UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS, ELECTRÓNICA E INDUSTRIAL
CARRERA DE INGENIERÍA INDUSTRIAL EN PROCESOS DE AUTOMATIZACIÓN

Tema:

"SISTEMA EMPOTRADO DE BAJO COSTO PARA ADQUISICIÓN DE DATOS BASADO EN EL ESTÁNDAR DE COMUNICACIÓN INDUSTRIAL OPC-UA."

Proyecto de Trabajo de Graduación Modalidad: Proyecto de Investigación, presentado previo la obtención del título de Ingeniero Industrial en Procesos de Automatización.

SUBLÍNEA DE INVESTIGACIÓN: Sistemas de control automatizados e instrumentos virtuales para procesos industriales de baja y alta potencia.

AUTOR: Héctor Fabian Baño Baño.

TUTOR: Ing. Marcelo Vladimir García Sánchez, Mg.

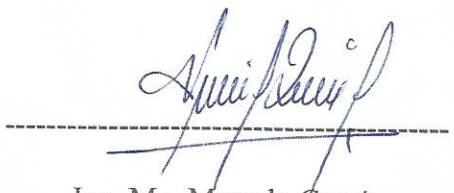
Ambato – Ecuador
Enero 2019

APROBACIÓN DEL TUTOR

En mi calidad de tutor del Trabajo de Investigación sobre el tema: "SISTEMA EMPOTRADO DE BAJO COSTO PARA ADQUISICIÓN DE DATOS BASADO EN EL ESTÁNDAR DE COMUNICACIÓN INDUSTRIAL OPC-UA", del señor Baño Baño Héctor Fabián, estudiante de la Carrera de Ingeniería Industrial en Procesos de Automatización, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, considero que el informe investigativo reúne los requisitos suficientes para que continúe con los trámites y consiguiente aprobación de conformidad con el numeral 7.2 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ambato Enero, 2019

EL TUTOR



Ing. Mg. Marcelo García

AUTORÍA

El presente Proyecto de Investigación titulado: "SISTEMA EMPOTRADO DE BAJO COSTO PARA ADQUISICIÓN DE DATOS BASADO EN EL ESTÁNDAR DE COMUNICACIÓN INDUSTRIAL OPC-UA", es absolutamente original, auténtico y personal, en tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato Enero, 2019

Héctor Fabián Baño Baño

CC: 1804570123

DERECHOS DE AUTOR

Autorizo a la Universidad Técnica de Ambato, para que haga uso de este Trabajo de Titulación como un documento disponible para la lectura, consulta y procesos de investigación.

Cedo los derechos de mi Trabajo de Titulación, con fines de difusión pública, además autorizo su reproducción dentro de las regulaciones de la Universidad.

Ambato Enero, 2019

Héctor Fabián Baño Baño

CC: 1804570123

APROBACIÓN DE LA COMISIÓN CALIFICADORA

La Comisión Calificadora del presente trabajo conformada por los señores docentes Ing. Mg. Franklin Salazar e Ing. Mg. Andrea Sánchez, revisó y aprobó el Informe Final del Proyecto de Investigación titulado "SISTEMA EMPOTRADO DE BAJO COSTO PARA ADQUISICIÓN DE DATOS BASADO EN EL ESTÁNDAR DE COMUNICACIÓN INDUSTRIAL OPC-UA", presentado por el señor Héctor Fabián Baño Baño, de acuerdo al numeral 9.1 de los Lineamientos Generales para la aplicación de Instructivos de las Modalidades de Titulación de las Facultades de la Universidad Técnica de Ambato.

Ing. Mg. Elsa Pilar Urrutia Urrutia
PRESIDENTA DEL TRIBUNAL

Ing. Franklin Salazar, Mg
DOCENTE CALIFICADOR

Ing. Andrea Sánchez, Mg
DOCENTE CALIFICADOR

DEDICATORIA:

*A mis **PADRES** quienes, con todo su cariño, amor y dedicación han logrado crear en mi un espíritu luchador sincero y humilde que es capaz de lograr grandes cosas en la vida y que ahora mismo lo puedo sentir y distinguir, gracias a que puedo culminar mi carrera de la manera más satisfactoria posible. Gracias por su esfuerzo y sacrificio totalmente invaluable, **ustedes son mi vida**.*

Héctor Fabian Baño Baño

AGRADECIMIENTO:

*A **Dios** por alimentar mi alma de fortaleza, curiosidad y amor, dándome la oportunidad de vivir experiencias irrepetibles como hermano, hijo, amigo y ahora como profesional.*

*A mis **Padres**, por ser los principales promotores de mis sueños, por confiar y creer en mí y en mis expectativas, por haber estado dispuestos a acompañarme en cada larga y agotadora noche de estudio, por siempre desear y anhelar lo mejor para mi vida, gracias por cada consejo y por cada una de sus palabras que me guiaron hasta alcanzar mis metas.*

*Al **Ing. Marcelo García,** por imponerme un reto que estuvo más allá de mis expectativas, pero que ahora cumplido, me doy cuenta que mi limite se compara solo con mi capacidad de soñar.*

*A la **Universidad Técnica de Ambato** en especial a la **FISEI**, por permitirme formar parte de tan prestigiosa institución y por ende adquirir las enseñanzas, valores y consejos de sus docentes.*

Héctor Fabian Baño Baño

ÍNDICE DE CONTENIDO

Paginas Preliminares

CAPÍTULO II
MARCO TEÓRICO

CAPÍTULO III
METODOLOGÍA

CAPÍTULO IV
DESARROLLO DE LA PROPUESTA

CAPÍTULO V
CONCLUSIONES Y RECOMENDACIONES

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El término “Industria 4.0” cada vez se está haciendo muy popular dentro de la comunidad que desarrolla tecnologías para la producción y control a nivel industrial, y con buena razón. La referencia de las diferentes etapas caracterizadas por grandes revoluciones industriales a lo largo de los años, revela la importancia y el papel fundamental que va a desempeñar la sustentabilidad de la nombrada “Fábrica del Futuro”. Con este fin, la investigación realizada trata sobre la implementación física y desarrollo aplicativo en base a la estructura planteada por *OPC-Foundation* y su estándar de arquitectura unificada (OPC-UA), integrado dentro de un sistema empotrado, basado en una tarjeta Single Board Computer Raspberry Pi, todo esto orientado a la adquisición de datos en tiempo real de diferentes sistemas de control.

Los desafíos del sistema planteado, radicaron principalmente en la selección apropiada del hardware y software de código abierto, a eso se le incluye los protocolos de comunicación entre los módulos y su complejidad de nivel medio en su programación necesaria para generar la aplicación deseada. A su vez se necesitaron varias pruebas para validar los requisitos de velocidad de comunicación y confiabilidad requeridos, con la finalidad de suplir objetivos importantes relacionados con la tecnología OPC-Clásico, comunicación multiplataforma, interoperabilidad de sistemas y mejora de la eficiencia en trasferencia y control de información.

Este sistema se implementó para la adquisición de señales tipo analógicas del módulo FESTO® MPS-PA Compact Workstation, aplicado al desarrollo de dos estructuras de lazo de control abierto y cerrado, todo esto bajo una arquitectura Servidor/Cliente. El servidor se compila dentro de la Raspberry Pi que, a su vez ejecuta un sistema operativo Debian/Linux, mientras que el cliente GUI se ejecuta dentro de un ambiente Windows, obteniendo una respuesta de control y trazado de datos en tiempo real aceptable y una operabilidad estable de todo el sistema en general.

Palabras Clave: Industria 4.0, OPC-UA, Sistemas CPS, Sistemas Empotrados.

ABSTRACT

The term "Industry 4.0" is becoming increasingly popular within the community that develops technologies for industrial production and control, and with good reason. The reference of the different stages characterized by great industrial revolutions over the years reveals the importance and the fundamental role that the sustainability of the so-called "Factory of the Future" will play. To this end, the research carried out deals with the physical implementation and application development based on the structure proposed by OPC-Foundation and its industrial communication standard (OPC-UA), integrated into a built-in system, based on a Single Board Computer Raspberry Pi, all this oriented to the acquisition of real-time data from different control systems.

The challenges of the proposed system were mainly in the appropriate selection of open source hardware and software, this includes the communication protocols between the modules and the extensive programming necessary to generate the desired application, in turn, several tests were needed to validate the required communication speed and reliability requirements, in order to meet important objectives related to OPC-Classic technology, multiplatform communication, systems interoperability and efficiency improvement in transfer and control of information.

This control system was implemented for the acquisition of analog type signals of the FESTO® MPS-PA Compact Workstation module, in addition to the development of a level PID control, all this under a Server/Client architecture, the server is compiled within the raspberry Pi, which in turn runs a Debian/Linux operating system, while the GUI client runs in a Windows environment, obtaining a control response and real-time data tracing of around 100ms, and a stable operability of all the system in general.

Keywords: Industry 4.0, OPC-UA, Embedded Systems, CPS Systems.

INTRODUCCIÓN

En la actualidad, se han presentado varias investigaciones acerca de OPC-UA y su relación fundamental dentro de la Industria 4.0, temática con la cual se trata de cumplir con las exigencias que surgen alrededor de la implementación de sistemas de interconexión en diferentes procesos de producción y manejo de datos a nivel de campo. Una parte central de estas definiciones, es su enlace transparente en todas las instancias de procesamiento de información en la que participa una cadena de valor basados en sistemas de control ciber-físicos (CPS) flexibles [1]. Los sistemas CPS son un paradigma emergente y en desarrollo exponencial dentro del diseño de sistemas industriales complejos, que integra un subconjunto de mecanismos de control y monitoreo de procesos mediante algoritmos basados en el desarrollo de aplicaciones software e incluidos estrechamente dentro dispositivos hardware de adquisición de datos adaptables y reconfigurables [2].

Esta última etapa desempeña un papel importante en dichos sistemas, debido a que su función principal es la de proporcionar una mejor dinámica en el manejo de datos y una fácil integración dentro de sistemas físicos reales. En mucho de los casos estos procesos suelen ser complejos y requieren mucho tiempo de implementación debido a su dificultosa integración heterogénea con distintos dispositivos de automatización. El actual trabajo de investigación presenta el diseño de un sistema de adquisición de datos (SAD), basado en la adaptabilidad de habilidades de control por software integrado dentro de un marco estructural presentado por la Arquitectura Unificada OPC, actuando como un sistema de comunicación, procesamiento y monitorización de datos mediante una interfaz de usuario genérica en donde los niveles de control integran fácilmente las habilidades disponibles en el sistema y sincronizan su ejecución en tiempo real de manera óptima.

El proyecto se encuentra estructurado por cinco capítulos, detallados de la siguiente manera:

Capítulo I, se detalla el planteamiento fundamental y razones de investigación del proyecto, en donde se explica el contexto actual de la problemática, cuyo desarrollo y expectativa se sustentan en satisfacer los objetivos enunciados.

Capítulo II, se puntualiza de manera breve una introducción acerca de las características actuales de los sistemas de control industrial y de las expectativas de desarrollo tecnológico que se están implementado dentro de la Industria 4.0. Además, en este capítulo se plantea y describe de forma general todo lo concerniente a tecnologías de control basadas en los sistemas CPS, así como un completo desarrollo textual del estándar de comunicación OPC-UA, temática principal y esencial para el desarrollo del objetivo principal de la investigación planteada.

Capítulo III, está constituido por la información de la metodología empleada para desarrollar la totalidad del proyecto.

Capítulo IV, se describe de manera sistemática, el proceso llevado a cabo para el desarrollo del sistema de adquisición de datos basado en el estándar OPC-UA, partiendo desde la descripción y características del hardware y software empleados como base fundamental para el desarrollo de la aplicación, todo esto dentro de un proceso sistémico orientado a la implementación del Servidor y Cliente.

Capítulo V, se cita las conclusiones y recomendaciones que se derivan del desarrollo del trabajo propuesto. Finalmente, se presentan la lista de referencias consultadas y anexos**.**

CAPÍTULO I
EL PROBLEMA

1.1 Tema

"SISTEMA EMPOTRADO DE BAJO COSTO PARA ADQUISICIÓN DE DATOS BASADO EN EL ESTÁNDAR DE COMUNICACIÓN INDUSTRIAL OPC-UA".

1.2 Planteamiento del problema

Una de las soluciones tradicionales dentro de la automatización de procesos industriales, se basan en la utilización de tres elementos principales: Buses de Campo, Controladores Lógicos Programables (PLC) y Sistemas de Interfaz Hombre-Máquina (HMI), eficientes hasta cierto punto [3]. En este escenario se destacan problemáticas diversas al trabajar con hardware de dificultosa iteración entre los elementos ya mencionados, incluyéndose una sobre-dependencia de proveedores software para el manejo de sus diferentes funciones, además de necesarios costes de implementación en temas de reingeniería [4].

Disponer de datos en tiempo real para la toma de decisiones es clave dentro de la industria. Es por ello que disponer de una tecnología interoperable que permita estandarizar múltiples protocolos, manejar diferentes fuentes de datos e implementar diferentes dispositivos empotrados que aumenten la conectividad, posibilita la creación de un nuevo modelo de control de procesos a nivel industrial, todo esto en busca de una convergencia más innovadora y rentable dentro de elementos básicos de automatización como lo son el Software y Hardware [5].

“La Cuarta Revolución Industrial” o “Industria 4.0” da respuesta a varias de las exigencias mencionadas y OPC-UA, una especificación pensada para mejorar las capacidades de OPC-Clásica, promete ser la tecnología adecuada para la impulsar la industria 4.0, gracias a la implantación de características como: a) La independencia de plataformas, eliminando la dependencia DCOM, b) Incorporando una mayor seguridad en el proceso de información en tiempo real, con OPC-binario y comunicación hacia internet con SOAP/HTTP mediante cifrado, c) Desplegando diferentes arquitecturas simplificadas sin necesidad de la llamada *Tunnelling*, que ya no es necesaria en el caso de OPC-UA puesto que está pensado para trabajar a través de *Firewall*, d) Escalabilidad a nivel hardware y software, e) Ahorro de costes de reingeniería, aprovechando los recursos ya disponibles, para crecer a medida que la industria lo requiera, mediante la simplificación de arquitecturas de control y su mantenimiento [5].

Si bien en ecuador los procesos de automatización se basan en implementaciones presentadas bajo estructuras tradicionales de control industrial [6], actualmente el gobierno ecuatoriano ha establecido cinco pilares fundamentales que coadyuvarán al cumplimiento del objetivo de alcanzar y entrar en el mundo de la Revolución Industrial 4.0, en los que se menciona: el emprendimiento e innovación, productividad, calidad, inversión y mercados, en donde la industria de alimentos, ensambladoras y cosméticos están a la vanguardia. Sobre este marco, la industria ecuatoriana debe cumplir un objetivo de negocio sustentado en la producción a través de la fabricación y comercialización de productos a medida que se adoptan nuevas tecnologías, principalmente, para automatizar la planificación de los procesos productivos, con la finalidad de ser más competitivas y garantizar su estabilidad a mediano y largo plazo en el mercado [7], [8], [9].

1.3 Delimitación

1.3.1 Delimitación de contenido

Área Académica: Ingeniería.
Línea de Investigación: Sistemas de Control.
Sublínea de investigación: Sistemas de control automatizados e instrumentos virtuales para procesos industriales de baja y alta potencia.

1.3.2 Delimitación Espacial

El presente proyecto de investigación se desarrolla en la provincia de Tungurahua dentro de las instalaciones de la Facultad de Ingeniería en Sistemas Electrónica e Industrial.

1.3.3 Delimitación Temporal

Este proyecto se realiza en el periodo académico en el que se apruebe por parte del consejo directivo de facultad.

1.4 Justificación

El principal componente de **interés** y estudio a nivel de software dentro del modelo de organización y control de la Industria 4.0 es la tecnología OPC-UA, protocolo de comunicación que permite la interoperabilidad en todos los niveles y dispositivos de una empresa. OPC-UA tiene independencia de implementación en base a su hardware o software, permitiendo una mejor conectividad entre varios productos de sistemas autómatas desarrollados en la actualidad [10], este campo de aplicación en un nivel empotrado es de suma **importancia**, ya que un gran número de sensores y dispositivos generan datos que son necesarios procesarlos y analizarlos. Con la nueva generación de sistemas de automatización y con el auge de la Industria 4.0, los usuarios podrán ser capaces de visualizar el comportamiento de procesos e instalaciones en tiempo real con el fin de comprenderlos de mejor manera y maximizar su rentabilidad.

La **contribución** de esta investigación destaca las capacidades integradas dentro de la arquitectura presentada por OPC-UA y enfocada dentro de sistemas elementales de adquisición de datos. En donde se integra una parametrización fluida dentro una aplicación Cliente puesta en marcha como controlador y de un Servidor integrado en un ordenador de placa reducida (SBC). Además, se valida la arquitectura de referencia con una implementación combinada de características software OPC-UA y *Stacks* de desarrollo de aplicaciones bajo lenguajes de programación orientada a objetos, demostrando un eficaz rendimiento en el control heterogéneo sin una excesiva sobrecarga

de recursos Hardware y Software, lo que en resumen se presenta como un enfoque de solución en especificaciones tecnológicas elementales dentro del subconjunto de mecanismos integrados en los sistemas ciber-físicos.

La **factibilidad** de estudio e implementación radica en el nivel de aplicación de diferentes conocimientos como: electrónica, sistemas de control, lenguajes de programación, redes industriales y sistemas computacionales, los mismos que son de suma importancia para su desarrollo técnico/experimental y que son complementados con la investigación que se muestra en este trabajo, que busca **beneficiar** a empresas públicas y privadas con necesidades relacionadas con sistemas industriales distribuidos, además de estar orientado a la comunidad de investigadores que se encuentran experimentando con la eficacia de ejecución del protocolo de comunicación OPC-UA de código abierto en la industria, como alternativa de aplicación a software desarrollado por empresas privadas.

1.5 Objetivos

1.5.1 Objetivo general

- Desarrollar un sistema empotrado de bajo costo para adquisición de datos basado en el estándar de comunicación industrial OPC-UA.

1.5.2 Objetivos Específicos

- Identificar las principales ventajas competitivas al ejecutar sistemas OPC-UA dentro de dispositivos empotrados.
- Implementar la arquitectura de software Servidor/Cliente OPC-UA para el control de procesos industriales.
- Generar pruebas de funcionamiento y eficiencia.

CAPÍTULO II
MARCO TEÓRICO

2.1 Antecedentes Investigativos

OPC-UA representa un creciente desarrollo exponencial de manera paralela en la creación de diferentes prototipos alternativos a los Controladores Lógico Programable (PLC's) con aplicación similares [11], atravez de dispositivos individuales conocidos como las SBC. Su implementación en diferentes campos de aplicación multidisciplinario, los están convirtiendo en una opción de hardware muy rentable y eficiente, cuyos objetivos de gestión se sustentan de manera directa con el manejo de sistemas CPS, como se detallan en los trabajos de investigación [12], [13], en los cuales se desarrolla una infraestructura que da sustentabilidad a la ejecución de servidores y clientes, basados en diferentes Pilas de aplicación OPC-UA y ejecutados en base a lenguajes de programación como Java y JavaScript.

Dentro de los sistemas de control distribuido, actualmente existe una revolución en términos de uso alrededor de conceptos del internet industrial de las cosas (IIoT). En donde investigadores proponen varios enfoques denominados "Conectar y Producir" (PnP) para su mejora y eficiencia. Por ejemplo, en [14] se propone una arquitectura de referencia novedosa para sistemas PnP dentro de la IIoT, que se basa en los estándares OPC-UA y el uso de hardware open source como base de su sistema, apoyado en el proyecto PLCopen, en donde se trata de reducir los tiempos de puesta en servicio de estos dispositivos industriales. De la misma forma, los sistemas multiagentes desempeñan un papel importante dentro de los sistemas de control distribuido en donde se busca proporcionar flexibilidad, robustez y adaptación.

La investigación presentada en [15], describe la implementación de un sistema basado en agentes dentro de un sistema de producción flexible, compuesto por un conjunto de dispositivos de automatización heterogéneos como mecanismo autómata dentro de sistemas CPS. De igual manera, aplicaciones de sistemas de comunicación con OPC-UA implementadas bajo la norma IEC-61499, detallado en [16], considera la utilización de plataformas hardware de bajo costo, como el núcleo de futuros sistemas de control y automatización de manera distribuida. Incluyendo a [17], en donde se describe una aplicación práctica con la implementación de una arquitectura CPPS cuyo objetivo principal es la de enlistar las ventajas al reducir los costos de implementación en reingeniería dentro de control de procesos, comunicación de red y adquisición de datos. En el caso de [18], se presenta un marco similar de manera experimental para la creación de una auto-organización de células de producción, en donde se desarrolla una estructura de software que organiza los servicios de los dispositivos autónomos incluidos en una célula de producción bajo el estándar UA.

Otro ejemplo de escalabilidad OPC-UA hasta el nivel de sistema en chip (SoC) integrado en una SBC, se detalla en [19], en donde se presenta una solución middleware para tales dispositivos de recursos limitados, con la implementación de un servidor OPC-UA basado en los "Perfiles de Servidor dentro de Dispositivos Nano Embebidos" presentados por la Fundación OPC. Por otro lado, proyectos open source actuales como open62541, proporcionan un marco flexible de software que implementa la pila del protocolo binario OPC-UA con la única finalidad de construir servidores y clientes independientes de plataforma [20]. Una aplicación dinámica de este software se presenta en [21], en donde se integran sugerencias en conceptos estructurales para acoplar modelos de simulación ya existentes que permiten la integración de modelos individuales en un entorno de co-simulación para escenarios proyectados en la Industria 4.0. Otro caso muy relevante de este proyecto se presenta en [22], en donde se plantea el uso de código abierto como habilitador del Modelo de arquitectura de referencia Industrie 4.0 (RAMI4.0). Esta investigación presenta la integración “Open Source Code” dentro de una aplicación de Servidor/OPC-UA personalizable en una placa Arduino Yun.

2.2 Fundamentación Teórica

2.2.1 Industria 4.0 (Cuarta Revolución Industrial)

Nuevos conceptos como la Industria 4.0 o el Internet Industrial de las Cosas (IIOT) implican una nueva forma de gestionar datos y mejorar la interconexión entre equipos. El objetivo es proveer una solución de interconexión entre equipos con diferentes arquitecturas usando las últimas tecnologías, de tal forma que la inversión en equipos se pueda ver reducida en la industria [23].

El término industria 4.0 se refiere a un nuevo modelo de organización y de control de la cadena de valor a través del ciclo de vida del producto y a lo largo de los sistemas de fabricación apoyado y hecho posible por las tecnologías de la información. Esta terminología es utiliza de manera generalizada en Europa, si bien se acuñó en Alemania, también es habitual referirse a este concepto con términos como “Fábrica Inteligente” o "Internet industrial". En definitiva, se trata de la aplicación a la industria del modelo "Internet de las cosas" (IoT). Todos estos términos tienen en común el reconocimiento de que los procesos de fabricación se encuentran en un proceso de transformación digital, una "revolución industrial" producida por el avance de las tecnologías de la información y, particularmente, de la informática y el software que ha tenido lugar a lo largo de las últimas décadas como se detalla en la Figura 1 [24].

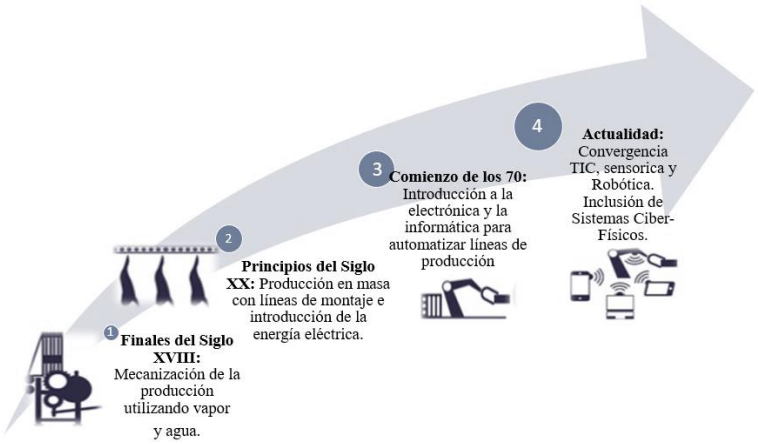


Figura 1. Evolución tecnológica a nivel Industria [2].

2.2.2 Sistemas Ciber-Físicos

Los sistemas ciber-físicos están intrínsecamente conectados con sistemas empotrados, que son partes de dispositivos completos que llevan a cabo funciones muy específicas, pero con restricciones de computación. Un sistema ciber-físico es todo aquel dispositivo que integra capacidades de computación, almacenamiento y comunicación para controlar e interactuar con un proceso físico. Los sistemas ciber-físicos están, normalmente, conectados entre sí y a su vez conectados con el mundo virtual y las redes digitales globales. Los CPS enlazan tres áreas de conocimiento esenciales, como son: Redes de Comunicación, Computación Embebida y Sistemas de control en Tiempo Real facilitando el procesamiento independiente de datos. Además, debido a la convergencia de estos sistemas, sensores y actuadores logran organizar la información recibida de forma automática y autónoma [25].

El control central del proceso puede ser eliminado ya que puede ser asumido por componentes basados en CPS. A este concepto de organización de cadena de valor también se conoce como Industria 4.0 ampliando sus gestión de control en diferentes áreas tecnológicas, su gestión de soporte se puede visualizar de forma detallada en la Figura 2 [26].

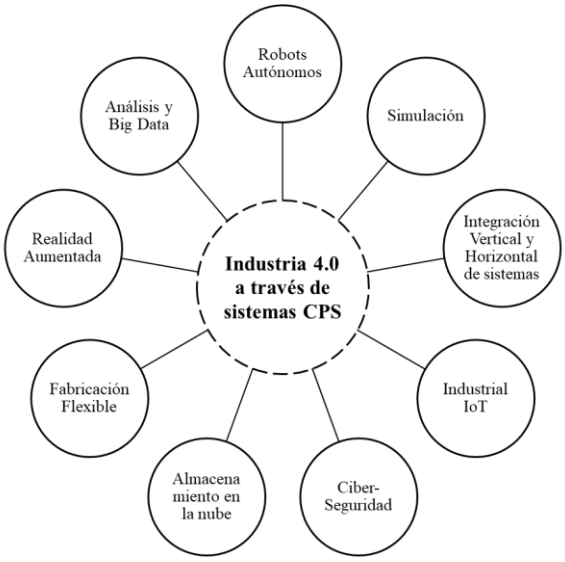


Figura 2. Nueve avances de la tecnología que soportan la Industria 4.0[26].

La llegada de los sistemas ciber-físicos a diferentes disciplinas destinadas tanto en la empresa o, así como en la sociedad, integra pautas de aplicaciones variadas que permite disuadir exigencias de desarrollo tecnológico, en la Tabla 1 se amplían sus áreas de ejecución.

Tabla 1. Aplicaciones de sistemas CPS a la Industria 4.0 [27].

PRINCIPALES ÁREAS DE APLICACIÓN DE LOS SISTEMAS CIBERFÍSICOS	
SOFTWARE	El desarrollo de software se verá claramente afectado y mejorado por los Sistemas Ciber físicos. Con la optimización de procesos y un grado mucho más alto de interconexión contaremos con programas más inteligentes, dinámicos y flexibles.
SERVICIOS	Unido al software, los servicios basados en él dispondrán de muchas más capacidades de infraestructura y generarán aplicaciones y productos de mayores prestaciones que los actuales.
NUBE	Si la nube y los servicios Cloud han cobrado mucha importancia en la última década, los Sistemas Ciber-físicos le concederán si cabe una dimensión más relevante. Los CPS permitirán ejecutar sistemas ubicuos, mayor adaptación y tenencia múltiple, algo de lo que todo tipo de dispositivos del mercado podrán beneficiarse.
BIG DATA	Parecía complicado que los datos pudieran cobrar todavía más importancia para algunas compañías o la propia sociedad, pero los Sistemas Ciber-físicos, con una mayor interconexión entre cualquier dispositivo, harán que los datos se puedan procesar o distribuir a mayor velocidad, en más cantidad, variedad y con mayor seguridad en todos los procesos.

2.2.3 Sistemas Empotrados (*Embedded Systems*)

Un sistema empotrado es un sistema informático con una función específica dentro de un sistema mecánico o eléctrico más grande, a menudo con restricciones de computación en tiempo real. Está integrado como parte de un dispositivo completo que a menudo incluye hardware y partes mecánicas. Estos sistemas integrados controlan muchos dispositivos de uso común en la actualidad. Entre el 80% y 90 % de todos los microprocesadores se fabrican como componentes de sistemas empotrados [28]. A la vez, tecnologías relacionadas con estos sistemas plantean cuestiones como: ¿Por qué incrustar un servidor OPC-UA en un dispositivo?, la respuesta a esto se reduce en una sola palabra: ***simplicidad***.

Para el usuario final, la objetividad final en la conexión de datos, es encontrar la manera más fácil, eficiente y rentable de hacerlo, de igual manera cuando y donde se los necesite, todo sin agregar hardware PC adicional, realizando configuraciones adicionales o mantenimiento. Hacer que OPC-UA se ejecute de forma nativa en los dispositivos, se interpreta como un sistema *Embedded OPC-UA* que, a su vez mejora la interoperabilidad y tecnología de conectividad OPC-UA redefiniendo las arquitecturas de automatización industrial y tratado de superar desafíos clave que se detallan en la Figura 3 [29].

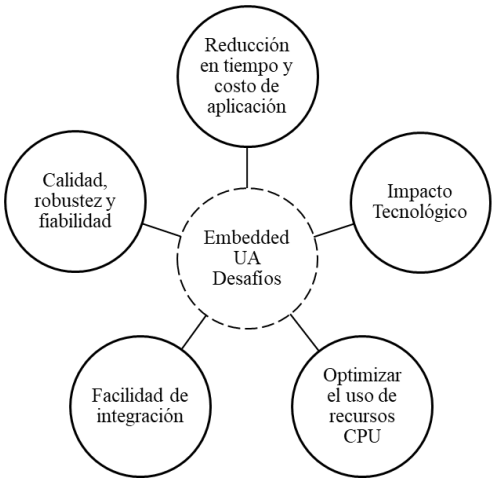


Figura 3. Desafíos clave de desarrollo de sistemas Embedded OPC-UA[29].

2.2.4 Plataforma Abierta de Comunicaciones (OPC)

La OPC Foundation administra una organización global donde los proveedores y sus usuarios colaboran para crear estándares de transferencia de datos seguros, confiables, independientes del proveedor y multiplataforma. Crean y mantienen las especificaciones, ofrecen pruebas de certificación para garantizar el cumplimiento de las especificaciones y colaborar con los líderes de la industria [30]. OPC Clásico fue lanzado en 1996 y fue el precursor de OPC-UA [31]. OPC fue presionado para evolucionar debido a la dependencia de la plataforma de Microsoft en clásicos de OPC, problemas de seguridad con el Modelo de objetos componentes (COM)/Modelo de objetos componentes distribuidos (DCOM) y problemas con la configuración del firewall. OPC-UA se lanzó en 2008 y ahora utiliza Arquitectura Orientada a Servicios (SOA) multiplataforma en lugar de COM/DCOM [32][33].

En la actualidad, con muchas soluciones OPC-UA disponibles, los usuarios se preguntan cuándo y si deberían migrar sus actuales procesos OPC clásicos a OPC-UA. Las nuevas especificaciones superan los problemas de DCOM que hacen que los productos OPC sean más seguros [34].

2.2.5 Arquitectura Unificada OPC

La Arquitectura Unificada OPC (OPC-UA) es un protocolo de comunicación independiente del proveedor para aplicaciones de automatización industrial. Se basa en el principio cliente-servidor y permite una comunicación continua desde los sensores y actuadores individuales hasta los Sistemas de Planificación de Recursos Empresariales (ERP) [35]. La capacidad de integrar servidores OPC-UA en microcontroladores ofrece un amplio esquema de posibilidades y elimina el equipo adicional para traducir protocolos patentados con el OPC clásico, ver Figura 4.

OPC-UA ofrece una solución completa para todos los requisitos en todas las capas verticales para el acceso de dispositivos remotos, la ampliación de sus aplicaciones se detalla en la Tabla 2. La cuarta revolución industrial está impulsada por tecnologías avanzadas de información y comunicación (TIC), que cada vez son más frecuentes en la automatización industrial. En los sistemas inteligentes distribuidos, los sistemas físicos, reales y los datos digitales/virtuales se fusionan dentro de los sistemas ciber-físicos. En donde los sistemas están interconectados y son autónomos, se reconfiguran y optimizan, además son expandibles sin ingeniería intervención o instalación manual [36].

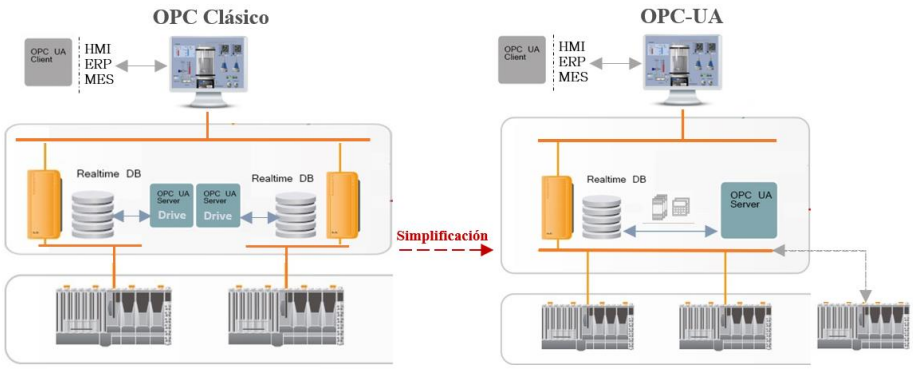


Figura 4. Cambio de OPC Clásico a OPC-UA[4].

Tabla 2. Soluciones OPC-UA para la Industria 4.0 [36].

Requerimientos	Soluciones OPC UA
Independencia de la tecnología de comunicación del fabricante, sector, sistema operativo o lenguaje de programación.	OPC-UA se ejecuta en todos los sistemas operativos, incluso hay implementaciones de capa de chip sin un sistema operativo. OPC-UA se puede implementar en todos los idiomas: en la actualidad hay disponibles apiladas en ANSI C / C ++, .NET y Java.
Transferencia segura y autenticación a nivel de usuario y aplicación.	OPC-UA utiliza certificados X.509, Kerberos o usuario/contraseña para la autenticación de la aplicación. La transferencia firmada y encriptada, así como un concepto de derechos a nivel de punto de datos con funcionalidad de auditoría, está disponible en la pila.
SOA, transporte a través de estándares establecidos como TCP/IP para intercambiar datos, comandos y eventos en vivo e históricos.	OPC-UA es independiente del método de transporte. Actualmente, hay dos enlaces de protocolo disponibles: protocolo binario optimizado basado en TCP para aplicaciones de alto rendimiento y servicio web HTTP/HTTPS con mensajes codificados binarios o XML. Además, se puede integrar el modelo de comunicación *Publish/Subscribe*.
Integración en ingeniería y extensión semántica.	La Fundación OPC ya colabora exitosamente con otras organizaciones (PLCopen, BACnet, FDI, AIM, etc.) y actualmente está expandiendo sus actividades de cooperación, por ej. MES-DACH, ISA95, MDIS (industria del petróleo y el gas), etc.
Verificación de conformidad con el estándar definido.	OPC-UA ya es un estándar IEC (IEC 62541) y existen herramientas y laboratorios de prueba para probar y certificar la conformidad. Los eventos de prueba adicionales (por ejemplo, *Plugfest*) mejoran la calidad y garantizan la compatibilidad.

Además, dentro de la arquitectura presentada por OPC-UA se destacan parámetros estandarizados como:

- **Seguridad OPC-UA:** Las arquitecturas OPC Unified, proporciona una nueva especificación de interfaz de software y un marco de aplicación basado en el servicio web para sistemas de automatización de planta que se comunican entre sí a través de Internet. El problema de seguridad es la clave de su desarrollo para la característica de tecnología basada en el servicio web. En este documento, los problemas de seguridad de OPC-UA se discuten desde las dos vistas de la seguridad del entorno de red y seguridad de comunicación en aplicaciones OPC-UA [37].

Se propuso la solución de implementación de seguridad de red basada en firewall distribuido para garantizar que el host del servidor y cliente OPC-UA contra los diferentes ataques. Secuencialmente, se presentó el modelo mejorado de seguridad OPC-UA basado en el modelo existente. Se agregó un módulo de gestión de estrategia de seguridad en el modelo, al configurar el módulo, la característica de seguridad del sistema de aplicaciones OPC-UA se puede adaptar a los diferentes niveles de seguridad. Finalmente, se diseña el modelo de información del modelo de seguridad. Por los medios anteriores, la eficiencia de comunicación y seguridad de comunicación de OPC-UA se puede equilibrar mejor y proporciona la guía para el desarrollo del servidor OPC-UA y las aplicaciones OPC-UA, su estructura se detalla en la Figura 5 [37].

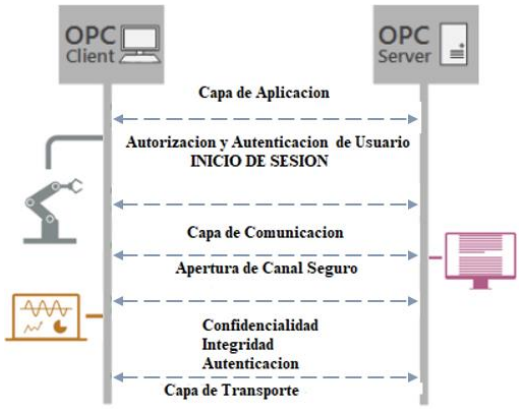


Figura 5. Modelo de Seguridad OPC-UA[37].

- **Comunicación Normalizada a Través de Internet y Firewalls:** OPC-UA amplía el estándar anterior de la industria OPC mediante varias funciones importantes, como la independencia de la plataforma, la escalabilidad, la alta disponibilidad y la capacidad de Internet. OPC-UA ya no se basa en la tecnología DCOM de Microsoft; se ha reinventado sobre la base de arquitectura orientada a servicios (SOA). OPC-UA es, por lo tanto, muy simple de adaptar. Hoy OPC-UA ya conecta el nivel empresarial hasta los sistemas integrados de los componentes de automatización, independientemente de Microsoft, UNIX o cualquier otro sistema operativo [37].

OPC-UA utiliza un protocolo binario optimizado basado en TCP para el intercambio de datos a través de un puerto 4840 registrado con IANA. El servicio web y HTTP también son compatibles opcionalmente. Los enlaces de protocolo adicionales como *Multicast o Message-Queuing* se pueden integrar fácilmente sin romper los conceptos de comunicación existentes. Los mecanismos de cifrado integrados garantizan una comunicación segura a través de Internet [37].

- **Protección Contra el Acceso no Autorizado:** La tecnología OPC-UA utiliza conceptos de seguridad comprobados que ofrecen protección contra el acceso no autorizado, contra el sabotaje, la modificación de los datos del proceso y contra el funcionamiento descuidado. Los conceptos de seguridad de OPC-UA contienen autenticación de usuarios y aplicaciones, la firma de mensajes y el cifrado de los datos transmitidos y un estándar de creación de objetos OPC-UA. La seguridad de OPC-UA se basa en estándares reconocidos que también se utilizan para la comunicación segura en Internet, como SSL o TLS. Los mecanismos de seguridad son parte del estándar y son obligatorios para los proveedores. El usuario puede combinar las diversas funciones de seguridad según su caso de uso; por lo tanto, los resultados de seguridad escalables en relación con la aplicación específica [37].

- **Confidencialidad, integridad y autenticación de la aplicación:** La autenticación de la aplicación necesita un diagrama de secuencia más detallado para mostrar una mejor visión general de la comunicación de arquitectura de seguridad. Muestra cómo el cliente realiza una solicitud de punto final al servidor y obtiene respuesta del servidor sobre sus configuraciones de seguridad y políticas. También envía su certificado de servidor al cliente. El cliente luego valida el certificado del servidor poniéndose en contacto con la Autoridad de certificación (CA). El cliente le pide al servidor un seguro canaliza y envía su certificado y clave privada que se cifra con la clave pública del servidor. El servidor valida el certificado con la CA. El canal seguro se abre entre el cliente y el servidor [37].

- **Espacio de direcciones:** Es un concepto nuevo e importante dentro de OPC-UA. Todos los bloques o dispositivos conectados se deben almacenar en un solo modelo jerárquico, dicho modelo de almacenamiento de datos puede ser establecido por el propietario del servidor de acuerdo con sus necesidades. En este sentido, cada objeto conectado es tratado como un nodo dentro del espacio de dirección. Normalmente, el servidor OPC-UA ofrecerá datos proporcionados por un sistema subyacente, como un dispositivo, una base de datos de configuración, un servidor OPC COM, etc. Por lo tanto, el modelado de los datos depende del modelo del sistema subyacente y de los requisitos de los clientes que acceden el servidor OPC-UA. Cada servidor puede modelar sus datos específicos de un proceso de manera que se ajuste a sus necesidades [38].

  El objetivo principal del espacio de direcciones OPC-UA es proporcionar una forma estándar para que los servidores representen objetos para los clientes, definiendo objetos en términos de Variables y Métodos. Además, los objetos pueden escribirse a máquina, es decir, OPC-UA proporciona una forma de definir y exponer tipos de objetos (clases con variables miembro y métodos miembros) e instancias de objetos. No es necesario utilizar este enfoque orientado a objetos. Se puede construir un modelo de espacio de direcciones simple como en el clásico OPC Data Access utilizando objetos y variables de carpeta. Pero la disponibilidad de características mejoradas orientadas a objetos hace que la representación de sistemas orientados a objetos sea mucho más fácil con OPC-UA [38].

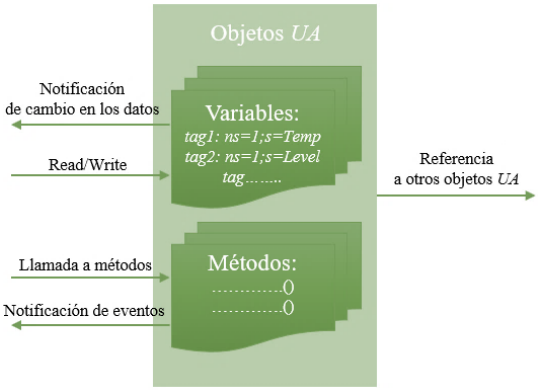


Figura 6. Modelo básico de objeto OPC-UA[38].

Los servicios UA se usan para acceder a los objetos y sus componentes, como leer o escribir un valor variable, llamar a un método o recibir eventos del objeto como se detalla en la Figura 6. El servicio de exploración se puede utilizar para explorar las relaciones entre los objetos y sus componentes, los elementos de este modelo están representados en el espacio de direcciones como nodos. Cada nodo se asigna a una clase de nodo, por ejemplo, objeto, variable y método, y representa un elemento diferente del modelo de objetos [38].

- **Modelo de nodo:** El conjunto de objetos y la información relacionada que el servidor OPC-UA pone a disposición de los clientes es su espacio de direcciones, los objetos y sus componentes se representan en el espacio de direcciones como un conjunto de nodos descritos por atributos e interconectados por referencias, como se muestra en la Figura 7 [38].

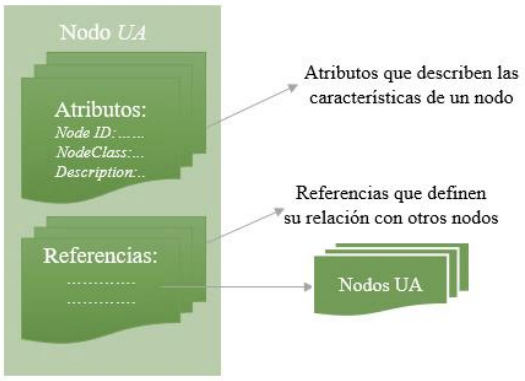


Figura 7. Modelo básico de Nodo OPC-UA[38].

Los atributos son elementos de datos que describen nodos. Los clientes pueden acceder a los valores de los atributos mediante los servicios de Lectura, Escritura, Consulta y Suscripción/Artículo Monitoreado. Los atributos son componentes elementales de las clases de nodo. Las definiciones de atributos forman parte de las definiciones de clase de nodo y, por lo tanto, son conocidas por los clientes y no son directamente visibles en el espacio de direcciones. Cada definición de atributo consiste en un identificador de atributo, un nombre, una descripción, un tipo de datos y un indicador obligatorio/opcional. Los clientes o servidores no

pueden extender el conjunto de atributos definidos para cada clase de nodo. Cuando se crea una instancia de un nodo en el espacio de direcciones, se deben proporcionar los valores de los Atributos de clase de nodo obligatorios. Po otra parte, las referencias se utilizan para relacionar nodos entre sí. Se puede acceder a ellos utilizando los servicios de navegación y consulta. Al igual que los atributos, se definen como componentes fundamentales de los nodos. A diferencia de los atributos, las referencias se definen como instancias de nodos *ReferenceType* [38].

Además, se especifica los atributos disponibles en todas las clases de nodo como se muestra en la Tabla 3. Los siguientes atributos están disponibles para cada clase de nodo.

Tabla 3. Atributos de la clase Nodo OPC-UA [38].

Atributo	Uso	Tipo de Dato	Descripción
NodeId	Obligatorio	NodeId	Identifica de manera única un Nodo en un servidor OPC UA y se usa para direccionar el Nodo en los Servicios OPC-UA
NodeClass	Obligatorio	NodeClass	Una enumeración que identifica el NodeClass de un nodo como objeto, variable o método
BrowseName	Obligatorio	QualifiedName	Identifica el nodo cuando navega por el servidor OPC-UA. No está localizado
Nombre para mostrar	Obligatorio	LocalizedText	Contiene el nombre del nodo que se debe usar para mostrar el nombre en una interfaz de usuario. Por lo tanto, está localizado
Descripción	Opcional	LocalizedText	Este atributo opcional contiene una descripción textual localizada del nodo
WriteMask	Opcional	UInt32	Es opcional y especifica qué Atributos del Nodo son modificables, es decir, pueden ser modificados por un cliente OPC-UA
UserWriteMask	Opcional	UInt32	Es opcional y especifica qué Atributos del Nodo puede modificar el usuario actualmente conectado al servidor

- **Suscripción OPC UA:** A diferencia de la lectura permanente de información, OPC-UA ofrece una funcionalidad más elegante, una llamada Suscripción. Un cliente de UA puede suscribirse a una selección de nodos de interés y dejar que el servidor controle estos elementos. Solo en caso de cambios, por ejemplo, a sus valores, el servidor notifica al cliente sobre dichos cambios. Este mecanismo reduce inmensamente la cantidad de datos transferidos. Además de la reducción del ancho de banda, este mecanismo presenta más ventajas y es el mecanismo recomendado para "leer" información de un servidor UA. Un cliente puede suscribirse a diferentes tipos de información proporcionada por un servidor OPC-UA. La Figura 8 muestra los servicios que están involucrados cuando un cliente se suscribe a cambios de datos y eventos [38].

  El propósito de una Suscripción es agrupar estas fuentes de información, llamadas Elementos Controlados, formando una información llamada Notificación. Una Suscripción consiste en al menos un Artículo Monitoreado, debe ser creado dentro del contexto de una Sesión y puede ser transferido a otra Sesión. Para crear una sesión, se debe establecer un canal seguro entre el cliente y el servidor [38].

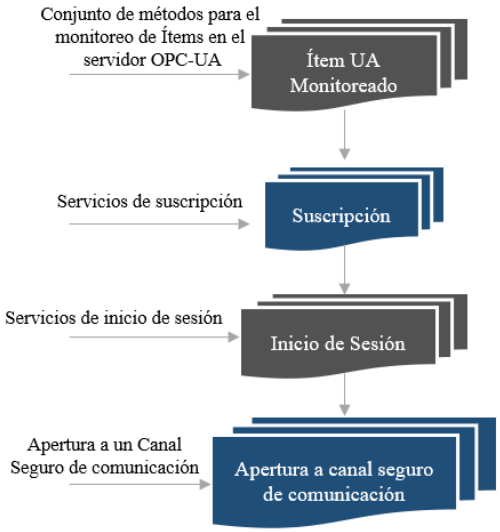


Figura 8. Contexto necesario para suscribirse a cambios de datos y eventos[38].

2.2.6 Herramientas Software

- **Interfaz de programación de aplicaciones** (API): Es un conjunto de rutinas, protocolos y herramientas para la construcción de aplicaciones software. Básicamente, una API especifica cómo deben interactuar los componentes de software en función de la aplicación deseada a desarrollarse [39].

  OPC-UA define un modelo de programación abstracta que puede implementarse en diferentes lenguajes de programación. Esto permite, tanto a empresas privadas, así como a la comunidad *Maker*, desarrollar diferentes paquetes API para aprovechar el modelo estándar propuesto por *OPC-Foundation* y agregar funciones que hacen que la API-OPCUA sea más útil y sencilla de implementar para los diferentes usuarios. En la actualidad, existen algunas implementaciones de código abierto OPC-UA desarrolladas en diferentes lenguajes y con diferentes licencias de uso. GitHub pretende colaborar con la comunidad *Maker* intentando intercambiar experiencias e ideas *Open Source* detallando una lista actualizada hasta la fecha con implementaciones OPC-UA de aplicación general, como se muestra de manera resumida en la Tabla 4 [40].

Tabla 4. Implementaciones OPC-UA [40].

Nombre	**Lenguaje**	**Licencia**	**Aplicación**	**Última Actualización**
open62541	C	MPL-2.0	Cliente/Servidor	Diciembre 2018
Estándar UA.net	C#	GPL, RPC	Cliente/Servidor	Julio 2018
Node-opcua	JavaScript	MIT	Cliente/Servidor	Noviembre 2018
FreeOpcUa	C++	LGPL	Cliente/Servidor	Junio 2018
Python FreeOpcUa	Python	LGPL	Cliente/Servidor	Agosto 2018
Eclipse Milo	Java	Eclipse Public License	Cliente/Servidor	Octubre 2018
OPC-UA Client	C#	MIT	Cliente.	Noviembre 2018

- **Cmake:** Es una familia de herramientas de plataforma abierta y multiplataforma diseñada para desarrollar, probar y empaquetar software. CMake se utiliza para controlar el proceso de compilación del software utilizando plataformas simples y archivos de configuración independientes del compilador, y genera makefiles y espacios de trabajo nativos que se pueden usar en el entorno de compilación de su elección [41].

- **Interfaces de Comunicación Serial:** La comunicación en serie es una técnica de comunicación utilizada en telecomunicaciones en la que la transferencia de datos se produce transmitiendo datos bit por bit en orden secuencial a través de un bus informático o un canal de comunicación. La comunicación en serie es el enfoque más utilizado para transferir información entre el equipo de procesamiento de datos y los periféricos. El protocolo serial es la forma segura y confiable de comunicación que tiene un conjunto de reglas dirigidas por el host de origen (emisor) y el host de destino (receptor). Algunas de las interfaces más conocidas y utilizadas para el intercambio de datos entre microcontroladores y microprocesadores son: I2C, SPI, las cuales se detallan a continuación [42]:

  a) **Comunicación a través de I2C:** El protocolo del circuito integrado (I2C) es un protocolo diseñado para permitir que múltiples circuitos integrados digitales "esclavos", se comuniquen con uno o más chips "maestros" [43].

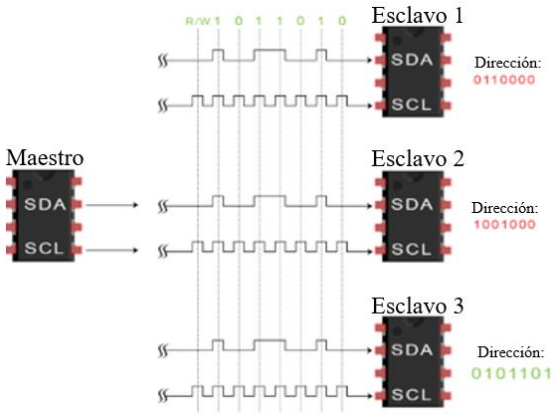


Figura 9. Configuración Multi Maestro-Esclavo mediante I2C[43].

I2C requiere una conexión física dual para SDA Y SCL, como serie asincrónica, pero esos dos cables pueden admitir hasta 1008 dispositivos esclavos. Además, a diferencia de SPI, I2C puede admitir un sistema multimaestro, permitiendo que más de un maestro se comunique con todos los dispositivos en el bus (aunque los dispositivos maestros no pueden comunicarse entre sí por el bus y deben turnarse para usar las líneas de bus) un detalle de su estructura se muestra en la Figura 9 [43].

Los mensajes se dividen en marcos de datos. Cada mensaje tiene un marco de dirección que contiene la dirección binaria del esclavo y uno o más marcos de datos que contienen los datos que se transmiten. El mensaje también incluye condiciones de inicio y parada, bits de lectura/escritura y bits de ACK/NACK entre cada marco de datos, como se detalla en la estructura de la Figura 10 [43].

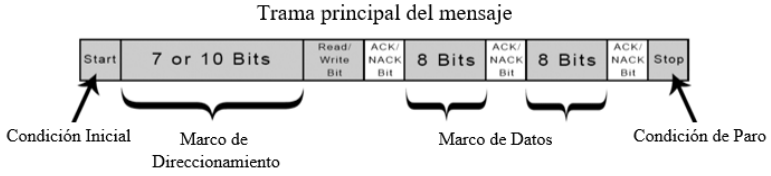


Figura 10. Trama principal de envió de mensajes por I2C [44].

Los mensajes se dividen en dos tipos de marcos: un marco de direcciones, donde el maestro indica el esclavo al que se envía el mensaje, y uno o más marcos de datos, que son mensajes de datos de 8 bits que pasan de maestro a esclavo o viceversa. Los datos se colocan en la línea SDA después de que SCL baja, y se muestrean después de que la línea SCL sube. El tiempo entre el borde del reloj y la lectura/escritura de datos está definido por los dispositivos en el bus y variará de un chip a otro [44].

b) **Comunicación a través de SPI:** SPI es un protocolo de comunicación común utilizado por muchos dispositivos diferentes. Un beneficio único de SPI es el hecho de que los datos se pueden transferir sin interrupción. Se puede enviar o recibir cualquier cantidad de bits en un flujo continuo. Con I2C o UART, los datos se envían en paquetes, limitados a un número específico de bits [45].

Los dispositivos que se comunican a través de SPI están en una relación maestro-esclavo, el maestro es el dispositivo de control (generalmente un microcontrolador), mientras que el esclavo (generalmente un sensor, pantalla o chip de memoria) recibe instrucciones del maestro. La configuración más simple de SPI es un solo sistema maestro, esclavo único, pero un maestro puede controlar más de un esclavo, su detalle de configuración se puede verificar en la Figura 11 [45].

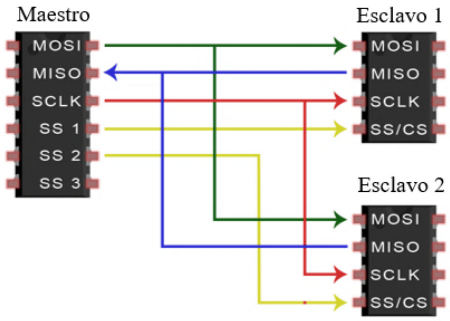


Figura 11. Multimaestro SPI [45].

La señal del reloj sincroniza la salida de los bits de datos desde el maestro hasta el muestreo de los bits por el esclavo. Se transfiere un bit de datos en cada ciclo de reloj, por lo que la velocidad de la transferencia de datos está determinada por la frecuencia de la señal del reloj. La comunicación SPI siempre es iniciada por el maestro ya que el maestro configura y genera la señal del reloj. Además, SPI se puede configurar para operar con un solo maestro y un solo esclavo, y se puede configurar con múltiples esclavos controlados por un solo maestro, en donde, el maestro envía datos al esclavo bit por bit, en serie a través de la línea MOSI [45].

El esclavo recibe los datos enviados desde el maestro en el pin MOSI. Los datos enviados desde el maestro al esclavo generalmente se envían con el bit más significativo primero. El esclavo también puede enviar datos al maestro a través de la línea MISO en serie. Los datos enviados desde el esclavo de vuelta al maestro generalmente se envían con el bit menos significativo primero [45].

2.2.7 Herramientas Hardware

- **Conversor Analógico-Digital:** Un ADC convierte una señal analógica de amplitud continua y continua en una señal digital de discreto tiempo y amplitud discreta. La conversión implica la cuantificación de la entrada, por lo que necesariamente introduce una pequeña cantidad de error o ruido. Además, en lugar de realizar continuamente la conversión, un ADC realiza la conversión periódicamente, muestreando la entrada, limitando el ancho de banda permitido de la señal de entrada. Un ADC también puede proporcionar una medida aislada, como un dispositivo electrónico que convierte una tensión o corriente analógica de entrada en un número digital que representa la magnitud de la tensión o corriente. Normalmente, la salida digital es un complemento de dos números binarios que es proporcional a la entrada, pero hay otras posibilidades. Hay varias arquitecturas ADC. Debido a la complejidad y la necesidad de componentes con una coincidencia precisa, todos los ADC más especializados se implementan como circuitos integrados (IC) como se muestra en la Figura 12 [46].

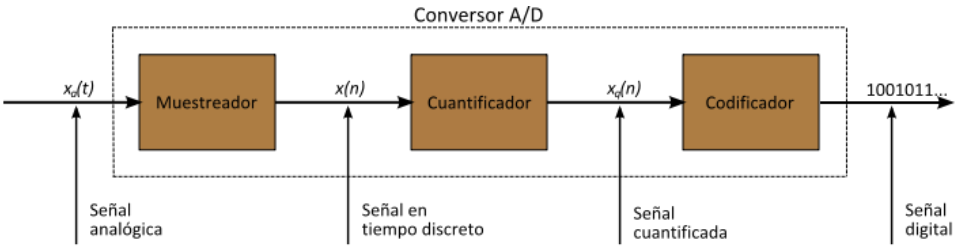


Figura 12. Estructura resumida de un IC de un conversor ADC [46].

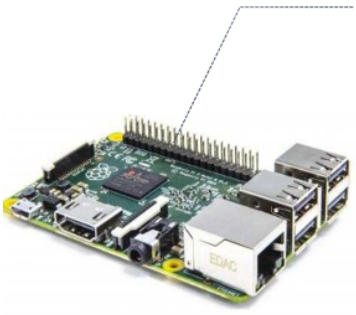
- **Single Board Computer (SBC):** Son dispositivos que funcionan como plataformas de sistemas de computación completas dentro de un solo paquete. En general son pequeñas y compactas, alrededor del tamaño de una tarjeta de crédito, pero cuentan con todo el hardware y software para funcionar como una computadora completamente integrada. El precio de las computadoras de placa única varía dependiendo de sus características, pero son conocidas por su economía y su versatilidad [47].

Los ordenadores de placa reducida están equipados con un procesador de aplicaciones capaz de ejecutar un sistema de operación avanzado como Linux, Windows o Android, y generalmente están listos para funcionar de inmediato, con una configuración mínima. Las SBC cuentan con conectividad cableada o inalámbrica y en general incluyen amplias interfaces de expansión. Generalmente tienen el respaldo de un ecosistema amplio y activo de usuarios, complementos de expansión y software de código abierto, aunque, a diferencia de una computadora personal, no se basa en expansiones para otras funciones que no sean de propósito específico. Los SBC se pueden producir fácilmente y tienen un tiempo de comercialización rápido en comparación con las computadoras personales o las computadoras portátiles. Son más livianos, de tamaño compacto, más confiables y mucho más eficientes que las computadoras con varios tableros, sin embargo, los computadores de placa reducida también tienen sus limitaciones [47].

Es posible que su formato estándar no sea adecuado o no se considere adecuado para las necesidades particulares de un cliente. También pueden ser difíciles de usar para aplicaciones que requieren la eliminación del cable o el uso de conectores de entrada/ salidas especiales. Las SBC se usan principalmente en aplicaciones integradas, también se usan en aplicaciones para control de procesos, como sistemas robóticos complejos y aplicaciones intensivas de procesador. A menudo se consideran una excelente alternativa a los microcontroladores [48].

Varias opciones de SBC de panel posterior permiten que estas computadoras se expandan tremendamente en sus capacidades, por lo que no están excesivamente limitadas por sus diseños. De hecho, con ciertos tipos de arreglos de placa base, las computadoras de placa reducida en realidad pueden proporcionar una capacidad más expandida que una computadora con varias placas estándar y en una cantidad mucho más compacta de espacio, a continuación, se enlistan algunas de las placas SBC más conocidas:

a) **Raspberry Pi:** Es una computadora del tamaño de una tarjeta de crédito diseñada originalmente para la educación, inspirada en la BBC Micro de 1981. El objetivo del creador Eben Upton era crear un dispositivo de bajo costo que mejorara las habilidades de programación y la comprensión del hardware en el nivel preuniversitario. Pero gracias a su pequeño tamaño y precio accesible, fue adoptado rápidamente por fabricantes de herramientas, fabricantes y entusiastas de la electrónica para proyectos que requieren más que un microcontrolador básico. Raspberry Pi es hardware abierto, con la excepción del chip principal, el *BROADCOM* SoC (*System on a Chip*), que ejecuta muchos de los componentes principales de la placa: CPU, gráficos, memoria, controlador USB, etc. Muchos de los proyectos hechos con Raspberry Pi también son de código abierto y cuentan con una excelente documentación, además de incorporar un gran apoyo en la comunidad *Maker* [49][50]. La Raspberry Pi 3 Model B forma parte de la tercera generación de estos modelos y mantiene el mismo formato de placa popular que el Raspberry Pi 2, pero cuenta con un SoC de 64 Bits, con una velocidad de 1,2 GHz, además de Wi-Fi y Bluetooth incorporado, una ampliación de sus características se detalla en la Figura 13 [51].



Procesador:
El conjunto de chips Broadcom BCM2387.
1.2 GHz QUAD-Core ARM Cortex-A53 (64 bits)
LAN inalámbrica 802.11 y Bluetooth 4.1 (Bluetooth Classic y LE)

GPU:
Co-procesador Multimedia Dual Core VideoCore IV®. Proporciona Open GL ES 2.0, OpenVG acelerado por hardware y decodificación de alto perfil 1080p30 H.264. Capaz de 1Gpixel/s, 1.5Gtexel/so 24GFLOPs con filtrado de texturas e infraestructura DMA

Ethernet:
Toma Ethernet 10/100 BaseT

Conector USB: x4 USB 2.0

Conector GPIO:
Cabezal de expansión de 40 pines de 2,54 mm (100 mil): cinta de 2x20
Proporciona 27 pines GPIO, así como líneas de suministro +3.3 V, +5 V y GND

Precio: Alrededor de los 60$

Figura 13. Raspberry Pi 3 Model B [52].

b) **Beaglebone Black:** La Beaglebone Black es un computador de placa reducida, como la Raspberry Pi. Una SBC ofrece todo el hardware que esperaría encontrar en una computadora de escritorio o portátil, reducido y soldado a una sola placa de circuito. Un procesador, memoria y aceleración de gráficos están presentes como chips en el tablero. BeagleBone Black es una plataforma de desarrollo de bajo costo, de código abierto y compatible con la comunidad para desarrolladores y aficionados a los procesadores ARM® Cortex™-A8. En la BeagleBone Black su proceso de inicio del sistema se lo puede realizar instalando Debian GNU/Linux™ en una FLASH incorporada. Muchas otras distribuciones de Linux y sistemas operativos también son compatibles con BeagleBone Black, incluyendo: Ubuntu, Androide, Fedora [53]. Las capacidades de BeagleBone Black se pueden ampliar utilizando placas de expansión llamadas "Shields" que se pueden conectar en los dos cabezales de expansión de doble fila de 46 pines de BeagleBone Black en donde se pueden manipular por software sus E/S tanto analógicas analógicos como digitales, lo que significa que puede admitir tarjetas complementarias adicionales [54], la ampliación de sus características se muestran en la Figura 14.

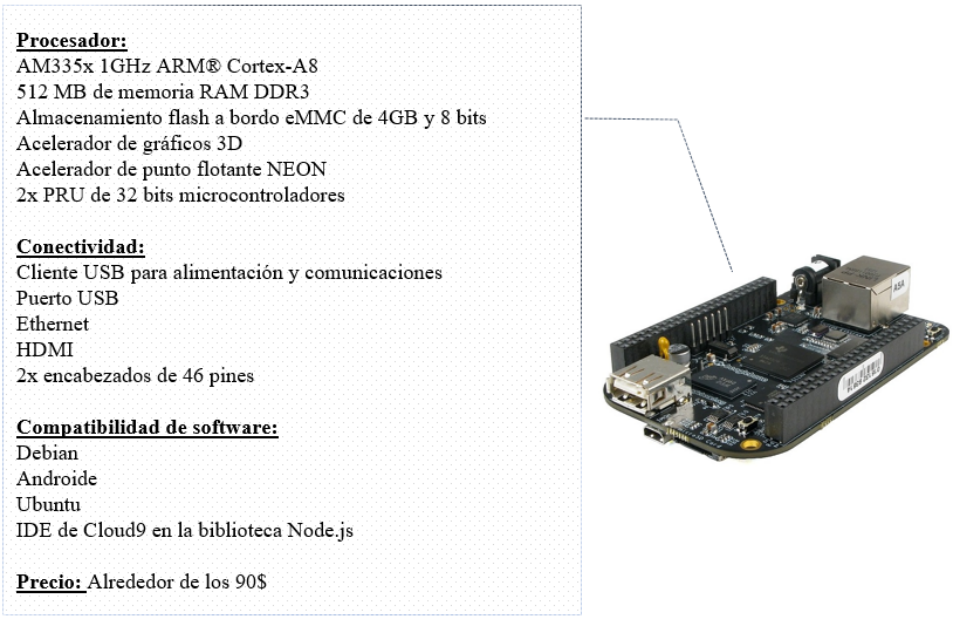


Figura 14. BeagleBone Black [55].

c) **PCduino 4 NANO:** Es un tablero ARM basado en el microcontrolador *Allwinner H3* diseñado y lanzado por *LinkSprite* para aficionados, fabricantes y fanáticos electrónicos. Tiene solo dos tercios del tamaño de la Raspberry Pi incluyendo su capacidad de integrar código abierto. El PCduino es una plataforma de mini PC rentable y de alto rendimiento que ejecuta sistemas operativos con todas las funciones, como Ubuntu y Android ICS. Es de fácil conexión, se alimenta con un voltaje de 5V para un correcto uso de sus periféricos conectados como un teclado y mouse. El PCduino emite video a cualquier televisor o monitor habilitado para HDMI a través de la interfaz HDMI incorporada. Fue diseñado específicamente para facilitar a la comunidad de código abierto el desarrollo de proyectos exigentes desde el punto de vista informático utilizando el amplio catálogo existente de *Arduino Shields* [56]. Además, se han desarrollado varias API para PCduino que permite al usuario acceder a todas las funciones que esperaría con un lenguaje simple al estilo Arduino, entre sus características novedosas se menciona su micrófono incorporado, un receptor de IR, un puerto de depuración de serie y un encabezado GPIO de 40 pines, una ampliación de sus características se detalladas en la Figura 15 [56].

CPU: 1GHz ARM Cortex A8
GPU: OpenGL ES2.0, OpenVG 1.1 Mali 400 core
DRAM: 1GB
Almacenamiento: 2GB Flash y slot de memoria SD expandible hasta 32GB
Salida de video: HDMI
Sistemas operativos compatibles: Linux3.0 + Ubuntu 12.04, Android ICS 4.0
Requisitos de alimentación: 5VDC, 2A
2 puertos USB
1 puerto Ethernet 10/100Mbps
Lenguajes de programación: C, C++, Python (Ubuntu) y Java (Android).
I/O:
14 pines digitales GPIO
1 UART RX
1 UART TX
2 pines PWM
6 pines ADC
4 pines SPI
2 pines I2C

Precio: Alrededor de los 100$

Figura 15. PCduino 4 NANO [56].

d) **ODROID-C2:** Es una computadora de placa reducida de cuatro núcleos a 64 bits, además puede funcionar como un decodificador de cine en casa, una computadora de propósito general para navegar en la web, jugar y socializar, una herramienta compacta para el trabajo universitario o de oficina, un dispositivo de prototipos para retoques de hardware, un controlador para domótica, una estación de trabajo para software desarrollo, y mucho más. Algunos de los sistemas operativos modernos que se ejecutan en el ODROID-C2 son Ubuntu, Android, *ARCHLinux*, Debian, con miles de paquetes de software de fuente abierta disponibles [57].

El pequeño tamaño del procesador ARM, la complejidad reducida y el bajo consumo de energía lo hacen muy adecuado para dispositivos miniaturizados como dispositivos portátiles y controladores integrados. Actualmente se la considera como la gran competidora directa de la Raspberry Pi. Con la única exclusión de no contar con conectividad Wifi de serie (algo que se soluciona con un adaptador), la versión de ODroid-C2 es su versión más reciente y la más potente de su familia, tiene más memoria RAM, puerto de infrarrojos y su salida HDMI permite una resolución de vídeo 4K a 60 Hz con soporte H.265, de igual manera la ampliación de sus principales características se describe en la Figura 16 [57].

Procesador: Amlogic S905 SoC ARM® Cortex®-A53 (ARMv8) arquitectura de ARMv8 cuádruple de 1,5 GHz @ oblea de 28nm
Memoria: 2Gbyte DDR3 SDRAM y un acelerador 3D ARM® Mali ™ -450 OpenGL ES 2.0/1.1
Almacenamiento flash: Conector eMMC 5.0 Socket: 8 ~ 64GB Módulo eMMC (opcional)
Ranura para tarjeta MicroSD: 8 ~ 128GB MicroSD UHS-1 (opcional)

- Host USB 2.0: Conector de tipo A de alta velocidad estándar x 4 puertos
- Ethernet / LAN Ethernet de: 10/100/1000 Mbps con conector RJ-45
- Salida de audio/video: HDMI2.0/HDMI / I2S
- Expansión de IO Puerto de: 40 pines (GPIO / UART / I2C / ADC)
- WiFi USB: IEEE 802.11b / g / n WLAN con antena (módulo USB) (opción)
- Poder Potencia : 5V 2A (opcional)
- Software del sistema: Ubuntu 16.04 o Android 5.1.x en Kernel 3.14

Precio: alrededor de 90$

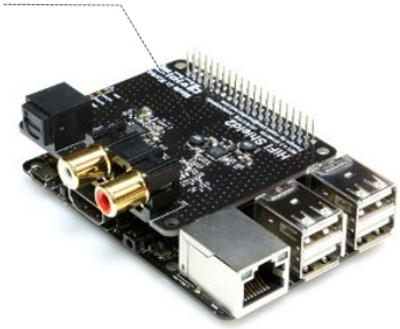


Figura 16. ODROID-C2 [57].

2.3 Propuesta de Solución

El propósito de diseñar un sistema de adquisición de datos, busca experimentar las capacidades de desempeño en versatilidad de control en tiempo real y robustez de construcción en hardware para sistemas empotrados de bajo costo con aplicaciones específicas a nivel industrial, para lo cual se aplican conceptos de código abierto en arquitecturas de comunicación determinadas por OPC-UA. Bajo este marco se propone implementar la arquitectura de referencia con una implementación combinada de características software bajo lenguajes de programación orientada a objetos, a la vez que, combinada con la nueva generación de sistemas SBC, se pretende demostrar la eficacia en rendimiento y control heterogéneo dentro de procesos industriales sin una excesiva sobrecarga ni dependencia de recursos Hardware y Software, mediante la aplicación de dos sistemas control basados en arquitecturas de lazo cerrado y abierto, con la finalidad de monitorear los procesos de nivel y temperatura del módulo FESTO® MPS-PA Compact Workstation como caso de estudio experimental.

CAPÍTULO III
METODOLOGÍA

3.1 Modalidad de investigación

Proyecto de investigación aplicada, con dependencia de información específica para el desarrollo del proyecto y aplicación de conocimientos previos, desarrollados en conjunto para resolver la problemática planteada utilizando las siguientes modalidades de investigación:

3.1.1 Investigación Bibliografía

Al recopilar información de libros, revistas, artículos científicos, tesis doctorales, web entre otros y tabularlos, con esto se deducen y concluyen los patrones de configuración sistemática para el correcto desarrollo del presente trabajo.

3.1.2 Investigación experimental

Se ejecuta este método, al desarrollar pruebas preliminares de trabajo en campo para el prototipo propuesto.

3.2 Población y Muestra

Por las características de la presente investigación no se requiere población y muestra.

3.3 Recolección de Información

Para el presente proyecto de investigación se considera la recopilación de información en base a artículos científicos, tesis, revistas, manuales, libros, páginas de internet, etc.

3.4 Procesamiento y Análisis de Datos

Una vez obtenida la información se procede con los siguientes pasos:

a) Análisis, revisión e interpretación de la información recopilada.
b) Selección de alternativas para dar solución al problema planteado.
c) Pruebas de funcionamiento.
d) Control de errores.
e) Interpretar y analizar los resultados.
f) Recomendaciones.

3.5 Desarrollo del Proyecto

El desarrollo del proyecto se basa en las siguientes actividades:

Revisión teórica:

1. Revisar e identificar características de aplicación y comunicación mediante el protocolo OPC-UA.
2. Revisar e identificar características de aplicación como Servidor OPC-UA dentro de ambientes de programación Linux y Windows.
3. Identificar las características de Hardware SBC como elemento base del sistema de adquisición de datos.

Producto: *Detalle y descripción de la estructura y configuración general del Servidor/Cliente OPC-UA.*

Software:

4. Detallar las características de desarrollo y soporte del software (SoC) elegido para crear la aplicación del Servidor/Cliente OPC-UA en sistemas Linux.

5. Generar la configuración necesaria de programación para la ejecución del Servidor y Cliente OPC-UA.
6. Generar la instancia de código para la lectura y escritura de datos Servidor/Cliente y su conexión con las variables físicas.
7. Generar las pruebas iniciales de comunicación OPC-UA Servidor/Cliente atravez de la conexión física ethernet.

Producto: *Enlace de comunicación OPC-UA Servidor/Cliente mediante el registro de envío y recepción de una variable.*

Hardware:

8. Determinar la correcta etapa de potencia y protección para la lectura y escritura en fusión de las variables físicas a controlar, haci como su correcto sistema conversor ADC auxiliar.
9. Generar el dimensionamiento final del sistema de adquisición de datos.

Producto: *Detalle del sistema físico final.*

Pruebas de Sistema:

10. Toma de evidencias de funcionamiento.
11. Corrección de posibles errores en el sistema final.

Producto: *Cuadro de análisis e interpretación de resultados de pruebas realizadas del sistema de adquisición de datos completo.*

CAPÍTULO IV
DESARROLLO DE LA PROPUESTA

4.1 RESUMEN DEL SISTEMA

La estrategia de diseño fue inspirada por diferentes esquemas actuales dentro de sistemas empotrados detallados en investigaciones realizadas, la comunidad Maker de código abierto y la investigación de protocolos de comunicación industrializados disponibles con perspectiva de aplicaciones futuras, como se detallan en la sección 2.1. del capítulo II. En la Figura 17. se puede observar un resumen de la descripción y configuración mínima del cliente, conectado al sistema empotrado con todos sus componentes hardware necesarios. Además, de la integración de un convertidor ADC y su bus de comunicación, la misma que amplía las características de la SBC seleccionada con la finalidad de obtener una correcta adquisición de datos tipo analógicas por parte del servidor. Comunicados a través de una red ethernet.

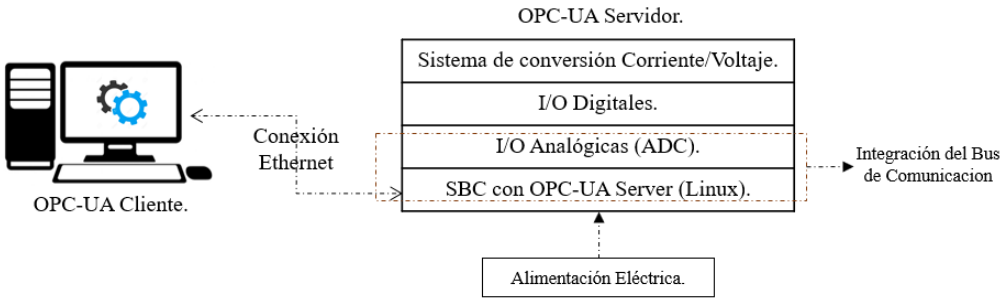


Figura 17. Resumen del sistema Cliente – Servidor.
Fuente: Elaborado por el Investigador.

Los módulos de hardware están dentro del límite del desarrollo del sistema y se eligen cuidadosamente para cumplir características como: precio, capacidades e interconectividad integradas de forma modular. La característica modular intercambiable asegura que el sistema se puede personalizar para cada proyecto, dependiendo de la cantidad y las tarjetas de ampliación de hardware I/O necesarias. Los módulos están divididos en estándar y no estándar.

El módulo estándar significa que no está personalizado entre proyectos y se pueden intercambiar sin programación adicional o cambio en el software, como es el caso de la etapa de potencia, control analógico, conversión Corriente/Voltaje y su alimentación desde la red principal de energía. El módulo no estándar tiene el software principal que puede variar entre proyectos y, por lo tanto, no puede simplemente sustituirlo sin cargarlo y compilarlo nuevamente. Como en el caso de los convertidores ADC, debido a que cada chip conlleva una exclusiva programación para su bus de comunicación con el servidor, sin dejar de mencionar al módulo de servidor y cliente OPC-UA.

En resumen, un computador personal (PC) ejecuta el cliente OPC-UA en Windows 10 para controlar y visualizar los datos del sistema. El servidor OPC-UA está programado en un ambiente Debian GNU/Linux dentro de un ordenador de placa reducida (SBC). Los módulos de I/O Analógicos y Digitales se comunican con el servidor, mientras controlan los actuadores y los sensores. Consulte la Figura 18. para ver la representación gráfica general de las conexiones entre los dispositivos hardware y software principal.

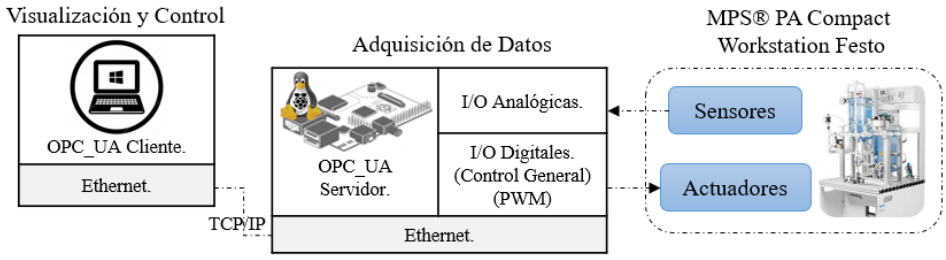


Figura 18. Representación gráfica de conexiones del Sistema Empotrado.
Fuente: Elaborado por el Investigador.

4.2 HARDWARE

La elección del hardware para el sistema se realiza con las siguientes características clave en mente:

- Cantidad de soporte y tamaño de la comunidad en línea.
- Open Source hardware y hardware extensible o "*Shields*".
- Open Source software de aplicación.
- Soporte de bus de comunicación
- Precio.

4.2.1 Elección de la SBC para el módulo de servidor OPC-UA

Existe una amplia gama de hardware SBC disponible en la comunidad Maker, *como se detalla en la sección 2.2.8,* una breve comparación se puede observar en la Figura 19. Los factores de elección más importantes fueron: el soporte de la comunidad en línea y la naturaleza de código permitido y ejecutable. La cantidad de soporte es crucial para que no haya necesidad de "Reinventar la Rueda", lo que acorta el tiempo de desarrollo de diferentes aplicaciones con funciones específicas como lo demuestra investigación planteada. La Raspberry Pi 3 Model B, muestra una ligera ventaja en comparación con sus competidores más cercanos debido a su cantidad de soporte en línea y su precio accesible. Tenga en cuenta que Raspberry Pi es hardware abierto con la excepción del chip principal el procesador de arquitectura "Advanced RISC Machines" (ARM), de Broadcom, que está en el corazón de la Placa.

Por otro lado, muchos otros aspectos de la Raspberry Pi son de naturaleza de código abierto y están disponibles públicamente. El servidor OPC-UA está desarrollado en un ambiente GNU/Linux y puede ser fácilmente portado a otras plataformas de hardware que también sean compatibles con este sistema operativo.

SBC Board	Open Source Hardware	Open Source Software	Soporte en la comunidad Maker	Precio	Soporte en Hardware/Software de ampliación	Implementación de Bus de Comunicación
Raspberry Pi 3 Model B	SI, (con excepción de su procesador)	SI	Muy Alto	Medio	Muy Alto	SPI / I2C
Beaglebone Black	SI	SI	Alto	Alto	Alto	SPI / I2C
PCduino 4 NANO	SI, (con excepción de su procesador)	SI	Medio	Alto	Medio	SPI / I2C
ODROID-C2	SI, (con excepción de su procesador)	SI	Medio	Alto	Alto	SPI / I2C

Figura 19. Comparación entre los diferentes tipos de hardware SBC actuales.
Fuente: Elaborado por el Investigador.

Sus competidores más cercanos son: la Beaglebone Black y la ODROID-C2, sus características hardware las hacen ideales para el control de procesos y la adquisición de datos de forma experimental, ya que en específico cuentan con un ADC integrado que facilitan mucho la adquisición de información física tipo analógica, pero con un mayor precio en el mercado y disponibilidad de adquisición. El sistema en general no necesita hardware más potente que el Raspberry Pi tiene para ofrecer. Por lo tanto, no se necesita más potencia y funciones adicionales, debido a que el módulo OPC-UA solo debe ejecutar un servidor OPC-UA relativamente ligero.

4.2.2 Elección del protocolo de comunicación entre la SBC y módulo de I/O análogas del ADC.

La Raspberry Pi 3 Model B, es compatible con el bus de Interfaz Periférica Serial (SPI) y el bus de Circuito Inter-Integrado (I2C). Pero como se menciona en capítulos anteriores OPC-UA se caracteriza por su velocidad y control estable de datos. Es decir que, para cumplir las exigencias propuesta por la norma UA, es necesario contar con la capacidad hardware necesaria para su cometido. Un punto a favor del proyecto se denota en la implementación del bus de comunicación que conecta los módulos de I/O Analógicas con el módulo de servidor, cuya capacidad nos permite manejar conexiones multimaestro. El motivo del requisito multimaestro se debe a que tanto los módulos de I/O del ADC, así como el *SoC BCM 2837* de la Raspberry Pi deben decidir quién posee el bus principal de control, es decir, quién proporciona la señal de reloj y trasferencia de datos serie asíncrona,

que en resumen genera menos procesos adicionales y pérdidas en los paquetes de información intercambiados. SPI e I2C son dos opciones excelentes de comunicación entre los elementos hardware mencionados, pero con algunas excepciones en específico. A continuación, se detallan características de velocidad de comunicación para ambos protocolos:

- **SPI:** El inconveniente más obvio es la cantidad de pines requeridos en su conexión. La conexión de un único maestro a un único esclavo con un bus SPI requiere cuatro líneas; cada esclavo adicional requiere un pin de I/O de selección de chip adicional en el maestro. La rápida proliferación de las conexiones de pines lo hace indeseable en situaciones donde muchos de los dispositivos deben ser esclavos de un maestro. A pesar de esto, SPI es bueno para conexiones de dúplex completo de alta velocidad de datos, soporta velocidades de reloj de más de 10MHz para algunos dispositivos [44].

- **I2C:** A diferencia de SPI, I2C puede admitir un sistema multimaestro, permitiendo que más de un maestro se comunique con todos los dispositivos en el bus, además su velocidad de comunicación datos se encuentra entre los rangos de: 100 kHz o 400 kHz [44].

Las características de estos buses de comunicación se ampliaron en la sección 2.2.7. Por lo cual, se eligió I2C como el protocolo de comunicación más adecuado, debido a su simplicidad de conexión y su estable velocidad de conexión y trasferencia de datos. Consulte la Figura 20. para obtener una descripción general de las conexiones del bus I2C entre el hardware SoC y el chip ADC.

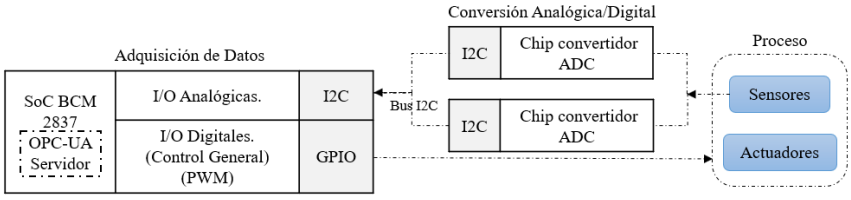


Figura 20. Representación gráfica de conexiones I2C.
Fuente: Elaborado por el Investigador.

Para las SBC que no cuentan con un convertidor Analógico/Digital, como es el caso de la Raspberry Pi o cuando se quiere un ADC de mayor precisión, el ADS1115 logra proporcionar una precisión de 16 bits a 860 muestras/segundo, su detalle físico de conexión se muestra en la Figura 21. Este chip se puede configurar como 4 canales de entrada de un solo extremo, o dos canales diferenciales. Además de esto, el chip cuenta con un amplificador de ganancia programable, hasta x16, para ayudar a impulsar la adquisición de señales individuales/diferenciales más pequeñas a todo el rango [58].

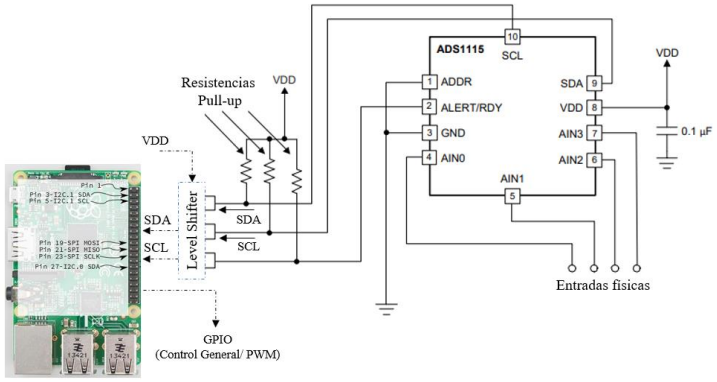


Figura 21. Conexión I2C Raspberry Pi – ADS1115.
Fuente: Elaborado por el Investigador.

El ADC ADS1115 puede operar en un amplio rango de voltajes que van desde 2.0Vdc hasta 5.5Vdc, con un consumo de corriente de 150 uA en modo continuo. Este amplio rango de voltaje le permite ser usado con la mayoría de las tarjetas de desarrollo embebido. Incluyendo una amplia gama de librerías de programación basadas en C, C++ o Python, que la hacen ideal para este proyecto de investigación.

4.3 SOFTWARE

El sistema de adquisición de datos planteado en general contiene un servidor, desarrollado en su totalidad en base a una Interfaz de Programación de Aplicaciones (*API)* diseñado para acceder a los valores de las variables físicas disponibles del proceso a controlar, ya sean de lectura o de escritura en función de sus sensores y sus actuadores. En cambio, el cliente se encarga de recibir la información del servidor y ejecutar las funciones de visualización y control. Un detalle grafico general se puede deducir en la Figura 22.

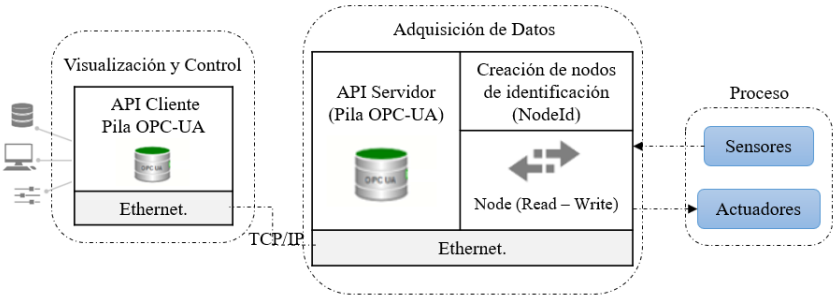


Figura 22. Configuración Servidor - Cliente.
Fuente: Elaborado por el Investigador.

4.3.1 Elección de la API para Servidor OPC-UA

OPC-UA es una especificación de código abierto que permite a los desarrolladores de software crear aplicaciones específicas en dependencia de las necesidades de investigación. En la actualidad existen diferentes grupos de investigación que generan proyectos de código abierto, con lo que, la elección del software para el sistema se realiza en base a las características que se detallan en la Tabla 5.

Tabla 5. Software OPC-UA.
Fuente: Elaborado por el Investigador.

Nombre	**Lenguaje**	**Aplicación**	**Soporte y Categorización en la Comunidad Github**
open62541	C	Cliente/Servidor	Alta (★ 536)
UA.net	C#	Cliente/Servidor	Alta (★ 260)
Node-opcua	JavaScript	Cliente/Servidor	Alta (★ 519)
FreeOpcUa	C++	Cliente/Servidor	Baja (★ 6)
Python FreeOpcUa	Python	Cliente/Servidor	Alta (★ 347)
Eclipse Milo	Java	Cliente/Servidor	Media (★ 208)

El proyecto open62541 se elige por presentar las mejores condiciones de soporte en la comunidad en línea, presentando una detallada documentación que especifica sus características y diseño de aplicación.

Su naturaleza asíncrona de lenguaje de programación “pure C” se considera, además, como otro punto fuerte en aspectos de velocidad de comunicación y lo hace una buena opción para aplicaciones basadas en Servidor/Cliente. open62541 proporciona un marco flexible para construir aplicaciones UA en donde su objetivo principal es tener una biblioteca central que se adapte a todos los casos de uso y se ejecute en todas las plataformas, con la característica esencial de ampliación fácil de su biblioteca principal, ajustándose así, a cada caso de uso a través de la configuración y el desarrollo de complementos.

4.3.2 Elección de la API para Cliente OPC-UA

Como ya se mencionó en la sección 2.2.6, las aplicaciones UA son de naturaleza multiplataforma, eso quiere decir, que puede ser implementado en una gran variedad de lenguajes de programación interpretado o estructurado, por lo que se pueden ejecutar en cualquier tipo de sistema que integre estas características. Esto en conclusión genera grandes ventajas al elegir *software open source*, debido a que es posible combinar varias Pilas API/OPC-UA para la creación, conexión y control de aplicaciones Servidor/Cliente en conjunto.

- **Open62541/Client,** es un software de Interfaz de programación de aplicaciones genérica y compatible con todos los principales modelos de información de OPC-UA mencionados en su estándar, opciones como, explorar los espacios de direcciones, selección de nodos para lectura y escritura de datos, suscripción de llamada y acceso a datos del servidor desde el nodo deseado, están incluidos dentro de sus líneas de programación. Todo esto a través de una aplicación de consola, open62541 aún no cuenta con una interfaz gráfica de procesos o interfaz gráfica de usuario intuitivo como opción para la generación de clientes de manera más amigable y opciones como el desarrollo de procesos basados en alarmas y eventos están en proceso de inclusión hasta la fecha de investigación de este proyecto con su versión v0.3. De otra manera, si se desea realizar aplicaciones gráficas, otros proyectos como *QT/OPC-UA* en combinación con open62541, son una excelente opción de investigación para

proyectos de otra naturaleza. En Resumen, los aspectos en contra mencionados anteriormente, fueron las razones principales por la cuales no se elegido open62541/Client, como herramienta de desarrollo final en esta investigación.

- Por otra parte, proyectos como **Node-OPCUA**, una implementación con una Pila UA completamente escrita en *JavaScript y Nodejs*, son una excelente opción de desarrollo de clientes UA, aprovecha la naturaleza asíncrona de node.js, creando aplicaciones altamente versátiles, además de su amplio impacto dentro la comunidad Maker con una gran acogida, valoración y apoyo, indicadores que representan puntos fuertes para su elección y experimentación en distintos temas de investigación.

- Finalmente, se menciona a **LGPL Pure Python OPC-UA**, una implementación del protocolo binario OPC-UA casi completa y probada en conjunto con una gran variedad de pilas UA diferentes. Su API ofrece una interfaz de bajo nivel para enviar y recibir todas las estructuras definidas en su estándar y clases de alto nivel que permiten escribir un servidor o un cliente con muy pocas líneas de código. Además, Python es un lenguaje de programación interpretado, orientado a objetos y de alto nivel. Sus estructuras de datos integradas de alto nivel, combinadas con la tipificación de forma dinámica, lo hacen muy atractivo para el desarrollo rápido de diferentes aplicaciones. Su sintaxis simple y fácil de aprendizaje, hacen hincapié en su legibilidad y versatilidad, por lo tanto, reduce el costo del mantenimiento de software.

En conclusión, la elección más adecuada para el desarrollo del proyecto presentado se inclina por la utilización de la Pila proporcionada por *Python OPC-UA*, cuya API presenta las herramientas de programación adecuadas para la creación de la aplicación planificada, además de un subconjunto de librerías que permiten ampliar las capacidades de procesamiento, control y visualización del cliente. A esto se le suma su amplio soporte en función de su factibilidad de ejecución dentro de la comunidad de investigadores como se muestra en la Tabla 5 de la sección 4.3.1.

4.3.3 Configuración inicial del servidor OPC-UA

Para llegar a la ejecución del servidor, la Pila debe reunir un conjunto de especificación en función de la plataforma utilizada y datos físicos de tiempo real a monitorear y controlar. Esto debido a que toda la interpretación de código “pure C”, debe ser compatible, incluyendo la creación de nodos, aplicación de Inputs/Outputs de control con los GPIO de la Raspberry Pi y la integración de código en los módulos ADC mediante el bus I2C.

Para iniciar el desarrollo del servidor, se descarga la versión de archivo único de open62541 desde su página principal en Github y se genera su configuración de "amalgamación", como se detalla en la Figura 23. A partir de ahora, se generan los archivos fuente open62541.c y open62541.h en la carpeta de trabajo actual, el uso de Cmake como empaquetador de los archivos generados es de suma importancia en esta instancia, *su detalle de configuración se muestra en el Anexo 1*. Además de esto es necesario obtener características de Multithreading y Build_Certificate, la primera usada para una programación multihilo, útil si se desea generar código que realice subprocesos en paralelo dentro una función principal y la segunda opción utilizada para la generación de certificados Handle usados para el control de acceso e intercambio de información entre el servidor y el cliente de forma segura.

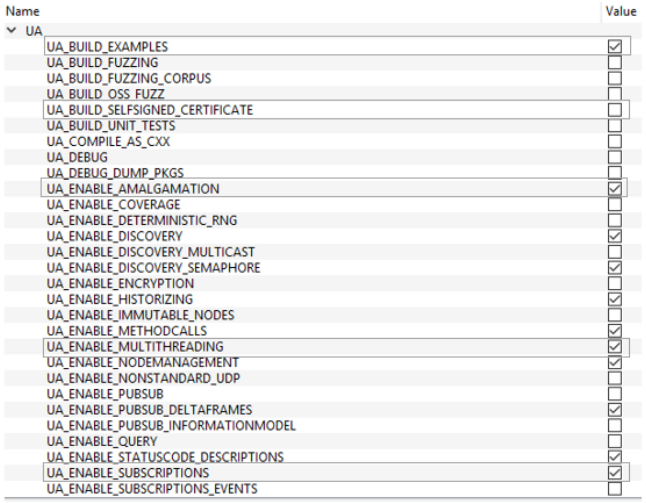


Figura 23. Configuración de amalgamación mediante Cmake.
Fuente: Elaborado por el Investigador.

4.3.4 Conectando el Servidor OPC-UA con un proceso físico.

En las arquitecturas basadas en OPC-UA, los servidores normalmente se encuentran cerca de la fuente de información, es decir, dentro de un contexto industrial, esto se traduce en que los servidores están cerca del proceso físico y los clientes consumen los datos en tiempo de ejecución. *Como se detalló en 4.2.2*, los módulos de I/O son responsables de leer los sensores y controlar los actuadores, además de recibir o enviar datos a través del bus I2C y las GPIO de la Raspberry Pi. En la Figura 24. se detalla de manera resumida la estrategia de programación orientada a la adquisición de datos y manejo de errores implementado en el servidor. En esta sección las siguientes bibliotecas son indispensables para la ejecución de los métodos de llamada y obtención de datos analógicos:

- "open62541.h": archivo tipo cabecera necesario para compilar el servidor.
- "Linux/i2c-dev.h": necesario en la comunicación a través del bus I2C.

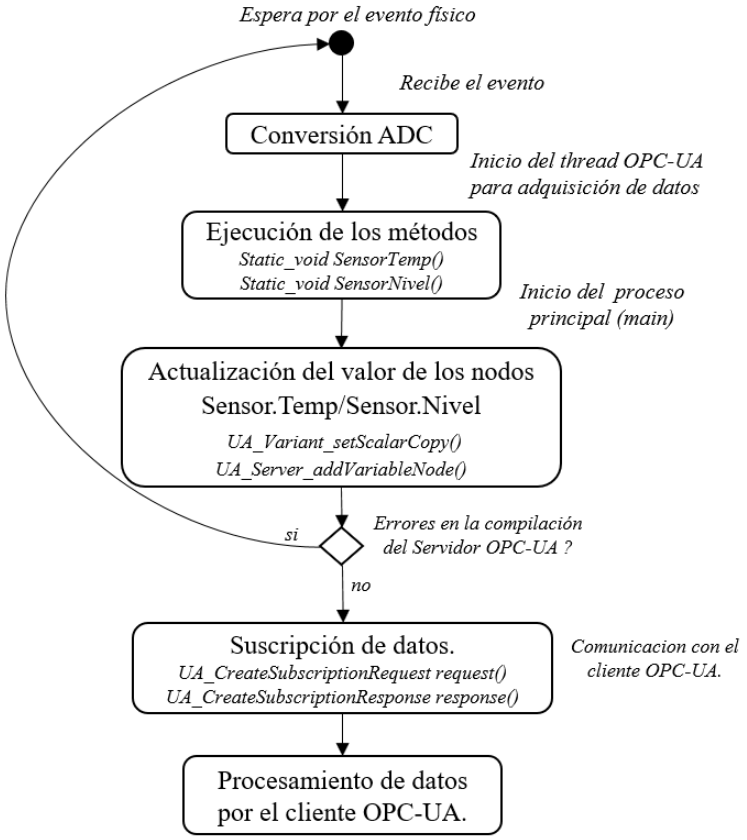


Figura 24. Adquisición de datos analógicos.
Fuente: Elaborado por el Investigador.

El siguiente paso es crear el fichero contenedor del servidor. Para su edición y fácil manejo, se utiliza el editor de texto GNU nano, un editor de código en la terminal de Raspberry Pi que permite compilar código de manera *sudo o administrador*. Dentro del mismo se genera el código de las siguientes secciones:

A

```
#include <fcntl.h>
#include <inttypes.h>
#include <linux/i2c-dev.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <inttypes.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <bcm2835.h>
#include <stdio.h>
#include "open62541.h"
```

B

```
UA_Boolean running = true;

static void stopHandler(int sign)
{
    UA_LOG_INFO(UA_Log_Stdout, UA_LOGCATEGORY_SERVER, "received ctrl-c");
    running = false;
}
```

Figura 25. Ciclo de vida del servidor.
Fuente: Elaborado por el Investigador.

El código en el ejemplo de la Figura 25, muestra la primera parte para la administración del ciclo de vida del servidor, además de los siguientes detalles:

A. Se incluyen las librerías extras necesarias para su correcta compilación.

B. Ejecuta la instancia de inicio del servidor en donde interactúa con la capa de red y procesa los mensajes de solicitud de información por el cliente. Para saber cuándo el servidor ha dejado de funcionar se crea una variable global *running*, que la inicializamos en verdadero. El método *stopHandler ()* capta la señal de interrupción que recibe el programa cuando el sistema operativo intentan cerrarlo, esto sucede, por ejemplo, cuando se presiona *ctrl-C*. El interpretador de señales establece luego la variable *running* en falso y el servidor termina su ejecución.

A esto, se le incluye los métodos *static void ()*, indispensables para la adquisición de datos externos generados por los sensores. La relación de conversión compleja ADC se muestra en las siguientes líneas de código:

```
static void SensorNivel(void *handle, const UA_NodeId nodeid,
                    const UA_Variant *data, const UA_NumericRange *range)

{
  int I2CFile;                                                    C
  uint8_t writeBuf[3];
  uint8_t readBuf[2];
  uint32_t nivel;

  I2CFile = open("/dev/i2c-1", O_RDWR);
  ioctl(I2CFile, I2C_SLAVE, ADS_address);
  writeBuf[0] = 1;
  writeBuf[1] = 0xC3;
  writeBuf[2] = 0x03;
  readBuf[0]= 0;
  readBuf[1]= 0;
  write(I2CFile, writeBuf, 3);
```

```
    while ((readBuf[0] & 0x80) == 0)                                  D
      {
         read(I2CFile, readBuf, 2);
      }
     writeBuf[0] = 0;
     write(I2CFile, writeBuf, 1);
     read(I2CFile, readBuf, 2);
     nivel = readBuf[0] << 8 | readBuf[1];

               UA_Server *server = (UA_Server*)handle;
               UA_Variant value;
               UA_Float con = nivel*5/32767.0;

               if(con >= 8)
               {
               CountNodeValue = 0;
               UA_Variant_setScalar(&value, &CountNodeValue,
                                    &UA_TYPES[UA_TYPES_DOUBLE]);
               UA_Server_writeValue(server, nodeid, value);

               }
               else
               {


               }
 close(I2CFile);
}
```

Figura 26. Adquisición de Datos de Sensores.
Fuente: Elaborado por el Investigador.

El código de la Figura 26, muestra la configuración a la cual está sujeto el *ADS1115* al manipular datos tipo analógicos. Un aspecto importante a tener en cuenta al momento de identificar el direccionamiento del dispositivo en el bus I2C, es que es variable según la configuración física de su pin *ADDR*, para la aplicación implementada se utiliza el direccionamiento *Address = 0x48* como variable global, además de describir los siguientes aspectos en ejecución:

C. Se habilita el buffer para el almacenamiento de datos de escritura y lectura por el dispositivo I2C, además de almacenar el valor de 16 bits de la conversión ADC y añadir el direccionamiento del esclavo. La configuración del puntero lanzado en el byte de escritura *writeBuf [1]* es de suma importancia, debido a que el direccionamiento de los canales A0, A1, A2, A3 encargados de las lecturas físicas en el chip, emplean un valor hexadecimal específico para cada uno de ellos.

   En la aplicación final se emplean los pines A0 y A2 del chip ADC como entradas físicas, para los cuales su direccionamiento hexadecimal es:

   writeBuf [1] = 0xB3 y *writeBuf [1] = 0xC3* respectivamente.

D. Se establece el registro de bytes leídos dentro de un ciclo loop indefinido en la variable *CountNodeValue*, para posteriormente convertirlo en un dato numérico entendible, escalarlo e integrarlo al servidor OPC-UA con la función *UA_Server_writeValue (),* finalmente se cierra el proceso de conversión y almacenamiento con la función de cierre de ciclo *close (I2CFile),* con lo cual se asegura la limpieza del buffer de datos históricos innecesarios con la finalidad de seguir actualizándolos.

El fragmento de código de la Figura 26, registra los datos analógicos de nivel, que son utilizados posteriormente para un control de lazo cerrado ejecutado en la sección 4.4.1. De igual manera otro método llamado *static void () SensorTemp,* se utiliza para el registro de temperatura del sistema en un control de lazo abierto ejecutado en la sección 4.4.5. implementando los cambios sugeridos en la referencia C. *El código completo de adquisición de datos se detalla en el anexo 2.*

En la programación del servidor se incluye un control de salida PWM y ON/OFF utilizado para la el control de lazo cerrado y abierto implementado, los mismos que serán manipulados por el cliente en su GUI correspondiente. En esta etapa de control las siguientes bibliotecas son necesarias para su correcta ejecución:

- "open62541.h": archivo tipo cabecera necesario para compilar el servidor.
- “BCM 2835.h”: biblioteca de uso estándar para el control general o PWM de los GPIO de la Raspberry Pi.

La Figura 27. Detalla el proceso de configuración PWM y su manejo de errores implementado en el servidor, la señal de control se escribe en el servidor mediante la función *out.set_value ()*, que proporciona una escritura de datos de manera constante por parte del cliente, generando a su vez, la configuración de retroalimentación implementada.

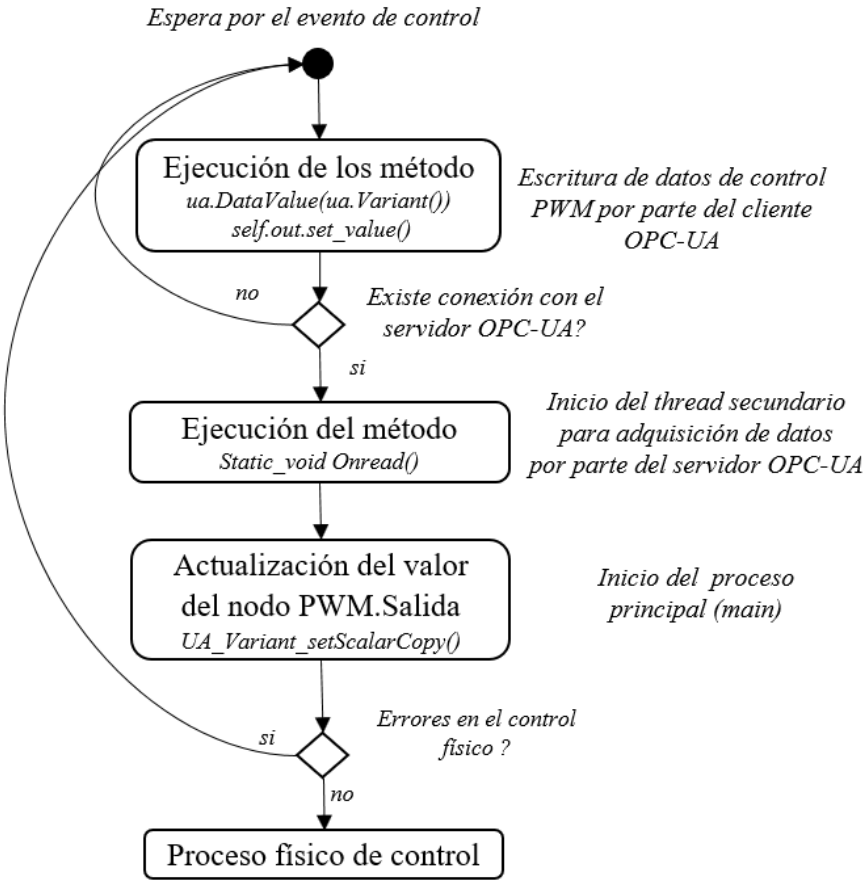


Figura 27. Control PWM.
Fuente: Elaborado por el Investigador.

```
static UA_StatusCode
       onRead(void *handle, const UA_NodeId nodeid, UA_Boolean
              sourceTimeStamp, const UA_NumericRange *range,
              UA_DataValue *dataValue)

 {

     UA_Variant_setScalarCopy(&dataValue->value, (UA_Float*)handle,
                              &UA_TYPES[UA_TYPES_FLOAT]);
```

E

```
//UA_Int32 dato =*(UA_UInt32*)handle;

/*
  if (!bcm2835_init())
        return 1;
    bcm2835_gpio_fsel(LED, BCM2835_GPIO_FSEL_OUTP);
    if(dato == 1)
        bcm2835_gpio_write(LED, HIGH);
   else
       if (dato == 0)
        bcm2835_gpio_write(LED, LOW);

bcm2835_close();
delay (500);
}
*/
```

F

```
if (!bcm2835_init())
        return 1;
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_ALT5);
    bcm2835_pwm_set_clock(BCM2835_PWM_CLOCK_DIVIDER_16);
    bcm2835_pwm_set_mode(PWM_CHANNEL, 1, 1);
    bcm2835_pwm_set_range(PWM_CHANNEL, RANGE);


    UA_Float dato =*(UA_Float*)handle;

        bcm2835_pwm_set_data(PWM_CHANNEL, dato);

        bcm2835_close();
        delay(0.1);

 }
```

Figura 28. Control de Actuadores.
Fuente: Elaborado por el Investigador.

En la Figura 28 se describe el método *static UA_StatusCode onRead ()*, el cual es utilizado para leer las señales de control de retroalimentación recibida por el cliente *UA*. A su vez, permite escribir un valor numérico variable de control en los pines PWM de la Raspberry Pi en uso con la función *setScalarCopy ()*.

E. Representa la etapa de control ON/OFF, su función es generar un pulso de control que puede estar en HIGH o LOW, dependiendo de la configuración indicada por el cliente y que es procesada por el servidor mediante la librería bcm2835 en los pines GPIO.

F. La librería bcm2835 admite hardware PWM en un subconjunto limitado de pines GPIO, proporcionando funciones para su configuración y control. Para que un pin GPIO emita una señal de salida desde su canal PWM, se debe habilitar sus dos canales de control por separado con la función *bcm2835_pwm_set_mode ()*, en alto (HIGH). A su vez, ambos canales deben estar controlados por el mismo reloj PWM, cuya frecuencia se puede modificar llamando a la función *bcm2835_pwm_set_clock ()*. Finalmente, la función *bcm2835_pwm_set_data ()* establece los datos de salida del proceso.

Un aspecto importante de la librería es que su señal de reloj se deriva del valor estándar de frecuencia a 19.2MHz que puede configurarse a cualquier divisor. La aplicación física del proyecto muestra una bomba de agua centrifuga con un motor de corriente continua (DC) el cual se controla con un estándar de PWM a aproximadamente 1 kHz para controlar su velocidad en incrementos de 1/1024 desde 0/1024 (detenido) hasta 1024/1024 (encendido total), para lo cual se configura en un divisor de reloj en 16. Lo que en conclusión genera una frecuencia de repetición del pulso: 1.2MHz/1024 = 1171.875Hz. La Figura 29. enlista las especificaciones de divisor, el período y la frecuencia de reloj a la que pueden ser ajustadas en función de las características del motor DC a controlar:

```
{
    BCM2835_PWM_CLOCK_DIVIDER_1024  = 1024,    /* 1024 = 18.75kHz
    BCM2835_PWM_CLOCK_DIVIDER_512   = 512,     /* 512 = 37.5kHz
    BCM2835_PWM_CLOCK_DIVIDER_256   = 256,     /* 256 = 75kHz
    BCM2835_PWM_CLOCK_DIVIDER_128   = 128,     /* 128 = 150kHz
    BCM2835_PWM_CLOCK_DIVIDER_64    = 64,      /* 64 = 300kHz
    BCM2835_PWM_CLOCK_DIVIDER_32    = 32,      /* 32 = 600.0kHz
    BCM2835_PWM_CLOCK_DIVIDER_16    = 16,      /* 16 = 1.2MHz
    BCM2835_PWM_CLOCK_DIVIDER_8     = 8,       /* 8 = 2.4MHz
    BCM2835_PWM_CLOCK_DIVIDER_4     = 4,       /* 4 = 4.8MHz
    BCM2835_PWM_CLOCK_DIVIDER_2     = 2,       /* 2 = 9.6MHz
    BCM2835_PWM_CLOCK_DIVIDER_1     = 1        /* 1 = 4.6875kHz
}
```

Figura 29. Valores de Frecuencia de Reloj en la librería bcm835.
Fuente: Elaborado por el Investigador.

Posterior a esto, se añade la configuración necesaria para la conexión con el cliente mediante la configuración de su capa de red TCP como se muestra en la Figura 30, en donde se genera su punto final y puerto de escucha. A demás de agregar el detalle de construcción de los nodos ligado a su espacio de direcciones que identifican las variables del proceso a controlar, un resumen de las características expuestas en los nodos creados en esta investigación se muestra en la *Tabla 3 de la sección 2.2.6.*

G

```
int main(void)

{
   signal(SIGINT, signalHandler);

        UA_ServerConfig config = UA_ServerConfig_standard;
        UA_ServerNetworkLayer nl = UA_ServerNetworkLayerTCP
                                          (UA_ConnectionConfig_standard,
                                                                4840);
        config.networkLayers = &nl;
        config.networkLayersSize = 1;
        UA_Server *server = UA_Server_new(config);
```

H

```
  UA_VariableAttributes attr2;

        UA_VariableAttributes_init(&attr2);
        UA_NodeId CountNodeId = UA_NODEID_STRING(1,"SensorNivel");
        UA_Variant_setScalar(&attr2.value, &CountNodeValue,
                                          &UA_TYPES[UA_TYPES_DOUBLE]);
        attr2.writeMask = 0xFF;
        attr2.description = UA_LOCALIZEDTEXT("en_US","SensorNivel");
        attr2.displayName = UA_LOCALIZEDTEXT("en_US","SensorNivel");
        UA_QualifiedName browseName2 = UA_QUALIFIEDNAME(1,"SensorNivel");


        UA_NodeId parentNodeId2 = UA_NODEID_NUMERIC(0,
                                                UA_NS0ID_OBJECTSFOLDER);
        UA_NodeId parentReferenceNodeId2 = UA_NODEID_NUMERIC(0,
                                                  UA_NS0ID_ORGANIZES);


        UA_Server_addVariableNode(server, CountNodeId, parentNodeId2,
                                  parentReferenceNodeId2, browseName2,
                                     UA_NODEID_NULL, attr2, NULL, NULL);

        UA_ValueCallback callbackCount = {server, SensorNivel, NULL};

        UA_Server_setVariableNode_valueCallback(server, CountNodeId,
                                                          allbackCount);
 }
```

Figura 30. Estructura para la creación de Nodos UA.
Fuente: Elaborado por el Investigador.

G. Se genera la configuración del servidor de manera predeterminada y se agrega su capa de red TCP estándar, que establece el proceso *handshaking* para determinar su comunicación desde su inicialización hasta su cierre, a su vez se declara el puerto 4840 por el cual el servidor empieza a escuchar las peticiones de recursos UA de diferentes clientes. La IANA declara el puerto 4840 como estándar en la comunicación de protocolos OPCUA-TCP y OPCUA-UDP, solo cuando la conexión es determinada y con un procesamiento de datos Servidor/Cliente de modo bidireccional. Este modo de comunicación garantiza la entrega de paquetes de datos en la misma orden en que fueron enviados.

H. La creación de nodos o modelos de información basado en objetos está creada para el acceso simplificado a los datos y es una característica fundamental en OPC-UA. Una ampliación de este tema se lo puede realizar en *la sección 2.2.6 en la temática "Modelo de Nodo",* para el caso de investigación se puntualiza en los aspectos característicos más importantes al momento de crear los nodos necesarios. Open62541 proporciona la función *UA_VariableAttributes attr (),* que permite la inicialización de un nodo, tenga en cuenta que la configuración predeterminada tiene un nivel de acceso del valor de la variable como lectura debido a que su dato numérico va a hacer controlado desde nuestro cliente. Su Node-ID, se declara con el uso de la función UA_NodeId *CountNodeId (),* en donde se puntualiza su atributo de identificación y su nombre con el cual será presentado a los clientes, finalmente el nodo objeto se adjunta a nuestro objeto servidor mediante la función *UA_Server_addVariableNode (),* y se integra una función de devolución de llamada de valor, para que al leer, la devolución de llamada proporciona cree una copia del valor actual, e internamente la fuente de datos implemente su propia administración de espacio de memoria para cada nodo.

De manera similar se crean las diferentes configuraciones para los nodos objeto planteados a manipular en los procesos físico detallados en el caso de estudio de la *sección 4.4*, estos parámetros de configuración de nodo se visualizan de mejor manera en la sección *4.6.2 y 4.6.5* de obtención de resultados en las pruebas realizadas.

Finalmente, la función principal *int main ()*, genera un ciclo de manera infinita durante todo el proceso de comunicación mediante el fragmento de código de la Figura 31, salvo el caso que se ejecute el proceso de interrupción manipulado por la variable global *running*, la misma que controla el ciclo de vida del servidor UA, valor de retorno controlado por la función *UA_StatusCode retval ()*.

```
UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return (int) retval;
   }
```

Figura 31. Loop para el Servidor.
Fuente: Elaborado por el Investigador.

4.3.5 Compilación del servidor

Para finaliza el proceso de construcción del servidor UA, se genera su ejecutable mediante el compilador nativo *GCC*, se incorpora la librería *bcm2835* y se lo ejecuta con permisos de *root* para poder acceder sin problemas a los ficheros de los GPIO de la Raspberry PI, como se muestra en la Figura 32:

```
$ gcc -std=c99 open62541.c Servidor_OPCUA.c -o Servidor_OPCUA -lbcm2835
$ sudo ./Servidor_OPCUA
```

Figura 32. Compilación del Servidor.
Fuente: Elaborado por el Investigador.

4.3.6 Configuración inicial del cliente OPC-UA

De igual manera, para iniciar el desarrollo del cliente es necesario adecuar algunos aspectos iniciales:

1) Al igual que el servidor, es necesario del uso de un entorno de desarrollo para la generación del código necesario para su interpretación. Se procede a utilizar un IDE de desarrollado específicamente para proyectos en Python, como lo es *PyCharm*, el

cual proporciona una facilidad de análisis de código, depuración gráfica y un comprobador de unidades de integración multiplataforma.

2) Se descarga la API completa en su única versión desde su página principal del proyecto *FreeOpcUa/python-opcua* en Github, desde la configuración estándar y manejo de librerías que ofrece el *IDE PyCharm*.

3) Además de la Pila Python OPC-UA, es necesario agregar librerías externas muy importantes para la generación de gráficos en tiempo real y la interfaz gráfica final de la GUI, como son: *Tkinter y Matplotlib*.

4.3.7 Conectando el Cliente OPC-UA.

La estructura lógica presentada en la Figura 33 se utiliza para establecer los parámetros de conexión y configuraciones adicionales utilizadas en la conexión con el servidor.

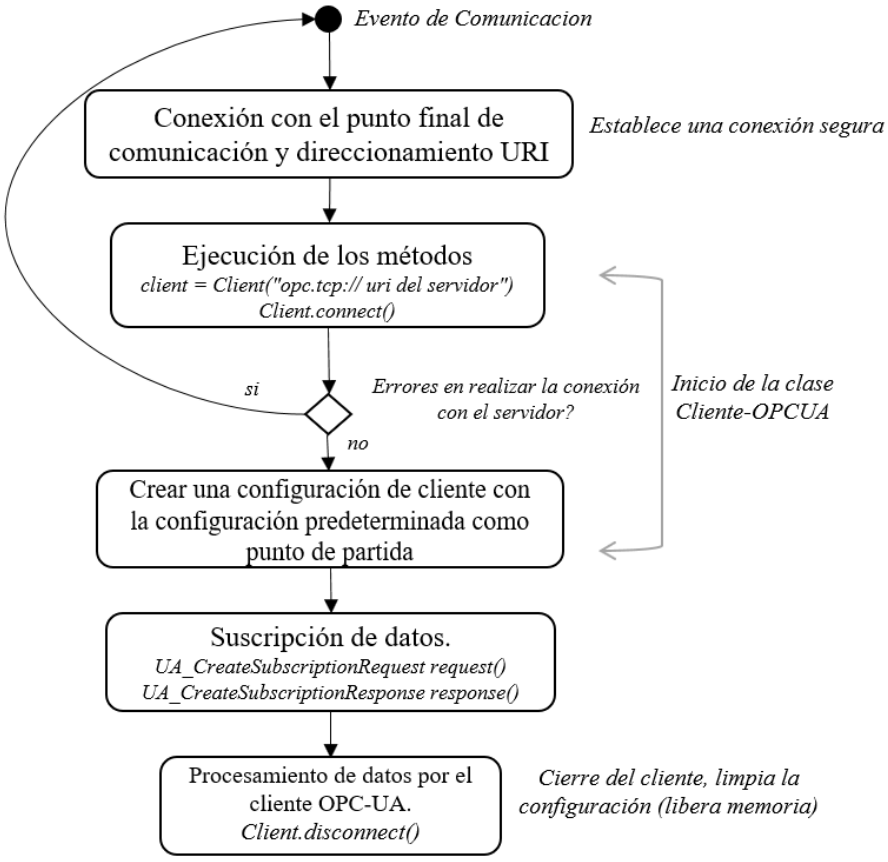


Figura 33. Estructura lógica general del cliente OPC-UA.
Fuente: Elaborado por el Investigador.

De igual manera, la arquitectura interna del cliente se basa en la aplicación de diferentes etapas *Multithread*, que permiten procesar y controlar de mejor manera los datos recibidos por el servidor, al manejar subprocesos individuales y compartir la información entre las clases implementadas. Los beneficios de esta estrategia de programación se derivan en una adecuada optimización de procesamiento de información en tiempo real y uso de recursos PC. Su arquitectura *Multithread* implementada se detalla en la Figura 34.

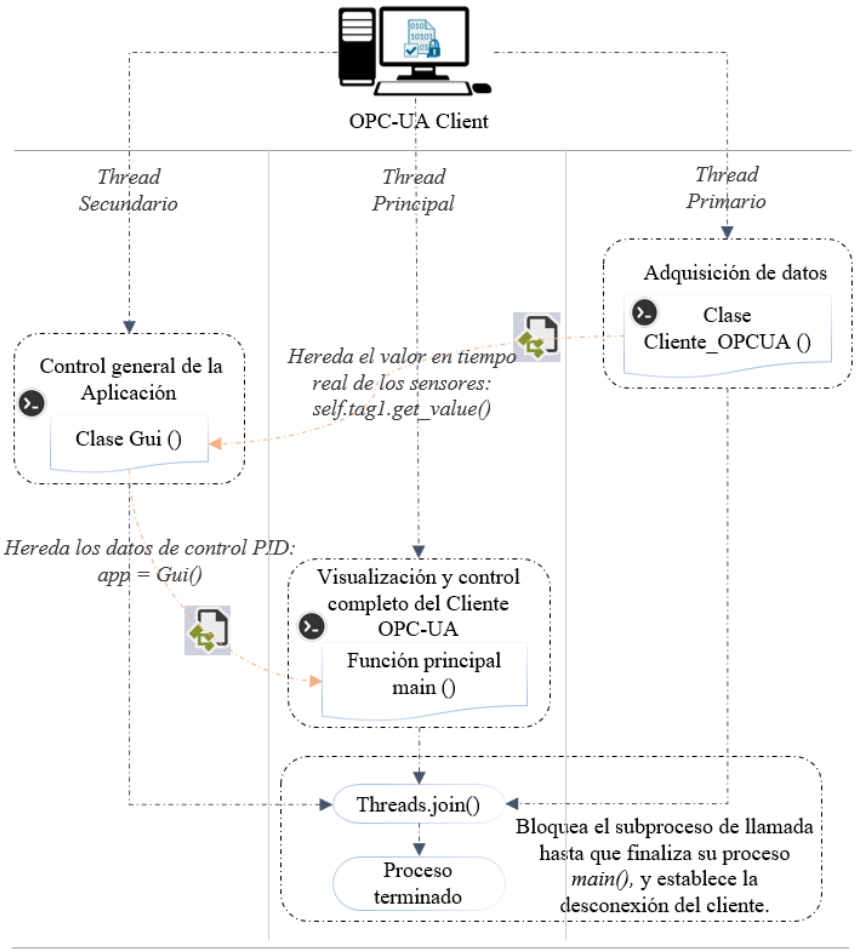


Figura 34. Estrategia Multihilo para el cliente OPC-UA.
Fuente: Elaborado por el Investigador.

La aplicación final del Cliente consta de tres hilos, en donde el proceso principal *main ()* se encarga de procesar todos los datos obtenidos en la etapa de adquisición con la *Clase Cliente_OPCUA ()* y en la etapa de control de retroalimentación y procesamiento grafico en tiempo real con la *Clase GUI ()*. En los siguientes apartados se detalla de manera explícita el código implementado en cada clase.

Como siguiente paso, se crea el fichero principal contenedor del Cliente OPC-UA y se genera código de las siguientes secciones:

A

```
class Cliente_OPCUA(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

        self.url = Client('opc.tcp://169.254.81.206:4840')
        self.url.connect()
```

B

```
        def get_children(node):
            children = extract_name(root.get_child(node).
                                                get_children_descriptions())
            return children

        def extract_name(description):
            qualifiedNames = re.findall(r"QualifiedName\(.*?\)",
                                                    str(description))
            nodeNames = re.findall("\d:[a-z,A-Z_]*", str(qualifiedNames))
            return nodeNames

        root = self.url.get_root_node()

        path = ['0:Objects']
        node = []


        for node in enumerate(get_children(path)):
            print(node[0], ": ", node[1])

        print('----------------------\n')

        max_Nodes = node[0] + 1

        objects = self.url.get_objects_node()
        val = objects.get_children()

        for i in range(max_Nodes):
            print(val[i])

        print('-------------------\n')
```

Figura 35. Inicialización de la clase Cliente_OPCUA.
Fuente: Elaborado por el Investigador.

De manera implícita se deduce la previa importación y configuración de librerías externas, como es el caso de OPC-UA Client y otros módulos del sistema necesarios para el registro y generación de Valores/Acciones en sus diferentes niveles de programación (depuración, información, error, etc) entre otros. Además de las siguientes características que se visualiza en la Figura 35.

A) Se define la clase *Cliente_OPCUA*, esta clase inicial se encarga de la conexión con el servidor. El programa principal establece una conexión segura mediante el uso de la configuración previa realizada en las características del punto final de nuestro servidor, con la función *self.url.connect ()*.

B) En esta sección de la clase se enlistan todos de nodos UA existentes dentro del servidor. El primer ciclo for genera el total de los nodos hijo que se puedan derivar de los objetos principales de la carpeta *root* de nuestro servidor, mientras que el segundo ciclo for, en función del número de nodos encontrado, genera un espacio de direcciones de forma vectorial para cada uno de ellos, es decir, imprime el total y la descripción completa de los nodos encontrados.

C

```
class Gui(threading.Thread):
    def __init__(self, plotLength = 100, origin_time=None):
        threading.Thread.__init__(self)
        self.con1 = Cliente_OPCUA()
        self.raiz = tk.Tk()
        self.raiz.geometry('1200x600')
        self.raiz.title('CLIENTE OPCUA')
        self.start()
```

D

```
    def run(self):

        while True:
            self.Kp = self.en_kp
            self.Ki = self.en_ki
            self.Kd = self.en_kd
            self.set_point = self.set_p
            if self.pwm <= 0:
                self.pwm = 0
                self.dv = ua.DataValue(ua.Variant(self.pwm,
                                                  ua.VariantType.Float))
                self.out.set_value(self.dv)
            else:
                if self.pwm >= 1020:
                    self.pwm = 1020

                    self.dv = ua.DataValue(ua.Variant(self.pwm,
                                                      ua.VariantType.Float))
                    self.out.set_value(self.dv)

                else:
                    self.dv = ua.DataValue(ua.Variant(self.pwm,
                                                      ua.VariantType.Float))
                    self.out.set_value(self.dv)
```

Figura 36. Inicialización de la clase GUI.
Fuente: Elaborado por el Investigador.

En la Figura 36 se describe la inicialización de la *clase GUI ()*, además de crear su propio hilo de subproceso, el mismo que presenta las siguientes características:

C) Se implementa la configuración principal para la creación de la interfaz de visualización y control, para esto se llaman a diferentes funcionalidades que se encuentran en la librería *tkinter*. self.raiz = tk.Tk()*, con la cual se crea una ventana principal llamada raíz, en donde se visualizaran todos los subprocesos implementados en los resultados finales, además de agregar sus dimensiones, color y etiquetación correspondiente.

D) Esta clase también se encarga de administrar el subproceso de control retroalimentado, integrando con un control PID dentro de un loop infinito. *self.start()*, inicializa el hilo secundario y adquiere los datos digitados por el usuario necesarios para obtener los datos de error en el proceso PID, generando las señales de control de salida, los mismos que son enviadas hacia el servidor por medio de la función ***self.out.set_value ()*** en el rango de frecuencia ya preestablecida en el control PWM, con los rangos determinados en la Figura 29 del *apartado 4.3.4.*

La *clase GUI ()* implementa diferentes funcionalidades para la creación de botones, etiquetas y control de herencia de información de los sensores obtenidos en la *clase Cliente_OPCUA* dentro de la presentación final, *este detalle se visualiza y se amplía de mejor manera en la sección 4.5*, en donde las funciones secundarias *def Botones () y def Etiquetas ()*, se encargan de administrar la diferentes funcionalidades y acciones de control de usuario a implementarse, todo esto con la finalidad de tener un control más preciso y dinámico de los subprocesos PID y ON/OFF.

Finalmente se detalla el proceso principal *main ()*, punto final de convergencia de los dos hilos de subproceso mencionados en la estructura presentada en la Figura 34, e interpretados dentro del código presentado en la Figura 37.

```
def main():
```

E

```
          maxPlotLength = 100
          xmin = 0
          xmax = maxPlotLength
          ymin = 0
          ymax = 10
          app = Gui()
          background_label = tk.Label(app.raiz, image=background_image)
          background_label.place(x=58, y=120)
          background_label = tk.Label(app.raiz, image=background_image1)

             fig = plt.figure(figsize=(12, 8), dpi=75)
             ax = fig.add_subplot(211)
             ax.set_xlim(xmin, xmax)
             ax.set_ylim(ymin, ymax)
             ax.set_title('Control PID Nivel')
             ax.set_xlabel("Tiempo")
             ax.set_ylabel("Variable de Control vs Variable de Proceso")

             ax1 = fig.add_subplot(212)
             ax1.set_xlim(xmin, xmax)
             ax1.set_ylim(-20, 120)
             ax1.set_xlabel("Tiempo")
             ax1.set_ylabel("Temperatura")

             lineLabel2 = 'Temperatura °C'
             timeText2 = ax1.text(0.80, 0.9, '', transform=ax1.transAxes)
             lines2 = ax1.plot([], [], color="green", linewidth=0.8,
                                                    label=lineLabel2)
             lineValueText2 = ax1.text(0.80, 0.82, '',
                                                transform=ax1.transAxes)
             plt.xticks([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
             plt.legend(loc="upper center")
             plt.grid(True)

             canvas = FigureCanvasTkAgg(fig, master=app.raiz)
             canvas.get_tk_widget().pack(side=RIGHT)

             anim = animation.FuncAnimation(fig, app.tag1, fargs=(lines,
                                  lineValueText, lineLabel1),interval=1,
                                                        blit=False)

             anim2 = animation.FuncAnimation(fig, app.tag2, fargs=(lines2,
             lineValueText2, lineLabel2, timeText2),interval=1,
                                                        blit=False)
             app.raiz.mainloop()
```

```
if __name__ == '__main__':
```

F

```
    th1 = threading.Thread(target=main)
    th1.start()
    th1.join()
```

Figura 37. Función principal main.
Fuente: Elaborado por el Investigador.

E) Establece el procesamiento de datos de manera gráfica dentro de la GUI, creado con las herramientas de *tkinter**, es decir que, incrustamos e incluimos a *Matplotlib* dentro de *tkinter** con la finalidad de obtener las funcionalidades de graficación en tiempo real en una sola ventana raíz, mediante el uso de la función *FigureCanvasTkAgg ()*, de esta manera se maneja la información del proceso físico de mejor manera y a comodidad del usuario final. La función *animation.FuncAnimation ()*, genera las animaciones de los datos obtenidos en los subprocesos *Cliente_OPCUA() y GUI ()* detalladas en las figuras 35 y 36 respetivamente, además de configurar su presentación y galería visual final. Finalmente, *app.raiz.mainloop ()*, genera el loop indefinido dentro de la ventana raíz de tkinter, haciendo que se actualice de manera automática en cada muestreo de datos graficado.

F) Se implementa el *thread* correspondiente para proceso principal y se inicializa su convergencia a la cola de subprocesos generados con *th1.join()*, con los cual se asegura de terminar e interrumpir los tres procesos creados al mismo tiempo, generando en conclusión una desconexión no forzada del cliente al momento de ejecutarse un enlace con él servidor.

Para una visualización de manera detallada de la estructura de código de la implementación final del cliente, se recomienda revisar el *anexo 3*.

4.3.8 Intérprete del código final del Cliente.

Una característica esencial de Python es que es un lenguaje interpretado. Es decir, que un intérprete es un tipo de programa que ejecuta código directamente sin necesidad de compilarlo antes. Para iniciar el programa con el IDE PyCharm, es necesario que el intérprete de Python seleccione la configuración del proyecto actual de nuestro cliente y ejecute nuestra *aplicacion.py*. Dentro de las configuraciones iniciales de PyCharm se implementan y se administran estos requisitos que no conllevan mayores complicaciones, para finalmente iniciar el intérprete de la aplicación desde nuestra IDE al *presionar Run* o ejecutarlo desde la terminal de Windows al ingresar la ruta: *C:\Codigo\opcua>python GUI_Client_OPCUA.py*, en nuestro caso.

4.4 CASO DE ESTUDIO

Una vez estructurado de manera completa el sistema Servidor/Cliente OPC-UA, se procede a detallar su etapa física de control. Como ya se mencionó en la sección 4.3 el sistema empotrado puede adquirir los datos de cualquier proceso físico previamente estudiado, con lo que para nuestro caso de estudio experimental se lo implementa para el control del módulo FESTO® MPS-PA Compact Workstation.

La maqueta de FESTO® MPS-PA Compact Workstation combina 4 procesos en lazo cerrado, los cuales se pueden operar individualmente, estos son: sistema de nivel controlado, sistema de caudal controlado, sistema de presión controlada y un sistema de temperatura controlada. Además de esto, el diseño de los sensores y actuadores admite una fácil manipulación de sus señales eléctricas análogas normadas industrialmente bajo la norma Namur NE 43 HART cuyos valores de corriente están entre *4 a 20mA* [59].

De igual manera, no es necesario entrar en detalle con las características del módulo Festo manipulado, debido a que ya existe una gran cantidad de información e investigaciones en línea con esta temática en común. Por lo que las siguientes secciones se centran únicamente en cómo se realiza el control de dos sistemas en específico y sus resultados obtenidos, los sistemas son: **Sistema de Nivel** con su correspondiente control PID simple y el **Sistema de Temperatura,** este último con una aplicación de control en lazo abierto.

4.4.1 Control PID del Sistema de Nivel

La bomba P101 suministra un fluido desde un tanque de almacenamiento B101 a un tanque de depósito B102 a través de un sistema de tuberías. El nivel del fluido dentro del tanque B102 se monitorea con un sensor ultrasónico analógico B101 en el punto de medición ‘LIC B101’ y se lee como valor real.

Su valor real debe mantenerse en un cierto nivel, esto de forma independiente de que ocurran alteraciones o cambios en el punto de ajuste. La cantidad de fluido de la bomba P101 puede ser controlado mediante un valor binario ON/OFF de encendido/apagado o también manipulado mediante un control por "Modulación por Ancho de Pulso" (PWM). Para perturbaciones es posible abrir o cerrar parcial o totalmente la válvula de bola V102 que drena el líquido de la parte superior hacia el tanque inferior o al mismo tiempo abrir/cerrar la válvula de mano V104. Su detalle grafico se puede deducir en la Figura 38.

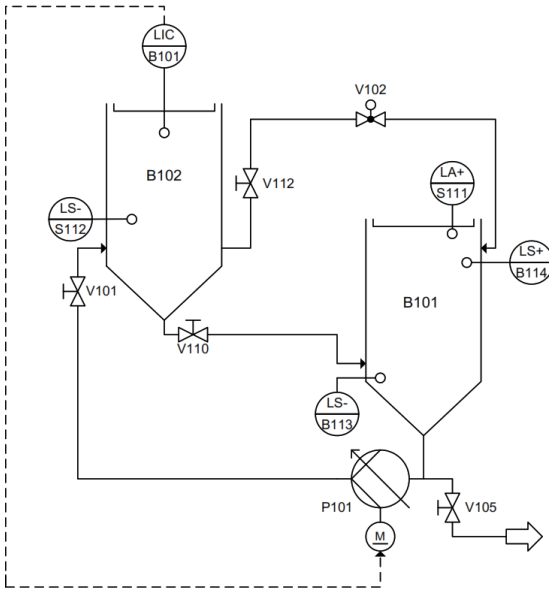


Figura 38. Sistema de nivel controlado (Diagrama P&ID) [59].

La señal de corriente analógica del sensor ultrasónico se conecta como una señal estándar al terminal analógico en el dispositivo transductor/transmisor **RMA42**, *verificar su manual de usurario en el anexo 4,* que a su vez trabaja como elemento trasmisor de procesos, para posteriormente convertir la señal de corriente analógica en una señal de voltaje estándar *de 0 a 5v* y entregarlo en el canal de salida *Analog Output 1 O15(+) y O16(-).* Este paso de transducción nos facilita mucho las cosas al eliminar la necesidad de implementar circuitos de conversión de señal Corriente/Voltaje y solo utilizar las diferentes funcionalidades del **RMA42.**

Implementado de manera física el *Control del Nivel*, se tiene como resultado la configuración que se detalla en la Figura 39. Aquí se denotan dos etapas de acondicionamiento muy importantes dedicados a la estabilización de la señal obtenida por el sensor ultrasónico B101. La primera etapa controla el subproceso conversión Corriente/Voltaje, mientras que la segunda etapa tiene como función principal proteger los canales GPIO de la Raspberry Pi.

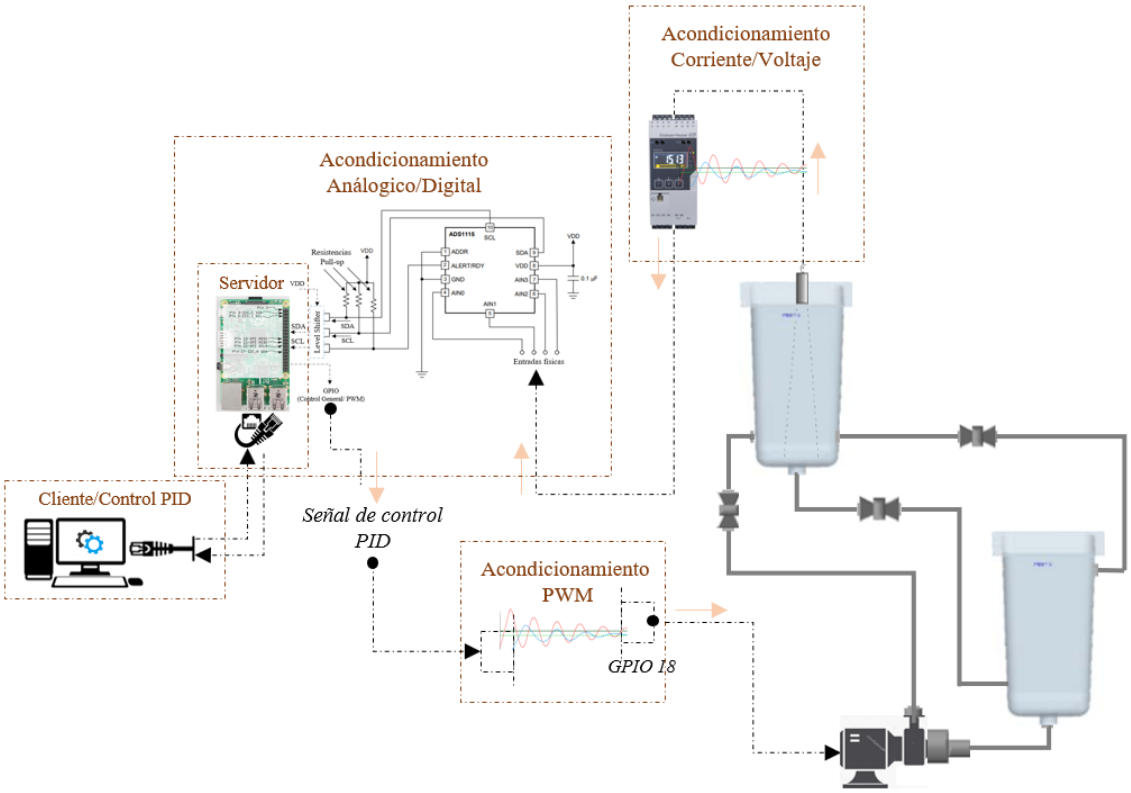


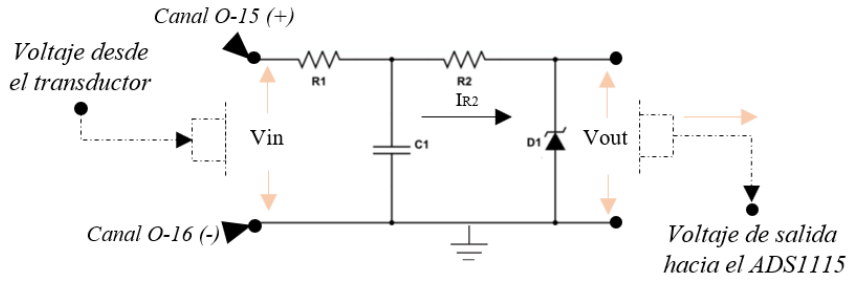
Figura 39. Conexión física del controlador PID.
Fuente: Elaborado por el Investigador.

4.4.2 Acondicionamiento de señal Corriente/Voltaje (Nivel)

Como ya se detalló en 4.4.1. el transductor **RMA42** realiza la conversión corriente/voltaje necesario para el sensor HART Ultrasónico, en una señal variable de rangos que van de: *0 a 5v procesado gracias a una reconfiguración interna de sus salidas analógicas.* Estos valores analógicos estándar a su vez, deben ser procesados por la SBC en donde se compila el servidor OPC-UA, para lo cual se deben cumplir características de estabilidad y nivel cero de voltaje al iniciar el arranque completo del sistema.

Las características físicas de la Raspberry Pi no permiten el manejo de voltajes altos ni corrientes elevadas, por lo que, con razones más que obvias se implementa el siguiente circuito de protección como se detalla en la Tabla 6.

Tabla 6. Circuito de protección de señal de nivel.
Fuente: Elaborado por el Investigador.

<table>
<tr><th colspan="2">Acondicionamiento de Corriente/Voltaje</th></tr>
<tr><td colspan="2"> 
</td></tr>
<tr><th>Ecuaciones y Resultados</th><th>Descripción</th></tr>
<tr><td>Previamente se obtienen los valores característicos del zener del circuito, para este caso se utiliza el zener 1N4732A:

$V_{zener} = 4.7v \quad R1 = 1k\Omega$
$V_{in} = 10v \quad C1 = 4.7uF$
$I_{zener} = 53mA$

En donde la $I_{zener} = I_{R2}$:

$V_{in} = V_{R2} + V_{zener}$ (1)
$V_{R2} = V_{in} - V_{zener}$
$I_{R2} * R2 = V_{in} - V_{zener}$
$R2 = \frac{V_{in} - V_{zener}}{I_{R2}}$
$R2 = \frac{10v - 4.7v}{53mA * 1.1}$
$R2 = 90.9\,\Omega$</td><td>El voltaje de entrada según las pruebas realizadas, presenta un sobre impulso que llega hasta los 10v que sería nuestro voltaje máximo a controlar, el mismo que necesita ser acondicionado a un valor de que no supere los 5v que serán leídos por los pines físicos de nuestro ADC.

El uso de 1watt en las resistencias, dan una buena capacidad de respuesta en el acondicionamiento de la señal, que es teóricamente la potencia total disipada por R2. Finalmente, la resistencia limitadora se sobre dimensiona en función de la facilidad de adquisición en el mercado con lo cual tenemos una R2 final de:
$R2 = 100\Omega$</td></tr>
</table>

Ecuaciones y Resultados	Descripción
La potencia dicipada por la resistenica es: $P_{R2} = \frac{(V_{in} - V_{zener})^2}{R2}$ (2) $P_{R2} = \frac{(10v - 4.7v)^2}{R2}$ $P_{R2} = 0.3364\, w$ La potencia dicipada por el zener es: $P_{zener} = V_{zener} * I_{zener}$ (3) $P_{zener} = 4.7v * 53mA$ $P_{zener} = 0.2491\, w$	Por diferentes fluctuaciones del sistema o subprocesos físicos transitorios, existe la posibilidad de que se cree ruido, los cuales serán eliminados por la etapa RC. El zener es de 1000mW y soportar sin problemas su carga en una condición de volcado para cualquier longitud de tiempo, en donde el circuito funciona por debajo de los 5v con una corriente de 25mA proporcionado por el transductor que a su vez es saturado hasta los 4.9v del voltaje nominal máximo del zener.

4.4.3 Acondicionamiento de señal Analógica/Digital (Nivel)

Como se detalló en la sección 4.4.2, el sensor de nivel funciona en un rango de voltaje de *0 a 5.0v continua*, en donde se escala el nivel físico del tanque, cuya capacidad máxima es de *10litros a 0.5v* por cada litro censado. Pero leer el valor de señal máximo de *5v*, no es una opción ventajosa si se desea generar lecturas continuas directamente con los pines de la SBC de la placa Raspberry Pi, al igual que muchas otras plataformas de bricolaje que funciona con *3.3v* en lugar de *5v*.

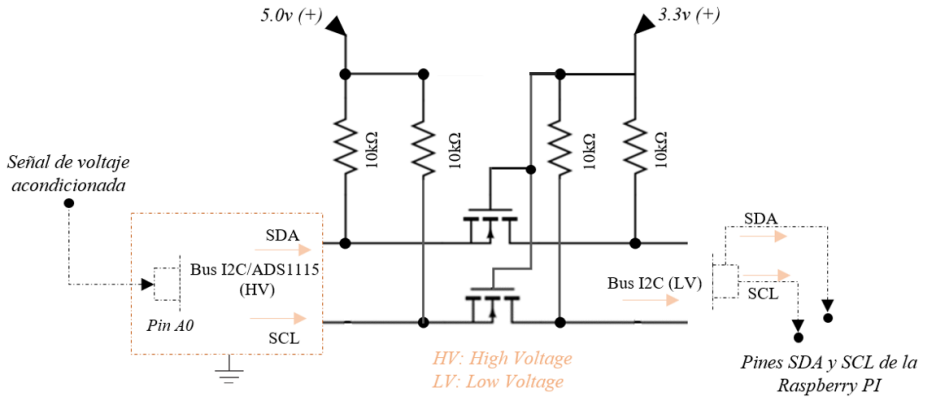


Figura 40. Circuito convertidor de nivel lógico (Pin A0).
Fuente: Elaborado por el Investigador.

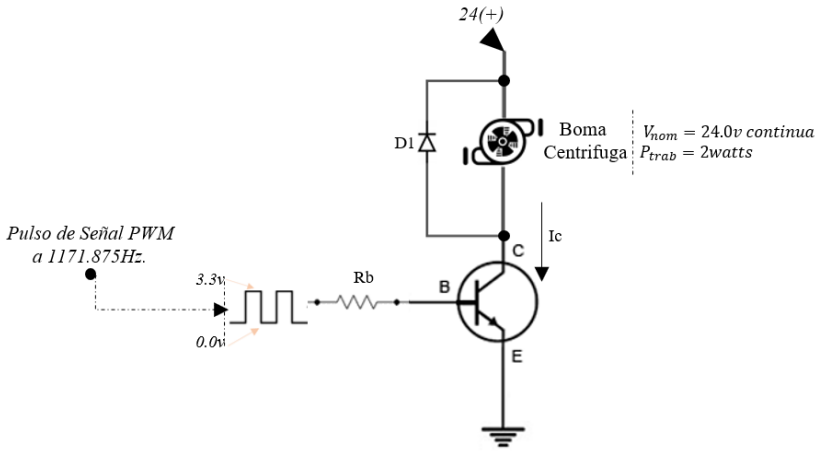
Por lo que, es necesario implementar un circuito *Convertidor de Nivel Lógico* con la finalidad de proteger los pines GPIO, un detalle de su implementación de muestra en la Figura 40. En resumen, un nivel lógico es el voltaje reconocido por un sistema como el voltaje típico de una señal alta. Este nivel de voltaje es utilizado por los chips para determinar si la señal es alta o baja para posteriormente poder realizar diferentes cálculos basados en ello. Tradicionalmente, *5v* era el nivel lógico predeterminado para la mayoría de los sistemas, sin embargo, como los chips de silicio se han vuelto más pequeños y más eficientes, *3.3v* es ahora el estándar para la mayoría de los microprocesadores actuales. El problema principal radica en que, para un chip que funciona a *3.3v*, si conecta *5v* hasta sus pines, a menos que sea específicamente tolerante a un voltaje más alto, puede dañar el pin o todo el chip. Por lo tanto, una manera óptima de proteger los circuitos de conversión, es generar una convergencia del voltaje de nivel lógico de los componentes externos, protectores, módulos u otras placas con la placa/chip al que se está conectando, para nuestro caso las GPIO del chip BCM2837 de la Raspberry Pi.

El circuito mostrado en la Figura 40, no conlleva ninguna dificultad de implementación. Aunque otra opción recomendada es adquirir su encapsulado, como es el caso de los *BOB-12009 de SparkFun*. De igual manera, sus resistencias integradas como control Pull-Up generan una elevación de tensión de entrada o salida que tiene un circuito lógico mientras éste está en reposo. Esto evita que se hagan lecturas erróneas si este pin ya no tiene nada conectado o no está recibiendo una señal. Finalmente, el control de nivel lógico se lo realiza mediante el MOSFET de canal N 2n7000, un mosfet bastante simple y versátil de usar y que para la investigación presentada cumple con todos los requisitos de aplicabilidad.

4.4.4 Acondicionamiento de señal PWM

El objetivo de esta etapa es manipular la bomba de agua de *24v* mediante una señal PWM controlada desde nuestro cliente y proporcionada desde la Raspberry Pi. Como se detalla en la *sección 4.4.3*, los pines GPIO solo proporcionan un voltaje de *3.3v a 50mA* máximo de salida, insuficiente para la manipulación de la bomba, por lo que es necesario implementar una etapa de potencia. En la Tabla 7 se deducen sus componentes.

Tabla 7. Circuito de potencia para el control de la bomba.
Fuente: Elaborado por el Investigador.

Acondicionamiento de señal PWM	
	
Ecuaciones y Resultados	**Descripción**
La bomba centrifuga consume una valor de corriente en etapa de encendido de: $$I_{con} = \frac{P_{trab}}{V_{nom}} \quad (4)$$ $$I_{con} = \frac{2\ watts}{24.0v}$$ $$I_{con} = 0.083\ A$$ $$I_{con} = 83.33mA = I_c$$ Posterior a esto, se calcula la resistencia que se colocar en la base del transistor. Para esto, es necesario conocer su hFe (ganancia de corriente) mínima que tiene nuestro transisto. Su datasheet muestra que su $H_{Fe} = 15$, por lo que la resistencia Rb sera: $$Rb = \frac{V_{in} - 0.7v}{\frac{Ic}{H_{Fe}}} \quad (5)$$ $$Rb = \frac{3.3v - 0.7v}{\frac{0.083A}{15}}$$ $$Rb = 469.87\ \Omega$$	La solución más simple para el caso de estudio, es el uso de un transistor de canal NPN como interruptor diferencial, para este caso se escoge el transistor TIP41-C que soporta una corriente $I_{cmax} = 6\ A$, ideal para el circuito implementado además de su diodo de protección D1 1N4007. La frecuencia PWM del pulso de señal debe ser lo suficientemente rápida como para que la corriente a través de motor no genere o entre en conducción discontinua. A 1171.875Hz, se genera un período de 85.33µs, que, en resumen, es una resolución adaptable y razonable para periféricos PWM presentados por microcontroladores de uso general, en este caso por la placa Raspberry Pi. Finalmente, la resistencia desaturación Rb, se sobre dimensiona en función de la facilidad de su adquisición en el mercado, con lo se tiene una Rb final de: $Rb = 470\Omega$.

4.4.5 Implementación del controlador PID

En este literal se detalla el ajuste del controlador sin entrar en cuestiones de aplicación matemática, pero siendo sustentado por implementaciones software que facilitan el análisis de la plata física de referencia detallada en la sección 4.4.1. La Figura 41 muestra el diagrama de bloques en función de los elementos hardware y software incluidos en el controlador.

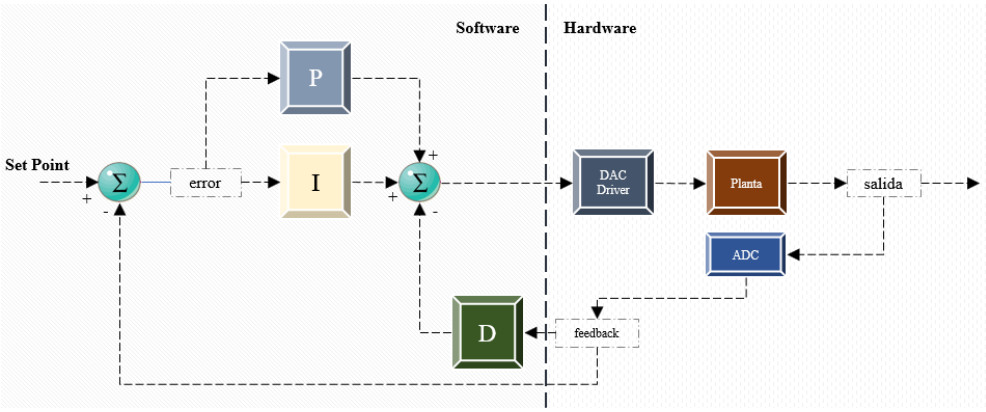


Figura 41. Diagrama de bloques Control PID.
Fuente: Elaborado por el Investigador.

Para el caso de aplicación el elemento derivativo está siendo impulsado solo por la retroalimentación de la planta, esta estrategia se implementa en función de la velocidad de reacción al cambio del error del proceso en donde las condiciones de conducción del fluido de la planta física se tornan inestables al no incorporar una válvula antirretorno. La retroalimentación de la planta se resta de la señal de comando para generar un error, esta señal de error impulsa los elementos proporcional e integral. Las señales resultantes se suman y se utilizan para controlar la variable al *Set Point* deseado.

Las ganancias Kp, Ki y Kd para el sistema se obtienen tras una evaluación y simulación de las condiciones de trabajo de la planta usando las herramientas *System Identification* y *PID Tuner* de Matlab, mediante la configuración de adquisición de datos inicial como se detalla en la Figura 42.

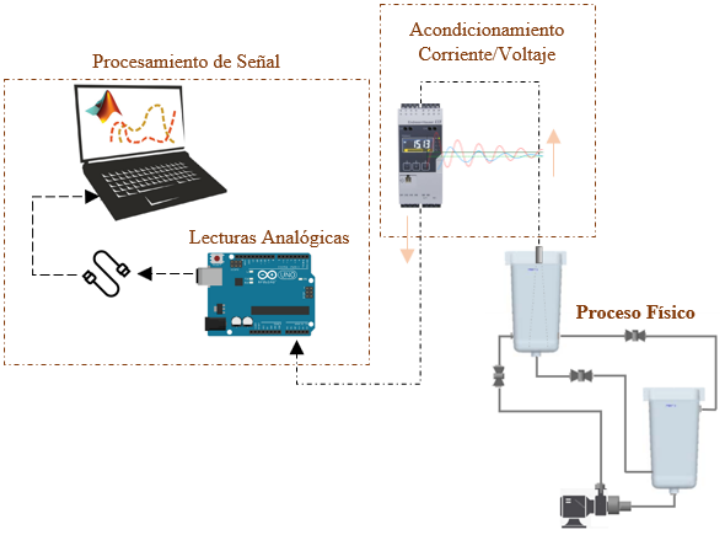


Figura 42. Evaluación inicial de datos con Matlab.
Fuente: Elaborado por el Investigador.

Una vez configurado el Arduino para lecturas análogas, los datos se envían por conexión USB serial siguiendo los siguientes pasos que se describen en la Figura 43:

a) Se inicializa el intervalo de lecturas, ajustando dos parámetros: el tiempo total de medida, y la velocidad de capturas por segundo.

b) A continuación, se genera una nueva ventana grafica o *chart* con sus respectivos ejes, y creamos el objeto gráfico de tipo línea, que se va a actualizar a medida que se tengan datos. De esta manera Matlab no se saturará, que es lo que pasaría si intentásemos utilizar la función *plot()* dentro del bucle *while*.

c) El núcleo del programa es el bucle de medida, en el cual se van leyendo del puerto serie los datos en formato ASCII, midiendo el tiempo de ejecución y actualizando el objeto línea creado anteriormente: los datos Y serán la medida en litros leídos hasta el momento y los datos X el tiempo de ejecución. Al salir del bucle, imprimiremos el dato de capturas por segundo que se ha estimado.

d) Por último, cerramos el puerto serie y eliminamos el objeto serie que se crea en el primer paso.

```
delete(instrfind({'Port'},{'COM5'}));
s = serial('COM5','BaudRate',9600,'Terminator','CR/LF');
warning('off','MATLAB:serial:fscanf:unsuccessfulRead');
fopen(s);
```

a

```
tmax = 50;
rate = 10;

f = figure('Name','Captura');
a = axes('XLim',[0 tmax],'YLim',[0 5.1]);
l1 = line(nan,nan,'Color','r','LineWidth',2);
l2 = line(nan,nan,'Color','b','LineWidth',2);
```

b

```
xlabel('Tiempo (s)')
ylabel('Litros (Lt)')
grid on
hold on

v1 = zeros(1,tmax*rate);
v2 = zeros(1,tmax*rate);
i = 1;
t = 0;

while t<tmax
    t = toc;
    a = fscanf(s,'%d,%d')';
    v1(i)=a(1)*5/1024;
    v2(i)=a(2)*5/1024;
    x = linspace(0,i/rate,i);
    set(l1,'YData',v1(1:i),'XData',x);
    set(l2,'YData',v2(1:i),'XData',x);
    drawnow
    i = i+1;
end
```

c

```
clc;
fprintf('%g s de captura a %g cap/s \n',t,i/t);
fclose(s);
delete(s);
clear s;
```

d

Figura 43. Matlab adquisición de datos.
Fuente: Elaborado por el Investigador.

Una vez se obtiene la respuesta del sistema, se procede a evaluarla en base a una entrada escalón unitario, con la finalidad de modelar sus datos físicos de cambio de nivel. Para el experimento, se aplica un impulso en escalón unitario con un valor de 100, que de manea física se representa como una respuesta de flujo de agua constante por parte de la bomba que funciona al 100% de su capacidad. En la Figura 44, se observa la respuesta del proceso al modelo de salida planteado y su función de transferencia en un sistema de primer orden.

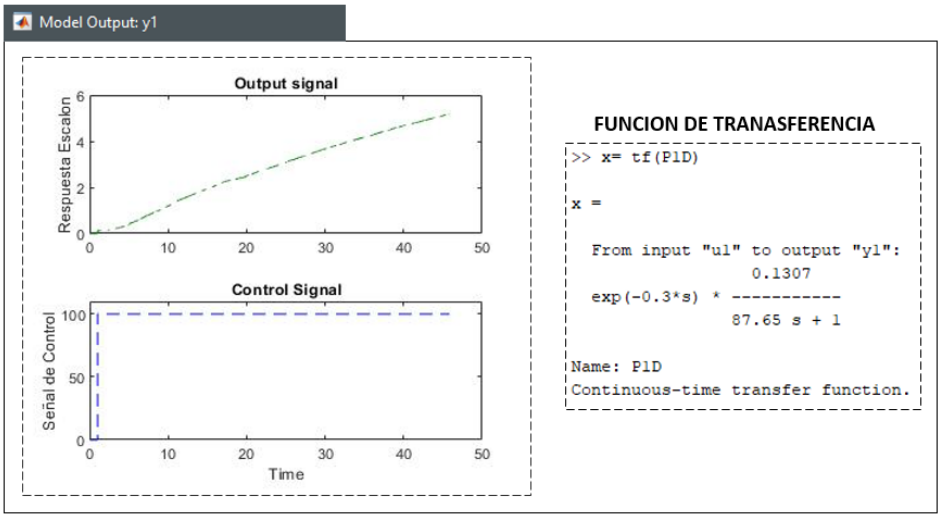


Figura 44. Respuesta Escalón Unitario.
Fuente: Elaborado por el Investigador.

Con el modelo de planta identificado, se ajustan los parámetros de estabilidad del controlador con las herramientas de *PID Tuner*, el mismo que calcula automáticamente sus ganancias a fin de ofrecer una respuesta rápida y estable. Dentro de las especificaciones como se detalla en la Figura 45, el ajuste de ganancias del controlador respectivamente es: Kp = 402.9425, Ki = 48.4258 y Kd = 0. Además de esto, detalla características de estabilidad de lazo cerrado como un sobre impulso del 12% que representa un error de 0,3 litros al punto de referencia y un tiempo de estabilización de alrededor de 30 seg.

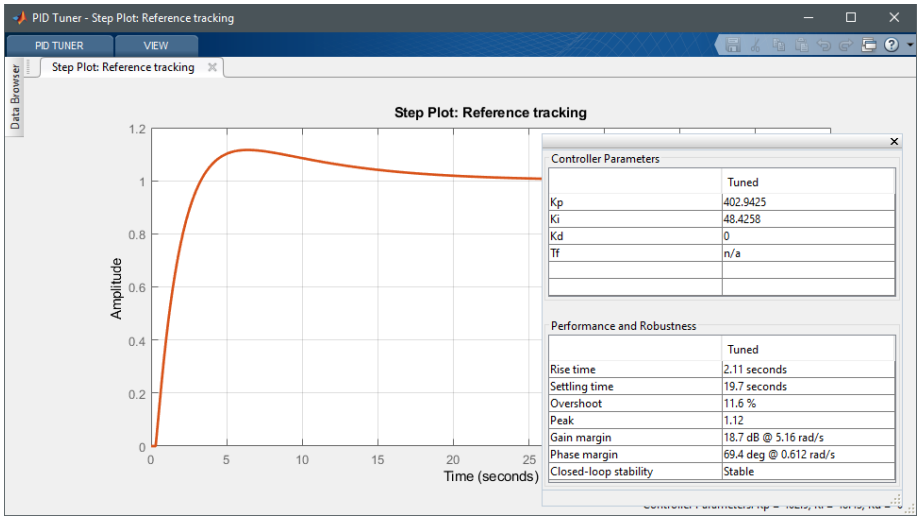


Figura 45. Ganancias PID.
Fuente: Elaborado por el Investigador.

4.4.6 Adquisición de datos del Sistema de Temperatura

El sistema de temperatura consta de un subproceso interno con una autorregulación que controla el valor máximo del líquido hasta sus 65 °C, que se recomienda no ser superado en pleno uso de los tanques. Por otro lado, debido a que la conversión de energía a señal eléctrica ocurre de manera lenta, se le añade un retraso en el proceso de envío de información. El agua en el recipiente del reactor B101 del intercambiador de calor E104 se calienta gracias a un elemento resistivo y se recircula por medio de la bomba P101. Además, se utiliza un sensor PT100 B104 para medir la temperatura del sistema a punto de medición en forma de un valor real. Un detalle de su configuración se muestra en la Figura 46.

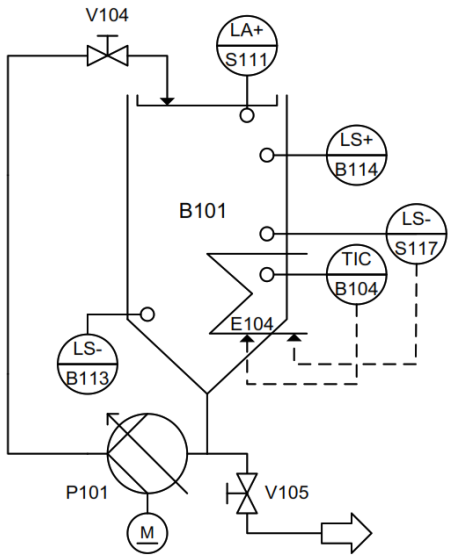


Figura 46. Sistema de temperatura controlado (Diagrama P&ID) [59].

El período de encendido/apagado del elemento calefactor E104, que es la variable manipulada, es determinada por el usuario final dentro del cliente OPC-UA. Para perturbaciones es posible usar fluido frío o simplemente mezclarlo con agua del tanque superior. Finalmente, la resistencia del sensor de temperaturas se conecta a nuestro transductor/transmisor **RMA42,** quien convierte la señal de resistencia en una señal de voltaje estándar de 0 a 5v y es entregado en el canal de salida *Analog Output 1 O25(+) y O26(-)*. Evitando nuevamente su proceso de conversión de señal de Corriente/Voltaje.

Implementado de manera física el sistema se tiene como resultado la configuración que se detalla en la Figura 47. De igual manera que en la sección 4.4.1, de describen dos etapas de acondicionamiento muy importantes dedicados a la estabilización de la señal obtenida por el sensor de temperatura B104 y la protección de las GPIO de la Raspberry Pi al momento de la adquisición de datos y su posterior procesamiento y control por el cliente.

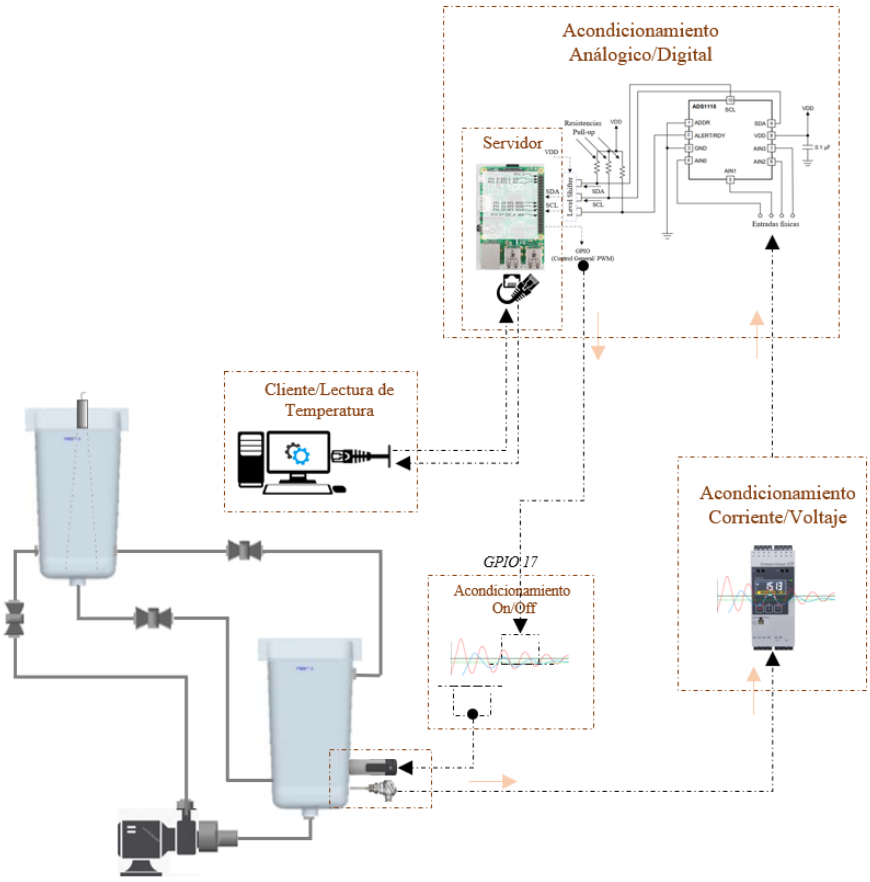


Figura 47. Conexión física adquisición de datos de temperatura.
Fuente: Elaborado por el Investigador.

4.4.7 Acondicionamiento de señal Corriente/Voltaje (Temperatura)

La sección 4.4.2. detalla de manera específica el control de señales realizado por el transductor/trasmisor **RMA42**, por lo que se implementan las mismas condiciones de funcionamiento y protección para la adquisición de temperatura del líquido en esta etapa del proyecto. Es decir que las referencias detalladas en la Tabla 6 son válidas para este literal sin ningún tipo de configuración extra.

4.4.8 Acondicionamiento de señal Analógica/Digital (Temperatura)

De la misma forma, en la sección 4.4.3, se detalla de manera general la etapa de acondicionamiento implementado en la correcta conversión de la señal obtenida por el sensor y controlado por el dispositivo **RMA42**. Además de implementarse la misma configuración de protección mostrado en la Figura 48, que muestra un circuito de control en la convergencia lógica bidireccional de señales de diferente magnitud, pero con un pequeño cambio en el pin físico de conexión en nuestro ADS1115.

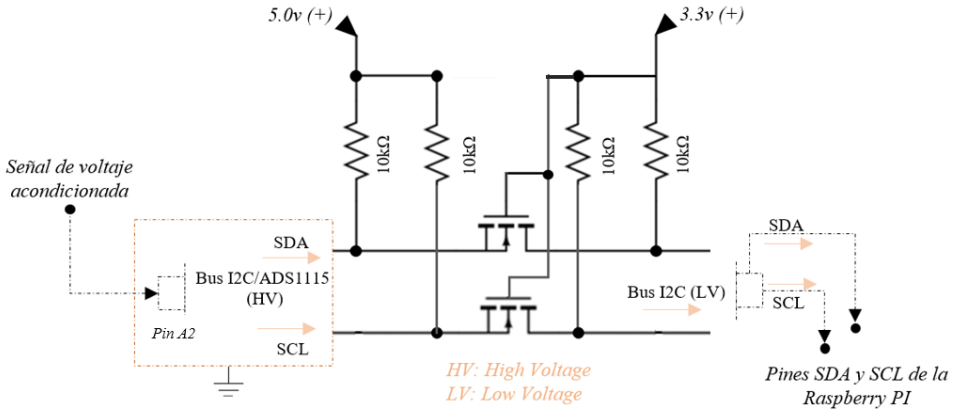
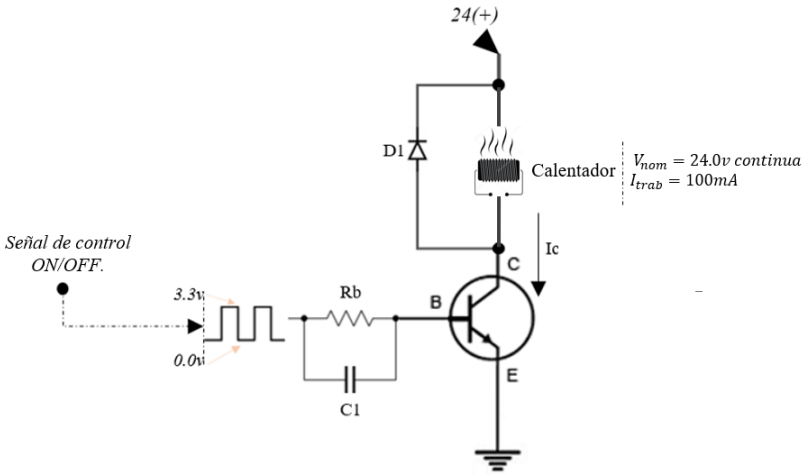


Figura 48. Circuito convertidor de nivel lógico (Pin A2).
Fuente: Elaborado por el Investigador.

4.4.9 Acondicionamiento de señal de control del Calentador

De igual manera, el objetivo de esta etapa es manipular la Calentador de *24v* mediante una señal binaria ON/OFF, controlada desde nuestro cliente y proporcionada desde la Raspberry Pi. Los pines GPIO no generan el voltaje ni la corriente necesaria para cumplir con los requerimientos de potencia del Calentador, por lo que es admisible y lógico implementar nuevamente una etapa de potencia, que aísle el consumo de altas cargas desde y hacia nuestra placa SBC y manipule el consumo de energía necesarias para su correcto funcionamiento. En la Tabla 8 de forma repetida se deducen sus componentes necesarios.

Tabla 8. Circuito de potencia para el control del Calentador.
Fuente: Elaborado por el Investigador.

Acondicionamiento de señal hacia el Calentador	
	
Ecuaciones y Resultados	**Descripción**
La bomba centrifuga genera una potencia en etapa de encendido de: $P_{trab} = V_{nom} * I_{trab}$ (4) $P_{trab} = 24.0v * 100mA$ $P_{trab} = 2.4watts$ En donde su: $I_{trab} = 100mA = I_c$ Posterior a esto, se calcula la resistencia que se debe colocar en la base del transistor. Su datasheet muestra que su gaacia de corriente es: $H_{Fe} = 100$, por lo que la resistencia Rb sera: $Rb = \frac{V_{in} - 0.7v}{\frac{Ic}{H_{Fe}}}$ (5) $Rb = \frac{3.3v - 0.7v}{\frac{100mA}{6}}$ $Rb = 156\,\Omega$	De igual manera, la solución más simple para el caso de estudio, es el uso de un transistor de canal NPN como interruptor diferencial, para este caso se elige el transistor D401 que soporta una corriente $I_{cmax} = 1.5\,A$, ideal para el circuito implementado además de su diodo de protección D1 1N4007. El capacitor C1 ayuda a acelerar el encendido y el apagado del calentador su valor experimental es de 4.7uF. Finalmente, la resistencia desaturación Rb, se sobre dimensiona en función de la facilidad de su adquisición en el mercado, con lo cual se tiene una Rb final de: $Rb = 180\,\Omega$.

4.5 IMPLEMENTACIÓN DEL SISTEMA

Una vez presentado de manera detallada el desarrollo de cada una de las etapas que componen la aplicación Servidor/Cliente OPC-UA, se procede a reflejar los resultados obtenidos y planteados en base a la problemática del objetivo principal de investigación implementada inicialmente en la *sección 1.5*, específicamente con el escenario de la Prueba 1 que se muestra en la *sección 4.6.2*.

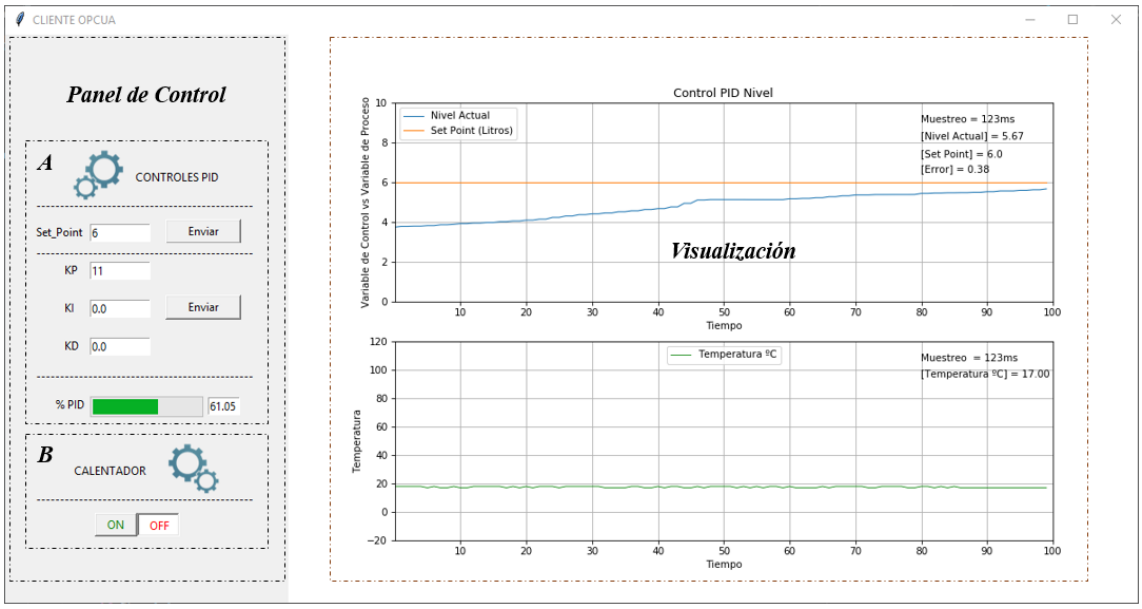


Figura 49. GUI de Usuario Final/Cliente OPC-UA.
Fuente: Elaborado por el Investigador.

De manera resumida la Figura 49, representa al cliente planteado en la *sección 4.3.7* ejecutándose en tiempo real en función de sus dos lazos de control implementados.

PANEL DE CONTROL: se visualizan dos secciones para el control y manipulación de los procesos de nivel y de temperatura por parte del usuario final:

A. Se encarga de generar las configuraciones PID necesarias para el control de lazo cerrado de nivel. El panel de control también incluye una depuración para el ingreso numérico de valores enteros o decimales, con la finalidad de que no existan errores de

lógica de interpretación y cálculos internos al ser procesados por el cliente para los campos de Set Point y Ganancias PID, todo esto, orientado a las ocasiones en donde el usuario ingresa, por ejemplo, letras o símbolos de manera errónea. Su detalle de *ADVERTENCIA* se puede visualizar en la Figura 50.

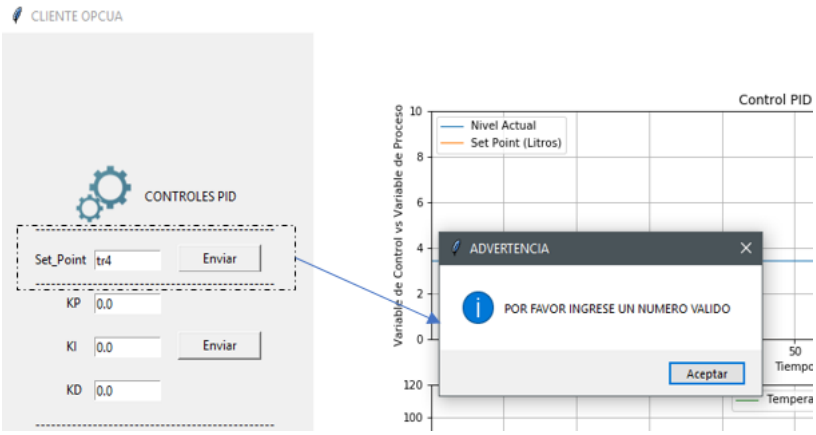


Figura 50. Control de Ingreso de Valores Numéricos.
Fuente: Elaborado por el Investigador.

B. Esta sección del Panel de Control, se encarga de la manipulación del Calentador incorporado en el control de lazo abierto detallado en la sección 4.4.5. con un sencillo control On/Off.

VISUALIZACIÓN: esta sección de la GUI representa de manera gráfica todos los procesos de control realizados en la programación del cliente de manera intrínseca y que se ejecuta en función del panel de principal, con un primer grafico en la parte superior que muestra la evolución del control PID de Nivel en función del tiempo, mientras que el segundo gráfico, muestra la temperatura del fluido en el tanque reservorio B101 del módulo. Además de incluir una referencia de las variables involucradas y su actualización en tiempo de muestreo ubicados en la parte superior derecha.

4.6 RESULTADOS OBTENIDOS

Las pruebas del sistema se han dividido en dos secciones. La primera sección detalla los aspectos de red OPC-UA esenciales en la conexión Servidor/Cliente. Mientras que, en la segunda sección enumeramos los aspectos de rendimiento y capacidad de ejecución del servidor en la Raspberry Pi.

4.6.1 PRUEBAS DE CONEXIÓN.

Una manera recomendable de verificar que se está trabajando bajo el protocolo OPC-UA, es mediante la captura del tráfico de paquetes de datos generado en la conexión y *WireShark* ofrece las herramientas de software de captura y visualización de datos necesarios al implementar nuestra red de conexión mediante TCP/IP.

4.6.2 Escenario de Prueba 1.

En el escenario de la prueba 1, se realiza la conexión física del sistema empotrado en el cual se ejecuta la aplicación *Servidor_OPCUA.c*, con una dirección IP: 169.254.81.206 y el puerto 4840, hacia el cliente en ejecución con la aplicación *GUI_Client_OPCUA.py*, que a su vez tiene una configuración de conexión sin modo de seguridad y un usuario anónimo, su detalle grafico se observar en la Figura 51.

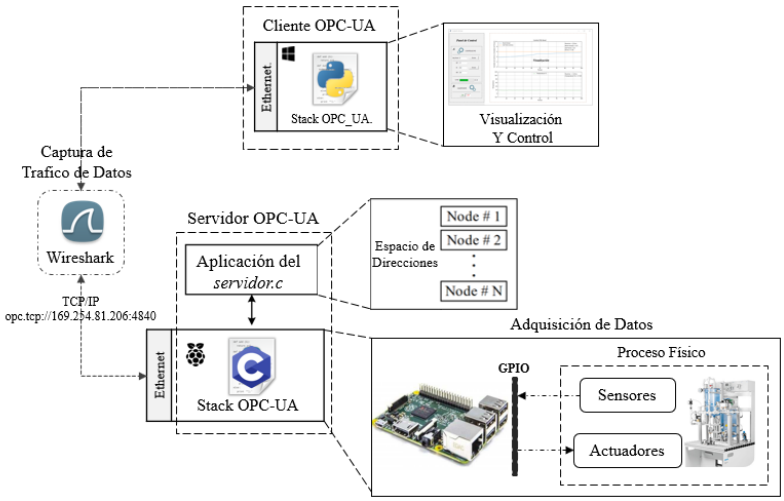


Figura 51. Escenario de Prueba 1.
Fuente: Elaborado por el Investigador.

Al iniciar la compilación del servidor desde la terminal de Raspberry Pi, se muestra la información en ejecución de la Figura 52. que detalla aspectos acerca de su fecha de inicio y punto final, lo que en conclusión especifica que el servidor está en espera de una conexión.

```
$ sudo ./Servidor_OPCUA
[10/16/2018 18:25:56.689]info/network TCP network layer listening on
                                                opc.tcp://raspberrypi:4840
```

Figura 52. Ejecución del Servidor.
Fuente: Elaborado por el Investigador.

Consecuentemente, se inicia la aplicación del cliente dentro de la *IDE PyCharm*, como se detalla en la Figura 53, presentando las siguientes características:

A. Representa el ambiente de desarrollo de la aplicación, específicamente se muestra un fragmento de la clase *Cliente_OPCUA*, en donde se definen los parámetros necesarios para la conexión y la navegación por el espacio de direcciones del servidor.

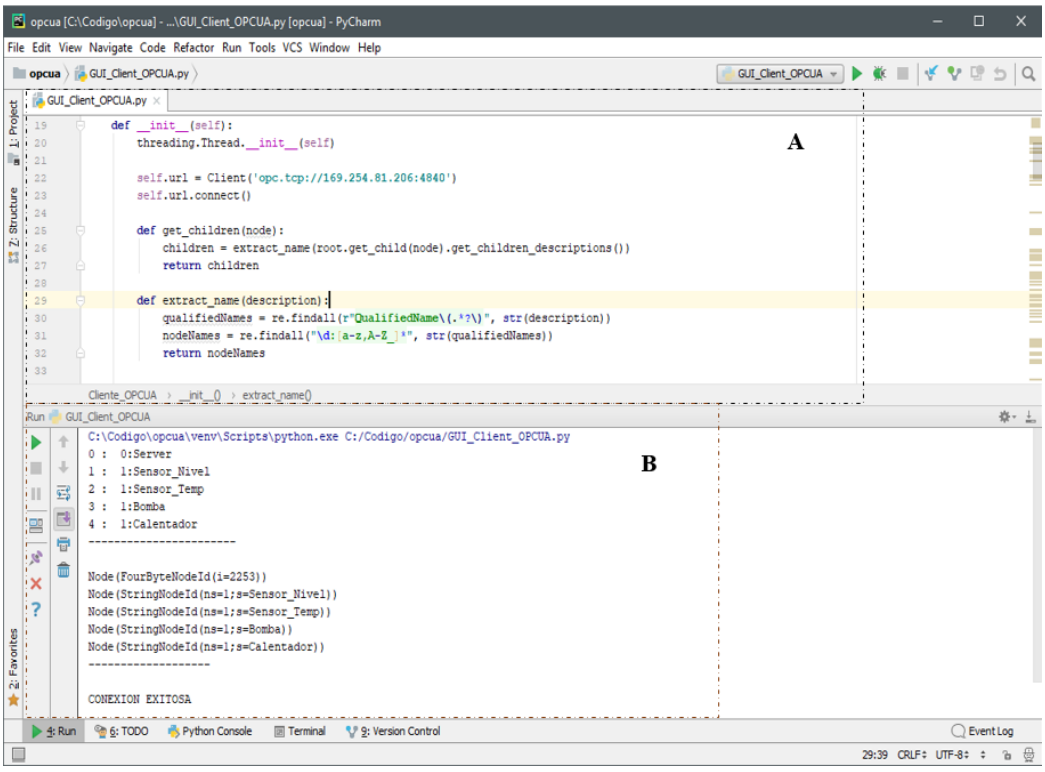


Figura 53. Conexión del Cliente OPC-UA.
Fuente: Elaborado por el Investigador.

B. Muestra el resultado al navegar por el espacio de direcciones. En primera instancia obtenemos la cantidad de objetos UA que controlan diferentes parámetros físicos desde la SBC, además de esto, se enlista los nodos UA referentes a las entradas y salidas del proceso controlado y sus características de identificación.

Lo que, en resumen, representa una prueba de conexión exitosa debido a que las condiciones de comunicación segura, navegación y obtención de datos con relación a los nodos implementados en el servidor se enlistan y se procesan dentro del cliente sin ningún problema. A demás de esto es necesario identificar las características y condiciones de la red OPC-UA que se generan de manera intrínseca en la aplicación Servidor/Cliente, para lo cual se plantea analizar su protocolo de comunicación como se detalla en las siguientes secciones.

4.6.3 Análisis del Protocolo.

En esta sección, se presenta una descripción general del protocolo binario OPC-UA. El mismo que se enfoca en una estructura binaria, que es lo que se ha implementado dentro del servidor con la API del proyecto open62541. El protocolo binario basado en TCP es, con mucho, la capa de transporte más común para OPC-UA en donde también se incluyen conceptos generales que también ligados a seguridad HTTP y etapas de comunicación basada en SOAP definida en el estándar.

El software de captura de paquetes de datos *Wireshark* incorpora un filtro OPC-UA que permite la captura de paquetes compartidos, siempre y cuando la comunicación no esté encriptada. Esto a su vez, permite solucionar problemas de comportamiento extraño o inesperado al realizarse el enlace de comunicación.

Para el escenario planteado se observan las siguientes características en los paquetes capturados, como se detalla en la Figura 54.

La estructura de conversación inicia con un mensaje de saludo del servidor y el reconocimiento del mismo por parte del cliente, como se muestran en los paquetes 283 y 291. Los enlaces *SecureChannels* o inicio de conexión segura, se crean posterior a la capa de reconocimiento de conexión TCP sin procesar y se establecen con los mensajes de solicitud y respuesta *OpenSecureChannel*, que se detallan entre los paquetes 292 y 329, en donde se obtienen, además, las características del punto final de conexión. Aunque un *SecureChanne*l es obligatorio, su cifrado de seguridad aún puede estar deshabilitado y sin modo de políticas de seguridad, como en este caso de prueba de conexión.

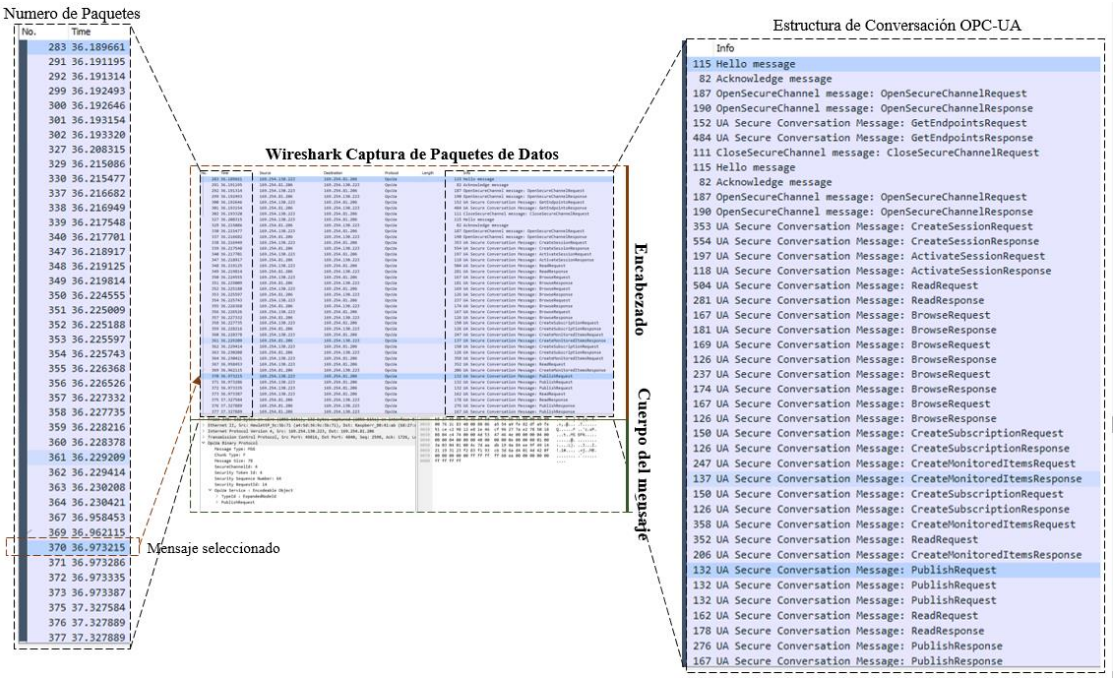


Figura 54. Conversación OPC-UA mostrada en Wireshark – Prueba 1.
Fuente: Elaborado por el Investigador.

Las sesiones *CreateSession* y *ActiveSession* se activan posteriormente a la creación del canal seguro de comunicación y representan la solitud de *Petición/Respuesta* hacia todas las características de intercambio de información procesada desde el servidor y presentada por el cliente presentados entre los paquetes 338 y 347. El servicio *ActivateSession* se puede usar para cambiar una sesión existente a un *SecureChannel* diferente, esto es importante, por ejemplo, cuando la conexión se rompe y es necesario una reconexión. Entre los paquetes 358 y 364, se inicia el proceso de suscripción y monitoreo a eventos que modifiquen el valor numérico o binario de los nodos UA identificados en el servidor.

De manera similar se describen tres tipos de intercambio de mensajes indispensables en la trama de conexión: la primera genera la petición de publicación *Publish Request/Response* de variables con su respuesta de manera continua durante su ejecución, una fragmento de este proceso muestran entre los paquetes 370 y 377, el segundo tipo genera una petición de lectura del estado de variables y la tercera que genera la apertura a escritura de variables *Read Request/Response* mostrada en los paquetes 373 y 375, además de exigir una devolución del estado del nodo y del servidor para cada caso y para cada uno de los paquetes capturado y la tercera conversación *Write Request/Response* que genera la apertura a escritura de datos en los nodos del servidor y su respuesta de estado.

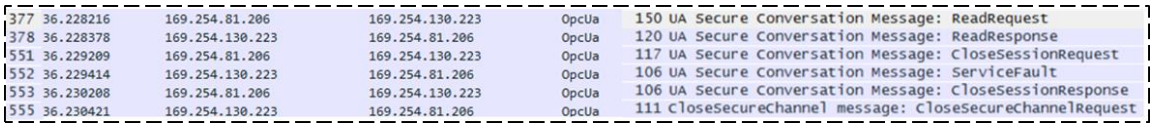


Figura 55. Estructura de cierre de sesión – Prueba 1.
Fuente: Elaborado por el Investigador.

La Figura 55 muestra la etapa final de la trama de conversación, en donde el cliente genera un proceso de cierre de sesión segura y de manera estándar mediante una petición *CloseSessionRequest/CloseSessionResponse* procesado entre los paquetes 551 y 555, caso contrario se generan errores de depuración o desconexiones forzadas.

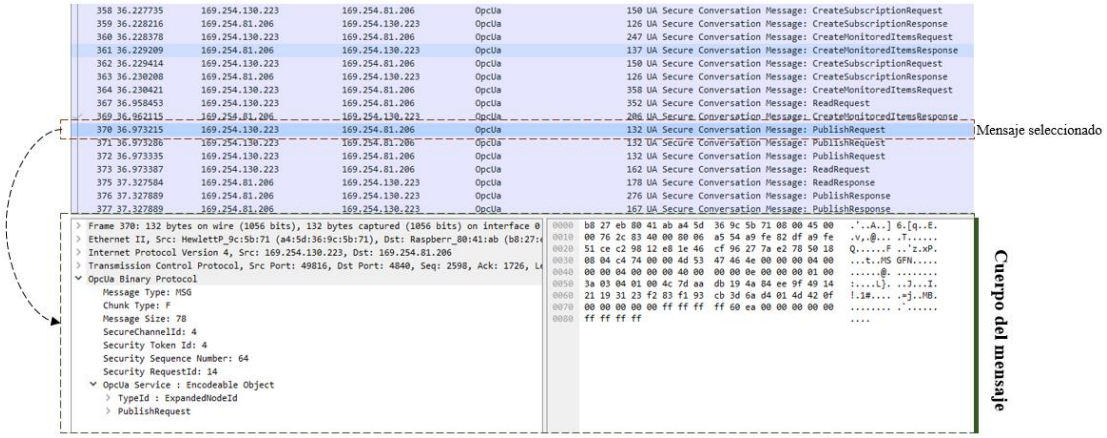


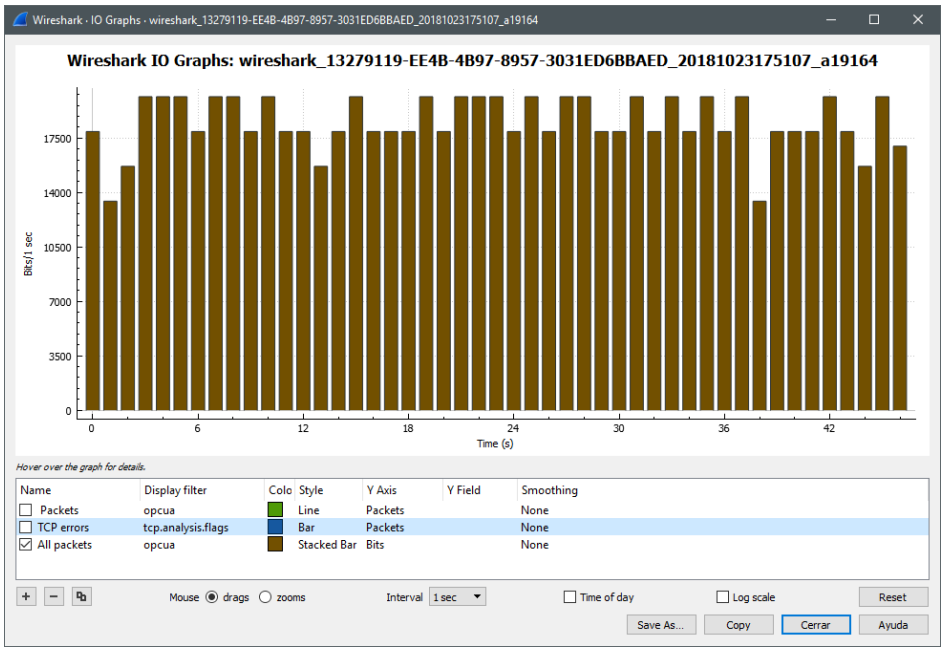
Figura 56. Estructura de mensaje mostrada en Wireshark – Prueba 1.
Fuente: Elaborado por el Investigador.

El cuerpo del mensaje identificado en la Figura 56 comienza con la ejecución del protocolo binario implementado dentro de la red TCP ethernet y detalla las principales

especificaciones del paquete seleccionado, en donde la primera sección inferior izquierda muestra el telegrama individual que incluye la marca de tiempo, dirección (origen a destino) y características numéricas de escritura o lectura del nodo UA, mientras que en la parte inferior derecha se enlistan sus características de encriptación y decodificación hexadecimal. En el ejemplo del paquete 370, se puede observar una petición enviada desde el servidor al cliente para un servicio de solicitud de publicación de estado de nodo UA.

4.6.4 Análisis de Paquetes de Datos.

Tabla 9. Ancho de banda OPC-UA.
Fuente: Elaborado por el Investigador.

Uso de Banda de Ancho

Resultados
Wireshark IO Graph muestra el tráfico general visto en el archivo de captura generado en la sección 4.6.3, que generalmente se mide en velocidad por segundo en bytes o paquetes. Al aplicar el filtro de datos *opcua*, describe un ancho de banda que alcanza un máximo de 18000 bits/s y un mínimo de 13500 bits/s, todo esto con un promedio de 16000 bits/s para todo el grupo de paquetes capturado en función del tiempo, que en conclusión es la cantidad de banda de ancho en uso cuando se genera la comunicación entre el Servidor/Cliente.

De la misma manera se analizan la cantidad de paquetes procesados y los posibles errores que se pueden generar dentro del proceso comunicación OPC-UA. Las estadísticas graficas se pueden deducir en la Tabla 10.

Tabla 10. Procesamiento de Paquetes OPC-UA.
Fuente: Elaborado por el Investigador.

Procesamiento de Paquetes de Datos

Resultados
▪ En la gráfica 1 se detalla la cantidad de paquetes procesados en una petición de solicitud/respuesta, con un máximo de 18 Packets/s y un mínimo de 12 Packets/s, que a su vez generan un promedio de 16 Packets/s por cada intervalo de tiempo mencionado. De la misma manera se intenta graficar la estadística de errores en el protocolo de comunicación con la perdida de paquetes de comunicación, pero al trabajar con un proceso dentro de un ambiente controlado no se generan errores visibles. ▪ La gráfica 2 mide el delta de tiempo entre la captura de un paquete TCP y el ACK retardado (Reconocimiento Retardado) correspondiente para cada paquete. Que en conclusión genera un retardo en el proceso de Solicitud/Respuesta que no supera los 1.4 ms. Este tiempo medido a su vez, guarda una relación importante con la gráfica 1 debido a que, si este tiempo es largo, podría indicar algún tipo de retraso en la red y generaría una pérdida de paquetes, congestión u otros aspectos negativos de comunicación, pero que, para el caso de estudio planteado no se tiene una mayor problemática.

4.6.5 Escenario de Prueba 2.

En el escenario de la prueba 2 se realiza la conexión física del sistema empotrado con las mismas condiciones que en la prueba 1, con la diferencia que en este caso se ejecuta una conexión hacia un cliente de la empresa *Unified Automation. UAExpert* puede ejecutarse en plataformas Windows y Linux, además de tener la capacidad de explorar datos de forma básica y estructurada, administrarlos mediante una interfaz gráfica con una configuración de tiempo de muestreo de variables OPC hasta de 100ms y un control dinámico de lectura de datos. Estas características lo hacen el software SDK indicado para pruebas con servidores OPC-UA y que para nuestro caso de estudio se desean realizar pruebas de confiabilidad y capacidad de comunicación atravez de la red. Su detalle de configuración inicial se observar en la Figura 57.

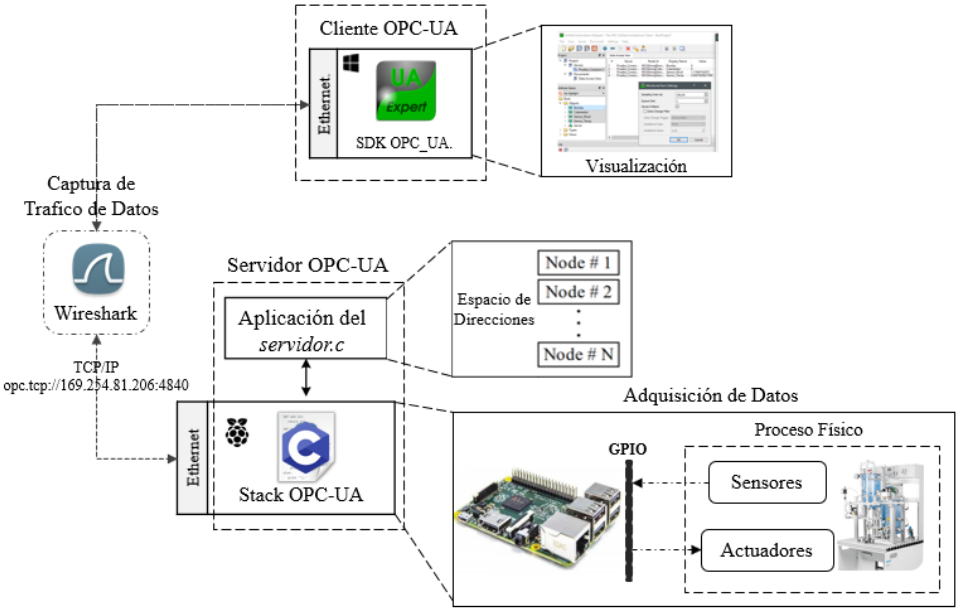


Figura 57. Escenario de Prueba 2.
Fuente: Elaborado por el Investigador.

De igual manera que en la *sección 4.6.2*, el servidor genera el proceso de inicio de espera y conexión con un cliente. Además de esto es necesario generar la configuración necesaria dentro del cliente *UAExpert* en función de las condiciones a las cuales se ejecutó el cliente con la aplicación *GUI_Client_OPCUA.py*. En la Figura 58, se detalla de mejor manera el proceso de configuración inicial para el cliente de la prueba 2.

Al configurarlo con el punto final y el puerto de comunicación correcto, se genera la conexión deseada, en donde se puede observar que los nodos planteados y preestablecido en el servidor se enlistan de manera correcta en el navegador de espacio de direcciones de *UAExpert*, así mismo, su lectura entiempo real y estado de las variables se visualizan sin ningún problema, esto en conclusión propone una conexión exitosa.

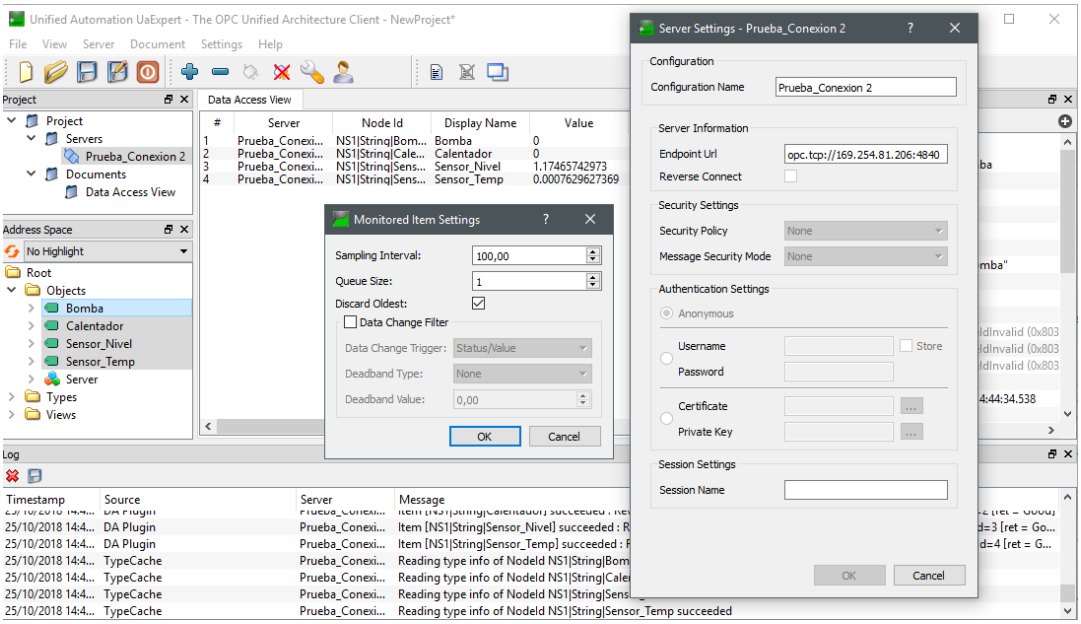


Figura 58. UAExpert Cliente OPC-UA - Prueba 2.
Fuente: Elaborado por el Investigador.

4.6.6 Análisis del Protocolo.

Se procede de manera similar al realizar el análisis del protocolo de conexión, generando la siguiente estructura:

- Filtro de captura: Puerto 4840
- Filtro de visualización: opcua
- Protocolo: OpcUa
- Negociación inicial, fase 1 (conexión con los puntos finales del servidor)
- Negociación inicial, fase 2 (inicio de sesión)
- Lectura, suscripción y recepción de variables.
- Cierre de sesión y canal.

El cuerpo del mensaje comienza con el identificador del tipo de dato, en el ejemplo se hace referencia a los paquetes 352 y 354 en donde se presentan las peticiones *Read Request/Response* y *Write Request/Response*, para la lectura y escritura del estado del valor binario del nodo UA destinado a la salida de control para el calentador en donde presenta una salida *Variant Type* de 1 o estado ON (encendido).

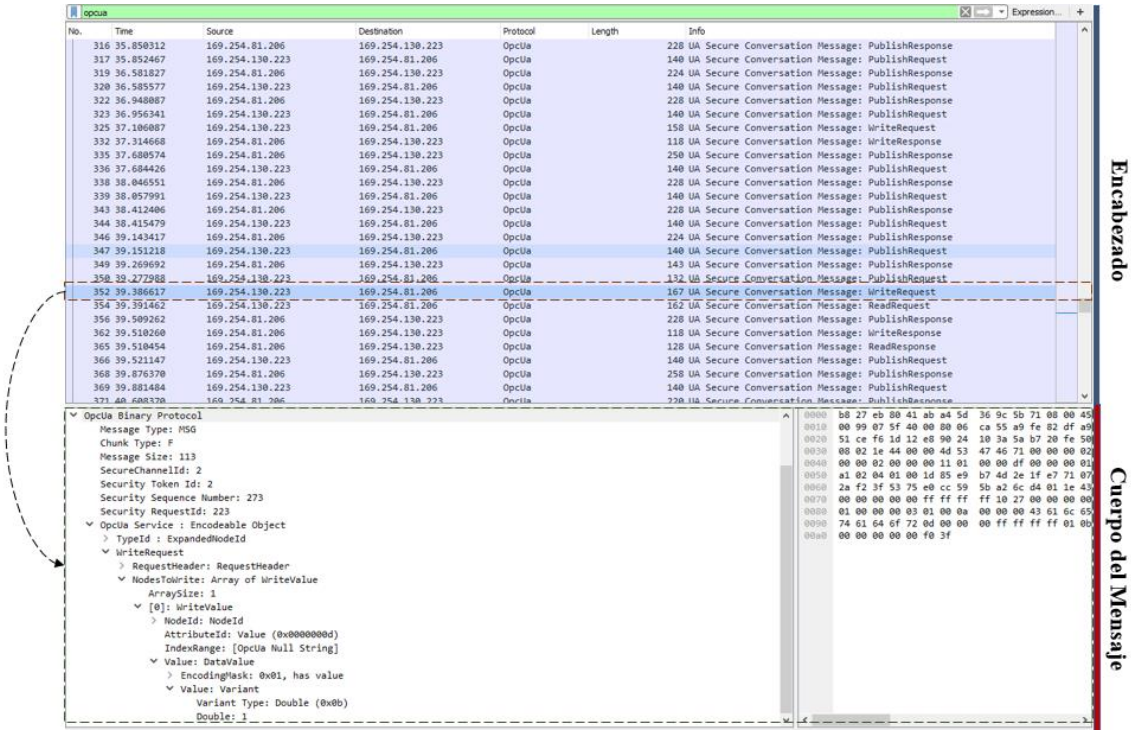


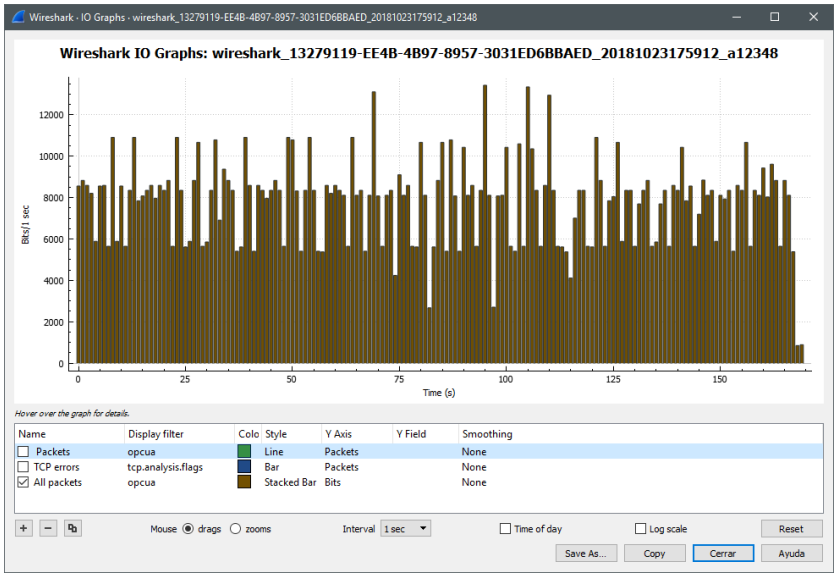
Figura 59. Captura de Paquete de datos – Prueba 2.
Fuente: Elaborado por el Investigador.

En conclusión, la Figura 59 contiene la misma estructura de captura de paquetes de datos OPC-UA que en la Figura 54 mostrada en el apartado 4.6.3. al aplicar el *filtro OpcUa*. Debido a esto, no es necesario repetir en detalle las características indicadas por *WireShark* para la sección actual y nos centraremos más bien, en los resultados estadísticos de comunicación a través de la red obtenidos.

4.6.7 Análisis de Paquetes de Datos.

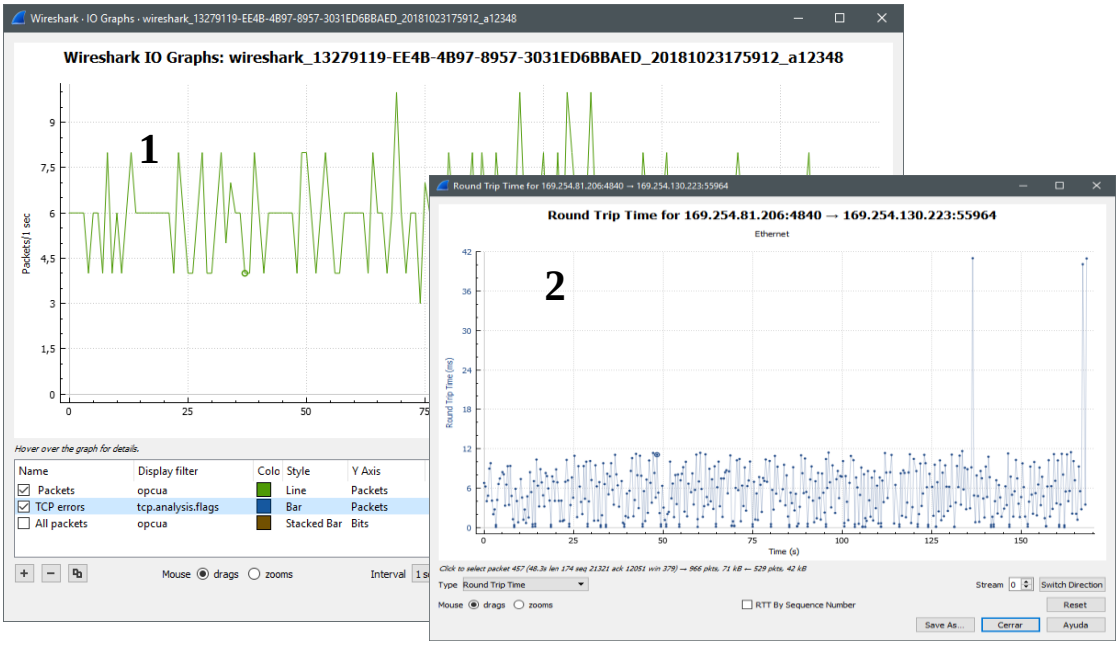
De igual manera, dentro de la captura de paquetes se detalla en la *sección 4.6.6*, tres tipos de conversación. La primera con la publicación de variables, la segunda conversación que con la petición de lectura del estado de variables y la tercera que genera la apertura a escritura de variables, que en términos generales representan la petición del proceso de *Solicitud/Respuesta* para cada paquete de información intercambiado. Todo esto con un enlace de muestreo de igual manera a 100ms, que en resumen genera el tráfico de paquetes en conexión del gráfico resultante de la Tabla 11.

Tabla 11. Ancho de banda OPC-UA.
Fuente: Elaborado por el Investigador.

Uso de Banda de Ancho

Resultados
▪ Al aplicar el filtro de datos *opcua* en la opción *Display Filter*, muestra un ancho de banda que alcanza un máximo de 13000 bits/s y un mínimo de 3000 bits/s, todo esto con un promedio de 8000 bits/s para todo el grupo de paquetes capturado en función del tiempo.

De la misma manera se analiza la cantidad de paquetes procesados y los posibles errores que se puedan generar dentro del proceso comunicación OPC-UA. Las estadísticas graficas se describen gráficamente en la Tabla 12.

Tabla 12. Procesamiento de Paquetes OPC-UA.
Fuente: Elaborado por el Investigador.

Procesamiento de Paquetes de Datos

Resultados
▪ La gráfica 1 se detalla la cantidad de paquetes procesados en una petición de solicitud/respuesta, que detalla un máximo de 10 Packets/s y un mínimo de 2 Packets/s, que a su vez generan un promedio de 6 Packets/s por cada intervalo de tiempo mencionado. De la misma manera se intenta graficar la estadística de errores en el protocolo de comunicación con la perdida de paquetes de comunicación, pero sin ningún resultado visible. ▪ Por otro lado, la gráfica 2 mide el delta de tiempo entre la captura de un paquete TCP y el ACK retardado correspondiente para cada paquete, generando un retardo en el proceso de Solicitud/Respuesta hasta los 11 ms. y con retrasos mayores en los paquetes: 1127, 1634 y 1636 que se promedian en los 40ms para la prueba realizada.

4.6.8 Resumen y Discusión de Resultados.

Tabla 13. Resultados de conexión OPC-UA.
Fuente: Elaborado por el Investigador.

Pruebas de Conexión y Análisis de Red

Objetivo:

- Comparar la eficiencia de comunicación Cliente/Servidor bajo el protocolo OPC-UA generado en la aplicación final del proyecto de investigación, frente a proyectos SDK de clientes desarrollados por empresas privadas.

Resultados:

Escenario de Prueba	Modo y Política de Seguridad	Autenticación de Usuario	Tiempo de muestreo (ms)	Ancho de banda total (Petición/Respuesta de datos y estado servidor)	Procesamiento Promedio de Paquetes de Información (Packets/s)	Round tripe Time (ms)
Prueba 1	Ninguno	Ninguno	100	16 kbits/s	16	1.4
Prueba 2	Ninguno	Ninguno	100	8 kbits/s	6	11

Conclusiones:

- En la tabla de resultados, se puede observar los dos escenarios de prueba en los cuales no se tiene firma digital ni cifrado en los datos. En la primera prueba se muestra una mayor segmentación en los paquetes bajo el monitoreo de Solicitud/Respuesta, es decir que, existe mayor número de respuestas por parte del servidor, lo que se concluye en un aumento de uso de banda de ancho, que en este caso se duplica en comparación a la segunda prueba. De igual manera se muestra la cantidad de Packets/s procesados, en donde la prueba 1 genera un mayor número de paquetes monitoreados por cada inérvalo de muestreo, resultado debido al mayor uso de ancho de banda ya mencionado.

- Finalmente, se describe el tiempo de vuelo o el delta de tiempo entre la captura de un paquete TCP y el ACK retardado correspondiente para cada paquete, que para este caso muestra un valor mucho más rápido en la prueba 1, esto se resume a que el monitoreo de Solicitud/Respuesta se genera de una manera más eficiente.

- El ancho de banda utilizado por nuestro cliente GUI desarrollado en Python para la prueba 1, se ejecuta de una manera muy aceptable en comparación con el software SDK UAExpert creado por la empresa Unified Automation, es decir que su conexión Servidor/cliente generan una comunicación continua y estable en el tiempo si una sobrecarga de uso de banda de ancho a un tiempo de muestreo aceptable.

- Por otro lado, cabe recalcar que en el escenario de la prueba 1 se generan procesos de control basado en sistemas PID y un procesamiento de trazado grafico en tiempo real, con lo cual se justifica el ancho de banda utilizado, que a diferencia del cliente utilizado en la prueba 2, se lo ejecuta de manera que permita visualizar de manera pasiva las variables del servidor como ejemplo de prueba de conexión.

4.6.9 Desempeño de la Raspberry Pi Como Servidor OPC-UA

Finalmente, se presenta un análisis estadístico de manera resumida de los recursos hardware y software de la Raspberry Pi monitoreados mientras el prototipo está en funcionamiento a plena carga durante 1 hora. Fue interesante ver cómo el hardware maneja el sistema a plena capacidad, en donde los datos fueron leídos/escritos lo más rápido posible. El porcentaje de potencia de procesamiento que la Raspberry Pi necesita manejar bajo carga completa, se observa en el gráfico de la Figura 60. Esta grafica muestra el uso promedio de la CPU que está en un 3.5%, con una frecuencia de reloj a 600Mhz y un voltaje de consumo sin periféricos extra de 1,2v, mínimo si se desea calificarlo de esa manera, por lo tanto, el hardware *Raspberry Pi 3 Model B* es totalmente capaz de ejecutar el servidor OPC-UA detallado en la sección 4.3.1.

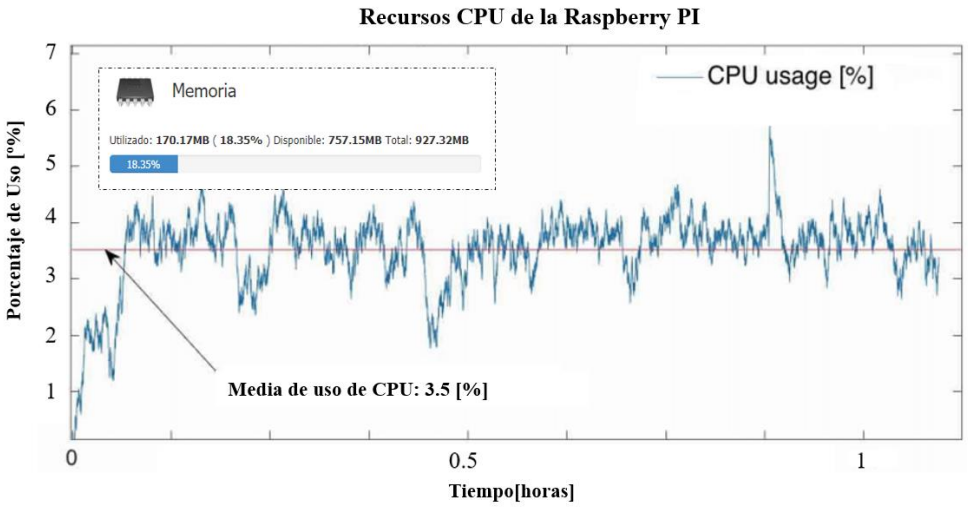


Figura 60. Recurso CPU Raspberry PI.
Fuente: Elaborado por el Investigador.

De la misma forma, se evalúa la cantidad de memoria interna usada para ejecutar la aplicación del servidor, recordemos que la *Raspberry Pi 3 Model B* cuenta con una memoria RAM de 1GB de manera teórica en sus especificaciones de fábrica, pero que en realidad tan solo cuenta con 927.32 MB de capacidad y que para el caso de estudio fue necesario solo el 18.35% (+/–10%) de su capacidad total, es decir que se usan alrededor de 170 MB de memoria interna para funcionar de manera óptima.

Por otro lado, es fundamental determinar a qué temperatura trabaja la Raspberry Pi, recordemos que esta SBC es un computador movil de gama baja. Por lo tanto, no cuenta con un adecuado sistema de enfriamiento como otras CPU de escritorio/laptop. Si su temperatura sube por encima de los 80 °C, se puede notar un pequeño termómetro en el escritorio Raspbian. Eso nos indica que la Raspberry Pi se está calentando demasiado. A medida que se aumenta la temperatura del núcleo central la CPU, se comienzan a acelerar sus procesos internos y se reduce la frecuencia de su reloj para aliviar la temperatura, lo que en pocas palabras significa que su rendimiento disminuirá. Debido a esto, fue necesario realizar un análisis de las condiciones de temperatura a la que trabaja con el servidor ejecutándose en tiempo real, las condiciones a las que se sometió fueron las más apropiadas, clima normal de medioambiente a 18 ºC, sin ningún elemento extra como disipadores de calor y una operabilidad continua de alrededor de 1 hora.

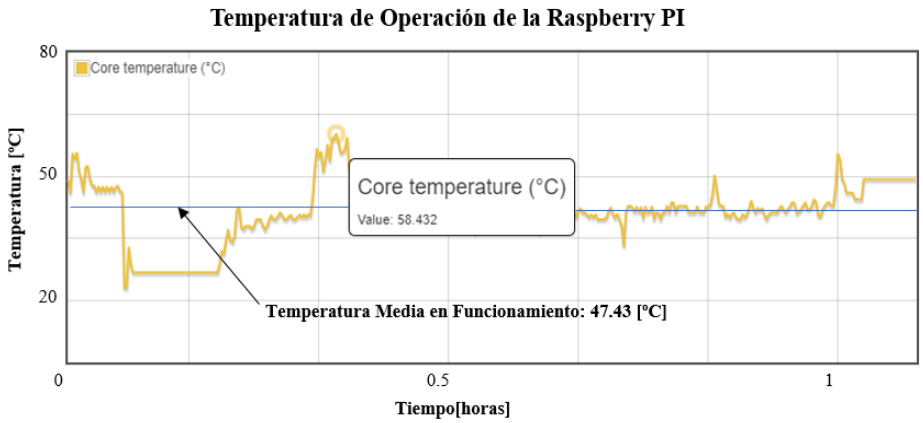


Figura 61. Temperatura de Núcleo de la Raspberry PI.
Fuente: Elaborado por el Investigador.

La Figura 61, muestra la evolución de la temperatura disipada por el núcleo de la Raspberry Pi durante el tiempo de ejecución planificada, en donde se puede observar que no se supera los 60 ºC que, en conclusión, menciona que no se superan los límites de seguridad, pero a su vez, es necesario una ayuda externa para poder disipar los picos de temperatura provocados en la placa. Y si se desea realizar pruebas adicionales con un mayor lapso de tiempo dentro de un sistema con las mismas características de esta investigación, se recomendaría implementar disipadores a los procesadores principales para evitar futuras complicaciones.

CAPÍTULO V
CONCLUSIONES Y RECOMENDACIONES

5.1 Conclusiones

- La implementación de la SBC Raspberry Pi como servidor OPC-UA presenta un sólido modelo estructural capaz de integrar y facilitar, la operatividad de los sistemas físico de nivel y temperatura dentro del módulo Festo MPS® PA Compact Workstation presentado como caso de estudio, integrando con facilidad conceptos de interoperabilidad, sincronización y uso multiplataforma sin un excesivo uso de recursos software y hardware. En donde se logra implementar diferentes tipos de control personalizables, convirtiéndola en una alternativa flexible y de bajo costo frente a diferentes dispositivos habituales utilizados en el control de procesos a nivel industrial. Esto da una gran apertura en el desarrollo de prototipos de sistemas CPS más robustos y multidisciplinarios, que, en conclusión, sustentan de manera objetiva las condiciones de automatización dentro de la Industria 4.0.

- Durante todo el proceso de desarrollo de la investigación, se centra en un espacio escalable de software con la finalidad de que la aplicación OPC-UA generada fuera portable para dispositivos integrados de características similares a la SBC utilizada. La implementación del servidor OPC-UA se basa en una API de red completamente asíncrona como una capa de abstracción del sistema operativo, como la presentada por el proyecto open62541. La naturaleza asíncrona del lenguaje de programación “pure C”, destaca un punto fuerte al evaluar la velocidad de comunicación dentro de la aplicación final. Además de permitir la ejecución de

un modelo de información continuo que define un soporte básico, pero completo, para la creación de diferentes tipos de nodos y métodos de adquisición de datos físicos, sin ningún contratiempo adicional, incluyendo un soporte adicional en el control de variables y objetos UA con un bajo consumo de recursos software. Por otro lado, la implementación del protocolo binario del proyecto Python OPC-UA presenta una interfaz dinámica para enviar y recibir todas las estructuras definidas dentro del servidor. Su arquitectura de código destaca clases de alto nivel que permiten desarrollar al cliente sin una sobrecarga en el uso de líneas código. En donde las principales funciones implementadas son: Conexión al Servidor, Apertura de Canal seguro de Comunicación, Apertura de Sesión, Obtención nodos por ruta, Suscripciones por eventos, Procesamiento de variables físicas para controles de lazo abierto y cerrado bajo condiciones experimentales y un proceso de desconexión segura.

- La adquisición de datos provenientes de los sensores físicos, son muestreados con un tiempo de 100ms de manera intrínseca en la programación del cliente, pero a pesar de esto, las gráficas en tiempo de real varían su intervalo de muestreo entre los 90ms y 120ms, esto debido explícitamente al uso de las librerías proporcionadas por *Matplotlib y tkinter*. En experimentaciones realizadas en paralelo a esta investigación, se pudo constatar un retraso en el trazado en tiempo real en función de las gráficas que se puedas implementar dentro de una GUI, es decir que mientras más graficas en tiempo real se deseen implementar más lento se tornara el trazado de datos. El control PID implementado genera una respuesta de control muy satisfactoria para diferentes valores de Set Point con un tiempo de estabilización alrededor de los 30 seg. La respuesta de control en general de la GUI se califica como aceptable, en las pruebas realizadas muestra una evolución de trazado de datos en tiempo real de manera fluida y sin cortes, al igual que el control de salida hacia en calentador que presenta una respuesta de manera rápida. Que en otras palabras se resume en puntos a favor en base a su operabilidad y control si se desea analizar la velocidad de respuesta del sistema en general.

5.2 Recomendaciones

- Los proyectos Open62541 y Python OPC-UA, son plataformas de creación de aplicaciones que obligatoriamente necesitan ser analizadas y puestas en experimentación las veces que sean necesarias, si bien los conceptos OPC-UA no son nuevos, las *API open source* no lo son de la misma manera y se siguen optimizando año tras año, versión tras versión. Debido a esto, tienen una serie de restricciones de aplicación para el control y manejo de datos, además de problemas de compatibilidad y ejecución en sus diferentes versiones, para esto, se recomienda generar una investigación previa de manera exhaustiva en función de los objetivos a satisfacer ya sea mediante el uso de las API implementadas en esta investigación, o también con la aplicación de otros proyectos OPC-UA mencionados en secciones anteriores, con la finalidad de que el investigador pueda suplir sus necesidades de ejecución para investigaciones futuras.

- Para solucionar aspectos de velocidad en trazado de datos dentro de la GUI, se puede implementar una programación más depurada implementando características Multiproceso en el código final, pero con una mayor complejidad en la estructura del programa, o a su vez, se podría elegir otras herramientas de trazado, como por ejemplo *PyQtGraph,* que presenta características igualmente dinámicas para el trazado y análisis de datos.

- Por otro lado, para evitar posibles complicaciones de operación causados por un sobrecalentamiento excesivo en la Raspberry Pi, se recomienda implementar un adecuado sistema de enfriamiento con disipadores de calor ubicados en el chip procesador, en especial para casos de pruebas en donde se requiera un tramo amplio de tiempo de uso y si es necesario se debe incorporar un blindaje extra de protección del hardware, con la finalidad de evitar posibles daños físicos, esto si se desea experimentar con procesos de control en entornos industriales no controlados. Además de un necesario dimensionamiento para las etapas de potencia con el fin de evitar cortos eléctricos dentro del sistema.

BIBLIOGRAFÍA

[1] N. Jazdi, "Cyber physical systems in the context of Industry 4.0," in *2014 IEEE International Conference on Automation, Quality and Testing, Robotics*, 2014, pp. 1–4.

[2] M. Riedl, H. Zipper, M. Meier, and C. Diedrich, "Cyber-physical systems alter automation architectures," *Annu. Rev. Control*, vol. 38, no. 1, pp. 123–133, Jan. 2014.

[3] Valeriy Vyatkin, "Software engineering in industrial automation: State-of-the-art review," *ieeexplore.ieee.org*, vol. 9, p. 16, 2013.

[4] M. H. Schwarz and J. Borcsok, "A survey on OPC and OPC-UA: About the standard, developments and investigations," in *2013 XXIV International Conference on Information, Communication and Automation Technologies (ICAT)*, 2013, pp. 1–6.

[5] Opiron-Electronics, "OPC UA, 5 razones para elegirlo como habilitador de la Industria 4.0," 2017. [Online]. Available: https://www.opiron.com/2017/01/05/5-razones-por-las-que-elegir-opc-ua/#3_Despliega_arquitecturas_simplificadas_No_mas_tunnelling. [Accessed: 08-Jan-2019].

[6] E. P. Patiño Ramón and J. C. Solano Minchalo, "Diseño e implementación de un prototipo de supervisión de un sisema de control industrial utilizando plataformas empotradas de bajo costo y controladores lógicos programables PLCs," 2016.

[7] itahora, "Ecuador, en el umbral hacia la industria 4.0 | ITAHORA," 2018. [Online]. Available: https://www.itahora.com/actualidad/ecuador-en-el-umbral-hacia-la-industria-4-0/. [Accessed: 08-Jan-2019].

[8] El Universo, "La era del robot se instala a paso lento en Ecuador | Informes | Noticias | El Universo," 2018. [Online]. Available: https://www.eluniverso.com/noticias/2018/08/17/nota/6909632/era-robot-se-instala-paso-lento-ecuador. [Accessed: 08-Jan-2019].

[9] La Revista Siemens, "Weblet Importer," 2018. [Online]. Available: https://www.siemens.com/la-revista/es/portada/industria/revolucion-industrial-40-en-ecuador.html. [Accessed: 08-Jan-2019].

[10] Matrikon OPC, "¿Qué es Internet de las cosas (IoT), Industry 4.0 y OPC UA?," 2018. [Online]. Available: https://www.matrikonopc.com/downloads/1308/webcasts/index.aspx. [Accessed: 24-Mar-2018].

[11] T. Rodrigues Alves, M. Buratto, F. M. de Souza, and T. V. Rodrigues, "OpenPLC: An open source alternative to automation," in *IEEE Global Humanitarian Technology Conference (GHTC 2014)*, 2014, pp. 585–589.

[12] S. Rúnarsson, "Open Source Hardware and Software Alternative to Industrial PLC," *130*, 2016.

[13] B. R. VILLARROYA ALFONSO, "Servidor OPC-UA sobre Beaglebone Black," Oct. 2015.

[14] H. Koziolek, A. Burger, and J. Doppelhamer, "Self-Commissioning Industrial IoT-Systems in Process Automation: A Reference Architecture," in *2018 IEEE International Conference on Software Architecture (ICSA)*, 2018, pp. 196–19609.

[15] J. Dias, J. Barbosa, and P. Leitao, “Deployment of industrial agents in heterogeneous automation environments,” in *2015 IEEE 13th International Conference on Industrial Informatics (INDIN)*, 2015, pp. 1330–1335.

[16] M. G.-C. de C. y T. ESPE and undefined 2015, “Desarrollo de Comunicación OPC-UA en Sistemas CPPS de Bajo Costo,” *journal.espe.edu.ec*.

[17] M. V. Garcia, E. Irisarri, F. Perez, E. Estevez, and M. Marcos, “OPC-UA communications integration using a CPPS architecture,” in *2016 IEEE Ecuador Technical Chapters Meeting (ETCM)*, 2016, pp. 1–6.

[18] S. Azaiez, M. Boc, … L. C.-P. C., and undefined 2016, “Towards flexibility in future industrial manufacturing: a global framework for self-organization of production cells,” *researchgate.net*.

[19] J. Imtiaz, J. J.-I. I. (INDIN), undefined 2013, and undefined 2013, “Scalability of OPC-UA down to the chip level enables ‘Internet of Things,’” *ieeexplore.ieee.org*.

[20] open62541.org, “open62541: una implementación de código abierto de OPC UA,” 2018. [Online]. Available: https://open62541.org/. [Accessed: 01-May-2018].

[21] S. Hensel, M. Graube, L. Urbas, T. Heinzerling, and M. Oppelt, “Co-simulation with OPC UA,” in *2016 IEEE 14th International Conference on Industrial Informatics (INDIN)*, 2016, pp. 20–25.

[22] M. Muller, E. Wings, and L. Bergmann, “Developing open source cyber-physical systems for service-oriented architectures using OPC UA,” in *2017 IEEE 15th International Conference on Industrial Informatics (INDIN)*, 2017, pp. 83–88.

[23] C. Catalán Cantero and A. Blesa Gascón, “Enseñanza de sistemas empotrados: de Arduino a Raspberry Pi,” *Actas las XXII JENUI*, pp. 351–354, Jul. 2016.

[24] L. del Val Román, “Industria 4.0: la transformación digital de la industria | Revista web de la Facultad de Ingeniería de la Universidad de Deusto,” *CONFERENCIA DE DIRECTORES Y DECANOS DE INGENIERÍA INFORMÁTICA*, España, p. 10, 2016.

[25] P. González, I. Calvo, I. Etxeberria, and J. M. López, “Hacia un framework basado en ROS para la implementación de Sistemas Ciberfísicos,” p. 8, 2015.

[26] infaimon, “Tecnología 4.0: aplicaciones y beneficios,” 2018. [Online]. Available: https://blog.infaimon.com/tecnologia-4-0/. [Accessed: 08-Jan-2019].

[27] ticnegocios, “Sistemas Ciberfísicos: la respuesta a las necesidades de la industria | TicNegocios.es,” 2017. [Online]. Available: https://ticnegocios.camaravalencia.com/servicios/tendencias/sistemas-ciberfisicos-la-respuesta-a-las-necesidades-de-la-sociedad-y-la-industria/. [Accessed: 22-May-2018].

[28] A. A. G. Craig Resnick, “Embedded OPC UA: Enhanced Potential for Interoperability | Automation World,” 2012. [Online]. Available: https://www.automationworld.com/article/technologies/communication-protocols-standards/embedded-opc-ua-enhanced-potential. [Accessed: 26-Sep-2018].

[29] L. Power, D. Kominek, and P. Eng, “Keys To Developing an Embedded UA Server,” Canada 2013.

[30] OPC Foundation, “¿Qué es OPC? - Fundación OPC,” 2018. [Online]. Available: https://opcfoundation.org/about/what-is-opc/. [Accessed: 24-Mar-2018].

[31] Novotek, “OPC and OPC UA explained,” 2018. [Online]. Available: https://www.novotek.com/en/solutions/kepware-communication-platform/opc-and-opc-ua-explained. [Accessed: 03-Apr-2018].

[32] L. U and I. F, “Historia - Fundación OPC,” *4th rev. Edición. Berlín*, 2018. [Online]. Available: https://opcfoundation.org/about/opc-foundation/history/. [Accessed: 24-Mar-2018].

[33] Matrikon, “OPC UA (Arquitectura Unificada),” 2018. [Online]. Available: https://www.matrikonopc.es/opc-ua/index.aspx. [Accessed: 30-Mar-2018].

[34] ControlEngineering, “Haciendo una transición suave de OPC Classic a OPC UA,” 2014. [Online]. Available: http://www.controlengeurope.com/article/80950/Making-a-smooth-transition-from-OPC-Classic-to-OPC-UA.aspx. [Accessed: 24-Mar-2018].

[35] BR-Automation, “¿Qué es OPC UA?,” 2015. [Online]. Available: https://www.br-automation.com/es/tecnologias/opc-ua/. [Accessed: 24-Mar-2018].

[36] T. J. Burke, “OPC Unified Architecture Interoperability for Industrie 4.0 and the Internet of Things 4.0 Industrie IoT M2M,” 2017.

[37] Inmation Wiki, “OPC Connectivity,” 2016. [Online]. Available: https://inmation.com/wiki/index.php?title=Sysdoc/OPC_Connectivity. [Accessed: 24-Mar-2018].

[38] Unified Automation, “Paquete de UA Server SDK C ++: Fundamentos de OPC UA,” 2016. [Online]. Available: http://documentation.unified-automation.com/uasdkcpp/1.4.0/html/L1OpcUaFundamentals.html. [Accessed: 26-Sep-2018].

[39] V. Beal, “What is API - Application Program Interface? Webopedia Definition,” 2018. [Online]. Available: https://www.webopedia.com/TERM/A/API.html. [Accessed: 27-Sep-2018].

[40] M. Buurmeijer, “List of Open Source OPCUA Implementations,” 2018. [Online]. Available: https://github.com/open62541/open62541/wiki/List-of-Open-Source-OPC-UA-Implementations. [Accessed: 27-Sep-2018].

[41] Cmake.org, “CMake,” 2018. [Online]. Available: https://cmake.org/. [Accessed: 26-Sep-2018].

[42] SWAROOP, “What is Serial Communication and How it works? [Explained],” 2018. [Online]. Available: https://www.codrey.com/embedded-systems/serial-communication-basics/. [Accessed: 27-Sep-2018].

[43] Circuit Basics, “BASICS OF THE I2C COMMUNICATION PROTOCOL,” 2016. [Online]. Available: http://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/. [Accessed: 26-Sep-2018].

[44] C. Towm, “I2C - learn.sparkfun.com,” 2017. [Online]. Available: https://learn.sparkfun.com/tutorials/i2c. [Accessed: 26-Sep-2018].

[45] Electrónica DIY, “Basics of the SPI Communication Protocol,” 2016. [Online]. Available: http://www.circuitbasics.com/basics-of-the-spi-communication-protocol/. [Accessed: 27-Sep-2018].

[46] E. Gomez, “Conversor analógico a digital - ADC - Rincón Ingenieril,” 2017. [Online]. Available: https://www.rinconingenieril.es/conversor-analogico-a-digital-adc/. [Accessed: 26-Sep-2018].

[47] Arrow, “Computadoras de placa única - SBC,” 2018. [Online]. Available: https://www.arrow.com/es-mx/categories/embedded-controllers-and-systems/embedded-systems/single-board-computers---sbcs. [Accessed: 27-Sep-2018].

[48] Techopedia Inc., “What is a Single-Board Computer (SBC)? - Definition from Techopedia,” 2018. [Online]. Available: https://www.techopedia.com/definition/9266/single-board-computer-sbc. [Accessed: 26-Sep-2018].

[49] Raspberrypi.org, “Raspberry Pi - Github,” 2018. [Online]. Available: https://github.com/raspberrypi. [Accessed: 27-Sep-2018].

[50] Raspberrypi.org, “Raspberry Pi — Teach, Learn, and Make with Raspberry Pi,” 2018. [Online]. Available: https://www.raspberrypi.org/. [Accessed: 27-Sep-2018].

[51] PC-Componentes, “Raspberry Pi 3 Modelo B,” 2016. [Online]. Available: https://www.pccomponentes.com/raspberry-pi-3-modelo-b. [Accessed: 27-Sep-2018].

[52] Back Andrew, “Raspberry Pi 3 Modelo B vs. 3 Modelo B +,” 2018. [Online]. Available: https://core-electronics.com.au/tutorials/raspberry-pi-3-model-b-plus-performance-vs-3-model-b.html. [Accessed: 19-Nov-2018].

[53] beagleboard.org, “BeagleBoard.org - black,” 2018. [Online]. Available: https://beagleboard.org/black. [Accessed: 27-Sep-2018].

[54] Texas Instruments Incorporated, “BEAGLEBK BeagleBone Black Development Board | TI.com,” 2018. [Online]. Available: http://www.ti.com/tool/BEAGLEBK. [Accessed: 27-Sep-2018].

[55] Mayank, “Introduction to Single Board Computing » maxEmbedded,” 2013. [Online]. Available: http://maxembedded.com/2013/07/introduction-to-single-board-computing/. [Accessed: 11-May-2018].

[56] linksprite, “pcDuino4 Nano - LinkSprite,” 2018. [Online]. Available: http://www.linksprite.com/pcduino4-nano/. [Accessed: 27-Sep-2018].

[57] ODRIOD, “ODROID | Hardkernel,” 2018. [Online]. Available: https://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438&tab_idx=1. [Accessed: 27-Sep-2018].

[58] Texas Instruments, “ADS111x Ultra-Small, Low-Power, I 2C-Compatible, 860-SPS, 16-Bit ADCs With Internal Reference, Oscillator, and Programmable Comparator,” 2016.

[59] Festo, “MPS® PA Compact Workstation,” 2018. [Online]. Available: https://www.festo-didactic.com/int-en/learning-systems/process-automation/compact-workstation/mps-pa-compact-workstation-with-level,flow-rate,pressure-and-temperature-controlled-systems.

ANEXOS

ANEXO 1: Servidor OPC-UA "Obtención de la librería y Compilación"

En primer lugar, se instala las dependencias necesarias para compilar la librería.

```
$ sudo apt-get install git build-essential gcc pkg-config cmake
python python-six
```

También puede ser de utilidad algunas dependencias necesarias para otras funcionalidades:

```
$ sudo apt-get install cmake-curses-gui# Necesaria para la interfaz
gráfica del ccmake
$ sudo apt-get install libmbedtls-dev # Para el cifrado
$ sudo apt-get install liburcu-dev # Para el uso de threads
$ sudo apt-get install check # Para los tests unitarios
$ sudo apt-get install python-sphinx graphviz # Documentación
$ sudo apt-get install python-sphinx-rtd-theme
```

A continuación, se clona dentro de un directorio cualquiera la librería del proyecto open62541 desde su página de GitHub:

```
$ git clone https://github.com/open62541/open62541.git
```

Para compilar la librería seguimos el procedimiento habitual con los proyectos en los que se emplea Cmake:

```
$ cd open62541
$ mkdir build
$ cd build
$ cmake ..
$ ccmake ..
```

Posterior a esto, se debe activar la opción UA_ENABLE_AMALGAMATION que permite obtener toda la librería en dos únicos ficheros open62541.c y open62541.h, además de las opciones que se muestran en la Figura 23 de la sección 4.3.3. Finalmente, se guarda esta configuración y se procede a compilar la librería:

```
$ make
```

ANEXO 2: Servidor_OPCUA.c

```
#include <sys/stat.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <inttypes.h>
#include <linux/i2c-dev.h>
#include <stdio.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <inttypes.h>
#include <signal.h>
#include <stdlib.h>
#include <unistd.h>
#include <time.h>
#include <bcm2835.h>
#include <stdio.h>
#include "open62541.h"


#define PIN RPI_GPIO_P1_12
#define PIN1 RPI_GPIO_P1_11
#define PWM_CHANNEL 0
#define RANGE 1024
#define ADS_address 0x48

UA_Boolean running = true;
UA_Double CountNodeValue = 0;
UA_Double CountNodeValue1 = 0;
UA_Logger logger = UA_Log_Stdout;

void signalHandler(int sig) {
   running = false;
}


/***********ETAPA DE COMUNICACION CON VARIABLES
               FISICA EXTERNAS DE ENTRADA*****************************/


static void SensorNivel(void *handle, const UA_NodeId nodeid, const
UA_Variant *data, const UA_NumericRange *range)

{

  int I2CFile;
  uint8_t writeBuf[3];
  uint8_t readBuf[2];
  uint32_t nivel;

  I2CFile = open("/dev/i2c-1", O_RDWR);
  ioctl(I2CFile, I2C_SLAVE, ADS_address);
  writeBuf[0] = 1;
  writeBuf[1] = 0xC3;
  writeBuf[2] = 0x03;
  readBuf[0]= 0;
  readBuf[1]= 0;
```

```
  write(I2CFile, writeBuf, 3);

     while ((readBuf[0] & 0x80) == 0)
       {
          read(I2CFile, readBuf, 2);
       }
      writeBuf[0] = 0;
      write(I2CFile, writeBuf, 1);
      read(I2CFile, readBuf, 2);
      nivel = readBuf[0] << 8 | readBuf[1];

                UA_Server *server = (UA_Server*)handle;
                UA_Variant value;
                UA_Double con = nivel*5/32767.0;

                if(con >= 8)
                {
                CountNodeValue = 0;
                UA_Variant_setScalar(&value, &CountNodeValue,
                                              &UA_TYPES[UA_TYPES_DOUBLE]);
                UA_Server_writeValue(server, nodeid, value);

                }
                else
                {

                CountNodeValue = con;
                UA_Variant_setScalar(&value, &CountNodeValue,
                                              &UA_TYPES[UA_TYPES_DOUBLE]);
                UA_Server_writeValue(server, nodeid, value);
                }

 close(I2CFile);


}



static void SensorTemp(void *handle, const UA_NodeId nodeid, const
UA_Variant *data1,
                   const UA_NumericRange *range)

{

  int I2CFile1;
  uint8_t writeBuf1[3];
  uint8_t readBuf1[2];
  uint32_t temp;

  I2CFile1 = open("/dev/i2c-1", O_RDWR);
  ioctl(I2CFile1, I2C_SLAVE, ADS_address);
  writeBuf1[0] = 1;
  writeBuf1[1] = 0xB3;
  writeBuf1[2] = 0x03;
  readBuf1[0]= 0;
  readBuf1[1]= 0;
  write(I2CFile1, writeBuf1, 3);
```

```
      while ((readBuf1[0] & 0x80) == 0)
       {
           read(I2CFile1, readBuf1, 2);
       }
      writeBuf1[0] = 0;
      write(I2CFile1, writeBuf1, 1);
      read(I2CFile1, readBuf1, 2);
      temp = readBuf1[0] << 8 | readBuf1[1];


                    UA_Server *server = (UA_Server*)handle;
                    UA_Variant value1;
                    UA_Double con1 = temp*5/32767.0;

          if(con1 >= 9)
                  {

                    CountNodeValue1 = 0;
                    UA_Variant_setScalar(&value1, &CountNodeValue1,
                                         &UA_TYPES[UA_TYPES_DOUBLE]);
                    UA_Server_writeValue(server, nodeid, value1);

                  }
          else
                  {
                    CountNodeValue1 = con1;
                    UA_Variant_setScalar(&value1, &CountNodeValue1,
                                         &UA_TYPES[UA_TYPES_DOUBLE]);
                    UA_Server_writeValue(server, nodeid, value1);
                  }

   close(I2CFile1);


}

/**************FIN DE ETAPA DE ADQUISICION DE DATOS********************/
/**********************ETAPA DE CONTROL PWM*****************************/

static UA_StatusCode
onRead(void *handle, const UA_NodeId nodeid, UA_Boolean sourceTimeStamp,
             const UA_NumericRange *range, UA_DataValue *dataValue)

{

    UA_Variant_setScalarCopy(&dataValue->value, (UA_Float*)handle,
                                           &UA_TYPES[UA_TYPES_FLOAT]);

if (!bcm2835_init())
        return 1;
    bcm2835_gpio_fsel(PIN, BCM2835_GPIO_FSEL_ALT5);
    bcm2835_pwm_set_clock(BCM2835_PWM_CLOCK_DIVIDER_64);
    bcm2835_pwm_set_mode(PWM_CHANNEL, 1, 1);
    bcm2835_pwm_set_range(PWM_CHANNEL, RANGE);

    UA_Float dato =*(UA_Float*)handle;
    bcm2835_pwm_set_data(PWM_CHANNEL, dato);
    bcm2835_close();
    delay(0.1);

}
```

```
static UA_StatusCode
onWrite(void *handle, const UA_NodeId nodeid,
             const UA_Variant *data, const UA_NumericRange *range) {

    if(UA_Variant_isScalar(data) && data->type == &UA_TYPES[UA_TYPES_FLOAT]
       && data->data){
                        *(UA_Float*)handle = *(UA_Float*)data->data;
    }
     UA_LOG_INFO(logger, UA_LOGCATEGORY_USERLAND, "Valor Escrito %f",
                  *(UA_Float*)handle);
}


/********************ON/OFF CALENTADOR DE AGUA************************/

static UA_StatusCode
onRead1(void *handle, const UA_NodeId nodeid, UA_Boolean sourceTimeStamp,
            const UA_NumericRange *range, UA_DataValue *dataValue)

{

  UA_Variant_setScalarCopy(&dataValue->value, (UA_Double*)handle,
&UA_TYPES[UA_TYPES_DOUBLE]);


  if (!bcm2835_init())
        return 1;
    bcm2835_gpio_fsel(PIN1, BCM2835_GPIO_FSEL_OUTP);
    UA_Double cal =*(UA_Double*)handle;

    if(cal == 1)
        bcm2835_gpio_write(PIN1, HIGH);
    else
       if (cal == 0)
        bcm2835_gpio_write(PIN1, LOW);

  bcm2835_close();
  delay(0.001);

}

static UA_StatusCode
onWrite1(void *handle, const UA_NodeId nodeid,
             const UA_Variant *data, const UA_NumericRange *range) {

    if(UA_Variant_isScalar(data) && data->type == &UA_TYPES[UA_TYPES_DOUBLE]
&& data->data){
        *(UA_Double*)handle = *(UA_Double*)data->data;
    }
//    UA_LOG_INFO(logger, UA_LOGCATEGORY_USERLAND, "Valor Escrito %d",
                  *(UA_Float*)handle);
}
```

```
/*******************CONTROL PRINCIPAL*************************/

int main(void)

{
/********CONFIGURACION DE PUERTO DE COMUNICACION***********/

   signal(SIGINT, signalHandler);

    UA_ServerConfig config = UA_ServerConfig_standard;
    UA_ServerNetworkLayer nl =
          UA_ServerNetworkLayerTCP(UA_ConnectionConfig_standard, 4840);
    config.networkLayers = &nl;
    config.networkLayersSize = 1;
    UA_Server *server = UA_Server_new(config);

/*****************CREACION DE NODOS UA************************/

 UA_VariableAttributes attr2;
        UA_VariableAttributes_init(&attr2);
        UA_NodeId CountNodeId = UA_NODEID_STRING(1,"Sensor_Nivel");
        UA_Variant_setScalar(&attr2.value, &CountNodeValue,
                                        &UA_TYPES[UA_TYPES_DOUBLE]);
        attr2.writeMask = 0xFF;
        attr2.description = UA_LOCALIZEDTEXT("en_US","Sensor_Nivel");
        attr2.displayName = UA_LOCALIZEDTEXT("en_US","Sensor_Nivel");
        UA_QualifiedName browseName2 = UA_QUALIFIEDNAME(1, "Sensor_Nivel");


          UA_NodeId parentNodeId2 = UA_NODEID_NUMERIC(0,
                                              UA_NS0ID_OBJECTSFOLDER);
          UA_NodeId parentReferenceNodeId2 = UA_NODEID_NUMERIC(0,
                                                 UA_NS0ID_ORGANIZES);


          UA_Server_addVariableNode(server, CountNodeId, parentNodeId2,
                               parentReferenceNodeId2, browseName2,
                               UA_NODEID_NULL, attr2, NULL, NULL);
          UA_ValueCallback callbackCount = {server, SensorNivel, NULL};
          UA_Server_setVariableNode_valueCallback(server, CountNodeId,
                                                        callbackCount);


 UA_VariableAttributes attr3;
        UA_VariableAttributes_init(&attr3);
        UA_NodeId CountNodeId1 = UA_NODEID_STRING(1,"Sensor_Temp");
        UA_Variant_setScalar(&attr3.value, &CountNodeValue1,
                                       &UA_TYPES[UA_TYPES_FLOAT]);
        attr3.writeMask = 0xFF;
        attr3.description = UA_LOCALIZEDTEXT("en_US","SensorTemp");
        attr3.displayName = UA_LOCALIZEDTEXT("en_US","Sensor_Temp");
        UA_QualifiedName browseName3 = UA_QUALIFIEDNAME(1, "Sensor_Temp");


          UA_NodeId parentNodeId3 = UA_NODEID_NUMERIC(0,
                                                 UA_NS0ID_OBJECTSFOLDER);
          UA_NodeId parentReferenceNodeId3 = UA_NODEID_NUMERIC(0,
                                                       UA_NS0ID_ORGANIZES);
```

```
        UA_Server_addVariableNode(server, CountNodeId1, parentNodeId3,
                                  parentReferenceNodeId3, browseName3,
                                  UA_NODEID_NULL, attr3, NULL, NULL);
        UA_ValueCallback callbackCount1 = {server, SensorTemp, NULL};
        UA_Server_setVariableNode_valueCallback(server, CountNodeId1,
                                                callbackCount1);


UA_VariableAttributes attr;
       UA_VariableAttributes_init(&attr);
       attr.displayName = UA_LOCALIZEDTEXT("en_US", "Bomba");
       UA_Int32 myInteger = 0;
       UA_Variant_setScalar(&attr.value, &myInteger,
                            &UA_TYPES[UA_TYPES_DOUBLE]);

       UA_NodeId newNodeId = UA_NODEID_STRING(1, "Bomba");
       UA_NodeId parentNodeId = UA_NODEID_NUMERIC(0,
                                UA_NS0ID_OBJECTSFOLDER);
       UA_NodeId parentReferenceNodeId = UA_NODEID_NUMERIC(0,
                                 UA_NS0ID_ORGANIZES);
       UA_NodeId variableType = UA_NODEID_NULL;
       UA_QualifiedName browseName = UA_QUALIFIEDNAME(1, "Bomba");


           UA_DataSource dateDataSource = (UA_DataSource) {.handle =
                        &myInteger, .read = onRead, .write = onWrite};

           UA_Server_addDataSourceVariableNode(server, newNodeId,
                                    parentNodeId,
                     parentReferenceNodeId, browseName, UA_NODEID_NULL,
                                      attr, dateDataSource, NULL);

 UA_VariableAttributes attr4;
       UA_VariableAttributes_init(&attr4);
       attr4.displayName = UA_LOCALIZEDTEXT("en_US", "Calentador");
       UA_Int32 myInteger4 = 0;
       UA_Variant_setScalar(&attr4.value, &myInteger4,
                                    &UA_TYPES[UA_TYPES_DOUBLE]);

       UA_NodeId newNodeId4 = UA_NODEID_STRING(1, "Calentador");
       UA_NodeId parentNodeId4 = UA_NODEID_NUMERIC(0,
                                       UA_NS0ID_OBJECTSFOLDER);
       UA_NodeId parentReferenceNodeId4 = UA_NODEID_NUMERIC(0,
                                          UA_NS0ID_ORGANIZES);
       UA_NodeId variableType4 = UA_NODEID_NULL;
       UA_QualifiedName browseName4 = UA_QUALIFIEDNAME(1, "Calentador");


           UA_DataSource dateDataSource1 = (UA_DataSource) {.handle =
                       &myInteger4, .read = onRead1, .write = onWrite1};

           UA_Server_addDataSourceVariableNode(server, newNodeId4,
                                    parentNodeId4,
                     parentReferenceNodeId4,browseName4, UA_NODEID_NULL,
                                      attr4, dateDataSource1, NULL);
```

```
    /* loop para el Sevidor */

    UA_StatusCode retval = UA_Server_run(server, &running);
    UA_Server_delete(server);
    nl.deleteMembers(&nl);
    return (int) retval;

}
```

ANEXO 2: GUI_Client_OPCUA.py

```
import tkinter as tk
from tkinter import *
from tkinter import ttk
from PIL import Image, ImageTk
import tkinter.messagebox as messagebox
import time
import collections
import matplotlib, matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
matplotlib.use('TkAgg')
import matplotlib.animation as animation
from opcua import Client, ua
import threading
import math
import random


class Cliente_OPCUA(threading.Thread):
    def __init__(self):
        threading.Thread.__init__(self)

        self.url = Client('opc.tcp://169.254.81.206:4840')
        self.url.connect()

        def get_children(node):
            children =
extract_name(root.get_child(node).get_children_descriptions())
            return children

        def extract_name(description):
            qualifiedNames = re.findall(r"QualifiedName\(.*?\)",
str(description))
            nodeNames = re.findall("\d:[a-z,A-Z_]*", str(qualifiedNames))
            return nodeNames

        root = self.url.get_root_node()

        path = ['0:Objects']
        node = []

        for node in enumerate(get_children(path)):
            print(node[0], ": ", node[1])

        print('----------------------\n')
```

```
        max_Nodes = node[0] + 1

        objects = self.url.get_objects_node()
        val = objects.get_children()

        for i in range(max_Nodes):
            print(val[i])

        print('-------------------\n')

        sen_a = str(val[1])
        sen_b = str(val[2])
        sen_c = str(val[3])
        sen_d = str(val[4])
        self.new_a = sen_a[18:37]
        self.new_b = sen_b[18:36]
        self.new_c = sen_c[18:30]
        self.new_d = sen_d[18:35]
        self.start()

    def run(self):
        self.tag1 = self.url.get_node(self.new_a)
        self.tag2 = self.url.get_node(self.new_b)
        self.tag3 = self.url.get_node(self.new_c)
        self.tag4 = self.url.get_node(self.new_d)
        self.ini = 0
        self.ini1 = 0

        while True:
            self.ini = self.tag1.get_value()
            self.ini1 = self.tag2.get_value()
            time.sleep(0.001)


class Gui(threading.Thread):
    def __init__(self, plotLength = 100, origin_time=None):
        threading.Thread.__init__(self)
        self.con1 = Cliente_OPCUA()
        self.raiz = tk.Tk()
        self.raiz.geometry('1200x600')
        self.raiz.title('CLIENTE OPCUA')
        self.plotMaxLength = plotLength
        self.data = collections.deque([0] * plotLength, maxlen=plotLength)
        self.data1 = collections.deque([0] * plotLength, maxlen=plotLength)
        self.data2 = collections.deque([0] * plotLength, maxlen=plotLength)
        if origin_time is None:
            origin_time = time.time()
        self.Cp = 0.0
        self.Ci = 0.0
        self.Cd = 0.0
        self.previous_time = origin_time
        self.previous_error = 0.0
        self.plotTimer = 0
        self.previousTimer = 0
        self.value = 0
        self.pwm = 0
        self.read_pwm = IntVar()
        self.switch_variable = tk.StringVar(value="OFF")
```

```
        self.Botones()
        self.Etiquetas()
        self.send_sp()
        self.send_ganacias()
        self.on()
        self.off()
        self.start()

    def on(self):

        out1 = self.con1.tag4
        dv = ua.DataValue(ua.Variant(1, ua.VariantType.Double))
        out1.set_value(dv)

    def off(self):

        out1 = self.con1.tag4
        dv = ua.DataValue(ua.Variant(0, ua.VariantType.Double))
        out1.set_value(dv)

    def Update(self, error, current_time=None):
        if current_time is None:
            current_time = time.time()
        dt = current_time - self.previous_time
        if dt <= 0.0:
            return 0
        de = error - self.previous_error

        self.Cp = error
        self.Ci += error * dt
        self.Cd = de/dt

        self.previous_time = current_time
        self.previous_error = error

        return (
                (self.Kp * self.Cp)
                + (self.Ki * self.Ci)
                + (self.Kd * self.Cd)
        )

    def run(self):

        while True:

            self.Kp = self.en_kp
            self.Ki = self.en_ki
            self.Kd = self.en_kd
            self.set_point = self.set_p
            self.nivel_actual = float(self.con1.ini)*1.65
            self.error = self.set_point - self.nivel_actual
            self.correction = self.Update(self.error)

            self.out = self.con1.tag3
            self.pwm = self.correction * 170
```

```
            if self.pwm <= 0:
                self.pwm = 0

                dv = ua.DataValue(ua.Variant(self.pwm,
                                  ua.VariantType.Float))
                self.out.set_value(dv)
                time.sleep(0.01)
            else:
                if self.pwm >= 1020:
                    self.pwm = 1020

                    dv = ua.DataValue(ua.Variant(self.pwm,
                                      ua.VariantType.Float))
                    self.out.set_value(dv)
                    time.sleep(0.01)

                else:
                    dv = ua.DataValue(ua.Variant(self.pwm,
                                      ua.VariantType.Float))
                    self.out.set_value(dv)
                    time.sleep(0.01)

            #print('Nivel Actual= ' + str(self.nivel_actual))
            #print('Set_Point = ' + str(self.set_point))
            #print('Error = ' + str(self.error))
            #print('PWM = ' + str(self.pwm))
            #print('------\n')

            self.progress.set(self.pwm)
            self.read_pwm.set("{0:.2f}".format(self.pwm/10.2))

    def Botones(self):

        Set_Point = tk.Button(self.raiz, text='Enviar', width=10,
                              command=self.send_sp)
        Set_Point.place(x=170, y=195)
        Send_Ganacias = tk.Button(self.raiz, text='Enviar', width=10,
                                  command=self.send_ganacias)
        Send_Ganacias.place(x=170, y=275)

        on_button = tk.Radiobutton(self.raiz, text="ON",
                                   variable=self.switch_variable,
                                    indicatoron=False, value="ON",
                      foreground="green", width=5, command=self.on)
        on_button.place(x=95, y=505)

        off_button = tk.Radiobutton(self.raiz, text="OFF",
                                    variable=self.switch_variable,
                                    indicatoron=False, value="OFF",
                        foreground="red", width=5, command=self.off)
        off_button.place(x=140, y=505)

    def Etiquetas(self):

        Label(self.raiz, text="CONTROLES PID").place(x=135, y=140)
        Label(self.raiz, text="-----------------------").place(x=30, y=170)
        Label(self.raiz, text="Set_Point").place(x=30, y=198)
        Label(self.raiz, text="-----------------------").place(x=30, y=220)
```

```
        Label(self.raiz, text="KP").place(x=60, y=238)
        Label(self.raiz, text="KI").place(x=60, y=278)
        Label(self.raiz, text="KD").place(x=60, y=318)
        Label(self.raiz, text="% PID").place(x=50, y=380)

        self.SP_entry = tk.Entry(self.raiz, width=10)
        self.SP_entry.insert(0, '0.0')
        self.SP_entry.place(x=90, y=200)
        self.KP_entry = tk.Entry(self.raiz, width=10)
        self.KP_entry.insert(0, '0.0')
        self.KP_entry.place(x=90, y=240)
        self.KI_entry = tk.Entry(self.raiz, width=10)
        self.KI_entry.insert(0, '0.0')
        self.KI_entry.place(x=90, y=280)
        self.KD_entry = tk.Entry(self.raiz, width=10)
        self.KD_entry.insert(0, '0.0')
        self.KD_entry.place(x=90, y=320)
        Label(self.raiz, text="--------------------").place(x=30, y=350)

        self.progress = tk.IntVar()
        level_pwm = ttk.Progressbar(self.raiz, orient="horizontal",
                            mode="determinate", variable=self.progress)
        level_pwm["maximum"] = 1020
        level_pwm.place(x=90, y=382, width=120)

        get_level_pwm = tk.Entry(self.raiz, width=5, textvariable=
                                                        self.read_pwm)
        get_level_pwm.place(x=215, y=383)

        Label(self.raiz, text="CALENTADOR").place(x=70, y=450)
        Label(self.raiz, text="--------------------").place(x=30, y=480)

    def send_sp(self):
            try:
                self.set_p = float(self.SP_entry.get())
                # self.set_p = random.randint(1, 10)
            except ValueError:
                messagebox.showinfo(message='POR FAVOR INGRESE UN NUMERO
                                    VALIDO', title='ADVERTENCIA')

    def send_ganacias(self):

            try:
                self.en_kp = float(self.KP_entry.get())
                self.en_ki = float(self.KI_entry.get())
                self.en_kd = float(self.KD_entry.get())

            except ValueError:
                messagebox.showinfo(message='POR FAVOR INGRESE UN NUMERO
                                    VALIDO', title='ADVERTENCIA')

    def tag1(self, frame, lines, lineValueText, lineLabel1):

        self.value = float(self.con1.ini) * 1.65
        self.data.append(self.value)
        lines.set_data(range(self.plotMaxLength), self.data)
        lineValueText.set_text('[' + lineLabel1 + '] = ' +
                               str("{0:.2f}".format(self.value)))
```

```
    def Set_Point(self, frame, lines1, lineValueText1, lineValueText3,
                  timeText1):

        currentTimer = time.clock()
        self.plotTimer = int((currentTimer - self.previousTimer) * 1000)
        self.previousTimer = currentTimer

        timeText1.set_text('Muestreo = ' + str(self.plotTimer) + 'ms')

        self.value1 = float(self.set_p)
        self.data1.append(self.value1)
        lines1.set_data(range(self.plotMaxLength), self.data1)
        lineValueText1.set_text('[Set Point] = ' + str(self.value1))
        lineValueText3.set_text('[Error] = ' +
             str("{0:.2f}".format(math.fabs(self.value1 - self.value))))

    def tag2(self, frame, lines2, lineValueText2, lineLabel2, timeText2):

        timeText2.set_text('Muestreo  = ' + str(self.plotTimer) + 'ms')
        value2 = random.randint(17, 18)
        #value2 = float(self.con1.ini1)*30
        self.data2.append(value2)
        lines2.set_data(range(self.plotMaxLength), self.data2)
        lineValueText2.set_text('[' + lineLabel2 + '] = ' +
                                        str("{0:.2f}".format(value2)))

def main():

            maxPlotLength = 100
            xmin = 0
            xmax = maxPlotLength
            ymin = 0
            ymax = 10
            app = Gui()
            image = Image.open(r"C:\Users\fabia\Desktop\im1.png")
            background_image = ImageTk.PhotoImage(image)

            background_label = tk.Label(app.raiz, image=background_image)
            background_label.place(x=58, y=120)

            image1 = Image.open(r"C:\Users\fabia\Desktop\im2.png")
            background_image1 = ImageTk.PhotoImage(image1)

            background_label = tk.Label(app.raiz, image=background_image1)
            background_label.place(x=160, y=430)

            fig = plt.figure(figsize=(12, 8), dpi=75)
            ax = fig.add_subplot(211)
            ax.set_xlim(xmin, xmax)
            ax.set_ylim(ymin, ymax)
            ax.set_title('Control PID Nivel')
            ax.set_xlabel("Tiempo")
            ax.set_ylabel("Variable de Control vs Variable de Proceso")

            lineLabel1 = 'Nivel Actual'
            lines = ax.plot([], [], linewidth=1.0, label=lineLabel1)[0]
            lineValueText = ax.text(0.8, 0.82, '', transform=ax.transAxes)
            plt.xticks([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
```

```
            plt.legend(loc="upper left")
            lineLabel = 'Set Point (Litros)'
            timeText1 = ax.text(0.8, 0.9, '', transform=ax.transAxes)
            lines1 = ax.plot([], [], linewidth=1.0, label=lineLabel)[0]
            lineValueText1 = ax.text(0.80, 0.73, '', transform=ax.transAxes)
            lineValueText3 = ax.text(0.80, 0.65, '', transform=ax.transAxes)
            plt.legend(loc="upper left")
            plt.grid(True)

            ax1 = fig.add_subplot(212)
            ax1.set_xlim(xmin, xmax)
            ax1.set_ylim(-20, 120)
            ax1.set_xlabel("Tiempo")
            ax1.set_ylabel("Temperatura")

            lineLabel2 = 'Temperatura °C'
            timeText2 = ax1.text(0.80, 0.9, '', transform=ax1.transAxes)
            lines2 = ax1.plot([], [], color="green", linewidth=0.8,
                                                label=lineLabel2)[0]
            lineValueText2 = ax1.text(0.80, 0.82, '',
                                      transform=ax1.transAxes)
            plt.xticks([10, 20, 30, 40, 50, 60, 70, 80, 90, 100])
            plt.legend(loc="upper center")
            plt.grid(True)

            canvas = FigureCanvasTkAgg(fig, master=app.raiz)
            canvas.get_tk_widget().pack(side=RIGHT)

            anim = animation.FuncAnimation(fig, app.tag1, fargs=(lines,
                                           lineValueText, lineLabel1),
                                              interval=1, blit=False)
            anim1 = animation.FuncAnimation(fig, app.Set_Point,
                        fargs=(lines1, lineValueText1, lineValueText3,
                                              timeText1), interval=1,
                                                    blit=False)
            anim2 = animation.FuncAnimation(fig, app.tag2, fargs=(lines2,
                                 lineValueText2, lineLabel2, timeText2),
                                               interval=1, blit=False)

            app.raiz.mainloop()


if __name__ == '__main__':

    th1 = threading.Thread(target=main)
    th1.start()
    th1.join()
```

ANEXO 4: RMA 42 Manual Técnico

Technical Information
RMA42

Process transmitter with control unit

Digital process transmitter for monitoring and visualizing analog measured values

Application

- Plant and apparatus engineering and construction
- Control rooms and cabinets
- Laboratories
- Process recording and supervision
- Process control
- Signal adjustment and signal conversion
- Overfill protection according to WHG

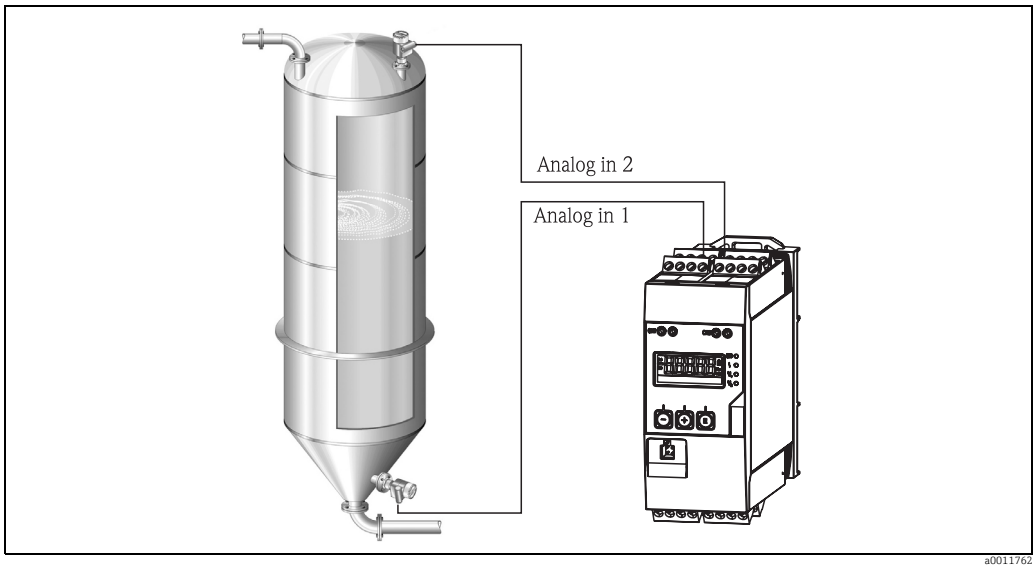
Your benefits

- 5-digit, 7-segment backlit LC display
- User-configurable dot matrix display range for bar graph, units and tag name
- 1 or 2 universal inputs
- 2 relays (optional)
- Min./max. value saved
- 1 or 2 calculated values
- One linearization table with 32 points for each calculated value
- 1 or 2 analog outputs
- Digital status output (open collector)
- Operation using 3 keys
- Configuration via interface and FieldCare or DeviceCare software

Endress+Hauser
People for Process Automation

Function and system design

Application



Example for "differential pressure" application

The RMA42 process transmitter powers the transmitter and processes analog signals from transmitters, particularly from the area of process instrumentation. These signals are monitored, evaluated, calculated, saved, separated, linked, converted and displayed. The signals, intermediate values and the results of calculations and analysis are transmitted by digital or analog means.

Measuring system

The RMA42 is a process transmitter, which is controlled by a microcontroller, and exhibits a display, analog inputs for process and status signals, analog and digital outputs, as well as an interface for configuration.
Connected sensors (e.g. temperature, pressure) can be powered by the integrated transmitter power supply system. The signals to be measured are converted from analog to digital signals, processed digitally in the device, and then converted from digital to analog signals and made available to the various outputs. All measured values, and values calculated in any way, are available as a signal source for the display, all outputs, relays and the interface. It is possible to make multiple use of the signals and results (e.g. a signal source as an analog output signal and limit value for a relay).

Input

Measured variable

Current, voltage, resistance, resistance thermometer, thermocouples

Calculated process variables

Mathematics functions

The following mathematics functions are available in RMA42:

- Sum
- Difference
- Mean
- Linearization
- Multiplication

Linearization function

Up to 32 user-definable points are available in the device per calculated value to linearize the input, e.g. for tank linearization. In the case of the two-channel device (option), mathematics channel M2 can be used to linearize mathematics channel M1.
Linearization is also available in the FieldCare configuration software.

Measuring ranges

Current:
- 0/4 to 20 mA +10% overrange
- Short-circuit current: max. 150 mA
- Load: 10 Ω

Voltage:
- 0 to 10 V, 2 to 10 V, 0 to 5 V, 0 to 1 V, 1 to 5 V, ± 1 V ± 10 V, ± 30 V, ± 100 mV
- Max. permitted input voltage:
  Voltage ≥ 1 V: ± 35 V
  Voltage < 1 V: ± 12 V
- Input impedance: > 1 MΩ

Resistance:
- 30 to 3000 Ω

Resistance thermometer:
- Pt100 as per IEC60751, GOST, JIS1604
- Pt500 and Pt1000 as per IEC60751
- Cu100, Cu50, Pt50, Pt46, Cu53 as per GOST
- Ni100, Ni1000 as per DIN 43760

Thermocouple types:
- Type J, K, T, N, B, S, R as per IEC60584
- Type U as per DIN 43710
- Type L as per DIN 43710, GOST
- Type C, D as per ASTM E998

Number of inputs

One or two universal inputs

Measurement cycle

200 ms

Galvanic isolation

Towards all other circuits

Output

Output signal

One or two analog outputs, galvanically isolated

Current/voltage output

Current output:
- 0/4 to 20 mA
- Overrange up to 22 mA

Voltage:
- 0 to 10 V, 2 to 10 V, 0 to 5 V, 1...5 V
- Overrange: up to 11 V, short-circuit proof, I_{max} < 25 mA

Loop power supply
- Open-circuit voltage: 24 V DC (+15% /-5%)
  Ex version: > 14 V at 22 mA
  Non-hazardous operation: > 16 V at 22 mA
- Maximum 30 mA short-circuit-proof and overload-proof
- Galvanically isolated from system and outputs

HART®:
HART® signals are not affected

Status output

Open Collector for monitoring of the device state and alarm notification. The OC output is closed in normal state. In error state, the OC output is opened.
- I_{max} = 200 mA
- U_{max} = 28 V
- $U_{on/max}$ = 2 V at 200 mA

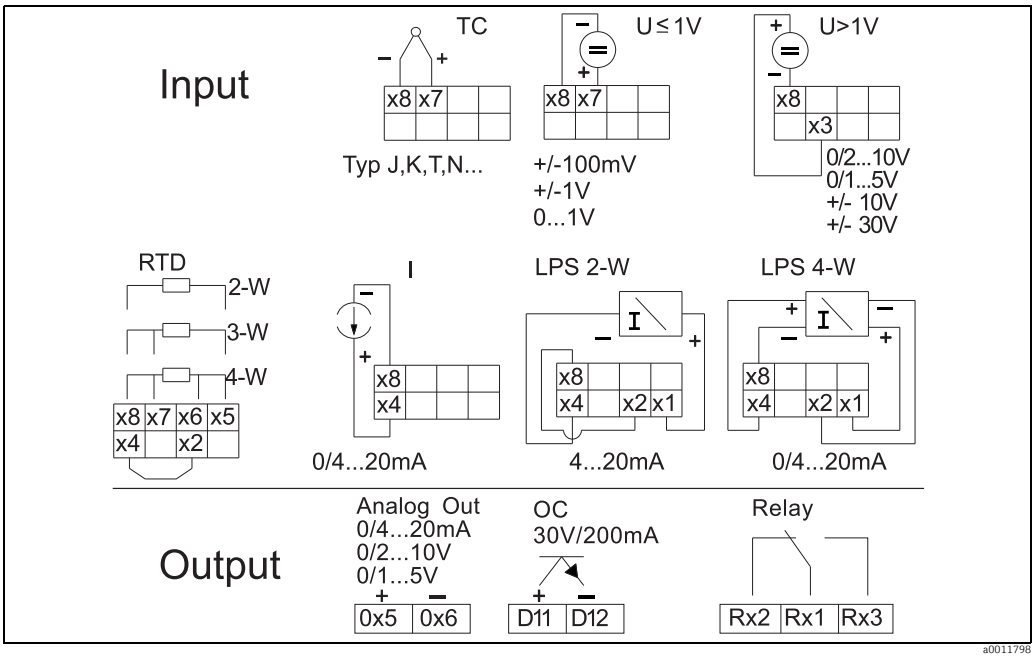
Galvanic isolation towards all other circuits; testing voltage 500 V

Limit function

Relay contact	Changeover
Maximum contact burden DC	30 V / 3 A (permanent state, without destruction of the input)
Maximum contact burden AC	250 V / 3 A (permanent state, without destruction of the input)
Minimum contact load	500 mW (12 V/10 mA)
Galvanic isolation towards all other circuits	Test voltage 1500V AC
Switching cycles	> 1 million

Power supply

Terminal assignment



Terminal assignment of the process transmitter (relays (terminals Rx1-Rx3) and channel 2 (terminals 21-28 and O25/O26) are optional)

Supply voltage

Wide-range power supply unit 24 to 230 V AC/DC (-20 % / +10 %) 50/60 Hz

Power consumption

Max. 21.5 VA / 6.9 W

Power loss

- 1-channel version: max. 2.8 W
- 2-channel version: max. 5.3 W

Connection data interface

Commubox FXA291 PC USB interface

- Connection: 4-pin connector
- Transmission protocol: FieldCare
- Transmission rate: 38,400 Baud

Interface cable TXU10-AC PC USB interface

- Connection: 4-pin connection
- Transmission protocol: FieldCare
- Delivery scope: Interface cable incl. FieldCare Device Setup DVD with all Comm DTMs and Device DTMs

HART®

- Connection sockets on the front of the device
- Internal communication resistor

Performance characteristics

Reference operating conditions

Power supply: 230 V AC, 50/60 Hz
Ambient temperature: 25 °C ± 5 °C (77 °F ± 9 °F)
Humidity: 20 % to 60 % rel. humidity

Maximum measured error

Universal input:

Accuracy	Input:	Range:	Maximum measured error of measuring range (oMR):
	Current	0 to 20 mA, 0 to 5 mA, 4 to 20 mA; overrange: to 22 mA	± 0.05%
	Voltage ≥ 1 V	0 to 10 V, 2 to 10 V, 0 to 5 V, 1 to 5 V, 0 to 1 V, ± 1 V, ± 10 V, ± 30 V	± 0.1%
	Voltage < 1 V	± 100 mV	± 0.05%
	Resistance measurement	30 to 3000 Ω	4-wire: ± (0.10% oMR + 0.8 Ω) 3-wire: ± (0.10% oMR + 1.6 Ω) 2-wire: ± (0.10% oMR + 3 Ω)
	Resistance thermometer	Pt100, -200 to 850 °C (-328 to 1562 °F) (IEC60751, α=0.00385) Pt100, -200 to 850 °C (-328 to 1562 °F) (JIS1604, w=1.391) Pt100, -200 to 649 °C (-328 to 1200 °F) (GOST, α=0.003916) Pt500, -200 to 850 °C (-328 to 1562 °F) (IEC60751, α=0.00385) Pt1000, -200 to 600 °C (-328 to 1112 °F) (IEC60751, α=0.00385)	4-wire: ± (0.10% oMR + 0.3 K (0.54 °F)) 3-wire: ± (0.10% oMR + 0.8 K (1.44 °F)) 2-wire: ± (0.10% oMR + 1.5 K (2.7 °F))
		Cu100, -200 to 200 °C (-328 to 392 °F) (GOST, w=1.428) Cu50, -200 to 200 °C (-328 to 392 °F) (GOST, w=1.428) Pt50, -200 to 1100 °C (-328 to 2012 °F) (GOST, w=1.391) Pt46, -200 to 850 °C (-328 to 1562 °F) (GOST, w=1.391) Ni100, -60 to 250 °C (-76 to 482 °F) (DIN43760, α=0.00617) Ni1000, -60 to 250 °C (-76 to 482 °F) (DIN43760, α=0.00617)	4-wire: ± (0.20% oMR + 0.3 K (0.54 °F)) 3-wire: ± (0.20% oMR + 0.8 K (1.44 °F)) 2-wire: ± (0.20% oMR + 1.5 K (2.7 °F))
		Cu53, -50 to 200 °C (-58 to 392 °F) (GOST, w=1.426)	4-wire: ± (0.30% oMR + 0.3 K (0.54 °F)) 3-wire: ± (0.30% oMR + 0.8 K (1.44 °F)) 2-wire: ± (0.30% oMR + 1.5 K (2.7 °F))
	Thermocouples	Type J (Fe-CuNi), -210 to 1200 °C (-346 to 2192 °F) (IEC60584)	± (0.1% oMR +0.5 K (0.9 °F)) from -100 °C (-148 °F)
		Type K (NiCr-Ni), -200 to 1372 °C (-328 to 2502 °F) (IEC60584)	± (0.1% oMR +0.5 K (0.9 °F)) from -130 °C (-202 °F)
		Type T (Cu-CuNi), -270 to 400 °C (-454 to 752 °F) (IEC60584)	± (0.1% oMR +0.5 K (0.9 °F)) from -200 °C (-328 °F)
		Type N (NiCrSi-NiSi), -270 to 1300 °C (-454 to 2372 °F) (IEC60584)	± (0.1% oMR +0.5 K (0.9 °F)) from -100 °C (-148 °F)
		Type L (Fe-CuNi), -200 to 900 °C (-328 to 1652 °F) (DIN43710, GOST)	± (0.1% oMR +0.5 K (0.9 °F)) from -100 °C (-148 °F)
		Type D (W3Re/W25Re), 0 to 2495°C (32 to 4523°F)(ASTME998)	± (0.15% oMR +1.5 K (2.7 °F)) from 500 °C (from 932 °F)
		Type C (W5Re/W26Re), 0 to 2320°C (32 to 4208°F)(ASTME998)	± (0.15% oMR +1.5 K (2.7 °F)) from 500 °C (932 °F)
		Type B (Pt30Rh-Pt6Rh), 0 to 1820 °C (32 to 3308 °F) (IEC60584)	± (0.15% oMR +1.5 K (2.7 °F)) from 600 °C (1112 °F)
		Type S (Pt10Rh-Pt), -50 to 1768 °C (-58 to 3214 °F) (IEC60584)	± (0.15% oMR +3.5 K (6.3 °F)) for -50 to 100 °C (-58 to 212 °F) ± (0.15% oMR +1.5 K (2.7 °F)) for 100 to 1768 °C (212 to 3214 °F)
		Type R (Pt13Rh-Pt), -50 to 1768 °C (-58 to 3214 °F) (IEC60584)	± (0.15% oMR +3.5 K (6.3 °F)) for -50 to 100 °C (-58 to 212 °F) ± (0.15% oMR +1.5 K (2.7 °F)) for 100 to 1768 °C (212 to 3214 °F)
		Type U (Cu-CuNi), -200 to 600 °C (-328 to 1112 °F) (DIN 43710)	± (0.15% oMR +0.5 K (0.9 °F)) from -100 °C (-148 °F)

AD converter resolution	16 bit
Temperature drift	Temperature drift: ≤ 0.01%/ K (0.1%/18 °F) oMR ≤ 0.02%/ K (0.2%/18 °F) oMR for Cu100, Cu50, Cu53, Pt50 and Pt46

Analog output:

Current	0/4 to 20 mA, overrange to 22 mA	± 0.05% of measuring range
	Max. load	500 Ω
	Max. inductance	10 mH
	Max. capacitance	10 µF
	Max. ripple	10 mVpp at 500 Ω, frequency < 50 kHz
Voltage	0 to 10 V, 2 to 10 V 0 to 5 V Overrange: up to 11 V, short-circuit proof, I_{max} < 25 mA	± 0.05% of measuring range ± 0.1% of measuring range
	Max. ripple	10 mVpp at 1000 Ω, frequency < 50 kHz
Resolution	13 bit	
Temperature drift	0.01%/K (0.1%/18 °F) of of measuring range	
Galvanic isolation	Testing voltage of 500 V towards all other circuits	

Installation

Installation instructions

Mounting location
Mounting on top-hat rail as per IEC 60715.
Orientation
Vertical or horizontal.

NOTICE
Heat accumulation when installing several devices on a vertically mounted top-hat rail
- ▸ Keep sufficient gaps between the individual devices.

Environment

Ambient temperature range
Non-Ex/Ex devices: -20 to +60 °C (-4 to 140 °F)
UL devices: -20 to 50 °C (-4 to 122 °F)
To avoid heat accumulation, always make sure the device is sufficiently cooled.
If the device is operated in the upper temperature limit range, this reduces the operating life of the display.

Storage temperature
-40 to +85 °C (-40 to 185 °F)

Operating height
< 2000 m above MSL (6561 ft)

Climate class
As per IEC 60654-1, Class B2

Degree of protection
Top-hat rail housing IP 20

Condensation
Not permitted

Electrical safety

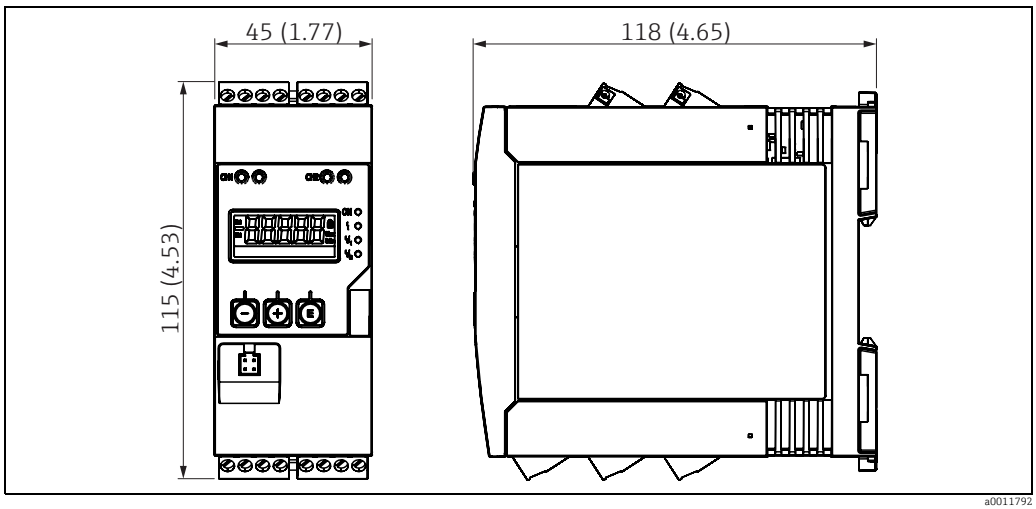
Protection class II, overvoltage category II, pollution level 2

Electromagnetic compatibility (EMC)

- Interference immunity:
  To IEC 61326 industrial environments / NAMUR NE 21
- Interference emissions:
  To IEC 61326 Class A

Mechanical construction

Design, dimensions



Dimensions of the process transmitter in mm (in)

Weight

Approx. 300 g (10.6 oz)

Material

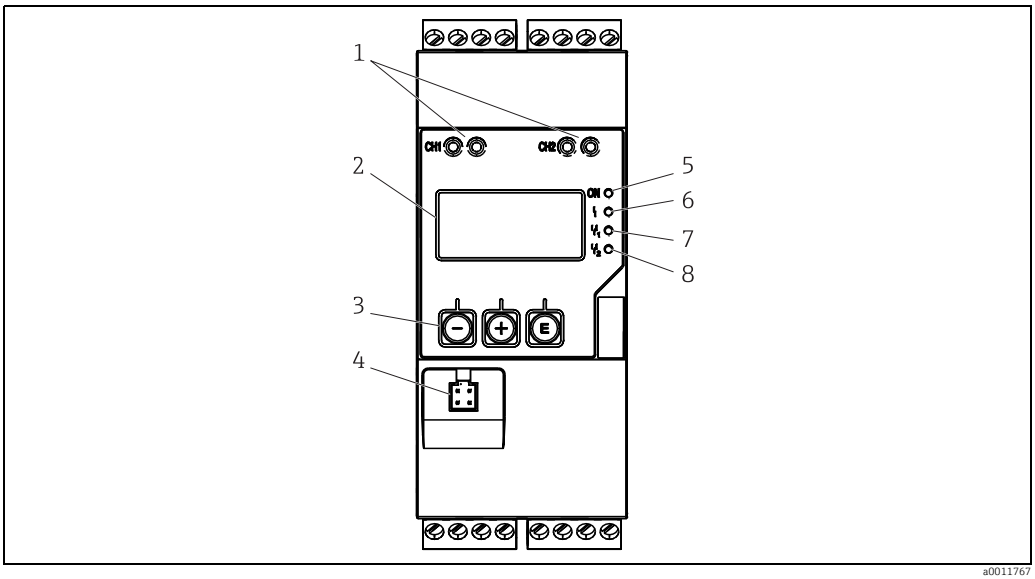
- Housing: plastic PC-GF10

Terminals

Screw terminals, plug-in; 2.5 mm^2 (30-12 AWG; torque 0.5-0.6 Nm (4.4-5.3 lb in))

Operability

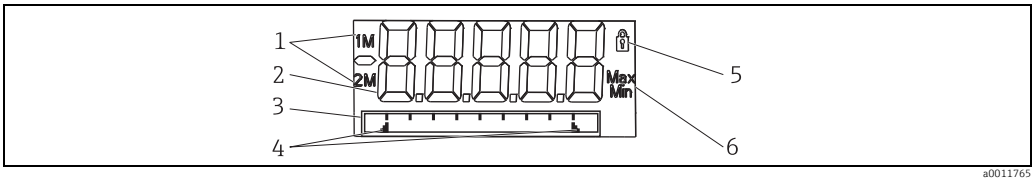
Local operation



a0011767

Display and operating elements of the process transmitter

1: HART® connection sockets
2: Display
3: Operating keys
4: PC interface connection port
5: Green LED; on = supply voltage applied
6: Red LED; on = error/alarm
7: Yellow LED; on = relay 1 energized
8: Yellow LED; on = relay 2 energized



a0011765

Display of the process transmitter

1: Channel display: 1: analog input 1; 2: analog input 2; 1M: calculated value 1; 2M: calculated value 2
2: Measured value display
3: Dot matrix display for TAG, bar graph and unit
4: Limit value indicators in the bar graph
5: "Operation locked" indicator
6: Minimum/maximum value indicator

- Display
  5-digit, 7-segment backlit LC display
  Dot matrix for text/bar graph
- Display range
  -99999 to +99999 for measured values
- Signaling
  – Setup security locking (lock)
  – Measuring range overshoot/undershoot
  – 2 x status relay (only if relay option was selected)

Operating elements

3 keys: -, +, E

Remote operation	**Configuration** The device can be configured with the PC software or on site using the operating keys. FieldCare Device Setup is delivered together with the Commubox FXA291 or TXU10-AC (see 'Accessories') or can be downloaded free of charge from www.endress.com. **Interface** 4-pin socket for the connection with a PC via Commubox FXA291 or TXU10-AC interface cable (see 'Accessories')

Certificates and approvals

CE mark	The device meets the legal requirements of the EU directives. Endress+Hauser confirms that the device has been tested successfully by affixing the CE mark.
Ex approval	Information about currently available Ex versions (ATEX, FM, CSA, etc.) can be supplied by your E+H Sales Center on request. All explosion protection data are given in a separate documentation which is available upon request.
UL	UL listed (optional)
Functional safety	SIL2 (optional)
Power plant	Seismic test acc. to KTA3505 (optional)
Overspill protection	Acc. to German WHG (optional)
Marine certificate	GL Marine certificate (optional)
Other standards and guidelines	▪ IEC 60529: Degrees of protection by housing (IP code) ▪ IEC 61010-1: 2001 Cor 2003 Safety requirements for electrical equipment for measurement, control and laboratory use ▪ EN 60079-11 Explosive atmospheres - Part 11: equipment protection by intrinsic safety "I"

Ordering information

Detailed ordering information is available from the following sources:

- In the Product Configurator on the Endress+Hauser website: www.endress.com → Select country → Instruments → Select device → Product page function: Configure this product
- From your Endress+Hauser Sales Center: www.endress.com/worldwide

Product Configurator - the tool for individual product configuration

- Up-to-the-minute configuration data
- Depending on the device: Direct input of measuring point-specific information such as measuring range or operating language
- Automatic verification of exclusion criteria
- Automatic creation of the order code and its breakdown in PDF or Excel output format
- Ability to order directly in the Endress+Hauser Online Shop

Accessories

PC operating software

FieldCare

Interface cable

Order No.	Name
FXA291	Commubox FXA291 incl. FieldCare Device Setup and DTM library
TXU10-AC	Commubox TXU10 incl. FieldCare Device Setup and DTM library

Documentation

- Overview brochure: System components - Indicators with control unit for field and panel mounting, power supplies, barriers, transmitters, energy managers and surge arresters: FA00016K/09
- Operating Instructions for 'Process transmitter RMA42': BA00287R/09
- Ex documentation:
  ATEX II (1)G [Ex ia] IIC, ATEX II (1)D [Ex ia] IIIC: XA00095R/09/A3
  FM AIS ANI Control Drawing: ZD00082R/09/EN
  CSA AIS, ANI, NI Control Drawing: ZD00083R/09/EN
- SIL - Functional Safety Manual: SD00025R/09

GLOSARIO TÉCNICO Y ACRÓNIMOS

PLC	Controladores Lógicos Programables
HMI	Interfaz Hombre-Máquina
CPPS	Sistemas de Producción Ciber-físicos
CPS	Sistemas Ciber-Físicos
OSI	Modelo de Interconexión de Sistemas Abiertos
OPC	Protocolo de Comunicación Abierta Clásico
OPC UA	Protocolo de Comunicación Abierta de Arquitectura Unificada
SOAP	Protocolo de Acceso a Objetos Simples
HTTP	Protocolo de transferencia de hipertexto
SCADA	Supervisión, Control y Adquisición de Datos
ERP	Planificación de Recursos Empresariales
MRP	Planificación de Materiales
SBC	Procesadores de Placa Reducida
IEC	Comisión Electrotécnica Internacional
IoT	Internet de las Cosas
IIOT	Internet Industrial de las Cosas
IT	Tecnología de la Información
OT	Tecnología Operacional
DCOM	Modelo de objetos y Componentes Distribuidos
AE	Alarmas y Eventos
DA	Acceso a Datos
HDA	Acceso a datos Históricos
TCP / IP	Protocolo de Control de Transmisión/Protocolo de Internet
XML	Lenguaje de Marcado Extensible
TIC	Tecnologías Avanzadas de Información y Comunicación
SSL	Capa de sockets de seguridad
TLS	Capa de Seguridad de transporte
CA	Certificado de Autorización