UNIVERSIDAD TÉCNICA DE AMBATO

FACULTAD DE INGENIERÍA EN SISTEMAS ELECTRÓNICA E INDUSTRIAL

CARRERA DE INGENIERÍA EN ELECTRÓNICA Y COMUNICACIONES

TEMA:

CONTROLADOR ELECTRÓNICO DE SOFTWARE LIBRE PARA LA OPERACIÓN DEL BRAZO ROBÓTICO SCORBOT ER_4U

Trabajo de graduación modalidad: Proyecto de Investigación, presentado previo a la obtención del título de Ingeniero en Electrónica y Comunicaciones.

SUBLÍNEA DE INVESTIGACIÓN: Robótica

AUTOR: Jennifer Elizabeth Chávez Chica

TUTOR: Ing. Franklin Salazar Mg.

AMBATO – ECUADOR

Octubre - 2020

APROBACIÓN DEL TUTOR

En calidad de tutor del trabajo de investigación sobre el tema: "CONTROLADOR ELECTRÓNICO DE SOFTWARE LIBRE PARA LA OPERACIÓN DEL BRAZO ROBÓTICO SCORBOT ER_4U", realizado por el señorita Jennifer Elizabeth Chávez Chica, estudiante de la carrera de Ingeniería en Electrónica y Comunicaciones de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, de la Universidad Técnica de Ambato, me permito indicar que el estudiante ha sido tutorado durante todo el desarrollo del trabajo hasta su conclusión, de acuerdo a lo dispuesto en el Artículo 15 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y el numeral 7.4 del respectivo instructivo.

Ambato Octubre, 2020

EL TUTOR

Firmado electrónicamente por:
FRANKLIN WILFRIDO SALAZAR LOGRONO

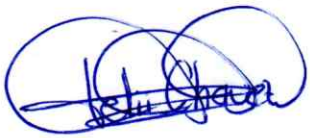
…….

Ing. Franklin Salazar Mg.

AUTORÍA

El presente proyecto de investigación titulado: "CONTROLADOR ELECTRÓNICO DE SOFTWARE LIBRE PARA LA OPERACIÓN DEL BRAZO ROBÓTICO SCORBOT ER_4U", es absolutamente original, auténtico y personal, en tal virtud, el contenido, efectos legales y académicos que se desprenden del mismo son de exclusiva responsabilidad del autor.

Ambato Octubre, 2020



.....

Jennifer Elizabeth Chávez Chica

CC: 1805385315

APROBACIÓN DEL TRIBUNAL DE GRADO

En calidad de par calificador del Informe Final del Trabajo de Titulación presentado por la señorita Jennifer Elizabeth Chávez Chica, estudiante de la Carrera de Ingeniería en Electrónica y Comunicaciones, de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial, bajo la Modalidad Proyecto de Investigación, titulado "CONTROLADOR ELECTRÓNICO DE SOFTWARE LIBRE PARA LA OPERACIÓN DEL BRAZO ROBÓTICO SCORBOT ER_4U, nos permitimos informar que el trabajo ha sido revisado y calificado de acuerdo al Artículo 17 del Reglamento para obtener el Título de Tercer Nivel, de Grado de la Universidad Técnica de Ambato, y al numeral 7.6 del respectivo instructivo. Para cuya constancia suscribimos, conjuntamente con la señora Presidenta del Tribunal.

Firmado electrónicamente por:
ELSA PILAR URRUTIA

.....

Ing. Elsa Pilar Urrutia Urrutia Mg.
PRESIDENTA DEL TRIBUNAL

Firmado electrónicamente por:
VICTOR SANTIAGO MANZANO VILLAFUERTE

.....

Ing. Santiago Manzano
DOCENTE CALIFICADOR

Firmado electrónicamente por:
EDGAR PATRICIO CORDOVA CORDOVA

.....

Ing. Patricio Córdova
DOCENTE CALIFICADOR

DERECHOS DE AUTOR

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

Cedo los derechos de mi Trabajo de Titulación en favor de la Universidad Técnica de Ambato, con fines de difusión pública. Además, autorizo su reproducción total o parcial dentro de las regulaciones de la institución.

Ambato, Octubre 2020.

.....

Jennifer Elizabeth Chávez Chica

CC: 1805385315

DEDICATORIA

Este estudio es dedicado de manera especial a mi padre Gonzalo Chávez, pues ha sido mi fuente de inspiración y de fuerza cuando he pensado en renunciar. Quien me da apoyo moral, espiritual, emocional y financiero.

Por mis hermanos Jason y Josué, mis tíos, primos, amigos y compañeros de clase con los que he compartido consejos para alentarnos a terminar este estudio.

Quiero dedicar también esta tesis a Dios y a mi Virgencita de Baños de Agua Santa, por la orientación, fuerza, poder mental, protección y por darme una vida saludable.

Jennifer Elizabeth Chávez Chica

AGRADECIMIENTO

Quisiera expresar mi sincero agradecimiento a todos los que me ayudaron, en completar con éxito mi trabajo de tesis.

Me gustaría agradecer a mi tutor Ing. Franklin Salazar, por su orientación y apoyo, sugerencias y estimulo en todo este proceso, de la misma manera a mi familia por su amor incondicional, apoyo continuo, paciencia y sobre todo por las palabras inspiradoras que me han ayudado para llegar a este punto.

Finalmente, me gustaría agradecer a mis amigos y a todos los que me han ayudado directa o indirectamente a colaborar para completar mi tesis.

Jennifer Elizabeth Chávez Chica

ÍNDICE GENERAL

PORTADA ..... **i**
APROBACIÓN DEL TUTOR ..... **ii**
AUTORÍA DEL TRABAJO DE TITULACIÓN ..... **iii**
APROBACIÓN DEL TRIBUNAL DE GRADO ..... **iv**
DEDICATORIA ..... **vi**
AGRADECIMIENTO ..... **vii**
ÍNDICE GENERAL ..... **viii**
ÍNDICE DE TABLAS ..... **xi**
ÍNDICE DE FIGURAS ..... **xii**
RESUMEN ..... **xiv**
ABSTRACT ..... **xv**
CAPÍTULO I **16**
MARCO TEÓRICO **16**
1.1 Antecedentes investigativos **16**
1.2 Contextualización del problema **18**
1.3 Fundamentación teórica **20**
1.3.1 Historia de la Robótica **20**
1.3.2 Robótica **21**
1.3.3 Robot **21**
1.3.4 Brazo Robótico **21**
1.3.5 Scorbot ER_4U **21**
1.3.6 Circuitos Electronicos **21**
1.3.7 Sisteas de Tarjeta Embebidas **30**
1.3.8 Hardware **31**
1.3.9 Tecnologías de Comunicación **30**
1.3.10 Protocolos de Comunicación **32**
1.3.11 Dispositivos Electrónicos **33**

Dispositivos Controladores **33**
Sensores de Corriente **37**
Puentes H **37**
1.4 Objetivos **38**
Objetivo general **38**
Objetivos específicos **38**
CAPÍTULO II **40**
METODOLOGÍA **40**
2.1 Materiales **40**
2.2 Métodos **41**
Modalidad de la Investigación **40**
Recolección de la Información **41**
Procesamiento y Análisis de Datos **41**
Desarrollo del Proyecto **41**
CAPÍTULO III **43**
RESULTADOS Y DISCUSIÓN **43**
3.1 INTRODUCCIÓN **43**
3.2 DESARROLLO DE LA PROPUESTA **44**
3.2.1 Brazo Robótico Scorbot ER_4U **44**
Hardware del Scorbot ER_4U **47**
a) Interfaz de Conexiones Eléctrica **47**
b) Características Eléctricas del Brazo **49**
Motores **49**
Microinterruptor **50**
Encoders **50**
Software de control del Scorbot ER_4U Scorbase **51**
a) Inicialización y Control. **52**
b) Grabación de Posiciones. **54**
3.2.2 Requerimientos Técnicos del Sistema **55**
3.2.3 Diseño General del Sistema **56**

Interfaz de Comunicación **57**
Interfaz Electrónica de Control **60**
3.2.4Arquitectura Final del Entorno de Control **61**
3.2.5 Módulo Electrónico **62**
Puentes H **68**
Sensores de Corriente **69**
Acondicionamiento de Entradas Digitales **72**
Reguladores de Voltaje **73**
Fuente de Poder **75**
Núcleo de Control **77**
Tarjeta Primaria **79**
Interfaz Angular **80**
3.2.6 Firmware de Control **81**
Firmware del Maestro **82**
Firmware del Esclavo **91**
3.2.7 Broker **94**
3.2.8 Host de Control **96**
3.3 ANALISIS DE PRESUPUESTO **104**
3.4 RESULTADOS **106**
CAPÍTULO IV **114**
CONCLUSIONES Y RECOMENDACIONES **114**
4.1 CONCLUSIONES **114**
4.2 RECOMENDACIONES **115**
Bibliografía **116**
ANEXOS **120**

ÍNDICE DE TABLAS

Tabla 3.1: Grados de libertad y funciones de los ejes de Scorbot ER_4U..... 46
Tabla 3.2: Descripción de las funciones de los pines del conector D50 del Scorbot ER_4U 32
Tabla 3.3: Mediciones promedio del consumo de corriente en los motores del Scorbot ER_4U. 49
Tabla 3.4: Voltajes y funciones de los pines de los encoders del Scorbot ER_4U. 50
Tabla 3.5: Funciones de los botones del panel de control de Scorbase 50
Tabla 3.6: Características técnicas de diferentes estándares de comunicación. 57
Tabla 3.7: Características de distintos protocolos de comunicación implementados sobre TCP/IP. 59
Tabla 3.8: Número de pines del microcontrolador requeridos para el módulo electrónico. 63
Tabla 3.9: Características técnicas de microcontroladores y módulos embebidos. 64
Tabla 3.10: Funciones y periféricos asignados a los microcontroladores del módulo electrónico. 65
Tabla 3.11: Características de operación de distintos puentes H. 68
Tabla 3.12: Características de los sensores de corriente. 69
Tabla 3.13: Fuentes requeridas para los componentes del módulo electrónico MQTT. 74
Tabla 3.14: Potencia estimada de consumo del módulo electrónico MQTT. 76
Tabla 3.15: Trama de recepcion UART del controlador maestro. 95
Tabla 3.16: Informamcion del byte de estado del motor. 95
Tabla 3.17: Bytes de payload pertenecientes a los sensores de corriente. 95
Tabla 3.18: Trama de recepcion de datos del micro controlador esclavo. 95
Tabla 3.19: Características de los servidores MQTT más reconocidos. 96
Tabla 3.20: Entornos de desarrollo para la interfaz HMI del controlador MQTT. 97
Tabla 3.21: Detalle del costo de materiales del módulo MQTT. 105
Tabla 3.22: Tiempos de respuesta de las peticiones de encendido y apagado de los motores. 108

ÍNDICE DE FIGURAS

Figura 1.1: Robots vedidos en America…… **7**
Figura 1.2: Brazo robot cartesiano…….. **23**
Figura 1.3: Brazo robot cilíndrico …….. **23**
Figura 1.4: Brazo robot esférico ……. **24**
Figura 1.5: Brazo robot SCARA……. **24**
Figura 1.6: Brazo robot articulado……..... **25**
Figura 1.7: Grados de libertad del Scorbot ER_4U…… **27**
Figura 1.8: Area de trabajo del Scorbot ER_4U……. **27**
Figura 1.9: Conector D50 del Scorbot……. **29**
Figura 1.10: Tarjeta ESP32 combinada con capacidad inalámbrica Wifi y Bluetooth **36**
Figura 1.11: Microcontrolador Pic…… **36**
Figura 1.12: Microcontrolador Arduino Uno…… **37**
Figura 1.13: Placa Raspberry…… **38**
Figura 1.13: Estructura de un Puente H……. **39**
Figura 3.1.: Diagrama de conexión del Scorbot ER_4U…… **44**
Figura 3.2 : Ejes de rotación del Scorbot ER_4U……... **45**
Figura 3.3 : Señal de pulsos de los encoders del Scorbot ER_4U…… **52**
Figura 3.4 : Interfaz gráfica de Scorebase que indica el proceso de referencia…….. **54**
Figura 3.5 : Cuadro de diálogo del movimiento de los ejes en Scorbase…… **55**
Figura 3.6 : Diagrama de bloques del sistema general…… **57**
Figura 3.7 : Interfaz electrónica del Scorbot ER_4U, terminal DD50…… **61**
Figura 3.8 : Arquitectura del entorno de control para el Scorbot ER_4U con interfaz Wifi – MQTT……... **62**
Figura 3.9 : Diagrama de bloques de módulo electrónico MQTT…….. **67**
Figura 3.10: Relación entre voltaje de salida y corriente medida en el sensor ACS712ELCTR-05BT…….. **71**
Figura 3.11: Circuito electrónico de puentes H y sensores de corriente……. **72**
Figura 3.12: Atenuadores tipo L para las entradas de los sensores del Scorbot…….. **72**
Figura 3.13: Reguladores de voltaje para alimentar los componentes…… **75**
Figura 3.14: Esquema eléctrico de las conexiones del núcleo de control…….. **62**
Figura 3.15: Imagen tridimensional de la tarjeta primaria…….. **79**
Figura 3.16: Esquema del circuito electrónico de la interfaz angular……. **80**

Figura 3.17: Visualización tridimensional de la interfaz angular…… **82**
Figura 3.18: Programa del Controlador Maestro……. **84**
Figura 3.19: Variables de suscripción del módulo MQTT……. **86**
Figura 3.20: Diagrama de flujo del firmware del esclavo……... **93**
Figura 3.21: Comandos para la Configuración de los repositorios……. **96**
Figura 3.22: Comando para la instalación de Emqx…… **97**
Figura 3.23: Modificaciones del archivo de configuración del bróker MQTT……... **97**
Figura 3.24: Funciones principales de node red utilizadas en la interfaz HMI……... **100**
Figura 3.25: Sección de desplazamiento de la interfaz HMI Demo…… **101**
Figura 3.26: Sección de posicionamiento de la interfaz Demo HMI…….. **102**
Figura 3.27: Panel de funciones de la interfaz Demo HMI…… **103**
Figura 3.28: Funcionamiento del sistema de control del Scorbot por MQTT……. **104**
Figura 3.29: Manipulación de Scorbot mediante MQTT…… **105**
Figura 3.30: Controlador MQTT del brazo robótico …….. **107**
Figura 3.31: Puerto de conexión del controlador MQTT…….. **108**
Figura 3.32 Tiempos de respuesta del encendido de los motores del brazo robótico. **110**
Figura 3.33: Tiempos de respuesta del cambio de velocidad de los motores …….. **112**
Figura 3.34: Tiempo de respuesta por eje…… **113**
Figura 3.35: Resumen general de la estadística de los tiempos de respuesta…….. **114**

RESUMEN

Un robot es un dispositivo mecánico que puede realizar tareas físicas bajo el control humano o bajo el control de una computadora preprogramada, por otra parte, los robots aplicados en educación o procesos industriales son de elevados costo, de igual forma, los equipos de código primitivo adquiridos limitan al usuario en la creación de aplicaciones con integración de nuevas tecnologías de comunicación, forzando a crear programas de bajo rendimiento en los tiempos de respuesta de interfaces de comunicación. El objetivo de esta investigación fue desarrollar un controlador electrónico para la operación del brazo robótico Scorbot ER_4U, los programas son elaborados en lenguaje C en el entorno de desarrollo ESP-IDF, el cual permitió la manipulación y control del brazo Scorbot ER_4U.

El desarrollo de este sistema alternativo para el control del Scorbot ER_4U, benefició de manera directa a los estudiantes, docentes e investigadores de la Universidad Técnica de Ambato, entregando un diseño de licencia abierta para la modificación de códigos y circuitos que apoyan a la evolución del conocimiento de la robótica. El proyecto de investigación utilizó software y hardware libre como fundamento para el desarrollo de los módulos requeridos en el sistema implementado, además, los elementos electrónicos y circuitos integrados requeridos fueron de carácter accesible al mercado ecuatoriano, ya sea por medio del comercio local o bajo importaciones. De esta manera el desarrollo de la investigación fue factible ya que se tuvo un soporte tecnológico inicial y la accesibilidad a los medios requeridos.

Palabras claves: Scorbot ER_4U, ESP 32, Wifi, MQTT, Servo motores DC.

ABSTRACT

A robot is a mechanical device that can perform physical tasks under human control or under the control of a pre-programmed computer, on the other hand, robots applied in education or industrial processes are high cost, likewise, the primitive code equipment acquired limits the user in creating applications with integration of new communication technologies, forcing to create programs of low performance in response times of communication interfaces. The objective of this research was to develop an electronic controller for the operation of the Scorbot ER_4U robotic arm. The programs are elaborated in C language in the ESP-IDF development environment, which allowed the manipulation and control of the Scorbot ER_4U arm.

The development of this alternative system for the control of the Scorbot ER_4U, directly benefited the students, teachers and researchers at the Technical University of Ambato, delivering an open license design for the modification of codes and circuits that support the evolution of knowledge in robotics. The research project used free software and hardware as the basis for the development of the modules required in the implemented system. In addition, the required electronic elements and integrated circuits were accessible to the Ecuadorian market, either through local commerce or under imports. In this way, the development of the research was feasible since there was an initial technological support and the accessibility to the required means.

Keywords: Scorbot ER_4U, ESP 32, Wifi, MQTT, DC servo motors.

CAPÍTULO I

MARCO TEÓRICO

1.1 Antecedentes investigativos

La automatización industrial es un conjunto de tecnologías que utilizan sistemas y equipos de control como computadoras o robots, los cuales permiten la operación automática de procesos dando como resultado beneficios en la productividad, calidad, seguridad, agilidad dando así un mayor rendimiento en la producción, una reducción de costos y un ahorro de tiempo. En Ecuador y alrededor del mundo se han realizado múltiples trabajos de investigación relacionados con la robótica de forma general y proyectos específicos para robots Scorbot, los mismos se detallan a continuación.

En junio de 2014 la revista International Journal of Materials Science and Engineering publica un artículo de Prasad Vinayak Patil y Shantipal Suresh Ohol con el tema: "Performance Analysis of SCORBOT ER 4u Robot Arm". Se realiza el análisis del brazo de este robot y se centra principalmente en su rentabilidad mediante el uso de métodos de re-acondicionamiento y programación. Como resultados se observa que, los gráficos de posición obtenidos por RoboCell, muestran que el robot funciona correctamente y con precisión, sin embargo, se produce un error de tiempo de ejecución durante el funcionamiento del brazo del robot [1].

En el año 2016, David Ortiz y Sergi Bermejo presentan una tesis con el tema: "Robótica para el Seguimiento de Líneas". Se realizó un robot que es diseñado para adquirir señales desde un dispositivo de carga acoplada (sensor CCD) y una cámara digital, obteniendo imágenes que permiten detectar líneas, colores y objetos. La funcionalidad del robot es de alta flexibilidad, el mismo permite la adición de nuevos sensores para dar más autonomía al robot. El proyecto de investigación desarrollado es una base para realizar una máquina autónoma, que funciona de manera similar a otros vehículos autónomos [2].

En el año 2017 la revista HindawiJournal of Robotics, publica un artículo científico de los autores J. Aroca Trujillo, A. Pérez Ruiz y R. Rodriguez Serrezuela con el tema: "Generation and Control of Basic Geometric Trajectories for a Robot Manipulator Using CompactRIOD", se realiza la implementación de tres tipos de algoritmos para generación de rutas. El primer algoritmo desarrolla una ruta conjunta, en la que el efector final no tiene ninguna relevancia. Los otros dos generan trayectorias cartesianas que dibujan con el efector final la forma de una línea o un arco. La programación se ejecuta a través de LabVIEW y se implementa en la misma una FPGA CompactRIOD. El usuario interactúa con el robot a través de una computadora conectada remotamente al puerto Ethernet, lo que facilita la visualización, creación y modificación de las tareas que el usuario desea replicar en el manipulador [3].

En el año 2018 la revista Contemporary Engineering Sciences publica un artículo científico de J. Aroca Trujillo y R. Rodriguez Serrezuela con el tema: "Kinematic Model of the Scorbot 4PC Manipulator Implemented in Matlab's Guide", se realiza el desarrollo y la implementación de los modelos de cinemática directa e inversa del robot manipulador de cinco grados de libertad Scorbot ER 4PC, bajo una interfaz diseñada en Matlab. Los modelos se procesan en Matlab, donde tienen una interfaz gráfica desarrollada en la GUIDE. Esto permite la verificación de la teoría y al mismo tiempo puede modificar los valores de las variables conjuntas, así como modificar la posición y orientación del efector final (griper) [4].

En abril de 2019, Franklin Salazar L. y Jorge Buele publican un artículo científico en la revista International Journal of Innovative Technology and Exploring Engineering con el tema: "Teleoperation and Remote Monitoring of a ScorBot ER-4U Robotic Arm in an Academic Environment", se realiza una investigación de un método de programación implementado para la teleoperación de un ScorBot ER-4U. El objetivo es que un operador humano tenga la posibilidad de crear rutinas para el robot desde un sitio remoto. Para lograr el propósito de las telecomunicaciones, se desarrollaron dos interfaces llamadas cliente y servidor en el software LabVIEW; que procesan la información a través de un protocolo de comunicación TCP / IP. El inconveniente del proyecto son los retrasos en la ejecución de los movimientos, producidos por las características del entorno de manejo y programación del robot [5].

1.2 Contextualización del problema

A nivel global, el avance tecnológico de las últimas décadas ha entregado expectativas de una transformación científica de pasos acelerados, es así que en los últimos 30 años se ha pasado de utilizar robustos computadores de escritorio a alcanzar una era digital en la que se usan equipos de bolsillo con altas características de procesamiento de datos de forma específica en el área de la robótica, en donde los investigadores empezaban sus trabajos en área industrial, migraron sus investigaciones hacia aplicaciones particulares con robots móviles destinados a trabajar fuera del área de manufactura [6], [7].

El diseño de sistemas autónomos inteligentes requiere de la integración de múltiples disciplinas, en donde se agrupan áreas como la electrónica, mecánica, ciencias físicas y computacionales para crear módulos y aplicaciones que ejecuten tareas sencillas como limpieza de pisos, transporte de material, trabajos de servicio en hospitales entre otros y actividades de alto riesgo con manipulación de elementos químicos peligrosos o exploración del espacio estelar. El conocimiento y estructura empleados en los diseños por la mayoría de empresas privadas, son de código cerrado, limitando a los investigadores externos la creación de nuevas características, retardando a la vez la evolución de los equipos diseñados [7].

En el área educativa el panorama es similar, la construcción de un robot requiere del dominio de mecánica para construir la estructura, electricidad para animarlo, electrónica y comunicaciones para establecer interfaces de control que permitan al humano enviar información, estableciendo un lenguaje entre el hombre y la máquina y conocimientos de programación para elaborar software que ejecuta las rutinas del robot. El desarrollo de las tecnologías de la información y comunicaciones ha llevado a establecer medios que permiten que cualquier objeto establezca conectividad con la Internet, así se percibe el auge del internet de las cosas o IOT. Los robots de aplicaciones educativas e industriales son de elevados costos, de la misma manera en instituciones educativas y de manufactura del Ecuador, se han adquirido con anterioridad robots para la investigación y desarrollo de la automatización; los equipos

de código privativo adquiridos limitan al usuario en la creación de aplicaciones con integración de nuevas tecnologías de comunicación, forzando a crear programas de bajo rendimiento en los tiempos de respuesta de interfaces de comunicación. Esto debido a que se requiere utilizar software y hardware controlador diseñado por el fabricante, mismo que para integrar aplicaciones no desarrolladas en implican en la creación de software de interfaz adicional; negando la programación directa de las rutinas en el hardware controlador del propietario [8], [9].

En la Universidad Técnica de Ambato se tiene un equipo Scorbot ER_4U, utilizado en el ámbito de enseñanza e investigación. El equipo en mención utiliza un módulo controlador de propietario en conjunto al software Scorbase para crear el firmware de ejecución de rutinas del brazo robótico. El software propietario limita la ejecución de acciones directas sobre el brazo robótico para utilizar protocolos de comunicaciones orientados a capas superiores a la de red del modelo OSI; es así que para desarrollar aplicaciones por ejemplo con TCP/IP, MQTT, websockets o protocolos similares orientados la conexión a internet, el programador debe utilizar de forma obligatoria una rutina de Scorbase que haga de interfaz entre la aplicación del nuevo protocolo, el módulo controlador del brazo.

Otra limitante en el área educativa que se tiene en el robot a la Universidad, es el uso de licencias, la misma que limita a los estudiantes el desarrollo libre de aplicaciones; el costo de la licencia de Scorbase representa una limitante al estudiante para ejecutar programas en los equipos personales. En adición los costos económicos e intelectuales del mantenimiento preventivo o correctivo del controlador de propietario del robot son elevados, ya que el propietario no entrega la información del software utilizado en el microcontrolador.

Finalmente se ha identificado que el hardware controlador de propietario del brazo robótico es demasiado grande y pesado, creando un esquema de estorbo e incomodidad en las rutinas de prueba desarrolladas.

1.3 Fundamentación teórica

La implementación de sistemas automatizados con células de fabricación robotizadas para el mejoramiento de procesos productivos a nivel mundial se detalla a continuación:

1.3.1 Historia de la Robótica

La historia del robot surgió de las novelas de ficción, donde los hombres máquinas imitaban a los humanos y sus acciones. Mary Shelly en Inglaterra publicó una novela en 1817 titulada "Frankenstein" que trataba sobre la historia de un científico que quería crear un monstruo humano. Un dramaturgo checoslovaco en 1922 escribió una historia a la que llamo "Rossunis Universal Robot" e introdujo la palabra robota que significa, un trabajador esclavo. Cuando Robota se tradujo al inglés, la palabra se convirtió en Robot [10].

En 1954, George Devol desarrolló el primer robot programable. En 1961, se emitió su patente estadounidense a George Devol. El primer robot industrial apareció en 1962 en Gencral Motors, EE. UU. En 1968, se construyó un robot inteligente llamado Shakey en el Stanford Research Institute (SRI) que tiene tres movimientos de rotación llamados brazo articulado. En 1972, IBM desarrolló un robot de coordenadas rectangular llamado IBM7565, es un robot cartesiano. En 1978, Unimation desarrolló un robot polar llamado PUMA. La mayoría de las empresas que fabricaron robots a mediados de 1980 ya no existen, excepto las que fabricaron robots industriales. Las empresas son Adapte Robotics, Stanford Robots. Robots Funuc y Norteamérica, Inc. Robots. Desde 1983 en adelante, la robótica se convirtió en un tema muy popular, tanto en la industria como en la academia [10].

La robótica es una tecnología con futuro. Los robots del futuro serán unidades móviles con uno o más brazos, capacidades de sensores múltiples y la potencia de procesamiento de datos computacionales. Serán capaces de responder al comando de voz humana. Podrán recibir instrucciones generales usando inteligencia artificial en un conjunto específico de acciones necesarias para llevarlas a cabo. Podrán ver, escuchar, sentir y aplicar una fuerza medida precisa al objeto y moverse bajo su propio poder. En breve, los futuros robots tendrán muchos atributos de los seres humanos. Pasando

del presente al futuro, la tecnología de robots requiere mucho desarrollo a través de la ingeniería mecánica, ingeniería eléctrica, informática, ingeniería industrial, tecnología de materiales, ingeniería de sistemas de fabricación y ciencias sociales [10].

1.3.2 Robótica

La robótica es una ciencia de ingeniería aplicada que se conoce como una combinación de tecnología de máquina y ciencias de la computación. Están inmersos en diversos campos como diseño de máquinas, teoría de control, microelectrónica, programación de computadoras, inteligencia artificial, factores humanos y teoría de la producción. El avance de la tecnología ampliará el alcance de aplicaciones industriales de los robots. Los robots son elementos muy poderosos de la industria actual, son capaces de realizar muchas tareas y operaciones diferentes con precisión y no requieren de seguridad y comodidad comunes que los humanos necesitan. El tema de robótica cubre muchas áreas diferentes, ya que se utilizan junto con otros dispositivos y periféricos. Generalmente están integrados en un sistema, que en su conjunto está diseñado para realizar una tarea o realizar una operación [10].

1.3.3 Robot

Los robots son máquinas que se pueden usar para hacer trabajos. Algunos robots son autónomas o tienen una persona que les diga que hacer, los robots pueden moverse dentro de su entorno físico y manipular objetos, tienen cuatro características principales: movilidad, interactividad, comunicación y autonomía. La palabra robótica abarca todos los campos relevantes para la robótica, como la visión artificial y por computadora, la inteligencia artificial y el aprendizaje automático, así como la automatización y los sistemas autónomos [11].

El uso de los robots permite mejorar la calidad de productos de manufactura y reducir los riesgos laborales, reduciendo así los costos de producción. Varios países del mundo invierten significativamente en el uso de nueva tecnología de producción, bien es así que Estados Unidos está invirtiendo cada vez más en nuevas formas de automatización para así poder aumentar y mejorar su productividad y reducir sus costos de fabricación. Los robots pueden trabajar de manera independiente o asistida por una computadora.

En la figura 1.1, se presenta un incremento de las ventas anuales de los robots en América, los robots se han convertido en una tendencia, utilizados principalmente en la automatización. En varios países los robots están reemplazando a las personas para llevar a cabo su trabajo, debido a que permite un aumento en ganancias, reducción de costos laborales, eliminación de trabajos peligrosos y permite una mejora en la calidad del producto [12].

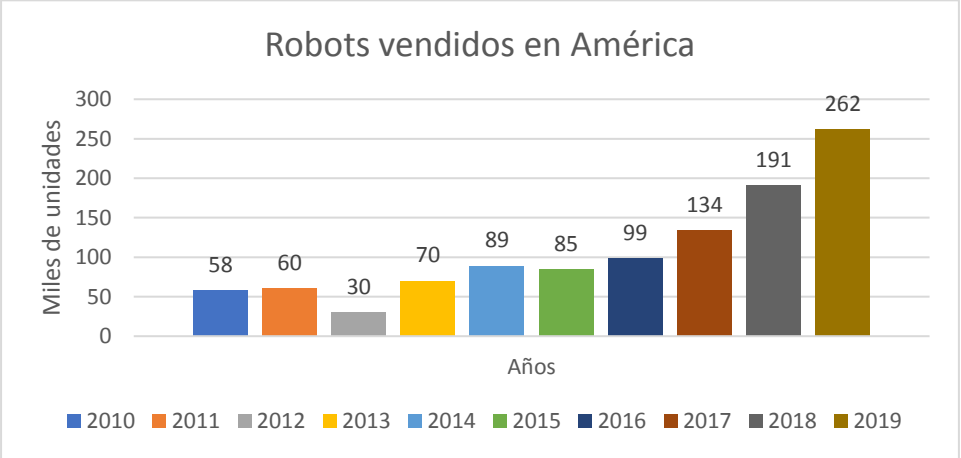


Figura 1.1: Robots vendidos en América.

Elaborado por: El Investigador en base a [12].

1.3.4 Brazo Robótico

El brazo robótico generalmente es programable con funciones similares a las de un brazo humano, posee una estructura mecánica que es diseñado para sostener, mover y agarrar objetos. La conexión para el movimiento del autómata permite un movimiento de rotación o de desplazamiento lineal [13].

Los brazos robóticos están divididos de la siguiente manera de acuerdo a su estructura mecánica:

✓ **Robot cartesiano:** El robot cartesiano es uno de los robots más utilizados, en la figura 1.2 se muestra que el brazo posee tres articulaciones cuyos ejes coinciden con unos sistemas de coordenadas cartesianas, estos robots son utilizados principalmente en el trabajo de colocación y recolección, aplicación como selladora o soldadura por arco, etc [14].

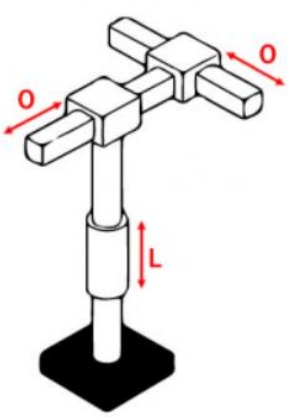


Figura 1.2: Brazo robot cartesiano [14].

✓ **Robot cilíndrico:** El robot cilíndrico es un robot que posee varias articulaciones que giran sobre una varilla fija como se muestra en la figura 1.3, el robot cilíndrico tiene tres ejes, dos son ejes lineales y un eje circular. Este tipo de robots cilíndricos son utilizados en operaciones de ensamblaje, soldadura, y en aplicaciones en donde se realizan tareas repetitivas [14].

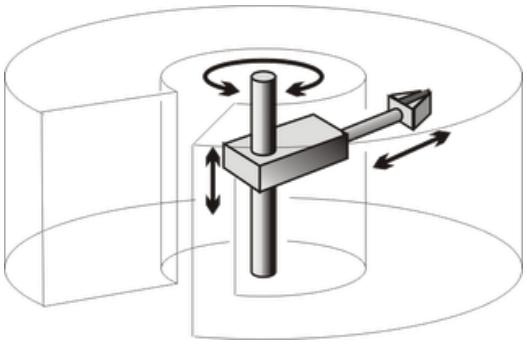


Figura 1.3: Brazo robot cilíndrico [14].

✓ **Robot esférico o polar:** El brazo esférico o polar son robots que permiten el movimiento de la rotación del brazo, en la figura 1.4, se observa que poseen articulaciones que permiten una rotación completa a lo largo de un rango esférico. Generalmente son utilizados en soldadura y principalmente en industria manufacturera [14].

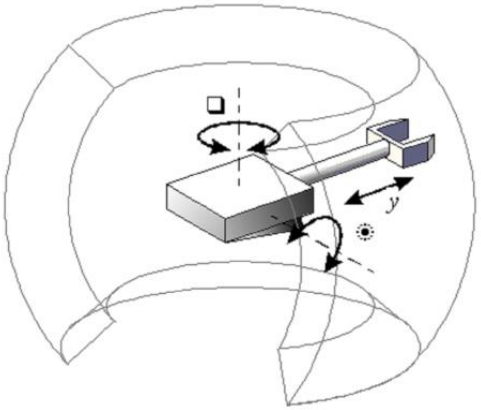


Figura 1.4: Brazo robot esférico [14].

✓ **Robot SCARA:** El robot SCARA tiene una configuración que permite realizar trabajos de ensamblaje moderno donde se necesita movimientos rápidos. Posee una combinación de un eje lineal que se mueve verticalmente y de dos ejes de rotación horizontales [14].

Figura 1.5: Brazo robot SCARA [14].

✓ **Robots articulados:** Son considerados robots articulados a cualquier robot cuyo brazo tenga al menos tres articulaciones, en la figura 1.6, se presenta un robot articulado, se utiliza para aplicaciones complejas donde se levanta objetos pesados, operaciones como soldadura, perforación o donde se manipulan químicos [14].

Figura 1.6: Brazo robot articulado [14].

1.3.5 Scorbot ER_4U

El brazo del robot SCORBOT-ER 4u es un sistema versátil y confiable para uso educativo que se puede montar sobre una mesa, un pedestal o una base deslizante lineal. La velocidad y la repetitividad del robot lo hacen muy adecuado tanto para operaciones independientes como para uso integrado en celdas de trabajo automatizadas y aplicaciones de manufactura simple como soldadura robótica, visión artificial y Máquina CNC [15].

- **Estructura Física del Scorbot ER_4U**

El SCORBOT-ER 4U fue diseñado y desarrollado para emular un robot industrial. La estructura abierta del brazo del robot permite a los estudiantes observar y aprender sobre sus mecanismos internos. El brazo es un robot articulado vertical, con cinco articulaciones giratorias y una pinza adicional conectada a modo de manipulador, teniendo así seis grados de libertad; cómo se puede visualizar en la Figura 1.7. Este diseño permite que el efector final se posicione y oriente arbitrariamente dentro de un gran espacio de trabajo [16].

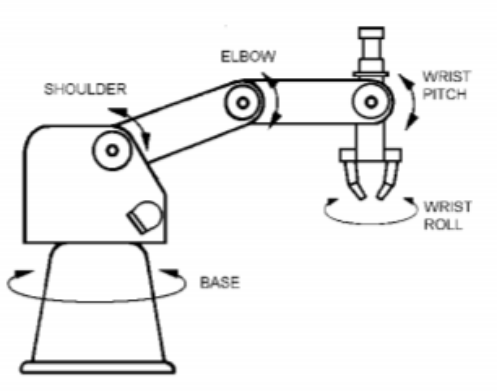


Figura 1.7: Grados de libertad del Scorbot ER_4U trabajo [16].

La longitud de los enlaces y el grado de rotación de las articulaciones determinan la envolvente de trabajo del robot. La Figura 1.8 presenta las dimensiones y el alcance del SCORBOT-ER 4u. La base del robot normalmente está fijada a una superficie de trabajo estacionaria. Sin embargo, puede estar conectado a una base deslizante, lo que resulta en un rango de trabajo extendido [16].

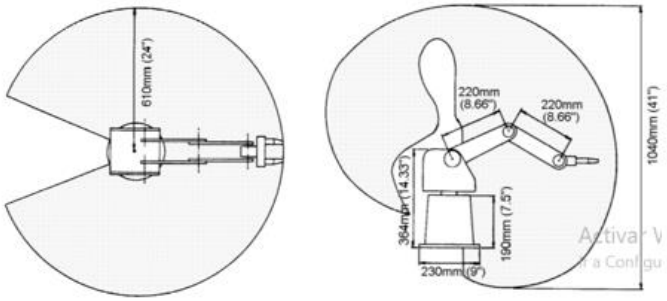


Figura 1.8: Área de trabajo del Scorbot ER_4U [16].

- **Motores del Scorbot ER_4U**

Los cinco ejes y la pinza del robot son operados por servomotores de corriente continua. La dirección de la revolución del motor está determinada por la polaridad del voltaje de funcionamiento: con el voltaje de corriente continua positiva gira el motor en una dirección, mientras a un voltaje de corriente continua negativa lo gira en la dirección opuesta El voltaje de operación nominal de los motores es de 12V DC y los mismos consumen una corriente de 500mA para desplazar cualquiera de los ejes sin carga. Cada motor está equipado con un codificador para control de circuito cerrado [16].

- **Elementos de Sensorización del Scorbot ER_4U**

El brazo robótico Scorbot ER_4U tiene integrado elemento de sensorización para determinar la posición de los ejes, entre ellos se encuentran los encoders que indican las posiciones intermedias de los ejes y finales de carrera que detectan los límites de posición de cada eje. La ubicación y el movimiento de cada eje se miden mediante un codificador electroóptico unido al eje del motor que acciona el eje. Cuando el eje del robot se mueve, el codificador genera una serie de señales eléctricas altas y bajas alternadas. El número de señales es proporcional a la cantidad de movimiento del eje. La secuencia de las señales indica la dirección del movimiento. El controlador lee estas señales y determina la extensión y dirección del movimiento del eje [16].

El SCORBOT-ER 4u tiene cinco micro interruptores o finales de carrera, uno en cada eje, que sirven para identificar la posición inicial del robot. Durante el procedimiento de referencia, las articulaciones del robot se mueven de una en una. Cada eje se mueve hasta que se activa su interruptor de inicio. Luego, el eje se mueve ligeramente hasta que se apaga el interruptor; en ese punto, la articulación está en "home". Cuando todas las articulaciones están en "home", el robot está en "home". Este es el punto de referencia para la operación del robot. Cada vez que se enciende el sistema, el robot se debe enviar a esta posición, mediante una rutina de referencia de software [16].

- **Cableado del Scorbot**

El brazo del Scorbot está enlazado al controlador por medio de un cable que va desde la base del brazo del Scorbot al conector D50 que se muestra en la Figura 1.9. Los cables de los motores del Scorbot están enlazados directamente al conector D50. Estos cables son particularmente flexible y resistente a la rotura, incluso después de un movimiento extenso del brazo del Scorbot [17].

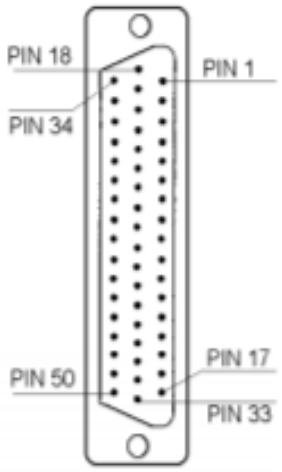


Figura 1.9: Conector D50 del Scorbot [17].

- **Características del Controlador-USB**

El controlador-USB, es una parte del sistema robótico Scorbot, es un microprocesador que tiene por objetivo realizar las tareas de control del brazo robótico. Esta tarjeta principal se conecta a un computador a través de un cable USB, el cual se controla por una fuente de alimentación de 24V para los motores del brazo y dos dispositivos periféricos. Además, posee 8 puertos de entrada digital y 8 puertos de salida digital con relay o con colector abierto, 4 puertos de entradas analógicas, 2 puertos de salidas analógicas y un botón de paro de emergencia e indicador LED [18].

- **Scorbase Software**

SCORBASE es un paquete de software de control robótico basado en Windows que ha sido diseñado para usarse con el SCORBOT-ER 4u. Su estructura basada en menús y capacidades fuera de línea facilita la programación y operación robótica. El programa se comunica con el controlador del robot por medio de un canal USB, operando en tres niveles: Nivel 1 y Nivel 2 se recomiendan para aquellos que desean aprender programación robótica desde las etapas más básicas mientras que el nivel Pro contiene comandos y opciones de programación para usuarios avanzados [19].

1.3.6 Circuitos Electrónicos

Los circuitos electrónicos son un conjunto de componentes interconectados entre sí, por medio de los cuales circulan corrientes eléctricas que representan información, que

se procesa o transmite hacia otros dispositivos. Los elementos electrónicos están compuestos en su mayoría por circuitos destinados a gestionar información codificada en una señal de voltaje [20].

- **Componentes Pasivos de Circuitos Eléctricos**

Los componentes pasivos de circuitos eléctricos son aquellos componentes que no tienen la capacidad de generar energía eléctrica, de esta forma, se requiere de una excitación de una corriente o tensión eléctrica externa para que actúen. Entre los elementos pasivos de un circuito se mencionan: Resistores, Capacitores e Inductores. Las resistencias o resistores son elementos que producen caídas de tensión, es decir limitan el paso de la corriente [21].

Los capacitores son dispositivos pasivos, utilizados en electricidad y electrónica, que tienen la capacidad de almacenar energía en forma de voltaje, sustentando un campo eléctrico. Estos elementos están formados por un par de superficies conductoras, en forma de láminas o placas paralelas, separadas por un material aislante o por la permitividad eléctrica del vacío. Los inductores son unos componentes utilizados en circuitos eléctricos y electrónicos, formados por espiras de material conductor. De forma concreta los inductores también llamados bobinas o solenoides, inducen un campo magnético cuando son atravesados por una corriente. El alambre enrollado es aislado con esmalte con la finalidad de que entre en cortocircuito al hacer contacto consigo mismo [21].

1.3.7 Sistemas de Tarjeta Embebidas

Las tarjetas embebidas en sistemas electrónicos, constituyen circuitos electrónicos y digitales que tienen por objeto controlar de forma automática procesos, máquinas u operadores. En todo sistema electrónico se tiene dispositivos de entrada, salida y de proceso. Los dispositivos de entrada entregan una señal eléctrica a partir de una señal física externa (por ejemplo, la humedad, el peso, el accionar de un pulsador). Los elementos de proceso reciben las señales de los dispositivos de entrada y deciden que acción a realizar ante el estímulo recibido. La función de los periféricos de salida es ejecutar las acciones decididas en el proceso [22].

1.3.8 Hardware

El hardware es un elemento físico de una computadora, que se encarga del procesamiento de datos y almacenamiento de estos. Posee periféricos de entrada para permitir el ingreso de datos y periféricos de salida, que posibilita la salida a los datos procesados. Es una unión de equipos mecánicos, magnéticos, eléctricos y electrónicos de una computadora para el procesamiento de la información general. Existen diversas unidades de micros controladores, de acuerdo a su gama de MCU con diferente rendimiento, RAM, memoria Flash, tamaños y conjuntos de protocolos de comunicación.

1.3.9 Tecnologías de Comunicación

- **Tecnologías de Comunicación Alámbrica**

La tecnología de comunicación alámbrica, utilizan soporte físico para la transmisión de información entre dos o más computadoras, impresoras y otros dispositivos conectados por cable Ethernet, son llamadas también redes Ethernet. Teniendo características como; su alta velocidad, seguridad y es utilizado para distancias comparativamente cortas.

Los tipos principales de medios guiados son:

✓ **Ethernet:** Ethernet, se desarrolló a partir de 1972 por los ingenieros Bob Metcalfe y Dr Boggs. Es una tecnología de redes de área local (LAN) más simples y rentables que existen actualmente, debido a su alta velocidad de transmisión. Emplea una topología tipo bus, que permite que todos los dispositivos se conecten a una misma línea de transmisión de la información como también las topologías tipo estrella o árbol, que dependen del cable a utilizar y de otros factores [23].

✓ **RS232:** RS 232, es un protocolo de comunicación serial donde se envía un bit a lo largo de una línea, a la vez, usado principalmente para conectar equipos externos a computadoras. Una de las principales ventajas que tiene la comunicación en serie sobre las comunicaciones en paralelas es que se necesita de un solo cable para la transmisión y recepción de datos. Permitiendo una velocidad de bits de 19600bps para una distancia máxima de 20 metros [24]. Sin embargo, una de las desventajas de este

protocolo se debe a los niveles de voltaje que no son compatibles con las tecnologías modernas TTL o CMOS, ya que se necesita un convertidor de nivel externo. Para establecer la comunicación bidireccional, se necesita las señales de control, cables de Tx, Rx, y GND. El protocolo RS 232, sigue la comunicación asíncrona, es decir, no hay señal de reloj para sincronizar la transmisión con el receptor. Este protocolo consiste en un conector tipo DB25 de 25 pines y DB9 de 9 pines [25].

✓ **RS485:** El RS485 posee hasta 32 pares de transmisor y receptor, es un sistema semidúplex de 2 hilos. Permite la conexión de hasta 32 transmisores con 32 receptores, con una transmisión simultanea full duplex. Las características eléctricas del conector poseen un margen de voltaje para trabajar cuando el emisor operan en estados de "1" lógico de -1.5V a -5V y con "0" lógico en el rango de +0.2V a +12V.

- **Tecnologías de Comunicación Inalámbrica**

La tecnología de comunicación inalámbrica permite la comunicación entre uno o más dispositivos, no necesita de un soporte físico para la transmisión de información, se clasifica en diferentes tipos de comunicación de acuerdo al tipo de dispositivos y al rango de datos que utiliza.

Características:

- La señal se transmite a través del aire
- Menos seguros
- Utilizados para distancias mayores

Los tipos de tecnología son las siguientes:

✓ **Wifi:** Wifi, conocida también por el estándar 802.11 a/b/g, es una tecnología de comunicación inalámbrica de bajo costo y de baja potencia, es la más usada actualmente en el mundo por varios dispositivos electrónicos como teléfonos inteligentes, computadoras portátiles, etc. Para la transmisión de datos trabajando a frecuencias de 2.4GHz [26]. Las ventajas principales que nos ofrece la tecnología wifia es la facilidad de integración, lo que nos permite que los usuarios puedan acceder a recursos de la red en casi cualquier lugar. Además de tener una capacidad de ampliación. Sin embargo, posee unas desventajas en la seguridad, la velocidad de

transmisión en redes inalámbricas comúnmente es más lentas que en las redes cableadas [27].

✓ **Bluetooth:** Bluetooth conocida también por el estándar IEE 802.15.1, basado en un sistema de radio inalámbrico de bajo costo, permite la comunicación de corto alcance para dispositivos inalámbricos. Opera a frecuencias de 2.4 GHz, con una velocidad de transferencia de datos de 1 a 2 Mbps. De esta tecnología se derivan tres grupos con diferentes rangos de alcance, el primer grupo está en el rango de 100 metros, el segundo en el rango y el más utilizado está en el rango de los10 metros y el tercero en el rango de 1 metro [28].

✓ **ZigBee:** ZigBee es también conocida como el estándar IEE 802.15.4, es una tecnología de comunicación inalámbrica de bajo costo, bajo consumo de energía y de baja velocidad de transmisión de datos. ZigBee permite la comunicación entre varios sensores que estén diseñados para ser usados con poca energía, trabaja en las bandas sin licencia 2.4GHz, 900MHz y 866MHz, con una velocidad de transmisión de 250Kbps. Además, soporta las topologías tipo estrella, malla y árbol [28].

1.3.10 Protocolos de comunicación

Los protocolos de comunicación son reglas de comunicación que permiten que diferentes hardware y software se comuniquen a través de una sola red mediante señales eléctrica, además de proporcionar la comunicación entre dispositivos que se encuentran al extremo del canal de comunicaciones, determinan un formato para la transmisión de datos.

- **MQTT**

El protocolo de Transporte de Telemetría MQ, fue creado a finales del siglo 90 por el Dr. Stanford y Alen Nipper, con el objetivo de comunicar dispositivos con muy pocos recursos, con un ancho de banda reducido, alta latencia o pocos confiables, es un protocolo de mensajería de publicación/suscripción, simple, liviano y flexible. En cuanto a su operación está compuesta por dos nodos: publicación/ suscripción, y el nodo central de gestión o Broker. Este protocolo es adecuado para aplicaciones de Internet de las cosas donde se envían pequeñas cantidades de información por lo tanto no se necesita gran cantidad de ancho de banda, usado principalmente por empresas e industrias para comunicaciones entre máquinas o sensores [29].

- **HTTP**

El protocolo de transferencia de HiperTexto HTTP, es un protocolo simple de cliente-servidor, que permite el intercambio de información entre los clientes web y los servidores HTT. Este protocolo fue propuesto en 1999 por Tim Berners-Lee para resolver las necesidades de un sistema global de distribución de información.

HTTP consiste en operaciones de solicitud/respuesta. El cliente permite una conexión con el servidor y envía un mensaje con los datos de la solicitud. El servidor responde con un mensaje similar, que contiene el estado de la operación y su probable resultado. [30].

Características del protocolo HTTP son:

✓ La comunicación entre el cliente y servidor se realiza por código US-ASCII de 7 bits.

✓ Permite la transferencia de objetos multimedia, codificando los archivos binarios en cadena de caracteres.

✓ Conexión permanente, no se cierra la conexión tras el envió de la información.

✓ Varias solicitudes simultaneas, un cliente puede realizar varias peticiones utilizando una única conexión, sin esperar a la respuesta del servidor para cada una de ellas [31].

1.3.11 Dispositivos electrónicos

- **Dispositivos controladores**

Los sistemas automáticos y los robots pueden iniciar, ejecutar procesos y detenerlos, para ello necesitan recibir información del exterior, procesarla y generar una respuesta, para eso se necesitan de dispositivos controladores. Los dispositivos controladores son sistemas que permiten el manejo de señales de entrada como de salida, es un software o programa que sirve de intermediario para la comunicación entre un dispositivo de hardware y el sistema operativo. Las principales funciones de los dispositivos controladores son: enviar comandos a los dispositivos, detectar interrupciones, controlar los errores, proporcionar una interfaz entre los dispositivos y el resto del sistema.

✓ **ESP32:** ESP32, fue desarrollado por Espressif Systems con varias características y capacidades combinadas (Sistema de doble chip) de Wi-Fi y Bluetooth. Este módulo consta de un microprocesador Tensilica Xtesa LX6 de doble núcleo, puede trabajar con una frecuencia de reloj de hasta 240MHz cuando se utiliza uno de ellos. ESP32 está integrado con switch de antena, amplificador de potencia, amplificador de recepción de bajo nivel de ruido, módulos de administración de energía y filtros, todos estos integrados dentro del mismo chip. Este módulo está diseñado para aplicaciones de electrónica, ya que logra un consumo de energía ultra bajo a través de condiciones de ahorro de energía. En la figura 1.10 se muestra a la tarjeta ESP32 combinada con capacidad inalámbrica wifi y Bluetooth [34].

Características:

- **Procesador principal:** El ESP32, presenta uno o dos microprocesadores Tensilica Xtensa de 32 bits LX6.
- ESP32, soporta tecnologías como:
- **Wifi:** Soporta las tecnologías estándar 802.11b/g/n que trabajan en frecuencias de 2.4GHz hasta 150Mbit/s
- **Bluetooth:** Soporta la tecnología v4.BR/EDR y además dispone de BLE (Bluetooth Low Energy).
- **Frecuencia de reloj:** Su frecuencia es programable, hasta 240MHz.
- **Rendimiento**
- En cuanto a la memoria, el dispositivo dispone de:
- **ROM:** 448KB, para arranque y funciones básicas.
- **SRAM:** 520KiB, para datos e instrucciones.

Figura 1.10: Tarjeta ESP32 combinada con capacidad inalámbrica Wifi y Bluetooth *[34]*.

✓ **PIC:** Los microcontroladores PIC, son circuitos integrados programables que integran en un solo chip las unidades de memorias para el almacenamiento de datos, lógica para el cálculo de operaciones, aritmética y los periféricos de entrada y salida para la comunicación con otros dispositivos. Existen en gamas de 8bit, 16bit y 32bit, se definen de una familia de bajo costo, bajo consumo de potencia y alta velocidad de operación. Su set de instrucciones es muy fácil de manejar. Entre estos controladores se tiene:

Los PIC16CX, son memorias EPROM o PROM internas, además posee un circuito de inspección interno para evitar que el programa se pierda, los pines trabajan con corrientes hasta de 25mA, pueden operar hasta una velocidad de 20Mhz, y el consumo de potencia es de 50micro vatios a una velocidad de 32Khz, posee temporizadores programables y algunos conversores análogos a digital. En la figura 1.11 se puede observar al PIC 16F84A, es un microcontrolador de la familia Microchip, su característica principal es que posee memoria flash en lugar de memoria EEPROM, pero su manejo es igual [35].

Figura 1.11: Microcontrolador PIC [35].

✓ **Arduino:** Arduino, es una placa para la realización de prototipos de electrónica, es de código abierto, de bajo costo, basada en software gratis, libre y multiplataforma que se instala en la computadora y permite escribir, verificar y guardar en la memoria del microcontrolador de la placa Arduino el conjunto de instrucciones que se desea que este empiece a ejecutar, además de ser flexible y fácil de usar, debido a que se encuentra empaquetado con el entorno de desarrollo integrado (IDE), posee un microcontrolador reprogramable que es programable a través de USB. Además, tiene serie de pines hembras, que permiten conectar allí de forma sencilla y cómoda diferentes sensores, actuadores y módulos. Este microcontrolador es un solo chip que se lo conoce como AVR, que funciona a solo 16MHz con un núcleo de 8 bits y tiene una cantidad limitada de memoria disponible con 32Kb de almacenamiento. El lenguaje de programación del software es lenguaje C o C++, esta placa es una implementación cableada, un ejemplo de las placas Arduino es el Arduino uno el cual se muestra en la figura 1.12 [36].

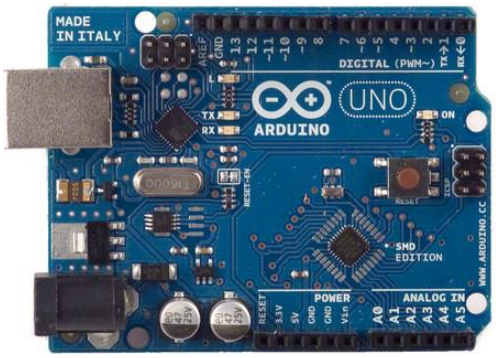


Figura 1.12: Microcontrolador Arduino Uno [36].

✓ **Computador de placa reducida:** La placa Raspberry, es una pequeña tarjeta que entrega las mismas características de una PC, fue elaborada en la Universidad de Cambridge en Reino Unido. Esta se conecta a una televisión o pantalla LCD con entrada HDMI y a un teclado. Por su tamaño puede ser usado para muchas cosas como una computadora de escritorio, además de reproducir videos de alta definición. Tiene un procesador grafico Video Core IV y 512 MiB de memoria RAM, el diseño no incluye disco duro ni unidad de estado sólido, usa una tarjeta SD pero es necesario que esta sea de al menos de 2GB de capacidad para almacenar todos los archivos requeridos [37].

Figura 1.13: Placa Raspberry [37].

- **Sensores de corriente**

El funcionamiento de los sensores de corriente es un dispositivo que permite la detección del campo magnético producido cuando una corriente alterna o continua que circula a través de un medio a medir, genera una señal proporcional que puede ser en voltaje, corriente análoga o una señal digital. Existen dos tipos de sensores de corriente, en lazo abierto que son menos costosos y son los preferidos en circuitos alimentados por baterías y por otro lado los sensores de corriente en lazo cerrado son sensores de gran precisión [33].

- **Puentes h**

El puente H adopta la letra H, por ello su nombre debido a la forma que presenta dentro de un circuito esquemático simplificado como se ve en la figura 1.14, es un circuito electrónico que permite el control de un motor eléctrico DC para un movimiento en ambos sentidos, horario y anti horario. Estos están disponibles en integrados y también se los puede construir a partir de componentes eléctricos y/o electrónicos. Principalmente son usados en robótica o en convertidores de potencia.

El funcionamiento del puente H para un cambio de giro se compone de 4 interruptores (S1, S2, S3, S4), cuando dos de ellos en este caso los interruptores S1 y S4 están cerrados y S2 y S3 están abiertos, se aplica un voltaje haciendo girar el motor en un

sentido. Abriendo los interruptores S1 y S4 y cerrado S2 y S3 el voltaje se invierte, permitiendo así un movimiento en sentido inverso del motor [32].

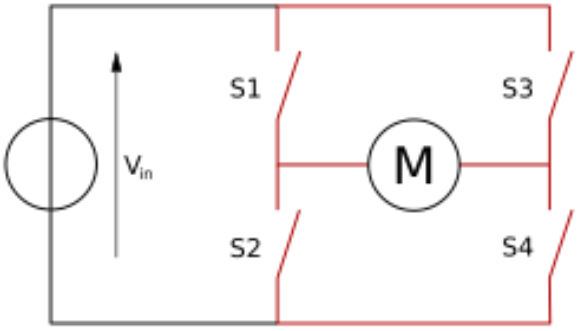


Figura 1.14: Estructura de un Puente H [32].

1.4 Objetivos

➢ Objetivo general

El objetivo principal del presente proyecto se enfocó en implementar un controlador electrónico para la operación del brazo robótico Scorbot ER_4U, mediante la selección de software y hardware libre, y del diseño de la estructura para uso en ambientes robustos, para finalmente realizar la validación de resultados del sistema de control implementado.

Para lograr dicho objetivo se realizó las actividades detalladas en cada objetivo específico

➢ Objetivos específicos

Analizar las características físicas y electrónicas del funcionamiento del brazo robótico Scorbot ER_4U.

✓ Análisis del funcionamiento del Scorbot ER_4U

✓ Identificación de los parámetros eléctricos y mecánicos que componen al Scorbot ER_4U

✓ Determinación de los componentes y etapas de control requeridas para la operación del brazo robótico Scorbot ER_4U

Determinar las diferentes tecnologías de dispositivos de procesamiento y transmisión de datos para sistemas microcontrolados.

- ✓ Análisis de métodos y protocolos de comunicación para el intercambio de información entre el brazo robótico Scorbot ER_4U y un computador.
- ✓ Selección de la tecnología de comunicación y procesamiento de datos a utilizarse en el módulo controlador.

Diseñar el controlador de software libre para la manipulación mecánica del brazo Scorbot ER_4U.

- ✓ Diseño del circuito control de interfaces del Scorbot ER_4U; integración de etapas de sensores, mecanismos de acción y modos de comunicación.
- ✓ Diseño del sistema de alimentación eléctrica para la controladora.
- ✓ Programación del software de la tarjeta electrónica del módulo controlador del Scorbot ER_4U.
- ✓ Elaboración de la tarjeta electrónica del módulo controlador.
- ✓ Ensamblado de componentes del módulo controlador en un contenedor de protección.
- ✓ Evaluación y pruebas de funcionamiento del prototipo.

CAPÍTULO II

METODOLOGÍA

2.1 Materiales

Para el diseño e implementación del presente proyecto de investigación, se requiere de medidas eléctricas realizadas en el Scorbot ER_4U, así como fundamentación bibliográfica disponible en libros, documentos científicos, tesis, repositorios tecnológicos, documentos web, revistas tecnológicas y documentación de fabricantes.

2.2 Métodos

Modalidad de la Investigación

En el presente trabajo de investigación se realizó de acuerdo a los conceptos de investigación aplicada ya que tiene por objeto la generación de conocimiento con aplicación directa a los problemas de la sociedad o el sector productivo. Los conocimientos adquiridos serán utilizados para dar solución a los generados para el brazo Robótico Scorbot ER_4U, utilizando tecnología de hardware y software libre, mediante los siguientes tipos de investigación:

Se aplicó la investigación bibliográfica documentada, para la adquisición de información aplicada sobre bases teóricas que facilite el diseño del módulo controlador de software libre para el Scorbot ER_4U. La explicación científica de las bases del proyecto se tomó de libros, artículos técnicos y proyectos desarrollados ya en otros países y en el Ecuador donde se realizaron estudios de: circuitos electrónicos, circuitos eléctricos, sistemas micro controlados, aplicaciones robóticas, programación, desarrollo de aplicaciones IOT y desarrollo de módulos de control.

Además, se utilizó la investigación de campo con la que se realizó un estudio sistemático para determinar las características del módulo y circuitos a diseñarse. La recolección de información, adquisición de datos y validación de funcionamiento se realizó de forma directa en el laboratorio de Robótica de la Facultad de Ingeniería en Sistemas, Electrónica e Industrial.

Se utilizó la investigación experimental para realizar el diseño de los circuitos requeridos para el controlador e implementación de los elementos de control del prototipo, se realizó pruebas de funcionamiento para validar que las características del controlador a implementarse sean adecuadas.

Recolección de la Información

La información para el estudio y dimensionamiento de equipos y materiales se obtendrá mediante medidas eléctricas realizadas en el Scorbot ER_4U, así como de la fundamentación bibliográfica disponible. La información fue levantada con fichas de observación; se utilizó bibliotecas afines a la documentación teórica requerida, de la Universidad Técnica de Ambato y documentación virtual.
La recolección de información se inició de forma previa a la presentación y reconocimiento del proyecto de investigación utilizando como recursos: tablas comparativas y fichas de observación.

Procesamiento y Análisis de Datos

Para el procesamiento y análisis de los datos se procedió con las siguientes actividades:

El procesamiento y análisis de datos se realizó mediante una clasificación de la documentación obtenida, presentando una descripción ordenada de los entornos a estudiarse en el proyecto. Se realizó un análisis crítico de los datos obtenidos durante la recolección de información, considerando los siguientes lineamientos:

- ✓ Obtener parámetros técnicos, específicos y concretos que determinen las características del sistema a ser diseñado.
- ✓ Interpretar la información que permite plantear estrategias de solución al problema.

Desarrollo del Proyecto

Para desarrollar la placa controladora electrónica fue necesario cumplir con las siguientes actividades:

- ✓ Análisis del funcionamiento del Scorbot ER_4U

✓ Identificación de los parámetros eléctricos y mecánicos que componen al Scorbot ER_4U

✓ Determinación de los componentes y etapas de control requeridas para la operación del brazo robótico Scorbot ER_4U

✓ Análisis de métodos y protocolos de comunicación para el intercambio de información entre el brazo robótico Scorbot ER_4U y un computador

✓ Diseño de un circuito de control para el actuador del Scorbot ER_4U

✓ Selección de la tecnología de comunicación y procesamiento de datos a utilizarse en el módulo controlador.

✓ Diseño del circuito control de interfaces del Scorbot ER_4U; integración de etapas de sensores, mecanismos de acción y modos de comunicación.

✓ Diseño del sistema de alimentación eléctrica para la controladora.

✓ Programación del software de la tarjeta electrónica del módulo controlador del Scorbot ER_4U.

✓ Elaboración de la tarjeta electrónica del módulo controlador

✓ Ensamblado de componentes del módulo controlador en un contenedor de protección.

✓ Evaluación y pruebas de funcionamiento del prototipo

✓ Corrección de errores

✓ Elaboración del informe final.

CAPÍTULO III

RESULTADOS Y DISCUSIÓN

3.1 INTRODUCCIÓN

La investigación en el área de la automatización y la robótica ha permitido el desarrollo de múltiples empresas dedicadas a la manufactura de equipos eléctricos y electrónicos. La ejecución de procesos de forma automática o manipuladas por computador requiere de la integración de varios dominios de conocimiento sobre física, mecánica, ciencias computacionales y electrónica. El sector industrial desarrolla equipos de robótica de campo abierto y para múltiples aplicaciones, sin embargo, se verifica que el software para la manipulación de los mismos está sujetos al uso de licencias. La preferencia de software libre en la comunidad universitaria y de investigación permite el desarrollo de trabajos colaborativos, impulsando el desarrollo de aplicaciones como: manejo de servidores, desarrollo de páginas web, visión artificial, elaboración de tarjetas embebidas y algunas áreas de la robótica. El desarrollo de aplicaciones de automatización y robótica con plataformas que utilizan software de propietario, con licencia pagada, limita la investigación y modificación libre de los componentes de los robots elaborados.

El brazo robótico Scorbot ER-4U está diseñado para manipularse de forma única mediante el programa de propietario, sea Scorbase o Robocell. Esta situación reduce las posibilidades en la expansión de aplicaciones y las características de las mismas; por ejemplo, para integrar programas con protocolos de comunicación RS232, TCP/IP, RS485 entre otros de la capa de red y aplicación del modelo OSI con el brazo robótico, se requiere el uso de Scorbase como interfaz entre el host y el controlador del brazo. El uso obligatorio del software de propietario implica dependencia de los tiempos de respuesta en las órdenes de ejecución de movimiento desde programas de terceros. También es obligatorio el uso de un ordenador y el brazo está obligado a operar de forma conjunta a un controlador, todo mediante una comunicación cableada, afectando la libertad de movimiento o el desarrollo de plataformas para el brazo.

La implementación de un controlador de hardware y software libre, explicada en el presente proyecto, elimina las barreras de diseño y genera la apertura a la elaboración de nuevas aplicaciones, que integren manejos de protocolos de red de forma directa, utilizando de forma única una tarjeta electrónica. El proyecto implementa un controlador que permite manipular el brazo de forma independiente mediante una memoria propia, o a través de un host externo desde donde se envían las rutinas a ejecutarse.

3.2 DESARROLLO DE LA PROPUESTA

3.2.1 Brazo Robótico Scorbot ER_4U

El brazo robótico Scorbot ER_4U es un robot diseñado para emular procesos de automatización en ambientes industriales. La estructura mecánica del brazo es de arquitectura abierta, para que los investigadores y estudiantes tengan la factibilidad de observar, analizar y aprender acerca de la estructura del robot.

Figura 3.1: Diagrama de conexión del Scorbot ER_4U.

Elaborado por: El Investigador

La manipulación del brazo robótico se realiza desde Scorbase o Robocell, instalado en un computador personal con un sistema operativo Windows, utilizando como interfaz un módulo de control; según lo indicado en la Figura 3.1. El módulo de control tiene interfaces de conexión para entradas y salidas digitales y analógicas, puerto DB9, interfaz para el encendido de dos motores DC externos a 24V DC un puerto serial para futuras implementaciones, puerto USB y un puerto D50 para manipular las señales eléctricas se los sensores y actuadores instalados en el brazo.

El software de control, sea Robocell o Scorbase, gestionan las señales de entrada y salida del controlador mediante comunicación USB, utilizando drivers de propietario. La información recibida por el módulo controlador es interpretada para encender, apagar salidas digitales o los motores y para leer los registros de las entradas analógicas, digitales y los sensores del brazo.

Figura 3.2 : Ejes de rotación del Scorbot ER_4U [27].

El Scorbot ER_4U es un robot articulado vertical que tiene 5 ejes de rotación y un gripper como actuador final, acuerdo a lo ilustrado en la Figura.3.2. En la imagen y en el presente proyecto se utiliza la notación en inglés de cada uno de los ejes, con la finalidad de mantener un estándar en la nomenclatura. En la Tabla 3.1 se indica las funciones específicas y los grados de libertad de movimiento que tienen cada uno de los ejes del brazo.

La base permite generar la rotación del cuerpo del robot en un total de 310°, el eje de rotación shoulder permite elevar el ante brazo en 130° y descender el mismo en 35° sobre la horizontal. Elbow tiene una libertad de rotación de 130° en sentido horario y anti horario medido desde la normal del eje shoulder. La rotación del eje picth es de ±130° en base a la línea perpendicular al eje elbow y paralela a shoulder. Finalmente, el eje roll tiene un diseño mecánico que permite una rotación infinita, sin embargo, debido a las conexiones eléctricas requeridas, la rotación se limita a 570° en sentido horario o anti horario en sentido paralelo a la perpendicular del brazo picth. El gripper es un actuador final manipulado con un servomotor que permite sujetar elementos externos mediante una brizna de dos lingotes de sujeción. Cuando el griper se encuentra con las almohadillas de caucho para protección tiene una apertura total entre lingotes de 65mm y en caso de ausencia de las mismas la apertura es de 75mm.

Tabla 3.1: Grados de libertad y funciones de los ejes de Scorbot ER_4U

Eje	Grados de Movimiento	Función
Base	310°	Gira el cuerpo
Shoulder	+130° / -35°	Eleva y desciende el brazo
Elbow	±130°	Eleva y desciende el brazo
Picth	±130°	Eleva y desciende la herramienta final
Roll	Ilimitada mecánica; ±570° eléctrica	Gira la herramienta final
Gripper	75mm sin rubber path 65mm con rubber path	Sujeta el elemento de trabajo

Elaborado por: El Investigador, en base a [16].

- **Hardware del Scorbot ER_4U**

Los componentes electrónicos integrados en el brazo del Scorbot ER_4U constituyen elementos de control que permiten al sistema programado manipular los movimientos de los ejes del brazo robótico. Estos elementos se clasifican en actuadores y sensores, teniendo así: motores, encoders y finales de carrera. El movimiento de los 5 ejes del brazo y el Gripper son operados por servo motores de corriente continua. La dirección del movimiento de los motores se determina por la polaridad del voltaje de operación aplicado, con una polarización positiva el motor gira en una dirección y en polarización negativa gira a la dirección opuesta.

Cada uno de los motores está asociado a un encoder eléctrico-óptico para tener un control de circuito cerrado; la ubicación y movimiento de cada uno de los ejes del brazo son medidos e identificados mediante el uso de éstos encoders. Cuando un eje del robot gira, el encoder asociado al motor de ése eje genera una serie de pulsos alternando señales eléctricas de "1" lógico y "0" lógico. La cantidad de pulsos generados es proporcional a la cantidad de movimiento efectuado, teniendo así la factibilidad de medir e identificar la posición de los ejes.

La operación del brazo robótico está sujeta a determinar una posición inicial antes de ejecutar las rutinas. La posición home es determinada mediante las señales eléctricas enviadas por microswitches asociados a cada eje del robot. Durante la ejecución de homming desde Scorbase, cada uno de los ejes se desplaza hasta la activación del microswitch o final de carrera. Cuando todos los finales de carrera se han activado, el robot se encuentra en posición de home y permite la ejecución de las rutinas programadas en Scorebase o Robocell.

a) Interfaz de Conexiones Eléctrica

El brazo robótico está conectado al controlador mediante un cable con un terminal D50, un conector de 50 pines al que se encuentran enlazados todos los cables de energía y señal de los encoders, motores y finales de carrera.

Tabla 3.2: Descripción de las funciones de los pines del conector D50 del Scorbot ER_4U.

#Pin	Función	#Pin	Función	#Pin	Función
1	Pulso B del encoder del eje	18	Pulso B del encoder del eje	34	Pulso B del encoder del eje
2	Pulso B del encoder del eje	19	Pulso A del encoder del gripper	35	Pulso B del encoder del eje
3	Pulso A del encoder del eje 5	20	Pulso A del encoder del eje 4	36	Pulso B del encoder del eje
4	Pulso A del encoder del eje 3	21	Pulso A del encoder del eje 2	37	5Vdc
5	Pulso A del encoder del eje 1	22	Final de carrera del eje	38	5Vdc
6	Final de carrera del Eje	23	Final de carrera del eje	38	5Vdc
7	Final de carrera del Eje	24	Final de carrera del eje	40	5Vdc
8	Final de carrera del Eje	25	Voltaje del encoder del eje	41	5Vdc
9	Voltaje del encoder del eje	26	Voltaje del encoder del eje	42	5Vdc
10	Voltaje del encoder del eje	27	Voltaje del encoder del eje	43	5Vdc
11	Voltaje del encoder del eje	28	GND	44	5Vdc
12	Motor del gripper -	29	GND	45	Motor del gripper +
13	Motor del eje 5 -	30	GND	46	Motor del eje 5 +
14	Motor del eje 4 -	31	GND	47	Motor del eje 4 +
15	Motor del eje 3 -	32	GND	48	Motor del eje 3 +
16	Motor del eje 2 -	33	GND	49	Motor del eje 2 +
17	Motor del eje 1 -			50	Motor del eje 1 +

Elaborado por: El Investigador.

Los cables del motor del gripper y los finales de carrera en el brazo se enlazan al D50 por medio de un conector molex de 12 pines ubicado en la base del robot. Los cables utilizados son particularmente flexibles y resistentes a la rotura, incluso después de un movimiento extenso del brazo robot. En la Tabla 3.2, se especifica las conexiones existentes entre los distintos componentes del brazo el conector D50.

b) Características Eléctricas del Brazo

Las características técnicas de los componentes electrónicos del brazo robótico se especifican en tres partes: motores, encoders y micro interruptores, de acuerdo a lo detallado a continuación:

Motores

Los servos motores utilizados en el brazo robótico tienen un voltaje de operación de 12V DC de acuerdo a las indicaciones de los manuales entregados por el fabricante. La corriente requerida para el funcionamiento correcto del brazo se determina mediante mediciones realizadas en durante la operación del mismo.

En la Tabla 3.3 se resumen las corrientes promedio medidas en los motores del Scorbot ER_4U. El consumo de corriente en condiciones normales es de 500mA, cuando se le aplica una carga de 1Kg, que es la carga máxima que soporta el brazo la corriente promedio sube hasta los 700mA y en el caso de que el eje al que se encuentra asociado el motor sufre un impacto, la corriente se incrementa sobre los 900mA.

Tabla 3.3: Mediciones promedio del consumo de corriente en los motores del Scorbot ER_4U.

Condición	Voltaje	Corriente Promedio
Desplazamiento normal	12V DC	500mA
Desplazamiento con carga	12V DC	700mA
Impacto	12V DC	>900mA

Elaborado por: El Investigador

Microinterruptor

Las características eléctricas de cada uno de los micros interruptores son determinadas mediante mediciones de voltaje y resistencia; se determina que los mismos son polarizados con un voltaje de 5V DC, y se encuentran en un estado normalmente abierto, entregando una señal de 0Vcd al controlador. Cuando el final de carrera es activado pasa al estado de cerrado, entregado a la salida una señal de 5V DC y una resistencia de 0Ω.

Encoders

Los encoders utilizados en cada uno de los ejes, son encoders de cuadratura; éstos entregan dos señales de pulsos rectangulares desfasados. La frecuencia de la señal es proporcional a la velocidad del movimiento y la con la cantidad de pulsos medidos se determina la variación de desplazamiento angular que efectúan los ejes.

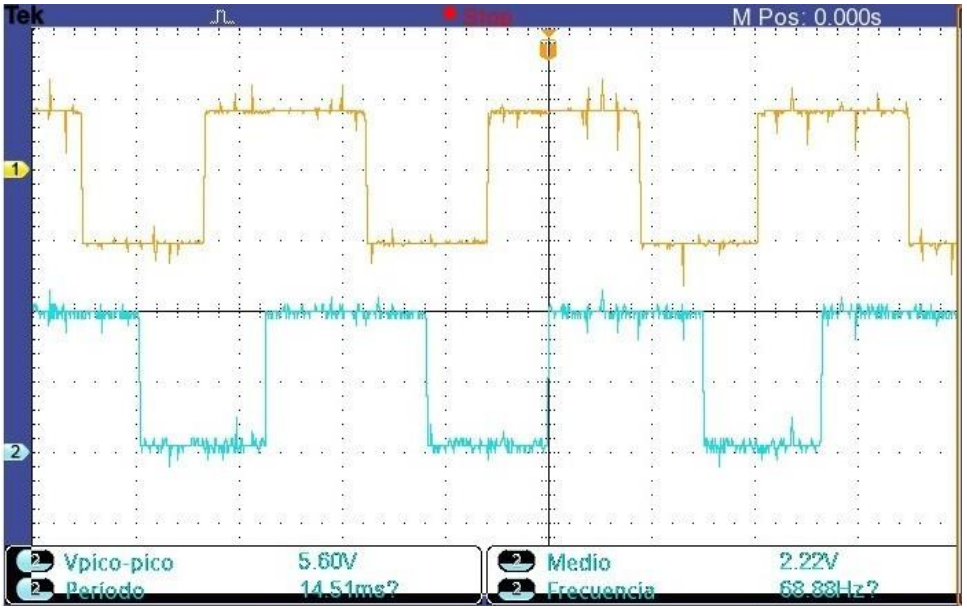
Todos los encoders del Scorbot ER_4U tienen las mismas características eléctricas, los voltajes y funciones de los pines de éstos, se detallan en la Tabla 3.4. El controlador entrega un voltaje de alimentación de 5V DC para los leds internos de los encoders y recibe dos señales de pulsos rectangulares.

Tabla 3.4: Voltajes y funciones de los pines de los encoders del Scorbot ER_4U.

Pin	Voltaje V DC	Función
Vled	5	Voltaje de alimentación
P0	5 - 0	Señal de pulsos
P1	5 - 0	Señal de pulsos desfasado
GND	0	Tierra

Elaborado por: El Investigador

En la Figura 3.3, se observa los pulsos rectangulares generados por los encoders medidas en un osciloscopio, durante el desplazamiento de los ejes del robot. De color amarillo se observa la señal del pin P0 y en color azúl la señal eléctrica el pin P1. En la imagen se aprecia algunas particularidades de las ondas de salida de los encoders, entre ellas: las señales se encuentran desfasadas en un 50% del valor del periodo, el voltaje pico es de 5.6 V DC y el voltaje medio de 2.22 V DC.



Elaborado por: El investigador

Figura 3.3: Señal de pulso de los encoders del Scorbot ER_4U.

- **Software de control del Scorbot ER_4U Scorbase**

Scorbase es un paquete de software de control de robótica utilizado para la programación y operación de robots, entre ellos el Scorbot ER_4U. Este aplicativo desarrollado por la empresa Intelitek, permite al usuario manipular las funciones del brazo robótico y de la interfaz de la controladora USB, bajo las siguientes características del software:

✓ Comunicación con el controlador del robot a través del canal USB.

✓ Control de ocho salidas digitales, cuatro entradas analógicas y dos salidas analógicas conectadas al controlador-USB.

✓ Definición y visualización de la posición, así como movimiento manual del robot en referencia al Sistema de coordenadas conjuntas.

✓ Definición del movimiento de robot como Ir a posición, Ir lineal o Ir circular, con ajustes de velocidad activa en porcentajes

✓ Ajuste predeterminado de 1000 posiciones y 10000 líneas de programas activos.

✓ Programación variable, en tres niveles de complejidad, para moderar la curva de aprendizaje.

✓ Guardar y cargar proyectos. proporciona simulación del robot y otros dispositivos en la celda de trabajo.

El mejor rendimiento de la herramienta Scorebase, se obtiene al instalar el programa en un equipo con las siguientes características: Procesador Intel Core i5 2400 GHz o superior, al menos 1 GB de RAM para Windows XP y 4 GB para Windows 7 y superior, un disco duro con al menos 200 MB de libre espacio, un mouse, puerto USB, sistema operativo Windows XP, 7, 8, 8.1 o 10; no existe una versión del software disponible para sistemas Linux.

a) Inicialización y Control.

La inicialización y control es el proceso utilizado por Scorebase para reconocer el brazo robótico y permitir la manipulación y movimiento de sus ejes. La ubicación del robot y los ejes periféricos se monitorea y controla mediante codificadores. Para inicializar los codificadores y obtener un rendimiento consistente, los ejes primero deben alcanzar una posición predefinida conocida como *"hard-home"* o posición inicial; todas las posiciones y movimientos registrados se realizan en referencia a esta posición de *home*.

En el procedimiento de búsqueda de referencia, se busca la posición de *home* para los 6 ejes del brazo robótico; en la ejecución de la rutina *"home"*, cada eje se calibra por separado, seleccionando los motores de cada eje de forma progresiva. El controlador activa el motor de eje seleccionado, hasta que se presiona el micro interruptor correspondiente. Luego, el controlador inicializa el contador del codificador del eje y procede con la ubicación del siguiente eje. Una vez que todos los ejes configurados están posicionados, se finaliza el procedimiento de had-home.

El procedimiento de referencia, se realiza desde el menú Run seleccionando la opción buscar home en todos los ejes. Se abre una ventana indicada en la Figura 3.4 que muestra el número del eje que se está referenciando actualmente. Cada vez que un eje se posiciona correctamente, se muestra en marca de verificación junto al número del eje.

Figura 3.4: Interfaz gráfica de Scorebase que indica el proceso de referencia. [Programa Scorebase].

Después de que los cinco ejes y la pinza se hayan ubicado en la posición de referencia, se muestra una marca de verificación junto al Robot. Si el procedimiento de referenciada falla, se muestra un mensaje. Cuando Scorebase esté en modo fuera de línea (el brazo no está conectado), o cuando RoboCell está instalado y en modo de simulación, no se requiere el procedimiento de referencia. Este comando envía el robot y los periféricos a una posición donde el valor de los codificadores de los ejes es igual a cero.

Scorebase tiene dos modos de operación: en línea y fuera de línea; éstas son formas de ejecución del programa que permiten utilizar las características del software bajo diferentes parámetros; con el controlador del brazo robótico conectado y desconectado respectivamente. En el modo en línea, Scorebase se comunica con el controlador a través del canal USB. Si se selecciona el estado Control activado, Scorebase controla el robot, los periféricos y el dispositivo de entradas y salidas. En el modo fuera de línea, solo se puede usar el estado de control apagado generalmente utilizado para la programación y depuración.

b) Grabación de Posiciones.

La grabación de una posición del robot (en coordenadas conjuntas) se realiza manipulando el movimiento del robot a la posición requerida y luego grabándola en una memoria interna del programa.

El cuadro de diálogo Movimiento manual permite el control directo y la manipulación del robot y los ejes periféricos, éste cuadro se abre automáticamente cuando se crea un proyecto. Para mostrar el cuadro de diálogo Movimiento manual cuando no hay proyectos abiertos, se selecciona el menú Ver y luego la opción movimiento manual, en donde se muestra el cuadro de diálogo de la figura 3.5.

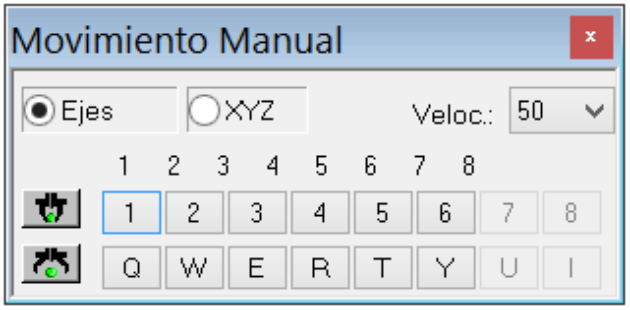


Figura 3. 5: Cuadro de diálogo del movimiento de los ejes en Scorebase.

En la Tabla 3. 5 se explica la forma en que los botones del cuadro de diálogo de movimiento manual (o presionar las teclas correspondientes en el teclado) controla el robot y los movimientos de los periféricos. Cuando se selecciona Juntas, hacer clic en los botones (o presionar las teclas correspondientes en el teclado) mueve un eje del robot a la vez, la distribución de las funciones se describe a continuación:

Tabla 3. 5: Funciones de los botones del panel de control de Scorbase.

Botones	Movimiento del Eje
1/Q	Gira el eje de base a la derecha o izquierda
2/W	Mueve el eje shoulder hacia arriba y abajo
3/E	Mueve el eje elbow hacia arriba y abajo
4/R	Mueve el eje pitch hacia arriba y abajo
5/T	Gira el eje de pitch en sentido horario o antihorario
6/Y	Abre y cierra el eje de gripper
7/U	Mueve los periféricos externos
8/I	Mueve los periféricos externos

3.2.2 Requerimientos Técnicos del Sistema

Al determinar las características eléctricas de los componentes del brazo robótico, es necesario especificar los requerimientos técnicos para el diseño del controlador electrónico de software libre para el brazo robótico Scorbot ER_4U; las mismas se detallan a continuación:

✓ Módulo de control para determinar la variación y dirección de desplazamiento angular generado en cada uno de los ejes. El controlador requiere la capacidad de lectura de 12 pines, 2 por cada eje. Se debe almacenar y transmitir la información de los desplazamientos ejecutados.

✓ Módulo como etapa de acondicionamiento para media potencia, que permita ejecutar funciones de encendido, apagado, control de velocidad y dirección de giro de cada uno de los motores del brazo. La etapa de acondicionamiento de motores recibe señales eléctricas de nivel lógico TTL y las transforma a señales eléctricas de 12V DC con una fuente de corriente superior a 600mA, se requiere el manejo de 12 señales, 2 pines por cada motor.

✓ Interfaz de alimentación y acondicionamiento eléctrico para las señales de los micros interruptores. Se requiere adaptar los valores de voltaje utilizados por el micro interruptor con los del microcontrolador a seleccionar; de forma similar para las señales de los encoders.

✓ Implementar un módulo de sensores que permita monitorear el consumo de corriente de los motores, con la finalidad de determinar impactos o colisiones de los ejes y diseñar un software de protección. El software ejecuta órdenes de apagado cuando se identifica un consumo de corriente superior a los 800mA, además debe generar alertas para identificar dichos eventos.

✓ Se requiere controlar la velocidad de desplazamiento de los ejes del Scorbot.

✓ Implementar un sistema de comunicación que utilice software libre y protocoles de comunicación de la capa de red del modelo OSI, para la manipulación del brazo robótico desde múltiples interfaces de programación.

✓ Implementar un software demo de la manipulación del Scorbot mediante un lenguaje de programación diferente a Scorbase o Robocell.

3.2.3 Diseño General del Sistema

El controlador del brazo robótico está diseñado de acuerdo a la arquitectura del sistema indicado en el diagrama de bloques de la Figura 3.6. El sistema está elaborado para que el brazo robótico sea manipulado desde un dispositivo electrónico externo al que se le ha denominado host, por ende, se requiere de un controlador que hace de interfaz entre los nodos (host y Scorbot), permitiendo intercambiar las señales eléctricas que generan los movimientos en el brazo e informan al host el estado del mismo.

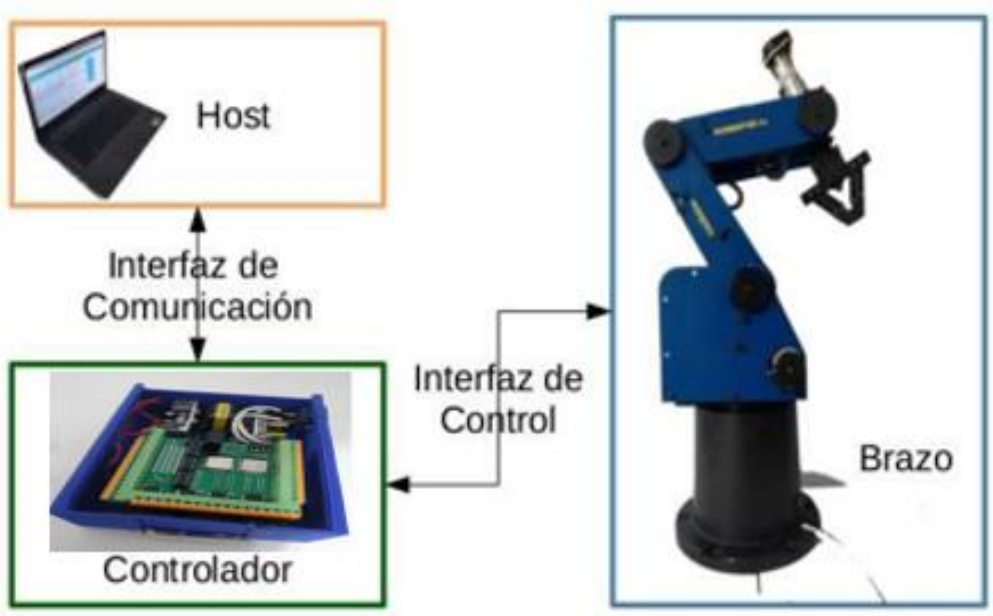


Figura 3.6: Diagrama de bloques del sistema general.

Elaborado por: El Investigador.

El host constituye un dispositivo electrónico de procesamiento de datos, el mismo puede ser un ordenador, computador de placa reducida, celular, Tablet, tarjetas de desarrollo electrónico o microcontroladores. El sistema se diseña en una arquitectura abierta, orientando a la capacidad de manipular el brazo desde equipos de diferentes sistemas operativos basados en Windows, Linux o firmwares de software libre.

- **Interfaz de Comunicación**

La interfaz de comunicación constituye el grupo de protocolos y medios de transmisión utilizados para la transferencia bidireccional de datos entre el ordenador o host y la tarjeta controladora. En la Tabla 3.6 se indica las características técnicas de diferentes protocolos de comunicación que se utilizan como interfaz de computador.

Entre las características, al analizar el medio de transmisión, la comunicación inalámbrica presenta ventajas sobre medios cableados, ya que entrega al robot libertad de movimiento e instalación y la factibilidad de integrarlo en procesos de automatización con plataformas de desplazamiento. En adición, una arquitectura inalámbrica permite crear fácilmente una topología de comunicación punto multipunto, ampliando las posibilidades en la creación de sistemas robóticos y procesos de automatización complejos; en donde se facilita la creación de múltiples nodos de control con múltiples estaciones de monitoreo.

Tabla 3.6: Características técnicas de diferentes estándares de comunicación.

Estándar	Tipo de Comunicación	Velocidad de Transmisión	Integración de Protocolos	Seguridad
RS232	Cableado Punto a Punto	1Mbps	Modbus RTU	Conexión Física
802.11 Wifi	Inalámbrico Punto Multipunto	600Mbps	TCP/IP, TCP, UDP, MQTT, HTTP, SSL	WPA, WPA2, SSL, TLS.
802.3 Ethernet	Cableado Punto a Multipunto	1Gbps	TCP/IP, TCP, UDP, MQTT, HTTP, SSL	SSL, TLS Conexión Física
RS485	Cableado Punto a Punto	10Mbps	Modbus RTU	Conexión Física
USB	Cableado Punto a Punto	480Mbps	N/A	Conexión Física
802.15.1 Bluetooth	Inalámbrico Punto a Multipunto	100Mbps	N/A	Autenticación

Elaborado por: El Investigador, en base a Datasheet.

Los protocolos de comunicación que utilizan medios inalámbricos tienen una mayor vulnerabilidad en referencia a la seguridad informática. En los estándares cableados como: RS232, RS485, USB y Ethernet, se requiere de una conexión física para tener

acceso a la información transmitida. Por otro lado, los medios inalámbricos como Wifi y Bluetooth son vulnerables en ataques de sniffing y hacking; en donde al romper las seguridades de la red inalámbrica se tiene acceso a la información de la red.

La integridad de la información en una transmisión punto a punto o punto multipunto se garantiza mediante protocolos de comunicación que integran algoritmos de detección y corrección de errores, también se utilizan protocolos orientados a conexión, como TCP. Los estándares Wifi y Ethernet permiten la integración de protocolos de la capa de transporte, red y aplicación a los sistemas diseñados con el Scorbot, entre los protocolos que se pueden integrar están: TCP/IP, TCP, UDP, MQTT, HTTP, TLS entre otros. En los estándares RS232 y RS485 existe la factibilidad de integrar el protocolo Modbus, mismo que también puede ser integrado en Wifi y Ethernet, presentando a éstos últimos con mayores ventajas.

La seguridad de la información transmitida se incrementa con métodos de encriptación. En las aplicaciones realizadas bajo los estándares Ethernet y Wifi, es posible implementar protocolos de seguridad como TLS y métodos de autenticación; incrementando de esta manera la seguridad e integridad de la información difundida en la red.

En referencia a la velocidad de transmisión de datos, los estándares con mejores características son Ethernet, Wifi y USB. A pesar de que Ethernet permite una velocidad de transmisión superior, una mayor seguridad y las mismas capacidades de integración de protocolos que Wifi, se selecciona el estándar 802.11 (Wifi) debido a su característica inalámbrica, ya que con los protocolos de encriptación se robustece la seguridad y la diferencia en la velocidad de transmisión con Ethernet, no es significativa para las aplicaciones a implementarse.

El sistema de comunicaciones se realiza utilizando la tecnología Wifi como medio de transmisión para la interfaz de comunicación. Se selecciona esta tecnología debido a que el estándar de comunicación inalámbrica ofrece una mayor velocidad de transmisión de datos, llegando hasta a 600Mbps; además permite la integración directa de protocolos como TCP e IP que implementan algoritmos de control de flujo de datos, entregando confiabilidad en la transmisión de información. Al utilizar una tecnología Wifi se abre la posibilidad de implementar sistemas con el brazo robótico mediante

protocolos de la capa de aplicación del modelo OSI como, por ejemplo: HTTP/HTTPS, MQTT y Websockets con la adición del protocolo de encriptación de datos TLS; asegurando que la información transmitida no pueda ser capturada.

Para la selección del protocolo adecuado para el sistema, en la Tabla 3.7 se determinan algunas características de diferentes protocolos de comunicación basados en el uso de la comunicación TCP/IP, aplicables a la tecnología Wifi.

En el análisis de los protocolos se verifica que HHTP y HTTPS, basan su funcionamiento en una arquitectura cliente servidor, limitando al sistema a implementarse a establecer una comunicación mediante peticiones unidireccionales; es decir el servidor no puede enviar información al cliente, a menos que éste lo solicite.

Tabla 3.7: Características de distintos protocolos de comunicación implementados sobre TCP/IP.

Protocolo	Arquitectura	Tipo de Comunicación	Orientado a Conexión	Seguridad
Socket TCP/UDP	Cliente Servidor	Petición Bidireccional	Si	No
HTTP	Cliente Servidor	Petición Unidireccional Interfaz Web	Si	Autenticación
HTTPS	Cliente Servidor	Petición Unidireccional Interfaz Web	Si	Autenticación TLS
MQTT	Brocker Publicador/Suscriptor	Petición Bidireccional Interfaz Web	Si	Autenticación TLS Reglas
Websockets	Cliente Servidor	Petición Bidireccional Interfaz Web	Si	TLS

Elaborado por: El Investigador, en base a Datasheet.

En sockets TCP o UDP, la transferencia de información es bidireccional entre clientes y servidores, sin embargo, no es factible implementar de forma directa el protocolo de encriptación como SSL o TLS, dejando vulnerable la información a ataques de hacking. El socket UPD no está orientado a conexión, lo que implica que los paquetes perdidos no son retransmitidos.

La comunicación mediante websockets o MQTT, garantiza la transmisión de la información entre múltiples nodos bajo ciertos parámetros. En websockets se puede utilizar el protocolo TCP y en MQTT una calidad de servicio QoS de 2, que implica una garantía de que la información es entregada a su destino. En adición estos estándares permiten la integración del protocolo SSL o TLS, llevando a establecer una comunicación segura. MQTT es desarrollado para aplicaciones de ambientes industriales y requiere de un menor ancho de banda que websockets; en consecuencia, se determina MQTT es la mejor plataforma para implementar sistemas con el controlador del Scorbot que se diseña en el presente proyecto.

- **Interfaz Electrónica de Control**

La interfaz electrónica de control es el grupo de conexiones cableadas requeridas entre el controlador y los dispositivos eléctricos y electrónicos integrados en el brazo robótico. La interfaz utilizada se diseña en base al conector DD50, mostrado en la Figura 3.7, el mismo es un terminal macho de un cable que proviene de las conexiones de cada elemento electrónico que integra el brazo robótico.

El módulo de control se diseña con un puerto DD50 tipo hembra que permite realizar la conexión directa al brazo, utilizando la misma interfaz diseñada por el fabricante para la conexión directa al brazo, utilizando cable de 62 hilos de cobre, siendo la misma para conectar el equipo al controlador USB diseñado por la empresa Intelitek, fabricantes del brazo robótico Scorbot ER_4U.

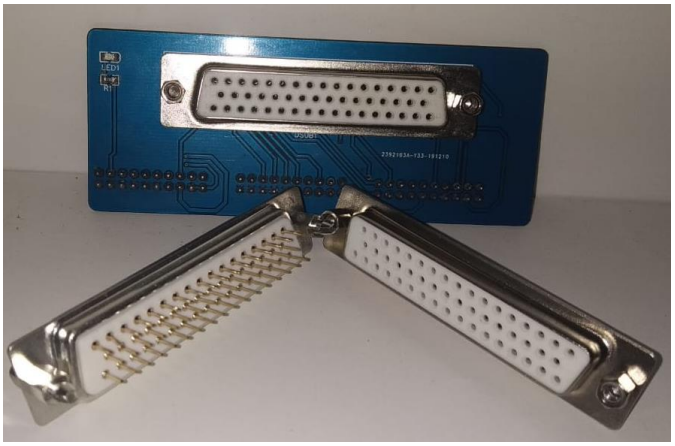


Figura 3.7: Interfaz electrónica del Scorbot ER_4U, terminal DD50.

3.2.4Arquitectura Final del Entorno de Control

La arquitectura final del entorno de control se refiere al sistema detallado a implementarse con el brazo del Scorbot ER_4U, integrando características y funciones del estándar Wifi y del protocolo MQTT, para el desarrollo de aplicaciones robóticas.

En la Figura 3.8 se detalla la estructura gráfica del ambiente de control con MQTT para aplicaciones con el Scorbot. En el diseño presentado el brazo robótico se conecta con un módulo electrónico mediante el cable del brazo a un puerto hembra DD50 del módulo.

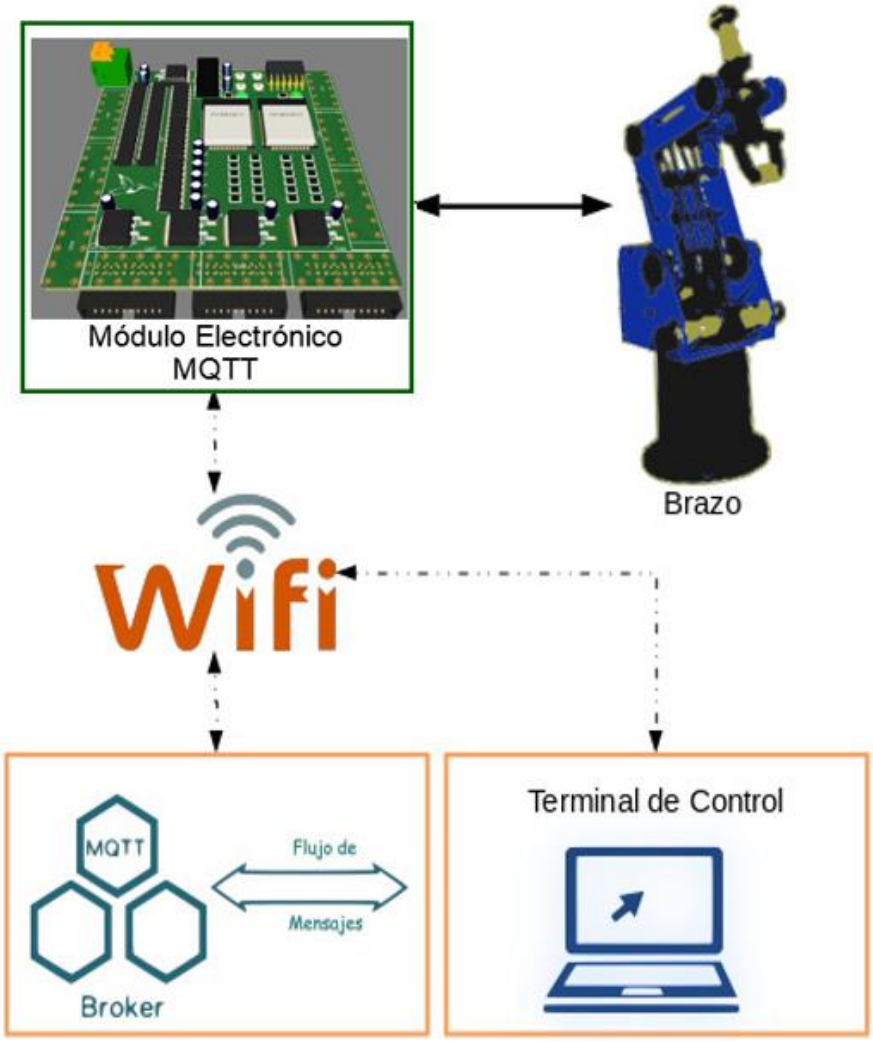


Figura 3.8: Arquitectura del entorno de control para el Scorbot ER_4U con interfaz Wifi - MQTT.

Elaborado por: El Investigador

El módulo electrónico MQTT de control constituye el grupo de circuitos integrados, microcontroladores y elementos electrónicos interconectados mediante circuitos impresos, el módulo incluye un firmware de control que integra las funciones de conexión inalámbrica Wifi y MQTT. La finalidad es recibir instrucciones desde equipos externos, interpretarlas y generar señales eléctricas que provoquen el movimiento de los ejes del Scorbot; y en forma inversa, procesar las señales eléctricas del brazo, generar instrucciones de control del movimiento y enviar alertas y estados de posiciones hacia el host de control.

El usuario controla y monitorea las funciones del brazo robótico conectándose al módulo de control con el protocolo MQTT a través de un punto de acceso inalámbrico. El protocolo MQTT permite el intercambio de información entre nodos de una red a través de una arquitectura de publicación y suscripción; la comunicación por medio de este protocolo requiere la implementación de un bróker, que recibe las publicaciones de los equipos terminales y la transfiere hacia los nodos suscritos utilizando “topics” para establecer distintos niveles y permisos de acceso. El brocker es instalado en un ordenador con la capacidad de ejecutar un sistema operativo de distribuciones Linux, Windows u Mac-Ox.

El terminal de control es un equipo terminal que permite al usuario monitorear, controlar y programar los movimientos del brazo robótico. Este nodo tiene la capacidad de ser implementado en el mismo computador que ejecuta el brocker, o a su vez en un dispositivo externo que tenga la capacidad de procesar funciones de publicación y suscripción con MQTT.

3.2.5 Módulo Electrónico MQTT

El módulo electrónico MQTT, es el conjunto de componentes analógicos y digitales encargados de gestionar el intercambio de información entre el brazo robótico y el terminal de control, además de ejecutar procesos de posicionamiento y protección de motores mediante el uso de microcontroladores.

El módulo de control requiere de un microcontrolador o tarjeta embebida que funcione como núcleo del procesamiento de datos y manipule los periféricos integrados en el

brazo robótico. En la Tabla 3.8 se detalla el número mínimo de señales de control que se requiere en el núcleo del módulo electrónico a implementarse.

El número total de señales requeridas es de 39, incluyendo en el circuito impreso módulos de comunicación wifi y sensores de corriente que permiten monitorear el comportamiento de los motores, detectando impactos. Utilizando sensores analógicos para medir la corriente, se requiere de un microcontrolador o una arquitectura de microcontroladores que tengan que entreguen 39 pines de control, 14 salidas digitales, una de ellas con modulación de ancho de pulso; 6 entradas analógicas, 17 entradas digitales y un puerto Usart.

Tabla 3.8: Número de pines del microcontrolador requeridos para el módulo electrónico.

Elemento del Scorbot	Cantidad	Pines de Control	Número Total de Pines	Tipo de Señal
Motores DC	6	2	12	Salida Digital
Encoders	6	2	12	Entrada Digital
Finales de Carrera	5	1	5	Entrada Digital
Sensores de Corriente	6	1	6	Entrada Analógica
Control de Velocidad	1	1	1	PWM
Señalización	1	1	1	Salida Digital
Módulo Wifi/USART	1	2	2	RS232
Total	26	10	39	

Elaborado por: El Investigador, en base a Datasheet.

En el controlador original del Scorbot ER_4U, Inteliteck utiliza como núcleo el microcontrolador NEC V853, un microcontrolador de 32 bits con una memoria RAM de 8KB, interfaz serial y hasta 100 pines de conexión. Este dispositivo trabaja a una frecuencia de operación de hasta 50 MHz, pero el circuito integrado no se ha encontrado en un mercado con la disponibilidad de importar hacia el Ecuador.

En la Tabla 3.9 se detalla las características técnicas de distintos microcontroladores y tarjetas electrónicas embebidos que pueden ser utilizados como núcleo del módulo a diseñarse.

Tabla 3.9: Características técnicas de microcontroladores y módulos embebidos.

Tipo de dispositivo	Módulo – Microcontrolador	Arquitectura	Frecuencia de Operación	Interfaces	Precio Aproximado
Microcontroladores	PIC18F4550	8 Bits 1KB RAM	48MHz	USB, EUART, PWM, I2C,SPI, 38 pines.	10.00 USD
	NEC V853	32 Bits 8KB RAM	33MHz	Serial, PWM, DSP, ADC, DAC,100 Pines	Sin Stock
	PIC32MX1xx	32 Bits 16KB RAM	50 MHz	USB, EUART, PWM, I2C, SPI, ADC,100 pines.	4.41 USD+ importación
Tarjetas de electrónicas embebidas	ESP WROOM32	32Bits 520KB RAM	240 Mhz 2 núcleos	Wifi, Ethernet, UART, SPI, I2C, DAC 6 ADC, 25 pines	3.15 USD + importación
	Arduino Mega 2560	8 Bits 8KB RAM	16MHz	USB, EUART, PWM, I2C,SPI, 54 pines.	19.00 USD
	ESP8266	32 Bits 36 KB RAM	160 MHz	Wifi, Ethernet, UART, SPI, I2C, DAC 6 ADC, 12 pines	11.00 USD

Elaborado por: El Investigador, en base a Datasheet.

El núcleo del Arduino Mega 2560 y el PIC18F4550 son microcontroladores de 8bits, además de 8 y 1KB de memoria RAM respectivamente; en comparación con los módulos ESP de Espressiff y la familia PIC32MX1xx las características de procesamiento son bajas. El Arduino Mega cumpliría con las características del número de pines requeridos, pero su frecuencia de operación es de 16MHz, la más baja entre los dispositivos comparados; el PIC18F4550 no tiene la cantidad suficiente de

pines por lo que el diseño del módulo electrónico con este dispositivo requiere del uso de dos microcontroladores.

Al ponderar de manera superior la frecuencia de operación, arquitectura y precio, descartando los computadores de placa reducida, el mejor dispositivo es el ESP WROOM 32. El ESP32 es un microcontrolador de 32 bits con dos núcleos, ejecuta los procesos a una frecuencia de hasta 240 MHz y en adición incluye conectividad Wifi. La desventaja de este dispositivo es que tiene solo 25 pines de control, sin embargo, por sus buenas características se selecciona este módulo como controlador de la tarjeta electrónica. Debido a que el dispositivo no tiene la cantidad de pines suficientes, se utilizan dos microcontroladores, intercomunicados mediante el protocolo RS232.

Tabla 3.10: Funciones y periféricos asignados a los microcontroladores del módulo electrónico.

Controlador	Periféricos	Funciones
Maestro	Encoders	Determinar la posición de los ejes Control de la Interfaz Wifi-MQTT Transferencia UART - MQTT - UART Control de la velocidad de los motores
	Finales de Carrera	
	Puerto de Programación	
	Indicador de Estado	
	Control de Velocidad	
Esclavo	Control de Motores	Generar el desplazamiento de ejes Medir las corrientes de los motores Transferencia de datos por UART
	Sensores de Corriente	
	Puerto de Programación	
	Indicador de Estado	

Elaborado por: El Investigador, en base a Datasheet

Al utilizar como núcleo del módulo electrónico MQTT dos ESP32, se distribuye las funciones y periféricos a controlar de forma equitativa, considerando los recursos de procesamiento requeridos en la ejecución de las distintas aplicaciones; así se consideran dos controladores: Maestro y Esclavo.

En la Tabla 3.10 se detalla los periféricos y funciones asignados a cada microcontrolador. El controlador maestro procesa las señales digitales de los encoders y finales de carrera (17 pines) para determinar la posición de cada uno de los ejes del robot. De forma adicional entrega una señal de pulso modulado en amplitud para controlar la velocidad de los motores y gestiona la comunicación por medio del puerto

de Transmisión y Recepción Asíncrono Universal (UART), Wifi y MQTT, utilizada para el intercambio de datos con el controlador esclavo y el terminal de control.

El controlador esclavo genera las señales eléctricas requeridas para ejecutar el movimiento de los motores del Scorbot; también procesa las señales analógicas de los sensores de corriente, detecta impactos y gestiona la transmisión de alertas en el apagado de emergencia de los motores. Cada micro-controlador tiene un puerto de programación RS232, que permite cargar o actualizar su firmware y un indicador de estado que se enciende en secuencia de pulsos para señalizar los errores de comunicación y de ejecución de procesos.

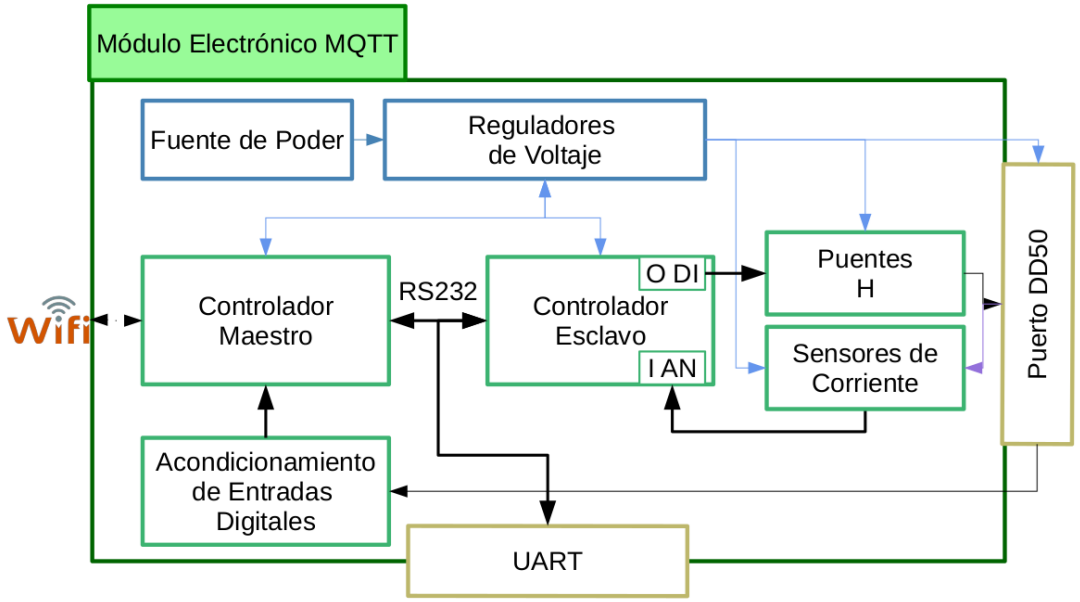


Figura 3.9: Diagrama de bloques de módulo electrónico MQTT.

Elaborado por: El Investigador

En la Figura 3.9 se define el diagrama de bloques de módulo electrónico MQTT, el mismo está compuesto por: fuente de poder, reguladores de voltaje, controladores, puentes H, sensores de corriente, etapa de acondicionamiento de entradas digitales, puerto DD50 para la conexión con el brazo y puerto UART para la programación del dispositivo.

Los elementos del circuito electrónico tienen diferentes voltajes de operación, en consecuencia, se utiliza una etapa de regulación de voltajes, para garantizar que cada componente tenga una fuente de tensión y corriente adecuadas.

Los encoders, finales de carrera, motores y fuentes de voltaje del brazo son conectados por medio del adaptador DD50. Las entradas digitales de los encoders y microswitches se conectan al controlador maestro por medio de una interfaz de acondicionamiento. Los puentes H, manejados por salidas digitales del esclavo, son una interfaz entre motores y controlador; desde los puentes H y el conector DD50 se alimenta la entrada de los sensores de corriente y la señal analógica de salida se envía al módulo ADC del microcontrolador.

En el ANEXO A, se visualiza el diagrama del circuito eléctrico del módulo MQTT, el mismo está divido en 4 secciones: drivers y sensores, en donde se tienen las conexiones de los puentes H y sensores de corriente; controlador maestro, que incluye los circuitos de las interfaces de entradas digitales y las conexiones del ESP32 maestro; fuente y reguladores, es la sección en donde se agrupa los conectores de la fuente de poder y los reguladores de voltaje requeridos.

Finalmente, en la sección del controlador esclavo se presenta las conexiones eléctricas del ESP32 esclavo y el diagrama de tres conectores header hembra de ángulo recto, (H3, H5 y H6), que son un puerto hacia una interfaz acoplada de forma perpendicular, la misma que integra un led que indica el encendido del módulo electrónico y el conector DD50 hembra que va hacia el brazo robótico.

Los detalles del diseño y funciones de cada uno de los periféricos que integra el módulo electrónico MQTT, se explica a continuación:

➢ **Puentes H**

Los puentes H son circuitos integrados utilizados para permitir a los motores de corriente continua del Scorbot girar en sentido horario y antihorario. El ESP32 tiene un voltaje de operación de 3.3V DC, y entrega una corriente máxima de 1A para todas las salidas; los puentes H se utilizan como interfaz de media potencia para pasar las señales del microcontrolador esclavo hacia los motores.

Los motores del Scorbot trabajan a un voltaje de operación de 12V DC, una corriente nominal de 500mA (700mA con carga) y una corriente de impacto mayor a 900mA, estas propiedades eléctricas determinan las características técnicas del driver de motores a utilizarse. En la Tabla 3.11 se detallan los parámetros eléctricos y físicos de operación diferentes puentes H. Todos los indicados están dentro de los parámetros adecuados en referencia al voltaje de operación, sim embargo se requiere analizar la corriente de operación para determinar el puente H adecuado.

Tabla 3.11: Características de operación de distintos puentes H.

Driver/ Características	Unidad	L293NE	L293D	L298	SN754410
Voltaje de salida	V	4.5--36	4.5--36	4.8--46	4.5--36
Corriente de salida continua	A	±1	±0.6	±2	±1
Corriente de salida pico	A	±2	±1.2	±3	±2
Nivel de 0 lógico	V	-0.3 a 1.5	-0.3 a 1.5	-0.3 a 1.5	-0.3 a 0.8
Nivel de 1 lógico	V	2.3 a 7	2.3 a 7	2.3 a 7	2 a 5.5
Temperatura de operación	°C	0 a 70	0 a 70	-25 a 130	-40 a 85
Potencia de Disipación	mW	1500	1500	25000	2075

Elaborado por: El Investigador, en base a Datasheet.

La fuente de corriente del L293D es de 600mA, inferior a los 700mA requeridos por los motores; por otro lado, el integrado L298 está sobre dimensionado, ya que entrega una corriente de 2A. Los dispositivos ideales son el L293NE y el SN754410, que entregan una corriente de 1A con picos de 2 amperios, estos dispositivos tienen la misma distribución de pines por lo que se consideran reemplazos mutuos. El

SN754410 tiene una mejor disipación de potencia y un mejor rango de temperatura de operación; sin embargo, para el mercado ecuatoriano es de difícil adquisición, aún con importación, por lo que se selecciona el puente H L293NE.

- **Sensores de Corriente**

Los sensores de corriente son circuitos integrados que permiten medir el consumo de corriente realizado por los motores del Scorbot. Debido a que el consumo de corriente de los motores supera los 900mA en condiciones de impacto en el movimiento de los ejes, estos sensores permiten detectar golpes en del brazo robótico.

Tabla 3.12: Características de los sensores de corriente.

Sensor/Características	Fuente Requerida	Rango de Medición	Sensibilidad
ACS710KLATR-6BB-T	3 -- 5.5 V DC 14.5 mA	±6A	151 mV/A
ACS711ELCTR-12AB-T	3 -- 5.5 V DC 5.5 mA	±12.5A	110mV/A
ACS712ELCTR-05B-T	4.5 – 5.5 13mA	±5A	185mV/A
ACS713ELCTR-20A-T	4.5 – 5.5 13mA	0 -- 20 A	185mV/A
ACS714ELCTR-05B-T	4.5 – 5.5 13mA	±5A	185mV/A

Elaborado por: El Investigador, en base a Datasheet.

En la Tabla 3.12 se detallan las características eléctricas de distintos sensores de corriente, entre los dispositivos indicados el ACS712ELCTR-05B-T y el ACS714ELCTR-05B-T tienen las mismas propiedades, estos sensores superan a los ACS710 y ACS711 en la sensibilidad de recepción, además su rango de medición que va de los -5 a 5 Amperios está más ajustado al rango consumido por los motores (-1A a 1A); el ACS713 solo mide corrientes positivas motivo por el que no sería adecuado.

El dispositivo seleccionado para el módulo electrónico MQTT es el ACS712ELCTR-05B-T.

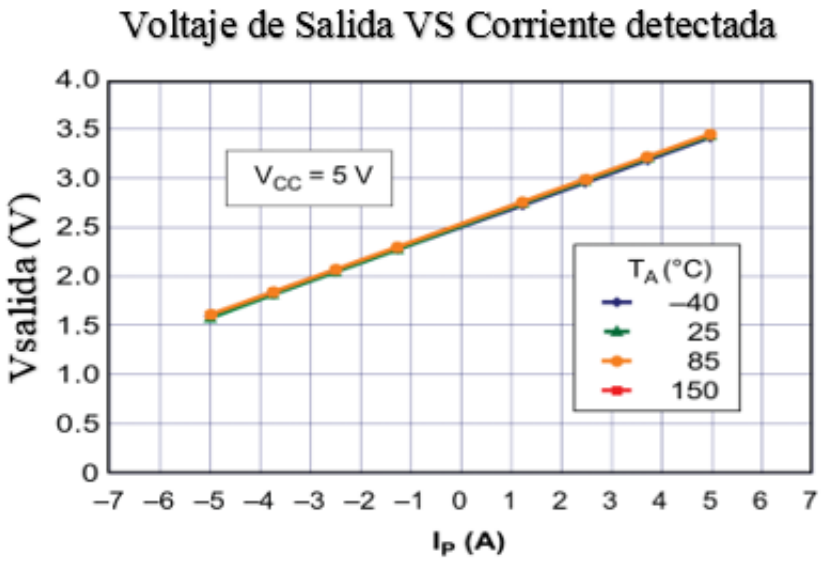


Figura 3.10: Relación entre voltaje de salida y corriente medida en el sensor ACS712ELCTR-05B-T [40].

La salida analógica del sensor utilizado entrega 185mV por cada amperio, la relación entre el voltaje y la corriente medida es la indicada en la Figura 3.10, en donde se visualiza que la diferencia de potencial en la salida del sensor va desde 1.5 a 3.5 voltios y responde a la ecuación 3.1, que es la ecuación de la recta para los puntos entregados por la imagen. Las entradas analógicas del microcontrolador ESP32 soportan hasta un voltaje de 3.3V DC, para evitar daños al dispositivo el voltaje de salida del sensor, es truncado para que no supere este valor umbral, limitando la corriente máxima detectada por el sensor a una magnitud de 4 amperios.

Ecuación 3.1

$$V_O = 0.2I + 2.5$$

$$3.3 = 0.2I + 2.5 \rightarrow I = 4\,A$$

En donde:

V_O Es el voltaje de salida del sensor analógico.

I Es la corriente consumida por la carga.

El valor absoluto de la corriente de los motores en la producción de impactos es mayor a 900mA, colocando como umbral este valor, se detecta los golpes en los ejes del brazo cuando V_o es mayor a 2.68V DC y menor a 2.32V DC.

El esquema de conexión electrónica de la interfaz de puentes H y sensores de corriente se indica en la Figura 3.11. Los integrados L293NE son alimentados con una fuente de 24V DC y cada uno de ellos maneja el giro de dos motores, la velocidad angular de los motores se controla con una señal PWM única y las señales de control (SMXX) se conectan de forma directa al ESP32.

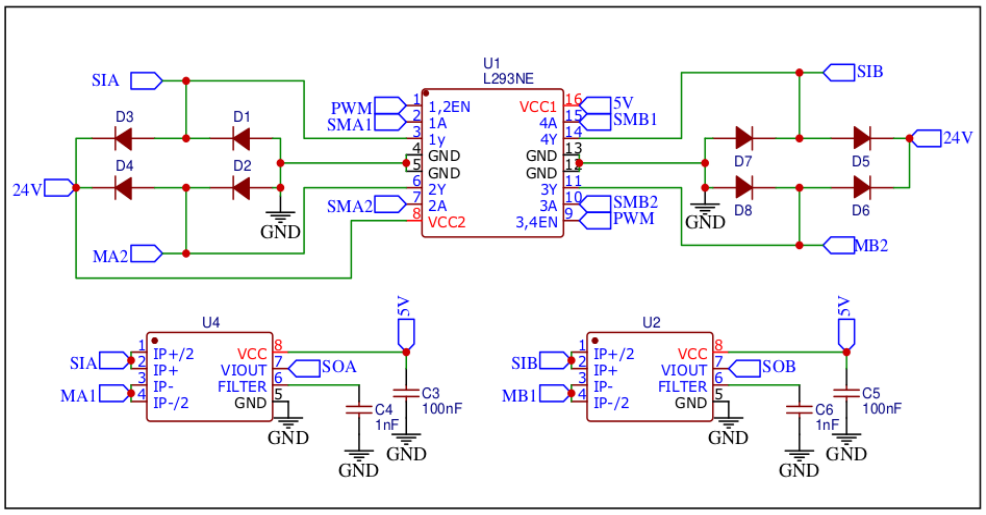


Figura 3.11: Circuito electrónico de puentes H y sensores de corriente.

Elaborado por: El Investigador en el software Easyeda.

Las salidas SIX del puente H se conectan en serie a las entradas positivas de los sensores de corriente; los terminales MXX, tanto de los sensores como de los drivers, van hacia los pines del motor en el conector DD50 y las salidas analógicas SOX se conectan directamente a las entradas analógicas del microcontrolador. Los ACS712ELCTR-05B-T son energizados con una fuente de 5V DC y se utilizan capacitores de 1nF y 100nF, de acuerdo a lo recomendado por el fabricante, para filtrar y estabilizar la señal de salida analógica.

➢ **Acondicionamiento de Entradas Digitales**

El acondicionamiento de entradas digitales es un grupo de atenuadores tipo L, que permiten reducir los voltajes de 5 V DC, pertenecientes a las señales de 1 lógico de los encoders y finales de carrera del Scorbot, a la tensión adecuada para las entradas del microcontrolador. Los niveles de 1 lógico de los ESP32 van desde los 2V DC hasta los 3.3V DC, en consecuencia, se debe reducir los 5 V DC que entregan los periféricos del Scorbot a un valor que se encuentre dentro del rango indicado.

En la Figura 3.12 se observa una sección de los atenuadores utilizados, en sí, para cada entrada digital se utiliza un par de resistencias conectadas en serie. Las señales E_XX se dirigen hacia los pines del conector DD50, los terminales opuestos de las resistencias en serie se conectan a tierra, creando un divisor de tensión y generando una tercera señal atenuada EXX que es la que se conecta hacia las entradas digitales del controlador maestro.

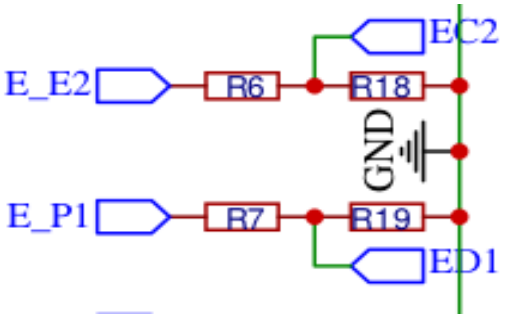


Figura 3.12: Atenuadores tipo L para las entradas de los sensores del Scorbot.

Elaborado por: El Investigador en el software Easyeda.

Los valores de las resistencias en los divisores de tensión se determinan utilizando las leyes de kirchhoff y de Ohm de acuerdo a la ecuación 3.2, que corresponde a la relación matemática de un divisor de tensión:

Ecuación 3.2

$$V_e = \frac{V_i R_g}{R_i + R_g}$$

En donde:

V_e Es la señal de voltaje en atenuado que se envía al microcontrolador.

R_g Es la resistencia conectada a tierra, desde la que se toma la señal atenuada.

R_i Es la resistencia que se conecta al terminal de entrada de 5V DC.

Para efectos de diseño y por las recomendaciones del fabricante se establece a un valor de 330 [Ω], por lo que el valor de se determina de acuerdo a las condiciones de la ecuación 3.2, en donde se establece que el valor del voltaje de entrada al módulo ESP 32, tiene que ser mayor a 2 voltios e inferior a 3.3.

Ecuación. 3.3

$$2.0 \leq V_e \leq 3.3$$

Entonces reemplazando V_e por la ecuación 3.2 y el valor de R_g al resolver la inecuación se tiene el rango de valores aceptables para R_i:

$$2.0 \leq \frac{5v * 330\Omega}{R_i + 330\Omega} \leq 3.3$$

$$186.79\Omega \geq R_i \geq 105.84[\Omega]$$

$$105284 \leq \text{Ri} \leq 186.79$$

El valor de la resistencia que se conecta al terminal con la señal de 5V DC debe ser superior a 105.84 [Ω]e inferior a 186.79 [Ω]. Se utiliza una resistencia con un valor comercial de 180 [Ω], entregando así al microcontrolador un voltaje V_e de 3.23 V DC.

- **Reguladores de Voltaje**

Las regulaciones de voltaje son circuitos integrados que permiten acondicionar la tensión eléctrica que entrega la fuente de alimentación al módulo electrónico, para que cada componente de la tarjeta electrónica reciba los valores de voltaje y corriente adecuados.

Tabla 3.13: Fuentes requeridas para los componentes del módulo electrónico MQTT.

Dispositivo	Voltaje V DC	Corriente mA	Regulador
Controlador Maestro	3.3	500 – 100	K7803
Controlador Esclavo	3.3	500 – 1000	K7803
Sensores de Corriente Puentes H	5	13 *6	L7805CDT2
Encoders	5	400 c/u	L7805CDT
Finales de Carrera Led de Encendido	5	450	L7805CDT

Elaborado por: El Investigador.

En Tabla 3.13 se detalla los valores de voltaje y consumo de corriente requeridos por los componentes que integran la tarjeta de control. Los microcontroladores maestro y esclavo trabajan a un voltaje nominal de 3.3V DC y consumen una corriente máxima de 1A cada uno, para cada controlador se utiliza un regulador K7803.

En la Figura 3.13 se observa los reguladores L7805 entregan un voltaje de salida de 5V a 1A, éstos integrados son utilizados para alimentar el resto de componentes de acuerdo a la siguiente descripción: los sensores de corriente y los niveles de voltaje lógico de los puentes H son alimentados con un regulador, calculando un consumo de corriente de 200mA; los finales de carrera se conectan en paralelo al led indicador de encendido, teniendo un consumo de corriente estimado de 450mA; los encoders consumen 400mA aproximadamente por lo que se utiliza 3 L7805 para los seis dispositivos.

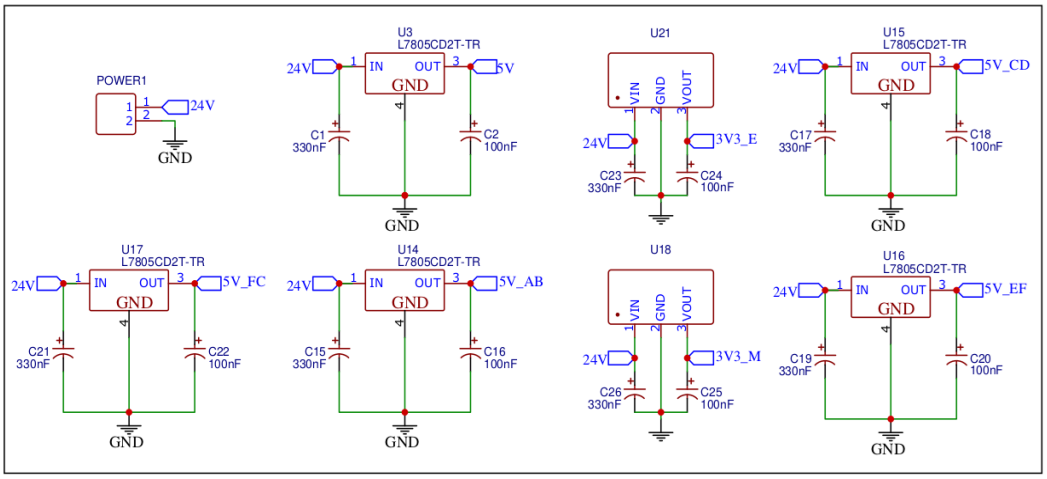


Figura 3. 13: Reguladores de voltaje para alimentar los componentes.

Elaborado por: El Investigador – Easyeda

➢ Fuente de Poder

La fuente de poder es un conjunto de componentes eléctricos que permiten transformar la energía eléctrica de corriente alterna, en una fuente de corriente continua con un voltaje de salida constante y estable en el tiempo. El módulo electrónico MQTT, es alimentado con una tensión eléctrica de 24 V DC siguiendo las características del controlador diseñado por Inteliteck para garantizar el correcto funcionamiento de los motores.

La corriente mínima de la fuente de poder se determina estimando el consumo total de potencia del módulo electrónico MQTT y los periféricos del brazo robótico. En la Tabla 3.14 se calcula la potencia total consumida, especificando el valor de voltaje y corriente requeridos para cada fuente que alimentan los circuitos integrados, sensores y motores que intervienen en el sistema. La potencia requerida es de 116.1 W, sin embargo, debido a que no se ha considerado el consumo de elementos electrónicos menores, y para asegurar el correcto funcionamiento del sistema se realiza un incremento del 20%, recomendando el uso de una fuente de 140W.

Tabla 3.14: Potencia estimada de consumo del módulo electrónico MQTT.

Elemento	Fuente	Voltaje [V] DC	Corriente [A]	Potencia [W]
ESP32 Esclavo	K7803	3.3	1	3.3
ESP32 Maestro	K7803	3.3	1	3.3
ACS712ELCTR L293D Lógico	L7805C	5	1.5	7.5
Encoders	L7805C (x3)	5	1.5	22.5
Microswicth Led de Encendido	L7805C	5	1.5	7.5
Motores DC	L293D (x6)	12	1	72
Total				**116.1**

Elaborado por: El Investigador

La corriente de salida requerida para la fuente de poder se calcula en función de la ecuación 3.4.

Ecuación 3.4: $$P_s = V_s * I_s \rightarrow I_s = \frac{P_s}{V_s}$$

$$I_s = \frac{140}{24} = 5.8A$$

En donde

P_s Es la potencia recomendada, 140 W

V_s Es la tensión de alimentación del módulo electrónico MQTT, 24V DC

I_s Es la corriente requerida por la fuente

Entonces se recomienda el uso de una fuente de alimentación de 24V DC a 6A; o una fuente alternativa con una potencia de 140W en un rango de voltajes de 15V DC, que es un voltaje mínimo recomendado para evitar caídas de tensión a 36V DC, que es el voltaje máximo soportado por los reguladores de tensión y los puentes H. El módulo diseñado se alimenta con la fuente de poder WX-DC2416, ésta tiene un voltaje de alimentación de 100 a 240 V AC a 150W y una salida de 24 V DC con una corriente de de 6 a 9 amperios.

- **Núcleo de Control**

El núcleo de control es un dúo de módulos ESP32 WROOM 32 intercomunicados por medio del protocolo RS232; encargados de gestionar la comunicación MQTT, administrar la ubicación de los ejes del brazo robótico e identificar las alertas en el funcionamiento del Scorbot ER_4U.

En la Figura 3.14, se observa el diagrama de las conexiones eléctricas del controlador maestro y esclavo. Los pines 1 y 2 de los microcontroladores son de polarización, a los mismos se conectan tierra y las fuentes independientes de 3.3V DC, respectivamente. A cada pin 3 de los ESP32 se conecta en paralelo una resistencia pull-up de 12kΩ (recomendación del fabricante) y un pulsador dirigido a tierra; al presionar en botón el microcontrolador entra en estado de reset. De forma similar, en los pines 25 (DIO_0) se conectan otros pulsadores con su respectiva resistencia, denominados pulsadores de boot; estos botones permiten a los ESP32 ingresar en modo programación.

En la configuración realizada los módulos ESP32 WROOM entran en modo de ejecución del firmware con solo energizar la tarjeta electrónica. Para ingresar en modo boot, que permite cargar o actualizar el código del firmware, se debe seguir el siguiente procedimiento:

1. Presionar los botones de boot y reset al mismo tiempo.
2. Soltar el botón de reset y esperar de 1 a tres segundos.
3. Soltar el botón de boot.
4. El microcontrolador está listo para cargar el código por medio del puerto UART configurado en la interfaz de programación.

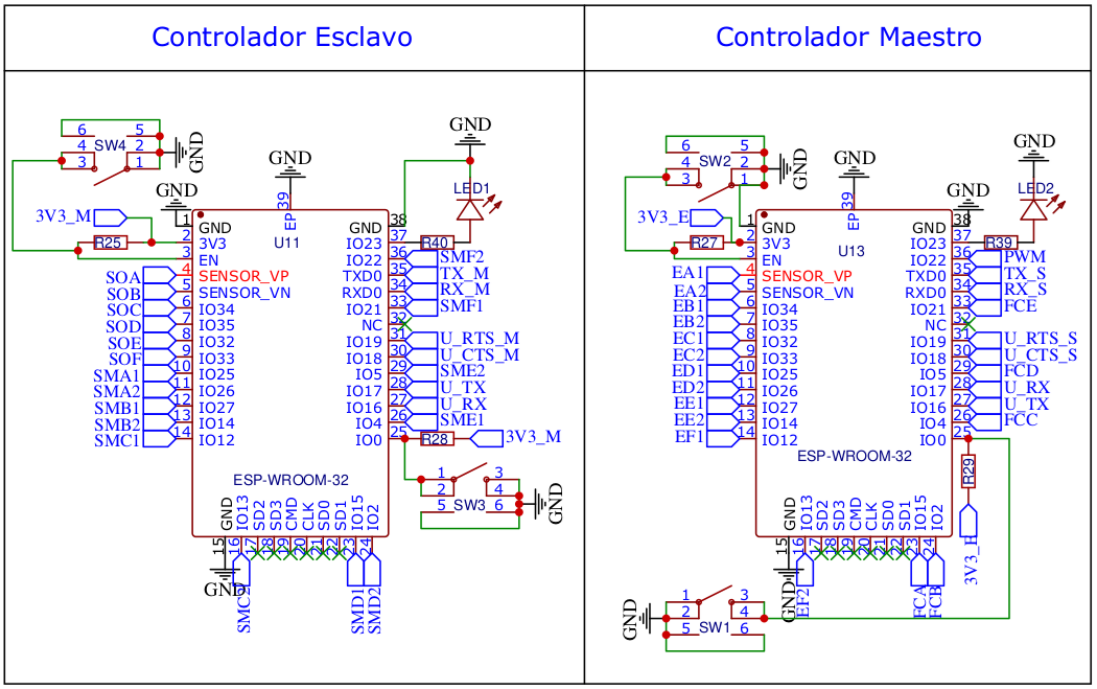


Figura 3.14: Esquema eléctrico de las conexiones del núcleo de control.

Elaborado por: El Investigador - Easyeda

Los microcontroladores del ESP32 tienen tres módulos de comunicación UART numerados de 0 a 2. El maestro y esclavo intercambian información por medio del puerto UART1 utilizando los pines 16 y 17 de cada dispositivo. Los pines 30,31,34 y 35 crean un puerto de comunicación al conectarse a un header hembra de dos filas, que permite añadir un dispositivo externo de comunicación RS232, I2C o SPI. La fila superior del conector pertenece al maestro y la inferior al esclavo, este puerto permite realizar la carga de los programas a los microcontroladores, por medio del puerto UART0 y utilizando un convertidor USB a RS232 de niveles lógicos TTL.

En el esclavo, el rango de pines 4-9 es utilizado como entradas de las señales analógicas de los sensores de corriente, el pin 37 es la salida del led indicador de estado y el grupo complementario de pines son las salidas que controlan el giro de los motores.

En el maestro, el grupo de pines 4-14 incluyendo el 16 son las entradas digitales de los encoders, el pin 37 se conecta al led indicador de estado, el 22 es la señal PWM que

controla la velocidad de los motores y el grupo complementario de pines son las señales de entrada de los finales de carrera.

➢ Tarjeta Primaria

La tarjeta primaria es el circuito impreso que contiene todos los dispositivos electrónicos del módulo MQTT a excepción de la interfaz DD50 para la conexión con el Scorbot. El circuito se realiza en un PCB cuadrado de 6 capas que ocupa una superficie total de 100 cm². La imagen tridimensional del circuito diseñado se observa en la Figura 3.15. La tarjeta primaria es alimentada eléctricamente por medio de un conector WJ142V de dos pines, para ello se debe utilizar una fuente externa de 24V DC.

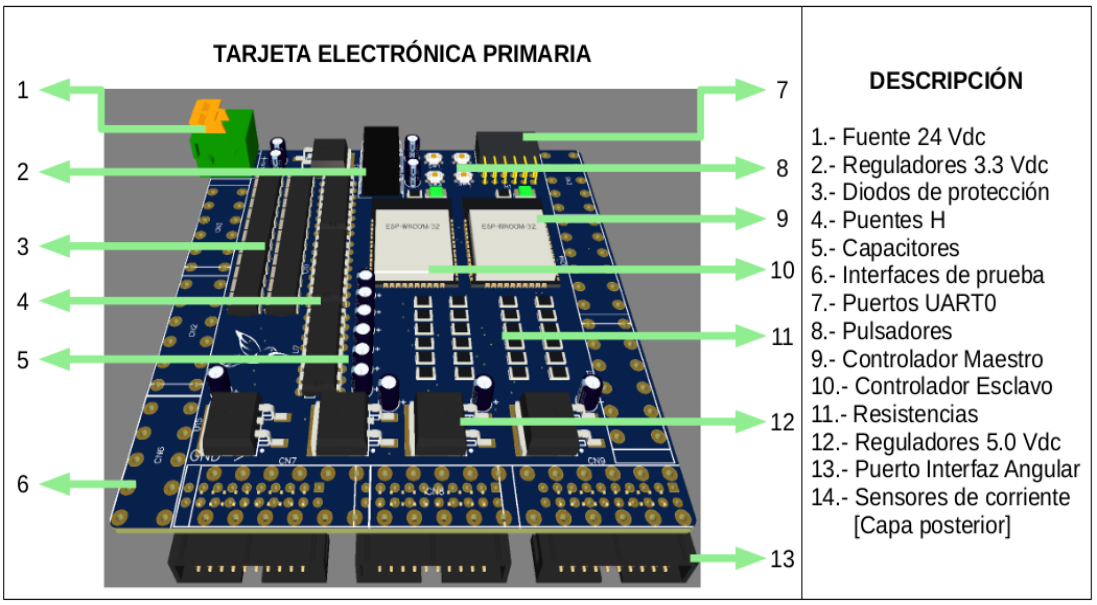


Figura 3.15: Imagen tridimensional de la tarjeta primaria.

Elaborado por: El Investigador – Easyeda.

El módulo diseñado posee tres interfaces de entrada y salida, de acuerdo a la imagen los puertos son el 6, 7 y13.

El puerto 6 es una interfaz paralela al puerto 13, el mismo permite conectar terminales de prueba para hacer mediciones en los pines del conector DD50. El puerto 13 es consta de tres headers macho horizontales que permiten añadir una interfaz angular; una tarjeta electrónica secundaria que acopla de la conexión con el Scorbot. El puerto 7 es un header hembra de 12 pines distribuidos en dos filas, cada fila conecta a 1 pin de tierra y 4 pines del controlador maestro y esclavo, esta

conexión facilita la creación de puertos de comunicación con los microcontroladores ESP32 y se utilizan para programar los mismos.

➢ Interfaz Angular

La interfaz angular es un circuito impreso utilizado como adaptador de conexión entre el circuito PCB de la tarjeta primaria y el conector DD50 macho del Scorbot ER_4U. La tarjeta de circuito impreso del módulo MQTT utiliza la interfaz angular para conectarse de forma perpendicular al cable del brazo robótico, teniendo de esta manera una conexión más ergonómica, menos invasiva y facilitando el diseño de una carcasa de protección para el módulo.

En la Figura 3.16 se muestra el esquema del circuito eléctrico de la interfaz angular, el mismo está compuesto por un conector DD50 vertical, un led indicador y tres headers verticales tipo hembra.

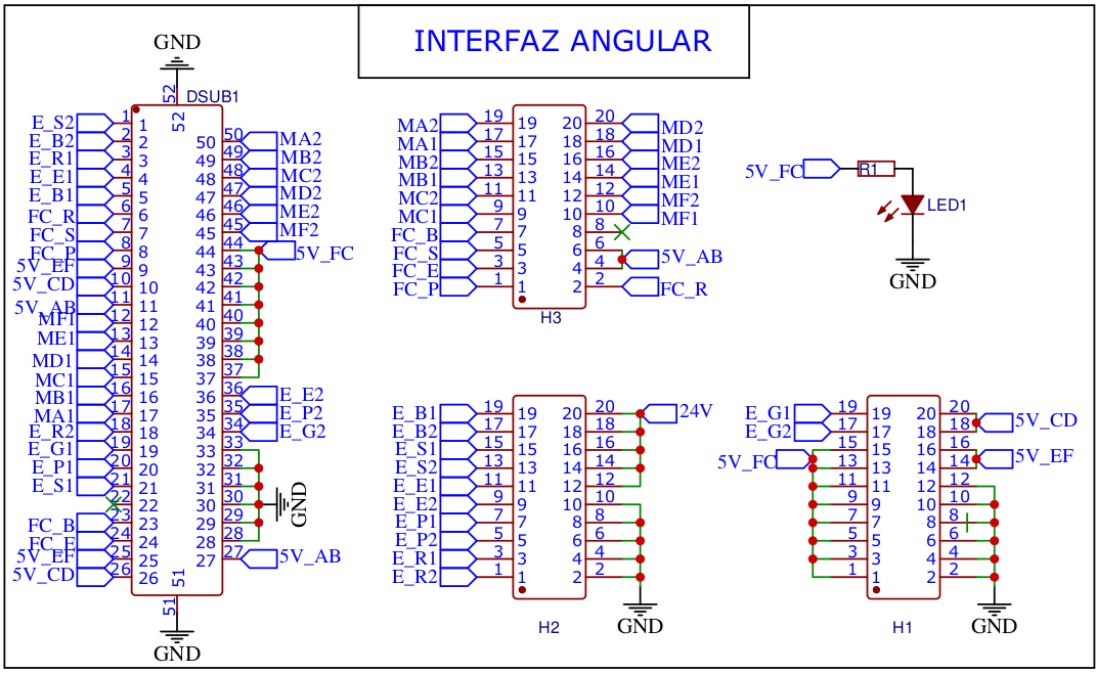


Figura 3.16: Esquema del circuito electrónico de la interfaz angular.

Elaborado por: El Investigador – Easyeda.

Los conectores H1, H2 y H3 se acoplan de forma directa a los headers horizontales H3, H5 y H6 de la tarjeta primaria, respectivamente. Desde los headers verticales se realizan las conexiones correspondientes para que las señales del módulo MQTT

coincidan con los pines adecuados del conector DD50 macho perteneciente al brazo robótico.

En la Figura 3.17 se ilustra la imagen tridimensional del circuito PCB de la interfaz angular. La tarjeta electrónica diseñada tiene una superficie de 40 cm² en una figura rectangular de 4 cm de altura por 10 cm de ancho y se acopla por el lado más largo a la tarjeta primaria.

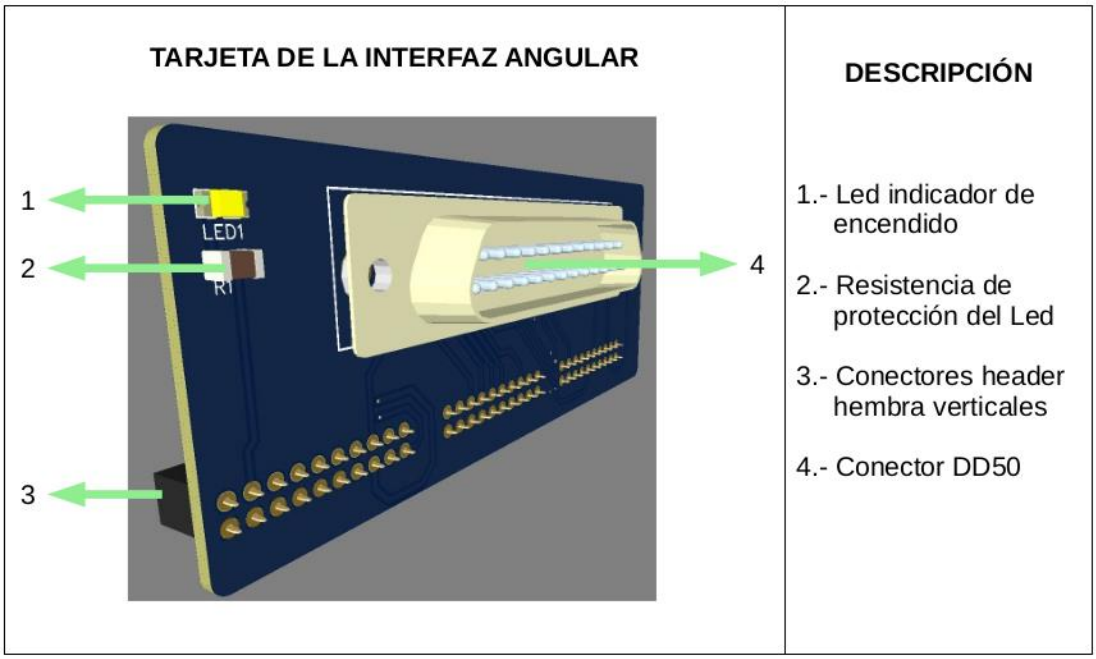


Figura 3.17: Visualización tridimensional de la interfaz angular.

Elaborado por: El Investigador – Easyeda.

La tarjeta electrónica consta de 4 elementos: puerto DD50 para la conexión directa con el cable del Scorbot, un diodo led que se enciende cuando el módulo es energizado y tres headers de 20 pines distribuidos en dos filas que forman un puerto para conectarse con los heardes horizontales de la tarjeta primaria.

3.2.6 Firmware de Control

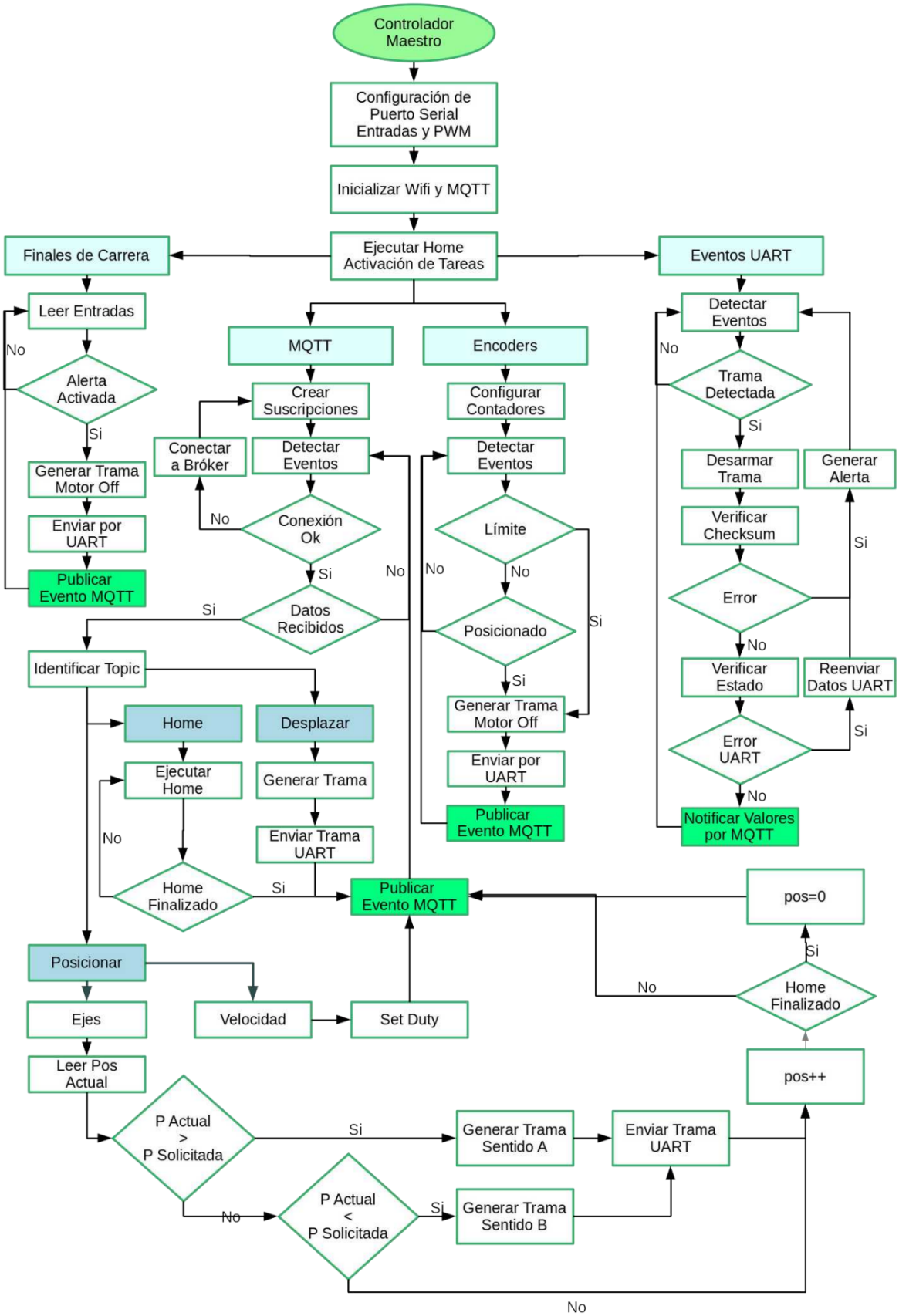
El Firmware de control es el software creado para controlar los periféricos que integran el brazo robótico. El módulo MQTT divide el software en dos secciones, la primera controla los periféricos conectados al Maestro y la segunda los del Esclavo. Los programas son elaborados en lenguaje C en el entorno de desarrollo ESP-IDF, que es

la interfaz de programación que provee la empresa Espressif Systems, fabricantes del microcontrolador.
La interfaz de programación de ESP-IDF permite manipular los micro-controladores mediante una ejecución de tareas, en donde se multiplexa las tareas de acuerdo a la prioridad para simular una ejecución en paralelo de múltiples ciclos infinitos.

- **Firmware del Maestro**

El firmware del Maestro es el software desarrollado en lenguaje C que gestiona la comunicación con el brocker MQTT y detecta de forma interna la posición en la que se encuentra el brazo, para generar los desplazamientos requeridos en los ejes del robot.

En la Figura 3.18 se ilustra el diagrama de flujo del software que ejecuta el controlador Maestro. El programa se divide en cuatro tareas: Finales de Carrera, MQTT, Encodres y Eventos UART; cada una de ellas son procesos ejecutados en los dos núcleos del ESP WROOM 32 de forma infinita e independiente. Al encender el Maestro se cargan las configuraciones iniciales de los pines que permiten manipular los periféricos.
En la configuración del puerto serial se establecen los parámetros del UART1, utilizado para intercambiar información con el esclavo. La comunicación se realiza a un baud_rate de 115200, con tramas de 8 bits y un bit de stop. De forma adicional se configura los pines de la ESP conectados a las señales de los encoders y finales de carrera, como entradas y los conectados al led de estado y señal PWM, como salidas digitales.



Elaborado por: El Investigador.

Figura 3.18: Programa del Controlador Maestro.

Al finalizar la configuración de pines para la gestión de periféricos se inicializa la red inalámbrica y administra la conexión con el broker MQTT. El microcontrolador Maestro es configurado como una estación y se conecta a un punto de acceso del que recibe una dirección IP mediante DHCP. Una vez establecida la conexión con el punto de acceso se crea un cliente MQTT mediante el protocolo TLS V1.2, utilizando un certificado alojado en la memoria del programa. La activación de las 4 aplicaciones o tareas que se ejecutan en el Maestro se generan solo si se ha establecido la conexión con el broker MQTT.

a. Aplicación Finales de Carrera

En la aplicación de finales de carrera se identifica si ha existido un cambio de estado en los cinco pines correspondientes a las conexiones con los micro-switches de los ejes del robot. En en caso de identificar un cambio de estado lógico se crea una trama para identificar el eje y se publica el evento enviando en el mensaje el valor de la entrada digital al topic "id_scorbot/S/F/#"; en donde S es el grupo de sensores, F el directorio de finales de carrera y # es la primera letra mayúscula que hace referencia al eje en donde se produjo el evento (B-S-E-P-R-G).

b. Aplicación MQTT

La aplicación MQTT es una función que detecta los eventos producidos en la recepción de mensajes mediante este protocolo. Al iniciar la aplicación se crean las suscripciones requeridas por el maestro para recibir la información desde el host de control.
En la Figura 3.19 se describe la estructura de las variables a las que se suscribe el módulo MQTT; el módulo del brazo robótico permite ejecutar funciones de desplazamiento y posicionamiento, para ello se debe publicar desde un host externo los valores de posición en pulsos, o de encendido y apagado en los topics correspondientes, a la función requerida. En la estructura de variables el signo # identifica a la raíz de directorios del broker utilizado en el intercambio de información. En el segundo nivel, la letra D es utilizada para las funciones de desplazamiento, la letra P para las de posicionamiento y el topic HOME permite tarar el brazo robótico.

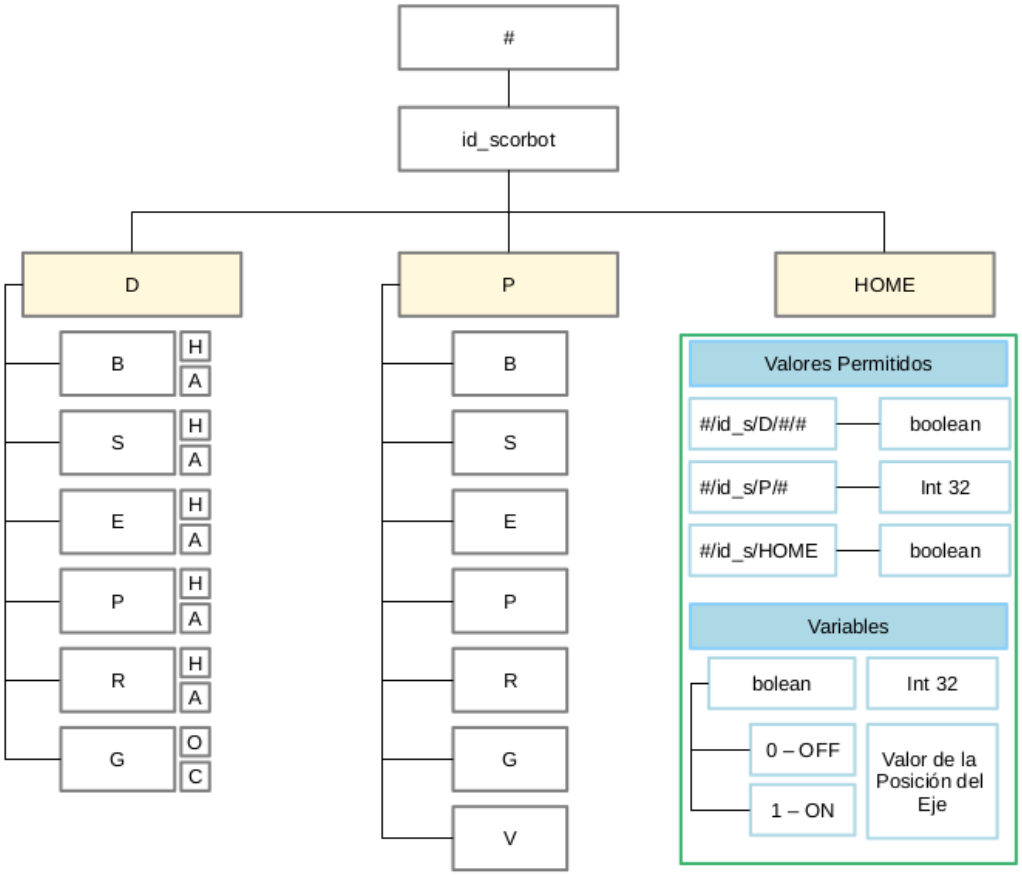


Figura 3.19: Variables de suscripción del módulo MQTT.

Elaborado por: El Investigador.

La función de desplazamiento permite mover los ejes de forma continua a voluntad del usuario; para utilizar esta función, desde un host externo se publica valores booleanos en los topics "id_scorbot/D/#/sentido", en donde # es la primera letra mayúscula del eje a desplazar y el subtopic sentido indica la dirección de giro; H se utiliza para mover el motor de forma horaria y A en forma antihoraria. El valor del mensaje determina si se enciende o se apaga el motor del eje, así por ejemplo, si se publica el valor de '1' en el topic "id_scorbot/D/B/H", el motor de base se encenderá en sentido horario y mantendrá su estado hasta recibir el valor '0' en el mismo topic.

En la función de posicionamiento el usuario indica de forma específica la ubicación que debe tomar cada uno de los ejes del brazo robótico. Para utilizar esta funcionalidad se debe publicar un valor entero de 16 bits en el topic "id_scorbot/P/#" en donde # es el indicador del eje del brazo. El número publicado en el mensaje es un entero con signo de 16 bits que corresponde a la cantidad de pulsos que debe desplazarse el eje a

partir de la posición de home, indicada por los finales de carrera; si el valor numérico es positivo el motor se desplaza en sentido horario y si es negativo el desplazamiento es en sentido antihorario.

En la aplicación MQTT activa la detección de eventos después de crear las suscripciones a los tres topics: “id_scorbot/D/#”, “id_scorbot/P/#” y “id_scorbot/HOME”. Con la detección de eventos es posible gestionar la re-conexión con el broker en el caso de perder la comunicación e identificar la recepción de mensajes.

Al recibir recibir un mensaje desde el broker, se identifica el topic al que pertenece y se ejecuta la función correspondiente. Si el topic pertenece a la función desplazar se envía la gestión al esclavo mediante una trama UART; en el caso de tratarse de una petición de posicionamiento, se lee la posición actual del eje, se compara para identificar si es mayor o menor a la posición de la solicitud y en función del resultado se crea una trama que indica al esclavo la dirección del desplazamiento. Cuando el controlador detecta que ha finalizado el posicionamiento de todos los ejes solicitados, publica el mensaje fin en el topic “id_scorbot/host”

Dentro del grupo de topics pertenecientes al desplazamiento se encuentra el subtopic para configurar el duty cylcle de la velocidad, este valor debe ser el primero en ser publicado antes de gestionar un posicionamiento y no debe superar el 50%. Si no se publica la velocidad deseada el programa utilizará el último valor configurado. El topic de velocidad permite crear paros de emergencia gestionados por software; para ejecutar esta función se debe publicar el valor de ‘0’ en la dirección “id_scorbot/P/V”.

A continuación, se detalla el código fuente utilizado en la aplicación ESP IDF, para crear las suscripciones a cada uno de los topics requeridos. La publicación en los topics se realiza con la función *esp_mqtt_client_publish (),* desde cualquier parte del código requerido en el Firmware.

Aplicación que controla los eventos de la comunicación MQTT en el ESP32

```
static esp_err_t mqtt_event_handler_cb(esp_mqtt_event_handle_t event) {
```

Creamos una variable cliente que almacena la información de los eventos

```
esp_mqtt_client_handle_t client = event->client;
```

int msg_id; // Identificador del evento

```
esp_mqtt_client_handle_t client = event->client;
```

switch (event->event_id) { // Separa las funciones según los eventos presentados en la comunicación MQTT

```
esp_mqtt_client_handle_t client = event->client;
```

case MQTT_EVENT_CONNECTED: // Evento de conexión exitósa al broker

```
esp_mqtt_client_handle_t client = event->client;
```

Indica al bróker que el brazo se ha conectado

```
msg_id = esp_mqtt_client_publish(client, "havel/led/conexion", "wifi",
```

Crea una suscripción general al directorio havel/D con una calidad de servicio QAS=2, en donde havel es el identificador del brazo, D el dominio de desplazamiento, P el dominio de Posicionamiento.

```
ESP_LOGI(TAG_W, "MQTT CONECTADO!!");msg_id = esp_mqtt_client_subscribe(client, "havel/D/#", 2);
ESP_LOGI(TAG_W, "sent subscribe successful, msg_id=%d", msg_id);
msg_id = esp_mqtt_client_subscribe(client, "havel/P/#", 2);
ESP_LOGI(TAG_W, "sent subscribe successful, msg_id=%d", msg_id);
msg_id = esp_mqtt_client_subscribe(client, "havel/HOME", 2);
ESP_LOGI(TAG_W, "sent subscribe successful, msg_id=%d", msg_id);
break;
```

Evento que indica problemas de conexión con el Broker

```
case MQTT_EVENT_DISCONNECTED:
ESP_LOGI(TAG_W, "MQTT_EVENT_DISCONNECTED");
break;
```

Evento que indica suscripción exitosa

```
case MQTT_EVENT_SUBSCRIBED: // Evento que indica suscripción exitosa
ESP_LOGI(TAG_W, "MQTT_EVENT_SUBSCRIBED, msg_id=%d", event->msg_id);
msg_id = esp_mqtt_client_publish(client, "havel/led/conexion", "mqtt", 0, 2, 0);
ESP_LOGI(TAG_W, "sent publish successful, msg_id=%d", msg_id);
break;
```

Evento que indica finalización de Suscripción

```
case MQTT_EVENT_UNSUBSCRIBED
ESP_LOGI(TAG_W, "MQTT_EVENT_UNSUBSCRIBED, msg_id=%d", event->msg_id);
break;
```

Evento que indica publicacion satisfactoria

```
case MQTT_EVENT_PUBLISHED
ESP_LOGI(TAG_W, "MQTT_EVENT_PUBLISHED, msg_id=%d", event->msg_id);
break;
```

Evento que controla la recepción de información por MQTT

```
case MQTT_EVENT_DATA:
 ESP_LOGI(TAG_W, "MQTT_EVENT_DATA");
 char mqtt_data[50];
char mqtt_topic[11];
snprintf(mqtt_data, event->data_len+1, "%s", event->data);
snprintf(mqtt_topic, event->topic_len+1, "%s", event->topic);    …}
return ESP_OK;
}
```

c. Aplicación Encoders

La aplicación encoders es un grupo de seis módulos contadores de pulsos que permiten

identificar la posición en la que se encuentra cada uno de los ejes. Al iniciar la ejecución de la aplicación, se requiere configurar cada unidad contadora de pulsos; para este fin se indican dos los valores límites, un estado triguer y los pines de señal y dirección de cada encoder. Los límites son los valores máximos y mínimos que puede alcanzar cada eje y triguer es el valor en pulsos que se utiliza como comparación para detectar que un eje ha llegado a la posición solicitada por MQTT.
Al inicializar los encoders se habilitan interrupciones que son ejecutadas cuando los registros alcanzan los valores límites o el valor de triguer. El programa cuenta los pulsos que se presentan en cada unidad, incrementando un registro de memoria interno de forma independiente. Si el pin que indica la dirección se encuentra en estado bajo, el conteo se realiza de forma incremental y en el caso contrario el conteo es decremental.

En la detección de una interrupción se identifica la causa y el eje correspondiente, luego se genera una trama UART para gestionar al esclavo el apagado del motor equivalente y se notifica por MQTT el evento, mediante una publicación en el topic "id_scorbot/host".

d. Aplicación UART

La aplicación UART es una tarea programada en el microcontrolador maestro, encargado de gestionar los eventos producidos en la recepción de datos provenientes del esclavo mediante el protocolo RS232. El microcontrolador esclavo debe informar de forma permanente el valor de los sensores de corriente, el estado de los motores y los errores producidos.

En la Tabla 3.15 se ilustra la trama de datos que recibe el maestro desde el esclavo, la misma consiste en un grupo de 5 bytes: los tres primeros tienen la información, el cuarto es una suma de comprobación y el último es un byte utilizado para detectar el fin de la trama.

Tabla 3.15: Trama de recepción UART del controlador maestro.

MOTOR								BITS DE VALOR								BITS DE VALOR								CHK								FIN							
1	1	R	D	D	X	X	X	1	1	X	X	X	X	X	X	1	1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	1	0	1	1	1	1	1	0
		ESTAD			ID					MSB								LSB								Verificación								Fin de trama							
192-255								192-255								192-255								0-189								190							

Elaborado por: El Investigador.

Los tres bits de payload transmiten la información del estado de cada motor, en conjunto a un valor binario de 12 bits, que corresponde al valor del sensor de corriente conectado al motor. Cada byte del payload utiliza los seis bits menos significativos para transmitir la información, los bits más significativos son rellenados con valores de ‘1’.

El valor de checksum se obtiene mediante una suma de los complementos de cada byte del payload; en estas condiciones los bytes de carga toman valores entre 192 y 255 y el byte de comprobación estará entre 0 y 189. El caracter que identifica la finalización de la trama es el correspondiente al valor decimal 190, ya que este valor no aparece dentro de los valores pertenecientes al payload o checksum.

Tabla 3.16: Información del byte de estado del motor.

1	1	R	D	D	X	X	X	Eje/Función
		ESTAD			ID			
		X	X	X	0	0	0	Base
		X	X	X	0	0	1	Shoulder
		X	X	X	0	1	0	Elbow
		X	X	X	0	1	1	Pitch
		X	X	X	1	0	0	Roll
		X	X	X	1	0	1	Griper
		X	X	X	1	1	0	Reservado
		X	X	X	1	1	1	Reservado

1	1	R	D	D	X	X	X	Eje/Función
		ESTAD			ID			
		0	0	0	X	X	X	Off
		0	0	1	X	X	X	Giro Horario
		0	1	0	X	X	X	Giro Antihorario
		0	1	1	X	X	X	Error Uart
		1	0	0	X	X	X	Reservado
		1	0	1	X	X	X	Reservado
		1	1	0	X	X	X	Reservado
		1	1	1	X	X	X	Reservado

Elaborado por: El Investigador.

El primer byte de la trama transmite la información que identifica al motor del eje e indica el estado en que se encuentra el mismo. En la Tabla 3.16 se ilustra la forma utilizada para identificar los ejes del brazo robótico y los estados que los mismos pueden presentar. Los tres bits menos significativos identifican al eje del Scorbot, utilizando los valores decimales desde el 0 al 6 para asignarlos a base, shoulder, elbow, pitch, roll y griper respectivamente. Cuando el bit R es ‘0’ los bits D informan la

dirección de giro del motor o a su vez la existencia de errores en la transmisión serial; los valores de D para cuando R en ‘1’ quedan reservados.

Los bytes complementarios del payload transmiten la información del sensor de corriente. Los ESP WROOM 32 tienen un convertidor análogo digital con una resolución de hasta 12 bits, en consecuencia, se divide el valor de la conversión para transmitirlos en dos partes, enviado 6 bits en cada byte, de acuerdo a lo indicado en la Tabla 3.17, en donde los caracteres x representan los bits del ADC. El segundo byte

Tabla 3.17: Bytes del payload perteneciente a los sensores de corriente.

BITS DE VALOR								BITS DE VALOR							
1	1	X	X	X	X	X	X	1	1	X	X	X	X	X	X
		MSB								LSB					

Elaborado por: El Investigador.

transmite los 6 bits más significativos y el tercero los menos significativos.

La aplicación UART recibe toda la trama y almacena la información en un buffer, cuando se detecta la recepción del carácter decimal ‘190’, se ejecuta la interrupción, desarmando la trama e identificando la existencia de errores. En el caso de las transmisiones correctas se identifica el eje, el estado del mismo y se retransmite la información hacia el brocker MQTT; el estado del eje se publica en el topic “id_scorbot/S/G/#” y el valor digital del sensor de corriente en el topic “id_scorbot/S/C/#” en donde # es el eje del brazo, representado por su primera letra mayúscula.

- **Firmware del Esclavo**

El firmware del esclavo, ilustrado en la Figura 3.20, es el software diseñado en lenguaje C, bajo el entorno de programación de ESP-IDF para el microcontrolador esclavo, permite transformar las señales analógicas de los sensores de corriente a señales digitales, manipular el encendido y apagado de los motores y controlar la dirección de giro de los mismos.

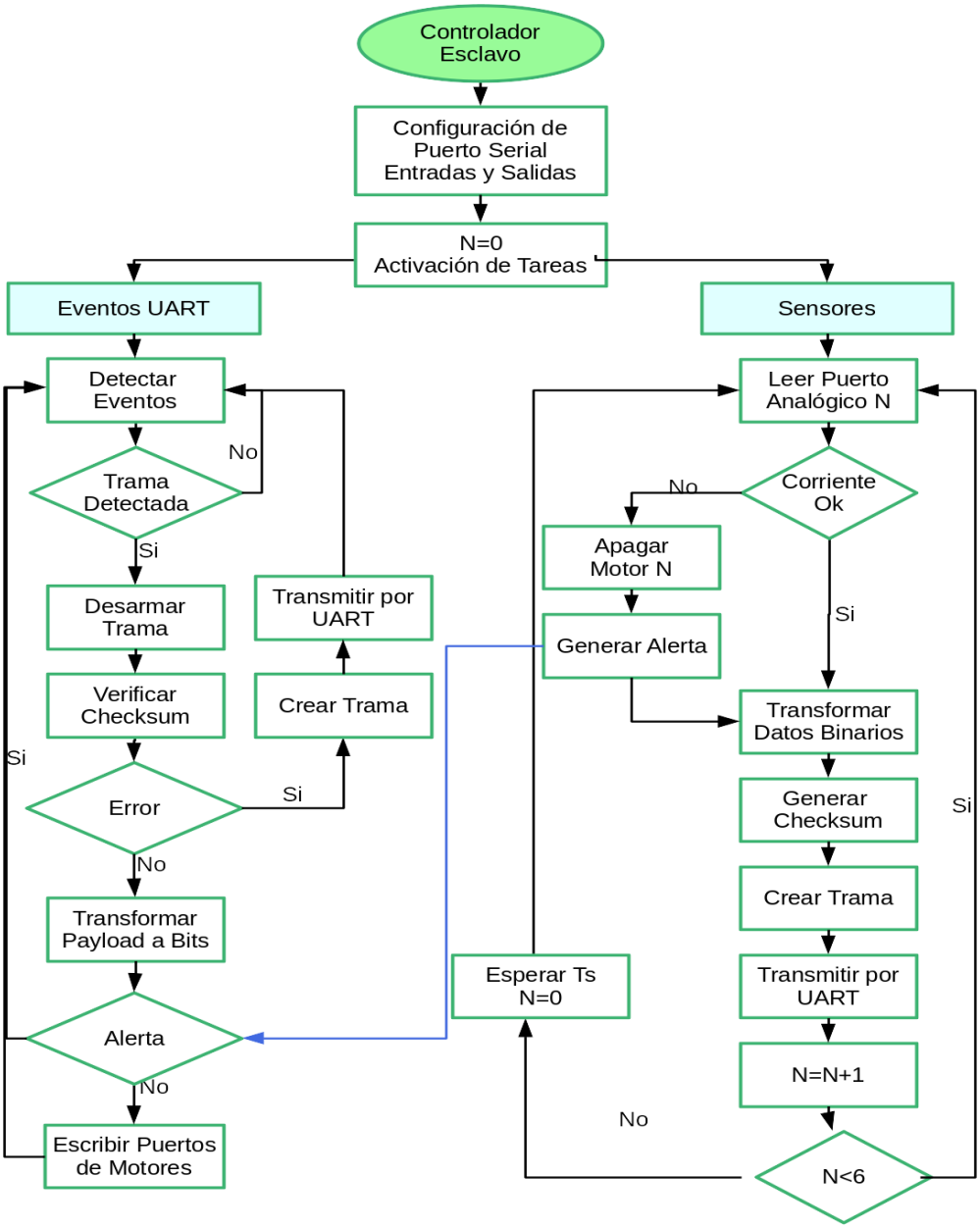


Figura 3.20: Diagrama de flujo del firmware del esclavo.

Elaborado por: El Investigador.

En el encendido del módulo esclavo se procede a configurar los pines de las entradas analógicas, pertenecientes a los sensores ACS712, y las salidas digitales para la activación de motores. Los parámetros de comunicación del puerto UART1 son los mismos utilizados en el maestro. Al finalizar la configuración de arranque del ESP WROOM 32, se inicializa la aplicación de eventos UART y sensores, que son tareas ejecutadas de forma infinita en los dos núcleos del microcontrolador.

a. Eventos UART

Eventos UART del esclavo, es una rutina de programación que detecta la transferencia de datos provenientes del módulo maestro, interpreta la información y ejecuta subrutinas que permiten encender o apagar los motores del brazo.

El microcontrolador esclavo utiliza la trama ilustrada en la Tabla 3.18 para gestionar el encendido y apagado de los seis motores conectados a sus salidas. Cada motor del robot es manipulado por dos señales digitales, en total se requiere de 12 bits para controlar todos los movimientos requeridos en el brazo. Las señales de encendido y apagado se distribuyen en los seis bits menos significativos de dos bytes, que constituyen el payload de la trama.

Tabla 3.18: Trama de recepción de datos del micro-controlador esclavo.

D1								D0								CHK								FIN							
1	1	G1	G0	R1	R0	P1	P0	1	1	E1	E0	S1	S0	B1	B0	0	X	X	X	X	X	X	X	1	0	1	1	1	1	1	0
192-255								192-255								0-127								190							

G1	G0	R1	R0	E1	E0	S1	S0	B1	B0	Estado
0	0	0	0	0	0	0	0	0	0	MotorApagado
0	1	0	1	0	1	0	1	0	1	Giro Horario
1	0	1	0	1	0	1	0	1	0	Giro Antihorario
1	1	1	1	1	1	1	1	1	1	Motor Bloqueado

Elaborado por: El Investigador.

La trama de recepción de datos del esclavo tiene una longitud de 4 bytes, los dos primeros son el payload, el tercero se utiliza enviar una suma de comprobación y el último es el indicador de finalización de trama. El valor de cheksum se calcula mediante la suma de los complementos de los bytes del payload.

Cada uno de los motores está relacionado a dos bits, y en función del valor que éstos toman se determina la dirección de giro o si el motor se apaga. En el caso de que los dos bits toman el mismo valor, el motor se apaga; si los bits son de distinto valor y el menos significativo es ‘1’ el motor se enciende en sentido horario y en el caso de que sea ‘0’ el sentido del giro es antihorario.

La aplicación UART almacena los datos de que recibe del maestro en un buffer hasta la detección del carácter de finalización de trama (‘190’). Al detectar la recepción

completa de la trama el programa separa los bytes y comprueba la existencia de errores; en el caso de que las transmisiones sean correctas se transforma el valor de los bytes a bits independientes y se envía la señal del bit al pin del motor correspondiente, siempre que la aplicación de sensores no haya detectado una colisión. Cuando existe un cambio en el estado de los motores, se notifica por UART al maestro.

b. Aplicación Sensores

La aplicación sensores, es una tarea que se ejecuta de forma infinita, cuyo objetivo es leer de forma permanente el estado de los sensores de corriente de todos los motores. Cuando existen impactos, el valor absoluto de la corriente medida supera 1 Amperio, entonces, se apaga el motor indexado al sensor y se notifica por el puerto UART el evento al maestro. En el caso de que no se detecte impactos, se notifica el estado de todos los sensores al maestro en un periodo de 500ms.

Para transformar el valor de voltaje entregado por el sensor de corriente a los amperios correspondientes, se realiza una regresión lineal a la gráfica de la relación voltaje a corriente entregado por el datasheet del ACS712ELCT, de donde se obtienen la ecuación 3.5.

Ecuación 3.5 :

$$i = 5v - 12.5$$

En donde:

i Es la corriente medida.

v Es el voltaje de salida del sensor de corriente.

3.2.7 Broker

El broker es un servicio instalado en un sistema operativo, que recibe los mensajes enviados por los clientes MQTT y los distribuye entre ellos de acuerdo a reglas de configuración, en entornos de suscripción y publicación. En el sistema diseñado para la manipulación del Scorbot mediante el protocolo MQTT, el broker es el software encargado de intercambiar la información entre el host y el módulo de control MQTT, mediante direcciones en las que se publican las variables que intervienen en el sistema. El servicio del broker es instalado en un sistema operativo Linux; al tratarse de un

sistema de software libre, no se tienen costos de licencias y permiten modificar el código fuente de acuerdo a las necesidades del proyecto.

En la Tabla 3.19 se detalla las características que soportan los servidores MQTT más reconocidos por los desarrolladores de software. La mayoría de ellos soportan los protocolos requeridos para la implementación del sistema con el brazo robótico, los que son: una transmisión con una calidad de servicio QoS de 2, soporte de autenticación y cifrado para la comunicación (SSL).

Tabla 3.19: Características de los servidores MQTT más reconocidos.

Servidor	QoS 0	QoS 1	QoS 2	SSL	auth	Free
Apache ActiveMQ	Si	Si	Si	Si	Si	Si
Mongoose	Si	?	?	?	?	Si
Mosquito	Si	Si	Si	Si	Si	Si
Mosca	Si	Si	No	?	Si	Si
RabbitMQ	Si	Si	No	Si	Si	Si
Emq X	Si	Si	Si	Si	Si	Si
HiveMQ	Si	Si	Si	Si	Si	Si

Elaborado por: El Investigador

El broker seleccionado es el Emq-x, este servidor permite realizar la configuración de reglas y aplicaciones mediante una interfaz web, también soporta la integración de aplicaciones con websockets y funciones de encriptación de datos transmitidos por medio del protocolo TLS V1.2.

En la figura 3.21 se observa la instalación de Emqx en sistemas Linux basados en Red Hat, es simple, solo requiere la configuración de los repositorios mediante los siguientes comandos:

```
#yum install -y yum-utils device-mapper-persistent-data lvm2
#yum-config-manager--add-repohttps://repos.emqx.io/emqxce/redhat/centos/7/emqx-ce.repo
```

Figura 3.21: Comandos para la Configuración de los repositorios

Luego en la figura 3.22, se obtiene el programa mediante la ejecución de:

```
#yum install emqx
```

Figura 3.22: Comando para la instalación de Emqx.

El archivo de configuración de Emqx está en el directorio /etc/emqx/emqx.conf, en donde la se debe establecer las siguientes opciones para habilitar la comunicación mediante el protocolo TLS, como se observa e la figura 3.23.

```
cluster.proto_dist = inet_tls
listener.ssl.external.tls_versions = tlsv1.2,tlsv1.1,tlsv1
listener.ssl.external = 8883
listener.ssl.external.keyfile = /etc/emqx/certs/key.pem
listener.ssl.external.certfile = /etc/emqx/certs/cert.pem
listener.ssl.external.cacertfile = /etc/emqx/certs/cacert.pem
~
```

Figura 3.23: Modificaciones del archivo de configuración del bróker MQTT

Con las configuraciones indicadas se habilita la encriptación de datos para la transferencia de información entre los clientes y el brocker. En la tarjeta electrónica se utiliza el protocolo TLS V1.2 ya que es el más seguro. Los directorios de keyfile, certfile y cacertfile son las direcciones de ubicación de los ficheros con las llaves y certificados firmados, utilizados para cifrar la información a ser transmitida.

El servicio de EMQX es administrado mediante los siguientes comandos:

#emqx start → Inicializa el servicio

#emqx stop → Detiene el servicio

3.2.8 Host de Control

El host de control es un equipo externo, que tiene la capacidad de ejecutar las funciones de un cliente MQTT, mediante el protocolo TLS V1.2. Este equipo terminal puede ser programado en un micro-controlador, computador de placa reducida o computador personal. En el presente proyecto se utiliza el mismo equipo de cómputo que aloja el servidor del brocker MQTT, ya que reduce los costos operativos y se garantiza óptimas características de procesamiento.

El software de programación requerido, es un conjunto de aplicaciones que permiten

desarrollar una interfaz de usuario para manipular las funciones del módulo controlador MQTT. De forma general un host de control requiere de forma mínima un lenguaje de programación o intérpretes de código que manipulen clientes MQTT, sin embargo, el desarrollo de aplicaciones completas, debería incluir un entorno gráfico y la gestión de almacenamiento de datos.

Tabla 3.20: Entornos de desarrollo para la interfaz HMI del controlador MQTT.

Entorno	Cliente MQTT	Interfaz Web	GUI	Tipo
Phython/Bash Glade	Si	Si	Si	Código escrito Free
Netbeans	Si	No	Si	Código escrito Free
Labview	Si	Si	Si	Código gráfico Licencia
Nodered	Si	Si	Si	Código híbrido Free
Visual Studio	Si	No	Si	Código escrito Licencia
Matlab	Si	Si	Si	Código escrito Licencia
OpenLab	Si	No	Si	Código Gráfico Free

Elaborado por: El Investigador.

El diseño elaborado puede ser controlador por cualquier lenguaje de programación que ejecute las funciones de un cliente MQTT, enviando las solicitudes de posicionamiento requeridas mediante publicaciones y leyendo el estado del brazo mediante suscripciones. En la Tabla 3.20 se presentan distintos entornos de programación que permiten elaborar interfaces de control para el brazo robótico. En el presente proyecto se desarrolla una interfaz Demo para indicar uno de los múltiples modos de programación que permite elaborar la tarjeta de desarrollo MQTT.

El entorno de programación seleccionado para el desarrollo de la Interfaz Demo es Nodered, una herramienta de programación web que utiliza el servidor Node Js y un

lenguaje de programación híbrido basado en Json. El lenguaje es seleccionado debido a que presenta ventajas en la facilidad de programación, permite crear interfaces web, es de licencia libre e integra funciones de gestión de base de datos. Matlab, Labview y Visual Studio son descartados debido a que implican elevados costos del pago de licencias.

Nodered debe ser ejecutado de forma infinita como una aplicación, para habilitar el servicio, se debe ejecutar en una terminal, con el comando node-red. Para acceder al entorno de programación se utiliza un navegador web, ingresando a la dirección: http://IP_SERVER:1880. Para habilitar la programación desde un terminal distinto al servidor se debe abrir el puerto del firewall para aplicaciones del protocolo TCP y UPD.

La programación en Nodered se realiza a través de una interfaz web, utilizando diagramas de bloques que representan distintas funciones. En el presente proyecto se añadieron tres librerías adicionales a las que vienen por defecto en el entorno de programación: Dashboard, que es una librería que permite diseñar entornos gráficos para interfaces HMI; Mysql, que es una función que permite administrar bases de datos Mysql o Mariadb y Led, que es un complemento de dashboard para simular la presencia de leds.

Las principales funciones de Nodered, utilizadas en la interfaz Demo HMI se ilustran en la Figura 3.24, en la imagen se tiene un bloque de suscripción que permite recibir datos de cualquier topic por MQTT de acuerdo a la configuración; también se indica un bloque de publicación que envía los datos hacia el brocker. La sección función permite ejecutar código en javascript, mientras que del lado izquierdo se observan los componentes más importantes del dashboard: botón, led y texto. Los bloques de funciones pueden ser de entrada, salida o entrada y salida y cada uno de ellos recibe datos por medio de estructuras de lenguaje javascript Json.

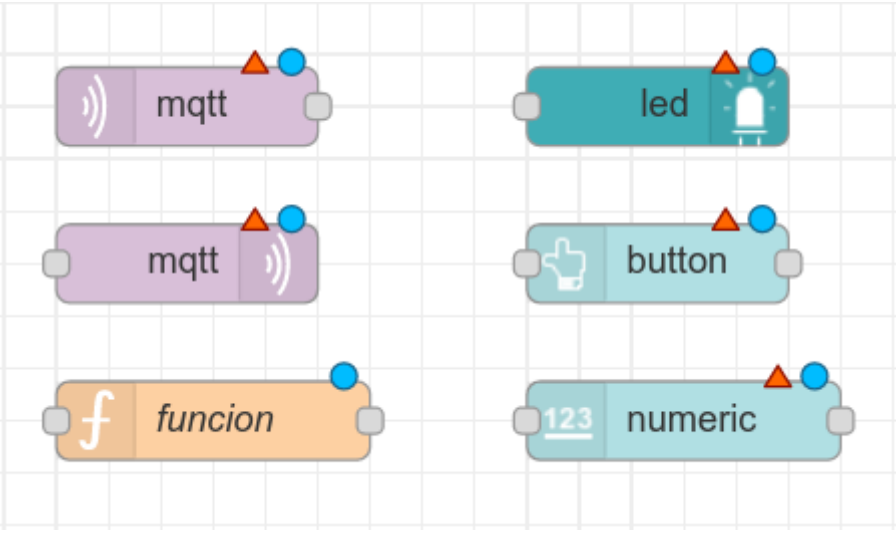


Figura 3.24: Funciones principales de node red utilizadas en la interfaz HMI.

Elaborado por: El Investigador.

La interfaz de demostración Humano Maquina (HMI) diseñada en el presente proyecto, es una interfaz de acceso web a la que se ingresa mediante un navegador, utilizando la dirección: http://IP_SERVER:1880/ui/. La misma está dividida en tres secciones: control de desplazamiento, control de funciones y control de posicionamiento.

La sección de control de desplazamiento, ilustrada en la Figura 3.25, permite al usuario posicionar de forma libre cada uno de los ejes. Esta sección de la interfaz, está compuesta por indicadores luminosos que simulan diodos led y botones que ejecutan movimientos en el brazo robótico. Existen 6 luces indicadoras de estado, una asignada a cada uno de los ejes, los diodos simulados toman distintos colores de acuerdo al estado de del motor del eje correspondiente en el brazo; así: cuando el motor del eje gira en sentido horario, la luz será de color verde, azúl cuando el giro es antihorario, rojo cuando el eje entra en estado de error por detección de impacto y gris cuando el motor del eje está apagado.

Figura 3.25: Sección de desplazamiento de la interfaz HMI Demo.

Elaborado por: El Investigador.

Cada uno de los 6 ejes del brazo robótico puede ser controlado por medio de dos botones para manipular el giro de los motores, uno para cada sentido (horario y antihorario). Al mantener presionado cualquiera de los botones, se enciende el motor asignado al eje, generando un movimiento continuo en el brazo; el movimiento se detiene cuando se detecta un impacto o el usuario libera la presión del botón. Los botones están programados para responder a la detección de eventos, cuando los botones son presionados envían publicaciones al broker solicitando el encendido del eje asignado, en un sentido determinado. Cuando el usuario libera la presión del mouse retirando el pulso, el sistema envía una publicación al broker en el topic del eje solicitando el apagado del motor.

El panel de posicionamiento, indicado en la Figura 3.26 está compuesto por siete interfaces de entradas numéricas, seis asignados a cada uno de los ejes y el séptimo utilizado para establecer la velocidad de desplazamiento; además de un botón de stop utilizado como paro de emergencia. Cada una de entradas numéricas está compuesta por una sección de ingreso por teclado y dos botones, los mismos controlan el incremento o decremento por unidades de las posiciones.

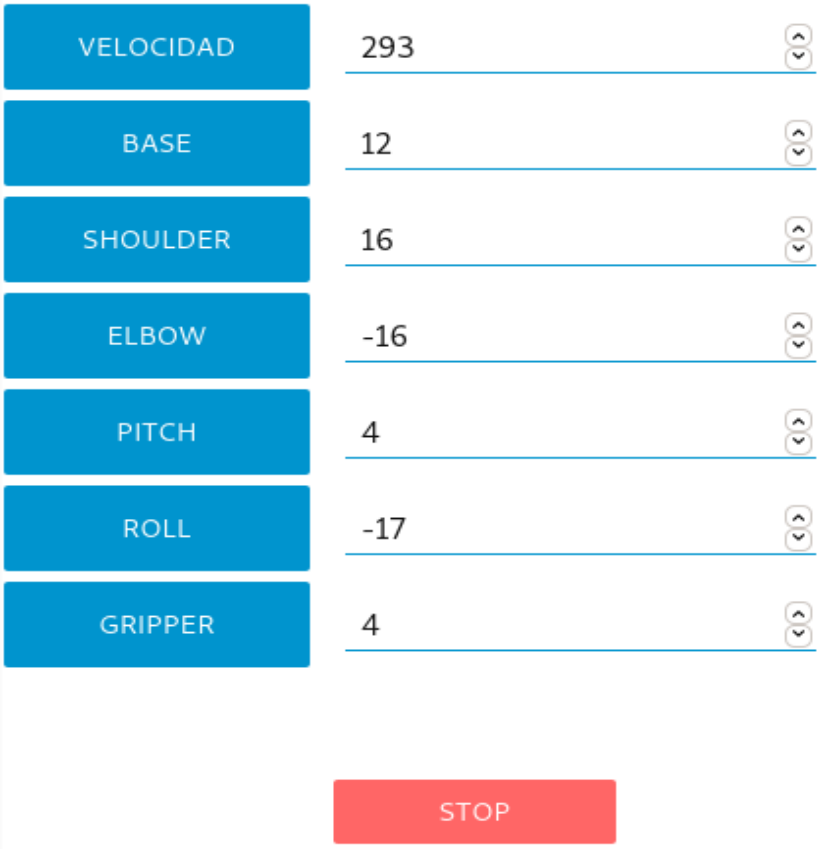


Figura 3.26: Sección de posicionamiento de la interfaz Demo HMI.

Elaborado por: El Investigador.

La posición de los ejes se establece en base al número de pulsos, tomado como referencia la posición de 0 para home, cuando el final de carrera es activado. En cada entrada numérica se establece la cantidad de pulsos que debe recorrer el eje asignado, moviendo el motor de acuerdo a los algoritmos de posicionamiento programados en la tarjeta de control.

Las entradas detectan cambios de estado en el valor numérico de cada eje; cuando se identifica un cambio, ya sea producido por los botones de la interfaz o por el teclado, se envía el dato mediante una publicación MQTT a un topic que identifica al eje o la velocidad en el brocker.

El botón de stop, publica en el brocker un valor de 0 para el topic de la velocidad, provocando que se cancele cualquier movimiento que se esté ejecutando y llevando a todos los ejes a un estado de libre movilidad.

La tercera sección corresponde al panel de funciones indicada en la Figura 3.27. En este panel se puede programar rutinas de movimiento, utilizando memorias almacenadas en una base de datos.

Figura 3.27: Panel de funciones de la interfaz Demo HMI.

Elaborado por: El Investigador

El panel de funciones tiene tres áreas: administración de memorias, ejecución de rutinas e indicador de estado. La administración de memorias está conformada por un selector numérico y tres botones, que permiten seleccionar un número de memoria y almacenarla en una base de datos. El botón guardar almacena la información del panel de posicionamiento en un registro identificado por el número de memoria, dentro de la tabla Memorias de una base de datos Mysql.

El botón borrar, elimina los registros del número de memoria indicado en el panel, finalmente, el botón mover envía las publicaciones requeridas al broker MQTT para ubicar cada uno de los ejes en la posición almacenada. En el indicador numérico al detectar un cambio de estado se realiza una consulta a la base de datos, preguntando por los valores almacenados, utilizando como índice el número de memoria. Si se detectan datos almacenados, se muestran en el panel de posicionamiento; en caso de no existir, todos los valores del panel serán ubicados a 0.

La sección de ejecución de rutinas tiene dos botones y un indicador para la referencia de posición final. El botón ejecutar, carga todos los datos almacenados en la base a un script Json, que se encarga de ir posicionando de forma cíclica cada una de las posiciones almacenadas en la base, pasando desde la memoria 1 hasta llegar a la memoria final; al terminar la rutina de memorias, repite el proceso de forma infinita. El botón detener, interrumpe el proceso cíclico y apaga el brazo robótico.

El resultado final del funcionamiento del sistema se refleja en la figura 3.28 en el lado izquierdo de la imagen se observa la interfaz gráfica que controla los movimientos del brazo robótico y al lado derecho se ilustra los resultados de una cámara que capta los movimientos del brazo. En la imagen de la cámara se observa que la relación de tamaños entre el brazo robótico y el controlador MQTT es ergonómico.

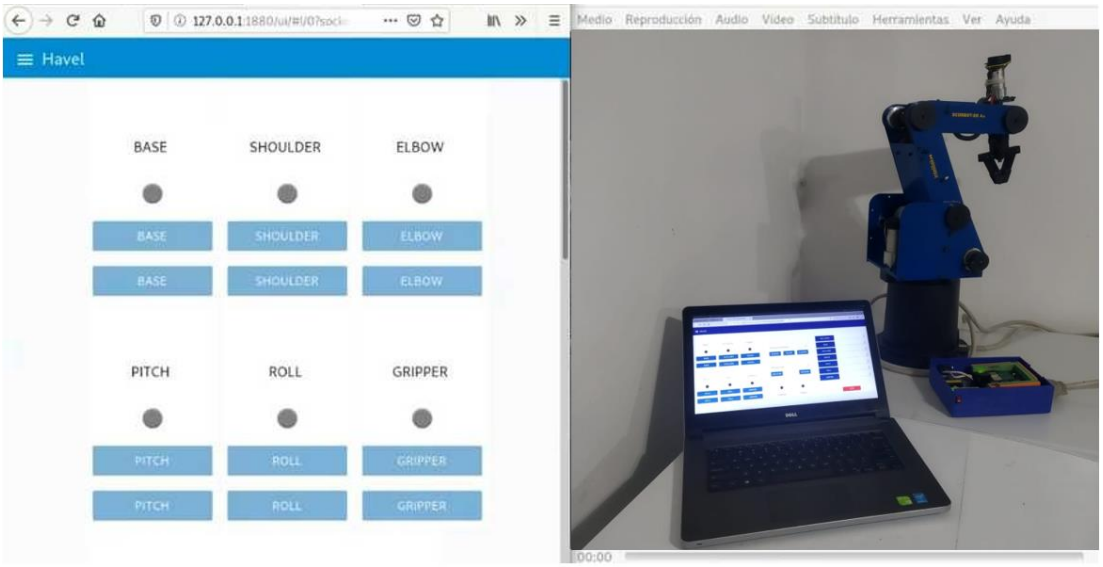


Figura 3.28: Funcionamiento del sistema de control del Scorbot por MQTT.

Elaborado por: El Investigador.

En la Figura 3.29, se observa todos los elementos requeridos para la operación del brazo robótico mediante el protocolo MQTT. El módulo electrónico, encapsulado por una carcasa de ABS de color azul, se conecta de forma directa a la red eléctrica de 110 VAC, y al brazo robótico mediante el terminal D50 del cable de color blanco. La conexión entre el computador y el módulo electrónico se realiza mediante una red inalámbrica utilizando el protocolo 802.11 n.

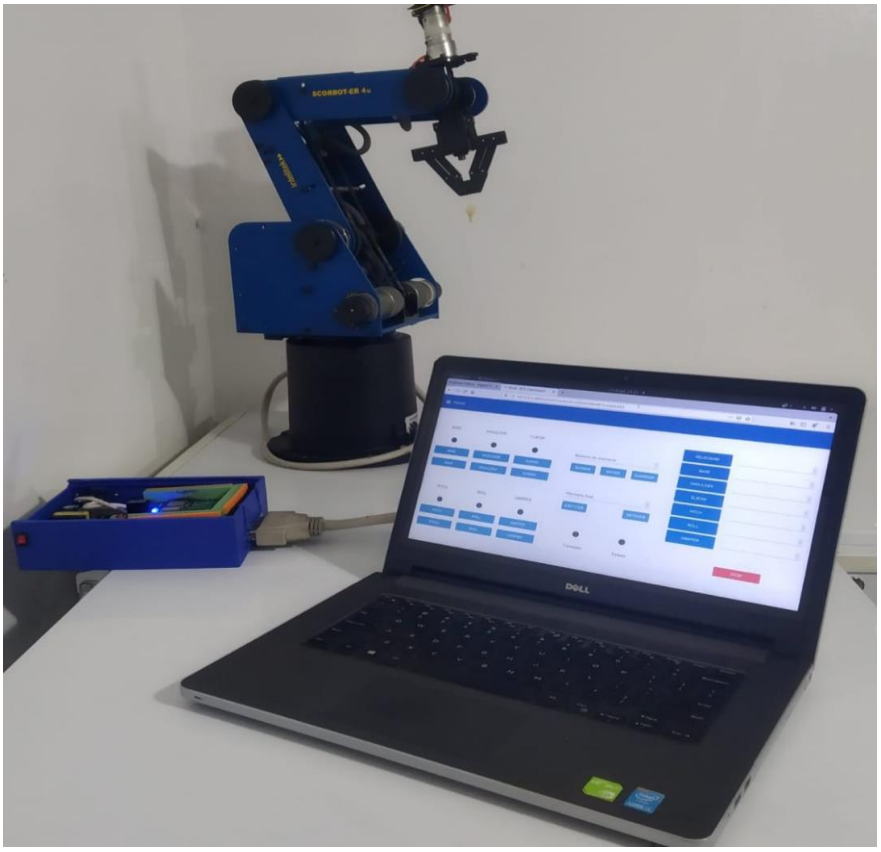


Figura 3.29: Manipulación del Scorbot mediante MQTT.

Elaborado por: El Investigador.

En el computador se observa la interfaz gráfica demostrativa diseñada en un servidor NodeJs mediante la herramienta Node-Red. El aplicativo de control diseñado permite ejecutar movimientos, desplazamientos y posicionamientos en los ejes del brazo.

3.3 ANALISIS DE PRESUPUESTO

El análisis del presupuesto del presente proyecto constituye un estudio económico del costo total que implica implementar de forma unitaria el módulo MQTT para el control del Scorbot ER-4U.

El detalle del presupuesto económico se analiza en la Tabla 3.21, donde se detalla los precios de los materiales necesarios para la implementación de la tarjeta electrónica del módulo MQTT. El costo total de los materiales es de 330.17 dólares.

Tabla 3.21: Detalle del costo de materiales del módulo MQTT.

Item	Descripción	Unidad	Cantidad	Precio Unitario	Total
1	Regulador de voltaje	c/u	5	0.21	1.05
2	Resistencias	c/u	41	0.02	0.82
3	Capacitores Ceramicos	c/u	14	0.02	0.28
4	Codesadores electroliticos	c/u	14	0.03	0.42
5	Bloques Terminales	c/u	60	0.18	10.08
6	Sensores de Corriente	c/u	6	1.18	7.08
7	Leds	c/u	3	0.02	0.06
8	Regulador lineal	c/u	2	2.37	4.73
9	Diodos	c/u	24	0.03	0.72
10	Interruptores	c/u	4	0.01	0.04
11	ESP-WROOM 32	c/u	2	3.50	7.00
12	Conector DB50	c/u	1	1.25	1.25
13	Baquelita de circuito impreso a 6 capas color verde, PCB módulo MQTT	c/u	1	12.12	12.12
14	Baquelita de circuito impreso a 2 capas color azúl, PCB interfaz angular	c/u	1	5.29	5.29
15	Fuente de poder	c/u	1	36.23	36.23
16	Ventilador	c/u	1	7	7.00
17	Impresión 3D de la carcasa	hora	12	3	36.00
				Subtotal	130.17
				Diseño del Prototipo	100.00
				Otros	100.00
				TOTAL ($)	**330.17**

Elaborado por: El Investigador.

3.4 RESULTADOS

La controladora elaborada mediante hardware y software libre para la manipulación del brazo robótico mediante el protocolo MQTT es una herramienta que permite el desarrollo de aplicaciones robóticas industriales y educativas a un bajo costo. El diseño final del dispositivo es el indicado en la figura 3.30; en la imagen se observa la distribución de los elementos que componen el módulo de control. En la parte superior se ubica la fuente regulada de tensión a 24Vdc y 6 Amperios. La tarjeta fabricada se ubica en la parte inferior de la fuente; se utiliza un ventilador debajo del circuito impreso para disipar el calor generado por los reguladores de tensión y drives que transfieren las altas temperaturas al cobre del PCB.

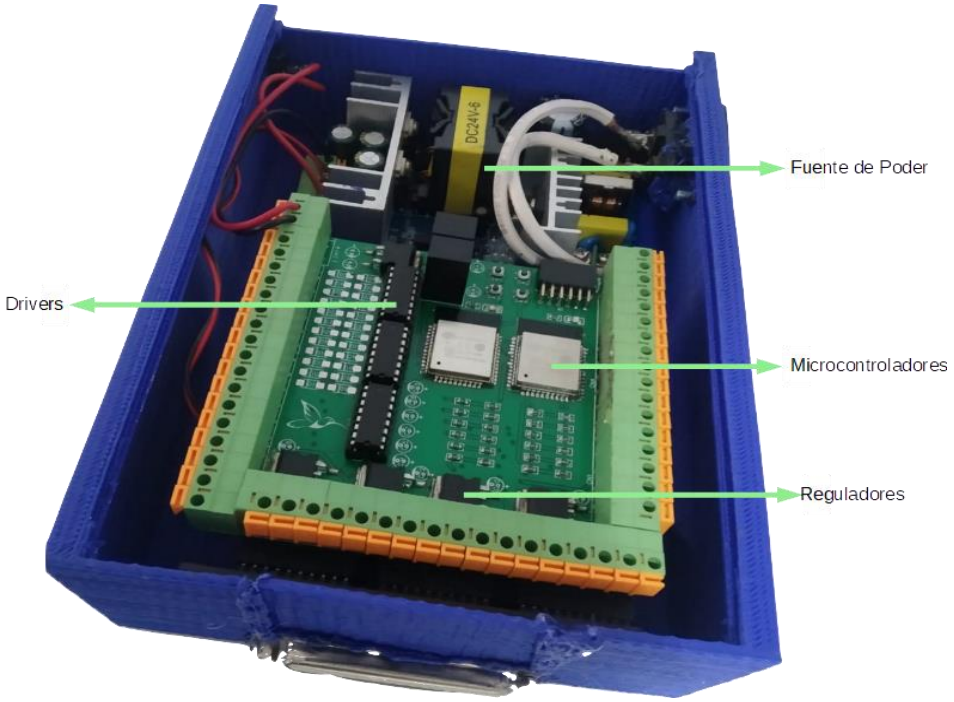


Figura 3.30: Controlador MQTT del brazo robótico.

Elaborado por: El Investigador.

Los terminales de conexión del controlador MQTT se ilustran en la Figura 3.31 El dispositivo elaborado tiene un terminal de conexión hembra para la alimentación eléctrica externa a una fuente de 110VAC; en el caso de utilizar baterías, los terminales de las mismas se deben conectar en paralelo a los cables de color rojo (+) y negro (GND) indicados en la imagen. El control de encendido y apagado del equipo se realiza

mediante un switch ubicado en la parte externa al costado superior izquierdo de la carcasa.

El controlador MQTT no está limitado a manipular los periféricos del brazo robótico ER-4U, este equipo tiene la capacidad de controlar la velocidad de motores de corriente continua y la posición de motores con cajas de reducción que operan a un voltaje de hasta 24VDC con un consume de corriente de 1A. De forma adicional tiene la factibilidad de leer señales digitales de 5Vdc provenientes de sensores y la posibilidad de conectar módulos de comunicación externos que utilicen protocolos RS232 o SPI. Las señales de salida pueden utilizarse para múltiples aplicaciones como activación de electroválvulas, señales analógicas para controladores lógicos programables, luces led indicadoras entre otras.

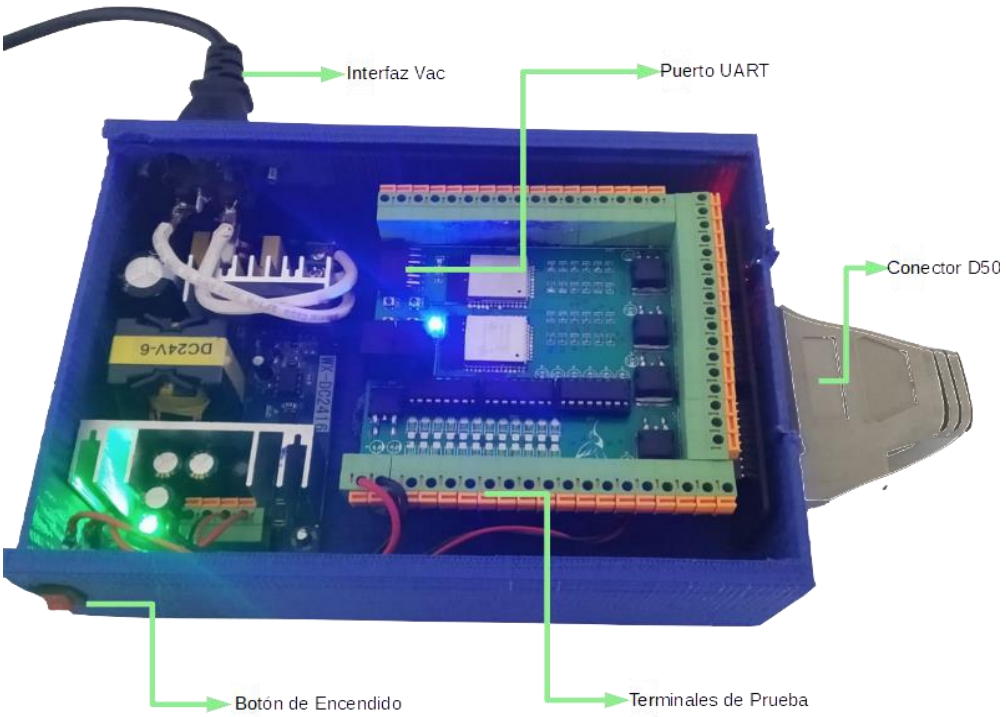


Figura 3.31: Puerto de conexión del controlador MQTT.

Elaborado por: El Investigador.

El controlador MQTT se conecta al cable del brazo robótico mediante el puerto D50 ubicado en el costado derecho de la carcasa. De forma interna se tiene acceso a los terminales de prueba, que son conexiones paralelas a todos los terminales de las señales enviadas y recibidas del brazo robótico. Finalmente, de forma similar en la

parte interna del controlador se tiene un terminal de conexión los puertos UART que permiten actualizar el firmware de los microcontroladores mediante un convertidor serial TTL a USB.

En los resultados de la investigación se especifica los parámetros técnicos que permiten evaluar el funcionamiento del módulo MQTT como una tarjeta de control del brazo robótico Scorbot ER-4U. Para esta finalidad se han realizado pruebas de funcionamiento, desplazando los ejes del brazo robótico desde la interfaz gráfica y evaluando los errores y tiempos de respuesta obtenidos en las peticiones realizadas.

En las pruebas de funcionamiento se realizaron 400 peticiones de desplazamiento para cada uno de los ejes y el cambio de velocidad durante un periodo de una semana. La calidad de respuesta del sistema se evalúa mediante mediciones de tiempo de cada una de las peticiones realizadas; para ello se programó en Nodered una rutina de javascript que calcula el tiempo de respuesta del sistema, midiendo el lapso en milisegundos desde que el usuario presiona el botón de la pantalla hasta que se recibe la notificación de encendido de los motores por MQTT desde la tarjeta electrónica. La información obtenida fue almacenada en una base de datos mysql para ser analizada y procesada; en la Tabla 3.24 se presenta la tabulación de datos de los tiempos de respuesta en milisegundos, medidos para cada una de las peticiones realizadas, diferenciados por el tipo de movimiento solicitado.

Tabla 3.22: Tiempos de respuesta de las peticiones de encendido y apagado de los motores.

Tiempo (ms)	Base	Shoulder	Elbow	Pitch	Roll	Gripper	Velocidad
1-100	139	87	1	0	0	0	273
101-200	170	181	231	157	91	0	114
201-300	67	82	112	142	188	210	13
301-400	22	41	37	73	83	112	0
401-500	2	9	18	22	27	53	0
501-600	0	0	0	5	11	24	0
601-700	0	0	0	0	0	1	0
701-800	0	0	0	0	0	0	0

801-900	0	0	0	0	0	0	0
901-1000	0	0	1	1	0	0	0
Total	400	400	400	400	400	400	400

Elaborado por: El Investigador.

En la Figura 3.32 se observa la gráfica de los tiempos de respuesta que le toma a cada uno de los ejes procesar una petición de movimiento. El tiempo máximo de respuesta del sistema está entre los 500 y 600 milisegundos y el mínimo es de 100; la moda está en el intervalo de 100 a 200 milisegundos, determinando que más del 50% de peticiones se encuentran con un tiempo de procesamiento adecuado.

Los tiempos del eje de base son inferiores a los de los ejes complementarios debido a que estas peticiones son las primeras en procesarse dentro del microcontrolador. De forma consecuente el algoritmo da prioridad a shoulder, elbow, picth, roll y al final a gripper. Si se observa la diferencia entre el eje prioritario y gripper se determina que el margen de los tiempos modales es de 100 ms, teniendo un tiempo pico de respuesta de 600 ms y un pico modal de 300.

Los procesos ejecutados para el eje de base tienen un tiempo de respuesta máximo de 500ms y un tiempo mínimo inferior a los 100ms, teniendo la mayor concentración de tiempos de respuesta en un rango de 100 a 200ms. En el caso de shoulder las estadísticas de tiempos mínimos y máximos son similares, evidenciando un desplazamiento en la concentración mayoritaria al rango comprendido entre los 100 y 300 ms, e identificando que se reduce el porcentaje de peticiones ejecutadas en menos de 100ms, con referencia al eje de base. Para el eje de pitch los tiempos de respuesta mínimos ya son superiores a 100ms, con una variación que llega hasta los 600ms; teniendo la mayor concentración de tiempos de respuesta en el rango comprendido entre los 100 a 300ms.

En los tiempos de respuesta del eje de roll se identifica una caída estadística de concentración de tiempos inferiores a 200ms, identificando que la mayor congregación se encuentra en un rango comprendido entre 200 y 500ms; también se observa la tendencia de crecimiento en los tiempos máximos a valores superiores a 600ms. Para

el eje de elbow el rango de tiempos de respuesta se extiende desde los 100 a 600ms, teniendo la mayor concentración de datos en el rango de 200 a 400ms. Finalmente el eje de griper, es el proceso que requiere de mayores tiempos de ejecución, por ende, sus tiempos de respuesta son superiores; los tiempos mínimos superan los 200ms con una concentración mayoritaria en el rango de 200 a 400 ms y un incremento en el número de peticiones que superan los 600 ms, en comparación a los ejes anteriores.

TIEMPOS DE RESPUESTA DEL ENCENDIDO DE MOTORES

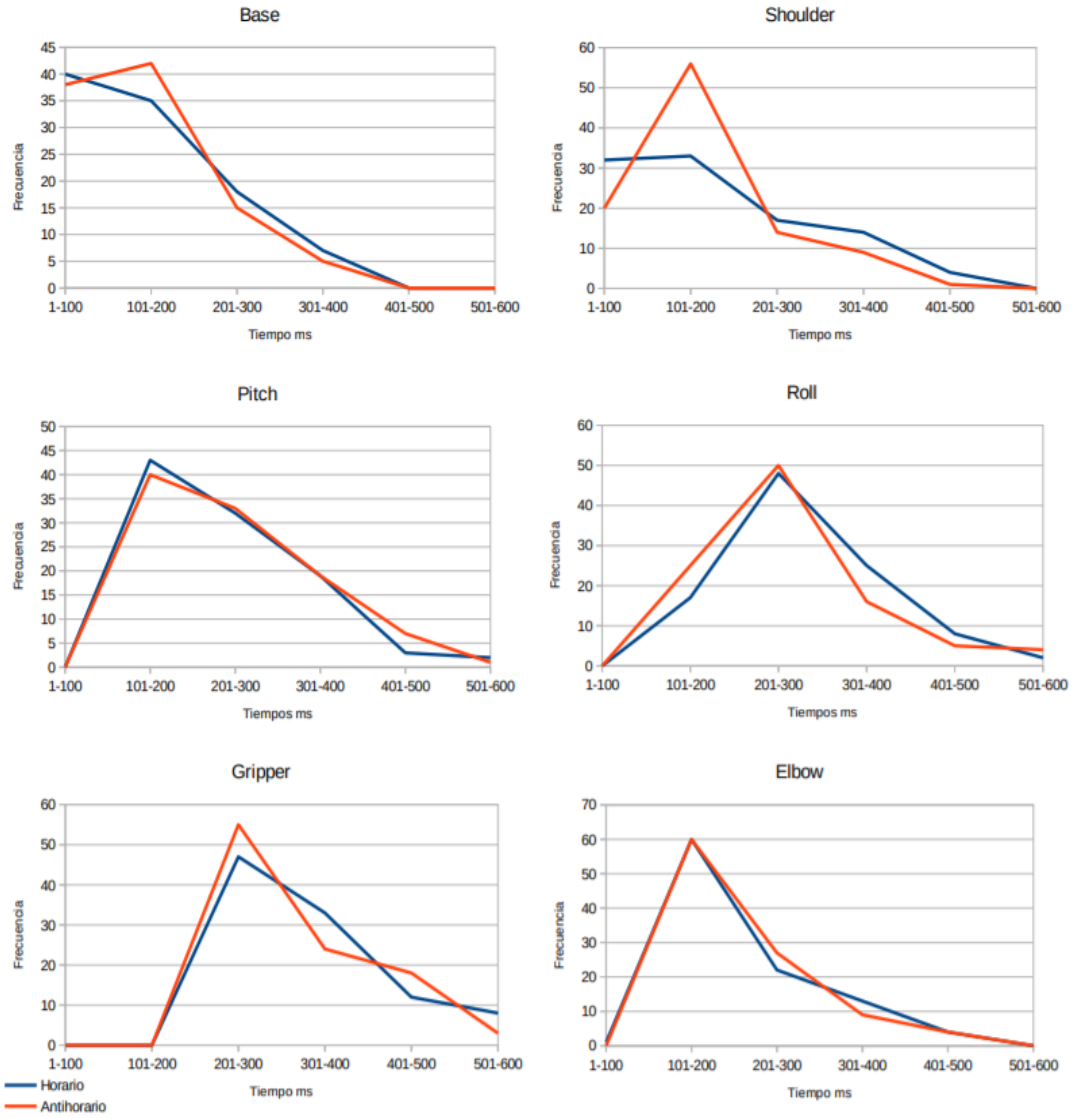


Figura 3.32: Tiempos de respuesta del encendido de los motores del brazo robótico.

Elaborado por: El Investigador.

Los tiempos de respuesta entre las peticiones de encendido de los motores en sentido horario (líneas azules), en comparación al sentido antihorario (líneas rojas), tienen un comportamiento similar en cada uno de los ejes; no existen evidencias representativas y las funciones tienen una variación muy cercana.

El proceso de datos del cambio de velocidad tiene una prioridad superior a la activación de los motores y el algoritmo requiere de menores recursos del microcontrolador; esto con la finalidad de ejecutar un paro de emergencia de forma inmediata. El gráfico de la Figura 3.33 representa los tiempos de respuesta de las peticiones de cambio de velocidad, en la imagen se determina que más del 70% de las peticiones tienen una respuesta inferior a los 100[ms] y solo un 2.5% está en el tiempo pico de 300 [ms].

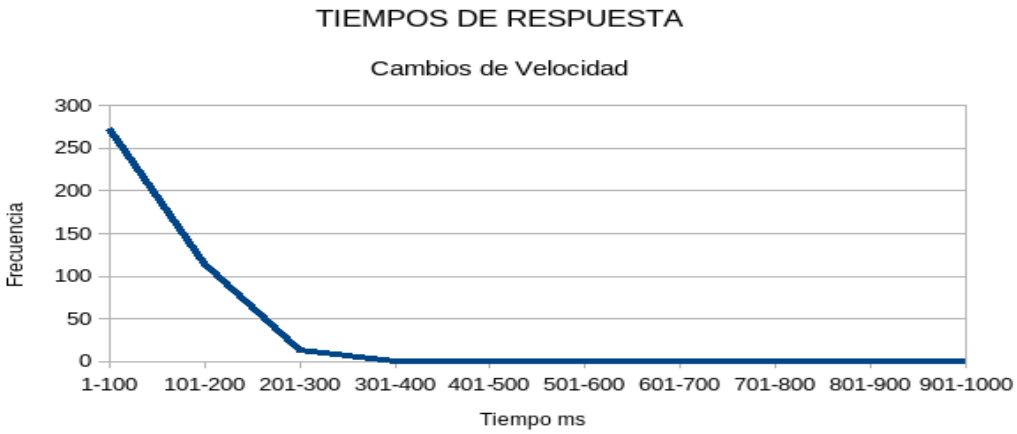


Figura 3.33: Tiempos de respuesta del cambio de velocidad de los motores.

Elaborado por: El Investigador.

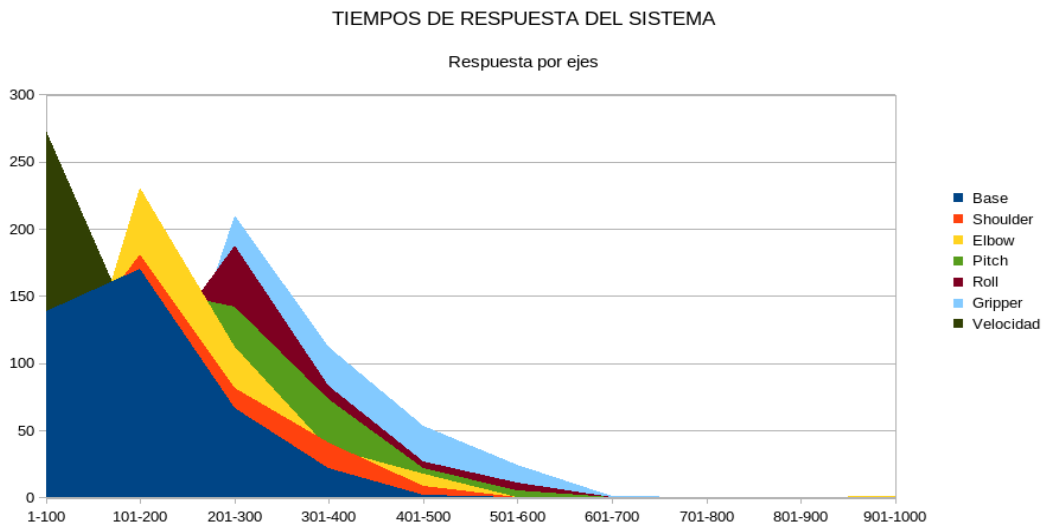


Figura 3.34: Tiempo de respuesta por ejes.

Elaborado por: El Investigador.

En la Figura 3.33 y Figura 3.34, se observa la representación general de todas las pruebas realizadas. En la Figura 3.34 se analiza que la todos los ejes mantienen su área de trabajo entre los 0 y 600[ms] con una concentración mayoritaria en los tiempos inferiores a 200[ms]. En la Figura 3.35 se observa el resumen general de los tiempos de respuesta de las 2400 pruebas ejecutadas, en donde la moda destaca un tiempo de respuesta inferior a 200[ms] y un tiempo pico de 1[s], provocado por error de la red de datos.

En las 2400 pruebas ejecutadas, la comunicación serial entre el maestro y el esclavo, no entregó un solo error, determinando que en esa sección el protocolo de comunicación RS232 es ideal. En la red Wifi solo se determinó un error que provocó que la respuesta del sistema a una de las peticiones sea lenta, (983[ms]) sin embargo la petición fue ejecutada.

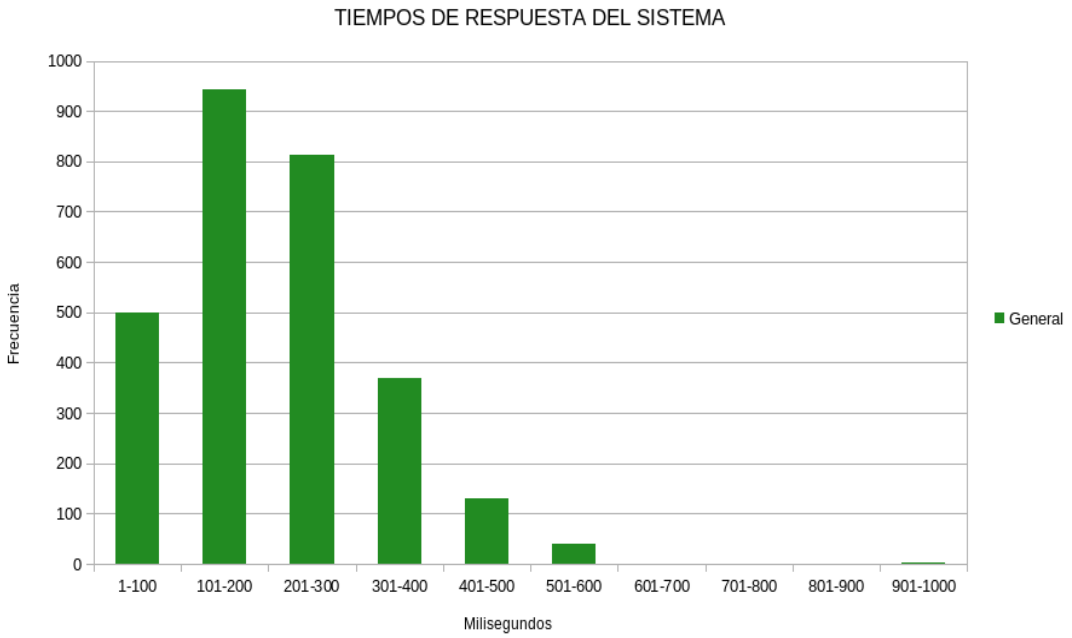


Figura 3.35: Resumen general de la estadística de los tiempos de respuesta.

Elaborado por: El Investigador.

El trabajo realizado logró cumplir los objetivos, elaborando un controlador electrónico que utiliza el protocolo MQTT para el control del brazo robótico o periféricos de características similares, teniendo la factibilidad de reemplazar el controlador USB del Scorbot ER-4U. Las aplicaciones que se ejecutan con el nuevo controlador no están limitadas a un único lenguaje de programación, permitiendo el desarrollo de programas en interfaces de Python, PHP, Javascript, Java, C, entre otras. El equipo diseñado tiene la facilidad de ser alimentado por baterías, generando una portabilidad del mismo.

CAPÍTULO IV

CONCLUSIONES Y RECOMENDACIONES

4.1 CONCLUSIONES

✓ El brazo robótico del Scorbot ER-4U está integrado por componentes electrónicos que se clasifican en sensores y actuadores, constituyendo elementos de control que permiten a un sistema programado manipular los movimientos del eje del brazo robótico, en función de 5 ejes de rotación y una pinza de elemento final. Las características de las conexiones físicas y eléctricas integradas en el brazo robótico permiten el desarrollo de tarjetas electrónicas adaptativas para manipular las señales de entrada y salida del brazo en una interfaz directa con los con sus componentes internos; permitiendo el desarrollo de aplicaciones integradas en sistemas de hardware y software libre

✓ El módulo controlador compuesto por un núcleo doble del microcontrolador ESP32 WROM32 resulta eficiente debido a que entrega al sistema una conexión directa entre periféricos mediante el protocolo MQTT, utilizando la interfaz inalámbrica del estándar 802.11 n. Los tiempos de respuesta medidos en la petición de movimientos a través del protocolo MQTT, al enviar un paquete desde el cliente y esperar la réplica resultan inferiores a los 600ms; una concentración mayor al 70% de peticiones tiene lapsos inferiores a 300ms.

✓ La comunicación MQTT utilizada como protocolo de transferencia de datos entre la controladora y el equipo terminal con una calidad de servicio de QOS de 2, garantizan la transmisión de la información entre los nodos, transmitiendo las tramas de peticiones de movimiento y señales de los sensores de forma bidireccional, con un porcentaje de error calculado en las pruebas de funcionamiento inferior al 0.025%; a un nivel de señal de -75 dBm. La comunicación entre los nodos se realiza sin inconvenientes de ancho de banda.

✓ La elaboración del módulo electrónico de control basado en el protocolo MQTT es desarrollado utilizando hardware y software libre. El equipo tiene la libertad de ser implementado en aplicaciones de ambientes educativos e industriales, así como la

factibilidad de modificar su diseño para su evolución o adaptación a nuevos protocolos de comunicación. El dispositivo diseñado permite el control del brazo robótico desde cualquier interfaz o lenguaje de programación que soporte el uso de clientes MQTT.

4.2 RECOMENDACIONES

A las personas que hagan uso del presente proyecto de investigación, se sugiere lo siguiente:

✓ Utilizar un ventilador completamente abierto y que tenga contacto con toda la PCB debido a que toda esta es un disipador de calor y se requiere que el mismo se mantenga frio, y así poder garantizar el funcionamiento adecuado de la tarjeta electrónica.

✓ Utilizar señales de alimentación de 5V con etapas de adaptación de señales que atenúen los 5V que entrega el brazo robótico a 3.3V, debido que si se alimenta con 3.3V no tiene un buen rendimiento.

✓ Todas las capas se rellenen con un plano a tierra de cobre para que estas capas se utilicen como disipador de calor y el calor generado en los reguladores de voltaje y en los puentes h se dispersen por esos sectores.

✓ Esta tesis podría ser un puente para realizar más experimentos robóticos, ya que permite desarrollar una interfaz en cualquier entorno donde el usuario interprete código MQTT para la manipulación de las funciones del módulo controlador MQTT.

Bibliografía

[1] P. Vinayak y S. Surech, «Performance Analysis of SCORBOT ER 4u Robot Arm,» International Journal of Materials Science and Engineering, vol. 2, nº 1, 2014.

[2] D. Bermejo, «Robótica para el Seguimiento de Líneas,» Universidad Politecnica de Catatuña, Barcelona, 2016.

[3] J. Aroca, A. Perez y R. Rodriguez, «Generation and Control of Basic Geometric Trajectories fora Robot Manipulator Using CompactRIOD,» HindawiJournal of Robotics, vol. 2017, nº 7508787, 2017.

[4] J. Aroca, R. Rodriguez, V. Ashmyakov y R. Sagaro, «Kinematic Model of the Scorbot 4PC ManipulatorImplemented in Matlab's Guide,» Contemporary Engineering Sciences, vol. 11, nº 4, 2018.

[5] F. Salazar, J. Buele, H. Velastegui, A. Soria, E. Tubon y G. Orejuela, «Teleoperation and Remote Monitoring of a ScorBot ER-4U Robotic Arm in an Academic Environment,» International Journal of Innovative Technology and Exploring Engineering, vol. 8, 2019.

[6] S. D. FRANCISCO SILVA, F. Silva y S. Deltiode, «Automatización Robótica de Proceso,» Deloitte, Febrero 2017. [En línea]. Available: https://www2.deloitte.com/content/dam/Deloitte/ec/Documents/deloitte-analytics/Estudios/Automatizacion_Rob%C3%B3tica_Procesos.pdf.

[7] C. Gallegos y E. S. Barahona, «Navegación Autónoma Basada en Maniobras Bajo Estimación de Posturas Humanas para un Robot Omnidireccional Kuka Youbot", Universidad Técnica de Ambato,» Universidad Tecnica de Ambato, Ingenieria en Electronica y Comunicaciones, Ambato, 2019.

[8] S. M. GONZALES, «A study on the usefulness of educationalrobotics from educators' perpective,» Revista de Pedagogía, vol. 32, nº 90, 2011.

[9] S. N. NAVARRO, «Smart Robots y Otras Máquinas Inteligentes en Nuestra Vida Cotidiana,» Revista CESCO de Derecho de Consumo, No 20, 2016. [En línea]. Available: vailable: http://www.revista.uclm.es/index.php/cesco.

[10] K. Apuu, Robotics, India: International Publishing House Pvt. Ltd, 2007.

[11] M. Kener, Australian Center for Robotic Vision, [En línea]. Available: http://roboticsinaustralia.com.au/what-is-a-robot/.

[12] D. Hunt, «Understanding Robotics,» Elsevier, 28 Agosto 1990. [En línea]. Available: https://www.elsevier.com/books/understanding-robotics/hunt/978-0-12-361775-0.

[13] G. Krutarth, K. Harsh, J. Susmit y P. Vikas, «Motion controlled robotic arm,» International Journal of Electronics andCommunication Engineering, Academia, vol. 2, nº 5, p. 86, 2013.

[14] V. Gayatri, «Technology Subjects Support Service,» 2018. [En línea]. Available: http://www.t4.ie/Technology/Resources%20-%20Options/Control_Technology/Introduction%20to%20Robotics.pdf.

[15] INTELITEK, «SCORBOT-ER 4u Educational Robot,» [En línea]. Available: http://www.intelitek.com/pdf/35-1005-8600_HW_ER4u-K.pdf.

[16] INTELITEK, «Scorbot ER-4U User Manual,» [En línea]. Available: http://www.intelitekdownloads.com/Manuals/Robotics/ER-4u/ER_4u_B.pdf. .

[17] J. S. Prabhakar, «Hand writing using scorbot arm,» Northern Illinois University, Department of Electrical Engineering, Illinois, 2014.

[18] «Controller-USB,» Intelitek, Manchester, 2007.

[19] INTELITEK, «Scorebase V7", SCORBASEVERSION 7AND HIGHERFORSCORBOT-ER 4USCORBOT-ER 2UER-400 AGV MOBILE ROBOTUSER MANUALCatalog No. 100342–Rev. I.,» [En línea]. Available: http://www.intelitekdownloads.com/Manuals/Robotics/ER-4u/Scorbase_USB_I.pdf. .

[20] J. G. GÓMEZ, «CIRCUITOS Y SISTEMAS DIGITALES, Departamento de Electronica y ComunicacionesUniversidad Pontifica de Salamanca en Madrid.,» 2002. [En línea]. Available: http://www.iearobotics.com/personal/juan/docencia/apuntes-ssdd-0.3.7.pdf. .

[21] Carlos, H. Pueyo y C. Marco, Circuitos electricos, España: Alfaomega, 2009.

[22] H y S. Altamirano, Sistema de Control de Acceso por Reconocimiento de Iris para el Ingreso dePersonal a la Empresa Electroservicios Querubín de la Ciudad de Puyo, Ambato: Universidad Técnica de Ambato, 2018.

[23] B. Mirchell, «lifewire,» 26 Septiembre 2019. [En línea]. Available: https://www.lifewire.com/introduction-to-ethernet-817550. [Último acceso: 25 Enero 2020].

[24] W. Buchanan, «Computer Busses,» [En línea]. Available: http://www.soc.napier.ac.uk/~bill/pdf/P_ch16.pdf. [Último acceso: 07 Abril 2020].

[25] O. E. Morales, «Departamento de Ingeniería Electrónica. Escuela de Ingeniería, Universidad de las Américas Puebla.,» 10 Diciembre 2003. [En línea]. Available: http://catarina.udlap.mx/u_dl_a/tales/documentos/lem/morales_h_oe/capitulo3.pdf. [Último acceso: 07 Abril 2020].

[26] L. Xiaoguang y W. Zhang, «IEEE Conference on Industrial Electronics and Applications. doi:10.1109/iciea.2009.5138237 ,» 2009. [En línea]. Available: https://sci-hub.tw/10.1109/ICIEA.2009.5138237. [Último acceso: 08 Febrero 2020].

[27] «Watelectronics,» 29 Julio 2019. [En línea]. Available: https://www.watelectronics.com/different-types-of-wireless-communication-technologies/. [Último acceso: 05 Marzo 2020].

[28] J. Salazar, «TechPedia,» Erasmus, [En línea]. Available: https://upcommons.upc.edu/bitstream/handle/2117/100918/LM01_R_ES.pdf. [Último acceso: 05 Marzo 2020].

[29] C. A. Hervas, «Universidad Nacional de la Plata,» Junio 2018. [En línea]. Available: http://sedici.unlp.edu.ar/bitstream/handle/10915/69435/Documento_completo.pdf-PDFA.pdf?sequence=1&isAllowed=y. [Último acceso: 07 Abril 2020].

[30] F. Prieto, «Universidad de Sevilla,» [En línea]. Available: http://bibing.us.es/proyectos/abreproy/11372/fichero/Memoria%252F05+-+Protocolo+HTTP.pdf. [Último acceso: 08 Abril 2020].

[31] J. N. Suarez , «Universidad Politecnica Salesiana,» 2012. [En línea]. Available: https://dspace.ups.edu.ec/bitstream/123456789/2146/16/UPS-CT002403.pdf. [Último acceso: 08 Abril 2020].

[32] F. Mecafenix, «La enciclopedia de la ingeniería,» 03 Mayo 2017. [En línea]. Available: https://www.ingmecafenix.com/electronica/puente-h-control-motores/. [Último acceso: 05 Marzo 2020].

[33] A. S. Salazar, «Universidad Tecnica de Ambato,» Febrero 2017. [En línea]. Available:

https://pdfs.semanticscholar.org/3eeb/cb724b1370b439e37f2ebd342b4c5c05bc98.pdf. [Último acceso: 09 Abril 2020].

[34] A. R. Bruno, «Microelectronicash,» 2019. [En línea]. Available: http://www.microelectronicash.com/downloads/ESP32_MANUAL.pdf. [Último acceso: 02 05 2020].

[35] J. A. Pichucho, «Repositorio Digital Escuela Politecnica Nacional,» Marzo 2017. [En línea]. Available: https://bibdigital.epn.edu.ec/bitstream/15000/1307/1/CD-0616.pdf. [Último acceso: 09 05 2020].

[36] K. E. Pekka, «Centria University of Applied Sciences,» Mayo 2016. [En línea]. Available: https://pdfs.semanticscholar.org/bb18/53c23fd556a859062cd5db5c6de1c9182620.pdf. [Último acceso: 20 05 2020].

[37] D. X. Gudiño, «Universidad Tecnologica Israel,» 2014. [En línea]. Available: http://157.100.241.244/bitstream/47000/915/1/UISRAEL%20-%20EC%20-%20SIS%20-%20378.242%20-%20118.pdf. [Último acceso: 23 05 2020].

[38] F. E. Ave, «Facultad de Educación, Universidad del Sur de Florida,» Centro de Tecnología Educativa de Florida, 2013. [En línea]. Available: https://fcit.usf.edu/network/chap2/chap2.htm. [Último acceso: 09 02 2020].

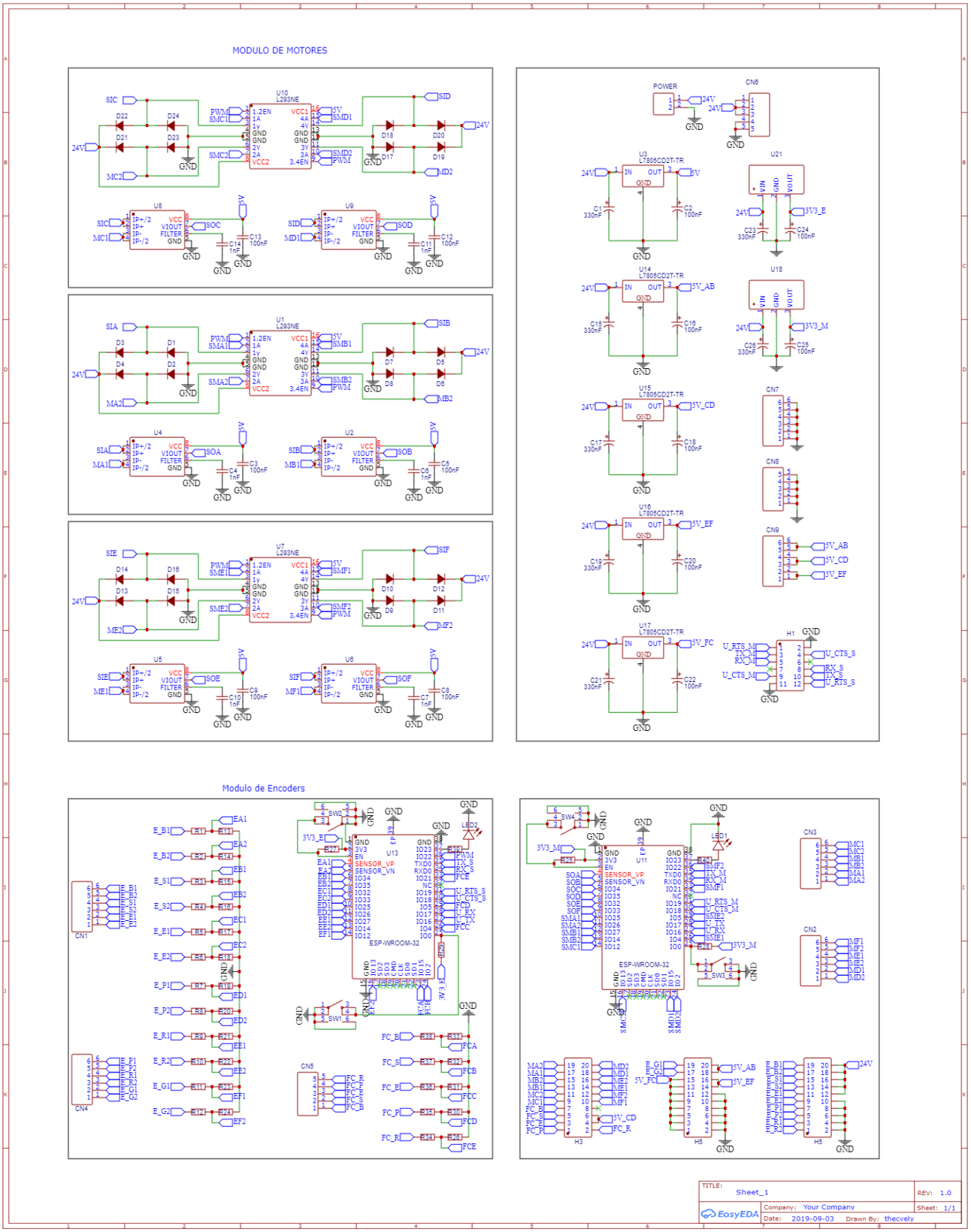
[39] J. Sonnenberg, «Raveon Technologies Corp,» 2018. [En línea]. Available: https://www.raveon.com/wp-content/uploads/2019/01/AN236SerialComm.pdf. [Último acceso: 23 Enero 2020].

[40] «Allegro MicroSystems,» 2010. [En línea]. Available: https://datasheet.octopart.com/ACS712ELCTR-05B-T-Allegro-MicroSystems-LLC-datasheet-8557243.pdf.

ANEXOS

ANEXO A

Diagrama del circuito electrónico del módulo MQTT



ANEXO B

EL SIGUIENTE CÓDIGO FUENTE CORRESPONDE AL SOFTWARE DEL MICROCONTROLADOR MAESTRO QUE PERMITE MANIPULAR LOS ENCODERS Y FINALES DE CARRERA GENERANDO LAS RUTINAS DE POSICIONAMIENTO

```
//Importación de librerías requeridas y definición de constantes globales
#include <stdio.h>
#include <stdint.h>
#include <stddef.h>
#include <string.h>
#include "esp_wifi.h"
#include "esp_system.h"
#include "nvs_flash.h"
#include "esp_event.h"
#include "tcpip_adapter.h"
#include "protocol_examples_common.h"
#include "driver/gpio.h"
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/semphr.h"
#include "freertos/queue.h"
#include <string.h>
#include "driver/uart.h"
#include "driver/periph_ctrl.h"
#include "lwip/sockets.h"
#include "lwip/dns.h"
#include "lwip/netdb.h"
#include "esp_log.h"
#include "mqtt_client.h"
#include "esp_tls.h"
#include "driver/ledc.h"
#define PWM 22
#define LED_GREEN 23
#include "freertos/portmacro.h"
#include "driver/pcnt.h"
#include "esp_attr.h"
#include "esp_log.h"
// Definición de constantes para la app de contador de pulsos
#define LB_H 7000
#define LB_L -7000
#define LS_H 7000
#define LS_L -7000
#define LE_H 7000
#define LE_L -7000
#define LP_H 7000
#define LP_L -7000
#define LR_H 7000
#define LR_L -7000
#define LG_H 7000
#define LG_L -7000
#define UNIT_BASE PCNT_UNIT_0
#defineUNIT_SHOULDERPCNT_UNIT
#define UNIT_ELBOW PCNT_UNIT_2
#define UNIT_PITCH PCNT_UNIT_3
#define UNIT_ROLL PCNT_UNIT_4
#define UNIT_GRIPPER PCNT_UNIT_5
staticesp_mqtt_client_handle_tclient_mqtt
static int16_t PB_M=10000, PS_M=0, PE_M=0, PP_M=0, PR_M=0, PG_M=0, VE_M=0, PB_B=-20000, PS_B=-2000,
```

```
PE_B=-20000, PP_B=-20000, PR_B=-20000, PG_B=-20000;
//Función que permite el intercambio de datos entre tareas (tasks)
xQueueHandle pcnt_evt_queue;
pcnt_isr_handle_t intr_service = NULL;
// Definición de la estructura la el intercambio de información (Se transfiere unidad y estado del contador)
typedef struct {
int unit;
uint32_t status;
}pcnt_evt_t;
//Función para la gestión de interrupciones del contador de pulsos
Static void IRAM_ATTR interruptService(void *arg){
uint32_t intr_status=PCNT.int_st.val;
pcnt_evt_t evt;
portBASE_TYPE HPTaskAwoken = pdFALSE;
for(int i=0;i<PCNT_UNIT_MAX;i ++){
if(intr_status & (BIT(i)) ){
evt.unit =i;
evt.status=PCNT.status_unit[i].val;
PCNT.int_clr.val=BIT(i);
xQueueSendFromISR(pcnt_evt_queue, &evt, &HPTaskAwoken);
if (HPTaskAwoken == pdTRUE) {
portYIELD_FROM_ISR();
}}}}
//Función para la inicialización y configuración de los contadores de pulsos
static void initContadores(void){
//Creación de Unidades
pcnt_config_t pcnt_base={
.pulse_gpio_num=36,
.ctrl_gpio_num=39,
.channel=PCNT_CHANNEL_0,
.unit=UNIT_BASE,
.pos_mode=PCNT_COUNT_INC,
.neg_mode=PCNT_COUNT_DIS,
.lctrl_mode=PCNT_MODE_REVERSE,
.hctrl_mode=PCNT_MODE_KEEP,
.counter_h_lim=LB_H,
.counter_l_lim=LB_L,};
pcnt_config_t pcnt_shoulder={
.pulse_gpio_num=34,
.ctrl_gpio_num=35,
.channel=PCNT_CHANNEL_0,
.unit=UNIT_SHOULDER,
.pos_mode=PCNT_COUNT_INC,
.neg_mode=PCNT_COUNT_DIS,
.lctrl_mode=PCNT_MODE_REVERSE,
.hctrl_mode=PCNT_MODE_KEEP,
 .counter_h_lim=LS_H,
.counter_l_lim=LS_L,
};
pcnt_config_t pcnt_elbow={
.pulse_gpio_num=32,
.ctrl_gpio_num=33,
.channel=PCNT_CHANNEL_0,
.unit=UNIT_ELBOW,
.pos_mode=PCNT_COUNT_INC,
.neg_mode=PCNT_COUNT_DIS,
.lctrl_mode=PCNT_MODE_REVERSE,
.hctrl_mode=PCNT_MODE_KEEP,
.counter_h_lim=LE_H,
.counter_l_lim=LE_L,};
```

```
pcnt_config_t pcnt_pitch={
.pulse_gpio_num=25,
.ctrl_gpio_num=26,
.channel=PCNT_CHANNEL_0,
.unit=UNIT_PITCH,
.pos_mode=PCNT_COUNT_INC,
.neg_mode=PCNT_COUNT_DIS,
.lctrl_mode=PCNT_MODE_REVERSE,
.hctrl_mode=PCNT_MODE_KEEP,
.counter_h_lim=LP_H,
.counter_l_lim=LP_L, };
pcnt_config_t pcnt_roll={
.pulse_gpio_num=27,
.ctrl_gpio_num=14,
.channel=PCNT_CHANNEL_0,
.unit=UNIT_ROLL,
.pos_mode=PCNT_COUNT_INC,
.neg_mode=PCNT_COUNT_DIS,
.lctrl_mode=PCNT_MODE_REVERSE,
 .hctrl_mode=PCNT_MODE_KEEP,
 .counter_h_lim=LR_H,
 .counter_l_lim=LR_L, };
pcnt_config_t pcnt_gripper={
.pulse_gpio_num=12,
.ctrl_gpio_num=13,
.channel=PCNT_CHANNEL_0,
.unit=UNIT_GRIPPER,
pos_mode=PCNT_COUNT_INC,
.neg_mode=PCNT_COUNT_DIS,
.lctrl_mode=PCNT_MODE_REVERSE,
.hctrl_mode=PCNT_MODE_KEEP,
.counter_h_lim=LG_H,
.counter_l_lim=LG_L, };

//Inicialización de Unidades

pcnt_unit_config(&pcnt_base);
pcnt_unit_config(&pcnt_shoulder);
pcnt_unit_config(&pcnt_elbow);
pcnt_unit_config(&pcnt_pitch);
pcnt_unit_config(&pcnt_roll);
pcnt_unit_config(&pcnt_gripper);

//Configuración de Filtros Pasa Bajos

pcnt_set_filter_value(UNIT_BASE,
100);//8KHz
pcnt_set_filter_value(UNIT_SHOULDER
, 100);//8KHz
pcnt_set_filter_value(UNIT_ELBOW,
100);//8KHz
pcnt_set_filter_value(UNIT_PITCH,
100);//8KHz
pcnt_set_filter_value(UNIT_ROLL,
100);//8KHz
pcnt_set_filter_value(UNIT_GRIPPER,
100);//KHz
pcnt_filter_enable(UNIT_BASE);
pcnt_filter_enable(UNIT_SHOULDER);
pcnt_filter_enable(UNIT_ELBOW);
pcnt_filter_enable(UNIT_PITCH);
pcnt_filter_enable(UNIT_ROLL);
pcnt_filter_enable(UNIT_GRIPPER);

//Configuración de eventos Posicionadores

pcnt_set_event_value(UNIT_BASE,
PCNT_EVT_THRES_1, PB_M);
pcnt_set_event_value(UNIT_SHOULDE
R, PCNT_EVT_THRES_1, PS_M);
pcnt_set_event_value(UNIT_ELBOW,
PCNT_EVT_THRES_1, PE_M);
```

```
pcnt_set_event_value(UNIT_PITCH,
PCNT_EVT_THRES_1, PP_M);
pcnt_set_event_value(UNIT_ROLL,
PCNT_EVT_THRES_1, PR_M);
pcnt_set_event_value(UNIT_GRIPPER,
PCNT_EVT_THRES_1, PG_M);

pcnt_event_enable(UNIT_BASE,
PCNT_EVT_THRES_1);
pcnt_event_enable(UNIT_SHOULDER,
PCNT_EVT_THRES_1);
pcnt_event_enable(UNIT_ELBOW,
PCNT_EVT_THRES_1);
pcnt_event_enable(UNIT_PITCH,
PCNT_EVT_THRES_1);
pcnt_event_enable(UNIT_ROLL,
PCNT_EVT_THRES_1);
pcnt_event_enable(UNIT_GRIPPER,
PCNT_EVT_THRES_1);

pcnt_event_enable(UNIT_BASE,
PCNT_EVT_H_LIM);
pcnt_event_enable(UNIT_BASE,
PCNT_EVT_L_LIM);
pcnt_event_enable(UNIT_SHOULDER,
PCNT_EVT_H_LIM);
pcnt_event_enable(UNIT_SHOULDER,
PCNT_EVT_L_LIM);
pcnt_event_enable(UNIT_ELBOW,
PCNT_EVT_H_LIM);
pcnt_event_enable(UNIT_ELBOW,
PCNT_EVT_L_LIM);
pcnt_event_enable(UNIT_PITCH,
PCNT_EVT_H_LIM);
pcnt_event_enable(UNIT_PITCH,
PCNT_EVT_L_LIM);
pcnt_event_enable(UNIT_ROLL,
PCNT_EVT_H_LIM);
pcnt_event_enable(UNIT_ROLL,
PCNT_EVT_L_LIM);
pcnt_event_enable(UNIT_GRIPPER,
PCNT_EVT_H_LIM);
pcnt_event_enable(UNIT_GRIPPER,
PCNT_EVT_L_LIM);
pcnt_counter_pause(UNIT_BASE);
pcnt_counter_clear(UNIT_BASE);
pcnt_counter_pause(UNIT_SHOULDER)
;
pcnt_counter_clear(UNIT_SHOULDER);
pcnt_counter_pause(UNIT_ELBOW);
pcnt_counter_clear(UNIT_ELBOW);
pcnt_counter_pause(UNIT_PITCH);
pcnt_counter_clear(UNIT_PITCH);
pcnt_counter_pause(UNIT_ROLL);
pcnt_counter_clear(UNIT_ROLL);
pcnt_counter_pause(UNIT_GRIPPER);
pcnt_counter_clear(UNIT_GRIPPER);
pcnt_isr_register(interruptService, NULL,
0, intr_service);///REVISAR WARNIG
pcnt_intr_enable(UNIT_BASE);
pcnt_counter_resume(UNIT_BASE);
pcnt_intr_enable(UNIT_SHOULDER);
pcnt_counter_resume(UNIT_SHOULDE
R);
pcnt_intr_enable(UNIT_ELBOW);
pcnt_counter_resume(UNIT_ELBOW);
pcnt_intr_enable(UNIT_PITCH);
pcnt_counter_resume(UNIT_PITCH);
pcnt_intr_enable(UNIT_ROLL);
pcnt_counter_resume(UNIT_ROLL);
pcnt_intr_enable(UNIT_GRIPPER);
pcnt_counter_resume(UNIT_GRIPPER);
```

```
}

/Funcion PWM
//Configuración de la aplicación PWM, la salida entrega una señal de 100Hz con una resolución de 10 bits

static void ledc_init(void){

 // Confuguración de valores y pines de la señal pwm

ledc_timer_config_t ledc_timer;
ledc_timer.speed_mode = LEDC_HIGH_SPEED_MODE;
ledc_timer.timer_num = LEDC_TIMER_1;
ledc_timer.duty_resolution = LEDC_TIMER_10_BIT;
ledc_timer.freq_hz = 100;

// Aplicación de las configuraciones al canal PWM

ledc_timer.clk_cfg = LEDC_AUTO_CLK;
ledc_timer_config(&ledc_timer);

 // Prepare and then apply the LEDC PWM channel configuration
ledc_channel_config_t ledc_channel;
ledc_channel.speed_mode = LEDC_HIGH_SPEED_MODE;
ledc_channel.channel = LEDC_CHANNEL_1;
ledc_channel.timer_sel=LEDC_TIMER_;
ledc_channel.intr_type=LEDC_INTR_DISABLE;
ledc_channel.gpio_num   = PWM;
ledc_channel.duty = 90; // set duty at about 25%
ledc_channel.hpoint     = 0;
ledc_channel_config(&ledc_channel);}

//Comunicación RS232
//Configuración y ejecución de tareas del módulo UART

static const char *TAG_U = "EVENTO SERIAL";
static QueueHandle_t uart_esp_queue;
void init_serial_esp(){
const uart_config_t serial_config = {
baud_rate = 115200,
.data_bits = UART_DATA_8_BITS,
parity = UART_PARITY_DISABLE,
stop_bits = UART_STOP_BITS_1,
flow_ctrl=UART_HW_FLOWCTRL_DISABLE,};
ESP_ERROR_CHECK(uart_param_config(CONFIG_UART_ESP, &serial_config));
ESP_ERROR_CHECK(uart_set_pin(CONFIG_UART_ESP,CONFIG_UART_TX,CONFIG_UART_RX,CONFIG_UART_RTS,CONFIG_UART_CTS));
ESP_ERROR_CHECK(uart_driver_instal(CONFIG_UAT_ESP,CONFIG_UART_SIZE,CONFIG_UART_SIZE,20,uart_esp_queue,0));
uart_enable_pattern_det_intr(CONFIG_U
```

```
ART_ESP, CONFIG_UART_PT, CONFIG_PT_N, 10000, 10, 10); uart_pattern_queue_reset(CONFIG_UART_ESP, 20);}

//Funciones de comunicacion rs232
void controlUart(uint8_t control[4]){

//Declaración de buffer de recepción
char data_mqtt[40];

//Declaración de variable para el control de estado del motor
char motor_state[40];

//Declaración de variables para chequeo de errores
uint8_t C0=255-control[0];
uint8_t C1=255-control[1];
uint8_t C2=255-control[2];

//Cálculo de cheqsum
uint8_t suma=C0+C1+C2;
uint8_t chksm=control[3]-suma;

//Verificación de errores de transmisión en UART
if(control[3]<190&&chksm==0){

//Transformación de la trama para encapsulación de 12 bits
control[1]=control[1]&63; //MSB

//ADC más la información cheqssum en dos bytes.
control[2]=control[2]&63; //LSB
control[0], control[1], control[2]);

//Se identifica el tipo de dato recibido para publicar el evento
uint16_t sensor_value=control[1]<<6;
sensor_value=sensor_value|control[2];
sensor_value);
sprintf(data_mqtt, "%u", sensor_value);

//EN CASO DE QUE NO TRANSMITA DATOS VERIFICAR AQUI

sprintf(motor_state, "%u", (56&control[0])>>3);
uint8_t eje=control[0]&7;

//En el topic del eje correspondiente
switch (eje){
case 0:
if(sensor_value!=0)
esp_mqtt_client_publish(client_mqtt, "havel/S/C/B", data_mqtt, 0, 2, 0);
else
esp_mqtt_client_publish(client_mqtt, "havel/S/G/B", motor_state, 0, 2, 0);
break;

case 1:
if(sensor_value!=0)
esp_mqtt_client_publish(client_mqtt, "havel/S/C/S", data_mqtt, 0, 2, 0);
else
esp_mqtt_client_publish(client_mqtt, "havel/S/G/S", motor_state, 0, 2, 0);
break;
```

```
case 2:
if(sensor_value!=0)
esp_mqtt_client_publish(client_mqtt,
"havel/S/C/E", data_mqtt, 0, 2, 0);
else
esp_mqtt_client_publish(client_mqtt,
"havel/S/G/E", motor_state, 0, 2, 0);
break;

case 3:
if(sensor_value!=0)
esp_mqtt_client_publish(client_mqtt,
"havel/S/C/P", data_mqtt, 0, 2, 0);
else
esp_mqtt_client_publish(client_mqtt,
"havel/S/G/P", motor_state, 0, 2, 0);
break;

case 4:
if(sensor_value!=0)
esp_mqtt_client_publish(client_mqtt,
"havel/S/C/R", data_mqtt, 0, 2, 0);
else
esp_mqtt_client_publish(client_mqtt,
"havel/S/G/R", motor_state, 0, 2, 0);
break;

case 5:
if(sensor_value!=0)
esp_mqtt_client_publish(client_mqtt,
"havel/S/C/G", data_mqtt, 0, 2, 0);
else
esp_mqtt_client_publish(client_mqtt,
"havel/S/G/G", motor_state, 0, 2, 0);
break;
default:
break;}
}else{
printf("Error en datos recibidos\n
Suma:%u\n chk:%u\n", suma, control[2]);
trama_err[5]={224,192,192,157,190};
}}

//Tarea de control de los eventos
que suceden en UART

static void uart_esp_event_task(void
*pvParameters){
uart_event_t event;
size_t buffered_size;
uint8_t* dtmp = (uint8_t*)
malloc(CONFIG_UART_SIZE);
for(;;) {
if(xQueueReceive(uart_esp_queue,(void *
)&event,
(portTickType)portMAX_DELAY)) {
bzero(dtmp, CONFIG_UART_SIZE);
switch(event.type) {

//Evento que detecta datos en
buffer
case UART_DATA:
ESP_LOGI(TAG_U, "[UART DATA]:
%d", event.size);
(const char*) dtmp, event.size);
break;

//Evento ejecutado en registro fifo
overflow
case UART_FIFO_OVF:
```

```
ESP_LOGI(TAG_U, "FIFO OVERFLOW! Se recomienda aÃ±adir flow control CTS RTS");
art_flush_input(CONFIG_UART_ESP);
xQueueReset(uart_esp_queue);
break;

//Evento ejecutado en buffer overflow
case UART_BUFFER_FULL:
ESP_LOGI(TAG_U, "Ring buffer full! Se debe incrementar el tamaño del buffer uart_buffer_size");
uart_flush_input(CONFIG_UART_ESP);
xQueueReset(uart_esp_queue);
break;

//Evento ejecutado cuando existe errores en UART
case UART_BREAK:
ESP_LOGI(TAG_U, "Uart rx break");
break;

//Evento ejecutado cuando existe errores de paridad
case UART_PARITY_ERR:
ESP_LOGI(TAG_U, "Error de paridad detectado");
break;

//Evento ejecutado cuando existe errores en la trama
case UART_FRAME_ERR:
ESP_LOGI(TAG_U, "Error de frame");
break;

//Evento que detecta patrones dentro de UART
caseUART_PATTERN_DET:
uart_get_buffered_data_len(CONFIG_UART_ESP, &buffered_size);
intpos=uart_pattern_pop_pos(CONFIG_UART_ESP);
if (pos == -1) {
ESP_LOGI(TAG_U, "Incremente tamaño de buffer queue");
uart_flush_input(CONFIG_UART_ESP);
}else{
uart_read_bytes(CONFIG_UART_ESP, dtmp,
pos, 100 / portTICK_PERIOD_MS);
uint8_t pat[CONFIG_PT_N + 1];
memset(pat,0,sizeof(pat));
uart_read_bytes(CONFIG_UART_ESP, pat,CONFIG_PT_N,100/portTICK_PERIOD_MS);
controlUart(dtmp);
= esp_mqtt_client_publish(client_mqtt, "havel/S/C/B", data_mqtt, 0, 2, 0);

//writeMotores(dtmp);// ELIMINAR ESTA LINEA PARA VERIFICAR EL FUNCIONAMIENTO SI ALGO FALLA AL RECIBIR DATOS
}
break;
default:
ESP_LOGI(TAG_U, "uart event type: %d", event.type);
break;
}}}
free(dtmp);
```

```
dtmp = NULL;
vTaskDelete(NULL);}

//Aplicación MQTT
//Sección de código referente a las aplicaciones ejecutadas con MQTT
//Integración el Certificado SSL (TLS v1.2)

static const char *TAG_W= "EVENTO MQTT";
#if CONFIG_BROKER_CERTIFICATE_OVERRIDDEN == 1
static const uint8_t ca_pem_start[]  = "-----BEGINCERTIFICATE-----\n"CONFIG_BROKER_CERTIFICATE_OVERRIDE"\n-----END CERTIFICATE-----";
#else
extern const uint8_t ca_pem_start[] asm("_binary_ca_pem_start");
#endif
extern const uint8_t ca_pem_end[] asm("_binary_ca_pem_end");

//Función que genera la trama de transmisión a UART para la ejecución de movimientos.
void controlMotores(uint8_t giro){ // 3 estados por 12 motores 36 casos
uint8_t B_L=192, B_H=192;
printf("\nValor de giro:%u\n", giro);
switch (giro){

//Base Off
case 1:
printf("\nBase off\n");
B_L=B_L&207;
break;

//Base Izquierda
case 2:
printf("\nBase Izquierda\n");
B_L=B_L|208;
break;

//Base Derecha
case 3:
printf("\nBase Derecha\n");
B_L=B_L|224;
break;

//Shoulder Off
case 4:
printf("\nShoulder Off\n");
B_L=B_L&243;
break;

//Shoulder Down
case 5:
printf("\nShoulder Down\n");
B_L=B_L|196;
break;

//Shoulder UP
case 6:
printf("\nShoulder UP\n");
B_L=B_L|200;
break;
```

```
//Elbow Off
case 7:
printf("\nElbow Off\n");
B_L=B_L&252;
break;

//Elbow Down
case 8:
printf("\nElbow Down\n");
B_L=B_L|193;
break;

//Elbow UP
case 9:
printf("\nElbow UP\n");
B_L=B_L|194;
break;

//Pitch Off
case 10:
printf("\nPitch Off\n");
B_H=B_H&207;
break;

//Pitch Down
case 11:
printf("\nPitch Down\n");
B_H=B_H|208;
break;

//Pitch UP
case 12:
printf("\nPitch UP\n");
B_H=B_H|224;
break;

//Roll Off
case 13:
printf("\nRoll Off\n");
B_H=B_H&243;
break;

//Roll Izquierda
case 14:
printf("\nRoll Izquierda\n");
B_H=B_H|196;
break;

//Roll Derecha
case 15:
printf("\nRoll Derecha\n");
B_H=B_H|200;
break;

//Gripper Off
case 16:
printf("\nGripper Off\n");
B_H=B_H&252;
break;

//Gripper Close
case 17:
intf("\nGripper Close\n");
B_H=B_H|193;
break;

//Gripper Open
case 18:
printf("\nGriper OPen\n");
B_H=B_H|194;
break;
default:
```

```
printf("Control de Motor no definido");
break;}
uint8_t TMR_CHK=255-B_L;
TMR_CHK+=255;
TMR_CHK-=B_H;
char trama[4]={B_L, B_H, TMR_CHK,
190};
uart_write_bytes(CONFIG_UART_ESP,
trama, 4);}

//Función que controla el giro de
los motores

void funcionDesplazar(char topic[11],
char data[4]){
char topic_eje=topic[8];
char topic_sentido=topic[10];
uint8_t motor_on=0;
uint8_t sentido=0;
printf("DATA: %s", data);
if(data[0]=='1'){
motor_on=1;
printf("\nOK D\n");
}else{
printf("ERR D\n");
motor_on=0;}
printf("topic: %c\n", topic_sentido);
if(topic_sentido=='H'){
printf("OK S\n");
sentido=1;
}else{
sentido=2;}
switch (topic_eje){
case 'B'://BASE 1
controlMotores(1+motor_on*sentido);
break;
case 'S'://SHOULDER 2
controlMotores(4+motor_on*sentido);
break;
case 'E'://ELBOW 3
controlMotores(7+motor_on*sentido);
break;
case 'P'://PITCH 4
controlMotores(10+motor_on*sentido);
break;
case 'R'://ROLL 5
controlMotores(13+motor_on*sentido);
break;
case 'G'://GRIPPER 6
controlMotores(16+motor_on*sentido);
break;
default:
break;
}}

//Función que permite controlar
el posicionamiento de los ejes.

void funcionPosicionar(char topic[9], char

//Enciende Motores

data[7]){
int16_t pos_f=atoi(data);
int16_t pos_i=0;
switch (topic[8]){
case 'V':
printf("Set Velocidad a %i\n", pos_f);
break;

//BASE
case 'B':
```

```
printf("**********************************************");
printf("Ubicando base a %i\n", pos_f);
pcnt_set_event_value(UNIT_BASE, PCNT_EVT_THRES_1, pos_f);
pcnt_get_counter_value(UNIT_BASE, &pos_i);
if(pos_i<pos_f && pos_i!= pos_f){
controlMotores(2);
printf("GIRO 1\n");
}else{
controlMotores(3);
printf("Giro 2\n");
}
break;

//SHOULDER

case 'S':
printf("Ubicando Shoulder a %i\n", pos_f);
pcnt_set_event_value(UNIT_SHOULDER, PCNT_EVT_THRES_1, pos_f);
pcnt_get_counter_value(UNIT_SHOULDER, &pos_i);
if(pos_i<pos_f && pos_i!= pos_f){
controlMotores(5);
}else{
controlMotores(6);
}
break;

//ELBOW

case 'E':
printf("Ubicando Elbow a %i\n", pos_f);
pcnt_set_event_value(UNIT_ELBOW, PCNT_EVT_THRES_1, pos_f);
pcnt_get_counter_value(UNIT_ELBOW, &pos_i);
if(pos_i<pos_f && pos_i!= pos_f){
controlMotores(8);
}else{
controlMotores(9);
}
break;

//PITCH

case 'P':
printf("Ubicando pitch a %i\n", pos_f);
pcnt_set_event_value(UNIT_PITCH, PCNT_EVT_THRES_1, pos_f);
pcnt_get_counter_value(UNIT_PITCH, &pos_i);
if(pos_i<pos_f && pos_i!= pos_f){
controlMotores(11);
}else{
controlMotores(12);
}
break;

//ROLL

case 'R':
printf("Ubicando roll a %i\n", pos_f);
pcnt_set_event_value(UNIT_ROLL, PCNT_EVT_THRES_1, pos_f);
pcnt_get_counter_value(UNIT_ROLL, &pos_i);
if(pos_i<pos_f && pos_i!= pos_f){
controlMotores(14);
```

```
}else{
controlMotores(15);}
break;

//GRIPPER

case 'G':
printf("Ubicando gripper a %i\n", pos_f);
pcnt_set_event_value(UNIT_GRIPPER,
PCNT_EVT_THRES_1, pos_f);
pcnt_get_counter_value(UNIT_GRIPPER
, &pos_i);
if(pos_i<pos_f && pos_i!= pos_f){
controlMotores(17);
}else{
controlMotores(18);
}
break;
default:
printf("Ubicando defaul data %i\n",
pos_f);
break;
}}
void funcionHome(){
}

//Función que controla los
eventos de la comunicación
MQTT

char tr_off[4]={192, 192, 126, 190};
char tr[4]="0000";
static esp_err_t
mqtt_event_handler_cb(esp_mqtt_event_
handle_t event)
{
esp_mqtt_client_handle_t client = event-
>client;
int msg_id;
switch (event->event_id) {
case MQTT_EVENT_CONNECTED:
msg_id = esp_mqtt_client_publish(client,
"havel/led/conexion", "wifi", 0, 2, 0);
ESP_LOGI(TAG_W, "MQTT
CONECTADO!!");msg_id =
esp_mqtt_client_subscribe(client,
"havel/D/#", 2);
ESP_LOGI(TAG_W, "sent subscribe
successful, msg_id=%d", msg_id);
msg_id =
esp_mqtt_client_subscribe(client,
"havel/P/#", 2);
ESP_LOGI(TAG_W, "sent subscribe
successful, msg_id=%d", msg_id);
msg_id =
esp_mqtt_client_subscribe(client,
"havel/HOME", 2);
ESP_LOGI(TAG_W, "sent subscribe
successful, msg_id=%d", msg_id);
break;

//Pédida de conexión con broker MQTT

case MQTT_EVENT_DISCONNECTED:
ESP_LOGI(TAG_W,
"MQTT_EVENT_DISCONNECTED");
break;

//Suscripción exitosa

case MQTT_EVENT_SUBSCRIBED:
```

```
ESP_LOGI(TAG_W,
"MQTT_EVENT_SUBSCRIBED,
msg_id=%d", event->msg_id);
msg_id = esp_mqtt_client_publish(client,
"havel/led/conexion", "mqtt", 0, 2, 0);
ESP_LOGI(TAG_W, "sent publish
successful, msg_id=%d", msg_id);
break;

//Suscripción eliminada

case MQTT_EVENT_UNSUBSCRIBED:
ESP_LOGI(TAG_W,
"MQTT_EVENT_UNSUBSCRIBED,
msg_id=%d", event->msg_id);
break;

//Publicación exitosa

case MQTT_EVENT_PUBLISHED:
break;

//Datos recibidos por MQTT

 case MQTT_EVENT_DATA:
 ESP_LOGI(TAG_W,
"MQTT_EVENT_DATA");
 char mqtt_data[50];
char mqtt_topic[11];
snprintf(mqtt_data, event->data_len+1,
"%s", event->data);
snprintf(mqtt_topic, event->topic_len+1,
"%s", event->topic);
//printf("TOPIC=%.*s\r\n", event-
>topic_len, mqtt_topic);
//printf("DATA=%.*s\r\n", event-
>data_len+1, mqtt_data);
if(strncmp(event->topic, "havel/D",
7)==0){ //********** Si el topic es
DesplazarfuncionDesplazar(mqtt_topic,
mqtt_data);
}else if(strncmp(event->topic, "havel/P",
7)==0){//******** Si el topic es
PosicionarfuncionPosicionar(mqtt_topic,
mqtt_data);
}else if(strncmp(event->topic,
"havel/HOME", 10)==0){//**** Si el
topic es funciÃ³n home
}else printf("Error!! Topic no definido")
break;

//Detección de errores en MQTT

case MQTT_EVENT_ERROR:
ESP_LOGI(TAG_W,
"MQTT_EVENT_ERROR");
int mbedtls_err = 0;
esp_err_t err =
esp_tls_get_and_clear_last_error(event-
>error_handle, &mbedtls_err, NULL);
ESP_LOGI(TAG_W, "Last esp error code:
0x%x", err);
ESP_LOGI(TAG_W, "Last mbedtls
failure: 0x%x", mbedtls_err);
break;
default:
ESP_LOGI(TAG_W, "Other event
id:%d", event->event_id);
break;
}
return ESP_OK;}
```

```
//Aplicacion de MQTT

static void mqtt_event_handler(void *handler_args, esp_event_base_t base, int32_t event_id, void *event_data) {

//Mostrar eventos que pasan en la aplicación MQTT

ESP_LOGD(TAG_W, "Event dispatched from event loop base=%s, event_id=%d", base, event_id);
mqtt_event_handler_cb(event_data);
}

// Rutina de configuración de la comunicación MQTT en ESP IDF(Interfaz de desarrollo de la ESP)

static void mqtt_app_start(void){

// creación de constante, define las competentes que se necesita para la conexión

const esp_mqtt_client_config_t mqtt_cfg = {

//Creación de la URL, dirección del servidor

.uri = CONFIG_BROKER_URI,

// certificado TLS, para el cifrado de la comunicación MQTT

.cert_pem = (const char *)ca_pem_start,
};
ESP_LOGI(TAG_W, "[APP] Free memory: %d bytes", esp_get_free_heap_size());

//Creación de variable, para la manipulación de los datos que entran por MQTT

client_mqtt = esp_mqtt_client_init(&mqtt_cfg);
esp_mqtt_client_register_event(client_mqtt,ESP_EVENT_ANY_ID,
mqtt_event_handler, client_mqtt);
esp_mqtt_client_start(client_mqtt);
}
static struct {
bool flagP[6];
bool flagN[6];
char eje[6];
uint8_t pin[6];
} fc_Scorbot;

//Función de control de los finales de carrera

void finalesdeCarrera(){
int msg_id;
char topic_mqtt[20];
for (int i=0; i<6; i++){
snprintf(topic_mqtt, 12, "havel/S/F/%c", fc_Scorbot.eje[i]);
if(gpio_get_level(fc_Scorbot.pin[i])&&fc_Scorbot.flagP[i]){
fc_Scorbot.flagP[i]=0;
```

```
fc_Scorbot.flagN[i]=1;
esp_mqtt_client_publish(client_mqtt,
topic_mqtt, "1", 0, 2, 0);
}else if(fc_Scorbot.flagN){
fc_Scorbot.flagP[i]=1;
fc_Scorbot.flagN[i]=0;
esp_mqtt_client_publish(client_mqtt,
topic_mqtt, "0", 0, 2, 0);
}}

//Función Principal
void app_main(void){

//inicialización de PWM
ledc_init();

//encendido de leds indicadores

gpio_pad_select_gpio(LED_GREEN);
gpio_set_direction(LED_GREEN,
GPIO_MODE_OUTPUT);
gpio_set_level(LED_GREEN, 1);
vTaskDelay(1000                    /
portTICK_PERIOD_MS);
gpio_set_level(LED_GREEN, 0);
gpio_pad_select_gpio(36);
gpio_set_direction(36,
GPIO_MODE_INPUT);
gpio_pad_select_gpio(39);
 gpio_set_direction(39,
GPIO_MODE_INPUT);
gpio_pad_select_gpio(34);
gpio_set_direction(34,
PIO_MODE_INPUT);
gpio_pad_select_gpio(35);
gpio_set_direction(35,
GPIO_MODE_INPUT);

//Notificación por UART0 del
estado de arranque

ESP_LOGI(TAG_W, "[APP] Startup..");
ESP_LOGI(TAG_W, "[APP] Free
memory: %d bytes",
esp_get_free_heap_size());
ESP_LOGI(TAG_W, "[APP] IDF version:
%s", esp_get_idf_version());
esp_log_level_set("*", ESP_LOG_INFO);
esp_log_level_set("MQTT_CLIENT",
ESP_LOG_VERBOSE);
esp_log_level_set("MQTT_EXAMPLE",
ESP_LOG_VERBOSE);
esp_log_level_set("TRANSPORT_TCP",
ESP_LOG_VERBOSE);
esp_log_level_set("TRANSPORT_SSL",
ESP_LOG_VERBOSE);
esp_log_level_set("TRANSPORT",
ESP_LOG_VERBOSE);
esp_log_level_set("OUTBOX",
ESP_LOG_VERBOSE);

//Configuración de memoria flash

ESP_ERROR_CHECK(nvs_flash_init());
tcpip_adapter_init();
ESP_ERROR_CHECK(esp_event_loop_c
reate_default());
ESP_ERROR_CHECK(example_connect
());

//Inicializa MQTT
```

```
mqtt_app_start();

//Inicializa UART
init_serial_esp();

//Configuración de pines de finales de carrera

for(int i=0; i<6; i++){
fc_Scorbot.flagP[i]=1;
fc_Scorbot.flagN[i]=1;}
fc_Scorbot.eje[0]='B';
fc_Scorbot.eje[1]='S';
fc_Scorbot.eje[2]='P';
fc_Scorbot.eje[3]='E';
fc_Scorbot.eje[4]='R';
fc_Scorbot.eje[5]='G';
fc_Scorbot.eje[0]='B';
fc_Scorbot.eje[1]='S';
fc_Scorbot.eje[2]='P';
fc_Scorbot.eje[3]='E';
fc_Scorbot.eje[4]='R';
fc_Scorbot.eje[5]='G';
xTaskCreate(uart_esp_event_task, "Tarea de Uart", 2048, NULL, 12, NULL);// 12 es la prioridad

//Contadores de pulsos

pcnt_evt_queue = xQueueCreate(10, sizeof(pcnt_evt_t));
initContadores();
int16_t countb =0, counts=0, counte=0, countp=0, countr=0, countg=0, count=0;
pcnt_evt_t evt;
portBASE_TYPE res;
int msg_id=0;
char data_mqtt[40];
while (true){
res=xQueueReceive(pcnt_evt_queue, &evt, 100/portTICK_PERIOD_MS);
if(res==pdTRUE){
pcnt_get_counter_value(evt.unit, &count);
printf("Evento Encoder: unidad[%d] valor: %d\n", evt.unit, count);
printf("Event PCNT unit[%d]; cnt: %d\n", evt.unit, count);
if (evt.status & PCNT_EVT_THRES_1) {
printf("PosiciÃ³n alzanzada horario\n");
controlMotores(evt.unit+1);}
if (evt.status & PCNT_EVT_THRES_0) {
printf("PosiciÃ³n alcanzada antihorario\n");
controlMotores(1+evt.unit);}
if (evt.status & PCNT_EVT_L_LIM) {
printf("Limite de impacto inferior\n");
controlMotores(1+evt.unit);}
if (evt.status & PCNT_EVT_H_LIM) {
printf("Limite de impacto superior\n");
controlMotores(1+evt.unit);}
if (evt.status & PCNT_EVT_ZERO) {
printf("Eje en Home\n");}
}else {
pcnt_get_counter_value(UNIT_BASE, &countb);
pcnt_get_counter_value(UNIT_SHOULDER,&counts);
pcnt_get_counter_value(UNIT_ELBOW, &counte);
pcnt_get_counter_value(UNIT_PITCH, &countp);
```

```
pcnt_get_counter_value(UNIT_ROLL,
&countr);
pcnt_get_counter_value(UNIT_GRIPPER
, &countg);
if(countb != PB_B){
PB_B=countb;
sprintf(data_mqtt, "%d", PB_B);
msg_id                              =
esp_mqtt_client_publish(client_mqtt,
"havel/U/B", data_mqtt, 0, 2, 0);}
if(counts != PS_B){
PS_B=counts;
sprintf(data_mqtt, "%d", PS_B);
msg_id                              =
esp_mqtt_client_publish(client_mqtt,
"havel/U/S", data_mqtt, 0, 2, 0);}
if(counte != PE_B){
PE_B=counte;
sprintf(data_mqtt, "%d", PE_B);
msg_id                              =
esp_mqtt_client_publish(client_mqtt,
"havel/U/E", data_mqtt, 0, 2, 0);}
if(countp != PP_B){
PP_B=countp;
sprintf(data_mqtt, "%d", PP_B);
msg_id                              =
esp_mqtt_client_publish(client_mqtt,
"havel/U/P", data_mqtt, 0, 2, 0);}
if(countr != PR_B){
PR_B=countr;
sprintf(data_mqtt, "%d", PR_B);
msg_id                              =
esp_mqtt_client_publish(client_mqtt,
"havel/U/R", data_mqtt, 0, 2, 0);}
if(countg != PG_B){
PG_B=countg;
sprintf(data_mqtt, "%d", PG_B);
msg_id                              =
esp_mqtt_client_publish(client_mqtt,
"havel/U/G", data_mqtt, 0, 2, 0);}}
gpio_set_level(LED_GREEN, 1);}
```

ANEXO C

EL SIGUIENTE CÓDIGO FUENTE CORRESPONDE AL SOFTWARE DEL MICROCONTROLADOR ESCLAVO QUE PERMITE MANIPULAR LOS MOTORES Y MONITOREAR LA CORRIENTE DE LOS MISMOS

```
//Importación de librerías requeridas

#include <stdio.h>
#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "driver/gpio.h"
#include "sdkconfig.h"
#include "driver/adc.h"
#include "esp_adc_cal.h"
#include <string.h>
#include "freertos/queue.h"
#include "driver/uart.h"
#include "esp_log.h"
#define LED_BLUE 23

//Definiciones y configuraciones
de comunicación UART

//Creación de variable reporte de eventos
de UART 2n en UART0

static const char *TAG = "Puerto Serial";

//Creación de buffer para
transferencia de datos UART
entre tareas
static QueueHandle_t uart_esp_queue;

//Funciones de control de motores
///Función que lee los datos
recibidos por UART desde el
maestro, verifica la existencia de
errores y convierte el valor de dos
bytes a valores binarios que
controlan el giro de los motores.
Cada dos bits, controlan el estado
del motor 00 -- 11 off 01 -- 10 ON
(D-I)
static void readSensors_task(void
*pvParameters){
int i=0;

//Detección de impactos mediante
el ADC

uint16_t acumulador;
for(;;){
i++;
acumulador=0;
for(int j=0;j<10;j++)
acumulador+=adc1_get_raw(CONFIG_C
MB);

//Detección de impactos mediante el ADC

uint16_t CMB = acumulador/10;
```

```
if(CMB<2854||CMB>3352||i==0){

//En el caso de que este fuera de rango (Valores binarios para corrientes de +- 1A)

uint8_t SMB=0;

//Si el caso no es por notificación temporal es exeso de corriente

if(i!=0){

//Apaga el pin de control del motor
gpio_set_level(CONFIG_MBP,0);

//Apaga el pin de control del motor
gpio_set_level(CONFIG_MBP,0);

//240 Valor binario de error  de corriente B3:0 B4:0
SMB=240;

//En el caso de que no exista impacto se procede con la petición recibida por UART
}else{

//En el caso de que no exista impacto se procede con la petición recibida por UART

SMB=192|(gpio_get_level(CONFIG_MBP)*16+gpio_get_level(CONFIG_MBN)*8); //192 Valor sin error
}
uint8_t

//Calculo de cheqsum para verificación de errores

CMB_MSB=192|((CMB&16320))>>6;

//Entramado de datos

uint8_t CMB_LSB=192|(CMB&255);
uint8_t TMB_CHK=255-CMB_LSB;
TMB_CHK+=255;
TMB_CHK-=CMB_MSB;
TMB_CHK+=255;
TMB_CHK-=SMB;
char trama_cmb[5]={SMB,CMB_MSB,CMB_LSB,TMB_CHK, 190};
uart_write_bytes(CONFIG_UART_ESP, trama_cmb, 5);
}

//Proceso indentico al anterior para el eje de shulder

acumulador=0;
for(int j=0;j<10;j++)
acumulador+=adc1_get_raw(CONFIG_CMS);
uint16_t CMS = acumulador/10;
if(CMS<2854||CMB>3352||i==0){
uint8_t SMS=0;
if(i!=0){
gpio_set_level(CONFIG_MSP,0);
gpio_set_level(CONFIG_MSN,0);
SMS=241;
}else{
```

```
SMS=193|(gpio_get_level(CONFIG_MS
P)*16+gpio_get_level(CONFIG_MSN)*8
);;
}
uint8_t
CMS_MSB=192|((CMS&16320)>>6);
uint8_t CMS_LSB=192|(CMS&255);
uint8_t TMS_CHK=255-CMS_LSB;
TMS_CHK+=255;
TMS_CHK-=CMS_MSB;
TMS_CHK+=255;
TMS_CHK-=SMS;
char
trama_cms[5]={SMS,CMS_MSB,CMS_
LSB,TMS_CHK, 190};
uart_write_bytes(CONFIG_UART_ESP,
trama_cms, 5);
}
//Proceso indentico al anterior
para el eje de Elbow

acumulador=0;
for(int j=0;j<10;j++)
acumulador+=adc1_get_raw(CONFIG_C
ME);
uint16_t CME = acumulador/10;
if(CME<2854||CME>3352||i==0){
uint8_t SME=0;
if(i!=0){
gpio_set_level(CONFIG_MEP,0);
gpio_set_level(CONFIG_MEN,0);
SME=242;
}else{
SME=194|(gpio_get_level(CONFIG_ME
P)*16+gpio_get_level(CONFIG_MEN)*8
);
}
uint8_t
CME_MSB=192|((CME&16320)>>6);
uint8_t CME_LSB=192|(CME&255);
uint8_t TME_CHK=255-CME_LSB;
TME_CHK+=255;
TME_CHK-=CME_MSB;
TME_CHK+=255;
TME_CHK-=SME;
char
trama_cme[5]={SME,CME_MSB,CME_
LSB,TME_CHK, 190};
uart_write_bytes(CONFIG_UART_ESP,
trama_cme, 5);
}

//Proceso idéntico al anterior para el eje de
pitch

acumulador=0;
for(int j=0;j<10;j++)
acumulador+=adc1_get_raw(CONFIG_C
MP);
uint16_t CMP = acumulador/10;
if(CMP<2854||CMP>3352||i==0){
uint8_t SMP=0;
if(i!=0){
gpio_set_level(CONFIG_MPP,0);
gpio_set_level(CONFIG_MPN,0);
SMP=243;
}else{
SMP=195|(gpio_get_level(CONFIG_MP
P)*16+gpio_get_level(CONFIG_MPN)*8
);
}
```

```
uint8_t
CMP_MSB=192|((CMP&16320)>>6);
uint8_t CMP_LSB=192|(CMP&255);
uint8_t TMP_CHK=255-CMP_LSB;
TMP_CHK+=255;
TMP_CHK-=CMP_MSB;
TMP_CHK+=255;
TMP_CHK-=SMP;
char
trama_cmp[5]={SMP,CMP_MSB,CMP_
LSB,TMP_CHK, 190};
uart_write_bytes(CONFIG_UART_ESP,
trama_cmp, 5);
}

//Proceso idéntico al anterior para
el eje de roll

acumulador=0;
for(int j=0;j<10;j++)
acumulador+=adc1_get_raw(CONFIG_C
MR);
uint16_t CMR = acumulador/10;
if(CMR<2854||CMR>3352||i==0){
uint8_t SMR=0;
if(i!=0){
gpio_set_level(CONFIG_MRP,0);
gpio_set_level(CONFIG_MRN,0);
SMR=244;
}else{
SMR=196|(gpio_get_level(CONFIG_MR
P)*16+gpio_get_level(CONFIG_MRN)*
8);
}
uint8_t
CMR_MSB=192|((CMR&16320)>>6);
uint8_t CMR_LSB=192|(CMR&255);
uint8_t TMR_CHK=255-CMR_LSB;
TMR_CHK+=255;
TMR_CHK-=CMR_MSB;
TMR_CHK+=255;
TMR_CHK-=SMR;
char
trama_cmr[5]={SMR,CMR_MSB,CMR_
LSB,TMR_CHK, 190};
uart_write_bytes(CONFIG_UART_ESP,
trama_cmr, 5);
}

//Proceso idéntico al anterior para
el eje de gripper

acumulador=0;
for(int j=0;j<10;j++)
acumulador+=adc1_get_raw(CONFIG_C
MG);
uint16_t CMG = acumulador/10;
if(CMG<2854||CMG>3352||i==0){
uint8_t SMG=0;
if(i!=0){
gpio_set_level(CONFIG_MGP,0);
gpio_set_level(CONFIG_MGN,0);
SMG=245;
}else{
SMG=197|(gpio_get_level(CONFIG_MG
P)*16+gpio_get_level(CONFIG_MGN)*
8);
}
uint8_t
CMG_MSB=192|((CMG&16320)>>6);
uint8_t CMG_LSB=192|(CMG&255);
uint8_t TMG_CHK=255-CMG_LSB;
```

```
TMG_CHK+=255;
TMG_CHK-=CMG_MSB;
TMG_CHK+=255;
TMG_CHK-=SMG;
char
trama_cmg[5]={SMG,CMG_MSB,CMG_
LSB,TMG_CHK, 190};
uart_write_bytes(CONFIG_UART_ESP,
trama_cmg, 5);
}
if(i<10){
i=-1;
}
vTaskDelay(1000 /
portTICK_PERIOD_MS);
}}

//Función auxiliar que permite
manipular los 6 motores como un
solo puerto.
//Recibe una vector de bytes
desde Uart y escribe en forma de
puertos los valores
// En los pines de salida para controlar el
movimiento de los motores.

void writeMotores(uint8_t control[3]){
printf("FunciÃ³n Write Motores\n");
uint8_t B0=255-control[0];
uint8_t B1=255-control[1];
uint8_t suma=B0+B1;
uint8_t chksm=control[2]-suma;
uint8_t RET;
uint8_t CHK;
if(control[2]<127&&chksm==0){
printf("Trama correcta\n Suma:%u\n
chk:%u\n datos%u\n", suma, control[2],
control[0]);
control[0]=control[0]&63;
control[1]=control[1]&63;
printf("B0%u\n B1:%u\n", control[0],
control[1]);

//Escribe el estado de base
gpio_set_level(CONFIG_MBP,
(control[0] & 0x20) >> 5);

//Escribe el estado de base
gpio_set_level(CONFIG_MBN,
(control[0] & 0x10) >> 4);

//Escribe el estado de sholder
gpio_set_level(CONFIG_MSP,
(control[0] & 0x08) >> 3);

//Escribe el estado de shoulder
gpio_set_level(CONFIG_MSN,
(control[0] & 0x04) >> 2);

//Escribe el estado de elbow
gpio_set_level(CONFIG_MEP,
(control[0] & 0x02) >> 1);

//Escribe el estado de elbow
gpio_set_level(CONFIG_MEN,
(control[0] & 0x01));

//Escribe el estado de picth
gpio_set_level(CONFIG_MPP,
(control[1] & 0x20) >> 5);
```

```
//Escribe el estado de picth
gpio_set_level(CONFIG_MPN,
(control[1] & 0x10) >> 4);

//Escribe el estado de roll
gpio_set_level(CONFIG_MRP,
(control[1] & 0x08) >> 3);

//Escribe el estado de roll
gpio_set_level(CONFIG_MRN,
(control[1] & 0x04) >> 2);

//Escribe el estado de gripper
gpio_set_level(CONFIG_MGP,
(control[1] & 0x02) >> 1);

//Escribe el estado de gripper
gpio_set_level(CONFIG_MGN,
(control[1] & 0x01));

//Verifica si el motor se enciende y reporta
el estado del motor por UART (base)

RET=192|(gpio_get_level(CONFIG_MB
P)*16+gpio_get_level(CONFIG_MBN)*
8);
CHK=255-192;
CHK+=255;
CHK-=192;
CHK+=255;
CHK-=RET;
char     trama_rb[5]={RET,192,192,CHK,
190};
uart_write_bytes(CONFIG_UART_ESP,
trama_rb, 5);
printf("GPIO B: %u --
%u\n",gpio_get_level(CONFIG_MBP),gp
io_get_level(CONFIG_MBN));
printf("%u", RET);

//Verifica si el motor se enciende
y reporta el estado del motor por
UART (shoulder)

RET=193|(gpio_get_level(CONFIG_MSP
)*16+gpio_get_level(CONFIG_MSN)*8);
CHK=255-192;
CHK+=255;
CHK-=192;
CHK+=255;
CHK-=RET;
char     trama_rs[5]={RET,192,192,CHK,
190};
uart_write_bytes(CONFIG_UART_ESP,
trama_rs, 5);
printf("GPIO S: %u",RET);

//Verifica si el motor se enciende
y reporta el estado del motor por
UART (elbow)

RET=194|(gpio_get_level(CONFIG_MEP
)*16+gpio_get_level(CONFIG_MEN)*8)
;
CHK=255-192;
CHK+=255;
CHK-=192;
CHK+=255;
CHK-=RET;
char     trama_re[5]={RET,192,192,CHK,
190};
```

```
uart_write_bytes(CONFIG_UART_ESP,
trama_re, 5);
printf("GPIO E: %u",RET);

//Verifica si el motor se enciende
y reporta el estado del motor por
UART (pitch)

RET=195|(gpio_get_level(CONFIG_MPP
)*16+gpio_get_level(CONFIG_MPN)*8);
CHK=255-192;
CHK+=255;
CHK-=192;
CHK+=255;
CHK-=RET;
char trama_rp[5]={RET,192,192,CHK,
190};
uart_write_bytes(CONFIG_UART_ESP,
trama_rp, 5);
printf("GPIO P: %u",RET);
//Verifica si el motor se enciende
y reporta el estado del motor por
UART (roll)

RET=196|(gpio_get_level(CONFIG_MR
P)*16+gpio_get_level(CONFIG_MRN)*
8);
CHK=255-192;
CHK+=255;
CHK-=192;
CHK+=255;
CHK-=RET;
char trama_rr[5]={RET,192,192,CHK,
190};
uart_write_bytes(CONFIG_UART_ESP,
trama_rr, 5);
printf("GPIO R: %u",RET);

//Verifica si el motor se enciende
y reporta el estado del motor por
UART (gripper)

RET=197|(gpio_get_level(CONFIG_MG
P)*16+gpio_get_level(CONFIG_MGN)*
8);
CHK=255-192;
CHK+=255;
CHK-=192;
CHK+=255;
CHK-=RET;
char trama_rg[5]={RET,192,192,CHK,
190};
uart_write_bytes(CONFIG_UART_ESP,
trama_rg, 5);
printf("GPIO G: %u",RET);
}else{
char
trama_err[5]={224,192,192,157,190};
uart_write_bytes(CONFIG_UART_ESP,
trama_err, 5);
}}

//Funciones de comunicación
rs232

static void uart_esp_event_task(void
*pvParameters){

//Buffer de almacenamiento de eventos

uart_event_t event;
size_t buffered_size;
```

```
uint8_t* dtmp = (uint8_t*) malloc(CONFIG_UART_SIZE);
for(;;) {
if(xQueueReceive(uart_esp_queue, (void * )&event, (portTickType)portMAX_DELAY)) {
bzero(dtmp, CONFIG_UART_SIZE);

//Variable de reporte de eventos

ESP_LOGI(TAG, "uart[%d] event:", CONFIG_UART_ESP);
switch(event.type) {
//Detección de datos en buffer

case UART_DATA:
break;

//Registro FIFO overflow

case UART_FIFO_OVF:
ESP_LOGI(TAG, "FIFO OVERFLOW! Se recomienda aÃ±adir flow control CTS RTS");
uart_flush_input(CONFIG_UART_ESP);
xQueueReset(uart_esp_queue);
break;

//Buffer UART overflow

case UART_BUFFER_FULL:
ESP_LOGI(TAG, "Ring buffer full! Se debe incrementar el tamaÃ±o del buffer uart_buffer_size");
uart_flush_input(CONFIG_UART_ESP);
xQueueReset(uart_esp_queue);
break;

//Errores en UART

case UART_BREAK:
ESP_LOGI(TAG, "Uart rx break");
break;

//Error de paridad

case UART_PARITY_ERR:
ESP_LOGI(TAG, "Error de paridad detectado");
break;
case UART_FRAME_ERR:
ESP_LOGI(TAG, "Error de frame");
break;

//Detección de trama

case UART_PATTERN_DET:
uart_get_buffered_data_len(CONFIG_UART_ESP, &buffered_size);
intpos=uart_pattern_pop_pos(CONFIG_UART_ESP);
ESP_LOGI(TAG, "[TRAMA DECTECTADA] Pos: %d, TamaÃ±o de la trama: %d", pos, buffered_size);
if (pos == -1) {
ESP_LOGI(TAG,"Incrementetamaño de buffer queue");
uart_flush_input(CONFIG_UART_ESP);
 } else {
uart_read_bytes(CONFIG_UART_ESP, dtmp, pos, 100 / portTICK_PERIOD_MS);
```

```
uint8_t pat[CONFIG_PT_N + 1];
memset(pat, 0, sizeof(pat));
uart_read_bytes(CONFIG_UART_ESP,
pat, CONFIG_PT_N, 100 /
portTICK_PERIOD_MS);
ESP_LOGI(TAG, "read data: %s", dtmp);
ESP_LOGI(TAG, "read pat : %s", pat);

// ELIMINAR ESTA LINEA PARA
VERIFICAR EL FUNCIONAMIENTO SI
ALGO FALLA AL RECIBIR DATOS

writeMotores(dtmp); }
break;
default:
ESP_LOGI(TAG, "uart event type: %d",
event.type);
break;
}}}
free(dtmp);
dtmp = NULL;
vTaskDelete(NULL);
}

//Función de configuración de la
comunicación serial.

void init_serial_esp(){
const uart_config_t serial_config = {
.baud_rate = 115200,
.data_bits = UART_DATA_8_BITS,
.parity = UART_PARITY_DISABLE,
.stop_bits = UART_STOP_BITS_1,
.flow_ctrl=UART_HW_FLOWCTRL_DI
SABLE,
 };
ESP_ERROR_CHECK(uart_param_confi
g(CONFIG_UART_ESP,
&serial_config));
ESP_ERROR_CHECK(uart_set_pin(CO
NFIG_UART_ESP,
CONFIG_UART_TX,
CONFIG_UART_RX,
CONFIG_UART_RTS,
CONFIG_UART_CTS));
ESP_ERROR_CHECK(uart_driver_instal
l(CONFIG_UART_ESP,
CONFIG_UART_SIZE,
CONFIG_UART_SIZE, 20,
&uart_esp_queue,0));
uart_enable_pattern_det_intr(CONFIG_U
ART_ESP, CONFIG_UART_PT,
CONFIG_PT_N, 10000, 10, 10);
uart_pattern_queue_reset(CONFIG_UAR
T_ESP, 20);
}

//Función principal

void app_main(void)
{
esp_log_level_set(TAG,
ESP_LOG_INFO);
init_serial_esp();
uint8_t motores[12]={CONFIG_MBP,
CONFIG_MBN, CONFIG_MSP,
CONFIG_MSN, CONFIG_MEP,
CONFIG_MEN,
CONFIG_MPP, CONFIG_MPN,
CONFIG_MRP, CONFIG_MRN,
CONFIG_MGP, CONFIG_MGN};
```

```
adc1_config_width(ADC_WIDTH_BIT_
12);
for(int i=0; i<12; i++){
gpio_pad_select_gpio(motores[i]);
gpio_set_direction(motores[i],
GPIO_MODE_INPUT_OUTPUT);
if(i<8 && i!= 2 && i!=1 )
adc1_config_channel_atten(i,
ADC_ATTEN_DB_11);
}
gpio_pad_select_gpio(LED_BLUE);
gpio_set_direction(LED_BLUE,
GPIO_MODE_INPUT_OUTPUT)
xTaskCreate(uart_esp_event_task, "Tarea
de Uart", 2048, NULL, 12, NULL);// 12 es
la prioridad
xTaskCreate(readSensors_task, "Tarea de
Sensores", 2048, NULL, 12, NULL);// 12
es la prioridad
while(1) {
gpio_set_level(LED_BLUE, 0);
vTaskDelay(500 /
portTICK_PERIOD_MS);
UART_PIN_NO_CHANGE);
    gpio_set_level(LED_BLUE, 1);
vTaskDelay(500 /
portTICK_PERIOD_MS);
}}
```

ANEXO D

HOJA DE DATOS TÉCNICOS DE LA ESP32

Especificaciones

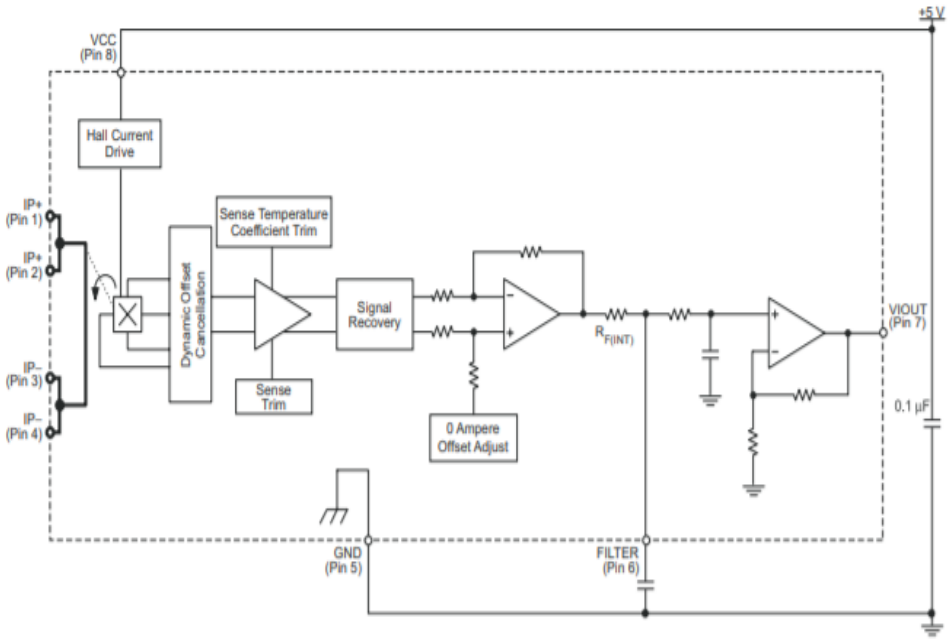
Atributo del producto	Valor de atributo
Fabricante:	Espressif
Categoría de producto:	Módulos WiFi (802.11)
RoHS:	RoHS Detalles
Protocolo admitido:	802.11 b/e/g/i/n, Bluetooth
Frecuencia:	2.4 GHz to 2.5 GHz
Velocidad de transmisión de datos:	150 Mb/s
Potencia de salida:	20 dBm
Voltaje de alimentación operativo:	2.7 V to 3.6 V
Temperatura de trabajo mínima:	- 40 C
Temperatura de trabajo máxima:	+ 85 C
Protocolo: WiFi - 802.11:	WiFi
Dimensiones:	18 mm x 25.5 mm x 3.1 mm
Empaquetado:	Cut Tape
Empaquetado:	MouseReel
Empaquetado:	Reel
Seguridad:	WPA/WPA2/WPA2-Enterprise/WPS
Serie:	ESP32-WROOM
Marca:	Espressif Systems
Sensibles a la humedad:	Yes
Tipo de producto:	WiFi Modules
Cantidad de empaque de fábrica:	550
Subcategoría:	Wireless & RF Modules
Peso de la unidad:	2.500 g

Categories	Items	Specifications
Certification	RF certification	FCC/CE-RED/IC/TELEC/KCC/SRRC/NCC
	Wi-Fi certification	Wi-Fi Alliance
	Bluetooth certification	BQB
	Green certification	RoHS/REACH
Test	Reliablity	HTOL/HTSL/uHAST/TCT/ESD
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps)
		A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with –97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
		AFH
	Audio	CVSD and SBC
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I^2C, LED PWM, Motor PWM, I^2S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range	–40 °C ~ +85 °C
	Package size	(18.00±0.10) mm × (25.50±0.10) mm × (3.10±0.10) mm
	Moisture sensitivity level (MSL)	Level 3

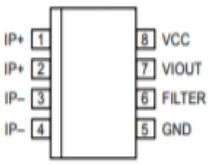
ANEXO E

HOJA DE DATOS TÉCNICOS DEL MODULO SENSOR DE CORRIENTE ACS712

Functional Block Diagram



Pin-out Diagram



Terminal List Table

Number	Name	Description
1 and 2	IP+	Terminals for current being sensed; fused internally
3 and 4	IP–	Terminals for current being sensed; fused internally
5	GND	Signal ground terminal
6	FILTER	Terminal for external capacitor that sets bandwidth
7	VIOUT	Analog output signal
8	VCC	Device power supply terminal

COMMON OPERATING CHARACTERISTICS[1] over full range of T_A, C_F = 1 nF, and V_{CC} = 5 V, unless otherwise specified

Characteristic	Symbol	Test Conditions	Min.	Typ.	Max.	Units
ELECTRICAL CHARACTERISTICS						
Supply Voltage	V_{CC}		4.5	5.0	5.5	V
Supply Current	I_{CC}	V_{CC} = 5.0 V, output open	–	10	13	mA
Output Capacitance Load	C_{LOAD}	VIOUT to GND	–	–	10	nF
Output Resistive Load	R_{LOAD}	VIOUT to GND	4.7	–	–	kΩ
Primary Conductor Resistance	$R_{PRIMARY}$	T_A = 25°C	–	1.2	–	mΩ
Rise Time	t_r	I_P = I_P(max), T_A = 25°C, C_{OUT} = open	–	5	–	µs
Frequency Bandwidth	f	–3 dB, T_A = 25°C; I_P is 10 A peak-to-peak	–	80	–	kHz
Nonlinearity	E_{LIN}	Over full range of I_P	–	1.5	–	%
Symmetry	E_{SYM}	Over full range of I_P	98	100	102	%
Zero Current Output Voltage	$V_{IOUT(Q)}$	Bidirectional; I_P = 0 A, T_A = 25°C	–	V_{CC} × 0.5	–	V
Power-On Time	t_{PO}	Output reaches 90% of steady-state level, T_J = 25°C, 20 A present on leadframe	–	35	–	µs
Magnetic Coupling[2]			–	12	–	G/A
Internal Filter Resistance[3]	$R_{F(INT)}$			1.7		kΩ

ANEXO F

HOJA DE DATOS TÉCNICOS DE L293B

Especificaciones

Atributo del producto	Valor de atributo	Buscar productos similares
Fabricante:	STMicroelectronics	☐
Categoría de producto:	Motor / Movimiento /Controladores y dispositivos de ignición	☑
RoHS:	RoHS Detalles	
Producto:	Fan / Motor Controllers / Drivers	☐
Tipo:	Half Bridge	☐
Voltaje de alimentación operativo:	4.5 V to 36 V	☐
Corriente de salida:	1 A	☐
Corriente de suministro operativa:	2 mA	☐
Temperatura de trabajo mínima:	- 40 C	☐
Temperatura de trabajo máxima:	+ 150 C	☐
Estilo de montaje:	Through Hole	☐
Paquete / Cubierta:	PDIP-16	☐
Empaquetado:	Tube	☐
Número de salidas:	4 Output	
Rango de temperatura de trabajo:	- 40 C to + 150 C	
Serie:	L293B	
Marca:	STMicroelectronics	
Tipo de producto:	Motor / Motion / Ignition Controllers & Drivers	
Cantidad de empaque de fábrica:	1000	
Subcategoría:	PMIC - Power Management ICs	
Peso de la unidad:	1.628 g	

ANEXO G

HOJA DE DATOS TÉCNICOS DEL REGULADOR DE VOLTAJE 7805

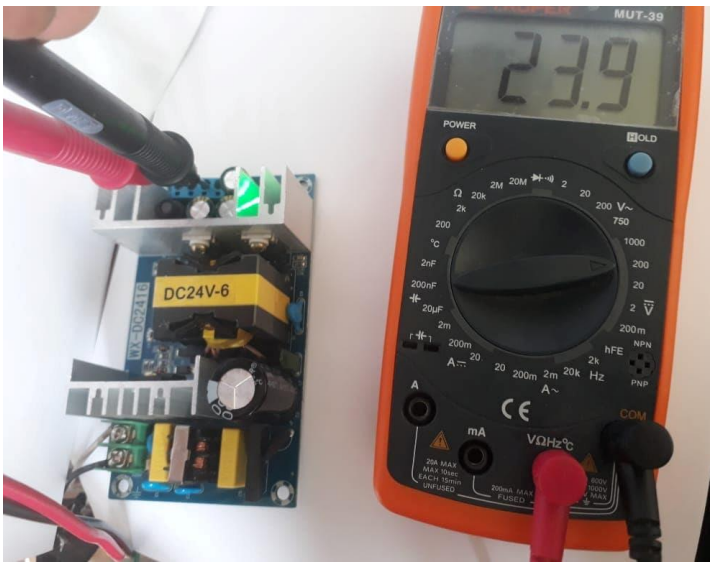
Especificaciones

Atributo del producto	Valor de atributo
Fabricante:	STMicroelectronics
Categoría de producto:	Reguladores de tensión lineal
RoHS:	RoHS Detalles
Estilo de montaje:	Through Hole
Paquete / Cubierta:	TO-220
Número de salidas:	1 Output
Polaridad:	Positive
Voltaje de salida:	5 V
Corriente de salida:	1 A
Tipo de salida:	Fixed
Voltaje de entrada MÁX.:	35 V
Voltaje de entrada MIN.:	7 V
Temperatura de trabajo mínima:	0 C
Temperatura de trabajo máxima:	+ 125 C
Regulación de carga:	100 mV
Regulación de línea:	100 mV
Serie:	L78
Marca:	STMicroelectronics
PSRR / Rechazo de propagación - Típica:	62 dB
Tipo de producto:	Linear Voltage Regulators
Cantidad de empaque de fábrica:	1000
Subcategoría:	PMIC - Power Management ICs
Peso de la unidad:	1.600 g

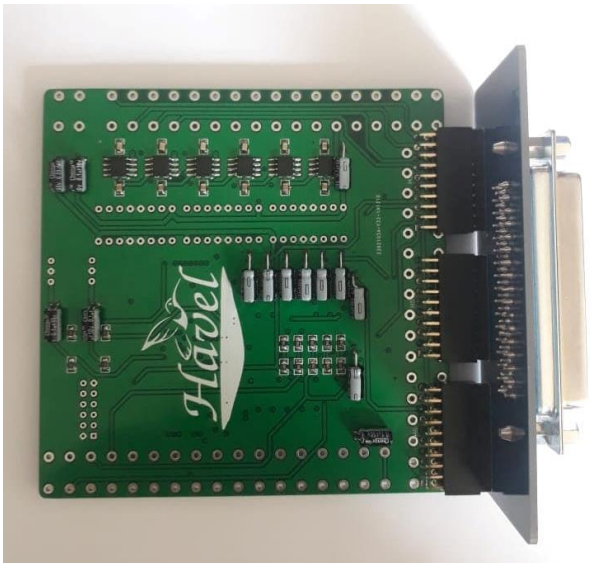
ANEXO H

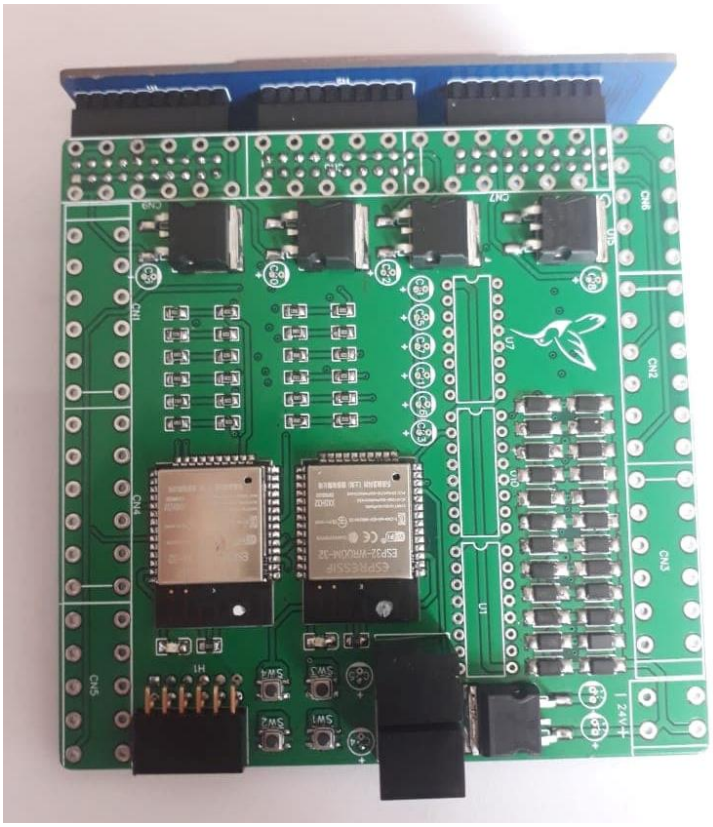
IMPLEMENTACION DEL CONTROLADOR ELECTRONICO DE SOFTWARE LIBRE PARA LA OPERACIÓN DEL BRAZO ROBOTICO SCORBOT ER_4U.

Fuente de poder de corriente continua con un voltaje de salida de 24V DC.

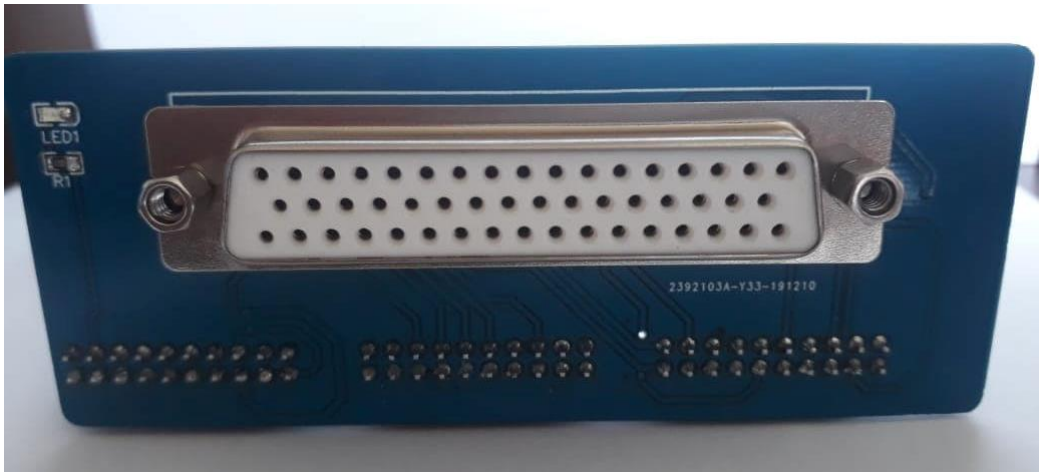


Tarjeta primaria, donde contiene todos los dispositivos electrónicos del módulo MQTT





Interfaz angular, usado como adaptador de conexión entre el circuito PCB de la tarjeta primaria y el conector DD50 macho del Scorbot ER_4U.



Encendido del módulo electrónico