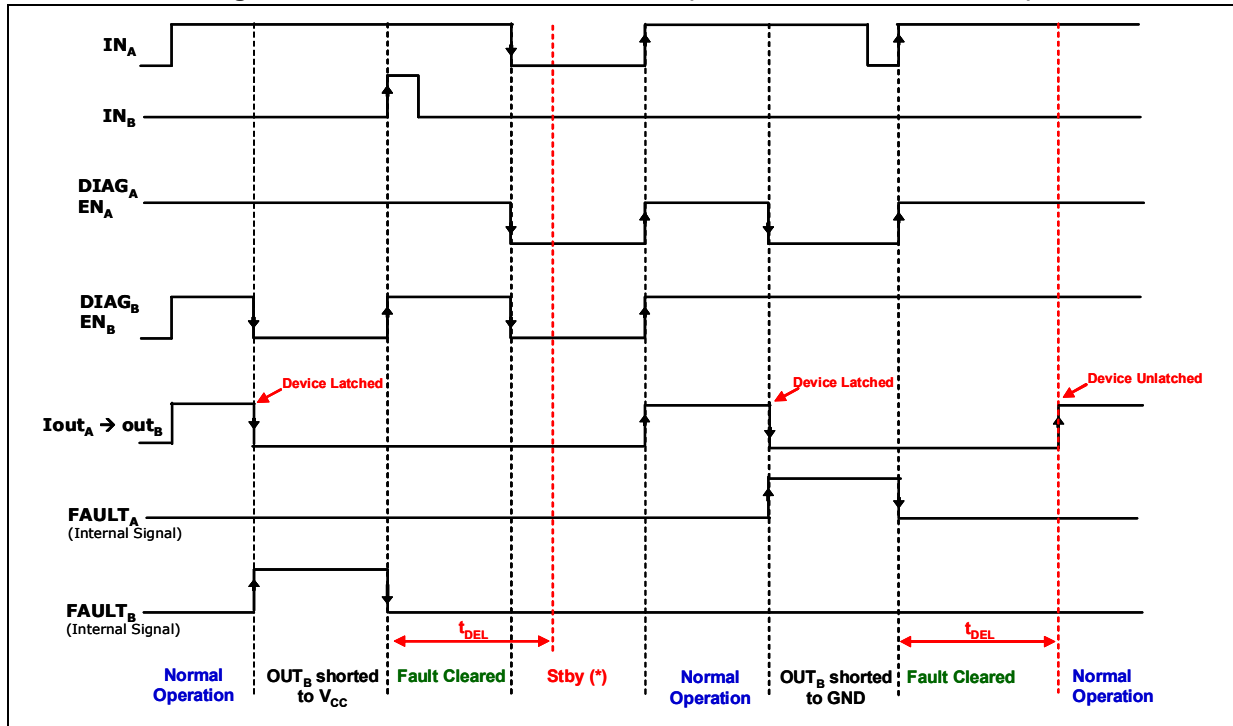
Possible origins of fault conditions may be:

- OUT_A is shorted to ground → overtemperature detection on high side A.
- OUT_A is shorted to V_{CC} → low side power MOSFET saturation detection.

When a fault condition is detected, the user can be informed of which power element is in fault by monitoring the IN_A, IN_B, DIAG_A/EN_A and DIAG_B/EN_B pins.

In any case, when a fault is detected, the faulty leg of the bridge is latched off. To turn on the respective output (OUT_x) again, the input signal must rise from low to high level.

Figure 33. Behavior in fault condition (how a fault can be cleared)



Note: In case of the fault condition is not removed, the procedure for unlatching and sending the device into Stby mode is:

- Clear the fault in the device (toggle : IN_A if EN_A=0 or IN_B if EN_B=0)
- Pull low all inputs, PWM and Diag/EN pins within t_{DEL}.

If the Diag/En pins are already low, PWM=0, the fault can be cleared simply toggling the input. The device will enter Stby mode as soon as the fault is cleared.

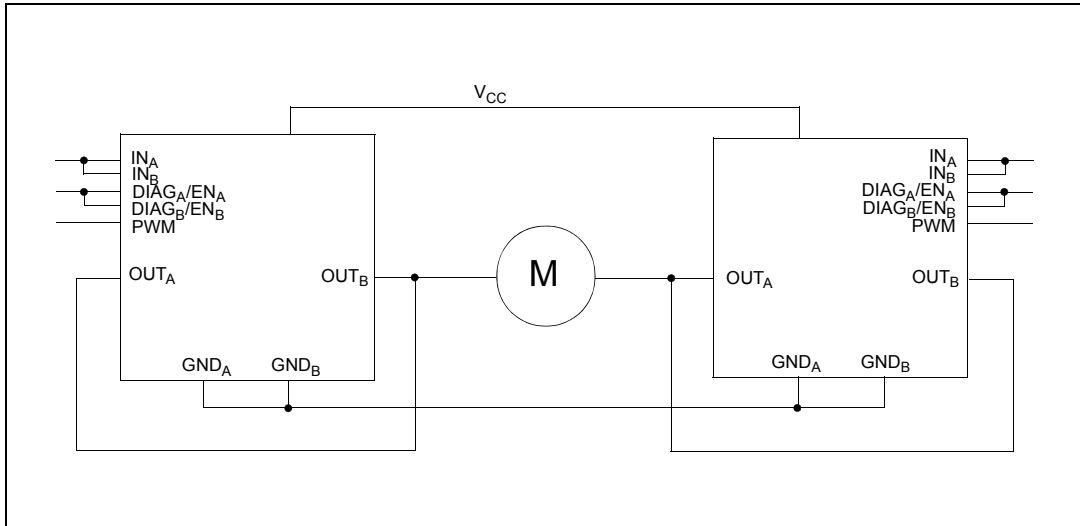
3.1 Reverse battery protection

Three possible solutions can be considered:

1. a Schottky diode D connected to V_{CC} pin
2. an N-channel MOSFET connected to the GND pin (see [Figure 32: Typical application circuit for DC to 20 kHz PWM operation short-circuit protection on page 21](#))
3. a P-channel MOSFET connected to the V_{CC} pin

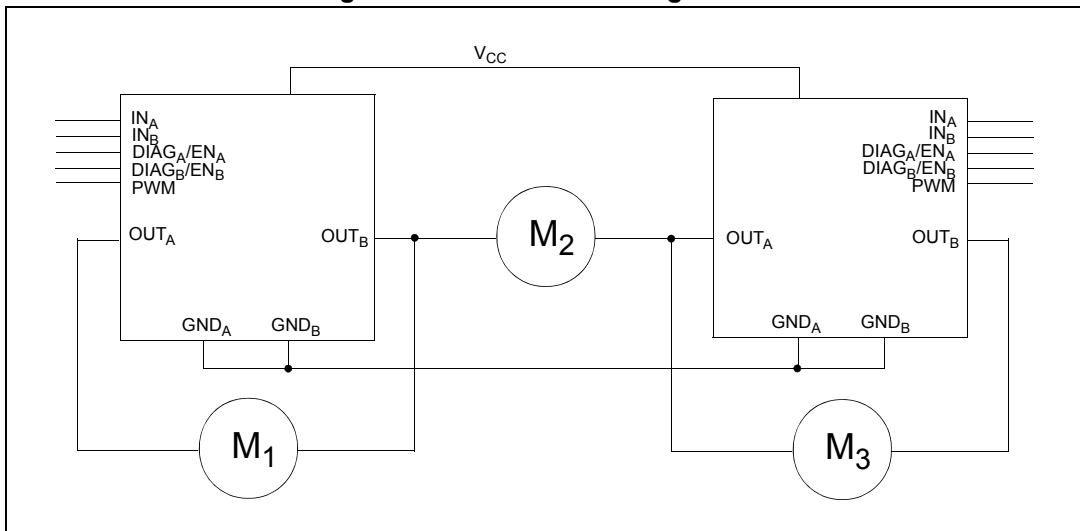
The device sustains no more than -30 A in reverse battery conditions because of the two body diodes of the power MOSFETs. Additionally, in reverse battery condition the I/Os of the VNH2SP30-E are pulled down to the V_{CC} line (approximately -1.5 V). A series resistor must be inserted to limit the current sunk from the microcontroller I/Os. If I_{Rmax} is the maximum target reverse current through μC I/Os, the series resistor is:

Figure 34. Half-bridge configuration



Note: The VNH2SP30-E can be used as a high power half-bridge driver achieving an On resistance per leg of 9.5 mΩ.

Figure 35. Multi-motor configuration



Note: The VNH2SP30-E can easily be designed in multi-motor driving applications such as seat positioning systems where only one motor must be driven at a time. The $DIAG_X/EN_X$ pins allow the unused half-bridges to be put into high impedance.

Figure 36. Waveforms in full bridge operation

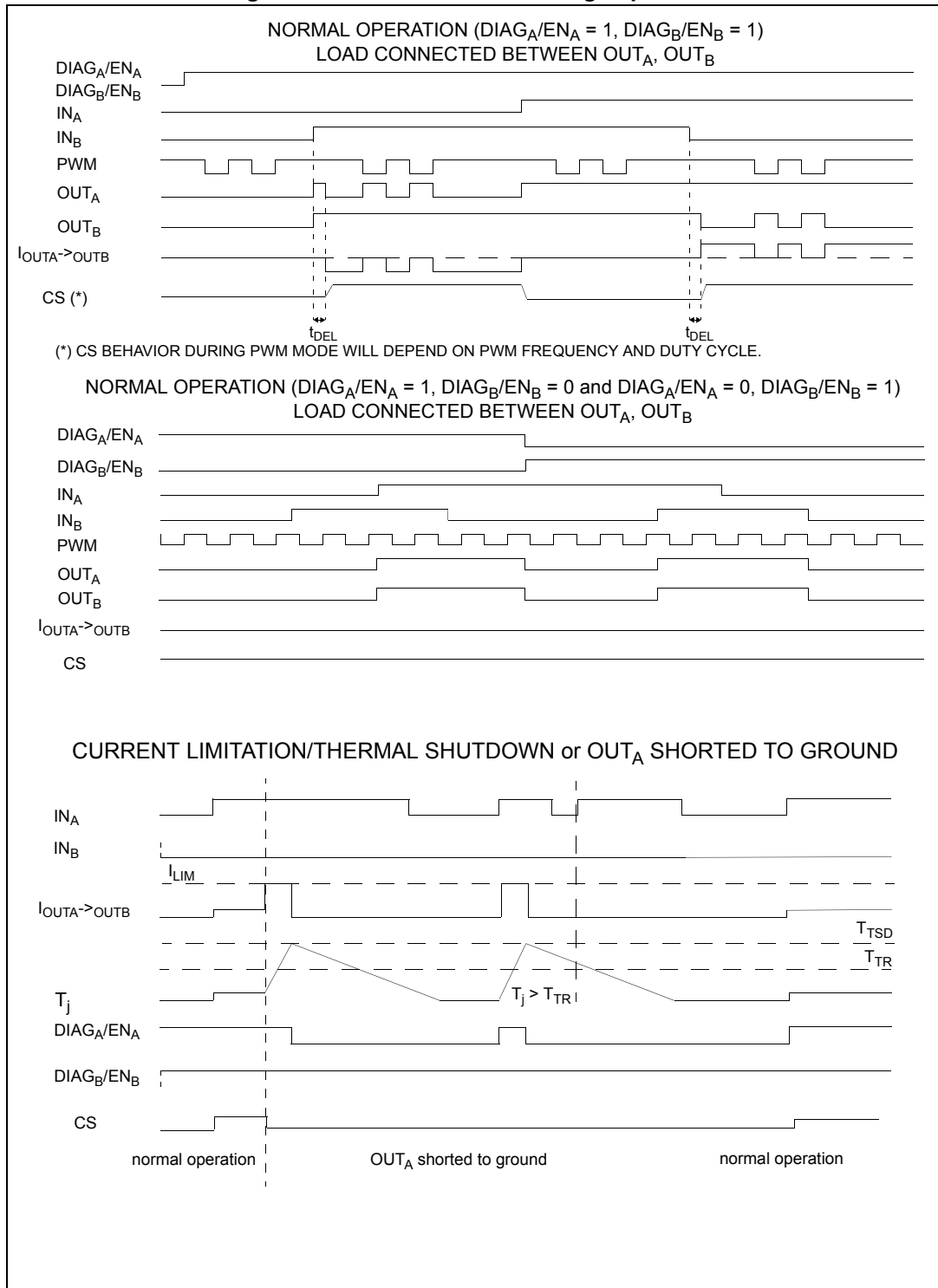
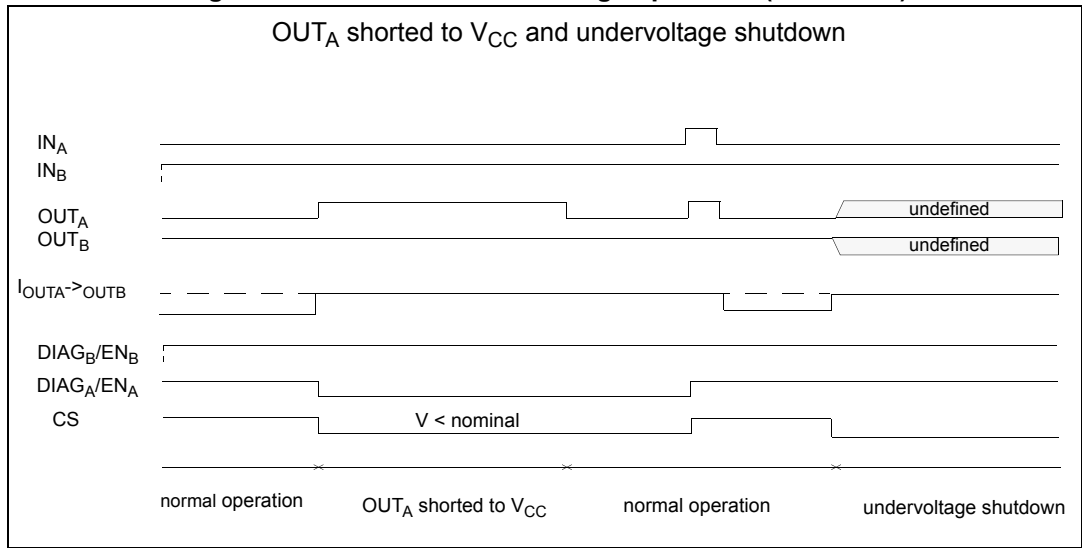


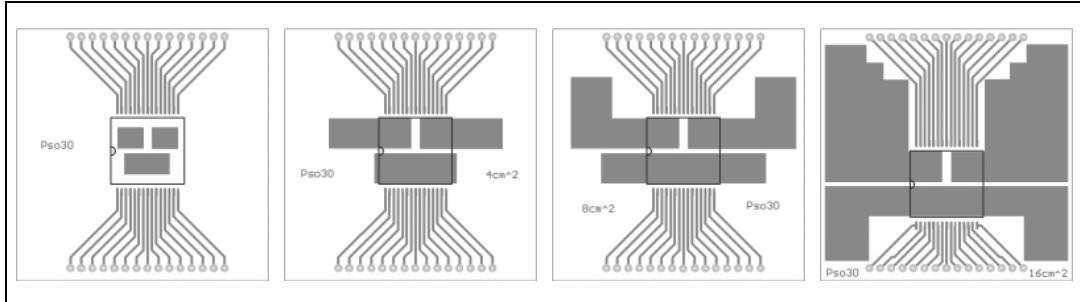
Figure 37. Waveforms in full bridge operation (continued)



4 Package and PCB thermal data

4.1 PowerSSO-30 thermal data

Figure 38. MultiPowerSO-30™ PC board



Note: Layout condition of R_{th} and Z_{th} measurements (PCB FR4 area = 58 mm x 58 mm, PCB thickness = 2mm. Cu thickness = 35 μ m, Copper areas: from minimum pad layout to 16 cm²).

Figure 39. Chipset configuration

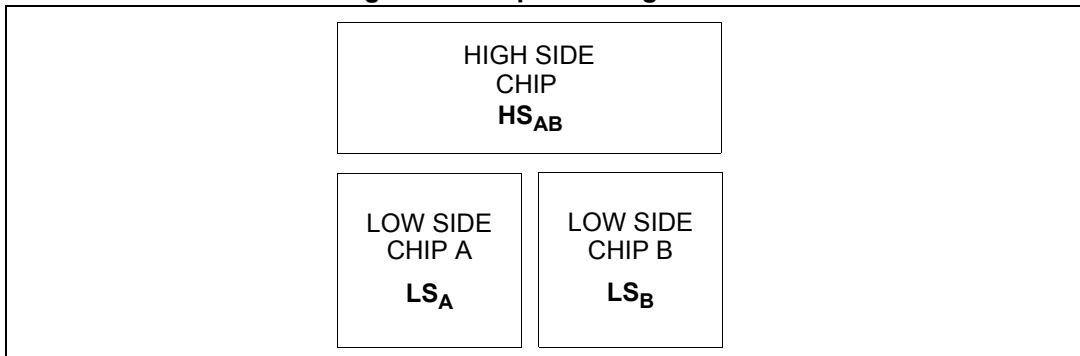
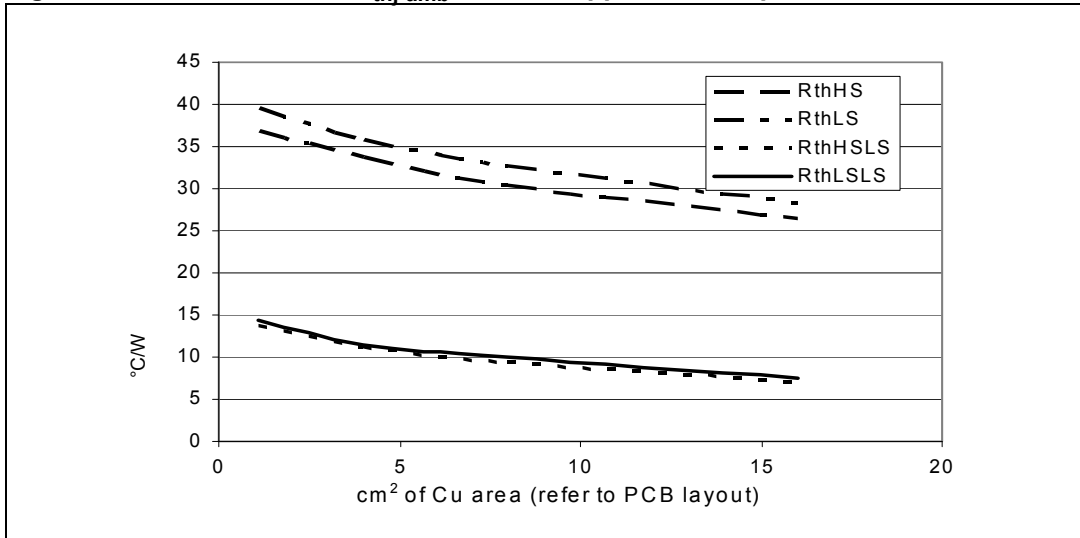


Figure 40. Auto and mutual $R_{thj-amb}$ vs PCB copper area in open box free air condition



4.1.1 Thermal calculation in clockwise and anti-clockwise operation in steady-state mode

Table 15. Thermal calculation in clockwise and anti-clockwise operation in steady-state mode

HS _A	HS _B	LS _A	LS _B	T _{jHSAB}	T _{jLSA}	T _{jLSB}
ON	OFF	OFF	ON	$P_{dHSA} \times R_{thHS} + P_{dLSB} \times R_{thHSL} + T_{amb}$	$P_{dHSA} \times R_{thHSL} + P_{dLSB} \times R_{thLSL} + T_{amb}$	$P_{dHSA} \times R_{thHSL} + P_{dLSB} \times R_{thLS} + T_{amb}$
OFF	ON	ON	OFF	$P_{dHSB} \times R_{thHS} + P_{dLSA} \times R_{thHSL} + T_{amb}$	$P_{dHSB} \times R_{thHSL} + P_{dLSA} \times R_{thLS} + T_{amb}$	$P_{dHSB} \times R_{thHSL} + P_{dLSA} \times R_{thLSL} + T_{amb}$

4.1.2 Thermal resistance definitions (values according to the PCB heatsink area)

$R_{thHS} = R_{thHSA} = R_{thHSB}$ = High Side Chip Thermal Resistance Junction to Ambient (HS_A or HS_B in ON state)

$R_{thLS} = R_{thLSA} = R_{thLSB}$ = Low Side Chip Thermal Resistance Junction to Ambient

$R_{thHSL} = R_{thHSALS} = R_{thHSBLS}$ = Mutual Thermal Resistance Junction to Ambient between High Side and Low Side Chips

$R_{thLSL} = R_{thLSALS} = R_{thLSBLS}$ = Mutual Thermal Resistance Junction to Ambient between Low Side Chips

4.1.3 Thermal calculation in transient mode^(a)

$$T_{jHSAB} = Z_{thHS} \times P_{dHSAB} + Z_{thHSL} \times (P_{dLSA} + P_{dLSB}) + T_{amb}$$

$$T_{jLSA} = Z_{thHSL} \times P_{dHSAB} + Z_{thLS} \times P_{dLSA} + Z_{thLSL} \times P_{dLSB} + T_{amb}$$

$$T_{jLSB} = Z_{thHSL} \times P_{dHSAB} + Z_{thLSL} \times P_{dLSA} + Z_{thLS} \times P_{dLSB} + T_{amb}$$

4.1.4 Single pulse thermal impedance definition (values according to the PCB heatsink area)

Z_{thHS} = High Side Chip Thermal Impedance Junction to Ambient

$Z_{thLS} = Z_{thLSA} = Z_{thLSB}$ = Low Side Chip Thermal Impedance Junction to Ambient

$Z_{thHSL} = Z_{thHSALS} = Z_{thHSBLS}$ = Mutual Thermal Impedance Junction to Ambient between High Side and Low Side Chips

$Z_{thLSL} = Z_{thLSALS} = Z_{thLSBLS}$ = Mutual Thermal Impedance Junction to Ambient between Low Side Chips

a. Calculation is valid in any dynamic operating condition. P_d values set by user.

Equation 1: pulse calculation formula

$$Z_{TH\delta} = R_{TH} p \delta + Z_{THtp}(1 - \delta)$$

where $\delta = t_p / T$

Figure 41. MultiPowerSO-30 HSD thermal impedance junction ambient single pulse

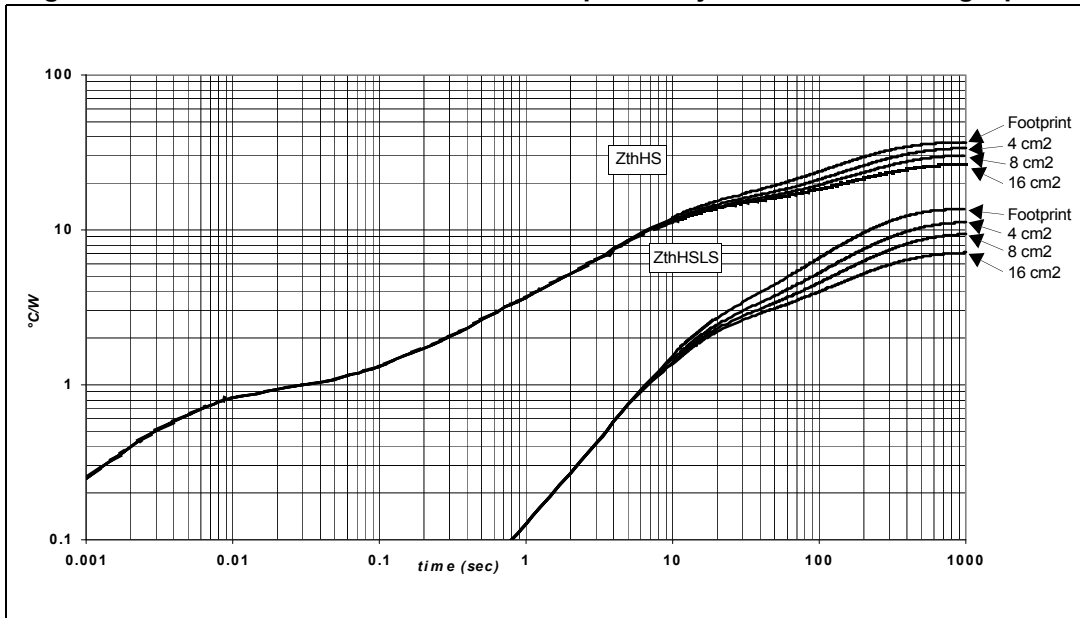


Figure 42. MultiPowerSO-30 LSD thermal impedance junction ambient single pulse

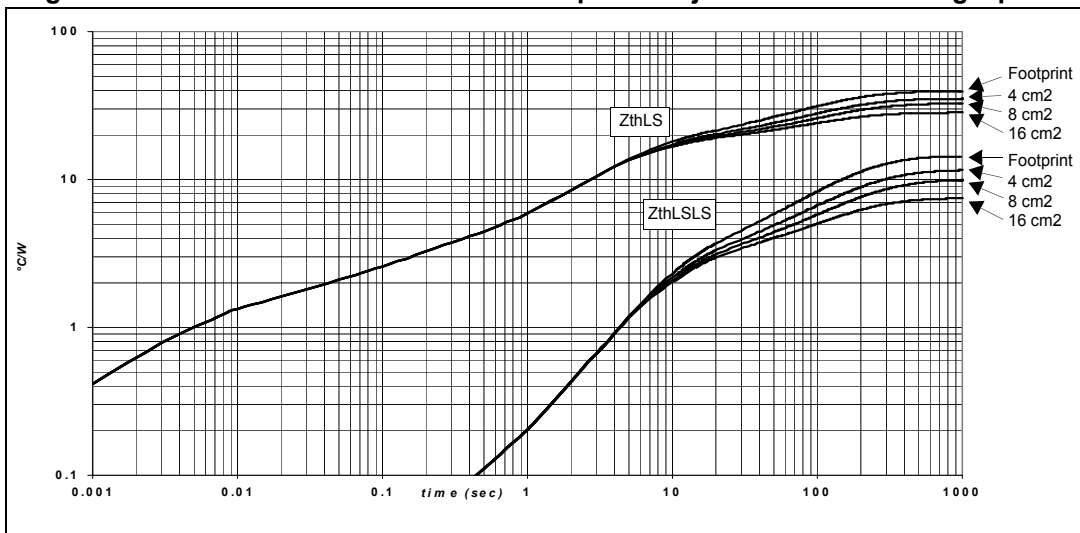


Figure 43. Thermal fitting model of an H-bridge in MultiPowerSO-30

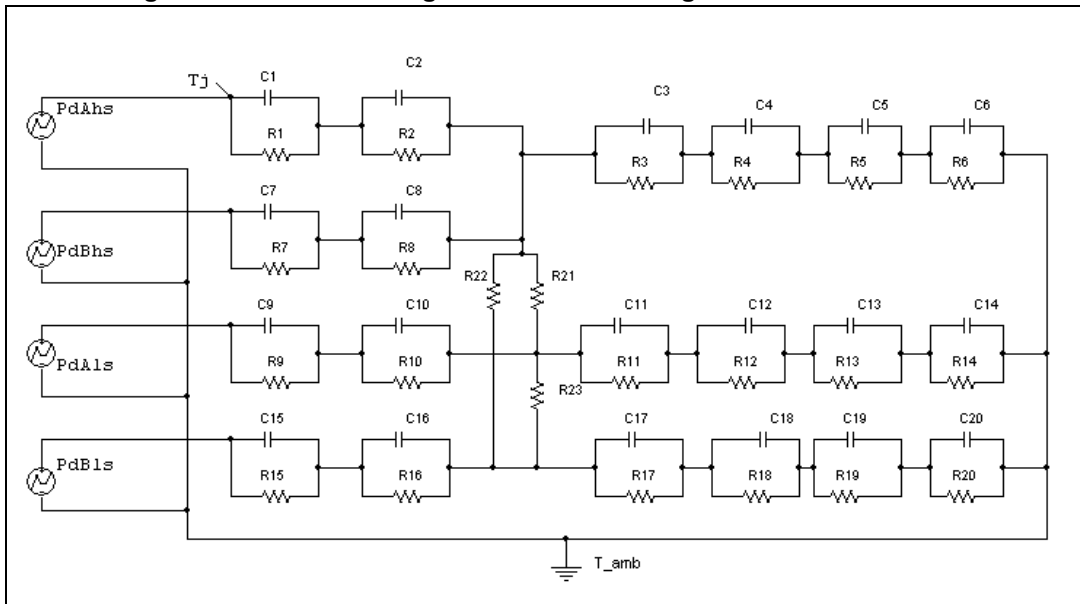


Table 16. Thermal parameters⁽¹⁾

Area/island (cm2)	Footprint	4	8	16
R1 = R7 (°C/W)	0.05			
R2 = R8 (°C/W)	0.3			
R3 (°C/W)	0.5			
R4 (°C/W)	1.3			
R5 (°C/W)	14			
R6 (°C/W)	44.7	39.1	31.6	23.7
R9 = R15 (°C/W)	0.2			
R10 = R16 (°C/W)	0.4			
R11 = R17 (°C/W)	0.8			
R12 = R18 (°C/W)	1.5			
R13 = R19 (°C/W)	20			
R14 = R20 (°C/W)	46.9	36.1	30.4	20.8
R21 = R22 = R23 (°C/W)	115			
C1 = C7 (W.s/°C)	0.005			
C2 = C8 (W.s/°C)	0.008			
C3 = C11 = C17 (W.s/°C)	0.01			
C4 = C13 = C19 (W.s/°C)	0.3			
C5 (W.s/°C)	0.6			
C6 (W.s/°C)	5	7	9	11
C9 = C15 (W.s/°C)	0.003			

Table 16. Thermal parameters⁽¹⁾ (continued)

Area/island (cm ²)	Footprint	4	8	16
C10 = C16 (W.s/°C)	0.006			
C12 = C18 (W.s/°C)	0.075			
C14 = C20 (W.s/°C)	2.5	3.5	4.5	5.5

1. The blank space means that the value is the same as the previous one.

5 Package information

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK[®] packages, depending on their level of environmental compliance. ECOPACK[®] specifications, grade definitions and product status are available at: www.st.com. ECOPACK[®] is an ST trademark.

5.1 MultiPowerSO-30 package information

Figure 44. MultiPowerSO-30 package outline

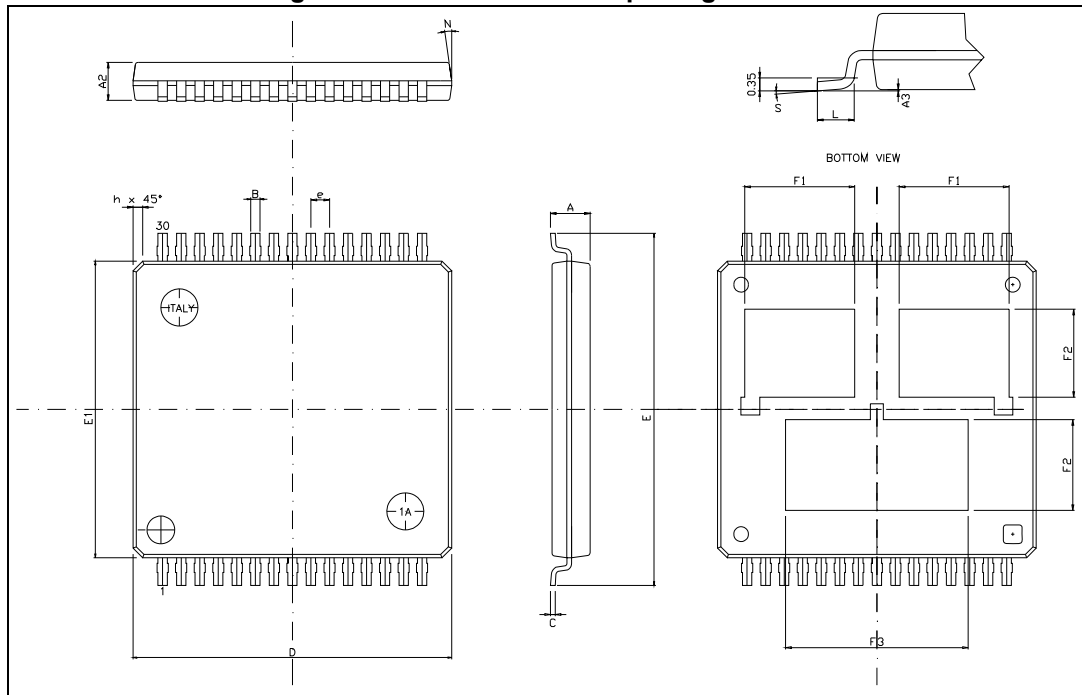


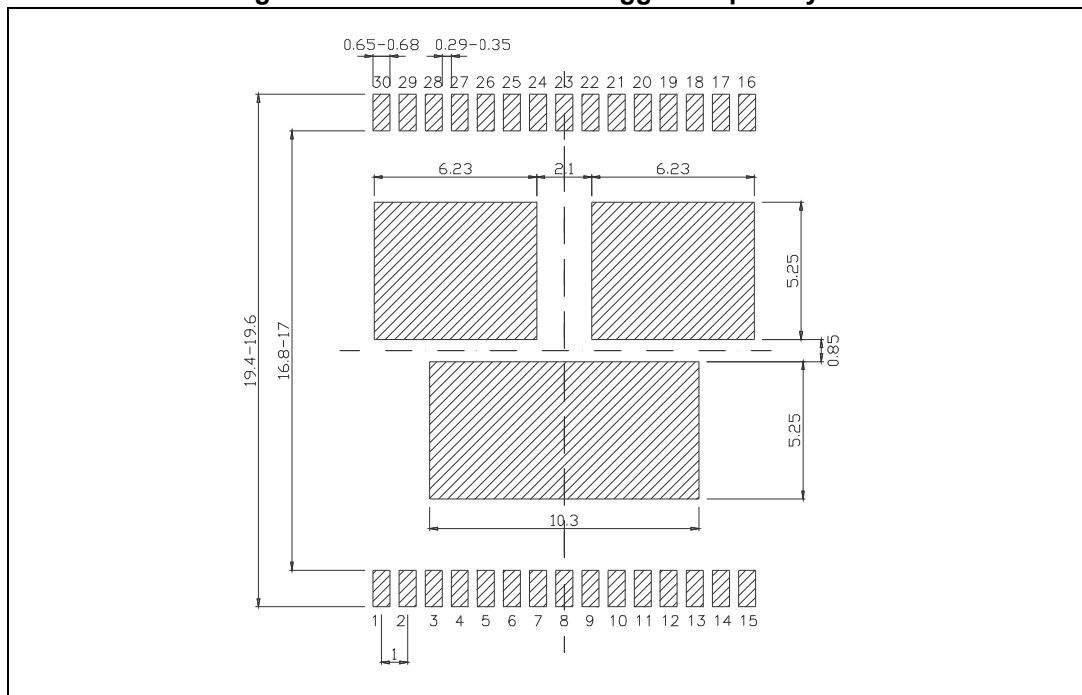
Table 17. MultiPowerSO-30 mechanical data

Symbol	Millimeters		
	Min	Typ	Max
A			2.35
A2	1.85		2.25
A3	0		0.1
B	0.42		0.58
C	0.23		0.32
D	17.1	17.2	17.3
E	18.85		19.15

Table 17. MultiPowerSO-30 mechanical data (continued)

Symbol	Millimeters		
	Min	Typ	Max
E1	15.9	16	16.1
e		1	
F1	5.55		6.05
F2	4.6		5.1
F3	9.6		10.1
L	0.8		1.15
N			10deg
S	0deg		7deg

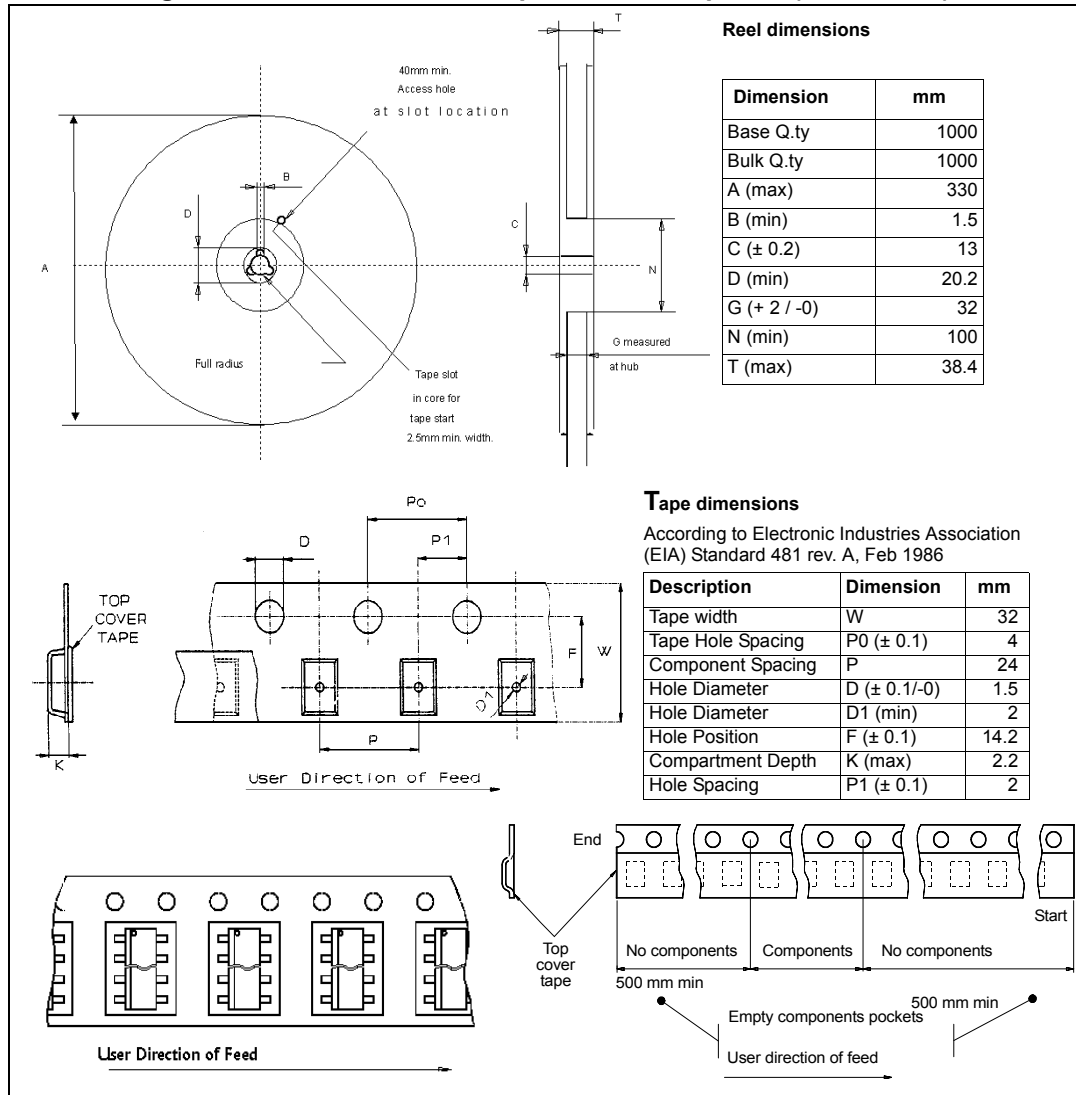
Figure 45. MultiPowerSO-30 suggested pad layout



5.2 Packing information

Note: The devices are packed in tape and reel shipments (see [Table 1: Device summary on page 1](#)).

Figure 46. MultiPowerSO-30 tape and reel shipment (suffix "TR")



6 Revision history

Table 18. Document revision history

Date	Revision	Description of changes
Sep-2004	1	First issue
Dec- 2004	2	Inserted $t_{off(min)}$ test condition modification and note Modified I_{RM} figure number
Feb-2005	3	Minor changes
Apr-2005	4	Public release
01-Sep-2006	5	Document converted into new ST corporate template. Added table of contents, list of tables and list of figures Removed figure number from package outline <i>on page 1</i> Changed <i>Features on page 1</i> to add ECOPACK [®] package Added <i>Section 1: Block diagram and pin description on page 5</i> Added <i>Section 2.2: Electrical characteristics on page 9</i> Added "low" and "high" to parameters for I_{INL} and I_{INH} in <i>Table 6 on page 9</i> Inserted note in <i>Figure 32 on page 22</i> Added vertical limitation line to left side arrow of $t_{D(off)}$ to <i>Figure 7 on page 14</i> Added <i>Section 4.1: PowerSSO-30 thermal data on page 27</i> Added <i>Section 5: Package and packing information on page 32</i> Added <i>Section 5.3: Packing information on page 34</i> Updated disclaimer (last page) to include a mention about the use of ST products in automotive applications
15-May-2007	6	Document reformatted and converted into new ST template.
06-Feb-2008	7	Corrected Heat Slug numbers in <i>Table 2: Pin definitions and functions</i> .
02-Oct-2008	8	Added new information in <i>Table 5: Power section</i> Added <i>Figure 33: Behavior in fault condition (How a fault can be cleared)</i>
20-Sep-2013	9	Updated Disclaimer.
11-Jan-2017	10	<ul style="list-style-type: none"> – Removed all information relative to tube packing of the product – Modified Section 5: Package information – Added AEC-Q100 qualified in the Features section – Minor text edits throughout the document

IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2017 STMicroelectronics – All rights reserved