UNIVERSIDAD NACIONAL DE LA PLATA

FACULTAD DE INFORMÁTICA

TESINA DE LICENCIATURA

TÍTULO: Integración continua en proyectos con microcontroladores

AUTORES: Alejo Alfredo Santi

DIRECTOR/A: Tinetti Fernando G.

CODIRECTOR/A: -

ASESOR/A PROFESIONAL: -

CARRERA: Licenciatura en Informática

Resumen

En los últimos años, la industria de la tecnología ha crecido de manera exponencial, generando aplicaciones donde antes no existían. Algunos ejemplos pueden ser el Internet de las Cosas (IoT), que conecta a Internet dispositivos de uso cotidiano, las flotas de vehículos conectadas, que pueden administrarse desde un sistema central, o ambientes como la industria 4.0, en donde se puede monitorizar la maquinaria y controlar procesos productivos.

Como soporte de muchas de estas aplicaciones, se encuentran los microcontroladores, un sistema de hardware con tamaño y prestaciones reducidas y específicas. Con el incremento en su utilización, la demanda de desarrollo sobre microcontroladores ha aumentado considerablemente y, proporcionalmente, también la capacidad de los microcontroladores. Pero este ambiente no goza de todas las facilidades que posee el desarrollo tradicional de software, y tiene un proceso más laborioso (menos automatizado) en muchos aspectos.

Este trabajo busca integrar una metodología llamada CI/CD, cada vez más popular en el desarrollo tradicional, al desarrollo con microcontroladores. Esta práctica, de manera resumida, consiste en automatizar diferentes partes del ciclo de software. De esta manera, se logra incrementar la productividad y disminuir la posibilidad de generar errores, automatizando tareas tediosas que suelen malgastar el tiempo del programador.

Palabras Clave

-Microcontroladores
-ESP32
-Sistemas distribuidos
-CI/CD
-Arduino
-OTA
-Sistemas en tiempo real

Conclusiones

Las prácticas de CI/CD, ampliamente utilizadas en el desarrollo tradicional, se pueden utilizar también en el desarrollo sobre microcontroladores, un contexto que con el incremento en su demanda necesita mejoras.

Como se puede ver en el proyecto, con ayuda de la actualización Over The Air (OTA), se logra tener un sistema que automatiza muchas partes del desarrollo, ahorrando mucho tiempo y esfuerzo al programador, que al subir su código al repositorio lo puede ver en producción en cuestión de minutos.

Trabajos Realizados

Se realizó un sistema capaz de automatizar varias etapas del desarrollo. Para ello, un servidor se encarga de interactuar con diferentes agentes. Cuando el desarrollador sube su código al controlador de versiones elegido, un webhook notifica al servidor, quien extrae el código, genera una compilación, la sube a un microcontrolador de testeo, testea sobre él, y cuando se pasan todos los controles se encarga de subir la compilación, de manera remota, al microcontrolador que esté en producción.

Trabajos Futuros

El trabajo futuro más importante es agregarle una capa de seguridad al sistema, discriminando quienes pueden realizar actualizaciones sobre el microcontrolador y encriptando las mismas.

También es recomendable construir una versión que utilice GitLab, ya que la actual hace uso de GitHub como controlador de versiones. Teniendo una versión de GitLab, se estarían cubriendo los 2 controladores de versiones más populares del mercado.

Otra propuesta es prescindir de la librería utilizada para llevar a cabo la actualización OTA, dando como resultado un desarrollo más independiente.

Fecha de la presentación: **Noviembre 2022**

Universidad Nacional de La Plata
Facultad de Informática

Tesina de Licenciatura en Informática

"Integración continua en proyectos con microcontroladores"

Autor: Alejo Alfredo Santi
Director: Tinetti Fernando G.

Índice

Resumen

La industria de la tecnología relacionada con el uso de microcontroladores (MCU) ha crecido y está creciendo a pasos agigantados. En este contexto, han evolucionado tanto los dispositivos electrónicos/hardware utilizados como el software desarrollado para los mismos, que cada vez es más complejo. En cierto modo, en los últimos años, los microcontroladores han dejado de ser utilizados casi exclusivamente como dispositivos electrónicos para, con el software apropiado, ser elementos de procesamiento muy similares a un sistema de cómputo (aunque con limitaciones y detalles específicos de uso).

Es común la utilización de microcontroladores en múltiples aplicaciones, en diferentes lugares de nuestra casa, con el Internet de las Cosas o Internet of Things (IoT), en los vehículos que utilizamos o en los procesos productivos de diferentes industrias.

Como es de esperar, el crecimiento en el uso de microcontroladores conlleva una creciente demanda de software en los mismos, y con ello los problemas asociados al desarrollo de software, algunos conocidos y otros propios de esta área de desarrollo particular. El proceso de desarrollo de software sobre microcontroladores no está tan avanzado como el desarrollo de software tradicional, el cual tiene IDEs excepcionales y diferentes herramientas que le facilitan el trabajo al programador. El desarrollo sobre microcontroladores, tiene IDEs básicos, específicos de cada plataforma, y no necesariamente tiene formas sencillas o asistidas de testear, además de que muchas veces falta documentación de lo que se está utilizando, por nombrar algunos de los problemas.

Este proyecto busca agilizar el proceso de desarrollo, utilizando una práctica que se popularizó en los últimos años, Continuous Integration/Continuous Delivery or Deployment (en español Integración Continua/Entrega o Despliegue Continuo), abreviado como CI/CD. Esta práctica plantea, de manera resumida, automatizar diferentes etapas del proceso de desarrollo para ahorrar tiempo, como la creación de una build, el testeo y su posterior puesta en producción. De esta manera, se genera un sistema que trabaja de manera fluida y continua, liberando parcialmente al desarrollador de tareas que pueden ser automatizadas.

Con esa meta, se diseñó un pequeño servidor que va a interactuar con diferentes agentes, creando un sistema capaz de automatizar, al menos, una parte del proceso de desarrollo de software, generando grandes ventajas para el programador.

Capítulo 1 - Introducción

En este capítulo se van a explicar las motivaciones y objetivos de la tesis. Se explicará por qué es tan importante y cada vez más demandado el desarrollo con microcontroladores, señalando las complicaciones que este contexto de desarrollo tiene aparejado. Como parte de los objetivos, se nombrará una metodología en auge, CI/CD, que puede ayudar a combatir estas desventajas y agilizar el proceso productivo.

1.1. Motivación

En los últimos años, el desarrollo con microcontroladores, sobre todo en sistemas distribuidos, se está popularizando. Ya sea por la creciente industria del IoT [1], en donde los objetos de uso cotidiano ahora se pueden conectar a internet y procesar información [2], por el crecimiento de la industria 4.0, la cual precisa ayuda de estos dispositivos electrónicos [3], por su incorporación cada vez más regular en automóviles, por su utilidad para controlar flotas de vehículos o por muchas otras aplicaciones [4], se requiere cada vez más desarrollo sobre estos dispositivos.

En comparación con el desarrollo tradicional, el desarrollo sobre microcontroladores presenta grandes desventajas. Para comenzar, el ciclo de producción de software es mucho más laborioso. Supongamos un ciclo genérico, en donde el programador, ya sea para agregar características o arreglar errores, desea incorporar un nuevo código al ambiente productivo, es decir, generar una actualización que llegue al dispositivo donde está corriendo el software.

Primero debe compilar el código que escribió, un proceso relativamente tardío, que corresponde a hacer una build o versión en el desarrollo tradicional, y para hacerlo tiene que interactuar físicamente con el dispositivo, presionando un botón del mismo en un momento preciso. El siguiente paso sería probar (o testear en inglés) esa build, lo cual en el contexto de los microcontroladores no es nada fácil. A día de hoy, se disponen de pocos simuladores, normalmente propietarios y con uso bajo licencia y/o con costo. Detalles importantes como los de tiempo real, por ejemplo, han llevado a que la opción más viable sea testear sobre un dispositivo real, lo que obliga al desarrollador a poseer un microcontrolador similar al utilizado en producción, cargar la nueva build al dispositivo, y ahí empezar a correr las pruebas que desee. Una vez las pruebas son exitosas, se debe subir la nueva versión al dispositivo en producción que, como nombramos, puede ser parte de una flota de vehículos, por lo que reunir los dispositivos en un punto central resulta altamente costoso, de modo que se debe buscar una alternativa inalámbrica para poder enviar esa actualización.

Con lo explicado, queda en evidencia que el proceso de crear una build, testearla y subirla a producción, en el desarrollo sobre microcontroladores, es notoriamente más laborioso y tardío que en el desarrollo tradicional. Estos requerimientos hacen más complejo y costoso todo el proceso.

1.2. Objetivos

El objetivo de este proyecto es combinar el desarrollo de software sobre microcontroladores con una práctica relativamente nueva, pero que se está utilizando cada vez más en el desarrollo tradicional [5], que es el CI/CD. Esta práctica consiste en automatizar diferentes partes del desarrollo, para facilitar el ciclo o evolución del software [6].

Más específicamente, la propuesta de este trabajo busca generar un sistema que agilice el desarrollo y puesta en producción del software de/en microcontroladores, asistiendo al desarrollador o a equipos de desarrollo en las tareas hoy consideradas imprescindibles en el ciclo de software [7]:

1. Integración de repositorios en el proceso de desarrollo con microcontroladores (subir el código de una actualización a un repositorio de código).
2. Automatización de la construcción del binario del proyecto (la “build” del proyecto).
3. Ejecución automatizada de pruebas de software (“testing”) con microcontroladores.
4. Automatización de la puesta en marcha/ejecución de una nueva versión de software (“deploy”) de microcontroladores.

Se debe tener en cuenta que esta propuesta está enfocada solo en el software, su desarrollo y evolución en sistemas basados en microcontroladores. En este contexto de desarrollo, es usual combinar partes de hardware (sensores y actuadores, básicamente) con software para construir el sistema completo. Originalmente, estos sistemas se consideraron muy específicos y, en cierto modo, más asociados a proyectos de electrónica que a proyectos de software. Expresado de otra manera: el software no necesariamente era considerado de tanta importancia como el hardware mismo y su funcionamiento, los microcontroladores estaban dedicados a implementar máquinas de estado para organizar o controlar el resto de hardware utilizado. A medida que estos fueron crecieron en capacidad, e integración de periféricos (ejemplo: interfaces WiFi), el software claramente se volvió más complejo. Entre las nuevas posibilidades que se incorporan en los microcontroladores se puede identificar OTA, y es lo que a su vez se propone aprovechar para asistir a los equipos de desarrollo. Claramente, los cambios en el uso de hardware no se pueden automatizar: si se incorpora un sensor o actuador al sistema, necesariamente se debe acceder o cambiar físicamente el mismo sin posibilidad de incorporar herramientas de software (asociadas a procesos de CI/CD).

El resto de este documento está organizado de la siguiente manera:

- El capítulo 2 contiene el marco teórico orientado a microcontroladores, en donde se va a explicar qué son los microcontroladores, qué tipos existen, cuál se va a utilizar en este proyecto y qué son las actualizaciones OTA, entre otras cosas.
- El capítulo 3 corresponde al marco teórico de CI/CD en sistemas distribuidos, en el cual se explicarán varios conceptos que, si bien son independientes, están interrelacionados. Se hará una explicación de qué significa CI/CD y qué ventajas tiene su implementación, y también se explicarán los diferentes tipos de sistemas que atraviesan esta tesis, los distribuidos, los de tiempo real y los de control de versiones.

- El capítulo 4 contiene la propuesta, se plantea un sistema capaz de automatizar las diferentes partes del ciclo de software, con gráficos que detallan los agentes involucrados y clasifican las tareas a realizar según corresponden a la parte de CI o a la de CD.
- El capítulo 5 contiene la implementación, se detallarán las tecnologías con las que se va a llevar a cabo el sistema planteado en el punto anterior, con la correspondiente justificación de su uso y nombrando algunas alternativas.
- El capítulo 6 contiene los resultados obtenidos, se muestra el sistema en funcionamiento, remarcando sus virtudes y desventajas. Con este fin, se hace una comparación con el desarrollo tradicional, demostrando las diferencias que se obtienen con la utilización de CI/CD en el proceso de desarrollo.
- El capítulo 7 contiene las conclusiones y trabajos futuros. La conclusión tiene una breve reseña del objetivo de este trabajo y lo que se pudo lograr y, en la sección de trabajos futuros, se abordan todos los aspectos que, por la dimensión del trabajo, no se pudieron contemplar, pero resultan de interés.

Capítulo 2 - Marco Teórico: Microcontroladores

En este capítulo se va a definir uno de los temas principales que atraviesan la tesis. Se explicará qué son los microcontroladores y para qué se utilizan. Se detallarán, además, los tipos de microcontroladores existentes, las placas de desarrollo y sus utilidades, el microcontrolador que se utilizará en el presente proyecto y la tecnología OTA de la que se va a hacer uso. Asimismo, se precisará qué es la plataforma Arduino y por qué es de interés para este trabajo.

2.1 Microcontroladores

En primer lugar, en pos de entender correctamente qué son y con qué fin se utilizan los microcontroladores, vamos a diferenciarlos de los microprocesadores.

Para entender la diferencia entre un microcontrolador y un microprocesador es necesario dividir la computadora en sus unidades funcionales básicas [8][9]:

- La unidad de control: que gobierna a todos los demás elementos.
- La unidad aritmético-lógica (UAL o ALU en inglés): encargada de realizar las operaciones aritméticas y lógicas.
- La memoria principal: donde se encuentra almacenado el programa que se está ejecutando y todos los datos que maneja.
- La entrada-salida (E/S o I/O en inglés): que permite introducir y obtener datos al y desde el ordenador, incorporando hardware en forma de periféricos.

Teniendo en cuenta estas partes, se puede identificar al microprocesador de la siguiente manera:

> ”Cuando se habla conjuntamente de la unidad de control y la unidad aritmético-lógica se le da el nombre de procesador o Unidad Central de Procesamiento (UCP o CPU de *Central Process Unit*). Y cuando estas dos unidades se integran en un microchip, se le da el nombre de microprocesador.” [8]

Los más grandes fabricantes de microprocesadores son Intel y AMD, cuyos productos están en la gran mayoría de computadoras alrededor del mundo. A diferencia de los microprocesadores, los microcontroladores incorporan todas las unidades funcionales anteriormente nombradas en un solo microchip y, por lo tanto, constituyen íntegramente una computadora.

Como relata Prieto [8], a lo largo de la historia los microprocesadores fueron evolucionando en potencia, es decir, su capacidad de ejecutar instrucciones y la memoria que pueden manejar, adaptándose a cualquier aplicación de propósito general, que es lo que requiere una computadora personal: edición de texto, hojas de cálculo, reproducción de música, videojuegos, etcétera. Los microcontroladores, a diferencia, tienen un propósito específico y no necesitan ser acompañados de hardware como una memoria o un módulo de E/S aparte, ya tienen ese hardware integrado en el microchip, por lo que la potencia no es su único objetivo. En palabras de Valdés y Areny [9]:

> “Los microcontroladores están concebidos fundamentalmente para ser utilizados en aplicaciones puntuales, es decir, aplicaciones donde el microcontrolador debe realizar un pequeño número de tareas, al menor costo posible. En estas aplicaciones, el microcontrolador ejecuta un programa almacenado permanentemente en su memoria, el cual trabaja con algunos datos almacenados temporalmente e interactúa con el exterior a través de las líneas de entrada y salida de las que dispone.”

Para hacer un resumen simplista, se podría decir que un microprocesador es una de las partes que componen una computadora personal, el cual se debe conectar con otros componentes de hardware (placa madre, memorias, discos, etc.), y tienen propósito general. Por otro lado, un microcontrolador es una computadora pequeña y menos costosa, montada sobre un microchip o circuito integrado. En consecuencia de estas limitaciones físicas, tiene una menor capacidad, pero se aprovecha su practicidad y eficiencia con aplicaciones específicas, sirviéndose de su pequeño tamaño, ligero peso, bajo consumo de batería y su capacidad de trabajar en tiempo real [10].

Los microcontroladores se pueden encontrar en una gran variedad de dispositivos, Tanenbaum y Austin [10] hacen una clasificación por categoría con algunos ejemplos:

- Electrodomésticos: radio reloj, lavadora, secadora, microondas, alarma antirrobo.
- Equipo de comunicaciones: teléfono inalámbrico, teléfono celular, fax.
- Periféricos informáticos: impresora, escáner, módem.
- Dispositivos de entretenimiento: DVD, estéreo, reproductor de MP3.
- Dispositivos de imagen: TV, cámara digital, fotocopiadora.
- Dispositivos médicos: rayos X, resonancia magnética, monitor cardíaco, termómetro digital.
- Sistemas de armas militares: misiles crucero, misiles balísticos intercontinentales, torpedos.
- Dispositivos de compra: máquina expendedora, cajero automático, caja registradora.
- Juguetes: muñeco parlante, videoconsola, auto o barco a control remoto.

2.2 Tipos de microcontroladores

Hay varias formas de clasificar a los microcontroladores, la principal se puede realizar considerando el propósito de los mismos. Siguiendo esta idea, se dividirían en:

- De propósito general
- De propósito específico, los cuales pueden subdividirse. Por ejemplo, un gran productor de microcontroladores, cómo lo es Texas Instruments, los subdivide en:
    - Real-time control
    - Industrial sensing
    - Industrial Communications
    - Automotive-qualified
    - High performance

En la Figura 1, se puede apreciar como este gran productor clasifica sus productos, en donde un mismo equipo puede servir para 1 o más propósitos:

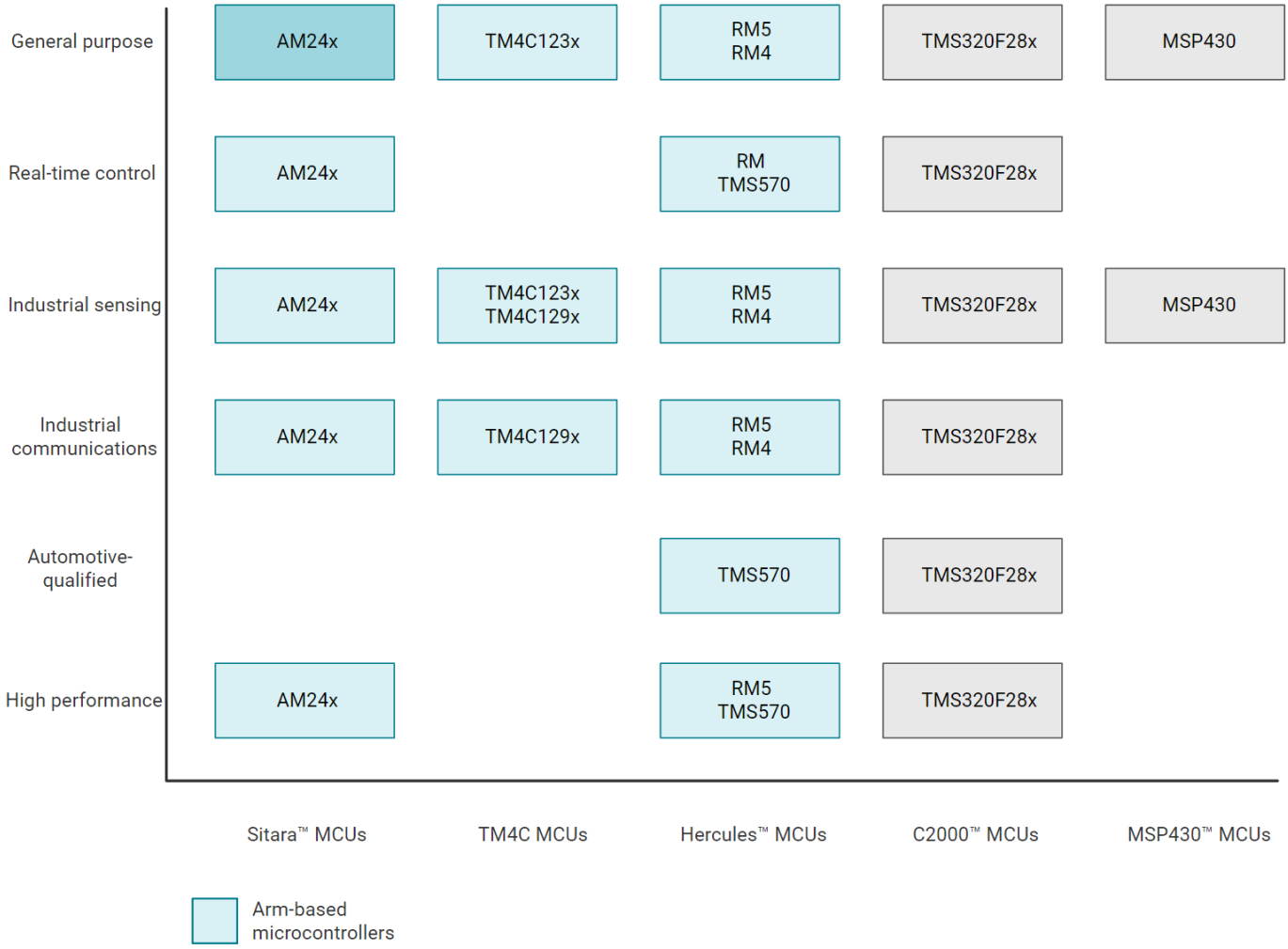

Figura 1. Microcontrollers for every design need. [11]

Otra manera de clasificarlos es según la cantidad de bits. Como muestran Tanenbaum y Austin [10], los microcontroladores vienen en versiones de 4-bit, 8-bit, 16-bit y 32-bit, en donde la capacidad del producto va en aumento. Microchip, otro gran productor, da una breve descripción de sus productos que puede servir para entender qué pretende lograr cada uno [12]:

- Microcontroladores de 4-bit (explicación externa, no se encuentra en el catálogo de Microchip): es el tipo de microcontrolador más pequeño, son de los primeros en surgir y, a día de hoy, se los considera obsoletos para la mayoría de las tareas. Generalmente son utilizados en pequeños dispositivos electrónicos, calculadoras de bolsillo, juegos electrónicos portátiles, controles remotos [13], cafeteras, etc.
- Microcontroladores de 8-bit: un chip con programación mínima, que sirve como puerta de entrada o nivel inicial para los programadores que están empezando en este tipo de desarrollos. Su diseño está orientado a aplicación en tiempo real.

- Microcontroladores de 16-bit: son ideales para aplicaciones que necesitan un rendimiento de bajo consumo extremo o que han superado el rendimiento o las capacidades de memoria de un microcontrolador 8-bit. Gozan de una mayor cantidad de instrucciones, una mayor memoria, una mayor integración con periféricos, incluyendo USB y LCD, y funciones de seguridad de hardware.
- Microcontroladores de 32-bit: brindan rendimiento y capacidades funcionales para satisfacer las necesidades en una amplia variedad de aplicaciones. Pueden correr aplicaciones multihilo o multi-thread, tienen capacidades de seguridad como arranque seguro, actualización segura de firmware, aislamiento de hardware y protección de claves, una mejor conectividad, incluyendo USB de alta velocidad y Ethernet, entre otras mejoras.

En este proyecto, debido a que busca facilitar el desarrollo de software complejo sobre microcontroladores, se va a hacer uso de un microcontrolador de 32-bit. Este tipo de microcontrolador, debido a que es el de mayor capacidad, es el idóneo para tareas de propósito general.

2.3 Placas de desarrollo

Las placas de desarrollo son piezas de hardware que acompañan al microcontrolador (u otro tipo de hardware) con las herramientas necesarias para comenzar un proyecto [14]. Quienes hacen las placas de desarrollo no tienen por qué ser el mismo fabricante del microcontrolador, sino que se encargan de acompañar al chip con hardware que no tiene integrado y puede ser de utilidad. Por ejemplo, el microcontrolador ESP32s posee casi todo lo necesario para comenzar a procesar, incluso una antena WiFi impresa, pero la placa de desarrollo se encarga de agregar regulaciones de voltaje, conexiones USB y lo necesario para que los pines sean accesibles desde el exterior, además del circuito impreso donde está todo conectado.

En general, las placas de desarrollo para microcontroladores deben proveer [14]:
- Un circuito auxiliar, necesario para alimentar y hacer funcionar el microcontrolador.
- Una forma de programar y comunicarse con el microcontrolador.
- Conectores para una fácil conexión a circuitos externos.
- Posiblemente, algunos circuitos integrados útiles para probar algunos de los periféricos.

En la Figura 2, se puede apreciar una placa de desarrollo que acompaña al microcontrolador ESP-32s, el cual está señalado con un círculo rojo, agregando hardware como la entrada Micro-USB, para alimentar el microcontrolador, o los pines que salen al exterior, para facilitar la conexión externa:

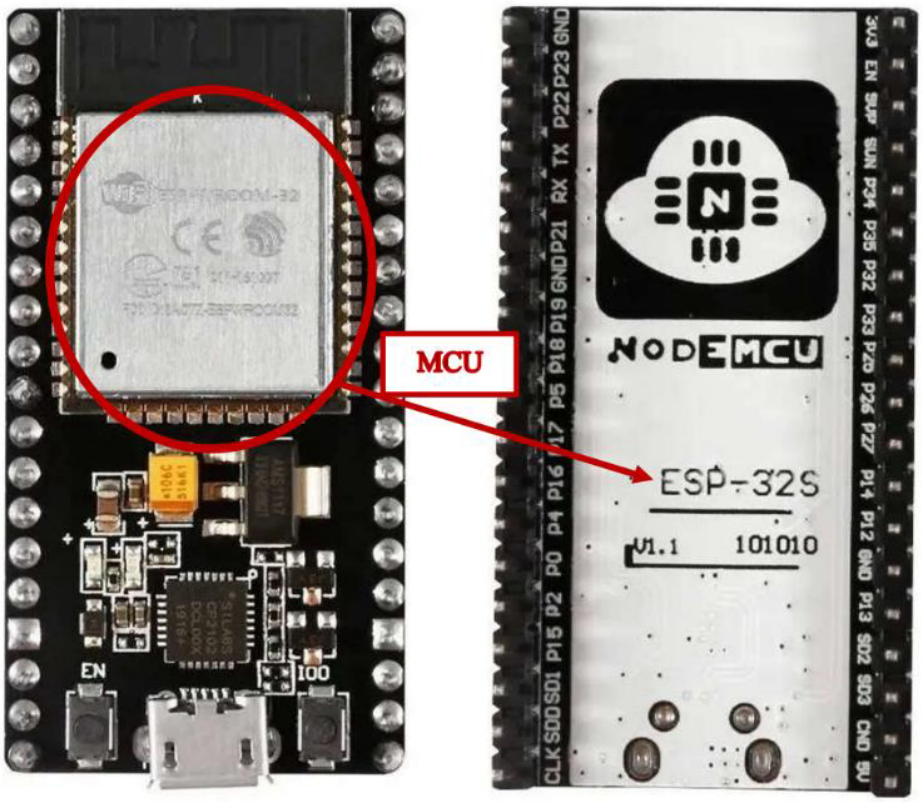

Figura 2. Placa de desarrollo de un ESP32s.

2.4 ESP32

El ESP32 [15], sucesor del popular ESP8266 [16], es un microcontrolador de la familia de los ESP, productos desarrollados por Espressif Systems [17]. Este microcontrolador es 32-bit, de propósito general, con buen rendimiento y bajo consumo. También, tiene la capacidad de interactuar con redes WiFi y Bluetooth, algo fundamental para el proyecto, ya que se va a hacer uso del tipo de actualización OTA, explicada posteriormente.

Ahondando en las especificaciones, posee un procesador dual core, con 2 Tensilica Xtensa LX6 32-bit, con una frecuencia de reloj de 160 MHz, tiene una memoria SRAM de 448 KB y una flash de 520 KB, y una gran cantidad de pines E/S. En la Figura 3 se puede observar una placa de desarrollo que contiene al microcontrolador, identificando cada pin y mostrando las diferentes partes del hardware:

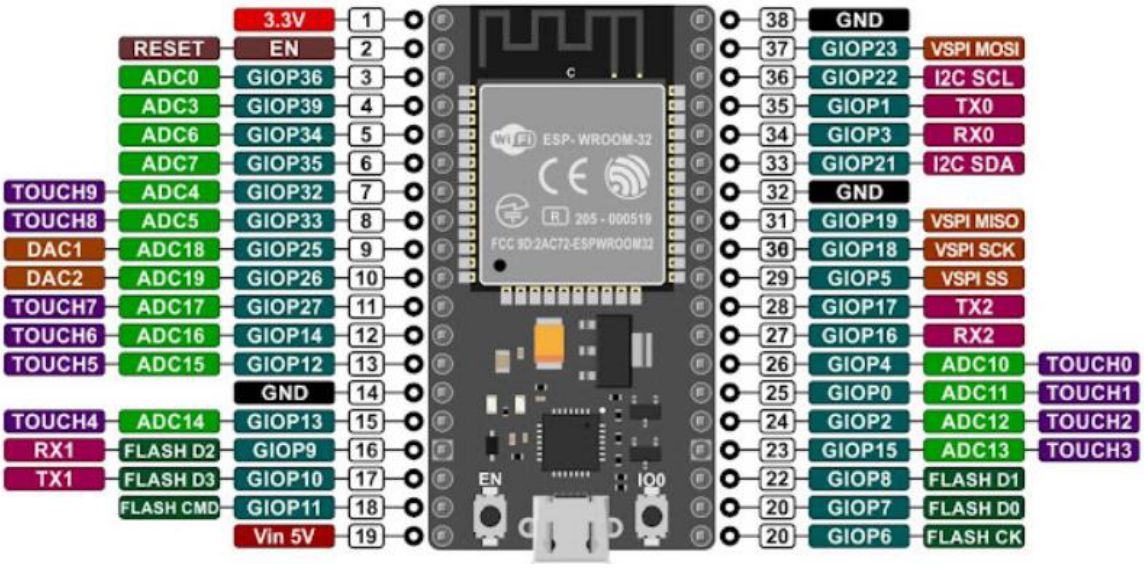

Figura 3. ESP32 y sus pines de E/S. [18]

El ESP32 posee un alto nivel de integración, dentro del chip incluye interruptores de antena, amplificadores de potencia, filtros y módulos de administración de energía, entre otras características. Hay que mencionar, además, que logra su funcionamiento con un bajo consumo, a través de funciones de ahorro de energía que incluyen sincronización de reloj y múltiples modos de operación. Por este motivo es considerado una herramienta ideal para proyectos energizados con baterías o para aplicaciones del IoT [19].

También incluye/contiene FreeRTOS, que es un sistema operativo orientado a las aplicaciones de tiempo real. Este sistema operativo permite manejar varias tareas independientes de manera paralela [20], y es el que efectivamente provee soporte para todo lo que se ejecuta, por ejemplo, habilitando el uso de OTA, el método de actualización que se explicará posteriormente, al final de este capítulo.

Se lo conoce como el sucesor del ESP8266, otro conocido microprocesador, ya que en comparación posee un núcleo adicional, mejor o más rápida conexión WiFi, mayor número de pines de entrada/salida y más opciones en la conexión Bluetooth [19]. Para mayor información técnica, se puede indagar en su hoja de datos técnicos o datasheet [21] o en el manual de referencias técnico [22], ambos ofrecidos por Espressif.

2.5 Arduino

Arduino [23] es una plataforma cuyo objetivo inicial era crear placas de desarrollo más baratas que las de la época, para hacerlas más accesibles al público. El sistema Arduino tiene un diseño de hardware de código abierto, lo que significa que todos sus detalles se publican y son gratuitos para que cualquiera pueda construir (e incluso vender) un sistema Arduino [10]. En un principio, los dispositivos se basaban en el microcontrolador RISC de 8 bits AVR de Atmel, y la mayoría de los diseños de placas también incluían soporte básico de E/S. A día de hoy, la realidad es distinta, hay una gran variedad de placas de desarrollo con diferentes microcontroladores y, por lo tanto, con diferentes capacidades [24].

Arduino no solo se encarga de crear placas, sino que es una plataforma que además brinda software y contenido educativo, cuya comunidad tiene alrededor de 30 millones de usuarios activos. Esta plataforma es de particular interés en este proyecto, ya que vamos a utilizar 2 de los softwares que proveen:

- Arduino IDE [25]: es un ambiente de desarrollo (funcional en Windows, Linux y macOS) que provee una gran cantidad de herramientas para trabajar con la mayoría de los microcontroladores, entre ellos el ESP32. Este programa permite, por ejemplo, escribir código y compilarlo, cargarlo al microcontrolador (vía cable), leer la salida, etc.
- ArduinoCLI [26]: la CLI (Command Line Interface) de Arduino es la interfaz de línea de comandos que provee esta plataforma, la cual ayudará en el proceso de integración continua. Particularmente, se va a hacer uso de la funcionalidad que permite compilar el código y generar el binario correspondiente, archivo que se cargará posteriormente en el ESP.

En comparación con los IDEs tradicionales, usados en el desarrollo de software popular o mainstream, el de Arduino es un IDE bastante limitado, carente de características como una ejecución paso a paso, breakpoints, visualización y navegación entre directorios del proyecto, entre otras funcionalidades. La documentación tampoco es un punto fuerte, muchas veces tanto las herramientas de Arduino como el Hardware destinado a usarse con él tienen una documentación incompleta o nula.

El potencial de Arduino, y por lo que generó una base tan grande de usuarios, reside en su simplicidad y su alta compatibilidad con todo tipo de hardware, algo que en otros contextos ha tomado muchos años y se ha realizado de manera propietaria. Arduino no solo es open source, sino que se lo podría considerar open hardware, utilizando hardware con especificaciones abiertas al público.

2.6 OTA (Over-The-Air)

Las actualizaciones de software son una parte esencial de casi cualquier sistema con computadoras. Existen diferentes tipos de estrategias para implementar estas actualizaciones, variando los métodos y prácticas utilizados para entregar los paquetes de actualización a los sistemas finales. Un tipo particularmente interesante es la actualización Over-The-Air (OTA), que consiste en actualizar de forma inalámbrica el código de un dispositivo integrado, como pueden ser los microcontroladores, enviando datos desde un servidor a diferentes nodos o clientes. Las actualizaciones de OTA se usan ampliamente en varios dominios, uno de los más populares es el IoT [27]. Hablando más técnicamente, OTA permite escribir la memoria flash del microcontrolador, que es donde se guarda el código que debe ejecutar cuando inicia (firmware).

Sin embargo, para cambiar el programa que corre un microcontrolador, no solo alcanza con escribir el nuevo código en su memoria flash, también hay que reiniciar el dispositivo. Repasando la evolución de las actualizaciones en los microcontroladores, se puede ver que, en un principio, las memorias que contenían el firmware no se podían modificar. Por lo tanto, para cambiar el programa que estaba corriendo el microcontrolador, había que cambiarle la memoria, y luego volverlo a prender. Con el paso del tiempo, se comenzaron a utilizar las memoria EPROM (Erasable Programmable Read-Only Memory), la cual podía

ser reescrita, pero se necesitaba un instrumento específico. Para modificar su código, había que remover la memoria, insertarla en el instrumento, borrar los datos¸ escribir el nuevo código, y luego volver a colocarla en el microcontrolador, volviendo a encender el dispositivo. La última mejora antes de OTA se puede ver, por ejemplo, en el ESP8266-01/07. Este dispositivo tenía 2 modos de arranque:

a. Flash mode: arranca con el código guardado en el flash.
b. UART mode: arrancar preparado para recibir un nuevo código en la flash, así el próximo arranque sería utilizando el nuevo código.

La elección del modo de arranque se realizaba según uno de los pines del microcontrolador (GPIO0), se arrancaba con UART mode si el pin estaba conectado a tierra, sino se usaba el Flash mode [28][29][30].

Como queda en evidencia, todas las versiones anteriores a OTA necesitaban un acceso físico al hardware, lo cual era altamente restrictivo para el planteamiento de sistemas distribuidos. OTA no solo permite escribir la memoria flash del microcontrolador, sino que también permite, mediante software, realizar el reinicio necesario para que el equipo comience a ejecutar el nuevo código.

Un claro ejemplo del beneficio de utilizar actualizaciones OTA se da en las flotas de vehículos con dispositivos integrados. En caso de que haya un error, o simplemente se quiera mejorar el servicio, realizar la actualización de manera tradicional implicaría que los vehículos deben ir a un punto central, un garaje o un servicio técnico, para que un técnico pueda cargar la nueva versión. Con la actualización OTA, esta situación no ocurre, el cliente solo debe buscar una red WiFi a la que conectarse, o la red móvil de su celular, y descargar directamente la actualización correspondiente. También, la empresa que hace el vehículo, puede instalarle una tarjeta SIM, como la que poseen los celulares, para que el usuario no tenga que preocuparse por buscar una red [31]. La Figura 4 provee un ejemplo, detallando los agentes que intervienen para actualizar el software de un automóvil:

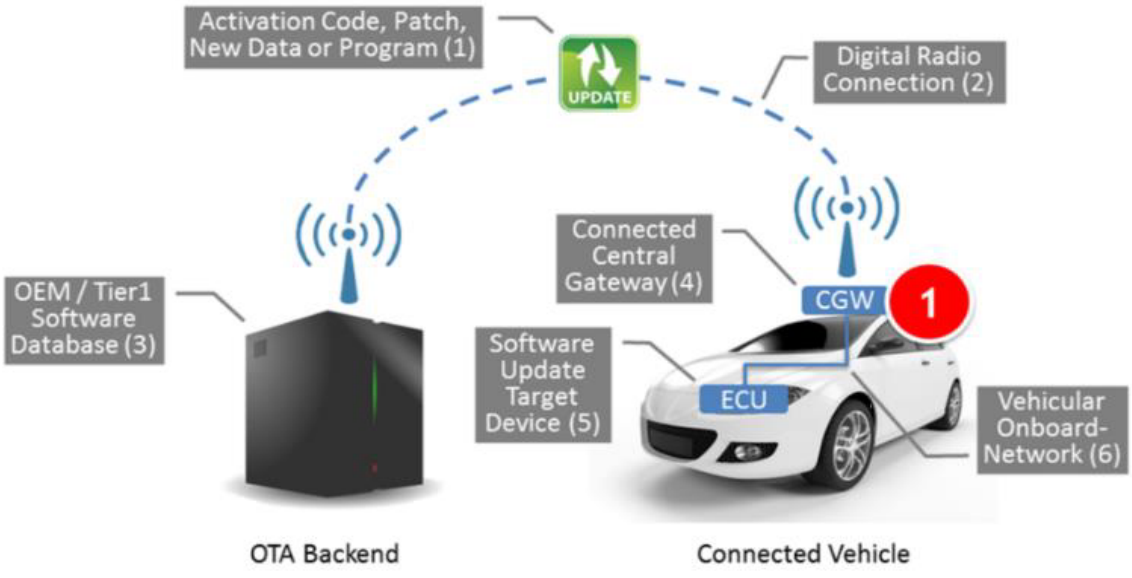

Figura 4. Vehicular Over-The-Air update components. [31]

Hay varias consideraciones que deben tenerse en cuenta al implementar OTA, pero la más relevante es sobre la seguridad. Por sí mismo, este método de actualización no tiene ningún tipo de seguridad incluida, cualquier persona sería capaz de subir una actualización al dispositivo, lo cual genera un gran riesgo [32]. Sin embargo, ya hay varias soluciones e implementaciones de OTA que le agregan una capa de seguridad a la misma, se comentará más al respecto en el Capítulo 7, en la sección de “Trabajos futuros”.

Capítulo 3 - Marco Teórico: CI/CD en sistemas distribuidos

En este capítulo se explicará en profundidad qué es CI/CD y las ventajas que puede traer su integración con los microcontroladores. Por otro lado, se explicarán diferentes tipos de sistemas: los sistemas distribuidos, los sistemas de tiempo real y los sistemas de control de versiones. Si bien estos temas se pueden considerar independientes, se interrelacionan debido a las vinculaciones que tienen con los microcontroladores.

3.1 CI/CD

El CI/CD es un método de desarrollo cuya adopción se está incrementando en los últimos años [5]. Este método automatiza una parte del trabajo que se necesita para llevar un código a producción. Con un canal o pipeline de CI/CD, los desarrolladores pueden lograr que se genere una build, que la misma se testee y luego se ponga en marcha en el ambiente productivo, todo de manera automática [33].

Este proceso es importante porque ayuda a los desarrolladores a trabajar de manera eficiente, disminuyendo el tiempo consumido en trabajo manual tedioso y, además, facilitando la integración con el ambiente de producción. Esta automatización hace que los procesos sean predecibles y repetibles, lo que se traduce en menos oportunidades de error por parte de la intervención humana [6].

El proceso de CI/CD se puede dividir en 3 etapas, donde cada una de ellas se encarga de automatizar una o más tareas del desarrollo tradicional [34]:

1. Continuous Integration: incluye las etapas de desarrollo, generar una build y los test de unidad.
2. Continuous Delivery: generalmente lleva a cabo los test de integración y se encarga de almacenar la nueva build en un repositorio de código.
3. Continuous Deployment: se encarga de las tareas que llevan a cabo el despliegue de ese proyecto en el ambiente de producción.

Sin embargo, la definición de estas etapas y las tareas que realizan puede variar, no es un límite estricto.

Al conjunto de todas las tareas que realiza en el proceso de CI/CD se lo suele llamar pipeline (Figura 5), y como se dijo, este pipeline puede variar según cada proyecto, omitiendo o agregando tareas. Para ejemplificar, en el artículo de Rangnau et al. [35], se realiza una descripción detallada de cómo sería un pipeline estándar en un proyecto convencional:

> “The Continuous Integration stage starts with a commit, followed by a build of the modified application which is then verified using unit tests. When all test cases pass, the tested application is deployed to a testing environment. If Continuous Delivery is implemented, a set of automated acceptance tests is executed to verify that there are no regressions in the system’s features. This step also helps to identify any errors that may occur due to a difference in run-time environment because the testing environment is usually a server with similar configuration to the production environment. Depending on the level of automation, this step can also involve manual testing and approval before the pipeline advances to the next stage. When all previous stages have passed, the system is automatically deployed to the production environment in the Continuous Deployment stage, where the users will have access to the new version. If a test fails, or when an error occurs during build or deployment, the pipeline is automatically stopped and developers are notified of the error.”

Figura 5. CI/CD pipeline. [33]

Si bien los microcontroladores han tenido tradicionalmente una capacidad reducida (sobre todo en comparación a las computadoras personales), la evolución en las tecnologías ha mejorado cada aspecto de los mismos. Se ha mejorado la potencia, logrando un nivel considerable, se ha aumentado la capacidad de memoria y se han agregado nuevas formas de E/S, como el WiFi o Bluetooth. Este crecimiento en las capacidades de los microcontroladores abre nuevas puertas, permite que el software que son capaz de ejecutar se vuelva cada vez más complejo. Esta es una de las razones del por qué empieza a ser atractiva la incorporación de metodologías como CI/CD al desarrollo sobre microcontroladores, anteriormente no solo el CI/CD no estaba tan consolidado en la industria del software tradicional como lo está ahora, sino que, debido a la poca capacidad de los microcontroladores y la carencia de la actualización OTA, resultaba inviable o inútil plantear el uso de este tipo de metodologías.

Lo hasta aquí explicado corresponde a la visión pragmática del CI/CD, que es lo que realmente se puede aplicar en el contexto de los microcontroladores, ya que, en la teoría, esta metodología no surge sólo con la idea de automatizar etapas del desarrollo, sino con conceptos más profundos. La primera concepción de la Integración Continua fue en el libro de Kent Beck en 1999 [36], un prestigioso defensor de las metodologías ágiles, como una de las prácticas deseadas en la metodología de desarrollo Extreme Programming. En su concepción, como explica Fowler [37], la Integración Continua apuntaba a que los programadores integren su código constantemente (al menos una vez por día), manteniendo un desarrollo sincronizado y fluido, para disminuir el tiempo que se perdía en la integración tradicional. Para que sea viable la idea de integrar diariamente, los programadores no podían perder demasiado tiempo, y de ahí surge la automatización de los procesos de build y de testeo. Con el tiempo, esta idea fue evolucionando, incluyendo el delivery y deployment continuo, completando el ciclo de CI/CD.

En el ambiente de los microcontroladores, las metodologías ágiles no son una preocupación, esta tesis no busca implementar el método de desarrollo CI/CD por sus fundamentos teóricos, sino que es la parte pragmática la que resulta interesante. Se logra una mayor eficiencia en diferentes etapas del desarrollo, ayudando al programador con tareas tediosas que pueden ser automatizadas y disminuyendo el riesgo de encontrarse con errores, independientemente de si se realiza con el fin de seguir la ideología de desarrollo ágil o no. Si bien las metodologías ágiles están enfocadas en proyectos de mayor magnitud, si la evolución en el campo de los microcontroladores sigue con un ritmo elevado, no resultaría extraño que empiece a ser una práctica habitual.

3.2 Sistemas distribuidos

Se han dado varias definiciones sobre los sistemas distribuidos, pero una en la que coinciden varios autores [4][38] es la siguiente: un sistema distribuido es una colección de computadoras o dispositivos independientes, que a vistas de los usuarios del sistema parecen una única computadora. Estos dispositivos o computadoras, en general, están dispersos geográficamente, y el tamaño del sistema puede variar desde algunos dispositivos a millones de ellos. Al igual que el tamaño, la red con la que se conectan también puede variar, puede ser cableada, inalámbrica o un híbrido entre ambos.

A cada computadora independiente se le suele llamar nodo, y los nodos no siempre son lo que entendemos tradicionalmente por computadoras, pueden ser sensores, servidores o, en relación a esta tesis, microcontroladores. Los nodos tienen un objetivo común, el cual logran intercambiando mensajes entre sí [4]. Esta combinación de dispositivos independientes que trabajan de manera colectiva se logra, en parte, integrando los diferentes protocolos de cada dispositivo en lo que se conoce como un middleware, que es una capa de abstracción mediante software que se sitúa entre los sistemas operativos y las aplicaciones distribuidas. El middleware, para un sistema distribuido, es, en esencia, lo que un sistema operativo es para una computadora, un administrador de recursos que los distribuye entre las aplicaciones de la red de la forma más eficiente posible [4].

Si bien los sistemas distribuidos tienen muchas ventajas, como puede ser incrementar el poder de cómputo uniendo computadoras, tener una escalabilidad mucho mayor, tener la capacidad de descentralización geográfica, entre otras, también vienen acompañados de un aumento en la complejidad. Al tener que interconectar varios nodos para que trabajen de forma conjunta, hay que tener en cuenta múltiples aspectos [39]:

1. Un sistema distribuido tiene muchos focos de control. El monitoreo secuencial y las técnicas de debugging (depuramiento), como el tracing (rastreo) y los breakpoints (puntos de control), que están basados en el program counter (contador de programa) y el process state (estado del proceso), necesitan ser adaptados para aplicarse a los sistemas distribuidos.
2. Los retrasos en la comunicación entre los nodos en un sistema distribuido dificultan determinar el estado de un sistema en un momento dado. Por ejemplo, el inicio de un intento de determinar el estado del sistema debe realizarse desde un nodo, y los otros nodos siempre serán notificados de este intento en momentos posteriores, que no se pueden predecir.

3. Los sistemas asincrónicos distribuidos son inherentemente no deterministas. Esto significa que dos ejecuciones del mismo sistema pueden producir un orden de eventos diferentes, pero no obstante válidos. Por lo tanto, es difícil reproducir errores y probar posibles situaciones.
4. El monitoreo de un sistema distribuido altera su comportamiento. El comportamiento de un programa secuencial no se ve afectado por la cantidad de tiempo transcurrido entre la ejecución de dos instrucciones sucesivas, por ejemplo, un debugger simbólico puede interrumpir un proceso secuencial en un punto de interrupción sin afectar la ejecución posterior del proceso. En un sistema distribuido, detener o ralentizar un proceso puede alterar el comportamiento de todo el sistema.
5. Las interacciones entre el sistema y el desarrollador del sistema pueden ser más complejas. Por ejemplo, cuando se conecta una terminal a cada procesador, es posible que el desarrollador del sistema deba moverse físicamente de una terminal a otra para iniciar procesos, establecer puntos de interrupción y examinar los rastros. Por lo tanto, es necesario proporcionar herramientas que abarquen un sistema distribuido y que puedan invocarse y controlarse desde un solo sitio.

3.2.1 Tipos de sistemas distribuidos

Hay diferentes tipos de sistemas distribuidos que, según van Steen y Tanenbaum [4], se pueden clasificar como orientados hacia la computación, el procesamiento de información o la omnipresencia:

1. Computación distribuida de alto rendimiento (High performance distributed computing): esta clase de sistemas distribuidos está pensada para realizar tareas de alto rendimiento, y tiene 2 subdivisiones:
   a. Computación en clúster (Cluster computing): el hardware que lo conforma consiste en una colección de dispositivos similares, corriendo el mismo sistema operativo, conectados por una conexión local de alta velocidad.
   b. Grid computing: el hardware que lo conforma consiste en una unión de dispositivos con diferentes características, tanto en el hardware, software o la red que utilizan. La ventaja de este tipo de sistemas es que permite la colaboración virtual de grupos de personas o equipos de diferentes entornos.

   Del grid computing surge posteriormente el popular Cloud Computing o computación en la nube, un sistema mediante el que se puede usar y acceder fácilmente a recursos virtuales [40].

2. Sistemas de información distribuida (Distributed information systems): esta clase de sistemas distribuidos se centra en, como su nombre lo indica, administrar información distribuida. Un ejemplo se da en una aplicación que esté naturalmente distribuida, como puede ser la aplicación de gestión de un supermercado que tenga varias sucursales. En este caso el sistema se encargaría de recopilar la información de cada sucursal y, por ejemplo, unificar la facturación. Otro utilidad que tienen estos sistemas es para integrar diferentes aplicaciones, ya existentes, de un sistema de información. Muchas veces las organizaciones se encuentran con una gran cantidad de aplicaciones en la red y diferentes bases de datos que la acompañan. La idea de estos sistemas es integrar todas esas aplicaciones y darle la posibilidad al usuario de enviar una única petición sin tener que pensar en cada aplicación en particular. Este sistema se debe encargar de procesar la petición del

cliente y descomponerla en varias otras diferentes, de un nivel más bajo, que puedan llegar a cada aplicación en particular. Luego, debe recomponer los resultados que da cada aplicación para lograr una respuesta satisfactoria, lo que, muchas veces, conlleva a que las aplicaciones deban comunicarse entre ellas.

3. Sistemas pervasivos (Pervasive systems): este tipo de sistema tiene la intención de integrarse con naturalidad al ambiente. Generalmente, un sistema pervasivo está compuesto por muchos sensores, los cuales obtienen información del comportamiento del usuario, y tienen actuadores, para proveer feedback, usualmente con el propósito de dirigir ese comportamiento. Una casa inteligente es un ejemplo perfecto de este tipo de sistemas, la casa tiene varios sensores que monitorizan el comportamiento del usuario y responden en consecuencia. Los dispositivos que se utilizan en este tipo de sistema pueden ser diversos, generalmente caracterizados por ser pequeños, alimentados por baterías, móviles y tener conexión inalámbrica, descripción que es totalmente compatible con los microcontroladores. Hay diferentes tipos de sistemas pervasivos, aunque compartan ciertas características entre ellos:
    a. Sistemas de computación ubicuos (ubiquitous computing systems): el sistema es pervasivo y continuo, esto significa que el usuario va a estar continuamente interactuando con el sistema, sin casi darse cuenta de ello. Es un sistema que trabaja de manera autónoma, conociendo y optimizando el ambiente del usuario, con una interacción muy poco intrusiva.
    b. Sistemas de computación móviles (mobile computing systems): son sistemas compuestos por dispositivos móviles, generalmente de conexión inalámbrica. Los dispositivos que forman este tipo de sistema pueden variar mucho, se suele utilizar dispositivos como smartphones o tablets, pero pueden incluirse dispositivos que puedan usar GPS, controles remotos, equipamiento de autos, etc.
    c. Redes de sensores (sensor networks): una red de sensores, desde el punto de vista de los sistemas distribuidos, puede ser más que solo una colección de inputs o entradas de información. En lugar de eso, los dispositivos pueden colaborar para procesar los datos detectados, dependiendo de cada aplicación específica.

Los microcontroladores generalmente están pensados para formar parte de los sistemas distribuidos pervasivos, en cualquiera de sus subclasificaciones. Pueden estar en la computación ubicua, por ejemplo, en una casa inteligente que tenga una gran presencia de dispositivos electrónicos. Puede, también, estar en un sistema de computación móvil, controlando una flota de vehículos que lleven integrado un microcontrolador. De igual modo, pueden estar presentes en una red de sensores, por ejemplo, controlando un proceso productivo, en donde se encargue de regular el funcionamiento de alguna máquina que posea diferentes sensores.

3.3 Sistemas de tiempo real

Los sistemas de tiempo real son aquellos sistemas en donde la correctitud del mismo no solo depende de los resultados de su computación, sino también del tiempo en el que esos

resultados se producen [41]. Los sistemas de tiempo real suelen ser sistemas que interactúan constantemente con factores externos, dando respuestas de manera reactiva. Aquellos sistemas que consideren un fracaso el no responder a tiempo, se consideran sistemas de tiempo real. Ahora, el cómo se define el fracaso puede ser muy amplio, puede variar tanto la ventana de tiempo que tienen los sistemas para dar la respuesta, como las consecuencias que provoca ese fracaso [14]. Atendiendo a esta idea, una de las maneras más comunes de clasificar los sistemas en tiempo real es la siguiente [14][42][43][44][45]:

- Sistemas de tiempo real duros (Hard real-time systems): un sistema de tiempo real duro debe cumplir con su plazo de respuesta (deadline) el 100% de las veces. En caso de tener una sola respuesta fuera de tiempo, ya se considera un fallo del sistema. Un ejemplo de este tipo de sistema se puede encontrar en dispositivos médicos, como los marcapasos, en donde no está permitido ni un solo fallo, el dispositivo debe adaptarse constantemente y dar los impulsos eléctricos que el paciente necesita dentro del tiempo estipulado, una omisión o atraso en la respuesta podría resultar crítico.
- Sistemas de tiempo real firmes (Firm real-time systems): un sistema de tiempo real firme debe cumplir con el deadline casi todo el tiempo. Por dar un ejemplo, supongamos un dispositivo que regula la temperatura de una máquina, realizando mediciones constantes. Que el sistema no responda a tiempo generará un breve aumento o disminución en la temperatura no es deseado en la máquina, pero eso no significa un accidente crítico (siempre y cuando la máquina no sea demasiado sensible), pero, si este error se realiza lo suficientemente seguido, puede terminar dañando la máquina.
- Sistemas de tiempo real blandos (Soft real-time systems): estos tipos de sistemas son los más laxos en cuanto al cumplimiento del deadline, suelen prometer solo realizar el mejor esfuerzo. Un buen ejemplo es el sistema de control crucero de un auto, en donde los conductores no eligen una cantidad determinada de km/h que el sistema puede variar de la velocidad fijada, no hay un requerimiento severo. Simplemente esperan que, dadas circunstancias razonables, el control de velocidad se mantenga parecido a lo fijado la mayor cantidad de tiempo posible, pero una variación temporal o una imprecisión razonable no resulta un problema.

Es recomendable utilizar los microcontroladores en sistemas de tiempo real tanto firmes como blandos, por ejemplo, para controlar flotas de vehículos, para ambientes industriales o para otras aplicaciones del estilo. Sin embargo, esto no significa que no se puedan utilizar en sistemas de tiempo real duro, se puede usar siempre y cuando cumpla con el requerimiento del deadline el 100% de las veces. Pero hay que tener en cuenta que, si ocurre una actualización, puede que el dispositivo no responda a tiempo. Para adaptarlo, hay que asegurarse que margen de respuesta que exige el sistema sea, por lo menos, mayor al tiempo que tarda la actualización sumado el tiempo que tarda el microcontrolador en generar la respuesta, sin mencionar que se debería implementar una funcionalidad para que el dispositivo sea consciente de que ocurrió un evento antes o mientras se estaba reiniciando y lo pueda recuperar. Otra alternativa, dependiendo de la frecuencia de los eventos, podría ser posponer las actualizaciones hasta que el microcontrolador identifique que hay una ventana de tiempo en la que no van a ocurrir eventos. Sin embargo, es poco probable que un sistema real duro tenga un margen de respuesta tan amplio, o una frecuencia de eventos tan baja, por ese motivo se suelen utilizar en sistemas firmes o blandos.

3.4 Sistema de control de versiones

Un sistema de control de versiones es un software que se utiliza para almacenar las diferentes versiones por las que pasa un proyecto durante el desarrollo. En cada versión puede haber ficheros nuevos, modificados o eliminados, y este sistema se encarga de guardar un historial de todos los cambios, facilitando a los usuarios la administración del proyecto [46].

Las funcionalidades básicas que debe proveer cualquier sistema de versiones son las siguientes [47][48]:

1. Mantener un historial: el sistema debe permitir que se suban diferentes versiones y generar un historial, pudiendo identificar cuál es la última versión del código, y quién y cuándo la ha cargado.
2. Trabajo de versiones: el sistema debe permitir comparar dos versiones de código diferentes. Generalmente, las versiones van acompañadas de una descripción para facilitar su comprensión.
3. Regresar atrás: si se desea, se debe poder restaurar el proyecto en base a alguna versión almacenada.
4. Creación de distintos flujos de trabajo: se debe poder crear ramas separadas de un proyecto para, por ejemplo, trabajar paralelamente en diferentes funcionalidades o versiones de una aplicación.
5. Concurrencia del trabajo: varias personas deben poder trabajar sobre un proyecto al mismo tiempo, por lo que se necesita determinar alguna política de resolución de conflictos (fusión o bloqueo).

El sistema de control de versiones moderno más utilizado del mundo es Git [49], un proyecto de código abierto creado por Linus Torvalds. Git presenta una arquitectura distribuida, en donde los clientes no solo descargan la última copia instantánea de los archivos, sino que se replica completamente el repositorio. De esta manera, si un servidor deja de funcionar, cualquiera de los repositorios disponibles en los clientes puede ser copiado al servidor con el fin de restaurarlo. Además, permite que varios desarrolladores trabajen en simultáneo, estableciendo varios flujos de trabajo o creando diferentes ramas con versiones diferentes del software [50].

En este proyecto se va a utilizar GitHub [51], una aplicación que provee cierta infraestructura y herramientas utilizando el sistema Git, aunque podría ser reemplazable por cualquier otro sistema que implemente Git.

En particular, GitHub provee dos herramientas que son interesantes y de las que se hará uso. La primera es la API, que permite obtener los archivos de un repositorio, y la segunda son los webhooks, que permiten notificar a un servidor cuando hay cambios en el repositorio. Para ambas herramientas hay alternativas, por eso es apropiado recalcar que no es obligatorio el uso de GitHub.

Capítulo 4 - Propuesta

La propuesta de este proyecto es integrar las prácticas CI/CD al desarrollo sobre microcontroladores, lo que resultaría en un sistema capaz de automatizar algunas tareas, generando un ambiente productivo mucho más eficiente. Como se plantea en los objetivos, hay 4 pasos que son los fundamentales en la mayoría de los desarrollos [7]:

1. Integración con el repositorio
2. Build
3. Testing
4. Deploy

Del paso 2 y 3 es responsable la Integración Continua (CI), y el paso 4 le corresponde al Despliegue Continuo (CD).

El primer paso se resuelve simplemente creando un repositorio en cualquier sistema de control de versiones. Este repositorio estará alocado en los servidores de la empresa dueña del sistema de control de versiones elegido, por lo que el desarrollador simplemente deberá tener una conexión con el mismo para usar las herramientas que se le provea y poder subir su código.

Para realizar los demás pasos, se deben realizar diferentes tareas, por lo tanto, se va a crear un módulo o agente llamado “Control”, quien va a ser el encargado de comunicarse con el repositorio y los microcontroladores. Por ese motivo, puede estar alojado en una computadora que no necesariamente sea la misma del desarrollador. En la implementación, este agente puede ser representado de diferentes formas, pero se entrará en detalle en el próximo capítulo.

Siguiendo con el paso 2, para realizar la build, este módulo de control debe encargarse de, mediante algún tipo de comunicación con el repositorio, conseguir los archivos que correspondan al nuevo código. Una vez obtenido el código, deberá compilarlo y generar la build, representada por un archivo binario. Comentar que, para saber cuándo realizar una build, debe haber algún tipo de supervisión que le indique al módulo de control cuándo se sube código nuevo, ya sea que mediante una notificación del repositorio o implementando algún sistema de sondeo (polling).

Una vez generado el archivo binario, el módulo de control también será el encargado de testear la nueva versión. Como se comentó anteriormente, el testeo de microcontroladores sobre simuladores no es una buena alternativa, siendo la forma óptima de realizarlo sobre hardware real. Por este motivo, el sistema que propone esta tesis va a incluir un microcontrolador aparte del que se use en producción, el cual, por una cuestión didáctica, se llamará microcontrolador de testeo. Entonces, el módulo de control debe encargarse de subir el nuevo archivo binario al microcontrolador de testeo, y luego comenzar a realizar los test que se hayan configurado.

Como se desea que el proceso sea automático, para subir el archivo binario al microcontrolador de testeo se va a utilizar OTA, tecnología mediante la cual se enviará el archivo de forma remota y no se necesitará intervención física en el dispositivo. Esto también habilita a que el ESP no deba encontrarse ubicado o conectado a la computadora del

desarrollador ni a la que corre el módulo de control, siempre y cuando se pueda conectar por WiFi a ésta última.

Si no hay ningún fallo al testear, se debe proceder al deploy, caso contrario se puede notificar al desarrollador o directamente cancelar el proceso de actualización.

Realizar el deploy, en este caso, es una tarea simple, también llevada a cabo por el módulo de control, que consiste en cargar la nueva versión al ambiente productivo. En este trabajo, el ambiente de producción es un microcontrolador el cuál, por una cuestión didáctica, se llamará ESP de producción, y debería estar en ejecución corriendo algún programa. Para cargar la nueva versión, tal cual se hizo con el microcontrolador de testeo, el módulo de control se encargará de enviar el binario mediante OTA, pero ahora al microcontrolador de producción.

Por facilitar la explicación, hasta aquí se ha hablado de un único desarrollador que sube código al repositorio, y que el ambiente productivo está representado por un solo microcontrolador, pero no es necesariamente así. Puede haber varios desarrolladores subiendo código al repositorio, lo cual no afectaría en absoluto al sistema, o también puede darse que el microcontrolador de producción sea, en lugar de uno solo, varios microcontroladores distribuidos, por ejemplo, en una flota de vehículos. Para esta situación, lo único que cambiaría es que, en lugar de hacer el envío del binario mediante OTA a un solo microcontrolador, el módulo de control debería realizar el envío a todos los microcontroladores que sean necesarios, lo cual tampoco representaría un problema.

En la Figura 6 se muestra, en resumen, el flujo de trabajo, dividiendo las tareas del proceso de CI/CD según el color.

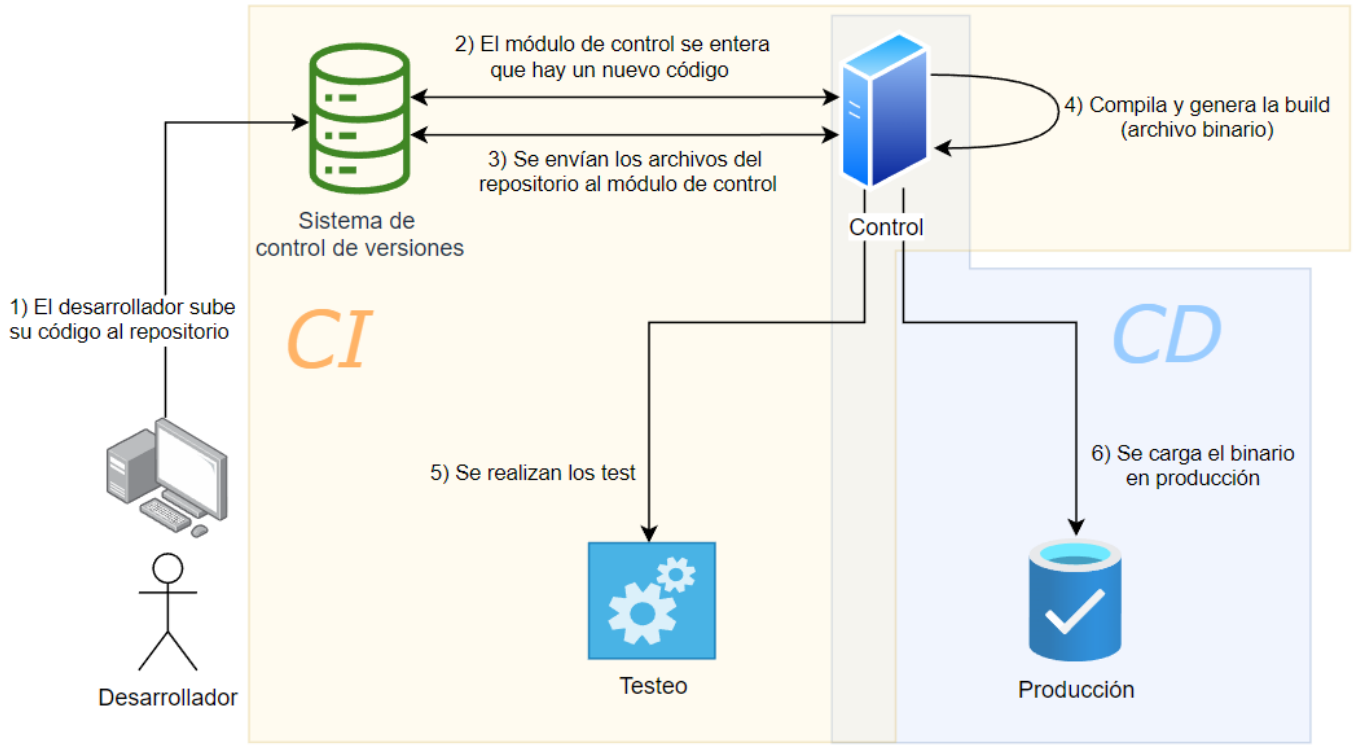

Figura 6. Flujo de trabajo dividiendo tareas de CI y CD.

Entrando un poco más en detalle, la Figura 7 muestra en profundidad las tareas que corresponden a la Integración Continua (CI) y la Figura 8 hace lo mismo con las tareas del Despliegue Continuo (CD).

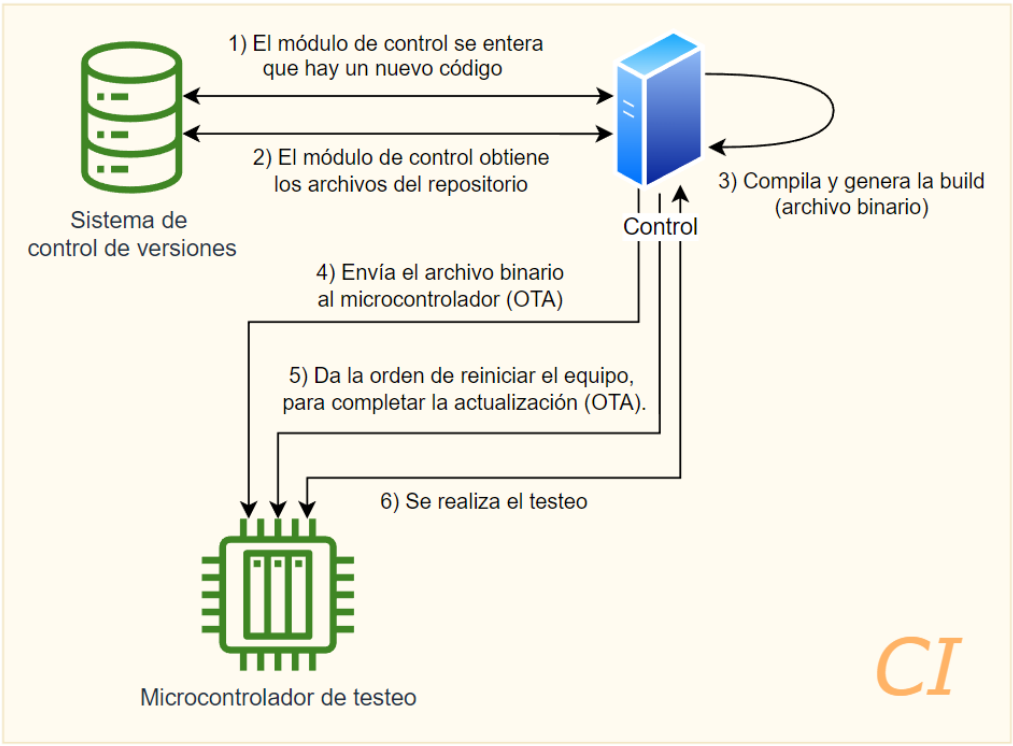

Figura 7. Proceso de CI.

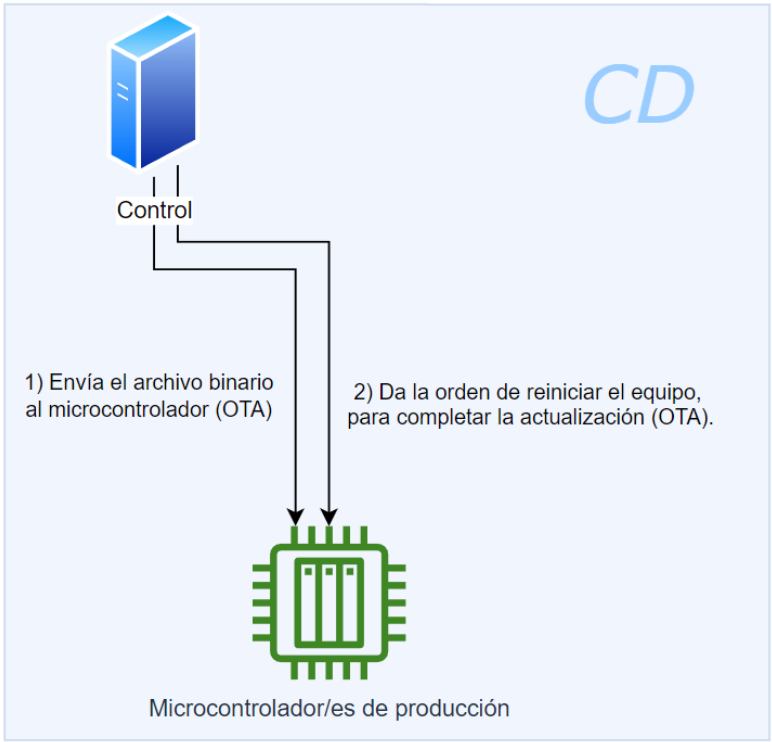

Figura 8. Proceso de CD.

Capítulo 5 - Implementación

En este capítulo se va a explicar cómo se intenta implementar todo lo planteado en la propuesta. Se va a recorrer el flujo de trabajo (Figura 6) dándole una entidad concreta a cada agente y detallando como se va a llevar a cabo cada uno de los pasos.

Antes de comenzar a recorrer el flujo, se considera apropiado explicar algunas de las elecciones de implementación. En primera medida, los microcontroladores, tanto para testeo como para producción, van a estar representados, cada uno, por un ESP32, cuyas características están detalladas en el marco teórico sobre microcontroladores. En términos generales, es un microcontrolador de 32-bit, con una potencia de procesamiento relativamente alta y conexión WiFi. Por otro lado, para simular un sistema en funcionamiento, se debe cargar a estos ESP32 con algún código que tenga habilitado OTA, para posteriormente poder actualizarlos de manera remota. En este caso, el programa elegido va a consistir en un servidor web con 4 endpoints:

- La ruta “/” va a funcionar como página principal o home, mostrando un HTML a elección.
- La ruta “/version” va a mostrar la versión actual del programa, que se configura dentro de una variable del código.
- La ruta “/test” está destinada a que el desarrollador la utilice para realizar los testeos que considere, por defecto devuelve un String prefijado.
- La ruta “/update” sirve para realizar las actualizaciones OTA, se explicará en detalle su funcionamiento posteriormente dentro de este capítulo.

Las 3 primeras rutas son provisionales, se usan como referencia de un posible desarrollo, pero podrían ser reemplazadas sin afectar en exceso el sistema. La elección del servidor web como programa del ESP tampoco tiene un motivo estricto, pero se eligió porque representa uno de los tipos de desarrollo más simples y flexibles, que puede ser adaptado fácilmente por cualquier programador.

Otro detalle a tener en cuenta es que el módulo de control estará representado por un servidor desarrollado sobre Python, el cual será el encargado de realizar múltiples tareas. La elección de Python como lenguaje de desarrollo es arbitraria, y puede ser reemplazada por cualquier otro.

Terminadas las aclaraciones, se puede comenzar a recorrer el flujo de trabajo. Para cumplir el primer paso, se debe utilizar un repositorio de código, en donde el desarrollador pueda subir los cambios del desarrollo. Por las herramientas que provee, que se describirán a la brevedad, se eligió utilizar GitHub. De aquí en adelante, hasta previo aviso, comienzan las tareas correspondientes a la Integración Continua (CI).

Una vez que el desarrollador sube su código al repositorio GitHub, el módulo de control debe enterarse de que hay una nueva versión que compilar. Este paso puede ser implementado de diferentes maneras:

1. La primera, y si se quiere, la menos refinada, es hacer polling sobre el contenido del repositorio. El polling consiste en chequear constantemente el contenido, con algún proceso que repita la revisión cada cierta cantidad de tiempo. Por ejemplo, se podría configurar un proceso que cada minuto corrobore si hay un nuevo commit dentro del repositorio y, en caso de encontrar uno, enviaría una notificación al módulo de control [52].
2. La segunda es utilizar algún tipo de herramienta orientada al CI/CD, como puede ser Jenkins [53]. Jenkins es un servidor que ayuda a realizar el ciclo de CI/CD, pero todavía no tiene mucha implicación en el mundo de los microcontroladores [54]. Sin embargo, puede cumplir la función de notificar al módulo de control cuando hay un cambio en el repositorio, lo cual bastaría para cumplir esta segunda tarea del flujo de trabajo.
3. La tercera es utilizar los webhooks que provee GitHub. Los webhooks se encargan de, cuando haya un evento específico en el repositorio, como puede ser una subida de código (push), notificar mediante un pedido HTTP POST a la dirección que se desee.

La alternativa elegida en este trabajo es la tercera, los webhooks de GitHub, y es una de las razones de por qué se eligió este repositorio de código. Tanto la alternativa de Jenkins como la de los webhooks son mejores que hacer polling, ya que trabajan de manera asincrónica, por lo que la notificación de un nuevo código se obtiene casi al instante de producido el evento.

En particular, se eligió utilizar los webhooks sobre Jenkins por 2 razones. La primera es que los webhooks son bastante fáciles de configurar, prácticamente solo se debe especificar a qué dirección se desea enviar la notificación y nada más, al final de este capítulo se explicará en detalle. En cambio, Jenkins lleva un poco más de trabajo y, además, es un servidor que debe correr sobre computadora, lo que va a consumir recursos que, si se utilizan los webhooks, estarían a cargo de GitHub. La ventaja de Jenkins, por lo general, es que ayuda al desarrollador en varios pasos del proceso de CI/CD, pero por la poca integración que tiene con los microcontroladores [54], el aporte sería el mismo que los webhooks. Pequeña aclaración, los webhooks se encuentran presente en la mayoría de sistemas de control de versiones avanzados, por lo que no es una elección que restringe el trabajo al uso de GitHub. Por dar un ejemplo, GitLab también provee esta forma de notificación [55].

Entonces, con la forma de notificación elegida, cuando el desarrollador suba un cambio al repositorio, GitHub enviará un pedido HTTP a la dirección que se le especifique, en este caso a la del módulo de control. En particular, el servidor va a tener un endpoint llamado /github-webhook, que será el que reciba la notificación del webhook. Una vez recibida la notificación, estaría completo el segundo paso del flujo de trabajo, quedando por delante la obtención de datos del repositorio.

Para obtener el código que subió el desarrollador, se va a hacer uso de la API que provee GitHub, la otra razón por la que se eligió este sistema. Sin embargo, la mayoría de los sistemas de control de versiones avanzados tienen una API, o forma alternativa, que permita la obtención de datos de sus repositorios, por lo que, de nuevo, esta decisión no restringe el trabajo al uso de GitHub de manera obligatoria. Continuando con la explicación, el módulo de control se encargará de mandar los pedidos necesarios para obtener el nuevo código del

microcontrolador y, una vez GitHub envíe los archivos correspondientes, se puede pasar al siguiente paso, la generación de la build.

Para la creación del binario, se deben compilar los archivos obtenidos en el paso anterior. De la manera tradicional, esto se hace con ayuda del IDE. En este trabajo, para replicar esa situación, se va utilizar Arduino CLI, la interfaz de línea de comandos de Arduino. El módulo de control se encargará de ejecutar en consola el comando correspondiente a la compilación de los archivos (Figura 9), obteniendo como resultado el archivo binario que representa la build del proyecto.

```
os.system("arduino-cli compile -b " + placa + " ./CodeFromGithub/otaesp/otaesp.ino -e  --clean")
```

Figura 9. Comando de Arduino CLI para compilar y generar el archivo binario correspondiente (línea 136)

Cómo último paso de esta etapa de CI, debe realizarse el testeo de la build. Cómo se explicó en repetidas ocasiones, la mejor forma de testear es sobre el hardware real, por lo que, antes de realizar ningún test, se debe cargar el archivo binario al microcontrolador de testeo. Para llevar a cabo esta actualización, se va a hacer uso de la tecnología OTA. Con ese objetivo, el ESP se conectará a la red WiFi que se le indique, medio por el que se transmitirán los datos para la actualización.

Hay diversas formas de implementar la actualización OTA. El primer intento se realizó con las librerías que provee Espressif, el creador del ESP, microcontrolador que se está utilizando, lo que corresponde a la forma más tradicional de realizar la actualización. Este intento, si bien funcionó en un principio, no fue exitoso. Con el correr del tiempo, surgió una actualización en las librerías que le agregaba seguridad y, desde ese momento, se necesitaba agregar una certificación al servidor que enviaba el binario al ESP, en nuestro caso el módulo de control, lo cual aumentaba la complejidad innecesariamente. Una alternativa para sortear esta restricción era subir el archivo al ESP directamente desde GitHub, pero era un movimiento no deseado. Por esta razón, se descartó la forma tradicional de realizar la actualización y, en su lugar, se comenzó a buscar librerías alternativas que la implementen.

Se probaron diferentes librerías, si bien algunas resolvían el problema, agregaban otras complicaciones que tampoco eran deseadas, hasta que se encontró una adecuada. En consecuencia, la actualización se llevará a cabo con ayuda de AsyncElegantOTA, una librería open source reconocida por Arduino [56]. Esta librería envuelve el código necesario para llevar a cabo la actualización, con el objetivo de simplificarlo, lo que resulta en un código mucho más simple y legible en la programación del microcontrolador. Más en detalle, esta librería se encarga de levantar un servidor web en el ESP y deja disponible un endpoint, el "/update", para recibir las actualizaciones. El tipo de petición que recibe es mediante el método POST, acompañado de 4 parámetros:

- "filename": este parámetro se debe acompañar del valor "firmware", ya que es lo que se desea enviar en este proyecto.
- "name": este parámetro se utiliza por cuestiones de compatibilidad, debe tener el mismo valor que el parámetro anterior.
- "file": en este parámetro se adjunta el archivo binario que se desea subir.
- "MD5": el valor de este parámetro debe ser el resultado de hashear con el algoritmo MD5 el archivo binario.

Este pedido post se puede realizar de la manera convencional o se puede hacer accediendo con el navegador al endpoint /update, utilizando una interfaz gráfica. En la Figura 10 se muestra un extracto de cómo se construye el pedido, adjuntando los parámetros señalados anteriormente y el archivo binario:

```
158     data = {'filename': 'firmware', 'name': 'firmware', 'MD5': md5}
162     files = {"file": open(path, "rb")}
165     post_resp = requests.post("http://" + ip + "/update", data=data, files=files)
```

Figura 10. Pedido POST para realizar actualización OTA.

Una vez que el ESP de testeo está corriendo el nuevo código, se pueden comenzar a realizar los test. En este proyecto, el testeo lo va a llevar a cabo el módulo de control, y consiste en que corrobore el funcionamiento de 2 endpoints:

1. Primero se va a testear el endpoint /version, el cual debe responder con la versión del sistema. El módulo de control enviará un pedido HTTP y comprobará que la respuesta esté expresada de la manera correcta, el formato debe ser “vX.X.X”, con X un número natural, por ejemplo, “v2.3.11”.
2. Como segundo test, se enviará un pedido a /test, el cual simplemente devuelve un string específico, por lo que el servidor debe comprobar que sea el string correcto.

Los test elegidos buscan ser genéricos, para así generar una estructura fácilmente modificable por cada desarrollador en su proyecto específico. Con ese objetivo en mira, todos los test que realice el servidor se modularizaron y están escritos en un archivo aparte, test.py. Por lo tanto, si se quiere testear algo diferente, no es necesario modificar el código del servidor que corresponde al módulo de control, simplemente se puede hacer modificando el archivo test.py.

Si ambos test resultaron exitosos, se finaliza la etapa de testeo y, con ello, el proceso de Integración Continua (CI). Si alguno de los test genera un error, el mismo se muestra en la consola del servidor y se detiene el proceso.

El último paso, correspondiente al Despliegue Continuo o CD, es realizar el deploy, el cual consiste en poner el nuevo programa en funcionamiento. En este caso, el ambiente de producción está representado por un único ESP, identificado en el flujo de trabajo cómo el microcontrolador de producción, pero podrían ser varios. En esta situación, tal como se hizo con el microcontrolador de testeo, el módulo de control se encargará de enviar el archivo binario generado en etapas anteriores al ESP de producción mediante OTA. En el caso de que hubiese más de un microcontrolador en producción, como es el caso de los sistemas distribuidos, el proceso sería el mismo con cada uno de ellos. Subido el código, el ambiente de producción estaría corriendo la nueva versión del programa, por lo que finaliza la etapa de deploy y, con ello, el ciclo de CI/CD.

Con el sistema en funcionamiento (Figura 11), cuando el programador suba su código a GitHub, verá reflejado ese código, ya testeado, en el ESP de producción, todo de manera automática.

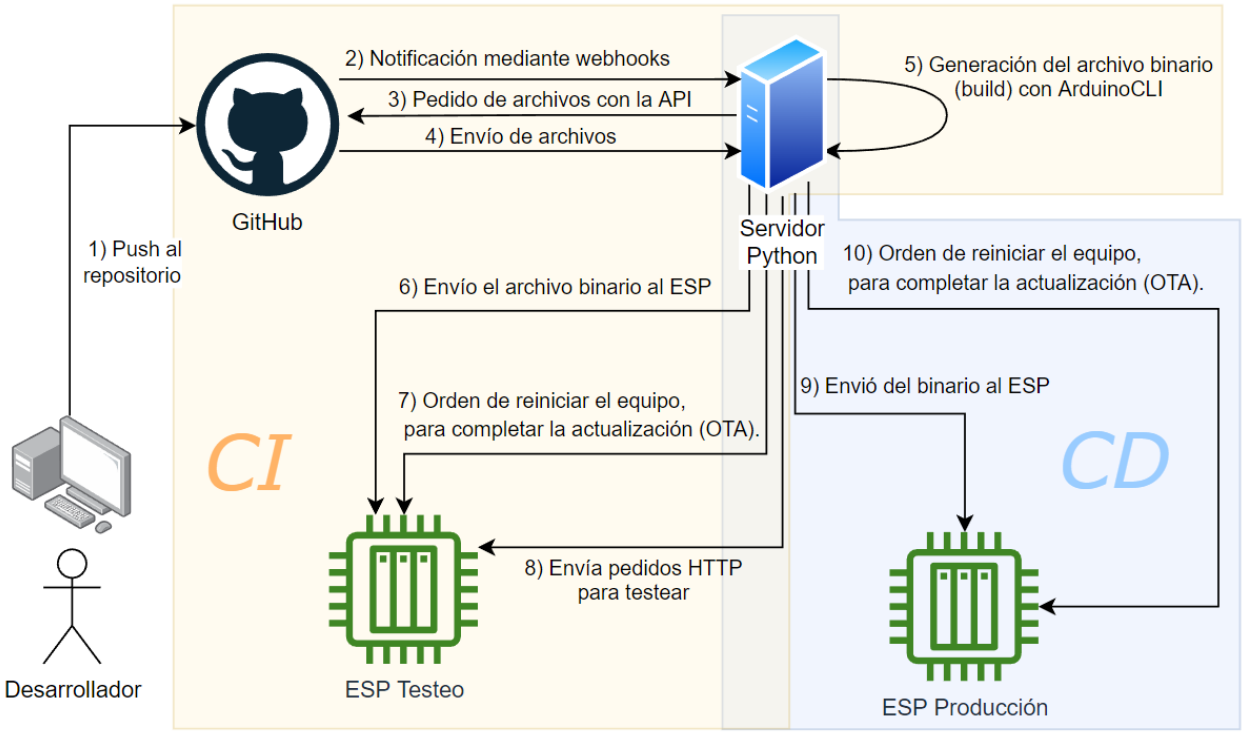

Figura 11. Flujo de trabajo de la implementación del sistema propuesto.

5.1 Configuración

Como ya se nombró, es necesario que GitHub notifique los cambios en el repositorio mediante los webhooks. Para configurar un webhook es necesario ir a “Settings” en el repositorio del proyecto, y dentro del menú desplegado a la izquierda, en la sección de “Code and automation”, aparece el botón que dice “Webhooks”. Al presionar esa opción, se listan los webhooks que tengamos configurados, en caso de no tener ninguno se puede presionar “Add webhook”, lo que abre una ventana que permite configurar uno nuevo. El webhook solicita la dirección URL a la que se debe comunicar y da la opción de configurar cuándo debe hacerlo, en el caso de este trabajo lo hará siempre que haya un push sobre el repositorio.

En una computadora personal, si va a ser quien corra el módulo de control, primero es necesario obtener la dirección IP, lo cual se puede realizar de diferentes maneras, una forma es mediante páginas online, como https://www.cual-es-mi-ip.net/ (consultado el 11/11/2022). Una vez obtenida la IP, es necesario elegir el puerto al que se va a comunicar, supongamos el 16000. Esa dirección + puerto es lo que se le tiene que brindar al webhook que se configura en GitHub. Dependiendo del proveedor de internet, es probable que sea necesario acceder al router y habilitar el Port Forwarding, conectando la dirección que ingresamos en el webhook, que es a la que se comunica GitHub, con la que posee el módulo de control (se puede encontrar el su código). Por último, al menos en el sistema operativo Windows, lo más probable es que sea necesario crear una excepción en el Firewall, para que no se rechace el pedido de GitHub.

Más detalles de configuración se encuentran explicados dentro del repositorio del proyecto, nombrado en el capítulo a continuación.

Capítulo 6 - Resultados

Los resultados de este proyecto se pueden encontrar en el siguiente repositorio: https://github.com/alejosanti/ESProject, el cual se divide principalmente en 2 partes.

Por un lado, está el código que va a utilizar el microcontrolador, tanto el de testeo como el de producción, cargado en el directorio “Arduino_code”. El programa del ESP está escrito en el archivo *otaesp.ino*, y dentro del mismo directorio se encuentra el archivo *index.h*, el cual contiene el HTML que el microcontrolador muestra en el endpoint “/”.

La otra parte del proyecto, contenido en el directorio “Server”, tiene el código necesario para levantar el módulo de control. Se listan brevemente las tareas que llevará a cabo:

1. **Recibir Webhook de GitHub**: utilizando los webhooks, GitHub puede notificar cuando se sube contenido en el repositorio. Por lo tanto, el trabajo del servidor comienza cuando recibe un pedido POST en el endpoint /github-webhook.
2. **Leer los archivos de GitHub**: el primer paso luego de recibir el webhook es buscar los archivos que corresponden al código del ESP en GitHub, contenidos en el directorio “Arduino_code”. Para esta tarea se hace uso de la API de GitHub.
3. **Hacer una copia local**: se escribe el contenido de los archivos en un directorio local, llamado “CodeFromGithub”.
4. **Crear archivo binario**: se crea el archivo binario a partir del código del microcontrolador. Para esto, se hace uso de interfaz de línea de comandos que provee Arduino, ArduinoCLI. Este archivo binario es lo que se debe subir al ESP.
5. **Subir binario al ESP de testeo**: como está explicado en la descripción de la librería AsyncElegantOTA, se manda un request POST particular al ESP, enviando el archivo binario correspondiente a la nueva actualización.
6. **Testear sobre el ESP de testeo**: para testear el servidor va a consultar diferentes endpoints disponibles que tiene el microcontrolador, corroborando que las respuestas sean las esperadas.
7. **Subir binario al ESP de producción**: si todos los test dieron resultados correctos, se sube el mismo binario que subió al ESP de testeo en el ESP de producción.

Con este servidor quedaría funcionando correctamente el flujo de trabajo descrito en la propuesta, por lo tanto, se pudo desarrollar un sistema que implementa las prácticas de CI/CD en el contexto de los microcontroladores. Esto significa que una persona que desarrolla sobre microcontroladores, utilizando esta herramienta, puede desvincularse de muchas de las adversidades que este tipo de desarrollo lleva aparejado. En primer lugar, el uso del sistema conlleva el uso de un repositorio de código, con todas las ventajas que el mismo trae aparejado. Además, como el desarrollador no debe encargarse de hacer la compilación del código de manera manual, puede prescindir de utilizar los IDEs como el de Arduino, que usualmente son los que se encargan de esta tarea, habilitándolo a utilizar IDEs mucho más avanzados, como los del desarrollo tradicional. Para testear, ya no necesita tener un acceso físico al dispositivo, sino que, se va a subir la nueva actualización de manera remota mediante OTA y los test van realizan de manera automática, solo se deben preconfigurar. La misma situación ocurre para el ambiente productivo, si el código pasa los test planteados, automáticamente se va a cargar en los microcontroladores que estén en producción, por más

que estén distribuidos geográficamente, solo deben asegurarse el poder establecer una conexión con el módulo de control.

Para demostrar pragmáticamente las ventajas del sistema planteado, se hará una comparación real, eligiendo como programa modelo un servidor web básico. Primero, se explicará el procedimiento del desarrollo clásico, ilustrado en el siguiente video de YouTube: https://youtu.be/TGOV-tXFH8s [57].

En el video, hay un microcontrolador conectado a la computadora del desarrollador, y el primer paso sería subir el código al microcontrolador de testeo para hacer las pruebas necesarios. Para eso, debe presionar el botón “Subir” en el IDE de Arduino, el cual desencadena la compilación del código. Esta compilación, al menos en el programa elegido (servidor web básico), tarda aproximadamente 35 segundos. Una vez compilado el código, el desarrollador tiene que esperar a ver un mensaje en la consola, como se puede apreciar en la Figura 12.

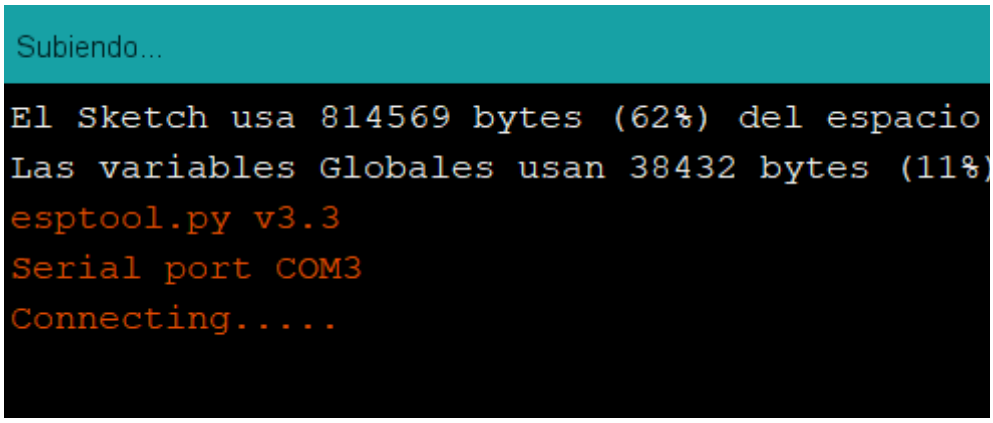

Figura 12. Estado de la consola antes comenzar la actualización.

Este mensaje le indica que tiene una ventana de, aproximadamente, 10 segundos para dejar presionado un botón en el microcontrolador, con el cuál se comienza a subir el nuevo software. Si no se presiona el botón tiempo, debe volver a comenzar el proceso y reiniciar la compilación. Si se presiona correctamente, el nuevo software comienza a cargarse en el microcontrolador (Figura 13), proceso que tarda, al menos en este programa, aproximadamente 1 minuto.

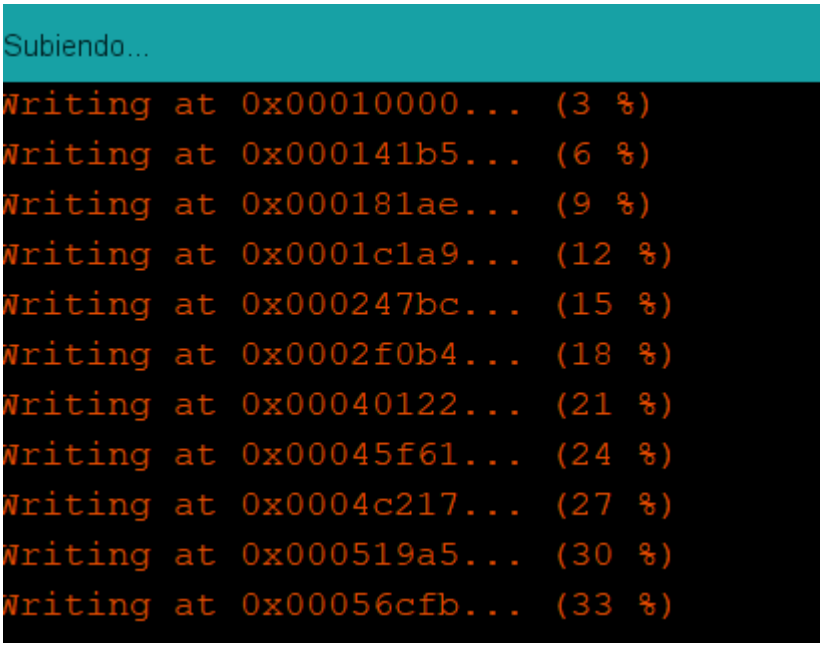

Figura 13. Estado de la consola al actualizar.

Una vez cargado el código en el microcontrolador de testeo, el desarrollador debe proceder a realizar las pruebas que considere necesarias, también de manera manual. Si se pasan los test correctamente, debe cambiar el microcontrolador que está conectado a la computadora y subir el código nuevamente, ahora al microcontrolador de producción. Esto significa realizar el proceso anterior nuevamente (compilación, presionar el botón y actualización), lo que conlleva una cantidad de tiempo considerable.

A modo de comparación, se explicará el proceso a realizar para obtener los mismos resultados, pero ahora haciendo uso del sistema desarrollado en esta tesis: https://youtu.be/t0aGn3nj774 [58].

En este video, el desarrollador tiene un microcontrolador conectado por WiFi, sin necesitar un acceso físico al mismo. El primer y único paso que debe hacer es subir la nueva versión del código al repositorio, que, teniendo la posibilidad de utilizar un IDE más avanzado, lo hace con ayuda de las herramientas que provee Visual Studio Code, pero también lo podría hacer por consola, de manera tradicional. El resto del proceso pasa inadvertido para él, a menos que desee mirar la consola del módulo de control, en donde se imprimen los pasos que se van realizando. Como pequeña aclaración, es importante señalar que, en el video, el servidor que representa al módulo de control está en la misma computadora del desarrollador, pero esto no es necesario, se podría correr en otra computadora y acceder a la consola, si así se lo quisiera, de manera remota.

El resultado de este proyecto no solo desvincula al programador de realizar diferentes tareas tediosas, sino que el proceso además es más rápido. Como se puede observar, en una misma situación, el desarrollo clásico requiere una participación semiactiva del desarrollador para completar el ciclo del software, tardando aproximadamente 4 minutos, y con la necesidad de poseer un acceso físico a los microcontroladores. En cambio, con el sistema planteado, cada actualización conlleva aproximadamente 2 minutos, la mitad, y el programador está exento de cualquier tipo de intervención durante el mismo. Se agrega, además, la ventaja de que el

código queda almacenado en un controlador de versiones, de que se tiene la facilidad para desarrollar en cualquier IDE y de que no hay necesidad de acceder físicamente a los dispositivos.

Capítulo 7 - Conclusiones y trabajos futuros

Como se planteó al principio del proyecto, el uso de microcontroladores está creciendo, se usan en diferentes electrodomésticos con el IoT, en sistemas distribuidos como flotas de vehículos, en procesos productivos de diferentes industrias, dentro de los automóviles, etcétera. Esta creciente demanda de uso genera, por lo tanto, una creciente demanda de desarrollo sobre estas plataformas, y se ha visto que desarrollar para sistemas distribuidos puede generar complicaciones. Parte de estas complicaciones son la falta de un ambiente de desarrollo avanzado, la ausencia de simuladores en donde testear o la necesidad de acceder físicamente a los dispositivos (si no se usa OTA).

Con el fin de atenuar todas estas adversidades, se planteó integrar estos desarrollos con una tecnología moderna, como es el CI/CD, haciendo uso de las actualizaciones OTA. Para llevar a cabo esta integración, se utilizó un servidor, desarrollado sobre Python, quién es el encargado de comunicarse con diferentes agentes y procesar diferentes datos, con el fin de automatizar todas aquellas partes del desarrollo que no necesitan la intervención constante de una persona, ahorrando una gran cantidad de tiempo y esfuerzo al programador.

El sistema resultante tiene varias virtudes que son de provecho para el desarrollador. Utiliza un repositorio de código, con las ventajas que eso trae aparejado, y permite que solo con un push en el mismo se vean reflejados los cambios en el microcontrolador del ambiente productivo. No solamente ahorra muchos pasos tediosos, lo que libera al programador de estar realizando todo el procedimiento manual, sino que también logra que el resultado esté en producción en una menor cantidad de tiempo. Además, la forma de actualización que utiliza es OTA, lo que le da la capacidad al desarrollador de prescindir del acceso físico a los dispositivos.

Si bien se finaliza el proyecto con un sistema completamente funcional, hay partes sobre las que se puede seguir trabajando:

1. **Agregar seguridad**: el principal agregado que se le puede hacer al proyecto es incluir una capa de seguridad. Como se ha explicado, para generar una nueva actualización en el ESP, simplemente se requiere enviar un pedido POST a un endpoint específico, por lo si el sistema se utiliza en un ambiente poco seguro, se podría vulnerar sin demasiada complejidad. Una buena referencia a seguir es la que propone Iserman [31] cuando describe SOTA (Safe & Secure OTA), básicamente una combinación de OTA con diferentes aspectos criptográficos, que le agregan esta capa de seguridad de gran importancia.
2. **Utilizar GitLab**: como el proyecto no depende de un controlador de versiones en particular, se podría tener una versión ya preparada para cada uno de los controladores de versiones más populares, GitHub y GitLab. La versión de GitHub es la que se eligió desarrollar en este proyecto, por lo que quedaría pendiente el desarrollo de una versión utilizando GitLab, buscando una manera de reemplazar la API y los webhooks que provee GitHub.
3. **Mayor independencia**: en el proyecto se utiliza la librería AsyncElegantOTA para simplificar el proceso de actualización, pero es prescindible. Se podría realizar una versión que no utilice la librería, haciendo uso solo de las herramientas que provee el ESP, para lograr una mayor independencia.

Referencias

[1] M. A. R. Trejo, *"Las 7 tendencias que marcarán el crecimiento del IoT en los próximos años"*. 2022. https://www.linkedin.com/pulse/las-7-tendencias-que-marcaran-el-crecimiento-del-iot-en-rivera-trejo/?originalSubdomain=es (accedido el 16/11/2022)

[2] J. Pizarro Peláez, *"Internet de las cosas (IOT) con ESP. Manual práctico."*. Editorial Paraninfo, 2020.

[3] L. D. Candia, A. S. Rodríguez, N. Castro, P. A. Bazán, V. M. Ambrosi y F. J. Díaz, *"Mejoras en maquinaria industrial con IoT: hacia la industria 4.0"*, en *XXIV Congreso Argentino de Ciencias de la Computación (CACIC)*, La Plata, Argentina. 2018. http://sedici.unlp.edu.ar/handle/10915/73348 (accedido el 13/11/2022)

[4] M. R. van Steen y A. S. Tanenbaum, *"Distributed systems"*, 3ª ed. Pearson, 2017.

[5] N. Forsgren, D. Smith, J. Humble, y J. Frazelle, *"Accelerate State of DevOps 2019"*. 2019. https://services.google.com/fh/files/misc/state-of-devops-2019.pdf (accedido el 16/11/2022)

[6] GitLab B.V., *"What is CI/CD?"*. https://about.gitlab.com/topics/ci-cd/ (accedido el 3/11/2022)

[7] IEEE, *"IEEE/ISO/IEC 15288-2015"*. 2015. https://standards.ieee.org/ieee/15288/5673/ (accedido el 10/11/2022)

[8] X. M. Prieto, *"Un viaje a la historia de la informática"*. Universidad Politécnica De Valencia, 2016.

[9] F. Valdés y R. P. Areny, *"Microcontroladores Fundamentos y Aplicaciones con PIC"*. Marcombo, 2007.

[10] A. S. Tanenbaum y T. Austin, *"Structured Computer Organization"*, 6ª ed. Pearson, 2012.

[11] Texas Instruments Incorporated, *"Microcontrollers (MCUs)"*. https://www.ti.com/microcontrollers-mcus-processors/microcontrollers/overview.html (accedido el 5/11/2022)

[12] Microchip Technology Inc., *"Microcontrollers (MCUs)"*. https://www.microchip.com/en-us/products/microcontrollers-and-microprocessors (accedido el 5/11/2022)

[13] Renesas Electronics Corporation, *"μPD67, 67A, 68, 68A, 69 4-bit single-chip microcontroller for infrared remote control transmission"*. 2016. https://web.archive.org/web/20160106205219/http://documentation.renesas.com/doc/DocumentServer/U14935EJ2V1DS00.pdf (accedido el 16/11/2022)

[14] B. Amos, *“Hands-On RTOS with Microcontrollers: Building real-time embedded systems using FreeRTOS, STM32 MCUs, and SEGGER debug tools”.* Packt Publishing, 2020.

[15] Espressif Systems, *“ESP32”.*
https://www.espressif.com/en/products/socs/esp32 (accedido el 5/11/2022)

[16] Espressif Systems, *“ESP8266”.*
https://www.espressif.com/en/products/socs/esp8266 (accedido el 5/11/2022)

[17] Espressif Systems, *“Espressif Home Page”*.
https://www.espressif.com/ (accedido el 5/11/2022)

[18] V. A. Asanza*, "Especificaciones del módulo ESP32".* 2021.
https://vasanza.blogspot.com/2021/07/especificaciones-del-modulo-esp32.html (accedido el 8/11/2022).

[19] J. G. Carmenate, *“ESP32 Wifi y Bluetooth en un solo chip”*.
https://programarfacil.com/esp8266/esp32/ (accedido el 16/11/2022)

[20] Amazon Web Services, Inc., *“FreeRTOS: eal-time operating system for microcontrollers”.*
https://www.freertos.org/ (accedido el 16/11/2022)

[21] Espressif Systems, *“ESP32 Series Datasheet”.*
https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
(accedido el 5/11/2022)

[22] Espressif Systems, *“ESP32 Technical Reference Manual”.*
https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf (accedido el 5/11/2022)

[23] Arduino, *"Arduino - Home".*
https://www.arduino.cc/ (accedido el 7/11/2022).

[24] Arduino, *"Arduino Documentation"*.
https://docs.arduino.cc/ (accedido el 8/11/2022).

[25] Arduino, *"Arduino IDE 2 Tutorials".*
https://docs.arduino.cc/software/ide-v2 (accedido el 8/11/2022).

[26] Arduino, *"Arduino CLI".*
https://arduino.github.io/arduino-cli/0.27/ (accedido el 8/11/2022).

[27] M. M. Villegas, Y H. Astudillo, *“OTA Updates Mechanisms: A Taxonomy and Techniques Catalog”*, en *Simposio Argentino De Ingeniería De Software (JAIIO)*. 2020.
https://49jaiio.sadio.org.ar/pdfs/asse/ASSE%2008.pdf (accedido el 16/11/2022)

[28] F. G. Tinetti, *"Programming (Flashing) and Using the ESP8266-[01/07] with Arduino UNO (Update 2022)"*. III-LIDI, Facultad de Informática, UNLP, 2022. http://fernando.scienceontheweb.net/links/embed-rt/2022-ESPFlash.pdf (accedido el 16/11/2022)

[29] Components101*, "ESP8266 - WiFi Module".* 2018. https://components101.com/wireless/esp8266-pinout-configuration-features-datasheet

[30] OLIMEX Ltd., *"Changing the Modes of MOD-WI-FI-ESP8266-DEV. Reference. Revision B".* 2018. https://www.olimex.com/Products/IoT/ESP8266/MOD-WI-FI-ESP8266-DEV/resources/MOD-WI-FIESP8266-DEV_jumper_reference.pdf (accedido el 3/11/2022).

[31] R. Isermann, *"Fahrerassistenzsysteme 2016: Von Der assistenz Zum automatisierten"* en *2ª Conferencia ATZ Internacional*. Springer Vieweg, 2018.

[32] M. Schwartz, *"ESP8266 internet of things cookbook"*. Packt Publishing, 2017.

[33] Red Hat Inc., *"¿Qué son la integración/distribución continuas CI/CD)?"*. 2018. https://www.redhat.com/es/topics/devops/what-is-ci-cd (accedido el 3/11/2022).

[34] J. Davis y D. Ryn, *"Effective DevOps: Building a Culture of Collaboration"*. Affinity. Shroff Publishers & Distributors, 2016.

[35] T. Rangnau, R. V. Buijtenen, F. Fransen y F. Turkmen, *"Continuous Security Testing: A Case Study on Integrating Dynamic Security Testing Tools in CI/CD Pipelines",* en *IEEE 24th International Enterprise Distributed Object Computing Conference (EDOC)*. 2020. https://doi.org/10.1109/edoc49727.2020.00026 (accedido el 16/11/2022)

[36] K. Beck y C. Andres, *"Extreme Programming Explained: Embrace Change"*, 1ª ed. Addison-Wesley Professional, 1999.

[37] M. Fowler, *"Continuous Integration".* 2006. https://martinfowler.com/articles/continuousIntegration.html (accedido el 16/11/2022)

[38] A. S. Tanenbaum, *"Sistemas Operativos Distribuidos"*. Prentice Hall & IBD, 1996.

[39] J. Joyce, G. Lomow, K. Slind y B. Unger, *"Monitoring distributed systems",* en *ACM Transactions on Computer Systems*. Association for Computing Machinery (ACM), 1987. https://dl.acm.org/doi/pdf/10.1145/13677.22723 (accedido el 16/11/2022)

[40] L. M. Vaquero, L. Rodero-Merino, J. Caceres y M. A. Lindner, *"A Break in the Clouds: Towards a Cloud Definition"*. ACM Computer Communications Review, 2008.

[41] J. A. Stankovic, *"Real-Time and embedded systems"*, en *ACM Computing Surveys, Vol. 28, No. 1*, 1996.

[42] A. Burns y A. Wellings, *"Real-Time Systems and Programming Languages: Ada, Real-Time Java and C/Real-Time POSIX"*, 4ª ed. Pearson Education Canada, 2009.

[43] G. C. Buttazzo, *"Hard RealTime Computing Systems"*, 3ª ed. Springer, 2011.

[44] H. Kopetz, *"Real-Time Systems, Design Principles for Distributed Embedded Applications"*, 2ª ed. Springer, 2011.

[45] P. A. Laplante y S. J. Ovaska, *"Real-Time Systems Design And Analysis. Tools for the Practitioner."*, 4ª ed. Wiley-IEEE Press, 2012.

[46] D. Otero Gutiérrez, *"Desarrollo de una aplicación web para control de versiones de software"*. Universidad Carlos III de Madrid, 2011. http://hdl.handle.net/10016/11936 (accedido el 5/11/2022)

[47] L. G. Lianet, Y. Medina Colina y N. Moreno Lemus, *"Nuevo software Controlador de Versiones para el Polo de Bioinformática"*. Universidad de las Ciencias Informáticas, 2009. https://repositorio.uci.cu/jspui/handle/ident/TD_2529_09 (accedido el 16/11/2022)

[48] Microsoft, *"¿Qué es el control de versiones?*". 2022. https://learn.microsoft.com/es-es/devops/develop/git/what-is-version-control (accedido el 17/11/2022)

[49] Git-scm.com, *"Git - Book"*. https://git-scm.com/book/es/v2 (accedido el 9/11/2022).

[50] Atlassian, *"Qué es Git | Atlassian Git Tutorial"*. https://www.atlassian.com/es/git/tutorials/what-is-git (accedido el 9/11/2022).

[51] GitHub, Inc, *"GitHub"*. https://github.com/ (accedido el 12/11/2022)

[52] DAVIS, LLC., *"Definition of polling"*. https://www.pcmag.com/encyclopedia/term/polling (accedido el 22/11/2022)

[53] Jenkins, *"Jenkins Home"*. https://www.jenkins.io/ (accedido el 22/11/2022)

[54] Jenkins, *"Jenkins in the Embedded World"*. https://www.jenkins.io/solutions/embedded/ (accedido el 22/11/2022)

[55] GitLab B.V., *"Webhooks"*. https://docs.gitlab.com/ee/user/project/integrations/webhooks.html (accedido el 22/11/2022)

[56] Arduino, *"AsyncElegantOTA – Arduino Reference"*. https://www.arduino.cc/reference/en/libraries/asyncelegantota/ (accedido el 13/11/2022)

[57] A. A. Santi, *“Desarrollo tradicional sobre microcontroladores (ESP32)”*. 2022. https://www.youtube.com/watch?v=TGOV-tXFH8s (accedido el 13/11/2022)

[58] A. A. Santi, *“CI/CD sobre microcontroladores (ESP32)”*. 2022. https://youtu.be/t0aGn3nj774 (accedido el 13/11/2022)