

Proyecto Fin de Carrera

Ingeniería Electrónica, Robótica y Mecatrónica

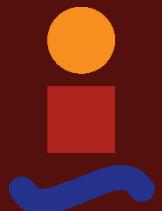
Diseño, fabricación y Control de un robot de
3GDL mediante visual servoing

Autor: Richard M. Haes-Ellis

Tutor: Ignacio Alvarado Aldea

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

Sevilla, 2020



Proyecto Fin de Carrera
Ingeniería Electrónica, Robótica y Mecatrónica

Diseño, fabricación y 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, 2020

Proyecto Fin de Carrera: Diseño, fabricación y 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 L^AT_EX 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 L^AT_EX 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, tanto 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

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
1 Introducción	1
1.1 Objetivos	1
1.2 Introducion al visual servoing (VS)	2
1.3 Aplicaciones	3
1.4 Alcance y limites del proyecto	3
2 Mecánica	5
2.1 Sistemas de transmision	5
2.2 Diseño CAD	6
2.3 Fabricación y ensamblaje	10
3 Electrónica	13
3.1 Motor	13
3.2 Pantalla tactil	18
3.3 Sensores	18
3.4 Microcontrolador	20
3.5 Placa de adaptación CNC	20
3.6 Fuente de alimentacion	21
4 Comunicación micro-pc	23
4.1 Introdución	23
4.2 Comunicacion UART	24
4.3 Protocolo a nivel de aplicación	25
5 Programación embebida	29
5.1 Sistema embebido	29
5.2 Interrupciones y Timers	30
6 Visión por computación	35
6.1 Introdución	35
6.2 OpenCV con python	36
7 Capítulo - Control mediante PID	43
7.1 Introdución	43
7.2 Implementacion de un PID en Codigo	44
7.3 Código python	46

7.4 Resultados	49
Apéndice A Sobre L^AT_EX	51
A.1 Ventajas de L ^A T _E X	51
A.2 Inconvenientes	51
Apéndice B Sobre Microsoft Word[®]	53
B.1 Ventajas del Word [®]	53
B.2 Inconvenientes de Word [®]	53
<i>Índice de Figuras</i>	63
<i>Índice de Tablas</i>	65
<i>Índice de Códigos</i>	67
<i>Bibliografía</i>	69
<i>Índice alfabético</i>	69
<i>Glosario</i>	71

Índice

<i>Resumen</i>	III
<i>Abstract</i>	V
<i>Índice Abreviado</i>	VII
1 Introducción	1
1.1 Objetivos	1
1.2 Introducion al visual servoing (VS)	2
1.3 Aplicaciones	3
1.4 Alcance y limites del proyecto	3
2 Mecánica	5
2.1 Sistemas de transmision	5
2.2 Diseño CAD	6
2.2.1 Tipos de articulaciones	7
2.2.2 pan-tilt	7
2.2.3 Eje 1 - Pan joint	7
2.2.4 Eje 2 - Tilt joint	8
2.2.5 Rail	9
2.3 Fabricación y ensamblaje	10
2.3.1 Impresion 3D	10
2.3.2 Ensamblaje	11
3 Electrónica	13
3.1 Motor	13
3.1.1 Funcionamiento	14
3.1.2 Driver	15
3.2 Pantalla tactil	18
3.3 Sensores	18
3.3.1 Sensores fin de carrera	18
3.3.2 Encoders magneticos	18
3.4 Microcontrolador	20
3.5 Placa de adaptación CNC	20
3.6 Fuente de alimentacion	21
4 Comunicación micro-pc	23
4.1 Introducción	23
4.2 Comunicacion UART	24
4.2.1 Expermiento	25
4.3 Protocolo a nivel de aplicación	25
4.3.1 Protocolo binario mediante maquina de estados	25

5 Programación embebida	29
5.1 Sistema embebido	29
5.2 Interrupciones y Timers	30
6 Visión por computación	35
6.1 Introducción	35
6.1.1 Tracking de objetos por visión	36
6.2 OpenCV con python	36
6.2.1 OpenCV	36
6.2.2 Python	37
6.2.3 Implementación	38
6.2.4 Resultados	40
7 Capítulo - Control mediante PID	43
7.1 Introducción	43
7.1.1 Bucle abierto - Bucle cerrado	44
7.1.2 Control PID	44
7.2 Implementacion de un PID en Código	44
7.3 Código python	46
7.4 Resultados	49
Apéndice A Sobre L^AT_EX	51
A.1 Ventajas de L ^A T _E X	51
A.2 Inconvenientes	51
Apéndice B Sobre Microsoft Word[®]	53
B.1 Ventajas del Word [®]	53
B.2 Inconvenientes de Word [®]	53
<i>Índice de Figuras</i>	63
<i>Índice de Tablas</i>	65
<i>Índice de Códigos</i>	67
<i>Bibliografía</i>	69
<i>Índice alfabético</i>	69
<i>Glosario</i>	71

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



Figura 1.1

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

El sistema se compone de un mecanismo motorizado de 2 grados de libertad al que se le conoce como *pan-tilt* montado encima de un rail motorizado que permite el desplazamiento lateral. El seguimiento de objetos mediante la cámara se realizará con los 2 grados de libertad del mecanismo pan-tilt. El tercer grado de libertad no se usa en este proyecto.

1.1 Objetivos

Se propone un robot al que se le indica mediante una interfaz gráfica una vista de la cámara, de aquí se selecciona con el puntero una zona de interés (ROI) que encierre el objeto que nos interesa seguir, una vez confirmada la selección el robot automáticamente centrará la imagen mediante sus motores.

El objetivo principal del proyecto es realizar este seguimiento mediante visión a tiempo real. Además se desarrollará todo el proceso de diseño y fabricación del robot, la programación del microcontrolador mediante interrupciones, la programación de comunicación entre el microcontrolador y el pc, la implementación de

un tracker para localizar en la imagen el objeto de interés y finalmente la implementación de un controlador PID en python para controlar los motores del robot. Este proyecto abarca varios ámbitos la robótica tales como la mecánica, electrónica, programación embebida, programación de alto nivel, control y visión artificial.

1.2 Introducción al visual servoing (VS)

Los sistemas de VS surgieron en los años 80. Visual servoing es una técnica que usa información visual mediante un sensor de visión para controlar un robot, sea controlando directamente sus articulaciones o comandando una trayectoria al robot. Las técnicas que usan el primer método se les llaman IBVS (image based visual servoing) mientras que el otro usa información geométrica extraída del sensor para estimar la posición del objetivo. Otras clasificaciones de VS derivan de las características de los componentes del sistema como por ejemplo la configuración de la cámara que pueden ser *eye-in-hand* o *hand-eye*, dependiendo en el bucle de control que pueden ser *end-point-open-loop* o *end-point-closed-loop*, dependiendo de que si se controla directamente o indirectamente mediante comandos de posición las articulaciones del robot.

Feddema et al. introdujo la idea de generar una proceso de generación de trayectorias con respecto a la velocidad del objeto con el fin de asegurar que los sensores no queden inefectivos para el movimiento del robot si estas fallan. Muchas de las técnicas de VS requieren que el objeto detectado sea conocido a priori y que garantizan la extracción de elementos visuales características del objeto.

Corke planteó una serie de preguntas críticas sobre VS con el fin de elaborar sobre sus implicaciones. En su artículo se enfoca en la dinámica del VS. El autor intenta combatir los problemas de latencia y de estabilidad que pueden surgir en este tipo de sistemas. Además habla de la generación de trayectorias en el bucle cerrado de control.

En el artículo de Chaumette se nos plantea dos problemas fundamentales de los IBVS. Uno es el servoing a algún mínimo local y el segundo es el acercamiento a puntos singulares. Las imágenes nos proporcionan suficiente información para combatir estos problemas por sí solos.

Durante los años se ha ido implementando sistemas híbridos que combinan varias técnicas basadas en múltiples cámaras, estadísticas para la estimación de posición, inteligencia artificial, etc..

Estos problemas son inherentes de brazos robóticos de múltiples grados de libertad, al simplificar el robot a dos grados de libertad nos libraremos de la mayoría de estos problemas. Durante la elaboración del proyecto se planteó usar perfiles trapezoidales para generar trayectorias. Sin embargo se ha descartado esa idea dado que se busca una respuesta rápida ante cambios de referencia o perturbaciones, luego la generación de trayectorias implica el cálculo iterativo y complica más aún el sistema.

Nuestro robot tendrá la cámara colocada en su efecto final tomando la configuración de *eye-in-hand* y controlaremos las articulaciones directamente con referencias de velocidad. En nuestro sistema se busca que el robot siga cualquier objeto que el usuario desee, por tanto los algoritmos de tracking solo son capaces de hacer seguimiento y no detección, dado que la detección suele venir acompañada de un entrenamiento de alguna IA. Esto es importante ya que si en cualquier momento el objeto deja de ser visible, habremos perdido nuestra referencia para comandar los motores. En este caso efectuaremos una parada inmediata a los motores.

1.3 Aplicaciones

La aplicación directa de este proyecto es la elaboración de tomas precisas y repetitivas con una cámara de video. En los montajes de publicidad cinematográfica se usan extensamente la robótica para conseguir videos estables, coordinados y precisos. El principal inconveniente de estos sistemas es el coste ya que un solo brazo robotico puede llegar a costar decenas de miles de euros. Con este proyecto se explora la posibilidad de conseguir similares resultados a un coste muy bajo, con el valor añadido de que el robot es capaz de seguir su objetivo sin intervención humana.



Figura 1.2

Es la razón por la que hay otro grado de libertad con el rail, para efectuar traslaciones con la cámara sin perder de vista al objetivo.

1.4 Alcance y límites del proyecto

El campo de visual-servoing entre en profundidad en técnicas de control aplicados a brazos robóticos de muchos grados de libertad para obtener resultados. Sin embargo, en este proyecto se pretende acercar al lector a este campo de la robótica de una forma simplificada y completa. Para ello se ha propuesto utilizar un robot de 2 grados de libertad montado en un rail de 1 grado de libertad.

2 Mecánica

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

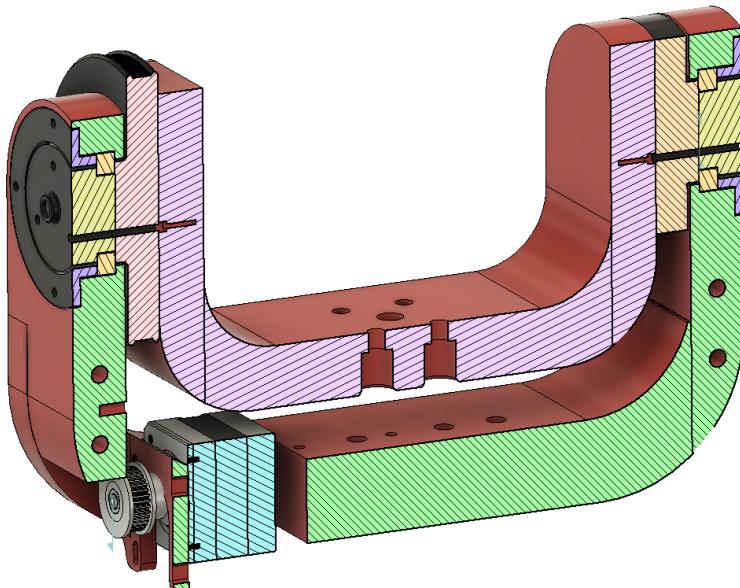


Figura 2.1

En este capítulo veremos todos los aspectos mecánicos del proyecto desde la análisis de los requisitos mecánicos hasta el diseño por CAD y fabricación de las piezas.

2.1 Sistemas de transmisión

Para poder mover las ejes del mecanismo de dos grados de libertad con los motores necesitaremos algún tipo de sistema de transmisión, ya que los motores producen un par bajo a la salida del eje. Si connectamos directamente este eje al elemento que acopla la cámara, el motor saltará pasos. Por tanto tenemos que buscar una transmisión que nos permita elevar el par resultante para soportar la carga del efecto final del mecanismo.

Hay varios tipos de transmisiones, seguidamente mostraremos los diferentes tipos y sus ventajas y desventajas.

- Engranajes: El movimiento rotacional del eje es transmitido a otro eje mediante engranajes. Es el más usado para sistemas de transmisión mecánica para robots. La potencia se transmite mediante el empuje



Figura 2.2

de dientes de un engranaje que engrana con los dientes de otro engranaje. En esta categoría hay varios tipos:

- Engranajes rectos: La configuración de dos engranajes paralelos y coplanarios se hace con engrajae rectos. El engranaje de entrada girará empujando el engranaje de salida cambiando su dirección y reduciendo o aumentando su velocidad dependiendo de la relación de transmisión. Es la configuración mas simple de esta categoría
- Engranajes cónicos: Son implementados usando dos ejes coplanarios que se interseccionan. Es capaz de transferir potencia muy elevada y además de cambiar el sentido de giro cambia la dirección del eje de salida por 90º.
- Engranajes helicoidales: Son iguales que los engranajes rectos con la diferencia de que sus dentado es oblicuo en relación a su eje de rotación. Esto permite trnasmitir muy elevadas potencias y consigue reducir de forma considerable las vibraciones con respecto a los engrajae rectos.
- Engranajes de tornillo: Estan formados por un engraje de entrada tipo tornillo y otro engranaje recto o helicoidal con un pitch igual al paso del tornillo. Esto permite una reducción muy elevada a costa de reducir considerablemente la velocidad. Tiene la propiedad de sólo se puede transmitir potencia en una dirección y se bloquea en la otra.

Las ventajas de este tipo de transmisión son la alta eficiencia de transmisión, alta durabilidad y la capacidad de conseguir altas relaciones de transmisión en poco espacio. La principal desventaja de este tipo son las vibraciones de engrane y la aparición de holgura o "*backlash*".

- Harmonicas: El un sistema de engranajes y "*splines*" que permiten una alta relación de transmisión en un espacio muy reducido y bajo peso. Es el sistema mas complejo de todos y el mas utilizado en la robotica industrial por su alta fiabilidad y precisión. Su principal desventaja es el alto coste.
- Poleas: Se trata de dos "ruedas" con una ranura en la superficie exterior, estas poleas giran en torno a un eje y están unidos por una correa que descansa en la ranura de cada polea. La correa, al estar en tensión, la polea motriz es capaz de transmitir potencia a la otra. Es uno de los sistemas de transmisión mas primitivos. Dentro de esta categoría existen un tipo llamado poleas dentadas. Estas poleas tienen unos dientes en vez de una ranura y la correa tiene unos dientes que engranan con los dientes de las poleas permitiendo transmitir potencia sin producirse ningún deslizamiento.

Para nuestro poryecto usaremos las poleas dentadas ya que nos proporciona una transmisión con posibilidad de reducir velocidad y elevar el par resultante. Además si se ajusta bien la correa obtendremos zero holgura entre ejes permitiendo un control preciso del efecto final.

2.2 Diseño CAD

Para diseñar el mecanismo se ha usado el programa de diseño Fusion 360. Es un programa de Autodesk que está disponible gratuitamente tanto para *makers* y estudiantes. Fusion es un programa "completo", es decir, incluye herramientas de CAD, CAM y CAE. Es intuitivo de utilizar con una curva de aprendizaje relativamente bajo.

2.2.1 Tipos de articulaciones

Antes de empezar a diseñar el robot, veremos diferentes tipos de articulaciones que nos podrán servir para nuestra aplicación. Para hacer un seguimiento de un objeto en una imagen 2D hace falta como mínimo dos grados de libertad para cumplir este requisito. Fijándonos en la Figura 2.3 podríamos seleccionar la articulación esférica, sin embargo este tipo de articulación es compleja de fabricar y controlar. Por tanto usaremos 2 articulaciones de tipo rotacional y 1 de tipo prismática para el rail.

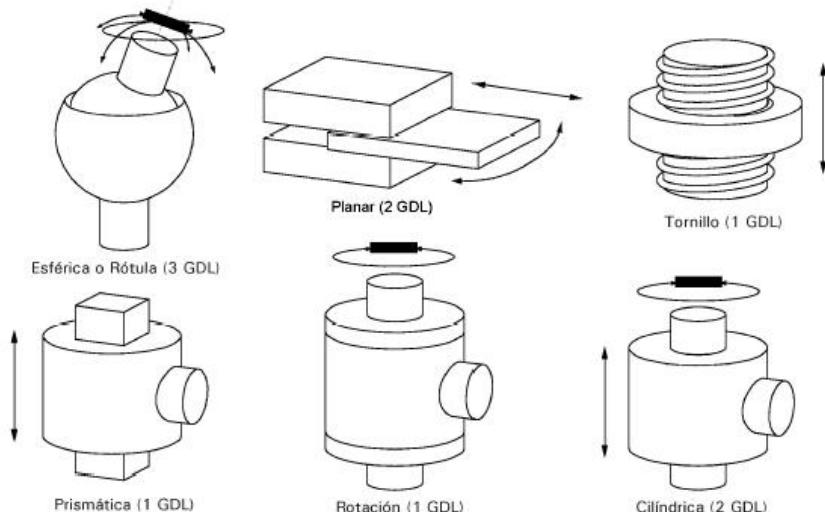


Figura 2.3

2.2.2 pan-tilt

En la figura Figura 2.4 podemos ver una ilustración del mecanismo que queremos realizar, consistiendo en dos articulaciones rotacionales con sus ejes coplanarios y que interseccionan con un ángulo de 90° .

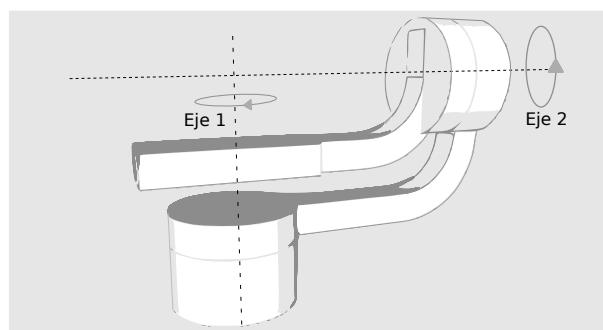


Figura 2.4

2.2.3 Eje 1 - Pan joint

El diseño final de la articulación 1 se puede ver en la figura Figura 2.5 . Como se puede ver en el corte, esta compuesto por una serie de piezas enumeradas. Las piezas 1 y 2 son rodamientos axiales cuya superficie exterior quedan fijadas entre la piezas 3,6 y 7 mediante tornillos. La parte interna del rodamiento queda fijada mediante la base 4 y el fijador 5 mediante tornillos. En la figura Figura 2.10 se ven un despiece del mecanismo completo. Se han usado 2 rodamientos para eliminar la holgura que existe entre las bolas de los rodamientos y anillo exterior e interior del rodamiento. La polea del eje motriz esta compuesta por ENE dientes y la polea de salida (pieza 7) esta compuesta por ENE dientes, con esto conseguimos una articulación sin holgura con una relación de transmisión de ENE y rotación continua. Esta articulación nos permite el movimiento lateral de la cámara.

Encima de la pieza 7 colocamos un mecanismo de conexión/desconexión rápida utilizada en la industria de cámaras llamada Manfrotto. Ver Anexo.

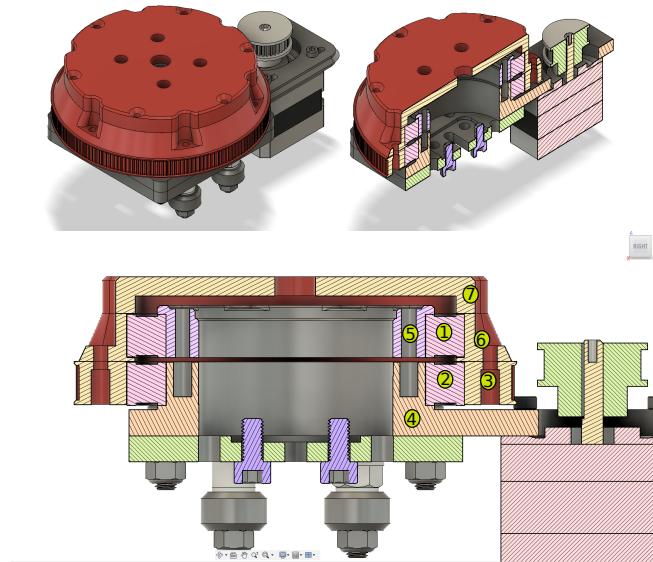


Figura 2.5

2.2.4 Eje 2 - Tilt joint

Para el eje 2 se ha realizado una estructura en forma de U tal y como se ver en la figura Figura 2.6 . La articulación esta compuesta por un rodamiento (pieza 3), una polea dentada de ENE dientes (pieza 5) o espaciador en el caso del lado opuesto, un retenedor para rodameinto (pieza 1) y un acoplador para fijar la estructura U al eje móvil del rodamiento (pieza 4). Al haber dos rodamientos en dos puntos de fijación no habra problemas de holgura por las tolerancias de los rodamientos. La polea del motor es de ENE dientes permitiendo una relación de transmisión de TAL. Esta articulación nos permite el movimiento vertical de la cámara.

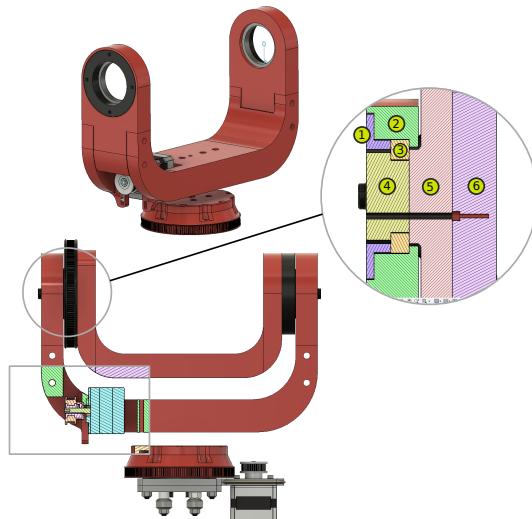


Figura 2.6

2.2.5 Rail

Para el rail se ha usado un perfil de aluminio estandarizado de 40x800 tipo C. Se le llama así por sus dimensiones y por su forma en C. Este perfil nos permite usar unos rodines que ruedan en la ranura interior del perfil (ver figura Figura 2.7). Estos rodines lo acoplamos a una plataforma permitiendo su giro. Esta plataforma podrá deslizarse por toda la longuitud del rail y nos permite el movimiento transversal del mecanismo de 2 grados de libertad.

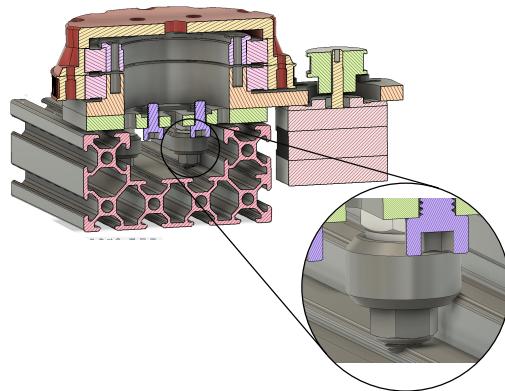


Figura 2.7

Para poder mover este eje se usa otra correa que se fija a la base de la plataforma y da la vuelta hacia el extremo alrededor de un tensor de correa, de aquí vuelve hasta el otro extremo del rail hasta la polea del motor y finalmente vuelve a la plataforma. Al tener la correa más larga debido a la longuitud del rail, la correa está propensa a estirarse con el tiempo. Por esa razón se ha realizado un mecanismo de autoajuste para mantener la correa en tensión. Este mecanismo está compuesto por un tornillo que va roscado al el cuerpo del mecanismo y retiene un resorte con una tuerca desde el interior. Este resorte expande la pieza al que esta fijada la polea loca. Si se rosca el tornillo, la correa se destensa mientras que si se desenrosca el tornillo se la correa se tensa.

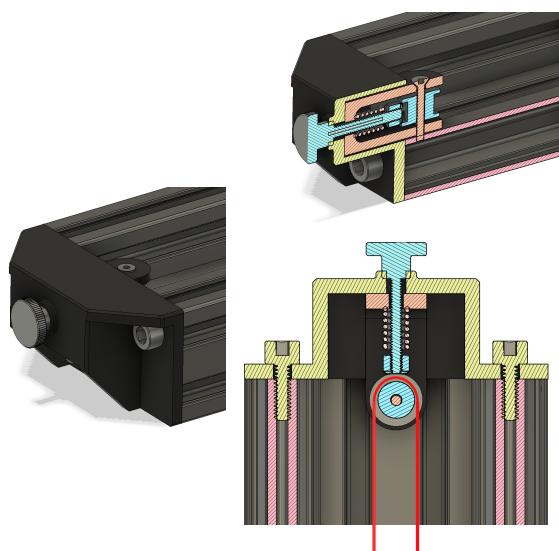


Figura 2.8

2.3 Fabricación y ensamblaje

Para la fabricación de la mayoría de las piezas del proyecto se ha recurrido a la impresión 3D.

2.3.1 Impresión 3D

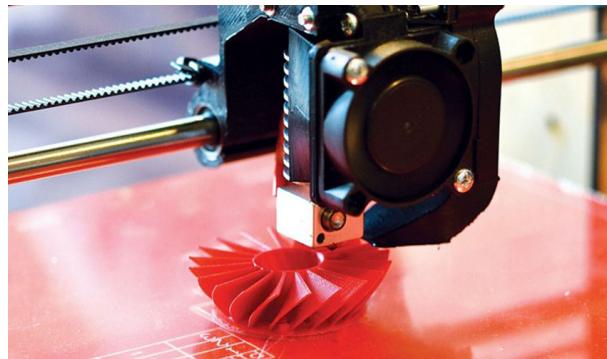


Figura 2.9

La impresión 3D o manufactura aditiva es un proceso automático de construcción de un objeto tridimensional generado a partir de un modelo CAD o modelo digital 3D. Esta reconstrucción se hace capa a capa hasta obtener el modelo completo. Por tanto el modelo 3D se corta en múltiples capas de un grosor determinado (normalmente 0.01-1mm). Estas capas o imágenes 2D obtenidas del modelo 3D son recorridas por un extrusor de plástico que va depositando un flujo determinado de material hasta cubrir la capa completa del modelo, luego sube hasta la siguiente capa y repite el proceso hasta completarse el modelo.

El principio de funcionamiento del proceso de manufactura mediante impresión 3D se puede resumir en tres partes:

- **Modelado:** El modelado de piezas puede realizarse mediante un programa de diseño asistido por ordenador (CAD), por un escáner 3D o por captación de imágenes y fotogrametría. El método por CAD es el método más exacto de los tres. Los errores debido a tolerancias de la máquina de impresión 3D se pueden corregir ajustando las dimensiones acorde. Además hay que tener en cuenta los límites de la impresión 3D, dado que se imprime capa por capa, hay ciertos casos en el que no es posible manufacturar una pieza dada con este método de fabricación. Una vez tengamos nuestro modelo lo podemos exportar en formato STL o 3MF. Estos formatos están diseñados para usarse para la fabricación aditiva.
- **Preparado del modelos:** Antes de poner a imprimir una pieza con nuestro archivo STL tenemos que preparar la pieza y generar las trayectorias que tiene que seguir la máquina para producir nuestra pieza. Este proceso se le suele denominar *Slicing*. Se refiere al rebanado de la pieza, esto es el corte sucesivo de la pieza en capas a partir del cual se puede autogenerar trayectorias 2D. En este proceso hay muchos parámetros que podemos ajustar para obtener una impresión óptima. Entre los parámetros más importantes están:
 - Altura de capa: Define la resolución de la impresión en el eje Z, cuanto más pequeño más resolución.
 - Temperatura de la cama: Para el PLA suele ser unos 50°C
 - Temperatura del extrusor: Para el PLA suele ser unos 220°C
 - Velocidad de impresión: En mm/minuto. Suele rondar los 100
 - Relleno: Define cuánto material usar para el interior de la pieza. 25% es suficiente en la mayoría de los casos.
 - Número de perimetros: Define el grosor de la cáscara de nuestra pieza.

Una vez ajustado los parámetros el *Slicer* autogenerará Código-G que nuestra impresora podrá interpretar. Una vez generada podemos encender la máquina y empezar la impresión.

- Acabado: Si nuestra pieza tiene algunos desperfectos o material de soporte, tendrémos que eliminar dicho material y lijar las piezas para obtener un acabado adecuado.

2.3.2 Ensamblaje

Uno de los aspectos más importantes durante el diseño de mecanismos y piezas es el orden de ensamblado. Es fácil diseñar un mecanismo cumpla los requisitos pero no nos servirá de mucho si es físicamente imposible ensamblar las piezas. En la figura Figura 2.10 podemos ver una vista despiezada del mecanismo.

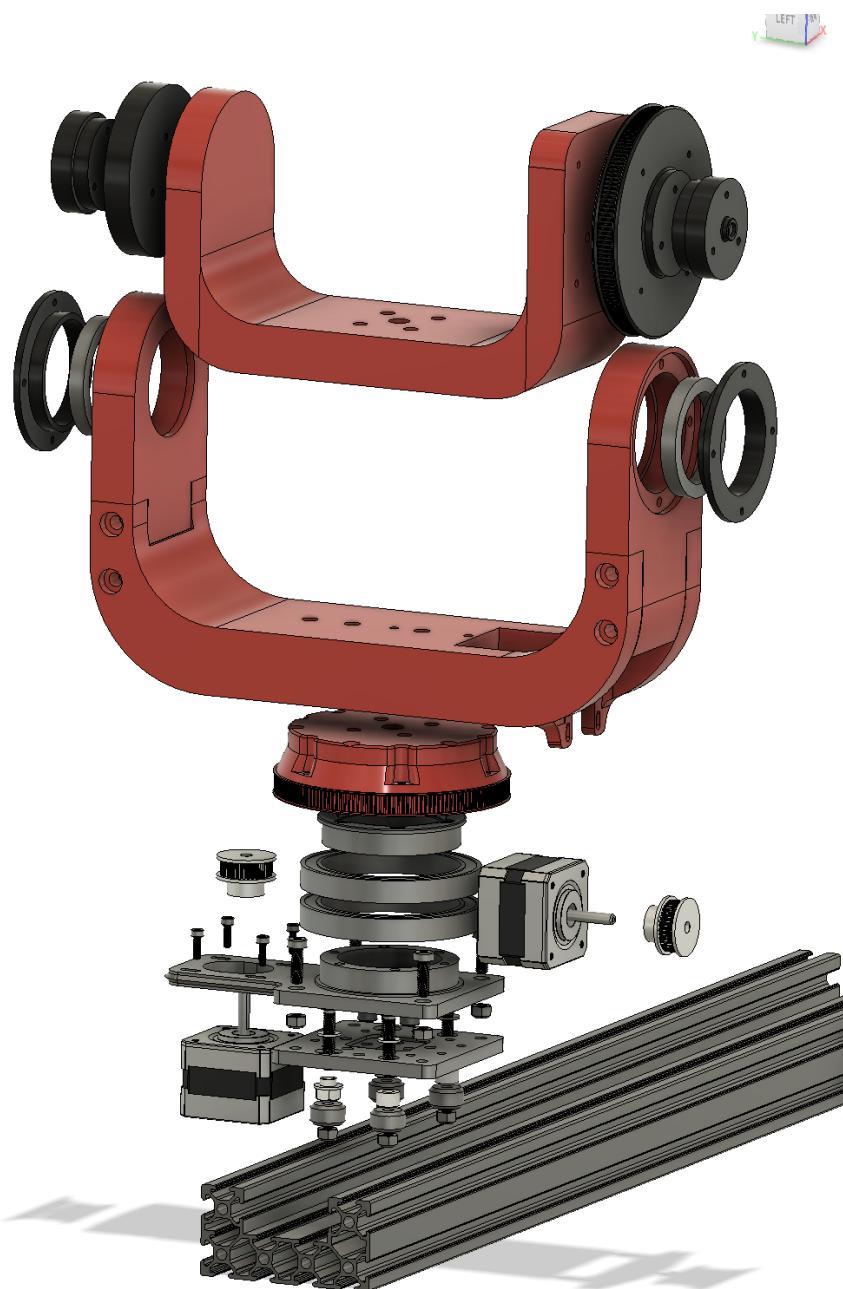


Figura 2.10

<https://www.roboticsbible.com/robot-mechanical-transmission-systems.html>

3 Electrónica

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

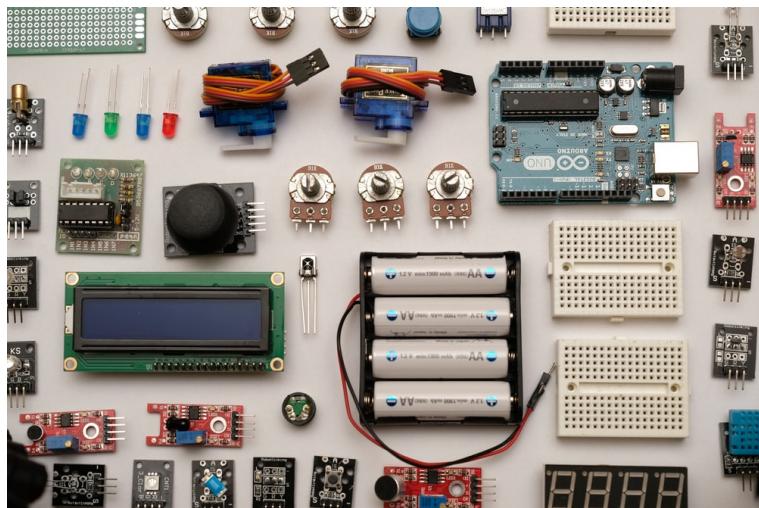


Figura 3.1

En esta sección analizaremos los diferentes componentes electrónicos y seleccionaremos aquellos que resultan más adecuados para este proyecto. Empezaremos detallando la parte electro-mecánica para accionar las articulaciones, veremos como interactuar con el sistema mediante una pantalla táctil, veremos que red de sensores usaremos para detectar el estado del sistema, selecciónaremos los drivers para alimentar y controlar de forma adecuada los motores, escojeremos un microcontrolador que será el que controla y monitoreará el sistema electrónico completo y por último veremos como alimentar el sistema.

3.1 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 ahí 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 magnético multipolar formado por dientes de hierro.

Cuando el estator se alimenta se induce un campo magnético en el rotor y genera un par en la dirección que minimiza la reluctancia magnética, por tanto atrae al polo más 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 están compuestos de un rotor que está axialmente magnetizado, es decir, tiene polos que alternan entre norte y sur paralelamente al eje. Su estator está compuesto por dos bobinados contenidos en unos entrehierros con dientes que interactúan con el rotor.

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

- **El motor de pasos híbrido:** La combinación de ambos motores de pasos dan lugar a los motores de pasos híbrido. Están compuestos de un rotor con dos partes magnetizadas y de polos opuestos con dientes que están desfasados entre sí. El estator está compuesto por bobinados y también están formados por unos dientes. Los dientes del rotor ayudan a orientar el flujo magnético a través de los dientes donde avanzan a las posiciones más favorables. Esto permite mejorar el par estático, dinámico 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 comparación con los otros dos, llegando a tener una precisión de hasta 1600 pasos por revolución con técnicas de comutación llamado "micro-stepping".

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

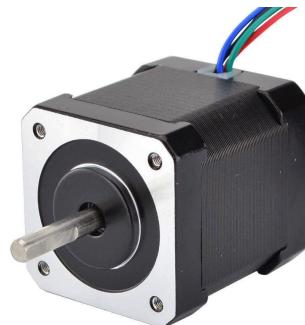


Figura 3.2 Motor de pasos NEMA-17.

3.1.1 Funcionamiento

Este motor cuenta con 4 cables, 2 para cada fase tal. En la figura Figura 3.3 vemos un esquema simplificado del motor mostrando dos fases uno para cada eje.

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. Esto provoca que el rotor se alinee con las fases rotando 90 grados por paso.

Sin embargo si alimentamos primero un bobinado, luego el otro manteniendo el primero habremos obtenido un nuevo paso intermedio. Esto se ve ilustrado en la figura Figura 3.3. Con esto hemos conseguido duplicar el número de pasos por revolución del motor y esta técnica es lo que llamamos "half-step". Pero ¿qué pasa si queremos más resolución?

Para obtener más precisión usamos una técnica llamada "micro-stepping", esto no es más que una extensión de lo anterior. Si miramos la ilustración de la figura Figura 3.3, podemos ver que la alimentación de los bobinados por igual, el vector magnético resultante 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 magnético resultante formando un ángulo de 26.56 grados con la horizontal. Para cualquier posición angular, las corrientes necesarias estarán definidas por el seno y coseno del ángulo requerido.

Cabe notar que si alimentamos dos bobinados por igual a su máxima corriente obtendremos un vector magnético más grande que si solo alimentamos un bobinado, esto se ve bien reflejado en el primer diagrama de la figura Figura 3.4 en los ángulos que están localizados en medio de cada bobinado.

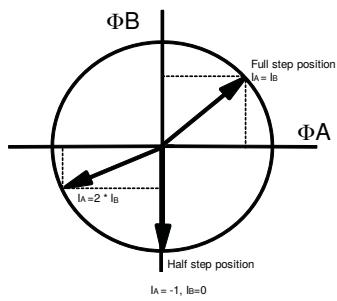


Figura 3.3 Ilustración microstepping.

Este fenómeno provoca vibraciones no deseadas, 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 operación del motor. De esta forma conseguiremos un par constante y evitará vibraciones indeseadas. Esto se ve claro en el segundo diagrama de la figura Figura 3.4.

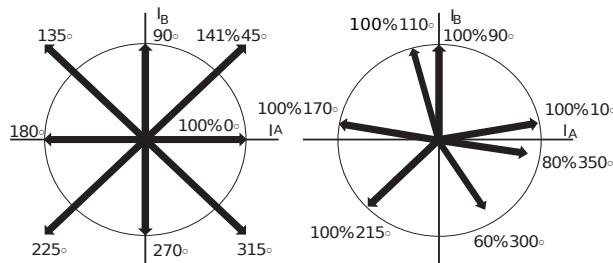


Figura 3.4 Ilustración vector magnético resultante.

3.1.2 Driver

Como hemos visto antes, para hacer girar el rotor del motor de pasos híbrido, tenemos que alimentar los bobinados para formar un vector magnético que oriente el rotor como queramos, para ello vamos a necesitar un driver que se encargará de convertir las señales de control en señales de potencia. Estos drivers suelen estar formados por una red de mosfets dispuestos en una configuración H. En la figura Figura 3.5 vemos el circuito de un puente-H para controlar un bobinado. En nuestro caso necesitaremos un driver que contega dos de estos puentes.

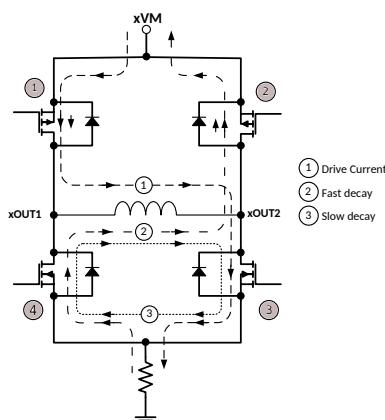


Figura 3.5 Circuito puente H.

Para alimentar una bobina tan solo tenemos que abrir o cerrar un par de transistores para que fluya corriente a través de él en una dirección u otra. Si queremos alimentar la bobina con una tensión positiva tan solo tendremos que abrir los transistores 1,3 y mantener cerrado los transistores 2,4, de esta forma la corriente fluirá

desde $xOUT1$ hasta $xOUT2$. Para alimentarlo al contrario simplemente cerramos 1,3 y abrimos 2,4. Con esto podemos controlar la dirección 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 3.6 se ilustra las corrientes en cada fase del motor, donde se ve claramente las senoides de corrientes del que hablamos anteriormente. Cabe notar el escalonado de dicha corriente, bien pues esto es debido a la cuantización de la señal PWM.

Las señales PMW se modulan con temporizadores que cuentan hasta un cierto valor de un registro de comparación (CCRX: Capture Compare Register X) y cambian la señal digital de estado, pasando de 1 a 0, cuando el timer llega hasta su valor máximo determinado (ARR) se resetea la señal PWM pasando de 0 a 1. Los contadores de los micros toman valores enteros, por tanto la señal PWM generada también tendrá 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 afecta directamente la resolución del motor. Otra consecuencia de esto es la aparición de ruido audible proveniente del motor.

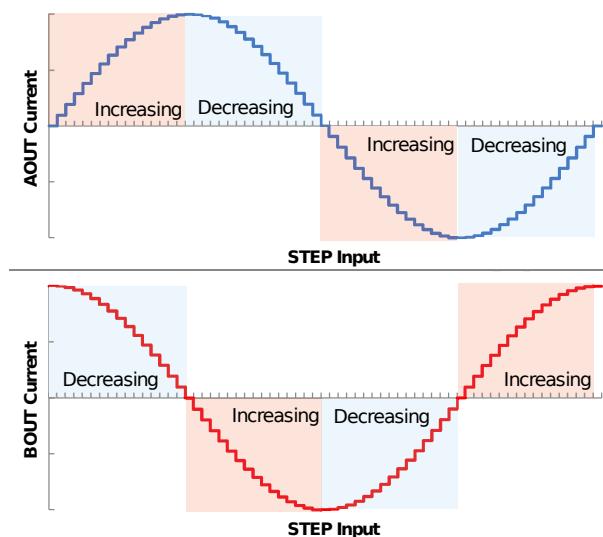


Figura 3.6 Diagrama de corrientes.

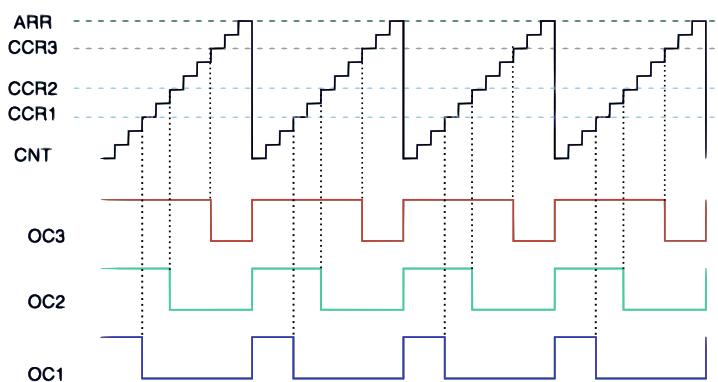


Figura 3.7 Diagrama temporal de señales PWM.

El driver que usaremos será uno de *Pololu*, el modelo A4988. Este driver es usado ampliamente en impresoras 3D por su bajo coste y alta fiabilidad. Cabe mencionar otros drivers como el *TMC22XX* de *Trinamic* que proporcionan otras funcionalidades tales como suavizado de corrientes para minimizar el ruido, detección 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. Fijándonos en el esquemático y guía de cableado de las figuras Figura 3.9 y Figura 6.1 del producto, se especifican 3 señales de control:



Figura 3.8 Driver pololu A4988.

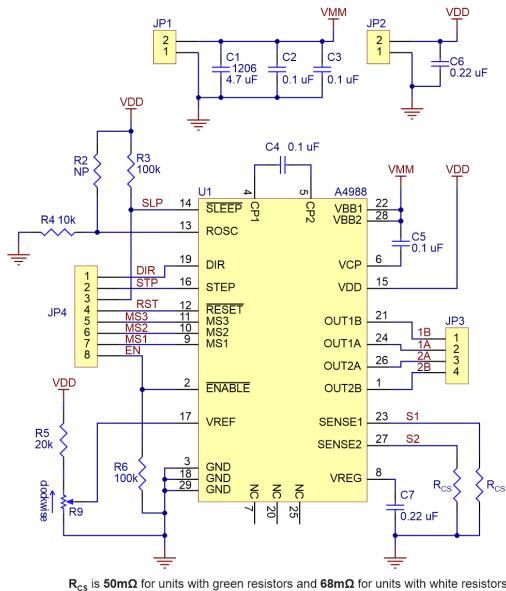


Figura 3.9 Esquematico del driver A4988.

- Señal STEP: Esta será la señal que marcará un paso del motor cuando pasemos de nivel bajo a nivel alto, dependiendo como este configurado los *micro-steps* hará que el motor gire una cierta cantidad.
- Señal DIR: Esta señal marcará la dirección del motor.
- Señal ENABLE: Activo a nivel bajo, será 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 y sobrecalentar el motor.

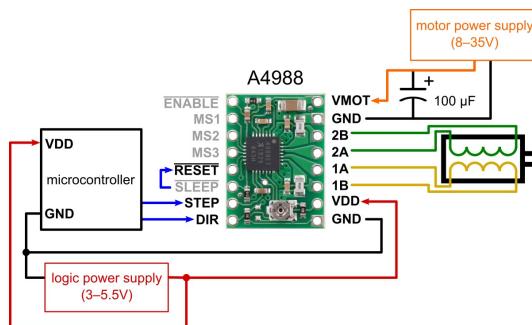


Figura 3.10 Guia de cableado.

Además contamos con unos pines de configuración que nos permitirá determinar el número de pasos por vuelta, dicha configuración se muestra en la Tabla 3.1. Los motores de pasos híbridos suelen tener 200

dientes, si tenemos una configuración de MS1=1, MS2=1 y MS3=1, tendremos un total de 3200 pasos por vuelta.

Tabla 3.1 Tabla de configuración *micro-stepping*.

MS1	MS2	MS3	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

3.2 Pantalla táctil

Para la pantalla se ha seleccionado una táctil de 320x240 pixeles de 2.4" de la marca Nextion. Esta pantalla permite crear interfaces gráficas muy competentes sin tener que cargar el microcontrolador. Esto lo consigue ya que todo el manejo de la pantalla lo hace un microcontrolador dedicado y todas las gráficas usadas en su interfaz están guardadas en una tarjeta microSD.

Para poder comunicarnos con esta pantalla usamos el puerto UART. Con tan solo mandar comandos simples podemos cambiar y actualizar cualquier elemento de la pantalla. Además contamos con una librería que nos permite leer y manejar cuando y donde se pulsa en la pantalla para reaccionar acorde.



Figura 3.11 Pantalla NEXTION.

3.3 Sensores

3.3.1 Sensores fin de carrera

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

3.3.2 Encoders magnéticos

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



Figura 3.12 Sensor montado en eje AS504X.

magnéticos de efecto hall. En la figura Figura 3.13 se muestra un diagrama del sensor elegido con una ilustración del su funcionamiento.

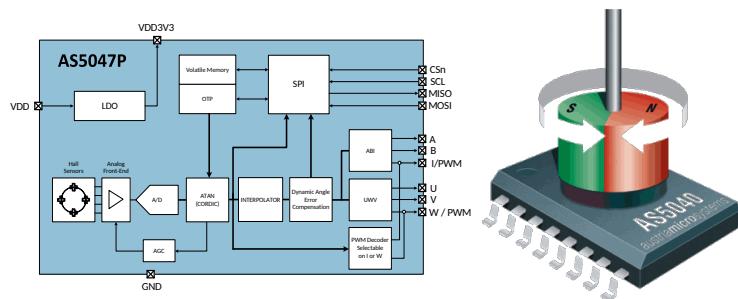


Figura 3.13 Sensor montado en eje AS504X.

El sensor magnético usado es uno que se monta en el propio eje del motor. Un imán que está magnetizado axialmente se coloca en el rotor del motor o eje móvil cuyo ángulo 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 con un puente de wheatstone, posee circuitería interna 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 disciplina de diseño de PCBs, se diseña 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.

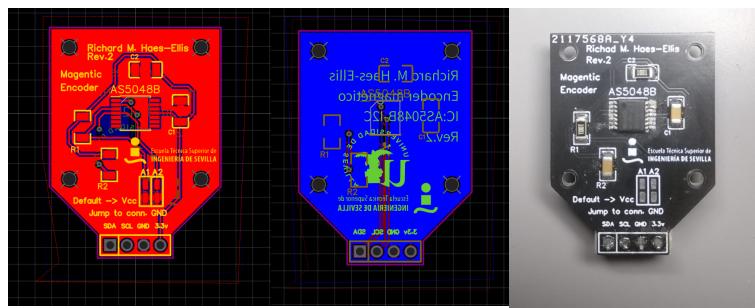


Figura 3.14 PCB para el sensor AS504X.

En la figura Figura 3.14 vemos una placa de adaptación para el sensor magnético integrado en configuración absoluta con interfáz i2C. Por tanto podemos leer mediante el bus i2C el valor absoluto de posición de la articulación deseada y como se conecta al bus i2C podemos conectar varios de estos sensores en cadena ya que esta dispuesto en la pcb unos jumpers para preseleccionar la dirección por defecto de dispositivo.

3.4 Microcontrolador

Para este proyecto necesitaremos un micro capaz 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.
- Un puerto I2C.
- Programación de interrupciones para el manejo de temporizadores.
- Velocidad de CPU 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 características mas importantes del micro:



Figura 3.15 Arduino DUE.

- Voltaje de operación: 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
- 4 USARTs y un UART
- 9 canales de 32-bit con Timer Counter (TC) para *capture, compare*

3.5 Placa de adaptación CNC

Para facilitar el interconexión de todos los componentes electrónicos se ha optado por usar una placa que esta destinada para el control de máquinas de control numérico (CNC) ya que cuentan con los condensadores de desacoplamiento, zócalos 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 3.16 se ve una imagen de dicha placa de adaptación, esta simplemente se coloca encima del microcontrolador y garantiza un conexiónado fijo.

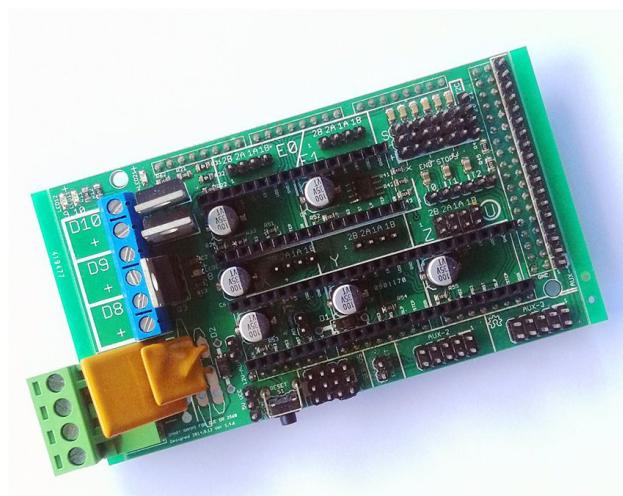


Figura 3.16 Placa de adaptación.

3.6 Fuente de alimentacion

Para poder alimentar el sistema entero basta con proporcionar una tensión de entre 7 y 12 voltios, ya que el microcontrolador lleva a bordo un regulador de tensión que regula la entrada jack a 5 y 3.3 voltios. Por tanto tenemos otras opciones a la hora de elegir una fuente.

El más versátil es la fuente regulable de tensión. Con él nos permite no solo alimentar la fuente a la tensión requerida sino también limitar y monitorear la corriente que absorbe nuestro sistema. Esto es útil a la hora de desarrollar el proyecto ya que si limitamos la corriente a un valor que evita que se quemen los componentes, si al conectar erroneamente algún componente sólo circulará la corriente especificada y el circuito quedará protegido.

Una vez terminada el proyecto, podemos fijarnos en la corriente que demanda en condiciones normales de operación y a partir de ahí determinar una solución más permanente. A partir de aquí podemos buscar un transformador o una batería de 12V capaz de suministrar la corriente absorbida en condiciones normales.

<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>

4 Comunicación micro-pc

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

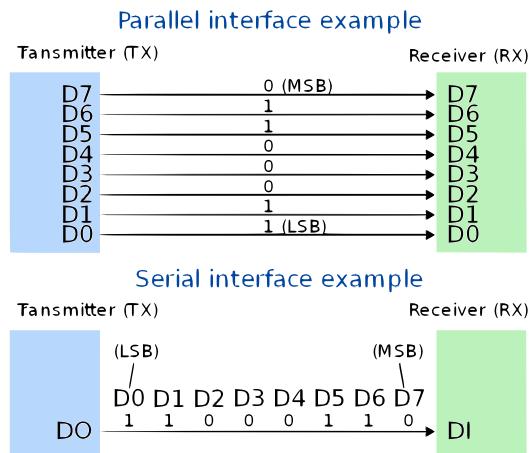


Figura 4.1

4.1 Introducción

Para poder controlar los motores desde un ordenador hace falta comunicarlos por alguno de los periféricos disponibles y compatibles entre el micro y el PC. Hay varias formas de comunicación entre dispositivos, podemos clasificar la comunicación entre 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 que existen varias pistas de datos o cables entre el transmisor y receptor para efectuar la transmisión de datos. Esta forma de comunicación es más rápida y también 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 de ordenadores para intercomunicar la RAM, CPU, GPU o puertos PCI, impresoras antiguas, etc..

**Figura 4.2 Serie vs Paralelo.**

La comunicación serie sin embargo envía datos secuencialmente por una sola vía o cable, empujando 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 única 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**.

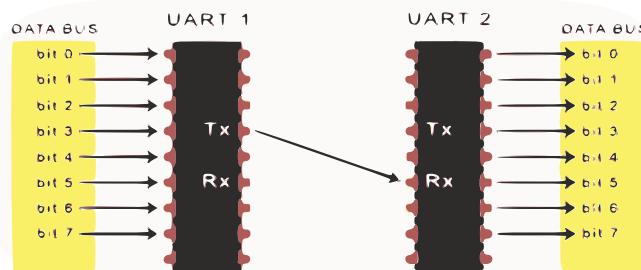
Con el avance en la tecnología de dispositivos de comunicación serial, estos se estan abarantando, se están haciendo más pequeño y más rápidos que nunca y poco a poco se esta sustituyendo los puertos paralelos por puertos serie. Esta será la comunicación que usaremos nosotros, en concreto la comunicación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

4.2 Comunicacion UART

UART o tambien conocido por sus siglas en ingés: *Universal Asynchronous Receiver Transmitter* 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 parámetros de la transmisión.

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 línea de reloj para sincronizar los datos. Cabe preguntarnos entonces como se sabe a que velocidad se comunican diferentes dispositivos. Bien pues ambos dispositivos de transmisión y recepción deben estar de acuerdo en los parámetros de temporización 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 4.3** ilustración de comunicación serie.

Fijándonos en la figura Figura 4.3 vemos una ilustración de UART. El puerto UART recibe 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 recibir/transmitir a la misma velocidad.

4.2.1 Experimento

Usando el microcontrolador, si lo programamos para que envíe 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 4.4

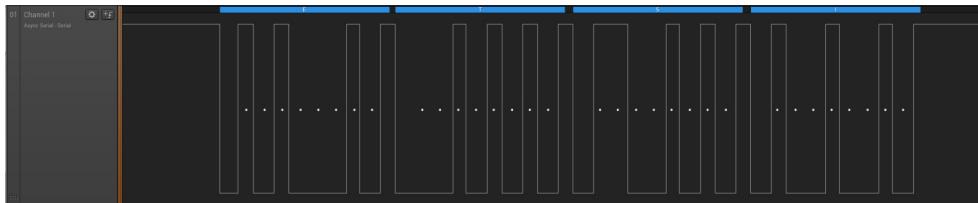


Figura 4.4 Análisis de una señal TX del microcontrolador DUE.

Si aplicamos un decodificador ASCII a esta señal vemos en fondo azul las letras decodificadas del dato y conseguimos leer el mensaje que hemos enviado.

4.3 Protocolo a nivel de aplicación

Como hemos visto antes podemos comunicar el PC con el micro mediante el puerto serie, por tanto podemos ya comunicar información de la velocidad de los motores para controlar la posición del robot. Pero hay que tener en cuenta que necesitamos mandar como mínimo tres variables de tipo entero que definirán la velocidades de los tres motores. Si enviamos los tres valores seguidos puede ocurrir que leamos el dato de forma errónea, sea por que leemos un bit de más, con tan solo eso ya se nos corrupta la información. Por tanto necesitamos algún 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.

4.3.1 Protocolo binario mediante maquina de estados

En la figura Figura 4.5 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 resulta no estar vacío quiere decir que nos esta llegando informacion desde el puerto serie.
- Estado inicio: En este estado se lee un sólo byte del registro RX y se comprueba que es igual al byte de comienzo, que resulta ser *0x7F* si resulta que no es igual, abortamos la lectura del paquete y empezamos de nuevo. Si resulta ser correcto saltamos al estado de lectura del payload.
- Estado leer *payload*: En este estado se leen los próximos 12 bytes del registro RX y nos pasamos al siguiente 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*, así nos aseguramos que hemos leído 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 desecharmos el paquete comenzando de nuevo el ciclo.

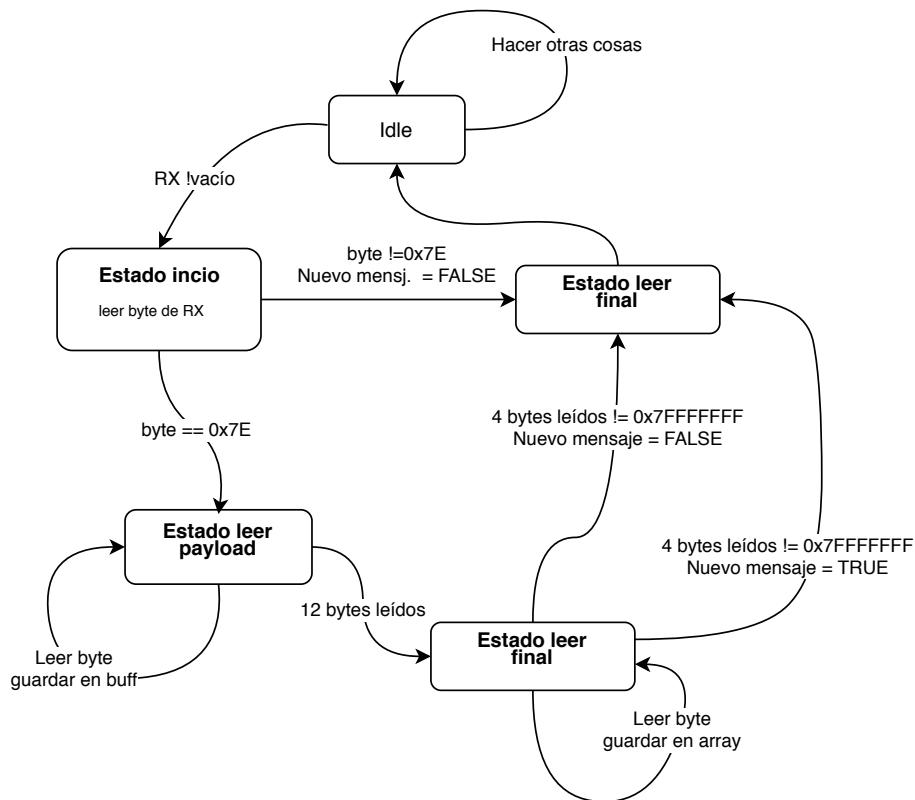


Figura 4.5 Máquina de estados del protocolo de comunicación.

Código 4.1 Código manejador de recepción de comandos.

```

1 // If there is data in our RX buffer
2 if (Serial.available()){
3
4 // We initialize some control variables
5 int16_t byten = 11;           // Index of our input buffer , we start a 11←
6 // because of big-endian (I think)
7 uint8_t state = LISTENING; // Start off on a listening state
8 uint8_t new_packet = 0;      // Flag variable that indicates a valid new←
9 // data packet
10 do{
11 // We wait for data to come in
12 while (Serial.available() == 0); // Wait for incomming data
13 // Then we read a byte of that RX buffer
14 uint8_t in_byte = Serial.read();
15
16 // STATE MACHINE //
17 switch (state)
18 {
19 case LISTENING: // Initially we are listenting for a start byte ←
20 // indicated by the 0x78 byte
21 if (in_byte == 0x7E){ // CHECK IF ITS START BYTE
22 // If the start byte is read then we proceed to read the ←
23 // payload
24 state = READ_LOAD;
25 }else{
26 // Wrong start command
  
```

```

23     // If we started listenting and the first byte wasnt a start ←
24     // byte then we restart the listening , the packen has begun ←
25     // without the start packet
26     state = END_CMD; // RESET IF IT ISN'T
27 }
28 break;
29
30 case READ_LOAD: // Reading payload state , payload has a fixed length←
31     // of 12 anything else will break the packet and restart the state ←
32     // machine
33     // Put data in our structure data
34     m_payload.array[byten] = in_byte;
35     byten--;
36     if (byten < 0){
37         // When we have read 12 bytes total we go to the end state
38         state = READ_END;
39         byten = 3;
40     }
41     break;
42
43 case READ_END: // This state must read a end command composed by 4 ←
44     // bytes
45     m_integer.array[byten] = in_byte;
46     byten--;
47     if (byten < 0){
48         // If the end command maches the predifined end command then we ←
49         // can say that the packet is good.
50         if (m_integer.number == (int32_t)0x7FFFFFFF){ // END COMMAND
51             // Data is correct
52             new_packet = 1;
53             state = END_CMD;
54         } else{
55             state = END_CMD;
56         }
57     }
58     break;
59 }
60 while (state != END_CMD);

```

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

5 Programación embebida

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE



Figura 5.1

5.1 Sistema embebido

Para describir como vamos a programar nuestro microcontrolador primero vamos a hacer una distinción entre un sistema embebido y un PC. A diferencia de un PC que tiene como propósito realizar tareas muy generales, 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 dándoles los trenes de pulsos a los drivers adecuadas para mover los motores a una velocidad específica.

Estas tareas se suelen programar directamente en la memoria del sistema mediante lenguajes de programación de bajo nivel tales como el lenguaje ensamblador o mediante un compilador específico a dicho sistema con C/C++ ya que no suelen tener un sistema operativo por encima regulando y gestionando los recursos para dicho programa. Por tanto tenemos mucha más 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 ningún otro proceso interrumpa su tarea.

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 capítulo 3, los motores de pasos funcionan alimentando los bobinados del estator, esto lo maneja el drive mediante MOSFETs de potencia. A su vez estos se controlan internamente por el driver en función de las señales de entrada del drive.

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 afecta un flanco de subida el driver actúa 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.

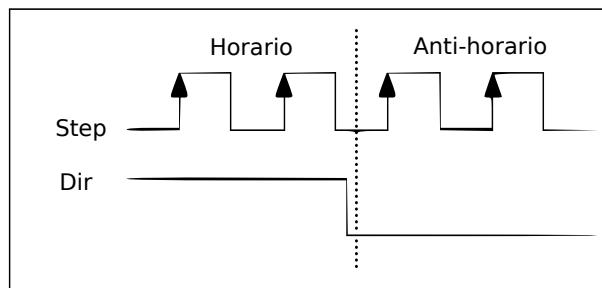


Figura 5.2

Se ve claro que la velocidad de los motores lo determinamos mediante la frecuencia de una señal cuadrada. Además tenemos que generar 3 de estas señales y asegurarnos que estemos mandando los pulsos en el tiempo adecuado. Realizar esto mediante GPIO activando y desactivando las señales en un bucle infinito es muy costoso, además tendremos al microcontrolador cargando con solo esta tarea si poder atender a otras tareas como la de atender a información de sus puertos de comunicación, leer los sensores, etc.. . Por tanto tenemos que buscar otra solución.

5.2 Interrupciones y Timers

Para gestionar el tren de pulsos para los drivers usaremos contadores con interrupciones, usaremos las interrupciones para ejecutar un manejador que enviará un pulso de señal de nivel bajo a nivel alto produciendo el flanco de subida necesario para que el driver actúe y se genere un paso en el rotor del motor. Además aprovecharemos esta interrupción para actualizar el registro de comparación 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 frecuencia de la señal de forma sincrona.

A raíz de lo anterior, esta claro que necesitaremos 3 contadores, uno para cada motor. Mirando el datasheet tanto del micro como el de la placa de adaptación vemos ver que pines están en uso:

Código 5.1 Definición de pines de la placa de adaptación.

```

1
2 #define PIN_0_DIR 48      // PORTC Bit 15
3 #define PIN_1_DIR 61      // PORTA Bit 2
4 #define PIN_2_DIR 55      // PORTA Bit 24
5
6 #define PIN_0_STEP 46     // PORTC Bit 17
7 #define PIN_1_STEP 60     // PORTA Bit 3
8 #define PIN_2_STEP 54     // PORTA Bit 16
9
10#define PIN_0_MAX 19      // PORTA Bit 10
11#define PIN_1_MAX 15      // PORTD Bit 5

```

```
12 #define PIN_2_MAX 2 // PORTB Bit 25
13
14 #define PIN_0_MIN 18 // PORTA Bit 11
15 #define PIN_1_MIN 14 // PORTD Bit 4
16 #define PIN_2_MIN 3 // PORTC Bit 28
17
18 #define PIN_0_ENABLE 62 // PORTB Bit 17
19 #define PIN_1_ENABLE 56 // PORTA Bit 23
20 #define PIN_2_ENABLE 38 // PORTC Bit 6
21
22 #define PIN_FAN 9 // PORTC Bit 21
```

Del datasheet y del ASF (Advanced Software Framework) del SAM3X8E podemos sacar la siguiente tabla de información acerca de los timers. Básicamente tenemos 3 timers con 3 canales por timer, cada canal tiene asociado su propia rutina de interrupción y su propio registro de comparación, por tanto tendremos en efectiva 9 timers de los cuales podemos elegir. Fijándonos en la tabla resumen seleccionaremos los timers que no tengan ninguna relación con los pines que vayamos a usar, ya que queremos usar esto para manejar el tiempo entre pasos e interrumpir internamente el programa para ajustar la velocidad, por tanto nos quedamos con los timers TC1 canal 0, TC1 canal 1, TC1 canal 2. Estos timers tienen las interrupciones TC3, TC4 y TC5 respectivamente.

Tabla 5.1 Tabla de interrupciones.

ISR/IRQ	TC	Channel	Due pins
TC0	TC0	0	2, 13
TC1	TC0	1	60, 61
TC2	TC0	2	58
TC3	TC1	0	none
TC4	TC1	1	none
TC5	TC1	2	none
TC6	TC2	0	4, 5
TC7	TC2	1	3, 10
TC8	TC2	2	11, 12

ISR: Interrupt service routine

IRQ: Interrupt request

TC: Timer counter

A continuación veremos un código plantilla de como se implementa estas interrupciones para generar el tren de pulsos que se enviará al driver del motor de pasos:

Código 5.2 Configuración de un timer.

```

1 void configureTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t frequency)
2 {
3     // Unblock the power manager controller
4     pmc_set_writeprotect(false);
5     pmc_enable_periph_clk((uint32_t)irq);
6     // Configure
7     TC_Configure(tc,           // Timer
8                  channel,       // Channel
9                  TC_CMR_WAVE | // Wave form is enabled
10                 TC_CMR_WAVSEL_UP_RC |
11                 TC_CMR_TCCLKS_TIMER_CLOCK4 // Settings
12 );
13
14     uint32_t rc = VARIANT_MCK / 128 / frequency; // 128 because we selected ←
15         TIMER CLOCK4 above

```

```

15     tc->TC_CHANNEL[channel].TC_RC = rc; //TC_SetRC(tc , channel , rc);
16     // TC_Start(tc , channel);
17
18     // enable timer interrupts on the timer
19     tc->TC_CHANNEL[channel].TC_IER = TC_IER_CPCS; // IER = interrupt ←
19     enable register // Enables the RC compare register.
20     tc->TC_CHANNEL[channel].TC_IDR = ~TC_IER_CPCS; // IDR = interrupt ←
20     disable register /// Disables the RC compare register.
21
22     // To reset a counter we set the TC_CCR_SWTRG (Software trigger) bit in ←
22     the TC_CCR
23     tc->TC_CHANNEL[channel].TC_CCR |= TC_CCR_SWTRG;
24
25     /* Enable the interrupt in the nested vector interrupt controller */
26     // NVIC_EnableIRQ(irq);
27
28 }
```

Fijandonos en el código 5.2, habilitamos primero el periférico IRQ mediante el pmc (*power management controller*), luego configuramos los relojes que vamos a usar para alimentar los contadores. Aquí indicamos que timer queremos usar, en que canal y el modo de la cuenta, en este caso va a ser en forma de onda (up-down), y usamos el CLK 4 que oscila a una frecuencia de $MCK/128$, es decir $84MHz/128 = 656.250Khz$.

Ya tenemos configurada la frecuencia de nuestro contador, ahora tenemos que fijar la cuenta máxima al que disparará la interrupción con el registro TC RC. Es aquí donde podremos cambiar la frecuencia de disparo de la interrupción. Inicialmente lo configuramos a 1, de este modo el contador cuando llegue a la cuenta 1 saltará la interrupción, esta interrupción saltará a una frecuencia de $656.250Khz$ que será el máximo.

Código 5.3 Arranque del timer.

```

1 void startTimer(Tc *tc, uint32_t channel, IRQn_Type irq)
2 {
3     NVIC_ClearPendingIRQ(irq);
4     NVIC_EnableIRQ(irq);
5     TC_Start(tc, channel);
6 }
```

Código 5.4 Parada del timer.

```

1 void stopTimer(Tc *tc, uint32_t channel, IRQn_Type irq)
2 {
3     NVIC_DisableIRQ(irq);
4     TC_Stop(tc, channel);
5 }
```

Para arrancar y parar el timer, y por tanto el motor, se usan 2 funciones mostradas en los códigos 5.3 y 5.4, uno para habilitar el contador que hemos configurado y otra para deshabilitarlo. Esto lo usaremos para parar momentáneamente la interrupción mientras estamos modificando los registros de la cuenta de comparación.

Código 5.5 Manejador de interrupción.

```

1 void TC3_Handler()
2 {
3     TC_GetStatus(axes[0].tc, axes[0].channel); // Timer 1 channel 0 -----> ←
3     TC3 it also clear the flag
4     digitalWrite(axes[0].step_pin, HIGH);
5     digitalWrite(axes[0].step_pin, LOW);
6     axes[0].step_position += axes[0].dir;
```

```
7 // axes[0].tc->TC_CHANNEL[axes[0].channel].TC_RC = axes[0].step_delay;  
8 }
```

Al saltar la interrupción se ejecutará el manejador de la interrupción, este manejador se encarga primero de borrar el flag de interrupción para que vuelva a saltar la proxima vez. Además realiza un pulso digital en la señal de *Step* provocando que efectue un paso en el rotor del motor. Y finalmente guarda la posición nueva en una variable.

6 Visión por computación

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

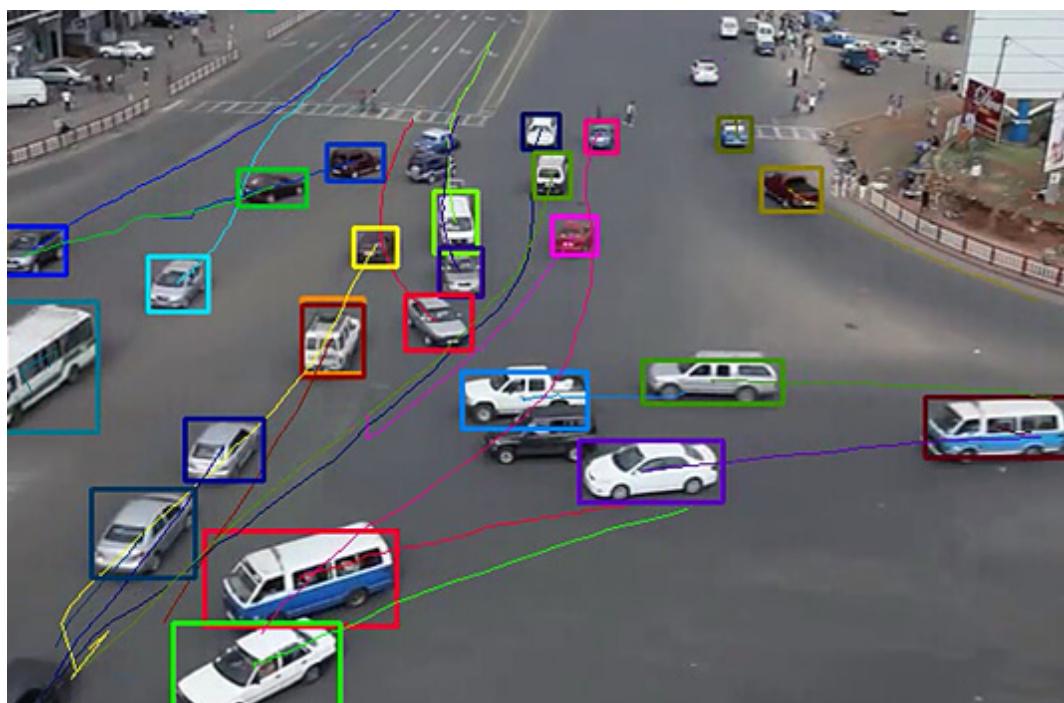


Figura 6.1 Reconocimiento y tracking de movimiento de vehículos.

En este capítulo explicaremos que es un sistema de visión por computación y su fundamento. Además veremos el hardware y el software necesario para conseguir capturar y procesar una imagen para luego realizar un seguimiento de un objeto.

6.1 Introducción

La visión por computación es una disciplina de la ciencia que se enfoca en como los ordenadores son capaces de interpretar imágenes digitales o vídeos a un nivel de abstracción alto. Trata de entender y automatizar las tareas que nosotros, los humanos somos capaces de hacer con nuestro sistema de visión.

Estas tareas incluyen la adquisición y procesamientos de imágenes, el análisis y entendimiento digital de las imágenes, y extracción de características de alto nivel del mundo real como por ejemplo detectar un

perro, o entender las líneas de una carretera etc.. La interpretación de esta información significa transformar imágenes en información descriptiva del mundo real que tiene sentido para la aplicación que se le quiere dar.

6.1.1 Tracking de objetos por visión

Tracking es el proceso de localizar un objeto móvil en una secuencia de imágenes o vídeo a lo largo del tiempo. Hoy en día esta disciplina es fundamental para las tecnologías emergentes tales como coches autónomos que necesitan visualizar y hacer un seguimiento de todos los coches que circulan en su entorno para predecir su comportamiento y evitar colisiones. También se usa extensamente para monitorear el comportamiento del tiempo y realizar predicciones.

El principal objetivo del tracking de video es hacer una relación de un objeto o zona de interés en un fotograma con el siguiente fotograma. Esta forma de relacionar y seguir el objetivo es especialmente difícil si el objetivo se está moviendo rápido y si cambia de orientación o tamaño en relación a la cámara.

Para realizar estas relaciones entre fotogramas, un algoritmo analiza secuencialmente los fotogramas y devuelve el movimiento del objetivo entre fotogramas. Hay muchos tipos de algoritmos que consiguen este resultado, pero hay que analizar más a fondo cuál usar dependiendo de las necesidades de nuestra aplicación ya que cada una tiene una serie de ventajas y desventajas. Veremos 2 componentes importantes en los que se pueden diferenciar estos algoritmos:

- **Filtrado y asociación de datos (Filtering and data association):** Es un método que incorpora datos sobre la escena o el objeto, requiere tratar la dinámica de la escena y depende de lo bueno o malo que sea la hipótesis. Estos métodos permiten hacer un seguimiento del objetivo incluso si este objeto es obstruido por otros elementos de la escena. La complejidad del algoritmo es aún más compleja si la cámara usada está localizada en una plataforma móvil. Esto requiere estabilización de la imagen antes de ser procesado. En comparación con el siguiente método este tiene un alto coste computacional. En seguida nombramos dos algoritmos comunes de este tipo:

- *Filtro de Kalman* : Es un filtro bayesiano recursivo. Se usan en sistemas lineales que están sujetos a ruidos con forma gaussiana. También hay variaciones de este filtro adaptadas para sistemas no lineales como el EKF (Extended Kalman Filter) o UKF (Uncentred Kalman Filter). Es un algoritmo que usa mediciones de diferentes sensores que contienen ruido y otras incertidumbres, y produce una estimación de la medida que tienden a ser más precisas que aquellas basadas en una sola medida. Un caso práctico de este filtro es fusionar el GPS y el odómetro de un vehículo para localizar su posición.
 - *Particle filter* : Este algoritmo solventa algunas de las limitaciones del filtro de kalman como la limitación a sistemas lineales. Es útil cuando se muestrea el espacio de estados del sistema no lineal de un proceso no gaussiano.

- **Representación y localización de objetivos (Target representation and localization):** Estos métodos aportan varias herramientas para identificar objetos móviles. Su eficacia es muy dependiente en el algoritmo en sí. Su coste computacional es bastante bajo en comparación con otras técnicas. Estos son los algoritmos más comunes:

- *Contour-tracking*, este método calcula iterando sobre un contorno inicial del fotograma anterior dando un nuevo contorno en el fotograma actual.
 - *Kernel-based tracking*, es un método iterativo basado en la maximización de la medida de similitud usando correlaciones entre el ROI anterior y una multitud de posibles ROI posteriores.

En nuestro proyecto usaremos exclusivamente algoritmos de tracking y no detección, ya que la detección abarca un tema que está fuera del alcance de este proyecto. Usaremos un algoritmo de representación y localización de objetivos usando *kernel correlation filters*. Para implementar este algoritmo usaremos una biblioteca de visión artificial llamada OpenCV para python.

6.2 OpenCV con python

6.2.1 OpenCV

OpenCV es una biblioteca de visión artificial desarrollada por Intel en 1999 por Gary Bradsky. En 2005 se usó en el vehículo *Stanley* que llegó a ganar la competición DARPA Grand Challenge. Desde entonces su desarrollo ha sido continuado como proyecto open-source.

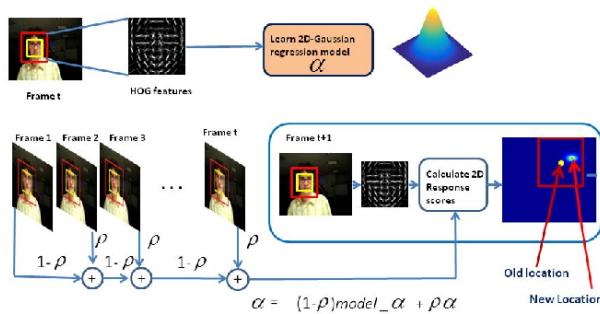


Figura 6.2 Esquema general del funcionamiento de KCF.

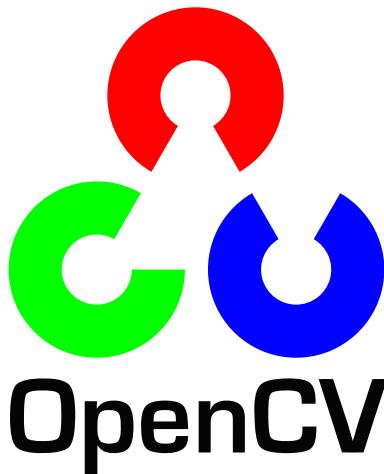


Figura 6.3 OpenCV.



Figura 6.4 Logo OpenCV.

Hasta en 2020 se siguen mencionando OpenCV como la librería de visión artificial más popular. Esto es debido a que su código fuente está publicado bajo la licencia BSD, que permite su uso libre tanto para uso comercial e investigación.

Además es multiplataforma, lo que permite desarrollar de forma más rápida una aplicación para la máxima cantidad de dispositivos. Otro de los puntos a favor es la documentación extensa de sus funciones y tutoriales. Lo que permite que sea accesible a un número mayor de personas.

6.2.2 Python

Python es un lenguaje de programación de uso general, interpretado y de alto-nivel. Fue creado por Guido Van Rossum en 1992. Se enfoca principalmente en la compresión de código y se le conoce por su uso extenso de indentaciones. Es un lenguaje con varios paradigmas de programación, esto permite a los programadores escribir código claro y conciso, tanto para proyectos pequeños como para proyectos a gran escala.

Python es un lenguaje interpretado al igual que *Matlab* y soporta varios paradigmas de programación

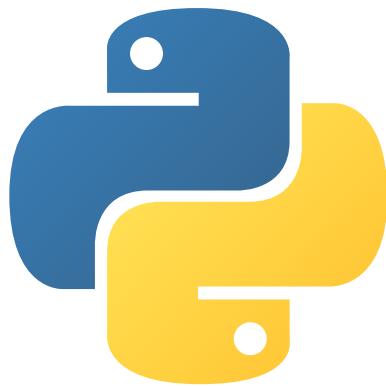


Figura 6.5 Logo python.

incluyendo estructurado, orientado a objetos y programación funcional. Posee una licencia de código abierto compatible con la licencia GNU y esta extensamente documentado. Además cuenta con una comunidad inmensa de desarrolladores lo cual encontrar soporte para este lenguaje es sencillo.

6.2.3 Implementación

Para nuestro proyecto hemos elegido la famosa cámara de modelo Logitech HD Pro Webcam C920 que nos proporciona un video stream de 60 fps a 1080p con autoenfoque. Para el PC hemos usado un portatil MSI GL65 9SEK con Intel® Core™ i7-9750H CPU @ 2.60GHz × 12, tarjeta gráfica GeForce RTX 2060/PCIe/SSE2 con 16GB de RAM. Para el sistema operativo usaremos Ubuntu 18.04LTS de Canonical.



Figura 6.6 Hardware y software.

Para nuestra aplicación queremos tener la posiblidad de seleccionar el objeto que deseamos seguir, esto lo haremos encerrando el objetivo deseado con un *bounding-box*, una vez que tenemos el bounding-box de la zona de interés, podrémos alimentar el algoritmo. La implementación se muestra en el código 7.8. Dicho código está explicado en comentarios.

Código 6.1 Código python para la visualizar y crear un bounding-box.

```
1 # Importamos las librerías necesarias
2 from imutils.video import VideoStream
3 import imutils
4 import time
5 import cv2
```

Primero incorporamos las librerías para manejar imágenes, el tiempo y el más importante la librería de OpenCV cv2.

Código 6.2 Código python para la visualizar y crear un bounding-box.

```
1 # Creamos nuestro tracker KCF
2 tracker = cv2.TrackerKCF_create()
3
```

```

4     # Inicializamos el bounding box del objetivo que queremos seguir
5     initBB = None
6
7     print("[INFO] Empezando video...")
8     vs = VideoStream(src=0).start()
9     time.sleep(1.0)

```

A continuación creamos nuestro tracker con opencv e inicializamos una variable que indica si se ha indicado un objetivo o no. Seguidamente arrancamos un stream de imágenes con la cámara.

Código 6.3 Código python para la visualizar y crear un bounding-box.

```

1     # Bucle principal
2     while True:
3         # Cojemos un frame de nuestro video
4         frame = vs.read()
5
6         # Reescalamos el frame para poder procesarlo mas rápido y ↵
7             # leemos su dimension final
8         frame = imutils.resize(frame, width=500)
9         (H, W) = frame.shape[:2]

```

En esta sección entramos en el bucle infinito y lo primero que hacemos es cojer un frame de la cámara y redimensionarla.

Código 6.4 Código python para la visualizar y crear un bounding-box.

```

1     # Nos preguntamos si ya estamos haciendo tracking de algún ↵
2         # objeto
3     if initBB is not None:
4         # Escojemos el nuevo bounding box seleccionado
5         (success, box) = tracker.update(frame)
6
7         # Si el tracking ha sido exitoso
8         if success:
9             # Pintamos un rectángulo en segun nos indica el ↵
10                # tracker
11             (x, y, w, h) = [int(v) for v in box]
12             cv2.rectangle(frame, (x, y), (x + w, y + h),
13                           (0, 255, 0), 2)
14
15         else: # Si no hay nada o si ha fallado
16             print("Lost target!")
17
18         # Creamos una tupula de información para imprimir en ↵
19             # pantalla
20         info = [
21             ("Tracker", "kcf"),
22             ("Success", "Yes" if success else "No"),
23             #("Control signals x:{} y:{}".format(x,y)),
24         ]
25
26         # Interamos por la tupula y pintamos sus elementos en el ↵
27             # frame
28         for (i, (k, v)) in enumerate(info):
29             text = "{}: {}".format(k, v)
30             cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
31                         cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
32             # show the output frame

```

En esta parte solo se ejecuta si se ha definido el objetivo con el bounding-box, si fuera el caso, recalculamos la nueva posición con el tracker KCF y vemos si ha sido existoso, en caso afirmativo recojemos el nuevo bounding-box calculado y lo pintamos en la pantalla, en caso contrario imprimimos por pantalla que hemos perdido el objetivo. Seguidamente se imprime por pantalla información del fps, el tipo de traker y su estado.

Código 6.5 Código python para la visualizar y crear un bounding-box.

```

1      # Enseñamos el fram procesado
2      cv2.imshow("Frame", frame)
3
4      # Leemos si se ha tecleado algo
5      key = cv2.waitKey(1) & 0xFF
6
7      # Si hemos pulsado la tecla s, seleccionaremos el bound box de la zona de interés.
8      if key == ord("s"):
9          # Guardamos la zona de interés seleccionada
10         initBB = cv2.selectROI("Frame", frame, fromCenter=False,
11                                showCrosshair=True)
12         # Arrancamos el tracker usando el bounding box seleccionado
13         tracker.init(frame, initBB)
14
15     # Si se teceló la q salimos del bucle y por tanto del programa
16     elif key == ord("q"):
17         break
18
19     # Paramos nuestro video
20     vs.stop()
21
22     # Cerramos todas las ventanas
23     cv2.destroyAllWindows()

```

Finalmente pintamos el nuevo frame en pantalla y leemos el teclado. La lectura del teclado se realiza para inicializar el tracker con un bounding-box y para terminar el programa.

6.2.4 Resultados

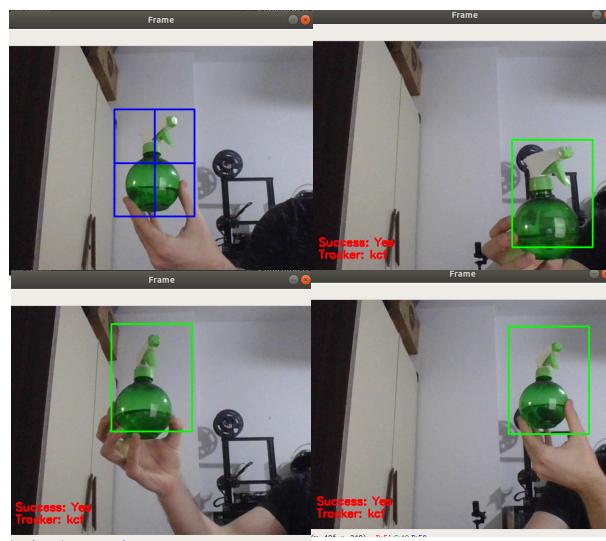


Figura 6.7 Resutados de selección de objetivos y tracking.

Como se puede apreciar en la Figura 6.7 hemos podido seleccionar un objetivo (en este caso un spray) con el bounding-box azul y al presionar la tecla enter se ha conseguido que el tracker KCF realice seguimiento del objeto encerrandolo en un bounding-box verde que nos da la posición a lo largo del timepo. En las figuras Figura 6.8 vemos estas posiciones graficadas tanto el el tiempo como en las coordenadas XY. En el siguiente capítulo se usará esta información para mover el robot de tal forma que se centre el objetivo en el centro de la imagen.

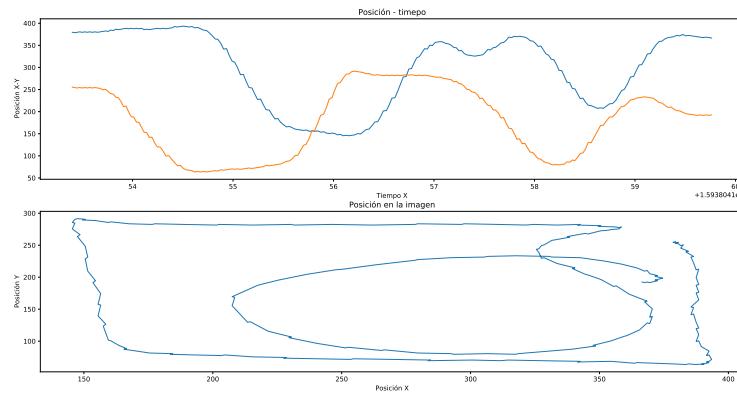


Figura 6.8 Resutados de selección de objetivos y tracking.

7 Capítulo - Control mediante PID

Una de las virtudes del ingeniero es la eficiencia.

GUANG TSE

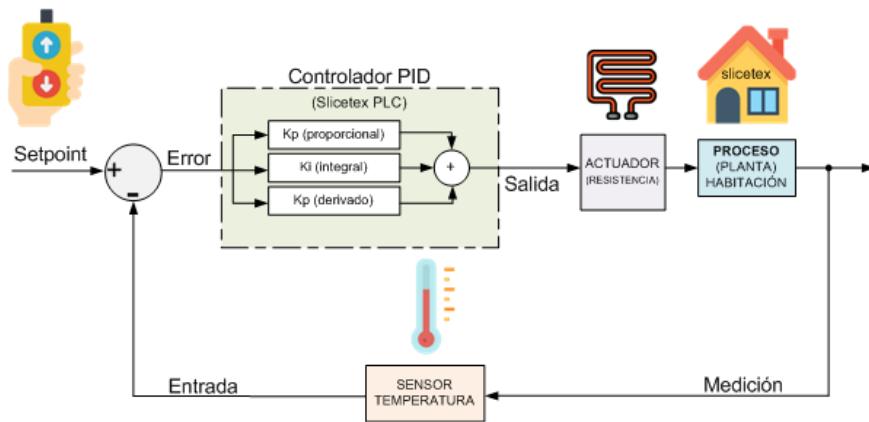


Figura 7.1

7.1 Introducción

La teoría de control se centra en el control de sistemas dinámicos en el tiempo en diversos procesos de ingeniería. El objetivo principal es desarrollar un modelo de control para controlar dichos sistemas usando una señal de actuación de la manera más óptima sin que haya retrasos, sobre disparos, oscilaciones indeseables y obteniendo al fin un sistema estable.

Para conseguir esto un controlador tiene que tener un comportamiento correctivo. El controlador monitoreiza a través de sus sensores la variable a controlar, y lo compara con la variable de referencia o *set-point*. La diferencia entre variable actual y la deseada se le llama error, esta señal es la que se realimenta a nuestro controlador para generar una señal de control que hace que se minimice tal valor de error hasta conseguir que el valor actual esté lo más cercano al valor deseado.

Hay otros aspectos a tener en cuenta como la controlabilidad y observabilidad. Este es la base de control avanzado que ha revolucionado la manufactura, control de aviones, cohetes y otras industrias. En la figura TAL se puede ver un diagrama de bloques de un sistema cualesquiera. Esta compuesta por el controlador C, el sistema G y el sensor S.

7.1.1 Bucle abierto - Bucle cerrado

Fundamentalmente hay dos tipos de bucles de control, bucle abierto y bucle cerrado (*feedback*).

- En un control de bucle abierto, la acción de control de la controladora es independiente de las variables de proceso o sistema. Un ejemplo sencillo de este tipo de control es el calentamiento de una tostadora con temporizador. En este caso la señal de control es el temporizador que delimita el tiempo de operación, cuando se pone en marcha se alimentan las resistencias y estas empiezan a calentarse, pero en ningún momento se está monitoreando la temperatura de estos elementos. Por tanto la temperatura de los elementos y la acción de apagado del temporizador no están relacionados.
- En un control de bucle cerrado, la acción de control de la controladora es dependiente de la señal retroalimentada del proceso. Volviendo al ejemplo de la tostadora, un control en bucle cerrado consistiría en añadirle un sensor de temperatura en las resistencias, este sensor monitorea la temperatura y realimenta esta información y lo compara con un valor deseado, la controladora entonces calculará una señal de control que mantenga la temperatura a la deseada.

7.1.2 Control PID

Una controladora PID, o por sus siglas en inglés, *Proportional-Integral-Derivative*, es un algoritmo de control ampliamente usado en miles de sistemas. Este tipo de controlador está implementado en miles de sistemas de control analógicos; originalmente en controladoras mecánicas, luego se pasó a electrónica discreta y más adelante en los procesos industriales con ordenadores. El controlador PID es probablemente el control en bucle cerrado más utilizado.

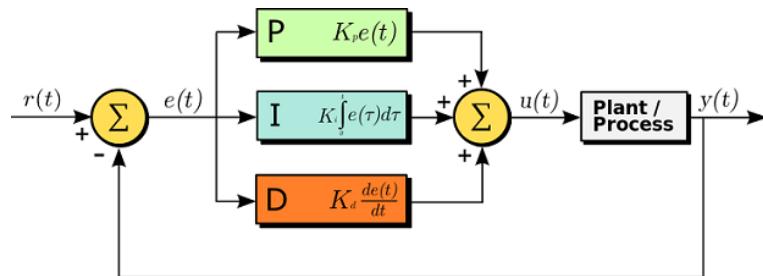


Figura 7.2 Esquema de un PID.

La controladora PID calcula de forma continua el valor de error $e(t)$ como la diferencia entre la variable deseada y la variable actual medida del proceso, luego calcula una señal de control basado en el valor proporcional al error, el valor de la integral del error y el valor derivativo del error.

Si llamamos $u(t)$ a la señal de control, $y(t)$ la señal medida de la salida y $r(t)$ a la señal de referencia entonces $e(t) = r(t) - y(t)$ será el error, la ecuación genérica de un controlador PID es:

$$u(t) = K_P e(t) + K_I \int e(\tau) d\tau + K_D \frac{de(\tau)}{dt} \quad (7.1)$$

La obtención de la dinámica deseada del control se puede conseguir ajustando los términos K_P , K_I y K_D , a menudo este proceso de "tuning" es iterativo y no requiere información del modelo dinámico del sistema. La estabilidad se puede conseguir ajustando adecuadamente el término proporcional. El término integral lo podemos ajustar para rechazar perturbaciones externas y obtener un error nulo en régimen permanente. El término derivativo se puede ajustar para conseguir un efecto de suavizado de la respuesta de la señal de control.

7.2 Implementación de un PID en Código

Ahora que tenemos un sistema en el cual podemos controlar la velocidad de los motores y mediante la cámara obtener el centro de nuestro objetivo de interés podremos implementar el controlador para centrar el robot en el objetivo. Para ello mostraremos la implementación del PID en forma discreta.

Como estamos trabajando en una computadora no podemos realizar la integración de forma continua por ello el cálculo del término integral se ha usado el método de Euler hacia adelante, ver Figura 7.3.

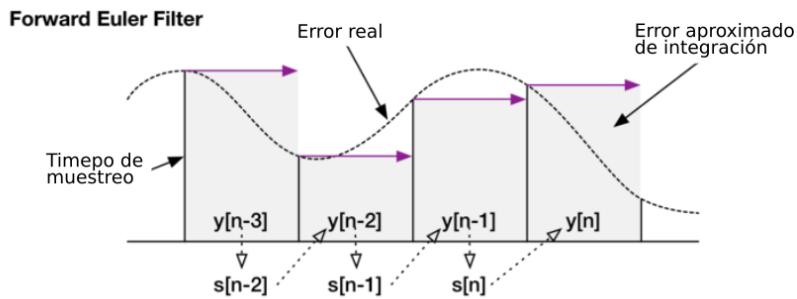


Figura 7.3 Método de integración numérica.

Código 7.1 Código python para la visualizar y crear un bounding-box.

```

1 # Inicializamos timepo del bucle y variables
2 loop_frequency = 30
3 dt = 1.0/loop_frequency # 30 fps
4
5 # Variables de error
6 epx_1 = 0
7 epy_1 = 0
8 eix = 0
9 eiy = 0
10
11 # Parametros del PID
12 kp = 60
13 ki = 0.25
14 kd = 1.8
15
16 while True:
17     # Calcular posicion central del objetivo
18     (x, y, w, h) = [int(v) for v in box]
19
20     # Calcular error p
21     epx = (x+w/2)-(W/2)
22     epy = (y+h/2)-(H/2)
23
24     # Calcular error i
25     eix = eix + epx*dt
26     eiy = eiy + epy*dt
27
28     # Calcular error d
29     edx = (epx-epx_1)/dt
30     edy = (epy-epy_1)/dt
31
32     # Calcular las señales de control
33     u_x = kp*epx + ki*eix + kd*edx
34     u_y = kp*epy + ki*eiy + kd*edy
35
36     # Enviar señales de control a los motores
37     ser.write(sendPacket(0,u_x,u_y))
38
39     # Actualizar varibales
40     epx_1 = epx
41     epy_1 = epy

```

7.3 Código python

En esta sección veremos el código completo del proyecto. Para empezar veremos las librerías que necesitaremos y las variables de inicialización y configuración.

Código 7.2 Código python para la visualizar y crear un bounding-box.

```

1 # import the necessary packages
2 from imutils.video import VideoStream
3 from imutils.video import FPS
4 import argparse
5 import imutils
6 import struct
7 import time
8 import serial
9 import binascii
10 import cv2

```

Las librerías de imutils nos dan herramientas para tratar las imágenes procedentes de la cámara. Struct lo usaremos para empaquetar los datos de las velocidades del queremos comandar a los motores, de esta forma podemos mandar ese paquete directamente por el puerto serie. Time lo usamos para manejar los tiempos del bucle. Serial nos permitirá abrir un puerto serie para comunicarnos con el micro. Binascii nos permitirá convertir datos binarios a ascii y viceversa. Cv2 es la librería de OpenCV de donde sacaremos los algoritmos de tracking.

Código 7.3 Código python para la visualizar y crear un bounding-box.

```

1 # grab the appropriate object tracker using our dictionary of
2 # OpenCV object tracker objects
3 tracker = cv2.TrackerKCF_create()
4
5 # initialize the bounding box coordinates of the object we are going
6 # to track
7 initBB = None
8
9
10 print("[INFO] starting video stream...")
11 vs = VideoStream(src=0).start()
12 time.sleep(1.0)
13
14 # initialize the FPS throughput estimator
15 fps = None
16 loop_frequency = 30
17 dt = 1.0 / loop_frequency # 30 fps
18 prev = 0
19
20 # Error variables
21 epx_1 = 0
22 epy_1 = 0
23 eix = 0
24 eiy = 0
25
26 # PID parameters Kp Ki Kd
27 kp = 60
28 ki = 0.25
29 kd = 1.8
30
31 # Open serial connection to arduino
32 ser = serial.Serial("/dev/ttyACM0", 9600)

```

```

33 time.sleep(3)
34
35 cv2.namedWindow("Frame", cv2.WINDOW_NORMAL)
36 # set your desired size
37 cv2.resizeWindow('Frame', 1080, 1920)

```

En esta sección creamos un objeto de la librería de openCV para el tracker. Usaremos este objeto para calcular la nueva posición de objetivo. InitBB es una variable tipo flag que nos indica si ya hemos indicado o no la zona de interés que queremos seguir. VideoStream lo llamamos para crear un video stream al que podemos referenciar con la variable que devuelves. Loop frequency sera la frecuencia al que funcionara nuestro control y seguimiento, por tanto cada 1/30 segundos se recalculará la nueva posición del objetivo, se calculará el error cometido y con ello se calculará la nueva comanda de velocidades para los motores para minimizar el error. Luego tenemos inicializado las variables de control de error $epx_1 = 0, epy_1 = 0, eix = 0, eiy = 0$. K_p, K_i, K_d serán los coeficientes del controlador PID. Serial.Serial nos abrirá el puerto serial con nuestro micro. Y finalmente abrimos una ventana con opencv con namedWindow y resizeWindow, llamando a la ventana "Frame".

Código 7.4 Código python para la visualizar y crear un bounding-box.

```

1
2 def sendPacket(v1,v2,v3):
3     """Packs a python 4 byte integer to an arduino unsigned long"""
4     packet = struct.pack('>bihi',126,int(v1),int(v2),int(v3),2147483647)
5     # print(binascii.hexlify(bytearray(packet)))
6     return packet    #should check bounds

```

La función sendPacket nos va a permitir mandar números enteros que representarán las velocidades de los motores y transformar y empaquetar esa información a binaria para luego mandarlo por el puerto serial.

Ahora veremos el bucle principal, lo vamos a dividir en 4 partes e iremos explicando cada una por separado:

Código 7.5 Código python para la visualizar y crear un bounding-box.

```

1
2 # loop over frames from the video stream
3 while True:
4     time_elapsed = time.time() - prev
5     if(time_elapsed > dt):
6         # print('Elapsed time {}'.format(time_elapsed))
7         prev=time.time()
8         # grab the current frame from VideoStream
9         frame = vs.read()
10        # resize the frame (so we can process it faster) and grab the frame ←
11        # dimensions
12        frame = imutils.resize(frame, width=500)
13        (H, W) = frame.shape[:2]

```

En esta primera parte calculamos el tiempo que ha pasado desde la última iteración, si ha pasado el tiempo de un periodo ejecutamos un ciclo de cálculo. En este ciclo se resetea el contador de tiempo desde última ejecución. Capturamos un frame de la cámara y le cambiamos el tamaño para poder procesar más rápidamente la zona de interés.

Código 7.6 Código python para la visualizar y crear un bounding-box.

```

1 # check to see if we are currently tracking an object
2 if initBB is not None:
3     # grab the new bounding box coordinates of the object
4     (success, box) = tracker.update(frame)
5     # check to see if the tracking was a success

```

```

6     if success:
7         (x, y, w, h) = [int(v) for v in box]
8         cv2.rectangle(frame, (x, y), (x + w, y + h),
9             (0, 255, 0), 2)
10        cv2.line(frame, (int(x+w/2), int(y+h/2)), (int(W/2), int(H/2)),
11             (0,255,0), 2)
12
13    ##### CONTROL #####
14    # Calculate p error
15    epx = (x+w/2)-(W/2)
16    epy = (y+h/2)-(H/2)
17
18    # Calculate i error
19
20    eix = eix + epx*dt
21    eiy = eiy + epy*dt
22
23    # Calculate d error
24    edx = (epx-epx_1)/dt
25    edy = (epy-epy_1)/dt
26
27    #print(' Errx:{} Erry:{} '.format(p_error_x,p_error_y))
28    #print(' iErrx:{} iErry:{} '.format(i_error_x,i_error_y))
29    #print(' dErrx:{} dErry:{} '.format(d_error_x,d_error_y))
30
31    # Calculate control signal
32    u_x = -(kp*epx + ki*eix + kd*edx)
33    u_y = (kp*epy + ki*eiy + kd*edy)
34
35    # Send to microcontroller
36    ser.write(sendPacket(0,u_x,u_y))
37
38    epx_1 = epx
39    epy_1 = epy
40 else:
41     # Nothing is being tracked
42     # Stop any controller variables or control signals
43     u_x = 0
44     u_y = 0
45     ser.write(sendPacket(0,0,0))
46 print("Lost target!")

```

Esta segunda parte sólo se ejecuta si hemos definido un bounding box inicial de nuestra zona de interés. Una vez definida esta zona de interés podremos calcular la nueva posición de la zona de interés con `tracker.update`. `Tracker.update` nos devolverá si ha sido existoso o si ha perdido de vista la zona de interés. Si ha sido existoso se calcula entonces los errores ep, ei, ed con la distancia desde el centro del objetivo de interés y el centro de la imagen de la cámara. Se calcula con la ley de control la nueva señal de control y se manda por puerto serie usando la función que hemos definido anteriormente. En caso de que se haya perdido el objetivo de vista, paramos los motores mandando comandas de velocidad zero.

Código 7.7 Código python para la visualizar y crear un bounding-box.

```

1  # update the FPS counter
2  # fps.update()
3  # fps.stop()
4  # initialize the set of information we'll be displaying on
5  # the frame
6  info = [
7      ("Tracker", "kcf"),
8      ("Success", "Yes" if success else "No"),

```

```

9      ("FPS", "{:.5f}".format(time_elapsed)),
10     #("Control signals x:{} y:{}".format(x,y)),
11   ]
12 # loop over the info tuples and draw them on our frame
13 for (i, (k, v)) in enumerate(info):
14   text = "{}: {}".format(k, v)
15   cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
16             cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
17   # show the output frame

```

En esta tercera parte imprimimos por pantalla la información relevante tal como el tiempo entre ciclos, el estado del tracker, y el tracker usado.

Código 7.8 Código python para la visualizar y crear un bounding-box.

```

1  cv2.imshow("Frame", frame)
2  key = cv2.waitKey(1) & 0xFF
3  # if the 's' key is selected, we are going to "select" a bounding
4  # box to track
5  if key == ord("s"):
6    # select the bounding box of the object we want to track (make
7    # sure you press ENTER or SPACE after selecting the ROI)
8    initBB = cv2.selectROI("Frame", frame, fromCenter=False,
9      showCrosshair=True)
10   # start OpenCV object tracker using the supplied bounding box
11   # coordinates, then start the FPS throughput estimator as well
12   tracker.init(frame, initBB)
13   fps = FPS().start()
14   # if the `q` key was pressed, break from the loop
15   elif key == ord("q"):
16     break
17
18 # release the pointer
19 vs.stop()
20
21 # close all windows
22 cv2.destroyAllWindows()

```

Por último en esta parte se maneja la inicialización del bounding box de la zona de interés y la terminación del programa. Esto lo hacemos leyendo las teclas, si pulsamos la tecla "s" entraremos en la configuración de la zona de interés y podremos usar el ratón para seleccionar dicha zona, una vez contento con la selección si le damos al teclado "espacio" se aceptará dicha zona y se inicializará el tracker empezando con esta zona de interés. Si en cualquier momento le damos a la tecla "q" terminamos el programa.

7.4 Resultados

Los resultados de dicho control se ven en la figura Figura 7.4. En la primera gráfica vemos la posición del objeto que está localizando el tracker y en la segunda gráfica se ve el error cometido que en este caso es la distancia del objeto al centro de la cámara. En este experimento se ha desplazado de un lado a otro un objeto seleccionado, vemos que al mover el objeto el error cometido se incrementa y el controlador empieza a compensar.

Para visualizar mejor los resultados del proyecto vease también los videos en los links siguientes:

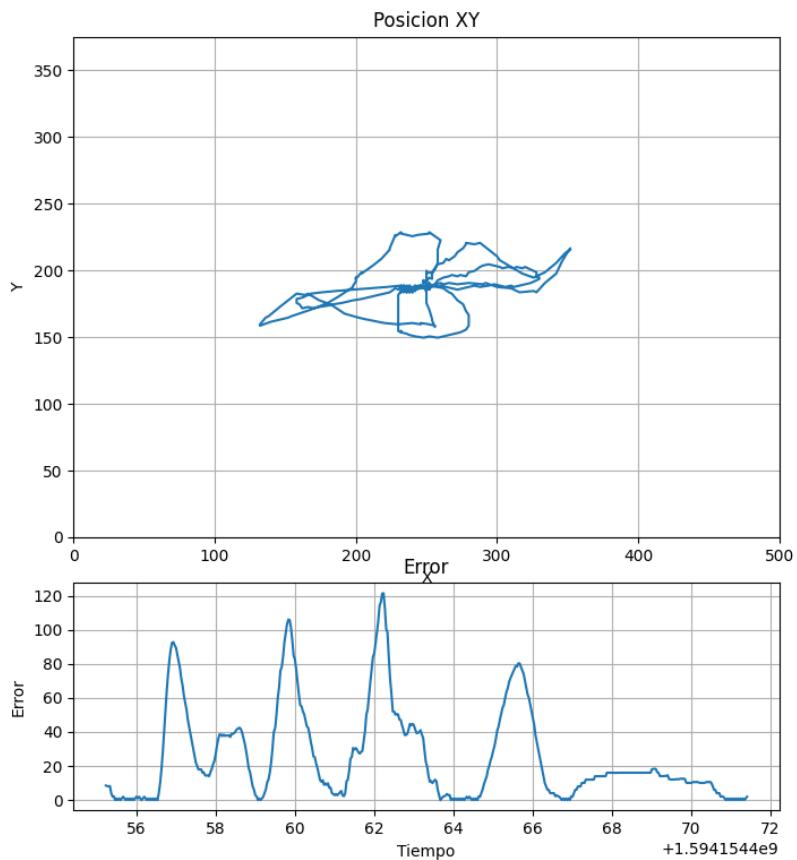


Figura 7.4 Resultados.

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)

$$x_1 = x_2. \quad (\text{A.1})$$

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:

$$x_1 = x_2 \quad (\text{A.2})$$

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

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,

$$x_1 = x_2, \quad (\text{B.1})$$

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 perdida 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 descontrulado 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.

```
1 #include <Arduino.h>
2
3 // #define debug_com
4 #define debug_control_msg
5 // #define debug_motor_data
6
7 #define T1_FREQ 656250
8 #define MIN_DELAY 200
9 #define TIMEOUT 500
10
11 #define PIN_2_DIR 55 // PORTA Bit 24
12 #define PIN_1_ENABLE 56 // PORTA Bit 23
13 #define PIN_2_STEP 54 // PORTA Bit 16
14 #define PIN_0_MIN 18 // PORTA Bit 11
15 #define PIN_0_MAX 19 // PORTA Bit 10
16 #define PIN_1_STEP 60 // PORTA Bit 3
17 #define PIN_1_DIR 61 // PORTA Bit 2
18
19 #define PIN_2_MAX 2 // PORTB Bit 25
20 #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; //! Period of next timer delay. At start ←
77     // this value set the acceleration rate c0.
78     volatile uint32_t min_delay; //! Minimum time delay (max speed)
79
80     // Interrupt handler
81     Tc *tc;
82     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 counter we set the TC_CCR_SWTRG (Software trigger) bit in ←
104     // the TC_CCR
105     tc->TC_CHANNEL[channel].TC_CCR |= TC_CCR_SWTRG;
106 }
107
108 void configureTimer(Tc *tc, uint32_t channel, IRQn_Type irq, uint32_t ←
109     frequency)
110 {
111     // Unblock the power management controller
112     pmc_set_writeprotect(false);
113     pmc_enable_periph_clk((uint32_t)irq);
114     // Configure
115     TC_Configure(tc,           // Timer
116                  channel,        // Channel
117                  TC_CMRR_WAVE | // Wave form is enabled
118                  TC_CMRR_WAVSEL_UP_RC |
119                  TC_CMRR_TCCLKS_TIMER_CLOCK4 // Settings
120 );
121
122     uint32_t rc = VARIANT_MCK / 128 / frequency; // 128 because we selected ←
123     // TIMER_CLOCK4 above
124
125     // enable timer interrupts on the timer
126     tc->TC_CHANNEL[channel].TC_IER = TC_IER_CPCS; // IER = interrupt enable←
127     // register // Enables the RC compare register.
128     tc->TC_CHANNEL[channel].TC_IDR = ~TC_IER_CPCS; // IDR = interrupt ←
129     // disable register // Disables the RC compare register.
130
131     // To reset a counter we set the TC_CCR_SWTRG (Software trigger) bit in ←
132     // the TC_CCR
133     tc->TC_CHANNEL[channel].TC_CCR |= TC_CCR_SWTRG;
134 }
135
136 void TC3_Handler()
137 {

```

```

138     TC_GetStatus(axes[0].tc, axes[0].channel); // Timer 1 channel 0 -----> ←
139         TC3 it also clear the flag
140     digitalWrite(axes[0].step_pin, HIGH);
141     digitalWrite(axes[0].step_pin, LOW);
142     axes[0].step_position += axes[0].dir;
143     // axes[0].tc->TC_CHANNEL[axes[0].channel].TC_RC = axes[0].step_delay ;
144 }
145
146 void TC4_Handler()
147 {
148     TC_GetStatus(axes[1].tc, axes[1].channel); // Timer 1 channel 0 -----> ←
149         TC3 it also clear the flag
150     digitalWrite(axes[1].step_pin, HIGH);
151     digitalWrite(axes[1].step_pin, LOW);
152     axes[1].step_position += axes[1].dir;
153     // axes[1].tc->TC_CHANNEL[axes[1].channel].TC_RC = axes[1].step_delay ;
154 }
155
156 void TC5_Handler()
157 {
158     TC_GetStatus(axes[2].tc, axes[2].channel); // Timer 1 channel 0 -----> ←
159         TC3 it also clear the flag
160     digitalWrite(axes[2].step_pin, HIGH);           // Step
161     digitalWrite(axes[2].step_pin, LOW);
162     axes[2].step_position += axes[2].dir; // Get motor position data
163     // axes[2].tc->TC_CHANNEL[axes[2].channel].TC_RC = axes[2].step_delay ;
164 }
165
166 void setup()
167 {
168     Serial.begin(9600);
169     SerialUSB.begin(9600);
170     delay(3000);
171
172     pinMode(PIN_0_ENABLE, OUTPUT);
173     pinMode(PIN_0_DIR, OUTPUT);
174     pinMode(PIN_0_STEP, OUTPUT);
175
176     pinMode(PIN_1_ENABLE, OUTPUT);
177     pinMode(PIN_1_DIR, OUTPUT);
178     pinMode(PIN_1_STEP, OUTPUT);
179
180     pinMode(PIN_2_ENABLE, OUTPUT);
181     pinMode(PIN_2_DIR, OUTPUT);
182     pinMode(PIN_2_STEP, OUTPUT);
183
184     delay(3000);
185
186     SerialUSB.println("Starting timer..");
187
188     configureTimer(/*Timer TC1*/ TC1, /*Channel 0*/ 0, /*TC3 interrupt ←
189         nested vector controller*/ TC3 IRQn, /*Frequency in hz*/ T1_FREQ);
190     configureTimer(/*Timer TC1*/ TC1, /*Channel 0*/ 1, /*TC3 interrupt ←
191         nested vector controller*/ TC4 IRQn, /*Frequency in hz*/ T1_FREQ);
192     configureTimer(/*Timer TC1*/ TC1, /*Channel 0*/ 2, /*TC3 interrupt ←
193         nested vector controller*/ TC5 IRQn, /*Frequency in hz*/ T1_FREQ);
194
195     // Axis 1
196     axes[0].enable_pin = PIN_0_ENABLE;
197     axes[0].step_pin = PIN_0_STEP;
198     axes[0].dir_pin = PIN_0_DIR;
199     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 at 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←
311     data packet
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←
329         // x78 byte
330         case LISTENING:
331             if (in_byte == 0x7E) // CHECK IF ITS START BYTE
332             {
333 #ifdef debug_com
334                 SerialUSB.println("\n START command read");
335 #endif
336                 // If the start byte is read then we proceed to read the ←
337                 // payload
338                 state = READ_LOAD;
339             }
340             else
341             {
342                 // Wrong start command
343 #ifdef debug_com
344                 SerialUSB.println("\nWrong START BYTE");
345 #endif
346                 // If we started listenting and the first byte wasnt a start ←
347                 // byte then we restart the listening , the packen has begun ←
348                 // without the start packet
349                 state = END_CMD; // RESET IF IT ISN'T
350             }
351             break;
352
353         // Reading payload state , payload has a fixed length of 12 anything ←
354         // else will break the packet and restart the state machine
355         case READ_LOAD:
356
357             // Put data in our structure data
358             m_payload.array[byten] = in_byte;
359             byten--;
360             if (byten < 0)
361             {
362 #ifdef debug_com
363                 SerialUSB.println("\nReceived 12 bytes");
364 #endif
365             // When we have read 12 bytes total we go to the end state
366             state = READ_END;
367             byten = 3;
368         }
369         break;

```

```

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("\nReceived 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 predefined end command then we ←
380             // 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                 state = END_CMD;
395             }
396         }
397         break;
398     }
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

1.1	1
1.2	3
2.1	5
2.2	6
2.3	7
2.4	7
2.5	8
2.6	8
2.7	9
2.8	9
2.9	10
2.10	11
3.1	13
3.2 Motor de pasos NEMA-17	14
3.3 Ilustracion microstepping	15
3.4 Ilustracion vector magnetico resultante	15
3.5 Circuito puente H	15
3.6 Digrama de corrientes	16
3.7 Digarama temporal de señales PWM	16
3.8 Driver pololu A4988	17
3.9 Esquematico del driver A4988	17
3.10 Guia de cableado	17
3.11 Pantalla NEXTION	18
3.12 Sensor montado en eje AS504X	19
3.13 Sensor montado en eje AS504X	19
3.14 PCB para el sensor AS504X	19
3.15 Arduino DUE	20
3.16 Placa de adaptación	21
4.1	23
4.2 Serie vs Paralelo	24
4.3 ilustración de comunicación serie	24
4.4 Análisis de una señal TX del microcontrolador DUE	25
4.5 Máquina de estados del protocolo de comunicación	26
5.1	29
5.2	30
6.1 Reconocimiento y tracking de movimiento de vehículos	35
6.2 Esquema general del funcionamiento de KCF	37

6.3	OpenCV	37
6.4	Logo OpenCV	37
6.5	Logo python	38
6.6	Hardware y software	38
6.7	Resutados de selección de objetivos y tracking	40
6.8	Resutados de selección de objetivos y tracking	41
7.1		43
7.2	Esquema de un PID	44
7.3	Método de integración numérica	45
7.4	Resultados	50

Índice de Tablas

3.1	Tabla de configuracion <i>micro-stepping</i>	18
5.1	Tabla de interrupciones	31

Índice de Códigos

4.1	Código manejador de recepción de comandos	26
5.1	Definición de pines de la placa de adaptación	30
5.2	Configuración de un timer	31
5.3	Arranque del timer	32
5.4	Parada del timer	32
5.5	Manejador de interrupción	32
6.1	Código python para la visualizar y crear un bounding-box	38
6.2	Código python para la visualizar y crear un bounding-box	38
6.3	Código python para la visualizar y crear un bounding-box	39
6.4	Código python para la visualizar y crear un bounding-box	39
6.5	Código python para la visualizar y crear un bounding-box	40
7.1	Código python para la visualizar y crear un bounding-box	45
7.2	Código python para la visualizar y crear un bounding-box	46
7.3	Código python para la visualizar y crear un bounding-box	46
7.4	Código python para la visualizar y crear un bounding-box	47
7.5	Código python para la visualizar y crear un bounding-box	47
7.6	Código python para la visualizar y crear un bounding-box	47
7.7	Código python para la visualizar y crear un bounding-box	48
7.8	Código python para la visualizar y crear un bounding-box	49
A.1	Escritura de una ecuación	51
	../../../../MotorDriver/src/main.cpp	53

Índice alfabético

Universal Asynchronous Transmitter, 24	Receiver half-step hyperpage, 14	Start bit, 24
Arduino DUE, 20	Matlab, 37	Step, 30, 33
capture, compare, 20	micro-stepping, 18	Stop bit, 24
detent hyperpage, 14	micro-stepping hyperpage, 14	TMC22XX, 16
Dir, 30	micro-steps, 17	Trinamic, 16
duty-cycle hyperpage, 16	NEMA 17 hyperpage, 14	UART, 24
Enable, 30	NEMA hyperpage, 14	Visual Servoing hyperpage, 1
full-step hyperpage, 14	pan-tilt, 1	xOUT1, 16
	pan-tilt hyperpage, 13, 18	xOUT2, 16
	paso a paso hyperpage, 13	
	Pololu, 16	

