ISBN 978–987–633–574–4

Entorno de contenedores con emuladores de sistemas embebidos STM32

Esteban Carnuccio[1], Waldo Valiente[1], Mariano Volker[1], Raúl Villca[1], Matías Adagio[1]

[1] Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas
Florencio Varela 1903 - San Justo, Argentina
{ecarnuccio, wvaliente, mvolker, rvillca, maadagio}@unlam.edu.ar
www.unlam.edu.ar

Resumen. Ante la gran importancia que han tenido los sistemas embebidos, en los últimos años, debido al auge de Internet de las Cosas. Resulta de vital importancia conocer y probar el hardware antes de adquirirlo, para saber si cumple las necesidades de un proyecto determinado. En ese contexto, esta investigación se centró en la creación de un entorno automatizado de emulación, que permita probar rápida y fácilmente determinadas placas de desarrollo, sin necesidad de adquirir el hardware físico. Con esa premisa, se desarrolló un entorno dentro de un contenedor Docker, que permite realizar la emulación de determinadas placas de la familia STM32 a través del programa Qemu, listo para funcionar. De esta forma se podrá realizar distintas pruebas, sin la necesidad de realizar una tediosa configuración e instalación de los componentes y las dependencias.

Palabras Clave. Docker, Emulación, Internet de Las Cosas, Qemu, Sistemas Embebidos, STM32.

1 Introducción

En los últimos años creció de forma exponencial la tecnología de Internet de las Cosas. El término IoT toma relevancia cuando se superó la cantidad de dispositivos conectados a internet, que el número de personas que existían en el mundo en ese momento. Según las proyecciones de [1], se estima que en el año 2025 habrá aproximadamente cien mil millones de sistemas embebidos conectados, que transmitirán los datos de sus sensores para ser procesados en servidores externos, a través de internet. En este sentido, el tiempo y los costos que incurren para configurar un entorno de trabajo de software encargado de construir el sistema embebido, se vuelve un factor importante. Ya que para determinar si la placa de desarrollo puede cumplir con las necesidades, es necesario adquirir la placa físicamente, con el agregado de la curva de aprendizaje de su utilización. Muchas veces se trabaja sobre un producto que, en una etapa avanzada de un proyecto, se descubre que no es la indicada para cumplir con sus requerimientos. Como consecuencia, se debe adquirir un nuevo sistema embebido, que pueda cumplir con sus objetivos, produciendo así atrasos y aumento en los costos.

Para tratar de minimizar estos riesgos existen simuladores y emuladores de sistemas embebidos, como *Thinkercad* y *Proteus*. Pero estos presentan ciertas limitaciones de uso, que dificultan su utilización en los distintos proyectos IT [2]. Por ese motivo en esta investigación se desarrolló un entorno de integración automatizado a través de contenedores Docker, que permiten ejecutar programas en distintas placas STM32 emuladas a través de Qemu. De forma tal que permita realizar rápidamente distintas pruebas de aprendizaje en el entorno y el sistema embebido. Estos ejemplos a su vez funcionan en el embebido real.

En este documento, inicialmente se describe brevemente la comparación con otros entornos. Posteriormente se brinda una introducción al funcionamiento del contenedor de Docker y su registro de contenedores llamado Docker Hub. Luego, se detalla la forma en que se implementó y configuró el entorno de trabajo utilizando Qemu, dentro de Docker para los sistemas embebidos seleccionados. Finalmente, se detalla los ejemplos de código que se pueden ejecutar dentro del entorno.

2 Trabajos relacionados

Esta investigación se sustentó en diferentes trabajos relacionados sobre el emulador de Qemu para diferentes sistemas embebidos. A continuación, se realiza un repaso de los principales. Existen desarrollos como [3], donde se presenta una extensión de Qemu para integrarlo con la aplicación Eclipse. Esta debe ser instalado manualmente por el usuario y es un poco complicada su configuración. Además, solo se explica el ejemplo de encendido y apagado del led de testeo de la placa, mejor conocido como "*Blinky Led*". Sin embargo, este no ofrece explicación de cómo se deben emular otros sensores, actuadores y componentes del embebido.

Por otro lado, existe el proyecto realizado por *Beckus* [4]. El cual presenta su propia versión adaptada del código fuente de Qemu, que a su vez fue modificada por otros autores, como se describen en [5]. Pero para lograr utilizarlos, es difícil hacerlos funcionar correctamente, ya que se debe hacer la instalación y compilación de forma manual. No obstante, el proyecto de *Beckus* ofrece la forma de crear un contenedor Docker, pero no se logró hacerlo funcionar.

En [6] se menciona el proyecto de *Pebble*, que es una adaptación de *Beckus*, también es difícil su configuración e instalación. Pero es un proyecto en proceso, que no registra cambios presentes de actualización.

Finalmente, en el registro de contenedores de Docker[7], se puede descargar una imagen que emplea el proyecto *Beckus*, pero este no permite ejecutar los ejemplos. Además, no posee la documentación detallada con las formas en que se deben ejecutar dichos ejemplos. Tampoco tiene habilitado el ingreso por protocolo *SSH* dentro del contenedor, para su utilización.

Para subsanar las falencias descritas de los proyectos antes expuestos, en este trabajo de investigación se detalla cómo se armó el entorno contenido en una imagen Docker, en el que se modificó y adaptó el proyecto de *Beckus*. Obteniendo así un entorno de emulación bien documentado y funcional. Para ello se describen los distintos cambios realizados y se comentan brevemente los distintos tutoriales explicativos, para la

emulación de los sensores y actuadores en ciertas placas de desarrollo de la familia de microcontroladores STM32. Todo esto se hizo de forma automatizada, para que se pueda probar el mismo programa, que funciona en forma física, dentro del contenedor en forma emulada.

3 Desarrollo

3.1 Docker

Como se comenta en [8], Docker es una de las herramientas más populares en estos días en el mundo de la tecnología (IT). Básicamente, hay dos corrientes principales en el paradigma de Docker. El Primero, la plataforma Docker es de código abierto, esto permite que se equipe continuamente con nuevas características y funcionalidades relevantes. Siendo aprovechada no solo por programadores, sino también por equipos operativos de IT. La segunda tendencia es la adopción sin precedentes de la tecnología de contenedores, que es complementada por varios proveedores de soluciones y servicios de IT en todo el mundo. Estos dos permiten una mayor simplicidad en el desarrollo de aplicaciones, gracias a la implementación automatizada y acelerada de contenedores Docker. Siendo ampliamente promocionada como la clave diferenciadora del éxito sin precedentes de este paradigma.

3.2 Docker Hub

Docker Hub[1] es un registro de contenedores mantenido por Docker Inc. En el repositorio se encuentran las principales imágenes oficiales, como por ejemplo *Busybox*, Sistemas operativos como Ubuntu o Windows, incluso programas ya empaquetados como Apache, Node.js o WordPress entre otras. También permite a cualquier usuario crear una cuenta en forma gratuita y subir sus propias imágenes [9]. La protección de seguridad brindada es muy simple. Solo los propietarios de las imágenes publicadas y los usuarios habilitados, pueden realizar cambios en ellas. Además, tiene un sistema de puntuación por estrellas, similar al utilizado en repositorio GitHub o incluso el que utiliza Android en sus aplicaciones. Esto permite en el caso de imágenes con similares funcionalidades, seleccionar a la imagen mejor posicionada.

El repositorio de Docker permite probar nuevas versiones de las aplicaciones publicadas, o buscar nuevas aplicaciones que sirvan para un propósito dado. Las imágenes de Docker son una forma fácil de experimentar sin interferir la configuración actual de la computadora, ni aprovisionar una máquina virtual y no tener que preocuparse por los pasos de instalación. Además, permite la centralización en un único canal, facilitando compartir públicamente la imagen entre diferentes integrantes [10].

3.3 Imagen de Docker Creada

Acorde a los objetivos antes mencionados, se creó una imagen Docker que ya dispone el entorno utilizable, evitando el proceso de instalación y configuración de las herramientas necesarias para su funcionamiento. De manera tal, que el desarrollador

[1] URL Docker Hub: http://hub.docker.com

pueda de forma rápida y sencilla emular sus programas para STM32, en un entorno a través de Qemu. Para esto, este trabajo se basó en el proyecto de *Beckus* [4], adaptándolo a los objetivos planteados. En este sentido, se modificó el código fuente de Qemu, para que pueda emular funcionalmente las placas *Stm32-p103*, *Stm32-Maple* y *Stm32-f103c8t6* (conocida como *BluePill*). Siendo publicado en un repositorio propio de Github, junto con los códigos modificados de los ejemplos de las placas antes mencionadas. Por esa razón, la siguiente es la dirección web de dicho repositorio.

https://github.com/soaunlam2021/soa-entorno-integracion **(1)**

Este se encuentra estructurado en tres directorios, de la siguiente manera:

- **Qemu_stm32:** Este directorio contiene el código fuente del emulador Qemu, que fue adaptado para esta investigación.
- **Stm32_demos:** Contiene diversos ejemplos de código fuente que fueron adaptados, para poder ser ejecutados en la emulación y en el hardware real de las placas mencionadas. A su vez contiene información acerca del hardware de dichos SE.
- **Workflows:** En donde se encuentra la configuración de la generación automática de la imagen en Docker Hub.

El método que se utiliza para construir la imagen de Docker de forma automática es a través del archivo *docker-image.yml.* Este archivo se configuró, para que se ejecuten automáticamente ciertos comandos, que se encuentran indicados dentro un archivo llamado *Dockerfile.* Esta secuencia de comandos, son ejecutados dentro de una máquina virtual (VM) que se encuentra en los servidores de Github, y son accionados cada vez que se actualiza el repositorio. Una vez que dentro de la VM se genera la imagen de Docker, automáticamente el archivo *yml* la carga y la publica dentro del registro de contenedores de Docker Hub. Para que de esta forma cualquier persona pueda descargarla y utilizarla fácilmente. Por consiguiente, la imagen publicada en esta investigación se puede encontrar desde línea de comando de la siguiente manera, mediante el nombre *soaunlam/qemu_stm2*:

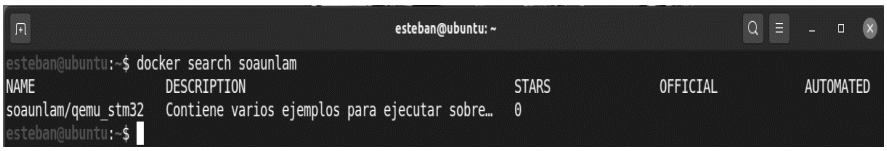


Fig. 1 Búsqueda de la Imagen Docker generada

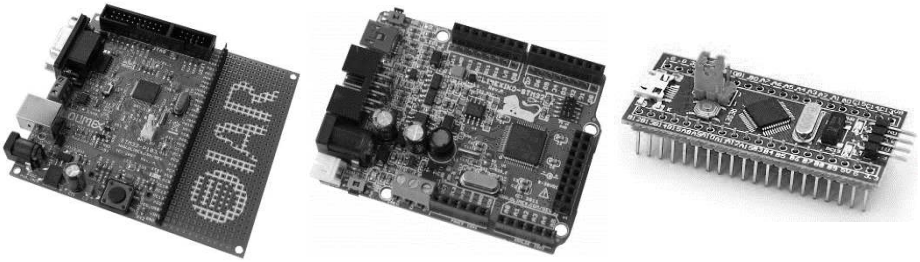
La ventaja de haber utilizado los *workflows* de Github, a través del archivo *yml,* es que la creación de la imagen se realiza en los servidores de Github y no en la máquina local. Con lo cual, esto nos evita tener que realizar el trabajo manualmente y el tiempo de procesarlo localmente.

3.4 Contenido del Archivo Dockerfile

Los archivos *Dockerfiles* permiten generar imágenes personalizadas. En estos archivos se especifican los comandos, en forma de meta instrucciones, que Docker interpretará para construir la imagen deseada. El archivo Dockerfile correspondiente a este trabajo se encuentra publicado en el repositorio de esta investigación. El contenido de dicho archivo fue organizado, para usar la menor cantidad de capas resultantes que conforman la imagen, como es explicado en la sección de mejores prácticas [10]. Para ello en el Dockerfile, se especifica como capa base a la versión adaptada del Sistema Operativo Ubuntu 20.04. Se consideró conveniente, utilizar una versión determinada y no la última existente, debido a que entre distintas versiones pueden variar sus dependencias, generando una imagen de Docker defectuosa. Luego se instalaron las dependencias y bibliotecas de Linux que necesita Qemu, junto con programas adicionales. Algunos de ellos son *apt-utils, gcc-arm-none-eabi, gcc, python2.7, pkgconf, git, make, libglib2.0-dev, libpixman-1-dev,* entre otros. Es importante mencionar que, para el correcto funcionamiento de algunos ejemplos, fue necesario instalar las dependencias *open-ssh* y *net-tools,* dado que estos deben ser ejecutados en parte a través de terminales ssh. Una vez instaladas todas las dependencias, se borra la cache utilizada para su instalación, de manera de poder liberar espacio en la imagen generada. Luego se descarga el repositorio de esta investigación desde Github (1). Posteriormente, dentro se configura y compila el código de Qemu, que se encuentra dentro del directorio *Qemu_stm32*. A su vez también se compilan y se generan los archivos binarios de los distintos ejemplos. Seguidamente se configura la imagen para que el usuario pueda acceder al contenido de este a través de protocolo SSH. Para ello se modifican los archivos *sshd_config* y *.bashrc* para ello. De esta forma se genera la imagen de Docker con todo configurado e instalado, para que el desarrollador pueda ejecutar ejemplos de código emulados en Qemu rápidamente. También para que tenga la posibilidad de modificar su código accediendo a estos de forma externa, a través de SSH. De tal manera que pueda acceder a los archivos contenidos dentro de la imagen, a través de distintos programas, como por ejemplo editores de código.

3.5 Diferencias entre las placas soportadas por la imagen de Docker

Las tres placas *Stm32-p103*, *Stm32-Maple* y *Bluepill,* Fig. 2 (a, b y c), pertenecen a la misma serie denominada “convencional” de microcontroladores de STM32F1 con 32 bits. Esta arquitectura se encuentra bien equilibrada y se adapta a las necesidades básicas que se esperan en los mercados de consumo, donde las limitaciones de costos y el tiempo de comercialización son esenciales. Además, están diseñadas para responder a los requisitos de forma simple, robusta y con larga vida de utilidad.



(a) Stm32-p103 (b) Stm32-Maple (c) BluePill

Fig. 2 Modelos de placas soportados por esta investigación

Estas placas de desarrollo tienen de similitud, el voltaje de trabajo (2.0V a 3.6V), su rango de temperaturas (-40 °C a 105 °C) y su velocidad de funcionamiento (hasta 72MHz). No obstante, sus diferencias radican en su tamaño y sus interfaces de entrada y salida. La placa Stm32-p103 [11], tiene las dimensiones de 100 x 90 mm. Además, internamente posee un botón y dos LEDs, uno de estado y otro de encendido. Como interfaces externas posee un conector JTAG, 3 puertos USART, 2 conectores SPI, 2 del tipo I2C, un puerto CAN, un conector USB y 2 conectores ADC. Mientras que la placa Stm32-Maple [12],posee las dimensiones de 69 x 54 mm. A su vez, internamente posee 2 botones, uno de reset y otro programable. Además posee 3 LEDs, dos de estado y uno de encendido. Como interfaces externas tiene un pin de conexión SWD, un conector de UEXT, otro de PWR JACK y distintos puertos CON1-POWER, CON2-ANALOG, ON3_DIGITAL y CON4 -DIGITAL. Además, posee un conector LI BAT, USB, un conector para tarjeta SD/MMC y un puerto CAN. Por otro lado, la placa BluePill [13]es la más pequeña de todas, con tan solo 23 x 53 mm. Además, internamente posee un botón de reset y contiene dos LED, uno de estado y otro de encendido. Adicionalmente, como conexiones externas posee un puerto USB, un pin de conexión SWD, 2 conectores del tipo I2C, 2 de SPI, un puerto CAN, un conector JTAG y 2 ADC.

3.6 Código fuente del contenido de la Imagen

Como se mencionó anteriormente, este trabajo se basó en el proyecto de *Beckus* y se lo adaptó para que pueda funcionar correctamente, en las emulaciones de las placas previamente mencionadas. En este apartado se describen brevemente las modificaciones que se debieron realizar en dicha adaptación. Estos ajustes fueron tanto en el código fuente base de Qemu y en los ejemplos. Dado que estos fueron creados para funcionar emulando solamente la placa *stm32-p103*, por lo que no estaban preparados para funcionar en su totalidad en las placas *stm32-Maple* y *BluePill*. Esto se debió principalmente a que los microcontroladores de esas placas poseen diferentes configuraciones internas. Por ese motivo en el repositorio de git (1), se crearon tres subdirectorios diferentes. Los cuales contienen los ejemplos de código fuente de programas, para ejecutar en cada una de las placas mencionadas. Ya que entre ellos presentan pequeñas modificaciones que lo hacen funcional. En este sentido, una de las adaptaciones que se debió

realizar, consistió en la asignación de la USART[2], la cual es diferente dependiendo del hardware que se esté utilizando. Los puertos de las USART en los sistemas embebidos resultan de vital importancia, dado que a través de ellos el desarrollador puede realizar una depuración indirecta de los programas que se ejecute en dicho hardware. La gran mayoría del código fuente de los ejemplos que se encuentran en la imagen de Docker creada, utilizan este mecanismo de depuración. Otra de las modificaciones que fue necesario realizar, fue la adaptación del manejador de interrupciones, dado que se mapea diferente dependiendo de la placa emulada. Además, se adaptó la conexión interna en determinados pines, como por ejemplo el LED de testeo. Por otra parte, se modificó el código fuente de Qemu, para poder generar la emulación de los eventos de pulsadores externos, en las placas *stm32-Maple* y *BluePill*. Estos eventos se crearon, de forma tal que se generen cuando el usuario envié un comando *sendkey,* desde la consola de Qemu, al ejecutar cualquier programa en él. De esta forma se podrá emular la acción de un actuador del tipo pulsador. Adicionalmente se adaptó el código fuente, para que la utilización de los registros de hardware sea a través de la utilización de funciones de bibliotecas. Las cuales corresponden al funcionamiento del hardware pertinente. Al mismo tiempo, se generó un mecanismo de compilación que permite generar los binarios de todos los ejemplos.

3.7 Ejecución del emulador Qemu dentro de la Imagen Docker

Cuando el usuario descargue la imagen de Docker de este trabajo, del repositorio Docker Hub, deberá asociarle un contenedor para poder trabajar con ella. Para ello una de las formas de uso, es a través del comando "`docker run -it`". El cual crea un contenedor, asociado a una pseudo-terminal interactiva, la cual permite interactuar por medio de línea de comandos. Esto se puede visualizar en la Fig. 3. Dependiendo de lo que desee hacer, desde dicha terminal se podrá ejecutar cualquiera de los ejemplos, que se encuentran dentro de alguno de los subdirectorios: *Stm32-p103, Stm32-Maple y BluePill*. Los ejemplos de código fuente que se encuentran en estos directorios, permiten entre otras cosas: emular el encendido y apagado de un led de testeo, emular un programa que trabaje con un pulsador, emular un sensor que trabaje con valores analógicos, hacer programas que emulen interrupciones por software y hardware, emular el trabajo del temporizador que posee cada placa, poder emular el trabajo de los puertos USART para la depuración remota indirecta y permitir trabajar con programas que funcionen con el Sistema Operativo de Tiempo Real (*FreeRTOS*). En esta última opción, se permite ejecutar, en los sistemas embebidos emulados, programas que funcionan en un único o múltiples hilos de ejecución.

[2] USART: Dispositivo que controla los puertos y dispositivos serie. Se encuentra integrado en la placa base o en la tarjeta adaptadora del dispositivo.

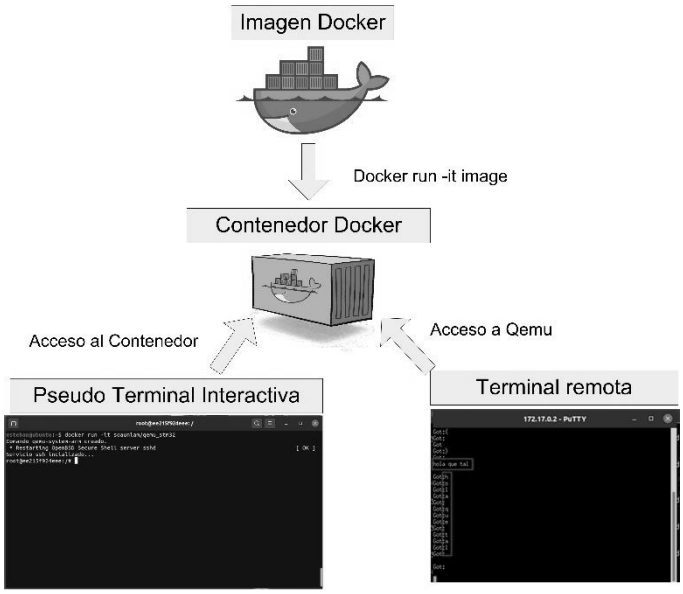


Fig. 3 Forma de ejecución de programas emulados en Qemu dentro del contenedor Docker

Para poder usar los binarios de los ejemplos en Qemu, que son generados cuando se compilan en el momento en que se genera la imagen, se debe ejecutar el emulador de una forma específica. Esta puede variar dependiendo del ejemplo que se quiera ejecutar, pero en la mayoría de los casos se debe seguir una forma base estándar de secuencia de pasos. Primero, el usuario deberá ejecutar el programa Qemu de la siguiente manera, desde la línea de comandos de la pseudo-terminal.

```
qemu-system-arm -M <nombre de la placa> -kernel <ar-
chivo.bin> -serial tcp::7777,server -nographic
```

El comando **qemu-system-arm,** es el archivo ejecutable del programa Qemu. Con el parámetro **-M,** se indica el modelo de placa que se desea emular. En nuestro caso, se puede emular las placas *stm32-p103, stm32-maple* y *stm32-f103c8.* Luego con la opción **-kernel,** se le indica la ubicación del programa binario que se desea ejecutar dentro del sistema embebido emulado. Adicionalmente con la opción **-serial,** se le dice al emulador que todas las entradas y salidas que se envíen al puerto serial de las placas sean redirigidas al puerto **7777** usando el protocolo TCP. Finalmente, con la opción **-nographic,** se deshabilita las salidas gráficas que genera el emulador.

Cuando se inicia la ejecución de Qemu, el emulador se quedará esperando una conexión externa a través del puerto 7777 del contenedor. Para ello el usuario deberá utilizar otra terminal remota y conectarse al puerto anteriormente indicado, utilizando la dirección IP que tenga asignado el contenedor de Docker, mediante el protocolo Telnet. Esto se muestra en la Fig. 4. Una vez que se establece la conexión, el programa emulado continuará su ejecución normalmente. Como ya se mencionó, la forma de ejecución puede variar, dependiendo del ejemplo que se desea ejecutar. Por ese motivo,

en esta investigación se generó un instructivo en forma de tutorial. En donde se detallan los pasos que se deben seguir, para poder ejecutar cada uno de los ejemplos disponibles dentro del entorno del contenedor. Este tutorial se encuentra dentro del repositorio generado (1), y por consiguiente, también se encuentra dentro de la imagen Docker generada.

En muchos de los ejemplos que se encuentran dentro de la imagen de Docker, se realiza depuración indirecta en forma remota a través de los puertos seriales. Como consecuencia de que el puerto serial es utilizado, tanto para mostrar datos en una terminal remota, como para ingresarlos a través de ella. Esto se puede visualizar en la siguiente figura, donde se muestra un caso de datos de entrada y salida a través de una terminal de este tipo.

Fig. 4 Terminal remota con datos de entrada y salida a través del puerto serial

4 Conclusiones

En este trabajo se presentó una alternativa, para que los desarrolladores de soluciones de sistemas embebidos, que pueden ser utilizados para IoT, realicen pruebas en placas STM32 emuladas a través del programa Qemu. Este trabajo les puede ayudar a los desarrolladores, establecer si determinado hardware le es o no de utilidad para sus proyectos, sin necesidad de adquirir el hardware físico. De manera que lo pueda realizar de forma rápida y sencilla, sin tener que realizar tediosas instalaciones y configuraciones de programas. Debido a que el emulador Qemu se encuentra empaquetado, configurado y automatizado dentro de una imagen Docker, fácil de emplear. Si bien este trabajo se centró en la emulación de tres placas: *stm32-p103, stm32-Maple* y *stm32-f103c8,* se planea en un futuro realizar el mismo trabajo de automatización, configuración y emulación mediante contenedores Docker para otros tipos de placas de desarrollo.

5 Referencias

1. Rose, K., Eldridge, S., Chapin, L.: La Internet De Las Cosas - Una Breve Reseña. , Reston, United State (2015).
2. Valiente, W., Carnuccio, E., Volker, M., de Luca, G., Villca, R., Adagio Matías: Entorno de contenedores de emuladores que contienen sistemas embebidos. In: XXIII Workshop de Investigadores en Ciencias de la Computación. pp. 12–17 (2021).
3. Eclipse Foundation: https://eclipse-embed-cdt.github.io/debug/qemu/.
4. Beckus: http://beckus.github.io/qemu_stm32/.
5. Lovric, D., Olsson, C.: Virtual Controllers (Tesis de Maestría). Department of Automatic Control, Lund University, Sweden (2016).
6. Muñoz, J.F., Goenaga, I.M.: Ofera Project: Open Framework for Embedded Robot Applications, European Union's Horizon 2020, Unión Europea (2019).
7. Amamory: https://hub.docker.com/r/amamory/qemu-stm32.
8. Chelladhurai, J.S., Singh, V., Raj, P.: Learning Docker - Second Edition: Build, ship, and scale faster. Packt Publishing, Birmingham, Reino Unido (2017).
9. Miell, I., Sayers, A.H.: Docker in practice. Manning Publications Co., Shelter Island, NY (2019).
10. Goasguen, S.: Docker Cookbook: Solutions and Examples for Building Distributed Applications. (2015).
11. Olimex: STM-P103 development board - User's manual., Plovdiv, Bulgaria (2016).
12. Olimex: OLIMEXINO-STM32 development board - Users Manual. , Plovdiv, Bulgaria (2011).
13. STMicroelectronics: STM32F103x8 DataSheet. (2015).