UC | Facultad de Ciencias

Sistema de soldadura industrial con comunicaciones ethernet

Industrial welding system with ethernet communications

Trabajo de Fin de Máster
para acceder al

MÁSTER UNIVERSITARIO EN INGENIERÍA INFORMÁTICA

Autor: Miguel Martinez Diaz

Director\es: Mario Aldea Rivas
Hector Perez Tijero

Julio - 2024

Agradecimientos

Muchas personas y organizaciones me han acompañado durante esta etapa de estudios, y no sería sin cada una de ellas que esto no hubiera sido posible.

Quiero agradecerle a Equipos Nucleares, por haber proporcionado el material necesario para la realización de este trabajo.

A los profesores del máster, los cuales han sabido transmitir sus conocimientos de una manera muy natural, cercana y entretenida, que siempre han estado disponibles para lo que necesitase, y de quienes me llevo lecciones de vida muy importantes, más allá incluso del contenido de este plan de estudios.

A Héctor Pérez y Mario Aldea, directores de este TFM, que han estado disponibles y me han ayudado siempre que lo he necesitado.

A mis compañeros del máster, con quienes he llegado a tener un sentimiento de fraternidad y amistad que no me imaginé que fuese posible, y han sabido hacerme disfrutar cada momento de esta etapa, y gracias a quienes me llevo de este máster algunos de los mejores recuerdos y amistades que uno pudiera tener.

Y, por último, y, sobre todo, a mi familia, que siempre ha estado presente dando todo el apoyo necesario y más, por mucho sacrificio que supusiese, y me han servido siempre de inspiración y guía para nunca tirar la toalla, por muy difícil que fuese la situación.

Resumen

En el momento actual, se está produciendo un gran cambio en la industria, denominado “Industria 4.0”, donde la automatización, inteligencia artificial (IA), monitorización y adquisición de datos están permitiendo no solo aumentar la eficiencia del proceso productivo, sino también facilitar y agilizar la toma de decisiones y aportar una visión global, detallada y vertical de todo el entorno fabril.

Una de las tecnologías clave de la industria 4.0 es el “internet de las cosas” (IoT, del inglés “internet of things”), donde elementos que tradicionalmente han operado de manera aislada del entono y han trabajado de manera autónoma, tienen capacidad de comunicación con el exterior, permitiendo una monitorización y configuración en tiempo real de todos los elementos implicados en el proceso industrial.

El presente trabajo de final de máster realiza un desarrollo para la implantación del protocolo MQTT sobre TCP/IP en un software de tiempo real, sobre una placa de desarrollo STM32F769I-DISCOVERY, para el control de un módulo hardware análogo a un sistema de soldadura industrial.

Este proyecto parte sobre el trabajo desarrollado por Asier Zulueta Barbadillo “*Controlador de un subsistema de oscilación de una antorcha de soldadura con un procesador de bajo coste*” [1], lo integra contra un módulo hardware y amplía su funcionalidad incorporando el protocolo MQTT para poder establecer comunicaciones mediante ethernet con otros sistemas externos. Finalmente, se explota la información obtenida mediante este protocolo mediante la implementación de un sistema de monitorización en tiempo real en una plataforma IoT.

Palabras Clave

Microcontrolador, Ada, MQTT, TCP/IP, Tiempo Real, Industria 4.0, IoT

Abstract

Currently, a change in the industry is taking place, known as “Industry 4.0”, where automation, artificial intelligence (AI), monitoring and data acquisition are allowing not only to increase the efficiency of the manufacturing process, but also ease decision making.

One of the core technologies of industry 4.0 is the “internet of things” (IoT), where elements that traditionally have been isolated from the environment and have operated in an autonomous way, now have communication capabilities, allowing for a real time monitorization and configuration of all elements present in the industrial process.

This master’s degree dissertation presents a development for the implementation of the MQTT protocol over TCP/IP in a real-time software, over a STM32F769I-DISCOVERY development board, for the control of an analogical hardware module analog to an industrial welding system.

This work is based upon previous work from Asier Zulueta Barbadillo “*Oscillation subsystem controller of a welding torch with a low-cost processor*” [1] integrating it with a hardware module, and also implements the MQTT protocol in order to establish ethernet communications with external systems. Finally, we exploit the information retrieved from the microcontroller by implementing a real time monitoring dashboard with an IoT platform.

Key Words

Microcontroller, Ada, MQTT, TCP/IP, Real-Time, Industry 4.0, IoT

Índice de Contenido

Índice de Ilustraciones

Índice de Listados

Glosario y acrónimos

IoT
Del inglés "***I**nternet **o**f **T**hings*", traducido al castellano como "el internet de las cosas", se trata de la red colectiva de dispositivos conectados y las tecnologías de interconexión entre ellas y sistemas externos como el cloud [2]

Sistema SCADA
Un sistema SCADA (por sus siglas en ingles ***S**upervisory **C**ontrol **A**nd **D**ata **A**cquisition*, en castellano, *Control de supervisión y adquisición de datos*) es un sistema informático que monitoriza, representa y almacena información en tiempo real del estado de un sistema industrial

ADC
Un conversor analógico-digital (del inglés ***A**nalog-to-**D**igital **C**onverter*) es un dispositivo electrónico que recibe señales analógicas de entrada y lo convierte en señales digitales de salida.

API
Una API (del inglés ***A**pplication **P**rogramming **I**nterface*, en castellano *Interfaz de Programación de Interfaces*) es un conjunto de procedimientos y funciones que permiten la integración entre varios sistemas, permitiendo que las funcionalidades de un sistema puedan ser empleadas por otro.

IDE
Un IDE (del inglés ***I**ntegrated **D**evelopment **E**nvironment*, en castellano: *Entorno de Desarrollo Integrado*) es software que proporciona las herramientas necesarias para el desarrollo, codificación, análisis, compilación, depuración, test y despliegue de software.

ARM
ARM hace referencia tanto a la empresa británica *ARM Holdings* como a la arquitectura de procesadores que desarrolla. Acrónimo de "***A**dvanced **R**ISC **M**achine*" (Traducido al castellano como *Maquina RISC Avanzada*) se trata de una arquitectura RISC licenciable, donde, por una parte, ARM Holdings desarrolla y comercializa diseños propios (o Propiedades Intelectuales, IP por sus siglas en inglés), por ejemplo, ARM Cortex, y por otra, ciertas organizaciones disponen de licencia para poder desarrollar sus propias IPs de arquitectura ARM, por ejemplo, *Qualcomm* con los procesadores *Snapdragon*.

Arquitectura RISC
Las arquitecturas RISC son arquitecturas de procesadores con un conjunto reducido de operaciones de tamaño fijo. RISC es acrónimo de "***R**educed **I**nstruction **S**et **C**omputer*", traducido al castellano como "*Computador de conjunto reducido de instrucciones*".

Arquitectura CISC
Las arquitecturas CISC son arquitecturas de procesadores con un conjunto muy amplio de operaciones, algunas de ellas complejas con acceso directo a memoria. CISC es acrónimo de "***C**omplex **I**nstruction **S**et **C**omputer*", traducido al castellano como "*Computador de conjunto complejo de instrucciones*".

ERP
Un ERP (del inglés ***E**nterprise **R**esource **P**lanner*, en castellano *Planificador de Recursos de la Empresa*) es un sistema de información empleado por una empresa u organización para gestionar las actividades diarias, tales como contabilidad, facturación, aprovisionamiento, planificación, gestión de proyectos, gestión de riesgos o gestión logística, entre otros. [3]

1. Introducción

Hoy en día cualquier sistema industrial de vanguardia pertenece a la denominada "Industria 4.0" [4] Entendemos como Industria 4.0 a la revolución industrial que está sucediendo en la que se están incorporando al tejido industrial tecnologías digitales como el internet de las cosas (IoT por sus siglas en inglés de Internet of Things), cloud computing, inteligencia artificial, análisis de datos o machine learning.

En cualquier entorno industrial de una fábrica que haya adoptado tecnologías de Indutria 4.0 nos encontramos el IoT, donde dispositivos que tradicionalmente trabajaban de manera autónoma y aislada ahora son capaces de reportar información en tiempo real. Si juntamos esta capacidad de tener una monitorización en tiempo real de toda una fábrica con técnicas de análisis de datos, seremos capaces de aumentar la eficiencia de los procesos industriales, agilizar la toma de decisiones o implementar técnicas de mantenimiento predictivo con las que poder aumentar al máximo el tiempo de actividad de los dispositivos, llevando al mínimo los costes de mantenimiento. También se puede integrar toda esta información contra sistemas globales de una organización como un ERP y tener una visión vertical y completa de los procesos de una compañía (manufactura, logística, compras, transporte, aprovisionamiento, finanzas…)

Una de las tecnologías que vertebran IoT es el protocolo MQTT [5], que sigue el paradigma publicador-suscriptor, el cual permite comunicaciones uno a uno (one to one) o uno a varios (one to many) de una manera sencilla y fácil de implementar, estas características lo hacen idóneo para múltiples situaciones como comunicaciones dispositivo a dispositivo (M2M, del inglés machine-to-machine) o IoT, caso que nos ocupa, donde memoria, almacenamiento y/o ancho de banda suelen ser recursos escasos.

Otro de los elementos presentes en cualquier entorno industrial son los sistemas empotrados de tiempo real. Por una parte, los sistemas de tiempo real son aquellos que no solo se espera que ofrezcan un resultado correcto, sino también en un plazo bien definido [6], por otra parte, los sistemas empotrados son sistemas de computación diseñados de manera específica para realizar una única tarea en concreto, o un bajo número de ellas. Es habitual que un sistema empotrado sea a su vez un sistema de tiempo real que interactúa y realiza tareas de control con el entorno físico en que se encuentra.

Estos sistemas en la industria suelen ser habituales encontrarlos siendo ejecutados sobre PLCs (del inglés ***P**rogrammable **L**ogic **C**ontroller*, traducido como *Controlador Lógico Programable*), controladores industriales con altas capacidades de entrada y salida, así como gran resistencia a temperaturas y vibraciones. Más sencillos que los PLCs, pero con una finalidad similar, encontramos los microcontroladores, estos dispositivos aúnan en una sola unidad todos los elementos básicos de cualquier sistema de computación: procesador, memoria volátil, almacenaje persistente de datos, dispositivos de entrada y salida y toda la circuitería pertinente. Para este proyecto hemos empleado el Microcontrolador STM32F769 [7], un procesador de prestaciones limitadas y bajo consumo de energía.

Para la programación de estos sistemas, es necesario garantizar que el software producirá las respuestas esperadas en el plazo definido, es por ello, que, dentro de los sistemas empotrados de tiempo real, el lenguaje de programación Ada [8] es una elección habitual en su desarrollo debido a su gran robustez, fiabilidad, prevención de errores y capacidad de análisis estático de código.

1.1 Objetivos

El objetivo general del proyecto es desarrollar un prototipo de sistema de soldadura industrial con soporte para comunicaciones.

Para el desarrollo del prototipo funcional, partimos del trabajo previo realizado por Asier Zulueta Barbadillo para su trabajo final de grado "Controlador de un subsistema de oscilación de una antorcha de soldadura con un procesador de bajo coste" [1]. En este trabajo previo se desarrolló un sistema de control de una antorcha de soldadura que operaba de manera autónoma y aislada sobre hardware simulado.

Considerando las enormes ventajas expuestas que presenta la industria 4.0 y partiendo de la base del proyecto de control de una antorcha de soldadura, queremos integrarlo contra el módulo de hardware físico y actualizarlo para poder introducirlo en el entorno de la industria 4.0, para ello hemos de dotarlo de capacidad de establecer comunicaciones externas mediante el protocolo MQTT, uno de los protocolos más empleados a nivel global en dispositivos IoT para la transmisión de información debido a su simplicidad, bajo consumo de recursos y amplio adopción por múltiples plataformas de referencia, como AWS IoT Core [9], Google Cloud IoT Core [10] o Microsoft Azure IoT Hub [11].

El objetivo general del proyecto se puede descomponer en los siguientes tres objetivos más concretos:

1. Ampliar el proyecto de un sistema de soldadura simulado, integrándolo contra un módulo de hardware análogo al de un entorno de soldadura real, compuesto por elementos físicos como motores paso a paso y sus controladores, potenciómetros y pulsadores entre otros elementos. Este módulo ha sido diseñado, fabricado y proporcionado por Equipos Nucleares SA (ENSA), y es análogo a un sistema de soldadura industrial empleado en sus instalaciones.

   La integración del software de control de soldadura ejecutado sobre el microcontrolador STM32F769 con el módulo hardware proporcionado por ENSA da como resultado la plataforma de control de soldadura

2. Dotar al sistema de soldadura de comunicaciones mediante el protocolo MQTT sobre el stack TCP/IP para lograr reportar el estado del sistema durante su ejecución. La integración de MQTT sobre esta plataforma industrial nos proporciona el soporte básico para poder explotar todas las ventajas que nos ofrece la industria 4.0 como son el mantenimiento predictivo, monitorización remota en tiempo real o adquisición y análisis de datos.

3. Integración de la plataforma industrial contra herramientas IoT. Para ello plasmaremos gráficamente los valores reportados por el microcontrolador sobre su estado a modo de sistema SCADA en la plataforma IoT ThingsBoard [12], tanto los valores reportados en vivo, como aquellos reportados en el pasado.

2. Tecnologías y elementos empleados

En este capítulo presentaremos todos los elementos, tecnologías, dispositivos y técnicas empleadas en el desarrollo de este proyecto, consistiendo, a nivel hardware, de un microcontrolador STM32F769 [13], con un procesador de arquitectura ARM, así como un módulo hardware industrial compuesto por varios elementos electromecánicos.

A nivel software, explicaremos en qué consisten el lenguaje de programación Ada, las librerías empleadas (Ada Drivers Library y ada-ipstack), el perfil Ravenscar-Full, el IDE GNAT Studio, el compilador arm-eabi, la interfaz ST-link, el protocolo MQTT, la plataforma IoT ThingsBoard y la base de datos PostgreSQL asociada.

En cuanto a técnicas de desarrollo, explicaremos en que consiste la compilación cruzada y por qué ha sido indispensable para el desarrollo de este proyecto.

Finalmente, también presentaremos las diferentes herramientas empleadas para la validación del funcionamiento del proyecto, como son el capturador y analizador de paquetes Wireshark y el bróker MQTT mosquitto.

2.1 STM32F769I-DISCOVERY

Microcontrolador desarrollado por el fabricante francoitaliano de semiconductores ST Microelectronics. Este microcontrolador dispone de un procesador ARM Cortex M7 a 216 MHz, 2Mb de Memoria Flash y 532Kb de memoria RAM, así como múltiples puertos de entrada y salida, como un conector ethernet o una interfaz GPIO

Resumimos en la siguiente tabla sus principales características:

Procesador	STM32F769NIH6, basado en ARM Cortex M-7 @ 216MHz, de 32 bits [7]
Memoria	2 Mb de memoria Flash 532 Kb memora RAM
Entrada/Salida	Pantalla táctil TFT de 4” y resolución 800x472 pixeles Conector Ethernet Conector micro-USB 16 pines de entrada/salida digital 6 pines de entrada analógica
Imagen	

Este dispositivo representa el elemento central del prototipo de soldadura industrial. Por un lado ejecutará el software embebido, enviando las señales de actuación de los motores al módulo hardware y recibiendo y procesando las señales digitales y analógica de entrada. Por otro lado, también actúa como nodo publicador en la comunicación MQTT (esto es, en el paradigma publicador-suscriptor, la parte que genera la información), enviando información sobre su estado.

2.2 Lenguaje de programación Ada

El origen de este lenguaje surge en la década de 1970 de la necesidad del departamento de defensa de EE. UU. de aunar bajo un mismo lenguaje de programación el desarrollo de todos sus sistemas empotrados [8]. Tras múltiples revisiones, finalmente en 1983 se publica la primera estandarización del lenguaje Ada. A día de hoy, se han publicado otras cuatro estandarizaciones: 1995, 2005, 2012 y 2022.

El lenguaje de programación Ada es un lenguaje de programación orientado a objetos fuertemente tipado, especialmente diseñado para sistemas críticos, donde la seguridad y fiabilidad sean requisitos primordiales y se requiera un análisis estático del código, con garantías de tiempo real, predictibilidad y certificable. Esto lo hace idóneo para un sistema de control industrial de tiempo real, como es el caso de este proyecto, de ahí su elección.

2.2.1 Perfil Ravenscar-Full

El perfil Ravenscar-Full [14] es un subconjunto del lenguaje Ada cuyo objetivo es restringir el uso de herramientas de tareas, con el fin de que el comportamiento del programa sea predecible y analizable. Este perfil está indicado para sistemas de tiempo real y empotrados que requieran análisis de tiempo real, certificación y garantías de tiempo real y seguridad. Debe su nombre a que se definió en el 8º congreso *International Real-Time Ada Workshops* (*IRTAW*), que tuvo lugar en el año 1997 en la localidad de Ravenscar, Yorkshire, Inglaterra [15]

Este perfil se emplea debido a la necesidad de alta integridad y requisitos de tiempo real del sistema, en el cual es necesario que la generación de las señales de movimiento de los motores y proceso de las señales de entrada se produzcan dentro de plazo razonable.

2.2.2 Tareas y Objetos protegidos

En el lenguaje Ada, la concurrencia se logra mediante el uso de tareas (*tasks*) y objetos protegidos (*protected objects -PO-*).

Por una parte, las tareas son programas que se ejecutan concurrentemente junto con la tarea principal. En otros lenguajes como C, reciben el nombre de hilos o *threads.* Estas tareas pueden ser completamente independientes del resto de tareas, incluyendo la principal, o bien compartir datos o sincronizarse.

Si varias tareas acceden a un mismo dato, puede ocurrir una corrupción de datos, por ejemplo, si durante el proceso de lectura de un dato por parte de una tarea, este es modificado por otra tarea. Con el fin de evitar estas situaciones, en Ada se emplean los objetos protegidos, los cuales encapsulan datos y proporcionan acceso a estos datos mediante operaciones protegidas, esto evita que los datos sean corrompidos debido a condiciones de carrera o accesos concurrentes.

2.3 Librería Ada Drivers Library

La librería Ada Drivers Library es la librería de drivers oficial de AdaCore para distintos microcontroladores en lenguaje Ada, entre ellas el microcontrolador STM32F769, ofreciendo acceso a distintos dispositivos externos como sensores, pantallas, cámaras o dispositivos de audio [16].

Es una librería de código abierto mantenida por la comunidad, donde cada miembro participante es libre de añadir drivers para el microcontrolador y dispositivo que estime oportuno.

Esta librería nos ofrece las funcionalidades necesarias para poder realizar el acceso a los dispositivos de entrada y salida, como la pantalla, los pines de entrada y salida y el puerto ethernet

Por simplicidad, nos referiremos a ella en el resto del documento como “ADL”

2.4 Librería ada-ipstack

Librería desarrollada por Manuel Iglesias Abbatemarco para el implementar el stack TCP/IP y el protocolo MQTT en el lenguaje Ada sobre un microcontrolador STM32F [17]. Esta librería se desarrolló para la competición *Make with Ada 2017* y dispone de todas las funciones necesarias para poder implementar MQTT: establecimiento de conexión, envío de mensajes y cierre de conexión.

Esta librería se ha empleado para integrar el protocolo MQTT y el stack TCP/IP sobre el sistema de control de soldadura.

2.5 IDE GNAT Studio

GNAT Studio es el entorno de desarrollo integrado (IDE) empleado para desarrollar el proyecto. Se trata de un IDE con capacidad para desarrollar software en lenguajes Ada, C, C++ y Python en todas las fases naturales de desarrollo de software: programación, integración, pruebas, depuración y análisis estático. También dispone nativamente de funcionalidad de compilación cruzada.

Es desarrollado y mantenido por AdaCore, empresa fundada en 1995 por miembros del departamento de ciencias de la computación de la universidad de Nueva York para poder comercializar las tecnologías que habían desarrollado para Ada [18]

Para este proyecto hemos empleado GNATStudio en su versión *Community*. Publicada por AdaCore para desarrolladores de software, aficionados y estudiantes. Esta versión incluye todas las funcionalidades que pudiéramos necesitar para el desarrollo de este proyecto.

2.6 MQTT

Protocolo de comunicación de red concebido para dispositivos IoT, debido a su pequeño incremento de código y bajo consumo de recursos, en especial ancho de banda. Es por esta idoneidad para dispositivos de pocos recursos y su fácil integración con plataformas IoT por la que se emplea este protocolo en la comunicación del estado del sistema.

Está basado en el paradigma publicador/suscriptor, donde existe un bróker como árbitro central de todas las comunicaciones. Los clientes publicadores envían mensajes con un tópico asociado, y los clientes suscriptores se suscriben a tópicos para que el bróker les reenvíe los mensajes que reciba con dicho tópico asociado [19].

En la Ilustración 1 podemos ver un sencillo ejemplo de los distintos elementos que intervienen en el protocolo, con un cliente publicador que genera información ("24ºC", asociado al tópico *temperature*), dos clientes suscriptores que se suscriben al tópico *temperature* y un bróker como nodo central a todos ellos, el cual recibe la información del publicador con el tópico *temperature* y se la redistribuye a los clientes suscritos al tópico *temperature*

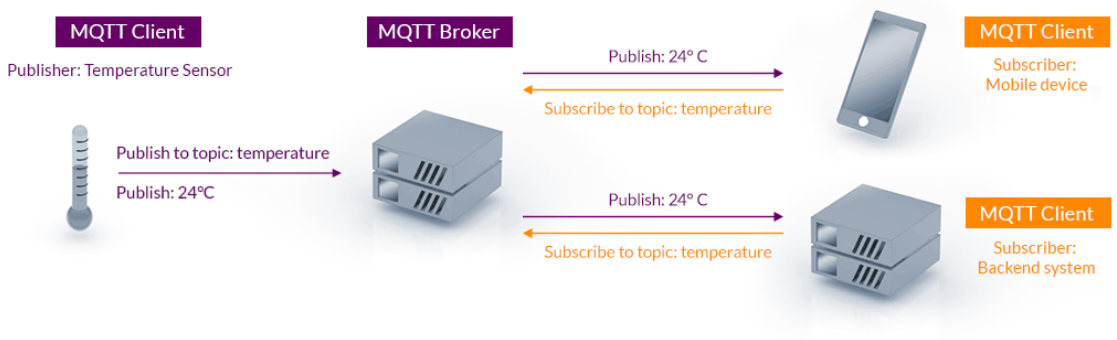


ILUSTRACIÓN 1 DIAGRAMA COMUNICACIÓN MQTT [19]

2.6.1 mosquitto

Bróker de código abierto que implementa los protocolos MQTT. Es parte de la fundación Eclipse y su desarrollo está liderado por Cedalo [20]

Se trata de un bróker muy ligero, capaz de ejecutarse en cualquier tipo de dispositivo, desde clientes ligeros hasta servidores.

Este bróker se ha empleado para validar el funcionamiento correcto de la integración de la librería ada-ipstack.

2.7 Wireshark

Programa de referencia en la captura y análisis de paquetes de red. Permite analizar el tráfico de red de una interfaz, capturando los paquetes que circulen a través de ella para posteriormente poder analizarlos, obteniendo, entre muchos otros valores, las direcciones de origen, destino, protocolo y contenido del mensaje.

Se ha usado para validar la emisión/recepción de paquetes MQTT y su correcto formato, capturando los paquetes enviados desde el microcontrolador hasta el equipo en que se ejecutaban paralelamente tanto este programa como mosquitto o la plataforma Thingsboard.

2.8 ThingsBoard

ThingsBoard es una plataforma IoT de código abierto que permite la captura, almacenamiento, proceso y representación de datos, así como gestión y administración de distintos dispositivos IoT empleando los protocolos más habituales de la industria como HTTP, MQTT o CoAP [12]

Esta plataforma se ha empleado para varias tareas:

- Ejecutar el microservicio de bróker MQTT. Este es un microservicio de la plataforma que actúa como bróker entre el microcontrolador y la propia plataforma
- Desarrollar paneles visuales para mostrar gráficamente el estado del sistema
- Integrar la base de datos PostgreSQL para almacenar de manera persistente la telemetría reportada por el microcontrolador

2.8.1 Microservicio MQTT

ThingsBoard integra un microservicio de transporte para el protocolo MQTT, esto es una API que actúa como bróker en las comunicaciones entre dispositivos y la plataforma.

2.8.2 PostgreSQL

PostgreSQL es una base de datos relacional, de código abierto con mas de 35 años de desarrollo. Tiene sus orígenes en el proyecto POSTGRES de la universidad de California en Berkeley y hoy en día, es una de las bases de datos de código abierto más populares tanto para desarrolladores como para organizaciones, debido a su robustez, fiabilidad y rendimiento [21].

ThingsBoard emplea PostgreSQL como base de datos en la capa de persistencia para el almacenamiento de entidades (configuración, perfiles, paneles, usuarios, dispositivos…) y telemetría (datos)

2.9 Compilación Cruzada

La compilación es el proceso en el cual se convierte el código fuente en código máquina, este proceso lo realiza el compilador y el resultado dependerá de la arquitectura objetivo para la que se realice este proceso. Es posible generar en un dispositivo con una arquitectura código para otro dispositivo con una arquitectura distinta y posteriormente cargarlo en este último para poder ejecutarlo. A este proceso lo conocemos como compilación cruzada [22].

En este caso, se ejecuta el IDE sobre un PC con arquitectura x86 de 64 bits, mientras que el controlador del sistema de control de soldadura dispone de un procesador de arquitectura ARM de 32 bits.

2.9.1 Arm-eabi

Compilador para arquitectura ARM. Se ha empleado para poder hacer compilación cruzada sobre un PC Windows con arquitectura x86_64 para un el microcontrolador, de arquitectura ARM de 32 bits.

Eabi es un acrónimo de Embedded Application Binary Interface, o, traducido a castellano, Interfaz Binaria de Aplicaciones Embebidas, esto es, un conjunto de reglas específicas a una arquitectura hardware sobre cómo el compilador ha de acceder en lenguaje máquina a las estructuras de datos o subrutinas.

2.9.2 St-link

Interfaz para programar el microcontrolador STM32, proporciona un entorno para poder leer, escribir y verificar software en dispositivos STM32 [23]. Gnat Studio lo integra de manera nativa [24]

3. Análisis

Tal y como se ha comentado previamente, este trabajo es una ampliación del proyecto "*Controlador de un subsistema de oscilación de una antorcha de soldadura con un procesador de bajo coste*" [1]. En este proyecto previo se simuló un dispositivo de soldadura por antorcha empleando para ello un microcontrolador ST32F769I-DISCOVERY, desarrollado en el lenguaje de programación Ada. En este proyecto se desarrolló software con garantías de tiempo real para operar dos motores paso a paso que simulaban ser parte de un sistema de soldadura industrial: Un motor sería el destinado al aporte de material a la soldadura y el otro controlaría el movimiento periódico oscilatorio de la antorcha.

En la Ilustración 2, obtenida de la memoria del proyecto del que partimos [1] podemos ver los diferentes elementos que componen el módulo hardware de soldadura industrial: el microcontrolador como elemento central, el cual recibe una señal analógica, en base a la cual calculará la velocidad y sentido del movimiento de los motores, y dos señales digitales de entrada para indicar fallo en alguno de los motores o borrar las alarmas existentes, también genera señales digitales de salida para mostrar información por pantalla y controlar el movimiento de los motores mediante el envío de las señales a los controladores STR2, los cuales, a su vez, son los encargados de enviar la energía eléctrica a cada estator del motor según corresponda.

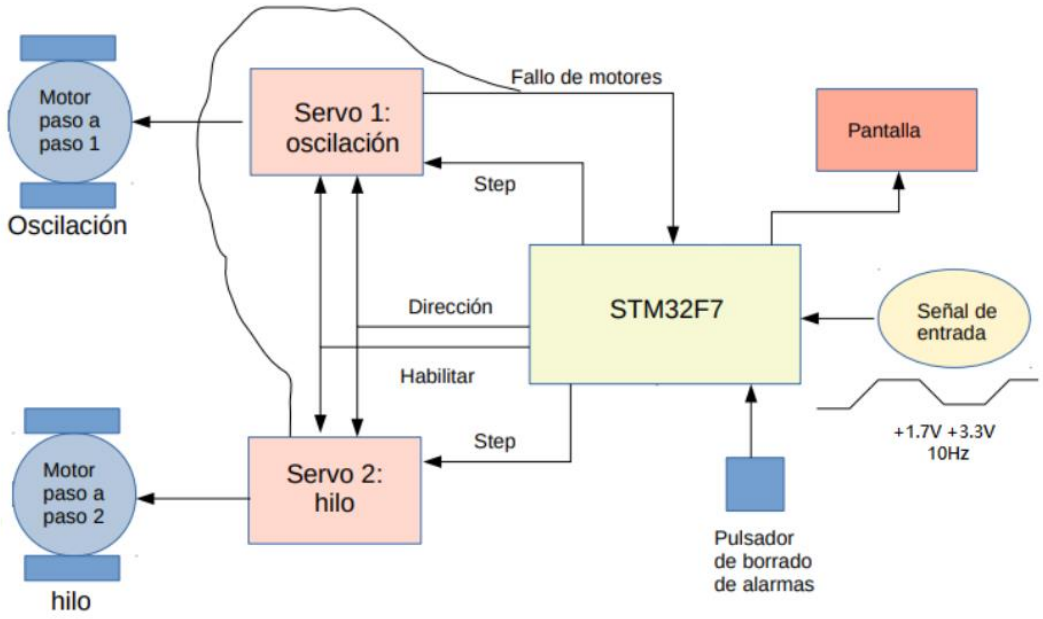


ILUSTRACIÓN 2 ELEMENTOS Y COMUNICACIÓN DEL MÓDULO DE SOLDADURA INDUSTRIAL

3.1 Software previo

El subsistema de control de soldadura simulado fue desarrollado en lenguaje Ada y en él, podemos encontrar los siguientes 9 paquetes desarrollados para este proyecto, junto con la librería Drivers_UC y su funcionalidad descrita a continuación:

3.1.1 Paquete Entrada_Salida

Este paquete gestiona la entrada y salida digital además de la entrada analógica. Se compone de un objeto protegido *Entrada_Salida_PO* con dos funciones: *Init_System*, que iniciará los pines necesarios de la placa, y *DInput_Read* que realiza la lectura de un pin digital de la placa. Fuera del objeto protegido tenemos las funciones *DOutput_Set*, que escribe el valor indicado (0 o 1) en el pin de salida digital indicado, *Configura_Analogica*

que configura el pin A0 como entrada analógica, y *AInput_Read*, que devuelve el valor leído por el ADC en el pin A0, así como funciones públicas *Init_System* y *DInput_Read* para poder invocar las funciones del objeto protegido desde otros paquetes.

3.1.2 Paquete Consola_Avanzada

Este paquete gestiona la pantalla LCD del microcontrolador, proporcionando una capa de abstracción sobre los paquetes *LCD_Std_Out, HAL.Bitmap y STM32.Board,* facilitando y simplificando en un único paquete la impresión por pantalla de mensajes.

3.1.3 Paquete Controlador.Servos

Este paquete recibe las consignas de velocidad de cada motor y genera los pulsos necesarios para que se muevan a la velocidad y dirección adecuada.

Se compone de dos procedimientos *Nuevas_Consignas_Velocidad*, que almacenará la velocidad a la que se tiene que mover los dos servomotores, hasta que la velocidad vuelva a ser sobrescrita en otra llamada al mismo procedimiento. Asimismo, la función *Leer_Consignas_Velocidad* sirve para retornar el valor de consigna del motor que especifiquemos.

También se dispone de un objeto protegido, *Servos_PO,* con la función *Leer_Consigna_velocidad* que retorna la consigna de velocidad que en ese momento haya para el eje que especifiquemos. El objeto protegido dispone a su vez de otros cuatro procedimientos: *Nueva_Consigna_Osc* y *Nueva_Consigna_Hilo,* para escribir la consigna de velocidad del motor destinado a la oscilación y a la alimentación de hilo, respectivamente, y *Nuevo_Evento_Oscilacion* y *Nuevo_Evento_Hilo,* los cuales calculan la frecuencia con la que se tiene que generar un nuevo flanco y lo generan por el pin digital correspondiente.

3.1.4 Paquete Controlador.Parametros

El paquete *Controlador.Parametros* contiene todas las variables globales al proyecto: Prioridad y periodo de cada tarea, canales destinados a cada funcionalidad del módulo hardware, variables estáticas y funciones para la conversión del valor de entrada del ADC a voltios y de voltios a ángulo, etc. También dispone de un objeto protegido con una variable para almacenar el ultimo valor de entrada leído por el ADC y sendos procedimiento y función para escribir y leer en dicha variable.

3.1.5 Paquete Controlador.Sensor_Entrada

Este paquete dispone de un objeto protegido *Sensor_Entrada_PO,* con la función *Leer_Consigna_Entrada,* la cual inicializa el ADC para realizar la lectura de su valor y lo lee, además de escribir en el objeto protegido de *Controlador.Parametros* el voltaje leido

3.1.6 Paquete Controlador.Reportero

El paquete *Controlador.Reportero* incluye la *Reportero_Task* la cual que se encarga de obtener información de los demás paquetes y tareas y mostrarla por pantalla empleando las subrutinas del paquete *Consola_Avanzada.* La información que muestra la velocidad en grados por segundo, número de pasos dado desde el inicio de la ejecución, alarmas activas (si las hubiera) y otros mensajes de relevancia.

3.1.7 Paquete Controlador.Alarmas

El paquete *Controaldor.Alarmas* contiene la tarea *Alarmas_Task* encargada de notificar si se genera alguna alarma en el sistema para ser mostrada por pantalla, y en caso de que exista una alarma y se pulse el pulsador analógico, borrarlas.

3.1.8 Paquete Controlador.Seguimiento_Entrada

El paquete *Controlador.Seguimiento_Entrada* contiene la tarea *Seguimiento_Task* encargada de, periódicamente, realizar la lectura de la señal de entrada para actualizar las consignas de velocidad de los motores.

3.1.9 Paquete Controlador.Guardian

El paquete Controlador.Guardian contiene la tarea *Guardian_Task*, la cual determina si se ha producido un fallo y se ha de notificar y deshabilitar los motores en base al tiempo máximo entre fallos determinado en *Controlador.Parametros.*

3.1.10 Librería Drivers_UC

Se emplea la librería Drivers_UC [25], desarrollada por Daniel Arranz Ortega para el trabajo de fin de grado *Marco para el desarrollo de aplicaciones Ada sobre microcontroladores STM32* [26], la cual simplifica el uso de dispositivos externos en microcontroladores STM32, actuando como capa de abstracción de la ADL

3.2 Módulo hardware

Debido a limitaciones por la pandemia del Covid-19, el proyecto original [1] nunca se llegó a integrar con hardware real y el módulo hardware fue simulado. Es por ello por lo que, uno de los principales objetivos de este proyecto, consiste en adaptar el trabajo previo para operar con un módulo hardware compuesto por distintos elementos electromecánicos, el cual podemos ver en la Ilustración 3:

- Dos motores paso a paso *Applied Motion Products HT17-075D*. Un motor sería el destinado a realizar la alimentación de hilo de soldadura a la antorcha y el otro realizaría el movimiento de la antorcha.

- Dos controladores *Applied Motion Products STR2* [27], uno por cada motor. Ambos controladores están configurados de manera que la resolución de una revolución sean 5000 señales y la dirección y sentido sean dos señales independientes

- Un pulsador

- Un buffer bipolar Texas Instruments TI-SN7407N, el cual convierte señales TTL de 5V de la placa a señales MOS de 3’3V para los controladores del motor.

- Dos potenciómetros BOURNS 3540S-1-502L para regular la frecuencia de oscilación de los motores, aunque solo se emplea uno en este proyecto. En un entorno productivo, serían sustituidos por un generador de señales

- Una fuente de alimentación de 5V y 12V para alimentar eléctricamente todos los elementos

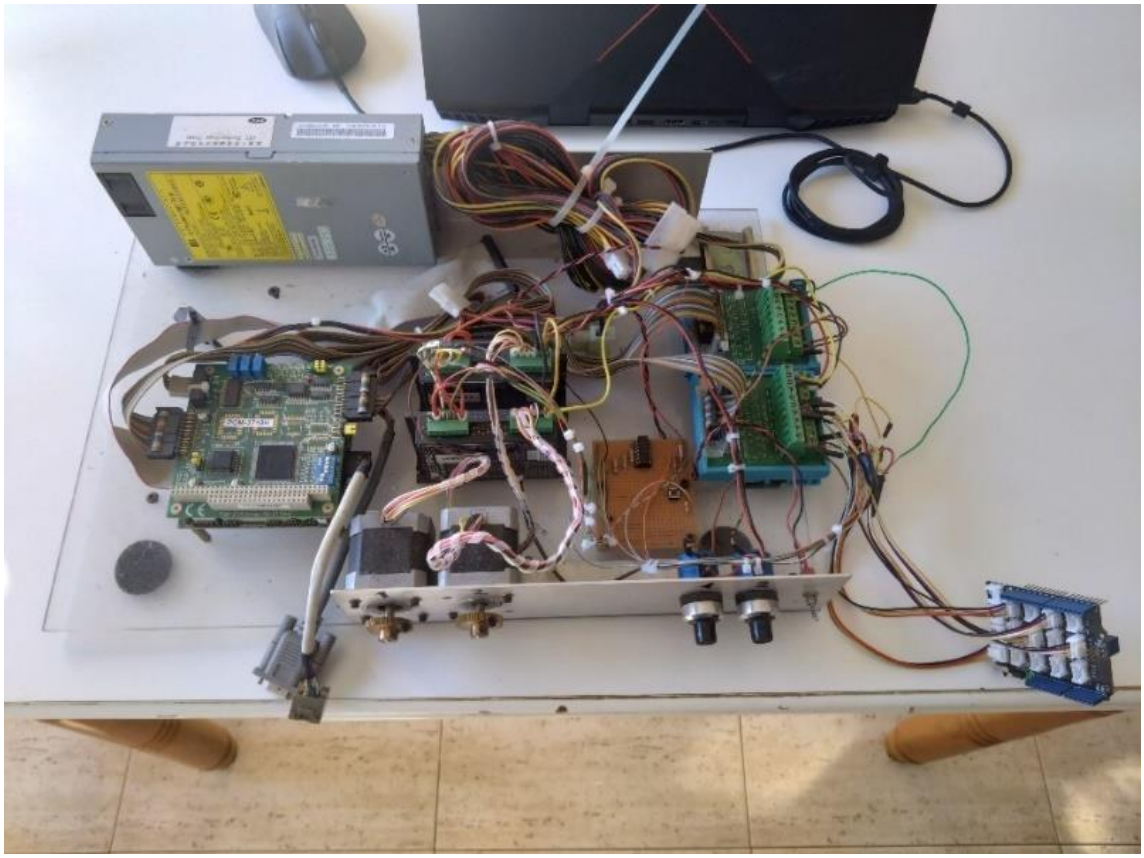


ILUSTRACIÓN 3 MÓDULO HARDWARE

En la Ilustración 4 se muestra el diagrama eléctrico y esquema de conexionado de los distintos elementos que componen el módulo hardware:

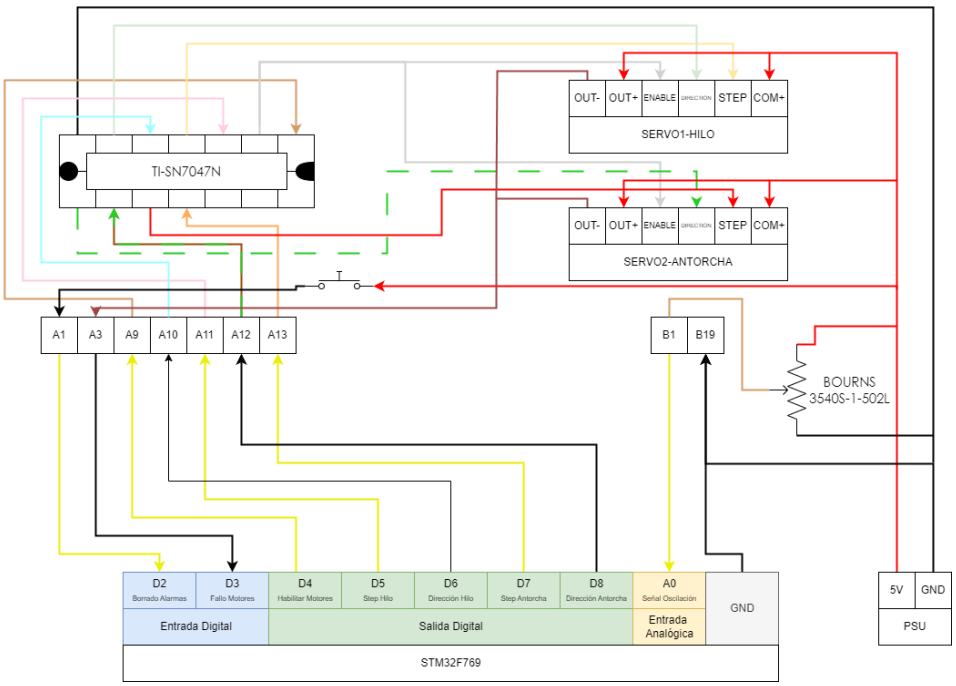


ILUSTRACIÓN 4 DIAGRAMA ELÉCTRICO DEL MÓDULO HARDWARE

En este diagrama podemos identificar los siguientes elementos principales:

- Microcontrolador STM32F769, incluyendo sus pines de entrada Digital (D2 y D3), salida digital (D4, D5, D6, D7 y D8) y entrada analógica (A0), así como una conexión a tierra común a todos los elementos del módulo hardware.

- Zócalos de interconexión (A1, A3, A9, A10, A11, A12, A13, B1 y B19), mostrados en detalle en la Ilustración 5. Simplemente son los elementos de interconexión entre el módulo hardware y los pines del microcontrolador. Se han denotado como "A" a los ubicados a la izquierda y "B" a los de la derecha. Solo se han representado los relevantes al proyecto

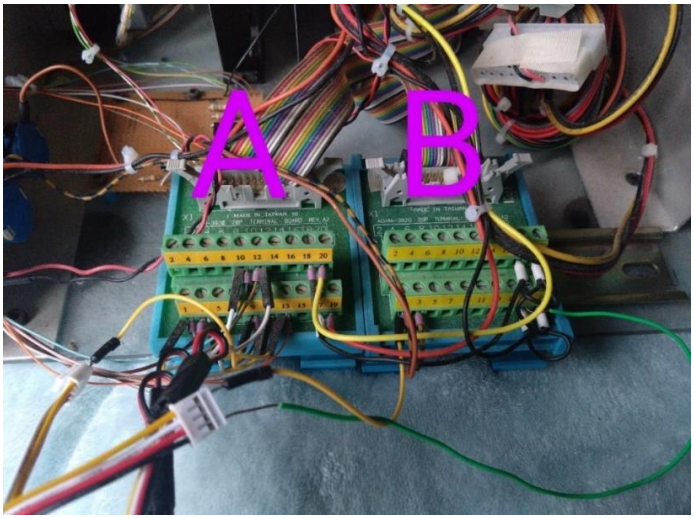


ILUSTRACIÓN 5 DETALLE DE LOS ZÓCALOS DE INTERCONEXIÓN

- Buffer bipolar Texas Instruments SN7047N, el cual se ubica entre los pines de salida digital del microcontrolador (a través de los zócalos de interconexión) y los controladores de los motores a fin de garantizar que el voltaje de entrada está dentro de los valores de electrónica MOS (3’3V) y no TTL (5V) en caso de que el microcontrolador dispusiese de salida de señales a 5V

- Pulsador para enviar la señal de borrado de alarmas, conectada entre la salida de 5V de la PSU y el pin D2 del microcontrolador. En la Ilustración 6 podemos ver en detalle tanto el pulsador como el buffer bipolar

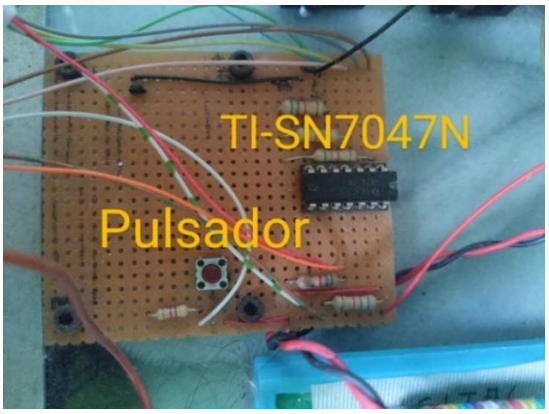


ILUSTRACIÓN 6 PULSADOR Y BUFFER BIPOLAR TI-SN7047N

- Dos controladores de motores paso a paso Applied Motion Products STR2, los cuales reciben las señales desde el buffer. Las señales que reciben indican si el motor está operativo o no (señal ENABLE, o EN), si ha de ejecutar un paso (señal STEP) y el sentido de este (señal DIRECTION o DIR).
  También disponen de dos pines de entrada de alimentación de 5V directa desde la PSU, por una parte, COM+ que actúa como fuente para las señales EN, STEP y DIR y por otra OUT+, el cual actúa como fuente de la señal OUT-, la cual se activa en caso de fallo de motores.
  Las señales OUT- de ambos controladores conectan al mismo pin A3 del controlador, por lo que un fallo de cualquier motor hará saltar la alarma de fallo de motores.

- Un potenciómetro BOURNS 3540S-1-502L, cuyos terminales fijos están conectados a la PSU (GND y 5V) y el terminal variable al pin A0 del microcontrolador mediante el zócalo B1. Solo se emplea uno de los dos disponibles en el módulo. En la Ilustración 7 podemos ver en detalle los potenciómetros

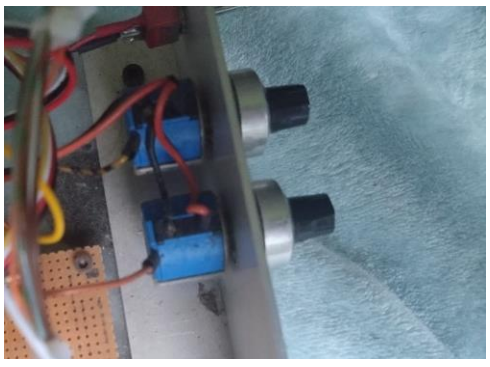


ILUSTRACIÓN 7 POTENCIÓMETROS BOURNS 3540S-1-502L

3.3 Comunicaciones

Posteriormente, a la hora de analizar la mejor solución para implementar las comunicaciones del sistema, debido a la naturaleza del proyecto, al tratarse de un sistema empotrado de recursos muy limitados, se determina que las comunicaciones se realizarán mediante el protocolo MQTT, un protocolo ampliamente usado a nivel industrial en dispositivos IoT debido a su bajo consumo de recursos, en especial ancho de banda. Para implementar este protocolo en Ada, se empleará la librería *ada-ipstack* [17]. Se decide que la manera en que se enviará la telemetría será mediante la creación de una nueva tarea periódica, de baja prioridad, que envíe el último valor de entrada leído por la tarea *Seguimiento_Task.*

Finalmente, para poder demostrar el potencial que supone disponer de dispositivos IoT en un área industrial, desplegaremos la plataforma ThingsBoard para poder capturar la telemetría del microcontrolador y poder representarla de manera gráfica en tiempo real análogo a un sistema SCADA.

4. Diseño, desarrollo e implementación

En este capítulo abordaremos las etapas de diseño, desarrollo e implementación del proyecto, primero de una manera global como proyecto IoT, y luego centrándonos en cada capa, que coincide con las fases de desarrollo del proyecto.

Comenzaremos explicando el diseño adoptado para la plataforma de control de soldadura industrial y cómo se ha logrado integrar el software de control de soldadura con el módulo hardware, así como los cambios que han sido necesarios realizar para su correcto funcionamiento.

Posteriormente, comentaremos el proceso que hemos realizado para conseguir integrar la librería *ada-ipstack* con el subsistema de control de soldadura y lograr así la capacidad de poder crear una tarea periódica que reporte el estado del sistema mediante el protocolo MQTT.

Finalmente, se abordará la implementación de la plataforma IoT ThingsBoard, la comunicación de la placa con esta y las funcionalidades que hemos logrado explotar de ella.

4.1 Diseño global del proyecto

A nivel de integración entre los diferentes sistemas, abordamos este proyecto como un proyecto IoT sencillo. Varios autores defienden que una arquitectura IoT se compone solamente de tres capas [28], como podemos ver en la Ilustración 8:

1. Capa de percepción: Se trata de la capa que interactúa con el entorno (sensores y actuadores).

2. Capa de red: Se trata de la capa que conecta dispositivos IoT con los controladores/hubs a los que se conectan.

3. Capa de aplicación: Se trata de la capa que opera con los datos obtenidos por la capa de percepción

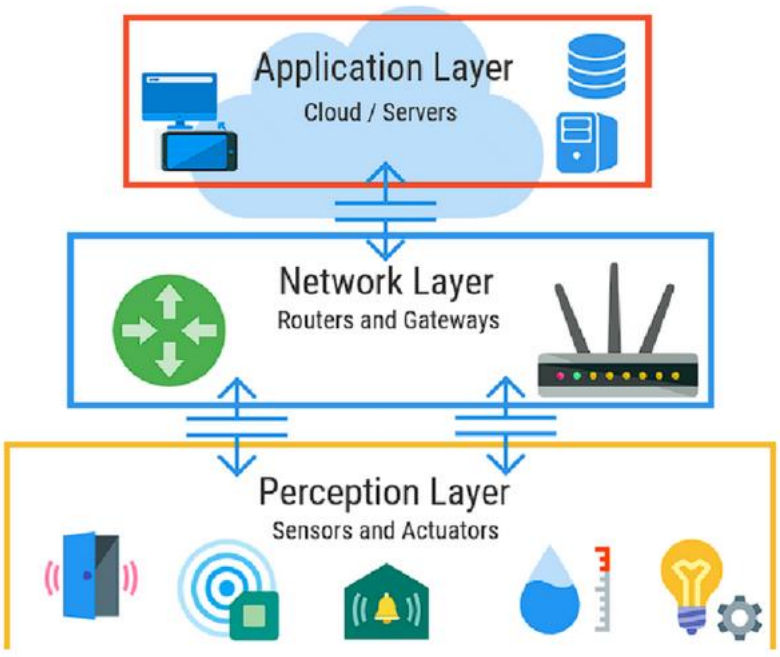
1. 

ILUSTRACIÓN 8 DIAGRAMA DE TRES CAPAS DE ARQUITECTURA IOT [29]

Estas tres capas en este proyecto se corresponderían con:

1. Capa de percepción: En esta capa encontramos el microcontrolador STM32F769, el cual realiza la lectura del voltaje de la señal analógica del módulo hardware y genera las señales de movimiento de los motores. Esta capa se correspondería con la plataforma del sistema de control de soldadura industrial.

2. Capa de red: En esta capa encontramos la red ethernet entre el PC y el microcontrolador STM32, empleando el protocolo MQTT sobre TCP/IP. La librería *ada-ipstack* proporciona la funcionalidad necesaria para el envío de mensajes mediante el protocolo MQTT sobre el stack TCP/IP, mientras que la tarea periódica envía un mensaje de manera periódica con la información del estado de la plataforma.

3. Capa de aplicación: En esta capa encontramos la plataforma ThingsBoard empleada para la representación de valores de señal de entrada del microcontrolador, junto con la base de datos PostgreSQL empleada para la persistencia de datos

La Ilustración 9 muestra una visión global de los elementos que componen la solución final adoptada, y la interacción entre cada uno de ellos:

Por una parte, en la capa de percepción encontramos el microprocesador STM32F769, ejecutando el software de control de sistema de soldadura y comunicándose mediante entrada y salida digital y analógica con el módulo hardware.

Por otro lado, nos encontramos la capa de red, en la que el microcontrolador STM32F769 envía mensajes mediante el protocolo MQTT sobre TCP/IP a la plataforma ThingsBoard, la cual dispone del microservicio MQTT como punto de entrada de estos mensajes.

Finalmente, en la capa de aplicación encontramos la propia plataforma ThingsBoard con el microservicio MQTT embebido y una base de datos PostgreSQL proporcionando la capa de persistencia.

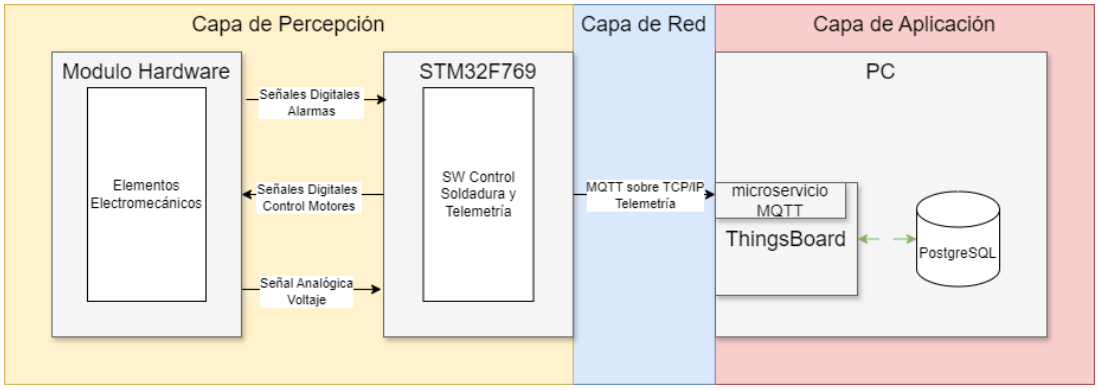


ILUSTRACIÓN 9 VISIÓN GLOBAL DEL SISTEMA

Una vez se ha proporcionado una visión global del diseño del sistema de soldadura industrial, se detallará a continuación el trabajo desarrollado en cada una de las capas que lo componen:

4.2 Capa de percepción

En este proyecto, la capa de percepción se corresponde con la plataforma del sistema de soldadura industrial. Tal y como hemos comentado con anterioridad, este trabajo parte del proyecto original del sistema de control de soldadura, en el cual el módulo hardware fue simulado. En este proyecto, se modifica el proyecto original hasta lograr integrarlo con un módulo hardware real.

A modo esquemático, en la Ilustración 10 vemos la integración entre los diferentes elementos hardware y software que lo componen. En este apartado abordamos solamente la capa de percepción por lo que nos centramos en la plataforma de control de soldadura y su integración con el módulo hardware, su integración con las capas de red y de aplicación lo abordaremos más adelante:

ILUSTRACIÓN 10 DIAGRAMA DE CAPAS DE LA CAPA DE PERCEPCIÓN

4.2.1 Plataforma de control de soldadura

Esta plataforma parte de [1] y se conserva su estructura básica, con las tareas y paquetes expuestos previamente en la parte de software de control de soldadura (Sección 3.2 Software previo).

Sin embargo, tras una primera integración del módulo hardware, se observa que las señales analógicas de entrada varían entre 0 y 3’3V, y se decide que este voltaje determinará, de manera lineal, la velocidad de giro de los motores, desde -15º/s hasta +15º/s (0V=-15º/s, 1’65V=0º/s y 3’3V = 15º/s). También se modifica el valor de consigna mostrado por pantalla, que pasa de ser el voltaje recibido por el ADC a la velocidad determinada para los motores.

Asimismo, en el proyecto original, la gestión del paso de cada motor era independiente, pero no así la de su sentido, por lo que un motor podría estar detenido y el otro girar en cualquier sentido, aunque no podrían girar ambos a la vez en sentidos contrarios. Se decide independizar las señales de sentido de cada motor, de modo que se dota al sistema de mayor versatilidad para futuros proyectos.

4.2.2 Adaptación del sistema de soldadura al módulo hardware

Debido a que el proyecto del que partimos nunca se integró contra hardware real, nos encontramos con que el conexionado que se definió en el proyecto original no era el más cómodo ni más organizado a la hora de conectarlo a nivel físico contra el módulo hardware, así como también, debido a la separación de la gestión de la dirección los motores, necesitaremos un pin extra de salida digital para gestionar el sentido del segundo motor. Es por ello por lo que, para integrar el módulo hardware, primeramente, se redefinen los pines del microcontrolador y su funcionalidad, tanto los de entrada y salida digital, como el de entrada analógica.

Describimos en la siguiente tabla cada pin, su sentido (entrada o salida) y tipo (analógico o digital) y su funcionalidad.

Pin	Sentido y tipo	Funcionalidad
D2	Entrada digital	Borrado de alarmas
D3	Entrada digital	Fallo de motores
D4	Salida digital	Habilitar motores
D5	Salida digital	Paso motor hilo
D6	Salida digital	Dirección motor hilo
D7	Salida digital	Paso motor antorcha
D8	Salida digital	Dirección motor antorcha
A0	Entrada analógica	Señal de oscilación

Además, originalmente, se determinó que la señal de entrada variaría entre 1’7V y 3’3V. Sin embargo, debido a que el conversor analógico-digital tiene capacidad de procesar valores entre 0V y 3’3V y esto nos permite trabajar con mayor precisión, se decide que estos serán los valores con los que trabajará la plataforma, lo cual conlleva redefinir las funciones que procesan las señales de entrada analógica

Existen funciones en el paquete *Controlador.Parametros* que, junto con variables estáticas dentro del mismo paquete, sirven para calcular, en base al valor leído del ADC (número entero entre 0 y 2045), el voltaje equivalente, velocidad angular de los motores en base al voltaje, desplazamiento realizado desde el inicio del programa, tanto en grados como en pasos.

A continuación, en el Listado 1 se muestra un resumen de estas variables y funciones que encontramos en el paquete *Cotrolador.Parametros* con su definición para el proyecto actual, donde los servos STR2 están configurados para que una revolución sean 5000 pasos.

Listado 1. Resumen del paquete Controlador.Parametros

```
1.   --  Rango de la señal de entrada

2.    Max_Senal_Entrada : constant Voltaje := 3.3;
3.    Min_Senal_Entrada : constant Voltaje := 0.0;
4.    Posiciones_Iniciales : constant Angulos_Ejes := (0.0, 0.0);

5.    --Velocidad maxima de los ejes en grados por segundo, tanto en un sentido como en
otro
```

```
 6.    Velocidad_Maxima : constant Velocidad_Angular := 15.0;

 7.    --  Factor y offset para conversion de pasos a angulos en grados/paso
 8.    --  Contolador STR2 configurado para 5000 pasos/revolucion
9.    Factores_Paso_Angulo : constant array (Tipo_Eje) of Float :=
10.      (360.0 / 5000.0, 360.0 / 5000.0);

11.   Offsets_Paso_Angulo : constant Angulos_Ejes := (0.0, 0.0);

12.   Ganancia_Convertidor_Consigna_Entrada : constant Voltaje := 3.3 / 4095.0; -- Valor
4095 para 3.3V

13.   Offset_Convertidor_Consigna_Entrada : constant Voltaje := 0.0; -- Para esta
aplicación, con este ADC el offset es 0

14.   --------------------
15.   -- Pasos_A_Angulo --
16.   --------------------
17.   --  Convierte de pasos a angulo en grados
18.   --  Angulo = Factor*Pasos+Offset
19.
20.   function Pasos_A_Angulo (P : Pasos_Motor; Eje : Tipo_Eje) return Angulo is
21.   begin
22.      --  Usamos Doble_Precision (type digits 15) para disminuir los errores de
redondeo
23.      return Angulo (Doble_Precision (Factores_Paso_Angulo (Eje)) *
24.                     Doble_Precision (P)) + Offsets_Paso_Angulo (Eje);
25.   end Pasos_A_Angulo;

26.   --------------------
27.   -- Angulo_A_Pasos --
28.   --------------------
29.   --  Convierte de angulo en grados a pasos de motor
30.   --  Pasos = (Angulo-Offset)/Factor
31.   function Angulo_A_Pasos (A : Angulo; Eje : Tipo_Eje) return Pasos_Motor is
32.   begin
33.      --  Usamos Doble_Precision para disminuir los errores de redondeo
34.      return Pasos_Motor (Doble_Precision (A - Offsets_Paso_Angulo (Eje)) /
35.                          Doble_Precision (Factores_Paso_Angulo (Eje)));
36.   end Angulo_A_Pasos;

37.   ---------------------------
38.   -- Convertidor_A_Voltios --
39.   ---------------------------
40.   --  Convierte cuentas de convertidor a voltios
41.   --  Voltios=ganancia*lectura_conversion+offset
42.   function Convertidor_A_Voltios (C : Unidades_AI) return Voltaje is
43.   begin
44.      if C < 15 then  -- en teoría, 0=0V, 2045=3.3V. Si el valor devuelto por el ADC
es menor a 15, entendemos que puede ser ruido, por ello lo damos como 0
45.         return 0.0;
46.      else
47.         return Voltaje (C) * Ganancia_Convertidor_Consigna_Entrada +
Offset_Convertidor_Consigna_Entrada;
48.      end if;
49.
50.   end Convertidor_A_Voltios;

51.   ----------------------
52.   -- Voltios_A_Angulo --
53.   ----------------------
54.   --  Convierte Voltios a Angulo deseado para el motor indicado
55.   --  Angulo=factores_voltios_angulo*voltios
65.
57.   function Voltios_A_Angulo (V : Voltaje; Eje : Tipo_Eje) return Angulo is
58.   begin
59.      return Angulo (V) * Factores_Voltios_Angulo (Eje) + Offset_Voltios_Angulo(Eje);
60.   end Voltios_A_Angulo;
```

Asimismo, con el fin de aumentar la versatilidad del sistema de control de soldadura, se pasa a gestionar el movimiento de cada motor de manera independiente. Originalmente ambos motores realizaban el mismo movimiento, tanto en velocidad como en sentido empleando las mismas señales para ambos controladores STR2. En proyecto se ha independizado la gestión de cada motor, tanto en paso como en sentido para potencialmente en un futuro, realizar movimientos completamente distintos en cada motor, gestionando cada uno con una señal de entrada independiente.

4.3 Capa de red

La capa de red de este proyecto se corresponde con la mensajería mediante el protocolo MQTT entre la plataforma de soldadura industrial y la plataforma ThingsBoard, para ello, se ha integrado la librería *ada-ipstack* y se ha creado una tarea periódica *TelemetryMQTT* poder integrar comunicaciones MQTT a la capa de percepción. Los nuevos elementos de la capa de red se muestran en color azul en la siguiente Ilustración 11

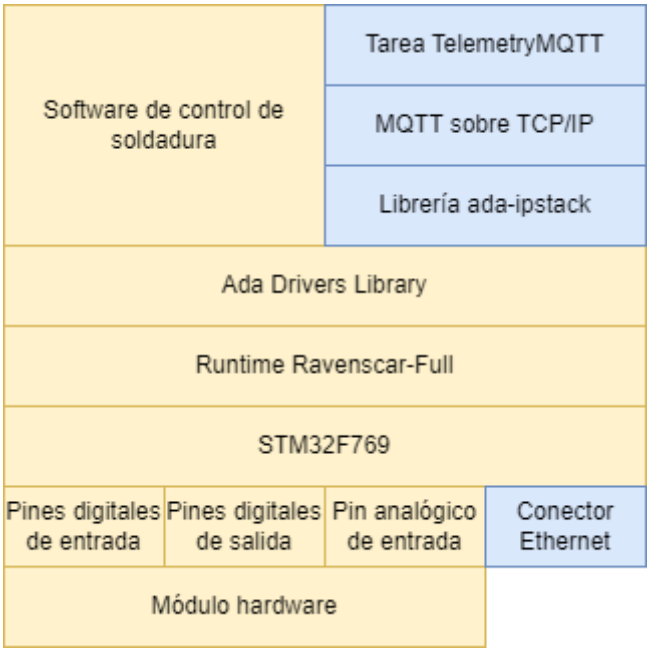


ILUSTRACIÓN 11 DIAGRAMA DE CAPAS DE LA CAPA DE PERCEPCIÓN Y RED

Para desarrollar esta capa, se han realizado las siguientes tareas:

4.3.1 Integración de la librería ada-ipstack

El proyecto original era un sistema que trabajaba de manera autónoma y aislada del entorno, a excepción de las señales de entrada y salida con el módulo hardware, sin embargo, carecía de comunicaciones a alto nivel que permitiesen el envío de información a otros sistemas y, por ello, no era posible integrar la plataforma en un entorno de la industria 4.0. Es por ello, que se decide implementar el protocolo MQTT y poder así monitorizar el estado de la plataforma, para ello, se emplea la librería *ada-ipstack*, que además del stack TCP/IP, implementa el protocolo MQTT, con todas las funcionalidades requeridas para el establecimiento de conexión, envío de mensajes y cierre de conexión necesarias.

Debido a que la librería *ada-ipstack* emplea una versión de la ADL de 2017 [30], así como el compilador arm-eabi de 2017, existen problemas de compatibilidad a la hora de integrarlo con el proyecto de control de soldadura, que emplea una versión de la ADL más moderna [16], en la cual se han omitido ficheros de configuración empleados por la librería *ada-ipstack*, además, también se compiló con el compilador arm-eabi de 2020 para el cual, ciertas directivas de compilación de la librería *ada-ipstack*, así como funcionalidades del lenguaje estaban obsolescentes.

Otra de las limitaciones encontradas es que Ada no permite que dos ficheros tengan el mismo nombre, aunque pertenezcan a librerías distintas, por lo que no es posible trabajar con dos versiones de ADL distintas. Se decide por ello, integrar ambos proyectos con una misma ADL, la de 2017 presente en la librería *ada-ipstack*, y compilarlo con el compilador arm-eabi 2017. Sin

embargo, con el fin de que ambos proyectos puedan funcionar con la misma ADL, se tuvieron que hacer ciertos cambios.

Por una parte, la librería *drivers_UC* pasa a importar el proyecto *stm32f769_discovery*, en lugar de *stm32f769_discovery_full*, este proyecto (*stm32f769_discovery*) estaba disponible en la ADL de 2017, pero se eliminó en futuras versiones en favor de las versiones del proyecto **_sfp* y **_full*. Este proyecto es empleado tanto por *ada-ipstack* como por la librería *Drivers_UC*. Al haber prescindido del proyecto *stm32f769_discovery_full*, las variables de configuración del Runtime a emplear ("*ravenscar-full-stm32f769disco*"), así como las directivas de compilación se han de especificar directamente en el fichero de configuración del proyecto de control de soldadura.

Por otra parte, en el proyecto del que partimos, se modificó el paquete *LCD_Std_Out*, al cual se le añadieron las funciones *Ultima_Fila, Ultima_Columna, Current_Character y Current_Line*, las cuales hemos de conservar.

Otros de los paquetes que sufrieron modificaciones entre una versión de ADL y otra fueron el *STM32.GPIO, HAL.GPIO, STMPE1600* y *mcp23x08*, los cuales conservamos el de ADL del proyecto original.

4.3.2 Integración de comunicaciones en el software de control de soldadura

En el apartado 4.2 anterior se han descrito los cambios y mejoras realizadas en el software de control de soldadura relacionadas directamente con la integración del módulo hardware. Además de estos cambios mencionados, es necesario añadir soporte para las comunicaciones en dicho software de control de soldadura, para ello, se ha creado el paquete *Controlador.TelemetriaMQTT*, el cual contiene la tarea *MQTT_Task*, la cual enviará los mensajes con información sobre el estado del sistema mediante MQTT, aunque antes, ha sido necesario crear un objeto protegido para realizar el intercambio de información entre el sistema de control de soldadura mediante la tarea *Seguimiento_Task* del paquete *Controlador.Seguimiento_Entrada* (la cual ha debido ser modificada) y esta nueva tarea

Tal y como se vio en el apartado 2.2.2, la manera que existe en Ada para garantizar el acceso concurrente a un dato es mediante el uso de tareas y objetos protegidos, es por ello por lo que se crea un nuevo objeto protegido en el paquete *Controlador.Parametros* llamado *Voltage_Value_PO* en el cual existe una variable privada llamada *Last_Senal_Entrada*, un procedimiento para asignarle valor llamado *Set_Value* y una función para leerlo llamada *Get_Value*. Esta variable tendrá el valor escrito por la función *Leer_Consigna_Entrada* del paquete *Controlador.Sensor_Entrada*, la cual es invocada por la tarea *Seguimiento_Task* del paquete *Controlador.Seguimiento_Entrada* en cada ejecución, y será leída por la tarea *MQTT_Task*, es así como se realiza el intercambio de información sobre el estado del sistema entre el sistema de control de soldadura y la nueva tarea.

Dentro del paquete *Controlador.Parametros* también se crea el procedimiento y función pública auxiliar para poder ejecutar el procedimiento y función del objeto protegido desde la tarea de lectura de señal del paquete *Controlador.Sensor_Entrada* y la tarea periódica *Controlador.TelemetryMQTT*, los cuales reciben el mismo nombre que las funciones del objeto protegido, *Set_Value* y *Get_Value*

A continuación, en el Listado 2 podemos ver el código fuente del objeto protegido *Voltage_Value_PO*, con su variable privada *Last_Senal_Entrada* de tipo *Voltaje* y funciones privadas de lectura y escritura *Get_Value* y *Set_Value* respectivamente.

Fuera del objeto protegido encontramos otras dos funciones de lectura y escritura con el mismo nombre, que solamente llaman a las funciones del objeto protegido para así poder acceder a ellas desde otros paquetes.

Listado 2. Código Fuente del Objeto Protegido con la variable compartida

```
1.    protected Voltage_Value_PO is
2.       --  Procedimiento que escribe la variable.
3.       procedure Set_Value (V:Voltaje);
4.       --  Funcion que lee la variable.
5.       function Get_Value return Voltaje;
6.    private
7.       Last_Senal_Entrada :  Voltaje;
8.    end Voltage_Value_PO;
9.
10.   protected body Voltage_Value_PO is
11.      procedure Set_Value (V:Voltaje) is
12.      begin
13.         Voltage_Value_PO.Last_Senal_Entrada:=V;
14.      end Set_Value;
15.      function Get_Value return Voltaje is
16.      begin
17.         return Last_Senal_Entrada;
18.      end Get_Value;
19.   end Voltage_Value_PO;
20.
21.   procedure Set_Value (V:Voltaje) is
22.   begin
23.      Voltage_Value_PO.Set_Value(V);
24.   end Set_Value;
25.
26.   function Get_Value return Voltaje is
27.   begin
28.      return Voltage_Value_PO.Get_Value;
29.   end Get_Value;
```

Asimismo, la función *Leer_Consigna_Entrada* del paquete *Controlador.Sensor_Entrada* se modifica para que escriba el valor leído del ADC en la variable *Last_Senal_Entrada* del objeto protegido *Voltage_Value_PO*. Esto se realiza debido a que el acceso al ADC es un acceso a un dispositivo, lo cual es más costoso en tiempo que si se hiciese un acceso a memoria, por lo que delegamos el acceso al ADC exclusivamente a esta función, invocada en cada ejecución por la tarea *Controlador.Seguimiento_Entrada*

Con este diseño logramos dos objetivos:

1. Evitar accesos innecesarios a un dispositivo hardware por parte de la tarea *TelemetryMQTT*

2. Evitar carreras críticas o corrupción de datos en la lectura/escritura de una variable compartida entre las tareas

Por último, como se ha comentado, se ha de enviar la telemetría de manera periódica, esto lo haremos mediante la creación de una nueva librería que se ocupe de establecer la conexión MQTT y el envío de la telemetría con una periodicidad de un segundo.

Partiendo de la librería *ada-ipstack* [17] ya integrada, se desarrolla el paquete *Controlador.TelemetryMQTT*, la cual contiene la tarea *MQTT_Task*, y cuyo pseudocódigo podemos ver en el Listado 3. Esta tarea comienza estableciendo la conexión MQTT y posteriormente, de manera periódica e indefinida, envía mensajes con el valor que en ese momento tenga la variable *Last_Senal_Entrada* leído por la tarea *Seguimiento_Task* y su valor equivalente en grados por segundo. En este proyecto, enviaremos el último voltaje leído por el

conversor analógico-digital y su valor equivalente en grados por segundo, lo cual es una correlación lineal.

Listado 3. Pseudocódigo de la tarea TelemetryMQTT

```
1. Begin MQTT_Task
2. Inicializa stack IP, TCP, MQTT...
3. Establece conexión MQTT
4. Tiempo siguiente es tiempo actual
5. Bucle
6.       Tiempo siguiente es Tiempo siguiente + Periodo
7.       Envia mensaje telemetría
8.       Duerme hasta Tiempo siguiente
9. Fin bucl
```

En la Ilustración 12 de la siguiente página podemos ver la interacción entre la tarea *Seguimiento_Task, MQTT_Task* y el objeto protegido *Voltage_Value_PO* del paquete *Controlador.Parametros*, donde, por una parte, la tarea *Seguimiento_Task* del paquete *Controlador.Seguimiento_Entrada* invoca a la función *Leer_Consigna_Entrada* del paquete *Controlador.Sensor_Entrada*, esta función realiza la lectura de valor del ADC y lo escribe en la variable *Last_Senhal_Entrada* del objeto protegido *Voltage_Value_PO* del paquete *Controlador.Parametros* mediante la función pública *Set_Value.*

Por otra parte, la tarea *MQTT_Task* del paquete *Controlador.TelemetryMQTT* realiza la lectura de la variable *Last_Senhal_Entrada* del objeto protegido *Voltage_Value_PO* del paquete *Controlador.Parametros* mediante la función pública *Get_Value.*

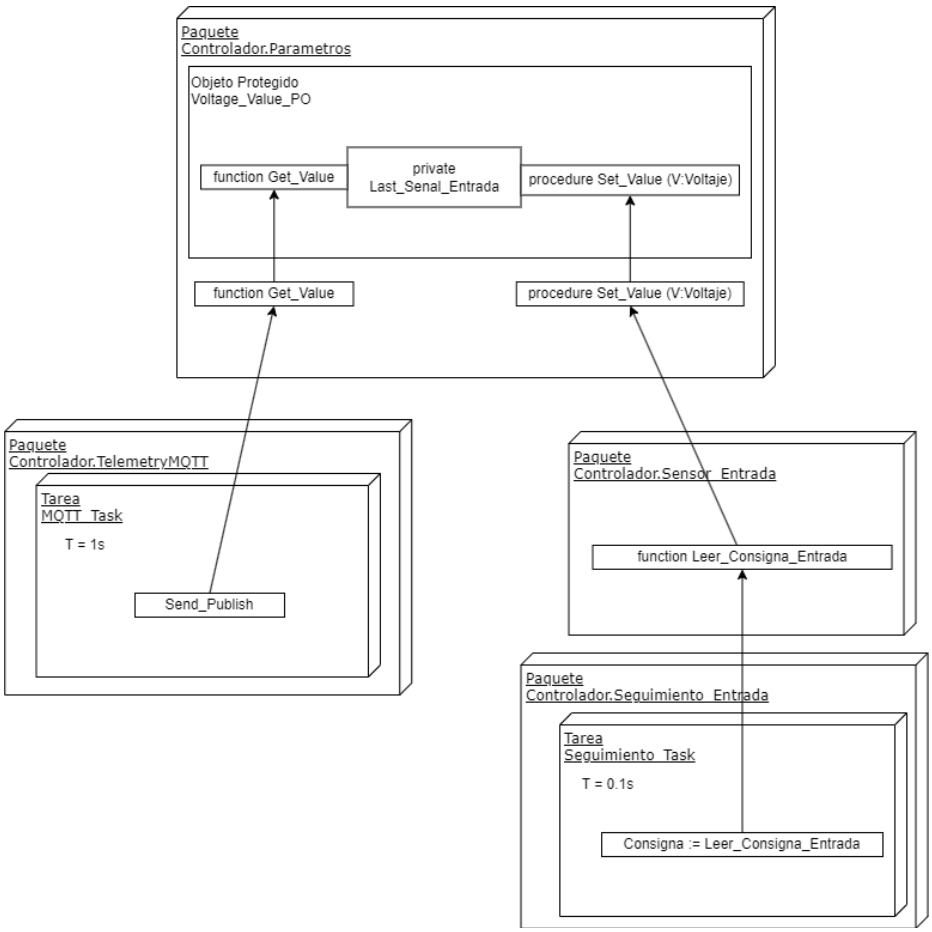


ILUSTRACIÓN 12 DIAGRAMA DE BLOQUES DE LA INTERACCIÓN TAREA SEGUIMIENTO ENTRADA Y TAREA TELEMETRYMQTT

4.4 Capa de aplicación

En este proyecto, la capa de aplicación se corresponde con la plataforma ThingsBoard. En esta plataforma emplearemos la telemetría enviada por el microcontrolador STM32 para mostrarla gráficamente en un panel de información en tiempo real y, además, será almacenada en una base de datos PostgreSQL para poder acceder a los datos en un futuro, aportando así además una capa de persistencia.

En la siguiente Ilustración 13 podemos ver la integración entre la capa de aplicación en color rojo y las capas de percepción y red presentadas anteriormente:

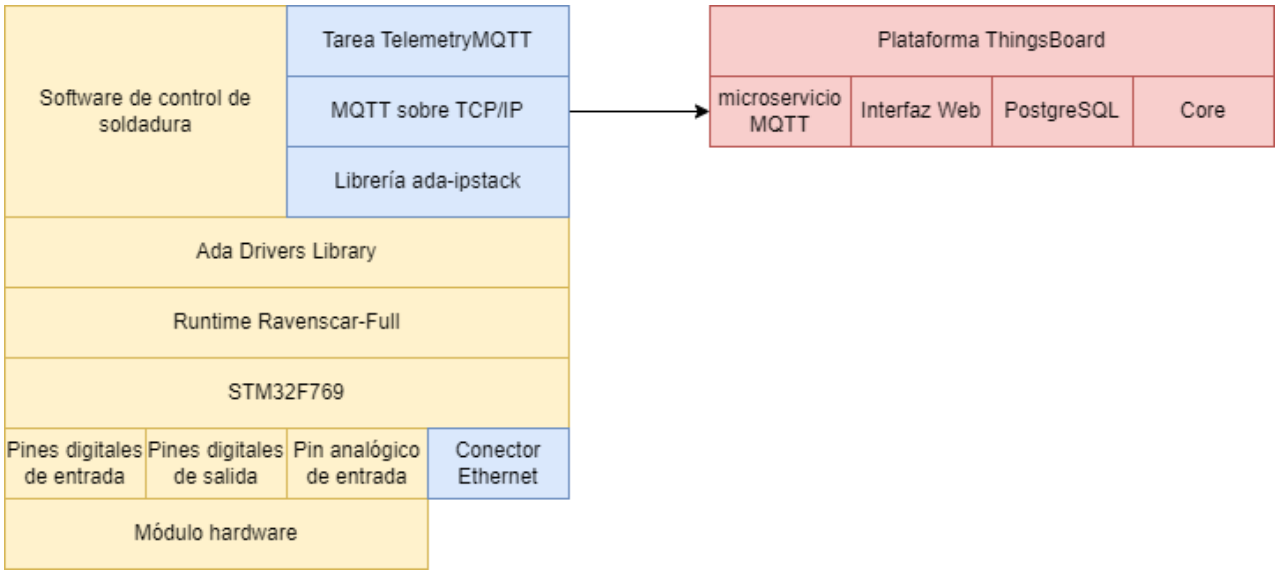


ILUSTRACIÓN 13 DIAGRAMA DE CAPAS DE LA SOLUCIÓN GLOBAL

4.4.1 Plataforma ThingsBoard

Esta plataforma está diseñada para ser escalable, tolerante a fallos, robusta, eficiente, persistente y personalizable, pudiendo ser desplegada en un único nodo o en varios, de manera local o en cloud, con varias opciones en la capa de persistencia y a nivel de comunicación con dispositivos externos.

En la Ilustración 14 podemos ver el diagrama de componentes de Thingsboard. Analizamos a continuación cuales hemos empleado para este desarrollo.

Podemos ver que los dispositivos se conectan con los microservicios de transporte, donde hemos empleado el de MQTT. Este microservicio actúa como broker de las comunicaciones, y se conecta con el *core* de la plataforma mediante Apache Kafka. Apache Kafka se trata de una plataforma de comunicación mediante cola de mensajes persistente distribuida, escalable y fiable.

El *core* de ThingsBoard luego almacena la información en la capa de persistencia (base de datos) y emplea la interfaz de usuario (ThingsBoard UI) para interactuar con el usuario final.

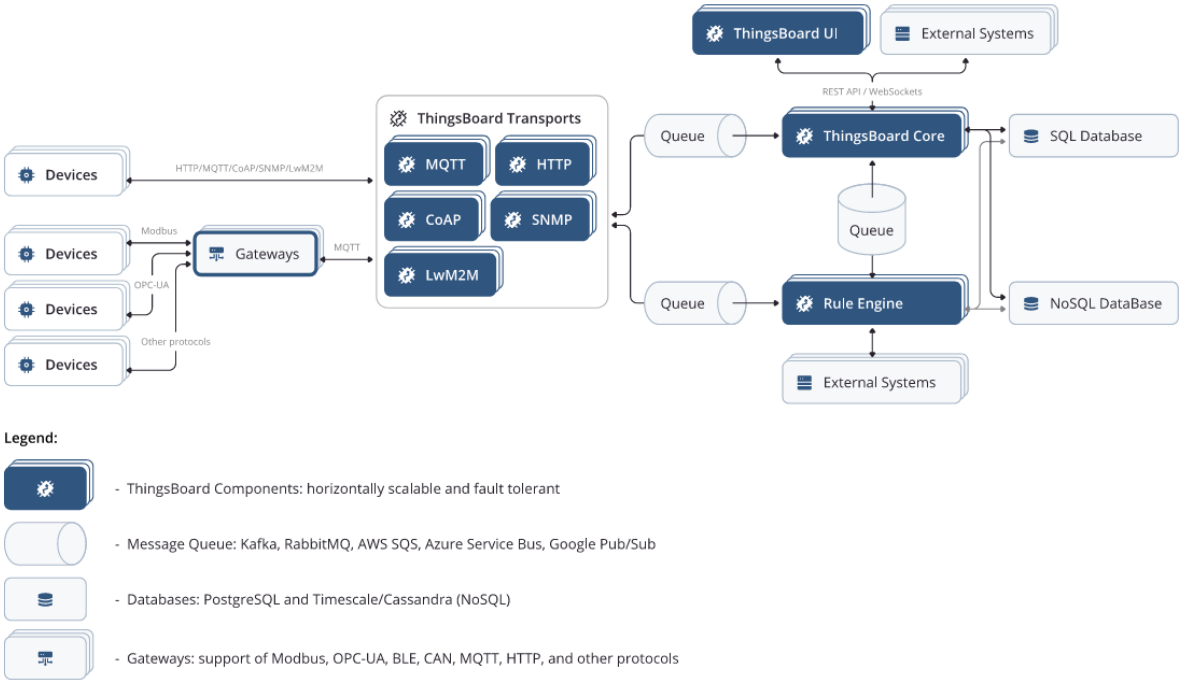


ILUSTRACIÓN 14 DIAGRAMA DE COMPONENTES DE LA PLATAFORMA THINGSBOARD [31]

Se implementa esta plataforma de manera local en el PC en su versión *Community* (Open Source), debido a su coste gratuito, así como el hecho de simplificar el despliegue de la plataforma al hacerlo de manera local frente a hacerlo en cloud. Esta plataforma se emplea para poder registrar la telemetría del microcontrolador, así como poder monitorizar y visualizar a modo de sistema SCADA el estado del sistema con esta telemetría.

Para ello, se han de crear sobre la plataforma los dispositivos que se conectarán a la misma. Para este proyecto solamente se ha creado un dispositivo, al que hemos denominado *STM32F769i*, dado que se corresponde con el microcontrolador que controla el sistema de soldadura industrial.

Se crea también un panel donde se mostrará gráficamente la información recibida o almacenada del microcontrolador, el cual enviará los valores de voltaje y velocidad angular en un mensaje MQTT, con los datos en formato JSON

A continuación, explicaremos cómo se han de enviar los mensajes a la plataforma ThingsBoard, el formato de su contenido y cómo está diseñada la capa de persistencia

4.4.1.1 Autenticación del dispositivo y tópico del mensaje

Como se explicó con anterioridad en el apartado 2.6, el protocolo MQTT está basado en el paradigma publicador/suscriptor con un broker como elemento central y árbitro de las comunicaciones. Un publicador envía un mensaje con un tópico asociado al broker, y este lo redistribuye a todos los suscriptores suscritos al tópico del mensaje.

En la plataforma ThingsBoard se han de crear los dispositivos que se conectarán a la plataforma, y estos se han de autenticar en la fase de establecimiento de conexión a fin de que la plataforma sea conocedora de que dispositivo es el que hay conectado en cada conexión. Posteriormente, en el envío de mensajes, se ha de especificar el tópico *v1/devices/telemetry*, este tópico es único para todos los dispositivos, y es mediante la autenticación que la plataforma conoce que dispositivo es el que está conectado

En este desarrollo se ha optado por la autenticación mediante tokens de acceso [32], que son el método de autenticación por defecto de la plataforma. Estos tokens los genera y asocia al dispositivo la plataforma ThingsBoard en el momento de la creación del dispositivo y son empleados por este en la fase de establecimiento de conexión, enviándolos en el campo usuario del mensaje MQTT. El establecimiento de la conexión se realiza en la tarea *MQTT_Task* del paquete *Contolador.TelemetryMQTT*.

Además, todos los dispositivos que envían información a la plataforma ThingsBoard lo hacen con el tópico *v1/devices/me/telemetry*

4.4.1.2 Formato mensaje JSON

Para el envío de información a la plataforma Thingsboard se pueden emplear valores booleanos, de coma flotante, enteros o estructuras más complejas en formato JSON.

En este desarrollo empleamos el formato JSON, debido a que enviaremos dos valores, voltaje de entrada y velocidad angular de los motores, y potencialmente en el futuro, simplificaría el aumento de información a enviar

La estructura del mensaje JSON se define directamente a la hora de enviar el mensaje mediante la función *Send_Publish*, del paquete *MQTT* de la librería *ada-ipstack*. El campo *Message*, para esta aplicación, la definición del siguiente Listado 4:

Listado 4. Definición del mensaje a enviar

```
"{Velocidad: " & Voltios_A_Angulo(Controlador.Parametros.Get_Value,
Eje_Oscilacion)'Image & "," & "Voltaje: " & Controlador.Parametros.Get_Value'Image &
"}
```

Lo que daría como resultado:

"{Velocidad: <*velocidad anglar*>,Voltaje: <*voltaje de entrada*>}"

4.4.1.3 Capa de Persistencia

Es necesario emplear una base de datos para proporcionar una capa de persistencia a la plataforma ThingsBoard donde almacenar tanto los valores de configuración de la plataforma como los valores de telemetría recibidos.

ThingsBoard soporta y recomienda el uso de la base de datos SQL PostrgreSQL como base de datos única, tanto para de entidades (configuración, perfiles, paneles, usuarios, dispositivos…) como para telemetría (datos), y es esta la opción elegida en este proyecto debido a estar soportada y recomendada por la propia plataforma, así como ser más sencillo de implantar y cubrir las necesidades de almacenar la telemetría de un único dispositivo.

El uso de una base de datos NoSQL ya no está soportado de manera oficial, a excepción de Cassandra, aunque debido a las múltiples limitaciones de este tipo de base de datos de poder hacer transacciones y uniones requeridas para habilitar la búsqueda avanzada en entidades de ThingsBoard, su rendimiento no era óptimo, y su complejidad de integración era mucho más elevada que con una base de datos relacional, es por ello que ha caído en desuso su uso como base de datos única, aunque existe la posibilidad de emplearla en una arquitectura híbrida: PostgreSQL como base de datos para entidades y Cassandra o TimescaleDB como base de datos para telemetría [33]. Este tipo de arquitectura híbrida la encontramos más indicada en entornos productivos, con un gran flujo de datos de telemetría y elevado número de consultas sobre los datos y así poder tratar la capa de telemetría de manera independiente a la capa de entidades.

5. Validación

En cada etapa del proyecto, tras su desarrollo, se ha ido validando el trabajo realizado, antes de pasar a la fase siguiente. Esto se ha hecho así para poder cambiar de fase con una base sólida y poder descartar que los problemas encontrados en una fase sean fruto de desarrollos anteriores.

5.1 Validación del módulo de Oscilación

Una vez realizada la integración del módulo hardware, se valida el correcto funcionamiento de los motores, que su movimiento sea correcto de acuerdo con el voltaje de entrada, y el número de pasos reportado en pantalla sea correcto, así pues, dado que la velocidad máxima de rotación es de +- 15º/s, se valida que una rotación en cualquier sentido a máxima velocidad (0V o 3'3V) tarda 24 segundos. También se comprueba que, al volver el motor a su posición de origen, el número de pasos mostrados en pantalla es 0, así como que cuando el voltaje de entrada es 1'65V los motores permanecen estáticos.

5.2 Validación del módulo de comunicaciones

Tras realizar la integración de la librería *ada-ipstack*, junto con los cambios de ADL, se crea un programa de prueba que envía un mensaje mediante MQTT con el tópico "*TFMControlSoldadura/test*" contra el PC de desarrollo, el cual ejecuta el bróker mosquitto y también tiene una instancia de un cliente suscrito al tópico "*TFMControlSoldadura/test*"

En la Ilustración 15 podemos ver la información reportada por el broker, donde podemos ver al inicio el establecimiento de conexión (*New connection/CONNACK*) de un cliente con IP 192.168.99.35, que se trata de una instancia de un cliente mosquitto que se suscribe (*SUBSCRIBE*) al tópico *TFMControlSoldadura/test*

Posteriormente se conecta (*New connection/CONNACK*) un dispositivo con IP 192.168.99.100, que se corresponde con el microcontrolador STM32, que envía un mensaje (*PUBLISH*) con el tópico *TFMControlSoldadura/test*. Cuando se recibe este mensaje, el broker se lo reenvía (*PUBLISH*) al cliente mosquitto de la máquina local.

Finalmente, el microcontrolador cierra la conexión (*DISCONNECT*)

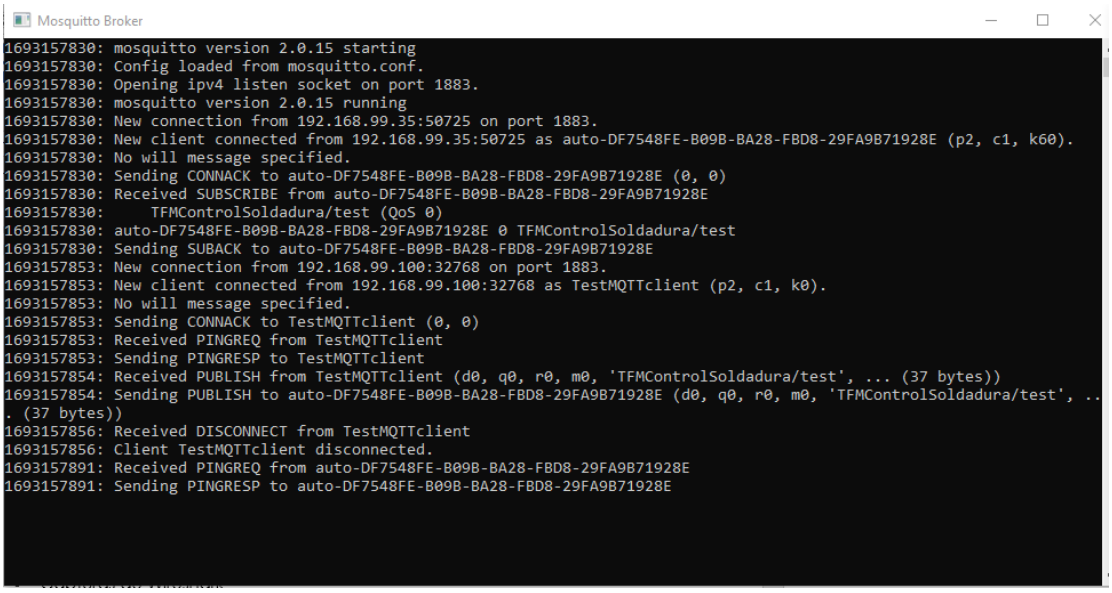


ILUSTRACIÓN 15 DETALLE DE EVENTOS EN EL BROKER MQTT

En la Ilustración 16 podemos ver el mensaje recibido en el cliente mosquitto que se ejecuta en la máquina local.

ILUSTRACIÓN 16 CLIENTE MQTT QUE MUESTRA LA RECEPCIÓN CORRECTA DE UN MENSAJE ENVIADO DESDE EL MICROCONTROLADOR

También empleamos WireShark para ver en detalle el contenido de los paquetes de red, con el tópico y el mensaje en texto plano, tal como podemos ver en la Ilustración 17

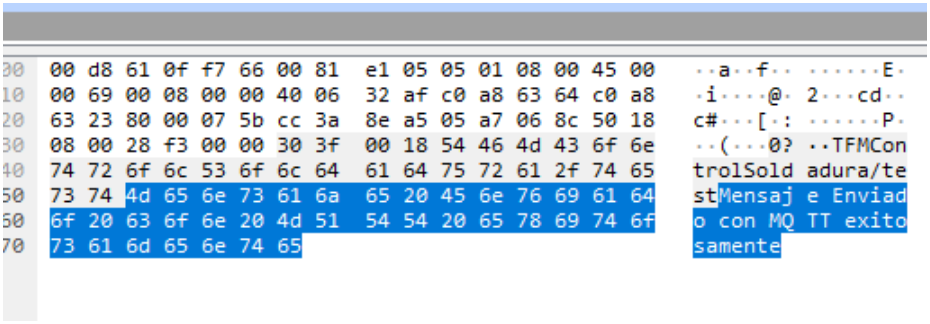
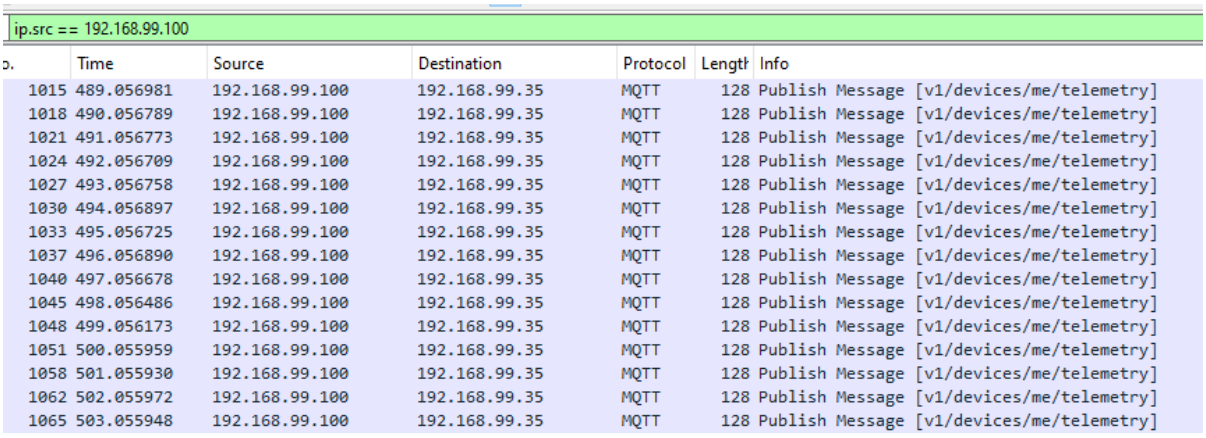


ILUSTRACIÓN 17 CAPTURAS WIRESHARK DE MENSAJES DE VALIDACIÓN DE MQTT

Una vez validada la capacidad de enviar mensajes MQTT, se desarrolla la tarea periódica, la cual, tras inicializar el stack TCP/IP y establecer una conexión MQTT con la plataforma ThingsBoard (empleo del token de acceso en el campo usuario), envía un mensaje MQTT con el valor de voltaje de entrada y su conversión a grados por segundo cada segundo

En las siguientes dos ilustraciones: Ilustración 18 e Ilustración 19, podemos ver, por una parte, la periodicidad de un segundo en el envío de cada mensaje MQTT con el tópico *v1/devices/me/telemetry*



ɔ.	Time	Source	Destination	Protocol	Length	Info
1015	489.056981	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1018	490.056789	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1021	491.056773	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1024	492.056709	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1027	493.056758	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1030	494.056897	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1033	495.056725	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1037	496.056890	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1040	497.056678	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1045	498.056486	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1048	499.056173	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1051	500.055959	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1058	501.055930	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1062	502.055972	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]
1065	503.055948	192.168.99.100	192.168.99.35	MQTT	128	Publish Message [v1/devices/me/telemetry]

ILUSTRACIÓN 18 FLUJO DE MENSAJES MQTT CON UNA PERIODICIDAD DE UN SEGUNDO

Y también el contenido de los paquetes de red con el contenido del mensaje MQTT, cuyo mensaje se adapta al formato JSON, tal y como especificamos en el apartado 4.4.1.2

```
08 00 3c 73 ab 75 30 48  00 17 76 31 2f 64 65 76   ··<s·u0H ··v1/dev
69 63 65 73 2f 6d 65 2f  74 65 6c 65 6d 65 74 72   ices/me/ telemetr
79 7b 56 65 6c 6f 63 69  64 61 64 3a 20 2d 31 2e   y{Veloci dad: -1.
35 30 30 30 30 45 2b 30  31 2c 56 6f 6c 74 61 6a   50000E+0 1,Voltaj
65 3a 20 20 30 2e 30 30  30 30 30 45 2b 30 30 7d   e:  0.00 000E+00}
```

ILUSTRACIÓN 19 CAPTURA WIRESHARK DE TELEMETRÍA MEDIANTE MQTT

Una vez validado el envío periódico de los mensjaes MQTT contra la plataforma ThingsBoard, desarrollamos un panel de visualización sobre el que se muestran ambos valores reportados en el momento sobre una gráfica, tal y como se muestra en la Ilustración 20, donde podemos observar una línea de color verde haciendo referencia al voltaje de entrada, con su eje de ordenadas en el extremo izquierdo, representando valores de entre -1V y 4V, aunque el sistema solo es capaz de trabajar entre 0V y 3’3V

En color turquesa se muestra la gráfica haciendo referencia a la velocidad angular, con el eje de ordenadas en la derecha, con valores de entre -20º/s y 20º/s, aunque las velocidades del sistema varían entre -15º/s y 15º/s.

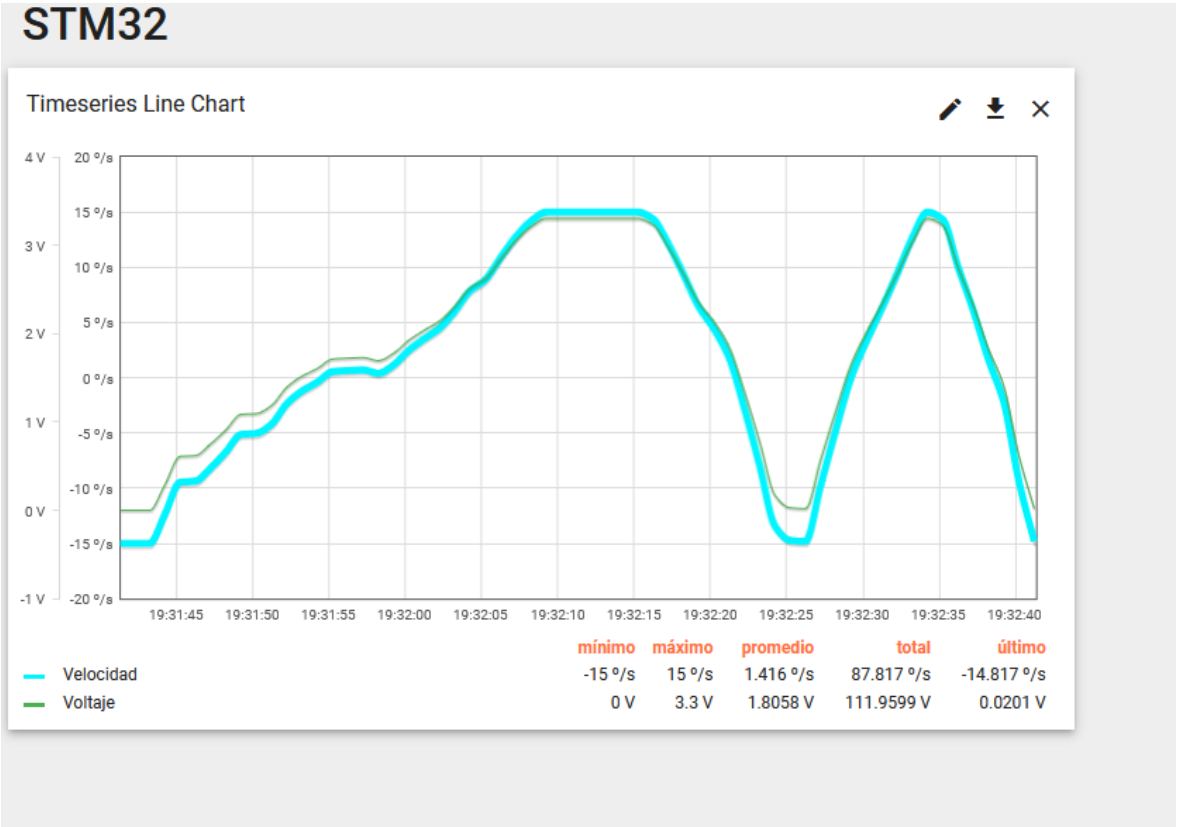


ILUSTRACIÓN 20 GRÁFICA MOSTRANDO VALORES DE ENTRADA (V) Y SALIDA (º/S)

Podemos observar, tal y como se diseñó, que para voltajes de entrada de 0V, la velocidad del sistema es de -15º/s, para voltajes de 1’65V, la velocidad es de 0º/s y para voltajes de 3’3V, la velocidad es de 15º/s

6. Conclusiones y trabajo futuro

En este proyecto se han logrado los objetivos propuestos inicialmente en cada fase.

Por una parte, se ha logrado integrar el software de control de soldadura con hardware real análogo a un sistema de soldadura industrial, realizando el movimiento de los motores correcto y de acorde a la especificación que definimos en base al voltaje de entrada, aumentando el rango de valores y con ello, la precisión de los movimientos frente al proyecto del que partíamos.

Por otra parte, se ha logrado dotar de comunicaciones ethernet mediante el empleo de la librería *ada-ipstack* a la plataforma de soldadura, la cual originalmente operaba de manera aislada y autónoma. Esta librería proporciona las herramientas necesarias para poder establecer comunicaciones mediante el protocolo MQTT con otros sistemas, pudiendo así explotar las múltiples posibilidades que ello permite, como son la adquisición de datos del estado del sistema en tiempo real para diferentes cometidos propios de sistemas IoT como son la monitorización en tiempo real de manera remota, mantenimiento predictivo, trazabilidad o análisis de datos entre otros. Esto queda demostrado mediante el desarrollo de un panel informativo que muestra en tiempo real el estado del sistema y el almacenaje persistente del mismo.

Por último, se ha demostrado el potencial de implementar estas comunicaciones implementando de manera local la plataforma ThingsBoard, en la cual hemos desarrollado un sistema SCADA mínimo, en el cual se muestra en tiempo real los valores de entrada del sistema.

6.1 Trabajo Futuro

Este proyecto se ha desarrollado como una prueba de concepto sobre la viabilidad y posibilidad de implantación de comunicaciones MQTT en un microcontrolador ARM de bajos recursos en lenguaje Ada. Todas las comunicaciones eran salientes del microcontrolador, y se han empleado para mostrar en un panel gráficamente información sobre el estado del sistema. También existen múltiples funcionalidades y capacidades de la plataforma ThingsBoard que no se han explotado, o la opción de desplegar ThingsBoard en Cloud. Todo da pie a poder desarrollar nuevos proyectos muy interesantes en el futuro:

- Establecer comunicaciones de entrada al microcontrolador, pudiendo enviar comandos, alterando su modo de trabajo y/o sustituyendo así la entrada de señales analógicas.

- Enviar otra información relevante de la instalación, como señales de alarma y generar un sistema que alerte a un usuario final mediante alarmas.

- Emplear la información almacenada en la base de datos para aplicar técnicas de mantenimiento predictivo, por ejemplo, capturar y analizar la intensidad del arco de soldadura para determinar el nivel de desgaste de la antorcha

- Desplegar ThingsBoard sobre algún servicio Cloud como Amazon Web Service y poder analizar diferencias de rendimiento entre una implantación on-premise y cloud.

- Realizar un análisis de rendimiento y escalabilidad entre dos despliegues de ThingsBoard, uno con una base de datos única PostgreSQL y otra con una arquitectura híbrida en la capa de persistencia.

7. Bibliografía

[1] A. Zulueta Barbadillo, Controlador de un subsitema de control de una antorcha de soldadura con un procesador de bajo coste, 2020.

[2] Amazon Web Services, [En línea]. Available: https://aws.amazon.com/es/what-is/iot/. [Último acceso: 28 Mayo 2024].

[3] Oracle, «Que es un ERP,» [En línea]. Available: https://www.oracle.com/es/erp/what-is-erp/. [Último acceso: 3 Julio 2024].

[4] IBM, «Industry 4.0,» [En línea]. Available: https://www.ibm.com/es-es/topics/industry-4-0. [Último acceso: 28 Mayo 2024].

[5] OASIS, «MQTT v5.0,» [En línea]. Available: https://docs.oasis-open.org/mqtt/mqtt/v5.0/mqtt-v5.0.html. [Último acceso: 20 Junio 2024].

[6] Universidad Politecnica de Madrid, «Sistemas de Tiempo Real,» [En línea]. Available: http://www.dit.upm.es/~jpuente/gstr/str.htm. [Último acceso: 24 Agosto 2023].

[7] ST Mictroelectronics, «Procesador STM32F769NIH6,» [En línea]. Available: https://www.st.com/en/microcontrollers-microprocessors/stm32f769ni.html. [Último acceso: 3 Junio 2024].

[8] Universidad de Michigan, «The Ada Programming Languaje,» [En línea]. Available: http://groups.umd.umich.edu/cis/course.des/cis400/ada/ada.html. [Último acceso: 30 Marzo 2024].

[9] Amazon Web Services, [En línea]. Available: https://aws.amazon.com/es/iot-core/. [Último acceso: 28 Marzo 2024].

[10] Google, «Google Cloud IoT Core,» [En línea]. Available: https://cloud.google.com/iot-core. [Último acceso: 28 Marzo 2024].

[11] Microsoft, «Azure IoT Hub,» [En línea]. Available: https://azure.microsoft.com/es-es/products/iot-hub. [Último acceso: 28 Marzo 2024].

[12] ThingsBoard, «Main Web Page,» [En línea]. Available: https://thingsboard.io/. [Último acceso: 17 Abril 2024].

[13] ST Microelectronics, «32f769idiscovery,» [En línea]. Available: https://www.st.com/en/evaluation-tools/32f769idiscovery.html. [Último acceso: 16 Abril 2023].

[14] Ada Information Clearinghouse, «Perfil Ravenscar,» [En línea]. Available: https://www.adaic.org/resources/add_content/standards/05rat/html/Rat-5-4.html. [Último acceso: 3 Junio 2024].

[15] AdaCore, «Ravenscar Profile,» [En línea]. Available: https://docs.adacore.com/spark2014-docs/html/ug/en/source/concurrency.html. [Último acceso: 23 Agosto 2023].

[16] AdaCore, «Github - Ada Drivers Library 2020,» [En línea]. Available: https://github.com/AdaCore/Ada_Drivers_Library. [Último acceso: 25 Abril 2024].

[17] M. Iglesias Abbatemarco, «Github,» [En línea]. Available: https://github.com/mhanuel26/ada-ipstack/tree/master. [Último acceso: 18 Abril 2024].

[18] AdaCore, «Timeline of Ada,» [En línea]. Available: https://www.adacore.com/about-ada/timeline-of-ada. [Último acceso: 28 Marzo 2024].

[19] MQTT, [En línea]. Available: https://mqtt.org/. [Último acceso: 15 Agosto 2023].

[20] Eclipse Mosquitto, «Mosquitto,» [En línea]. Available: https://mosquitto.org/. [Último acceso: 30 Marzo 2024].

[21] The PostgreSQL Global Development Group, «About PostgreSQL,» [En línea]. Available: https://www.postgresql.org/about/. [Último acceso: 3 Julio 2024].

[22] Microsoft, «Compilación Cruzada,» [En línea]. Available: https://learn.microsoft.com/es-es/dotnet/core/deploying/native-aot/cross-compile. [Último acceso: 29 Marzo 2024].

[23] ST Microelectronics, «ST Link,» [En línea]. Available: https://www.st.com/en/development-tools/stsw-link004.html. [Último acceso: 30 Marzo 2024].

[24] AdaCore, «StLink,» [En línea]. Available: https://docs.adacore.com/gnat_ugx-docs/html/gnat_ugx/gnat_ugx/arm-elf_topics_and_tutorial.html. [Último acceso: 30 Marzo 2024].

[25] D. Ortega Arranz, «GitHub,» [En línea]. Available: https://github.com/dao703/Robot_Car_UC/tree/master/Drivers_UC. [Último acceso: 25 Abril 2024].

[26] D. Ortega Arranz, Marco para el desarrollo de aplicaciones Ada sobre microcontroladores STM32, 2019.

[27] Applied Motion Products, [En línea]. Available: https://appliedmotion.s3.amazonaws.com/STR2-Hardware-Manual-920-0059C.pdf. [Último acceso: 18 Abril 2023].

[28] T. Yousuf, R. Mahmoud, F. Aloul y I. Zualkernan, «Internet of Things (IoT) Security: Current Status, Challenges and Countermeasures,» vol. 5, pp. 608-616, Diciembre 2015.

[29] M. Hassan y J. Li, «Demonstration of cyber security through Penetration testing on IP camera,» Febrero 2023.

[30] AdaCore, «Github - Ada Drivers Library 2017,» [En línea]. Available: https://github.com/AdaCore/Ada_Drivers_Library/tree/3251327a7e72d50699a733a3b110d60b704cd759. [Último acceso: 13 Febrero 2024].

[31] ThingsBoard, «Diagrama arquitectura,» [En línea]. Available: https://thingsboard.io/docs/reference/. [Último acceso: 3 Julio 2024].

[32] ThingsBoard, «Access Token,» [En línea]. Available: https://thingsboard.io/docs/user-guide/access-token/. [Último acceso: 24 Junio 2024].

[33] ThingsBoard, «SQL vs NoSQL vs Hybrid Database Approach,» [En línea]. Available: https://thingsboard.io/docs/reference/#sql-vs-nosql-vs-hybrid-database-approach. [Último acceso: 28 Marzo 2024].