Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Control de un robot de 3GDL mediante visual servoing

Autor: Richard M. Haes-Ellis

Tutor: Ignacio Alvarado Aldea

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Departamento de Teoría de la Señal y Comunicaciones

Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Control de un robot de 3GDL mediante visual servoing

Autor:

Richard M. Haes-Ellis

Tutor:

Ignacio Alvarado Aldea

Profesor Titular

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2019

Proyecto Fin de Carrera: Control de un robot de 3GDL mediante visual servoing

Autor: Richard M. Haes-Ellis
Tutor: Ignacio Alvarado Aldea

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

El diseño de una hoja de estilo en LaTeX para un texto no es en absoluto trivial. Por un lado hay que conocer bien los usos, costumbres y reglas que se emplean a la hora de establecer márgenes, tipos de letras, tamaños de las mismas, títulos, estilos de tablas, y un sinfín de otros aspectos. Por otro, la programación en LaTeX de esta hoja de estilo es muy tediosa, incluida la selección de los mejores paquetes para ello. La hoja de estilo adoptada por nuestra Escuela y utilizada en este texto es una versión de la que el profesor Payán realizó para un libro que desde hace tiempo viene escribiendo para su asignatura. Además, el prof. Payán ha participado de forma decisiva en la adaptación de dicha plantilla a los tres tipos de documentos que se han tenido en cuenta: libro, tesis y proyectos final de carrera, grado o máster. Y también en la redacción de este texto, que sirve de manual para la utilización de estos estilos. Por todo ello, y por hacerlo de forma totalmente desinteresada, la Escuela le está enormemente agradecida.

A esta hoja de estilos se le incluyó unos nuevos diseños de portada. El diseño gráfico de las portadas para proyectos fin de grado, carrera y máster, está basado en el que el prof. Fernando García García, de la Facultad de Bellas Artes de nuestra Universidad, hiciera para los libros, o tesis, de la sección de publicación de nuestra Escuela. Nuestra Escuela le agradece que pusiera su arte y su trabajo, de forma gratuita, a nuestra disposición.

Juan José Murillo Fuentes
Subdirección de Comunicaciones y Recursos Comunes

Sevilla, 2013

Resumen

En nuestra Escuela se producen un número considerable de documentos, tantos docentes como investigadores. Nuestros alumnos también contribuyen a esta producción a través de sus trabajos de fin de grado, máster y tesis. El objetivo de este material es facilitar la edición de todos estos documentos y a la vez fomentar nuestra imagen corporativa, facilitando la visibilidad y el reconocimiento de nuestro Centro.

Abstract

In our school there are a considerable number of documents, many teachers and researchers. Our students also contribute to this production through its work in order of degree, master's theses. The aim of this material is easier to edit these documents at the same time promote our corporate image, providing visibility and recognition of our Center.

... -translation by google-

Índice Abreviado

Índice

Notación

$\mathbb{R}$	Cuerpo de los números reales
$\mathbb{C}$	Cuerpo de los números complejos
$\|\mathbf{v}\|$	Norma del vector $\mathbf{v}$
$\langle \mathbf{v}, \mathbf{w} \rangle$	Producto escalar de los vectores $\mathbf{v}$ y $\mathbf{w}$
$\|\mathbf{A}\|$	Determinante de la matriz cuadrada $\mathbf{A}$
$\det(\mathbf{A})$	Determinante de la matriz (cuadrada) $\mathbf{A}$
$\mathbf{A}^\top$	Transpuesto de $\mathbf{A}$
$\mathbf{A}^{-1}$	Inversa de la matriz $\mathbf{A}$
$\mathbf{A}^\dagger$	Matriz pseudoinversa de la matriz $\mathbf{A}$
$\mathbf{A}^{\mathrm{H}}$	Transpuesto y conjugado de $\mathbf{A}$
$\mathbf{A}^*$	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
□	Fin de la solución
e.o.c.	En cualquier otro caso
e	número e
$\mathrm{e}^{\mathrm{j}x}$	Exponencial compleja
$\mathrm{e}^{\mathrm{j}2\pi x}$	Exponencial compleja con 2π
$\mathrm{e}^{-\mathrm{j}x}$	Exponencial compleja negativa
$\mathrm{e}^{-\mathrm{j}2\pi x}$	Exponencial compleja negativa con 2π
$\mathbb{R}\mathrm{e}$	Parte real
$\mathbb{I}\mathrm{m}$	Parte imaginaria
sen	Función seno
tg	Función tangente
arc tg	Función arco tangente
$\sin^y x$	Función seno de x elevado a y
$\cos^y x$	Función coseno de x elevado a y
Sa	Función sampling
sgn	Función signo
rect	Función rectángulo
Sinc	Función sinc
$\frac{\partial y}{\partial x}$	Derivada parcial de y respecto a x
x°	Notación de grado, x grados.
$\Pr(A)$	Probabilidad del suceso A
$E[X]$	Valor esperado de la variable aleatoria X
σ_X^2	Varianza de la variable aleatoria X
$\sim f_X(x)$	Distribuido siguiendo la función densidad de probabilidad $f_X(x)$
$\mathcal{N}(m_X, \sigma_X^2)$	Distribución gaussiana para la variable aleatoria X, de media m_X y varianza σ_X^2

$\mathbf{I}_n$	Matriz identidad de dimensión *n*
$\mathrm{diag}(\mathbf{x})$	Matriz diagonal a partir del vector **x**
$\mathrm{diag}(\mathbf{A})$	Vector diagonal de la matriz **A**
SNR	Signal-to-noise ratio
MSE	Minimum square error
$:$	Tal que
$\stackrel{\mathrm{def}}{=}$	Igual por definición
$\lVert\mathbf{x}\rVert$	Norma-2 del vector **x**
$\lvert\mathbf{A}\rvert$	Cardinal, número de elementos del conjunto **A**
$\mathbf{x}_i, i = 1,2,\ldots,n$	Elementos *i*, de 1 a *n*, del vector **x**
$\mathrm{d}x$	Diferencial de *x*
$\leqslant$	Menor o igual
$\geqslant$	Mayor o igual
$\backslash$	Backslash
$\Leftrightarrow$	Si y sólo si
$x = a+3 \underset{\substack{\uparrow \\ a=1}}{=} 4$	Igual con explicación
$\frac{a}{b}$	Fracción con estilo pequeño, a/b
Δ	Incremento
$b \cdot 10^a$	Formato científico
$\underset{x}{\rightarrow}$	Tiende, con x
O	Orden
™	Trade Mark
$\mathbb{E}[x]$	Esperanza matemática de x
$\mathbf{C}_\mathbf{x}$	Matriz de covarianza de **x**
$\mathbf{R}_\mathbf{x}$	Matriz de correlación de **x**
σ_x^2	Varianza de x

1 Introducción

The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point.

CLAUDE SHANNON, 1948

En este trabajo se presentará el desarrollo de un sistema de tracking de objetos/personas mediante el control de un robot de 3 grados de liberatad usando como sensor de realimentación una camara de visión. A dicho sistema tambien se le llama *"Visual Servoing"* o por su acrónimo VS.

El sistema se compone de un mecanismo motorizado de 2 grados libertad al que se le conoce como *pan-tilt* montado encima de un rail motorizado que permite el desplazamiento lateral. En este capítulo se le presentara las diversas técnicas de control de robots mediante visión que se usan en la robótica, el estado actual del mismo y las posibles aplicaciones que podrían tener.

1.1 Guía de uso de este TFG

1.1.1 A quién esta dirigido

Texto texto

1.1.2 Estrucutra de este documento

Texto texto

1.2 Objetivos

En este trabajo se presentará un proyecto que entra en diversos campos de la robótica tales como la mecánica, electrónica, programación, control y visión artificial. Nos centramos principalemtne en las tres primeras partes: la mecánica, la electrónica, la programación a bajo nivel y control del sistema. En cuanto a la visión usaremos algoritmos ya establecidos como herramienta para cerrar el bucle de control del sistema.

El objetivo sera controlar un mecanismo de 2 grados de libertad montado en una plataforma descizante para centrar la camara en alguna zona de interes indicada por el usuario. Esta zona se marcará mediante una interfaz gráfica mostrada en un PC en la que encuadra con un bounding-box la zona de interés, una vez selecionada se arma el sistema y se controlan los motores a tiempo real para centrar en todo momento la camara sobre la zona selecionada.

Se busca que el sistema se comporte de forma rápida y que sea capaz de hacer un seguimiento preciso de diferentes objetos.

1.3 Alcance y limites del proyecto

Texto texto

1.4 Introducion al visual servoing (VS)

Visual servoing o VS, es una técnica que usa información visual mediante un sensor de visión para controlar un robot.

1.4.1 Estado del arte del VS

La instrucción que sigue a la declaración de la clase es \usepackage{LibroETSI}. Con ella cargamos y definimos las principales características tipográficas y de muy diversa índole que hemos propuesto para el diseño de los documentos de la Escuela. A lo largo del presente documento se irán revelando diversos aspectos del mismo, pero se empieza aquí con una pequeña introducción. En el Capçitulo correspondiente se describen ordenadamente todas sus características.

Debemos observar antes que nada que es un fichero con la extensión sty y siempre debe estar antes del comando \begin{document}. En él se cargarán un conjunto de paquetes que hemos considerado necesarios y se definirán un conjunto de comandos que facilitan la escritura del texto. Una buena práctica para escribir un libro o cualquier documento que posea una extensión considerable es agrupar en un fichero como el presentado el conjunto de elementos que necesitamos para su escritura: paquetes y comandos.

1.4.2 Problema del VS

Como ya hemos dicho, el primer paquete importante (existen otros anteriores, pero de carácter mucho más técnico que otra cosa) es el paquete babel, que se carga en nuestro fichero mediante la instrucción

\usepackage [english, spanish, es-nosectiondot, es-noindentfirst, es-nolists, activeacute]{babel}.

Su papel fundamental es declarar que el texto estará escrito en español, que podemos utilizar sin restricción los acentos (no sería posible en LaTeX si no lo declarásemos como idioma preferente) y que adoptaremos los usos convencionales de mayúsculas, acentos en expresiones matemáticas, etc recomendados por la RAE. A todo esto contribuye también la sentencia \spanishdecimal{.}. Ya se ha comentado que intercambiando las palabras english por spanish obtenemos los nombres de capítulo, sección y otros en inglés.

1.4.3 Aplicaciones del VS

Aunque LaTeX no es sólo un sistema de edición para textos científicos, su aplicación para ellos es prácticamente universal. En el estilo de libro que hemos propuesto, la utilización de las fuentes en los textos matemáticos y el posible uso de diversos símbolos y herramientas propias para los textos científicos está recogido en diversos paquetes, entre los que cabe destacar \usepackage[cmex10]{amsmath}, \usepackage{amssymb} y \usepackage{mathptmx}.

2 Esquema general y análisis del sistema

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

En este capítulo trataremos de describir los diferentes componentes del sistema completo, para ello hemos creado un esquema general del objetivo y hemos subdivido el sistema en varias partes para analizar de forma independiente.

2.1 Esquema general

En la Figura 2.1 se muestra un esquema general del sistema a analizar:

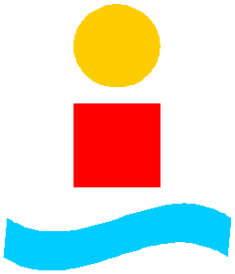

Figura 2.1 Logo de la ETSI.

Si nos detenemos en los comandos que hemos utilizado, con `width` se controla el ancho, y se escala así el tamaño de la imagen. En LaTeXexisten diversas opciones para situar la figura en la página: con `t` o `b` se le indica que las incluya arriba o abajo (top/bottom) y con `!` se le pide que la deje dónde está, tras el texto anterior.

2.2 Mecanismo pan-tilt

2.2.1 Cinematica

Si nos detenemos en los comandos que hemos utilizado, con `width` se controla el ancho, y se escala así el tamaño de la imagen. En LaTeXexisten diversas opciones para situar la figura en la página: con `t` o `b` se le indica que las incluya arriba o abajo (top/bottom) y con `!` se le pide que la deje dónde está, tras el texto anterior.

2.3 Cámara

Si nos detenemos en los comandos que hemos utilizado, con `width` se controla el ancho, y se escala así el tamaño de la imagen. En LaTeXexisten diversas opciones para situar la figura en la página: con `t` o `b` se le indica que las incluya arriba o abajo (top/bottom) y con `!` se le pide que la deje dónde está, tras el texto anterior.

2.4 PC

Si nos detenemos en los comandos que hemos utilizado, con `width` se controla el ancho, y se escala así el tamaño de la imagen. En LaTeXexisten diversas opciones para situar la figura en la página: con `t` o `b` se le indica que las incluya arriba o abajo (top/bottom) y con `!` se le pide que la deje dónde está, tras el texto anterior.

Si no desea que se numere una ecuación puede poner asterisco, tanto en el entorno `equation` como `align`.

3 Mecánica

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

El *formato de capítulo* abarca diversos factores. Un capítulo puede incluir, además de texto, los siguientes elementos:

3.1 Diseño CAD

3.1.1 simulacion de cinematica

3.2 Motores

3.2.1 sincronos

3.2.2 asincronos

3.3 Sistemas de transmision

3.3.1 Engranajes

3.3.2 Levas

3.3.3 Poleas

3.3.4 Poleas dentadas

3.4 Diseño de mecanismos de 2 grados de libertad

3.4.1 Tipos de articulaciones

3.4.2 pan-tilt

3.5 Diseño de ejes

3.5.1 rodamientos

3.6 rail

3.7 Manufactura de piezas

3.7.1 Manufactura sustractiva

3.7.2 Manufactura aditiva

3.8 Impresion 3D

3.8.1 Principio de funcionamiento

3.8.2 Código G

3.8.3 Preparacion de piezas (Slicer)

3.9 Ensamblaje

4 Electrónica

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

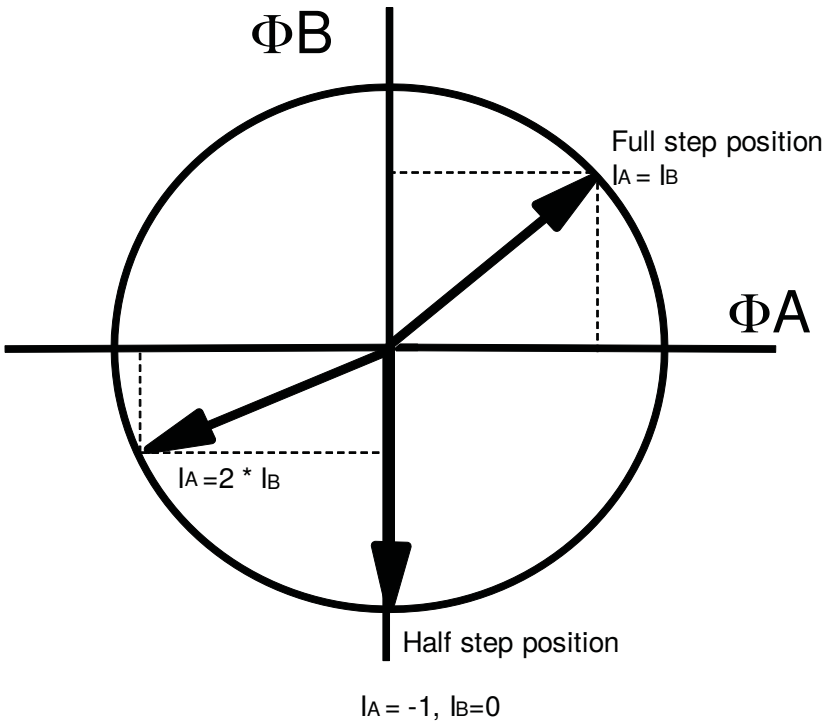

Figura 4.1 Ilustraccion microstepping.

En esta sección analizaremos los difrentes componentes electrónicos y selececionaremos aquies que resutan mas adecuados para este proyecto. Empezaremos detallando la parte electro-mecánica para accionar los ejes, esta la haremos mediante motores, veremos como se interactua con el sistema mediante una pantalla tátctil, veremos que red de sensores usaremos para detectar el estado del sistema, selececionaremos los drivers para alimentar y controlar de forma adecuada los motores, escojeremos un microcontrolador que sera el que controle y monotorize el sistema electronico completo y por ultimo veremos como alimetar el sistema.

4.1 Motor

Para el accionamiento de los ejes del mecanismo *"pan-tilt"* y el rail usaremos un motores de corriente continua. En concreto usaremos unos motores llamados motores *"paso a paso"*, tambien conocido como motor de pasos. Estos dispositivos son motores sin escobillas que tienen dividido el giro completo del rotor en un numero determinado de pasos, de ahi el nombre.

Hay tres tipos de motores paso a paso:

- **El motor de pasos de reluctancia variable (VR):** Este motor está compuesto por un estator devanado que suele estar laminado y un rotor no magnetico multipolar formado por dientes de hierro.

  Cuando el estator se alimenta se induce un campo magnetco en el rotor y genera par par en la direccion que minimiza la reluctancia magnetica, por tanto atrae al polo mas cercano del rotor y gira. La respuesta de este motor es muy rápida, pero la inercia permitida en la carga es pequeña. Cuando el estator no se alimenta, el par estático es cero.

- **El motor de pasos de rotor de imán permanente:** Este tipo de motores estan compuestos de un rotor que esta axialmente magnetizado, es decir, tiene polos que alternan entre norte y sur paralelamente al eje. Su estator esta compuesto por dos bobinados contenidos en unos entrehierros con dientes que interactuan con el roto.

  Este tipo de motores son capaces de aplicar un par estatico al rotor, pero operan a bajas velocidades.

- **El motor de pasos híbrido:** La combinación de ambos motores de antes dan lugar a los motores de pasos híbrido. Están compuestos de un rotor con dos partes magnetizadas y de polos opuestos con dientes que estan desfasados entre si. El estator esta compuesto por bobinados y tambien estan formados por unos dientes. Los dientes del rotor ayudan a alienar el flujo magnetico a traves de los dientes donde avanzan a las posiciones mas favorables. Esto permite mejorar el par estatico, dinamico y el par de *"detent"* que es el par del motor sin alimentar.

  Además del par, es capaz de obtener mejor resolución por paso en comparacion con los otros dos, llegando a tener una precision de hasta 1600 pasos por revolucion con técnicas de conmutacion llamado *"micro-stepping"*.

Para nuestro caso usaremso el motor de pasos hibrído por las ventajas de precision y par que tiene frente a los otros. En concreto usaremos los *"NEMA 17"*, el nombre *"NEMA"* viene del estandar NEMA ICS 16-2001 especificada por la asociacion NEMA, y 17 viene del tamaño del frontal del motor siendo de 1.7" x 1.7".

4.1.1 Funcionamiento

Este motor cuenta con 4 cables, 2 para cada fase tal y como se ve en la figura Figura 4.2

Para girar el rotor tenemos que alimentar cada fase en una secuencia dada, en la figura vemos una secuencia *"full-step"*, esto quiere decir que alimentamos una fase por cada paso intermedio. Esto provoca que el rotor se alinee con las fases de paso a paso rotando 90 grados entre ellos.

Además podemos alimentar primero un bobinado, luego el otro manteniendo el priemero para obtener un nuevo paso intermedio. esto se ve ilustrado en la figura BLABLA. Con esto hemos conseguido duplicar el numero de pasos por revolucion del motor y esta tecnica es lo que llamamos *"half-step"*.¿Pero que pasa si queremos mas resolución?

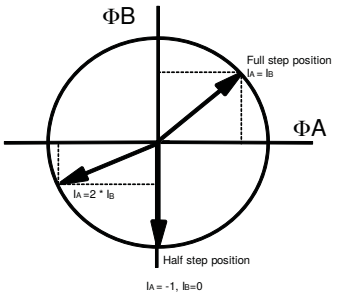

Figura 4.2 Ilustraccion microstepping.

Para obtener mas precision usamos una tecnica llamado *"micro-stepping"*, esto no es mas que una extension de lo anterior. Miremos la ilustracion de la figura Figura 4.2 , podemos ver que la alimentacion de los bobinados el vector magnetico resultant que orienta el rotor forma 45 grados con la horizontal. Pues bien si se alimentara el bobinado B con el doble de corriente que el bobinado A, conseguiremos un vector magnetico que formara un angulo de 26.56 grados con la horizontal. Para cualquier posicion angular, las corrientes necesarias estaran definidas por el seno y coseno del angulo requerido.

Cabe notar que si alimetnamos dos bobinados por igual a su maxima corriente obtendremos un vector magnetico mas grande que si solo alimentamos un bobinado, esto se be bien reflejado en el primer diagrama de la figura Figura 4.3 en los anglos que estan localizados en medio de cada bobinado.

Este fenomeno provoca vibraciones, ya que el par no es constante mientras rota el motor, para evitar este problema, lo que se suele hacer es limitar la corriente de los bobinados para que, al combinarse formen un vector de igual magnitud en todos el rango de operancion del motor. De esta forma conseguiremos un par constante mientras este funcionando y evitara vibraciones inesperadas. Esto se ve claro en el segundo diagrama de la figura Figura 4.3.

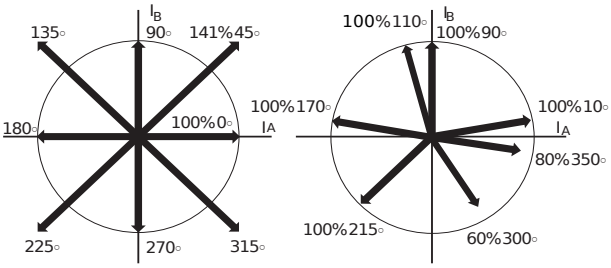

Figura 4.3 Ilustraccion vector magnetico resultante.

4.1.2 Driver

Como hemos visto antes para hacer girar el rotor del motor de pasos hibrido, tenemos que alimentar los bobinados para formar un vector magnetico que oriente el rotor como queramos, para ello vamos a necesitar un driver que se encargara de convertir las señales de control en señales de potencia. Estos drivers suelen estar formados por una red de transistores dispuestos en una configuracion H. En la figura Figura 4.4 vemos el circuito de un puente-H para controlar un bobinado. En nuestro caso necesitaremos un driver que contega dos de estos puentes.

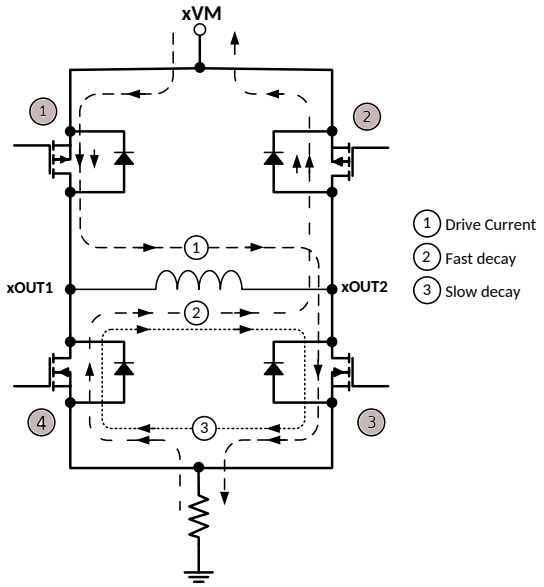

Figura 4.4 Circuito puente H.

Para alimentar una bobina tan solo tenemos que abrir o cerrar un par de transistores para que fluya corriente

a través de el en una direccion u otra. Si queremos alimentar la bonina con una tension positiva tan solo tendremos que abrir los transistores 1,3 y manter cerrado los transistores 2,4, de esta forma la corriente fluira desde *xOUT1* hasta *xOUT2*. Para alimentarlo al contrario simplemente cerramos 1,3 y abrimos 2,4. Con esto podemos ya controlar la direccion de la corriente, pero ¿Y la magnitud de corriente?, para esto usamos PWM, con esto podemos modular la cantidad de corriente que pasa por la bobina.

En la figura Figura 4.5 se ilustra las corrientes en cada fase del motor, donde se ve claramente las senoides de corrientes que habramos anteriormente. Cabe notar el escalonado de dicha corriente, bien pues esto es debido a la cuantizacion de la señal PWM.

Las señales PMW se modulan con temporizadores que cuentan hasta un cierto valor de un registro de comparacion (CCRX) y cambian la señar digital de estado, pasando de 1 a 0, cuando el timer llega hasta su valor maximo determinado (ARR) se resetea la señar PWM pasado de 0 a 1. Los contadores de los micros toman valores enteros, por tanto la señar PWM generada tambien tendra anchura de pulso o *"duty-cycle"* discreto, esto finalmente se traduce a saltos discretos en la corriente efectiva que pasa por los bobinados y en consecuencia efecta directamente la resolución del motor.

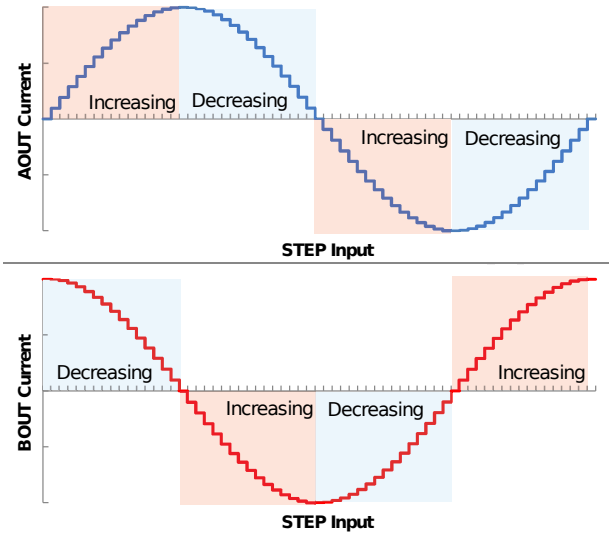

Figura 4.5 Digrama de corrientes.

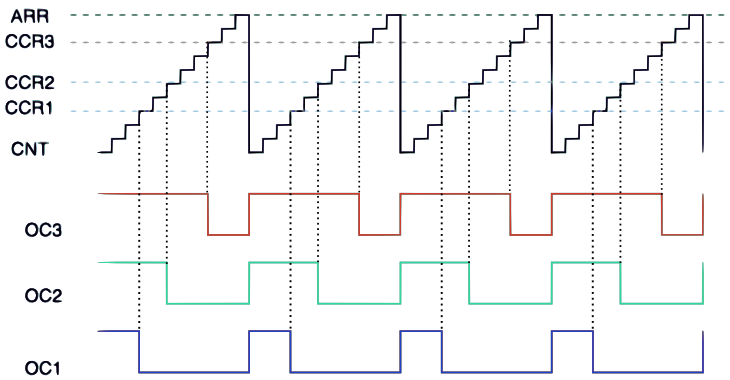

Figura 4.6 Digarama temporal de señales PWM.

El driver que usaremos sera uno de *Pololu*, el modelo A4988. Este driver es usado ampliamente en impresoras 3D por su bajo coste y alta fiabilidad. Cabe mensionar otros dirvers como el *TMC22XX* de *Trinamic* que proporcionan otras funcionalidades tales como suavizado de corrientes para minimizar el ruido, deteccion de bloqueo etc.. Para nuestro caso no necesitaremos estas funcionalidades.

Este driver es el que implementa el control de los motores, de esta forma solo tenemos que mandarle comandos básicos para controlar el motor. Fijandonos en el esquematico y guia de cableado de las figuras

Figura 4.7 Driver pololu A4988.

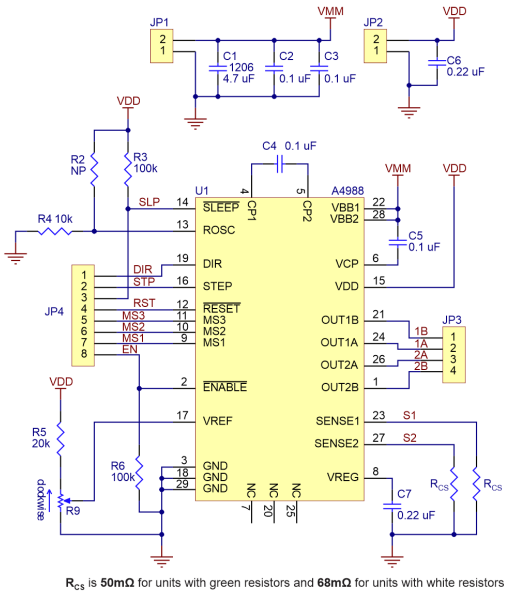

Figura 4.8 Esquematico del driver A4988.

Figura 4.8 y Figura 4.9 del producto, se especifican 3 señales de control:

- Señal STEP: Esta sera la señal que marcara un paso del motor cuando pasemos de nivel bajo a nivel alto, dependindo como este configurado los *micro-steps* girará una cantidad de grados.
- Señal DIR: Esta señal marcara la dirección del motor.
- Señal ENABLE: Activo a nivel bajo, sera la que habilita o deshabilita las salidas del driver hacia el motor. Es importante ya que si no estamos cargando nada en el eje del motor conviene deshabilitarlo para no circular corriente.

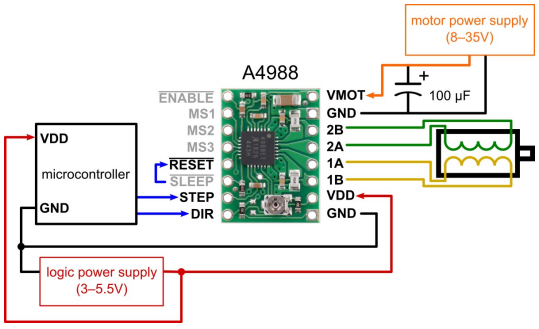

Figura 4.9 Guia de cableado.

Además contamos con unos pines de configuración que nos permitira determinar el numero de pasos por vuelta, dicha configuracion se muestra en la Tabla 4.1. Los motores de pasos hibridos suelen tener 200 dientes por tanto si tenemos una configuracion de MS1=1, MS2=1 y MS3=1, tendremos un total de 3200 pasos por vuelta.

Tabla 4.1 Tabla de configuracion *micro-stepping*.

MS1	MS2	MS2	pasos por diente
0	0	0	1
1	0	0	1/2
0	1	0	1/4
1	1	0	1/8
1	1	1	1/16

4.2 Pantalla tactil

4.2.1 Tipos

4.2.2 Nextion

4.3 Sensores

4.3.1 Sensores fin de carrera

Para poder determinar y calibrar las posiciones de lo motores se precisa un sensor que nos indique en que estado o posicion esta. Para conocer este estado hay varias formas de hacerlo, empezando con los interruptores mecanicos llamados sensores fin de carrera. Estos sesores son muy sencillos ya que solo hace falta colocar el sensor en una posicion conocida y que el eje movil contacte con el interruptor para activar el sensor, de esta forma se puede guardar ese instante como la posicion zero.

Figura 4.10 Sensor montado en eje AS504X.

4.3.2 Encoders magneticos

Para los ejes del mecanismo *"pan-tilt"* no podemos usar ese tipo de sensor ya que son ejes que tienen rotacion continua sin limites, por tanto no podemos colocar unos topes mecanicos con el sensor acoplado para activar el sensor. Por ello hemos recurrido a otro tipo de sensores usados ampliamente en la róbotica, los sensores

magneticos de effecto hall. En la figura Figura 4.11 se muestra un diagrama del sensor elegido con una ilustracion del su funcionamiento.

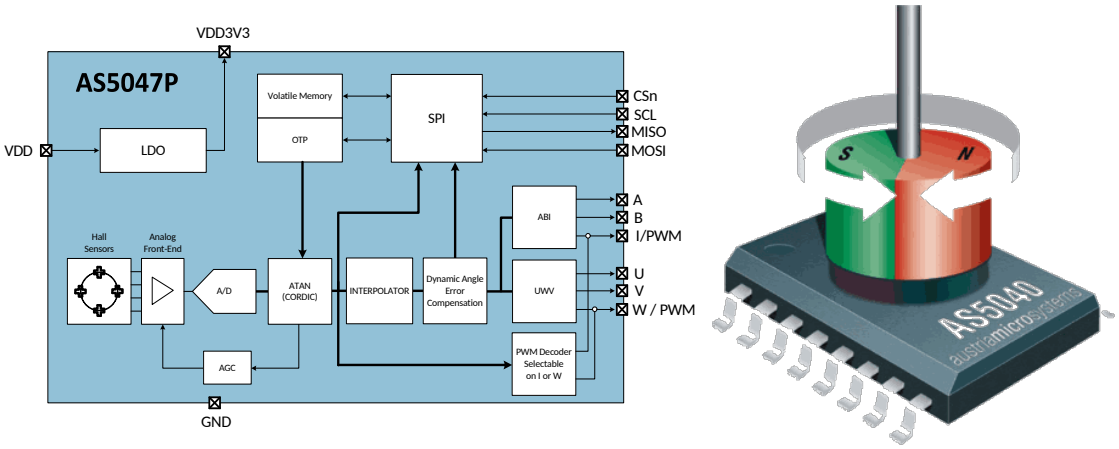

Figura 4.11 Sensor montado en eje AS504X.

El sensor magnetico usado es uno que se monta en el propio eje del motor. Un imán que está magnetizado axialemtne se coloca en el rotor del motor o eje movil cuyo angulo es el que se quiere medir y se coloca el sensor concentricamente al eje debajo del imán. Este sensor esta dispuesto de 4 sensores hall combinadas mediante un puente de wheatstone. Con circuitería que se encarga del acondicionamiento de las señales del puente, además nos proporciona una interfaz digital por el cual podemos hacer la lectura del ángulo.

Estos tipos de sensores suelen ser mas caros, por ello y con motivo de aprender la gran diciplina de diseño de PCBs, se diseño una placa de adaptación para el sensor. El diseño se ha realizado mediante EasyEDA. El resultado del diseño y su Funcionamiento esta reflejado en la figura.

[FOTON DEL SENSOR AS4050 Y CON GRAFICAS MOSTRANDO QUE FUNCIONA]

4.4 Microcontrolador

Para este proyecto necesitaremos un micro capz de manejar todos los componentes electrónicos que hemos mencionado antes. Para ello hemos listado los requisitos que debe de cumplir dicho microcontrolador.

- Mínimo 3 temporizadores para manejar las señales de los motores.
- Mínimo 2 puertos serial para comunicarnos con la pantalla y con el PC
- Varios GPIO para los sensores fin de carrera.
- Programación de interrupciones para el manejo de temporizadores.
- Velocidad moderada capaz de lanzar el tren de impulsos necesarios para manejar los motores a una velocidad aceptable.

EL microcontrolador elegido es el *Arduino DUE* dado que cumple todos los requisitos necesarios para nuestro proyecto. Mirando el datasheet estas son las carácteristicas mas importantes del micro:

- Voltaje de operanción: 3.3V
- Voltaje de entrada admitida: 7-12V
- GPIOs: 54 (12 de ellos PWM)
- Entradas analógicas: 12
- CPU: ARM Cortex-M3 revision 2.0 a 84 MHz
- SRAM: 96KB
- Memoria flash: 512KB
- USB 2.0 Device/Mini Host

Figura 4.12 Arduino DUE.

- 4 USARTs y un UART

- 9 canales de 32-bit con Timer Counter (TC) para *capture, compare*

4.5 Placa de adaptación CNC

Para facilitar el interconexionado de todos los componetnes electronicos se ha optado por usar una placa que esta destinada para el control de máquinas de control numérico (CNC) ya que cuentan con los condesadores de desacoplo y zocalos y pines de configuración disponibles para los drivers de motores de pasos junto con entradas y salidas para todo tipo de sensores o actuadores. En la figura Figura 4.13 se ve una imagen de dicha placa de adaptación, esta simplemente se coloca encima del microcontrolador y garantiza un conexionado fijo.

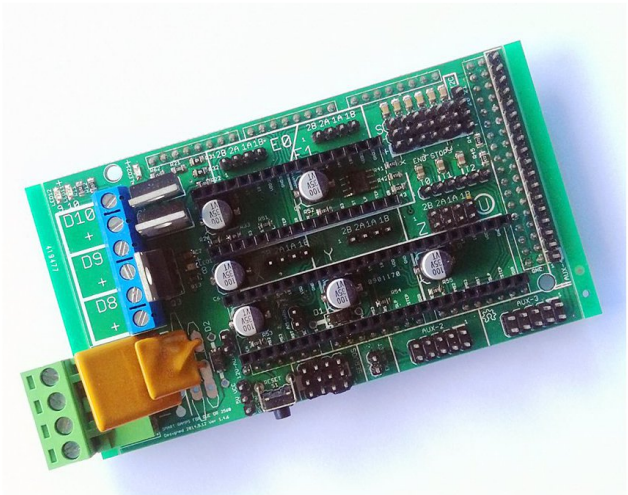

Figura 4.13 Placa de adaptación.

4.6 Fuente de alimentacion

4.6.1 Regulable

4.6.2 Bateria

https://www.elprocus.com/stepper-motor-types-advantages-applications/
https://www.nema.org/Standards/SecureDocuments/ICS16.pdf
Instrument Engineers'Handbook, Vol. 2: Process Control and Optimization, 4th Edition. Ed. Liptak, Bela G, N.p.: CRC Press. Print
https://www.arduino.cc/
https://www.pololu.com/product/2980

5 Comunicación micro-pc

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

El *formato de capítulo* abarca diversos factores. Un capítulo puede incluir, además de texto, los siguientes elementos:

5.1 Introdución

Para poder controlar los motores desde un ordenador hace falta comunicarlos por aguno de los perifericos disponibles y compatibles entre el micro y el PC. Hay varias formas de comunicación entre dispositivos, estos se pueden clasificar en comunicación paralela o serie.

En una comunicación paralela todos los bits del dato que se quiere transmitir son enviados a la vez, esto es posible ya existen varias pistas de datos o cables entre el transmisor y receptor para efectuar la transimsion de datos. Esta forma de comunicación es más rápida y tambien la mas cara ya que se requiere mas cableado y dispositivos. Las aplicaciones mas comunes de este tipo de comunicación se ven en placas base para intercomunicar la RAM con la CPU, o puertos PCI, impresoras antiguas, etc..

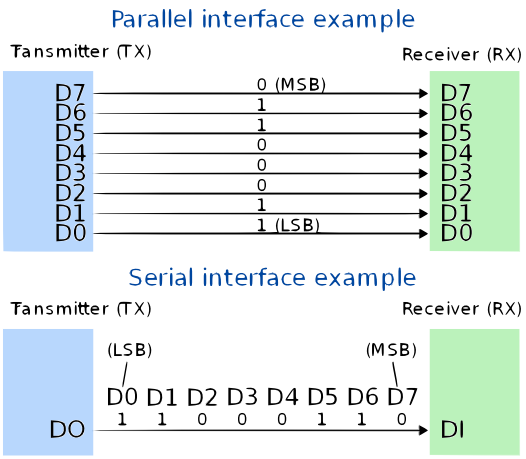

Figura 5.1 Serie vs Paralelo.

La comunicación serie sin embargo envia datos secuencialmente por una sola via o cable, empujado los bits uno a uno hasta llegar a mandar el mensaje completo. Como este tipo de comunicación solo requiere el uso de 1 o 2 cables, el hardware necesario para su funcionamiento es mas barato. La unica medida limitante de este tipo de comunicación es la velocidad, ya que el dato se manda bit a bit secuencialmente. Hoy en dia

las velocidades de comunicación serie ya son lo suficientemente altos para no tener que preocuparnos por latencias **en algunos casos**.

Con el avance en la tecnología de dispositivos de comunicación serial, estos se estan abarantando, haciendo mas pequeño y más rapidos que nunca y poco a poco se esta sustituyendo los puertos paraelos por puertos serie. Esta será la comunicación que usaremos nosotros, en concreto la comunicación ***UART***

5.2 Comunicacion UART

UART o también conocido por sus siglas en ingés: ***U**niversal **A**synchronous **R**eceiver **T**ransmitter* es un dispositivo de comunicación que es capaz de convertir datos que vienen en paralelo y pasarlo a serie en lado del transmisor y tambien en viceversa para el lado del receptor. Se le dice que es universal por que se pueden configurar varios parametros de la transimsion.

Se trata de un dispositivo que juega el papel de puente entre dispositivos que tienen otras formas de comunicación tales como USB, RS-232, etc.. De este modo podemos conectar nuestro PC al microcontrolador via USB y comunicarlos con UART.

Como se indica en el nombre, la comunicación es asíncrona, no hay linea de reloj para sincronizar los datos. Cabe preguntarnos entonces como se sabe a que velocidad se comunican diferentes dispositivos. Bien pues abos dispositivos de transmision y recepcion deben estar de acuerdo en los parametros de temporizacion de los datos. Además el UART posee bits especiales para sincronizar ambos dispositivos. Estos bits se llaman *Start bit* y *Stop bit*. Estos bits se añaden al paquete al principio y al final respectivamente.

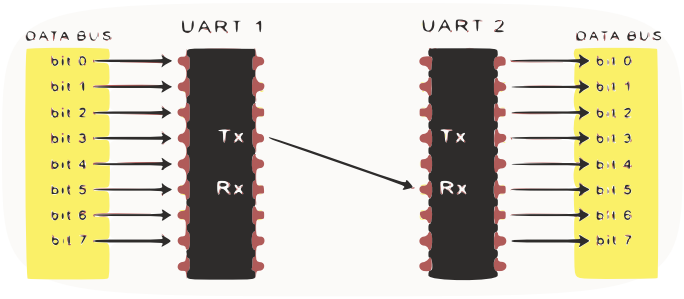

Figura 5.2 ilustración de comunicación serie.

Fijandonos en la figura Figura 5.2 vemos una ilustración de UART. El puerto UART recive los datos del dispositivo transmisor y este convierte los datos de paralelo a serie con un registro de desplazamiento donde se transmite bit a bit por la linea TX. El puerto UART del receptor recibe el dato bit bit por la linea RX y lo convierte de serie a paralelo para que el dispositivo receptor pueda leerlo.

Para que la comunicación se establezca es necesario configurar ambos dispositivos para que puedan transimitir los mismos bits de paridad y recivir/transmitir a la misma velocidad.

5.2.1 Prueba

Usando el microcontrolador, si lo programamos para que envie por puerto serie la frase *"ETSI* y conectamos un analizador lógico en el el pin del puerto serie nos encontraremos con un diagrama que se muestra en la figura Figura 5.3

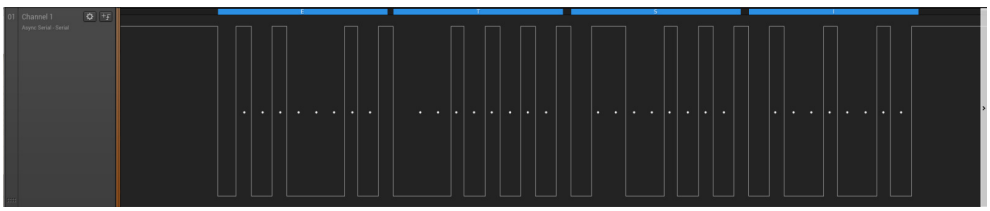

Figura 5.3 Análisis de una señal TX del microcontrolador DUE.

Si aplicamos un decodificador ASCII a esta señal conseguios leer el mensaje que hemos enviado.

5.3 Protocolo a nivel de aplicacion

Como hemos visto antes podemos comunicar el PC con el micro mediante el puerto serie, por tanto podemos ya comuncar información de la velocidad de los motores para controlar la posición del robot. Pero hay que tener en cuenta que necesitamos mandar minimo tres variables de tipo entero que definiran la velocidades de los tres motores. Si enviamos los tres valores seguidos puede ocurrir que leamos el dato de forma erronea, sea por que leemos un bit de mas y ya se nos corrupta la informacion. Por tanto necesitamos algun protocolo a nivel de aplicación que permita transmitir datos y que asegure que se este leyendo el paquete en orden y sin errores. Este protocolo a nivel de aplicación lo realizaremos con una maquina de estados.

5.3.1 Protocolo binario mediante maquina de estados

En la figura Figura 5.4 se muestra un diagrama de la máquina de estados. La máquina consta de 5 estados:

- Estado *Idle* o de reposo: En este estado el programa revisa si el registro de RX esta vacío o no. Si resutla no estar vacío quiere decir que nos esta llegando informacion desde el puerto serie.
- Estado inicio: En este estado se lee 1 solo byte del registro RX y se comprueba que es igual al byte de comienzo, en este caso igual al byte *0x7F* si resutla que no es igual, abortamos la lectura del paquete y empezamos de nuevo. Si resutla ser correcto saltamos al estado de lectura del payload.
- Estado leer *payload*: En este estado se leen los proxmos 12 bytes del registro RX y nos pasamos al siguente estado.
- Estado leer final: En este estado se leen 4 bytes que conjuntamente forman un entero de 32 bits y se comprueba que éste tenga el valor de *0x7FFFFFFF* asi nos aseguramos que hemos leido bien el paquete y podemos poner el flag de *Nuevo mensaje* a *True*, en caso contrario significará que ha habido un problema en la lectura y desechamos el paquete.

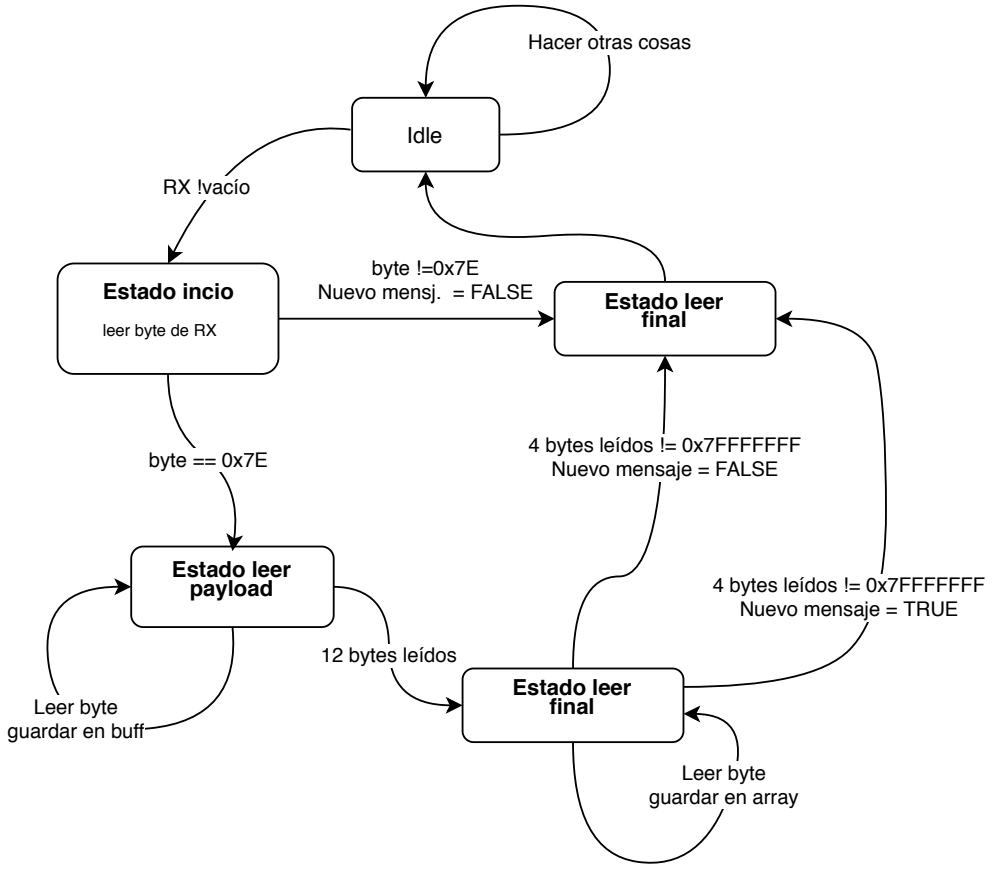

Figura 5.4 Máquina de estados del protocolo de comunicación.

Código 5.1 Código manejador de recepción de comandos.

```
    // If there is data in our RX buffer
  if (Serial.available()){

    // We initialize some control variables
    int16_t byten = 11;        // Index of our input buffer, we start a 11 because of big-endian (I think)
    uint8_t state = LISTENING; // Start off on a listening state
    uint8_t new_packet = 0;    // Flag variable that indicates a valid new data packet
    do{
      // We wait for data to come in
      while (Serial.available() == 0); // Wait for incomming data
      // Then we read a byte of that RX buffer
      uint8_t in_byte = Serial.read();

      // STATE MACHINE //
      switch (state)
      {
      case LISTENING: // Initially we are listenting for a start byte indicated by the 0x78 byte
        if (in_byte == 0x7E){ // CHECK IF ITS START BYTE
          // If the start byte is read then we procceed to read the payload
          state = READ_LOAD;
        }else{
          // Wrong start command
          // If we started listenting and the first byte wasnt a start byte then we restart the listening, the packen has begun without the start packet
          state = END_CMD; // RESET IF IT ISN'T
        }
        break;

      case READ_LOAD: // Reading payload state, payload has a fixed length of 12 anything else will break the packet and restart the state machine
        // Put data in our structure data
        m_payload.array[byten] = in_byte;
        byten--;
        if (byten < 0){
          // When we have read 12 bytes total we go to the end state
          state = READ_END;
          byten = 3;
        }
        break;

      case READ_END: // This state must read a end command composed by 4 bytes
        m_integer.array[byten] = in_byte;
        byten--;
        if (byten < 0){
          // If the end command maches the predifined end command then we can say that the packet is good.
          if (m_integer.number == (int32_t)0x7FFFFFFF){ // END COMMAND
            // Data is correct
            new_packet = 1;
            state = END_CMD;
          }else{
            state = END_CMD;
          }
```

```
51          }
52          break;
53        }
54      } while (state != END_CMD);
```

https://www.electronicshub.org/basics-uart-communication/

6 Programacion embebida

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

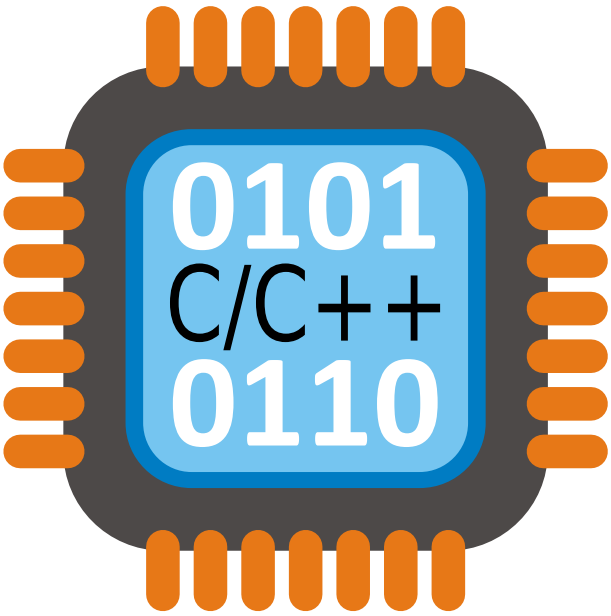

Figura 6.1

6.1 Sistema embebido

Para describir como vamos a programar nuestro microcontrolador primero vamos a hacer una distincion entre un sistema embebido y un PC. A diferencia de un PC que tiene como proposito realizar tareas muy general, un sistema embebido es un sistema de computación diseñado exclusivamente para realizar una tarea muy específica. En nuestro caso esa tarea será la de controlar los motores dandoles las señales adecuadas a los driver de los mismos.

Estas tareas se suelen programar directamente en la memoria del sistema mediantes lenguajes de programación de bajo nivel tales como el lenguaje ensamblador o mediante un compilador especifíco a dicho sistema con C/C++. sin tener un sistema operativo por encima regulando y gestionando los recursos para dicho programa. Por tanto tenemos mucha mas libertad para controlar de forma muy específica tanto las entradas y salidas como los periféricos de comunicación, esto es vital para obtener una respuesta rápida de control sobre los motores sin que ningun otro proceso interrumpa su función.

Para hacernos una mejor idea de como programar el microcontrolador, veremos primero como interactuar con los drivers de los motores.

Como hemos visto en un capitulo anterior los motores de pasos funcionan alimentado los bobinados del estator, esto lo maneja el drive mediante MOSFETs de potencia. A su vez estos se controlan internamiente por el driver en función de las señales de entrada del drivel.

La interfaz que nos proporciona se le llama *Step-direction*. Básicamente tenemos 3 señales para control el motor:

- La señal *Enable* que habilita o deshabilita el motor, esto permite controlar si circula corriente por los bobinados o no.
- La señal de *Step*, cuando esta señal afectua un flanco de subida el driver actua sobre los MOSFETs para hacer girar el rotor por un paso. De esta forma si alimentamos una señal cuadrada a una frecuencia de 1Khz a esta entrada estaremos girando el rotor a una velocidad de 1000 pasos por segundo.
- Y por último la señal *Dir*, esta señal determina en que sentido gira el rotor. En la figura **??** vemos una ilustración de dichas señales.

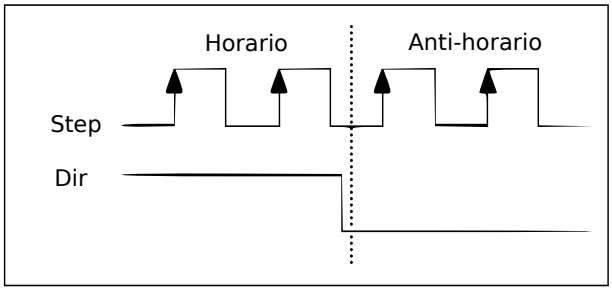

Figura 6.2

Por tanto vemos que la velocidad de los motores lo determinamos mediante la frecuancia de una señal cuadrada. Además tenemos que generar 3 de estas señales y asegurarnos que estemos mandado los pulsos en el timepo adecuado. Realizar esto mediante GPIO activando y desactivado las señales es muy costoso, además tendremos al microcontrolador cargando con solo esta tarea, por tanto tenemos que buscar otra solución.

6.2 Interrupciones y Timers

Para gestionar el tren de pulsos para los drivers usaremos contadores con interrupciones, usaremos las interrupciones para ejecutar un manejador que cambiar la señal de nivel bajo a nivel alto produciendo el flanco de subida necesario para que el driver actue un paso. Además aprovecharemos esta interrupcion para actualizar el registro de comparacion con una variable que nos llega del puerto serial, es decir, actualizaremos el valor de la cuenta que hace que se dispare la interrupción. De esta forma conseguimos variar la frecuancia de la señal de forma sincrona.

A raíz de lo anterior, esta claro que necesitaremos 3 contadores

6.3 Experimentos

7 Percepción

Una de las virtudes del ingeniero es la eficiencia.

Guang Tse

El *formato de capítulo* abarca diversos factores. Un capítulo puede incluir, además de texto, los siguientes elementos:

7.1 Cámara

7.2 Técnicas de tracking en Percepcion

7.3 Eleccion de algoritmos

7.4 OpenCV

7.5 Experimentos

8 Capítulo - Control mediante PID

Una de las virtudes del ingeniero es la eficiencia.

Guang Tse

El *formato de capítulo* abarca diversos factores. Un capítulo puede incluir, además de texto, los siguientes elementos:

8.1 Teoria de PID

8.2 Implementacion de un PID en Codigo

8.3 Python

8.4 Expermimentos

Apéndice A
Sobre LaTeX

Este es un ejemplo de apéndices, el texto es únicamente relleno, para que el lector pueda observar cómo se utiliza

A.1 Ventajas de LaTeX

El gusto por el LaTeX depende de la forma de trabajar de cada uno. La principal virtud es la facilidad de formatear cualquier texto y la robustez. Incluir títulos, referencias es inmediato. Las ecuaciones quedan estupendamente, como puede verse en (A.1)

$$x_1 = x_2. \tag{A.1}$$

A.2 Inconvenientes

El principal inconveniente de LaTeX radica en la necesidad de aprender un conjunto de comandos para generar los elementos que queremos. Cuando se está acostumbrado a un entorno "como lo escribo se obtiene", a veces resulta difícil dar el salto a "ver" que es lo que se va a obtener con un determinado comando.

Por otro lado, en general será muy complicado cambiar el formato para desviarnos de la idea original de sus creadores. No es imposible, pero sí muy difícil. Por ejemplo, con la sentencia siguiente:

Código A.1 Escritura de una ecuación.

```
1 \begin{equation}\LABEQ{Ap2}
2 x_{1}=x_{2}
3 \end{equation}
```

obtenemos:

$$x_1 = x_2 \tag{A.2}$$

Esto será siempre así. Aunque, tal vez, esto podría ser una ventaja y no un incoonveniente.

Para una discusión similar sobre el Word®, ver Apéndice B.

Apéndice B
Sobre Microsoft Word®

B.1 Ventajas del Word®

La ventaja mayor del Word® es que permite configurar el formato muy fácilmente. Para las ecuaciones,

$$x_1 = x_2, \tag{B.1}$$

tradicionalmente ha proporcionado pésima presentación. Sin embargo, el software adicional Mathtype® solventó este problema, incluyendo una apariencia muy profesional y cuidada. Incluso permitía utilizar un estilo similar al LaTeX. Además, aunque el Word® incluye sus propios atajos para escribir ecuaciones, Mathtype® admite también escritura LaTeX. En las últimas versiones de Word®, sin embargo, el formato de ecuaciones está muy cuidado, con un aspecto similar al de LaTeX.

B.2 Inconvenientes de Word®

Trabajar con títulos, referencias cruzadas e índices es un engorro, por no decir nada sobre la creación de una tabla de contenidos. Resulta muy frecuente que alguna referencia quede pérdida o huérfana y aparezca un mensaje en negrita indicando que no se encuentra.

Los estilos permiten trabajar bien definiendo la apariencia, pero también puede desembocar en un descontrolado incremento de los mismos. Además, es muy probable que Word® se quede colgado, sobre todo al trabajar con copiar y pegar de otros textos y cuando se utilizan ficheros de gran extensión, como es el caso de un libro.

```
#include <Arduino.h>

// #define debug_com
#define debug_control_msg
// #define debug_motor_data

#define T1_FREQ 656250
#define MIN_DELAY 200
#define TIMEOUT 500

#define PIN_2_DIR 55     // PORTA Bit 24
#define PIN_1_ENABLE 56  // PORTA Bit 23
#define PIN_2_STEP 54    // PORTA Bit 16
#define PIN_0_MIN 18     // PORTA Bit 11
#define PIN_0_MAX 19     // PORTA Bit 10
#define PIN_1_STEP 60    // PORTA Bit 3
#define PIN_1_DIR 61     // PORTA Bit 2

#define PIN_2_MAX 2      // PORTB Bit 25
#define PIN_0_ENABLE 62  // PORTB Bit 17
```

```
21
22 #define PIN_2_MIN 3      // PORTC Bit 28
23 #define PIN_FAN 9        // PORTC Bit 21
24 #define PIN_0_STEP 46    // PORTC Bit 17
25 #define PIN_0_DIR 48     // PORTC Bit 15
26 #define PIN_2_ENABLE 38 // PORTC Bit 6
27
28 #define PIN_1_MAX 15 // PORTD Bit 5
29 #define PIN_1_MIN 14 // PORTD Bit 4
30
31 // Direction of stepper axis1 movement
32 #define CW -1
33 #define CCW 1
34
35 float t = 0.0;
36 float a = 0.0;
37 float b = 0.0;
38 float c = 0.0;
39 long packetTimer = 0;
40
41 union payload {
42
43   int32_t numbers[4];
44   uint8_t array[12];
45
46 } m_payload;
47
48 union integer {
49
50   int32_t number;
51   uint8_t array[4];
52
53 } m_integer;
54
55 enum protocolState
56 {
57   LISTENING, // blink disable
58   READ_LOAD, // blink enable
59   READ_END,  // we want the led to be on for interval
60   END_CMD    // we want the led to be off for interval
61 };
62
63 typedef struct
64 {
65   int max_vel;
66   // Hardware declarations
67   uint8_t enable_pin;
68   uint8_t step_pin;
69   uint8_t dir_pin;
70
71   // Motor status at any given time
72   volatile int8_t dir; //! Direction stepper axis1 should move.
73   volatile int32_t step_position;
74
75   // Interrupt variables
76   volatile uint32_t step_delay; //! Peroid of next timer delay. At start ↩
       this value set the accelration rate c0.
77   volatile uint32_t min_delay;  //! Minimum time delay (max speed)
78
79   // Interrupt handler
80   Tc *tc;
81   uint32_t channel;
```

```
82    IRQn_Type irq;
83
84  } m_motor_data;
85
86  m_motor_data axes[3];
87
88  void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq)
89  {
90    NVIC_ClearPendingIRQ(irq);
91    NVIC_EnableIRQ(irq);
92    TC_Start(tc, channel);
93  }
94
95  void stopTimer(Tc *tc, uint32_t channel, IRQn_Type irq)
96  {
97    NVIC_DisableIRQ(irq);
98    TC_Stop(tc, channel);
99  }
100
101 void restartCounter(Tc *tc, uint32_t channel)
102 {
103   // To reset a conter we se the TC_CCR_SWTRG (Software trigger) bit in ↩
        the TC_CCR
104   tc->TC_CHANNEL[channel].TC_CCR |= TC_CCR_SWTRG;
105 }
106
107 void configureTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t ↩
      frequency)
108 {
109   // Unblock thee power managent cotroller
110   pmc_set_writeprotect(false);
111   pmc_enable_periph_clk((uint32_t)irq);
112   // Configure
113   TC_Configure(tc,           // Timer
114                channel,      // Channel
115                TC_CMR_WAVE | // Wave form is enabled
116                    TC_CMR_WAVSEL_UP_RC |
117                    TC_CMR_TCCLKS_TIMER_CLOCK4 // Settings
118   );
119
120   uint32_t rc = VARIANT_MCK / 128 / frequency; //128 because we selected ↩
        TIMER_CLOCK4 above
121
122   tc->TC_CHANNEL[channel].TC_RC = rc; //TC_SetRC(tc, channel, rc);
123   // TC_Start(tc, channel);
124
125   // enable timer interrupts on the timer
126   tc->TC_CHANNEL[channel].TC_IER = TC_IER_CPCS;  // IER = interrupt enable↩
        register // Enables the RC compare register.
127   tc->TC_CHANNEL[channel].TC_IDR = ~TC_IER_CPCS; // IDR = interrupt ↩
        disable register /// Disables the RC compare register.
128
129   // To reset a conter we se the TC_CCR_SWTRG (Software trigger) bit in ↩
        the TC_CCR
130   tc->TC_CHANNEL[channel].TC_CCR |= TC_CCR_SWTRG;
131
132   /* Enable the interrupt in the nested vector interrupt controller */
133   // NVIC_EnableIRQ(irq);
134 }
135
136 void TC3_Handler()
137 {
```

```
138   TC_GetStatus(axes[0].tc, axes[0].channel); // Timer 1 channel 0 ----> ↩
          TC3 it also clear the flag
139   digitalWrite(axes[0].step_pin, HIGH);
140   digitalWrite(axes[0].step_pin, LOW);
141   axes[0].step_position += axes[0].dir;
142   // axes[0].tc->TC_CHANNEL[axes[0].channel].TC_RC = axes[0].step_delay;
143 }
144
145 void TC4_Handler()
146 {
147   TC_GetStatus(axes[1].tc, axes[1].channel); // Timer 1 channel 0 ----> ↩
          TC3 it also clear the flag
148   digitalWrite(axes[1].step_pin, HIGH);
149   digitalWrite(axes[1].step_pin, LOW);
150   axes[1].step_position += axes[1].dir;
151   // axes[1].tc->TC_CHANNEL[axes[1].channel].TC_RC = axes[1].step_delay;
152 }
153
154 void TC5_Handler()
155 {
156   TC_GetStatus(axes[2].tc, axes[2].channel); // Timer 1 channel 0 ----> ↩
          TC3 it also clear the flag
157   digitalWrite(axes[2].step_pin, HIGH);       // Step
158   digitalWrite(axes[2].step_pin, LOW);
159   axes[2].step_position += axes[2].dir; // Get motor position data
160   // axes[2].tc->TC_CHANNEL[axes[2].channel].TC_RC = axes[2].step_delay;
161 }
162
163 void setup()
164 {
165   Serial.begin(9600);
166   SerialUSB.begin(9600);
167   delay(3000);
168
169   pinMode(PIN_0_ENABLE, OUTPUT);
170   pinMode(PIN_0_DIR, OUTPUT);
171   pinMode(PIN_0_STEP, OUTPUT);
172
173   pinMode(PIN_1_ENABLE, OUTPUT);
174   pinMode(PIN_1_DIR, OUTPUT);
175   pinMode(PIN_1_STEP, OUTPUT);
176
177   pinMode(PIN_2_ENABLE, OUTPUT);
178   pinMode(PIN_2_DIR, OUTPUT);
179   pinMode(PIN_2_STEP, OUTPUT);
180
181   delay(3000);
182
183   SerialUSB.println("Starting timer..");
184
185   configureTimer(/*Timer TC1*/ TC1, /*Channel 0*/ 0, /*TC3 interrupt ↩
          nested vector controller*/ TC3_IRQn, /*Frequency in hz*/ T1_FREQ);
186   configureTimer(/*Timer TC1*/ TC1, /*Channel 0*/ 1, /*TC3 interrupt ↩
          nested vector controller*/ TC4_IRQn, /*Frequency in hz*/ T1_FREQ);
187   configureTimer(/*Timer TC1*/ TC1, /*Channel 0*/ 2, /*TC3 interrupt ↩
          nested vector controller*/ TC5_IRQn, /*Frequency in hz*/ T1_FREQ);
188
189   // Axis 1
190   axes[0].enable_pin = PIN_0_ENABLE;
191   axes[0].step_pin = PIN_0_STEP;
192   axes[0].dir_pin = PIN_0_DIR;
193   axes[0].max_vel = 15000; // 30k is the max
```

```
194   axes[0].tc = TC1;
195   axes[0].channel = 0;
196   axes[0].irq = TC3_IRQn;
197   axes[0].min_delay = T1_FREQ / axes[0].max_vel;
198
199   // Axis 2
200   axes[1].enable_pin = PIN_1_ENABLE;
201   axes[1].step_pin = PIN_1_STEP;
202   axes[1].dir_pin = PIN_1_DIR;
203   axes[1].max_vel = 15000; // MAX for this axes
204   axes[1].tc = TC1;
205   axes[1].channel = 1;
206   axes[1].irq = TC4_IRQn;
207   axes[1].min_delay = T1_FREQ / axes[1].max_vel;
208
209   // Axis 3
210   axes[2].enable_pin = PIN_2_ENABLE;
211   axes[2].step_pin = PIN_2_STEP;
212   axes[2].dir_pin = PIN_2_DIR;
213   axes[2].max_vel = 15000; // MAX for this axes
214   axes[2].tc = TC1;
215   axes[2].channel = 2;
216   axes[2].irq = TC5_IRQn;
217   axes[2].min_delay = T1_FREQ / axes[2].max_vel;
218
219   digitalWrite(PIN_0_ENABLE, LOW);
220   digitalWrite(PIN_1_ENABLE, LOW);
221   digitalWrite(PIN_2_ENABLE, LOW);
222
223   axes[2].tc->TC_CHANNEL[axes[2].channel].TC_RC = 100000;
224   axes[1].tc->TC_CHANNEL[axes[1].channel].TC_RC = 100000;
225   axes[0].tc->TC_CHANNEL[axes[0].channel].TC_RC = 100000;
226
227   startTimer(axes[2].tc, axes[2].channel, axes[2].irq);
228   startTimer(axes[1].tc, axes[1].channel, axes[1].irq);
229   startTimer(axes[0].tc, axes[0].channel, axes[0].irq);
230
231   SerialUSB.print("\nReady");
232 }
233
234 int setMotorSpeed(int32_t speed, uint8_t motor)
235 {
236   if (0 < motor && motor < 4) // If motor is in range (1,2,3)
237   {
238     // Enable motor output
239     digitalWrite(axes[motor - 1].enable_pin, LOW);
240
241     // GET AND SET DIRECTION VALUES
242
243     digitalWrite(axes[motor - 1].dir_pin, speed > 0 ? HIGH : LOW);
244
245     // Register those directions in the motor data structures
246     axes[motor - 1].dir = speed > 0 ? CCW : CW;
247
248     // ABS VALUES
249     speed = abs(speed);
250
251     /// SAT ON AXIS 1 ///
252     if (speed == 0) // VELOCITY ZERO
253     {
254       // Disable motor -> speed is zero
```

```
255       stopTimer(axes[motor - 1].tc, axes[motor - 1].channel, axes[motor - 1].irq);
256     }
257     else // VELOCITY != ZERO
258     {
259       // If its too high we saturate
260       if (speed > axes[motor - 1].max_vel)
261       {
262         speed = axes[motor - 1].max_vel;
263       }
264
265       // Calculate the delay according to speed
266       axes[motor - 1].step_delay = T1_FREQ / speed;
267
268       // Edit counting register with new delay time
269       stopTimer(axes[motor - 1].tc, axes[motor - 1].channel, axes[motor - 1].irq);
270       axes[motor - 1].tc->TC_CHANNEL[axes[motor - 1].channel].TC_RC = (uint32_t)axes[motor - 1].step_delay;
271       startTimer(axes[motor - 1].tc, axes[motor - 1].channel, axes[motor - 1].irq);
272     }
273 #ifdef debug_motor_data
274     SerialUSB.print("M: " + String(motor) + " ");
275     SerialUSB.print("ENA: " + String(axes[motor - 1].enable_pin) + " ");
276     SerialUSB.print("DIR: " + String(axes[motor - 1].dir_pin) + " ");
277     SerialUSB.print("SPEED: " + String(speed) + " ");
278     SerialUSB.print("DELY: " + String((uint32_t)axes[motor - 1].step_delay) + " ");
279 #endif
280     return 1;
281   }
282   else
283   {
284     return -1;
285   }
286 }
287
288 void loop()
289 {
290   if (millis() - packetTimer > TIMEOUT)
291   {
292     stopTimer(axes[2].tc, axes[2].channel, axes[2].irq);
293     stopTimer(axes[1].tc, axes[1].channel, axes[1].irq);
294     stopTimer(axes[0].tc, axes[0].channel, axes[0].irq);
295     digitalWrite(PIN_0_ENABLE, HIGH);
296     digitalWrite(PIN_1_ENABLE, HIGH);
297     digitalWrite(PIN_2_ENABLE, HIGH);
298   }
299
300   // If there is data in our RX buffer
301   if (Serial.available())
302   {
303 #ifdef debug_com
304     SerialUSB.println("\nIncomming data:");
305 #endif
306
307     // We initialize some control variables
308     int16_t byten = 11;         // Index of our input buffer, we start a 11 because of big-endian (I think)
309     uint8_t state = LISTENING; // Start off on a listening state
```

```
310     uint8_t new_packet = 0;     // Flag variable that indicates a valid new↩
            data packet
311     do
312     {
313       // We wait for data to come in
314       while (Serial.available() == 0)
315         ; // Wait for incomming data // ADD TIMEOUT GOD DAMN IT
316       // Then we read a byte of that RX buffer
317       uint8_t in_byte = Serial.read();
318
319 #ifdef debug_com
320       SerialUSB.print("State: " + String(state) + " Byte: ");
321       SerialUSB.print(in_byte, HEX);
322       SerialUSB.println();
323 #endif
324
325       // STATE MACHINE //
326       switch (state)
327       {
328       // Initially we are listenting for a start byte indicated by the 0↩
            x78 byte
329       case LISTENING:
330         if (in_byte == 0x7E) // CHECK IF ITS START BYTE
331         {
332 #ifdef debug_com
333           SerialUSB.println("\n START command read");
334 #endif
335           // If the start byte is read then we procceed to read the ↩
                payload
336           state = READ_LOAD;
337         }
338         else
339         {
340 // Wrong start command
341 #ifdef debug_com
342           SerialUSB.println("\nWrong START BYTE");
343 #endif
344           // If we started listenting and the first byte wasnt a start ↩
                byte then we restart the listening, the packen has begun ↩
                without the start packet
345           state = END_CMD; // RESET IF IT ISN'T
346         }
347         break;
348
349       // Reading payload state, payload has a fixed length of 12 anything ↩
            else will break the packet and restart the state machine
350       case READ_LOAD:
351
352         // Put data in our structure data
353         m_payload.array[byten] = in_byte;
354         byten--;
355         if (byten < 0)
356         {
357 #ifdef debug_com
358           SerialUSB.println("\nRecived 12 bytes");
359 #endif
360           // When we have read 12 bytes total we go to the end state
361           state = READ_END;
362           byten = 3;
363         }
364         break;
365
```

```
366       // This state must read a end command composed by 4 bytes
367       case READ_END:
368         m_integer.array[byten] = in_byte;
369         byten--;
370         if (byten < 0)
371         {
372 #ifdef debug_com
373           SerialUSB.println("\nRecived end command");
374           for (int i = 0; i < 4; i++)
375             SerialUSB.print(m_integer.array[i], HEX);
376           SerialUSB.println();
377           SerialUSB.println(m_integer.number, HEX);
378 #endif
379           // If the end command maches the predifined end command then we ↩
                can say that the packet is good.
380           if (m_integer.number == (int32_t)0x7FFFFFFF) // END COMMAND
381           {
382             // Data is correct
383             new_packet = 1;
384             state = END_CMD;
385 #ifdef debug_com
386             SerialUSB.println("\nEnd packet");
387 #endif
388           }
389           else
390           {
391 #ifdef debug_com
392             SerialUSB.println("\nWrong packet");
393 #endif
394
395             state = END_CMD;
396           }
397         }
398         break;
399       }
400     } while (state != END_CMD);
401
402     // Only when we have a new packet we can do somthing with it
403     if (new_packet)
404     {
405       // Reset flag
406       new_packet = 0;
407
408       // Set a timer to keep track of the amount of time since last packet
409       packetTimer = millis();
410
411       /// SET MOTOR SPEED ///
412       setMotorSpeed(m_payload.numbers[2], 1);
413       setMotorSpeed(m_payload.numbers[1], 2);
414       setMotorSpeed(m_payload.numbers[0], 3);
415
416 #ifdef debug_motor_data
417       SerialUSB.println();
418 #endif
419
420 #ifdef debug_control_msg
421       SerialUSB.println();
422       SerialUSB.print("Delay M1 = ");
423       SerialUSB.print(m_payload.numbers[2]);
424       SerialUSB.print(" M2 = ");
425       SerialUSB.print(m_payload.numbers[1]);
426       SerialUSB.print(" M3 = ");
```

```
427         SerialUSB.println(m_payload.numbers[0]);
428 #endif
429       }
430     }
431 }
```

Índice de Figuras

Índice de Tablas

Índice de Códigos

Índice alfabético