Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO

GRADO EN INGENIERÍA EN TECNOLOGÍA INDUSTRIAL

TRABAJO FIN DE GRADO

PUESTA EN MARCHA DE UN SISTEMA DE AUTOMATIZACIÓN

SIEMENS SIMATIC IOT2040

Alumno/Alumna: Torre Escudero, Aner

Director/Directora (1): Pérez González, Federico

Curso: 2017-2018

Fecha: 12 de Febrero de 2018

Universidad del País Vasco
Euskal Herriko Unibertsitatea

BILBOKO
INGENIARITZA ESKOLA

ESCUELA
DE INGENIERÍA DE BILBAO

TRABAJO DE FIN DE GRADO

Grado en Ingeniería en Tecnología Industrial

PUESTA EN MARCHA DE UN SISTEMA DE AUTOMATIZACIÓN SIEMENS SIMATIC IOT2040

Alumno: Torre Escudero, Aner
Fecha: Marzo 2018
Director: Pérez González, Federico
Curso académico: 2017/2018

DATOS BÁSICOS DEL TRABAJO DE FIN DE GRADO:

- *Alumno:* Aner Torre Escudero
- *Director:* Federico Pérez González
- *Departamento:* Departamento de Ingeniería de Sistemas y Automática
- *Título:* "Puesta en marcha de un sistema de automatización: SIEMENS SIMATIC IOT2040"
- *Resumen:* Este Trabajo se basa en la puesta en marcha de un sistema de automatización mediante el dispositivo SIMATIC IOT2040 de SIEMENS, de forma que se obtenga un sistema que lea, procese y envíe automáticamente la información de los datos de un proceso a un ordenador/servidor remoto a través del uso de protocolos de comunicación y la conexión a Internet, para facilitar la actualización de cualquier planta a la Industria 4.0 mediante el Internet de las Cosas.
- *Palabras clave:* automatización, IOT, Modbus, MQTT, Simatic, Industry4.0, GNU/Linux

- *Izenburua:* "Automatizazio sistema bat martxan jartzea: SIEMENS SIMATIC IOT2040"
- *Laburpena:* Lan hau automatizazio sistema bat martxan jartzean oinarrituta dago, SIEMENS-ren SIMATIC IOT2040 aparatuaren bitartez, prozesu baten datuak automatikoki irakurri, prozesatu, eta urruneko ordenagailu/zerbitzari batera bidaltzeko, komunikazio protokoloen eta Interneterako konexioaren bidez, horrela edozein industria baten Industria 4.0-ri eguneratzea errazagoa egiteko Gauzen-Internetaren bitartez
- *Hitzgakoak:* automatizazioa, IOT, Modbus, MQTT, Simatic, Industry4.0, GNU/Linux

- *Title:* "An automation system start up: SIEMENS SIMATIC IOT2040"
- *Abstract:* This project is all about an automation system start up through the SIEMENS SIMATIC IOT2040 device so it can read, process and send the data of any process into a remote computer/server with the help of some communication protocols and Internet connection, in order to make the Industry 4.0 update easier for any factory thanks to the Internet Of Things
- *Keywords:* automation, IOT, Modbus, MQTT, Simatic, Industry4.0, GNU/Linux

ÍNDICE DE CONTENIDOS:

ÍNDICE DE DEFINICIONES Y ABREVIATURAS:

- *Siemens Simatic IOT2040:* Plataforma abierta diseñada y fabricada por la empresa SIEMENS, orientado al procesado, almacenamiento y transmisión de datos, basada en Linux y compatible con cualquier dispositivo Arduino, especialmente creada para la automatización y conexión de sistemas y procesos, y para facilitar la actualización de las plantas a la Industria 4.0 mediante el IOT (Internet de las Cosas)

- *IOT (Internet Of Things):* Conexión entre dispositivos físicos, así como ordenadores, dispositivos móviles, electrodomésticos, vehículos, etc… a la red de Internet, para facilitar la automatización y transmisión de datos a través del mundo

- *Industry 4.0:* Denominación de la actual tendencia en cuanto a automatización e intercambio de información que se está produciendo en la industria gracias a la revolución de Internet. Se tiende a centralizar toda la información disponible de los procesos en servidores conectados a Internet, de forma que la información esté disponible en cualquier parte del mundo, a la vez que se permiten controlar esos procesos

- *Modbus TCP:* Protocolo de comunicación desarrollado por *Modicon systems*. Es un método usado para transmitir información mediante comunicación por Ethernet entre dispositivos electrónicos. Consiste en un dispositivo Maestro (Master) que solicita la información, y uno o varios Esclavos (Slave) que se encargan de suministrar la información solicitada por el maestro

- *MQTT (Message Queue Telemetry Transport):* Protocolo de transmisión de datos ligero y pensado para dispositivos sencillos o situaciones con mala o escasa conectividad, basado en el protocolo TCP/IP. Definido por la norma ISO/IEC PRF 20922

- *Yocto Linux:* Proyecto de software colaborativo, de código abierto y gratuito diseñado para crear plataformas personalizadas para sistemas embebidos (independientemente de su arquitectura) basadas en Linux mediante plantillas y distintas herramientas

- *SDK (Software Development Kit): Conjunto* de herramientas proporcionadas en forma de kit utilizadas para llevar a cabo la creación de programas destinados a los dispositivos a los dispositivos especificados por él. También puede ser utilizado para la creación de dispositivos hardware

- *Java JRE (Java Runtime Environment):* Conjunto de herramientas proporcionadas con el fin de permitir la ejecución de programas basados en el lenguaje de programación Java.

- *GPIO (General Purpose Input Output):* Pin genérico incluido en ciertos dispositivos electrónicos cuya finalidad se puede escoger a la hora de la programación o durante la ejecución de un programa

- *IDE (Integrated Development Environment):* Programa software cuya finalidad es crear nuevos programas. Para ello cuenta con distintas herramientas como editores de texto, compiladores, o depuradores y suele funcionar en conjunto con determinados SDKs

ÍNDICE DE FIGURAS:

ÍNDICE DE TABLAS:

1. INTRODUCCIÓN:

Este documento detalla el Trabajo de Fin de Grado denominado "*PUESTA EN MARCHA DE UN SISTEMA DE AUTOMATIZACIÓN BASADO EN SIEMENS SIMATIC IOT2040*", el cual se basa, como su nombre indica, en la puesta en marcha de un sistema de automatización mediante el dispositivo SIMATIC IOT2040 de SIEMENS, de forma que se obtenga un sistema que lea, procese y envíe automáticamente la información de los datos de un proceso a un ordenador/servidor remoto a través del uso de protocolos de comunicación y la conexión a Internet, para facilitar la actualización de cualquier planta a la Industria 4.0 mediante el Internet de las Cosas.

Figura 1. Siemens SIMATIC IOT2040

Primero, se explicará los pasos a seguir para obtener, grabar y configurar una imagen de arranque Linux (distribución Yocto Linux) desarrollada, adaptada y distribuida por Siemens para el Siemens SIMATIC IOT2040, en una tarjeta Micro SD para su correcto encendido y funcionamiento.

Después, se detallará la configuración de la estación de trabajo (ordenador) para la programación del SIMATIC IOT2040 mediante un lenguaje de alto nivel.

Finalmente, en el apartado de los anexos, se propondrá una serie de programas que faciliten la comunicación entre el SIMATIC IOT2040 y otros dispositivos, mediante los protocolos de comunicación *ModBus* y *MQTT* para la transmisión de información, a través de las interfaces Wi-Fi y Ethernet. Estos programas, serán de código abierto, disponible para cualquier usuario o desarrollador para su observación, uso o incluso mejora de estos y estarán publicados en la página web de GitHub.

2. CONTEXTO

En los últimos años la industria ha adquirido una tendencia hacia la automatización y digitalización de prácticamente cualquier proceso en el que se pueda obtener un mínimo de información, con el objetivo de recopilar datos, para calcular estadísticas/analíticas, encontrar ineficiencias, retrasos, fallos o errores, o incluso implementar cualquier mejora donde sea posible, ya sea controlar uno o varios robots o incluso implementar una inteligencia artificial. Para ello, la industria se basa en plataformas de automatización, que son las que se encargan de leer los datos de todo tipo de sensores y acceder a los actuadores para, procesar, analizar, enviar y almacenar información.

Cabe destacar que, debido a la globalización industrial, gran parte de las empresas tienen distintas plantas deslocalizadas en un número elevado de países, pudiendo estar unas a miles de kilómetros de otras. Debido a esto, es preciso crear un sistema basado en *La Nube*, de forma que la organización entre las distintas plantas sea más sencilla para lo cual, la industria se ayuda de la conexión global a Internet. De este modo, se conectan los distintos dispositivos de cada planta a Internet para que, de esta forma, reciban o envíen automáticamente toda la información del/al sistema central basado en *La Nube*. Así es como se introduce el concepto de *IoT*, más concretamente *IIoT* (Industrial Internet of Things), que está ejerciendo una gran influencia en la llamada *Cuarta Revolución Industrial, Industria Inteligente* o *Industria 4.0*.

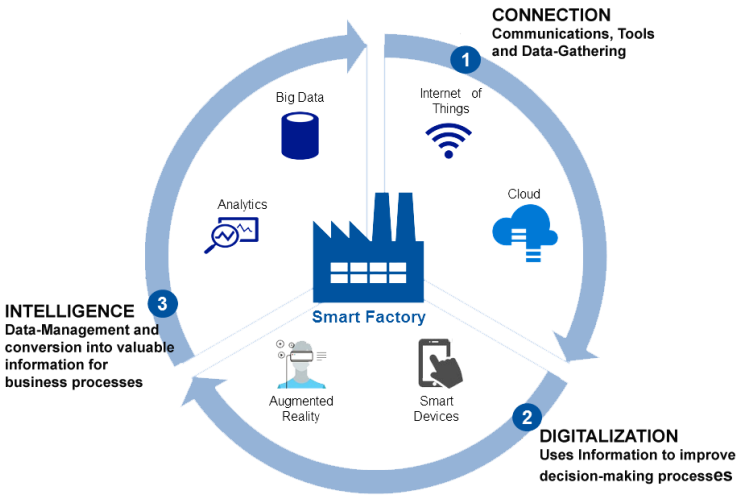


Figura 2. Esquema de una industria 4.0

El concepto de IIoT, es básicamente el uso de tecnologías dedicadas al IoT aplicadas a los procesos industriales, que aporta una gran cantidad de beneficios a la industria, tanto a nivel organizativo, como a nivel de eficiencia, e incluso a nivel económico ya que, normalmente, este tipo de sistemas de automatización no suelen requerir de inversiones importantes y sus costes de mantenimiento son ínfimos en comparación con los de cualquier otra parte de los procesos.

El presente proyecto tiene como objetivo la puesta en marcha de un sistema de automatización, que haga más sencilla la implementación del IIoT para prácticamente cualquier proceso. Para ello, se basa en la utilización de la plataforma SIMATIC IOT2040 de SIEMENS, la cual presenta las siguientes características entre otras:

CARACTERÍSTICAS BÁSICAS DEL SIMATIC IOT2040
Placa base basada en la Intel Galileo 2ª generación: procesador Intel Quark con ahorro de energía, 1 GB de memoria RAM, 2 puertos Ethernet RJ45, 2 interfaces RS232/485 y un reloj RTC respaldado por una pila
Compatible con el sistema operativo Linux (distribución Yocto)
Compatible con distintos lenguajes de programación
Compatible con las interfaces de Arduino (Arduino Shields) y tarjetas miniPCIe

Tabla 1. Características básicas del SIMATIC IOT2040

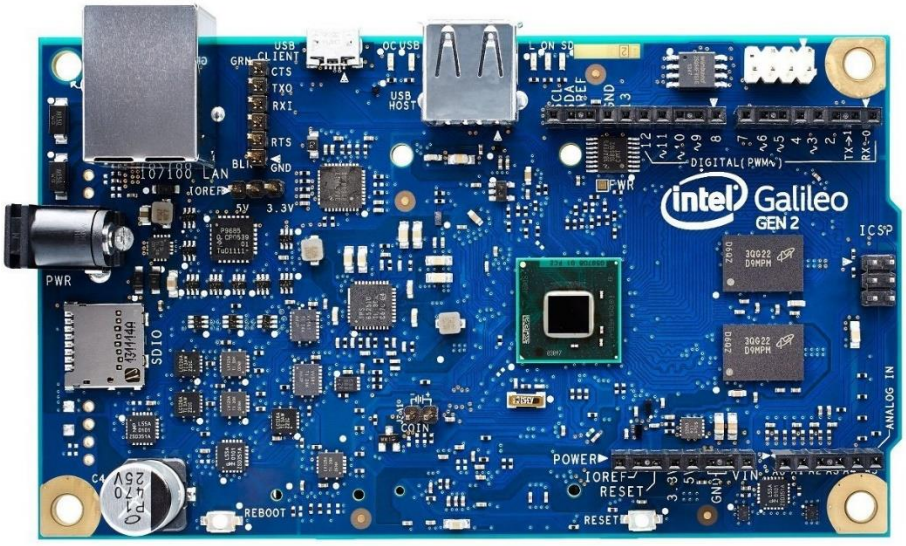


Figura 3. Detalle de la Intel Galileo 2ª generación

Estas características permiten cumplir todos los requisitos básicos que podría tener una planta industrial sencilla cumpliendo, por lo tanto, el objetivo marcado por el proyecto: gracias a sus puertos Ethernet y compatibilidad con tarjetas miniPCIe se puede proveer al SIMATIC IOT2040 de conexión a Internet mediante cable o de manera inalámbrica. También se puede conectar a él una gran cantidad de sensores/actuadores gracias a su elevado número de entradas/salidas, tanto digitales como analógicas (mediante PWM) como se puede observar en la figura superior, y programar su funcionamiento mediante distintos lenguajes de programación como C, C++, Java, NodeRED, Python, etc…

Cabe destacar también que, gracias a este dispositivo, se obtiene una plataforma de automatización modular, que permite la integración y comunicación con prácticamente cualquier dispositivo al cumplir distintos estándares industriales y de programación.

3. ALCANCE DEL PROYECTO

Debido a la naturaleza de la temática, este proyecto garantiza al usuario/cliente una gran variedad de oportunidades y posibilidades ya que, básicamente, se trata de crear un sistema de automatización modular que sea fácil de adaptar para prácticamente cualquier proceso. Por lo tanto, se puede decir que el proyecto tiene un alcance muy amplio, prácticamente impuesto por el propio usuario, por lo que éste no está totalmente determinado puesto que depende del uso que se le vaya a dar.

Sin embargo, las potenciales limitaciones existentes vienen impuestas por las limitaciones técnicas del SIMATIC IOT2040 ya que, por ejemplo, carece de gran potencia computacional y un número limitado de entradas/salidas, por lo que podría ser necesario añadir un ordenador o uno o más dispositivos SIMATIC IOT2040 para cada caso, respectivamente. Destacar también como limitación técnica las tensiones máximas de los pines de entrada/salida, que restringe la compatibilidad a dispositivos que trabajen con tensiones igual o menores con el fin de evitar daños al dispositivo.

En cuanto a las comunicaciones prácticamente no hay limitación alguna (más allá que el ancho de banda de la conexión), puesto que al estar el funcionamiento del SIMATIC IOT2040 basado en Linux, éste puede adoptar una gran cantidad de librerías que incluyan la programación para los distintos protocolos de comunicación estándar existentes (por ejemplo *ModBus TCP* o *MQTT*), otorgando al sistema compatibilidad con prácticamente cualquier dispositivo conectado a la red global de Internet y en cualquier parte del mundo. Además, el dispositivo posee 2 puertos Ethernet (para cables RJ45) y un puerto miniPCIe, en el que se podría insertar una tarjeta Wi-Fi, por ejemplo, para añadir conectividad inalámbrica a Internet, y así prescindir de instalaciones de cables de red en la planta.

A continuación se expondrán algunos casos de estudio para ver las posibilidades del alcance del proyecto, no limitándose únicamente conexiones con otros dispositivos mediante Internet:

- Controlador: El sistema se podría programar de forma que, en función de la lectura de los sensores conectados a los pines de entrada, controle o manipule ciertos actuadores, de forma que se obtenga una respuesta deseada en el proceso a controlar. Mediante programas que trabajen con ecuaciones en diferencias, se podría simular cualquier tipo de controlador

- Intermediario entre sensores y sistemas distribuidos de control: El sistema podría trabajar recopilando toda la información disponible de los sensores a los que está conectado, procesándola y enviándola a uno o varios servidores remotos con el objetivo de almacenarla para posteriormente obtener analíticas y estadísticas

- Control/Mando remoto: También es posible que, gracias a la conectividad a Internet, se pueda controlar lo que ocurre en un proceso de forma remota siendo posible manipular sus distintas variables desde un ordenador que esté a kilómetros de distancia

- Combinación de los tres casos: Mediante un ordenador o servidor remoto, se puede leer los datos de los sensores enviados por el sistema, actuar según sea necesario sobre el proceso cambiando las variables del controlador, de forma que se obtiene un control total sobre el proceso, sin la necesidad de estar presente en él

Se debe incluir también, la posibilidad de este sistema de acercarse a las últimas tendencias tanto a nivel industrial como no industrial que están surgiendo a partir de las nuevas tecnologías y la conectividad así como el *cloud-computing*, *fog-computing* o *edge-computing*. Gracias a la compacidad del SIMATIC IOT2040, su fácil integración en cualquier proceso y su poder de computación, hace más fácil integrar estas últimas tendencias ya que su conectividad le permite convertirse en un nodo más de estas redes de computación.

4. BENEFICIOS DEL PROYECTO

1. Beneficios técnicos

Debido al alcance del proyecto, el sistema de automatización basado en el SIEMENS SIMATIC IOT2040, trae una serie de ventajas que pueden ser muy interesantes de cara a las empresas industriales:

- En primer lugar, la sencillez, robustez y amplia compatibilidad del dispositivo hace que pueda ser situado en prácticamente cualquier lugar de una planta, lo cual otorga un sistema de automatización implantable en casi cualquier tipo de proceso, permitiendo la implementación de sensores en los procesos en lugares donde antes no se podrían colocar y, por lo tanto ayuda a recibir una mayor cantidad de la planta. Añadir también que, el hecho de ser modular facilita la movilidad de cada proceso o puesto de trabajo.
- En segundo lugar, además de reducir la necesidad de ordenadores en lugares no adecuados, la posibilidad de tener un sistema de automatización inalámbrico hace posible simplificar en gran medida la instalación eléctrica y de comunicaciones de la planta, al prescindir de menor cableado, lo cual contribuye también a la liberación de espacio en la planta, permitiendo un mayor aprovechamiento de éste.
- En tercer lugar, la gran compatibilidad de este sistema de automatización con distintos estándares, proporciona a la propia planta una mayor compatibilidad con otros sistemas o máquinas, siendo posible utilizar otros más eficientes, que ocupen menos espacio o que incluso trabajen más rápido, aumentando así la producción y, por lo tanto aportando mayor beneficio económico indirectamente.
- Por último, hay que añadir que como sistema de automatización, este proyecto trae todos los beneficios que podría otorgar uno de este tipo, como por ejemplo una mayor organización a nivel de trabajo, disminución de los tiempos de espera o retrasos, e incluso la posibilidad de alertar a un operario en caso de un fallo o error en el funcionamiento del proceso, favoreciendo la eficiencia a la vez que disminuye la tasa de errores en la producción.

2. Beneficios económicos

Éste es posiblemente el apartado más importante a destacar, puesto que el objetivo principal de prácticamente cualquier empresa es maximizar el beneficio en la medida de lo posible. Como cualquier sistema de automatización, éste brinda la posibilidad de reducir el trabajo que normalmente se encargaría de hacer un operario, por lo tanto reduciendo el número de horas de trabajo, o incluso prescindiendo de él para cierta tarea, disminuyendo el gasto que esta podría suponer. Por otro lado, el potencial aumento en la eficiencia de la planta que trae este proyecto puede ayudar a aumentar la producción a la vez que se disminuyen los tiempos de espera y los retrasos, traduciéndose en una disminución del gasto.

El coste de la instalación que podría suponer este proyecto es prácticamente despreciable frente al gasto que puede tener cualquier planta de procesado o fabricación. Si se añade además, el hecho de que el sistema consume muy poca energía y que el mantenimiento de este es prácticamente nulo, se puede decir que la rentabilidad de este es máxima, suponiendo un coste ínfimo a la vez que aporta un gran beneficio.

Añadir también que la amplia compatibilidad de este sistema abre las puertas a la posibilidad de elegir entre una mayor variedad de máquinas o dispositivos en el mercado, incluyendo los más económicos o de menor mantenimiento que con otro sistema con menor compatibilidad no funcionaría permitiendo, de nuevo, la disminución del gasto.

Por último, gracias a la comunidad Open Source, es posible encontrar librerías y programas, que faciliten la programación de este sistema, sin necesidad de adquirir licencias o programas que supongan un coste importante

3. Beneficios sociales

El sistema de automatización basado en el SIMATIC IOT2040, trae un posible beneficio social a destacar:

- El propio sistema está basado en Linux, un sistema operativo de código abierto que cualquiera puede ver o incluso modificarlo y proponer mejoras sobre él y que está basado bajo la licencia GPL (General Public License), lo obliga a que todo código basado en él debe ser publicado, a la vez que se debe facilitar el acceso al código original. Gracias a esta licencia, las propias modificaciones o mejoras que cometa el usuario sobre este sistema se convierten en contribuciones a la comunidad Open Source (Código abierto), de forma que cualquier persona o entidad sea pueda beneficiar de estas, facilitando indirectamente la programación del mismo o incluso otro tipo de software.

Figura 4. Logotipo de la Fundación Linux

Figura 5. Logotipos de la Licencia GPL

5. ANÁLISIS DE ALTERNATIVAS

1. Introducción

Para realizar un análisis detallado de las posibles alternativas que sean preferiblemente competitivas con este proyecto, es preciso mirar posibilidades con características parecidas. Por lo tanto, se deben buscar alternativas sencillas y de bajo coste, fáciles de implementar y, por supuesto, que incluya compatibilidad con los distintos protocolos de comunicación presentes en la industria, contando preferiblemente con interfaces Wi-Fi o Ethernet por su amplia presencia. Mirando las diferentes soluciones con estas características en el mercado, se encuentra una gran variedad de dispositivos, pero debido a su gran popularidad, y por contar con un gran soporte y con mucha documentación por parte de los fabricantes y de la comunidad Open Source, se eligen los siguientes dispositivos para el análisis de alternativas: Arduino y Raspberry Pi. Por supuesto, también se debe incluir la versión inferior del dispositivo utilizado para este proyecto: el Siemens SIMATIC IOT2020.

2. Siemens SIMATIC IOT2020

Es obvio que, por pertenecer a la misma familia, el Siemens SIMATIC IOT2020 debe estar incluido en el análisis de alternativas. Tanto el IOT2020 como el IOT2040, están diseñados y fabricados por el mismo fabricante, por lo que el soporte y servicio técnico de ambos debería de ser, en principio similar, contando con un foro online dedicado a estos dispositivos (https://support.industry.siemens.com). Sucede lo mismo con el apartado de software y, de hecho, ambos dispositivos cuentan con la misma imagen de arranque, mismos binarios, y mismas librerías. Es por esto que este detalle no estará incluido en la comparativa y, por lo tanto, para hacer el análisis comparativo en este caso, se debe mirar únicamente hacia las características del hardware y, evidentemente, el precio.

	SIMATIC IOT2020	**SIMATIC IOT2040**
Procesador	Intel Quark x1000	Intel Quark x1020 (con arranque seguro)
Memoria RAM	512MB	1024MB
Interfaces Ethernet	1	2 x RJ
Puertos serie	0	2 x RS232/485
Reloj interno	No	Mediante pila
Precio	±80€	±180€

Tabla 2. Comparativa entre los SIMATEC IOT2020 e IOT2040

Como se puede ver en la tabla 3, las características el IOT2020 son menores aunque el precio también es menor. Sin embargo, teniendo en cuenta dónde se aplica este proyecto, sería más recomendable elegir el IOT2040, ya que la diferencia de precio es relativamente pequeña en

comparación con los movimientos monetarios de una planta convencional y el hecho de tener dos puertos de Ethernet para la comunicación ayuda en gran medida a la programación, a su depuración, y a la búsqueda de errores (en caso de que los haya), evitando potenciales pérdidas de tiempo y dinero. Esto se debe a que, al tener dos puertos, es posible conectarse al IOT2040 sin tener que cortar la conexión que tenía previamente.

En cuanto a la capacidad de almacenamiento, ésta viene dada por la tarjeta Micro SD que se utilice para grabar la imagen de arranque, ya que toda la información generada será almacenada en esta. Por lo tanto, la capacidad de almacenamiento es de entre 8 y 32 GB.

3. Arduino (+ shields)

Arduino es una plataforma de hardware libre, lo cual significa que cualquiera puede acceder a sus planos de fabricación y su diseño. Esta plataforma se compone principalmente de microcontroladores y otros elementos, siendo estos primeros programables por el usuario de una manera sencilla con un lenguaje de programación específico para *Arduino* basado en el lenguaje C y con un entorno de programación (IDE) abierto y gratuito.

Las principales ventajas que puede ofrecer la plataforma Arduino sobre el IOT2040 son su precio y su tamaño. Esta plataforma, incluye la compatibilidad con distintos módulos llamados *Arduino Shields* que otorgan nuevas funcionalidades a estas placas, como por ejemplo interfaces Wi-Fi o Ethernet o mayor facilidad para conectar distintos sensores y actuadores como el caso del *TinkerKit! Shield*.

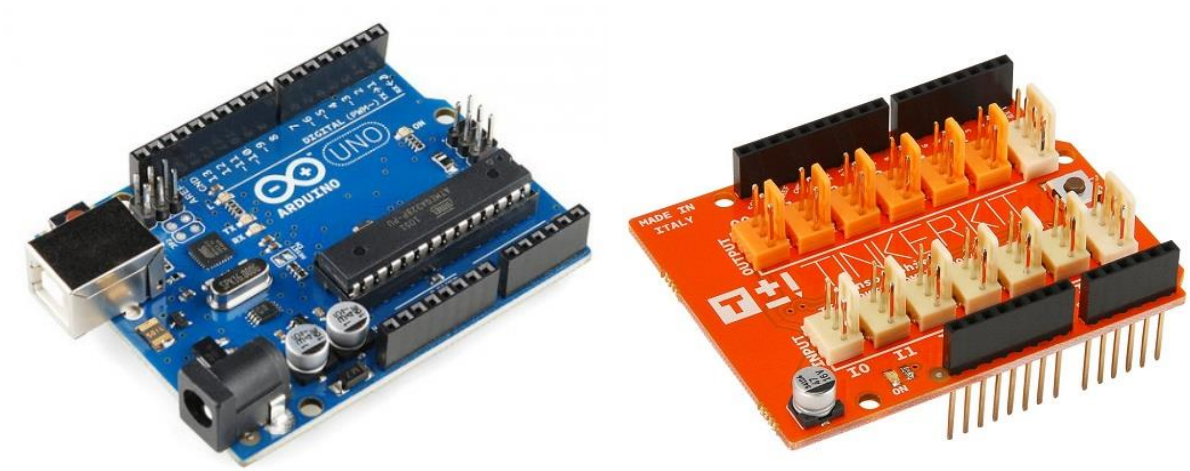


Figura 6. Arduino UNO y la TinkerKit! Shield

Entre los objetivos de este proyecto de un sistema de automatización, se encuentra el de la comunicación con otros dispositivos, por lo que es necesario utilizar, de base, un *Shield* que proporcione conectividad Wi-Fi o Ethernet a la placa Arduino. Teniendo en cuenta que la

extensión de funcionalidades mediante estos *Shields* está limitada por contener un número limitado de pines de entrada/salida, se presenta una desventaja de esta plataforma frente al IOT2040 porque este último ya contiene puertos dedicados a tal conectividad además de ser compatible con la mayor parte de los *Arduino Shields*, haciendo posible prescindir de los *Shields* de conectividad y, por lo tanto, pudiendo extender otro tipo de funcionalidades.

Además, en cuanto a hardware, la plataforma *Arduino* es considerablemente inferior, con especificaciones muy simples, lo que las hace perfectas para tareas sencillas debido a su tamaño y su bajo consumo. En la siguiente tabla, se muestran las características de la placa *Arduino* más popular:

	ARDUINO UNO (rev3)
Dimensiones	68.6 x 53.4 mm
Peso	25 g
Precio	20 €
Microcontrolador	ATmega 328P (16 MHz)
Memoria RAM	2 KB
Memoria Flash	32 KB
Tensión de operación	5 V
Tensión de entrada	7-12 V
Pines de E/S digitales	14 (6 de ellos compatibles con PWM)
Pines de entrada analógicos	6
Corriente máxima por pin (CC)	20 mA

Tabla 3. Especificaciones de la Arduino UNO (rev3)

Como se puede ver en la tabla, las especificaciones quedan limitadas al almacenamiento de un simple programa y muy poca información, por lo que para este proyecto no serían adecuadas, puesto que en caso de una pérdida de conexión, no podría almacenar la información recibida para mandarla tras la reconexión, por lo que se produciría una pérdida potencialmente importante. Sin embargo, hay que tener en cuenta que las placas *Arduino* son una interesante propuesta de cara a ampliar el proyecto porque, gracias a su pequeño tamaño y bajo peso, pueden ampliar el sistema de automatización de forma modular, ayudando a implementarlo en lugares difíciles donde previamente no se podría. Más adelante se verá cómo es posible lograr la comunicación entre el *Simatic IOT2040* y una placa *Arduino* con el *Ethernet Shield* mediante el protocolo *ModBus*.

4. Raspberry Pi

Esta plataforma es también una opción muy interesante a tener en cuenta, debido a su relación, capacidad y su versatilidad. De hecho, se puede decir que se acerca mucho a la plataforma SIMATIC en cuanto a posibilidades

Raspberry Pi es una plataforma de hardware similar a *Arduino*, pero sin estar tan centrada en ser libre. Sin embargo, tanto en hardware como en software es muy superior, puesto que tiene unas

características muy superiores e incluso compatibilidad con distintos sistemas operativos, como se verá posteriormente. Cuenta con una gran comunidad Open Source, lo cual hace que tenga una compatibilidad y funcionalidad muy extendida gracias a las distintas librerías y programas existentes y también cuenta con una gran cantidad de accesorios existentes de los cuales hay mucha variedad debido a su extendida popularidad. Destacar también su bajo coste, ya que la última versión comercializada se vende a partir de los aproximadamente 40 euros.

Se puede decir que compite directamente con el SIMATIC IOT2040, ya que incluye, entre otros, cuatro puertos USB, un puerto Ethernet y un puerto HDMI, lo que lo puede llegar a convertir en prácticamente un ordenador de características básicas. Por supuesto también incluye 40 pines GPIO, disponibles para conectar distintos accesorios como sensores y actuadores.

Sin embargo, este tipo de placas, aunque tienen distintos accesorios y carcasas que les proporcionan una mayor robustez, no están diseñadas para el entorno industrial y, por lo tanto, sus condiciones de operación pueden no ser adecuadas en algunas industrias, lo que lleva a restringir el uso de esta plataforma como complementos o módulos para este proyecto ya que, como las placas *Arduino*, tienen unas dimensiones muy pequeñas, son ligeras y se pueden obtener por precios muy bajos. Además posee conectividad Bluetooth, lo cual permite comunicarse con el sistema de automatización de forma inalámbrica.

Otra desventaja de considerable importancia es que ésta no está dotada de pines de entrada analógicos, lo cual se requiere para cualquier tipo de sensor. Por lo tanto se necesitaría un circuito ADC externo, para poder implementarla

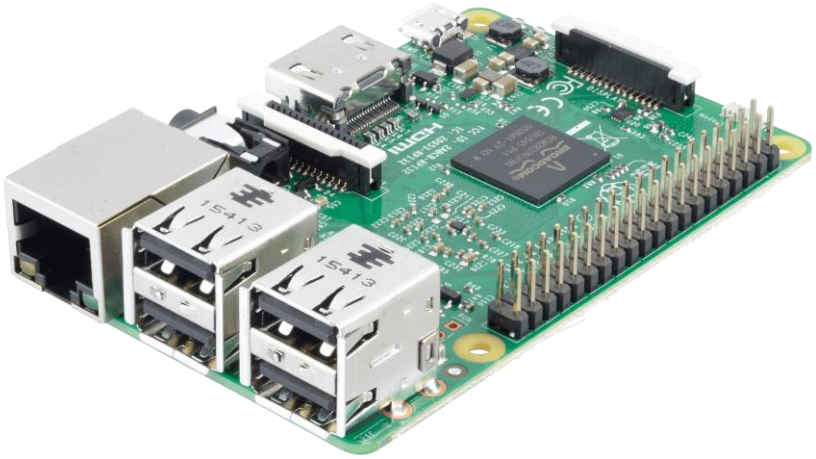


Figura 7. Raspberry Pi 3 (Modelo B)

En la siguiente tabla se pueden observar las características del último modelo comercial de la Raspberry Pi:

	RASPBERRY PI 3 (Modelo B)
Dimensiones	86.9 x 58.5 x 19.1 mm
Peso	41.2 g
Precio	40 €
Microcontrolador	Broadcom BCM2837 64bit (4 núcleos x 1.2GHz)
Memoria RAM	1 GB
Memoria Flash	Ranura para MicroSD
Tensión de operación	5V
Tensión de entrada	5V
Pines de E/S digitales	40 (1 compatible con PWM)
Pines de entrada analógicos	0 (requiere chips ADC externos)
Corriente máxima por pin (CC)	50 mA

Tabla 4. Especificaciones de la Raspberry Pi 3 (Modelo B)

5. Selección de la solución

Para hacer una selección entre las distintas alternativas propuestas es necesario realizar una comparación entre estas, realizando un análisis detallado de las ventajas y desventajas propuestas previamente para cada alternativa. Para ello, se ha elaborado una tabla con los posibles requerimientos que se pueden presentar en lugares donde se va a implementar el sistema de automatización propuesto en el proyecto, introduciendo SÍ en caso favorable o que suponga una ventaja y NO en caso desfavorable o en caso que suponga una desventaja:

	SIMATIC IOT2000	ARDUINO	RASPBERRY PI
Diseñado para ámbito industrial	SÍ	NO	NO
Compatibilidad con sistemas operativos	SÍ	NO	SÍ
Compatibilidad con multitud de protocolos	SÍ	NO	SÍ
Conectividad	SÍ	SÍ (requiere Arduino Shield)	SÍ
Capacidad de información	SÍ	NO	SÍ
Capacidad de procesamiento	SÍ	NO	SÍ
Memoria RAM	SÍ	NO	SÍ
Pines de entrada analógicos	SÍ	SÍ	NO
Pines de salida analógicos	SÍ	SÍ	NO
Precio	NO	SÍ	SÍ
Tamaño y peso	NO	SÍ	SÍ
TOTAL	9	5	8

Tabla 5. Ventajas y desventajas de las alternativas

Como se puede ver, la propuesta más adecuada es la serie SIMATIC IOT2000 de Siemens, de la cual se escogerá el IOT2040 por razones ya comentadas en el correspondiente apartado. Se puede decir que la plataforma Raspberry Pi también podría ser adecuada aunque, pese a tener pocas desventajas, estas son de gran importancia lo cual lleva a descartarla como elección.

6. DESCRIPCIÓN DE LA SOLUCIÓN

Tras hacer un análisis detallado de las posibles alternativas y, comparando las ventajas y desventajas de cada una se optará por elegir el Siemens SIMATIC IOT2040 como plataforma para el sistema de automatización, como estaba propuesto en el proyecto. Tras la puesta en marcha, se obtendrá un sistema de automatización preparado para que le sean conectados, directa o indirectamente, distintos sensores y actuadores, cuyo comportamiento vendrá definido por un programa que el usuario escriba. También será posible, como se ha sugerido previamente, utilizar varias placas *Arduino*, de forma que los sensores y actuadores se conecten a ellas, y definir su comportamiento mediante las órdenes que reciban, de forma alámbrica o inalámbrica, a través distintos protocolos desde el SIMTATIC IOT2040 gracias a un sencillo programa.

En resumen, se obtendrá un sistema de automatización completamente programable y modular que, gracias a la conectividad a Internet pueda comunicarse con otros dispositivos acomodándose a los requisitos del usuario.

En primer lugar, se deberá descargar, grabar, y configurar, la imagen de arranque Linux proporcionada por SIEMENS para el SIMATIC IOT2040 en una tarjeta Micro SD. Para ello se utilizarán las herramientas: Win32 Disk Imager y PuTTY.

Posteriormente, se configurará la estación de programación del SIMATIC IOT2040 con el software Eclipse IDE for C/C++ developers y el plugin para Eclipse IOT-2000 proporcionado por SIEMENS para trabajar con el lenguaje de alto nivel C++.

Finalmente, en el apartado de anexos, se incluirá una serie de programas para demostrar el correcto funcionamiento del sistema y algunas de las posibilidades que este podría otorgar.

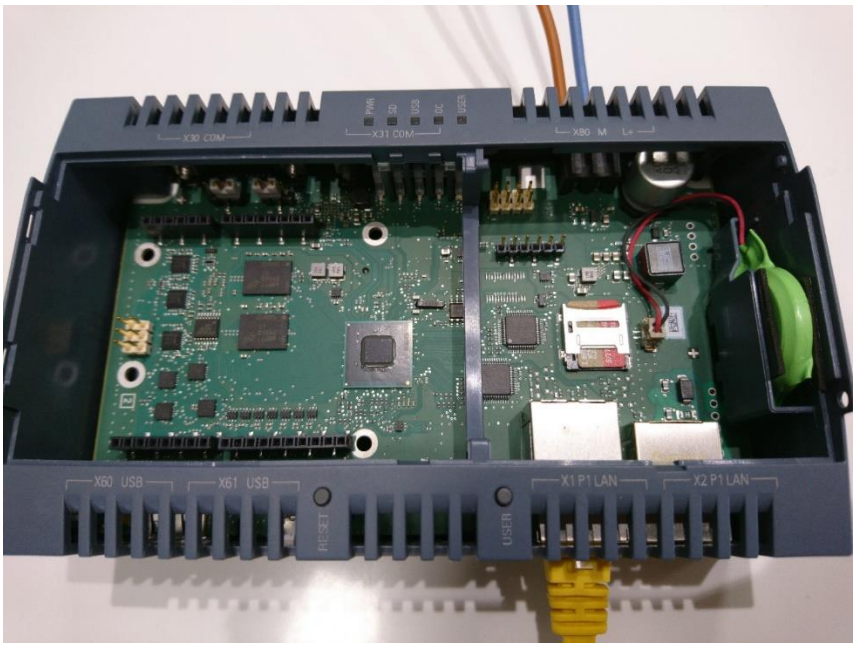


Figura 8. Interior del SIMATIC IOT2040

REQUISITOS

Para llevar a cabo este proyecto, serán necesarios los siguientes dispositivos/accesorios y programas:

HARDWARE
○ Siemens SIMATIC IOT2040 ○ Tarjeta Micro SD de entre 8 y 32 gigabytes de capacidad ○ Ordenador con puerto Ethernet y con lector de tarjetas Micro SD ○ Cable Ethernet RJ-45 ○ Fuente de alimentación de entre 9 y 36 voltios de corriente continua
SOFTWARE
○ PutTTY ○ Win32 Disk Imager (en el caso de usar Windows) ○ Eclipse IDE for C/C++ Developers y el plug-in de IOT2000 para Eclipse ○ El SDK para el IOT2000 ○ Java Platform JRE (Java Runtime Environment)

Tabla 6. Requisitos del proyecto

7. METODOLOGÍA

1. Introducción

A continuación se detallarán los pasos a seguir para la puesta en marcha del sistema. Primero, se empezará por la descarga de todo el software necesario. Después, se grabará la imagen de arranque en una tarjeta de memoria Micro SD y se procederá a la configuración del IOT2040 para alojar los programas que se compilarán mediante el entorno de programación Eclipse. Posteriormente, se configurará el IDE Eclipse, para insertar el plug-in necesario para la integración del compilador y el SDK necesarios y la comunicación con el SIMATIC IOT2040 a través de cable Ethernet. Por último, se creará un programa inicial de prueba *Simple Test*, para comprobar que todo esté instalado correctamente y funcionando como debería. Destacar también, que en la sección de anexos se incluirá una serie de programas con el fin de ayudar a la explicación del desarrollo de distintos programas para la integración con el protocolo ModBus TCP, comunicación serie y demás.

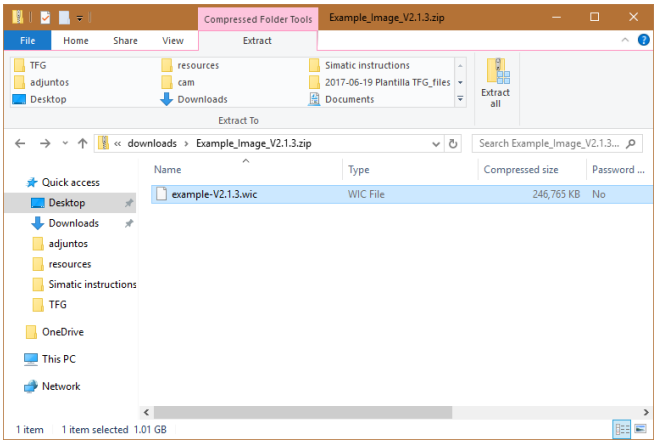
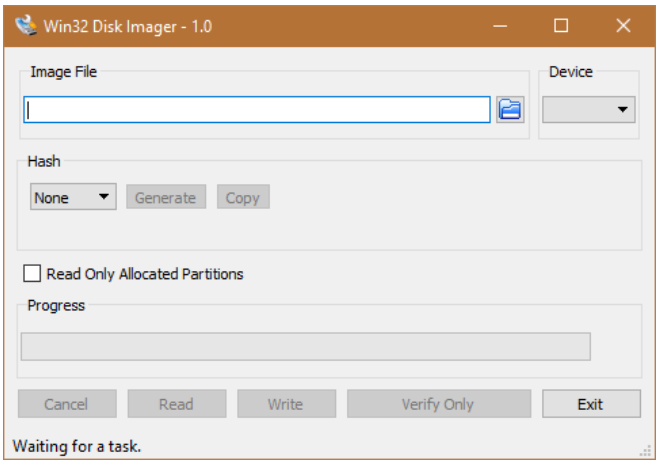
2. Descarga del software necesario

La siguiente lista de software está ordenada según los pasos que se vayan a seguir posteriormente:

- Imagen de arranque Linux: Es la un archivo de imagen que se grabará en la tarjeta Micro SD para la carga del sistema operativo del SIMATIC IOT2040 (NOTA: A día de hoy 9 de Octubre de 2017, la última versión disponible es la v2.1.3)
    - https://support.industry.siemens.com/cs/document/109741799/simatic-iot2000-sd-card-example-image
- Win32 Disk Imager: Programa que facilita la grabación de la imagen de arranque Linux en la tarjeta de memoria Micro SD
    - https://sourceforge.net/projects/win32diskimager/
- PuTTY: Programa que facilita la comunicación entre el ordenador y el terminal Linux del SIMATIC IOT2040 mediante TTY
    - http://www.putty.org
- Eclipse for C/C++: Distribución de Eclipse compatible con la programación en lenguajes C y C++
    - https://www.eclipse.org/downloads/eclipse-packages/
- IOT2000 for Eclipse plug-in y SDK: Son archivos necesarios para la creación y compilación de los programas para el IOT2040. Ambos se pueden obtener en el mismo enlace
    - https://support.industry.siemens.com/cs/document/109744106/simatic-iot2000-eclipse-plugin
- 7-Zip: Se requiere específicamente para extraer el *IOT2000 for Eclipse plug-in* y el SDK (NOTA: Se debe utilizar una versión igual o superior a la v16.0. De lo contrario, el SDK no se extraerá correctamente y se obtendrá un SDK corrupto)
    - http://www.7-zip.org/download.html
- Java JDK: Se requiere para ejecutar Eclipse
    - https://oracle.com/technetwork/java/javase/downloads/index.html

3. Grabación de imagen y configuración del SIMATIC IOT2040

Para que el SIMATIC IOT2040 pueda ejecutar programas que definan el comportamiento de los pines GPIO y las comunicaciones, es necesario que éste ejecute un sistema operativo. El propio fabricante proporciona un archivo de imagen del sistema operativo basado en la distribución Yocto Linux. Para el arranque de este sistema operativo, es necesario grabar dicha imagen de arranque en una tarjeta de memoria Micro SD e insertarla en su respectivo compartimento, de forma que pueda ser leída por el SIMATIC IOT2040 durante su encendido. Después, se debe configurar para su funcionamiento, conexión y programación y, para ello, primero se cambiará la contraseña del acceso *root*, que es el usuario Linux con mayor privilegio para modificar archivos del sistema, para después configurar la conexión con el dispositivo cambiando su dirección IP y, finalmente crear un directorio que aloje los programas creados con Eclipse.

1.-	**Extraer la imagen de arranque Linux.** Para obtener la imagen de arranque, es necesario extraer el archivo *.zip* descargado previamente. Tras esto, se obtendrá un archivo *.wic* de aproximadamente 1 GB  *Figura 9. Archivo de imagen Linux*
2.-	**Instalar y ejecutar Win32 Disk Imager.** El propio programa incluye un asistente de instalación muy sencillo. Tras la instalación, el programa presenta el siguiente aspecto:  *Figura 10. Win32 Disk Imager*

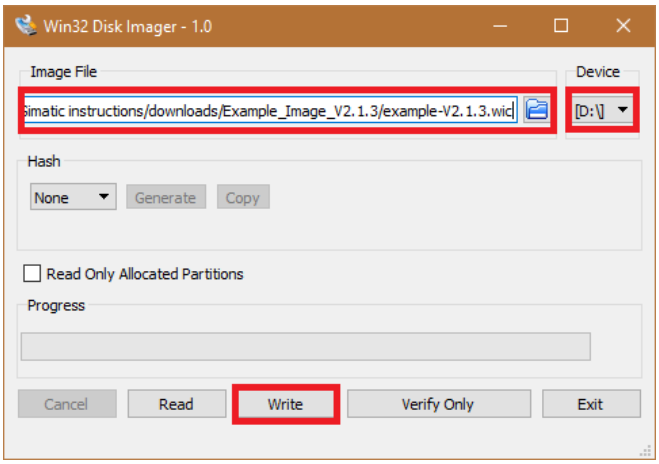
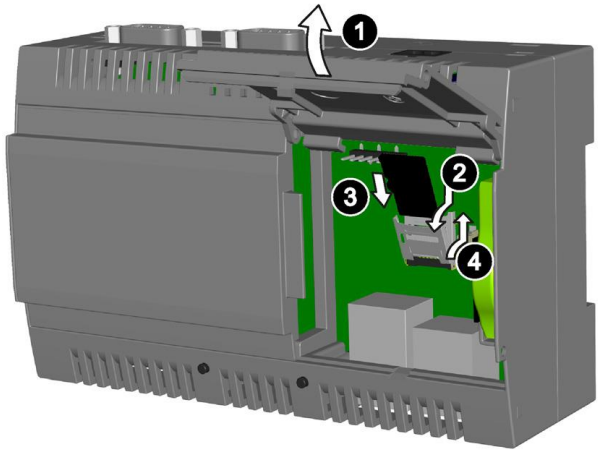
3.-	**Grabar la imagen de arranque.** Para ello, hay que seleccionar el archivo *.wic* y la letra de la unidad en la que está la tarjeta Micro SD y hacer click en *write*. Esperar a que finalice la operación*:*  *Figura 11. Grabando la imagen de arranque*
4.-	**Insertar la tarjeta Micro SD en el IOT2040.** Las instrucciones para hacerlo vienen detalladas en el manual proporcionado por el fabricante (adjunto)  *Figura 12. Insertando la tarjeta de memoria Micro SD*

Tabla 7. Pasos para la grabación de imagen Linux del IOT2040

Tras estos pasos, el SIMATIC IOT2040 debería leer la tarjeta Micro SD y cargar el sistema operativo escrito en ella. A continuación, se procederá a la configuración para las comunicaciones con el ordenador y el almacenamiento de los programas:

1.-	**Conectar el IOT2040 a una fuente de alimentación.** El primer arranque puede tomar hasta 5 minutos, ya que ajusta automáticamente el tamaño del sistema de archivos.
2.-	**Conectar el IOT2040 a un ordenador mediante cable Ethernet.** Se debe utilizar el puerto 1 del SIMATIC IOT2040, ya que tiene asignada una dirección IP por defecto.
3.-	**Iniciar la comunicación.** Para ello, hay que instalar el programa PuTTY descargado previamente. Utilizar la dirección IP por defecto: 192.168.200.1, guardar la configuración, y abrir la conexión. Aparecera un mensaje de advertencia acerca de una clave RSA2, hacer click en Yes. Se abrirá un terminal SSH que permitirá la conexión con el terminal Linux del IOT2040

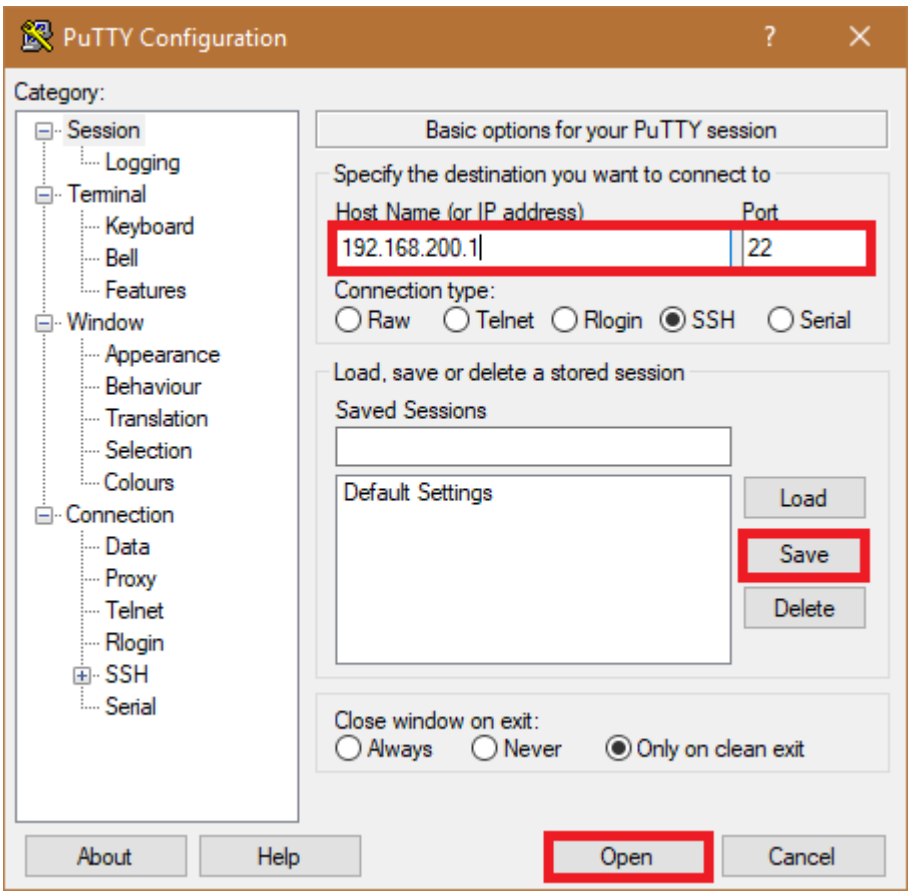


Figura 13. Iniciando la conexión mediante PuTTY

4.- **Cambiar la contraseña del usuario *root*.** Este paso es opcional, pero muy recomendable para evitar cualquier tipo de ataque proveniente de Internet. Primero, hay que iniciar sesión, por lo que hay que introducir '*root*'. Después se cambia la contraseña mediante el comando '*passwd*'.

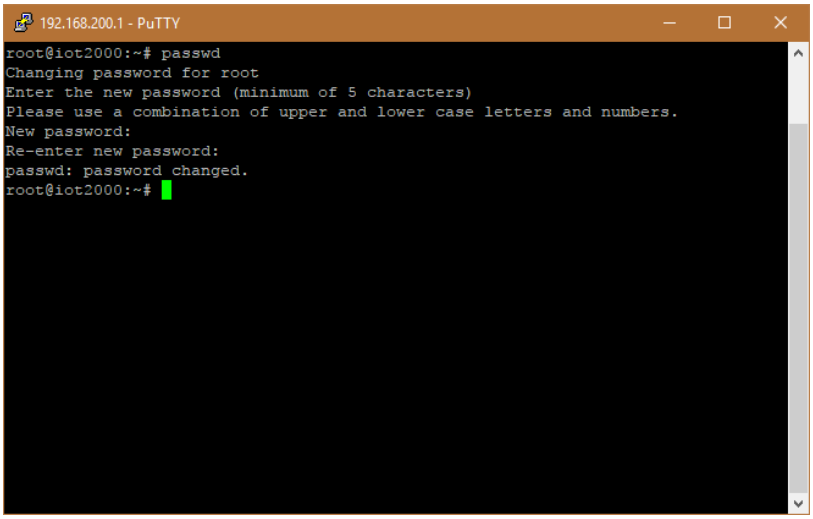


Figura 14. Configurando una contraseña mediante PuTTY

5.- **Cambiar la dirección IP por defecto.** Es recomendable para configurar la dirección IP del dispositivo según la red en la que se vaya a utilizar (por defecto, está asignado el protocolo *dhcp* para el puerto Ethernet 2). Para ello, introducir el comando '*iot2000setup*', ir a Networking->Configure Interfaces y seguir las instrucciones:

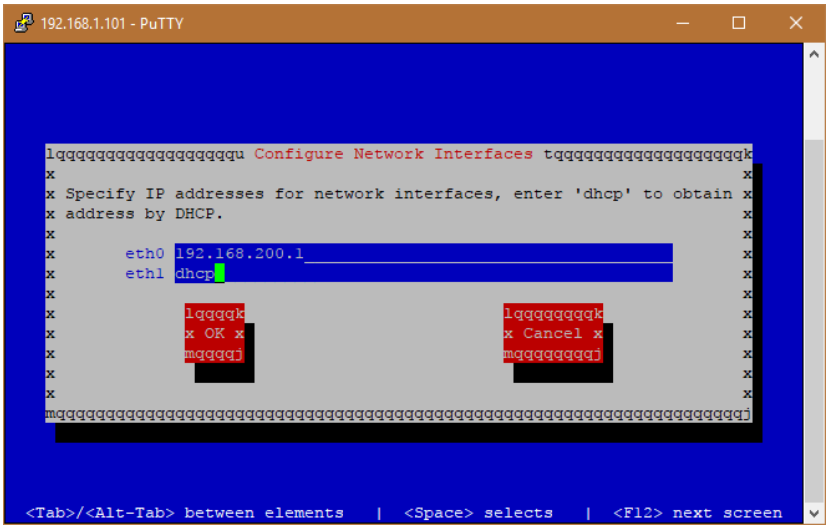


Figura 15. Cambiando la dirección IP por defecto mediante PuTTY

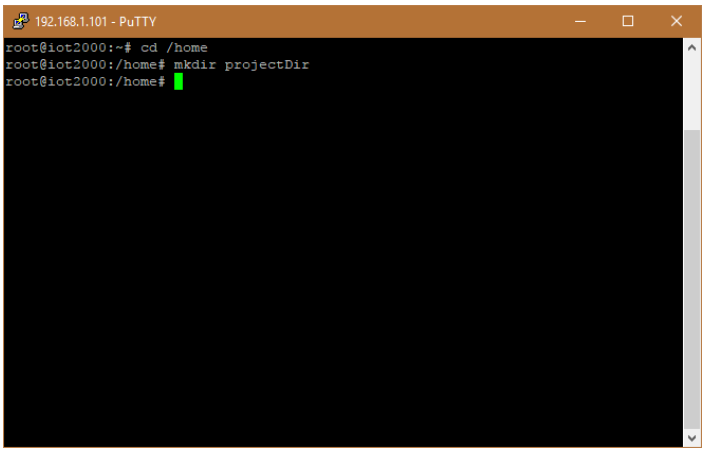
5.-	**Crear un directorio para el almacenamiento de los programas.** Se ha elegido una carpeta llamada '*projectDir*' dentro de '*/home'*. Para ello, se deben introducir los comandos de la siguiente figura:  *Figura 16. Creación del directorio de proyectos mediante PuTTY*

Tabla 8. Pasos para la configuración del SIMATIC IOT2040

4. Instalación y configuración del entorno de desarrollo (IDE)

Una vez seguidos los pasos del apartado anterior, ya se debería tener el SIMATIC IOT2040 completamente configurado y listo para ser programado mediante el IDE Eclipse. Por lo tanto, se dejará de lado el SIMATIC IOT2040 para proceder a la instalación y configuración del entorno de desarrollo.

1.-	**Instalar el IDE Eclipse.** Para ello, basta con extraer el archivo *.zip* de *Eclipse for C/C++* descargado previamente

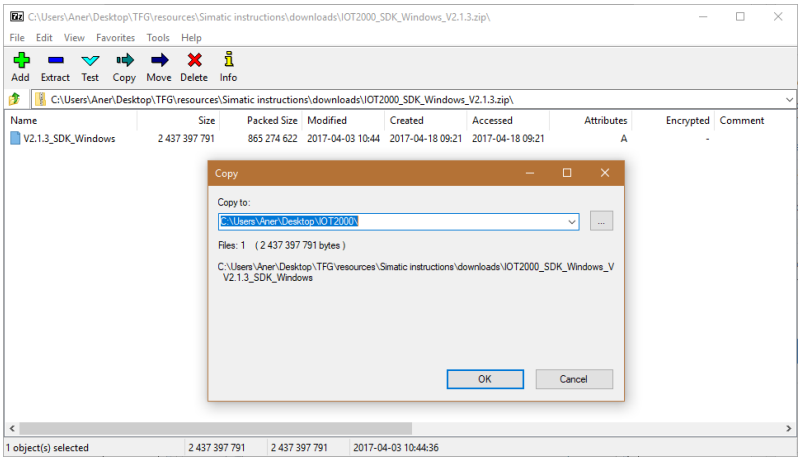
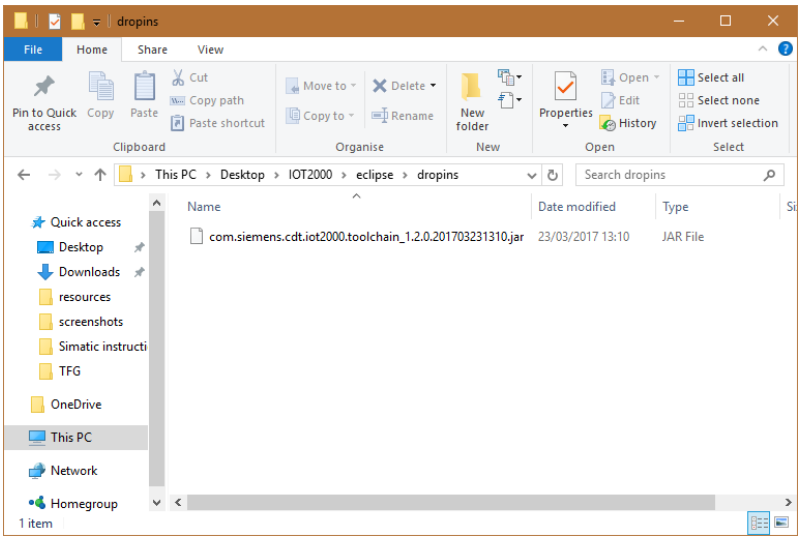
2.-	**Extraer el *SDK IOT2000*.** IMPORTANTE: Para ello, iniciar *7-Zip* como admnistrador, seleccionar '*Extraer*' y se obtendrá un archivo *.jar*. Extraer este último de nuevo y confirmar '*Sí a todo*' a la hora de sobrescribir. Ignorar las advertencias mostradas  *Figura 17. Extracción del SDK IOT2000 mediante 7-Zip*
3.-	**Extraer el *plug-in IOT2000* para *Eclipse*.** Ejecutando *7-Zip* como administrador de nuevo, extraer el archivo '*com.siemens.cdt.**.jar' del archivo *.zip* del plug-in descargado previamente en la carpeta '*dropins*' del directorio de *Eclipse* extraído previamente, de forma que resulte como en la siguiente figura:  *Figura 18. Extracción del plug-in IOT2000*

Tabla 9. Instalación del entorno de desarrollo

Tras completar esta tabla, ya se tiene el entorno de desarrollo completamente instalado con el SDK y el plug-in IOT2000, y sólo queda configurarlo para prepararlo para la programación del SIMATIC IOT2040.

1.-	**Iniciar *Eclipse*.** Hacer doble-click en el archivo '*eclipse.exe*'. Es posible que se requiera instalar *Java JDK* descargado previamente para su ejecución

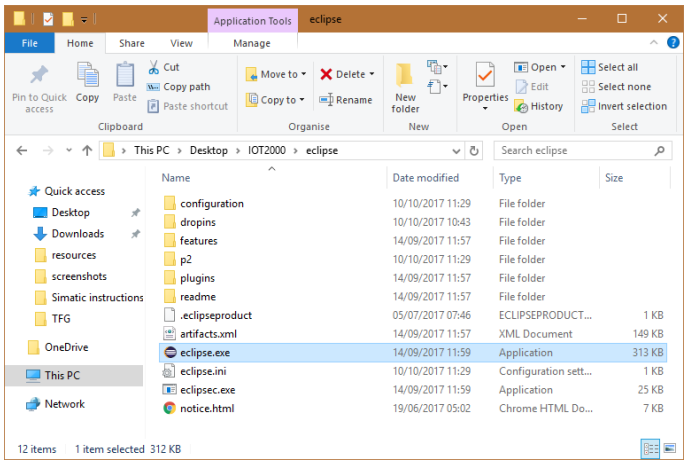


Figura 19. Abriendo Eclipse

2.-	**Añadir un espacio de trabajo.** Durante la primera ejecución, *Eclipse* solicitará un directorio en el que almacenar los proyectos
3.-	**Instalar *TCF Target Explorer*.** Necesario para que *Eclipse* pueda enviar los binarios al IOT2040, para ello ir a *Help->Install new software->Work with: All available sites->TCF Target Explorer*. Después de ello, seguir el asistente y reiniciar *Eclipse:*

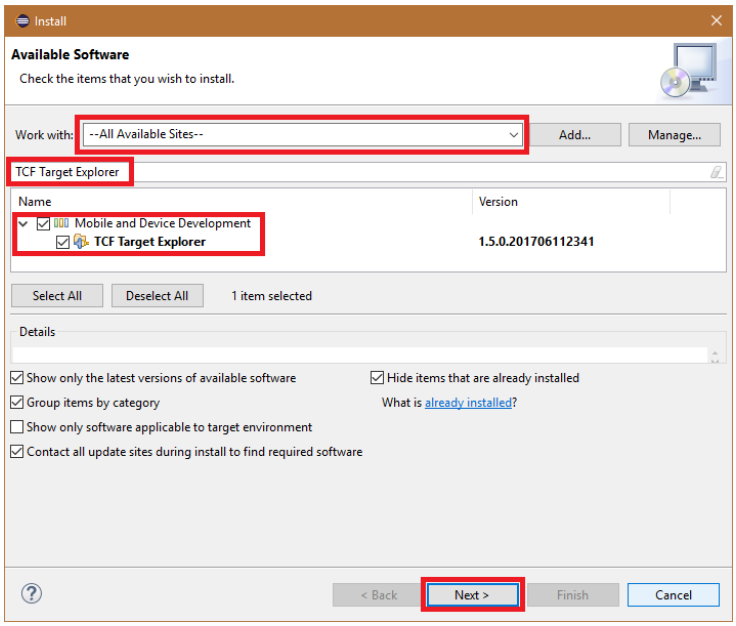


Figura 20. Instalando TCF Target Explorer

4.-	**Crear una conexión con el IOT2040.** Para ello, cerrar la pestaña de bienvenida y hacer click en *New connection…>Generic connection->Next*. Después, hacer click en '*browse*' y seleccionar la IP del IOT2040 (dependerá de la configuración previa). Por último click en '*Finish*'. Obviamente, el IOT2040 deberá estar conectado mediante Ethernet.

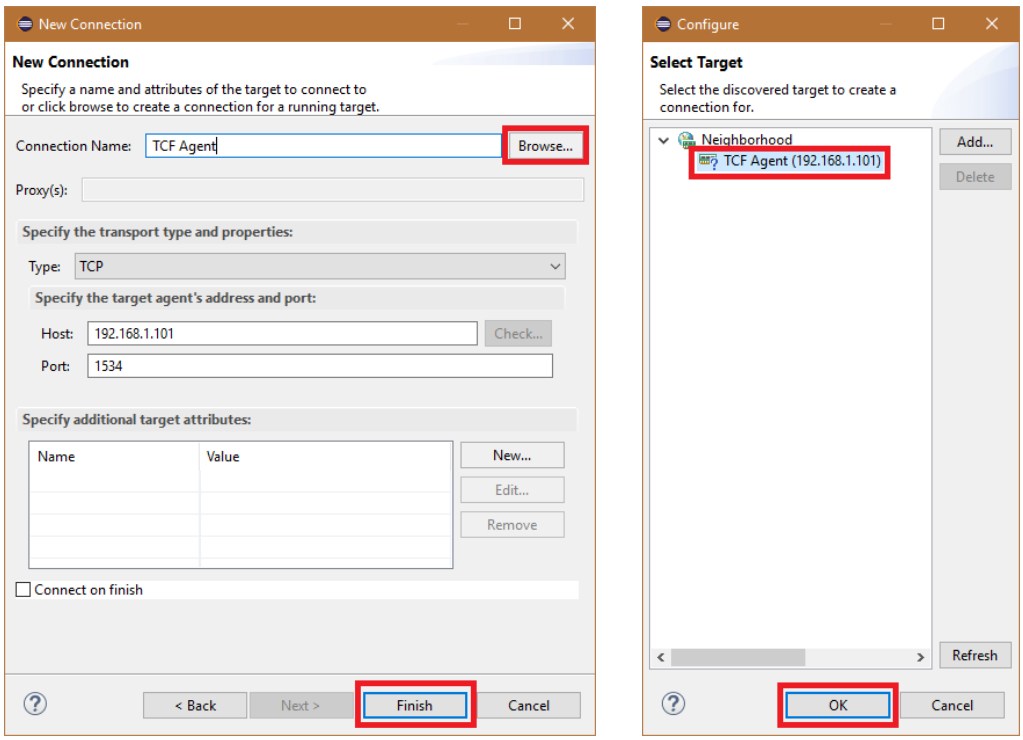


Figura 21. Creando una conexión con el IOT2040

Tabla 10. Configurando TCF Target Explorer en Eclipse

5. Creación de un programa *Simple Test*

En este punto, el ordenador ya dispone de un entorno de desarrollo completamente capaz de crear y compilar programas para al SIMATIC IOT2040. Por lo tanto, se creará un programa de prueba llamado *Simple Test* para comprobar que todo funciona correctamente. El código fuente de este programa está proporcionado a modo de plantilla por Siemens la cual vendrá incluida por defecto siempre que se cree un nuevo proyecto

1.-	**Crear un nuevo proyecto.** Ir a *File->New->Other…->Simatic IOT2000 project*

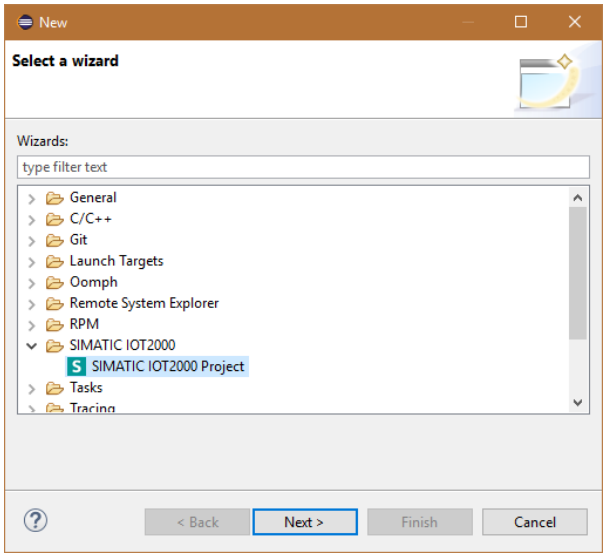


Figura 22. Creando un nuevo proyecto IOT2000 mediante Eclipse

2.-	**Configurar el nuevo proyecto.** Seleccionar un nombre para el proyecto y definir la ruta del SDK (donde se ha extraído previamente el archivo *.tar*). Elegir el lenguaje C++ y '*Finish*'. IMPORTANTE: Se debe utilizar '/' para definir la ruta del SDK, el símbolo por defecto '\' no es válido y resultará en compilaciones fallidas

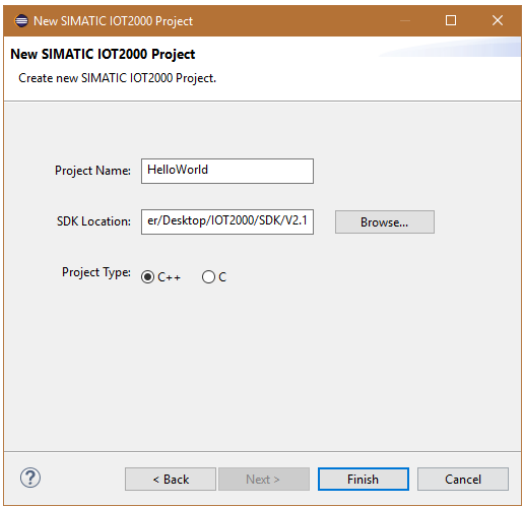


Figura 23. Configurando el nuevo proyecto IOT2000 mediante Eclipse

3.- **Compilar el proyecto.** Ir a '*Proyect Explorer*', hacer click derecho en el nombre del proyecto y seleccionar '*Build*'

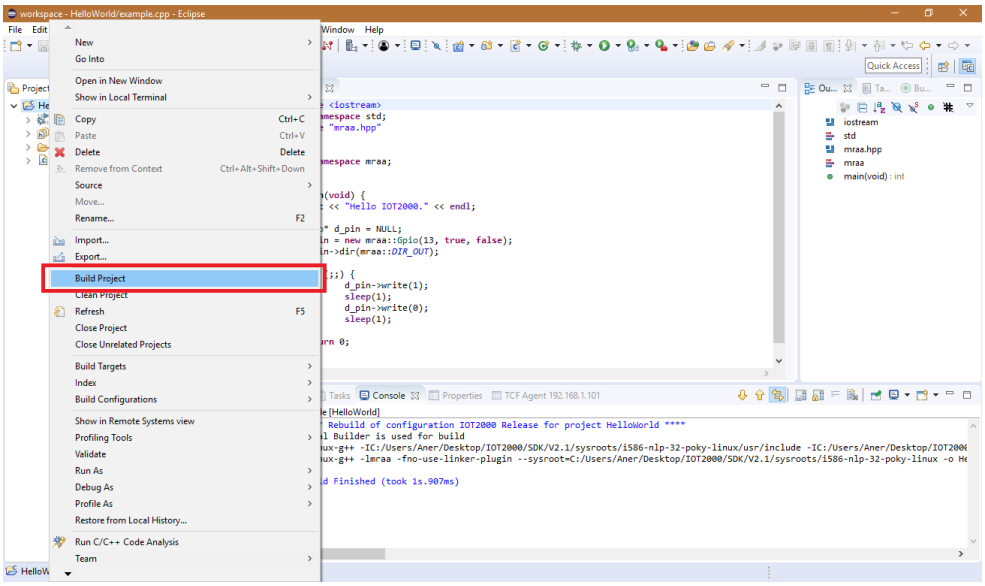


Figura 24. Compilando el proyecto

4.- **Ejecutar el proyecto.** Una vez compilado correctamente, ir a '*Proyect Explorer*', hacer click derecho en el nombre del proyecto y seleccionar *Run as…->Remote application*, y después:

1. En la pestaña '*File tranfers*' hacer click en '*Add*' y seleccionar '*Host to target*'. En '*Host*', seleccionar la ruta completa al binario con el nombre del proyecto en la carpeta '*IOT2000 Release*' de este. Si no existe, es necesario compilar el proyecto. En '*Target*', seleccionar la carpeta creada para almacenar proyectos en el IOT2040:

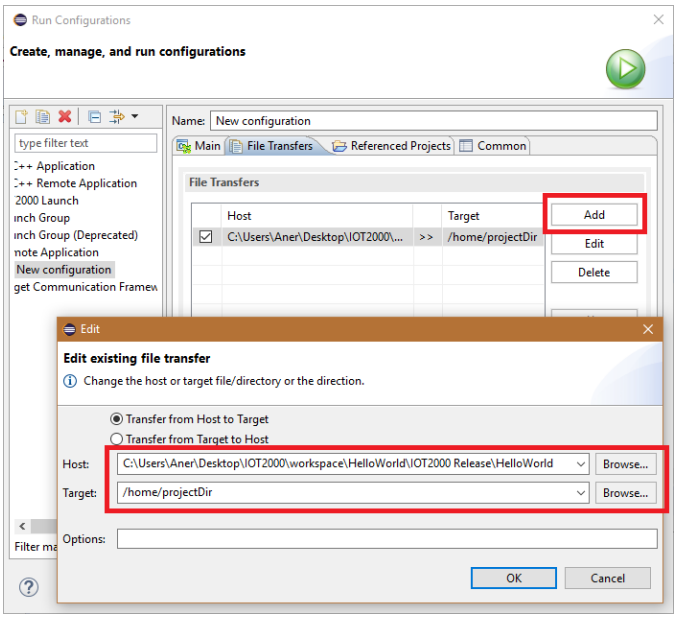


Figura 25. Definiendo la transferencia del binario

2. En la pestaña '*Main*', en '*Image*', seleccionar la ruta en la que se ha almacenado el binario generado, que debería estar en la misma carpeta que la definida para '*Target*' en el paso anterior. Si no está presente, hay que reconectar el IOT2040 y volver a compilar el programa

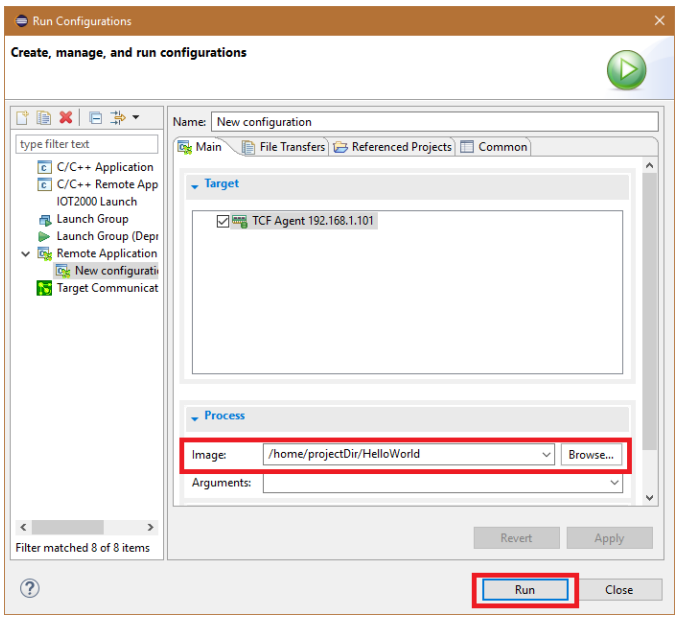


Figura 26. Seleccionando el binario transferido al IOT2040

5.- **Comprobar los resultados.** El programa debería escribir el mensaje definido por el ejemplo en el terminal, como se puede ver en la captura. En otros programas, el terminal se puede utilizar para definir argumentos para el programa o leer sus salidas. La explicación de las funciones ofrecidas por el SDK vienen detalladas en el Anexo

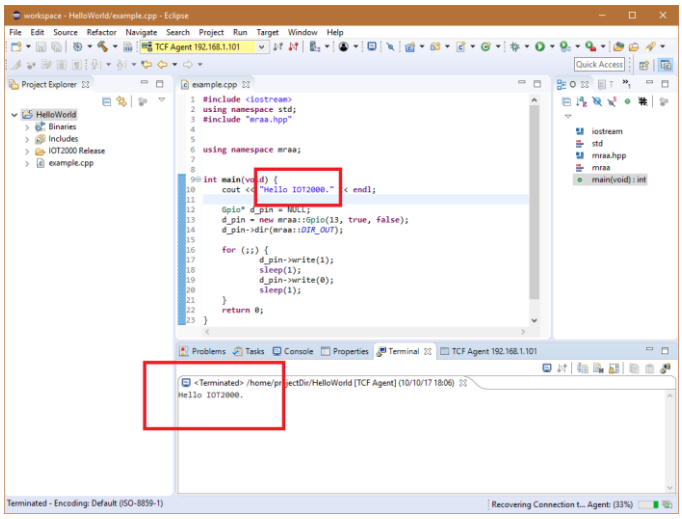


Figura 27. Mensaje de salida del programa

Tabla 11. Creación de un programa Simple Test

8. DESCRIPCIÓN DE LAS TAREAS. DIAGRAMA DE GANTT

El desarrollo de este Trabajo de Fin de Grado se puede dividir en las siguientes fases:

1. **Análisis de alternativas**
   Se buscan y estudian las distintas opciones disponibles en el mercado a nivel de hardware (compatibilidad, capacidad y potencia), a nivel de software y a nivel de conectividad, ya que son los principales requisitos para un proyecto de este tipo para así escoger la solución más adecuada. Esta fase tiene una duración de 15 días laborables.

2. **Establecer objetivos**
   En esta fase se definen los objetivos del proyecto, de forma que se pueda aclarar de qué trata, y cuál es la finalidad y el alcance del proyecto, para así estructurar de forma precisa los pasos a seguir para llevarlo a cabo. Esta fase tiene una duración de 5 días laborables.

3. **Estudio de viabilidad y aprendizaje**
   Antes de empezar con el diseño de la plataforma, es necesario hacer un estudio de la viabilidad de este proyecto, tanto en sentido económico como académico, ya que para su realización requiere de conocimientos de bajo nivel tanto de informática como de programación. Para ello, se analiza la documentación oficial disponible en la web del fabricante, y se analiza si los conocimientos cumplen con los requisitos necesarios para llevarlo a cabo. Esta fase tiene una duración de 10 días laborables.

4. **Diseño de la plataforma**
   Una vez verificada la viabilidad del proyecto, definidos los objetivos principales de éste, y habiendo comprendido el funcionamiento y las características del SIMATIC IOT2040, se define cómo se va a llevar a cabo la puesta en marcha y qué pruebas se van a realizar con el fin de asegurar el perfecto funcionamiento del sistema. Esta fase tiene una duración de 20 días laborables.

5. **Puesta en marcha del sistema de automatización**
   Tras tener la definición completa del proyecto, el siguiente paso es poner en marcha el sistema de automatización, para lo cual se deben descargar los archivos del sistema operativo y grabarlos posteriormente en la tarjeta Micro SD previamente formateada de acuerdo a la compatibilidad del dispositivo, de forma que los lea y se ejecute su arranque satisfactoriamente. Además, se debe preparar una fuente de alimentación adecuada para el dispositivo. Esta fase tiene una duración de 5 días laborables.

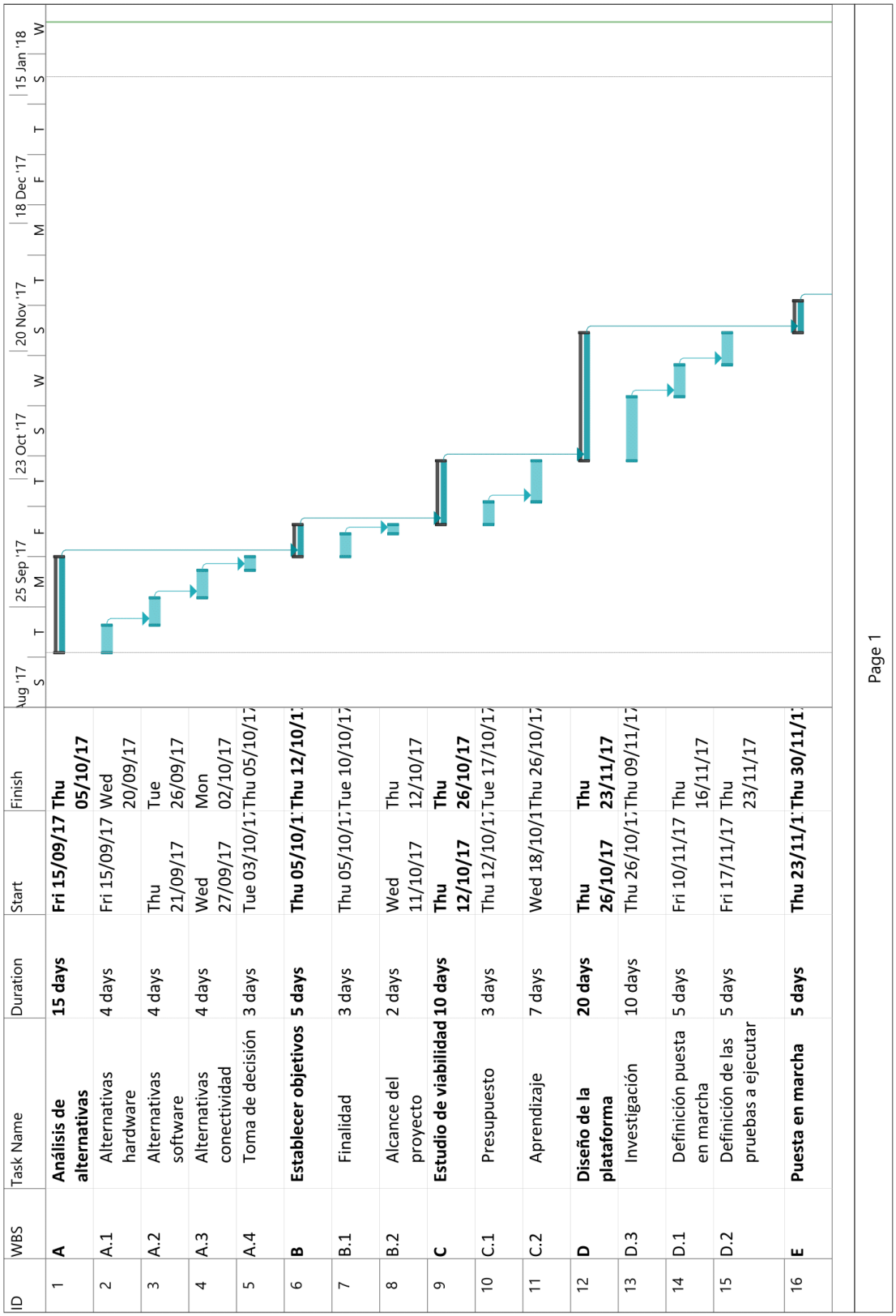
6. **Descarga e instalación del software de desarrollo**
   Una vez puesto en marcha, y verificando que el dispositivo ejecuta el sistema operativo durante el arranque con éxito, se procede a descargar e instalar el entorno de programación necesario para la creación de los programas que se van a llevar a cabo para ejecutar las pruebas. Esta fase tiene una duración de 5 días laborables.

7. **Fase de pruebas**
   Esta fase es fundamental para asegurar un correcto funcionamiento del sistema. Se trata de crear una serie de programas para aprender a utilizar las entradas y salidas del sistema tanto digitales como analógicas, además de las comunicaciones mediante RS232 y Ethernet y la aplicación del protocolo ModBus para la transmisión de información entre varios dispositivos. Para ello, este proyecto se apoyará en un dispositivo Arduino UNO, capaz de leer e intercambiar información con el SIMATIC IOT2040. Estos programas vendrán definidos en el anexo. Esta fase tiene una duración de 30 días laborables.

8. **Redacción de la memoria**
   El Trabajo de Fin de Grado finaliza con la redacción de la memoria del proyecto, tras haber realizado la parte práctica, asegurando y garantizando el correcto funcionamiento del sistema de automatización, cumpliendo los objetivos establecidos, y habiendo seguido el diseño definido en las fases previas. Esta fase tiene una duración de 30 días laborables.

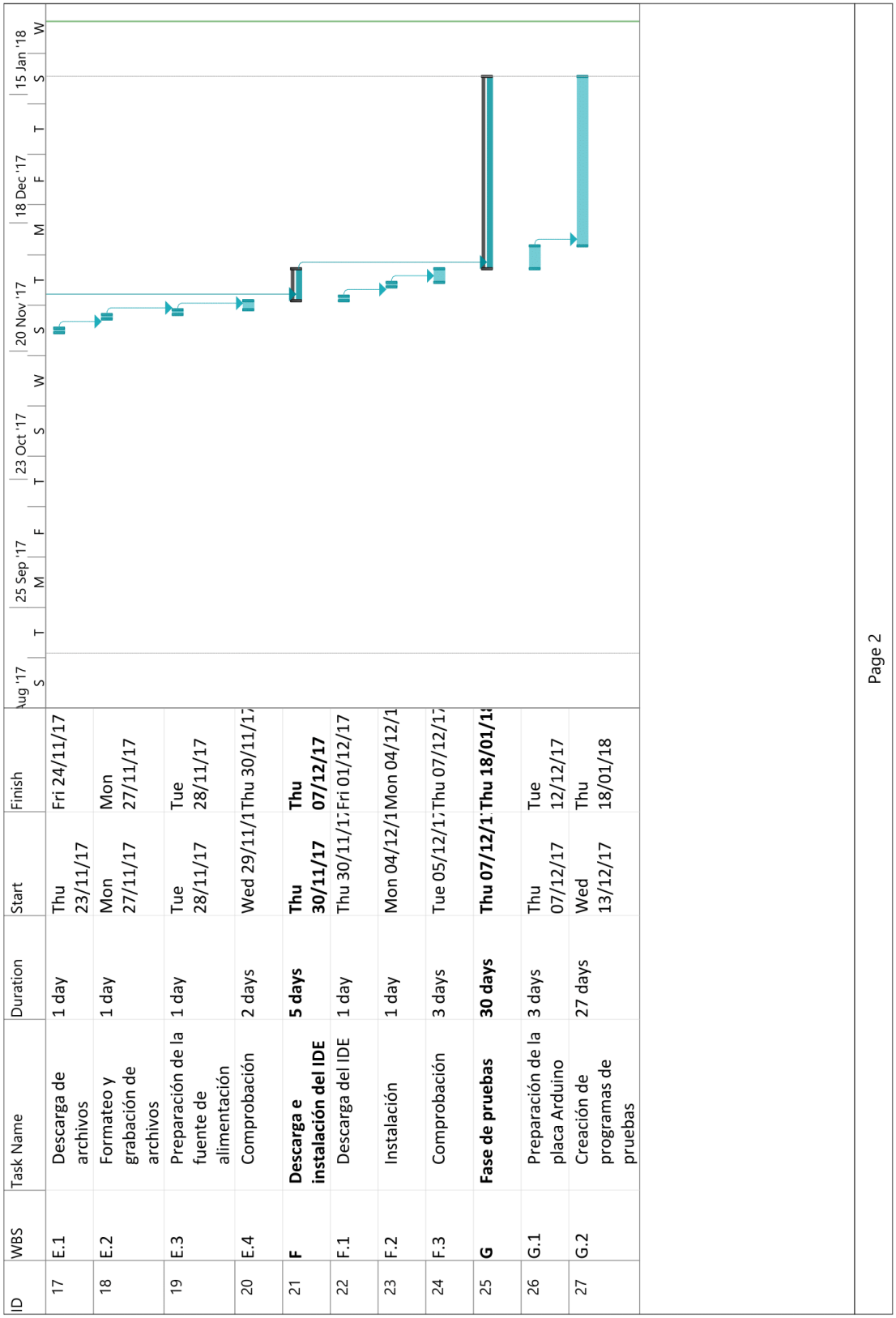
El proyecto comenzó el día 15 de septiembre de 2017 y finalizó el día 31 de enero de 2018.

El Trabajo de Fin de Grado lleva un total de 120 días laborables para su ejecución, lo cual supone 180 horas de trabajo considerando 1.5 horas por día de trabajo.



ID	WBS	Task Name	Duration	Start	Finish
1	**A**	**Análisis de alternativas**	**15 days**	**Fri 15/09/17**	**Thu 05/10/17**
2	A.1	Alternativas hardware	4 days	Fri 15/09/17	Wed 20/09/17
3	A.2	Alternativas software	4 days	Thu 21/09/17	Tue 26/09/17
4	A.3	Alternativas conectividad	4 days	Wed 27/09/17	Mon 02/10/17
5	A.4	Toma de decisión	3 days	Tue 03/10/17	Thu 05/10/17
6	**B**	**Establecer objetivos**	**5 days**	**Thu 05/10/17**	**Thu 12/10/17**
7	B.1	Finalidad	3 days	Thu 05/10/17	Tue 10/10/17
8	B.2	Alcance del proyecto	2 days	Wed 11/10/17	Thu 12/10/17
9	**C**	**Estudio de viabilidad**	**10 days**	**Thu 12/10/17**	**Thu 26/10/17**
10	C.1	Presupuesto	3 days	Thu 12/10/17	Tue 17/10/17
11	C.2	Aprendizaje	7 days	Wed 18/10/17	Thu 26/10/17
12	**D**	**Diseño de la plataforma**	**20 days**	**Thu 26/10/17**	**Thu 23/11/17**
13	D.3	Investigación	10 days	Thu 26/10/17	Thu 09/11/17
14	D.1	Definición puesta en marcha	5 days	Fri 10/11/17	Thu 16/11/17
15	D.2	Definición de las pruebas a ejecutar	5 days	Fri 17/11/17	Thu 23/11/17
16	**E**	**Puesta en marcha**	**5 days**	**Thu 23/11/17**	**Thu 30/11/17**

Figura 28. Diagrama de Gantt (1 de 3)



ID	WBS	Task Name	Duration	Start	Finish
17	E.1	Descarga de archivos	1 day	Thu 23/11/17	Fri 24/11/17
18	E.2	Formateo y grabación de archivos	1 day	Mon 27/11/17	Mon 27/11/17
19	E.3	Preparación de la fuente de alimentación	1 day	Tue 28/11/17	Tue 28/11/17
20	E.4	Comprobación	2 days	Wed 29/11/17	Thu 30/11/17
21	**F**	**Descarga e instalación del IDE**	**5 days**	**Thu 30/11/17**	**Thu 07/12/17**
22	F.1	Descarga del IDE	1 day	Thu 30/11/17	Fri 01/12/17
23	F.2	Instalación	1 day	Mon 04/12/17	Mon 04/12/17
24	F.3	Comprobación	3 days	Tue 05/12/17	Thu 07/12/17
25	**G**	**Fase de pruebas**	**30 days**	**Thu 07/12/17**	**Thu 18/01/18**
26	G.1	Preparación de la placa Arduino	3 days	Thu 07/12/17	Tue 12/12/17
27	G.2	Creación de programas de pruebas	27 days	Wed 13/12/17	Thu 18/01/18

Figura 29. Diagrama de Gantt (2 de 3)

Figura 30. Diagrama de Gantt (3 de 3)

9. PRESUPUESTOS

A la hora de elaborar el presupuesto del Trabajo de Fin de Grado, se ha decidido clasificarlo en tres partidas: horas internas, amortizaciones, y gastos, incluyendo así un resumen con el sumatorio de las tres partidas y el coste total del proyecto al final de este apartado:

HORAS INTERNAS			
Concepto	**Coste unitario (€/h)**	**Número de unidades (h)**	**Coste**
Ingeniero técnico	30	180	5400.00 €
TOTAL			**5400.00 €**

Tabla 12. Presupuesto: Horas internas

AMORTIZACIONES				
Concepto	**Precio de adquisición (€)**	**Vida útil (años)**	**Utilización (h)**	**Coste**
Ordenador	800	5 *(9125 h/año)*	180	15.78 €
TOTAL				**15.78 €**

Tabla 13. Presupuesto: Amortizaciones

GASTOS			
Concepto	**Coste unitario (€/ud)**	**Número de unidades**	**Coste**
SIEMENS SIMATIC IOT2040	180	1	180 €
Arduino UNO	21	1	21 €
Arduino Ethernet shield	22	1	22 €
Cable Ethernet RJ45	9.31	2	18.62 €
Fuente de alimentación 12V	24.83	1	24.83 €
Cable USB Type A	2.49	1	2.49 €
Protoboard y otros materiales de electrónica	25.89	1	25.89 €
TOTAL			**294.83 €**

Tabla 14. Presupuesto: Gastos

RESUMEN	
Horas internas	5400.00 €
Amortizaciones	15.78 €
Gastos	294.83 €
SUBTOTAL	**5710.61 €**
Costes derivados (3%)	171.32 €
TOTAL	**5881.93 €**

Tabla 15. Presupuesto: Resumen

Cabe destacar que para realizar estos presupuestos, no se han incluido los periféricos del ordenador tales como el monitor y el teclado, ya que su coste de amortización resulta despreciable y, además solo resultarían necesarios en caso de tener que reprogramar el SIMATIC IOT2040.

A la hora de calcular el coste final del presupuesto, se ha añadido un 3% de costes derivados la realización del proyecto tales como la electricidad, o la conexión a Internet.

En el siguiente gráfico se muestra la ponderación de cada partida del presupuesto:

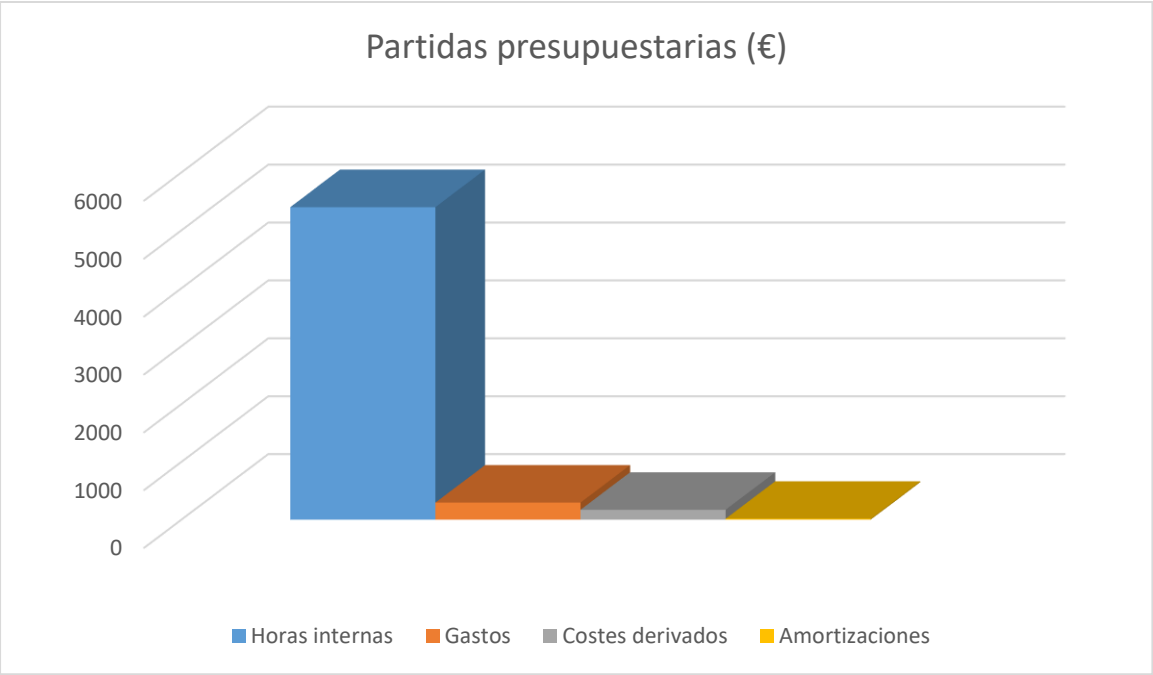


Figura 31. Partidas presupuestarias

Como se puede ver, la partida con mayor peso sobre el total, se debe a las horas internas ya que, debido a la necesidad de investigar, aprender, diseñar, y desarrollar, se ha requerido una gran cantidad de horas de trabajo. Además, cabe destacar el bajo coste en cuanto a materiales que supone este proyecto, demostrando así que se cumple una de las principales ventajas de este proyecto gracias, en parte, al bajo coste del SIMATIC IOT2040 y a la ausencia de necesidad de adquisición de licencias de software u otro tipo.

10. CONCLUSIONES

En este documento, se ha detallado cómo se lleva a cabo la puesta en marcha de un sistema de automatización basado en el dispositivo SIMATIC IOT2040 de SIEMENS, que brinda la posibilidad de dotar a este sistema de conexión a otros dispositivos en una red o incluso a Internet, de forma que sea posible acceder a los datos de un proceso desde cualquier parte del mundo, acercando el concepto de la industria actual hacia la nueva revolución ya en marcha llamada *Industria 4.0*.

Además, el bajo coste que supone este sistema, hace que la implementación de este sistema sea factible para prácticamente cualquier tamaño de negocio. Cabe destacar que el bajo coste de este sistema no sólo se debe precisamente al bajo coste del propio dispositivo y su sencillez, sino también a la no necesidad de adquirir ningún tipo de licencias a la hora de utilizar su entorno de programación, que además es completamente gratuito.

El hecho de ser compatible con un funcionamiento basado en Linux, permite otorgar al sistema de mucha mayor flexibilidad, ya que este es un sistema operativo de software libre y cualquiera puede acceder a él, aplicar sus propias personalizaciones y crear sus propias compilaciones, abriendo las puertas a implementar nuevos protocolos de comunicación, y también a una compatibilidad con prácticamente un número infinito de dispositivos disponibles en el mercado, reforzando el carácter modular de este sistema. Como consecuencia de este hecho, resulta obvio que es posible hacer el sistema compatible con una gran cantidad de estándares industriales.

Un sistema de automatización de este tipo, no sólo sirve para negocios a nivel industrial, sino que también resulta de gran utilidad en otros campos como pueden ser la domótica o incluso el campo de la docencia.

Por último, se debe tener en cuenta que se predice que el futuro de la automatización industrial pase por el uso de distintas plataformas modulares, de forma que se pueda proveer a los distintos procesos que se llevan a cabo de un gran nivel de flexibilidad y adaptabilidad. Estas plataformas modulares son conjuntos de hardware y software que forman parte del nuevo concepto *Industria 4.0*.

11. FUENTES DE INFORMACIÓN. BIBLIOGRAFÍA

[1] Siemens. *SIMATIC IOT2040: La pasarela inteligente para las soluciones de IoT industrial.*

https://w5.siemens.com/spain/web/es/industry/automatizacion/simatic/PC_industriales/Pages/SIMATIC-IOT2040.aspx

[2] Siemens. *SIMATIC IOT2000 support forums.*

https://support.industry.siemens.com/tf/ww/en/conf/60/

[3] Siemens. *Setting up the SIMATIC IOT2000.*

https://support.industry.siemens.com/tf/ww/en/posts/setting-up-the-simatic-iot2000/155642

[4] Siemens. *Getting started with SIMATIC IOT2000.*

https://support.industry.siemens.com/tf/ww/en/posts/getting-started-with-simatic-iot2000/155643

[5] Intel. *Developer Zone. IOT home.*

https://software.intel.com/en-us/iot/home

[6] Intel. m*raa library documentation. Low Level Skeleton Library for Communication on GNU/Linux platforms*

http://iotdk.intel.com/docs/master/mraa/

[7] Intel, GitHub. *Open source C++ mraa library examples.*

https://github.com/intel-iot-devkit/mraa/tree/master/examples/

[8] ModBus. *ModBus library documentation.*

http://libmodbus.org/docs/v3.0.6/

[9] Arduino. *Arduino language reference.*

https://www.arduino.cc/reference/en/

[10] Intel. *Intel Galileo Gen 2 Board specs*

https://ark.intel.com/products/83137/Intel-Galileo-Gen-2-Board

[11] Wikipedia. *Industry 4.0.*

https://en.wikipedia.org/wiki/Industry_4.0

[12] Aner Torre, GitHub. *Open source SIMATIC IOT2040 example programs*

https://github.com/bamsbamx/FinalDegreeProject

12. ANEXOS

Este apartado tiene como finalidad mostrar ciertos programas que se han creado y utilizado durante la ejecución de este proyecto con el objetivo de demostrar que la puesta en marcha del sistema ha sido efectivamente satisfactoria. Estos programas también sirven como ejemplo o plantilla a la hora de crear otros programas con fines relacionados. Para cada programa de este apartado se proporcionará una breve descripción acerca de su funcionamiento y objetivos además de tener líneas con comentarios explicando la finalidad de cada sentencia.

Cabe destacar que estos programas son de código abierto y están disponibles a través de un repositorio Git disponible en: *https://github.com/bamsbamx/FinalDegreeProject*. Este repositorio cuenta con varias ramas (branch), en la que cada una contiene el código de cada hito o programa de ejemplo.

ANEXO 1: PROBANDO LAS ENTRADAS/SALIDAS

Este anexo contiene un simple programa que consiste en utilizar un pin de salida digital para iluminar un LED y otro de entrada analógica, para leer el valor de un potenciómetro. Su finalidad es verificar que efectivamente el SIMATIC IOT2040 está en marcha, y se ha logrado una comunicación entre él y la estación de programación. Para ello, lo que hace este programa es cambiar el estado de encendido de un LED, leer el valor de un potenciómetro (en rango de 0 a 1023 y de 0 a 1, siendo el valor máximo 5V y el mínimo 0), y enviar un mensaje con el valor leído a la estación de trabajo.

Conexiones:

- El SIMATIC IOT2040 debe estar conectado a la fuente de alimentación de 12V DC
- El SIMATIC IOT2040 debe estar conectado mediante su puerto de Ethernet #1 con la estación de programación a través de un cable RJ-45
- Partiendo del GPIO D0 un LED, en serie con una resistencia de 1KOhm conectada al pin GND
- Un potenciómetro entre el pin 5V y el pin GND, conectando su salida a la entrada analógica al pin A0

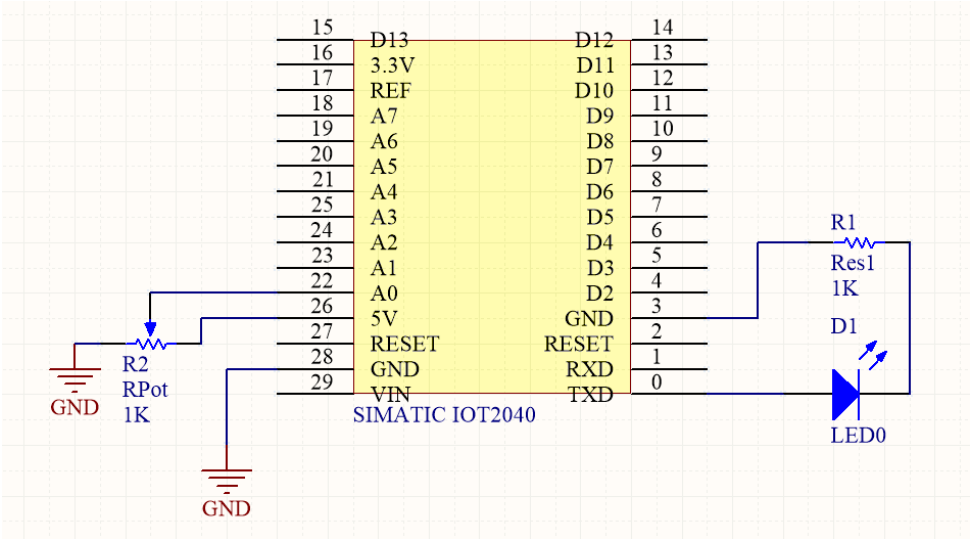


Figura 32. Anexo 1: Conexiones necesarias

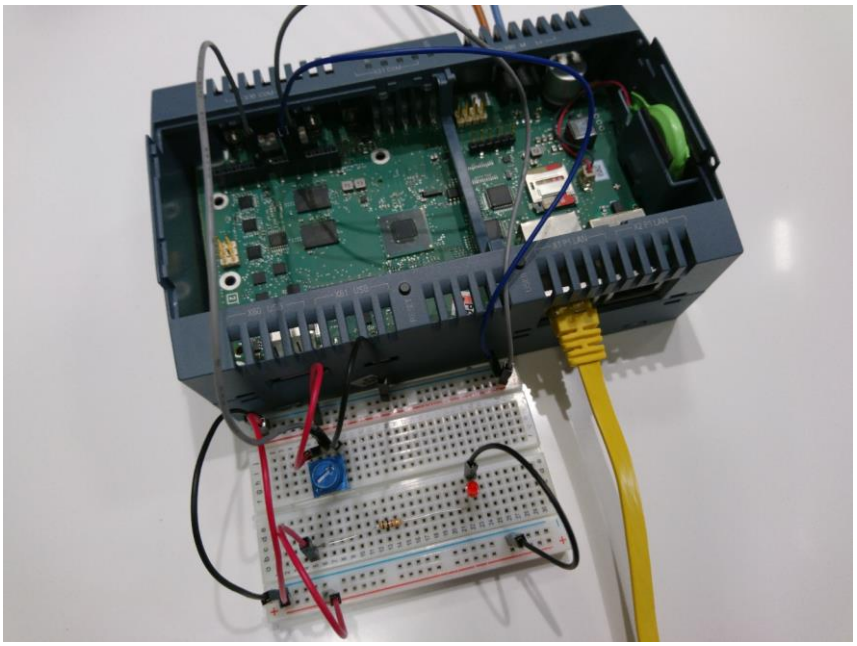


Figura 33. Anexo 1: Prueba con LED y potenciómetro

Programa:

```
#include <iostream>
#include "mraa.hpp"

using namespace std;
using namespace mraa;

int main(void) {

	// Mostrar un mensaje de inicio
	cout << "Program initialized." << endl;

	// Inicializar variables
	uint16_t value1;
	float value2;

	// Inicializar los pines de entrada y salida
	Gpio led_gpio(0);
	Aio pot_aio(0);

	// Definir el pin D0 como pin de salida
	led_gpio.dir(mraa::DIR_OUT);
	// Apagar el led en caso de estar encendido antes de la ejecucion del
programa
	led_gpio.write(0);

	// Bucle del programa
	for (;;) {
			// Encender LED
			led_gpio.write(1);
```

```
                    sleep(1);
                    // Apagar LED
                    led_gpio.write(0);
                    sleep(1);
                    // Leer la tension de entrada en el pin A0 (desde 0 hasta
1023)
                    value1 = pot_aio.read();
                    // Leer la tension de entrada en el pin A0 (desde 0 hasta 1)
                    value2 = pot_aio.readFloat();
                    // Mostrar dos mensajes con los valores leidos
                    cout << "A0 read %d" << value1 << endl;
                    cout << "A0 read %.5f" << value2 << endl;
                    sleep(1);
        }

        return 0;
}
```

ANEXO 2: COMPATIBILIDAD CON EL SHIELD ARDUINO TINKERKIT

Este anexo contiene el procedimiento llevado a cabo para utilizar el *shield* para Arduino *TinkerKit!* (hay que recordar que la disposición de pines del SIMATIC IOT2040 es la misma que la del estándar Arduino. Por lo tanto, la compatibilidad está asegurada en cuanto a hardware). Para ello, la idea es utilizar las librerías existentes para el uso de este *shield*. Sin embargo, éstas están adaptadas para el código fuente de Arduino, por lo que se requiere hacer ciertas modificaciones para que estas librerías puedan ser compiladas para el SIMATIC IOT2040. Para ello, es necesario cambiar las llamadas a las funciones que esta hace (definidas en Arduino), para utilizar aquellas que vienen definidas en la librería *mraa,* así como los constructores y las variables predefinidas. La adaptación de esta librería se mostrará más adelante en este anexo.

También se ha creado un programa para comprobar el funcionamiento de esta adaptación. En el que, de manera análoga al anexo anterior, se cambiará el estado de encendido de un LED y se leerá el valor de un potenciómetro cada segundo, esta vez estando ambos conectados al shield en lugar del SIMATIC IOT2040.

Conexiones:

- El SIMATIC IOT2040 debe estar conectado a la fuente de alimentación de 12V DC
- El SIMATIC IOT2040 debe estar conectado mediante su puerto de Ethernet #1 con la estación de programación a través de un cable RJ-45
- Con el *TinkerKit!* Shield en el SIMATIC IOT2040, conectar un LED de *TinkerKit!* en el pin O0 del shield. Tener en cuenta que la resistencia ya está integrada en los LED de *TinkerKit!*, por lo que no es necesario añadirla a la conexión
- Con el *TinkerKit!* Shield en el SIMATIC IOT2040, conectar un potenciómetro de *TinkerKit!* en el pin I0 del shield

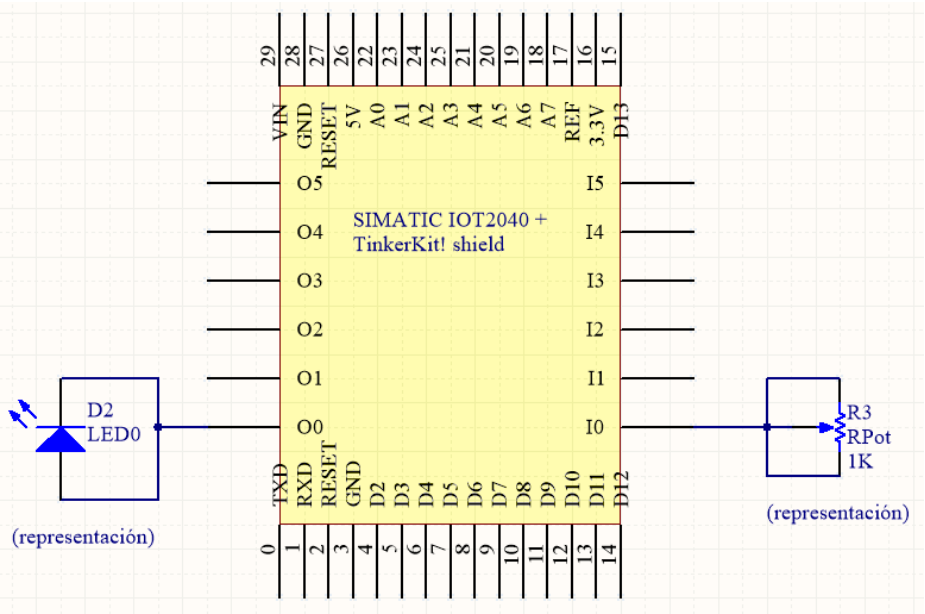


Figura 34. Anexo 2: Conexiones necesarias

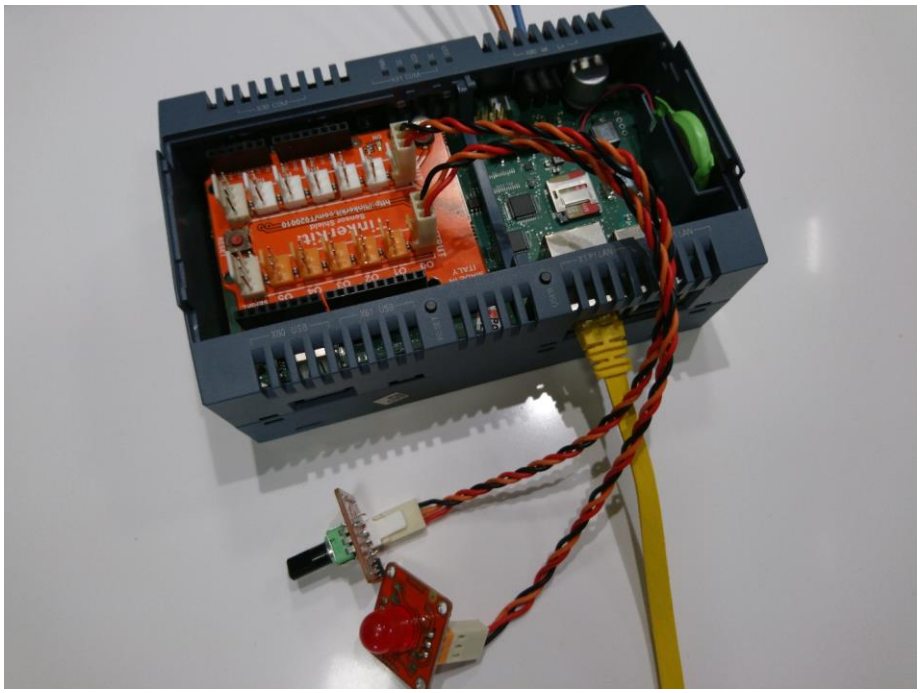


Figura 35. Anexo 2: Prueba con el Shield TinkerKit! y sus accesorios

Programa:

- Librerías adaptadas:
    - tinkerkit.cpp

```
#include <iostream>
#include <Math.h>      // PI
#include "TinkerKit.hpp"
#include <unistd.h>    // usleep()

TKInputDigital::TKInputDigital(uint8_t _pin) {
    mPin = NULL;
    mPin = new mraa::Aio(_pin);
    if (mPin == NULL) throw MRAA_ERROR_UNSPECIFIED;
}

TKInputAnalog::TKInputAnalog(uint8_t _pin) {
    mPin = NULL;
    mPin = new mraa::Aio(_pin);
      if (mPin == NULL) throw MRAA_ERROR_UNSPECIFIED;
}

TKInputAnalog2::TKInputAnalog2(uint8_t _pinX, uint8_t _pinY) {
      mPinX = NULL;
      mPinX = new mraa::Aio(_pinX);
      if (mPinX == NULL) throw MRAA_ERROR_UNSPECIFIED;
      mPinY = NULL;
      mPinY = new mraa::Aio(_pinY);
      if (mPinY == NULL) throw MRAA_ERROR_UNSPECIFIED;
      mPinZ = NULL;
}
```

```
TKInputAnalog2::TKInputAnalog2(uint8_t _pinX, uint8_t _pinY, uint8_t _pinZ) {
       mPinX = NULL;
       mPinX = new mraa::Aio(_pinX);
       if (mPinX == NULL) throw MRAA_ERROR_UNSPECIFIED;
       mPinY = NULL;
       mPinY = new mraa::Aio(_pinY);
       if (mPinY == NULL) throw MRAA_ERROR_UNSPECIFIED;
       mPinZ = NULL;
       mPinZ = new mraa::Aio(_pinZ);
       if (mPinZ == NULL) throw MRAA_ERROR_UNSPECIFIED;
}

TKOutput::TKOutput(uint8_t _pin) {
    _state = LOW;
       mPin = NULL;
       mPin = new mraa::Pwm(_pin);
       if (mPin == NULL) throw MRAA_ERROR_UNSPECIFIED;
       mPin->enable(true); // Initialise as output with low state
}


// Returns 0 or 1
boolean TKInputDigital::read() {
    return mPin->read() > 512 ? HIGH : LOW;
}

int TKInputAnalog::read() {
    return mPin->read(); // 0..1023
}

int TKInputAnalog2::readX() {
    int val = mPinX->read(); // 0..1023
    return val;
}

int TKInputAnalog2::readY() {
    return mPinY->read(); // 0..1023
}

int TKInputAnalog2::readZ() {
    return mPinZ->read(); // 0..1023
}

void TKOutput::write(float value) {
    if (value <= TK_MAX && value >= 0) {
       float actualValue = map(value, 0, 1023, 0, 1.0f);
       mPin->write(actualValue);
    } else throw;
}

/*      Button      */
TKButton::TKButton(uint8_t _pin) : TKInputDigital(_pin) {}

/*      Tilt Sensor         */
TKTiltSensor::TKTiltSensor(uint8_t _pin) : TKInputDigital (_pin){}

/*      Touch Sensor        */
```

```
 TKTouchSensor::TKTouchSensor(uint8_t _pin) : TKButton(_pin) {}


/*
 ------------------------------------------------------------------------------
                                Analog Inputs
 ------------------------------------------------------------------------------
 */

/*      Potentiometer       */
TKPotentiometer::TKPotentiometer(uint8_t _pin) : TKInputAnalog(_pin) {}

int TKPotentiometer::read() {
    return TKInputAnalog::read(); // From 0 to 1023
}

int TKPotentiometer::readStep(int steps) {
       return floor(map(read(), 0, 1023, 0, steps)); // From 0 to 'steps'

}

/*      Light Sensor        */
TKLightSensor::TKLightSensor(uint8_t _pin) : TKInputAnalog(_pin){}

/*      Temperature Sensor       */
TKThermistor::TKThermistor(uint8_t _pin) : TKInputAnalog(_pin) {}

float TKThermistor::readCelsius() {
       float Rthermistor = Rb * (ADCres / TKThermistor::read() - 1);
       float _temperatureC = Beta / (log( Rthermistor * Ginf )) ;
       return _temperatureC - Kelvin;
}

float TKThermistor::readFahrenheit() {
       float _temperatureF = (TKThermistor::readCelsius() * 9.0)/ 5.0 + 32.0;
       return _temperatureF;
}


/*      Hall Sensor     */
TKHallSensor::TKHallSensor(uint8_t _pin) : TKInputAnalog(_pin) {}

boolean TKHallSensor::polarity() {
       int value = read();
       if(value >= _zeroValue) return NORTH;
       else return SOUTH;
}

/*      Joystick        */
TKJoystick::TKJoystick(uint8_t _pinX, uint8_t _pinY) : TKInputAnalog2 (_pinX,
_pinY) {}

int TKJoystick::readX() {
    return TKInputAnalog2::readX();
}

int TKJoystick::readY() {
    return TKInputAnalog2::readY();
```

```
}


/*      Gyro        */
TKGyro::TKGyro(uint8_t _pinX, uint8_t _pinY, boolean _model) : TKInputAnalog2
(_pinX, _pinY) {
       _sensitivityInCount = 14633;  // 4.88mV / (0.167mV/dps * 2)
       model = _model;
       if (_model == TK_X4) _sensitivityInCount /= 4;
    // default values
    _xZeroVoltage = 503;  // 2.46V expressed in ADC counts
    _yZeroVoltage = 503;
}

void TKGyro::calibrate() {
       _xZeroVoltage = 0;
       _yZeroVoltage = 0;

       for (uint8_t i=0; i<50; i++) {
              _yZeroVoltage += readY();
              _xZeroVoltage += readX();
              usleep(20 * 1000); // 20ms
    }
   _yZeroVoltage /= 50;
   _xZeroVoltage /= 50;
}

long TKGyro::readXAxisRate() {
       return ((long)(readX() - _xZeroVoltage) * _sensitivityInCount) / 1000;
}

long TKGyro::readYAxisRate() {
       return ((long)(readY() - _yZeroVoltage) * _sensitivityInCount) / 1000;
}


/*      Accelerometer        */
TKAccelerometer::TKAccelerometer(uint8_t _pinX, uint8_t _pinY) :
TKInputAnalog2(_pinX,_pinY){}
TKAccelerometer::TKAccelerometer(uint8_t _pinX, uint8_t _pinY, uint8_t _pinZ) :
TKInputAnalog2(_pinX,_pinY,_pinZ){}

int TKAccelerometer::inclination() {
       int xVal = readX() - _zeroOffset;
       int yVal = readY() - _zeroOffset;

       if(xVal <= 96 && yVal <= 96) {
              int inclination = atan2(xVal, yVal)*180/M_PI;
              return (int) inclination;
       } else return 0; //TODO Test
}

/*
------------------------------------------------------------------------------
                                        Outputs
------------------------------------------------------------------------------
*/
```

```
/*      LED      */
TKLed::TKLed(uint8_t _pin) : TKOutput(_pin) {}


/*      MosFet      */
TKMosFet::TKMosFet(uint8_t _pin) : TKOutput(_pin) {}


/*      Relay      */
TKRelay::TKRelay(uint8_t _pin) : TKOutput(_pin) {}

float map(float x, float in_min, float in_max, float out_min, float out_max) {
  return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

- tinkerkit.hpp

```
#ifndef TinkerKit_hpp
#define TinkerKit_hpp

#include <stdint.h>
#include "mraa.hpp"

#define I0 0
#define I1 1
#define I2 2
#define I3 3
#define I4 4
#define I5 5

#define O0 11
#define O1 10
#define O2 9
#define O3 6
#define O4 5
#define O5 3

#define TK_MAX 1023
#define TK_X1 0     // 1x TKGyro model
#define TK_X4 1     // 4x TKGyro model
#define NORTH 1     // north pole in TKHallSensor
#define SOUTH 0     // south pole in TKHallSensor

// Arduino core libs stuff
#define LOW  0x0
#define HIGH 0x1

typedef uint8_t boolean;

#define constrain(amt,low,high) ((amt)<(low)?(low):((amt)>(high)?(high):(amt)))


/*
---------------------------------------------------------------------------
                                Generic Classes
---------------------------------------------------------------------------
*/

class TKInputDigital {

      public:
            TKInputDigital(uint8_t _pin);
            boolean read();

      protected:
            mraa::Aio* mPin;

};


class TKInputAnalog {

      public:
```

```
            TKInputAnalog(uint8_t _pin);
            int read();

      private:
            mraa::Aio* mPin;

};

class TKInputAnalog2 {

      public:
            TKInputAnalog2(uint8_t _pinX, uint8_t _pinY);
            TKInputAnalog2(uint8_t _pinX, uint8_t _pinY, uint8_t _pinZ);
            int readX();
            int readY();
            int readZ();

      protected:
            mraa::Aio *mPinX, *mPinY, *mPinZ;

};

class TKOutput {

      public:
            TKOutput (uint8_t _pin);
            void write(float value);
            inline int state() { return _state; }
            void on() {
                  write(1023);
                  _state = HIGH;
            }
            void off() {
                  write(0);
                  _state = LOW;
            }

      protected:
            int _state;
            mraa::Pwm* mPin;
};

/*
 ------------------------------------------------------------------------------
                              Digital Inputs
 ------------------------------------------------------------------------------
*/


/*      Button      */
class TKButton: public TKInputDigital {

      public:
            TKButton(uint8_t _pin);
};


/*      Tilt Sensor     */
```

```
class TKTiltSensor: public TKInputDigital {

	public:
		TKTiltSensor(uint8_t pin);

};

/*	Touch Sensor	*/
class TKTouchSensor : public TKButton {

	public:
		TKTouchSensor(uint8_t _pin);

};

/*
 ------------------------------------------------------------------------------
 				Analog Inputs
 ------------------------------------------------------------------------------
 */

/*	Potentiometer	*/
class TKPotentiometer: public TKInputAnalog {

	public:
		TKPotentiometer(uint8_t pin);
		int read();
		int readStep(int steps);

	protected:
		// mraa::Aio->read() returns values from 0 to 1023
		const static int _minVal = 0;
		const static int _maxVal = 1023;

};


/*	Light Sensor	*/
class TKLightSensor : public TKInputAnalog {

    public:
        TKLightSensor(uint8_t _pin);

};

/*	Temperature Sensor	*/
class TKThermistor : public TKInputAnalog {

	public:
		TKThermistor(uint8_t _pin);
		float readCelsius();
		float readFahrenheit();

	protected:
		const static float ADCres = 1023.0;
		const static int Beta = 3950;			// Beta parameter
		const static float Kelvin = 273.15;	// 0Â°C = 273.15 K
		const static int Rb = 10000;			// 10 kOhm
```

```
            const static float Ginf = 120.6685;     // Ginf = 1/Rinf
            // Rinf = R0*e^(-Beta/T0) = 4700*e^(-3950/298.15)

};

/*      Hall Sensor        */
class TKHallSensor : public TKInputAnalog {

      public:
            TKHallSensor(uint8_t _pin);
            boolean polarity();

      protected:
            const static uint16_t _zeroValue = 512;

};


/*      Joystick        */
class TKJoystick : public TKInputAnalog2 {

      public:
            TKJoystick(uint8_t _pinX, uint8_t _pinY);
            int readX();
            int readY();

};


/*      Gyro Sensor        */
class TKGyro : public TKInputAnalog2 {

      public:
            TKGyro(uint8_t _pinX, uint8_t _pinY, boolean model);
            void calibrate();
            long readXAxisRate();
            long readYAxisRate();

      protected:
            boolean model;

            //const static int _ADCresolution = 4880;     // [mV/count]multiplied
by 1000 to avoid float numbers
            // minimum sensitivity for the 1x module value (from datasheet is
0.167 mV/deg/s but the TinkerKit module has the outputs amplified 2x)
            //const static int _sensitivity = 334; // Sensitivity is expressed
in mV/degree/seconds, multiplied by 1000 to avoid float numbers.
            // This value represent the sensitivity of the 1x module. The
sensitivity of the 4x module is 4x of this one
            long _sensitivityInCount; // we obtain the sensitivity expressed in
ADC counts
            // [counts/dps]
            int _yZeroVoltage;
            int _xZeroVoltage;

};

/*      Accelerometer        */
```

```
class TKAccelerometer : public TKInputAnalog2 {
	public:
		TKAccelerometer(uint8_t _pinX, uint8_t _pinY);
		TKAccelerometer(uint8_t _pinX, uint8_t _pinY, uint8_t _pinZ);
		inline float readXinG() { return (float)(readX() - _zeroOffset)/96;
}
		inline float readYinG() { return (float)(readY() - _zeroOffset)/96;
}
		inline float readZinG() { return (float)(readZ() - _zeroOffset)/96;
}
		int inclination();

	protected:
		const static float _gain = 1.414;
		const static int _zeroOffset = 478;

};

/*
 ------------------------------------------------------------------------------
                                    Outputs
 ------------------------------------------------------------------------------
 */

/*      LED     */
class TKLed : public TKOutput {

	public:
	  TKLed(uint8_t _pin);
	  inline void brightness(int value) { write(value); }

};

/*      MosFet      */
class TKMosFet : public TKOutput {

	public:
		TKMosFet(uint8_t _pin);

};

/*      Relay       */
class TKRelay : public TKOutput {

	public:
		TKRelay(uint8_t _pin);

};


float map(float x, float in_min, float in_max, float out_min, float out_max);


#endif
```

- Programa principal

```
#include <iostream>
#include "mraa.hpp"
#include "TinkerKit.hpp"

using namespace std;
using namespace mraa;

int main(void) {

	// Mostrar un mensaje de inicio
	cout << " Program initialized." << endl;

	// Inicializar variables
	int pot_value;

	// Inicializar los pines de entrada y salida
	TKPotentiometer pot(I0);		// Potenciometro TinkerKit!
	TKLed led(O0);					// Led TinkerKit!

	// Apagar el led en caso de estar encendido antes de la ejecucion del
programa
	led.off();

	// Bucle del programa
	while (true) {
		pot_value = pot.read();		// Leer el valor ADC del
potenciometro (de 0 a 1023)
		cout << "Potentiometer value: " << pot_value << endl; // Mostrar el
valor leido
		if (led.state()) led.off();		// Led encendido, apagar
		else led.on();				// Led apagado,
encender
		usleep(1000*1000);		// Esperar 1 segundo (10^6 ms) para
la siguiente lectura
	}

	return 0;
}
```

ANEXO 3: CONECTIVIDAD MEDIANTE EL PROTOCOLO RS-232

Este anexo tiene como objetivo lograr la conectividad serie RS-232 entre el SIMATIC IOT2040 y otro dispositivo externo, con el fin de conseguir una transmisión de información. Como para una comunicación es necesario más de un dispositivo, se ha optado por una placa Arduino UNO para llevar a cabo las pruebas. El principal motivo que ha llevado a utilizar esta placa es básicamente el bajo coste que esta supone y, principalmente, porque estaba disponible a la hora de realizar este trabajo.

El funcionamiento de este programa consiste en utilizar las librerías *mraa* para la comunicación serie y las *TinkerKit!* adaptadas previamente para que, a la vez que se tiene el shield *TinkerKit!* conectado, haga al SIMATIC IOT2000 establecer una comunicación con la placa Arduino UNO y transmita cada cierto periodo a esta placa el valor leído por el potenciómetro *TinkerKit!* utilizado ya anteriormente, logrando así uno de los principales objetivos de este trabajo que se basa en la conectividad de dispositivos.

La placa Arduino iluminará el LED incorporado según el valor que reciba del SIMATIC IOT2040 mediante PWM.

Conexiones:

- El SIMATIC IOT2040 debe estar conectado a la fuente de alimentación de 12V DC
- El SIMATIC IOT2040 debe estar conectado mediante su puerto de Ethernet #1 con la estación de programación a través de un cable RJ-45
- Con el *TinkerKit!* Shield en el SIMATIC IOT2040, conectar un potenciómetro de *TinkerKit!* en el pin I0 del shield
- Conectar la placa Arduino a una fuente de alimentación (por ejemplo, un puerto USB de la estación de programación)
- Conectar los pines 0 y 1 (TX y RX) del shield *TinkerKit!* con los pines 1 y 0 (RX y TX) de la placa Arduino respectivamente

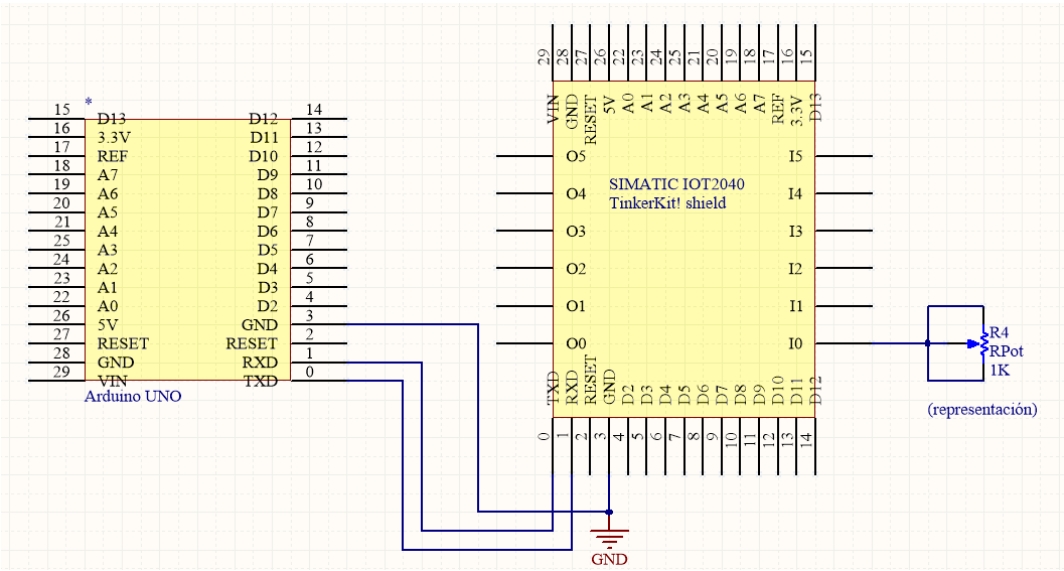


Figura 36. Anexo 3: Conexiones necesarias

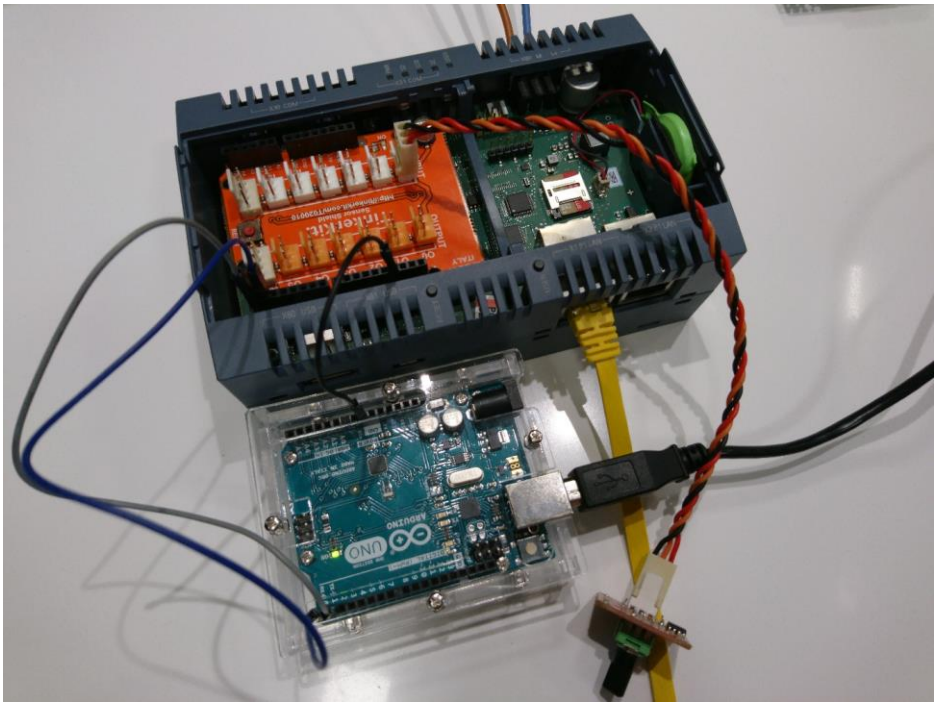


Figura 37. Anexo 3: Conexión serie con una placa Arduino UNO

Programa:

- SIMATIC IOT2040:

```
#include <iostream>
#include "mraa.hpp"
#include "TinkerKit.hpp"

using namespace std;
using namespace mraa;

int main(void) {

	// Mostrar un mensaje de inicio
	cout << " Program initialized." << endl;

	// Inicializar variables
	int pot_value;
	char buffer [4];		// La longitud del String no debería superar los
4 caracteres

	// (intentar) Inicializar la conexion UART
	Uart* serial;
	try {
		serial = new Uart(0);	// Initialise UART port 0 (pins 0 and 1 of
the Arduino extension)
	} catch (exception& e) {
		cout << e.what() << endl;
		return ERROR_UNSPECIFIED;
	}
```

```
	// Ajustar el baud rate a 9600 bit/s
	if (serial->setBaudRate(9600) != SUCCESS) {
		cout << "Error setting baud rate" << endl;
		return ERROR_UNSPECIFIED;
	}

	// Ajustar el modo de paridad
	if (serial->setMode(8, UART_PARITY_NONE, 1) != SUCCESS) {
		cout << "Error setting parity" << endl;
		return ERROR_UNSPECIFIED;
	}

	// Ajustar el flowControl
	if (serial->setFlowcontrol(false, false) != SUCCESS) {
		cout << "Error setting flow control" << endl;
	}


	// Inicializar los pines de entrada y salida
	TKPotentiometer pot(I0);		// Potenciometro TinkerKit!
	TKLed led(O0);					// Led TinkerKit!

	// Apagar el led en caso de estar encendido antes de la ejecucion del
programa
	led.off();

	// Bucle del programa
	while (true) {
		pot_value = pot.read();		// Leer el valor ADC del
potenciometro (de 0 a 1023)
		cout << "Potentiometer value: " << pot_value << endl; // Mostrar el
valor leido
		sprintf(buffer, "%d", pot_value); // Almacenar el valor en buffer
		serial->write(buffer, 4);	// Enviar el valor por la conexion
serie establecida
		if (led.state()) led.off();		// Led encendido, apagar
		else led.on();				// Led apagado,
encender
		usleep(1000*1000);		// Esperar 1 segundo (10^6 ms) para
la siguiente lectura
	}

	return 0;
}
```

- Arduino

```
const int LED = 13; //Use the Arduino built-in LED

char inputChar;


void setup() {

  pinMode(LED, OUTPUT);

  Serial.begin(9600);    // Abrir la conexión serie (IDE)
  Serial1.begin(9600);   // Abrir la conexión serie (SIMATIC IOT2040)
}

void loop() {
  if (Serial1.available()) {
    delay(25);                          // Esperar a recibir todos los bytes

    String valueString = "";
    while (Serial1.available()) {
      inputChar = Serial1.read();
      valueString += inputChar;         // Concatenar cada caracter recivido
    }

    float value = valueString.toInt();  // Crear una variable numerica (sera de 0
a 1023)
    value = value * 255 / 1024;         // Interpolarla al intervalo (0 a 255)
    int brightness = round(value);      // Redondear el valor
    analogWrite(LED, brightness);       // Encender el led con el brillo calculado

    Serial.print(valueString);          // Mostrar un mensaje con el valor
recibido en el IDE Arduino
  }
}
```

ANEXO 4: CONECTIVIDAD MEDIANTE ETHERNET: TCP/IP

Este anexo tiene como objetivo demostrar la conectividad del SIMATIC IOT2040 mediante Ethernet a través del protocolo TCP/IP. Para ello, de manera análoga al anexo anterior, la conexión se ha realizado con una placa Arduino UNO esta vez con su *Ethernet Shield* instalado. Sin embargo, esta ocasión se ha decidido ir más allá y se ha optado por crear un simple y sencillo protocolo de comunicación, cuyo concepto se basa en conjuntos de comandos y argumentos con el fin de ser transmitidos entre varios dispositivos. Su funcionamiento es simple, se envía el comando y se recibe de vuelta el mismo comando con los mismos argumentos (para identificar y verificar el mensaje enviado), con el añadido del resultado del comando.

Teniendo en cuenta que este sistema tiene como finalidad controlar determinadas entradas y salidas, los comandos considerados son los siguientes (la documentación se encuentra disponible en GitHub: *https://github.com/bamsbamx/FinalDegreeProject/blob/milestone_3/README.md*):

```
"ar <analogPinNumber>\n": Analog read command. The client will read the analog
value of the <pinNumber> assigned pin, and reply back with the read value
concatenated to the actual command (Example: "ar 1\n" makes the client read the
pin 6 analog value and reply "ar 1 347\n") NOTE: The ADC value range may depend on
the client so, in case of using an Arduino board, the actual range will be 0-1023

"aw <analogPinNumber> <dacValue>\n": Analog write command. Sets the <pinNumber>
assigned analog pin the analog <dacValue> The client will reply with the same
command if the execution was successful. (Example: "aw 2 916\n" makes the analog
pin 5 set the analog value 916) NOTE: The DAC value range may depend on the client
so, in case of using an Arduino board, the actual range will be 0-1023

"dr <pinNumber>\n": Digital read command. The client will read the digital value
of the <pinNumber> assigned pin, and reply back with the read value concatenated
to the actual command (Example: "dr 6\n" makes the client read the digital pin 6
value and reply "dr 6 0\n")

"dw <pinNumber> <value>\n": Digital write command. Sets the <pinNumber> assigned
pin the digital on/off <value>. The client will reply with the same command if the
execution was successful. (Example: "dw 5 1\n" makes the digital pin 5 set the
digital value 1)
```

COMANDO	ARGUMENTOS	EFECTO
ar **(Analogic Read)**	<*analogPinNumber*>: Número del pin de entrada del ADC	El dispositivo destino lee el valor analógico en el pin determinado y reenvía el mensaje seguido de dicho valor
aw **(Analogic Write)**	<*analogPinNumber*>: Número del pin de salida del DAC (o PWM) <*dacValue*>: Valor del DAC para establecer una tensión (en caso del Arduino de 0 a 1023)	El dispositivo destino establece la tensión determinada en el pin determinados por los argumentos del comando
dr **(Digital Read)**	<*pinNumber*>: Número del pin GPIO	El dispositivo destino lee el valor digital (HIGH o LOW) en el pin determinado y reenvía el mensaje seguido de dicho valor
dw **(Digital Write)**	<*pinNumber*>: Número del pin GPIO <*value*>: Valor a establecer	El dispositivo destino establece el valor digital determinado en el pin determinados por los argumentos del comando

Tabla 16. Anexo 4: Documentación del protocolo de comunicación

Conexiones:

- El SIMATIC IOT2040 debe estar conectado a la fuente de alimentación de 12V DC
- El SIMATIC IOT2040 debe estar conectado mediante su puerto de Ethernet #1 con la estación de programación a través de un cable RJ-45
- Conectar la placa Arduino a una fuente de alimentación (por ejemplo, un puerto USB de la estación de programación)
- Conectar el SIMATIC IOT2040 (puerto Ethernet #2) y el Arduino UNO (mediante el *Ethernet Shield*) mutuamente

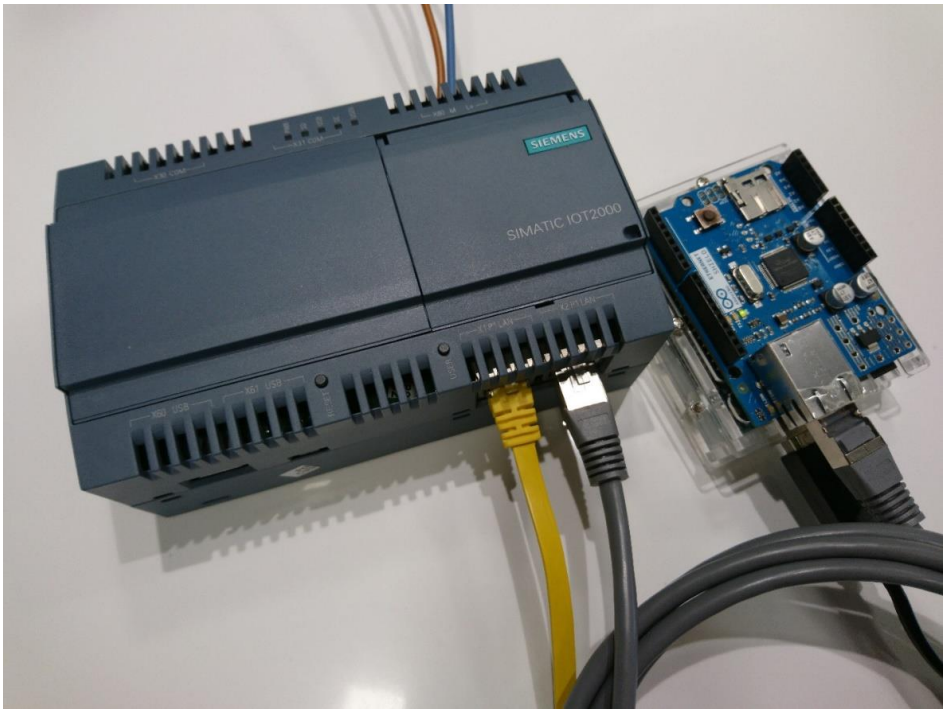


Figura 38. Conexión con una placa Arduino UNO mediante Ethernet

Programa:

- SIMATIC IOT2040:

```
#include <mraa.hpp>
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>

#include <string>
#include <sstream>
#include <vector>
#include <string.h>
#include <csignal>

const int PORT_NUMBER = 55056;
const int BUFFER_LENGTH = 256;

// format: "ar <analogPinNumber>\n"
const char* COMMAND_FORMAT_ANALOG_READ = "ar %d\n";
// format: "aw <analogPinNumber> <dacValue>\n" (depends on client dacValue)
const char* COMMAND_FORMAT_ANALOG_WRITE = "aw %d %d\n";
// format: "dr <digitalPinNumber>\n"
const char* COMMAND_FORMAT_DIGITAL_READ = "dr %d\n";
// format: "dw <digitalPinNumber> <digitalValue>\n"
const char* COMMAND_FORMAT_DIGITAL_WRITE = "dw %d %d\n";
```

```
void error(const char *);
int digitalReadCommand(char*, int);
int digitalWriteCommand(char*, int, int);
int analogReadCommand(char*, int);
int analogWriteCommand(char*, int, int);
void parseReply(char*, int);

bool stopped = false;

// Program close signal handler
void procSigIntHandler(int s) {
	stopped = true;
}

// Define input elements, which values are sent to the Arduino board
mraa::Gpio* button = NULL;
mraa::Aio* potentiometer = NULL;


int main(void) {

	// Initialize input elements
	button = new mraa::Gpio(0, true, false);
	button->dir(mraa::DIR_IN);
	potentiometer = new mraa::Aio(0);

	// Print TCP port number to console
	std::cout << "TCP port " << PORT_NUMBER << " has been opened" << std::endl;

	// Listen for program termination, so it can close sockets properly
	struct sigaction sigIntHandler;
	sigIntHandler.sa_handler = procSigIntHandler;
	sigemptyset(&sigIntHandler.sa_mask);
	sigIntHandler.sa_flags = 0;
	sigaction(SIGINT, &sigIntHandler, NULL);

	// Create I/O char buffer
	char buffer[BUFFER_LENGTH];

	// Open TCP connection socket
	int sockfd = socket(AF_INET, SOCK_STREAM, 0);
	if (sockfd < 0) error("ERROR opening socket");

	// Initialize server socket
	struct sockaddr_in serv_addr;
	memset(&serv_addr, 0, sizeof(serv_addr));
	serv_addr.sin_family = AF_INET;
	serv_addr.sin_addr.s_addr = INADDR_ANY;
	serv_addr.sin_port = htons(PORT_NUMBER);

	// Initialize client socket
	struct sockaddr_in cli_addr;
	socklen_t clilen = sizeof(cli_addr);

	// Bind server socket and start listening for clients
	if (bind(sockfd, (struct sockaddr *) &serv_addr, sizeof(serv_addr)) < 0)
error("ERROR on binding");
```

```
	listen(sockfd, 5);

	// Wait until a client is connected
	int newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr, &clilen);
	if (newsockfd < 0) error("ERROR on accept");

	int n = BUFFER_LENGTH;

	int length;
	int buttonValue;
	int potValue;
	memset(buffer, 0, BUFFER_LENGTH);
	bool error = false;

	// Keep transmitting input values until program is terminated or client is
disconnected
	while (!stopped) {

		if (error == true) {
			// Wait until any client is connected again
			newsockfd = accept(sockfd, (struct sockaddr *) &cli_addr,
&clilen);
			error = false;
		}

		// Read button value
		buttonValue = button->read();
		// Write output command into buffer
		length = digitalWriteCommand(buffer, 5, buttonValue);
		// Send digital write command to client (Arduino board)
		n = write(newsockfd, buffer, length);
		if (n < 0) {
			printf("ERROR writing to socket");
			error = true;
		}

		// Write input command into buffer
		length = analogReadCommand(buffer, 0);
		// Send analog read command to client (Arduino board)
		n = write(newsockfd, buffer, length);
		if (n < 0) {
			printf("ERROR writing to socket");
			error = true;
		}


		// Read potentiometer value
		potValue = (potentiometer->read() * 255) / 1024;
		// Write output command into buffer
		length = analogWriteCommand(buffer, 3, potValue);
		// Send analog write command
		n = write(newsockfd, buffer, length);
		if (n < 0) {
			printf("ERROR writing to socket");
			error = true;
		}

		// Read client command replies
```

```
            n = read(newsockfd, buffer, BUFFER_LENGTH - 1);

            // Parse input replies (if any)
            if (n > 0) {
                parseReply(buffer, n);
                memset(buffer, 0, BUFFER_LENGTH);
            }

            if (n < 0) {
                printf("ERROR reading from socket");
                error = true;
            }

            usleep(250 * 1000); // 2 seconds
        }

        std::cout << "Program terminated, closing sockets..." << std::endl;
        close(newsockfd);
        close(sockfd);

        return 0;
}


// Terminate program in case of errors
void error(const char *msg) {
    perror(msg);
    exit(1);
}


// Split received to read command arguments
std::vector<std::string> split(std::string str, std::string sep) {
    char* cstr = const_cast<char*>(str.c_str());
    char* current;
    std::vector<std::string> arr;
    current = strtok(cstr, sep.c_str());
    while (current != NULL) {
        arr.push_back(current);
        current = strtok(NULL, sep.c_str());
    }

    return arr;
}


// Create analog read command
int analogReadCommand(char* buffer, int pinNumber) {
        return sprintf(buffer, COMMAND_FORMAT_ANALOG_READ, pinNumber);
}


// Create analog write command
int analogWriteCommand(char* buffer, int pinNumber, int dacValue) {
        return sprintf(buffer, COMMAND_FORMAT_ANALOG_WRITE, pinNumber, dacValue);
}
```

```
// Create digital read command
int digitalReadCommand(char* buffer, int pinNumber) {
	return sprintf(buffer, COMMAND_FORMAT_DIGITAL_READ, pinNumber);
}


// Create digital write command
int digitalWriteCommand(char* buffer, int pinNumber, int digitalValue) {
	return sprintf(buffer, COMMAND_FORMAT_DIGITAL_WRITE, pinNumber,
digitalValue);
}


// Split replies from each command
void parseReply(char* replies, int length) {

	printf("Received reply: %s\n", replies);
	char commandName[10];
	int pin;
	int value;
	int n;

	char* reply;
	reply = strtok(replies, "\n"); // Separate each received reply
	while (reply != NULL) {
		n = sscanf(reply, "%s %d %d", commandName, &pin, &value);

		if (n == 0) {
			std::cout << "Unknown reply: " << reply << std::endl;
			return;
		}

		if (strcmp(commandName, "ERROR:") == 0) {
			std::cout << "Client replied with error: " << reply <<
std::endl;
		} else if (strcmp(commandName, "dr") == 0) {
			// Client replied with digital reading result
			std::cout << "Digital pin value: " << pin << "-" << value <<
std::endl;
		} else if (strcmp(commandName, "dw") == 0) {
			// Command was successful, nothing to do
		} else if (strcmp(commandName, "ar") == 0) {
			// Client replied with analog reading result
			std::cout << "Analog pin value: " << pin << "-" << value <<
std::endl;
		} else if (strcmp(commandName, "aw") == 0) {
			// Command was successful, nothing to do
		} else {
			std::cout << "ERROR: Unknown reply: " << reply << std::endl;
		}

		reply = strtok(NULL, "\n");
	}
}
```

- Arduino UNO:

```
#include <SPI.h>
#include <Ethernet.h>

// Client config (this board)
const byte MAC_ADDRESS[] = {0x90, 0xA2, 0xDA, 0x0D, 0xBF, 0x61};
const byte IP_ADDRESS[] = {192, 168, 1, 143};
const byte GATEWAY[] = {192, 168, 1, 140};
const byte SUBNET[] = {255, 255, 255, 0};

// Server config (Simatec IOT2040)
const byte SERVER_IP_ADDRESS[] = {192, 168, 1, 140};
const int SERVER_PORT = 55056;

const int MAX_LENGTH = 256;

EthernetClient client;


void setup() {

  Serial.begin(9600);

  for (int i = 0; i < 9; i++) pinMode(i, OUTPUT);

  Serial.println("Initializing client...");
  // Try DHCP connection
  if (Ethernet.begin(MAC_ADDRESS) == 0) {
    Serial.println("Failed to configure Ethernet using DHCP");
    // Try using static IP address instead of DHCP
    Ethernet.begin(MAC_ADDRESS, IP_ADDRESS, GATEWAY, SUBNET);
  }

  delay(500);
}


void loop() {

  // Wait until server becomes online
  while (!client.connect(SERVER_IP_ADDRESS, SERVER_PORT)) {
    Serial.println("Server not found, retrying...");
    delay(1000);
  }

  // At this time, server is online and client (this board) is connected to it
  Serial.println("Client connected");
  char command[MAX_LENGTH];
  int i = 0;
  char receivedByte;

  // Keep reading inputs while client is connected
  while (client.connected()) {
    // Read received bytes if available
    while (client.available()) {
      receivedByte = client.read();
      switch (receivedByte) {
```

```
        case 10: // char 10 means line end (\n), which means the end of command
          Serial.print("Command received, parsing: ");
          Serial.println(command);
          command[i] = 0; // Add string terminator at the end
          parseCommand(command);
          i = 0; // Reset counter
          break;
        default:
          if (i >= MAX_LENGTH) {
            reportError("Maximum byte count reached");
            i = 0;
          }
          // Add read byte into buffer
          command[i] = receivedByte;
          delayMicroseconds(1);
          i++;
      }
    }
  }
  client.stop();
}


void parseCommand(char* command) {

  // Extract command arguments
  char commandName[2];
  int pin;
  int value;
  int n = sscanf(command, "%s %d %d", commandName, &pin, &value);

  if (n == 0) {
    reportError("No command received"); //Unknown command
    return;
  }

  // ANALOG WRITE COMMAND
  if (strcmp(commandName, "aw") == 0) {
    if (n != 3 || value < 0 || value > 255) {
      reportError("Usage: aw <pinNumber(3,5,6,9)> <analogicValue(0-255)>");
      return;
    }
    if (pin != 3 && pin != 5 && pin != 6 && pin != 9) {
      reportError("PWM available pins are digital 3,5,6,9");
      return;
    }
    analogWrite(pin, value);
    client.println(command);
  }
  // ANALOGIC READ COMMAND
  else if (strcmp(commandName, "ar") == 0) {
    if (n != 2 || pin < 0 || pin > 5) {
      reportError("Usage: ar <pinNumber(0-5)>");
      return;
    }
    // First analog pin A0 is equivalent to 14. See:
https://github.com/arduino/Arduino/blob/master/hardware/arduino/avr/variants/stand
ard/pins_arduino.h#L56
```

```
    int value = analogRead(pin + 14);
    // Reply with read value
    client.print("ar ");
    client.print(pin, DEC);
    client.print(" ");
    client.println(value, DEC);
  }
  // DIGITAL WRITE COMMAND
  else if (strcmp(commandName, "dw") == 0) {
    if (n != 3 || value < 0 || value > 1 || pin < 0 || pin > 9) {
      reportError("Usage: dw <pinNumber> <digitalValue(0-1)>");
      return;
    }
    if (pin == 4) {
      reportError("Digital pin 4 is reserved for the Arduino ethernet shield");
      return;
    }
    digitalWrite(pin, value == 1 ? HIGH : LOW);
    client.println(command);
  }
  // DIGITAL READ COMMAND
  else if (strcmp(commandName, "dr") == 0) {
    if (n != 2 || pin < 0 || pin > 9) {
      reportError("Usage: dr <pinNumber(0-9)>");
      return;
    }
    if (pin == 4) {
      reportError("Digital pin 4 is reserved for the Arduino ethernet shield");
      return;
    }
    int state = digitalRead(pin);
    // Reply with read value
    client.print("dr ");
    client.print(pin, DEC);
    client.print(" ");
    client.println(state == HIGH ? 1 : 0, DEC);
  }
  // UNKNOWN COMMAND
  else {
    reportError("Unknown command. Available commands: dr, dw, ar, aw");
    return;
  }
}


// Reply with error to the server
void reportError(char* error) {
  Serial.println(error);
  if (client.connected()) {
    client.print("ERROR: ");
    client.println(error);
  }
}
```

ANEXO 5: APLICACIÓN DEL PROTOCOLO MODBUS

Este anexo tiene como aplicar el protocolo de comunicación estándar ModBus, que es un protocolo muy utilizado a nivel industrial, sobre todo en controladores lógicos programables (PLCs). Es por esto que resulta de gran importancia para este proyecto ya que un sistema de automatización requiere obviamente compatibilidad con este tipo de dispositivos.

Para habilitar la comunicación mediante el protocolo ModBus en el SIMATIC IOT2040, es necesario implementar las librerías oficiales proporcionadas por el organismo oficial que se encarga de su mantenimiento, las cuales se encuentran disponibles en el siguiente enlace: *http://libmodbus.org/download/*. Hay que destacar que, al ser una librería destinada para múltiples dispositivos con distintos sistemas operativos, es necesario compilar dichas librerías para el dispositivo en el que se vayan a utilizar, y luego incorporar las definiciones (header files / .h / .hpp) para el programa en el que se vayan a utilizar, para ello se deben realizar los siguientes pasos:

1. Establecer una conexión con el programa PUTTY utilizado durante la puesta en marcha
2. Descargar la librería mediante el comando *wget*
3. Extraer la librería en el SIMATIC IOT2040 mediante el comando *tar*
4. Ejecutar los comandos *./configure; make; make install* para instalar las librerías
5. Actualizar la imagen *pocky Linux* de forma que incluya las nuevas definiciones, para que el compilador las tenga en cuenta

Conexiones:

- El SIMATIC IOT2040 debe estar conectado a la fuente de alimentación de 12V DC
- El SIMATIC IOT2040 debe estar conectado mediante su puerto de Ethernet #1 con la estación de programación a través de un cable RJ-45
- Conectar la placa Arduino a una fuente de alimentación (por ejemplo, un puerto USB de la estación de programación)
- Conectar el SIMATIC IOT2040 (puerto Ethernet #2) y el Arduino UNO (mediante el *Ethernet Shield*) mutuamente

Ya que el SIMATIC IOT2040 tiene capacidad para ello, para este anexo se incluirán dos programas: El primero consistirá en la aplicación del protocolo ModBus de manera que el SIMATIC IOT2040 actúe como MAESTRO/CLIENTE, mientras que el segundo hará que actúe como ESCLAVO/SERVIDOR.

Programa de ejemplo:

- SIMATIC IOT2040 como MAESTRO/CLIENTE:

```
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace mraa;

#define LED_ADDRESS 10

int main(void) {

	// Mostrar un mensaje de inicio
	cout << " Program initialized." << endl;

	// Inicializar variables
	char buffer [3];
	uint16_t tab_reg[32];
	uint8_t dest[1];

	// Inicializar la conexion mediante TCP/IP
	modbus_t *mb;
	mb = modbus_new_tcp("192.168.1.40", MODBUS_TCP_DEFAULT_PORT);
	modbus_connect(mb);

	while (true) {
		modbus_read_bits(mb, LED_ADDRESS, 1, dest);                    //
Guardar el estado del led del esclavo en 'dest'
		if (dest) modbus_write_bit(mb, LED_ADDRESS, FALSE);        // El led
esta encendido, apagarlo
		else modbus_write_bit(mb, LED_ADDRESS, TRUE);              // El led
esta apagado, encenderlo
		modbus_read_input_registers(mb, 0, 3, tab_reg);
	// Leer el valor de los potenciometros del esclavo
		sprintf(buffer, "POT1: %d; POT2: %d; POT3: %d", tab_reg);  // Crear
un array con caracteres con los valores leidos
		cout << "Potentiometer values: " << buffer << endl;        // Enviar
un mensaje con los valores leidos
		sleep(1);
	}

	// Cerrar la conexion
	modbus_close(mb);
	modbus_free(mb);

	return 0;
}
```

- SIMATIC IOT2040 como ESCLAVO/SERVIDOR:

```
#include <iostream>
#include <stdio.h>

using namespace std;
using namespace mraa;

int main(void) {

	// Mostrar un mensaje de inicio
	cout << " Program initialized." << endl;

	modbus_mapping_t *mapping = modbus_mapping_new(1, 0, 3, 0);
	if (!mapping) {
		// No se ha podido guardar la tabla de registros en la memoria
	    spritnf(stderr, "Failed to allocate the mapping: %s\n",
modbus_strerror(errno));
	    exit(1);
	}

	//Example: set register 12 to integer value 623
	mapping->tab_registers[12] = 623;

	// Inicializar variables
	uint8_t req[MODBUS_TCP_MAX_ADU_LENGTH];
	int len;


	// Inicializar la conexion mediante TCP/IP
	modbus_t *mb;
	mb = modbus_new_tcp("192.168.1.40", MODBUS_TCP_DEFAULT_PORT);
	modbus_connect(mb);

	while(1) {
	    len = modbus_receive(ctx, req);                 // Recibir las
peticiones
	    if (len == -1) break;                           // En caso de error,
salir
	    len = modbus_reply(ctx, req, len, mapping);           // Responder a
las peticiones
	    if (len == -1) break;                           // En caso de error,
salir
	}
	cout << "Error during transmission" << endl;

	// Cerrar la conexion
	modbus_close(mb);
	modbus_free(mb);

	return 0;
}
```