



Universidad Nacional de Quilmes

Departamento de Ciencia y Tecnología
Ingeniería en Automatización y Control Industrial

CONTROL AUTOMÁTICO DEL EQUIPO UPDOWN

Olivieri, Ian Paulo

Director:

Pernia, Eric Nicolás

Jurado:

Juarez, José
Oliva, Damián
Safar, Felix

Presentación: Septiembre de 2017
Quilmes, Buenos Aires, Argentina.

Resumen

Resumen del proyecto (1 carilla)

Índice de contenidos

1 Introducción	10
1.1 Marco temático	10
1.1.1 Esculturas cinéticas	10
1.1.1.1 Definición	10
1.1.1.2 Aplicaciones y estado actual del arte	11
1.1.2 Sistemas de iluminación	12
1.1.2.1 Equipos de luces	12
1.1.2.2 Consolas de control de luminaria	13
1.2 DMX	14
1.2.1 Definición e historia	14
1.2.2 Capa física	14
1.2.2.1 Cableado y conectores	14
1.2.2.2 Topología	15
1.2.2.3 Señal	15
1.2.3 Capa de enlace de datos	15
1.2.3.1 Subcapa de control de enlace lógico	15
1.2.3.2 Subcapa de control de acceso al medio	17
1.3 Updown	17
1.3.1 Definición	17
1.3.2 Descripción del sistema	18
1.3.2.1 Fuente de alimentación	18
1.3.2.2 Entrada y salida DMX	18
1.3.2.3 Dip-switch	19
1.3.2.4 Freno	19
1.3.2.5 Fin de carrera	19

1.3.2.6	Sistema motor	19
1.3.2.7	Placa de control	20
1.3.3	Resumen de entradas y salidas del sistema	20
1.3.3.1	Entradas	20
1.3.3.2	Salidas	20
1.4	Justificación del proyecto	21
1.5	Objetivos	21
1.5.1	REQ-01	21
1.5.2	REQ-02	21
1.5.3	REQ-03	21
1.5.4	REQ-04	22
2	Diseño	23
2.1	Descripción del capítulo	23
2.2	REQ-01	23
2.3	REQ-02	24
2.3.1	Esquema de control	24
2.3.2	Modelo de la planta	25
2.3.3	Procedimiento para el diseño de control	26
2.3.4	Relación entre cuentas de encoder y distancia	26
2.3.5	Velocidad máxima	27
2.4	REQ-03	27
2.4.1	Corte de correa	27
2.4.2	Fin de carrera	28
2.4.3	Pérdida de DMX	28
2.5	REQ-04	28
2.6	Firmware del updown	29

2.6.1	Aviso de confidencialidad	29
2.6.2	Convenciones	29
2.6.3	Librerías de bajo nivel	30
2.6.4	Librerías de alto nivel	31
2.6.5	Diagrama de módulos	32
3	Desarrollo	33
3.1	Descripción del capítulo	33
3.2	Firmware del updown - Librerías de bajo nivel	33
3.2.1	DigitalIO	33
3.2.1.1	Objetivo	33
3.2.1.2	Desarrollo	33
3.2.1.3	Acceso	34
3.2.2	ADC	35
3.2.2.1	Objetivo	35
3.2.2.2	Desarrollo	36
3.2.2.3	Acceso	37
3.2.3	PWM	38
3.2.3.1	Objetivo	38
3.2.3.2	Desarrollo	38
3.2.3.3	Acceso	40
3.2.4	EXINT	40
3.2.4.1	Objetivo	40
3.2.4.2	Desarrollo	41
3.2.4.3	Acceso	41
3.2.5	UART	41
3.2.5.1	Objetivo	41

3.2.5.2	Desarrollo	42
3.2.5.3	Acceso	43
3.2.6	Tick	44
3.2.6.1	Objetivo	44
3.2.6.2	Desarrollo	44
3.2.6.3	Acceso	45
3.2.7	SUART	45
3.2.7.1	Objetivo	45
3.2.7.2	Desarrollo	46
3.2.7.3	Acceso	47
3.3	Controlador	48
3.3.1	Relación entre cuentas de encoder y distancia	48
3.3.1.1	Pruebas y resultados	48
3.3.1.2	Selección de la función de ajuste	48
3.3.2	Determinación de la velocidad máxima	50
3.3.3	Obtención del período de muestreo	51
3.3.4	Obtención del modelo de la planta	52
3.3.4.1	Toma de muestras	52
3.3.4.2	Modelos propuestos	52
3.3.4.3	Determinación de parámetros	55
3.3.4.4	Validación de los modelos	56
3.3.4.5	Ánálisis de resultados	56
3.3.5	Obtención del controlador	58
3.3.5.1	Ánálisis	58
3.3.5.2	Controlador de velocidad	58
3.3.5.3	Controlador de posición	59
3.3.6	Ajuste de los controladores	59

3.3.6.1	Condiciones iniciales	59
3.3.6.2	Ajuste del controlador de velocidad	59
3.3.6.3	Ajuste del controlador de posición	60
3.3.6.4	Sistema de control final	61
3.4	Dipswitch	61
3.5	Firmware del updown - Librerías de alto nivel	64
3.5.1	Encoder	64
3.5.1.1	Objetivo	64
3.5.1.2	Desarrollo	64
3.5.1.3	Acceso	67
3.5.2	Grua	67
3.5.3	Controlador	68
3.5.3.1	Objetivo	68
3.5.3.2	Seteo de referencias	69
3.5.3.3	Cálculo de los controladores	70
3.5.3.4	Acceso	71
3.5.4	Dipswitch	71
3.5.4.1	Objetivo	71
3.5.4.2	Desarrollo	72
3.5.4.3	Acceso	73
3.5.5	DMX	73
3.5.5.1	Objetivo	73
3.5.5.2	Desarrollo	73
3.5.5.3	Acceso	75
3.6	Firmware del updown - Capa de Aplicación	76
3.6.1	Objetivo	76
3.6.2	Desarrollo	76

3.6.2.1	Inicialización del sistema	76
3.6.2.2	Planificador y despachador de tareas	76
3.6.2.3	Bucle principal	78
4	Validación	81
4.1	Procedimiento	81
4.2	Procedimiento	81
5	Conclusiones	83
5.1	Conclusiones	83
5.2	Mejoras a futuro	83
5.3	Referencias utilizadas	83

Índice de figuras

1.1	Ejemplo de escultura cinética movida por aire, por Anthony Howe. Fuente: LINK al video	10
1.2	Escultura cinética en el museo BMW. Fuente: LINK al video	11
1.3	Escultura cinética por parte de Build Up. Fuente: LINK al video	12
1.4	Shapeshifter, de High End Systems. Fuente: LINK al video	12
1.5	Consola Hog4, de High End Systems. Fuente: LINK a la imagen	13
1.6	Cable DMX con conector XLR5. Fuente: wikipedia	14
1.7	Conexionado en una red DMX. Fuente: wikipedia	15
1.8	Formato de la trama DMX. Fuente: LINK	16
1.9	Imagen del Updown, desarrollado por la empresa Blackout	17
1.10	Diagrama conceptual del equipo Updown	18
1.11	Diagrama del sistema motor	19
1.12	Diagrama de la placa de control y su interacción con el hardware externo	20
2.1	Esquema de control a implementar	25
2.2	Modelo mecánico de la planta	26
2.3	Diagrama de uno de los 3 divisores de tensión del dipswitch	29
2.4	Diagrama de módulos del firmware	32
3.1	Diagrama del módulo DigitalIO	35
3.2	Diagrama del módulo ADC	37
3.3	Ejemplo variación ciclo de trabajo al cambiar ICR1	39
3.4	Diagrama del módulo PWM	40
3.5	Diagrama del módulo EXINT	42
3.6	Diagrama del módulo UART	43
3.7	Diagrama del módulo Tick	45

3.8	Forma de la señal en el canal para una 'p' con formato 8N1	46
3.9	Diagrama del módulo SUART	47
3.10	Ajuste mediante función cuadrática	49
3.11	Ajuste mediante una única recta	50
3.12	Ajuste mediante una recta cada 25cm	51
3.13	Ajuste mediante una recta cada 100cm	52
3.14	Respuestas del sistema en bajada	53
3.15	Respuestas del sistema en subida	54
3.16	Entrada del sistema para la identificación de parámetros	54
3.17	Salida de velocidad del sistema	55
3.18	Comparación entre datos predichos con el modelo 2 y los datos reales . .	57
3.19	Resultados para el controlador de velocidad inicial	60
3.20	Resultados del equipo 1 para el controlador de velocidad final	61
3.21	Resultados del equipo 2 para el controlador de velocidad final	62
3.22	Resultados para el controlador de posición inicial	63
3.23	Resultados para el controlador de posición final	64
3.24	Ejemplo estado de los canales AB del encoder de disco	66
3.25	Diagrama del módulo Encoder	67
3.26	Diagrama del módulo Grua	68
3.27	Diagrama del módulo Controlador	72
3.28	Diagrama del módulo Dipswitch	73
3.29	Maquina de estados ejecutada en cada interrupción de la UART	74
3.30	Diagrama del módulo DMX	75
3.31	Máquina de estados ejecutada en el bucle principal de la aplicación	79
4.1	Diagrama del sistema motor	82

Introducción

1.1 Marco temático

1.1.1 Esculturas cinéticas

1.1.1.1 Definición

Las esculturas cinéticas (kinetic sculpture en inglés) son estructuras tridimensionales en donde el movimiento es una parte fundamental del conjunto. Para lograr el efecto de movimiento en el espacio estos sistemas se construyen con partes móviles que pueden cambiar de posición ya sea naturalmente por acción del viento, como se ve en la figura 1.1, o de manera forzada.



Figura 1.1: Ejemplo de escultura cinética movida por aire, por Anthony Howe. Fuente: [LINK al video](#)

1.1.1.2 Aplicaciones y estado actual del arte

Al ser obras que caen dentro del campo artístico suelen presentarse en museos y utilizarse para fines decorativos ya sea en parques o eventos. Sin embargo, el nivel de ingeniería y diseño que algunas de ellas requieren las tornan un interesante desafío intelectual y creativo.

Las aplicaciones puntuales de estructuras cinéticas a las que se hará foco en este informe, debido a la naturaleza del proyecto final, son aquellas en donde el efecto espacial se logra a través del movimiento en el eje vertical de objetos esféricos mediante motores.

Un ejemplo de aplicación de estas características se puede ver en la figura 1.2. Allí se muestra una escultura presentada en el Museo de BMW, en Munich, Alemania, en donde 714 esferas metálicas son coordinadas para formas figuras como olas, gotas, y hasta la silueta de un auto.

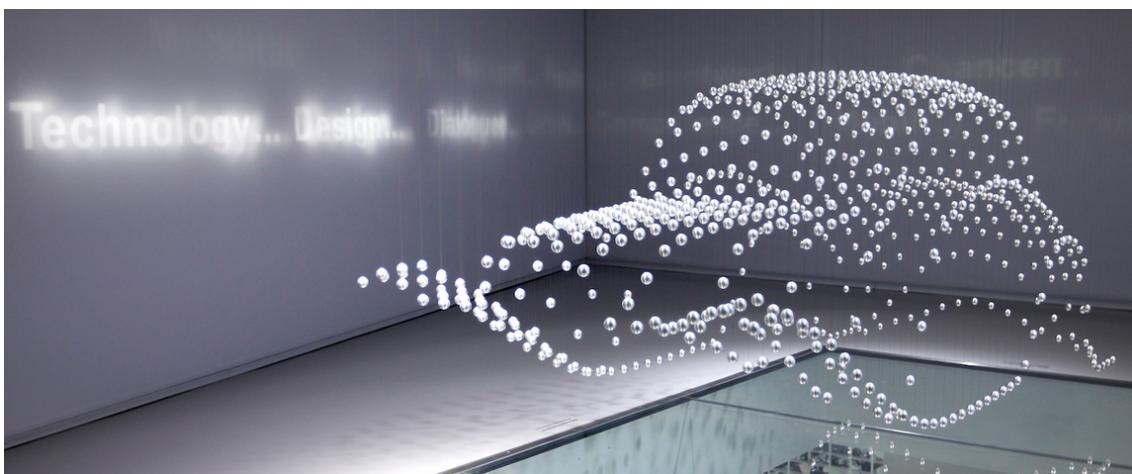


Figura 1.2: Escultura cinética en el museo BMW. Fuente: [LINK al video](#)

Otro ejemplo de aplicación se puede ver en la figura 1.3, en una obra presentada por la empresa Build Up en un centro comercial en Fukuoka, Japón. Allí se instalaron 1000 luminarias esféricas RGB dispuestas en una matriz de 25x40 para generar figuras tridimensionales como planos y gausseanas, entre otras. En este caso los efectos espaciales se logran coordinando el movimiento de cada esfera independientemente, cada una manejada por un equipo motorizado.

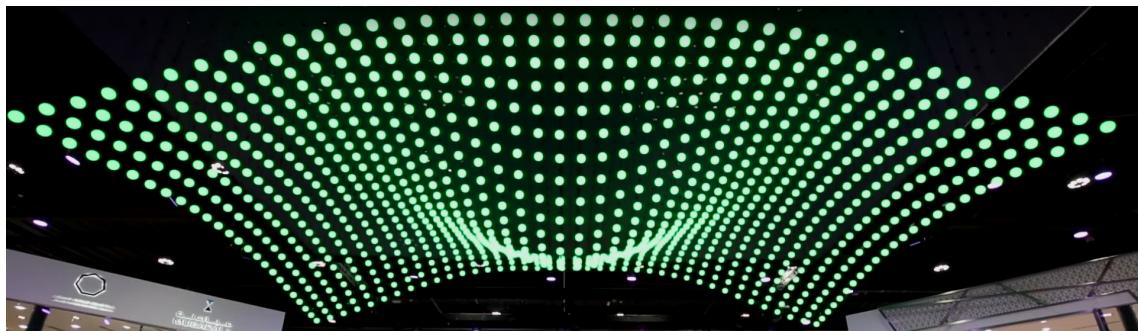


Figura 1.3: Escultura cinética por parte de Build Up. Fuente: [LINK al video](#)

1.1.2 Sistemas de iluminación

1.1.2.1 Equipos de luces

En cualquier espectáculo o evento la iluminación es una parte vital del show, y a medida que estos fueron evolucionando también lo hicieron los equipos de luces. Partiendo de aparatos fijos en donde solo se podía variar la intensidad de luz, se pueden conseguir hoy en día dispositivos complejos con decenas de parámetros controlables.

Un notable ejemplo es el **Shapeshifter**, figura 1.4, que cuenta con 7 módulos leds que pueden ser manejados independientemente.



Figura 1.4: Shapeshifter, de High End Systems. Fuente: [LINK al video](#)

En el caso del sistema visto en la figura 1.3, los parámetros controlables de los equipos son la posición, velocidad y colores de cada esfera.

1.1.2.2 Consolas de control de luminaria

Para controlar los sistemas de luces es necesario utilizar unas consolas especiales. Estas se comunican con las luminarias utilizando el estándar **DMX** y le indican a cada equipo el valor de sus parámetros en todo momento.

La manera más común para generar un efecto es indicando la progresión de uno o más parámetros desde un tiempo inicial a uno final. Al cambio de los parámetros entre 2 instantes de tiempo se las llama *cues*, o entradas, y cuyo conjunto forma los efectos. Dentro de las consolas que hay en el mercado para este tipo de control de equipos se pueden destacar las **consolas hog 4** de High End Systems, como la que se muestra en la figura 1.5.

Otra manera generarlos es a partir de equipos y softwares, como el **Madrix**, que tienen la capacidad de convertir videos a variaciones de parámetros, lo cual lo hace especialmente útil cuando se quieren crear **efectos lumínicos complejos**.



Figura 1.5: Consola Hog4, de High End Systems. Fuente: [LINK a la imagen](#)

1.2 DMX

1.2.1 Definición e historia

DMX, de *Digital MultipleX*, es un estándar de comunicación digital ampliamente utilizado para el control de sistemas de iluminación.

El estándar DMX512, donde 512 significa que se envían 512 piezas de información, fue creado por la *United States Institute for Theatre Technology* (USITT) en 1986 y transformado en DMX512/1990 tras una revisión de la USITT. En 1998 la *Entertainment Services and Technology Association* (ESTA) cuadró DMX dentro de los estándares ANSI, modificación que fue aprobada por el instituto (ANSI) en 2004. Finalmente, en 2008 DMX tuvo una nueva revisión y se llegó a la versión actual llamada "E1.11 – 2008, USITT DMX512-A", o simplemente DMX512-A. A pesar de esto, el nombre comúnmente conocido del estándar es simplemente DMX, aunque no es indistinto ya que hay diferencia de compatibilidad entre las diferentes versiones.

1.2.2 Capa física

1.2.2.1 Cableado y conectores

DMX emplea el estándar EIA-485 como capa física, por lo que emplea por lo menos 3 líneas; A, B y C, en donde A y B son los datos que se transmiten, y C es masa. Los conectores utilizados son los XLR, tanto de 5 como de 3 pinos. Un ejemplo del cable se puede ver en la figura 1.6



Figura 1.6: Cable DMX con conector XLR5. Fuente: wikipedia

1.2.2.2 Topología

La red de DMX consiste en un maestro y varios esclavos, conectados con una topología de bus multidrop (MDB) con nodos conectados entre sí, lo que normalmente se denomina como topología *daisy chain*. En otras palabras, todos los equipos a controlar tienen una entrada y una salida conectadas entre sí, de manera tal de que se puede conectar un equipo y apartir de este equipo conectar el siguiente, y así sucesivamente, como se ve en la figura 1.7. Esto permite que el conexionado sea simple y que la red pueda ser fácilmente extendida.

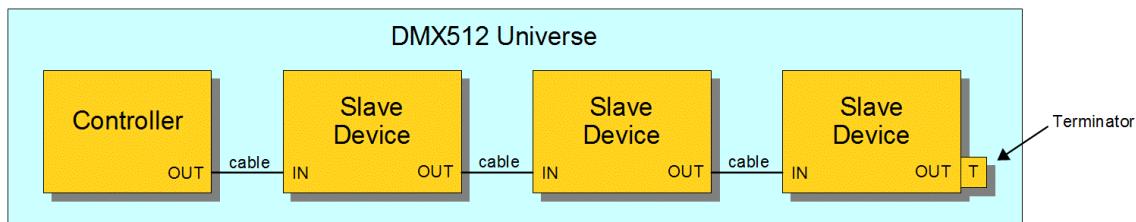


Figura 1.7: Conexión en una red DMX. Fuente: wikipedia

1.2.2.3 Señal

La señal de DMX es de tipo diferencial, como es indicado en EIA-485, de 5 Volts de pico, y los datos se envían asincrónicamente de manera serie con una tasa de transmisión de 250Kbits por segundo que equivale a una duración de bit de $4\mu s$.

1.2.3 Capa de enlace de datos

1.2.3.1 Subcapa de control de enlace lógico

La trama DMX, visible en la figura 1.8 consta de las siguientes partes:

- **Idle** y **MTBP** (Mark Time Between Packets): estado de HIGH en la línea que indica la ausencia de señal de DMX. En caso de que se supere el tiempo máximo de 1 segundo sin datos, se considera que hubo una pérdida en la conexión.
- **Break**: estado de LOW en la línea utilizado para separar las tramas entre sí.
- **MAB** (Mark After Break): estado en HIGH enviado luego de un Break. Este estado suele traer problemas de compatibilidad ya que fue cambiado de una duración de $4\mu s$ a $8\mu s$ en la versión de DMX512 de 1990.

- **Slots:** son datos con formato 8N2 (un bit de start (LOW), 8 bits de datos, 2 bits de stop (HIGH) y sin paridad), y pueden ser:
 - **SC** (Start Code): es el Slot 0, enviado luego de un MAB, para indicar el principio del payload. Todos los bits de datos equivalen a LOW en este estado.
 - **Channels:** contienen la información que quiere ser enviada a los equipos de DMX. El conjunto de los 8 bits de datos pueden tomar un valor de 0 a 255. En total pueden haber un máximo de 512 canales enviados.
- **MTBF** (Mark Time Between Frames): estado opcional de HIGH en la linea que puede agregarse antes del bit de start de cada canal.

Señal	Mínima	Máxima	Típica
Idle/MTBP	0	1 seg	No especificada
Break	88μs	1seg	88μs
MAB	-	-	8μs
Bit	-	-	1/250KHz = 4μs
Slot (11 bits)	-	-	44μs
MTBF	0	1 seg	No especificada

Tabla 1.1: Duración de las señales que componen la trama de DMX

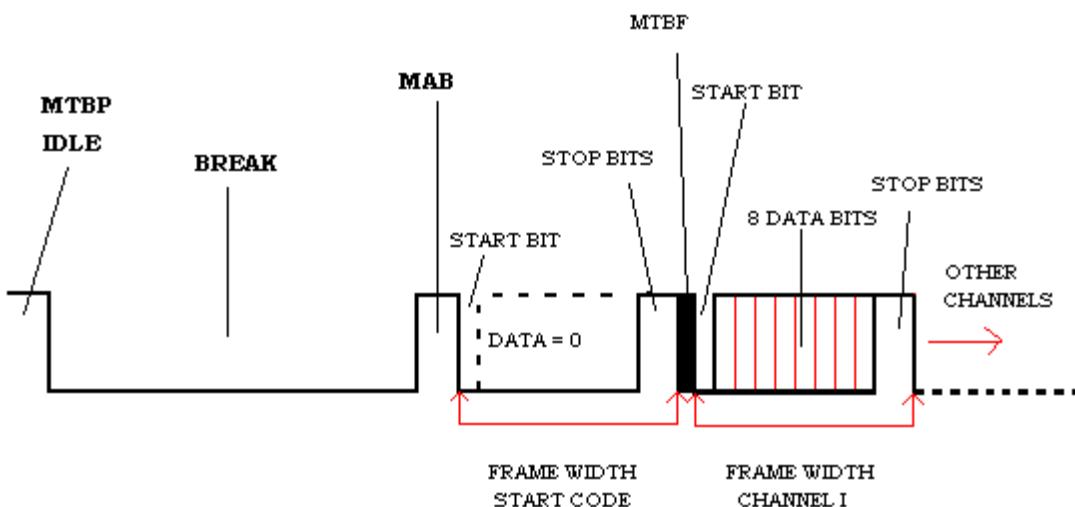


Figura 1.8: Formato de la trama DMX. Fuente: [LINK](#)

A cada trama con 512 canales se la llama "universo de DMX". Si se necesitan enviar más de 512 canales se utilizan más universos.

1.2.3.2 Subcapa de control de acceso al medio

DMX no implemente ningún mecanismo de control de acceso al medio debido a que el único transmisor es la red es el maestro y todos los esclavos reciben la misma información.

1.3 Updown

1.3.1 Definición

El **Updown**, que se puede ver en la figura 1.9, es básicamente una grúa que sube y baja una carga conforme a comandos recibidos por una equipo que utilice el protocolo DMX, como puede ser una consola de control de luminaria. La distancia máxima a la que la carga puede bajar es de 4 metros.

Este producto fue concebido en **Blackout**, una empresa productora y proveedora de tecnología cuyo objetivo es generar contenido audiovisual para grandes eventos. Blackout tomó interés en esculturas cinéticas como la de la figura 1.3 pero se encontró con el problema que el costo de importación de los equipos utilizados para tales fines, sumado a su precio unitario, era muy elevado. Por este motivo, decidió comenzar el desarrollo de un producto propio y nacional para alcanzar su objetivo.

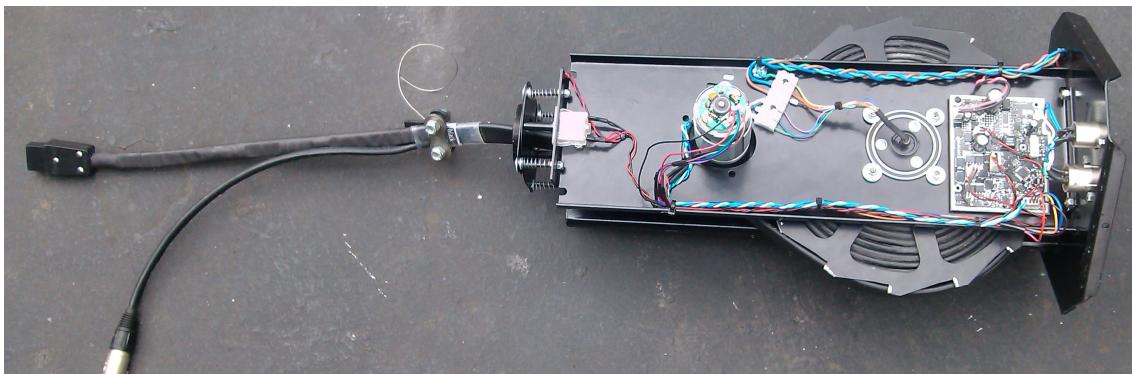


Figura 1.9: Imagen del Updown, desarrollado por la empresa Blackout

1.3.2 Descripción del sistema

En la figura 1.10 se presenta el diagrama general del equipo updown. Allí se pueden identificar todos elementos electrónicos y mecánicos que conforman el sistema, los agentes externos del sistema, y la comunicación entre ellos.

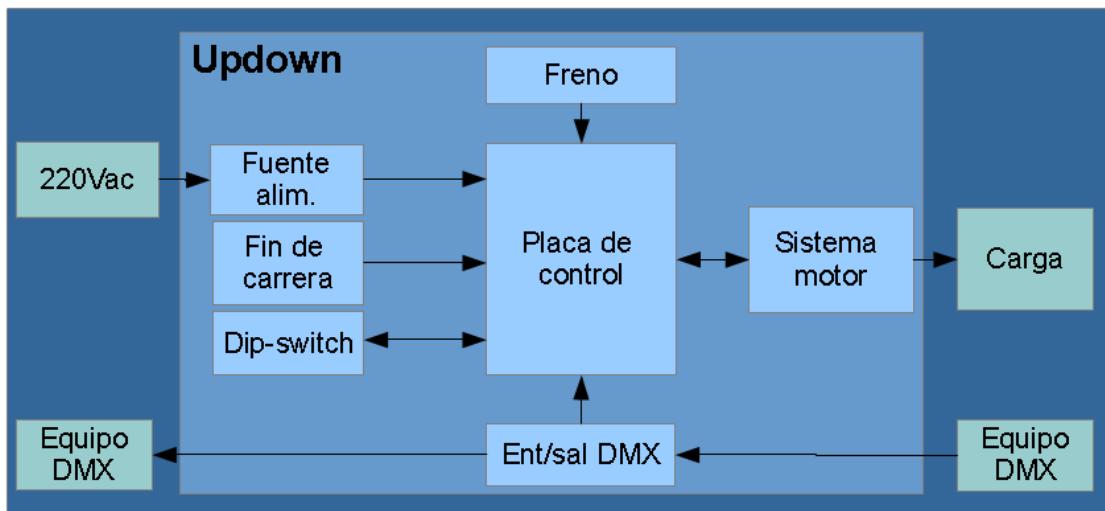


Figura 1.10: Diagrama conceptual del equipo Updown

1.3.2.1 Fuente de alimentación

La fuente de alimentación es la encargada de proveer la potencia necesaria para el funcionamiento del sistema motor, y de placa de control junto con sus periféricos. Esto lo logra convirtiendo la tensión de red de 220Vac a una señal continua de 24Vcc, entregando 4.5A máximo.

1.3.2.2 Entrada y salida DMX

La señal DMX proveniente del master de la red, otro Updown, o cualquier otro equipo esclavo dentro del mismo universo DMX ingresa al sistema para ser procesada por la placa de control. Su función es indicarle al equipo la referencia de posición y velocidad, y los parámetros que correspondan a la carga.

Además, para cumplir con el estándar DMX, la señal de entrada se replica mediante un puente a una salida para que otro equipo pueda ser conectado a la red.

1.3.2.3 Dip-switch

El dipswitch consta con varios divisores resistivos, 10 switchs y un led indicador. Su función es seleccionar el canal inicial de DMX con el que el equipo trabajará.

1.3.2.4 Freno

El freno consta de una bobina, desactivada por defecto, que mueve una varilla metálica que traba el carrete que contiene la polea. Al activar la bobina la varilla libera al carrete, permitiendo el libre movimiento de la carga.

1.3.2.5 Fin de carrera

El fin de carrera consta de un par de pulsadores en la base del equipo. Eventos como enrollar completamente la polea activa alguno de los pulsadores, dandole aviso a la placa de control.

1.3.2.6 Sistema motor

El elemento principal del sistema motor es un motor de continua de 24V con un encoder AB en su eje. Este motor mueve un carrete en donde se enrolla el cable que sostiene la carga, que tiene un peso de 3Kg. El carrete, llamado disco, tiene unas marcas que son sensadas por un encoder en la placa de control para tener una segunda referencia de posición. La dirección de giro del motor se selecciona a través de un puente H.

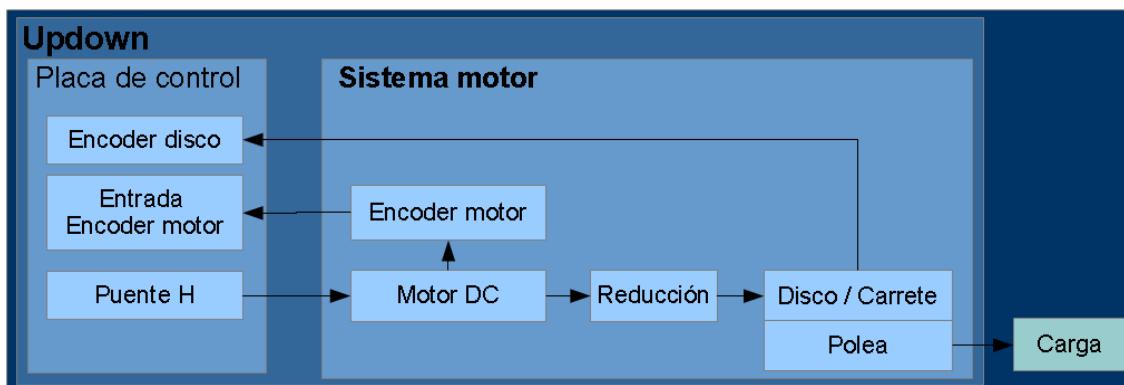


Figura 1.11: Diagrama del sistema motor

1.3.2.7 Placa de control

La placa de control contiene electrónica para el manejo de las entradas y salidas del sistema, y un controlador para manejar todos los periféricos siguiendo la lógica deseada.

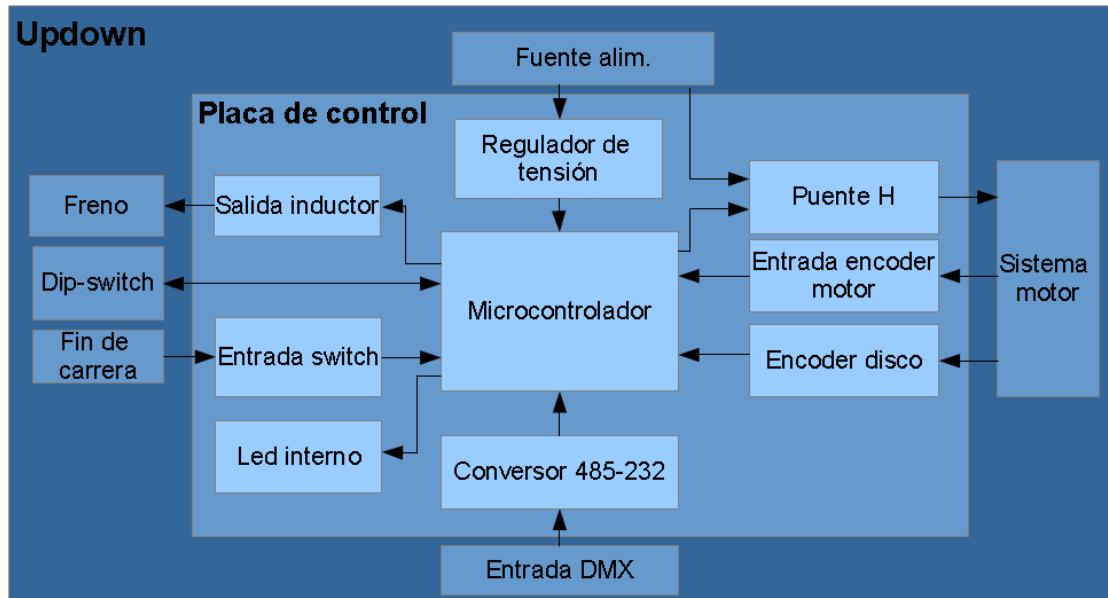


Figura 1.12: Diagrama de la placa de control y su interacción con el hardware externo

1.3.3 Resumen de entradas y salidas del sistema

1.3.3.1 Entradas

Señal de DMX, Fin de carrera, Dip-switch, Encoder AB de disco, Encoder AB de motor.

1.3.3.2 Salidas

Freno, Motor, Led interno (led indicador de la Placa de control), Led externo (led indicador del Dipswitch), Puente H.

1.4 Justificación del proyecto

Durante el desarrollo del updown la empresa se dió cuenta que con los recursos que contaban, tiempo en particular, no podían concretar los requerimientos que buscaban que el updown tuviera.

Por este motivo, Blackout se vió en la necesidad de buscar a alguien externo a la empresa para darle fin al proyecto, lo que presentó la oportunidad de realizar el trabajo de concluir el desarrollo del equipo, que es en lo que este proyecto final se basa.

1.5 Objetivos

El objetivo del proyecto es que el Updown suba y baje una carga de 3Kg, siguiendo las referencias de posición y velocidad dadas por un master DMX. Además, debe contar con ciertos mecanismos de detección y manejo de errores para hacer que el producto sea seguro. También se tiene que terminar el desarrollo del dipswitch.

Como la mayoría del hardware del equipo resuelto las tareas a cumplir para terminar el producto, y dar por concluido el proyecto, se pueden resumir en los siguientes 4 requisitos:

1.5.1 REQ-01

Contar con el software necesario para el manejo de todas las entradas y salidas del sistema.

1.5.2 REQ-02

Lograr que las cargas manejadas por los equipos se muevan a la velocidad y posición indicadas mediante una consola DMX.

1.5.3 REQ-03

Manejar errores y excepciones de hardware para lograr que el producto sea seguro, siendo que será instalado en eventos con un alto nivel de concurrencia.

Dentro de estos errores se encuentran: corte de correa, pérdida de señal DMX y accionamiento indebido del fin de carrera.

1.5.4 REQ-04

Concluir el desarrollo del dipswitch.

Diseño

2.1 Descripción del capítulo

En este capítulo se analizarán los requerimientos presentados en la sección 1.5 del capítulo 1 con el objetivo de determinar qué se planea hacer para cumplirlos.

2.2 REQ-01

Para cumplir este requerimiento se debe contar con herramientas de software que permita el manejo de: freno, leds indicadores, fin de carrera, dipswitch, encoders AB de disco y motor, motor y DMX. Además, como la placa de control no cuenta con un puerto de debug es necesario tener algún canal de comunicación alternativo, como un puerto serie por software. También será necesario temporizar ciertas partes del programa, por lo que se deberá poder manejar un timer.

El microcontrolador utilizado en la placa de control es el Atmega328p, y cuenta con periféricos para manejar todas las entradas y salidas del sistema por lo que es una buena elección para el proyecto. El único inconveniente que presenta es que es un microcontrolador de 8 bits con relativamente baja memoria y sin optimizaciones para operaciones de punto flotante.

En sistemas embebidos, la manera de manejar el hardware del microcontrolador, entre otras cosas, es a través de librerías. Estas son un conjunto de funciones o procedimientos externos a la aplicación a desarrollar que le permiten al usuario manejar ciertos aspectos del sistema de forma más fácil.

Una opción sería utilizar librerías ya desarrolladas, entre las cuales se destacan las de Arduino. Arduino, una plataforma de programación de sistemas embebidos muy popular, utiliza en sus placas microcontroladores de marca Atmel. En particular, el *Arduino UNO* utiliza el Atmega328p, por lo que existen funciones provistas por Arduino para el manejo de los periféricos del microcontrolador que se utiliza en el proyecto.

La otra opción sería hacer librerías a medida para el proyecto, desarrollando las funciones necesarias para el manejo de los periféricos que se necesiten para que se comporten exactamente como uno desea.

En la tabla 2.1 se presenta una comparación entre ambas opciones.

El mayor problema con las librerías de Arduino es que para que cuadren en proyectos grandes como estos requieren modificaciones, y las funciones provistas deben ser analizadas para verificar que no se usen elementos como delays bloqueantes ni operaciones

de punto flotante indiscriminadamente, ya que deterioran el rendimiento.

Como el objetivo es hacer que el sistema sea lo más confiable posible, y teniendo en cuenta que el tiempo de desarrollo no es un factor crítico, **se desarrollarán librerías propias para manejar los periféricos.**

	Arduino	Propia
Pros	<ul style="list-style-type: none"> - Listas para usar - Altamente testeadas - Mantenidas por una comunidad 	<ul style="list-style-type: none"> - Bajo uso de recursos - Confiabilidad - Predictibilidad
Cons	<ul style="list-style-type: none"> - Genéricas (poco optimizadas) - Necesitan modificaciones para ser útiles - Requieren análisis 	<ul style="list-style-type: none"> - Tiempo de desarrollo alto por defecto - Investigación y estudio

Tabla 2.1: Comparación entre el uso de librerías de Arduino vs el desarrollo de unas propias

2.3 REQ-02

2.3.1 Esquema de control

Para cumplir con este requerimiento la carga se debe mover en respuesta a una referencia de posición y otra de velocidad indicadas por una consola DMX. Para esto se utilizará el esquema de control mostrado en la figura 2.1, siendo: p la posición, v la velocidad, C_p el controlador de posición, C_v el de velocidad, ϵ_p y ϵ_v los errores de posición y velocidad, u la acción de control, G la planta, y rp_dmx y rv_dmx las referencias de posición y velocidad dadas por la consola DMX.

La elección de este esquema fue debido a su simpleza: un controlador de velocidad se encarga de mantener la velocidad igual a la referencia, mientras que uno de posición le indica al de velocidad que debe frenar cuando la posición objetivo está por ser alcanzada. Para evitar que el controlador de posición le indique al de velocidad una referencia mayor a la permitida, se utiliza un limitador en la referencia de velocidad. Similarmente, como la acción de control será el ciclo de trabajo del pwm que mueve al motor, se utiliza un segundo limitador para evitar que el ciclo de trabajo supere el 100 %.

La simpleza en el controlador y su modelo son un factor muy importante en este proyecto, ya que el objetivo es controlar el equipo mediante una consola DMX, y los efectos enviados por consola tienen una dinámica que el equipo debe seguir lo más fielmente posible. Introducir mucha dinámica por parte del controlador, por más que haga al sistema más estable, es indeseado en este caso.

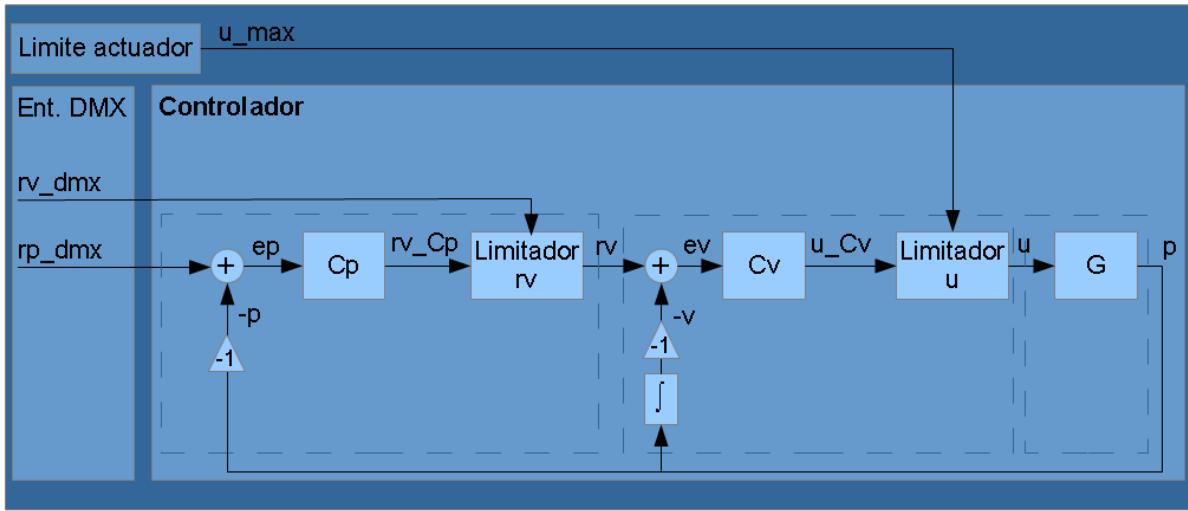


Figura 2.1: Esquema de control a implementar

2.3.2 Modelo de la planta

Para este proyecto la planta será considerada como el conjunto motor-transmisión-carga. Un esquema mecánico de la planta se muestra en la figura 2.2. Su funcionamiento es el siguiente: parte de la potencia eléctrica que ingresa al motor se transforma a mecánica y se utiliza para mover el piñón que se encuentra conectado al eje del motor. El piñón, a través de una cadena ANSI 25, transfiere su energía a la corona, que hace girar un disco adosado a ella. Este disco, o carrete, contiene en su interior el cable que sostiene a la carga, enrrollandolo o desenrrollandolo para subirla o bajarla. Como la carga del updown puede necesitar ser alimentada y recibir señal de DMX, el cable utilizado contiene en su interior contienen 2 cables con ambas señales.

Ahora, para diseñar el controlador se debe encontrar un modelo matemático de la planta. Una opción es determinar las ecuaciones diferenciales que gobiernan el sistema aplicando la segunda ley de newton, y determinando el modelo matemático de un motor de continua mediante identificación. El problema es que por motivos constructivos del equipo existen muchas piezas que generan roces en diferentes partes del recorrido de la carga, lo que quiere decir que hay una perturbación cuya dinámica es totalmente desconocida y que depende de varios factores, como la distancia y el peso de la carga. Por lo tanto, se abordará el problema con un método más simple: determinar el modelo de todo el conjunto mediante identificación, midiendo la respuesta del sistema ante ciertas entradas. Como el objetivo es que varios updown se comporten lo más parecido posible, se realizarán las pruebas para por lo menos 2 updown.

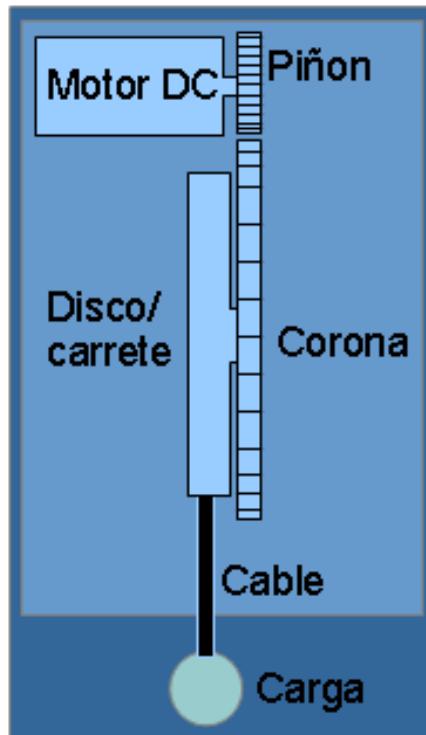


Figura 2.2: Modelo mecánico de la planta

2.3.3 Procedimiento para el diseño de control

Primero se determinará el período de muestreo y actuación inyectándole un escalón unitario a planta.

Luego, se obtendrá el modelo de la planta mediante identificación paramétrica, haciendo uso de un método de estimación de tipo offline (no recursivo).

Luego se diseñarán los controladores de posición y velocidad en base al modelo de la planta obtenido

Finalmente se validarán los controladores hayados haciendo pruebas directamente sobre el updown.

2.3.4 Relación entre cuentas de encoder y distancia

A medida que el motor gira desenrolla el cable para subir o bajar la carga. El problema es que dependiendo cuán enrrollada está el cable en el disco un mismo número de cuentas de encoder se puede traducir en distintas distancias recorridas por la carga. Por ejemplo, si el cable está completamente enrollada cuando se gira 360 grados el carrete el largo desenrollado será uno, mientras que si el cable se encuentra parcialmente desenrollada

la distancia descendida en un giro de 360 grados del carrete será menor que la anterior. Para determinar esta relación se harán marcas sobre en el cable y para cada una se anotará el valor de cuentas del encoder del motor, que es el que más resolución tiene. Con estos datos se construirá una tabla, se encontrarán varios polinomios que ajusten los datos y se determinará cuál de ellos es el que se implementará en software.

2.3.5 Velocidad máxima

Una tarea a resolver es determinar cuál es la velocidad máxima posible para una carga determinada.

Para determinar este valor se medirá la velocidad para una carga de 3Kg en subida aplicando sobre el motor un PWM con ciclo de trabajo de 100%.

La posición máxima posible queda dada por condiciones de diseño: 4 metros

2.4 REQ-03

Los posibles errores detectables que se pueden tener en el equipo son:

2.4.1 Corte de correa

La transmisión de potencia entre el piñón en el eje del motor y la corona en el disco se hace a través de una correa. El problema es que durante las pruebas, la empresa observó que la correa podría cortarse.

La manera que tiene el equipo de detectar este error es mediante un segundo encoder AB que mide el giro del disco. Como la relación de vueltas entre el disco y el motor es proporcional, cualquier desviación grande de esta proporcionalidad indica que uno se está moviendo más rápidamente que el otro. Esto podría interpretarse como que la correa fue cortada o que alguno de los encoders dejó de funcionar, 2 errores válidos de hardware.

Para poder detectar estos errores lo primero que se hará es determinar la relación entre las cuentas del encoder del disco y las del motor. Luego verificará en el firmware que esta se cumpla en todo momento, bajo un cierto nivel de error aceptable. En caso de no cumplirse se accionará el freno, se detendrá el equipo completamente y se indicará sobre la existencia del error mediante el led de la placa de control (led interno).

2.4.2 Fin de carrera

La función principal del fin de carrera, un par de pulsadores en la base del equipo, es indicar cuándo el cable que sostiene la carga está completamente enrollada para determinar la posición 0 o "home" del equipo, dado que este puede estar en cualquier posición al ser encendido.

La función secundaria del fin de carrera es detectar posibles eventos indeseados. Por ejemplo, en su punto más bajo la carga se encontrará a 4 metros por debajo del 0 del updown, por lo que podría comenzar a oscilar debido a fuertes vientos o personas que muevan la carga. Por cómo está construido el fin de carrera si la carga oscila más allá de un ángulo de aproximadamente 15 grados el fin de carrera se accionará dando aviso de esto.

Para poder detectar estos errores se verificará que el fin de carrera no sea presionado fuera de la rutina de calibración. En caso de que sea presionado en estas condiciones se detendrá el equipo, se esperará un tiempo a que la oscilación baje, y intentarán reanudar las operaciones normales. En caso de que el fin de carrera siga presionado se accionará el freno, se detendrá el equipo completamente y se indicará sobre la existencia del error mediante el led de la placa de control (led interno).

2.4.3 Pérdida de DMX

El estándar DMX establece que el tiempo máximo entre un paquete de datos y otro, tiempo de IDLE o MTBP (ver tabla 1.1), es de 1 segundo. Esto quiere decir que si por 1 segundo no me llegaron nuevos datos se considera que se perdió la comunicación con el dispositivo DMX.

Para detectar la pérdida de señal de DMX se verificará constantemente que el último paquete haya llegado hace menos de 1 segundo. En caso de no cumplirse se asumirá que las referencias de posición y velocidad son 0 (equipo parado). Además, por una convención en equipos DMX, se indicará mediante el led en el dipswitch (led externo) cuándo el equipo recibe correctamente la señal de DMX y cuándo no, encendiéndolo y apagándolo rápidamente en el primer caso y lentamente en el segundo.

2.5 REQ-04

El propósito del dipswitch en el equipo es poder seleccionar el canal principal de DMX a partir del cual leer las referencias de posición y velocidad, dentro de los 512 canales que tiene un universo DMX. Para lograr esto cuenta con 10 switchs, 9 que determinan el canal de DMX ($2^9 = 512$) y uno extra para ampliaciones futuras. Estos conmutadores

conectan o desconectan resistencias de unos divisores de tensión resistivos, variando el valor entregado por cada uno de ellos. Como el dipswitch cuenta con 3 divisores se tienen 3 salidas analógicas, que dan información de qué switchs están activados y cuales no. El esquema de uno de los divisores se puede ver en la figura 2.3. Para completar con el desarrollo del dipswitch se deben encontrar 5 valores de resistencias (R_1, R_2, R_3, R_4 y R).

Para hallar estos 5 valores se simulará el divisor de tensión en Matlab, probando con distintas combinaciones de resistencias hasta que los valores analógicos de la salida para cada combinación de resistencia estén lo suficientemente separados.

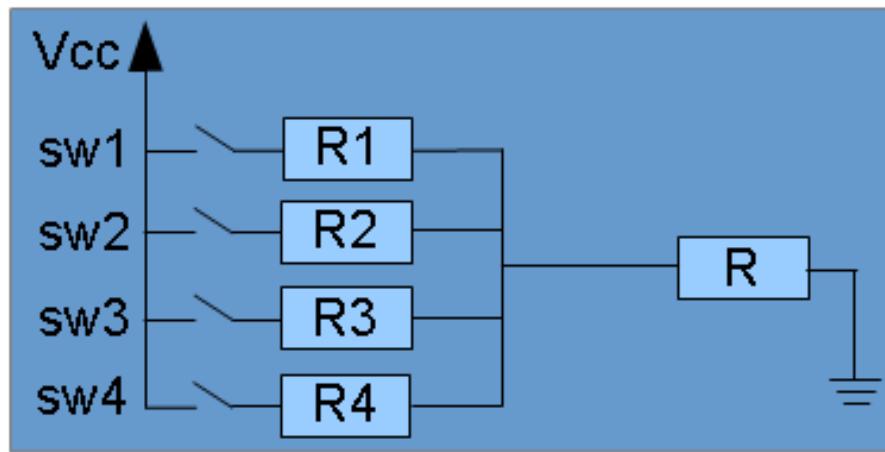


Figura 2.3: Diagrama de uno de los 3 divisores de tensión del dipswitch

2.6 Firmware del updown

2.6.1 Aviso de confidencialidad

Debido a políticas de confidencialidad de la empresa Blackout, el código fuente del firmware desarrollado para este proyecto no puede ser distribuido libremente. Es por este motivo que únicamente se describirá cómo se solucionaron los problemas, sin analizar el código puntualmente.

2.6.2 Convenciones

Con el objetivo de maximizar el encapsulamiento, las funciones dentro de los módulos se dividirán en internas y externas. Las internas podrán ser accedidas únicamente por otras funciones dentro del módulo, mientras que las externas también podrán ser accedidas por funciones de otros módulos. Para implementar esto se hará uso del sistema de separación

de información en archivos .c (código) y .h (headers). Los .h tendrán las declaraciones de funciones externas y definiciones para que, al ser incluido por otros módulos, tengan acceso a las mismas.

En cuanto a variables, solo se utilizarán variables internas al módulo. En caso de que tengan que poder ser accedidas externamente se implementarán setters (para escribir un valor en la variable) y getters (para escribir el valor de la variable) según corresponda. Las funciones tendrán su nombre en formato *camelCase* (primera letra minúscula y el resto de las primeras letras de las palabras utilizadas en el nombre en mayúscula), los Define en *MAYUSCULA*, y los módulos en formato PascalCase (como el camel case pero con la primera letra en mayúscula). Finalmente, todas las funciones y definiciones que pueden ser accedidas desde afuera de un módulo llevarán como prefijo el nombre del módulo seguido de un guión bajo.

Además, debido a políticas de Blackout, todo el software desarrollado debe ser entendible por todo el equipo de trabajo. Esto quiere decir que una de las metas para las funciones de las librerías a crear es que permitan un flujo entendible, y que los nombres de las funciones sean parecidas a las de Arduino ya que lo que se acostumbraba a utilizar en la empresa para la programación de sistemas embebidos.

Para lograr este objetivo todas los módulos desarrollados tendrán una función de inicialización llamada "init", y de lectura y escritura, según corresponda, llamadas "read" y "write", respectivamente.

A continuación se presentan algunos ejemplos de las convenciones que se usarán. Función de inicialización del módulo 1: Modulo1_init(); Define del largo de un buffer genérico del modulo 3: Modulo3_BUFFERGENEROICO.

2.6.3 Librerías de bajo nivel

Como se mencionó en la sección 2.2, se desarrollarán librerías para manejar los periféricos del Atmega328p y así poder controlar el hardware del sistema. Esto implica analizar la [hoja de datos del Atmega328p](#) e interactuar con los registros, un lugar de memoria utilizado para guardar información, que correspondan.

Asociadas a las librerías viene de la mano el concepto de módulo, que tiene que ver con agrupar funciones similares en un mismo paquete para desacoplar dependencias y separar al sistema en varios subsistemas que sean lo más independientes que se pueda. Esto ayuda a que el programa sea escalable y mantenable, por lo que todo el software desarrollado se hará de manera modular.

Entonces, para manejar los periféricos se creará una librería de bajo nivel que constará de los siguientes módulos:

- **Entradas y salidas digitales**, para leer el estado del fin de carrera y comandar el freno, los leds indicadores interno y externo, y el puente H. Este módulo se llamará

DigitalIO.

- **Entradas analógicas**, para leer las 3 salidas analógicas del dipswitch y poder determinar la posición de los selectores. Este módulo se llamará *ADC*, por *Analog to Digital Converter*.
- **Interrupciones externas** por cambio de estado de un pin, para llevar cuenta de los cambios de estado de los encoders AB y así saber la posición relativa. Este módulo se llamará *EXINT*, por *EXternal INTerrupts*.
- **PWM**, para el manejo del motor de continua. Este módulo se llamará *PWM*, por *Pulse Width Modulation*.
- **UART**, para la lectura de la señal DMX. Este módulo se llamará *UART*, por *Universal Asynchronous Receiver/Transmitter*
- **UART por software**, una UART implementada en pines genéricos que se utilizará para el debuggeo de la placa. Este módulo se llamará *SUART*, por *Software UART*.
- **Base de tiempo**, un timer utilizado para la temporización tareas y eventos. Este módulo se llamará *Tick*.

2.6.4 Librerías de alto nivel

Si bien la librería de bajo nivel permite acceder a las entradas y salidas del equipo, se necesita otra librería, de alto nivel, que utilice sus módulos para manejar el hardware como se espera. Por ejemplo, no es lo mismo obtener un valor analógico leyendo un *registro* (lugar para el almacenamiento de datos) del microcontrolador, que determinar qué conmutadores del dipswitch están activados leyendo 3 valores analógicos, aunque ambas acciones involucren leer una entrada analógica.

Entonces, se creará una librería más acorde a la aplicación y no tan genérica como la de bajo nivel, que constará de los siguientes módulos:

- **Dipswitch**, que incluye la lectura de las llaves del dipswitch, y la escritura del led indicador externo.
- **Grúa**, que incluye el manejo del motor, del led interno, del freno y del fin de carrera.
- **DMX**, para la recepción de datos de la consola DMX.
- **Encoder**, para el manejo de los encoders AB del disco y del motor.
- **Controlador**, para la implementación del sistema de control

2.6.5 Diagrama de módulos

De estas 2 librerías se obtiene el diagrama de módulos presentado en la figura 2.4, el cual muestra su relación, dependencia y alcance. Por encima de ellas se encuentra la capa de aplicación, que hará uso de estas librerías, relacionandolas con el objetivo de que el Updown funcione como debe.

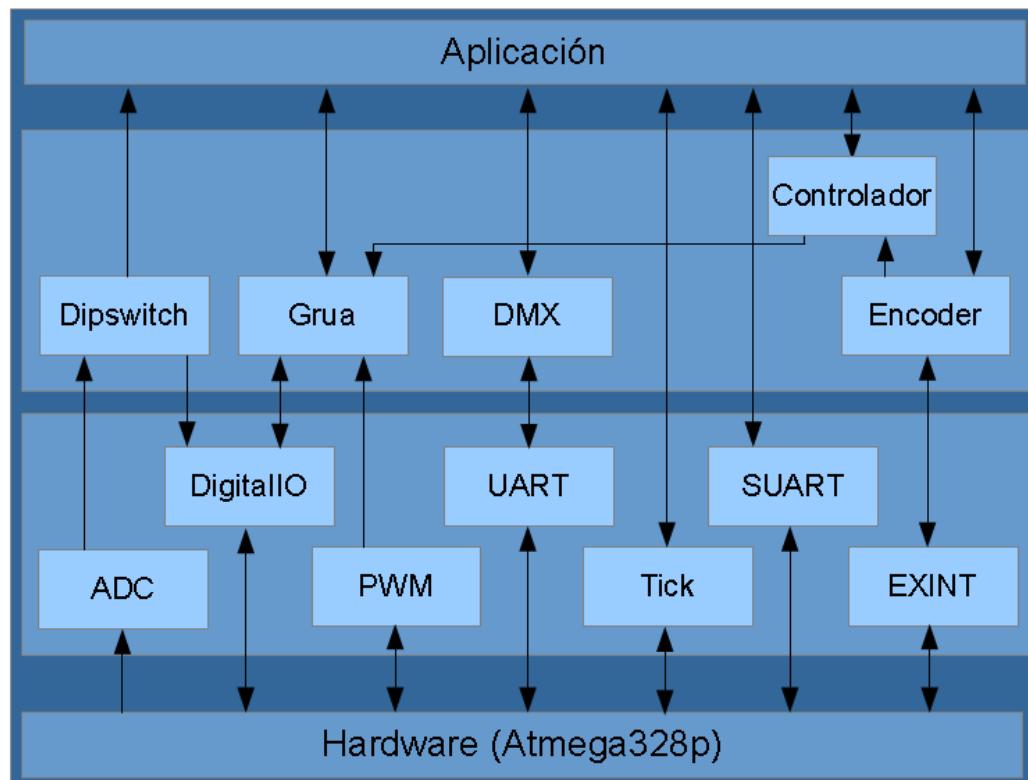


Figura 2.4: Diagrama de módulos del firmware

Desarrollo

3.1 Descripción del capítulo

En este capítulo se desarrollarán las soluciones propuestas en el capítulo de diseño. Esto implica detallar el proceso realizado, describir los cambios que se debieron hacer en caso de que el planteo inicial no haya funcionado, análisis de resultados, etc.

3.2 Firmware del updown - Librerías de bajo nivel

3.2.1 DigitalIO

3.2.1.1 Objetivo

La función de este módulo es manejar el periférico de entradas y salidas del microcontrolador con el propósito de leer el estado del fin de carrera y seleccionar el estado del freno, los leds indicadores interno y externo, y el puente H.

3.2.1.2 Desarrollo

Las entradas y salidas digitales del Atmega328p se manejan mediante 3 registros: el DDRn (Data Direction Register n), PORTn (Port n Data Register) y PINn (Port n Input Pins Address). La "n" en los registros se refiere al registro específico a ser accedido, que en el caso de este microcontrolador puede ser B, C o D.

Antes de poder leer o escribir el estado de un pin es necesario inicializarlo. Para inicializar un pin como entrada o salida digital primero se elige la dirección del mismo, o sea, si se utilizará como entrada O como salida. Para esto se utiliza el registro DDRn, en donde escribir un bit de este registro en 1 significa configurar el pin asociado a este bit como Output (salida) o , mientras que si se escribe en 0 significa configurar al pin como Input (entrada). Para el caso de las entradas se puede optar por habilitar una resistencia pull-up interna del microcontrolador. El pull-up se encuentra deshabilitado por defecto, pero puede ser habilitado escribiendo un 1 en el bit análogo del registro PORTn. Para escribir el valor de un pin configurado como salida se setea su valor mediante el registro PORTn. Un 1 en un bit de este registro significa poner en HIGH (5 Volts) la salida asociada a ese bit, mientras que un 0 es un estado LOW (0 Volts).

La lectura del estado de un pin configurado como entrada se hace a través del registro PINn. Enmascarando un bit específico en el puerto se puede obtener ese valor en particular.

Por ejemplo, si se quiere inicializar el bit 3 del puerto B como entrada con pull-up y el bit 6 del puerto D como entrada en lenguaje C, se tiene:

```

1 /* —— Configuracion bit 3 puerto B como Input – Pullup —— */
2 // Inicializacion del pin
3 DDRB &= ~(1 << 3); // Inicializacion como entrada
4 PORTB |= (1 << 3); // Habilitacion pullup
5
6 // Lectura del estado del pin
7 estadoBit3PuertoB = PINB & (1 << 3);
8
9 /* —— Configuracion bit 6 puerto D como Output —— */
10 // Inicializacion del pin como salida
11 DDRD |= (1 << 6);
12
13 // Escritura de un 1 en el pin
14 PORTD |= (1 << 6);

```

Ahora, conocer el bit, puerto y nombres de los registros para cada pin a configurar es muy tedioso. Por lo tanto, para facilitar la lecto-escritura y configuración de los pines se mapearon los bits de los puertos B,C y D a números, como se muestra en la tabla 3.1. El criterio de enumeración fue basado en los pines del Arduino UNO.

Puerto	Bit	Número mapeado
D	0, 1, 2, 3, 4, 5, 6, 7	0, 1, 2, 3, 4, 5, 6, 7
B	0, 1, 2, 3, 4, 5	8, 9, 10, 11, 12, 13
C	0, 1, 2, 3, 4, 5	14, 15, 16, 17, 18, 19

Tabla 3.1: Mapeo de bits de los puertos a número

3.2.1.3 Acceso

El acceso al módulo se realiza a través 3 funciones, como se muestra en la figura 3.1. A continuación se presenta la descripción de cada una:

- **DigitalIO_init(pin,modo)**, que inicializa el pin deseado en el modo salida o entrada, que a su vez se divide en con pull up y sin pullup. Estas 3 posibles configuraciones se puede seleccionar haciendo uso de los macros DigitalIO_OUTPUT, DigitalIO_INPUT_PULLUP y DigitalIO_INPUT, respectivamente.
- **DigitalIO_read(pin)**, para leer el estado de un pin configurado como salida.

- **DMX_write(pin, estado)**, para escribir en un pin el estado deseado (HIGH = 1 o LOW = 0).

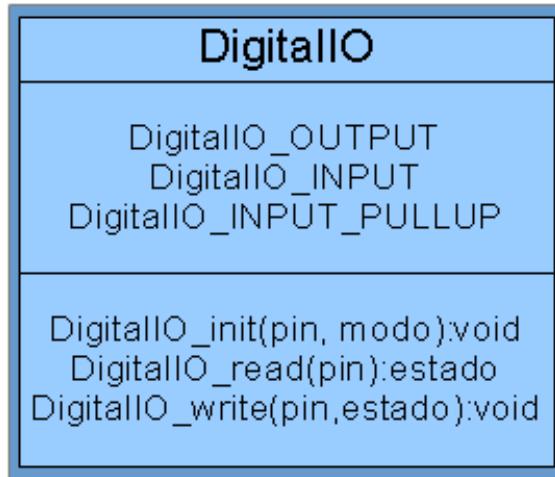


Figura 3.1: Diagrama del módulo DigitalIO

Con este nuevo esquema, el ejemplo resuelto mediante registros queda de la siguiente manera:

```

1  /* --- Configuracion bit 3 puerto B como Input – Pullup --- */
2  // Inicializacion del pin como entrada con pullup
3  DigitalIO_init(11, DigitalIO_INPUT_PULLUP); // Puerto B bit 3 = 11
4
5  // Lectura del estado del pin
6  estadoBit3PuertoB = DigitalIO_read(11);
7
8  /* --- Configuracion bit 6 puerto D como Output --- */
9  // Inicializacion del pin como salida
10 DigitalIO_init(6, DigitalIO_OUTPUT); // Puerto D bit 6 = 6
11
12 // Escritura de un 1 en el pin
13 DigitalIO_write(6, 1);
  
```

3.2.2 ADC

3.2.2.1 Objetivo

La función de este módulo es manejar el periférico lectura de canales analógicos del microcontrolador con el objetivo de poder leer las 3 salidas del dipswitch.

3.2.2.2 Desarrollo

El periférico de lectura de entrada analógica del Atmega328p es un ADC por *aproximaciones sucesivas* de 10bits de resolución. Esto quiere decir que la conversión se realiza a una frecuencia determinada, y con una tensión de referencia dada. Además, este microcontrolador cuenta con 1 solo módulo de ADC y 8 canales analógicos, por lo que el usuario debe elegir a qué canal irá el resultado de la conversión mediante un multiplexor de entradas. El diagrama completo del sistema de conversión se encuentra en la figura 28.1 del datasheet del atmega328p.

La inicialización del periférico se logra mediante los registros ADMUX (ADC Multiplexer Selection Register) y ADCSRA (ADC Control and Status Register). Con el primero se selecciona la tensión de referencia, mientras que con el segundo se habilita el periférico y se selecciona la frecuencia de la conversión mediante la selección de un preescalador. Por ejemplo, si se desea utilizar como referencia de tensión la Vcc del micro (5V) y se quiere una frecuencia de 125KHz el código, en lenguaje C, sería:

```

1 /* — Inicializacion del periferico — */
2 // Seleccion de Vref = Vcc
3 ADMUX = (1 << REFS0);
4
5 // Habilitacion del adc y seteo del prescaler a 128 => f_adc = 125KHz
6 ADCSRA = (1 << ADEN) | (1 << ADPS2) | (1 << ADPS1) | (1 << ADPS0);
```

Cabe aclarar que REFS0 es el bit asociado a la selección de $V_{ref} = V_{cc}$. Es el 6to bit del registro ADMUX, por lo que $REFS0 = 6$. Similarmente, $ADEN = 7$, $ADPS2 = 2$, $ADPS1 = 1$ y $ADPS0 = 0$.

Luego, para la lectura del valor analógico se:

1. Selecciona el canal mediante el multiplexor de entradas con el registro ADMUX. Los 3 bits menos significativos, llamados MUX0/1/2, de este registro permiten seleccionar los canales: si $MUX[2:0] = 000 \Rightarrow$ selecciono el canal 0, si es igual a 001 selecciono el canal 1, y así hasta llegar a 111 para el canal 7.
2. Da comienzo a la conversión poniendo un 1 en bit 6 (llamado ADSC) del registro ADCSRA.
3. Esperar a que termine la conversión. Cuando la conversión termina ADSC, que había sido seteado en 1, pasa a valer 0.
4. Recupera el resultado leyendo el registro ADC.

El ejemplo de lectura del canal 2, en lenguaje C, sería:

```
1 /* — Lectura del canal 2 — */
```

```

2 // Seleccion del canal que se desea leer conservando el resto de los
3 // valores del registro
4 ADMUX = (ADMUX & 0b11111000) | 2; // 2 debido al canal a ser leido
5
6 // Inicio de conversion
7 ADCSRA |= (1<<ADSC);
8
9 // Espera del fin de la conversion
10 while(ADCSRA & (1<<ADSC));
11
12 // Lectura del valor resultante de la conversion
13 valorCanal2 = ADC;

```

3.2.2.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.2. A continuación se presenta la descripción de cada una:

- **ADC_init()**, que inicializa el periférico ADC con una frecuencia de muestreo de 125KHz y una tensión de referencia de 5V.
- **ADC_read(canal)**, para leer el valor analógico del canal deseado.

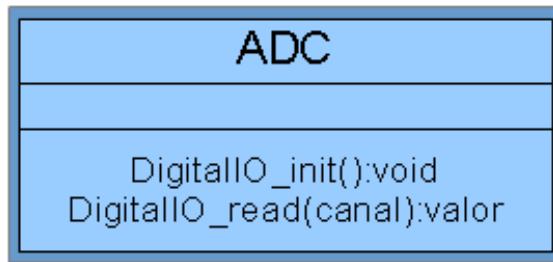


Figura 3.2: Diagrama del módulo ADC

Con este nuevo esquema, los ejemplos resueltos mediante registros quedan de la siguiente manera:

```

1 /* —— Inicializacion del periferico —— */
2 ADC_init();
3
4 /* —— Lectura del canal 2 —— */
5 valorCanal2 = ADC_read(2);

```

3.2.3 PWM

3.2.3.1 Objetivo

La función de este módulo es generar una señal de tipo tren de pulsos con ciclo de trabajo (tiempo encendido sobre período de la señal) variable con el objetivo de manejar la velocidad del motor de continua que mueve la carga en el updown.

3.2.3.2 Desarrollo

En el Atmega328p no existe un periférico exclusivo para generación de PWM. Por lo tanto, se generará con el periférico Timer del microcontrolador. De los Timers que tiene este microcontrolador, el más potente para generación de PWM es el Timer1, puesto que cuenta con un modo de generación de PWM de fase correcta. Este modo es preferible por sobre el modo de pwm soportado por los otros 2 timers (0 y 2) ya que no provoca corrimientos de fase durante la variación del ancho del pulso, efecto que hay que evitar en el control de motores.

Para un Timer, la selección de la frecuencia es un factor fundamental. En esta aplicación la frecuencia elegida para la señal de PWM es de 25KHz, ya que:

- Es lo suficientemente rápida como para que el motor integre la señal y la tome como continua.
- Al encontrarse por encima de 20KHz, que es el tope del rango audible, no provoca ruidos indeseados.
- Es soportada por los mosfets utilizados en el puente H (NTD3055L104).

La generación de PWM en modo fase correcta se hace a través de la comparación de un contador (TCNT1) con un valor definido por el usuario (OCR1A). La señal de PWM se manifiesta por una salida digital especial llamada OC1A, y la forma de la señal depende del tipo de comparación. Si el tipo de comparación es no-invertida, OCR1 vale 1 si el TCNT1 es menor a ICR1, y 0 en caso contrario. De esta manera, se puede ajustar el ciclo de trabajo al variar el valor de ICR1. En la figura 3.3 se muestra la señal en OC1A si el timer se encuentra en modo fase correcta con comparación tipo no-invertida, al variar el valor de OCR1A. A medida que este valor sube también lo hace el ciclo de trabajo. La configuración de estos modos de trabajo se hace a través del registro TCCR1A.

La frecuencia de la señal se puede elegir mediante el registro ICR1. Su valor depende de la frecuencia del microcontrolador (f_{cpu}), de la frecuencia objetivo (f_{pwm}) y de un preescalador (N) seleccionable mediante el registro TCCR1B, como se puede ver en la



Figura 3.3: Ejemplo variación ciclo de trabajo al cambiar ICR1

ecuación 3.1. Para este proyecto el atmega328p trabaja a $f_{cpu} = 16Mhz$, y la frecuencia objetivo es $f_{pwm} = 25Khz$, por lo que tomando un preescalador de $N=1$ se tiene que ICR1 vale exactamente 320.

$$ICR1 = \frac{f_{cpu}}{2.N.f_{pwm}} \quad (3.1)$$

A continuación se presenta un ejemplo de inicialización del periférico y selección del ciclo de trabajo, en lenguaje C:

```

1  /* — Inicializacion del periferico — */
2  // Inicializacion del pin asociado a OC1A como salida , que es el 9 (
3  // puerto B bit 1).
4  DigitalIO_init (9,OUTPUT);
5
6  // Configuracion para pwm de fase correcta
7  TCCR1A |= (1 << WGM11)|(0 << WGM10); TCCR1B |= (1 << WGM13)|(0 << WGM12);
8
9  // Configuracion para tipo de comparacion no-invertida
10 TCCR1A |= (1 << COM1A1)|(0 << COM1A0)|(1 << COM1B1)|(0 << COM1B0);
11
12 // Configuracion para prescalador = 1
13 TCCR1B |= (0 << CS12) | (0 << CS11) | (1 << CS10);
14
15 // Configuracion para frecuencia de 25KHz
16 ICR1 = 320;

```

```

17  /* — Seleccion de ciclo de trabajo — */
18  // Ciclo de trabajo de x%: ICR1*x/100. Para 50% se tiene
19  OCR1A = 160;

```

3.2.3.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.4. A continuación se presenta la descripción de cada una:

- **PWM_init()**, que activa la generación de la señal cuadrada de 25KHz por la salida OC1A.
- **PWM_write(cicloDeTrabajo)**, para seleccionar el ciclo de trabajo del PWM.

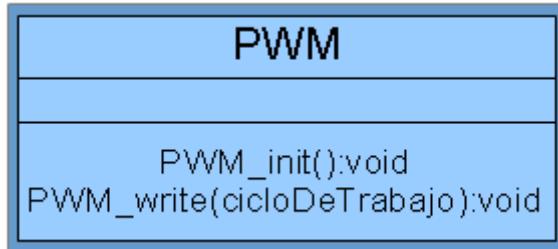


Figura 3.4: Diagrama del módulo PWM

De esta forma, el ejemplo resuelto mediante registros queda de la siguiente manera:

```

1  /* — Inicializacion del periferico — */
2  // Inicializacion del pin asociado a OC1A como salida , que es el 9 (
3  // puerto B bit 1).
4  PWM_init();
5
6  /* — Seleccion de ciclo de trabajo — */
7  PWM_write(50); // ciclo de trabajo a 50%

```

3.2.4 EXINT

3.2.4.1 Objetivo

El función de este módulo es generar interrupciones ante cambios de estados en un pin con el objetivo de poder contar los pulsos de los encoders de disco y motor.

3.2.4.2 Desarrollo

En el atmega328p hay 2 tipos de interrupciones externas:

- Las EXINT, que pueden ser configuradas para que se accionan por cualquier tipo de cambio en un pin (flanco ascendente, descendente y ambos), pero que solo están disponibles en 2 pines, el pin 2 y el 3.
- Las PCINT que siempre se activan con un cambio en el pin (no se puede elegir como en las EXINT), pero que están disponibles para la mayoría de los pines del microcontrolador. En total hay 24 pines que pueden activar esta interrupción.

Lo único que se busca de este módulo es habilitar interrupciones, por lo que solo se necesitan funciones de inicialización. Para inicializar las EXINT se selecciona el tipo de disparo en el registro EICRA, se habilita la entrada digital correspondiente, y activa la interrupción en el registro EIMSK.

Para las PCINT, por otro lado, solo se habilita la entrada digital correspondiente y se activa la interrupción en el registro PCICR.

Para ayudar al encapsulamiento lo ideal es que este módulo no implemente las rutinas de cuenta de los encoders, sino que esa lógica debería estar en otro módulo de mayor nivel. Para lograr esto se hace uso de *punteros a función*. Los punteros a función son variables que guardan la dirección de una función, haciendo que la función pueda ser llamada a través del puntero, sin necesidad de ejecutarla directamente. Esto permite que una función pueda ser pasada como parámetro, haciendo que la implementación de dicha función no tenga que estar en el módulo en donde se ejecuta, desasociando un módulo de otro. Por lo tanto, las funciones de inicialización de los periféricos EXINT y PCINT reciben como parámetro una función, que será la ejecutada durante la interrupción.

3.2.4.3 Acceso

El acceso al módulo se realiza a través 2 función, como se muestra en la figura 3.5. Ambas funciones habilitan la interrupción y le asocian la función pFuncion para que se ejecute cuando un cambio en el estado del pin ocurra.

3.2.5 UART

3.2.5.1 Objetivo

La función de este módulo es configurar el periférico UART para que pueda leer los datos recibidos por el maestro DMX.

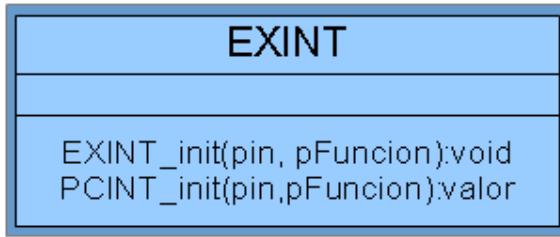


Figura 3.5: Diagrama del módulo EXINT

3.2.5.2 Desarrollo

El Atmega328p tiene un periférico de UART bastante estándar, en donde se puede configurar su tasa de transmisión (baudrate), modo de operación y formato de trama. Esto se logra por medio de los registros UBRR0, UCSR0A, UCSR0B y UCSR0C.

UBRR0, que se separa en UBRR0H u UBRR0L, sus partes alta y baja respectivamente, de 8 bits cada una, se utiliza para configurar el baudrate. La ecuación 3.2 sirve para obtener el valor de UBRR0, en donde el 16 pasa a ser 8 si el bit U2X0 en el registro UCSR0A está seteado.

$$UBRR0 = \frac{f_{clk}}{(16.BAUD)} - 1 \quad (3.2)$$

Como DMX trabaja a una tasa de 250KHz y la frecuencia del microcontrolador es de 16MHz tenemos que $UBRR0 = 3$, o sea que $UBRR0L = 3$ y $UBRR0H = 0$.

Mediante el registro UCSR0B se habilita o deshabilita la transmisión y recepción de datos. Como los esclavos DMX solo reciben datos, solo hace falta habilitar la recepción, lo cual se logra seteando el bit RXEN0 de este registro. También se pueden habilitar las interrupciones por recepción poniendo en 1 el bit RXCIE0, lo cual es necesario en esta aplicación debido a que durante la interrupción se tiene que analizar la trama DMX. Al igual que para el módulo EXINT, aquí también se utilizará el tipo de dato voidFunctionPointer_t para que el módulo que llame a este pueda indicar qué función quiere ejecutar durante las interrupciones de recepción.

Finalmente, el registro UCSR0C permite configurar el modo (sincrónico o asincrónico) y el formato de la trama. DMX trabaja en modo asincrónico y con formato de trama 8N2.

En cuanto a la recepción, los datos que llegan por el puerto serie se almacenan en el registro UDR0. Para leerlo se verifica si el bit RXC0 en el registro UCSR0A se encuentra en 1, ya que este bit indica si existen datos no leídos en el buffer de recepción.

A continuación se presenta un ejemplo de inicialización y lectura de datos del periférico,

, en lenguaje C:

```

1  /* —— Inicializacion del periferico —— */
2  // Configuracion de tasa = 250KHz
3  UBRROH = 0;
4  UBRROL = 3;
5  UCSR0A = (0<<U2X0);

6
7  //Habilitacion de Rx y su interrupcion
8  UCSR0B = (1<<RXEN0)|(0<<TXEN0)|(1<<RXCIE0);

9
10 //Seleccion de modo asincronico , con formato de trama 8N2
11 UCSR0C = (1<<USBS0)|(1<<UCSZ01)|(1<<UCSZ00);

12
13 /* —— Lectura de un dato —— */
14 // Se espera a que un dato sea recibido
15 while( !(UCSR0A & (1<<RXC0)) );

16
17 // Se saca el dato del buffer de recepcion
18 datoRecibido = UDR0;

```

Como en el módulo EXINT en este módulo también la función de inicialización recibe como parámetro una función para que sea ejecutada durante la interrupción de recepción.

3.2.5.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.6. A continuación se presenta la descripción de cada una:

- **UART_init(pFuncion)**, que activa el periférico UART con un baudrate de 250KHz y formato 8N2. Además activa la interrupción por recepción y le asocia la función pFuncion para que sea ejecutada cuando un dato sea recibido.
- **UART_read()**, para leer el dato serie recibido.

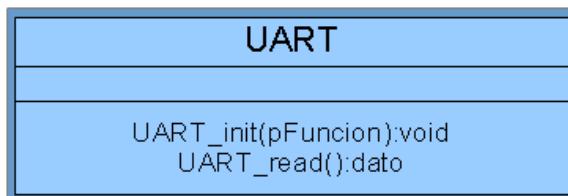


Figura 3.6: Diagrama del módulo UART

De esta forma, el ejemplo resuelto mediante registros queda de la siguiente manera:

```

1  /* — Inicializacion del periferico — */
2  UART_init(funcion);
3  // "funcion" es la rutina a ejecutar durante la interrupcion
4
5  /* — Lectura de un dato — */
6  datoRecibido = UART_read();

```

3.2.6 Tick

3.2.6.1 Objetivo

La función de este módulo es generar una base de tiempo con el objetivo de temporizar tareas.

3.2.6.2 Desarrollo

Para implementar una base de tiempo como es el módulo Tick es necesario un Timer. El Atmega328p cuenta con 3 timers, de los cuales 1 (el timer 1) ya fue utilizado para el módulo PWM. Los restantes son el 0 y el 2, y ambos pueden generar una señal de 1KHz sin problemas y con mínimo error, por lo que se utilizará el Timer2.

La base de tiempo será de 1ms, equivalente a una frecuencia de 1KHz, ya que si bien la mayoría de las acciones del equipo serán lentas, la actualización del controlador probablemente será del orden de los milisegundos.

Para generar la base de tiempo se utiliza el modo de comparación (CTC) del timer, seleccionable mediante el registro TCCR2A. En este modo un contador (TCNT2) aumenta hasta igualar un valor definido por el usuario (OCR2A). Cuando esto sucede se reinicia la cuenta y se genera una interrupción si el bit OCIE2A del registro TIMSK2 se encuentra en 1.

La frecuencia se configura mediante los registros TCCR2B (seteo del prescalador) y OCR2A, siguiendo la ecuación 3.3. Como la frecuencia deseada es $f_{tick} = 1KHz$ y la del microcontrolador es $f_{cpu} = 16MHz$, con un N = 128 se obtiene que OCR2A = 124.

$$OCR2A = \frac{f_{cpu}}{N \cdot f_{tick}} \quad (3.3)$$

A continuación se presenta un ejemplo de inicialización del periférico con una frecuencia de 1KHz e interrupción habilitada, en lenguaje C.

```

1  /* — Inicializacion del periferico — */
2  // Configuracion para el modo CTC
3  TCCR2A |= (1 << WGM21);

```

```

4 // Seteo de la frecuencia a 1KHz
5 TCCR2B |= (1 << CS22) | (0 << CS21) | (1 << CS20); // N = 128
6 OCR2A = 124;
7
8 // Habilitacion de la interrupcion
9 TIMSK2 = (1 << OCIE2A);
10

```

Como en el módulo EXINT en este módulo también la función de inicialización recibe como parámetro una función para que sea ejecutada durante la interrupción de recepción. Además de ejecutar esta función en la interrupción se incrementa la base de tiempo, contando cada milisegundo que transcurra.

3.2.6.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.7. A continuación se presenta la descripción de cada una:

- **Tick_init(pFuncion)**, que habilita una interrupción cada 1 milisegundo en donde se incrementa la base de tiempo y se ejecuta la función pFuncion.
- **Tick_read()**, para leer el valor de la base de tiempo.

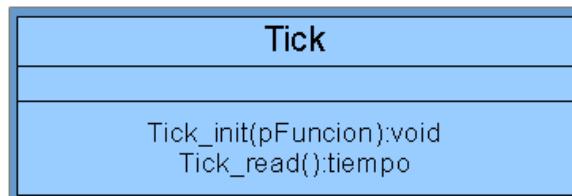


Figura 3.7: Diagrama del módulo Tick

3.2.7 SUART

3.2.7.1 Objetivo

La función de este módulo es generar un canal de comunicación serie bidireccional utilizando cualquier par de pines de la placa.

3.2.7.2 Desarrollo

Este módulo implementa la SUART haciendo uso del timer 0, que es el único que queda disponible (el 1 genera PWM y el 2 la base de tiempo). La función del timer es generar interrupciones cada cierto tiempo y verificar el estado de los pines asociados a Rx (entrada) y Tx (salida).

Para saber cómo implementar la lógica de lectura y escritura de datos se analiza la forma de la señal en un bus serie. Un ejemplo puede ser la señal del carácter 'a' con formato 8N1, como se ve en la figura 3.8 .

Durante una comunicación serie el canal se encuentra en estado High (1) por defecto, hasta que se envía el bit de stop, que siempre es un estado Low (0). Luego se envían los datos, que en el caso de la 'a' sería 01100001 en binario, empezando por el bit menos significativo (LSB) hasta el más significativo (MSB), y finalmente el bit de stop, que siempre es un estado High (1).

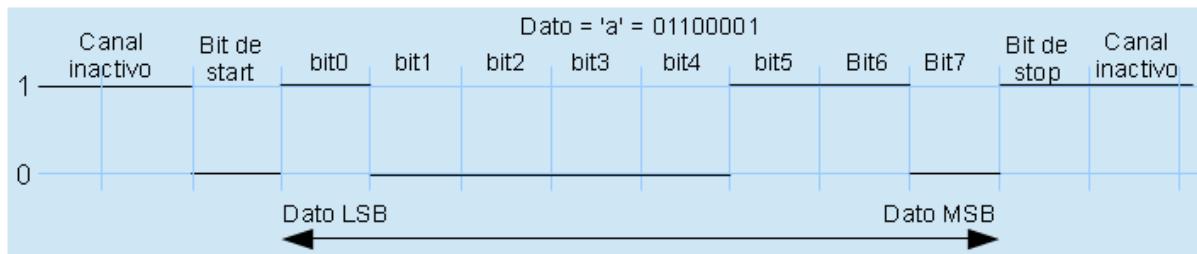


Figura 3.8: Forma de la señal en el canal para una 'p' con formato 8N1

En base a esto, para el envío de datos simplemente se cambia el estado del pin de Tx según el formato de la trama y del dato a enviar. Por ejemplo, si se quiere enviar una 'a', que equivale a 01100001, con una trama 8N1, se enviará 0 (bit de start), luego del bit menos al más significativo del dato (empiezo con 1, luego envío el 0.. etc) y finalmente 1 (bit de stop). Cada cambio de estado debe ocurrir a la tasa de símbolos (baudrate) deseada.

En cuanto a la entrada de datos lo que se hace es muestrear el pin de Rx cada cierto tiempo e ir reconstruyendo el dato. Este muestreo es conveniente realizarlo a la mitad del tiempo de bit, debido a que se podrían llegar a detectar datos inválidos si la interrupción no se llega a ejecutar a tiempo.

Por defecto el canal se encuentra en 1, por lo que lo muestreo hasta detectar el bit de start (0). Luego de detectado este bit se suele volver a verificar el canal para asegurarse que sea un bit de start y no un error en la línea, error comúnmente conocido como *line glitch*. Luego de verificar que no hubo un error se muestrea el canal a la tasa de símbolos, y una vez recopilados los 8 bits de datos, se verifica que el último bit sea el de stop (1).

La comunicación serie tendrá las siguientes características: formato de trama 8N1 (8

bit de datos, sin paridad, 1 bit de stop) y tasa de símbolos de 9600baudios. Se elige una tasa chica ya que las interrupciones del timer ocurrirán por lo menos 2 veces más rápido que esta tasa, y con interrupciones como la de DMX que tienen una frecuencia de 250Khz los recursos deben cuidarse.

Para la transmisión la tasa podría ser simplemente 9600 baudios, pero para la recepción se necesita una tasa de por lo menos el doble ya que se debe muestrear entre bits para detectar y evitar errores. Por lo tanto, se probó duplicando, triplicando y cuadruplicando la velocidad de muestreo.

Para decidir cuál es la tasa óptima se probó enviandole a la placa de control por puerto serie el carácter 'a' y replicando lo recibido hacia el transmisor. Para 10000 datos enviados los resultados fueron que con la tasa duplicada hubo una alta cantidad de información mal recibida, mientras que para la triplicada y cuadruplicada se recibieron 0 datos erroneos. Para decidir qué multiplicador utilizar se evaluó que el Atmega328p consumen 20 ciclos de clock solo para entrar en y salir de la interrupción, que equivale a $1.25\mu s$ para una frecuencia de cpu de 16MHz. 9600 triplicado y cuadruplicado es 28800 y 38400, que equivalen a una interrupción cada $34\mu s$ y $26\mu s$, respectivamente. O sea que cuanto mayor es el multiplicador más pesan los 20 ciclos comparativamente. Por lo tanto, como para ambos multiplicadores la cantidad de datos incorrectos recibidos es 0 de 10000, se eligió el multiplicador más chico, quedando la tasa de símbolos en 28800 baudios.

3.2.7.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.9. A continuación se presenta la descripción de cada una:

- **SUART_init(pinRx,pinTx)**, que inicializa la comunicación serie en los pines indicados.
- **SUART_read()**, para leer el dato serie presente en el pin de recepción.
- **SUART_write(dato)**, para escribir un dato en el pin de transmisión.

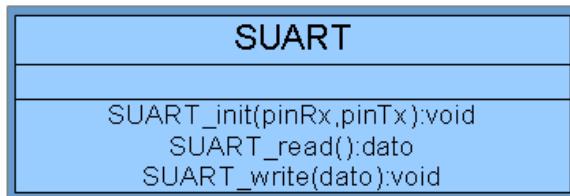


Figura 3.9: Diagrama del módulo SUART

3.3 Controlador

3.3.1 Relación entre cuentas de encoder y distancia

3.3.1.1 Pruebas y resultados

Como se mencionó en la sección 2.3, subsección 3, la distancia lineal recorrida por la carga varía según cuán enrollado está el cable en el carrete. La forma que se tiene de detectar que el cable se desenrolla es a través del encoder del motor, que mide la posición angular de su eje, y por consiguiente del disco, que contiene el cable que sostiene la carga. Lo que hay que hacer entonces es determinar la relación entre posición angular y lineal.

Para determinar la relación entre estas 2 variables se marcó el cable que sostiene la carga cada 25cm, tomando como 0 el cable completamente enrollado. Luego, a medida que se desenrollaba el cable se midió el valor de la cuenta de encoder del motor y de disco. Los resultados obtenidos se pueden encontrar en la tabla 3.2.

Distancia [cm]	0	25	50	75	100	125	150	175	200
Encoder mot.	0	1130	2250	3430	4630	5830	7130	8400	9730
Encoder disc.	0	8	17	26	35	44	54	64	74
Distancia [cm]	225	250	275	300	325	350	375	400	
Encoder mot.	11120	12530	14020	15530	17130	18770	20520	22330	
Encoder disc.	85	96	106	118	131	143	157	170	

Tabla 3.2: Relación entre cuentas del encoder motor y distancia

De estos resultados se puede ver que la relación entre las cuentas de motor y de disco es de aproximadamente 131 a 1. Cualquier gran desviación de esta relación significa que ocurrió algún problema o con las cuentas del motor o con la del disco. Esto puede incluir situaciones como: que un sensor del encoder de disco dejó de funcionar, que un cable se haya cortado, o la más importante, que la correa se corte. Si esto último sucede las cuentas del motor incrementarán a una tasa mucho mayor que las del disco, generando una gran diferencia entre la relación normal de ambas cuentas.

3.3.1.2 Selección de la función de ajuste

De la figura 3.10 se puede ver que la relación entre cuentas de encoder y distancia puede ser ajustada con un polinomio de grado 2 (función cuadrática). En dicha figura la función de ajuste cuadrática es $cuentaEncoder = f(distancia) = 0.0356 * dist^2 + 41.0180 * dist + 101.2797$.

El problema de este método es que la conversión se implementa en un microcontrolador de 8 bits en donde la posición va de 0 a 500 (variable de 16 bits). Esto implica que para obtener las cuentas de encoder una variable de 16 bits, que eventualmente podría tener signo, debe ser potenciada al cuadrado, y luego multiplicada por un número racional, lo cual implica operaciones de punto flotante o al menos divisiones. En la siguiente [nota de aplicación de ATMEL](#) se puede ver que dichas operaciones no están optimizadas, lo cual implicaría un altísimo costo computacional cada vez que se tenga que calcular la referencia de posición, que como DMX trabaja a 250KHz será muy seguido, haciendo la implementación de una función cuadrática inviable.

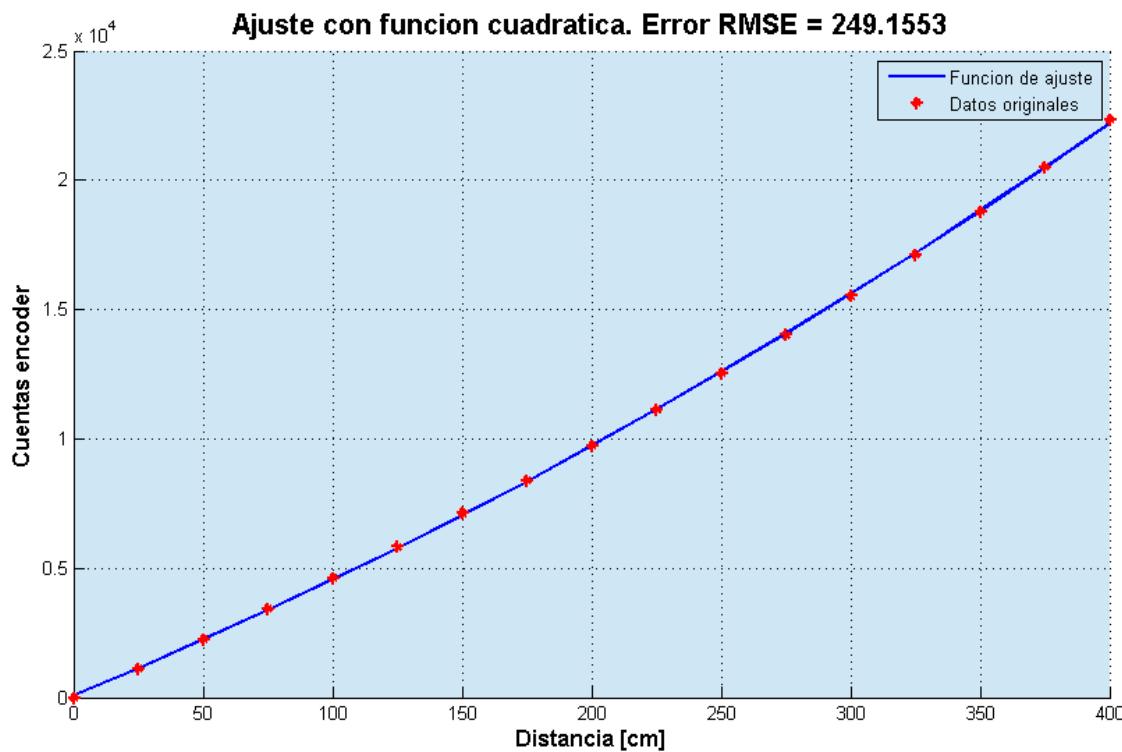


Figura 3.10: Ajuste mediante función cuadrática

La siguiente opción más simple es realizar la conversión mediante polinomios de grado 1. En este caso el costo computacional es mucho más bajo ya que la conversión implica únicamente la multiplicación con un escalar.

Los casos extremos serían hacer el ajuste mediante una única recta, como se ve en la figura 3.11, o una recta cada 25cm, como se ve en la figura 3.12. En el primer caso el error es muy alto al ser la aproximación muy grosera, y en el segundo se tiene que se ajustan tanto los datos que también se está ajustando el ruido.

Una solución intermedia es ajustar los datos con una recta por cada 100cm (1 metro),

como se ve en la figura 3.13. Esta trae como ventaja: error comparable con el del ajuste cuadrático, bajo costo computacional, solo se necesitan 4 mediciones por equipo (una por metro). Por estos motivos, **el ajuste de distancia a cuentas de encoder se realizará con 4 rectas**, una por metro.

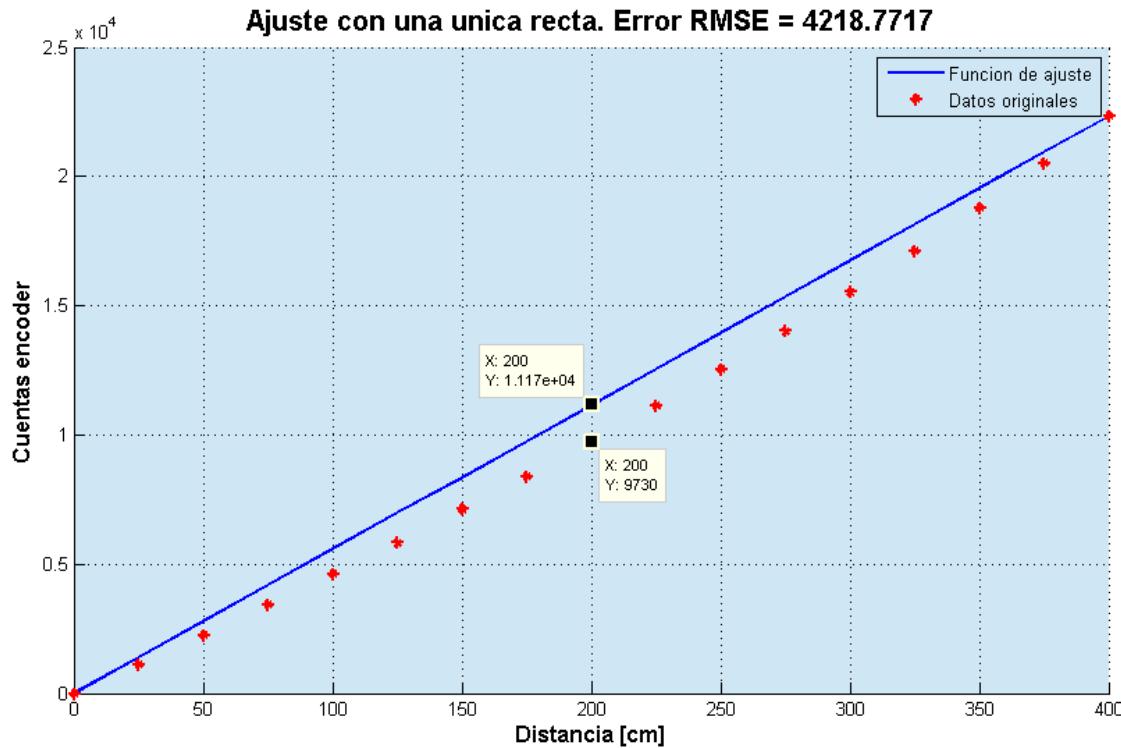


Figura 3.11: Ajuste mediante una única recta

3.3.2 Determinación de la velocidad máxima

El método utilizado para determinar la velocidad máxima a la que el equipo puede funcionar es hacer subir una carga de 3Kg inyectándole al driver del motor un PWM de ciclo de trabajo de 100%. Es necesario que la prueba sea en subida y no en bajada ya que la máxima velocidad posible debe ser igual en ambos casos, y en bajada seguro necesito menos fuerza para alcanzar la máxima velocidad en subida, por lo que la limitante es la segunda.

Luego, se mide cada cierto tiempo, que en este caso será de 50ms, cuantas cuentas del encoder del motor se tienen, y se calcula la velocidad como $velocidad = \Delta\text{posicion}/\Delta\text{tiempo}$.

Del experimento se obtuvo que cada 50ms se tienen 100cuentas, por lo que la velocidad máxima es $vel = 2\text{cuentas}/ms$.

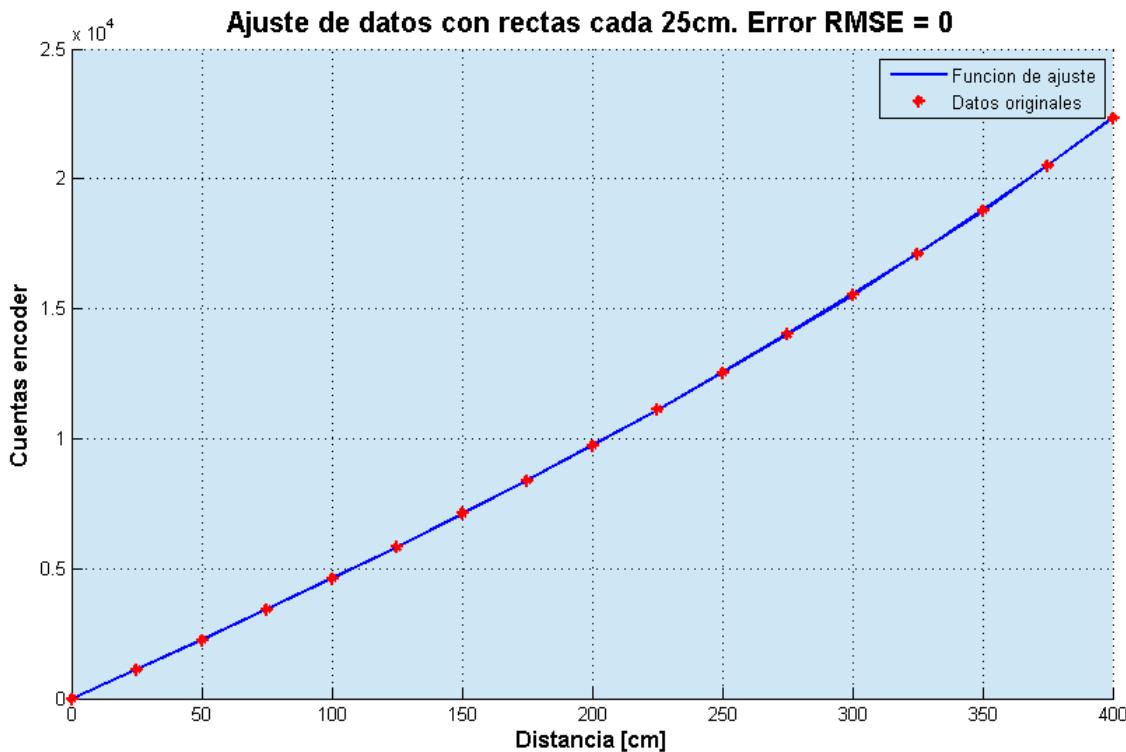


Figura 3.12: Ajuste mediante una recta cada 25cm

3.3.3 Obtención del período de muestreo

Para poder diseñar los controladores de posición y velocidad el primer paso es determinar el período de muestreo.

Para esto se realizaron varias pruebas tomando como 0 de posición 2000 cuentas de encoder (aproximadamente 50cm según la tabla 3.2): 2 pruebas en bajada, en donde las cuentas aumentan ya que el positivo se define para abajo, y 2 pruebas en subida. La fuerza aplicada al motor es de 15 y 33% del máximo de fuerza posible en bajada, y de 66 y 100% en subida.

Los resultados de estas pruebas se pueden ver en las figuras 3.14 y 3.15. Allí se ve que durante la bajada la salida se estabiliza a los 250ms, mientras que para la subida lo hace a los 50ms. Esto genera un problema, ya que el período de muestreo suele tomarse de 5 a 10 veces más rápido que el tiempo de establecimiento del sistema, y aquí tenemos 2 tiempos distintos.

Como la solución no es directa se probarán distintos períodos de muestreo durante la caracterización de la planta y diseño del controlador para determinar cuál es el indicado.

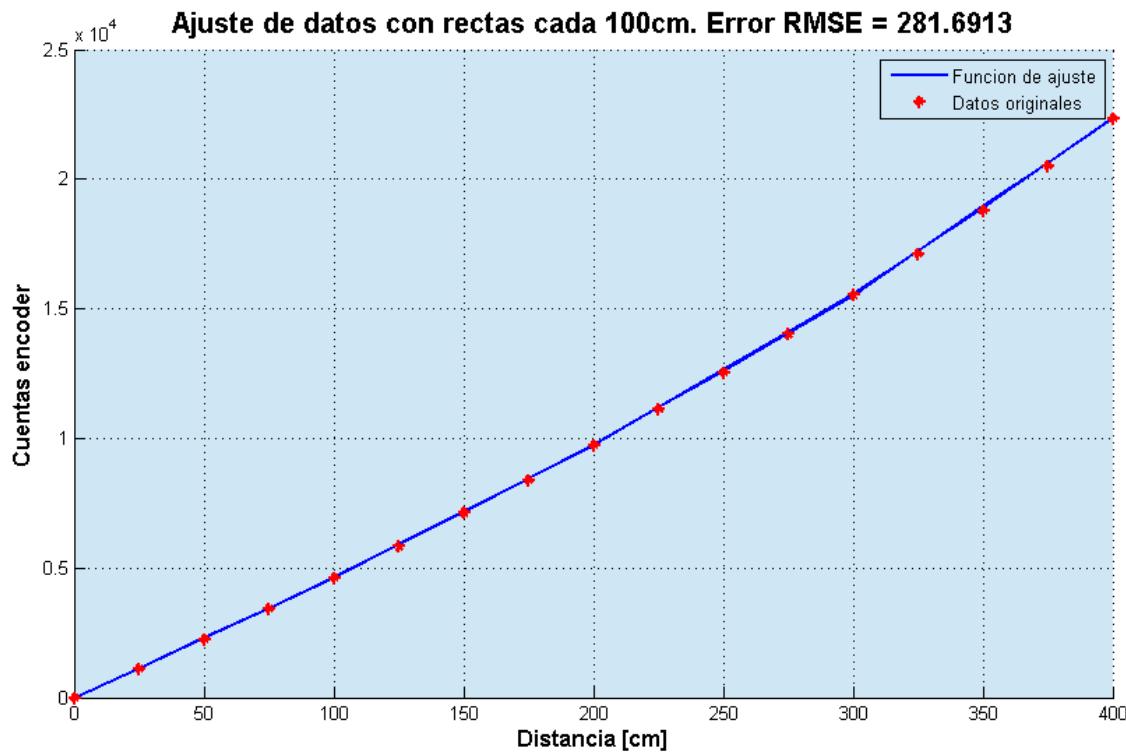


Figura 3.13: Ajuste mediante una recta cada 100cm

3.3.4 Obtención del modelo de la planta

3.3.4.1 Toma de muestras

El modelo de la planta a obtener tiene que ser tal que su salida sea de velocidad para poder diseñar el lazo de control interno de velocidad del modelo propuesto en la figura 2.1.

Para obtener este modelo se le inyecta al equipo la entrada mostrada en la figura 3.16, en donde la fuerza es positiva cuando el motor hace bajar la carga y negativa cuando la hace subir. La carga del equipo es de 3Kg

De esta entrada se midió la velocidad de salida en 2 pruebas separadas con el fin de obtener un set de datos para identificación y otro para validación. Estas respuestas se muestran en la figura 3.17.

3.3.4.2 Modelos propuestos

Para realizar la identificación de la planta se propusieron 12 modelos discretos de la forma $v(k) = \theta \cdot \phi(k)^T$:

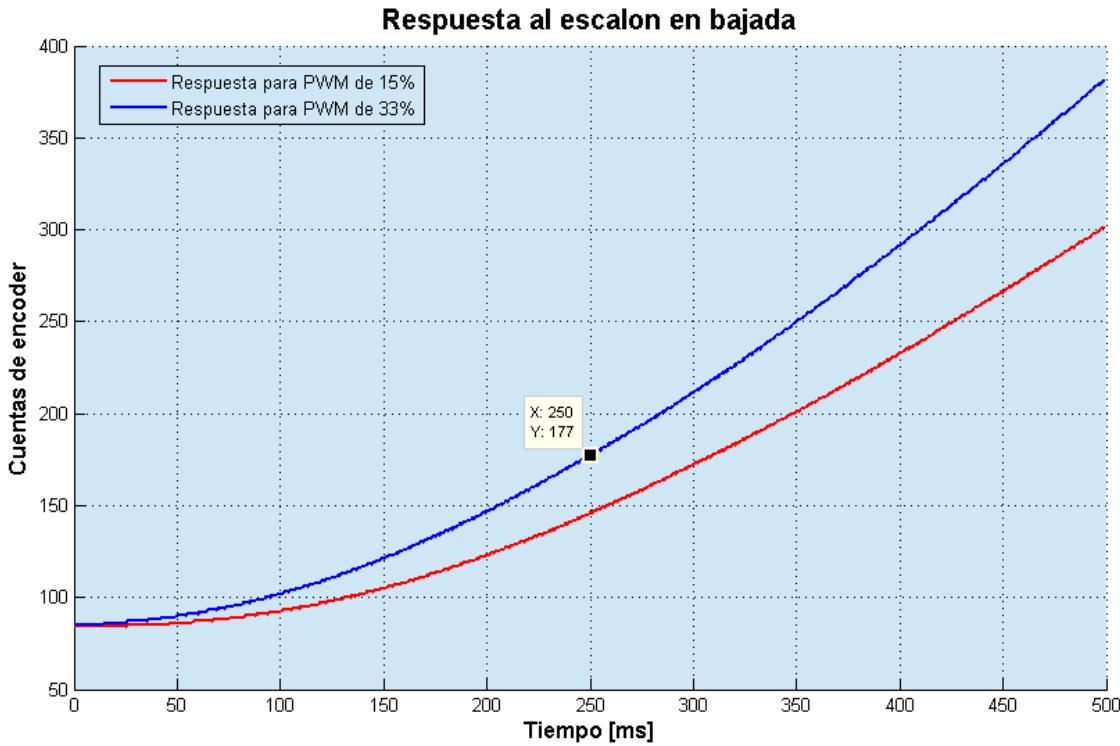


Figura 3.14: Respuestas del sistema en bajada

- Modelo 1: $y(k) = -a1 * y(k-1) + b1 * u(k-1) = [-a1, b1] * [y(k-1), u(k-1)]^T = b_0 \phi(k)^T$
- Modelo 2: $y(k) = -a1 * y(k-1) + b1 * u(k-1) + c$
- Modelo 3: $y(k) = -a1 * y(k-1) - a2 * y(k-2) + b1 * u(k-1)$
- Modelo 4: $y(k) = -a1 * y(k-1) - a2 * y(k-2) + b1 * u(k-1) + c$
- Modelo 5: $y(k) = -a1 * y(k-1) - a2 * y(k-2) + b1 * u(k-1) + b2 * u(k-2)$
- Modelo 6: $y(k) = -a1 * y(k-1) - a2 * y(k-2) + b1 * u(k-1) + b2 * u(k-2) + c$
- Modelo 7: $y(k) = -a1 * y(k-1) - a2 * y(k-2) - a3 * y(k-3) + b1 * u(k-1)$
- Modelo 8: $y(k) = -a1 * y(k-1) - a2 * y(k-2) - a3 * y(k-3) + b1 * u(k-1) + c$
- Modelo 9: $y(k) = -a1 * y(k-1) - a2 * y(k-2) - a3 * y(k-3) + b1 * u(k-1) + b2 * u(k-2)$
- Modelo 10: $y(k) = -a1 * y(k-1) - a2 * y(k-2) - a3 * y(k-3) + b1 * u(k-1) + b2 * u(k-2) + c$
- Modelo 11: $y(k) = -a1 * y(k-1) - a2 * y(k-2) - a3 * y(k-3) + b1 * u(k-1) + b2 * u(k-2) + b3 * u(k-3)$

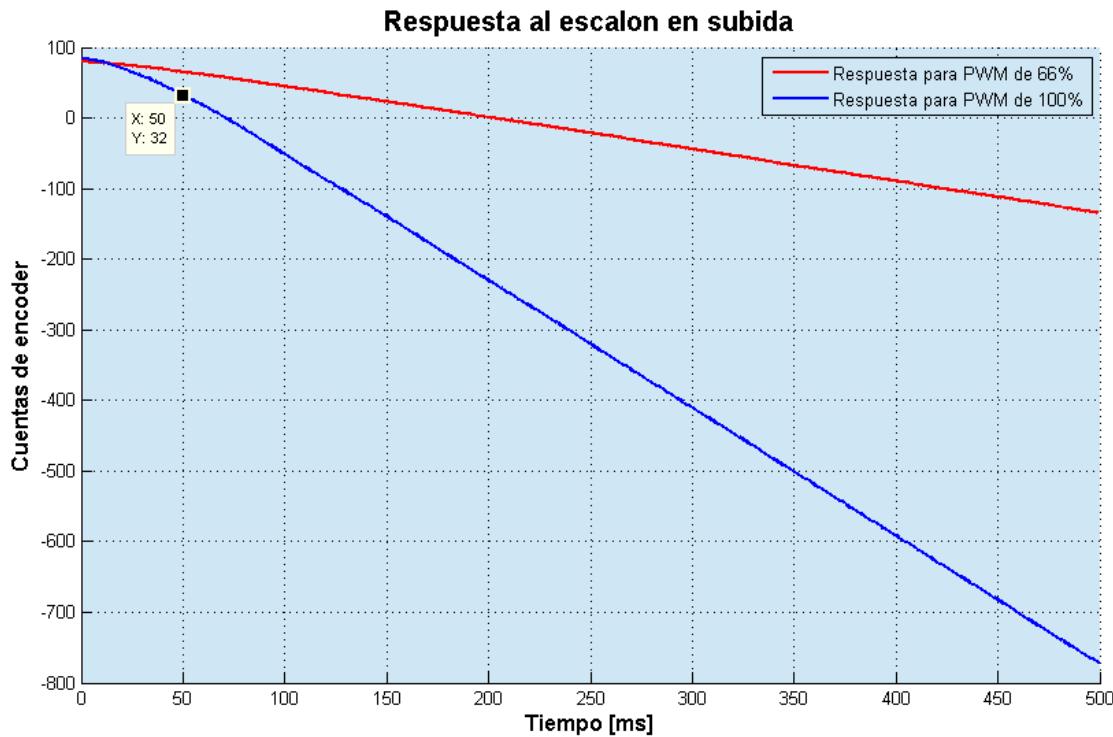


Figura 3.15: Respuestas del sistema en subida

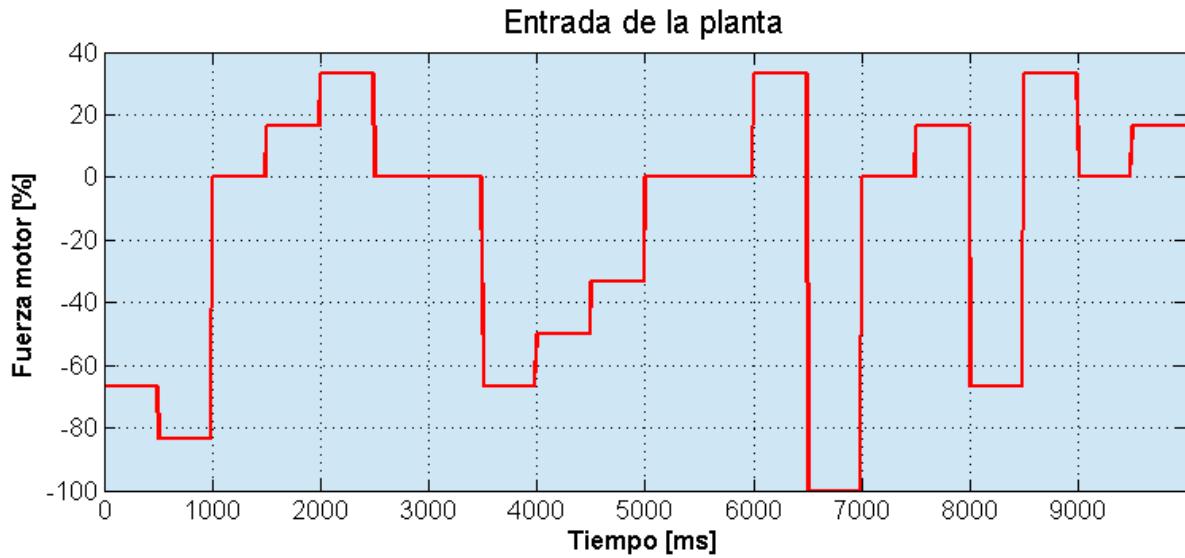


Figura 3.16: Entrada del sistema para la identificación de parámetros

- Modelo 12: $y(k) = -a1 * y(k-1) - a2 * y(k-2) - a3 * y(k-3) + b1 * u(k-1) + b2 * u(k-2) + b3 * u(k-3) + c$

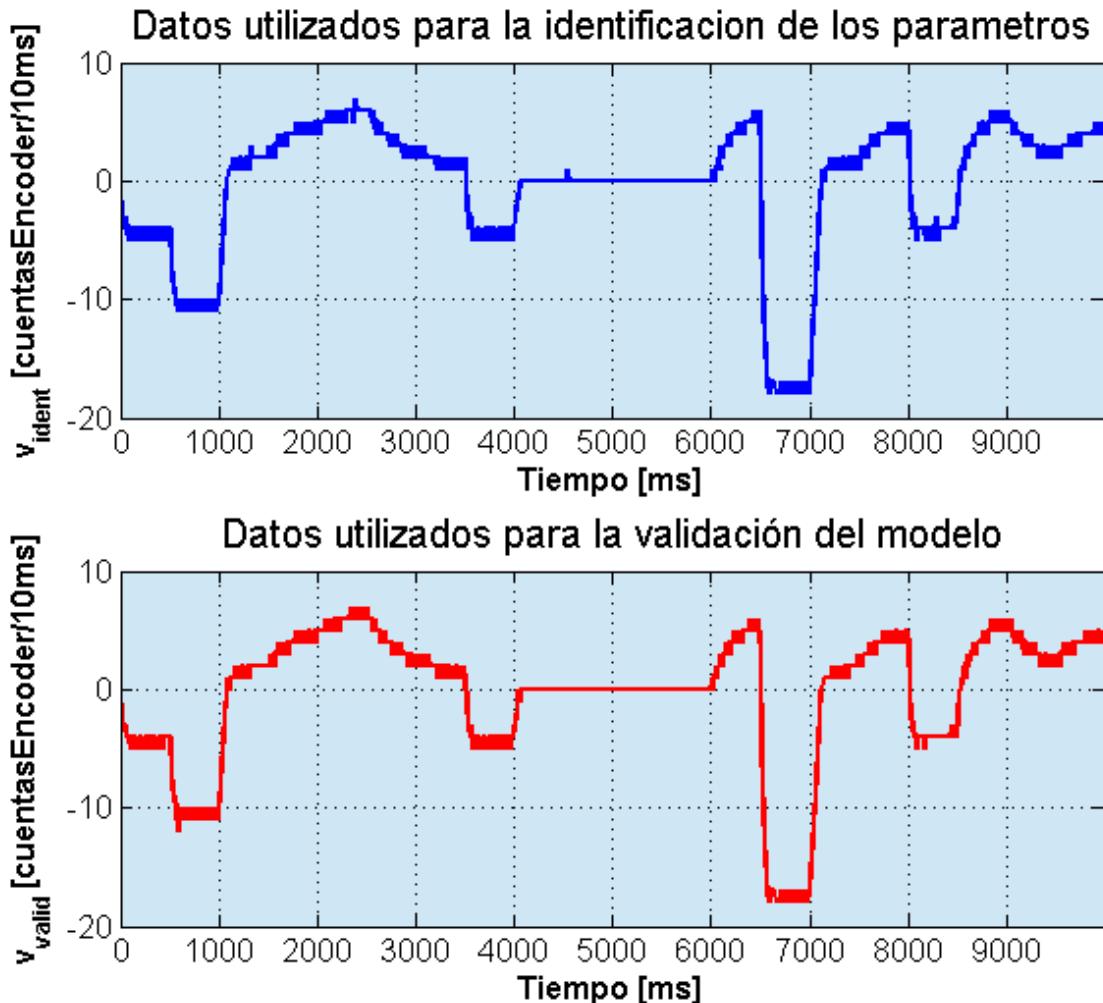


Figura 3.17: Salida de velocidad del sistema

3.3.4.3 Determinación de parámetros

La forma de determinar los parámetros es mediante un método de estimación de tipo offline basado en el principio de mínimos cuadrados tal que minimiza el funcional de error cuadrático $J = (\|\varepsilon\|^2)$. La solución al problema de mínimos cuadrados que minimiza el funcional J es la ecuación 3.4, en donde V y Φ se construyen con las $v(k)$ y $\phi(k)$ de los modelos discretos propuestos.

$$\hat{\theta} = (\Phi^T \Phi)^{-1} \cdot \Phi^T \cdot V \quad (3.4)$$

$$V = [v(1), v(2), \dots, v(\text{cantMuestras})]^T$$

$$\Phi = [\phi(1), \phi(2), \dots, \phi(cantMuestras)]^T$$

Con esta ecuación se obtuvieron los siguientes resultados de la tabla 3.3

Modelo	a1	a2	a3	b1	b2	b3	c
1	-0.9383	0	0	0.0028	0	0	0
2	-0.9183	0	0	0.0039	0	0	0.1643
3	-0.7233	-0.2289	0	0.0021	0	0	0
4	-0.6938	-0.2399	0	0.0032	0	0	0.1376
5	-0.6540	-0.2980	0	0.0118	-0.0094	0	0
6	-0.6201	-0.3115	0	0.0120	-0.0085	0	0.1520
7	-0.8079	-0.3914	0.2296	0.0011	0	0	0
8	-0.7946	-0.3851	0.2207	0.0017	0	0	0.0751
9	-0.7257	-0.4101	0.1701	0.0069	-0.0055	0	0
10	-0.7038	-0.4031	0.1551	0.0073	-0.0051	0	0.0964
11	-0.6485	-0.4890	0.1710	0.0122	-0.0043	-0.0062	0
12	-0.6230	-0.4819	0.1541	0.0123	-0.0040	-0.0057	0.1090

Tabla 3.3: Parámetros obtenidos para los 12 modelos propuestos

3.3.4.4 Validación de los modelos

Para validar los modelos se utiliza el set de datos de validación (figura 3.17). Simplemente se calcula iterando la salida del sistema utilizando los parámetros encontrados y se contrastan los resultados con este set de datos. Para medir el grado de relación entre los datos obtenidos con los modelos propuestos y los de validación se calcular el error RMSE para cada caso, resultados que se pueden ver en la tabla 3.4.

3.3.4.5 Análisis de resultados

De los resultados se puede ver que el error RMSE es muy grande, ya que la señal real del sistema toma amplitudes de como mucho 20, y el error es de por lo menos 3, un 15% de error (aprox). Esto que indica que la identificación no fue buena, por lo que no es recomendable diseñar los controladores basandose en los datos obtenidos.

Un ejemplo gráfico se puede ver en la figura 3.18 en donde se contrastan la salida predicha con los parámetros identificados para el modelo 2 (que es el que menor error RMSE tuvo), con los datos de validación.

Modelo	Error RMSE
1	3.6939
2	2.945
3	4.2602
4	3.452
5	4.0055
6	3.1642
7	4.461
8	3.8115
9	4.1403
10	3.4381
11	3.8415
12	3.084

Tabla 3.4: Error RMSE para los 12 modelos propuestos

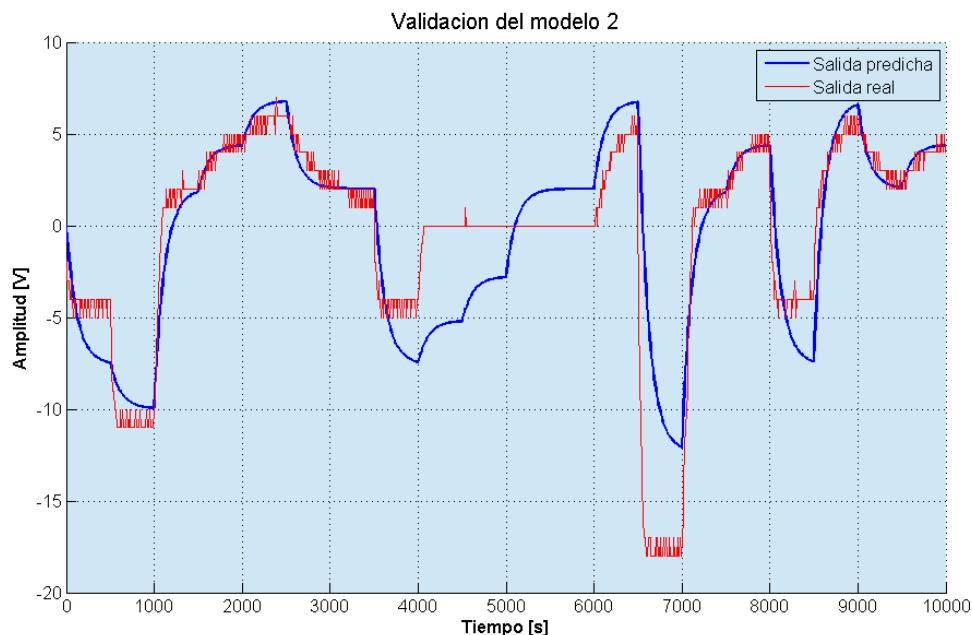


Figura 3.18: Comparación entre datos predichos con el modelo 2 y los datos reales

3.3.5 Obtención del controlador

3.3.5.1 Análisis

Como la planta identificada no se ajusta al modelo diseñar los controladores basado en ella no tiene mucho sentido. Como se mencionó en la sección 2.3, subsección 1, el controlador debe ser lo más simple posible, así que propondrán controladores estándar (PID) y se ajustarán empíricamente mediante prueba y error.

También es importante recordar que el Atmega328p no tiene optimizadas las operaciones de punto flotante, por lo que cualquier multiplicación por números no enteros tendrá que ser aproximada. En el caso de las divisiones se hará mediante la operación corrimiento (bit shift). Esto tiene como ventaja que se reduce mucho el uso del CPU, pero se pierde resolución y cambiar el código se hace complicado.

En cuanto al período de muestreo, se comenzará con un valor de 50ms y se ajustará dependiendo los resultados que se vayan obteniendo.

3.3.5.2 Controlador de velocidad

Basándose en la figura 2.1, el controlador de velocidad tiene como entrada el error de velocidad y como salida el ciclo de trabajo del pwm que irá al motor. Si bien el ciclo de trabajo no puede ser negativo, a términos prácticos un valor negativo indicaría que el motor sube la carga y uno positivo que la baja.

El rango de la variable de entrada es de ± 20 , siendo $+20$ velocidad de descenso y -20 de ascenso, como se vió en la sección 3.3 subsección 2, en donde se determinó la velocidad máxima. Por otro lado, el valor de la salida puede ser cualquiera, no necesariamente $\pm 100\%$ por ser un ciclo de trabajo, pero estará acotado por el limitador.

Basándose en esta información y en el esquema de control planteado en la sección 2.3 subsección 1, se propone como controlador de velocidad inicial la siguiente ecuación en diferencias:

$$u_Cv[k] = u_Cv[k - 1] + (ev[k] \gg 1) - (ev[k] \gg 2)$$

Donde $u[k]$ es la salida del controlador, $ev[k]$ el error de velocidad, y " $x \gg n$ " es el operador corrimiento, que equivale a $floor(x/2^n)$.

3.3.5.3 Controlador de posición

Basándose en la figura 2.1, el controlador de posición tiene como entrada el error de posición y como salida la referencia de velocidad.

El error de posición está relacionada con las cuentas del encoder del motor, por lo que su rango es de aproximadamente ± 20000 (basándose en la tabla 3.2). Por otro lado, el valor de la salida puede ser cualquiera, no necesariamente ± 20 por ser la velocidad máxima permitida, pero estará acotado por el limitador según lo que indique la referencia dada por DMX.

Basándose en esta información y en el esquema de control planteado en la sección 2.3 subsección 1, se propone como controlador de posición inicial la siguiente ecuación en diferencias:

$$rv_Cp[k] = (ep[k] >> 4)$$

Donde $rv_Cp[k]$ es la salida del controlador, $ep[k]$ el error de posición, y " $x \gg n$ " es el operador corrimiento, que equivale a $floor(x/2^n)$.

3.3.6 Ajuste de los controladores

3.3.6.1 Condiciones iniciales

Las condiciones iniciales para el sistema de control son un controlador de posición con la forma $rv_Cp[k] = ep[k] >> 4$, un controlador de velocidad con la forma $u_Cv[k] = u_Cv[k - 1] + (ev[k] >> 1) - (ev[k] >> 2)$, y un período de muestreo $T = 50\text{ms}$.

Como el objetivo es calibrar el controlador de velocidad, se mide su entrada (solamente la velocidad) y su salida (ciclo de trabajo del pwm después del limitador).

Para obtener estos datos se setea manualmente la referencias de velocidad en $40\% = 40\text{cuentas}/50\text{ms}$ ($100\% = 2\text{cuentas}/\text{ms}$) y la de posición que alterne entre 0 y 1 metro, cambiando apenas se alcance una. La carga utilizada tiene un peso de 3Kg.

Bajo estas condiciones se obtiene la respuesta de la figura 3.19, en donde se ve que la velocidad de la carga nunca llega a estabilizarse. Esto sucede en parte porque el el período de muestreo no ayuda a estabilizar el sistema, por lo que .

3.3.6.2 Ajuste del controlador de velocidad

Luego de varios procesos de ajuste del período de muestreo y controlador de velocidad se alcanzó una combinación cuyo desempeño es aceptable: un período de muestreo de 20ms y un controlador de velocidad de la forma $u_Cv[k] = u_Cv[k - 1] + (ev[k] >> 1)$.

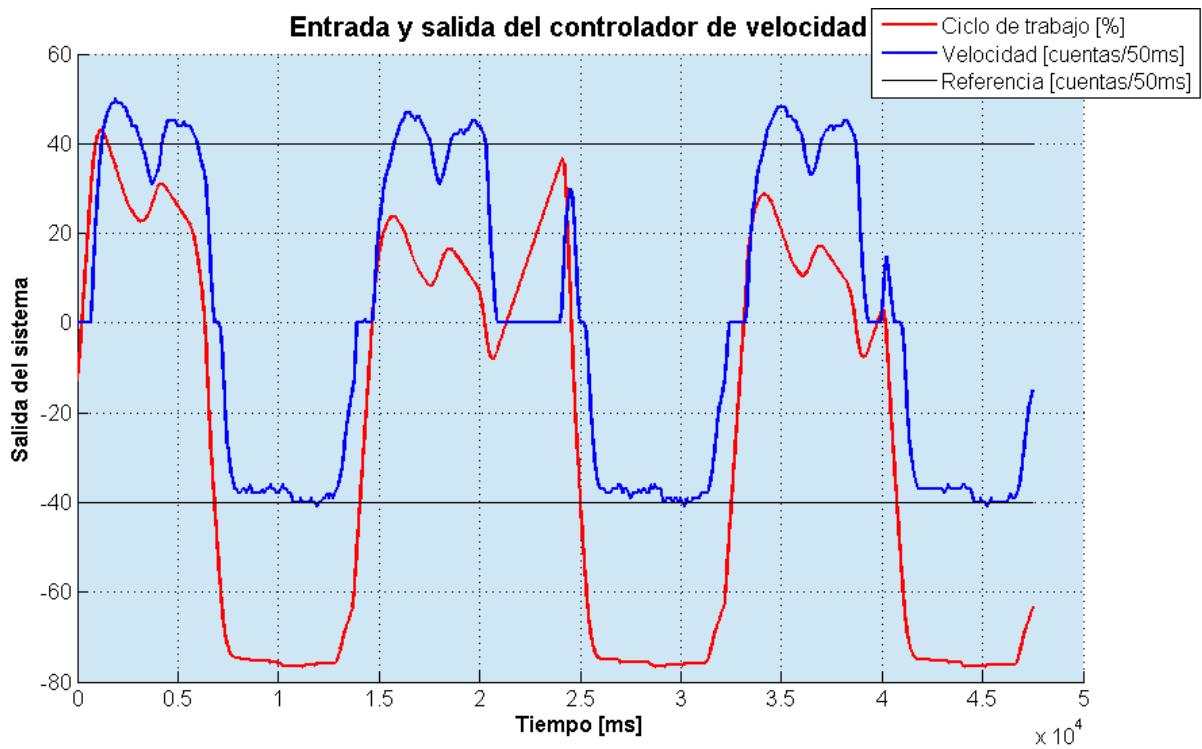


Figura 3.19: Resultados para el controlador de velocidad inicial

Para verificar esta nueva configuración se efectúa nuevamente la prueba con la referencia de velocidad al 40% y la de posición que alterna entre 0 y 1 metro. En este caso, sin embargo, se realizó la prueba con 2 equipos distintos, ambos con una carga de 3Kg. La respuesta del equipo 1 y 2 se presentan en las figuras 3.20 y 3.21, en donde se puede ver que ahora la referencia de velocidad se sigue exitosamente y de manera similar para ambos equipos, y que la salida del controlador no es tan errática.

3.3.6.3 Ajuste del controlador de posición

Una vez ajustado el controlador de velocidad se pasa a ajustar el de posición. Igual que en los casos anteriores se realiza una prueba con una referencia de velocidad del 40%, y una de posición que alterna entre 0 y 1 metro (que equivale a aproximadamente 4600 cuentas según la tabla 3.2). La diferencia en este caso es que se pasa a analizar la entrada (posición únicamente) y salida (referencia de velocidad rv) del controlador de posición.

Los resultados de esta prueba, conservando el controlador de posición inicial y utilizando el de velocidad y ciclo de trabajo ajustados, se pueden ver en la figura 3.22. Allí se observa que el controlador necesita un poco menos de ganancia para eliminar el sobre error.

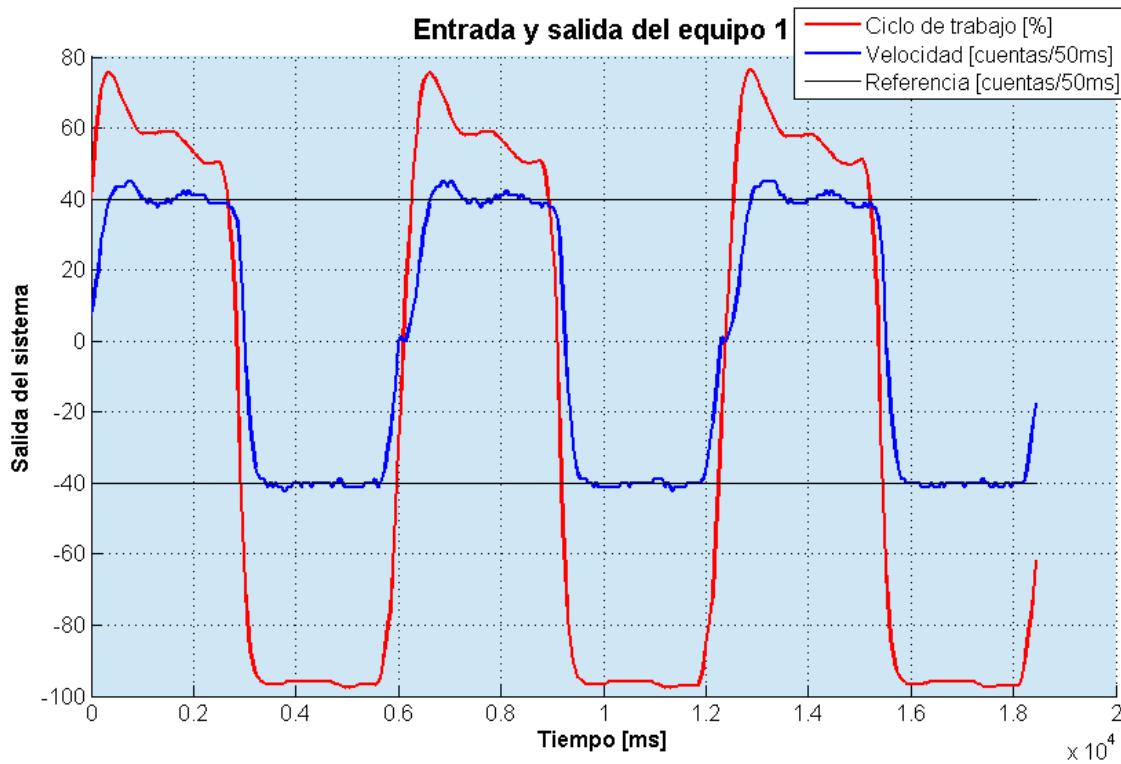


Figura 3.20: Resultados del equipo 1 para el controlador de velocidad final

Luego de varios procesos de ajuste del controlador se encontró que el siguiente controlador $rv_Cp[k] = (ep[k] \gg 4) - (ep[k] \gg 5)$ mejora el rendimiento del sistema, como se ve en la figura 3.23

3.3.6.4 Sistema de control final

En conclusión, el sistema de control consta de:

Período de muestreo: $T = 20ms$

Controlador de velocidad: $u_Cv[k] = u_Cv[k - 1] + (ev[k] \gg 1)$ (3.5)

Controlador de posición: $rv_Cp[k] = (ep[k] \gg 4) - (ep[k] \gg 5)$ (3.6)

3.4 Dipswitch

Como se mencionó en la sección 2.5, la obtención de los valores del dipswitch se hace mediante una simulación en Matlab.

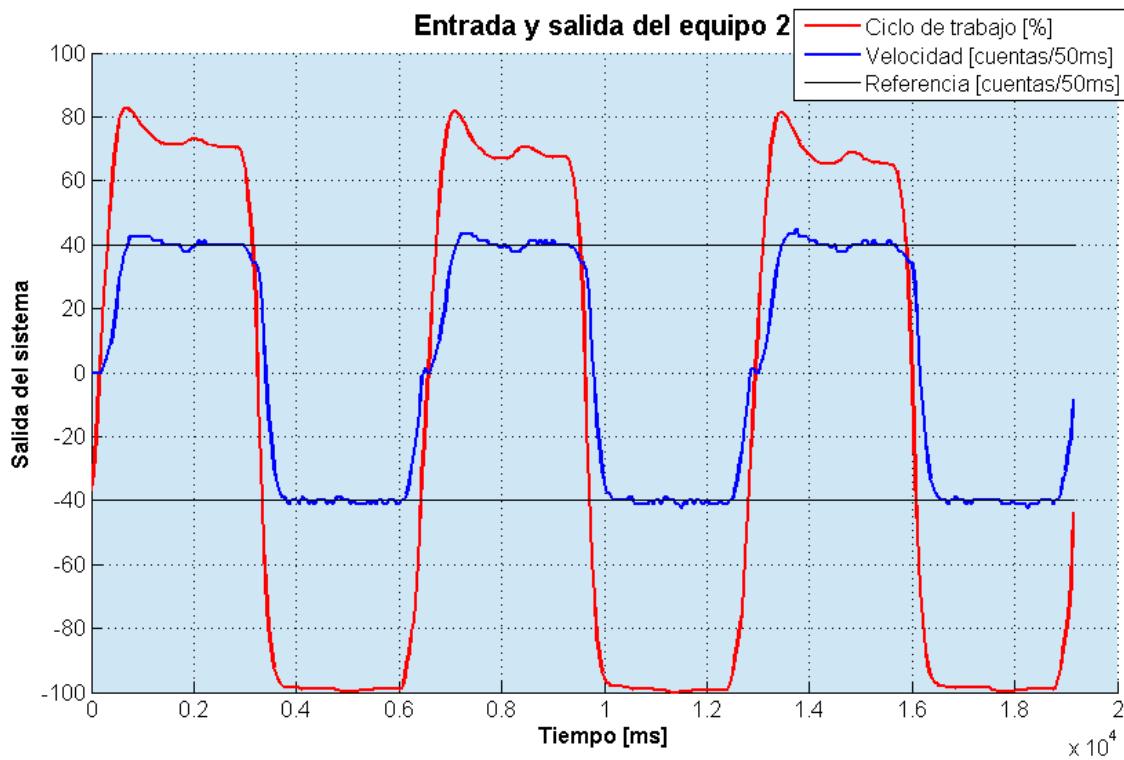


Figura 3.21: Resultados del equipo 2 para el controlador de velocidad final

El procedimiento para determinar una posible combinación de R1, R2, R3, R4 y R (ver figura 2.3) es el siguiente: se propone un valor para cada resistencia y se calcula la caída de tensión en la resistencia R, que llamaremos Vo, para cada combinación de R1, R2, R3 y R4. Como son 4 resistencias hay 16 posibles combinaciones, por lo que se obtendrán 16 valores de Vo.

Ahora, lo importante es encontrar R1, R2, R3, R4 y R tales que los 16 valores de Vo encontrados sean distintas y estén lo más separados posibles

Como el método propuesto es de prueba y error se establece una diferencia mínima que todas las Vo deben tener entre sí. Como el canal analógico del microcontrolador tiene una resolución de 10 bits (1024) y el rango de tensión de entrada va de 0 a 5V el paso más chico que puede medir el ADC es de $5V/1024 \approx 5mV$. Entonces, se elige como diferencia mínima 10 unidades del ADC, que equivalen a aproximadamente 50mV.

Para determinar los valores se usó como tensión de alimentación Vcc (referirse a la figura 2.3) 5V, ya que es la tensión de trabajo de la placa de control, y de varios intentos de prueba y error se encontró que para la combinación de resistencias R1=10KΩ, R2=4,7KΩ, R3=2,2KΩ, R4=1KΩ y R=1KΩ la mínima diferencia entre todas las Vo es de 67mV.

En la tabla 3.5 se presentan los valores analógicos resultantes de todas las combinaciones

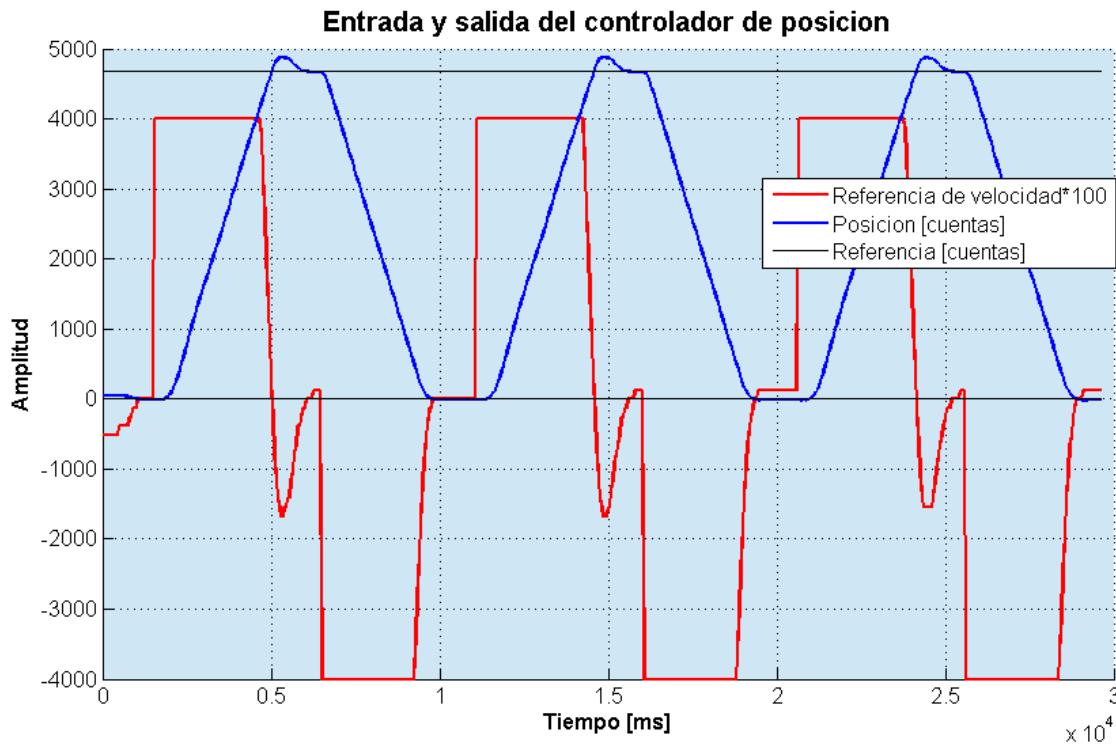


Figura 3.22: Resultados para el controlador de posición inicial

posibles de las resistencias encontradas. La primera fila de dicha tabla muestra los bits, ordenados de mas significativo (msb) a menos significativo (lsb), que equivalen al estado de R4 (msb),R3,R2 y R1 (lsb). Por ejemplo, si el switch de R4 está activado el bit msb estará en 1. En la segunda fila se muestra el valor analógico calculado teóricamente con el script de Matlab.

Con el objetivo de verificar que los valores analógicos obtenidos son lógicos se armó un dipswitch con estos valores de resistencia y se midió su salida haciendo uso del módulo ADC, datos que se ven en la fila 3 de la tabla.

bits (R4,R3,R2,R1)	0000	0001	0010	0011	0100	0101	0110	0111
Valor analógico calculado	0	93	179	243	320	365	409	444
Valor analógico real	0	91	177	241	320	365	409	444
bits (R4,R3,R2,R1)	1000	1001	1010	1011	1100	1101	1110	1111
Valor analógico calculado	512	536	561	581	606	623	640	653
Valor analógico real	510	535	559	579	606	622	639	653

Tabla 3.5: Valores analógicos teóricos y reales para cada combinación de R4,R3,R2,R1

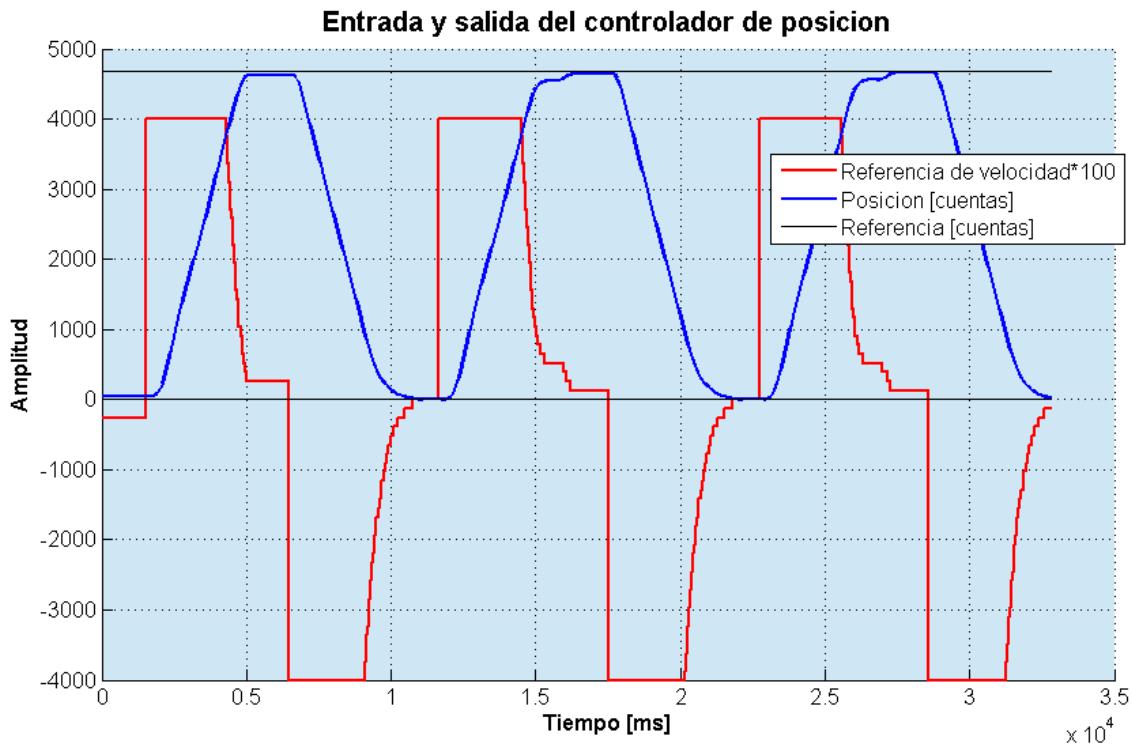


Figura 3.23: Resultados para el controlador de posición final

3.5 Firmware del updown - Librerías de alto nivel

3.5.1 Encoder

3.5.1.1 Objetivo

La función de este módulo es implementar la lógica de cuenta de los encoders para determinar cuál es la posición angular de los mismos.

3.5.1.2 Desarrollo

Para leer los encoders se utilizan las interrupciones del módulo de bajo nivel EXINT. Por como están conectado los componentes electrónicos en la placa de control, el encoder del motor está asociado a las EXINT y el encoder del disco a las PCINT.

Ambos 2 encoders, el de motor y el de disco, son incrementales de tipo AB. Estos cuentan con 2 canales (A y B) que son en esencia 2 entradas digitales que físicamente están separadas de tal manera que el tren de pulsos que generan se encuentran desfasadas

en el tiempo.

Para entender el funcionamiento de los encoders incrementales AB se describe como funciona el del disco en la figura 3.24. Allí se ve que el disco tiene marcas que los sensores A y B, asociados a los canales A y B, van detectando a medida que este gira. Estas marcas son lo suficientemente grandes como para que algún momento ambos sensores estén activados (válido para ambos encoders). Cuando un sensor detecta una marca el canal se pone en HIGH y cuando no, en LOW.

Basandose en la imágen 3.24 se puede ver que el orden en que se activan los canales A y B depende del sentido de giro. Si el disco gira en sentido horario primero se activa el canal A y luego en B, mientras que lo contrario pasa en sentido antihorario.

Siguiendo este patrón se implementó la siguiente lógica de cuenta para ambos encoders:

1. Se asocia al canal A una interrupción por cambio en el estado de un pin.
 2. Cuando el evento de cambio de pin ocurre y la interrupción se activa se verifica si el cambio del canal A fue de HIGH a LOW o viceversa.
 3. Luego leo el estado del canal B para determinar si el sentido de giro es horario y anti horario. Las posibles combinaciones de estados se resumen en la tabla 3.6.

	Canal B en HIGH	Canal B en LOW
Canal A de HIGH a LOW	Sentido horario	Sentido antihorario
Canal A de LOW a HIGH	Sentido antihorario	Sentido horario

Tabla 3.6: Sentido según el estado del canal B durante las transiciones del canal A

Físicamente si el disco gira en sentido horario la carga baja, mientras que cuando gira en sentido antihorario el cable se enrrolla. Como la posición crece cuando la caja baja en los casos en que se detecta un cambio que indique que el sentido de giro es horario las cuentas del encoder se incrementan, y se decrementan en caso contrario.

Por lo tanto, la implementación en C de la lógica de cuentas sería:

```
1 void funcionEjecutadaDuranteLaInterrupcionPorCambioDePin (void) {
2     if ( Canal_A == HIGH ) { // El canal A cambio de LOW a HIGH
3         if ( Canal_B == LOW ) { // Leo el canal B
4             cuentasEncoder++; // Canal B = LOW => Sentido horario (la carga baja)
5         } else {
6             cuentasEncoder--; // Canal B = HIGH => Sentido antihorario (la carga
7             sube)
8         }
9     } else { // El canal A cambio de HIGH a LOW
10        if ( Canal_B == LOW ) { // Leo el canal B
11            cuentasEncoder++; // Canal B = LOW => Sentido horario (la carga
12            sube)
13        } else {
14            cuentasEncoder--; // Canal B = HIGH => Sentido antihorario (la carga
15            sube)
16        }
17    }
18 }
```

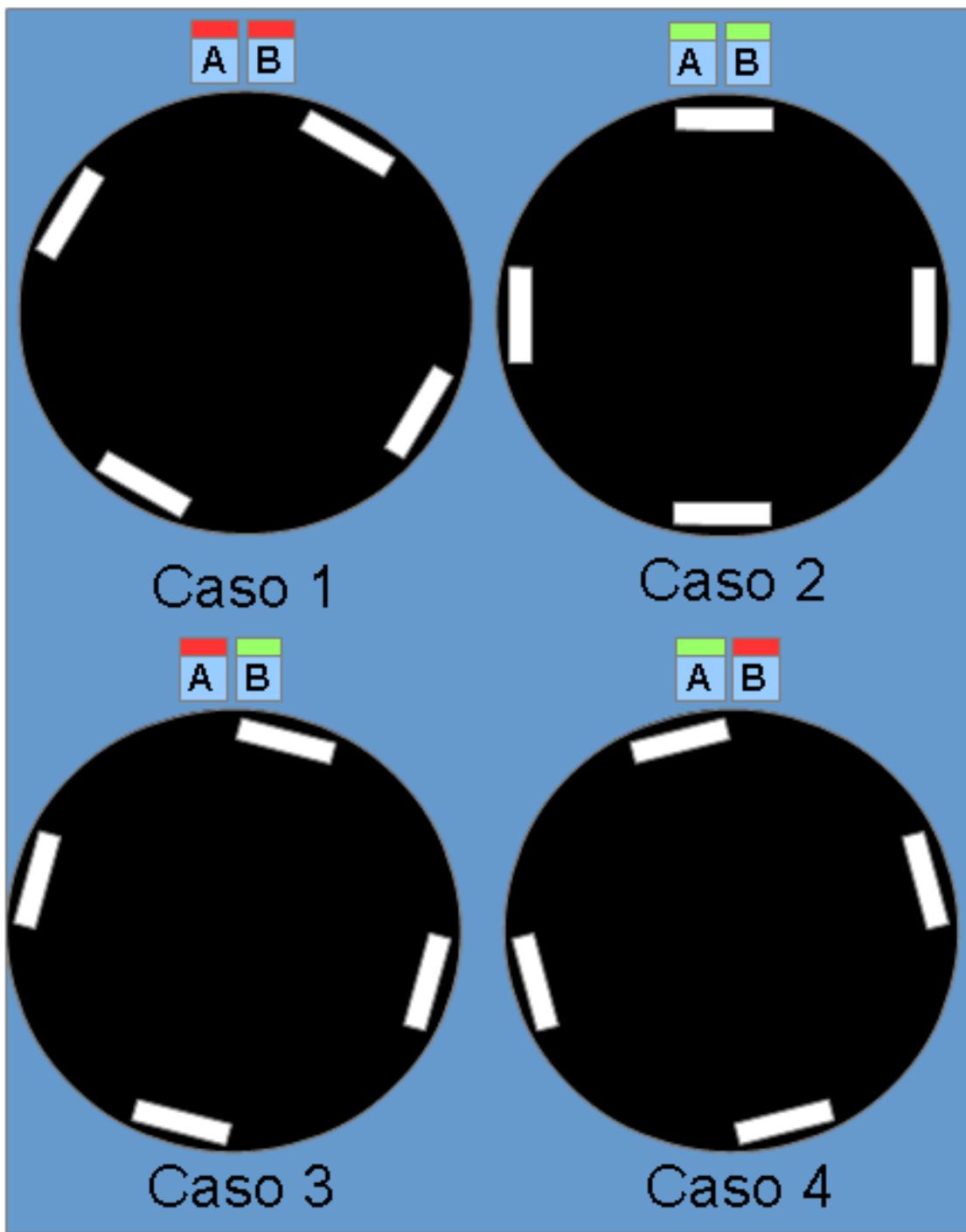


Figura 3.24: Ejemplo estado de los canales AB del encoder de disco

10 cuentasEncoder--; // Canal B = LOW => Sentido antihorario (la carga sube)

```

11 } else {
12     cuentasEncoder++; // Canal B = HIGH => Sentido horario (la carga baja
13 }
14 }
15 }

```

3.5.1.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.25. A continuación se presenta la descripción de cada una:

- **Encoder_init()**, que habilita el periférico EXINT e inicializa las cuentas para ambos encoder de motor y de disco.
- **Encoder_read(encoder)**, para leer las cuentas de uno de los 2 encoders. Se puede seleccionar cuál haciendo uso de los macros Encoder_MOTOR y Encoder_DISCO.
- **Encoder_write(encoder,cuentas)**, para escribir las cuentas de uno de los 2 encoders.

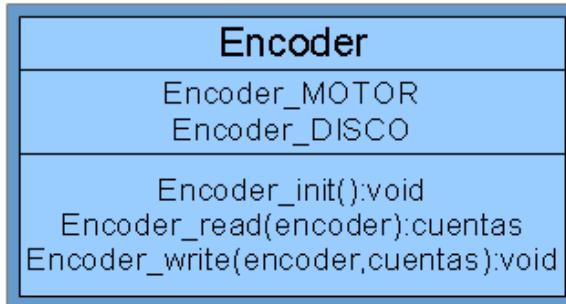


Figura 3.25: Diagrama del módulo Encoder

3.5.2 Grua

El módulo Grua es simplemente un paquete que agrupa el uso de elementos del equipo updown que no requieren mucha inteligencia. Entre estos se encuentran:

- Los leds internos y externo, que al ser salidas digitales solo necesitan una función de escritura cada uno.

- El motor, del cual se puede controlar la velocidad mediante el PWM, y el sentido de giro.
- El freno, que al ser una salida digital solo necesita una función de escritura.
- El fin de carrera, que al ser una entrada digital solo necesita una función de lectura.

El acceso al módulo se realiza a través 6 funciones, como se muestra en la figura 3.26. A continuación se presenta la descripción de cada una:

- **Grua_init()**, que inicializa, a través del módulo DigitalIO, los pines asociados al hardware que este módulo maneja. También se inicializa el módulo PWM para manejar la velocidad del motor.
- **Grua_read_fdc()**, para leer el estado del fin de carrera.
- **Grua_write_...(estado)**, para cambiar el estado de dispositivos como el led interno, led externo y freno.
- **Grua_write_pwm(cicloTrabajo)**, para setear el ciclo de trabajo del pwm.
- **Grua_write_puenteH(direccionCarga)**, para seleccionar si la carga sube o baja. Para facilitar esto están las macros Grua_CARGA_SUBIR y Grua_CARGA_BAJAR.

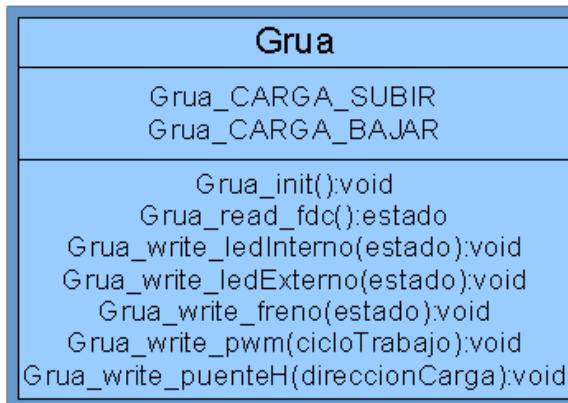


Figura 3.26: Diagrama del módulo Grua

3.5.3 Controlador

3.5.3.1 Objetivo

El módulo de controlador tiene que encargarse de convertir comandos de DMX en referencias para los controladores y calcular los mismos para mover al motor.

3.5.3.2 Seteo de referencias

Lo primero es convertir los comandos DMX en referencias. Se tienen 2 comandos, ambos de 8 bits, uno de posición y otro de velocidad.

El comando DMX de posición indica la distancia lineal que tiene que recorrer la carga, de 0 a 4 metros, por lo que se debe hacer es convertir esta información a distancia angular, o sea, a cuentas de encoder. Para hacer esto se utilizarán 4 rectas que ajusten datos como los de la tabla 3.2. La implementación en psudocódigo de C sería:

```

1  uint16_t conversionPosicionDMXaPosicionEncoder (uint8_t posicion_dmx) {
2      uint16_t posicion_cm = 0, posicion_referencia = 0;
3      /* —— Convierto posicion DMX a posicion lineal —— */
4      posicion_cm = ((uint16_t)posicion_dmx)*400/255;
5
6      /* —— Convierto posicion lineal a posicion angular —— */
7      if(0 < posicion_cm <= 100){
8          posicion_referencia = f1(posicion_cm);
9          // f1 = recta de ajuste para las posiciones de 0 a 1 metro
10     }
11    if(100 < posicion_cm <= 200){
12        posicion_referencia = f2(posicion_cm);
13        // f2 = recta de ajuste para las posiciones de 1 a 2metros
14    }
15    if(200 < posicion_cm <= 300){
16        posicion_referencia = f3(posicion_cm);
17        // f3 = recta de ajuste para las posiciones de 2 a 3metros
18    }
19    if(300 < posicion_cm <= 400){
20        posicion_referencia = f4(posicion_cm);
21        // f4 = recta de ajuste para las posiciones de 3 a 4metros
22    }
23    return(posicion_encoder);
24 }
```

Luego se tiene el comando DMX de velocidad, que tiene que ser mapeado de 0 a la velocidad máxima del equipo. Ahora, la velocidad puede ser positiva o negativa, en donde positiva quiere decir que el equipo debe bajar y negativa que debe subir. El signo se determina comparando la posición actual y la de la referencia de posición. Si la referencia de posición es menor a la posición actual la carga debe subir, por lo que la velocidad será negativa. La implementación en psudocódigo de C sería:

```

1  int16_t conversionVelocidadDMXaVelocidadEncoder (uint8_t velocidad_dmx) {
2      int16_t velocidad_referencia = 0;
3
4      /* —— Convierto velocidad DMX a velocidad angular —— */
5      referencia_velocidad = velocidad_dmx*VELOCIDAD_MAXIMA/255;
6      // Este resultado es siempre positivo
7
8      /* —— Determino el signo de la velocidad —— */
9      if(referenciaDePosicion - posicionActual > 0){
```

```

10    // La carga esta por debajo de la referencia:
11    // Entonces tengo que subir => la velocidad es negativa
12    referencia_velocidad = -referencia_velocidad;
13 } else {
14    // La carga esta por encima de la referencia:
15    // Entonces tengo que bajar => la velocidad es positiva
16    // La referencia ya es positiva, asi que no hago nada
17 }
18 return(velocidad_referencia);
19 }
```

3.5.3.3 Cálculo de los controladores

El siguiente paso es implementar los controladores, que tienen que actualizar su estado cada $T = \text{período de muestreo}$.

La manera más simple y modular de hacer esto es crear una función que actualice los valores de los controladores. Luego, esta función será llamada por otro módulo, el de aplicación, que se encargará de temporizar todas las tareas, desasociando el manejo de tiempo del módulo Controlador.

La forma de los controladores de posición y velocidad se hallaron en la sección 3.1, y se pueden ver en las ecuaciones 3.5 y 3.6. La implementación en pseudocódigo de C, siguiendo el esquema de la figura 2.1, sería:

```

1 void actualizacionControladores(void){
2
3     /* — Calculo el controlador de posicion — */
4     posicion[k-1] = posicion[k]; // me guardo la posicion pasada
5     posicion[k] = Encoder_read(Encoder_MOTOR); // leo la posicion actual
6     ep[k] = posicion_referencia - posicion[k]; // Calculo el error de posicion
7     rv_Cp[k] = (ep[k] >> 4) - (ep[k] >> 5); // Calculo el controlador de
9      posicion
8
9     /* — Aplico el limitador de la salida del controlador de posicion — */
10    if(rv_Cp[k] > referencia_velocidad_dmx){
11        rv = referencia_velocidad_dmx;
12    } else {
13        rv = rv_Cp[k];
14    }
15
16    /* — Calculo el controlador de velocidad — */
17    velocidad[k] = posicion[k] - posicion[k-1]; // Calculo velocidad actual
18    ev[k] = velocidad_referencia - velocidad[k]; // Calculo el error de
19      velocidad
20    u_Cv[k] = u_Cv[k-1] + (ev[k] >> 1); // Calculo el controlador de
      velocidad
21
22    //u_Cv puede ser positiva o negativa
```

```

21  /* —— Aplico el limitador de la salida del controlador de velocidad ——
22  */
23  if( |u_Cv[k]| > limite_actuador){
24      u = signo(u_Cv[k])*limite_actuador;
25  } else {
26      u = u_Cv[k];
27  }
28
29  /* —— Actuo sobre el motor —— */
30  // u puede ser positiva o negativa.
31  // Si es positiva tengo que bajar la carga, y si es negativo subirla.
32  // Ademas, el ciclo de trabajo siempre es positivo
33  if(u < 0){
34      Grua_write_puenteH(Grua_CARGA_SUBIR);
35      Grua_write_pwm(-u);
36  } else {
37      Grua_write_puenteH(Grua_CARGA_BAJAR);
38      Grua_write_pwm(u);
39  }
40 }
```

3.5.3.4 Acceso

El acceso al módulo se realiza a través 3 funciones, como se muestra en la figura 3.27. A continuación se presenta la descripción de cada una:

- **Controlador_init()**, que inicializa todas las variables de los controladores.
- **Controlador_update(encoder)**, que actualiza los controladores de posición y velocidad. El tiempo que debe transcurrir entre una actualización y otra, el período de muestreo, puede ser accedido a través del macro Controlador_TIEMPO_UPDATE.
- **Controlador_write(posicionDMX,velocidadDMX)**, para escribir las cuentas de uno de los 2 encoders.

3.5.4 Dipswitch

3.5.4.1 Objetivo

La función de este módulo es inferir qué llaves del dip-switch están activados a partir de su salida de tensión, que se traduce a valores analógicos en la placa.



Figura 3.27: Diagrama del módulo Controlador

3.5.4.2 Desarrollo

Para lograr esto simplemente se lee el valor analógico del dipswitch y utilizando la tabla 3.5 se determina qué resistencias están activadas, equivalente a la posición de los switchs. Como estos valores tienen, como cualquier señal, un nivel de ruido de ± 3 unidades (aproximadamente 15mV) se considera válida una combinación de valores de resistencias dentro de un rango de valores analógicos. Por ejemplo, se considerará que la combinación es 0001 si el valor analógico leído está dentro del rango $round([(91 - 0)/2, (177 - 91)/2])$. Estos valores no son arbitrarios, si no que es el promedio entre el valor analógico asociado a una combinación dada y los valores superior e inferior a este. Para el caso de 0001, el valor analógico asociado es 91, y el inferior y superior son 0 y 177, asociados a las combinaciones 0000 y 0010, respectivamente.

De esta manera se implementa una función de lectura simplemente mirando en qué rango cae el valor analógico leido. La implementación en psudocódigo de C sería:

```

1 if( 0 <= valor_analogico_dipswitch < (91-0)/2 ){
2   combinacion_dipswitch = 0000;
3 }
4 if( (91-0)/2 <= valor_analogico_dipswitch < (177-91)/2 ){
5   combinacion_dipswitch = 0001;
6 }
7 ...
8 if( (91-0)/2 <= valor_analogico_dipswitch < (177-91)/2 ){
9   combinacion_dipswitch = 0001;
10 }
11 if( (640-623)/2 <= valor_analogico_dipswitch < (653-640)/2 ){
12   combinacion_dipswitch = 1110;
13 }
14 if( (653-640)/2 <= valor_analogico_dipswitch <= 1023 ){
15   combinacion_dipswitch = 1111;
16 }
  
```

Como el dipswitch tiene 10 llaves, no 4, que entregan un total de 3 salidas analógicas simplemente se replica esta función para cada una de ellas y se concatenan las combinaciones obtenidas. La única consideración a tomar es que 2 de las 3 salidas están asociadas

a 2 llaves, no a 4, por lo que las combinaciones de ellas serán de 3 bits. Por ejemplo, si los valores analógicos de los 3 canales leidos son 559,91,444, la combinación de llaves asociada será 1010 001 111.

3.5.4.3 Acceso

El acceso al módulo se realiza a través 2 funciones, como se muestra en la figura 3.28. A continuación se presenta la descripción de cada una:

- **Dipswitch_init()**, que inicializa el periférico ADC.
- **Dipswitch_read()**, para leer el estado de los switchs.



Figura 3.28: Diagrama del módulo Dipswitch

3.5.5 DMX

3.5.5.1 Objetivo

La función de este módulo es leer los datos DMX que se reciben de un equipo DMX externo.

Para lograr esto se implementa un decodificador del formato de la trama de DMX que se presentó en la figura 1.8 de la sección 1.3, subsección 3 (capa de enlace de datos). Como en DMX se envían 512 canales con información y el Updown solo necesita 2, la referencia de posición y velocidad, también se filtran los datos sin importancia con el objetivo de optimizar el uso de memoria (se guardan 2 bytes en vez de 512).

3.5.5.2 Desarrollo

La solución que se aplicó para resolver esto fue ejecutar una función durante la interrupción por recepción de la UART. Cabe aclarar que, como se explicó en la sección 3.2,

subsección 5, el módulo UART se inicializa con un baudrate de 250KHz y formato de trama 8N2, por lo que cumple con la especificación del estándar DMX. La función ejecutada actualiza la máquina de estados de la figura 3.29. En esta MEF . .

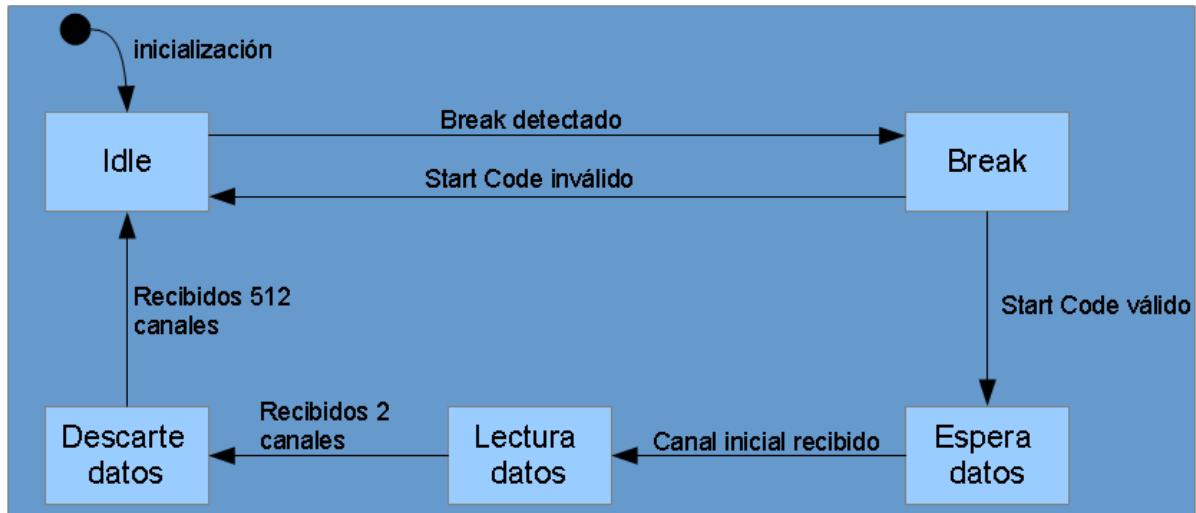


Figura 3.29: Maquina de estados ejecutada en cada interrupción de la UART

A continuación se detalla el funcionamiento de la MEF:

- El estado inicial es el Idle, igual que en la trama DMX, en donde simplemente se espera sin hacer nada a que llegue un break.
- Cuando un break es detectado se verifica que el primer dato recibido sea el Start Code (0x00). Si no lo es, hubo un error y se vuelve al estado Idle, pero si lo es se comienzan a escuchar los canales DMX.
- Como se explicó anteriormente, de los 512 canales solo importan 2, y a partir de cuál canal tomo los 2 es determinado mediante el dipswitch. Justamente, la función del dipswitch en el sistema es seleccionar el canal inicial a leer, a partir del cual se leerán las referencias de posición y velocidad. Entonces, se espera hasta llegar al canal seteado por dip-switch en el estado Espera Datos, dejando pasar todos los datos innecesarios.
- Una vez recibido el canal inicial se guardan ese canal y el próximo, que serán las referencias de posición y velocidad, respectivamente. Adicionalmente, en esta etapa se guarda el tiempo, obtenido del módulo Tick, en el que se recibieron los datos para poder determinar cuándo la señal de DMX fue perdida.
- Luego se descartan el resto de los canales (hasta llegar al 512) y se vuelve al estado Idle, a la espera de una nueva trama.

De esta manera con la inicialización del módulo, que implica la inicialización del módulo UART, siempre que el master DMX envíe una trama de 512 bytes el módulo de DMX se encarga de guardar los datos que el sistema necesita. Para acceder a ellos se implementa una función de lectura en donde el usuario selecciona uno de 2 datos disponibles (posición o velocidad). Además se implementa una función para setear el canal inicial de lectura, para independizar este módulo del de Dipswitch.

3.5.5.3 Acceso

En conclusión, el acceso al módulo se realiza a través 3 funciones, como se muestra en la figura 3.30. A continuación se presenta la descripción de cada una:

- **DMX_init()**, que inicializa el periférico UART con la función que actualiza la máquina de estados de la figura 3.29 y que pone por defecto el canal inicial en 1.
- **DMX_read(canal)**, para recuperar los datos de posición y velocidad, accesibles a través de los macros DMX_CANAL_POSICION y DMX_CANAL_VELOCIDAD.
- **DMX_write(canalInicial)**, para seleccionar el canal inicial
- **DMX_estaActivo()**, que indica mediante un LOW si transcurrió más de un segundo desde la última vez que datos válidos fueron recibidos, y HIGH en caso contrario. Con esto se puede detectar la pérdida de DMX.

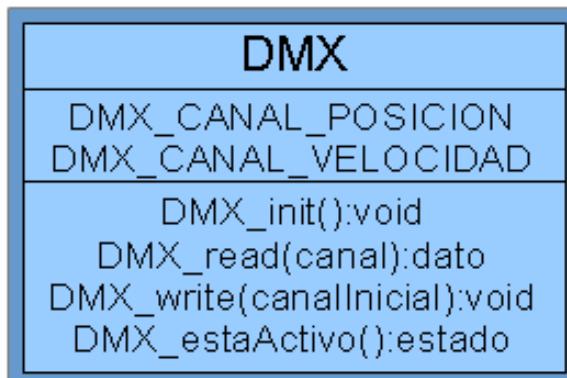


Figura 3.30: Diagrama del módulo DMX

3.6 Firmware del updown - Capa de Aplicación

3.6.1 Objetivo

La función de la capa de aplicación, la capa más alta de la figura 2.4 es hacer uso de las librerías desarrolladas para hacer que el Updown se comporte como es deseado, o sea, que cumpla con los objetivos presentados en la sección 1.5.

3.6.2 Desarrollo

La capa de aplicación es la que tiene la función principal del programa, la función *main*. Dentro de ella, como en cualquier firmware para sistemas embebidos, hay 2 grandes secciones: la sección de inicialización o *setup* y la del bucle principal, o "super loop". Adicionalmente, como hay acciones que deben suceder cada un tiempo definido se hace uso de un esquema planificador-despachador, siguiendo la arquitectura *time-triggered cooperativa*.

3.6.2.1 Inicialización del sistema

Dentro de la sección de inicialización simplemente se habilitan todos los módulos que el sistema necesite para funcionar con su configuración correspondiente. Esto incluye los módulos: Encoder, Grua, Tick, Dipswitch y DMX, que además se configura para tener el address indicado por el dipswitch. Adicionalmente, se asocia al módulo Tick el planificador de tareas, sobre la cual se desarrollará más adelante.

El módulo SUART, por otro lado, solo se utilizó para la etapa de pruebas y debug. En el firmware final del equipo no se inicializa ya que no hay nada que depurar y solo deterioraría el rendimiento del sistema a causa de los recursos que consume.

3.6.2.2 Planificador y despachador de tareas

En este proyecto hay varias tareas que tienen que ser ejecutadas periódicamente, como por ejemplo la actualización del controlador, que se hace cada 20ms. Para resolver esto se implementa el esquema de planificador-despachador, en donde el planificador determina el momento en el que ciertas tareas están listas para ejecutarse y el despachador las ejecuta cuando ese momento.

En esta aplicación al planificador se lo ejecuta durante cada interrupción del Tick. Como cada tick ocurre cada 1 milisegundo, sencillamente se cuentan los ticks y se habilita

la ejecución de las tareas que correspondan cuando se alcance el tiempo de ejecución asociada a ellas. Por ejemplo, la actualización del controlador se activa una vez cada 20 ticks (que equivalen a 20ms).

Para evitar el solapamiento de tareas se hace uso de un retardo que se activa únicamente en la primera ejecución de la tarea, desfasandolas de forma tal que nunca se pisen.

Las tareas que el planificador habilita para que se ejecuten en este proyecto son:

- Actualización del controlador, ejecutada cada 20 ticks (20ms) con un retardo inicial de 0 ticks.
- Actualización del canal inicial de DMX, cada 1000 ticks (1 segundo) con un retardo inicial de 1 tick.
- Comparación de los valores del encoder de motor y de disco, cada 100 ticks (100ms) con un retardo inicial de 2 ticks.
- Cambiar el estado del led externo si hay señal de DMX presente, cada 250 ticks (250ms) con un retardo inicial de 3 ticks.

Un ejemplo de planificador en psuedocódigo de C sería:

```

1 void planificador(void){
2     // funcion ejecutada en cada interrupcion del Tick
3     static int16_t tarea1_contador = tarea1_retardo;
4     static int16_t tarea2_contador = tarea2_retardo;
5
6     // Todas las cuentas se miden en multiplos de la base de tiempo
7     if(++tarea1_contador == tarea1_tiempo_ejecucion){
8         tarea1_contador = 0;
9         gTarea1_ejecutar = TRUE; // Variable global, visible por el despachador
10    }
11    if(++tarea2_contador == tarea2_tiempo_ejecucion){
12        tarea2_contador = 0;
13        gTarea2_ejecutar = TRUE; // Variable global, visible por el despachador
14    }
15 }
```

El despachador en este proyecto simplemente ejecuta las tareas que el planificador habilita, sin utilizar un método de priorización complejo debido a que, como se mencionó antes, cada tarea tiene asociado un retardo de tiempo para evitar que se solapen.

A continuación se detalla qué hacen las 4 despachar y el orden en que aparecen en la rutina despachador:

1. Actualización del controlador: se llama a la función *Controlador_update()*.
2. Actualización del canal inicial de DMX: se asocia el valor del dipswitch con el canal inicial de DMX haciendo uso de las funciones en C *DMX_write(Dipswitch_read()*

- $+ 1$). El $+1$ es debido a que la salida de Dipswitch_read va de 0 a 511, y los canales DMX van de 1 a 512.
3. Comparación de los valores del encoder de motor y de disco: se verifica que la relación hallada en la sección 3.3, subsección 1, a través de la tabla 3.2 (que 1 cuenta de encoder de disco equivale a aproximadamente 131 del de motor), se mantenga. Como no es una relación que se cumple estrictamente para todo el rango se permite una tolerancia de ± 500 cuentas de encoder de motor, haciendo que la comparación sea: $cuentas_encoder_motor = cuentas_encoder_disco * 131 \pm TOLERANCIA$.
 4. Cambiar el estado del led externo si hay señal de DMX presente: simplemente se alterna el estado del led externo, acción muy común para indicar que la señal llega correctamente en esclavos DMX.

Un ejemplo de despachador en psuedocódigo de C sería:

```

1 void despachador (void){
2     if (gTarea1_ejecutar) {
3         gTarea1_ejecutar = FALSE;
4         tarea1 ();
5     }
6     if (gTarea2_ejecutar) {
7         gTarea2_ejecutar = FALSE;
8         tarea2 ();
9     }
10 }
```

3.6.2.3 Bucle principal

En el bucle principal del programa se ejecuta la máquina de estados que se muestra en la figura 3.31.

A continuación se detalla el funcionamiento de la MEF:

El estado inicial, luego de configurar todos los módulos, es el Modo de calibración. En este modo se desactiva el freno (se permite que la carga pueda bajar) y se sube la carga muy lentamente hasta tocar el fin de carrera (FDC). Cuando esto ocurre se bajan 10cm e inicializa las cuentas de los encoders en 0, haciendo que este sea el 0 del sistema. En caso que la carga no toque el fin de carrera dentro de los 2 minutos, o que no baje los 10 cm dentro de 10 segundos, se considerará que hubo un error de calibración y se pasa al modo error. En caso contrario, el sistema pasa al Modo de funcionamiento normal. El freno queda desactivado de aquí en adelante para permitir el libre movimiento de la carga.

El estado en el cual el sistema se encontrará la mayoría del tiempo, siempre que todo funcione correctamente, es el Modo normal. En este modo se:

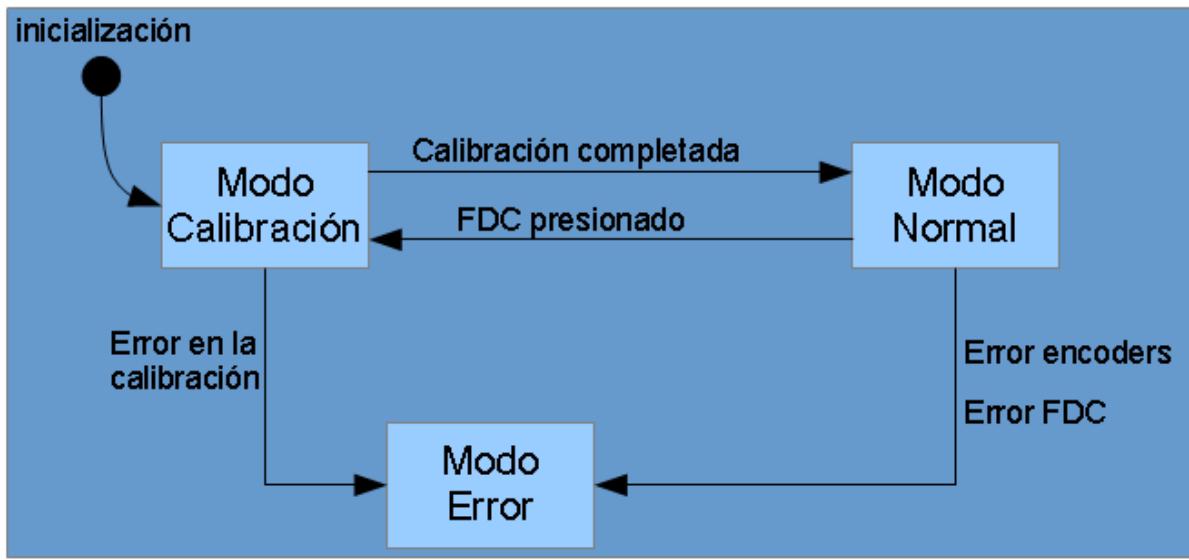


Figura 3.31: Máquina de estados ejecutada en el bucle principal de la aplicación

- Ejecuta el despachador de tareas. Como se mencionó anteriormente esto incluye la actualización del controlador, cambiar de estado el led externo si hay señal de DMX presente, verificar que el fin de carrera no haya sido presionado incorrectamente, y que la relación de cuentas se mantenga dentro de una tolerancia. En caso que se pierda la relación entre los encoders se pasa al Modo error.
- Comprueba continuamente el fin de carrera para verificar que haya habido una activación indebida del mismo. En caso de que esto suceda se parará el sistema por 6 segundos y se volverá al estado de calibración. Esto suprime las oscilaciones en la carga, que es la causa más probable de la activación indebida del fin de carrera. En caso de que el FDC se mantenga más de 1 minuto activado se pasa al Modo error.
- Verifica que el DMX esté activado. En caso de estarlo se lee la información de posición y velocidad del módulo DMX y se la pasa al módulo controlador haciendo uso de las funciones en C `Controlador_write(posicionDMX, velocidadDMX)`, siendo estos 2 parámetros `DMX_read(DMX_CANAL_POSICION)` y `DMX_read(DMX_CANAL_VELOCIDAD)`, respectivamente. Si no está activo se sube la carga lentamente hasta el 0 de posición y se mantiene allí hasta recuperar la señal.

Ante cualquier falla el equipo entra a un modo de error, en donde se acciona (se evita que la carga baje) el freno y se deshabilita el motor. No hay forma de volver de este modo, por lo que el equipo debe ser por lo menos reiniciado para que pueda seguir. Según el error causante de que este estado haya sido accedido se hace parpadear al led interno

a una determinada frecuencia: por error en la calibración el led interno parpadea a una frecuencia de 10 veces por segundo, si es por error de encoders 2 veces por segundo, y si es por fin de carrera 1 vez cada 2 segundos.

Validación

4.1 Procedimiento

En este capítulo se mostrarán los procedimientos realizados para validar los requerimientos presentados en la sección 1.5. Esto incluye:

- **REQ-01:** validar que firmware desarrollado para el equipo maneja correctamente todas las entradas y salidas.
- **REQ-02:** validar que la carga sigue la posición y velocidad indicada por DMX .
- **REQ-03:** validar que el comportamiento del sistema ante el descubrimiento de uno de los 3 errores detectables es la esperada.
- **REQ-04:** verificar que el funcionamiento del dipswitch es correcto.

4.2 Procedimiento

Los requerimientos REQ-01, REQ-02 y REQ-04 se validan simplemente controlando al updown mediante DMX, puesto que de esta manera se utilizan todas las entradas y salidas (que el firmware debe manejar), el sistema de control (que debe asegurar que la carga siga las referencias enviadas por DMX) y el dipswitch (para seleccionar la dirección inicial de DMX).

La prueba se realiza con 9 equipos updown dispuestos en una matriz de 3x3, como se muestra en la figura 4.1. Todos tienen en su extremo una carga de 3Kg, que es una luminaria cuyas intensidades R, G y B pueden ser controladas mediante 3 canales de DMX.

Para controlarlos se utiliza la consola HedgeHog 4, de HighEnd Systems.

Explicá: masomenos qué hiciste con la consola, q movimiento configuraste, y poné imágenes comparativas del movimiento en la consola y de los equipos

Por otro lado, para el requerimiento REQ-01

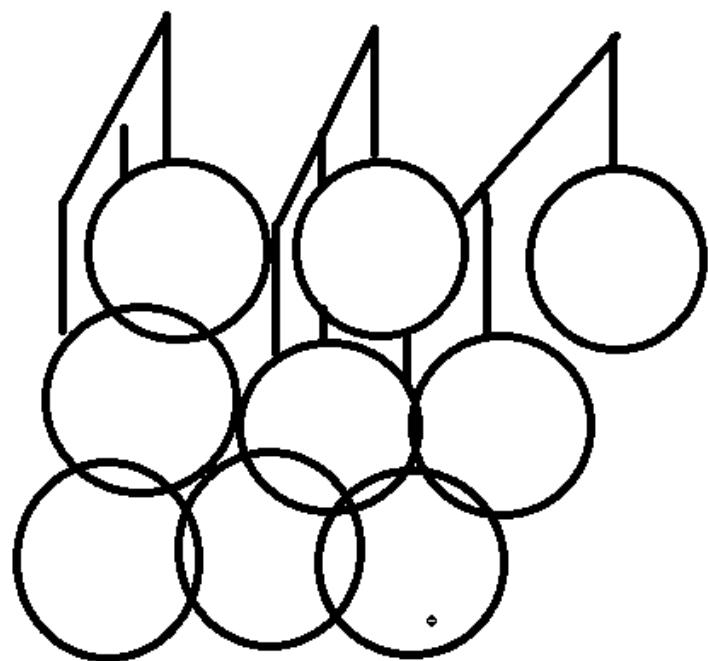


Figura 4.1: Diagrama del sistema motor

Conclusiones

5.1 Conclusiones

5.2 Mejoras a futuro

5.3 Referencias utilizadas