Sistema de control digital de una maqueta Ferroviaria (ID.24679)

Sergi Molina Grau

Resumen– En el mundo del modelismo ferroviario a escala, las unidades motrices están formadas por un motor eléctrico conectado a las ruedas del modelo. Con dos sistemas de control disponibles, analógico y digital, en este documento se plantea el desarrollo de un prototipo Hardware y Software de una central digital con la que poder controlar un conjunto variable de decodificadores. Se plantean metodologías y algoritmos para el control de los registros y la señal, así como dos métodos de control del sistema: una aplicación móvil, y el uso de un asistente de voz. A modo de continuación, también se plantean un conjunto de líneas de mejora con las que seguir en el futuro.

Palabras clave– Modelismo ferroviario, central digital, DCC, Digital Command Control, microcontrolador AVR, gestion de trenes, control por voz, señal digital.

Abstract– In the world of scale model railways, locomotives are based on an electrical motor connected to the model wheels. There are two control systems, analog and digital, this document proposes the development of a Hardware and Software digital station prototype that allows to control a set of decoders. There are proposed some metodologies and algorithms to control all the registers and digital signal. Also, two methodologies to control the system: a Smartphone application and a voice assistant. As a continuation line, it is proposed some ideas to improve the system capabilities in the future.

Keywords– Model railway, digital station, DCC, Digital Command Control, AVR microcontroller, train management, voice assistance, digital signal.

1 Introducción

En el mundo del modelismo ferroviario a escala, las unidades motrices están formadas por un motor eléctrico conectado a las ruedas del modelo. En los sistemas analógicos, todos los motores conectados a un mismo raíl, se moverán en el mismo sentido de la marcha, y a la misma velocidad en proporción al voltaje suministrado por el regulador de tensión.

A inicios de los años 40, con la aparición de los microcontroladores de bajo coste, empezaron a surgir proyectos para solventar la problemática de los sistemas analógicos; todos ellos son de carácter propietario y con bastantes limitaciones. Posteriormente, a principios de los años 80, Lenz [1] diseña un protocolo, el cual, entrados los años 90 fue adoptado por la NMRA (National Model Railroad Association) [2] como un estándar, el cuál es conocido como DCC (Digital Command Control).

Sin embargo, para que las locomotoras puedan funcionar con este nuevo sistema, es necesario que estas lleven instalado un decodificador compatible con DCC. Este se encargará de leer la señal, decodificar los mensajes y actuar en consecuencia.

La señal DCC es una señal cuadrada de amplitud fija y frecuencia modulada, transmitida por los raíles con polaridad opuesta, es decir, cuando en un raíl hay +Vcc en el otro hay 0V y viceversa). Los 1 y 0 lógicos se diferencian entre si por la longitud total del periodo de la señal [3](ver figura 1). Esta sucesión de 1s y 0s forma mensajes que encapsulan instrucciones que serán interpretados por los decodificadores.

El objetivo de este TFG se basa en el diseño e implementación de una central digital, siguiendo las recomendaciones del estándar, capaz de generar una señal DCC válida, junto a una aplicación móvil y una aplicación de reconocimiento de voz con Alexa, con la que poder gestionar el sistema.

En este documento se exponen las fases de desarrollo seguidas para llegar a un primer prototipo funcional de sistema, junto con las metodologías empleadas para llegar al

• E-mail de contacto: s.molinagrau@hotmail.com
• Mención realizada: Enginyeria de Computadors
• Trabajo tutorizado por: Màrius Montón Macián (Microelectrónica y Sistemas Electrónicos)
• Curs 2021/22

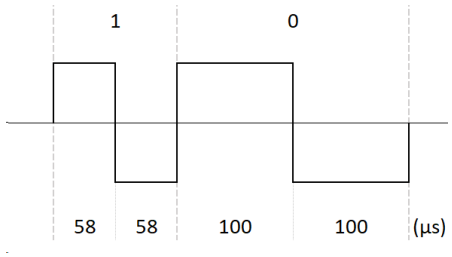

Fig. 1: Ejemplo bits 0 y 1 DCC

objetivo final, pasando por un análisis del mercado actual. Finalmente, también se analizará una simulación de los costes asociados al proyecto de cara a poder sacar un producto al mercado.

2 Estado del arte

En el desarrollo de este proyecto se han considerado diferentes tipos de fuentes, como lo son la documentación, así como los sistemas existentes en la actualidad. Para el primer caso, se ha analizado la documentación oficial, disponible en la web de la *NMRA (National Model Railroad Association)*, asociación sin ánimo de lucro que gestiona el estándar tanto del sistema digital como de otros campos del modelismo ferroviario a escala. En cuanto a la señal DCC, se pueden encontrar los documentos oficiales [4][5], los cuales tratan sobre las recomendaciones, especificaciones y el funcionamiento de esta. También existen varios foros enfocados a la temática de la señal digital. Entre los primeros resultados se pueden encontrar la web *DCC-Wiki* (en inglés), y *Wiki iGuadix* (versión en castellano).

Por el lado de los sistemas actuales, existen múltiples marcas que comercializan este tipo de central digital, cada una con unas funcionalidades y prestaciones distintas. Entre ellas, se puede encontrar el *Roco/Fleischmann[6][7] Multimaus[8]*, un mando de control con el generador de señal incorporado, que junto con un booster compatible, permite tener un sistema totalmente funcional. Se pueden encontrar sistemas más complejos, como pueden ser la Z21[9], comercializada también por *Roco/Fleischmann*, o las centrales comercializadas por *Digikeijs*[10]. Ambos dispositivos permiten funcionalidades más avanzadas que el dispositivo mencionado anteriormente, permitiendo por ejemplo, el uso de múltiples buses de comunicación, así como la gestión mediante un *Smartphone* o del ordenador, a través de una red LAN usando las aplicaciones específicas.

3 Objetivos de desarrollo

Para el desarrollo completo del proyecto se plantearon los cinco objetivos que se muestran a continuación.

1. **Definición de las bases del proyecto:** Decidir el alcance del proyecto, selección de los componentes necesarios para el desarrollo y preparación del entorno de trabajo.
2. **Generación de señal:** Desarrollar las bases del proyecto, implementando un algoritmo capaz de generar una señal DCC compatible.
3. **Comunicación y funcionamiento básico:** Generar paquetes de información en tiempo real; según el alcance del proyecto, movimiento de locomotoras. También se diseña un protocolo de comunicación simple entre periféricos.
4. **Aplicación móvil (Android)**: Diseño de una aplicación capaz de comunicarse mediante WiFi con el sistema, y que pueda gestionar múltiples decodificadores.
5. **Asistente de voz Alexa:** Diseño de una Skill de Alexa[11][12] que permita controlar el sistema mediante comandos de voz.

Entre todos estos objetivos, se han considerado los puntos 2 y 3 como los más críticos, debido a que sin ellos, el sistema no respondería a su función principal; Controlar una maqueta ferroviaria de forma digital.

En la fase de definición de las bases del proyecto, se decidió reducir su alcance para poder trabajar en varios campos del sistema, y aprovechar mejor así el tiempo disponible.

En la figura 2 se muestra un diagrama de bloques con los distintos elementos que formarían en conjunto el sistema completo (marcados en rojo), y los escogidos en el alcance de proyecto (en blanco).

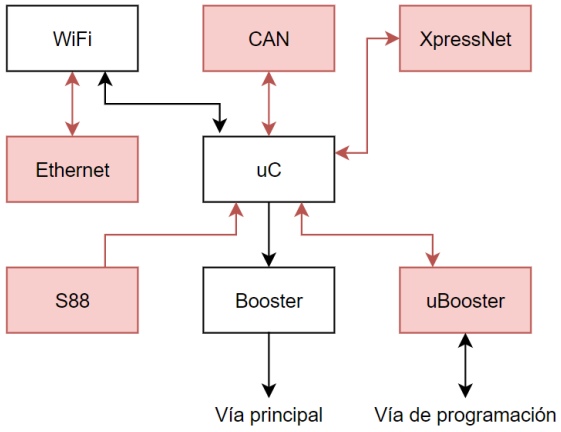

Fig. 2

Este diseño completo incluiría la posibilidad de comunicarse con otros periféricos a través de otros buses de comunicación existentes (XpressNet y CBUS). Además, se tendría en cuenta la inclusión de más funcionalidades. Para ello, se considerarán estas necesidades de cara a que el resultado de este trabajo permita seguir expandiendo a estos nuevos objetivos.

4 Metodología

Para el correcto desarrollo del proyecto, este se plantea sobre la metodología ágil Scrum [13] adaptada y planteada a las necesidades del TFG [14]. A pesar de que Scrum es originalmente una metodología colaborativa entre equipos, es interesante por el proceso que conlleva.

Cada dos semanas se ha realizado una iteración (*Sprint*), en las cuales se ha pasado por cada una de las distintas

fases de diseño, implementación y revisión del trabajo. Al finalizar cada sprint, se ha analizado la evolución del trabajo (*retrospective*). Finalizada cada iteración, se ha comprobado como se ha desarrollado esta, y definido el trabajo del nuevo sprint (*Sprint Backlog*).

Sobre cada uno de los objetivos comentados en el anterior apartado, se definieron los requisitos de desarrollo (*Product Backlog*), del que en cada iteración se estrajeron elementos para formar los Sprint Backlog.

Para la gestión y control de las tareas se ha utilizado el entorno Jira [15], una herramienta de Atlassian [16]. En esta se ha definido un tablero con el siguiente conjunto de columnas:

- **To do:** Lugar por defecto en el que aparecen las tareas que están disponibles para desarrollar.
- **In progress:** Aquí se añaden las tareas que están en ejecución. Se añadió una limitación de 2 a esta columna para evitar tener demasaidas tareas abiertas a la vez.
- **In review:** Una vez se ha implementado las funcionalidades, entran en fase de revisión. También se añadió una limitación de 5 para que los bloques de revisión no fuesen demasiado extensos.
- **Done:** Tareas finalizadas que han pasado la fase de revisión.

En el apéndice A.2, figura 10, se puede observar el flujo de trabajo establecido para las tareas. Con ello, se consigue que las tareas deban pasar por todos los estados en el orden correcto, es decir, una tarea en estado *To do* no puede pasar directamente al estado *In review*, pero si al revés.

Se ha hecho coincidir el final de cada iteración, con una reunión de seguimiento con el tutor, en las que se ha comentado la situación del proyecto en cada fase, y los planes de las siguientes semanas.

En relación al desarrollo, se ha trabajado sobre una metodología TDD (Test-Driven Development) [17][18]. Esta práctica consiste en escribir primero el test y posteriormente el código fuente, permitiendo así asegurar que este es correcto desde las primeras fases de su implementación.

Por último, para llevar un correcto control de versiones del código, todo el proyecto se ha subido a un repositorio en GitHub.

5 IMPLEMENTACIÓN

Esta sección está formada por diferentes bloques, los cuales recogen toda la documentacion relacionada con las diferentes fases de desarrollo del proyecto.

5.1. Elección de componentes

En la elección de componentes se han tenido en cuenta varios factores, por un lado la compatibilidad con las funcionalidades del sistema, y por otro lado, la disponibilidad del material.

Se ha decidido comparar microcontroladores *AVR*[19] debido a la experiencia con estos, y porqué al ser los chips que incorporan los Arduino, existe mucha documentación. Por otro lado, el método de programación: estos microcontroladores llevan un bootloader [20], un software muy liviano que permite programar el microcontrolador mediante el puerto serie. Además, se puede reprogramar el bootloader usando otro Arduino, evitando así tener que adquirir un programador específico [21][22].

En la tabla 1 se comparan algunas de las especificaciones de los microcontroladores que llevan el *Arduino UNO* y el *Arduino Pro Micro* respectivamente.

	Atmega328p[23]	Atmega32U4[24]
Tensión	2.7V - 5.5V	2.7V - 5.5V
Flash	32KB	32KB
SRAM	2KB	2.5KB
Freq.	16MHz (max. a 5V)	16MHz (max. a 5V)
GPIO	23	26
ADC	8x 10-bit	12x 10-bit
Timer	2x 8-bit 1x 16-bit	1x 10-bit 2x 16-bit
PWM	6 canales	14 canales
USART	Sí	Sí
SPI	Sí	Sí
I2C	Sí	Sí
CAN	No	No
USB	No	Sí

TABLA 1: COMPARACIÓN ENTRE MICROCONTROLADORES

Tal como se puede observar, ambos microcontroladores son muy parecidos. Finalmente, debido a que se quiere mantener la comunicación USB (para debug y control por PC) y el puerto serie (para la comunicación con el módulo WiFi) de forma independiente, se ha decidido escoger el microcontrolador *ATmega32U4*.

En referencia a la parte de potencia, se ha escogido el L298[25], un driver doble de motor, que puede ser alimentado hasta 46V y una intensidad de salida máxima de 2A por canal en DC. Para facilitar el montaje, se está usando una placa Plug&Play que tiene todos los componentes necesarios para que esta funcione[26]. La conexión entre Arduino y el driver de motor L298 consta de tres cables: el primero corresponde a la señal Enable, lo que permitirá desconectar la salida del driver en caso necesario; los otros dos cables, corresponden a la señal DCC. Sin olvidar un cable al transformador de 15V (tensión de alimentación típica en la escala de modelismo utilizada), otro al pin de 5V del Arduino (para tomar como referencia, para la entrada de señal del microcontrolador) y otro para la masa común.

De la placa salen otros dos cables, que corresponden a la salida amplificada de señal.

Para la comunicación inalámbrica se ha escogido el microcontrolador *ESP32* [27], el cual incorpora un módulo de comunicación WiFi y Bluetooth en el mismo *SoC*. Este servirá de puente entre el microcontrolador *ATmega32U4* y la comunicación inalámbrica.

La comunicación entre ambos microcontroladores se realiza mediante sus respectivos puertos serie. Hay que tener en cuenta que ambos funcionan a una tensión de alimentación distinta, por este motivo, y debido a que los pines del *ESP32* no son tolerantes a 5V, el cable de transmisión del ATmega32U4 (5V) al de recepción ESP32 (3.3V) pasa por un divisor de tensión resistivo como el de la figura 3.

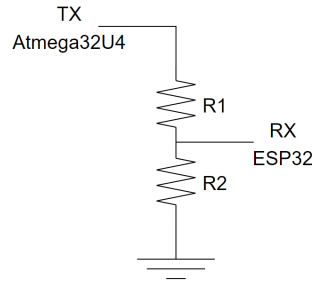

Fig. 3: Divisor de tensión entre TX y RX de ambos microcontroladores

La fórmula para calcular el valor de las resistencias es:

$$RX_{ESP32} = \frac{R2}{R1+R2} * TX_{ATmega32U4}$$

Partiendo de un valor de R2 = $4K7\Omega$, el valor de R1 debe ser:

$$R1 = \frac{TX_{ATmega32U4} * 4700\Omega}{RX_{ESP32}} - 4700\Omega = \frac{5V * 4700\Omega}{3{,}3V} - 4700\Omega = 2421\Omega = 2K4\Omega,$$

Analizados y escogidos todos los componentes, el prototipo en protoboard queda tal como se ve en el diagrama de la figura 4. En el anexo A.2, figura 11, se podrá observar el montaje final de este primer prototipo.

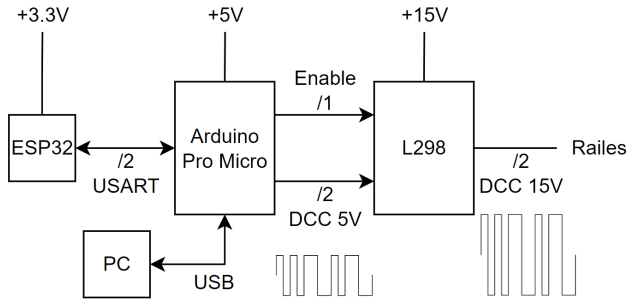

Fig. 4: Diagrama del prototipo del sistema

5.2. Entorno de trabajo

Para trabajar de forma cómoda y más eficiente se ha usado el IDE de Microsoft *Visual Studio Code*[28], usando los plugins para poder desarrollar sobre el entorno de *Arduino*, así como las herramientas necesarias para trabajar en *C/C++*, *Intelli-Code* para las ayudas del lenguaje y de conexión entre clases, y la herramienta *Git Graph*, con la que se puede ver el estado de los commits y las posibles ramificaciones forma clara.

La decisión de usar este IDE con los plugins de Arduino, en lugar del propio IDE de Arduino, se deben a que el editor oficial ofrece muy pocas funcionalidades de ayuda a la programación, que resultan útiles para programar de forma más eficiente, y resulta en un editor más simple.

5.3. Desarrollo: El sistema

Esta sección se ha dividido en cuatro partes, la primera enfocada a la generación de la señal, la segunda a la gestión de los registros, la tercera para la gestión y uso de la señal, y por último, la cuarta a un sencillo protocolo de comunicación, empleado para que el microcontrolador se comunique con el exterior.

5.3.1. Generación de señal

Antes de empezar, una breve explicación del formato de los mensajes compatibles con la señal DCC. Todos los paquetes están formados por un preámbulo, y un conjunto de bytes separados por bits con valor 0, y por último, un bit de finalización de paquete, con valor 1. El preámbulo indica a los decodificadores que se está a punto de enviar un paquete, de esta forma se pueden sincronizar con la señal. De forma general, se puede decir que la señal se construye siguiendo el siguiente patrón:

{preámbulo} 0 {address} [0 {bytes de instrucciones}] 1.

El estándar define que pueden haber entre 2 y 5 bytes de datos de instrucción. A continuación, en la imagen 5 se muestra un ejemplo de paquete DCC:

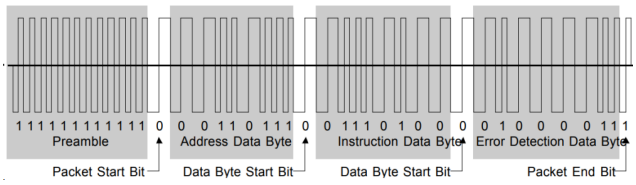

Fig. 5: Ejemplo de señal [29, p. 2]

El estándar indica que la duración del bit '1' debe ser de alrededor de 58µs; para el bit '0', el tiempo debe ser de al menos 100µs.

Para generar la señal, en la clase *NmraDCC* se podrá encontrar el método *signal*. Este será llamado a cada interrupción de un timer configurado para saltar cada 58µs. El algoritmo se encarga de preparar los bits que se deben enviar, hacer los cambios pertinentes en los pines de salida del microcontrolador, y gestionar cada cuantas interrupciones debe hacer el cambio de bit.

El tiempo de ejecución del código ejecutado en una interrupción debería ser lo más reducido posible. Esto es así, debido a que el microcontrolador abandona inmediatamente el flujo normal del programa, para atender la interrupción y ejecutar entonces el código asociado a esta.

Tras una ejecución en bucle de 1.000 iteraciones, se ha podido comprobar que el tiempo medio de ejecución del método es de unos 7µs. Debido a que las interrupciones ocurren cada 58µs, esto significa que el tiempo empleado en la ejecución de la interrupción con respecto al tiempo entre interrupciones es de un 12 %, por lo que no debería interferir en exceso con el flujo normal del programa.

La señal generada se puede observar en la captura de pantalla del osciloscopio, en el apéndice A.2, figura 14.

5.3.2. Registro de locomotoras

Para poder controlar múltiples locomotoras de forma simultanea, se ha implementado un registro en el que se guarda la información necesaria, asociada a cada decodificador disponible. Este registro devolverá los valores de cada locomotora, de forma cíclica, de modo que los decodificadores puedan estar siempre actualizados a la última información almacenada.

Debido a la reducida capacidad de memoria del microcontrolador, las variables se han ajustado al máximo a las necesidades y especificaciones:

- **Address:** El estándar permite direcciones en el rango [1, 10239]. Para ello son necesarios $\log_2 10239 = 13{,}32 = 14$ bits, por lo cual, el tipo de dato escogido es *uint16_t*.
- **Velocidad y Dirección:** El estándar admite 14, 28 y 128 pasos de velocidad (de 0 % a 100 %). Considerando el caso de mayor precisión, son necesarios $\log_2 128 = 7$ bits, por lo cual el tipo de dato escogido es *uint8_t*. Como para representar la dirección se necesita 1 bit, se añade al byte anterior, quedando el siguiente formato *DSSSSSSS* (D: dirección, S: Velocidad).
- **Funciones:** El estándar permite el control de hasta 28 funciones[30, p. 7]. A cada función le corresponde 1 bit, por lo cual el tipo de dato escogido es *uint32_t*.
- **Configuración:** El tamaño de esta variable dependerá de la información de configuración que se quiera almacenar. Por el momento solo se guarda la configuración de los pasos de velocidad, por lo cual el tipo de dato escogido es *uint8_t*, restando otros 7 bits para futuras versiones.

En total, cada registro ocupa 8 bytes.

Si se compila el programa, con solo un registro de locomotora, la salida del IDE indica que el programa completo necesita un 28 % de la memoria SRAM (717/2560 bytes disponibles). Si se plantea usar la memoria restante para el vector (1843 bytes), se pueden tener hasta 230 registros.

Considerando un **supuesto** paquete estándar con 4 bytes de datos, donde cada byte de datos está formado por 1s y 0s intercalados, se puede calcular el tiempo necesario para enviar la trama completa. Un paquete está formado por un preámbulo (12x1s), cada byte de datos contendrá 4x1s y 4x0s, por tanto, 16x1s y 16x0s, 4 bits de separación (4x0s) y un bit de finalización (1x1). En total son necesarios 29x1s y 20x0s, que aplicando los tiempos de transmisión de cada tipo de bit, se obtiene: $t_{total} = 29 * (2 * 58\mu s) + 20 * (2 * 100\mu s) = 3{,}36ms + 4ms = 7{,}36ms$.

Suponiendo que un paquete de velocidad y dirección tarde 7.36ms en ser transmitido, y considerando la disponibilidad de 230 registros, se necesitaría 1.69s para enviar la información de todos ellos.

El estándar indica que se debe volver a transmitir la información de velocidad y dirección tan rápido como sea posible, para evitar la pérdida de información ocasionada por fallos de alimentación cuando los raíles estén sucios. Si los decodificadores reciben su información cada 1.69s, a velocidades lentas podría ocasionar parones demasiado perceptibles, pudiendo restar en la experiencia de usuario. Por otro lado, no es habitual tener ese volumen de decodificadores en movimiento.

Es por estos motivos, que el registro no debería ser mucho mayor de 32 elementos. Con este nuevo valor, el tiempo para realizar un ciclo es de $32 * 7{,}36ms = 235{,}52ms$, actualizando la información 4 veces por segundo.

Para complementar el registro, se ha añadido un buffer de entrada (*input_buffer* en la figura 6). Por cada mensaje recibido por los distintos canales disponibles, se añade una nueva entrada al buffer y si es necesario se actualiza el registro. Con esto se consigue que los nuevos mensajes sean procesados lo antes posible.

5.3.3. Gestión de salida

Tal como se ha comentado anteriormente, una de las grandes virtudes del sistema digital es el de poder controlar múltiples decodificadores a la vez. Además, no solo permite gestionar velocidad y dirección de las locomotoras, sino que ofrece la posibilidad de enviar una gran variedad de mensajes distintos a través de los raíles, como puede ser: activar o desactivar salidas de funciones (luces, sonidos...), control de accesorios (iluminación estática, semáforos, controladores de desvío...), comandos de fecha y hora, y configuración del comportamiento de los decodificadores mediante la programación de CV (Configuration-Variables[31]).

Para solventar este problema, se ha diseñado el procedimiento *NmraDCC::update()*, que se encarga de gestionar qué paquete se debe enviar a cada momento.

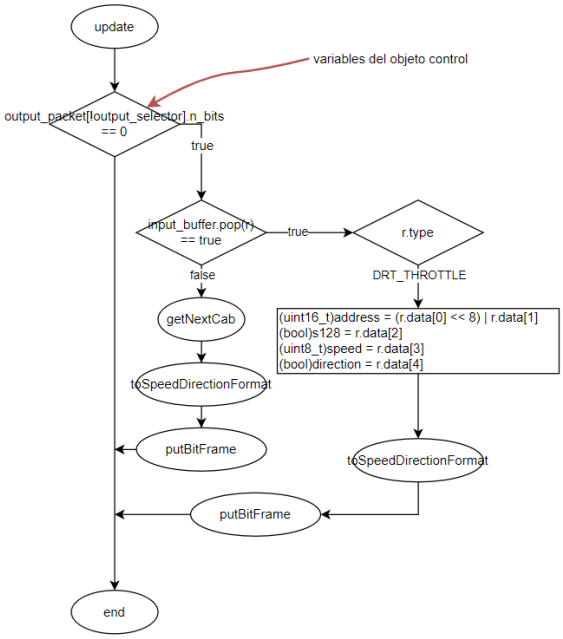

Fig. 6: Diagrama de flujo del método *NmraDCC::update*

El método *NmraDCC::signal()* explicado en el punto 5.3.1 trabaja sobre dos registros de salida; estos registros contienen la información necesaria para controlar los bits que se están transmitiendo. Uno de los dos registros se considera como el ”registro activo”, es decir, el que se está

transmitiendo actualmente. El otro se utiliza para precargar la información del siguiente paquete. Así cuando se transmite el último bit del registro, el "registro actual" pasa a ser el que estaba disponible. De esta forma, y tal como se observa en la figura 6, cuando el registro libre está disponible (no se ha precargado otro mensaje), el método *NmraDCC::update()* inserta el nuevo paquete en ese espacio.

Tal como se observa en la figura 6, cuando uno de los dos registros de salida está disponible, se define qué mensaje debe ocupar ese espacio.

En orden de preferencia, primero se comprueba si existe algún mensaje en el buffer de entrada; esto es debido a que los nuevos mensajes deben ser transmitidos lo antes posible; de este modo la acción del usuario se ve ejecutada al instante.

Si existe un mensaje en el buffer de entrada se comprueba el tipo, y se prepara el nuevo paquete DCC. Para el alcance del proyecto actual, solo se ha implementado el tipo *DRT_THROTTLE*, el cual contiene información de velocidad y dirección. En el apéndice A.2, figura 15, se puede observar una amplicación del método *NmraDCC::update*, en el que se podría trabajar con paquetes de funciones, accesorios y realizar paradas de emergencia, entre otros.

Por otro lado, si no existe ningún mensaje en el buffer de entrada, solicita al registro de locomotoras un nuevo registro (*getNextCab*); de este objeto se extrae la información de velocidad y dirección, se genera un nuevo paquete, y añade al registro de salida.

Los métodos *toXFormat* convierten los datos de entrada a un vector cuyos elementos corresponden al formato específico de cada tipo de mensaje (por ejemplo, los paquetes de velocidad y dirección tienen el formato [0AAAAAAA, 01DSSSSS, EEEEEEEE], donde A es Address, D dirección de movimiento, SSSSS velocidad y E el byte de detección de errores) . El método *putBitFrame* toma el vector generado en las funciones *toXFormat*, y configura el registro de salida con la trama de bits del paquete DCC final, y la información asociada a este.

Al separar los dos tipos de método, se ofrece mucha flexibilidad a la hora de poder añadir nuevos formatos en el futuro.

5.3.4. Protocolo de comunicación

Se ha diseñado un protocolo simple de comunicación entre los dispositivos del sistema, así como entre el microcontrolador *Atmega32U4* y *ESP32*.

Para su implementación se ha decidido trabajar con los caracteres alfanuméricos [0, 9] y [a, f] para representar valores hexadecimales, en grupos de dos para representar bytes completos, y los caracteres ':' y ';' para el inicio y finalización de un mensaje respectivamente.

El principal motivo de usar este formato se debe a la facilidad de interpretación por parte de un humano, y por otro lado para poder enviar comandos a través de una terminal cualquiera mediante teclado.

El formato que toma un mensaje es el siguiente:

:<OpCode>[Data];

donde *OpCode* es un byte con el que se indica el contenido del mensaje, y es siempre obligatorio. En cambio, los bytes correspondientes a la parte de *data* son opcionales, y dependen del mensaje. En el apéndice A.1 se podrá observar todos los mensajes compatibles actualmente con el sistema.

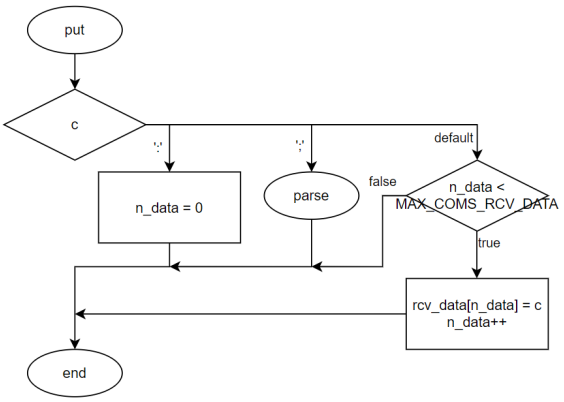

Fig. 7: Diagrama de flujo procedimiento *Coms::put*

Para la recepción y tratamiento de los mensajes se ha implementado el algoritmo de la figura 7. Por cada carácter recibido por alguno de los canales de comunicación disponibles, se llama al método *put*, pasando el carácter en la variable *'c'*. La clase tiene un vector *rcv_data* y un contador *n_data*. Cuando el carácter recibido es el símbolo *':'*, se reinicia el contador; con ello se pierden los datos anteriormente contenidos en el vector. Por otro lado, cuando el carácter recibido es el símbolo *';'*, se procesa y decodifica el mensaje llamando al método *Coms::parse*; en este se hace la llamada a las funciones necesarias para utilizar los datos. Por último, cada carácter recibido distinto a los símbolos de inicio y final de mensaje, se añade al vector. Se puede ver más detalles de la clase en el anexo A.2, en la figura 12.

5.4. Aplicación Android

Para poder controlar el sistema, se ha desarrollado una aplicación sencilla para Android. Esta permite gestionar un conjunto de decodificadores de una forma muy gráfica.

Esta aplicación se ha desarrollado mediante AppInventor[32], un entorno de desarrollo web que permite crear aplicaciones mediante bloques de programación tipo Scratch, formando un puzle[33].

Tal como se puede ver en la figura 8, la interfaz permite controlar la velocidad y dirección del decodificador, mediante la barra verde y gris, y los botones inferiores; también permite controlar las salidas de funciones del decodificador. Se puede seleccionar qué registro se está controlando, pulsando el nombre de la locomotora ("BR442" en el caso de la figura). Cada vez que se realiza una acción, la aplicación envía dicha información, junto con la dirección asignada al decodificador instalado en la locomotora.

5.5. Control mediante asistente de voz

Además, de poder controlar el sistema mediante una aplicación móvil, se ha desarrollado una Skill para el asistente de voz Alexa.

El principal motivo para desarrollar en la plataforma de Amazon ha sido la posibilidad de hacer toda la implemen-

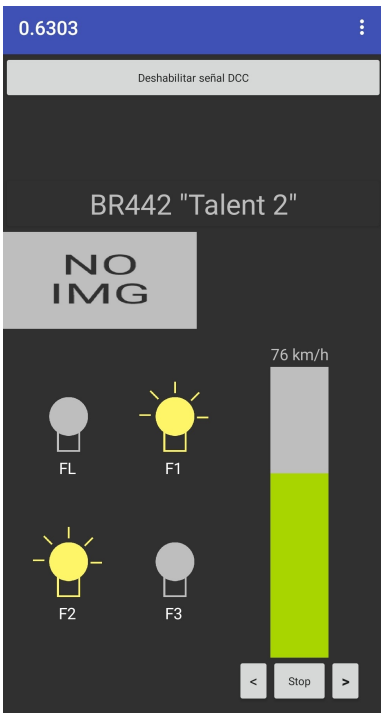

Fig. 8: Interfaz principal de control

tación de forma oficial en su propia plataforma, así como la extensa documentación y ejemplos que ofrecen en la página web y canales audiovisuales.

Para la implementación de la propia Skill, Amazon ofrece dos lenguajes de programación: *Python* y *Javascript*, de los cuales se ha escogido el primero. Esto se debe a que quería ganar más experiencia con el lenguaje.

La idea original era poder conversar con Alexa de forma fluida, por este motivo, y siguiendo los consejos de la documentación, se ha escrito un pequeño guión donde se muestra cada una de las posibles peticiones que podría demandar un usuario, así como posibles variaciones de estas. También se definen las posibles respuestas que puede devolver el asistente de voz. A continuación, se muestra un breve ejemplo de este guión:

USUARIO
≪Alexa,≫ abre *control de trenes* y ajusta la velocidad de la *cabName*.

ALEXA
Por favor, dime a qué velocidad y dirección quieres que la ajuste.

USUARIO
Al *cabSpeed* porciento hacia *cabDirection*.

ALEXA
Eso está hecho, los datos han sido enviados a la central.

La primera y tercera frase, son lo que se conoce como ’declaración’ (*utterance*). En este caso, contienen los slots *cabName*, *cabSpeed*, y *cabDirection*, que se usan para obtener la información concreta que ha dado el usuario. Las frases tercera y cuarta son respuestas por parte de Alexa.

Estos *slots* tienen un formato fijo, y con ello un conjunto de posibles entradas. Por ejemplo, *cabName* es una lista de nombres de locomotoras y composiciones concretas. Por este motivo, la conversión del nombre de la locomotora a la dirección física se ha tenido que hacer directamente sobre el código.

Hasta ahora, se han desarrollado comandos para gestionar el movimiento de las locomotoras (como en el caso del ejemplo), la detención general de todos los decodificadores, así como la posibilidad de activar un modo noche/día más enfocado al sistema de iluminación.

5.6. Comunicación WiFi

En este último punto, se documentan las implementaciones que han sido necesarias para hacer que el microcontrolador *ATmega32U4* acabe actuando según los comandos de la aplicación y de Alexa.

Para la comunicación WiFi se ha diseñado un Sketch para el módulo *ESP32*. Este ha necesitado usar las librerías *WiFi.h* de Arduino[34], *AsyncUDP*[35], para poder generar un servidor UDP en el microcontrolador, y la librería *HTTPClient*[36], para generar un cliente HTTP con el que comunicarse con un servidor Web.

El módulo ESP32 permite conectarse (u ofrecer conexión) de tres formas distintas: *STAtion*, en este caso se conecta a un router WiFi, *SoftAP*, es decir, actuar como un *AccessPoint* o punto de acceso, e incluso en modo *SoftAP-STA* caso en el que mantendría ambos métodos de conectividad. Para las funcionalidades que se quieren implementar, se ha decidido configurar el módulo en modo *Station*, de esta forma se puede conectar al router WiFi y obtiene así acceso a Internet.

5.6.1. Comunicación con la aplicación Android

Para la comunicación con la aplicación de Android, se ha implementado un servidor web **en el módulo WiFi** mediante la librería *AsyncUDP*. Al levantar el servidor en el puerto *8080*, este se mantiene a la escucha de los mensajes que entren por dicho puerto.

Cuando se recibe un paquete, el contenido de este se pasa al método *Coms::put*. Como el objeto se ha creado indicando que actúe como *’gateway’*, todos los mensajes recibidos se reenvían por el puerto serie. De esta forma, el módulo WiFi está actuando únicamente como puente entre la comunicación *WiFi* y el microcontrolador *Atmega32U4*.

5.6.2. Comunicación con Alexa

Debido a que la Skill de Alexa almacena la información en una base de datos intermedia, para tomar estos valores, se crea un cliente HTTP con el que lanzar solicitudes a dicho servidor. El programa envía una solicitud cada segundo.

En este caso, y para reducir carga a la función Lambda, se almacena directamente los datos en formato *JSON*. Para ello, cuando el cliente HTPP recibe datos nuevos, debe decodificar la información, para acabar transformándola al formato compatible con el protocolo descrito en la sección 5.3.4. Una vez transformados los datos, se envían al microcontrolador mediante el mismo proceso que se ha explicado en la sección 5.6.1.

5.7. Servicio MQTT

Este servicio se ha implementado para sustituir el protocolo simple de comunicación de cara a la transmisión de información mediante la red WiFi, entre el dispositivo móvil, y el sistema, así como evitar el uso de una base de datos intermedia para comunicar Alexa con el sistema.

MQTT son las siglas de *Message Queuing Telemetry Transport*; este es un protocolo de mensajería muy extendido en el mundo del IoT, usando el paradigma de publicación/suscripción. Este protocolo está formado por un conjunto de normas que definen cómo se debe publicar y suscribir a los datos a través de la red [37].

Para el correcto funcionamiento del servicio, es necesario tener un Broker. Este es un software que actúa a modo de centro de comunicación. Los clientes, se suscriben a este elemento del sistema para así poder recibir la información publicada por otros clientes. En este caso, se ha instalado el software libre Mosquitto[38] en una Raspberry Pi, la cual actuará como broker del sistema.

Se ha adaptado el Sketch del módulo WiFi añadiendo una librería que permite la funcionalidad de publicación/suscripción. En este caso, este se suscribe a los topics cab/# y station/#. Esto es así debido a que la central debe poder recibir toda la información asociada a cualquier dirección de locomotora; también debe poder recibir cualquier cambio que afecte al funcionamiento del sistema. En este punto del desarrollo no se ha creído conveniente que la central realice publicaciones, debido a que no hay implementado ningún evento que así lo requiera.

En cuanto a la aplicación, las modificaciones han sido las mismas: se ha añadido las librerías necesarias para poder realizar las comunicaciones, y los eventos asociados a los botones y slider, que ahora se transmiten a través de las nuevas librerías.

En este caso, la aplicación móvil solo se suscribe a la dirección de locomotora activa (para todos sus mensajes) y a los topics de cambio de estado del sistema. Sin embargo, a diferencia del sketch del módulo WiFi, la aplicación Android sí publica los distintos mensajes usando el formato adecuado.

En el caso de la skill de Alexa, no se ha conseguido implementar una librería Python que permita la comunicación directa mediante MQTT. Es por este motivo, que en la Raspberry Pi se ha desarrollado un pequeño servidor que recibe los siguientes comandos de la skill:

- "topic": "ruta del topic".
- "message": "valor que se quiere publicar"

Este pequeño software hace la traducción de esta información directamente en una publicación MQTT al broker.

Para que la skill de Alexa pudiese comunicarse con el broker, se ha abierto un puerto en el router, de modo que las peticiones de la skill fuesen atendidas por el servidor implementado en la Raspberry Pi.

A continuación, se muestran los topics a los que se publica y se suscribe cada uno de los elementos del sistema.

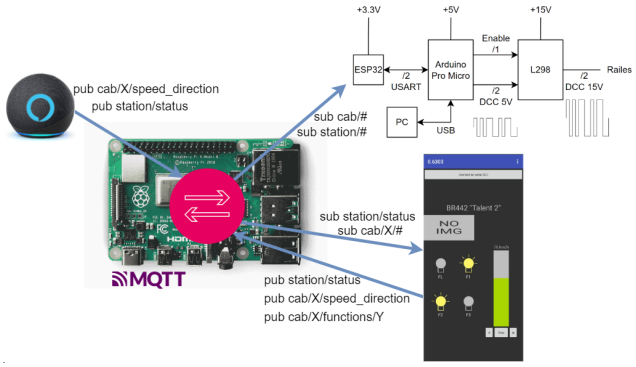

Fig. 9: Diagrama comunicación MQTT del sistema

6 'EVALUACIÓN DE COSTES'

En esta sección se va a evaluar el coste de desarrollar un dispositivo comercial, considerando el hardware empleado hasta la fecha, con el coste actual de los componentes y considerando su disponibilidad inmediata. Tal como se comenta en la sección 7, la intención es expandir este proyecto y mejorar sus capacidades, por lo que llegado el momento se deberá recalcular este presupuesto teniendo en cuenta los nuevos compoentes.

Tras comparar los precios y disponibilidad de los componentes en varios distribuidores, se ha decidido tomar los valores de *Farnell*[39]. En la tabla 2 se puede observar los precios de los componentes seleccionados. Hay que tener en cuenta que algunos componentes tienen una reducción de precio menores a x1000 unidades, por lo que probablmenente si se contacta con el distribuidor, se pueda ajustar todavía más algunos de ellos.

Componente	*Referencia*	*Precio (x1000)*
Atmega23U4	2857665	4760€
L298	403295	6020€
ESP32	3932669	4480€
L7805	1652336	430€
LM317	2464246RL	616€
Transformador 15V	3790301	12370€
Otros componentes	-	7000€
	Total:	35676€

TABLA 2: COSTE DE COMPONENTES

La fila *otros componentes* hace referencia a componentes varios como condensadores, resistencias, u otros circuitos integrados. Estos se han escogido observando los diagramas de Arduino y de la placa del driver *L298*.

Por otro lado, hay que tener en cuenta el coste de fabricación de las PCB, así como el proceso de ensamblado de los componentes. La empresa seleccionada es *JLCPCB*[40][41]. Para la primera parte, considerando un tamaño de PCB de 100x70cm, dos capas de cobre, y el resto de valores por defecto, el coste de fabricación asciende a

335,16€ + 121,26€ de envío. Para la parte de ensamblado, se ha considerado el montaje de 30 componentes por placa. Según la página de ayuda, el coste de ensamblado de los componentes empieza por 8€ de preparación y configuración, 6,65€ por la fabricación de la plantilla de soldadura, y después 0,0017€ por componente [42].

Finalmente, el coste total de producción de las PCB con ensamblado de componentes, tiene un coste de $335,16€ + 65,65€ + 121,26€ = 522,07€$.

Por último, en la tabla 3 se muestra el coste de los distintos recursos:

Recurso	*€/h*	*horas total*	**€ total**
Ing. Informático	15€	300h	4500€
Ingeniero HW	20€	30h	600€
Coste PC	0,1372€	330h	45,28€
		Total:	5145,28€

TABLA 3: COSTE DE RECURSOS

Se ha considerado el uso de ordenadores con un consumo constante de alrededor de 350W, y un coste medio de 0.3919€/kWh (marzo 2022 [43]).

Con todo ello, el coste total de producción de los 1000 dispositivos asciende a $35676€ + 522,07€ + 5145,28€ = 41343,35€$, es decir, 41,34€ por unidad. Para poder cubrir costes y obtener un margen de beneficio, el coste base de venta sería de 55€. Con ello, se consigue un beneficio de 13,66€ por unidad, es decir, de 13660€ vendiendo todas las unidades.

Para este ejemplo se ha obviado el coste de marketing, del embalaje del producto.

7 CONCLUSIONES Y LÍNEAS DE CONTINUACIÓN

Para concluir, tal como se ha demostrado a lo largo del documento, se ha conseguido cumplir todos los objetivos propuestos en la planificación original, consiguiendo así un sistema funcional capaz de controlar un conjunto variable de locomotoras de forma simultánea; lo cual se puede considerar lo mínimo indispensable a controlar por un sistema de estas características. También remarcar la importancia en el cumplimiento de los estándares desarrollados por empresas o asociaciones reguladoras, con las que se puede conseguir un correcto entendimiento, como es en este caso, entre dispositivos de distintos fabricantes.

En último término, indicar que este proyecto se propuso con la intención de continuar trabajando en el futuro, añadiendo nuevas funcionalidades y conectividades. Durante el desarrollo se ha tenido en consideración la posibilidad de añadir nuevos tipos de mensajes compatibles con DCC. Un ejemplo de ello se puede observar con la gestión de la señal de salida (sección 5.3.3), donde se ha tenido muy en cuenta la posibilidad de incluir nuevos paquetes en el futuro. Esta posibilidad se demuestra en la captura que se encuentra en el A.2, figura 15. Por otro lado, una rama que se considera importante de cara a nuevas versiones del sistema, sería la inclusión de algunos buses de comunicación, de modo que el sistema se pueda comunicar con otras placas de accesorios. Para finalizar, se definiría mejor el concepto de gestión de comandos de voz mediante Alexa, haciendo que estos se enfoquen a realizar una serie de acciones más específicas, como lo son la parada general, o activar algún tipo de funcionalidad concreta al sistema.

AGRADECIMIENTOS

Quiero agradecer a mi tutor de TFG Màrius Montón por los consejos que me ha dado desde el primer día; espero haber conseguido cumplir todos ellos. También por la tranquilidad transmitida así como la libertad para explorar campos.

A mi pareja, Laura, por acompañarme en esta travesía, a su indudable ayuda para buscar frases inconexas en el documento, y por recordarme siempre que el vaso está medio lleno. A mis padres y familia por el apoyo y la fuerza que han tenido al aguantarme en los momentos de mayor estrés.

Y a mis compañeros de carrera, con los que he compartido muy buenos momentos, una pandemia mundial, y muchas historias.

REFERENCIAS

[1] Startseite - Lenz Elektronik GmbH. (s. f.). Startseite - Lenz Elektronik GmbH. https://www.lenz-elektronik.de/index.php

[2] About the NMRA. (s. f.). National Model Railroad Association. https://www.nmra.org/about

[3] (s. f.). National Model Railroad Association. https://www.nmra.org/sites/default/files/standards/sandrp/pdf/s-.1_electrical_standards_for_digital_command_control_2021.pdf

[4] RP9 NMRA standards and recommended practices. (s. f.). National Model Railroad Association. https://www.nmra.org/index-nmra-standards-and-recommended-practices

[5] DCCWiki. (2022, 20 de mayo). History of digital command control. https://dccwiki.com/DCC_History#Command_Control_in_the_1980s

[6] Roco Modelleisenbahn Quién somos Roco Modelleisenbahnen: Productos y modelos ferroviarios para generaciones. (s. f.). Roco Modelleisenbahn. https://www.roco.cc/es/aboutus/company/index.html

[7] Fleischmann Modelleisenbahn Quién somos Productos y modelos ferroviarios de FLEISCHMANN conectan a generaciones. (s. f.). Fleischmann Modelleisenbahn. https://www.fleischmann.de/es/aboutus/company/index.html

[8] Fleischmann Modelleisenbahn Productos MANDO DE CONTROL Mandos de control digitales 10835 Z21 multiMAUS. (s. f.). Fleischmann Modelleisenbahn. https://www.fleischmann.de/es/product/242299-0-0-0-0-0-0-004001-0/products.html

[9] General Information - Z21 System - Roco z21. (s. f.). Roco Z21 Modellbahnsteuerung https://www.z21.eu/en/z21-system/general-information

[10] Digikeijs DR5000 - DCC Multi-bus command station. (s. f.). Digikeijs (Model railway accessories manufacturer) https://www.digikeijs.com/en/dr5000-dcc-multi-bus-central.html

[11] What is the alexa skills kit? — alexa skills kit. (s. f.). Amazon (Alexa). https://developer.amazon.com/en-US/docs/alexa/ask-overviews/what-is-the-alexa-skills-kit.html

[12] Amazon.es. (s. f.). Amazon.es: compra online de electrónica, libros, deporte, hogar, moda y mucho más. https://www.amazon.es/b?ie=UTF8&node=13944662031

[13] Scrum: Qué es, cómo funciona y por qué es excelente. (s. f.). Atlassian. https://www.atlassian.com/es/agile/scrum

[14] https://medium.com/medium-en-espanol/scrum-para-uno-a895de6010a4

[15] ¿Para qué se utiliza jira software? — atlassian. (s. f.). Atlassian. https://www.atlassian.com/es/software/jira/guides/use-cases/what-is-jira-used-for#jira-for-agile-teams

[16] Herramientas para desarrollo de software y gestión de proyectos. (s. f.). Atlassian. https://www.atlassian.com/es

[17] Test-driven development: Así funciona este método. (s. f.). IONOS Digitalguide. https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-el-test-driven-development/

[18] TDD: Los fundamentos en 10 minutos[2020]. (s. f.). Software Crafters - Craftsmanship, Devops & Tech. https://softwarecrafters.io/javascript/tdd-test-driven-development

[19] AVR microcontroller : All you need to know- (part 1/46). (s. f.). Engineers Garage. https://www.engineersgarage.com/avr-microcontroller-all-you-need-to-know-part-1-46/

[20] Bootloader — arduino documentation — arduino documentation. (s. f.). Arduino Docs — Arduino Documentation — Arduino Documentation. https://docs.arduino.cc/hacking/software/Bootloader/

[21] Dynamic dev tool page — microchip technology. (s. f.). Smart — Connected — Secure — Microchip Technology. https://www.microchip.com/en-us/development-tool/PG164140

[22] (s. f.). https://es.farnell.com/microchip/atatmel-ice-basic/debugger-atmel-arm-avr-basic-kit/dp/2407172

[23] (s. f.). Smart — Connected — Secure — Microchip Technology. https://www.microchip.com/en-us/product/ATmega328P

[24] (s. f.). Smart — Connected — Secure — Microchip Technology. https://www.microchip.com/en-us/product/ATmega32u4

[25] L298 - STMicroelectronics. (s. f.). STMicroelectronics. https://www.st.com/en/motor-drivers/l298.html

[26] L298N motor driver board. (s. f.). Makerfabs. https://www.makerfabs.com/l298n-motor-driver-board.html

[27] ESP32 wi-fi & bluetooth MCU I espressif systems. (s. f.). Wi-Fi & Bluetooth MCUs and AIoT Solutions I Espressif Systems. https://www.espressif.com/en/products/socs/esp32

[28] Microsoft. (2021, 3 de noviembre). Visual studio code - code editing. redefined. Visual Studio Code - Code Editing. Redefined. https://code.visualstudio.com/

[29] (s. f.). National Model Railroad Association. https://www.nmra.org/sites/default/files/s-92-2004-07.pdf

[30] (s. f.). National Model Railroad Association. https://www.nmra.org/sites/default/files/s-9.2.1_2012_07.pdf

[31] (s. f.). National Model Railroad Association. https://www.nmra.org/sites/default/files/standards/sandrp/pdf/s-.2.2_decoder_cvs_2012.07.pdf

[32] MIT app inventor. (s. f.). MIT App Inventor. https://appinventor.mit.edu/

[33] Hello codi! (s. f.). MIT App Inventor. http://appinventor.mit.edu/explore/ai2/hello-codi.html

[34] WiFi - arduino reference. (s. f.). Arduino - Home. https://www.arduino.cc/reference/en/libraries/wifi/

[35] Arduino-esp32/libraries/AsyncUDP at master · espressif/arduino-esp32. (s. f.). GitHub. https://github.com/espressif/arduino-esp32/tree/master/libraries/AsyncUDP

[36] HttpClient - arduino reference. (s. f.). Arduino - Home. https://www.arduino.cc/reference/en/libraries/httpclient/

[37] MQTT Broker - Enterprise ready MQTT broker to move IoT data. (s. f.). HiveMQ https://www.hivemq.com/hivemq/mqtt-broker/

[38] Eclipse Mosquitto. An open source MQTT broker (s. f.) Mosquitto https://mosquitto.org/

[39] (s. f.). https://es.farnell.com

[40] Most Efficient, Economic PCB Solutions for engineers and hobbyists - JLCPCB. (s. f.). PCB Prototype & PCB Fabrication Manufacturer - JLCPCB. https://jlcpcb.com/aboutUs

[41] PCB prototype & PCB fabrication manufacturer - JLCPCB. (s. f.). PCB Prototype & PCB Fabrication Manufacturer - JLCPCB. https://cart.jlcpcb.com/quote

[42] Surface mount (SMT) PCB assembly - JLCPCB. (s. f.). PCB Prototype & PCB Fabrication Manufacturer - JLCPCB. https://jlcpcb.com/smt-assembly?_ga=2.170647328.1857858446.1654905171-643337078.1654905171

[43] Precio kWh hoy: Precio de la luz hora a hora en España. (s. f.). Selectra. https://selectra.es/energia/info/que-es/precio-kwh

APÉNDICE

A.1. Comandos del protocolo de comunicación

- **:00;** Fail command.
- **:01;** Ok command.
- **:02;** Búsqueda de una central.
- **:03;** Deshabilitar booster.
- **:04;** Habilitar booster.
- **:05AAAA;** Stop de emergencia, donde AAAA es 0 para enviar por broadcast, o bien la dirección de un decodificador concreto.
- **:06AAAASS0T;** Velocidad y dirección, donde A es la dirección física del decodificador, SS la velocidad y dirección: 0bDSSSSSSS, y T indica los pasos de velocidad.
- **:07AAAAFF0V;** Ajustar funciones del decodificador, donde A es la dirección física del decodificador, F el identificador de función y V, 1 o 0 según si se quiere activar o desactivar dicha función.

A.2. Diagramas complentarios

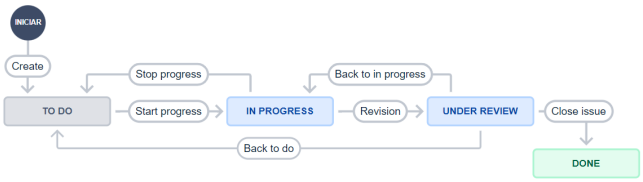

Fig. 10: Diagrama del flujo de trabajo

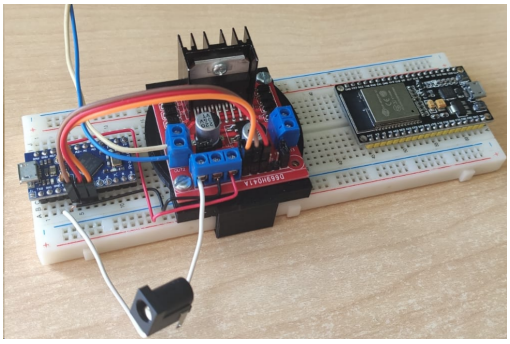

Fig. 11: Diagrama electrónico del sistema

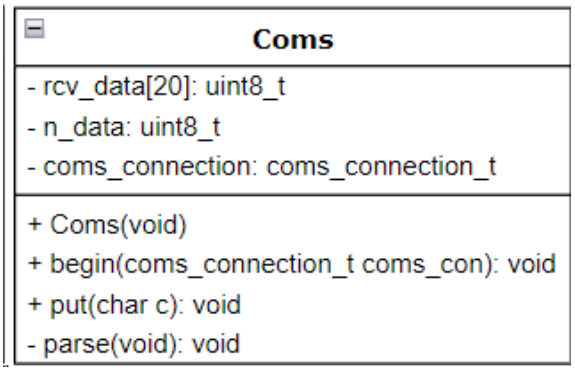

Fig. 12: Diagrama de clases correspondiente a Coms

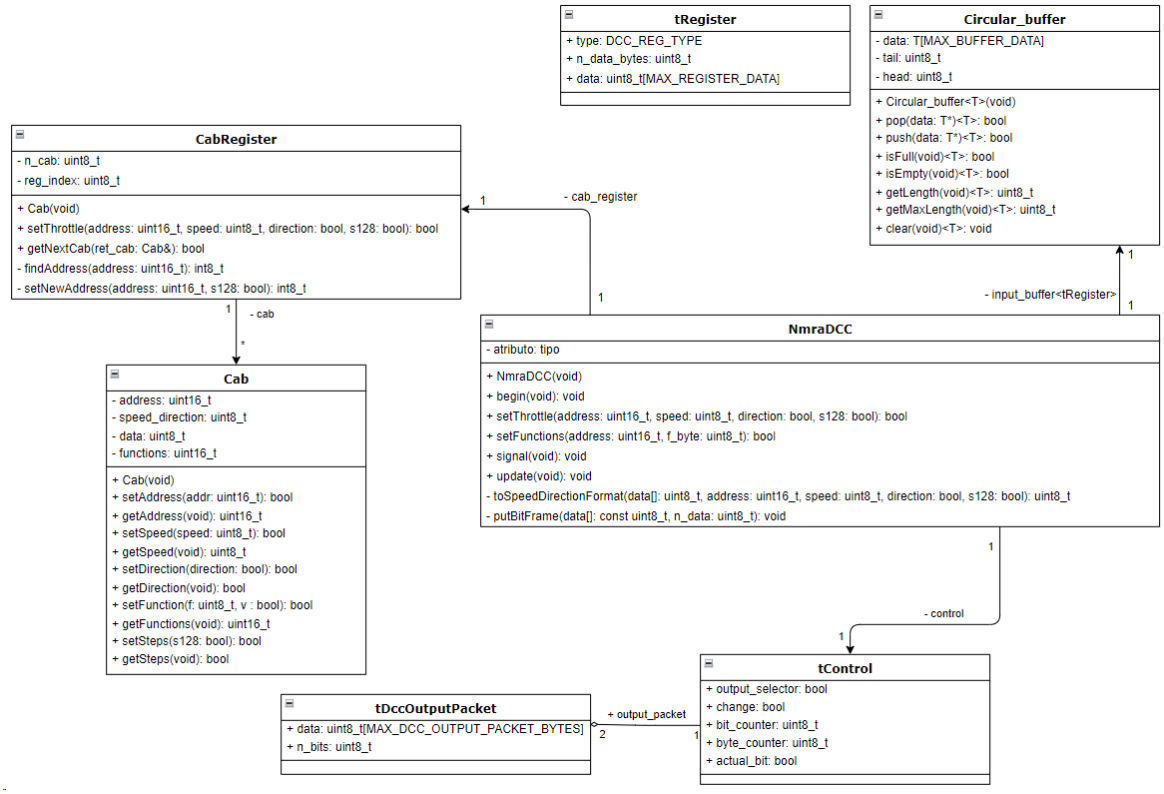

Fig. 13: Diagrama de clases del sistema (mover a los Apéndices)

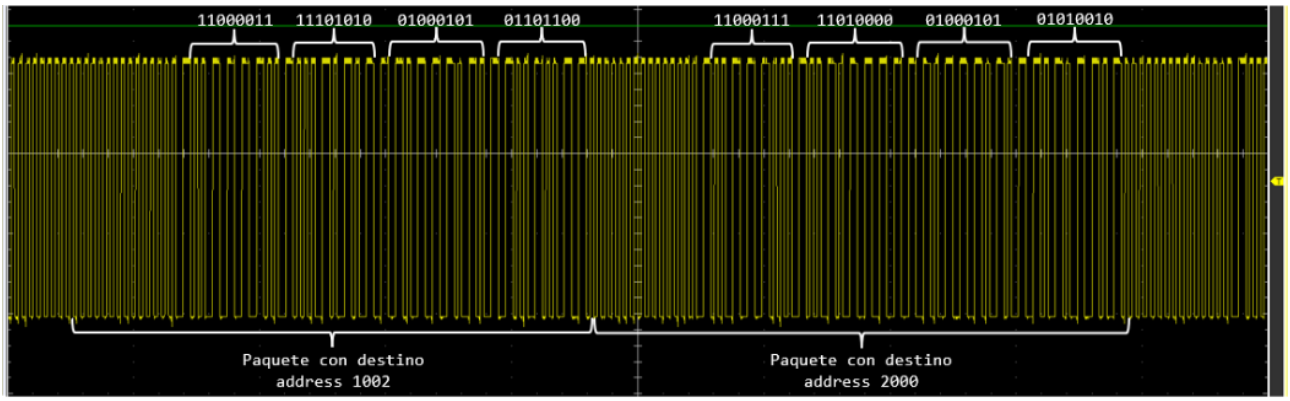

Fig. 14: Señal DCC en osciloscopio

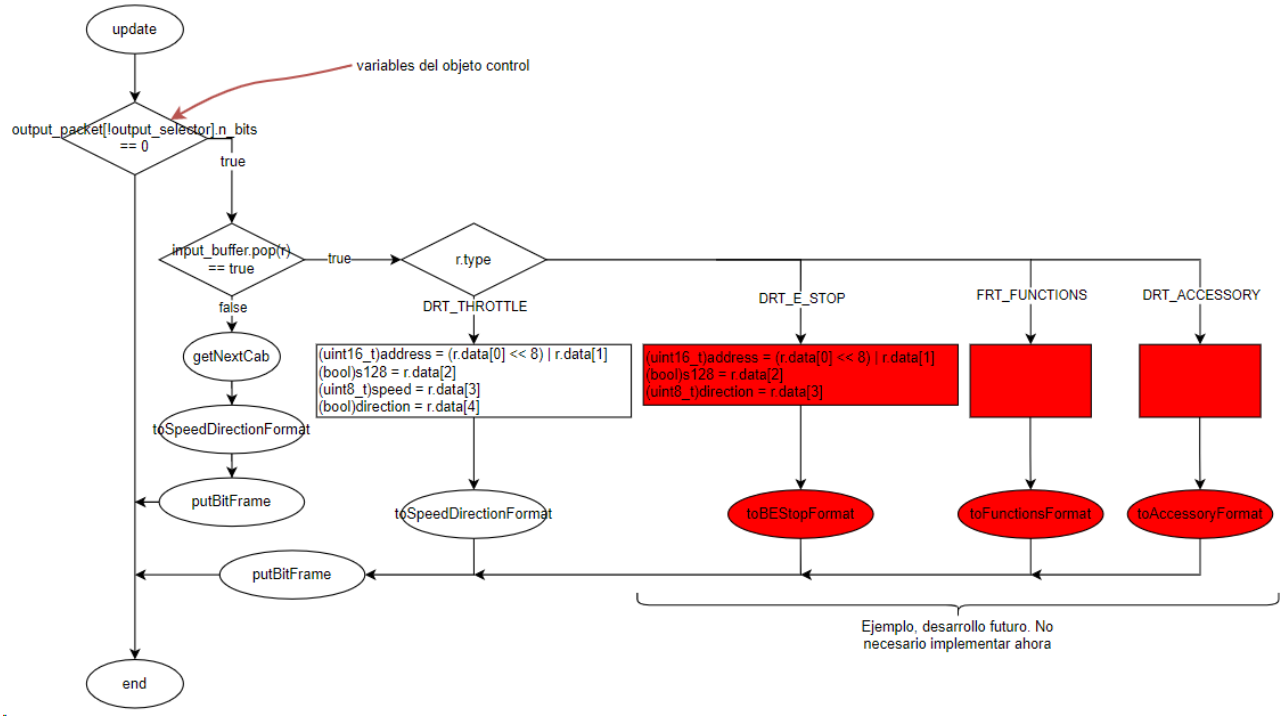

Fig. 15: Diagrama de flujo del método *NmraDCC::update* con más funcionalidades