Universidad de Cuenca

Facultad de Ingeniería

Escuela de Electrónica y Telecomunicaciones

"Diseño e implementación de un control tolerante a fallos activo utilizando plataformas de bajo costo para control, comunicación y supervisión de un proceso mediante programas de código libre y controladores lógicos programables."

TESIS PREVIA A LA OBTENCIÓN DEL TÍTULO DE INGENIERO EN ELECTRÓNICA Y TELECOMUNICACIONES

Autores :

Daniel Felipe Merchán Piedra
CI:*0105162788*
Jonnathan Adrián Peralta Sarmiento
CI:*0106695257*

Director :

Ing. Luis Ismael Minchala Ávila, PhD
CI:*0301453486*

Cuenca - Ecuador

2017

Resumen

En la industria es importante supervisar y controlar todos los procesos involucrados en la producción. Para lograr este objetivo usualmente se debe hacer uso herramientas informáticas especializadas, cuyos costos son generalmente elevados. Por tanto, para comunicar dispositivos de planta y efectuar una correcta supervisión de los procesos es necesario disponer de un sistema de adquisición de datos y control supervisado (SCADA, por sus siglas en inglés) con redes de campo para comunicar varios dispositivos, lo que implica una inversión importante en *hardware* y *software*, limitando su penetración en empresas pequeñas y medianas. Este proyecto propone el desarrollo de un sistema de supervisión, comunicación y control tolerante a fallos haciendo uso de controladores lógicos programables (PLCs) y plataformas de bajo costo para efectuar dicha tarea. Los resultados de validación del enfoque presentado en este trabajo se comprueban en un prototipo de supervisión y control de velocidad de un motor de corriente directa. El sistema de control propuesto cumple de manera efectiva los objetivos de diseños utilizando plataformas de software libre para su implementación.

El sistema propuesto consiste de un SCADA que integre el protocolo de comunicación industrial (OPC), y en el nivel de ejecución de sistemas de manufactura, se desarrollará un algoritmo de control activo tolerante a fallos (AFTC) con el método de espacio de paridad para el motor DC, con el fin de supervisar cambios dinámicos y corregir fallos en tiempo real.

Palabras claves :AFTC, DC, encoder, *hardware*, OPC, PLCs, SCADA .

Abstract

In industry, specially manufacturing, it is important to supervise and control all processes involved in production. To achieve this goal, it is usually necessary to make use of proprietary software whose costs are high in most cases. Therefore, to communicate the devices of the plant and to perform a proper supervision of the processes it is necessary to have a SCADA system with the field networks to communicate several devices, which implies the acquisition of hardware and software licenses, impliying an important investment and thus limiting its penetration on small and medium enterprises. This project proposes the development of a system of supervision, communication and fault tolerance control that make use of programmable logic controllers (PLCs) and low cost platforms to perform this task. A direct current (DC) motor is used as the case study. The speed measurement is performed through an incremental encoder.

In addition, it is proposed the development of a SCADA system that integrates the industrial communication protocol (OPC), and in the level of execution of manufacturing systems, an active control algorithm is developed with the parity space method for the DC motor , In order to monitor dynamic changes and accommodate faults in real time.

Keywords :AFTC, DC, encoder, *hardware*, OPC, PLCs, SCADA

Índice general

Índice de figuras

Índice de tablas

Yo, Daniel Felipe Merchán Piedra, autor de la tesis "Diseño e Implementación de un Control Tolerante a Fallos Activo utilizando plataformas de bajo costo para control, comunicación y supervisión de un proceso mediante programas de código libre y controladores lógicos programables.", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, Abril de 2017.

Daniel Felipe Merchán Piedra.
C.I: 010516278-8

Yo, Jonnathan Adrián Peralta Sarmiento, autor de la tesis "Diseño e Implementación de un Control Tolerante a Fallos Activo utilizando plataformas de bajo costo para control, comunicación y supervisión de un proceso mediante programas de código libre y controladores lógicos programables.", certifico que todas las ideas, opiniones y contenidos expuestos en la presente investigación son de exclusiva responsabilidad de su autor.

Cuenca, Abril de 2017

Jonnathan Adrián Peralta Sarmiento
C.I:010669525-7

Yo, *Daniel Felipe Merchán Piedra*, autor de la tesis *"Diseño e Implementación de un Control Tolerante a Fallos Activo utilizando plataformas de bajo costo para control, comunicación y supervisión de un proceso mediante programas de código libre y controladores lógicos programables."*, reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de *Ingeniero en Electrónica y Telecomunicaciones*. El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

Cuenca, Abril de 2017

Daniel Felipe Merchán Piedra.
C.I: 010516278-8

Yo, *Jonnathan Adrián Peralta Sarmiento*, autor de la tesis *"Diseño e Implementación de un Control Tolerante a Fallos Activo utilizando plataformas de bajo costo para control, comunicación y supervisión de un proceso mediante programas de código libre y controladores lógicos programables."*, reconozco y acepto el derecho de la Universidad de Cuenca, en base al Art. 5 literal c) de su Reglamento de Propiedad Intelectual, de publicar este trabajo por cualquier medio conocido o por conocer, al ser este requisito para la obtención de mi título de *Ingeniero en Electrónica y Telecomunicaciones*. El uso que la Universidad de Cuenca hiciere de este trabajo, no implicará afección alguna de mis derechos morales o patrimoniales como autor.

Cuenca, Abril de 2017

Jonnathan Adrián Peralta Sarmiento
C.I: 010669525-7

CERTIFICO

Que el presente proyecto de tesis: "Diseño e Implementación de un Control Tolerante a Fallos Activo utilizando plataformas de bajo costo para control, comunicación y supervisión de un proceso mediante programas de código libre y controladores lógicos programables." fue dirigido por mi persona.

Ing.Luis Ismael Minchala Ávila,PhD.

Dedicatoria

A Dios.
Por llenarme de bendiciones, por mostrarme que no existe la suerte o las coincidencias, es tu infinita bondad y el camino en el que nos pones día a día.

A mis Padres.
Por su ejemplo. Por el esfuerzo realizado durante todos estos años para que no me falte nada, porque han dado todo de ellos para que todo mi trayecto tenga la menor cantidad de obstáculos.

A mis hermanos.
Por hacer de mí una mejor persona, enseñándome todos los días a ser más paciente, dedicado y obediente. Porque creciendo juntos aprendimos mucho, aplicando los valores que nuestros padres nos enseñaron.

A mis dos ángeles.
A mi ángel en la Tierra, mi hermoso sobrino Dylan. Por sacarme siempre una sonrisa en los momentos de tristeza y estrés, por anular los problemas a los que me enfrento todos los días con tan solo un abrazo y un beso.
A mi ángel en el cielo, mi abuela Dina. Por tu bondad, porque siempre transmitiste calma en momentos de zozobra. Estoy seguro que desde el cielo nos cuidas y que como en todos mis logros académicos estarás diciendo, “Te felicito mijo”.

Daniel Merchán P.

Dedicatoria

A Dios.
Por darme fuerza en los momentos mas difíciles, permitiéndome ver el lado positivo de cada caída, para ser mejor persona y seguir adelante.

A mi abuelo.

Por ser la persona que me forjó desde pequeño, todos tus consejos y enseñanzas me han llevado a ser la persona que hoy soy, se que estarías orgulloso de mi por mis logros, y a pesar de que ya no estés conmigo te recuerdo siempre diciéndome que no me de por vencido.

A mi madre.
Que a pesar de todos los problemas que hemos pasado juntos, ha cuidado de mi, permitiendo que no me falte lo necesario para poder cumplir mis sueños, deseándome lo mejor siempre.

A mi novia.
Por ser la persona, que supo acompañarme en gran parte de este trayecto, siendo esa amiga y compañera ideal, que supo aconsejarme y darme ánimos cuando más lo necesitaba,y nadie más lo hacia. En especial cuando me daba por vencido, permitiéndome seguir adelante con mis metas.

Jonnathan Peralta S.

Agradecimientos

A Dios por la vida y la salud.
A mi familia por el apoyo entregado durante toda mi carrera estudiantil.
A los todos los profesores que han formado parte de mi formación académica, por impartir sus conocimientos y reforzar los valores inculcados en el hogar.
Un agradecimiento especial al Dr. Ismael Minchala por el tiempo invertido para apoyar el desarrollo de esta tesis.

Daniel Merchán P.

Agradecimientos

Gracias a dios por permitirme tener salud para cumplir con esta meta. A todas esas personas cercanas, que me han brindado siempre su apoyo durante mi vida estudiantil, dándome ánimos para tratar de sobresalir. A gran parte de los profesores que supieron impartirme sus conocimientos y valores. Agradezco al Dr. Ismael Minchala por brindarnos su tiempo y ayuda para el desarrollo de este trabajo.

Jonnathan Peralta S.

Abreviaciones y Acrónimos

CAN *Controller Area Network*. 20
CI circuito integrado. 26
CIM *Computer Integrated Manufacturing*. 1, 7, 11, 15

DC *Direct Current*. 2, 25, 26, 28

ERP *Enterprise Requirements Planning*. 10, 11

FCEM fuerza contra-electromotriz. 28

GUI *Graphical User Interface*. 18

HMI *Human Machine Interface*. 9, 17, 18

MES *Manufacturing Execution System*. 10
MRPII *Manufacturing Resources Planning*. 10

OPC *OLE for Proccess Control*. 3, 12–14
OPC-UA *Open Platform Communications Unified Architecture*. 2, 5, 15
OSACIM *Open Software Architecture fro Advanced CIM*. 3
OSI *Open Systems Interconnection*. 18, 19

PID proporcional-integrativo-derivativo. 21
PLC *Programmable Logic Controllers*. 7, 16, 18, 20
PWM pulse width modulation. 27

RTU *Remote Terminal Unit*. 20

SCADA *Supervisory Control And Data Acquisition*. 1–4, 9, 16, 18, 19, 26
SCD *Sistemas de Control Distribuido*. 8
SOA *Service Oriented Architecture*. 15

TCP *Transmission-Control-Protocol*. 18

UDP *User Datagram Protocol*. 18

ZOH zero order hold. 22

Capítulo 1

Introducción

1.1. Introducción

Las empresas e industrias requieren tener una visión clara de todas las fases y etapas de elaboración de un determinado producto. Los sistemas de manufactura integrada por computador (*Computer Integrated Manufacturing* (CIM) por sus siglas en inglés) emergen en los años 80 [1]. Estos sistemas permiten tener procesos totalmente automatizados y ofrecen una visión global de los mismos.

Los procesos que se desarrollan hoy en día en la industria involucran componentes como: computadoras, controladores, sensores, actuadores, maquinarias, etc. Los elementos que intervienen en estos procesos generan una gran cantidad de datos que requieren ser controlados y verificados constantemente.

El sistema de supervisión, control y adquisición de datos (SCADA por sus siglas en inglés) provee un manejo de todos los datos en tiempo real, implementa paradigmas de control más eficientes, mejora una planta de producción y reduce costos [2]. Además al ser un *softawre* que opera en un computador, otorga al operador el control de una planta por medio del teclado del ordenador [3].

El uso de arquitecturas de comunicación como OPC (*OLE for Proccess Control* (OPC) por sus siglas en inglés), ofrece una mejora en la gestión de procesos industriales, optimizando el monitoreo y control de los mismos. Protocolos como OPC-UA, permiten unificar la comunicación entre dispositivos de campo, eliminando el uso de varios controladores dentro del sistema de monitoreo.

La implementación de un sistema SCADA no es viable para determinado tipo de empresas, debido al alto costo de sus licencias. Por otro lado, existe la necesidad de adquirir *drivers* para cada uno de los dispositivos que se comuniquen con el sistema. Por esta razón, típicamente, las empresas pequeñas y medianas posponen o rechazan sistemas de integración de manufactura por computador (CIM, por sus siglas en inglés), posponiendo también los beneficios de esta tecnología en la eficiencia de sus procesos.

El presente trabajo presenta el diseño e implementación un sistema SCADA desarrollado en Python un entorno de programación en *software* libre. El sistema hace uso de un servidor OPC-UA desarrollado en el mismo lenguaje de programación, para la supervisión, control y monitoreo velocidad de un motor DC a través de plataformas industriales de bajo costo, con el fin de crear un prototipo de manufactura integrada accesible y funcional para el control de procesos industriales. El prototipo consta de un controlador de velocidad, y un sistema de detección de fallos, desarrollado utilizando el enfoque de ecuaciones de paridad.

1.2. Definición del problema

Un proceso industrial es controlado totalmente cuando se establece una comunicación entre todos los dispositivos que intervienen en él. Los vendedores de equipos de automatización desarrollan sus propios protocolos de comunicación, lo que limita a utilizar dispositivos de un único fabricante.

El estándar de comunicación industrial OPC solventa el problema de incompatibilidad de protocolos, lo que permite que dispositivos de distintos fabricantes intercambien información. En este punto, el costo de las licencias de los servidores OPC representa un potencial obstáculo para su implementación en industrias pequeñas y medianas. Este trabajo utiliza el servidor desarrollado en [4], el cual es de código abierto, como base para la implementación de un servidor de bajo costo.

Adicionalmente, se desarrolla e implementa un sistema de SCADA de código libre, el cual permite tener supervisión, que permite supervizar, monitorizar y controlar un proceso industrial. El propósito de este trabajo es superar el obstáculo de costos a través de la creación de un sistema de *software* libre.

1.3. Estado del arte

Los elementos que intervienen en los procesos industriales en la actualidad son cada vez más complejos. La necesidad de monitorear y controlar cada uno de estos procesos en tiempo real, ha hecho que estas tareas crezcan en complejidad.

En [5] desarrollan un prototipo basado en la arquitectura de 3 capas de un SCADA (Figura 1.1). El prototipo visualiza y configura datos en tiempo real de un proceso industrial. El proceso elegido es el de la pasteurización de un líquido como la leche. El proceso se apoya en 4 pasos:

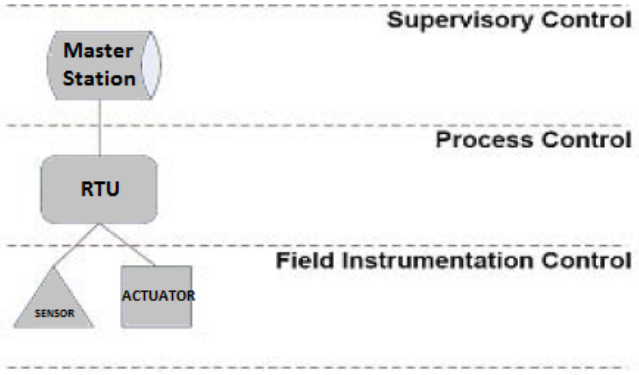


Figura 1.1: Arquitectura de 3 capas de un sistema SCADA.
Fuente: [5]

1. Llenar un tanque hasta cierto nivel.
2. Una vez puesto en marcha el horno, seguirlo calentando hasta llegar a la temperatura deseada.
3. Mantener el líquido en temperatura.
4. Después de un tiempo apagar el horno.

El prototipo utiliza una caja de simulación de sensores para la obtención de datos (capa 1). Un PLC S7-1200 programado con el *software* WIN32 se conecta a una computadora para la controlar el proceso (capa 2). El sistema de supervisión y control está elaborado en LabVIEW, tiene la capacidad de almacenar datos históricos, visualizar tendencias, alarmas, etc(capa 3). Un sistema OPC se usa para intercambiar datos entre la capa de supervisión y la capa de control. LabVIEW trabaja como un cliente OPC y el paquete de software del PLC Siemens S7-1200 como un servicio OPC.

En [6] desarrollan e implementan un sistema de manufactura integrada por computador con el uso de *software* libre (*Open Software Architecture fro Advanced CIM* (OSACIM) por sus siglas en inglés). Las plataformas utilizadas son Java y Python.
Los componentes pricipales del OSACIM son:

1. **Servidor OPC:** EL servidor es desarrollado en Python.
2. **Sistema SCADA:** Desarrollado en Java. El sistema emula una etapa de premolienda de una industria de fabricación de cemento.
3. **Cliente OPC-UA:** Emplean un cliente comercial, específicamente UaExpert, para comprobar la interoperabilidad del sistema.

El OSACIM fue expuesto a varias pruebas en las que se obtuvieron resultados favorables en tares como:

- Inicio de sesión
- Cambio de estados de un motor
- Revisión de una alarma
- Gráfica de tendencias
- Generar reportes
- Visualización del historial de una alarma
- Interoperabilidad de el sistema

El sistema se compara con una aplicación SCADA desarrollada con SIMATIC WinCC. Los datos dados por el software de WinCC sirven para la comparación entre ambos sistemas obteniendo resultados similares en todas las áreas.

Otro trabajo de comparación se encuentra en [7], en este caso se contrastan 2 sistemas SCADA comerciales con dos sistemas de *software* libre. El proceso es la producción de iones con el uso de dos prototipos, el primero llamado SPIDER destinado para pruebas. El segundo equipo es MITICA el cual es un generador a gran escala. El funcionamiento de SPIDER y MITICA requiere ser programado, sincronizado y supervisado por una planta de control. El proceso de automatización se realiza con PLCs cuyos datos varían cada 50ms. Los SCADA comerciales elegidos son: FactoryTalk View Site Edition (FTV-SE) de Rockwell Automation and PVSS II de ETM. Por otra parte los sistema de *software* libre escogidos son: EPICS desarrollado por Los Angeles National Lab y Argonne National Lab, y *TANGO* (TACO New Generation Objects) de la *European Synchrotron Radiation Facility.*

Las pruebas a las que fueron expuestos son: ensayo de arquitectura, problemas que presenten los sistemas con respecto a la configuración e instalación de *software* y prueba de rendimientos. El experimento concluye que el sistema FTV-SE es una herramienta de fácil desarrollo y que alta calidad gráfica, pero tiene problemas de operabilidad con dispositivos que no sean de un solo fabricante. El primer sistema de *software* libre, TANGO tiene como ventaja las herramientas que dispone para la personalización del producto, pero su estructura no ofrece un intercambio de datos efectivo.
En términos de rendimiento los sistemas que demuestran equivalencia son PVSS y EPICS, siendo comercial y de *software* libre respectivamente. En el área de la comunicación PVSS saca ventaja con respecto a EPICS, ya que el último necesita de una previa configuración. Acerca del rendimiento de la red, EPICS tuvo un mejor comportamiento, lo que hace que compita en iguales condiciones con sistemas comerciales.

Con el fin de abaratar costos en la industria, servidores OPC-UA de código abierto son el objetivo de recientes investigaciones. En esta búsqueda [8] desarrolla e implementa un servidor embebido en una plataforma de bajo costo.
Los objetivos son recolectar datos relacionados a un proceso y transmitir comandos a los dispositivos de control con el uso del protocolo MODBUS. El *hardware* elegido para alojar el servidor es la tarjeta Beagle Bone Black (BBB) la cual soporta cualquier distribución. Para los dispositivos de planta utiliza tarjetas Arduino.

El servidor OPC-UA y el cliente están integrados en una librería desarrollada en el lenguaje C++. El trabajo logra cumplir los objetivos planteados, emulando una línea de ensamblaje de tres etapas (Figura 1.2).

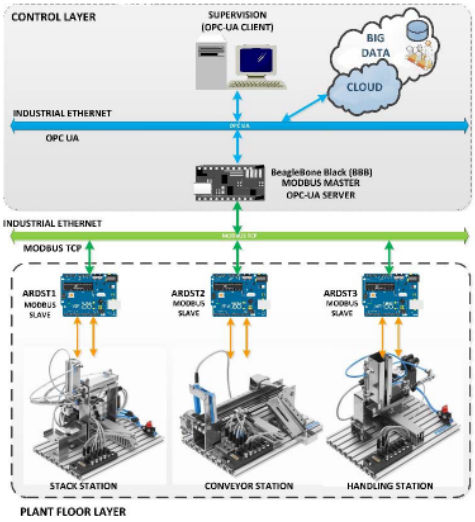


Figura 1.2: Emulación de una linea de ensamblaje.
Fuente: [8]

1.4. Objetivos

1.4.1. Objetivo general

Desarrollar e implementar un sistema de control tolerante a fallas activo utilizando un sistema SCADA y comunicación OPC con software libre, aplicado a un sistema de control de velocidad de un motor DC.

1.4.2. Objetivos específicos.

- Desarrollar una aplicación para control y supervisión de un motor DC, con el uso de plataformas de código libre.
- Mejorar y actualizar el servidor OPC-UA desarrollado en [4].
- Integrar el sistema SCADA con el servidor OPC-UA.
- Modelar el proceso para el diseño de un controlador digital de velocidad .
- Desarrollar un sistema de detección y diagnóstico de fallas para un motor DC con el método de espacios de paridad.

1.5. Contribuciones de la tesis

Este trabajo de investigación presenta las siguientes contribuciones:

- Solventar los problemas de comunicación de equipos industriales a través de un servidor de código libre.
- Reducción de costos en la adquisición de sistemas de monitoreo a nivel industrial.
- Metodología de desarrollo de herramientas de software libre para monitoreo y control supervisorio de procesos industriales

Capítulo 2

Sustento teórico

2.1. Manufactura integrada por computador

Manufactura integrada por computador (CIM, por sus siglas en inglés) es un concepto que describe un proceso completamente automatizado, es decir toda la planta opera bajo el total control de computadoras y otros sistemas digitales. CIM incluye todas las actividades que se realizan para la fabricación de un producto, desde la percepción de la necesidad, la concepción, el diseño y su desarrollo, incluyendo también su producción, marketing y soporte del producto en uso[9].
El concepto CIM expone la conocida pirámide, la cual divide una unidad de producción en niveles de comunicación para facilitar su reconocimiento. La Figura 2.1 muestra el nombre de los niveles que forman la pirámide [1].

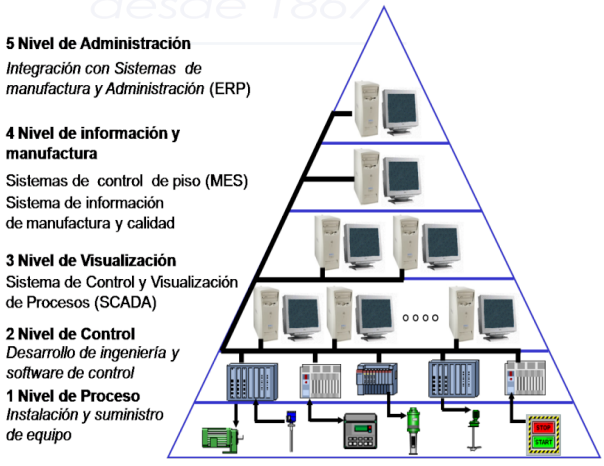


Figura 2.1: Pirámide CIM.
Fuente: Autores

2.1.1. Nivel de proceso

Comprende los instrumentos con los que se realiza las operaciones de producción de la empresa. En este nivel se adquieren las variables del proceso gracias a los sensores, y se toman acciones sobre el proceso mediante los elementos de control. La Figura 2.2 ilustra un conjunto de actuadores y sensores: En [10] se listan las características más

Figura 2.2: Sensores y actuadores desplegados en el nivel de campo.
Fuente: Autores

sobresalientes en este campo, las cuales son:

1. Descripción y propiedades detalladas de los procesos (tipo de variable, rangos).
2. Determinación de los lazos de control del proceso.
3. Implementación de respaldos mecánicos para garantizar el correcto funcionamiento de la planta en caso de un fallo en el sistema eléctrico.

2.1.2. Nivel de control

Todos los dispositivos de control tales como PLCs, RTUs, controladores industriales, *Sistemas de Control Distribuido* (SCD) están en este nivel. Se recibe y proporciona información al nivel de proceso (nivel inferior) y al nivel de visualización (nivel superior). La Figura 2.3 muestra ejemplos de elementos que forman parte del nivel de control. Las funciones que se rescatan en este nivel son:

1. Garantizar la seguridad y control del proceso.
2. Mantener una comunicación constante con el nivel de visualización que es en donde se programan sus acciones.

La característica principal de este nivel se da gracias a su arquitectura. La arquitectura

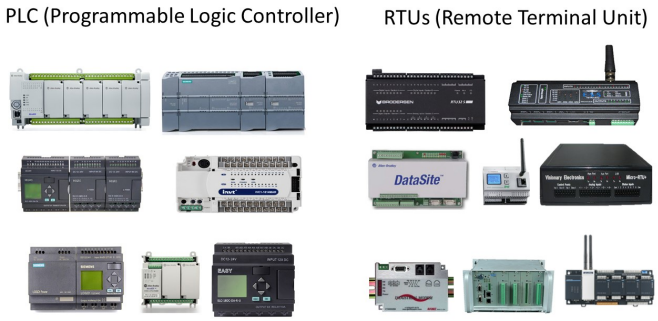


Figura 2.3: Dispositivos de nivel de control.
Fuente: Autores

permite expandir las variables de proceso de la planta y establecer comunicación con controladores del mismo nivel.

2.1.3. Nivel de visualización

Conocido también como nivel sistema SCADA. En este punto se realizan las tareas de adquisición y procesamiento de datos, control y supervisión del proceso, gestión de alarmas entre otras. Este nivel establece la interfaz humano-máquina (*Human Machine Interface* (HMI) por sus siglas en inglés) [4]. La Figura 2.4 muestra el *Human Machine Interface* (HMI) de un sistema SCADA.

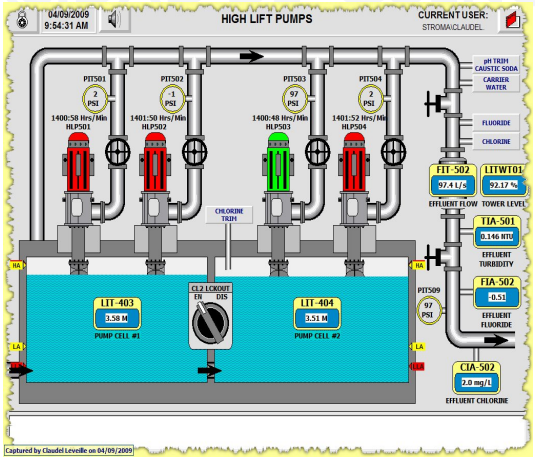


Figura 2.4: Nivel de visualización.
Fuente: Autores

2.1.4. Nivel de información y manufactura

Este nivel identifica las fortalezas y debilidades de todos los procesos en planta[10]. Los sistemas de ejecución de manufactura (*Manufacturing Execution System* (MES), por sus siglas en inglés) funcionan en este nivel. El sistema MES acorta la distancia entre el nivel de administración y de control utilizando la información en línea para gestionar la aplicación actual de los recursos de fabricación: personas, equipo e inventario[11].

2.1.4.1. Sistemas de ejecución de manufactura.

MES significa más que una herramienta de planificación como los sistemas de planificación de recursos empresariales (*Enterprise Requirements Planning* (ERP) por sus siglas en inglés) o los de planificación de recursos de manufactura (*Manufacturing Resources Planning* (MRPII) por sus siglas en inglés), es un sistema de planificación *on-line* con énfasis en la ejecución de un plan, entiéndase como ejecución a: la fabricación de productos, encendido y apagado de máquinas, fabricación y medición de piezas, etc. Las funciones de MES (Figura 2.5) son:

1. **Interfaz del sistema de planificación:** El MES debe estar directamente asociado al sistema de planificación para aceptar pedidos de trabajo y actualizar datos.
2. **Pedidos de trabajo:** El sistema gestiona los cambios en los pedidos,establece y modifica los horarios, y mantiene un plan ordenado según prioridades.
3. **Estaciones de trabajo:** Es responsable de la dirección en la que se implementa un plan de trabajo y la configuración lógica de las estaciones de trabajo.
4. **Seguimiento y manejo de inventario:** Mapea todo el inventario y los lugares de almacenamiento del mismo.
5. **Movimiento de material:** Una de las importantes contribuciones del sistema MES es el de movimiento de inventario o información de la ubicación en el piso de la planta.
6. **Recopilación de datos:** A través de varios tipos de dispositivos sensores e interfaces de control, los datos de las operaciones son almacenados.
7. **Manejo de excepciones:** Una de las funciones más importantes de la MES se refiere a cómo una empresa responde a las excepciones que se presentan en un plan. Cuando una estación de trabajo se desploma repentinamente, o cuando el material no está disponible, el sistema debe ser capaz de tomar estos cambios en tiempo real y responder con acciones alternativas.

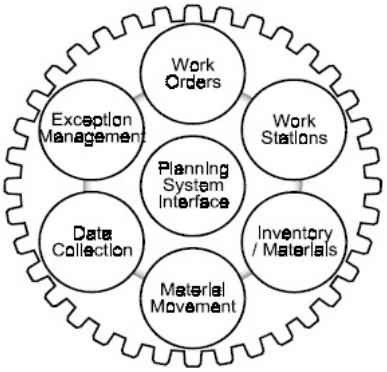


Figura 2.5: Funciones de el sistema MES.
Fuente:[11]

2.1.5. Nivel de administración

Este es el nivel de manejo corporativo. El nivel de administración realiza la planificación corporativa, la gestión de los recursos y tiene la misión de optimizar las finanzas [12]. Para lograr este objetivo se comunican una o varias plantas, obteniendo la información y los datos en tiempo real de las variables que intervienen en los procesos de producción. Los sistemas ERP se desempeñan en este nivel.

2.1.5.1. Sistemas de planificación de recursos empresariales

Los sistemas ERP tienen diferentes definiciones, en [13] encontramos tres perspectivas. En primer lugar define a ERP como un producto en forma de software informático. Como segunda definición expresa que ERP puede ser visto como un producto que tiene como objetivo mapear todos los procesos y los datos de una empresa en una estructura comprensiva. La última definición manifiesta que, ERP llega a ser la clave de una infraestructura que entrega una solución a un negocio. La segunda definición se ajusta perfectamente a la función que ejerce el nivel de administración. La Figura 2.6 ilustra todo lo que un sistema ERP engloba. Este es el nivel superior de la pirámide el cual tiene las siguientes funciones:

1. Gestionar los niveles inferiores del modelo CIM.
2. La comunicación de todas las plantas de producción de la empresa.
3. Administrar toda la producción de la empresa.

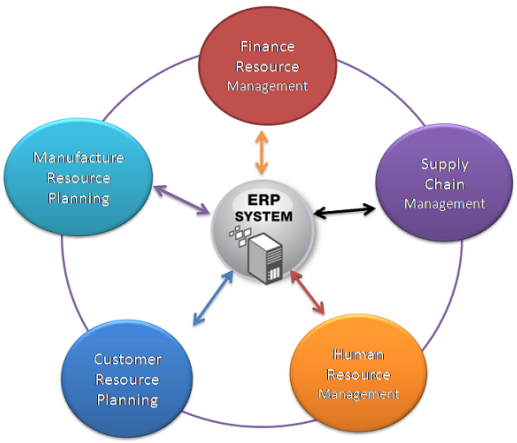


Figura 2.6: Estructura ERP.
Fuente:Autores

2.2. Estándar de comunicación OPC

2.2.1. Descripción

OPC es un conjunto de estándares y especificaciones para el uso de comunicaciones industriales. OPC permite la comunicación de los datos de planta con los dispositivos de control de diferentes fabricantes [14].

2.2.2. Propósito de OPC

Una necesidad común de las aplicaciones es el acceso a las fuentes de información desde cualquier dispositivo. Cada uno de los dispositivos requiere un *driver* independiente para que las aplicaciones interactúen con las fuentes de información, esto causa los siguientes problemas:

- Cada aplicación debe incluir un *driver* en particular para cada dispositivo (Figura 2.7) [15].
- Los *drivers* independientes causan conflictos al acceder remotamente a varios dispositivos de manera simultanea.
- Cambios en las características del *hardware* puede desestabilizar el funcionamiento del *driver* [14].

Los desarrolladores de *hardware* producen sus propios *drivers* para solucionar estas problemáticas, sin embargo no se puede elaborar *drivers* eficientes que puedan usarse por todos los clientes. Un servidor con una interfaz OPC permite que el acceso a la

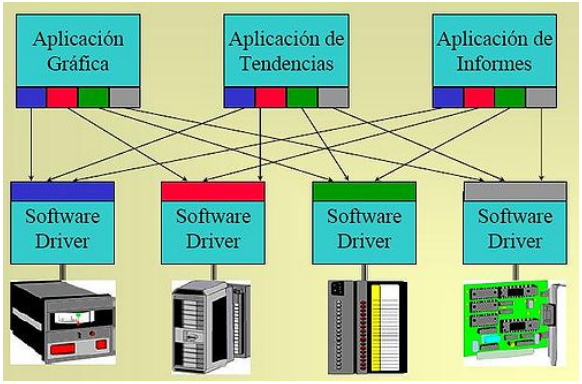


Figura 2.7: Problemática sin OPC.
Fuente: [4]

fuente de datos desde cualquier aplicación sea estándar. Con esta solución los proveedores crean servidores optimizados que permitan mantener un solo mecanismo para el acceso de datos de una manera eficiente.

2.2.3. Solución propuesta por OPC

OPC define una interfaz abierta sobre la cual un *software* de computador es capaz de intercambiar información, siendo una base para la conectividad de aplicaciones industriales y los dispositivos de campo en el nivel de automatización (Figura 2.8). La introducción de una interfaz estandarizada para la conectividad de aplicaciones, permite que la implementación de varios *drivers* sea reducido a uno solo, en este caso el necesario para la interfaz del cliente OPC.

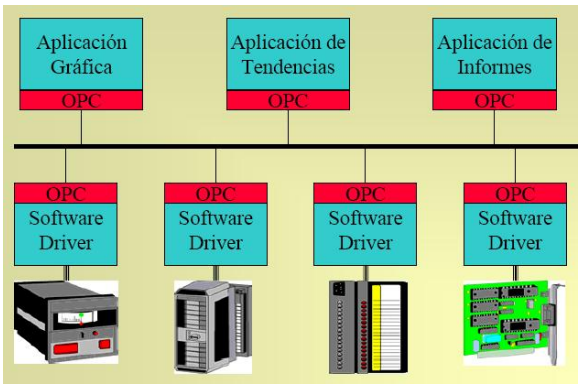


Figura 2.8: OPC solución al problema.
Fuente: [4]

2.2.4. Funcionamiento de OPC

OPC implementa una arquitectura cliente-servidor la cual promueve el intercambio de información entre estos componentes. El servidor es la base dentro de esta arquitectura dado que interactúa de manera directa con la fuente de información, a ésta se accede mediante aplicaciones cliente.
Un servidor puede ofrecer servicio a varios clientes, siempre y cuando éste posea la capacidad de conexión y velocidad de respuesta respecto a las peticiones del cliente. La Figura 2.9 muestra una topología de red en la cual todos los elementos comparten un solo medio físico, en la cual cada cliente puede acceder a cada uno de los servidores de la red [4].

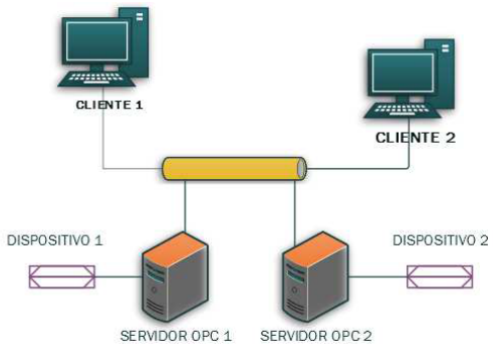


Figura 2.9: Topología de red.
Fuente: Autores

2.2.5. Tipos de servidores OPC

2.2.5.1. OPC estándar

El estándar OPC incluye varias especificaciones las cuales se distinguen por el tipo de datos y forma de acceso. Los más comunes son:

OPC DA (Data Access).
Permite compartir datos en tiempo real. Los clientes son capaces de leer y escribir información en el servidor. De igual manera el servidor permite enviar conjuntos de datos hacia el cliente cuando existe algún cambio dentro de las variables.

OPC HDA (Historical Data Acccess).

Permite el acceso a los datos que han sido almacenados y retirarlos de una manera uniforme por parte de sistemas SCADA o aplicaciones cliente OPC.

OPC A&E (Alarms and Events).
Ofrece notificaciones de alarmas y eventos hacia las aplicaciones cliente. Lo cual incluye alarmas de procesos, acciones por parte de un operador y mensajes informativos. Esta clase de servidor detecta las condiciones de los dispositivos conectados para generar dichas alarmas o notificaciones[16].

OPC-UA.
Permite acceder a varios tipos de datos, mediante el intercambio de información en un entorno cliente-servidor multi-funcional. El intercambio de información no se restringe por las limitaciones que brindan los distintos protocolos de comunicación comerciales. De esta manera, permite que los datos puedan ser transferidos entre los distintos niveles de la pirámide CIM.
La Figura 2.10 muestra la interacción de OPC-UA entre los distintos niveles de una fábrica para el intercambio de información. Grandes beneficios se obtienen con este tipo de servidor y que permite el acceso a la información incluso en el nivel corporativo. OPC-UA es una plataforma independiente, que permite unificar la comunicación en-

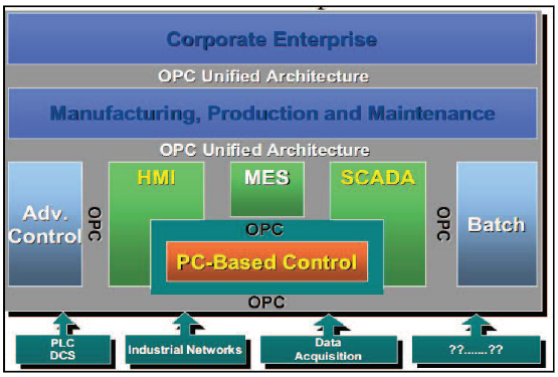


Figura 2.10: Interacción entre niveles de comunicación.
Fuente: [17]

tre dispositivos de varios fabricantes. Ofreciendo seguridad e interoperabilidad lo cual convierte a OPC-UA en un candidato para el uso de aplicaciones basadas en una arquitectura orientada a servicios (*Service Oriented Architecture* (SOA) por sus siglas en inglés) [17].

2.3. Sistema SCADA

Es una arquitectura de sistema de control que utiliza computadoras, comunicaciones de datos en red e interfaces gráficas de usuario para la supervisión de procesos. Utiliza dispositivos como *Programmable Logic Controllers* (PLC) para interactuar con dispositivos de planta o maquinaria de proceso. La Figura 2.11 expone un sistema SCADA básico.

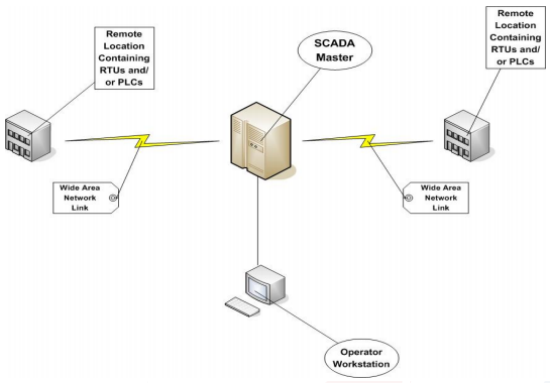


Figura 2.11: Sistema SCADA básico.
Fuente: [18]

2.3.1. Arquitectura

2.3.1.1. Arquitectura de *hardware*

El sistema SCADA se compone de dos capas, la capa cliente, que abastece a la interacción hombre-máquina y la capa de servidor de datos, que maneja la información de los procesos. Redes Ethernet LAN conectan a los servidores con las estaciones cliente [19].

2.3.1.2. Arquitectura de *software*

Los servidores están encargados del manejo y la adquisición de datos.

2.3.2. Elementos que componen un sistema SCADA

En [18] indica que un sistema SCADA consiste de:

- Uno o más dispositivos de campo, que interactúan con sensores, conmutadores y actuadores.

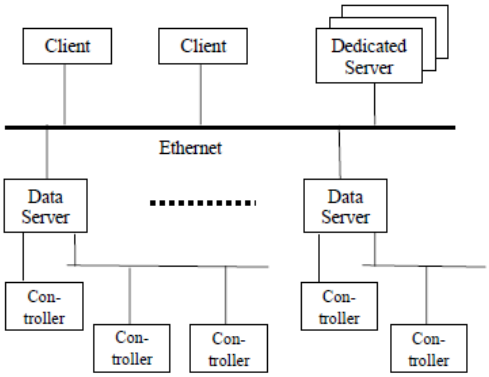


Figura 2.12: Arquitecura típica de *hardware*.
Fuente: [19]

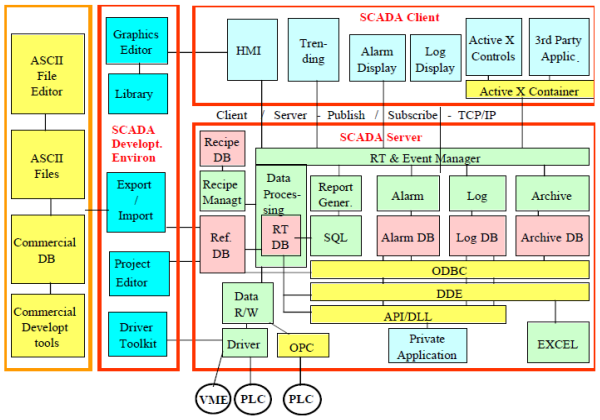


Figura 2.13: Arquitecura típica de *software*.
Fuente: [19]

Un PLC es un dispositivo de campo. Tiene un microcontrolador que ejecuta un programa de control para alterar el estado de una salida. Se puede conectar directamente a dispositivos tales como interruptores, pequeños motores, relés y solenoides, y está construido para soportar interferencias y ruido que se produce en un ambiente industrial [20].

- Un sistema de comunicaciones utilizado para transferir datos entre dispositivos de interfaz de datos de campo, unidades de control y computadoras centrales.
- Uno o varios servidores centrales.
- Una interfaz humano-máquina (HMI por sus siglas en inglés) utilizado para controlar y monitorear los dispositivos de campo. En [21] se listan las funciones de un HMI, y son:

1. **Visualización de procesos:** La pantalla del HMI se actualiza dinámicamente dependiendo de las transiciones de un proceso.
2. **Control operativo de un proceso:** El operador puede controlar el proceso por medio de una interfaz gráfica de usuario (*Graphical User Interface* (GUI) por sus siglas en inglés), por ejemplo, el operador puede pre-establecer valores de referencia para un motor, iniciarlo o apagarlo.
3. **Mostrar alarmas:** Los estados críticos del proceso activan automáticamente una alarma, por ejemplo, cuando se excede el valor un *setpoint*.
4. **Archivar valores de procesos y alarmas:** El sistema puede registrar alarmas y valores de proceso. Esto permite registrar secuencias de proceso y recuperar datos de producción anteriores.
5. **Registro de alarmas y valores de procesos:** El sistema HMI puede generar alarmas y reportes de valores de proceso. Esto permite imprimir los datos de producción.
6. **Gestión de parámetros de procesos y maquinaria:** El sistema HMI puede almacenar los parámetros de procesos y maquinaria. Por ejemplo, estos parámetros se pueden descargar en un paso desde el panel de operador al PLC para cambiar la versión del producto para la producción.

2.3.3. Comunicación

2.3.3.1. Modelo OSI

Los dispositivos tales como PLCs y RTUs están pre-programados para comunicarse con estaciones SCADA y otros sistemas de red. La comunicación usa protocolos diseñados para entregar información sobre el estado de todos los dispositivos en planta [22]. Las arquitecturas de red están basadas en el modelo de interconexión de sistemas abiertos (*Open Systems Interconnection* (OSI) por sus siglas en inglés). El modelo OSI es un estándar de 7 capas (Figura 2.14) las cuales son:

Capa 7.-Capa de aplicación: Define los protocolos que usan las aplicaciones para intercambiar datos. Los dispositivos usan protocolos típicos (por ejemplo MODBUS) para enviar información al servidor.

Capa 6.-Capa de presentación: Se encarga de unificar la representación de los datos, es decir, si dos equipos utilizan una representación diferente de caracteres, gracias a esta capa los datos son reconocibles para ambos.

Capa 5.-Capa de sesión: Esta capa establece y mantiene la conexión entre dos computadoras.

Capa 4.-Capa de transporte: Transporta los datos sin importar la red física que se utiliza. Los protocolos que usa son: Protocolo orientado a la conexión (*Transmission-Control-Protocol* (TCP)) o no orientado a la conexión (*User Datagram Protocol* (UDP)).
Capa 3.-Capa de red: Se encarga de identificar el enrutamiento existente entre una o más redes. Utiliza protocolos enrutables (enruta cada paquete ejemplo: IP) y protocolos de enrutamiento (selecciona las rutas ejemplo: OSPF, EIGRP, BGP, IGRP)
Capa 2.-Capa de enlace de datos: Se encarga del acceso al medio, direccionamiento físico de la detección de errores, de la distribución ordenada de tramas y del control del flujo.
Capa 1.-Capa física: Indica la forma en la que se transmite la información.
El objetivo del modelo OSI es el de intercambiar paquetes, señales, direcciones entre

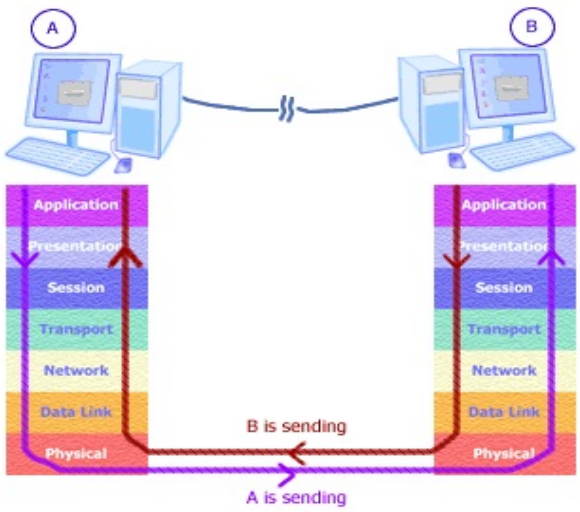


Figura 2.14: Modelo OSI.
Fuente: Autores

cualquier red o sistema interconectado.

2.3.3.2. Comunicación interna

Se fundamenta en una comunicación cliente-servidor y servidor-servidor. Esta comunicación se basa en eventos y utiliza un protocolo TCP/IP. Una aplicación cliente se suscribe a un parámetro que es propiedad de una aplicación de servidor, solo los cambios en ese parámetro se notifican al cliente [19].

Protocolo TCP/IP

Es un conjunto de reglas y protocolos que permiten el intercambio de información entre

dos ordenadores que pertenecen a una red. El protocolo basa en la idea de brindar a todos los equipos de una red una dirección IP para el enrutamiento de paquetes.

2.3.3.3. Comunicación con dispositivos

Un sistema SCADA incluye la adaptación de protocolos de comunicación entre dispositivos de conexión. Existen alrededor de 200 protocolos de aplicación ya sean propietarios y no propietarios, los más importantes son:

- MODBUS.
- Profibus.
- *Controller Area Network* (CAN).
- Fieldbus.
- DNP 3.0.
- Direct Net.

Los protocolos MODBUS, Profibus y CAN son los protocolos que se utilizan comúnmente en la industria, a continuación se realiza una breve descripción de cada uno de ellos.

- **MODBUS**

El protocolo MODBUS punto a punto se ha convertido en un estándar virtual para comunicar dispositivos como PLC y *Remote Terminal Unit* (RTU). Durante la comunicación en una red MODBUS, el protocolo determina cómo cada controlador conocerá la dirección del dispositivo, reconocerá un mensaje dirigido a él, determinará la acción que se tomará y extraerá cualquier información o datos que solicite. MODBUS no puede manejar grandes números positivos y negativos lo que no representa un impedimento para utilizarlo [22].

- **Profibus**

Profibus es el protocolo de mayor popularidad, ya que soporta los fabricantes de PLC más importantes. La propiedad sobresaliente del protocolo es su alto nivel de seguridad de datos. El ciclo de mensajes es un aspecto relevante en el protocolo Profibus. El dispositivo maestro puede iniciar transacciones de mensajes, mientras que el dispositivo esclavo no transmiten por propia iniciativa, sino sólo a petición del maestro. La transacción de mensajes consiste en la petición de una trama desde el iniciador (casi siempre el

dispositivo maestro), seguida de una respuesta de *acknowledgment* del otro dispositivo. La respuesta debe llegar en un intervalo de tiempo, de no darse se repite la petición[23].

- **CAN**

Este protocolo está diseñado para realizar una transmisión en *broadcast*. La velocidad de transmisión es de hasta 1Mbps. Los datos se transmiten en mensajes de 0 a 8 bytes, se destinan 11 bits para asociarlo a cada mensaje como identificador. El identificador debe ser único ya que dos mensajes que se transmiten simultáneamente pueden venir de fuentes diferentes [24]. El identificador tiene 2 propósitos:

1. Asignar una prioridad al mensaje.
2. Permitir filtrar los mensajes en los receptores.

Los mensajes CAN no tienen destino explícito, ya que cualquier estación con un filtro apropiado puede recibir un mensaje. Actuadores, sensores y controladores hoy en día están diseñados para soportar el protocolo CAN. El estándar CAN estuvo originalmente creado para aplicaciones en sistemas automáticos, con el tiempo han llegado a ser útiles para sistemas de control [25].

2.4. Controlador PID.

Los controladores PID son usados constantemente en el control industrial, debido a su sencillez al ajustar parámetros sin la necesidad de conocer el modelo del sistema que se controla.
La acción P actúa de manera proporcional al error producido entre la señal de referencia y la salida actual, y ajusta la salida del controlador de acuerdo al tamaño del error. La acción I elimina el error de estado estacionario inducida por el efecto proporcional; esta acción, por el contrario, provoca una respuesta más lenta del controlador generando un aumento en el tiempo de establecimiento. La acción D se anticipa ante una futura tendencia de la señal de referencia, que resulta en una disminución del exceso de oscilaciones [26]. El controlador PID se ajusta mediante los parámetros k_c, T_i y T_c (correspondientes a las acciones proporcional-integral-derivativa, respectivamente), todos estos agrupados en la ecuación característica que modela un controlador PID (ecuación 2.1), y cuyo diagrama de bloques se observa en la Figura 2.15.

$$\mu(t) = k_c \left[e(t) + \frac{1}{T_i} \int e(t)dt + T_d \frac{d}{dt} [e(t)] \right] \tag{2.1}$$

Debido al control digital de la interfaz, el modelo del PID del dominio continuo se lo

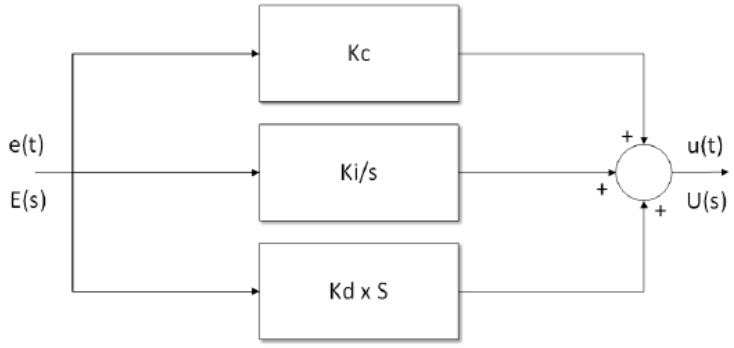


Figura 2.15: Diagrama de bloques del PID.
Fuente: Autores

traslada al dominio discreto. El desarrollo para la discretización de un controlador PID se muestra a continuación.

Se transforma la ecuación 2.1 al dominio de Laplace.

$$U(s) = k_c \left[E(s) + \frac{1}{sT_i} E(s) + sT_d E(s) \right] \tag{2.2}$$

Se utiliza un retenedor de orden cero (zero order hold (ZOH) por sus siglas en inglés) para la transformación al dominio Z, el cual contiene un periodo de muestreo, característico de los sistemas discretos.

$$s = \frac{1 - z^{-1}}{T} \tag{2.3}$$

Reemplazando tenemos:

$$U(z) = k_c \left[E(z) + \frac{1}{T_i \frac{1-z^{-1}}{T}} \left[E(z) + T_d \frac{1-z^{-1}}{T} E(z) \right] \right]$$

$$U(z) = k_c \left[E(z) + \frac{T}{T_i} \frac{1}{1-z^{-1}} E(z) + \frac{T_d}{T} (1 - z^{-1}) E(z) \right]$$

$$U(z) = k_c \left[\frac{E(z) - z^{-1}E(z) + \frac{T}{T_i}E(z) + \frac{T_d}{T}(1-z^{-1})^2 E(z)}{1-z^{-1}} \right]$$

Agrupando términos se tiene:

$$U(z) \quad - z^{-1}U(z) \quad =$$
$$k_c \left[E(z) - z^{-1}E(z) + \frac{T}{T_i}E(z) + \frac{T_d}{T}E(z) + \frac{T_d}{T}(1 - 2z^{-1} + z^{-2})E(z) \right]$$

$$U(z) \quad - z^{-1}U(z) \quad = \quad k_c\left[(1+\tfrac{T}{T_i}+\tfrac{T_d}{T})E(z) \quad - \quad (1+2\tfrac{T_d}{T})z^{-1}E(z) \quad + \tfrac{T_d}{T}z^{-2}E(z)\right]$$

Finalmente, a la salida del controlador digital se obtiene:

$$u_k = u_{k-1} + k_c\left[(1+\frac{T}{T_i}+\frac{T_d}{T})e_k - (1+2\frac{T_d}{T})e_{k-1} + \frac{T_d}{T}e_{k-2}\right] \tag{2.4}$$

En la ecuación 2.4, se observa que el tiempo de muestreo T, juega un papel fundamental en los sistemas de control digitales, puesto que este valor depende del tiempo de respuesta del sistema, la correcta elección de este valor determinara el comportamiento del controlador.

Capítulo 3

Desarrollo e implementación del prototipo

3.1. Componentes que intervienen en el sistema de control

3.1.1. Motor

Se implementa un control de velocidad al motor Maxon 353144 (Figura 3.1), para simular un proceso industrial el cual se monitoriza con el desarrollo de un sistema SCADA. Las especificaciones del motor se encuentran en el apéndice A.

Figura 3.1: Motor Maxon 353144.
Fuente: Autores

3.1.2. Driver del motor

Para controlar la velocidad del motor, se ajusta el voltaje y la corriente en la armadura del mismo. Mediante el uso de Ardumoto Shield, el cual es un controlador de cargas inductivas diseñado para Arduino, diseñado para el fácil manejo de motores DC (Figura 3.2).

Ardumoto posee un circuito integrado (CI) L298 puente H, capaz de tolerar cargas de

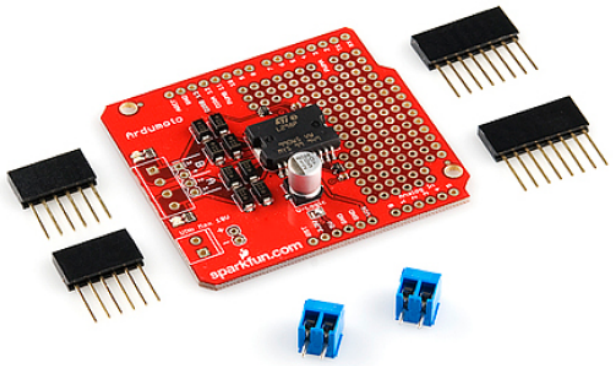


Figura 3.2: Ardumoto Shield.
Fuente: Autores

hasta 18V con corrientes de hasta 2A, además acepta niveles lógicos TTL, permitiendo habilitar o deshabilitar el dispositivo o efectuar cambios en el sentido de rotación del motor (Figura 3.3). El módulo utiliza la técnica de modulación por ancho de pulso

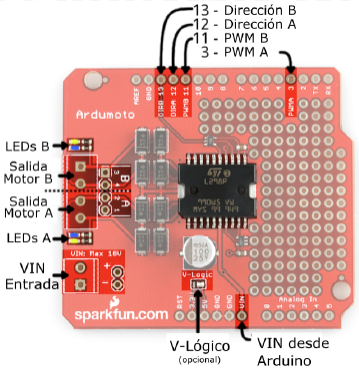


Figura 3.3: Distribución de pines del Ardumoto Shield.
Fuente: Autores

(pulse width modulation (PWM) por sus siglas en inglés) el cual consta posee una resolución de 8 bits.

Las características de funcionamiento y distribución de pines del Ardumoto se especifican en la Tabla 3.1.

Tabla 3.1: Distribución de pines Ardumoto shield.

Pines	Función Ardumoto	Control
3	PWM A	0 = apagado, 255 = máx. velocidad
11	PWM B	0 = apagado, 255 = máx. velocidad
12	Dirección A	Señal digital de control (uno/cero =derecha/izquierda)
13	Dirección B	Señal digital de control (uno/cero =derecha/izquierda)

3.1.3. Sensor de velocidad

Con el fin de añadir retroalimentación al sistema se utiliza un *encoder* de cuadratura incremental E6B2-CWZ3E (Figura 3.5). Este *encoder* posee una resolución de 1024 pulsos por rotación, los cuales se interpretan con el uso de un microcontrolador para determinar la dirección de giro y la velocidad de rotación del motor. La hoja de especificaciones técnicas de este dispositivo se encuentra detallado en el apéndice B.

Figura 3.4: Encoder incremental de cuadratura.
Fuente: Autores

3.1.4. Dispositivo de control

A nivel industrial, el uso de PLCs es común para el control de procesos. Existe un sin número de plataformas disponibles en el mercado, sin embargo, la mayor parte de estos dispositivos tienen un costo elevado, y ademas utilizan *software* propietario para la programación de los mismos.

Para la implementación del sistema de control, se hace uso de una plataforma de bajo costo, basada en Arduino, plataforma *open source*.

- **PLC MDunio Ethernet.**

M-Duino posee varias de las características de un PLC, esta plataforma esta desarrollada a base de Arduino Mega, razón por la cual posee 21 puertos entre entradas/salidas digitales y analógicas (Figura 3.5).

Figura 3.5: PLC M-Duino.
Fuente: Autores

M-Duino posee el protocolo de comunicación MODBUS, siendo este uno de los más utilizados por dispositivos industriales para interactuar con sistemas SCADA. Una de las principales características con las que cuenta esta plataforma es la capacidad de programarla directamente desde la IDE de Arduino, soportando un amplio número de librerías y secuencias de programación.

3.1.5. Interfaz maquina-humano (HMI)

Se hace uso de un dispositivo HMI para la manipulación del sistema de control del motor DC. Adicionalmente, se usa un PLC Siemens S7-1200, enlazado con una pantalla *touch KTP-600 Basic Color*. El objetivo de la interfaz es permitir la manipulación del sistema de control por parte de un operador a nivel de planta.
La programación de estos dispositivos se realiza con TIA PORTAL v12 de Siemens. La ventana principal del *software* se muestra en la Figura 3.6.

La Figura 3.7 muestra la implementan del lenguaje de programación KOP, para el primer bloque del programa del PLC. Donde cada contactor está asignado a una *tag*, la cual esta encargada de controlar el estado de las salidas del PLC asignadas.

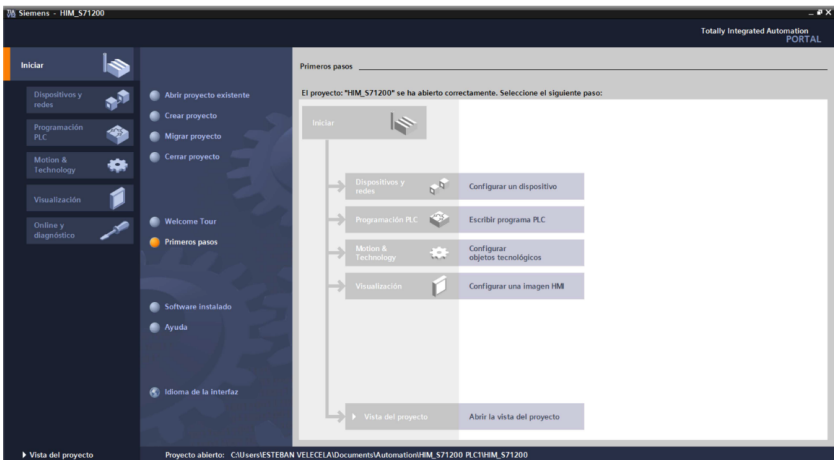


Figura 3.6: Interfaz TIA PORTAL 12.
Fuente: Autores

Por otra parte la Figura 3.8 muestra la implementación del lenguaje basado en diagrama de funciones, cada bloque cumple una tarea en específica, la cual recibe argumentos de entrada para procesarlos en su salida.

El primer bloque de programa, establece la manipulación del control de velocidad del motor. Donde las salidas asignadas por este PLC, son monitoreadas por parte del PLC M-Duino.

Mientras que el segundo bloque de programa, se usa para emitir una señal de referencia al PLC-Mduino con el uso de la salida analógica del PLC S7-1200. Dentro de este PLC se crea una variable de tipo entero cuyos valores fluctúan entre 0 a 6000, es así que en el primer diagrama de funciones de este bloque de programa se normaliza el valor de esta variable, para ser escalada posteriormente por un segundo diagrama de funciones, creando una relación de revoluciones por minuto (rpm) a voltaje a la salida del PLC S7-1200.

Con la configuración y programación del PLC lista, se procede a la configuración de la pantalla KTP-600 Basic Color PN, dentro del proyecto. La Figura 3.9 muestra la interfaz HMI desarrollada para el control de velocidad de el motor *DC*.

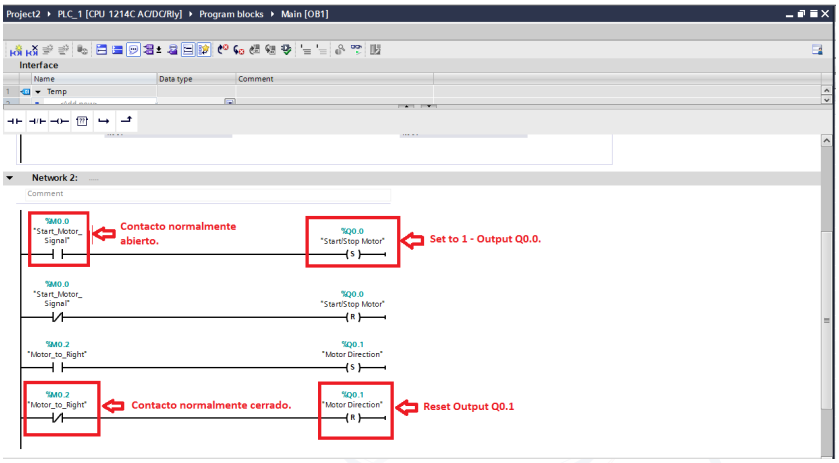


Figura 3.7: Primer bloque de programa con el esquema de contactores de TIA PORTAL 12.

Fuente: Autores

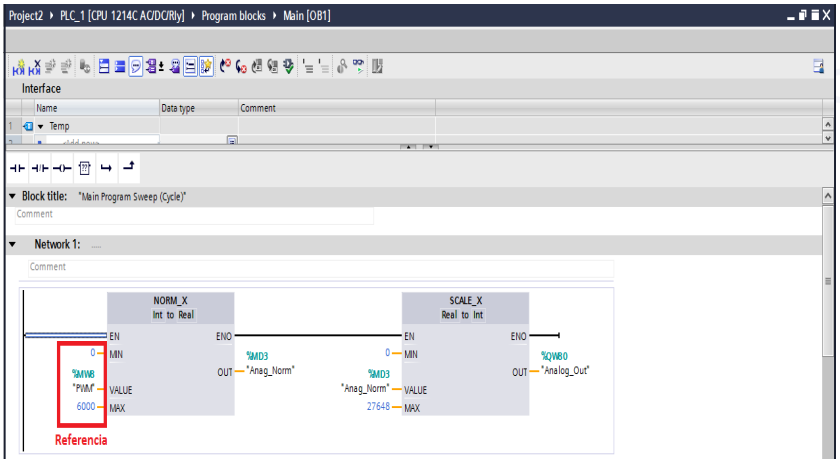


Figura 3.8: Segundo bloque de programa con el esquema de diagrama de funciones de TIA PORTAL 12.

Fuente: Autores

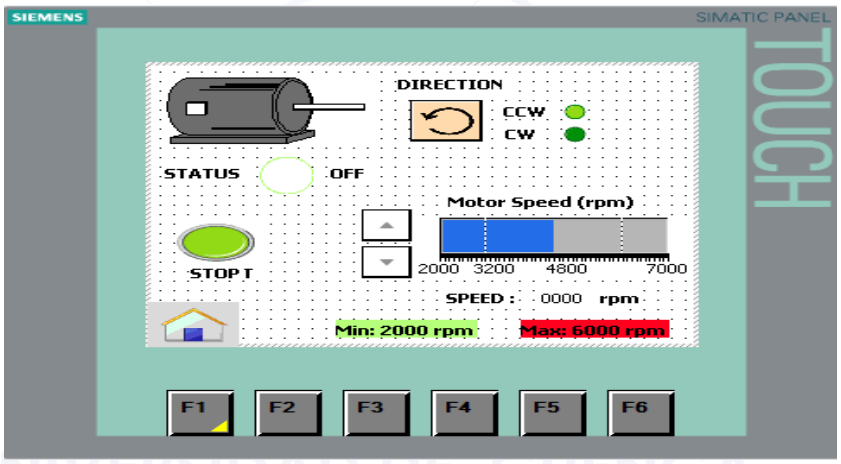


Figura 3.9: Interfaz HMI para el control de velocidad de un motor DC.
Fuente: Autores

3.2. Desarrollo del controlador de velocidad

3.2.1. Control de velocidad del motor DC

Las técnicas de control para motores DC, se usan para controlar la velocidad, el par y el suministro de potencia de estas máquinas.
En la industria, la alimentación de estos motores se efectúa de manera directa en sus devanados, con lo cual sus características de operación se mantienen inalterables al tener una tensión de entrada constante. Es decir, el motor trabaja en condiciones nominales cuando se alimenta con la tensión indicada en su hoja de especificaciones; con lo cual entrega potencia constante a la carga conectada a su eje.
La carga que se acople al eje del motor define el comportamiento del mismo, de tal manera que, para una carga liviana, la velocidad que entrega el motor será relativamente alta con un par de giro bajo. Por el contrario, si se dispone de una carga pesada que dificulte la movilidad del eje, la velocidad entregada por el motor será notablemente menor, incrementando el par según la exigencia de la carga, teniendo en cuenta las limitaciones que indica el fabricante. De esta forma, si una carga se mantiene constante, la operación del motor también lo hará, impidiendo controlar su velocidad.
Por estas razones existen casos en la industria donde se requiere el manejo de las características de operación de un motor.

3.2.1.1. Métodos que permiten variar la velocidad de un motor DC

Los métodos para variar la velocidad de un motor DC son:

- Ajustar el voltaje y la corriente aplicado al devanado del campo. Al aumentar el voltaje de campo, el motor desacelera.
- Ajustar el voltaje y la corriente aplicado a la armadura. Al aumentar el voltaje en la armadura el motor acelera.

3.2.2. Modelación del motor DC

La Figura 3.10 muestra el sistema de control de velocidad del motor, para el desarrollo del modelo, para esto se asume que la entrada del sistema es la fuente de voltaje U_A proveniente del *driver*, el cual es aplicado a la armadura del motor, la salida del sistema es la velocidad de rotación del eje $w(t)$. Posteriormente se asume un coeficiente de fricción viscosa casi nula.

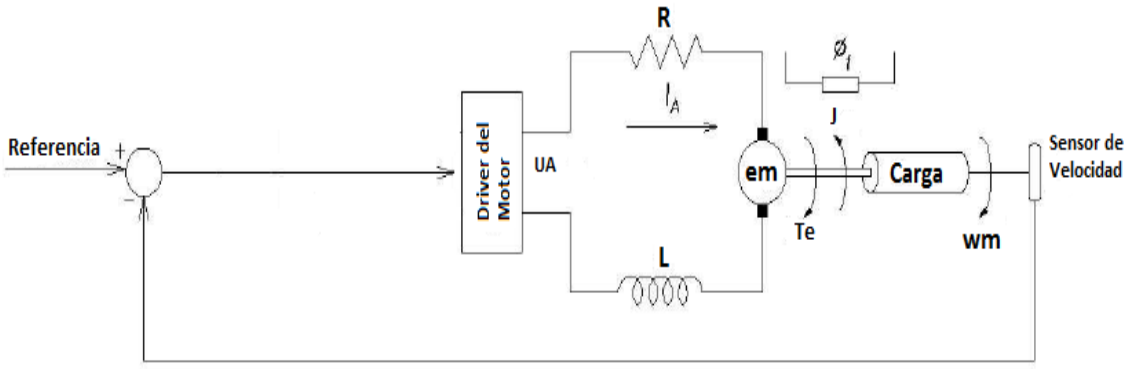


Figura 3.10: Sistema de control de velocidad para un motor DC.
Fuente: Autores

3.2.2.1. Sistema de ecuaciones

El torque generado por un motor DC, es proporcional a la corriente de la armadura y la fuerza del campo magnético. Para este modelo se asume que el campo magnético es constante, y además que el torque es solamente proporcional a la corriente de armadura i, por un factor constante k_t como se muestra en la ecuación 3.1.

$$T_e = K_t \times i \tag{3.1}$$

La fuerza contra-electromotriz (FCEM) e, es proporcional a la velocidad angular del eje por un factor constante K_e (ecuación 3.2).

$$e = K_e w_m \tag{3.2}$$

De la Figura 3.10, se derivan las siguientes ecuaciones basadas en la segunda ley de Newton (ecuación 3.3) y en la ley de voltajes de Kirchhoff (ecuación 3.4).

$$T_e = K_f w_m + \quad J\frac{dw_m}{dt} \quad + \quad T_L \tag{3.3}$$

$$U_A = Ri \quad + L\frac{di}{dt} + e \tag{3.4}$$

De las ecuaciones anteriores, se describen los siguientes términos:

- J= Inercia rotor
- K_f= Constante de fricción
- w_m = Velocidad angular

- T_L= Carga supuesta
- U_A= Voltaje en los terminales del motor
- R= Resistencia de armadura del motor

Reescribiendo las ecuaciones 3.3 y 3.4,

$$\frac{di}{dt} = \frac{U_A}{L} - \frac{Ri}{L} - \frac{K_e w_m}{L} \tag{3.5}$$

$$\frac{dw_m}{dt} = \frac{K_t i}{J} - \frac{K_f w_m}{J} - \frac{T_L}{J} \tag{3.6}$$

3.2.2.2. Función de transferencia

Aplicando la transformada de Laplace, las ecuaciones 3.5 y 3.6 pueden ser expresadas en términos de la variable de Laplace s.

$$si = \frac{U_A}{L} - \frac{Ri}{L} - \frac{K_e w_m}{L} \tag{3.7}$$

$$sw_m = \frac{K_t i}{J} - \frac{K_f w_m}{J} - \frac{T_L}{J} \tag{3.8}$$

Considerando $T_L \approx 0$, y despejando 3.8 en función de la corriente de armadura i,reemplazando en la ecuación 3.7 se obtiene:

$$\left(\frac{sw_m + \frac{K_f w_m}{J}}{\frac{K_t}{J}}\right)\left(s + \frac{R}{L}\right) = \frac{-K_e w_m}{L} + \frac{U_A}{L}$$

$$w_m\left(\frac{s^2 J w_m}{K_t} + \frac{sK_f}{K_t} + \frac{sJR}{K_t L} + \frac{K_f R}{K_t L} + \frac{K_e}{L}\right) = \frac{U_A}{L}$$

$$w_m\left(\frac{s^2 LJ + sLK_f + sRJ + K_f R + K_e K_t}{K_t}\right) = U_A \tag{3.9}$$

Despejando la ecuación 3.9, se obtiene la función de transferencia a lazo abierto, donde la velocidad de rotación se considera la salida del sistema, y el voltaje de armadura la entrada del sistema.

$$G_p(s) = \frac{w_m}{U_A} = \frac{K_t}{s^2 LJ + S(LK_f + RJ) + K_f R + K_e K_t}$$

Considerando $K_f \approx 0$, simplificamos la ecuación, de tal manera que la función de transferencia que describe el comportamiento de la planta es la siguiente:

$$G_p(s) = \frac{K_t}{s^2 LJ + SRJ + K_e K_t} \quad \left(\frac{rad/seg}{V}\right) \tag{3.10}$$

3.2.3. Desarrollo del controlador PID

El uso de un controlador digital PID se implementa para el control de velocidad del motor DC. A partir de la función de transferencia en tiempo continuo (ecuación 3.10), se efectúa el diseño de la ecuación de control en tiempo discreto. La selección del tiempo de muestreo T depende del tiempo de respuesta del sistema (Figura 3.11, para este caso se considera un tiempo de muestreo de 0.1 segundos.

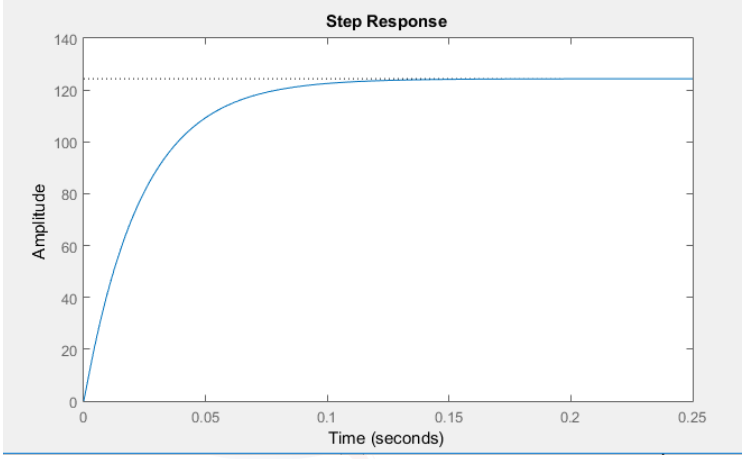


Figura 3.11: Respuesta lazo abierto del sistema.
Fuente: Autores

Los parámetros del motor se describen en la Tabla 3.2, de esta forma la función de transferencia a usarse es la siguiente:

$$G_p(s) = \frac{8.07\times10^{-3}}{(1.5042\times10^{-10})s^2+(1.54\times10^{-6})s+6.49\times10^{-5}}$$

Tabla 3.2: Parámetros del motor.

Parámetro	Valor	Unidad
Resistencia armadura (R)	1.41	Ω
Inductancia terminales (L)	0.138	mH
Constante de torque (K_t)	8.07×10^{-3}	Nm/A
Constante de velocidad	1180	rpm/V
Inercia rotor (J)	1.09×10^{-6}	Kgm^2

Con el uso de la función `c2d(Gp,0.1)` en MATLAB, se discretiza la función de transferencia, empleando el tiempo de muestreo seleccionado anteriormente, lo cual re-

sulta en la ecuación 3.11.

$$HG_p(z) = \frac{Y(z)}{U(z)} = \frac{0.0075z^{-2} + 122.6z^{-1}}{1 - 0.0145z^{-2}} \tag{3.11}$$

3.2.3.1. Sintonización del controlador PID

Para sintonizar el controlador se usa la herramienta *pidtool* de MATLAB, Figura 3.13, se fija un tiempo de estabilización de 3 segundos para el sistema sin sobre impulso. Como resultado los parámetros del controlador PID, se describen en la Tabla 3.3.

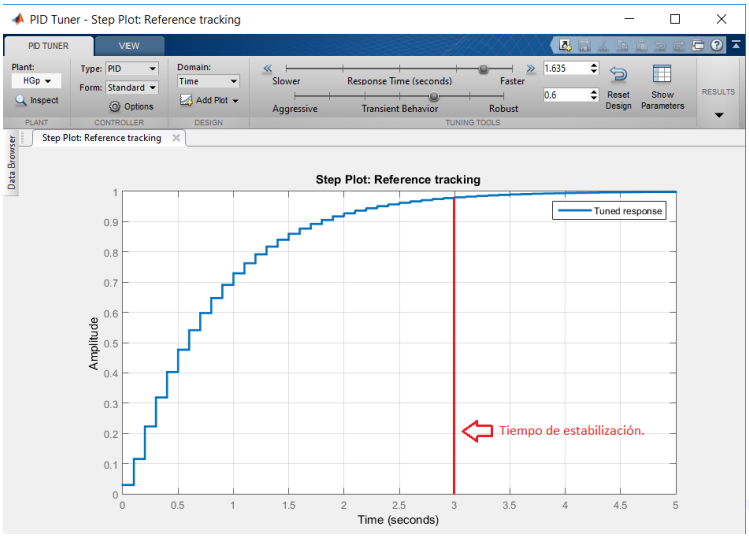


Figura 3.12: Sintonizacion del controlador PID usando MATLAB.
Fuente: Autores

Tabla 3.3: Parámetros del controlador PID.

Parámetros	Valor
K_p	0.00172427
T_i	0.1
T_d	0.025

3.2.4. Anti-windup PID

El control integrativo se aplica al sistema con el fin de eliminar el error de estado estacionario, la no linealidad presente en el sistema, produce un efecto indeseado conocido

como integrador *windup*, debido a cambios de referencia grandes, este efecto conlleva a que la respuesta del sistema presente sobre impulsos y tiempos de estabilización lentos.

Para evitar esta situación, una estrategia *anti-windup* se aplica al controlador, la cual consiste en re-calcular el término integral, cuando el controlador se satura. En general, el valor integral se reduce o se incrementa cuando la salida del controlador supera el límite superior u_{max} o cuando es menor que el límite u_{min}, retroalimentando la diferencia de la señal de control saturada y la no saturada, como se muestra en la Figura 3.13, donde T_t se denomina constante de tiempo de seguimiento y esta definido como $T_t = \sqrt{T_i T_d}$ para el caso de un PID y $T_t = T_i$ en caso de un controlador PI.

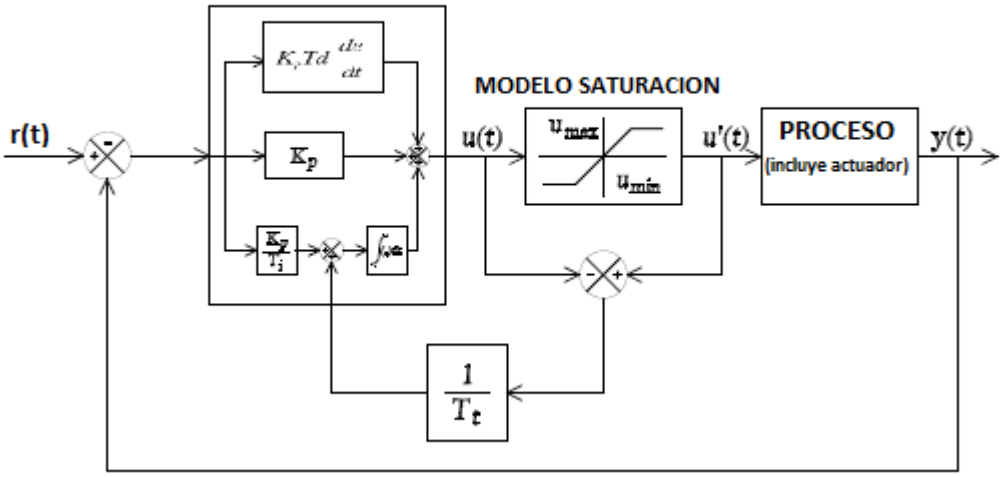


Figura 3.13: Esquema *anti-windup*.
Fuente: Autores

La ecuación del controlador PID que incluye la implementación de la técnica *anti-windup*, se define como a continuación donde $v_k = u'(t) - u(t)$.

$$u_k = u_{k-1} + K_P \left[e_k - e_{k-1} + \frac{T}{T_i} e_k + \frac{T_d}{T} (e_k - 2e_{k-1} + e_{k-2} \right] + \frac{T}{T_t} v_k \tag{3.12}$$

La Figura 3.14, simula la respuesta del controlador de velocidad para el motor DC,

con el uso de MATLAB.

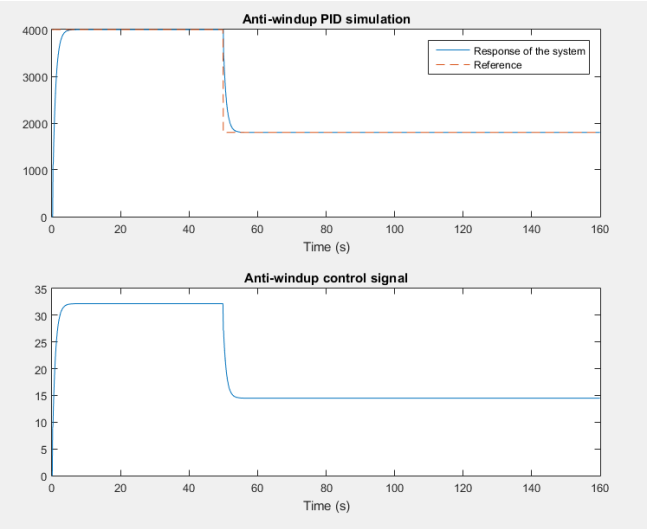


Figura 3.14: Simulación del PID con anti-windup.
Fuente: Autores

3.2.5. Implementación del controlador *anti-windup* PID en PLC

La rutina para el control de velocidad se programa dentro del PLC MDuino, esta plataforma es la encargada del control local de la planta.

Para establecer un tiempo de muestreo constante, el uso de una interrupción interna permite la ejecución de una rutina de código cada 0.1 segundos, la cual se genera a partir del contador de instrucciones Timer1 que posee el dispositivo.

Por otra parte para retroalimentar el sistema de control, se debe efectuar lecturas del sensor de velocidad cada periodo de muestreo. El *encoder* usado para esta tarea genera pulsos dependiendo de la posición del motor (Figura 3.15), para este caso genera 1024 pulsos por cada revolución.

La interpretación de la señal dentro del microcontrolador se efectúa con un proceso de conversión, que resulte en rpm. El proceso consiste en determinar la cantidad de pulsos generados por parte del *encoder*, con el uso de una interrupción externa, de tal manera que para cada flanco de subida generado por los pulsos de la señal del *encoder*, el microcontrolador genera una interrupción en el código, ejecutando una rutina en la

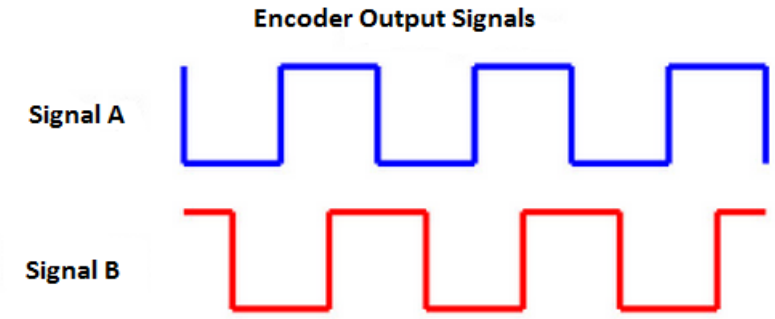


Figura 3.15: Señales generadas por el encoder.
Fuente: Autores

cual se almacena el número de pulsos generados durante un periodo de muestreo. De esta forma el cálculo de velocidad del motor se la efectúa de la siguiente manera:

$$rpm = \frac{num\_pulsos * N * 60}{1024} \tag{3.13}$$

Donde N, representa el número de muestras en un 1 segundo, para este caso $N = 10$. Una vez que se han establecidos estas rutinas, basta con la implementación de la ecuación 3.14 de control dentro de la subrutina que se ejecuta cada 0.1 segundos, para establecer la secuencia de control de velocidad.

$$u_k = u_{k-1} + 0.0017427\,[e_k - e_{k-1} + e_k + 0.025(e_k - 2e_{k-1} + e_{k-2}] + 20v_k \tag{3.14}$$

La Figura 3.16, muestra la respuesta del sistema de control implementado en el PLC MDuino, como se puede ver el tiempo de estabilización es cercano a los 3 segundos, de igual manera cuando se presenta cambios de referencia no existe sobre impulsos.

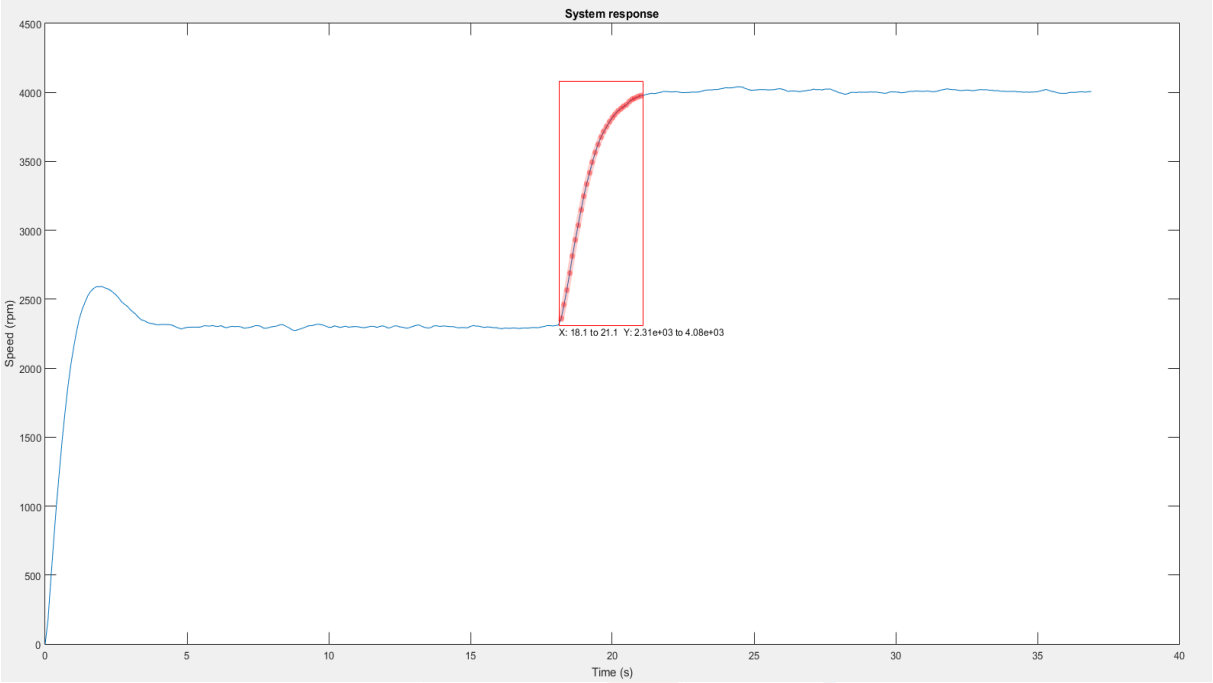


Figura 3.16: Respuesta del sistema con el controlador anti-windup PID.
Fuente: Autores

3.3. Sistema de detección de fallas

La Figura 3.17 muestra el esquema de un sistema de diagnóstico de fallas el cual consta un módulo de detección y diagnóstico de fallas. Las fallas pueden ocurrir tanto en el sistema como en actuadores y sensores. Además, el sistema incluye un mecanismo de reconfiguración que permite mantener las características iniciales del controlador una vez que se presenten fallas.

3.3.1. Detección de fallos

Los métodos de detección de fallos se basan en la generación de señales residuales. La diferencia entre la medida del sistema y su estimación se llama residuo. Los valores del residuo son cero o cercanos a cero si no ocurren fallos, en el caso de tener fallos en el sistema, los valores del residuo son diferentes de cero [27].

De manera analítica, las relaciones de redundancia del sistema se usan para crear una señal residual. El cálculo de esta señal residual puede realizarse a partir de las funciones de transferencia sobre una formulación de espacio de estados. El método de ecuaciones de paridad permite el cálculo de la señal residual basándose en modelos de

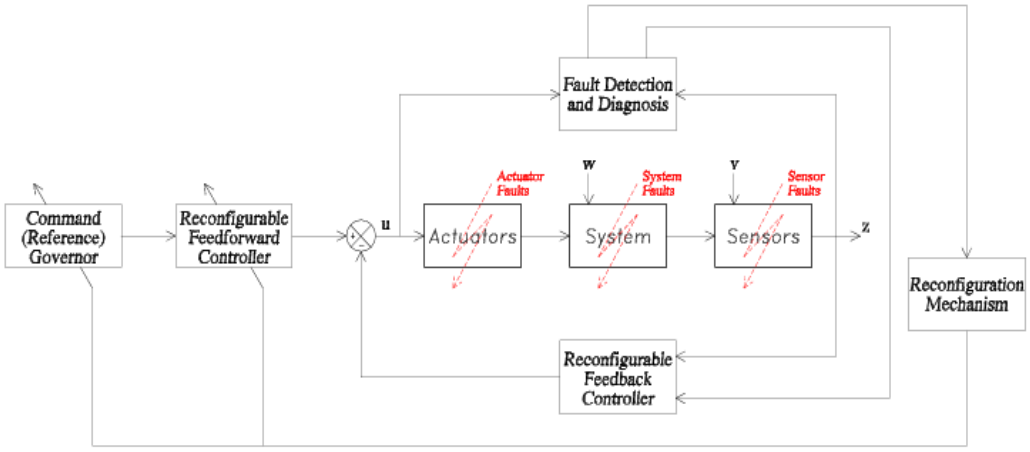


Figura 3.17: Esquema de un sistema de diagnóstico de fallas.
Fuente: Autores

espacio de estados [28].

3.3.2. Aproximación de las ecuaciones de paridad en tiempo discreto

Las ecuaciones en tiempo discreto para un proceso lineal con múltiples entradas y múltiples salidas están dadas por:

$$\begin{aligned} x(k+1) = &\quad Ax(k) + Bu(k) + Vv(k) + Lf(k) \\ y(k) = &\quad Cx(k) + Nn(k) + Mf(k) \end{aligned}$$

donde $v(k)$ y $n(k)$ son perturbaciones no medibles, mientras que $f(k)$ representa fallas aditivas tanto en la entrada y la salida $f_l(t)$ y $f_m(t)$.
Para simplificar el desarrollo del modelo en espacio de estado, no se considera fallas ni perturbaciones (ecuaciones 3.15 y 3.16).

$$x(k+1) = \quad Ax(k) + Bu(k) \tag{3.15}$$

$$y(k) = \quad Cx(k) \tag{3.16}$$

Se remplaza la ecuación 3.15 en 3.16 de tal manera que:

$$y(k+1) = \quad CAx(k) + CBu(k)$$

De la ecuación anterior se tiene que la salida para el siguiente instante de tiempo esta determinada por:

$$y(k+2) = \quad Cx(k+2) = \quad CAx(k+1) + CBu(k+1) = \\ CA^2x(k) + CABu(k) + CBu(k+1)$$

Donde para la q-ésima muestra $q <= m$ (donde m es el tamaño de la matriz de estados):

$$y(k+q) = \quad CA^qx(k) + CA^{q-1}Bu(k) + ... + CBu(k+q-1)$$

En este punto las ecuaciones de redundancia son generadas para diferentes instantes de tiempo. Donde las ecuaciones para una ventana de tiempo de longitud $q+1$ se suman obteniendo.

$$Y(k+q) = \quad Tx(k) + QU(k+q)$$

Cuando se produce un desplazamiento de q se tiene:

$$Y(k) = Tx(k-q) + QU(k) \tag{3.17}$$

Donde los vectores para la entrada y la salida están definidos como.

$$Y(k) = \begin{bmatrix} y(k-q) \\ y(k-q+1) \\ . \\ . \\ y(k) \end{bmatrix} \qquad U(k) = \begin{bmatrix} u(k-q) \\ u(k-q+1) \\ . \\ . \\ u(k) \end{bmatrix}$$

Las matrices del espacio de paridad se definen de la siguiente manera.

$$T = \begin{bmatrix} C \\ CA \\ CA^2 \\ . \\ . \\ CA^q \end{bmatrix} \tag{3.18}$$

$$Q = \begin{bmatrix} 0 & 0 & . & . & 0 \\ CB & 0 & . & . & . \\ CAB & CB & . & . & . \\ . & . & . & . & . \\ . & . & . & . & . \\ CA^{q-1}B & CA^{q-2}B & . & CB & 0 \end{bmatrix} \tag{3.19}$$

La ecuación 3.17 describe la entrada $(q+1)$ y las señales de salida. El estado inicial del vector $x(k-q)$ sobre el intervalo de longitud $(q+1)$, forma una redundancia temporal.
Dado que se desconoce el vector $x(k-q)$, la ecuación 3.17 es multiplicada por un vector w^T, con lo que se obtiene:

$$w^T Y(k) = \quad w^T T x(k-q) + w^T Q U(k) \tag{3.20}$$

La selección del vector w^T se da de tal manera que $w^T T = 0$ eliminando la dependencia de la entrada y los estados del sistema, dando como resultado la ecuación 3.21.

$$r(k) = \quad w^T Y(k) - w^T Q U(k) \tag{3.21}$$

La dimensión de w^T es $1 \times (q+1)r$ donde r es el número de salidas. Si el orden de A es m, la matriz T tiene orden $m \times (q+1)r$. Mediante la condición $w^T T = 0$ los m elementos de w^T son determinados. De cualquier manera los elementos restantes de w^T se eligen libremente. Más residuos pueden ser determinados multiplicando la ecuación 3.17 por una matriz W con la condición de que:

$$WT = 0 \tag{3.22}$$

De esta manera el vector residuo resulta en:

$$r(k) = \quad W\left[Y(k) - QU(k)\right] \tag{3.23}$$

3.3.3. Desarrollo del método de espacio de paridad discreto para el modelo del motor DC

La representación de un sistema, en el espacio de estados esta definido de la siguiente manera.

$$\dot{x(t)} = Ax(t) + Bu(t)$$
$$y(t) = Cx(t) + D$$

A partir de las ecuaciones 3.5 y 3.6 que representan el modelo del sistema en tiempo continuo, se define los estados y entradas de la matriz de estados.

$$x(t) = [i \quad w_m]^T$$
$$u(t) = [U_A]$$

De tal manera que la matriz de estados, que representa el comportamiento del sistema de control es el siguiente.

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{bmatrix} = \begin{bmatrix} \frac{-R}{L} & \frac{-K_e}{L} \\ \frac{K_T}{J} & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(t) \tag{3.24}$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix}$$

Para desarrollar un modelo con el uso las ecuaciones en espacio de paridad, se consideran las siguientes fallas en el sistema de control:

1. Un cambio en la resistencia de la armadura del motor R, este cambio representa un falla eléctrica en el motor.

$$R = R_o + \triangle R$$

2. Cambio en la constante de par del motor K_t. Lo cual se relaciona a una falla mecánica o sobre carga en el motor.

$$K_t = K_{to} + \triangle K_t$$

A partir de la matriz 3.24, se incorporan las fallas descritas al sistema, considerándolas como fallas aditivas en el modelo. De esta forma tenemos:

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{bmatrix} = \begin{bmatrix} -\frac{R_o+\triangle R}{L} & -\frac{K_{to}+\triangle K_t}{L} \\ \frac{K_{to}+\triangle K_t}{J} & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(t) \tag{3.25}$$

$$\begin{bmatrix} \dot{x_1}(t) \\ \dot{x_2}(t) \end{bmatrix} = \begin{bmatrix} -\frac{R_o}{L} & -\frac{K_{to}}{L} \\ \frac{K_{to}}{J} & 0 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} \frac{1}{L} \\ 0 \end{bmatrix} u(t) + \begin{bmatrix} -\frac{\triangle R}{L} x_1(t) & -\frac{\triangle K_t}{J} x_2(t) \\ \frac{\triangle K_t}{J} x_1(t) & 0 \end{bmatrix} \tag{3.26}$$

Reacomodando la representación en espacio de estados se obtiene la ecuación 3.27, la cual modela las fallas aditivas del sistema.

$$\begin{bmatrix}\dot{x_1}(t)\\ \dot{x_2}(t)\end{bmatrix} = \begin{bmatrix}-\frac{R_o}{L} & -\frac{K_{to}}{L}\\ \frac{K_{to}}{J} & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}\frac{1}{L}\\ 0\end{bmatrix}u(t) + \begin{bmatrix}-\frac{1}{L}\\ 0\end{bmatrix}\underbrace{\triangle R x_1}_{f_1(t)} + \begin{bmatrix}-\frac{1}{L}\\ 0\end{bmatrix}\underbrace{\triangle K_t x_2}_{f_2(t)} + \begin{bmatrix}0\\ \frac{1}{J}\end{bmatrix}\underbrace{\triangle K_t x_1}_{f_3(t)} \tag{3.27}$$

Las matrices de coeficientes de fallas $f_1(t)$ y $f_2(t)$ de la ecuación 3.27 son idénticas, por lo que se encuentran acopladas. Para la expresión descrita anteriormente se incluyen las fallas como entradas dentro del sistema, del tal manera que el modelo generado considere cada una de los errores que pueden presentarse en el modelo de control descrito.

$$\begin{bmatrix}\dot{x_1}(t)\\ \dot{x_2}(t)\end{bmatrix} = \begin{bmatrix}-\frac{R_o}{L} & -\frac{K_{to}}{L}\\ \frac{K_{to}}{J} & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}\frac{1}{L} & -\frac{1}{L} & -\frac{1}{L} & 0\\ 0 & 0 & 0 & \frac{1}{J}\end{bmatrix}\begin{bmatrix}u(t)\\ f_1(t)\\ f_2(t)\\ f_3(t)\end{bmatrix} \tag{3.28}$$

Debido a que se usa la técnica de detección de espacio de paridad discreto, es necesario discretizar la ecuación de estados del sistema. Incluyendo los parámetros del motor de la tabla 3.2, se reescribe la ecuación 3.28.

$$\begin{bmatrix}\dot{x_1}(t)\\ \dot{x_2}(t)\end{bmatrix} = \begin{bmatrix}-10217.4 & -58.49\\ 7403.7 & 0\end{bmatrix}\begin{bmatrix}x_1(t)\\ x_2(t)\end{bmatrix} + \begin{bmatrix}7246.4 & -7246.4 & -7246.4 & 0\\ 0 & 0 & 0 & 917431.2\end{bmatrix}\begin{bmatrix}u(t)\\ f_1(t)\\ f_2(t)\\ f_3(t)\end{bmatrix} \tag{3.29}$$

Se discretiza la ecuación 3.29 empleando un tiempo de muestreo T=0.1 segundos. Empleando la función de MATLAB `c2d` se obtienen las siguientes expresiones.

$$\begin{bmatrix}x_1(k+1)\\ x_2(k+1)\end{bmatrix} = \begin{bmatrix}-5.96\times10^{-5} & -8.19\times10^{-5}\\ 0.010137 & 0.001425\end{bmatrix}\begin{bmatrix}x_1(k)\\ x_2(k)\end{bmatrix} + \begin{bmatrix}0.001015 & -0.001015 & -0.001015 & -122.1\\ 122.1 & -122.1 & 122.1 & 917431.\end{bmatrix} \tag{3.30}$$

$$\begin{aligned} x(k+1) &= Ax(k) + Bu(k) + E_1 f_1(k) + F_1 f_2(k) + G_1 f_3(k)\\ y(k) &= Cx(k) + Du(k) + E_2 f_1(k) + F_2 f_2(k) + G_2 f_3(k)\end{aligned}$$

Una vez discretizado el modelo que contiene fallas, se calculan las matrices 3.18 y

3.19 con el modelo en espacio de estados que contiene fallas.

Con el fin de reducir los tiempos de procesamiento en el dispositivo en el cual se implementa la detección de fallos, se escoge un retardo en tiempos de muestreo de $q = 1$, des esta manera se tiene:

$$Q = \begin{bmatrix} 0 & 0 \\ 122.15 & 0 \end{bmatrix} \qquad T = \begin{bmatrix} 0 & 1 \\ 0.0104 & 0.0143 \end{bmatrix}$$

$$Y(k) = \begin{bmatrix} y(k-1) \\ y(k) \end{bmatrix}$$

$$U(k) = \begin{bmatrix} u(k-1) \\ u(k) \end{bmatrix} \tag{3.31}$$

Para la definición de un residuo que permita la detección de fallas, se procede con el cálculo de la matriz W. Esta matriz debe cumplir la condición 3.22, para eliminar la dependencia de condiciones iniciales.

$$W = \begin{bmatrix} -0.0143 & 1 \end{bmatrix} \tag{3.32}$$

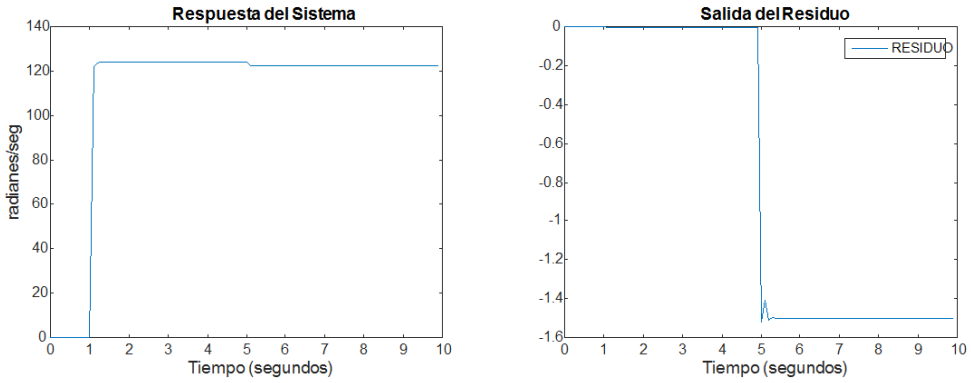
3.3.4. Simulación del método de espacios de paridad para la detección de fallas

A partir de la ecuación 3.21, se simula el comportamiento del modelo para la detección de fallos, evaluando el comportamiento del residuo en MATLAB. Se introducen fallas al sistema para determinar si el modelo permite detectarlas.

La Figura 3.31 (a) muestra la respuesta del sistema cuando se produce un cambio en la resistencia de la armadura del motor, simulando una falla eléctrica.
Por otra parte la Figura 3.31 (b) muestra el comportamiento del residuo ante la presencia de fallas en el sistema, el valor del residuo se mantiene en 0 ante la ausencia, pero este difiere de 0 cuando una falla ocurre en el sistema, lo que permite la detección de una o varias fallas en el sistema de control.

3.3.5. Implementación de ecuaciones de paridad para detección de fallos, en una plataforma micro-controlada

La generación de un residuo a partir del método de espacios de paridad, para la detección de fallas, se encuentra alojado en el PLC M-Duino.



(a) Respuesta del sistema.

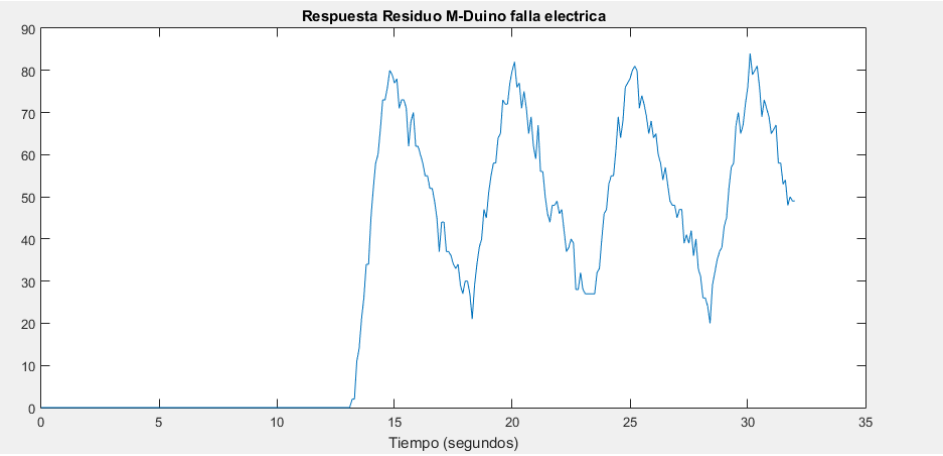
(b) Respuesta de residuo ante presencia de fallas.

Figura 3.18: Detección de fallos simulado en MATLAB.
Fuente: Autores

Para permitir la implementación de la ecuación 3.21, en el programa del PLC, es necesario conocer los vectores de redundancia que intervienen en esta ecuación. Donde $Y(k)$, representa la salida del sistema, en distintos tiempos de muestreo. La asignación de valores a este vector, se efectúa mediante la lectura del *encoder*.

Por otra parte el vector $U(k)$, representa la entrada del sistema, es decir el voltaje que proviene del *driver* que alimenta al motor. La medición de este voltaje se efectúa con el uso de las entradas analógicas del PLC. La asignación de una entrada analógica para cada terminal proveniente del *driver*, permite obtener un diferencial de voltaje, con lo cual se obtiene una lectura aproximada del voltaje en la armadura del motor.

La Figura 3.19 (a), muestra el comportamiento del residuo generado en el PLC MDuino con el uso de las ecuaciones de paridad. Como se puede observar el residuo difiere de 0, ante un cambio en la resistencia del motor, los valores generados por el residuo son considerablemente altos ante la presencia de este tipo de falla. Por otro lugar en la Figura 3.19, se observa el comportamiento del residuo cuando existe la presencia de una falla mecánica en el motor.



(a) Presencia de una falla eléctrica en el motor.

(b) Presencia de falla mecánica en el motor.

Figura 3.19: Detección de fallas del sistema implementado en un PLC.
Fuente: Autores

3.4. Desarrollo e implementación del *software* de supervisión y control

El sistema se desarrolla en su totalidad con el lenguaje de programación Python versión 3.4.

3.4.1. Inicio de sesión

Para hacer uso del sistema SCADA, los clientes se registran en una ventana de inicio de sesión (Figura 3.20). Esto permite establecer un control de seguridad mediante la gestión de usuarios.

La gestión de los usuarios se realiza por medio de una base de datos en MySQL. El

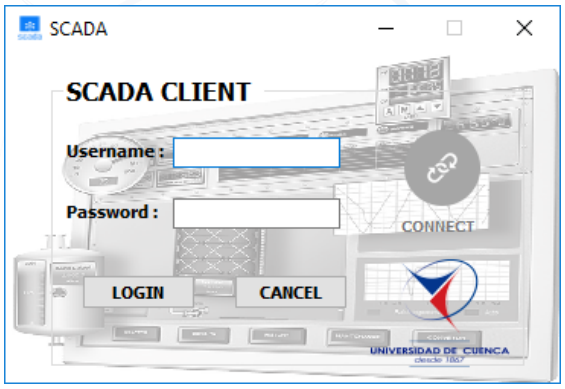


Figura 3.20: Ventana de inicio de sesión.
Fuente: Autores

servidor de la base de datos está instalado en un Raspberry Pi, en la cual se encuentra el servidor OPC-UA.

3.4.1.1. Instalación de MySQL en Raspberry Pi

Para la instalación de la base de datos se ejecuta la siguiente línea de comando.

$$sudo \quad apt-get \quad install \quad mysql-server \quad python-mysqldb$$

Durante la instalación se asigna una contraseña a la cuenta *root*.

Una vez instalada la base de datos se crean cuentas de usuario con todos los privilegios. Estas cuentas pertenecen a los usuarios SCADA que hacen uso de los datos alojados en el servidor. Para esto se ejecutan las líneas de comando que se muestran a continuación:

mysql> CREATE USER 'usuario'@'direccionIP' IDENTIFIED BY 'password';
mysql> GRANT ALL PRIVILEGES ON temps.* TO 'usuario'@'direccionIP'

```
mysql> FLUSH PRIVILEGES;
```

La tabla de la base de datos que gestiona los usuarios se muestra en la Figura 3.21. La ventana de inicio de sesión se rige bajo el funcionamiento de las siguientes clases:

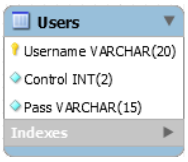


Figura 3.21: Tabla de gestión de usuarios.
Fuente: Autores

3.4.1.2. Clase LoginWindow

Esta clase tiene dos objetivos:

1. Validar los usuarios del sistema SCADA
2. Realizar la conexión con el servidor OPC-UA

- **Validar usuarios**

Para validar los usuarios se realiza una consulta a la base de datos. La ejecución de esta tarea se efectúa con el uso del método *AttempLogin*(). El método recibe dos parámetros: nombre y contraseña. El conector de MySql para Python se utiliza en este método para establecer la conexión con el servidor de la base de datos.

- **Conexión con el servidor OPC-UA**

El método utilizado para esta tarea es el *connect_clicked*(). La librería de FreeOPCUA disponible en *github* proporcpiona una clase para el cliente OPC-UA desarrollada en C++.
La clase del cliente lleva como nombre *opcua.client.client.Client*. La clase citada proporciona el método *connect*(), el cual recibe como parámetro la *Uniform Resource Identifier* (URI) del servidor.
Este método permite la creación y habilitación de un usuario dentro del servidor con las especificaciones OPC.

3.4.2. *Main Window*

Esta ventana se encarga de visualizar y revisar alarmas (figura 3.22). El método que realiza esta función tiene el nombre de *check_alarm*().

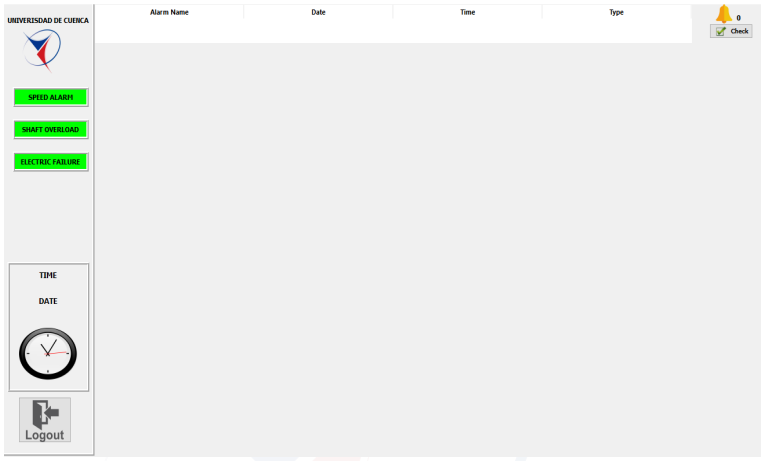


Figura 3.22: *Main Window*.
Fuente: Autores

El método *check_alarm*() permite al usuario registrar en la base de datos las alarmas que han sido resueltas. La Figura 3.23 muestra la tabla de registro de alarmas.

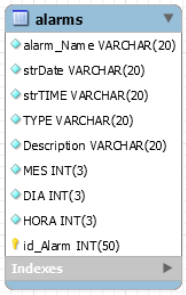


Figura 3.23: Tabla de registro de alarmas.
Fuente: Autores

Adicionalmente, esta ventana realiza dos tareas complementarias. Una de ellas es la de visualizar la fecha y hora actual, el método que se encarga de esta tarea es *hora*(). La otra tarea complementaria es la de desconectar al usuario del servidor OPC-UA. El método implementado para esto es *closeWindow*(), que ejecuta a su vez el método *disconnect*() de la librería FreeOPCUA.

3.4.3. *Monitor Window*

La Figura 3.24 muestra la ventana que realiza el monitoreo de las variables de la planta. Esta labor es la más importante en este apartado, pero no la única. Registro de alarmas que no están revisadas, control de la planta, visualización de alarmas, graficar en tiempo real la velocidad del motor, señal de control y residuo básico son otras de las tareas que desempeña esta ventana.

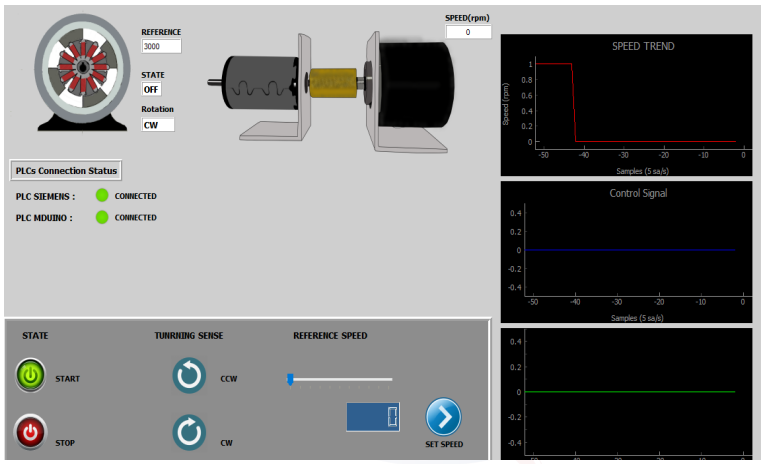


Figura 3.24: *Monitor Window.*
Fuente: Autores

3.4.3.1. Monitoreo de variables

Estado del motor, sentido de giro, velocidad, velocidad de referencia, monitoreo del residuo, señal de control, y estados de conexión de los dispositivos de control son variables que permiten supervisar la planta. El monitoreo en tiempo real de estas variables es responsabilidad de esta ventana, el método encargado de esta labor es $readTagsThread()$.

$$readTagsThread().$$

Este método es un hilo que permite conocer los valores de las variables con un tiempo de adquisición de $140ms$. Las variables están alojadas en el servidor, cada una de éstas se relaciona con un nodo. Estos nodos tienen identificadores únicos otorgados por el servidor. La librería del cliente OPC-UA ofrece dos métodos que facilitan la tarea

de adquisición de datos, estos son: *get_node*() y *get_value*.
El método *get_node*() recibe como parámetro el identificador del nodo correspondiente a una variable, accediendo al registro en el servidor de la misma. El método *get_value* toma el valor de la variable.

3.4.3.2. Tareas complementarias

La tarea de monitoreo no es la única funcionalidad de la ventana. La Tabla 3.4 resume los métodos complementarios con su respectiva función. El método implementado para que el motor realice las tareas deseadas es *set_value*() de la librería cliente OPC-UA. Esta herramienta se usa por parte de los 5 primeros métodos de la Tabla 3.4.

Tabla 3.4: Métodos complementarios *Monitor Window*.

Método	Función
stop	Detener el motor
start	Iniciar el motor
ccw()	Giro de motor antihorario
cw()	Giro de motor horario
sendSpeed()	Cambio de velocidad
waning_alarm()	Regitrar en la base de datos una alarma de precaución
electric_alarm()	Regitrar en la base de datos una alarma crítica no revisada
shaft_alarm()	Regitrar en la base de datos una alarma de sobrecarga en el eje no revisada
updatePlots()	Actualización de graficas en tiempo real
girar()	Animación de rotor

3.4.4. *Trends Window*

Esta ventana permite graficar tendencias de: velocidad del motor, señal de control y el residuo (Figura 3.25).

3.4.4.1. Método *plotThread*

El método es un hilo que a partir del día, hora inicial y hora final de la tendencia, realiza una consulta al servidor para obtener los datos históricos de la velocidad del motor, señal de control y residuo. Estos datos se guardan en una tupla para posteriormente graficarlos. La función principal de esta ventana se realiza gracias a este método.

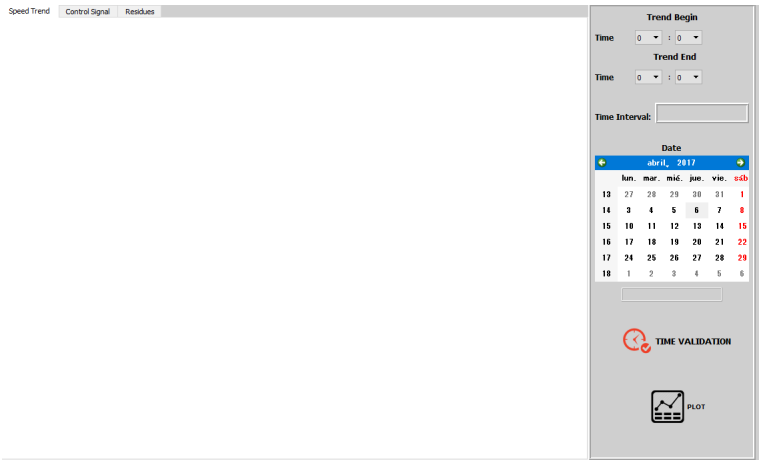


Figura 3.25: *Trends Window.*
Fuente: Autores

3.4.4.2. Métodos complementarios

El usuario tiene que validar los datos ingresados para evitar errores en la obtención de los datos. La Tabla 3.5 muestra los métodos utilizados para determinar si los datos ingresados son correctos.

Tabla 3.5: Métodos complementarios *Trends Window.*

Método	Función
validateData()	Mostar mensaje de error si se detecta datos erróneos
showDate()	Mostrar el día y mes seleccionado en el calendario
Interval()	Mostrar al usuario el intervalo de tiempo seleccionado

3.4.5. *Historial Window*

El método accede a la base de datos para obtener el historial de acciones y alarmas. La Figura 3.26 muestra la ventana de historiales.

3.4.5.1. Métodos destacados

El sistema SCADA desarrollado permite tener un control de los cambios realizados en las variables de control en un determinado día y hora. Las variables sujetas a cambios son: el estado, sentido de giro y velocidad del motor DC.

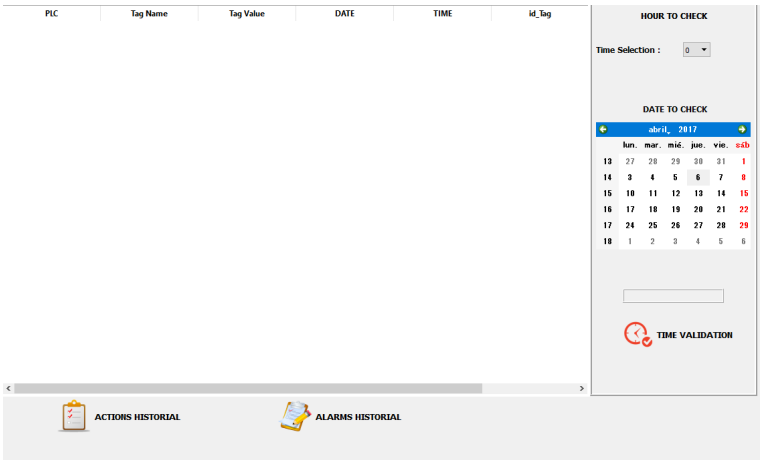


Figura 3.26: *Historial Window.*
Fuente: Autores

El método *actionsHistorial*() extrae de la base de datos las acciones realizadas en una fecha determinada. Estos datos se almacenan temporalmente en una tupla para luego listarse en una tabla.
Además de listar las acciones de control, esta ventana lista las alarmas generadas en un día a una hora específica. *alarmsHistorial*() es el método que realiza esta tarea.

3.4.5.2. Métodos complementarios.

El usuario elige el día, mes y hora en la que desea observar las acciones y alarmas suscitadas. Los métodos de validación que usa la ventana de tendencias se replican en esta ventana. En esta ventana no se calcula un intervalo de visualización, razón por la cual existen solo dos métodos complementarios los cuales se listan en la Tabla 3.6.

Tabla 3.6: Métodos complementarios *Historial Window*

Método	Función
validateData()	Mostar un mensaje de error en caso de que los datos ingresados sean erróneos
showDate()	Mostrar el día y mes seleccionado en el calendario

3.5. Rediseño y actualización de *software* del servidor OPC-UA

En base al servidor OPC-UA desarrollado en [4], se implementan varias mejoras considerando propuestas mencionadas en su trabajo futuro. Una de las principales limitaciones que presentaba el servidor, es la capacidad de establecer comunicación solamente con dispositivos de la familia Siemens, lo que reduce el rango de aplicaciones del servidor.

La mejora que se propone en este trabajo, es la capacidad de inclusión de varios protocolos de comunicación usados a nivel industrial, ampliando las capacidades de comunicación del servidor, con cualquier dispositivo.

3.5.1. Inclusión de protocolos de comunicación

El método $CrearObj()$ de la clase $UaServer()$ del servidor, la cual gestiona la conexión de un nuevo dispositivo, limitaba la capacidad de crear canales de comunicación con otros protocolos que difieran del protocolo S7 de Siemens.

En el presente trabajo, se reestructura las funcionalidades de este método, en donde se gestiona la adición de *drivers* correspondientes a cada dispositivo que se conecte al servidor.

La Figura 3.27 muestra la ventana de conexión de dispositivos dentro del servidor, la cual permite permite elegir el protocolo de comunicación que posee el dispositivo a conectarse.

El estado actual del servidor, permite la gestión de librerías en *python*, para la adición de protocolos de comunicación industrial disponibles en este lenguaje. El presente trabajo comprueba las capacidades del servidor para comunicarse con dispositivos que usan el protocolo MODBUS, para lograr esta tarea se hace uso de la librería *pymodbus*, distribuida por *github*. La librería soporta la mayoría de funciones de código MODBUS, descritas en la Figura 3.28.

Cabe recalcar que en el trabajo presente, únicamente se realiza la adición del protocolo de comunicación MODBUS, ya que para otro tipo de protocolos de comunicación no se dispone del *hardware* para efectuar pruebas de funcionamiento. Sin embargo, basta con importar librerías escritas en *python* que contengan protocolos de comunicación

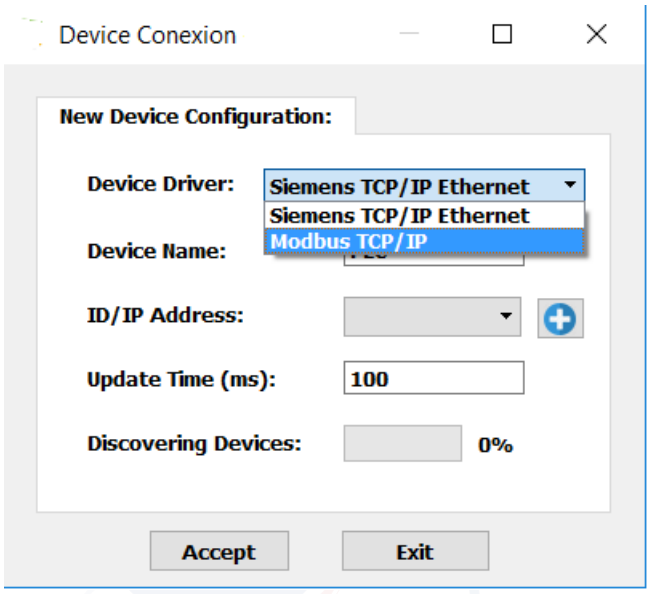


Figura 3.27: Conexión nuevo dispositivo con el servidor.
Fuente: Autores

industrial, para permitir que el servidor pueda comunicarse incluso con más dispositivos.

Por ejemplo la librería *pycan* provee un soporte CAN, brindando varias abstracciones para varios dispositivos, que puedan enviar o recibir mensajes en un *bus* tipo CAN. El método *CrearObject*() gestiona de manera sencilla cada una de las librerías de comunicación, permitiendo que la conexión de un dispositivo al servidor sea lo más sencillo y eficiente.

3.5.2. Conexión y desconexión de dispositivos

Una vez que un dispositivo ha sido configurado de manera exitosa dentro de la interfaz del servidor, la interfaz se encarga de crear automáticamente variables asociadas a cada dispositivo, con el fin de efectuar un monitorio del estado de cada dispositivo de manera remota.

La Figura 3.29 muestra lo anteriormente descrito, donde la carpeta *ConexionStatus* posee *tags* de tipo *booleano* que permite verificar si un dispositivo se encuentra activo o inactivo en el servidor.

Por otra parte, el servidor OPC-UA permite efectuar una conexión o desconexión manual de cada dispositivo, con el fin de que solo los dispositivos que deseen ser mo-

Function type			Function name	Function code
Data Access	Bit access	Physical Discrete Inputs	**Read Discrete Inputs**	2
		Internal Bits or Physical Coils	**Read Coils**	1
			Write Single Coil	5
			Write Multiple Coils	15
	16-bit access	Physical Input Registers	**Read Input Registers**	4
		Internal Registers or Physical Output Registers	**Read Multiple Holding Registers**	3
			Write Single Holding Register	6
			Write Multiple Holding Registers	16
			Read/Write Multiple Registers	23
			Mask Write Register	22
			Read FIFO Queue	24
	File Record Access		Read File Record	20
			Write File Record	21

Figura 3.28: Funciones de código MODBUS.
Fuente: Autores

nitoreados establezcan una conexión constante con el servidor, permitiendo optimizar recursos (Figura 3.30).

La capacidad de reiniciar conexiones con dispositivos que hayan presentado fallas dentro del sistema y no puedan establecer una conexión con el servidor de manera automática, se lo puede efectuar con esta tarea de igual manera.

3.5.3. Creación de *tags* en el servidor OPC-UA

Al momento de crear una *tag* el servidor identifica el protocolo usado por el dispositivo, donde se muestra una ventana de configuración acorde a cada protocolo de comunicación o tipo de dispositivo.

Así la Figura 3.31 (a), muestra la ventana de creación de *tags* para dispositivos de la familia Siemens que usen el protocolo S7, mientras que en la Figura 3.31 (b), se muestra la ventana de creación de *tags*, donde se especifican los parámetros del protocolo MODBUS.
De esta manera si se efectúa la inclusión de otro tipo de comunicación, se deberá crear ventanas acordes que contengan las propiedades de dicho protocolo.

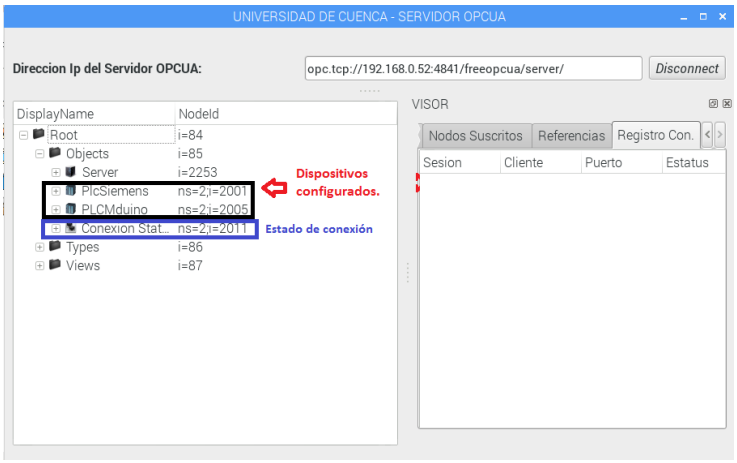


Figura 3.29: PLCs configurados en la interfaz del servidor.
Fuente: Autores

3.5.4. Lectura y escritura de variables en tiempo real.

Una de las mejoras significativos del servidor, es la capacidad de lectura de variables en tiempo real de los dispositivos conectados. Se optimizó el programa, reduciendo las líneas de código que realizan esta tarea.

El método $actualizarVarObj()$ dentro de la clase $UaServer()$, permite la ejecución de varios procesos para monitorizar el comportamiento de cada dispositivo, creando un hilo para cada dispositivo en un tarea de lectura, mientras que se ejecuta la creación un nuevo proceso encargado para las tareas de escritura, lo que permite al programa ocuparse de cada tarea por separado, permitiendo que la actualización de variables sea en tiempo real.

3.5.5. Gestión de usuarios.

Como medida de seguridad, se implementa la gestión y control de usuarios que tienen acceso a las variables del servidor. Debido a que el servidor carecía de la implementación de los protocolos de seguridad implantados por el protocolo OPC. De esta forma para que un cliente acceda al servidor es necesario un nombre de usuario y contraseña de tal manera que el servidor permita la creación de una sesión para cada cliente que se encuentre registrado en la base de datos.

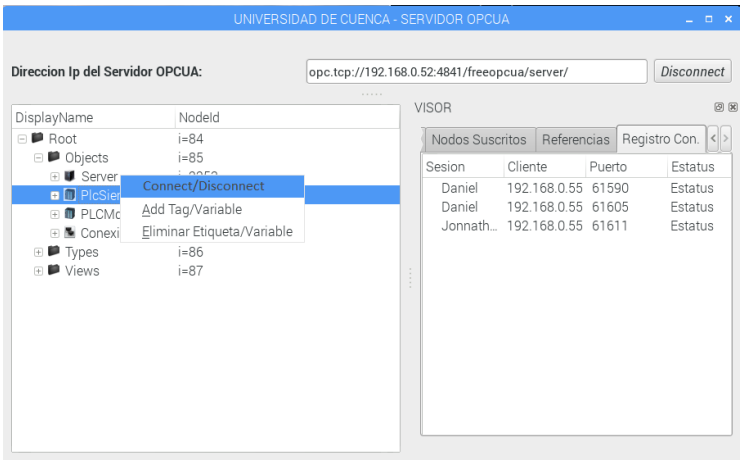
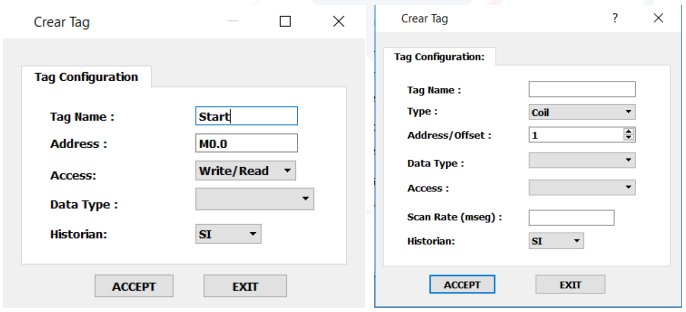


Figura 3.30: Conexión o desconexión de dispositivos en el servidor.
Fuente: Autores



(a) Ventana para dispositivos Siemens.

(b) Ventana para dispositivos con protocolo MODBUS.

Figura 3.31: Creación de tags en el servidor OPC-UA.
Fuente: Autores

3.5.6. Historiador

Se reestructura la forma en que se almacenan los datos de variables correspondientes a cada dispositivo que se encuentra conectado con el servidor. La Figura 3.32 muestra la tabla de la base datos, donde cada variable o *tag* esta relacionada con el PLC al que pertenece esta variable, de esta manera no puede existir dos variables con el mismo identificador del servidor OPC-UA dentro de la base datos.

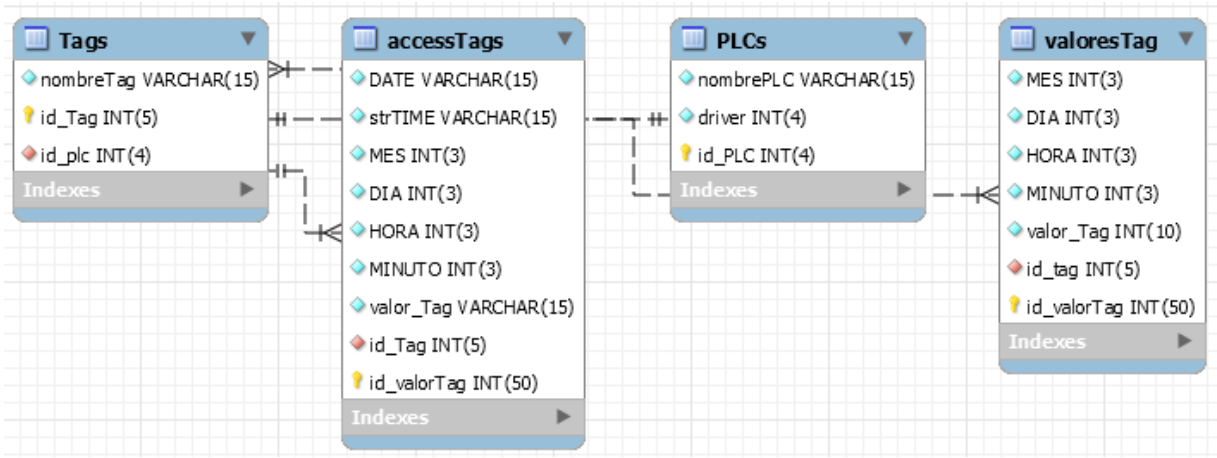


Figura 3.32: Diagrama para la base de datos del historiador en MySql.
Fuente: Autores

3.6. Arquitectura general del sistema SCADA

La Figura 3.33 muestra la interacción entre los niveles de proceso, control y visualización. El nivel de proceso conformado por el motor y *encoder*, proporciona información al nivel de control. El servidor OPC-UA adquiere los datos de los controladores lógicos programables poniéndolos a disposición de la interfaz del cliente.

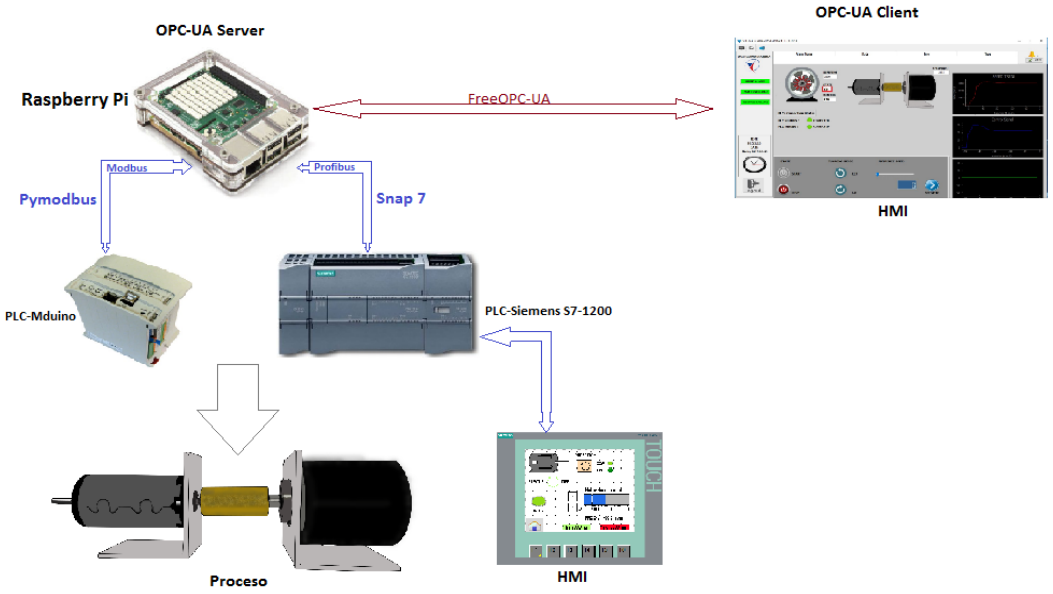


Figura 3.33: Arquitectura general del sistema SCADA.
Fuente: Autores

Capítulo 4

Pruebas de funcionamiento y evaluación de resultados

4.1. Servidor OPC-UA

4.1.1. Ventana de inicialización del servidor

El servidor OPC-UA se debe iniciar antes de usar el sistema SCADA. La Figura 4.1 muestra la ventana de inicio de sesión del servidor. Se especifica la dirección IP del servidor y el puerto al que accede un cliente. El botón conectar da inicio al servidor.
Una vez iniciado el servidor, se abre la ventana principal del mismo (Figura 4.2). La

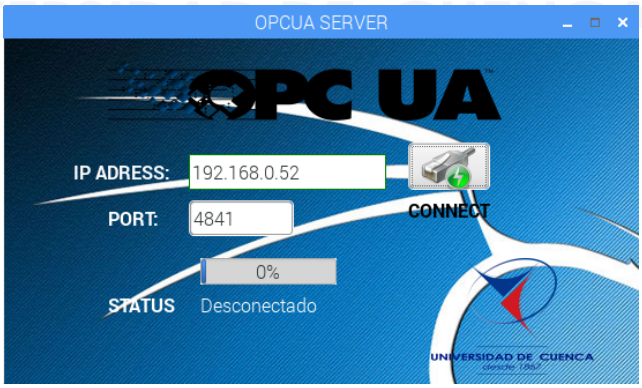


Figura 4.1: Ventana inicio de sesión del servidor.
Fuente: Autores

ventana esta formada por:

- **URI:** El rectángulo amarillo muestra el esquema URI del servidor.

- **Objetos:** Los objetos se visualizan en la parte lateral izquierda (rectángulo azul). Además, las *tags* agregadas a cada objeto se visualizan en este apartado.
- **Visor:** El rectángulo rojo señala el área de visualización de atributos de una variable, nodos suscritos a cambios, registro de conexiones y referencias.

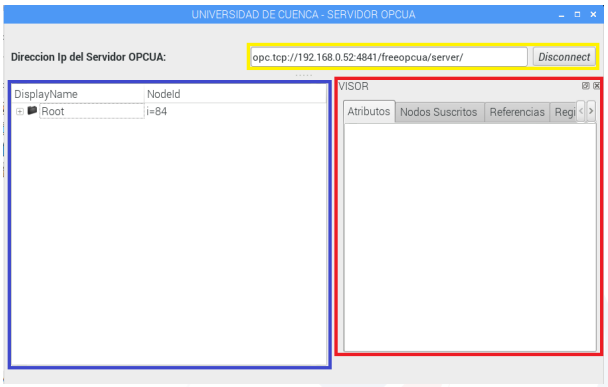


Figura 4.2: Ventana principal del servidor.
Fuente: Autores

4.1.2. Registro de usuarios y objetos predeterminados

Los clientes validados por la base de datos, se conectan al servidor. La Figura 4.3 señala el nombre del usuario conectado al servidor en la parte lateral derecha.
En la parte lateral izquierda se aprecian los objetos agregados predeterminadamente. Los objetos mencionados son los PLCs del nivel de control. El recuadro azul encierra el objeto *"Conexion Status"*, el cual señala el estado de la conexión de cada uno de los PLC.

4.1.3. Suscripción a cambios

Todas las variables que intervienen en la tarea de controlar y detectar fallos del motor DC se pueden visualizar en el servidor. Estas variables, llamadas *tags*, para ser visualizadas deben estar suscritas a cambios, esto se realiza seleccionando la opción "Suscribirse a Cambios". La Figura 4.4 expone en la parte lateral derecha las *tags* del proceso simulado.

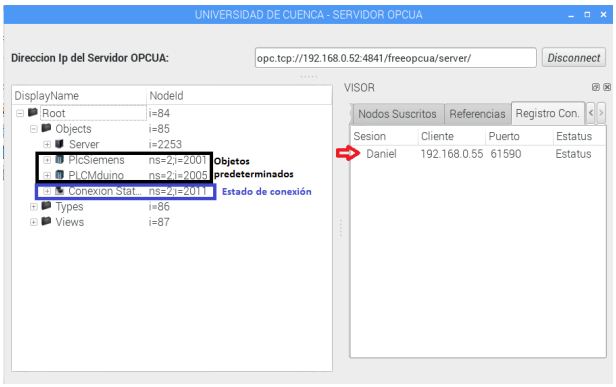


Figura 4.3: Registro de usuarios en el servidor.
Fuente: Autores

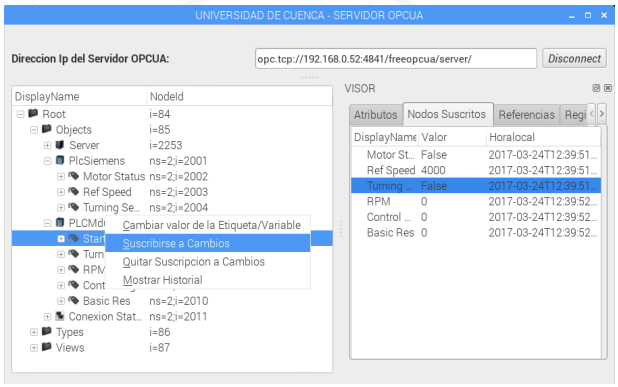


Figura 4.4: *Tags* suscritas a cambios.
Fuente: Autores

4.2. Sistema SCADA

4.2.1. Inicio de sesión

Una vez iniciado el servidor, el cliente puede iniciar sesión en el sistema SCADA. La Figura 4.5 indica los campos que se deben llenar para acceder al sistema, estos son:

- **Campo "Username":** En este campo se ingresa el nombre de usuario. El nombre de usuario es un campo obligatorio, si existe un intento de inicio de sesión y este campo esta vacío, se muestra el mensaje de error de la Figura 4.6 (a).
- **Campo "Password":** Al igual que el nombre de usuario, este es un campo obligatorio y no debe estar en blanco al registrarse. De no cumplirse con el requisito se tiene el mensaje de error de la Figura 4.6 (b).
- **Botón "Login":** Intenta el inicio de sesión, si los datos son válidos se muestra

el mensaje de la Figura 4.7 y se habilita el botón *"Connect"* (Figura 4.8), por el contrario se muestra el mensaje de la Figura 4.9.

- **Botón "Connect":** Inicia la conexión con el servidor. Una vez conectado, se da inicio al sistema SCADA.
- **Botón "Cancel":** Cierra la ventana de inicio de sesión.

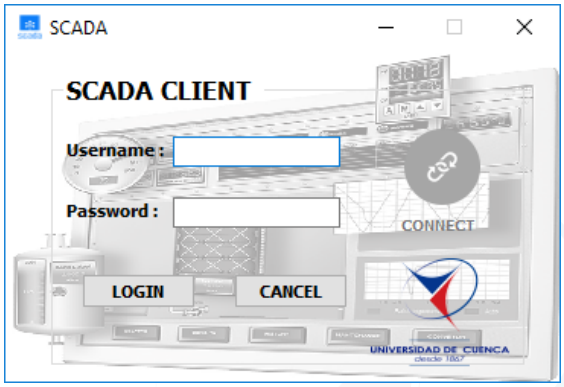
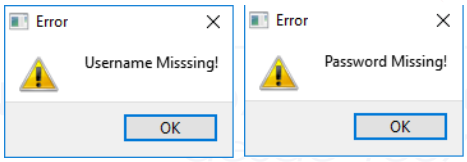


Figura 4.5: Ventana de inicio de sesión de cliente.
Fuente: Autores



(a) Usuario vacío. (b) Contraseña vacía.

Figura 4.6: Campos en ventana vacío.
Fuente: Autores

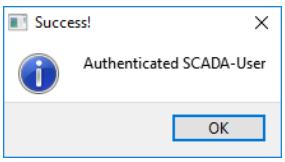


Figura 4.7: Datos de inicio de sesión validados.
Fuente: Autores

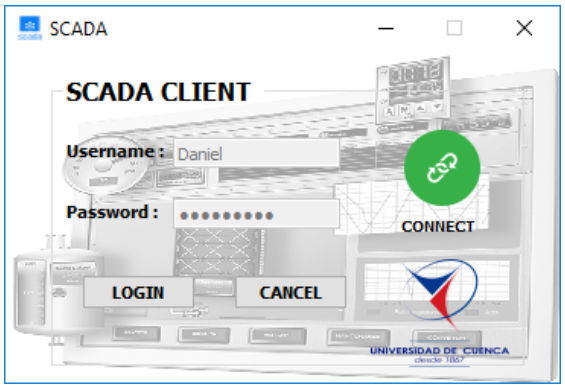


Figura 4.8: Botón de conexión habilitado.
Fuente: Autores

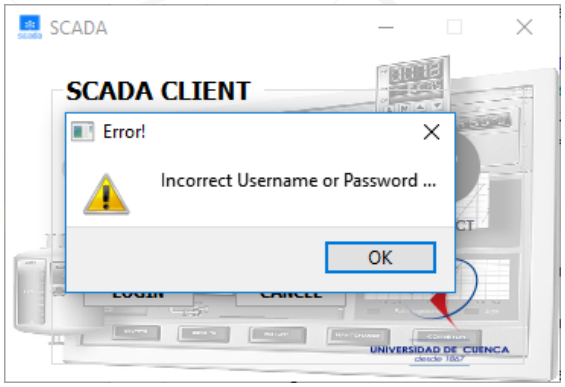


Figura 4.9: Datos de inicio de sesión incorrectos.
Fuente: Autores

4.2.2. Ventana principal

El usuario que se haya registrado correctamente accede a la ventana principal del sistema SCADA. Esta se divide en 6 partes enmarcadas como se muestra en la Figura 4.10, las cuales tienen las siguientes funciones:

1. **Barra de selección de ventana.-** Encerrada por el recuadro de color gris. Esta barra permite seleccionar entre la ventana principal, la ventana de tendencias o la ventana de historiales.
2. **Sección de fecha, hora y alerta de alarmas.-** Ubicada en la parte lateral izquierda (recuadro café) muestra la hora y la fecha actual. Además, se puede apreciar los tres tipos de alarmas, de presentarse una de ellas ésta cambiaría a un color rojo parpadeante. El botón de "Logout" ubicado en la parte inferior de

este recuadro cierra el sistema, desconectándolo del servidor.

3. **Barra de alarmas.-** Esta barra (recuadro naranja) contiene una lista de todas las alarmas que se hayan presentado, con sus respectivas características las cuales son: nombre, fecha, hora y tipo de alarma. La barra de alarmas contiene además un contador de alarmas que no hayan sido revisadas, y el botón "Check" que permite eliminar una determinada alarma de la lista antes mencionada.
4. **Planta.-** El recuadro negro encierra la planta (motor DC y *encoder*), en este se ven el estado del motor, sentido de giro, velocidad de referencia y la velocidad real medida por el sensor.
5. **Sección de gráficas.-** Las gráficas en tiempo real de: velocidad, señal de control y residuo básico se encuentran en la parte lateral derecha de la ventana (recuadro rojo).
6. **Panel de control y estado de conexión de PLCs.-** Los recuadros verdes enmarcan dos partes de la ventana. La primera es el panel de control, la segunda indica el estado de la conexión de los dispositivos de control (PLCs).

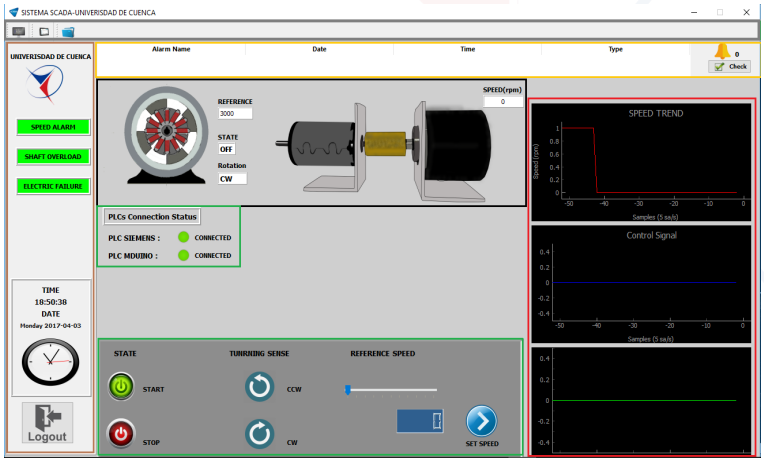


Figura 4.10: Ventana principal SCADA.
Fuente: Autores

Cabe recalcar que las secciones 1,2 y 3 están a su vez presentes en las ventanas de tendencias e historiales.

4.2.3. Ventana de tendencias

La Figura 4.11 muestra la distribución espacial de las secciones que conforman la ventana de tendencias. La ventana está conformada por dos secciones:

1. **Intervalo de Tendencia.-** Esta sección esta ubicada en la parte lateral derecha de la ventana (recuadro amarillo). En este espacio se selecciona el día, la hora de inicio y fin de la tendencia. En la parte inferior de esta sección se encuentra el botón de validación del intervalo de tiempo y el botón que permite graficar las tendencias.
2. **Gráfica de tendencias.-** Ubicado en la parte central (recuadro rojo), en este sitio se grafican las tendencias. A través de pestañas se puede elegir la tendencia que se desea ver (velocidad, señal de control, residuo).

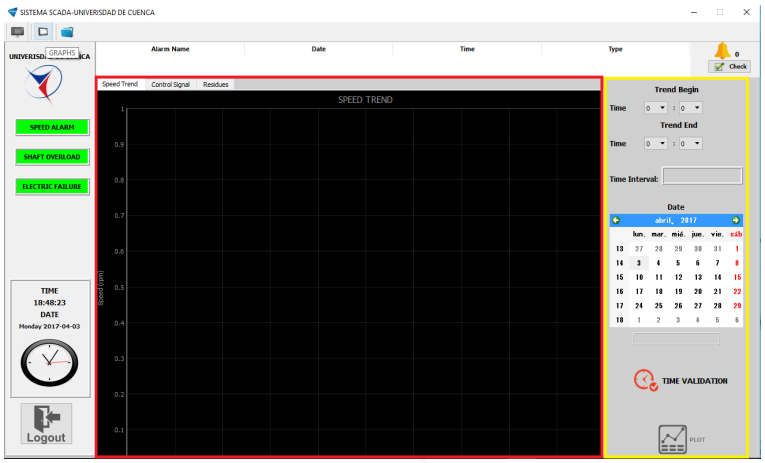


Figura 4.11: Ventana de tendencias.
Fuente: Autores

4.2.4. Ventana de historiales.

Esta ventana (Figura 4.12) consta de 3 secciones:

1. **Selección de fecha.-** El recuadro amarillo encierra el espacio en donde se selecciona la fecha para la cual se quieren listar las acciones o alarmas. El botón que se observa en la parte inferior habilita la sección de botones una vez que se selecciona una fecha.
2. **Sección de botones.-** Se encuentran en la parte inferior (recuadro azul), una vez que se selecciona un día, se habilitan los botones.
3. **Sección de historiales.-** Encerrada por el recuadro rojo, en este espacio se listan las acciones o alarmas que se dieron en un día, según el botón presionado.

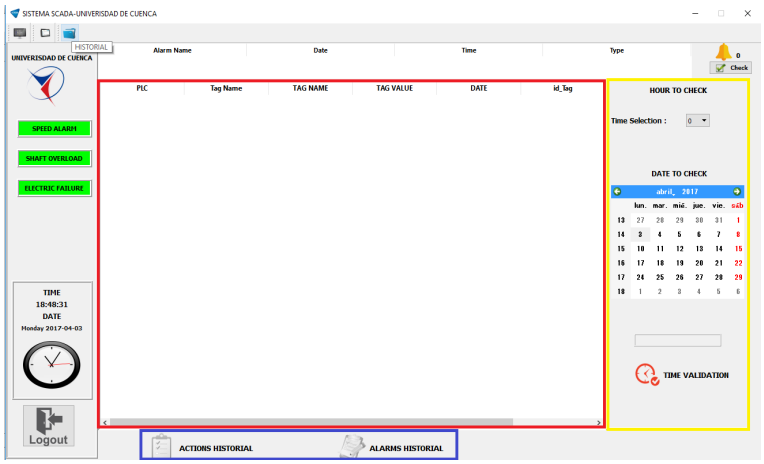


Figura 4.12: Ventana de historiales.
Fuente: Autores

4.2.5. Validación de usuarios

El sistema está en la capacidad de distinguir entre dos perfiles de usuarios. Los perfiles son el de usuario administrador y usuario estándar. La Tabla 4.1 indica los usuarios, contraseñas y la prioridad de cada uno. Una prioridad de "1" se refiere a un usuario administrador, por el contrario, una prioridad de "2" refleja un usuario estándar.

Tabla 4.1: Base de datos de usuarios del sistema SCADA.

Usuario	Contraseña	Prioridad
Daniel	daniel123	1
Jonnathan	jonh123	1
Brian	brian123	2

4.2.5.1. Usuario administrador

Los usuarios tienen perfiles para distinguir los privilegios de cada uno de ellos. Un usuario administrador tiene la capacidad de controlar la planta, en nuestro caso, el estado del motor, sentido de giro y velocidad de referencia del motor DC. Adicionalmente, una alarma que se hace presente en el sistema puede ser revisada únicamente por el usuario administrador.
Las funciones generales del sistema tales como: monitoreo de variables en tiempo real,

visualización de tendencias, historial de alarmas e historial de acciones también se incluyen como acciones que realiza este usuario.
La Figura 4.13 ilustra la habilitación del panel de control y el botón de revisión de alarmas del sistema. Cabe recalcar que los dispositivos de control deben estar conectados como se puede observar en el recuadro azul, de no ser así el panel de control se deshabilita para cualquier usuario.

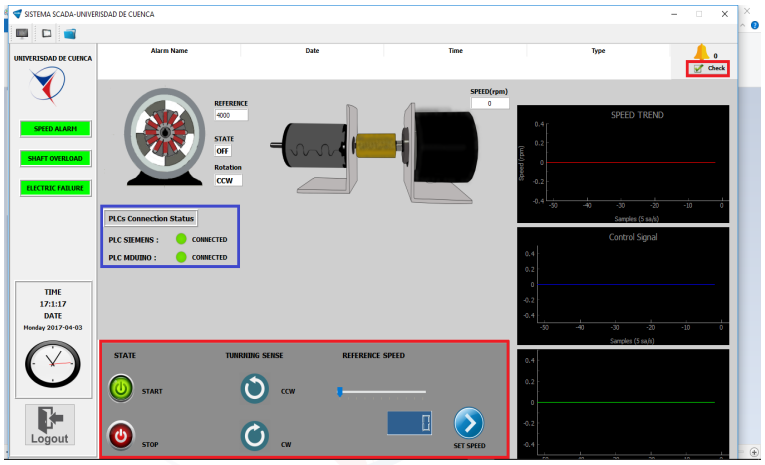


Figura 4.13: Ventana de un usuario administrador.
Fuente: Autores

4.2.5.2. Usuario estándar

Un usuario estándar se distingue en la base de datos por tener prioridad "2". Este tipo de usuario no esta en la capacidad de controlar la planta, ni revisar alarmas. Las funciones generales del sistema no se restringen para estos usuarios.
La Figura 4.14 detalla el bloque del botón de revisión de alarmas y de el panel de control de la planta.

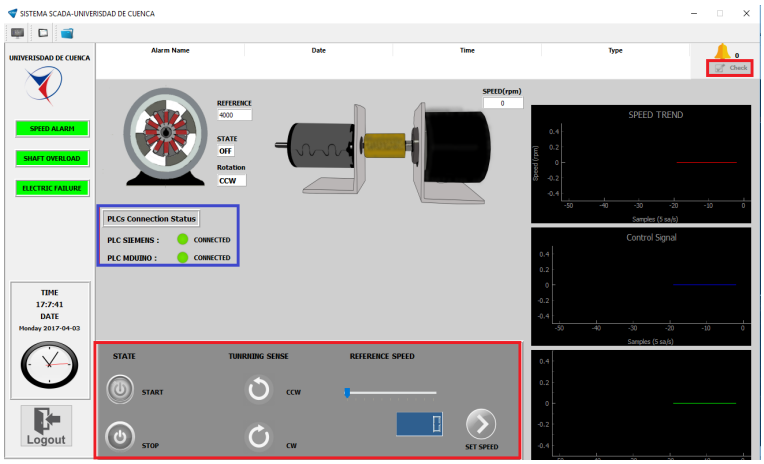


Figura 4.14: Ventana de un usuario estándar.
Fuente: Autores

4.3. Experimentación y pruebas de funcionamiento del prototipo

La sección de pruebas se basan en el funcionamiento del sistema SCADA con un usuario administrador, y la comunicación con el servidor OPC-UA.
Las pruebas del sistema SCADA se dividen en 5 partes. Las pruebas de desempeño del servidor se analizan al final de esta sección.

4.3.1. Control de planta y detección de fallos

4.3.1.1. Control de la planta

- **Cambio de estado**

Una vez realizada la conexión con el servidor, el sistema SCADA analiza el estado de conexión de los dispositivos de control (PLC Siemens S7-1200, PLC MDuino).
Si y solo si los dos PLCs están conectados se habilita el panel de control. La Figura 4.15 señala el estado del motor (“OFF”) en un instante.
La Figura 4.16 muestra el inicio del motor DC una vez que se presiona el botón “START”. En la sección de gráficas se observa el arranque del motor en la parte superior, en la parte inferior se muestra el cambio en la señal de control.

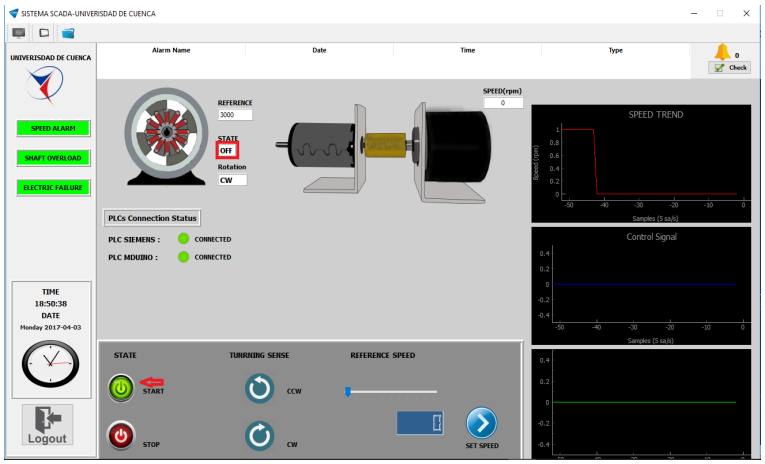


Figura 4.15: Motor DC estado OFF.
Fuente: Autores

- **Cambio de sentido de giro**

El panel de control proporciona al usuario la capacidad de escoger el sentido de giro del motor DC. En la Figura 4.17 se indica en el recuadro rojo el sentido de giro en ese instante.
En la Figura 4.18 se aprecia el cambio de sentido de giro realizado.

- **Cambio de referencia**

El cambio de referencia de la velocidad del motor es otra de las funciones que se le faculta a un usuario administrador. En la Figura 4.19 se muestra la referencia de velocidad inicial, además, encierra en el círculo rojo el botón de cambio la velocidad ("SET SPEED"). La barra permite al usuario elegir la velocidad.
El cambio de velocidad realizado en esta prueba es de 2800 rpm a 3500 rpm (Figura 4.20).

4.3.1.2. Detección de fallos

La detección de fallos clasifica a las alarmas en dos tipos: alarmas de precaución ("Warning Alarm"), y alarmas críticas ("Critical Failure").

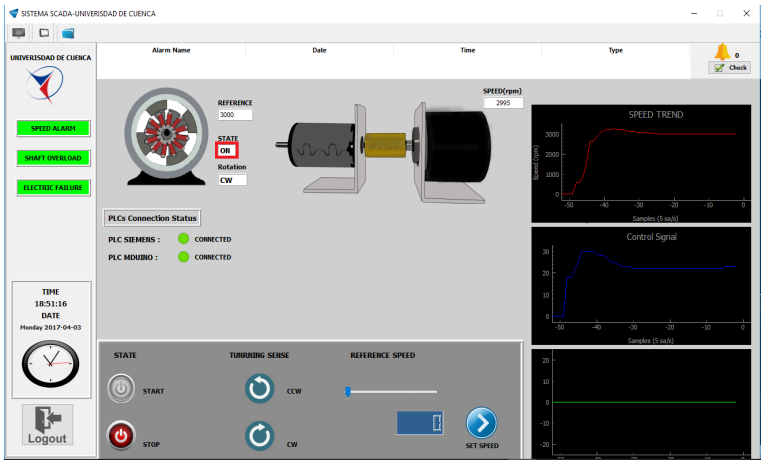


Figura 4.16: Cambio de estado del motor DC.
Fuente: Autores

4.3.1.3. Alarma de precaución

La alarma de precaución brinda la posibilidad de detectar fallas en la velocidad del motor DC. Esta alarma permite reconocer la existencia perturbaciones (Figura 4.21).

4.3.1.4. Alarmas críticas

El sistema está en la capacidad de clasificar las alarmas críticas en dos: falla eléctrica ("Electric Failure") y sobrecarga en el eje ("Shaft Overload"). Una falla eléctrica significa el desgaste del devanado del motor. Una sobrecarga en el eje implica una falla mecánica.
La presencia de cualquier fallo de esta naturaleza conlleva a un mayor esfuerzo por parte del controlador, por ende es necesario un mayor consumo energético que al final representa mayores gastos. Por esta razón estas fallas son denominadas críticas.

- **Falla eléctrica**

Para insertar una falla eléctrica, se agrega una resistencia en serie al motor DC. El valor de la resistencia es de 5 Ω la cual produce un residuo básico promedio mayor a 20. La Figura 4.22 expone el resultado (parte inferior de la sección de gráficas) del residuo cuyo valor permite detectar una falla eléctrica.

- **Falla mecánica**

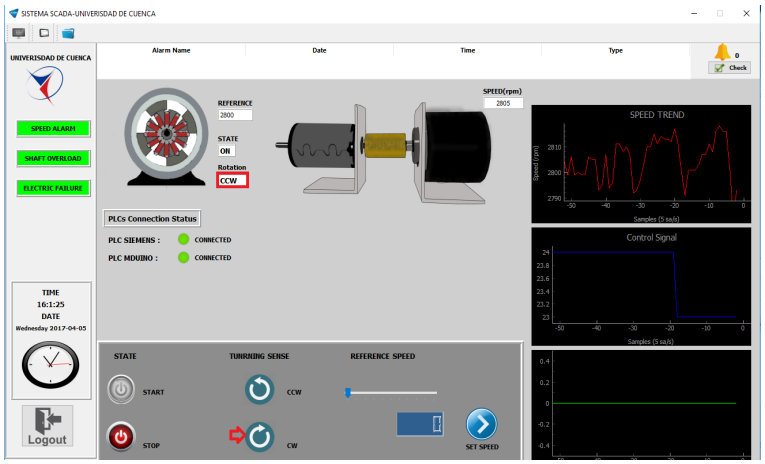


Figura 4.17: Motor DC girando en sentido antihorario.
Fuente: Autores

Para introducir una falla mecánica, se agrega un peso al eje del motor. El valor promedio del residuo básico en presencia de esta falla está entre 0 y 16. En la Figura 4.23 se logra ver el comportamiento del residuo una vez que se ha dado la falla.

4.3.2. Visualización de tendencias

La visualización de tendencias permite graficar el comportamiento de la velocidad del motor, la señal de control y el residuo básico en intervalos de tiempo deseados. Para esto se selecciona el día y las horas de inicio y fin de la tendencia.

Para evitar problemas al momento de consultar la base de datos se realizan dos validaciones. La primera validación impide tener una hora inicial de tendencia mayor a la hora final, de darse esto se muestra el mensaje de error de la Figura 4.24 (a).

La segunda validación evita que el usuario intente graficar tendencias sin especificar el día y el mes. El intento de validar la fecha sin indicar en el calendario el día produce el mensaje que se muestra en la Figura 4.24 (b).

El botón “PLOT” se habilita (Figura 4.26) una vez que se haya ingresado exitosamente la fecha y hora para las cuales se desean visualizar las tendencias (Figura 4.25).

En esta prueba se escogen los datos del día 3 de abril, con un intervalo de tiempo de 6 minutos, los cuales están entre las 17h41 como hora final y las 17h35 como hora inicial. La Figura 4.27 muestra el resultado para cada una de las tendencias.

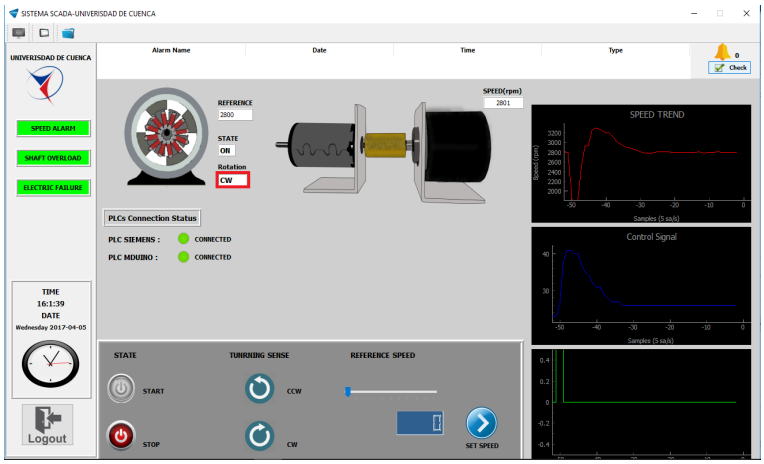


Figura 4.18: Motor DC girando en sentido horario.
Fuente: Autores

4.3.3. Visualización de historiales

En esta ventana se muestran los cambios en las variables en el panel de control o una lista de alarmas. Cabe recalcar que solo se pueden listar cambios y alarmas que se hayan dado en una hora específica.
La única validación que se realiza en esta ventana es la del día, es decir, los botones de la parte inferior se habilitan únicamente cuando se haya seleccionado un día en el calendario (Figura 4.28).

4.3.3.1. Historial de acciones

La tabla de historial de acciones tiene las siguientes columnas:

- **PLC:** En esta columna se especifica el PLC a la que pertenece la *tag*.
- **Tag Name:** Indica el nombre de la *tag*. Estas pueden ser : el estado del motor ("Motor Status"), el sentido de giro ("Turning Sense") y la velocidad de referencia ("Ref Speed").
- **Tag Value:** Muestra el estado de las variables.
- **DATE:** Indica la fecha.
- **TIME:** Indica la hora exacta del cambio del valor de la *tag*.
- *id_Tag*: Cada *tag* tiene un identificador único, en esta columna se lo especifica.

La prueba que se realiza en este apartado pertenece al día 3 de abril a las 17h00. La Figura 4.29 muestra todos los cambios que se realizaron en ese día. La tabla de historial

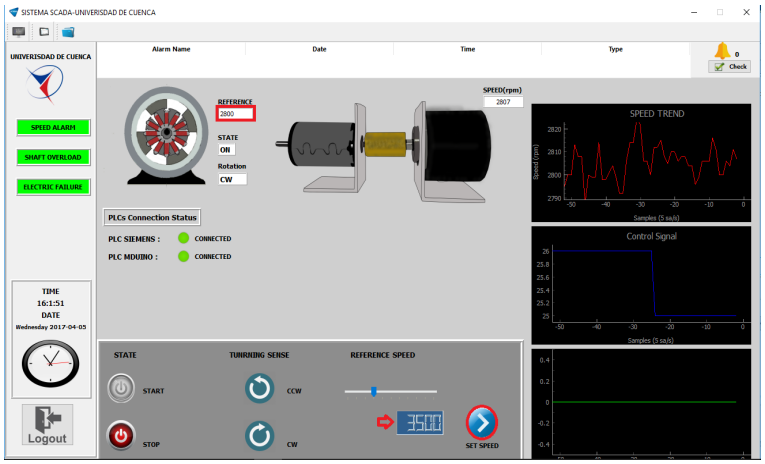


Figura 4.19: Referencia de velocidad inicial del motor DC.
Fuente: Autores

de acciones concede a los usuarios la oportunidad de tener un control detallado de todos los cambios a los que se somete el motor DC.

4.3.3.2. Historial de alarmas

La tabla de historial de alarmas consta de 5 columnas, las cuales se detallan a continuación:

- **Alarm Name:** Esta columna indica el nombre de la alarma.
- **DATE:** Muestra la fecha en la que se produjo la alarma.
- **TIME:** Especifica la hora exacta a la que se produjo la alarma.
- **TYPE:** Indica el tipo de alarma, puede ser alarma de precaución (“Warning Alarm”) o alarma crítica (“Critical Failure”).
- **Description:** Esta columna describe si una alarma ha sido revisada o no (“Acknowleged” y “Unacknowleged” respectivamente).

Las alarmas de precaución tendrán por defecto “Unacknowleged” en la columna de descripción ya que no tienen efectos nocivos en el motor.
Las alarmas críticas deben ser revisadas por el usuario administrador. Una vez que se hayan revisado, en la tabla se especifica la hora y el minuto, además en la columna de descripción aparece “Acknowleged” que indica que la alarma ha pasado por la etapa de revisión. La Figura 4.30 muestra un ejemplo de una lista de alarmas que se dieron el día 3 de abril a las 17h00.

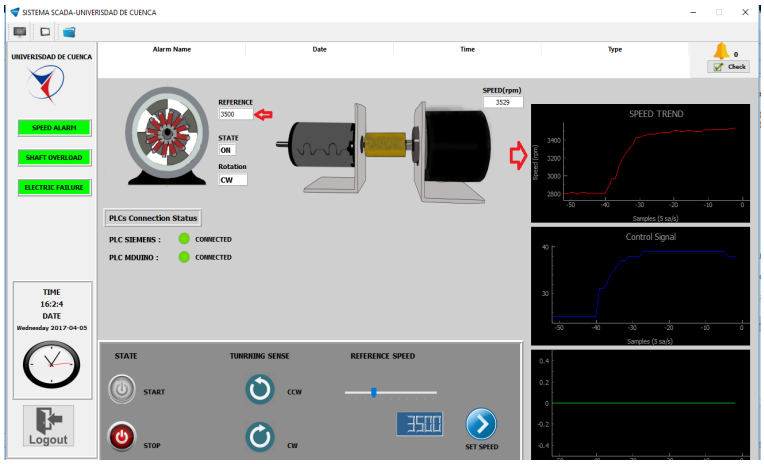


Figura 4.20: Referencia de velocidad final del motor DC.
Fuente: Autores

4.3.4. Desempeño del servidor

El código fuente del servidor OPC-UA implementado en [4] se utiliza como base para el desarrollo del servidor OPC-UA empleado en este trabajo.

Gracias a la actualización realizada en el servidor OPC-UA, se elimina la barrera que limitaba al anterior servidor al uso de un único protocolo de comunicación. La versión previa era compatible solamente con PLCs de marca Siemens S7-1200, los cuales utilizan el protocolo Profibus.

El PLC MDuino se utiliza para el control y detección de fallos del motor DC, este dispositivo utiliza el protocolo de comunicación MODBUS, lo que llevó a realizar cambios al antiguo servidor. La actualización de software permite al servidor aceptar cualquier otro tipo de protocolo, por ejemplo el protocolo CAN.

Para comprobar la funcionalidad del servidor y su idoneidad para trabajar en conjunto con los dispositivos de control utilizados en este trabajo, se realizaron varias pruebas las cuales permitieron descubrir otra de las limitaciones a las cuales estaba sujeto el predecesor. El código heredado no permitía hacer lecturas de variables con un tiempo menor a $500ms$.

El controlador de velocidad está implementado con un tiempo de muestreo de $100ms$, variables como la velocidad del motor necesitan ser muestreadas con este tiempo. Además, el sistema SCADA adquiere variables en tiempo real, lo que obliga a tener tiempos de adquisición en el orden de los milisegundos.

La incapacidad de la versión previa del servidor OPC-UA de adquirir el valor de una variable en tiempos menores a $500ms$, hace imposible el uso del mismo como herra-

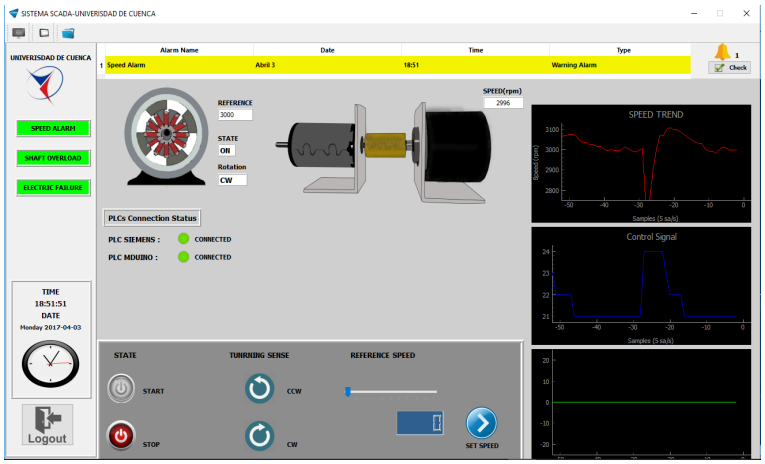


Figura 4.21: Alarma de precaución.
Fuente: Autores

mienta en este trabajo. Este impedimento llevo a la necesidad de actualizar y mejorar el servidor.

La Figura 4.31 exhibe la funcionalidad de la nueva versión del servidor. Se aprecia la velocidad del motor, la señal de control y el residuo básico, variables que se muestrean cada 100ms con el protocolo MODBUS. Por otra parte, se tienen los estados del motor variables a las que se accede gracias al protocolo PROFIBUS.

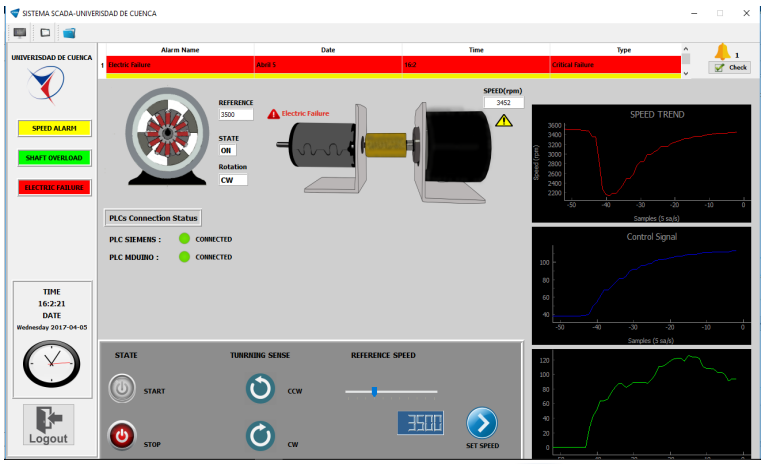


Figura 4.22: Falla eléctrica.
Fuente: Autores

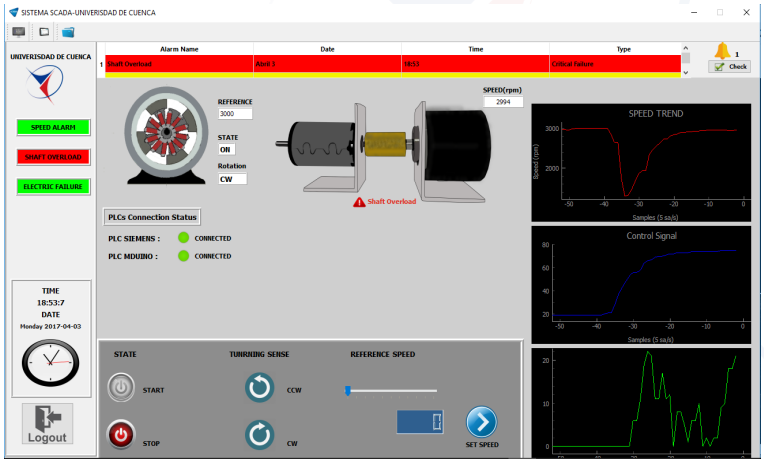
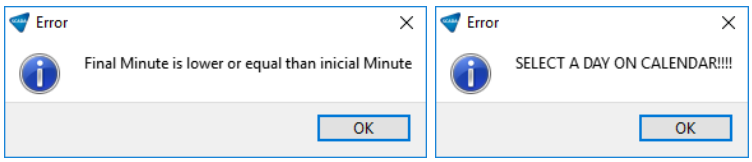


Figura 4.23: Falla mecánica.
Fuente: Autores



(a) Error en intervalo. (b) Error en día.

Figura 4.24: Datos inválidos.
Fuente: Autores

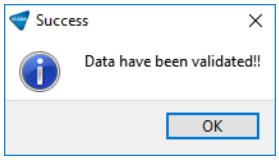


Figura 4.25: Fecha y horas validadas.
Fuente: Autores

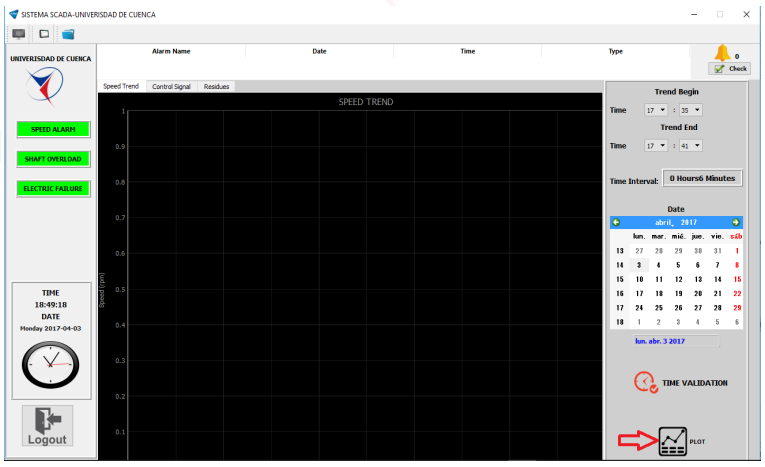
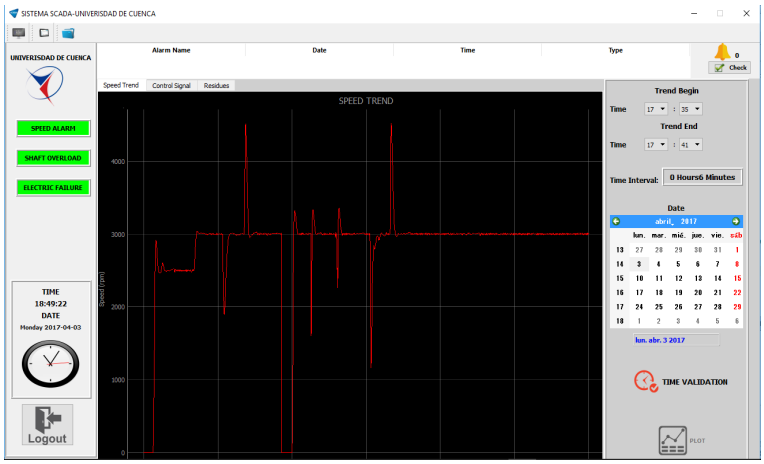
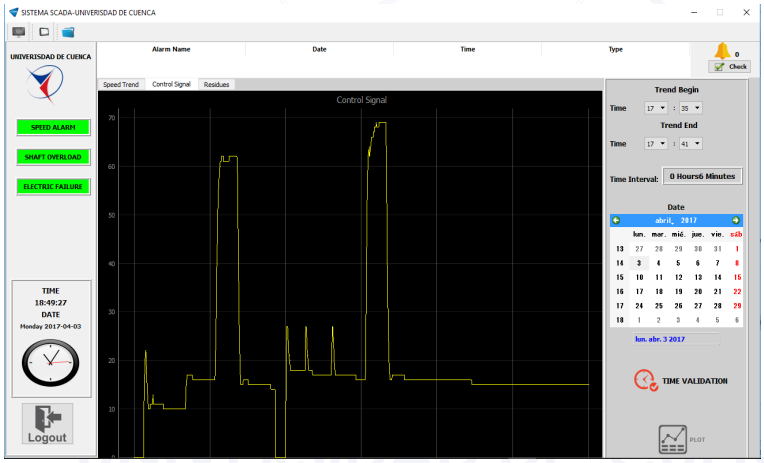


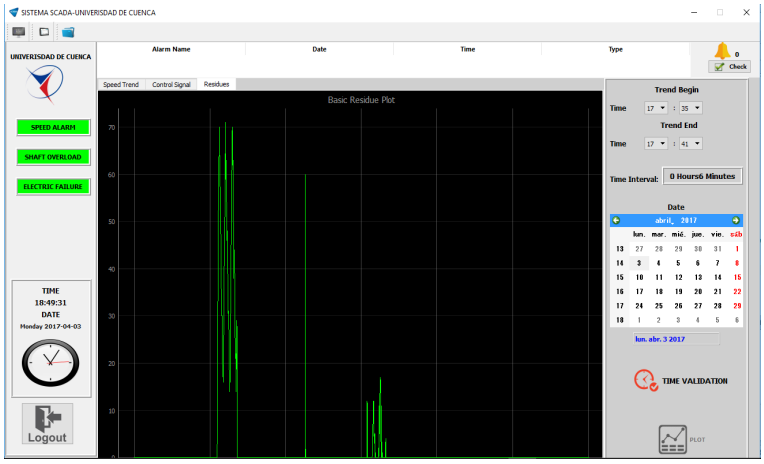
Figura 4.26: Botón PLOT habilitado.
Fuente: Autores



(a) Tendencia de velocidad.



(b) Tendencia de señal de control.



(c) Tendencia de residuo básico.

Figura 4.27: Visualización de tendencias.
Fuente: Autores

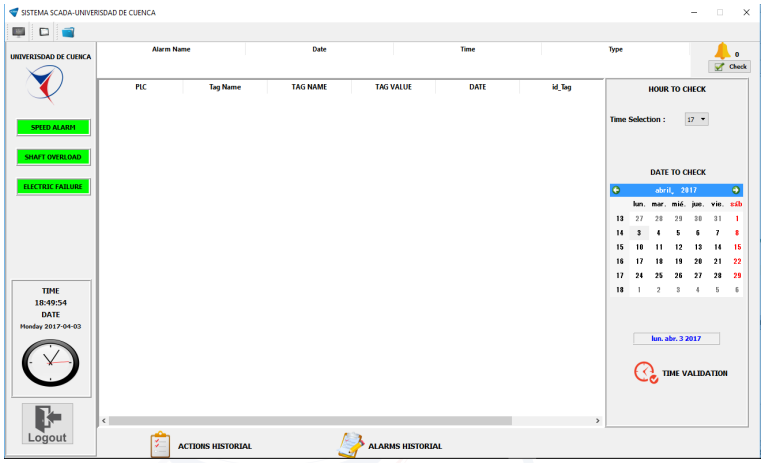


Figura 4.28: Botones de historiales habilitados.
Fuente: Autores

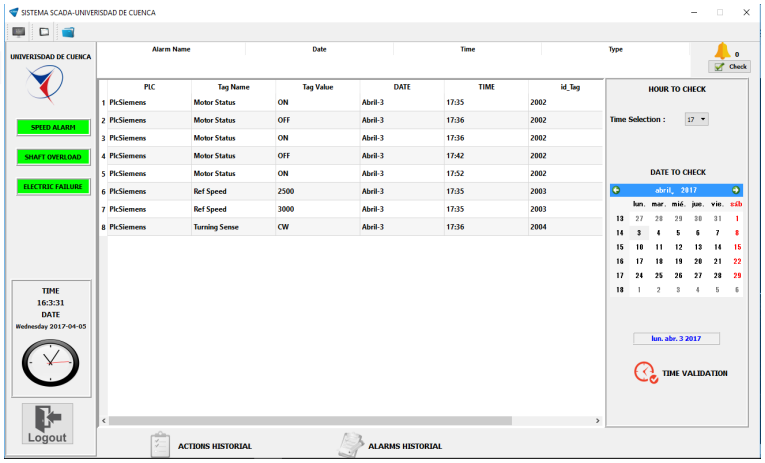


Figura 4.29: Tabla de historial de acciones.
Fuente: Autores

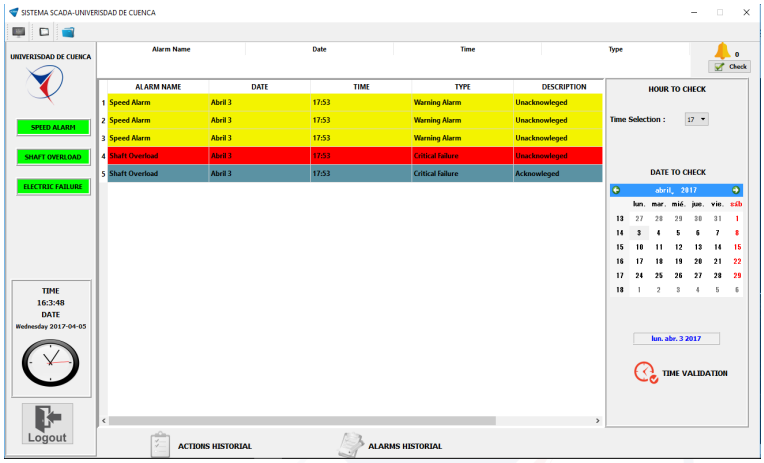


Figura 4.30: Tabla de historial de alarmas.
Fuente: Autores

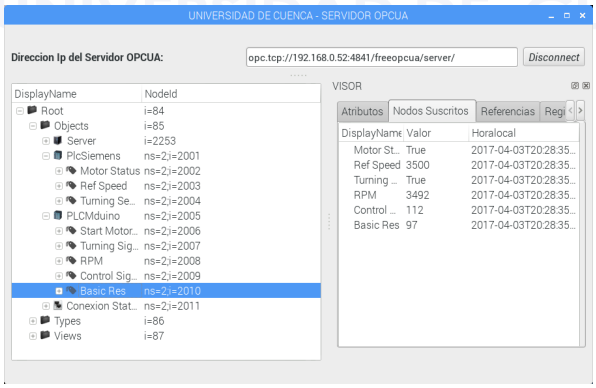


Figura 4.31: Versión actualizada del servidor.
Fuente: Autores

4.4. Resultados

Los resultados obtenidos por el sistema SCADA, son evaluados de acuerdo a las funciones que debe cumplir una HMI. La Tabla 4.2 exhibe los objetivos cumplidos por el sistema desarrollado en este trabajo.

Cabe mencionar que el último objetivo se refiere a la manipulación de los parámetros

Tabla 4.2: Objetivos cumplidos por el sistema.

Objetivo	**Resultado**
Visualización del proceso	Satisfecho
Control operativo de un proceso	Satisfecho
Mostrar alarmas	Satisfecho
Archivar valores de procesos y alarmas	Satisfecho
Registro de alarmas y valores de procesos	Satisfecho
Gestión de parámetros de procesos y maquinaria	No satisfecho

de un proceso para alterar la elaboración de un determinado producto. En este trabajo no se fabrica ningún producto, por lo que no se cumple ese objetivo. El trabajo desarrollado satisface cinco de los seis objetivos principales de un sistema SCADA.

El servidor OPC-UA, el sistema SCADA y la planta estuvieron en funcionamiento por una hora, en donde se realizaron cambios de referencia de la velocidad del motor DC, cambios de sentido de giro, manipulación del estado del motor DC, visualización de tendencias tanto de velocidad, señal de control y residuo básico.

Adicionalmente, fue posible visualizar historiales de alarmas y acciones realizadas sobre la planta. La detección de alarmas fue exitosa, permitiendo diferenciar entre alarmas de precaución, y alarmas críticas. Se introdujeron fallas eléctricas y mecánicas durante el periodo de prueba, realizando la tarea de clasificación de manera exitosa.

Las alarmas se sometieron a revisión por parte de un usuario administrador, almacenando los tiempos de revisión y posteriormente listados en la ventana de historiales.

El servidor estuvo en la capacidad de soportar los dos tipos de usuarios (administrador y estándar) sin ningún tipo de problema. La lectura de las variables de los PLCs se realizó correctamente, corroborando la robustez del servidor.

Capítulo 5

Conclusiones

5.1. Conclusiones

Un modelo CIM permite a las industrias automatizar los procesos de producción. Los 7 niveles que conforman este modelo facilitan el reconocimiento de todas las etapas que intervienen en dicha automatización.

El trabajo desarrollado abarca los tres primeros niveles del modelo CIM estos son: nivel de proceso, nivel de control y el nivel de visualización. La particularidad del desarrollo tiene que ver con el uso de *software* libre y plataformas industriales de bajo costo.

En el nivel de proceso se usa un *encoder* incremental como sensor y el motor Maxon como un actuador, simulando un proceso.

El nivel de control emplea los controladores lógicos programables MDuino y el Siemens S7-1200, éstos hacen uso de dos de los protocolos de comunicación más empleados a nivel industrial, MODBUS y Profibus respectivamente. El sistema logra manipular los dos protocolos de comunicación gracias a la intervención de el servidor OPC-UA, desarrollado en Python.

El servidor se monta en un Raspberry Pi para cumplir con el objetivo de reducción de costos. Gracias al trabajo del servidor los tres niveles del modelo mantienen una comunicación constante, manteniendo la transparencia con los usuarios.

El nivel de visualización corresponde al sistema SCADA desarrollado en Python.

Este nivel permite tener el control de la planta y monitorear las variables que intervienen en el proceso.

El sistema SCADA es capaz de generar alarmas y gestionarlas. Las alarmas críticas detectadas permiten evitar fallos que de no ser revisados a tiempo representan una amenaza a la integridad física de los dispositivos de campo, y a su vez refleja un incremento en gastos. El SCADA desarrollado cumple con 5 de los 6 objetivos que demanda una interfaz humano-máquina (HMI por sus siglas en inglés) haciéndolo un sistema de gran utilidad en relación al costo que representa su adquisición.

El hecho de que el servidor OPC-UA y el sistema SCADA estén desarrollados en Python, facilitan su ejecución en cualquier plataforma. Por supuesto, el uso de código libre evita que industrias se aten a licencias costosas, permitiéndoles aprovechar las ventajas que brinda el uso de estos sistemas. Como consecuencia, aumentan los niveles productividad y competitividad en el mercado.

5.2. Trabajo futuro

- Comprobar la funcionalidad de todo el sistema en un proceso industrial.
- Implementar otros métodos de detección de fallos para distinguir todas las amenazas presentes en una etapa de producción.
- Comparar el rendimiento de un sistema SCADA comercial con el sistema SCADA desarrollado.
- Verificar el funcionamiento del servidor con protocolos de comunicación que no estén presentes en este trabajo, por ejemplo el protocolo CAN.
- Permitir al sistema SCADA agregar de manera dinámica variables que intervienen en un proceso, siempre y cuando estén presentes en el servidor.

Apéndices

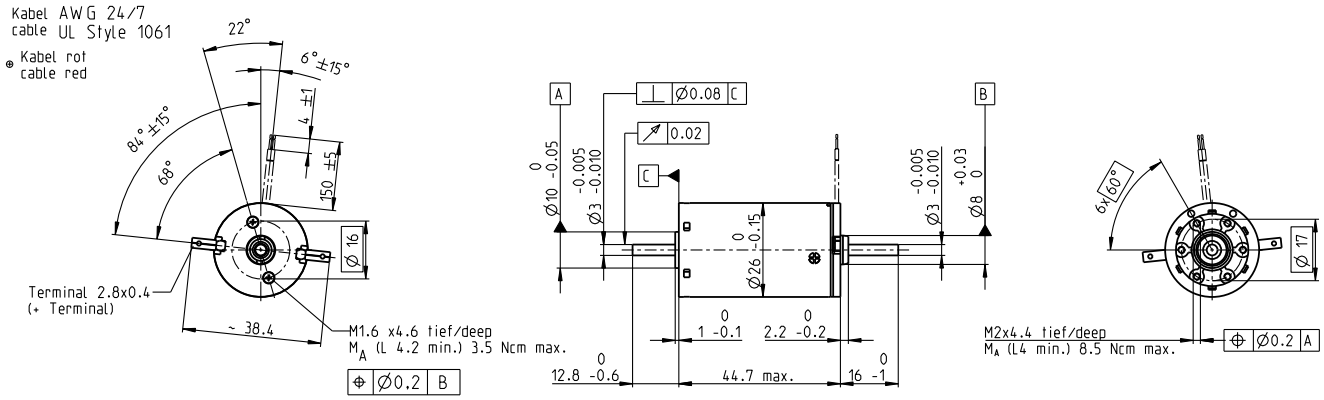
UNIVERSIDAD DE CUENCA

desde 1867

Apéndice A

Hoja de especificaciones Motor MAXON

A-max 26 ∅26 mm, Graphite Brushes, 6 Watt



M 1:2

Stock program
Standard program
Special program (on request)

Part Numbers

with terminals	110946	110947	110948	110949	**110950**	110951	110952	**110953**	110954	110955	110956	110957	
with cables	353143	353144	353145	353146	353147	353148	353149	353150	353151	353152	353153	353154	

Motor Data

	Values at nominal voltage												
1	Nominal voltage	V	7.2	9	12	12	18	18	24	24	30	36	42	48
2	No load speed	rpm	9790	10500	10300	8510	8380	7510	8680	7950	8890	8500	8230	6280
3	No load current	mA	121	106	77.7	60.2	39.4	34.2	31	27.7	25.5	20.1	16.5	10.3
4	Nominal speed	rpm	8580	8840	8510	6210	5890	5000	6050	5250	6350	5950	5630	3590
5	Nominal torque (max. continuous torque)	mNm	6.67	7.91	11	13.6	14.5	14.6	13.7	13.4	14.1	14.1	13.9	13.8
6	Nominal current (max. continuous current)	A	1.08	1.08	1.08	1.08	0.755	0.679	0.554	0.498	0.467	0.373	0.305	0.203
7	Stall torque	mNm	54.6	51.4	63.4	50.9	49.4	44	45.7	39.8	49.8	47.6	44.6	32.9
8	Stall current	A	7.89	6.36	5.79	3.84	2.45	1.96	1.76	1.41	1.57	1.2	0.931	0.461
9	Max. efficiency	%	77	76	78	77	76	76	76	74	76	76	76	73
	Characteristics													
10	Terminal resistance	Ω	0.912	1.41	2.07	3.13	7.36	9.19	13.6	17	19.1	30.1	45.1	104
11	Terminal inductance	mH	0.101	0.138	0.254	0.372	0.861	1.07	1.42	1.69	2.13	3.35	4.85	10.8
12	Torque constant	mNm/A	6.92	8.07	11	13.3	20.2	22.5	25.9	28.3	31.7	39.8	47.9	71.4
13	Speed constant	rpm/V	1380	1180	872	720	473	425	368	338	301	240	199	134
14	Speed / torque gradient	rpm/mNm	182	207	165	170	173	174	193	204	181	181	188	195
15	Mechanical time constant	ms	23.5	23.7	23.4	23.5	23.6	23.6	23.8	24	23.8	23.8	23.9	24.1
16	Rotor inertia	gcm²	12.3	10.9	13.6	13.2	13.1	13	11.8	11.2	12.5	12.5	12.2	11.8

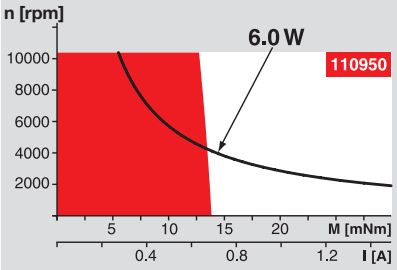
Specifications

	Thermal data	
17	Thermal resistance housing-ambient	13.2 K/W
18	Thermal resistance winding-housing	3.2 K/W
19	Thermal time constant winding	12.5 s
20	Thermal time constant motor	660 s
21	Ambient temperature	-30...+85°C
22	Max. winding temperature	+125°C
	Mechanical data (ball bearings)	
23	Max. speed	10400 rpm
24	Axial play	0.1 - 0.2 mm
25	Radial play	0.025 mm
26	Max. axial load (dynamic)	5 N
27	Max. force for press fits (static)	75 N
	(static, shaft supported)	1200 N
28	Max. radial load, 5 mm from flange	20 N
	Mechanical data (sleeve bearings)	
23	Max. speed	10400 rpm
24	Axial play	0.1 - 0.2 mm
25	Radial play	0.012 mm
26	Max. axial load (dynamic)	1.7 N
27	Max. force for press fits (static)	80 N
	(static, shaft supported)	1200 N
28	Max. radial load, 5 mm from flange	5.5 N
	Other specifications	
29	Number of pole pairs	1
30	Number of commutator segments	13
31	Weight of motor	100 g

Values listed in the table are nominal.
Explanation of the figures on page 151.

Option
Sleeve bearings in place of ball bearings

Operating Range



Comments

Continuous operation
In observation of above listed thermal resistance (lines 17 and 18) the maximum permissible winding temperature will be reached during continuous operation at 25°C ambient.
= Thermal limit.

Short term operation
The motor may be briefly overloaded (recurring).

Assigned power rating

maxon Modular System

Overview on page 20–27

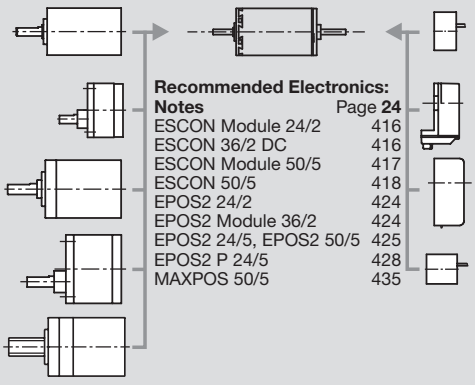
Planetary Gearhead
∅26 mm
0.75 - 4.5 Nm
Page 336

Spur Gearhead
∅30 mm
0.07 - 0.2 Nm
Page 337

Planetary Gearhead
∅32 mm
0.75 - 6.0 Nm
Page 338/339/342

Spur Gearhead
∅38 mm
0.1 - 0.6 Nm
Page 348

Spindle Drive
∅32 mm
Page 370–372



Recommended Electronics:

Notes	Page 24
ESCON Module 24/2	416
ESCON 36/2 DC	416
ESCON Module 50/5	417
ESCON 50/5	418
EPOS2 24/2	424
EPOS2 Module 36/2	424
EPOS2 24/5, EPOS2 50/5	425
EPOS2 P 24/5	428
MAXPOS 50/5	435

Encoder MR
128 - 1000 CPT,
3 channels
Page 392

Encoder Enc
22 mm
100 CPT, 2 channels
Page 398

Encoder HED_ 5540
500 CPT,
3 channels
Page 400/402

Encoder MEnc
∅13 mm
16 CPT, 2 channels
Page 410

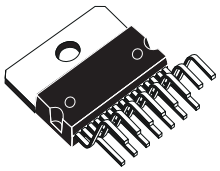
Apéndice B

Hoja de especificaciones integrado L298

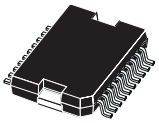
L298

DUAL FULL-BRIDGE DRIVER

- OPERATING SUPPLY VOLTAGE UP TO 46 V
- TOTAL DC CURRENT UP TO 4 A
- LOW SATURATION VOLTAGE
- OVERTEMPERATURE PROTECTION
- LOGICAL "0" INPUT VOLTAGE UP TO 1.5 V (HIGH NOISE IMMUNITY)



Multiwatt15



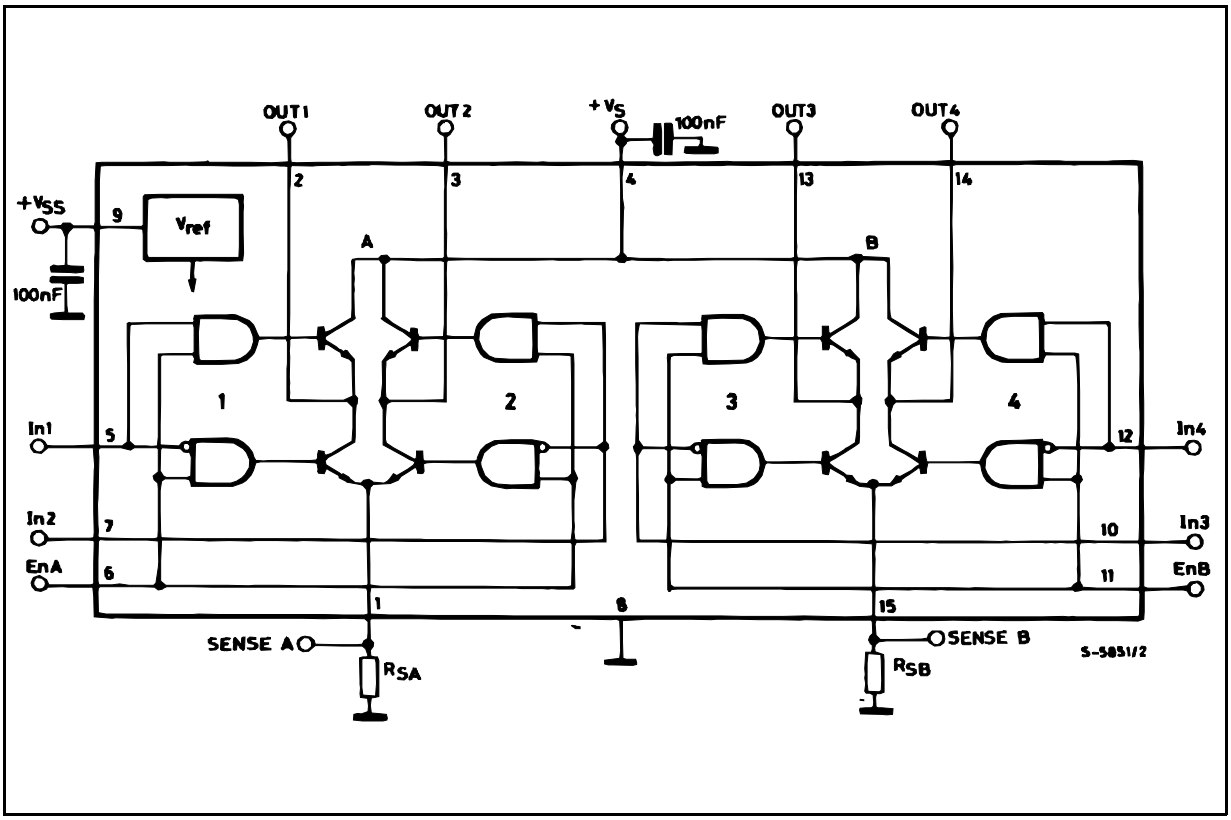
PowerSO20

ORDERING NUMBERS : L298N (Multiwatt Vert.)
L298HN (Multiwatt Horiz.)
L298P (PowerSO20)

DESCRIPTION

The L298 is an integrated monolithic circuit in a 15-lead Multiwatt and PowerSO20 packages. It is a high voltage, high current dual full-bridge driver designed to accept standard TTL logic levels and drive inductive loads such as relays, solenoids, DC and stepping motors. Two enable inputs are provided to enable or disable the device independently of the input signals. The emitters of the lower transistors of each bridge are connected together and the corresponding external terminal can be used for the connection of an external sensing resistor. An additional supply input is provided so that the logic works at a lower voltage.

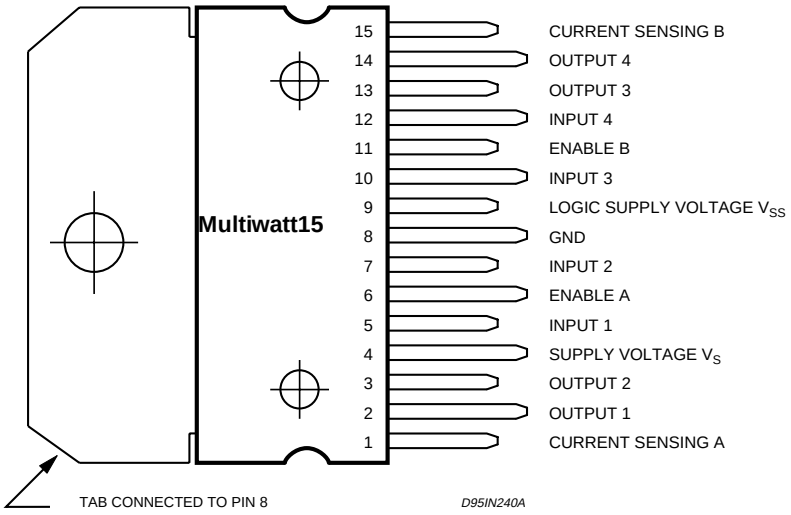
BLOCK DIAGRAM

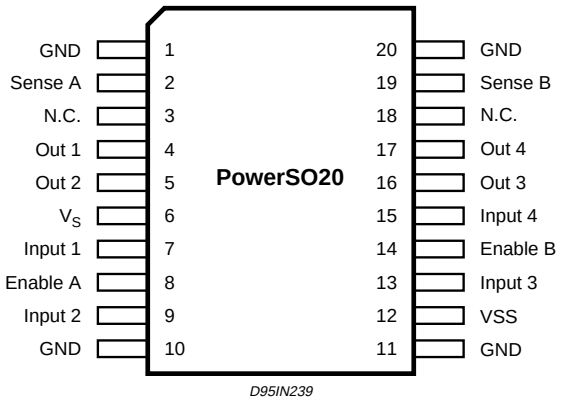


ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	Power Supply	50	V
V_{SS}	Logic Supply Voltage	7	V
V_I,V_{en}	Input and Enable Voltage	–0.3 to 7	V
I_O	Peak Output Current (each Channel) – Non Repetitive (t = 100µs) –Repetitive (80% on –20% off; t_{on} = 10ms) –DC Operation	 3 2.5 2	 A A A
V_{sens}	Sensing Voltage	–1 to 2.3	V
P_{tot}	Total Power Dissipation (T_{case} = 75°C)	25	W
T_{op}	Junction Operating Temperature	–25 to 130	°C
T_{stg}, T_j	Storage and Junction Temperature	–40 to 150	°C

PIN CONNECTIONS (top view)





THERMAL DATA

Symbol	Parameter		PowerSO20	Multiwatt15	Unit
$R_{th\ j\text{-}case}$	Thermal Resistance Junction-case	Max.	–	3	°C/W
$R_{th\ j\text{-}amb}$	Thermal Resistance Junction-ambient	Max.	13 (*)	35	°C/W

(*) Mounted on aluminum substrate

Apéndice C

Hoja de especificaciones encoder incremental modelo A6B2

增量型旋转编码器

增量型　外径φ40　型号：A6B2

INCREMENTAL ROTARY ENCODERS,OUTSIDE DIAM φ40 MODEL：A6B2

替代型号SUBSTITUTE：E6B2

通用编码器

■种类Ordering Information

◆本体ABSOLUTE ROTARY ENCODERS

电源电压 Supply Voltage	输出状态 Output Configuration	分辨率（脉冲/旋转） Resolution(p/r)	型号 model
DC5~24V	NPN开路集电极输出 Open collector NPN output	10、20、30、40、50、60、100、200、300、360、400、500、600	型号A6B2-CWZ6C
		1,000	
		1,200、1,500、1,800、2,000	
DC12~24V	PNP开路集电极输出 Open collector PNP output	100、200、360、500、600	型号A6B2-CWZ5B
		1,000	
		2,000	
DC5~12V	电压输出 Voltage output	10、20、30、40、50、60、100、200、300、360、400、500、600	型号A6B2-CWZ3E
		1,000	
		1,200、1,500、1,800、2,000	
DC5V	线性驱动输出 Push Pull output	10、20、30、40、50、60、100、200、300、360、400、500、600	型号A6B2-CWZ1X
		1,000	
		1,200、1,500、1,800、2,000	

注.订货时，除型号外，还一定要指定分辨率。

■：表格中为标准分辨率、不标准的可订货生产。

■种类ACCESSORIES

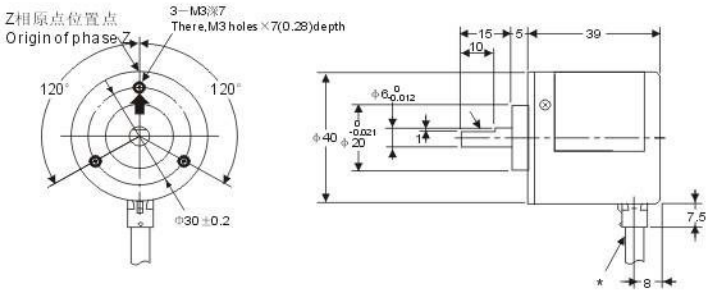
◆附件(零售)

种类Description	型号 Part number	备　注 Remark
耦合器 Shaft coupler	◎型号E69-C06B	附于商品 Fits two 6 mm (0.24 in) dia. shafts; supplied with each encoder.
	◎型号E69-C68B	不同直径型 Shaft coupler Fits one 6 mm (0.24 in) and 8 mm (0.32 in) dia. Shaft
	◎型号E69-C10B	不同直径型 **Fits one 6 mm (0.24 in) and 10 mm (0.39 in) dia. Shaft**
	◎型号E69-C06M	金属型 material：aluminum metal
法兰盘 Mounting flange	◎型号E69-FBA	—
	◎型号E69-FBA02	伺服装置用安装配件 附属于型号E69-2
金属压钩配件 Mounting bracket, set of three	型号E69-2	—

■外形尺寸Dimensions　(单位Unit：mm)

◆本体

型号model:A6B2



＊型号A6B2－CWZ6C/5C/83E：
PVC绝缘圆形导线Φ5、5芯
(导体截面积：0.2mm²、绝缘体直径：Φ1.0mm)标准长500mm
型号A6B2－CWZ1X：
PVC绝缘圆形导线Φ5、5芯
(导体截面积：0.2mm²、绝缘体直径：Φ1.0mm)标准长500mm

PVC shielded cable, 0.5 m (1.64 ft) standard length

增量型旋转编码器

增量型　外径φ40　型号：A6B2

INCREMENTAL ROTARY ENCODERS,OUTSIDE DIAM φ40 MODEL：A6B2

替代型号SUBSTITUTE：E6B2

■额定/性能SPECIFICATIONS

项目 ITEM / 型号 MODEL	型号A6B2-CWZ6C	型号A6B2-CWZ5B	型号A6B2-CWZ3E	型号A6B2-CWZ1X
电源电压 Powersupplyvoltage	DC5V-5%~24V+15% 脉冲(p-p)5%以下	**DC12V**-10%~24V+15% 脉冲(p-p)5%以下	DC5V-5%~12V+10% 脉冲(p-p)5%以下	DC5V±5% 脉冲(p-p)5%以下
消耗电流 Currentconsumption	80mA以下	100mA以		160mA以下
分辨率(脉冲/旋转) Resolution(See Note 1)	10、20、30、40、50、60、100、200、300、360、400、500、600、1,000、1,200、1,5000、1,8000、2,000、	100、200、360、500、600、1,000、2,000	10、20、30、40、50、60、100、200、300、360、400、500、600、1,000、1,200、1,5000、1,8000、2,000、	
输出相 Output phases	A、B、Z相			A、$\overline{A}$、B、$\overline{B}$、Z、$\overline{Z}$相
输出相位差 Phase difference of output	A相、B相的相位差90±45°(1/4±1/8T)			
输出状态 Output form	NPN开路输出 Open collectorNPN output	PNP开路输出 Open collector PNP output	电压输出 Voltage output	线性驱动输出*2 Line driver output
输出容量 Output capacity	外加电压：DC30V以下 In Voltage：DC30V max 同步电流：35mA以下 In- sink：：35 mA max 残留电压：0.4V以下 Residual voltage: 0.4 V max (输出电流) 35mA时 (In-sink：35mA))	同步电流：35mA以下 In- sink：：35 mA max 残留电压：0.4V以下 Residual voltage: 0.4 V max (输出电流) 35mA时 (In-sink：35mA))	输 出 电 阻：2kΩ Output resistance: 2 kΩ 输出电流：35mA以下 Sink current: 35 mA max 残留电压：0.4V以下 Residual voltage: 0.4 V max (输出电流)：35mA时	AM26LS31相当品 输出电流 H位：Ic=-20mA High level L位：Is=20mA 输出电压 V0=2.5V以上min Low level Vs=0.5V以下max
最高应答频率 Maximum response frequency	100kHz	50kHz	100kHz	
输出上升、下降时间 Output rise and fall times	1ms以下 (控制输出电压：5V 负荷电阻1kΩ、导线长：2m)	1ms以下 (导线长：2m 同步电流：10mA)		1ms以下 (导线长：2m Io=-20mA、Is=20mA)

机械性能 参数 Mechanical Spec		
起动转矩 Startion Torque		0.98mN·M以下
惯性力矩Moment of Inertia		1×10-6kg·m2以下(脉冲/旋转以下：3×10-7kg·m2以下
允许容力 Shaft loading	倾向Radial	29.4N
	推力Thrust	19.6N
允许安装精度 Mounting Tolerance		侧面误差 Raclial:0.03mm IIR Max； 正向误差 Axial：0.2mm Max； 角度误差Shaft Runout：0.1°Max
轴最大负荷 Allowable Shaft Load		径向Radial ：5N，轴向 Axial ：3N
允许最高转速Maxirnun Rotating Speed		6,000r/min
环境温度 Operating Temp Range		工作时：-10~+70℃、保存时：-25~85℃(不结冰)
环境湿度Humidity		工作时，保存时：各35~85%RH(不结露)
绝缘电阻 Insulation resistance		100MΩ以上(DC500V摇表)充电部整体与外壳间
漏电压		AC500V 50/60Hz 1min充电部整体与外壳间
振动(耐久)Vibration resistance		10~500Hz上下一步振幅2mm或150m/s²、X、Y、Z各方向1次11min 3次
冲击(耐久)Shock resistance		1,000m/s²、X、Y、Z各方向3次
保护结构 Degree of protection		IEC规格 Ip50
连接方式 Connection		导线引出型(标准导线长cablelength：500mm)
质量 Weight		约100g
附件 Accessories		耦合器、六角板手、使用说明书

*1.接通电压时，约有9A的冲流流过。(时间：约0.3ms)

*2.所谓线性驱动输出就是根据RS-422A的数据输送回路。可通过双股绞合线导线进行长距离输送。(内置AM 26LS31相当品)

*3.电气的最高响应转速(r/min)= $\frac{\text{最高响应频率}}{\text{分辨率}} \times 60$

因此，旋转超过最高响应转速时无法对电信号进行追踪。

Note:

- **The maximum electrical response revolution is determined by the resolution and maximum response frequency as follows:**
- **Maximum electrical response frequency (rpm) = Maximum response frequency ÷ resolution x 60**
- **This means that the A6B2 encoder will not operate electrically if its shaft speed exceeds the maximum electrical response revolution.**

Apéndice D

Manual de usuario MDduino

2 General Description M-DUINO FAMILY product

A compact PLC based in Open Source Hardware technology. With different Input/Outputs Units.

CONECTABLE PLC ARDUINO 24Vcc M-DUINO				
MODEL TYPE	**21 I/Os**	**42 I/Os**	**58 I/Os**	
Input Voltage	12- 24Vdc			Fuse protection (1A) Polarity protection
I max.	0,5A			
Size	101x119.5x70.1	101x119.5x94.7	101x119.5x119.3	
Clock Speed	16MHz			
Flash Memory	256KB of which 8KB used by bootlader			
SRAM	8KB			
EEPROM	4KB			
Communications	I2C[1] – Ethernet Port – USB – RS485 – RS232 -- SPI – (2x) Rx,Tx (Arduino pins)			Max232-Max485-W5100
TOTAL Input points	13	26	36	
TOTAL Output points	11	22	30	
Type of signals				
An/Dig Input 10bit (0-10Vcc)	6	12	16	0-10V Input Impedance: 39K Separated PCB ground
Digital Isolated Input (24Vcc)	7	14	20	5/12/24Vdc I min: 2/6/12 mA Galvanic ISOLATION
*** Interrupt isolated Input HS** (24Vcc)	2	4	6	5/12/24Vdc I min: 2/6/12 mA Galvanic ISOLATION
Analog Output 8bit (0-10Vcc)	3	6	8	0-10 Vdc I max: 40 mA Separated PCB ground
Digital Isolated Output (24Vcc)	8	16	22	5/12/24 Vdc I max: 0.3 A Galvanic ISOLATION Diode Protected for Relay
PWM Isolated Output 8bit (24Vcc)	3	6	8	5/12/24 Vdc I max: 0.3 A Galvanic ISOLATION Diode Protected for Relay
Expandability	I2C - 127 elements - Serial Port RS232/RS485			

[1] Pull-up resistance required (IS.ACI2C-4.7K)

4 Specifications

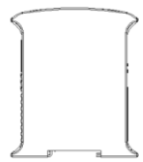
4.1 General Specifications:

Item		M-DUINO 21 IOs	M-DUINO 42 IOs	M-DUINO 58 IOs
Power supply voltage	**DC power supply**	12 - 24Vdc		
Operating voltage range	**DC power supply**	11.4 to 25.4Vdc		
Power consumption	**DC power supply**	30VAC max.		
External power supply	**Power supply voltage**	24Vdc		
	Power supply output capacity	700Ma		
Insulation resistance		20MΩ min.at 500Vdc between the AC terminals and the protective earth terminal.		
Dielectric strength		2.300 VAC at 50/60 HZ for one minute with a leakage current of 10mA max. Between all the external AC terminals and the protective earth terminal.		
Shock resistance		80m/s2 in the X, Y and Z direction 2 times each.		
Ambient temperature (operating)		0º to 45ºC		
Ambient humidity (operating)		10% to 90% (no condensation)		
Ambient environment (operating)		With no corrosive gas		
Ambient temperature (storage)		-20º to 60ºC		
Power supply holding time		2ms min.		
Weight		445g max.	542g max.	850g max.

4.2 Performance Specification:

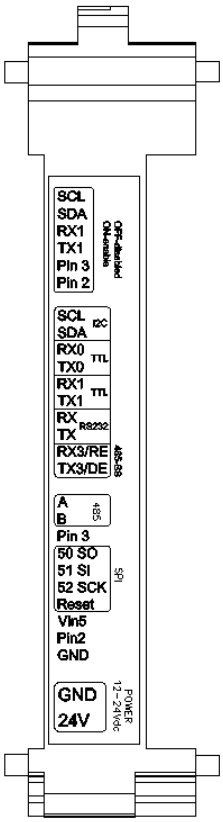
Item	M-DUINO 21 IOs	M-DUINO 42 IOs	M-DUINO 58 IOs
Arduino Board	**ARDUINO MEGA 2560**		
Control method	Stored program method		
I/O control method	Combination of the cyclic scan and immediate refresh processing methods.		
Programming language	Arduino IDE. Based on wiring (Wiring is an Open Source electronics platform composed of a programming language. “similar to the C”. http://arduino.cc/en/Tutorial/HomePage		
Microcontroller	ATmega2560		
Flash Memory	256kb of which 8 kb used by bootloader		
Program capacity (SRAM)	8kb		
EEPROM	4kb		
Clock Speed	16MHz		
Clock Speed	16MHz		

6 M-duino 21/42/58 I/O Pinout:



6.1 A Zone connection (21/42/58 I/Os)

Base (common unit)		
A Zone		
M-Duino Connector	*Arduino Pin*	Function
SCL	*21*	I2C/SS
SDA	*20*	I2C/SS
RX0	*1*	RX0/SS
TX0	*0*	TX0/SS
RX1	*19*	RX1/SS
TX1	*18*	TX1/SS
RX	*17*	RX2(serial 2)
TX	*16*	TX2(serial 2)
RX3/RE	*15*	RX3/RS485/SS
TX3/DE	*14*	TX3/RS485/SS
A	-	RS485
B	-	RS485
PIN3	*3*	Arduino Pin/ Select SPI
50 SO	*50*	SPI
51 SI	*51*	SPI
52 SCK	*52*	SPI
Reset	*Reset*	SPI
Vin5	*Vin5*	SPI
PIN2	*2*	Arduino Pin/ Select SPI
GND	-	Gnd
GND	-	Gnd
24Vdc		

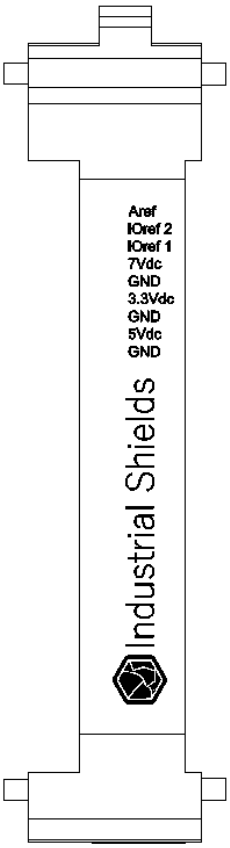


Configuration Switch* (see section 8 for Communications configuration.
Enabling Communications disable s some I/Os)

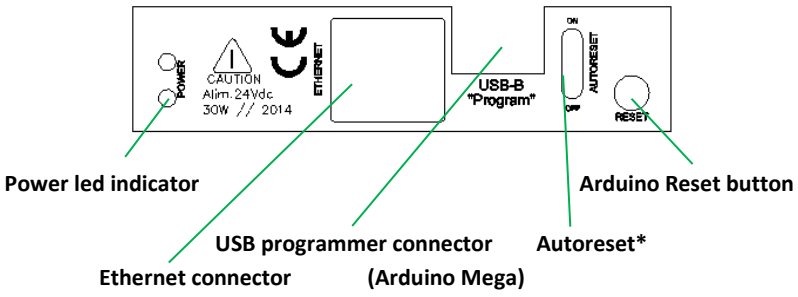
Communication Pinout

Power supply connectors (24Vdc – Gnd)

Base (common unit)		
A Zone		
M-Duino Connector	*Arduino Pin*	Function
AREF	*AREF*	Arduino PIN
IOREF2	*IOREF2*	Arduino PIN
IOREF1	*IOREF1*	Arduino PIN
7Vdc	*7Vdc*	-
Gnd	*Gnd*	GND
3.3Vdc	*3.3Vdc*	Arduino PIN
GND	*Gnd*	GND
5Vdc	*5Vdc*	-
GND	*Gnd*	GND



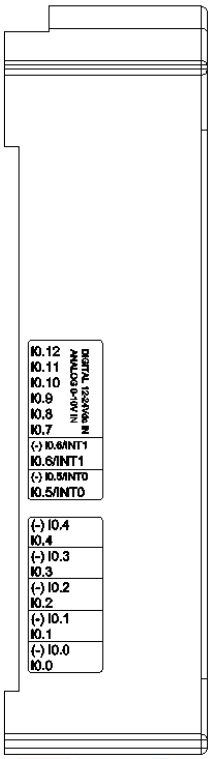
6.2 A Zone top (21/42/58 I/Os)



**NOTE: Autoreset. Arduino mega has auto reset when using serial communication code. Set switch to OFF when using serial communication. When uploading code to Arduino Mega set switch to ON.*

6.3 B Zone (21/42/58 I/Os)

B Zone		
M-Duino Connector	*Arduino Pin*	Function [2]
I0.12	*A5*	Analog/ Digital In
I0.11	*A4*	Analog/ Digital In
I0.10	*A3*	Analog/ Digital In
I0.9	*A2*	Analog/ Digital In
I0.8	*A1*	Analog/ Digital In
I0.7	*A0*	Analog/ Digital In
(-)I0.6/INT1	*NC*	GND I0.6
I0.6/INT1 [3]	*3*	Interrupt 1 In
(-)I0.5/INT0	*NC*	GND I0.5
I0.5/INT0 [3]	*2*	Interrupt 0 In
(-)I0.4	*NC*	GND I0.4
I0.4	*26*	Digital Input
(-)I0.3	*NC*	GND I0.3
I0.3	*25*	Digital Input
(-)I0.2	*NC*	GND I0.2
I0.2	*24*	Digital Input
(-)I0.1	*NC*	GND I0.1
I0.1	*23*	Digital Input
(-)I0.0	*NC*	GND I0.0
I0.0	*22*	Digital Input

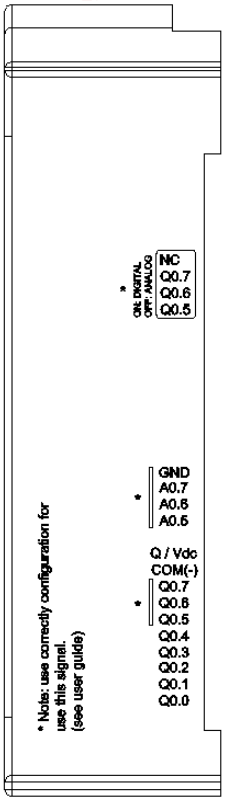


ANALOG/DIGITAL Inputs

INTERRUPT Inputs (isolated)

DIGITAL Inputs (isolated)

B Zone		
M-Duino Connector	*Arduino Pin*	Function [2]
GND	*GND*	GND
A0.7 [2]	*6*	Analog Out
A0.6 [2]	*5*	Analog Out
A0.5 [2]	*4*	Analog Out
Q/Vdc	-	External Isolated Out Vdc
COM(-)	-	External Isolated Out Gnd
Q0.7	*6*	Digital/PWM Out
Q0.6	*5*	Digital/PWM Out
Q0.5	*4*	Digital/PWM Out
Q0.4	*40*	Digital Out
Q0.3	*39*	Digital Out
Q0.2	*38*	Digital Out
Q0.1	*37*	Digital Out
Q0.0	*36*	Digital Out



Configuration Switch*
(see section 8 select correct configuration for outputs).

ANALOG Outputs

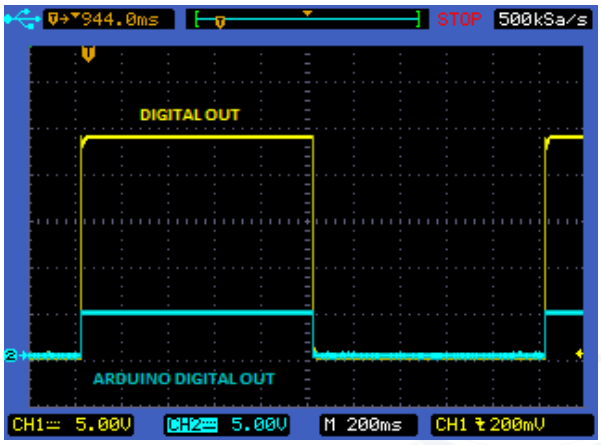
VOLTAGE SUPPLY/REFERENCE for DIGITAL/PWM Outputs (isolated)

PWM/DIGITAL Outputs

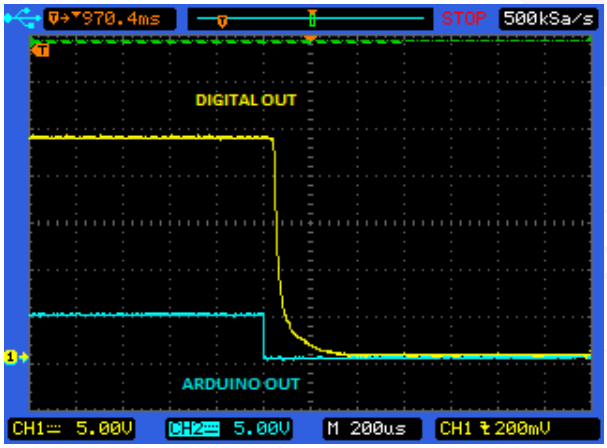
[2] See section 8 to select suitable switch configuration for (10-24Vdc/An-Dig) configurable I/Os.
[3] See section 8 to enable these connections.

10 I/O technical details:

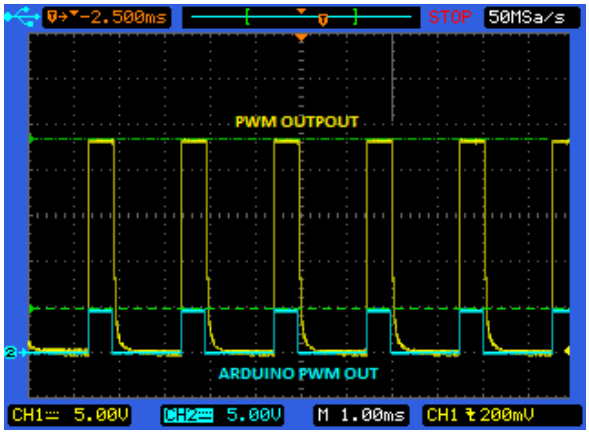
Digital Output Waveform:



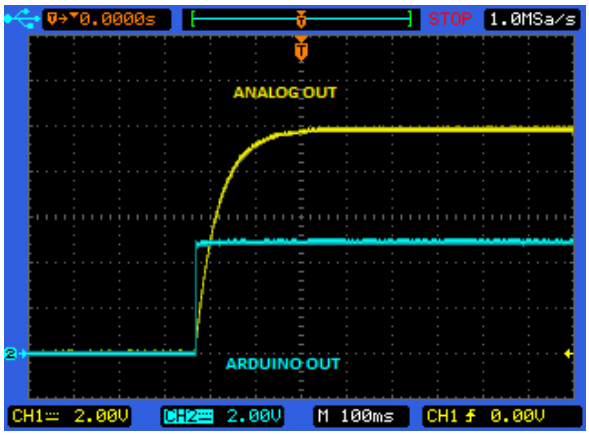
Digital Out-put Turn-off:



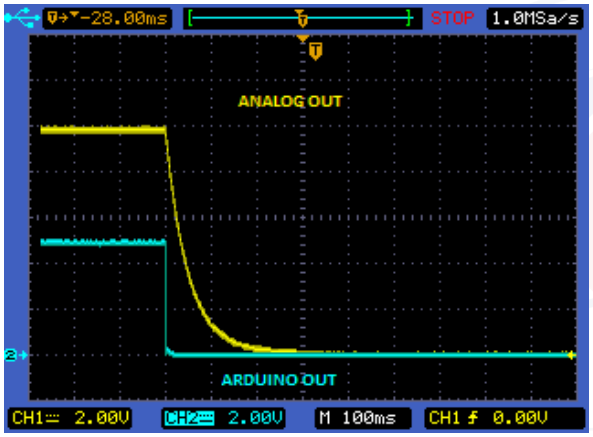
PWM Waveform:



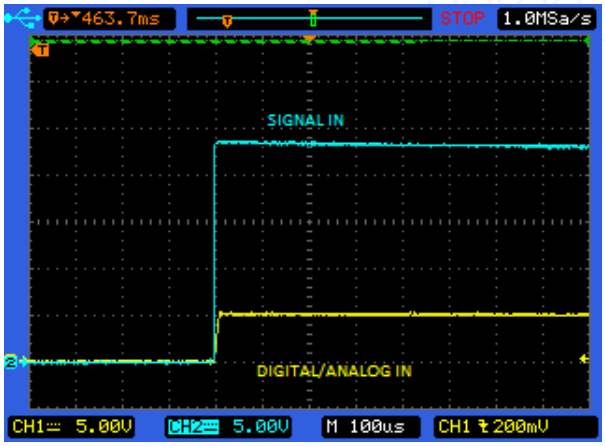
Analog Out Turn On:



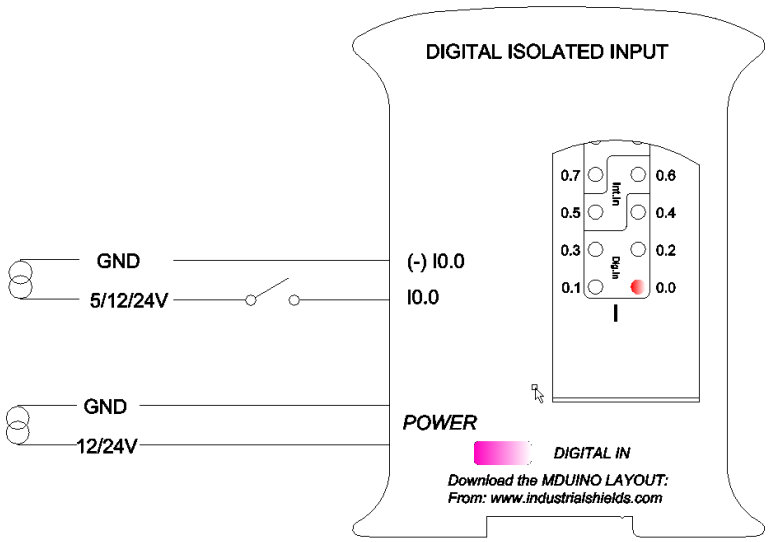
Analog Out Turn-Off:

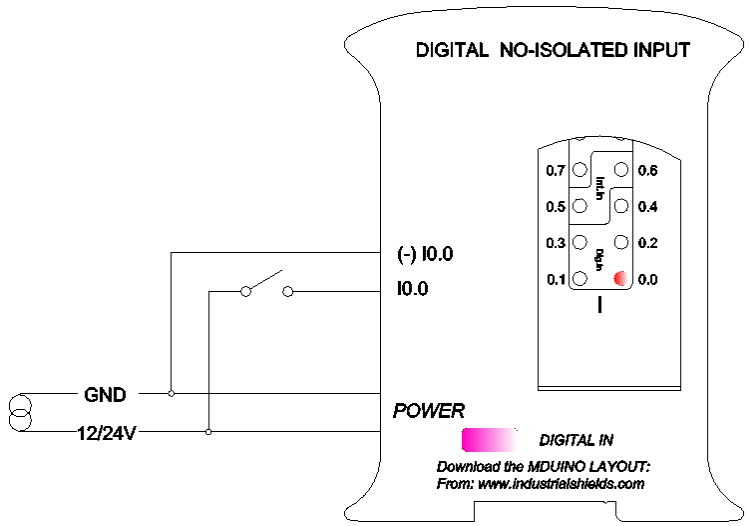


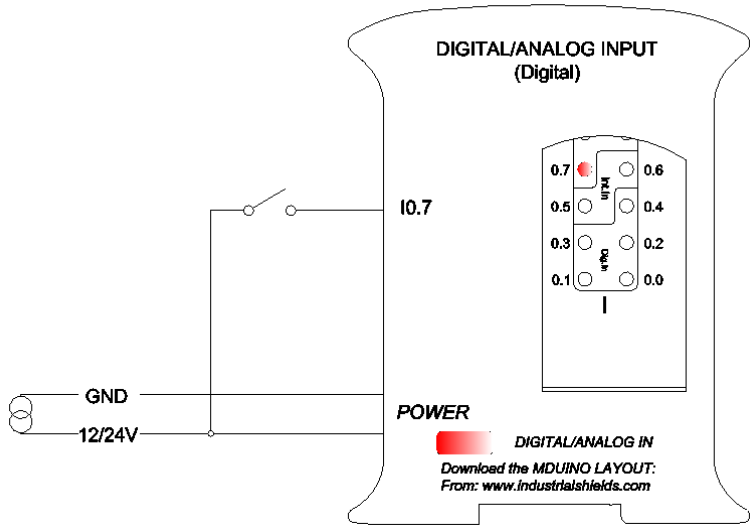
Analog /Digital input Turn-on:



9. Typical Connections

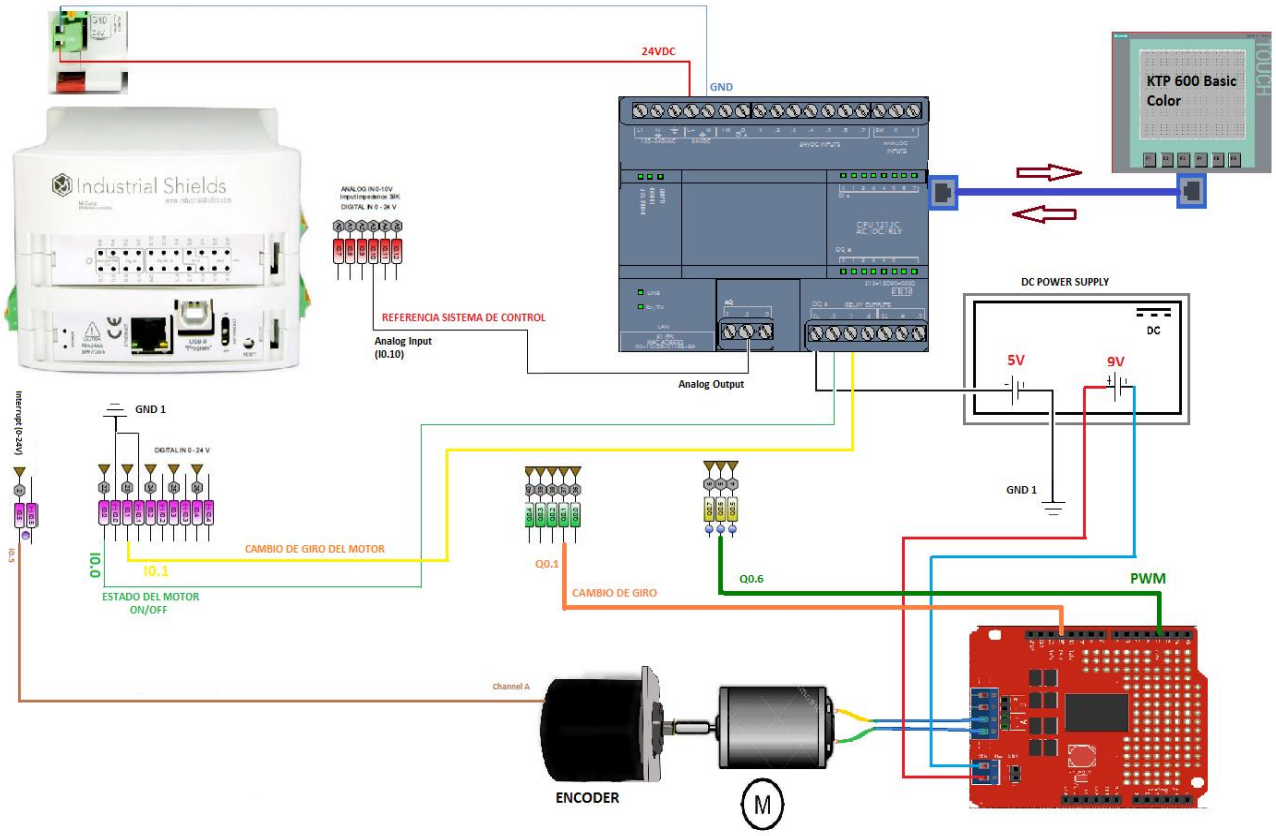






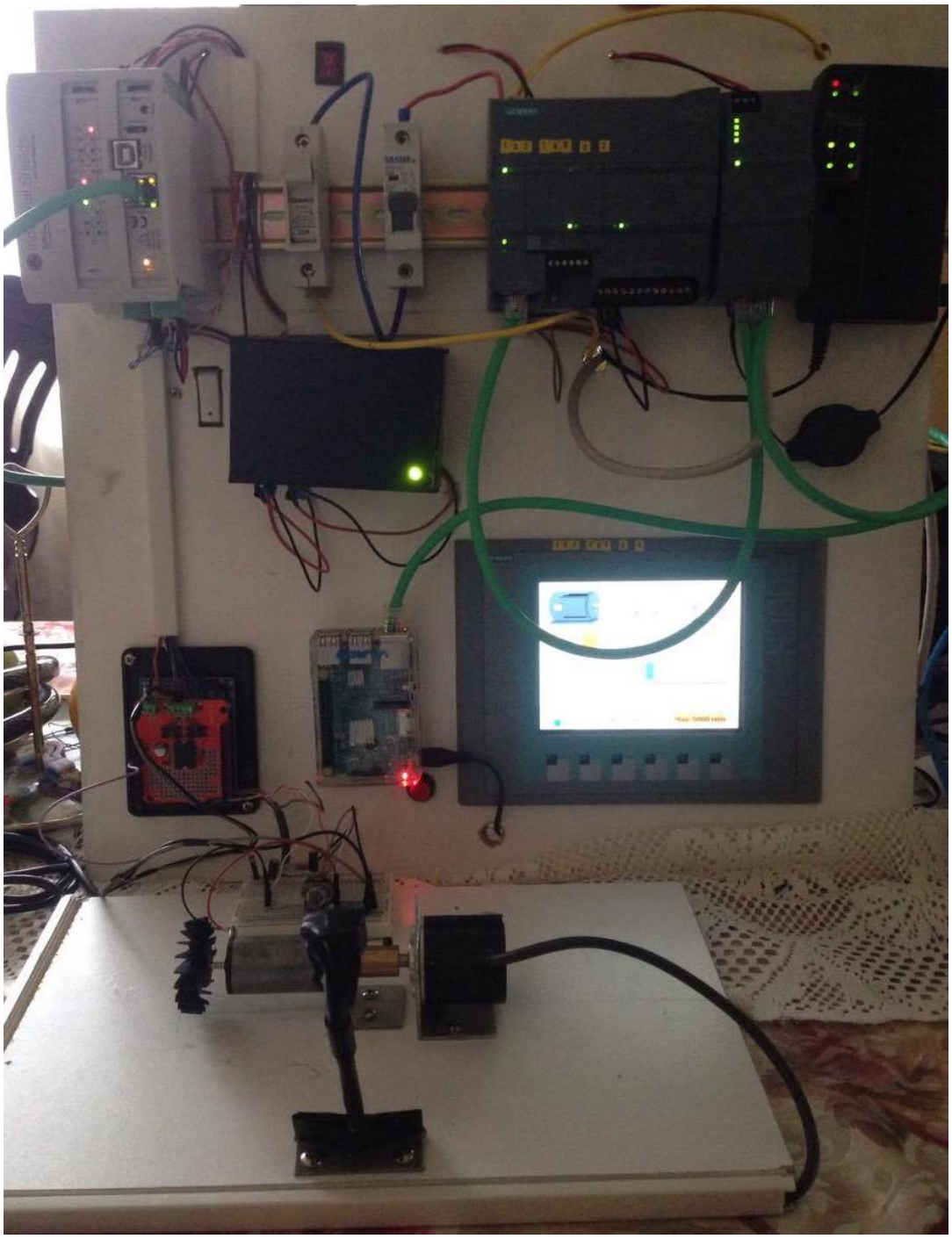
Apéndice E

Esquema de conexiones del prototipo


24VDC
GND
KTP 600 Basic Color
TOUCH
Industrial Shields
REFERENCIA SISTEMA DE CONTROL
Analog Input
(I0.10)
Analog Output
DC POWER SUPPLY
DC
5V
9V
GND 1
GND 1
DIGITAL IN 0 - 24 V
I0.0
I0.1
ESTADO DEL MOTOR
ON/OFF
CAMBIO DE GIRO DEL MOTOR
Q0.1
CAMBIO DE GIRO
Q0.6
PWM
Channel A
ENCODER
M

Apéndice F

Ensamblado del prototipo



Bibliografía

[1] Abdennebi TALBI Abdelhafid RACHIDI, Abdellah KHATORY. Toward an automation increasingly interconnecting. *International Journal of Scientific and Engineering Research*, 2014.

[2] Ronald L Krutz. *Securing SCADA systems*. John Wiley & Sons, 2005.

[3] David Bailey and Edwin Wright. *Practical SCADA for industry*. Newnes, 2003.

[4] Juan Carlos Solano Minchalo Edwin Patricio Patiño Ramón. Diseño e implementación de un prototipo de supervisión de un sistema de control industrial utilizando plataformas empotradas de bajo costo y controladores lógicos programables PLCs., 2016.

[5] Mohamed Endi, YZ Elhalwagy, et al. Three-layer plc/scada system architecture in process automation and data monitoring. In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, volume 2, pages 774–779. IEEE, 2010.

[6] Luis I Minchala, Saul Ochoa, Esteban Velecela, Darwin F Astudillo, and Javier Gonzalez. An open source scada system to implement advanced computer integrated manufacturing. *IEEE Latin America Transactions*, 14(12):4657–4662, 2016.

[7] O Barana, P Barbato, M Breda, R Capobianco, A Luchetta, F Molon, M Moressa, P Simionato, C Taliercio, and E Zampiva. Comparison between commercial and open-source scada packages—a case study. *Fusion Engineering and Design*, 85(3):491–495, 2010.

[8] Marcelo V García, Edurne Irisarri, Federico Pérez, Elisabet Estévez, and Marga Marcos. Opc-ua communications integration using a cpps architecture. In *Ecuador Technical Chapters Meeting (ETCM), IEEE*, volume 1, pages 1–6. IEEE, 2016.

[9] Velásquez Antonio, José Costa. Computer integrated Diseño Asistido por Computadora Fabricación Asistida por Computadora. page 3, 2008.

[10] Esteban Eduardo Velecela Gallegos Saúl Genaro Ochoa Ochoa. Propuesta e implementación de una solución alternativa para el control, supervisión y comunicación en procesos industriales mediante programas de código libre, 2015.

[11] Michael McCLELLAN. Introduction to manufacturing execution systems. In *MES Conference & Exposition, Baltimore, Maryland*, pages 1–7, 2001.

[12] A. G. Higuera and F. J. C. García. *CIM, El Computador en la automatización de la producción*. Universidad de Castilla de La Mancha, 2007.

[13] Helmut Klaus, Michael Rosemann, and Guy G Gable. What is erp? *Information systems frontiers*, 2(2):141–162, 2000.

[14] Y. Shimanuki. OLE for process control (OPC) for new industrial automation systems. *IEEE SMC'99 Conference Proceedings. 1999 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.99CH37028)*, 6:1048–1050, 1999.

[15] Li Zheng Li Zheng and H. Nakagawa. OPC (OLE for process control) specification and its developments. *Proceedings of the 41st SICE Annual Conference. SICE 2002.*, 2:917–920, 2002.

[16] Song Xin Jing Chunguo, Wang Yan. Development of an OPC Server for Remote Monitoring and Control Based on GPRS Networks. *The Tenth International Conference on Electronic Measurement and Instruments*, page 5, 2011.

[17] Serger Konstantinov Eduardo Moraes, Hernan Lepikson. Improving Connectivity for Runtime Simulation of Automation Systems via OPC UA. page 6.

[18] Supervisory Control and Data Acquisition (SCADA) Systems. *Technical Information Bulletin 04-1*, (October):76, 2004.

[19] A Daneels and W. Salter. What Is Scada ? *International Conference on Accelerator and Large Experimental Physics Control Systems, Trieste, Italy*, pages 339–343, 1999.

[20] Mohammad Salah. Programmable logic controller.

[21] Cristina Anita Bejan, Mihai Iacob, and Gheorghe-Daniel Andreescu. Scada automation system laboratory, elements and applications. In *Intelligent Systems*

and Informatics, 2009. SISY'09. 7th International Symposium on, pages 181–186. IEEE, 2009.

[22] Rao Kalapatapu. Scada protocols and communication trends. *ISA2004*, 2004.

[23] Rua Dr Antonio Bernardino de Almeida. Profibus protocol extensions for enabling inter-cell mobility in bridge-based hybrid wired/wireless networks. *Fieldbus Systems and Their Applications 2003*, page 277, 2003.

[24] Nicolas Navet, Y-Q Song, and Françoise Simonot. Worst-case deadline failure probability in real-time applications distributed over controller area network. *Journal of systems Architecture*, 46(7):607–617, 2000.

[25] Ken Tindell, H Hanssmon, and Andy J Wellings. Analysing real-time communications: Controller area network (can). In *RTSS*, pages 259–263, 1994.

[26] P. M. Meshram and R. G. Kanojiya. Tuning of pid controller using ziegler-nichols method for speed control of dc motor. In *IEEE-International Conference On Advances In Engineering, Science And Management (ICAESM -2012)*, pages 117–122, March 2012.

[27] WANG Hong-yu TIAN Zuo-hua and SHI Song-jiao WENG Zheng-xin. A fault detection and isolation scheme based on parity space method for discrete time-delay system.

[28] Rolf Isermann. *Fault-diagnosis systems: an introduction from fault detection to fault tolerance*. Springer Science & Business Media, 2006.