



SECRETARÍA DE EDUCACIÓN PÚBLICA
TECNOLÓGICO NACIONAL DE MÉXICO
INSTITUTO TECNOLÓGICO DE MÉRIDA

ITM

**“SISTEMA PARA AUTOMATIZAR EL MONITOREO DE
PARÁMETROS EN POZOS”**

OPCIÓN
TITULACIÓN INTEGRAL
(TESIS)

PARA OPTAR AL TÍTULO DE:
INGENIERO ELECTRÓNICO

PRESENTA:
HÉCTOR JOSÉ SOLÍS CASTILLO

ASESOR:
DR. JOSÉ RAMÓN ATOCHE ENSEÑAT

CO ASESOR:
DR. ALEJANDRO ARTURO CASTILLO ATOCHE

MÉRIDA, YUCATÁN, MÉXICO



ANEXO XXXIII. FORMATO DE LIBERACIÓN DE PROYECTO PARA LA TITULACIÓN INTEGRAL

Lugar y fecha: Mérida, Yuc., 23 de Noviembre de 2022.

Asunto: Liberación de proyecto para la titulación integral.

MPEDR. FÉLIX JOSÉ AGUILAR VÁZQUEZ.
JEFE DE LA DIVISIÓN DE ESTUDIOS PROFESIONALES
PRESENTE

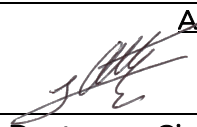
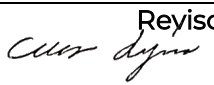
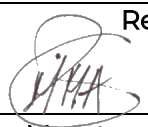
Por este medio informo que ha sido liberado el siguiente proyecto para la titulación integral:

Nombre del estudiante y/o egresado:	SOLÍS CASTILLO HÉCTOR JOSÉ
Carrera:	INGENIERÍA ELECTRÓNICA
No. de control:	E17081613
Nombre del proyecto:	SISTEMA PARA AUTOMATIZAR EL MONITOREO DE PARÁMETROS EN POZOS
Producto:	TESIS

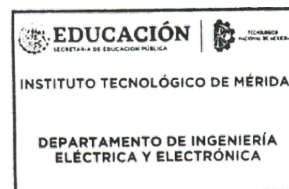
Agradezco de antemano su valioso apoyo en esta importante actividad para la formación profesional de nuestros egresados.

ATENTAMENTE

ING. JOSÉ FIDEL RODRÍGUEZ HUERTA
JEFE DEL DEPTO. DE ING. ELÉCTRICA Y ELECTRÓNICA

 Asesor	 Revisor	 Revisor
Doctor en Ciencias en Ingeniería Electrónica José Ramón Atoche Enseñat	Doctor en Ciencias en Ingeniería Electrónica Carlos Alberto Luján Ramírez	Maestra en Tecnología Educativa María Margarita Álvarez Cervera

* Solo aplica para el caso de tesis o tesina
c.c.p.- Expediente.



AGRADECIMIENTOS

Quiero agradecer a mi alma máter el Tecnológico Nacional de México/Instituto Tecnológico de Mérida por brindarme las herramientas y el conocimiento necesario para poder realizar este trabajo de ingeniería; de igual forma a la Universidad Nacional Autónoma de México y a la Universidad Autónoma de Yucatán por ofrecerme la oportunidad de desarrollar este proyecto y aplicar los conocimientos adquiridos hasta el momento.

También quiero agradecer a mis profesores y asesores: el Dr. José Ramón Atoche Enseñat, el Dr. Alejandro Castillo Atoche, al I.E. Rubén Rosado y al Dr. Víctor Sandoval Curmina por su confianza, paciencia y dedicación en su enseñanza y acompañamiento desde el inicio hasta el término de este trabajo. De igual forma, agradecerles las oportunidades que me brindaron, mismas que me capacitaron para ser el profesionista que soy.

Le agradezco a las profesoras, profesores y al personal humano del ITMérida que me brindaron consejo, así como su tiempo y esmero en su enseñanza, preparándome para la vida más allá de la institución.

Quiero agradecer a mis amigos y mi familia, especialmente a mis papás, mis abuelitas y mi hermana por apoyarme en los momentos de mayor duda, por regalarme el don de la amistad y compañerismo; así como el afecto y cariño del hogar. Por todas las risas, enojos, tristezas y experiencias que juntos pasamos y me han hecho la persona que soy.

De igual forma, quiero agradecer a Didier y Daniel por impulsarme a tomar las decisiones orientadas a mis metas y sueños, por motivarme a no rendirme cuando el “viento soplaba en contra”, por cada consejo dado y cada plática hecha.

También, quiero agradecer a Senna por mostrarme que, “la luz debería enseñarnos a no temerle a la oscuridad” ya que, durante los momentos de mayor duda y miedo, entendí que no estaba solo y que siempre se puede superar.

Por último, agradezco a Isabel, Leandro, Irving, Samuel, Ariadne y Mario por sus consejos, experiencias y oportunidades que mediante la amistad y el Movimiento Scout me han brindado y me han preparado para “remar mi propia canoa”.

Índice

RESUMEN	1
1 PRESENTACIÓN	2
1.1 Planteamiento del problema	3
1.2 Objetivos.....	3
1.2.1 General.....	3
1.2.2 Específicos	3
1.3 Justificación	3
1.4 Estado del arte.....	4
2 Marco teórico.....	9
2.1 Sistemas Embebidos.....	9
2.2 Microcontrolador	12
2.2.1 Interrupciones.....	15
2.2.2 Subsistema de tiempos: PWM.....	17
2.2.3 Subsistema de Comunicación.....	19
3 Desarrollo del Prototipo.....	23
3.1 Estructura metálica.....	24
3.1.1 Arreglo de poleas de tensión	26
3.1.2 Carrete.....	27
3.1.3 Polea del codificador	29
3.2 Hardware	31
3.2.1 ESP32-DevKitC	31
3.2.2 Motor	35
3.2.3 Codificador rotativo incremental.....	37

3.2.4	Controlador de motor BTS7960.....	40
3.2.5	Fuente de alimentación	44
3.2.6	Circuito sensor de corriente	44
3.2.7	Switch fin de carrera.....	47
3.2.8	Elementos de protección	48
3.2.9	Diagrama a bloques completo del sistema.....	49
3.2.10	Tarjeta de circuito impreso.....	51
3.3	Software.....	53
3.3.1	Función Principal.....	53
3.3.2	Control	55
3.3.3	Comunicación Serial.....	60
3.3.4	Sensor de corriente	62
4	Resultados.....	63
4.1	Primer y segundo evento	64
4.2	Tercer evento	64
4.3	Cuarto y quinto evento	66
5	Conclusiones y trabajos futuros	67
6	Referencias	69
7	Anexos.....	71
7.1	Planos de la estructura metálica.....	71
7.2	Algoritmo del funcionamiento del sistema.....	83

Índice de tablas

Tabla 1 Frecuencias y resoluciones más usadas.....	34
Tabla 2 Características del motor CHW-GW4058-555.....	37
Tabla 3 Funcionamiento de los pines físicos del codificador	39
Tabla 4 Distribución y descripción de pines físicos del módulo BTS7960.....	42
Tabla 5 Relación de las conexiones al ESP32	50

Índice de ilustraciones

Ilustración 1 Diagrama a bloques del robot móvil (MR) para el monitoreo de la calidad del agua en tanques de acuicultura con función para dispensar alimento [3].	6
Ilustración 2 Características del producto 6950 vendido por YSI como solución al monitoreo en columnas de agua [8].	8
Ilustración 3 Ejemplo de la arquitectura de un sistema embebido [14]	11
Ilustración 4 Diagrama a bloques de un microcontrolador genérico [15].	12
Ilustración 5 Arquitecturas del CPU [15].	14
Ilustración 6 Máquina de estados del orden secuencial de un programa [15].	16
Ilustración 7 Diagrama a bloques de operación del subsistema de tiempos [15].	17
Ilustración 8 Diagrama a bloques del contador de salida del sistema [15].	18
Ilustración 9 Señales de PWM [15].	19
Ilustración 10 Comunicación serial asíncrona [15].	20
Ilustración 11 Paquetes de datos en la transmisión serial.	21
Ilustración 12 Paquete de datos con bit de paridad.	22
Ilustración 13 Elementos críticos para el funcionamiento del prototipo.	25
Ilustración 14 Vistas laterales y medidas en milímetros del prototipo	25
Ilustración 15 Vista superior del arreglo de poleas tensoras	26
Ilustración 16 Estructura móvil absorbiendo la máxima pérdida de tensión del cable.	27
Ilustración 17 Diseño del carrete	28
Ilustración 18 Localización de la polea del codificador respecto a los demás elementos de la estructura	30
Ilustración 19 Vista superior de la estructura, enfocando el acople, los baleros y el eje.	31
Ilustración 20 Mapa de ubicación y propósito de los pines del ESP32-DevKitC [9]	32
Ilustración 21 Arquitectura del controlador LED PWM del ESP32 [19].	33
Ilustración 22 Diagrama de configuración del Controlador LED PWM [19]	34
Ilustración 23 Motor utilizado de 430rpm.	36
Ilustración 24 Codificador rotativo incremental con resolución de 400ppr	38
Ilustración 25 Circuito del divisor de tensión de 5V a 3.3V	39

Ilustración 26 Principio de funcionamiento del transductor óptico incremental [20].....	40
Ilustración 27 Diagrama de bloques funcionales del integrado BTS7960 [21]	41
Ilustración 28 Distribución de los pines físicos del módulo BTS7960	43
Ilustración 29 Diagrama de interconexión del módulo controlador BTS7960.....	44
Ilustración 30 Diagrama de conexión de la resistencia sensor respecto al controlador de motores BTS7960	45
Ilustración 31 Circuito sensor de corriente completo conectado al ADC1 del ESP32	47
Ilustración 32 Diagrama a bloques de los componentes funcionales del sistema.....	49
Ilustración 33 Diagrama esquemático de la tarjeta de interconexión	51
Ilustración 34 Previsualización del ruteo de las pistas y del solder mask de la placa.....	53
Ilustración 35 Diagrama de flujo del funcionamiento principal del prototipo	54
Ilustración 36 Algoritmo de cálculo de la velocidad.....	56
Ilustración 37 Función "control_subida()"	58
Ilustración 38 Función "control_bajada()"	59
Ilustración 39 Función "comunicación_serial()"	61
Ilustración 40 Función "senzar_corriente()"	62
Ilustración 41 Prototipo montado sobre el pozo a monitorear.....	63
Ilustración 42 Arreglo de poleas tensoras.....	65

RESUMEN

El Laboratorio Nacional para la Resiliencia Costera de la Unidad de Ingeniería de la Universidad Nacional Autónoma de México Campus Sisal, dentro de sus diversos proyectos de investigación trabaja en el monitoreo de la salinidad en el manto acuífero de la costa del estado de Yucatán. Debido a esto, existen pozos perforados de aproximadamente 20m de profundidad en diferentes ubicaciones del estado.

Debido al número de pozos de interés, el LANRESC requiere de un prototipo que permita al investigador (usuario/operador) subir y bajar la sonda a una velocidad calculada previamente; así como la posibilidad del funcionamiento automático, ya que los investigadores requieren monitorear el comportamiento del manto acuífero durante el día y la noche.

Este trabajo de tesis implementa las tecnologías existentes para dar una solución personalizada al problema que presentan los investigadores, por esto, en el estado del arte se revisan las aportaciones científicas en sistemas robóticos de monitoreo de cuerpos de agua y en el capítulo dos se definen los conceptos de microcontrolador, sistemas embebidos, interrupciones, subsistema de tiempos, PWM y comunicación serial para desarrollar este proyecto.

En el tercer capítulo se diseñan y definen los elementos que integran el prototipo propuesto en este documento. Este proyecto se divide en tres áreas que funcionan en conjunto: la estructura metálica, el hardware de control (tarjeta electrónica y componentes de potencia) y el software (lógica de funcionamiento).

En el capítulo cuatro se presentan los resultados del prototipo montado sobre un pozo de 18m de profundidad ubicado dentro de las instalaciones del Campus Sisal de la UNAM; donde se obtiene que el sistema logra operar de forma automática aún después de una interrupción en la alimentación. De igual forma, se obtiene que durante la subida de la sonda se tiene una velocidad media de 0.055m/s propuesta por los investigadores.

1 PRESENTACIÓN

La obtención de la salinidad en el manto acuífero cerca de las costas del estado de Yucatán permite conocer ciertos indicadores de contaminación humana, así como el historial del comportamiento de esa zona; en especial cerca de los manglares de la región debido a su importancia y al impacto que tiene este ecosistema en el medio ambiente. Por esto, la Unidad de Ingeniería de la Universidad Nacional Autónoma de México en el Campus Sisal en Yucatán, México; tiene diversos pozos ubicados en diferentes zonas del área costera de Sisal ya que estos le permiten conocer a los investigadores el comportamiento del ecosistema y generar estadísticas de interés para los diferentes trabajos de investigación.

Para poder obtener estos datos necesitan introducir sondas en los pozos para realizar las mediciones, sin embargo no cuentan con un equipo capaz de subir y bajar una sonda con sensores de manera automática o con precisión; haciendo así que la toma de muestras sea una tarea complicada y la depuración de las muestras tome demasiado tiempo, ya que si bien la sonda obtiene los datos de profundidad con la variable de presión; sería de mayor utilidad para los investigadores obtener estos resultados de manera automática. Además, ellos necesitan que el equipo trabaje de forma continua para poder observar el comportamiento durante el día y la noche, así como durante sucesos climáticos de interés.

Por esto, se propone un prototipo mecatrónico que automatice la acción de sumergir y hacer emerger la sonda a una velocidad preestablecida por el usuario, para que ésta pueda tomar las muestras necesarias cada cierto tiempo durante el recorrido.

Este proyecto es la respuesta a la necesidad que presenta el Laboratorio Nacional de Resiliencia Costera (LANRESC) del Laboratorio de Ingeniería y Procesos Costeros del Instituto de Ingeniería de la Universidad Nacional Autónoma de México (I.I.C. – U.N.A.M.) la cual es presentada por el Departamento de Ingeniería Eléctrica y Electrónica (D.I.E.E.) del Tecnológico Nacional de México/Instituto Tecnológico de Mérida (TecNM/ITMérida) en colaboración con el Laboratorio de Robótica e Industria 4.0 de la Facultad de Ingeniería de la Universidad Autónoma de Yucatán (U.A.D.Y.).

1.1 Planteamiento del problema

En el LANRESC se requiere de un dispositivo que permita sumergir y hacer emerger de manera automática las sondas YSI EXO 1 y 2, para que estas realicen mediciones. Adicionalmente, el usuario debe poder establecer los parámetros del experimento de muestreo tales como, la velocidad de movimiento de la sonda y los periodos de tiempo y distancias de funcionamiento, garantizando la seguridad de las sondas en las diferentes etapas del recorrido. Esta necesidad se presenta ya que las opciones semejantes de ámbito comercial que se pueden conseguir mediante compañías fabricantes de equipos de instrumentación para investigación se encuentran en el extranjero y a costos sumamente altos.

1.2 Objetivos

1.2.1 General

Esta tesis tiene como objetivo diseñar un prototipo que controle la posición vertical de las sondas YSI EXO 1 y 2 dentro de un pozo, y que permita al usuario configurar el funcionamiento automático del sistema.

1.2.2 Específicos

1. Diseñar un sistema electrónico para controlar el ascenso y descenso de una sonda con sensores.
2. Implementar un software de control de lazo cerrado que reciba del usuario la velocidad máxima deseada, la profundidad máxima, las repeticiones de operación y el tiempo entre repeticiones.
3. Diseñar una estructura mecánica que permita integrar el hardware y software para levantar un peso máximo de 5kg a una velocidad de 0.05m/s.

1.3 Justificación

En el LANRESC del Instituto de Ingeniería de la UNAM ubicado en la zona costera de Sisal, Yucatán; existen diversos trabajos de investigación dedicados al estudio integral de

las zonas costeras, para esto requieren obtener datos provenientes de situaciones climáticas, así como de los comportamientos marinos de interés dentro de los temas de investigación de la institución. Dentro de estos trabajos, se realiza el monitoreo del manto acuífero a través de pozos de, aproximadamente, 18 metros de profundidad; y estos están localizados en diversas áreas alrededor de la costa yucateca con la finalidad de poder estudiar indicadores de actividad humana en estas zonas y el impacto ambiental que estas representan.

Los investigadores obtienen estos datos al analizar diversas variables dentro de los pozos como son la conductividad eléctrica, presión barométrica, pH, temperatura, entre otras. Para esto, utilizan las sondas EXO 1 y EXO 2 de la marca YSI, las cuales son equipos especializados para medir y monitorear la calidad del agua en diferentes condiciones.

Durante los tiempos de medición, los investigadores necesitan estar presentes en todo momento sumergiendo la sonda dentro del pozo, lo que hace que estos procesos carezcan de precisión y sea difícil recolectar los datos durante sucesos críticos climáticos, tales como temporales y tormentas; ya que es en estos fenómenos donde ellos necesitan monitorear el comportamiento del manto acuífero. A su vez, el LANRESC cuenta con pozos en zonas del manglar de difícil acceso continuo, haciendo que la adquisición de datos sea una tarea que les consume mucho tiempo.

Por lo antes mencionado, este trabajo de tesis pretende facilitar a los investigadores el proceso de obtención de los datos al hacer un prototipo automático que sumerja y haga emerger la sonda dentro del pozo, de esta forma, se puede dejar operativo el sistema durante varias horas consecutivas permitiendo la adquisición de datos continuos.

1.4 Estado del arte

Alrededor del mundo se han desarrollado, desde 1940 [1], diversos proyectos de monitoreo del agua en ríos, presas, costas y otros cuerpos de agua para generar información confiable que pueda interpretarse para la toma de acciones con el fin de preservar este valioso recurso. Sin embargo, muchas veces el monitoreo que se ha realizado es poco confiable e impreciso con respecto a la variación del tiempo, de aquí la importancia de adoptar soluciones que contemplen plataformas robóticas que permitan automatizar con precisión este proceso [2].

En la actualidad se han desarrollado diferentes sistemas para el análisis de cuerpos de agua donde se han involucrado soluciones autómatas mediante equipo robótico para el monitoreo de la calidad de esta. Los autores en [2] hacen un análisis de estos sistemas o programas, siendo el primero conocido como “RTRM” (Real Time Remote Monitoring), el cual tiene como propósito la adquisición remota de datos para el análisis de la calidad del agua. El segundo que se analiza es el “NEMRP” (Neuse Estuary Monitoring and Research Program) y consta de 4 plataformas autónomas de monitoreo remoto. Este programa provee un sistema basado en web que almacena la información en bases de datos y está disponible para consulta. Y el MARVIN (Merhab Autonomous Research Vessel for In-Situ Sampling) que es un vehículo marino no tripulado para el monitoreo en tiempo real de diferentes variables como la temperatura, corriente, conductividad, características relativas a las propiedades ópticas y la cantidad de nitrato en agua [2]. De igual forma, los autores proponen un vehículo superficial no tripulado (Surface Unmanned Vehicle) utilizando como sistema embebido una Raspberry Pi B, sondas de la marca Atlas Scientific y un robot-velero (N-Boat – The Sailboat Robot), el cual monitoreará en tiempo real diferentes lagos de Brasil.

Los autores en [3] diseñan una guía, utilizando un lenguaje de modelado unificado (UML), para automatizar el monitoreo del agua y dispensar el alimento para peces en tanques de acuicultura, según el horario de alimentación de los especímenes. Logran hacerlo utilizando sondas conectadas a un robot móvil (Ilustración 1) instalado en un monorriel sobre los contenedores de agua, el cual hará las mediciones cada vez que se soliciten los parámetros mediante un servidor web. Dentro de este prototipo utilizan diferentes tipos de comunicación inalámbrica entre los servidores de internet y el robot móvil para garantizar el monitoreo y aplicación remota.

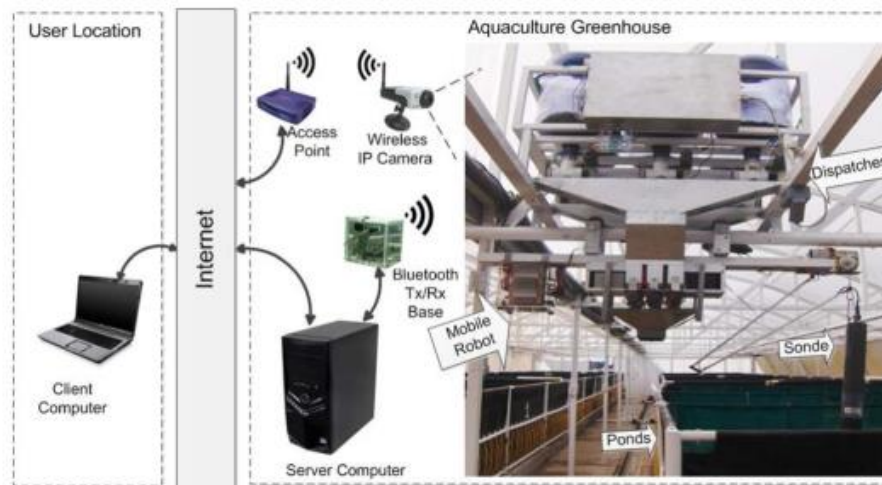


Ilustración 1 Diagrama a bloques del robot móvil (MR) para el monitoreo de la calidad del agua en tanques de acuicultura con función para dispensar alimento [3].

En el estudio presentado en [4] se utilizan los datos recopilados por boyas y estaciones climáticas en nueve playas para uso recreativo de Chicago para desarrollar un modelo empírico que permita informar a la población las condiciones de contaminación fecal del agua en esas zonas. Para lograrlo, los autores desarrollaron un programa ejecutable que recibe la información en un “host service” de las sondas enviadas por telemetría utilizando la red celular. Posteriormente este software calcula los resultados utilizando el modelo empírico propuesto, ya que este presentó mayor precisión que el modelo de persistencia analizado.

En el artículo [5] se describe el proyecto piloto “Bristol Is Open (BIO)”, operado por el Concejo de la Ciudad de Bristol, del monitoreo con multi sensores y cámaras de video del puerto flotante de dicha ciudad; el cual prueba el concepto de una red de sensores inalámbricos (Wireless Sensor Network) basada en tecnología Wi-Fi como solución para el monitoreo de la calidad del agua en una ciudad inteligente. Este sistema consiste en el proceso completo de adquisición, transmisión, almacenamiento y visualización de datos utilizando cómputo en la nube desarrollado en plataformas de código libre (IoT). Además, los autores analizan la evolución del monitoreo del agua junto con los retos que se han solucionado conforme los avances tecnológicos, siendo la integración de la robótica y WSN en los vehículos autónomos submarinos y de superficie los aportes más recientes al estado del arte.

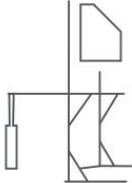

Al mencionar la implementación del monitoreo con la robótica, es necesario revisar el estado del arte existente de los diferentes métodos de control de los motores de corriente directa. Se encuentra que los autores en [6] desarrollaron tres experimentos. “El primero es controlar la velocidad del motor a través de un teléfono inteligente, el segundo es la relación entre la fuerza electromotriz y la velocidad del motor DC y el tercero es el monitoreo de la velocidad del motor utilizando la plataforma de Thingspeak y las aplicaciones de Blynk” [6]. De igual forma, implementaron un algoritmo de control PID de la medición de la fuerza electromotriz del motor para que modifique la señal de PWM generada por el microcontrolador Node-MCU. Además, utilizaron energía fotovoltaica como fuente principal de alimentación para la recarga de la batería instalada en el sistema.

Los autores en [7] proponen un controlador de velocidad basado en una red neuronal artificial (ANN), la cual “es un paradigma de proceso de información que está inspirado por la forma biológica en la que el sistema nervioso, como el cerebro, procesa la información”. Para comprobar la propuesta revisan el desempeño de un motor DC con diferentes cargas al aplicarle tres controladores diseñados: PI, PID y un Controlador Neuronal Artificial (ANC). El ANC provee el mejor desempeño en los modelados realizados en el software MATLAB/Simulink.

Los sistemas robóticos que se mencionan anteriormente no solucionan el problema particular de esta tesis debido a que estos sistemas fueron diseñados para mar abierto o utilizan vehículos autónomos para realizar las mediciones. Por esto, es necesario incluir en el estado del arte el equipo “6950 Fixed Vertical Profiler” (Ilustración 2) de la empresa YSI [8], ya que este producto es el más cercano al prototipo propuesto en este trabajo. El 6950 es parte de un sistema que provee la recolección de datos automatizados a través de una columna de agua en diferentes condiciones tales como presas, ríos, lagos, pozos, costas, etcétera. También permite, en ciertas situaciones, transmitir la información de los sensores de la sonda mediante telemetría. Este sistema puede ser configurado para que sumerja y haga emerger una sonda dentro de una columna de agua en diferentes intervalos de tiempo y profundidad configurables por el usuario.

Fixed Vertical Profiler

- Ideal for drinking water reservoirs
- Attach to a bridge, building, or railing
- NEMA4X enclosure



[Learn more](#)

Ilustración 2 Características del producto 6950 vendido por YSI como solución al monitoreo en columnas de agua [8].

Sin embargo, el 6950 Fixed Vertical Profiler, al ser un producto fabricado y vendido a través de YSI Incorporated con base en Ohio, Estados Unidos; así como los elevados costos de envío, importación y del mismo producto, hace que el acceso al 6950 en México sea difícil. Además, el LANRESC tiene la intención de establecer una red de sensores inalámbricos en cada pozo que tiene alrededor de la costa yucateca para el monitoreo de estos, entonces vuelve irrealizable esta Red de Sensores Inalámbricos (WSN) si el producto tiene un elevado costo de compra y mantenimiento.

2 Marco teórico

Debido a las características de este trabajo de investigación, se presenta un marco conceptual, ya que no es la finalidad de este comprobar las teorías de otros autores si no la aplicación de conceptos para la elaboración de una solución específica en el contexto de la ingeniería electrónica; por lo que en este apartado se definirán temas como los sistemas embebidos y el control por PWM, debido a que estos son los temas teóricos que se deben tomar en cuenta para realizar este proyecto de tesis.

2.1 Sistemas Embebidos

Según diversos autores dentro del estado del arte; se puede definir a los sistemas embebidos como:

- “La denominación aplicable a los equipos electrónicos que incluyen procesamiento de datos, pero que, a diferencia de una computadora de propósito general, están diseñados para satisfacer una función específica” [9]
- “Un sistema embebido es un sistema electrónico que contiene un microprocesador o microcontrolador; sin embargo, no pensamos en ellos como un computador” [10]
- “Un sistema embebido es un sistema cuya función principal no es computacional, pero es controlado por un computador integrado. Este computador puede ser un microcontrolador o un microprocesador. La palabra embebido implica que se encuentra dentro del sistema general, oculto a la vista, y forma parte de un todo de mayores dimensiones” [11]

En síntesis, un sistema embebido es un conjunto de hardware y software diseñado para un uso específico que requiera de procesamiento de datos, tal como pueden ser los computadores de un automóvil, de una lavadora o de un refrigerador inteligente; sin embargo, este computador no está disponible para que el usuario final pueda programarlo. Retomando el ejemplo del refrigerador; este realiza funciones específicas como activar o desactivar el

compresor dependiendo del sensor de temperatura, ya que está programado para que al llegar a ciertas condiciones internas de la nevera se modifique la variable de control por estímulos externos [9].

Por lo que se puede definir que es un dispositivo de análisis de datos, monitoreo y/o de control que consta de un microcontrolador o microprocesador diseñado para realizar funciones únicas y especializadas, que suelen ser cíclicas y estas son definidas mediante software por el programador.

Por lo que se puede definir que es un dispositivo de análisis de datos, monitoreo y/o de control que consta de un microcontrolador o microprocesador diseñado para realizar funciones únicas y especializadas, que suelen ser cíclicas y estas son definidas mediante software por el programador.

Pérez en [12] describe las partes que debe contener un sistema embebido; además que estos componentes principales se pueden encontrar dentro de las definiciones de los otros tres autores, sin embargo, se enlistan a continuación los tres componentes principales según el autor en [12]:

- Hardware.
- Software/Aplicación principal.
- Sistema operativo que supervisa las aplicaciones y gestiona la ejecución de procesos.

El hardware de estos dispositivos está diseñado para realizar tareas específicas, por ejemplo, la placa de control de un refrigerador o la de una lavadora; si bien pueden tener el mismo microcontrolador tienen diferentes componentes y periféricos específicos para el funcionamiento único de cada dispositivo electrónico. Además, estos (microcontroladores) cuentan con fuertes limitaciones los cuales pueden ser [13]: costo, tamaño, desempeño y consumo de energía; ya que suelen ser utilizados en implementaciones donde solo son una parte del todo, cuentan con espacio reducido y/o necesitan gestionar eficientemente los recursos energéticos.

De igual forma, muchos de estos sistemas deben poder reaccionar en tiempo real a estímulos que ocurran en el ambiente mediante sensores especializados en dichas áreas; por

lo que se necesita de un hardware que procese la información de manera rápida y sin retrasos, ya que si esto llegase a ocurrir pudiese ocasionar una falla fatal en todo el sistema poniendo en riesgo al mismo sistema o a los usuarios.

A continuación, se presenta en la Ilustración 1 las partes de un sistema embebido típico [14]; esta muestra un punto de acceso inalámbrico y su arquitectura de hardware representada en un diagrama a bloques. Los elementos que lo conforman son [12]:

- Procesador RISC (Reduced Instruction Set Computer), el cual es el procesador central del sistema basado en arquitectura de 32 bits.
- Memoria FLASH, esta es utilizada para almacenar los datos necesarios para la ejecución de programas configurados por el programador.
- Memoria Principal, en esta se almacenan valores temporales para la ejecución de los programas y suele tener cientos de megabytes (MB).

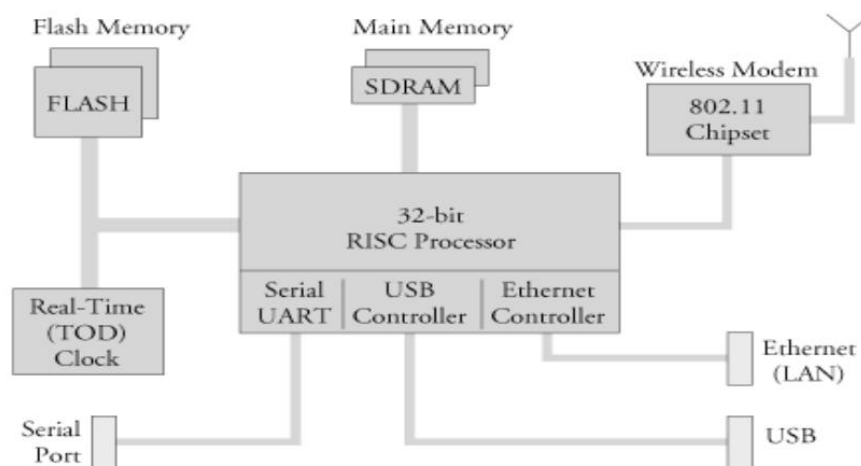


Ilustración 3 Ejemplo de la arquitectura de un sistema embebido [14]

Por lo tanto, un sistema embebido es un dispositivo diseñado, desde la concepción de la idea, para realizar funciones específicas ya que los elementos que lo componen suelen estar incluidos dentro de la placa de evaluación o dentro de System on Chip (SoC), por lo que no pueden ser modificables salvo por ciertas excepciones. Estos embebidos igual pueden contar con sensores, puertos y protocolos de comunicación para facilitar el uso de estos, tal como

sucede con las placas de desarrollo, evaluación y de educación; donde su finalidad es ayudar al usuario a comprender, diseñar y evaluar sistemas embebidos.

2.2 Microcontrolador

En su forma más fundamental, mencionado en [15], un microcontrolador es un sistema completo de cómputo en un solo chip de procesamiento, el cual contiene todos los subsistemas de un sistema de cómputo más grande, pero a tamaño y potencia escala. Estos tienen la capacidad de contar con puertos de entrada y salida, sistema de tiempo, memoria, una Unidad de Aritmética y Lógica (ALU por sus siglas en inglés) que provee la capacidad de ejecutar procesos lógicos y aritméticos; también este chip tiene la capacidad de generar señales de control en los puertos de salida de este. Todas estas características varían dependiendo de las funciones específicas que este vaya a ejecutar.

Existe gran variedad de microcontroladores según su rango en bits, empezando por pequeños de 4 bits con características limitadas en sus componentes, así como de procesadores de 32 bits. A continuación, se presenta un diagrama a bloques [15] de un microcontrolador genérico en la Ilustración 4

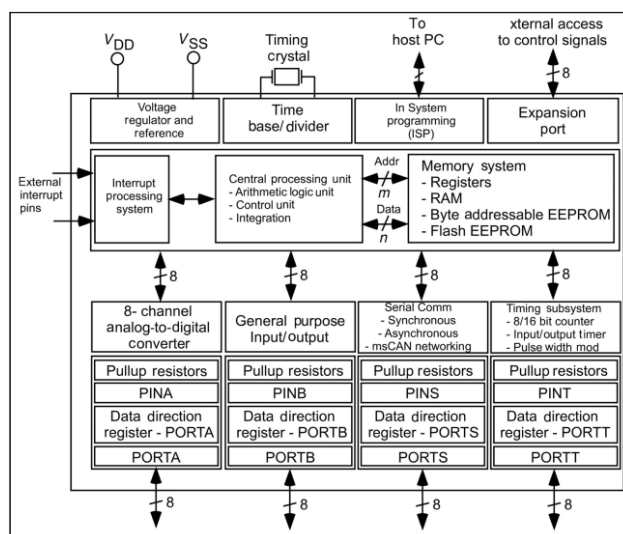


Ilustración 4 Diagrama a bloques de un microcontrolador genérico [15].

Muchos microcontroladores están equipados con los subsistemas que se muestran en la ilustración superior. Los puertos son usados para las interconexiones de elementos externos con el microcontrolador, como indicadores LED, sensores, relevadores, actuadores, interfaces, entre otros elementos necesarios para la aplicación deseada. En la mayoría de los casos, los puertos son bidireccionales y también pueden alternar funciones, es decir, pueden tener más de una función dedicada para ese puerto o en ciertas terminales de ellos, por ejemplo, pueden utilizarse para la conversión analógica a digital, comunicaciones seriales y del sistema de tiempos.

Además, los microcontroladores pueden interrumpir la secuencia lógica del código que está ejecutando mediante eventos externos utilizando los pines de interrupción dedicados. Esto permite al microcontrolador responder a eventos de alta prioridad, todo depende de la aplicación de este, por ejemplo, el botón de paro de emergencia en una bomba de gasolina debe detener la bomba de distribución para evitar accidentes.

De igual forma, el microcontrolador puede ser programado mediante la interfaz “In System Programming” (ISP) el cual utiliza a una computadora como host personal. En las placas de desarrollo y de educación suelen tener conexiones USB para esta función debido a la facilidad que estos presentan. Así mismo, el microcontrolador necesita de un cristal oscilador externo o resonador externo que le provea de tiempo base al chip [15].

La arquitectura básica de la unidad central de procesamiento incluida dentro del microcontrolador es un complejo circuito secuencial, la cual tiene como función principal ejecutar los programas que están almacenados en la Flash EEPROM (Electrically Erasable Programmable Read Only Memory). Un programa no es más que una serie de instrucciones secuenciales que ejecutan una o varias tareas diseñadas por el programador. Este se carga mediante un equipo de cómputo al microcontrolador una vez que se haya interpretado todas las líneas de código por el compilador, el microcontrolador se reinicia y se convierte en un sistema de procesamiento independiente para las tareas especificadas. En la Ilustración 5 [16] se observan los diferentes tipos de arquitecturas de CPU:

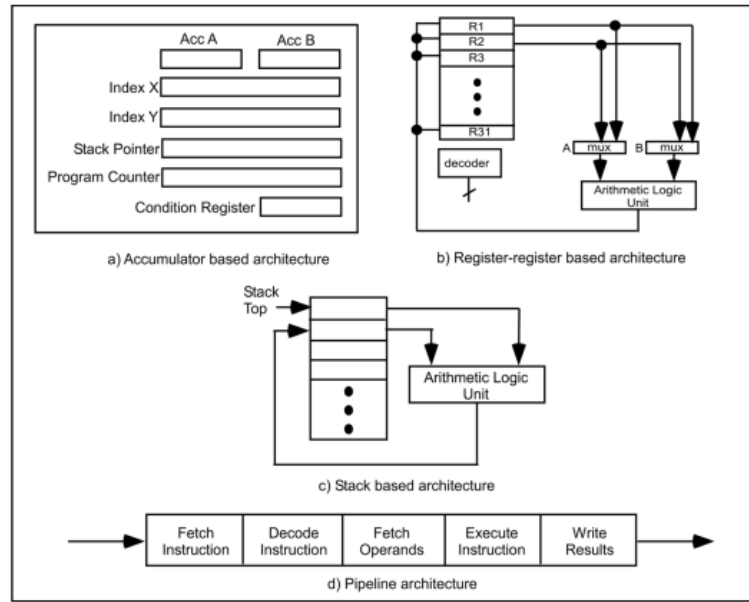


Ilustración 5 Arquitecturas del CPU [15].

Los microcontroladores igual se diferencian unos de otros por la arquitectura diseñada de su CPU, estas se muestran a continuación de la lista presentada en [15]:

- Arquitectura basada en acumuladores: en este bloque las instrucciones inician y terminan en registros especiales diseñados para esta tarea, estos se llaman “acumuladores” (A y B). Usualmente una operación es realizada en la cual un operando se encuentra en un acumulador y el otro se obtiene de la memoria. Esta arquitectura tiene la habilidad de ejecutar instrucciones bastante complicadas. La memoria corre a una velocidad menor que la del procesador principal, por lo que el procesador debe reducir su velocidad para acomodar la obtención de los operandos desde la memoria.
- Arquitectura basada en registros: ambos operandos están almacenados en registros que son colocados junto con la unidad central de procesamiento. El resultado de una operación dada se almacena en el registro. Debido a que el CPU y los registros operan a la misma velocidad, el procesador no necesita disminuir la velocidad de su funcionamiento y tanto la lectura como la escritura de datos ocurren en procesos de segundo plano.

- Arquitectura basada en pilas: ambos operandos y la operación a ser desarrollada son almacenadas en la pila. La pila puede estar basada en registros dedicados o en una partición especial de la memoria de acceso aleatorio (RAM)
- Arquitectura de “pipeline”: esta arquitectura consiste en separar los subsistemas de los hardware llamados “estadios a obtener” de una instrucción de la memoria, decodifica la instrucción, obtiene instrucciones de la memoria o registros, ejecuta comandos, entre muchas otras funciones. Usualmente, las instrucciones en el procesamiento de “pipeline” son instrucciones simples fáciles de implementar en un solo estado.

Muchos microcontroladores tienen registros complementarios diseñados dentro del mismo bloque de registros, ya que estos sirven de interfaz entre el usuario y los diferentes subsistemas dentro del microcontrolador. Cada registro consiste en una colección de flip-flops que pueden ser configuradas con un uno o cero lógicos, esto permite configurarlos como interruptores configurables y así acceder a las diferentes funciones que presentan. Dentro de estos registros se encuentran los de la comunicación serial y del subsistema de tiempos, ya que es en este último donde se encuentran los contadores y las señales de PWM.

2.2.1 Interrupciones

Para comprender las interrupciones, se debe mencionar que un programa consiste en un número de líneas de código (instrucciones) que son ejecutadas en orden secuencial, es decir, una después de la otra. El orden normal de ejecución del código [16] se muestra en la Ilustración 6:

- Fetch: la unidad de control solicita instrucciones de la memoria principal que esta almacenada en la ubicación de memoria indicada por el programa.
- Decode: una vez recibidas las instrucciones, estas se decodifican en los registros de instrucciones.

- Execute: las instrucciones previamente decodificadas se ejecutan de manera secuencial por el hardware correspondiente. Cuando termina este, se reinicia la máquina de estados.
- Interrupt: ocurre una interrupción en la secuencia lógica del código.

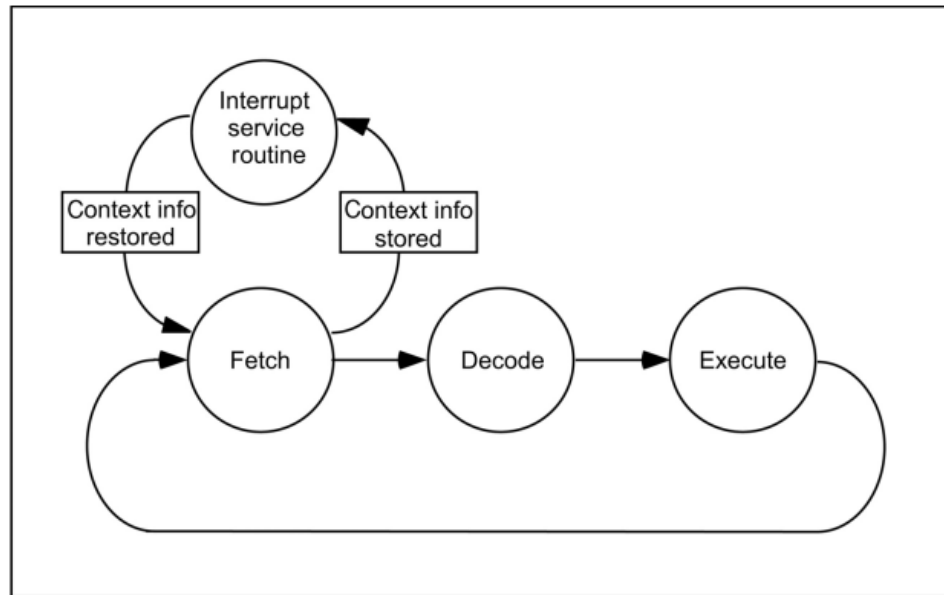


Ilustración 6 Máquina de estados del orden secuencial de un programa [15].

Cuando ocurre un evento de alta prioridad, la secuencia normal puede interrumpirse para ejecutar una tarea corta y regresar a la operación que estaba realizando previamente. Estos eventos pueden ser externos o internos al microcontrolador. Se debe hacer énfasis en que la interrupción temporal suspende la operación de la secuencia normal de los eventos y no puede regresar al orden secuencial hasta que todas y cada una de las instrucciones se hayan ejecutado propiamente, por lo que se recomienda que la rutina de interrupción sea corta y concisa.

Existe gran variedad de interrupciones disponibles en los microcontroladores, algunas son generadas por fallas o errores en el hardware o software, otras están disponibles para el programador o diseñador del sistema y hay dos que son de mucha utilidad [15], las cuales son las interrupciones en tiempo real (IRT) y las solicitudes de interrupciones externas (IRQ). La primera provee una interrupción de forma regular in el flujo del programa principal, mientras

que la segunda puede ser configurada por el usuario mediante un evento programado, por ejemplo, la detección de un flanco de subida en cierto pin.

2.2.2 Subsistema de tiempos: PWM

Las dos finalidades del sistema de contadores en un microcontrolador son [15]: detectar y capturar eventos de tiempo externos; y generar eventos de tiempo para controlar y acceder a sistemas externos. Esto ocurre debido a que para este sistema se utiliza un reloj que actualiza los contenidos de registros especiales denominados “contadores libres”, el trabajo de estos es contar de manera incremental cada vez que ocurra un flanco de subida/bajada de una señal de reloj. En la Ilustración 7 se observa la operación del subsistema, donde todo depende del cristal u oscilador físico del microcontrolador, pasa por el reloj del sistema al contador libre y de ahí se puede utilizar tanto la característica de entrada como de salida.

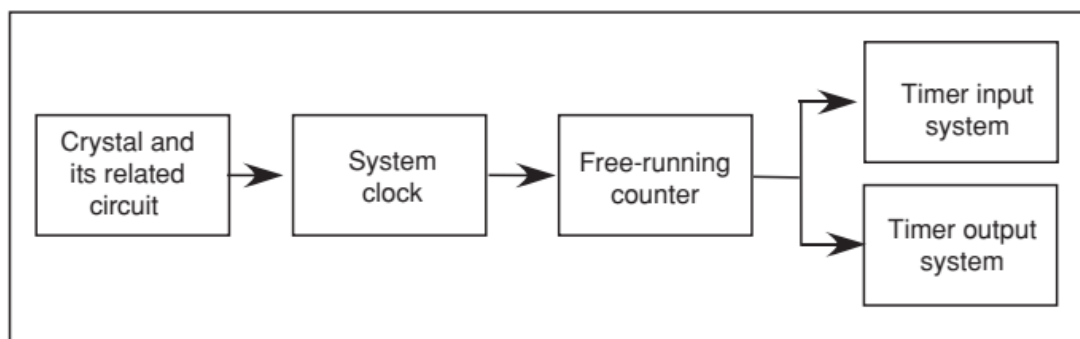


Ilustración 7 Diagrama a bloques de operación del subsistema de tiempos [15].

Siguiendo el diagrama a bloques [15], la salida del contador tiene la finalidad de generar señales que permitan controlar dispositivos externos. Un evento significa un cambio lógico en los pines de salida del microcontrolador en un tiempo específico. En la Ilustración 8 se observa un ejemplo de esta característica, donde el contador se compara con los datos almacenados en un registro especial para así ejecutar los eventos programados y asignarlos al pin físico o para ejecutar una interrupción de salida del sistema.

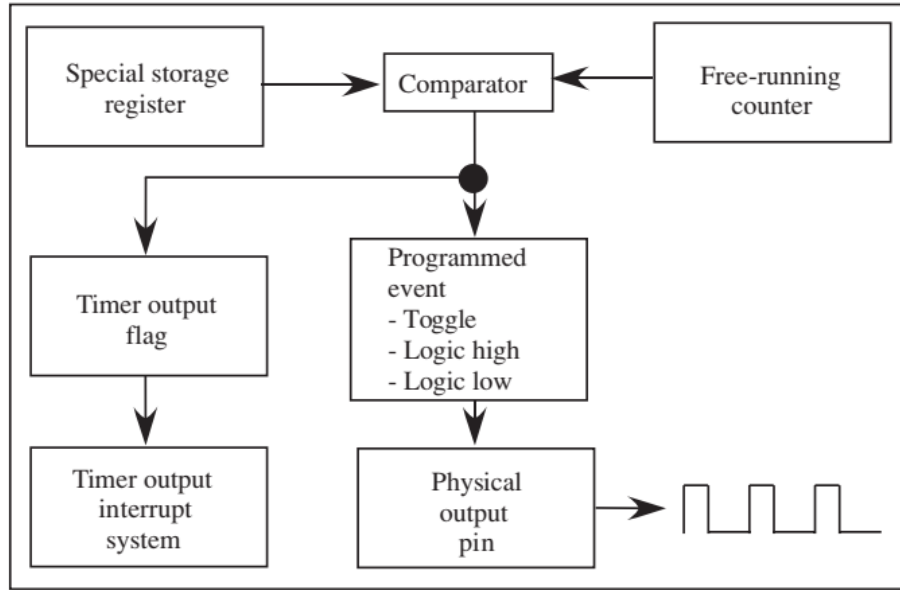


Ilustración 8 Diagrama a bloques del contador de salida del sistema [15].

Dentro de las aplicaciones de este subsistema están las señales de PWM (Modulación por Ancho de Pulso) son usadas frecuentemente para controlar la velocidad de un motor. La señal digital de PWM es convertida en un valor efectivo de corriente directa por el mecanismo de inercia del motor, así como las características de inductancia en los filtros pasivos inherentes presentes en el motor [15]. El ciclo de trabajo hace referencia al ancho del pulso positivo. El concepto de PWM se ejemplifica en la Ilustración 9 y se observa que se requiere saber del periodo el cual está dado por:

$$T = \frac{1}{f} \quad (1)$$

Donde T es el periodo y f es la frecuencia de la señal de salida, mientras que el ciclo de trabajo está dado por:

$$\text{Ciclo de trabajo} = \left(\frac{\text{tiempo encendido}}{T} \right) (100\%) \quad (2)$$

Cuando se menciona el “tiempo encendido” es la cantidad de tiempo que la señal está en su estado lógico alto.

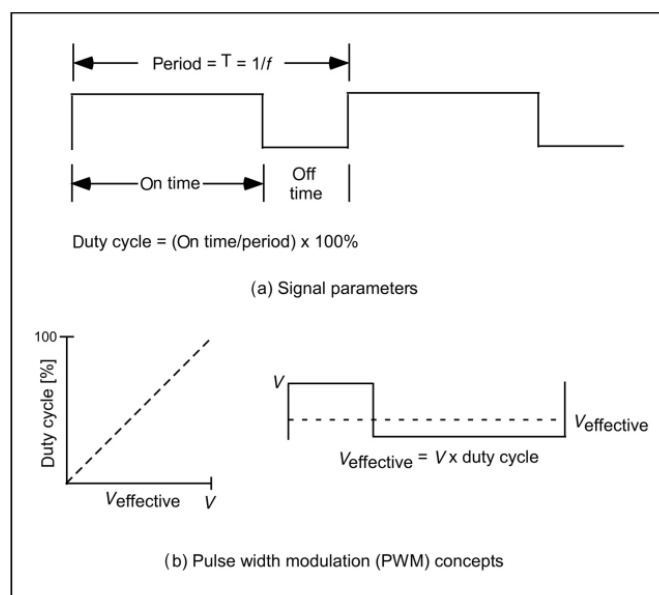


Ilustración 9 Señales de PWM [15].

De igual forma, esta técnica se utiliza en la transmisión de señales analógicas, las cuales la señal portadora es digital; porque se modifica el ciclo de trabajo de una señal periódica (en este caso en particular, será una señal cuadrada), ya sea para transmitir la información utilizando un canal de comunicaciones o para controlar la cantidad de energía que se le enviará a una carga.

2.2.3 Subsistema de Comunicación.

Existen dos tipos de comunicación entre sistemas, según su arquitectura puede ser paralelo y/o serial. La diferencia entre una y otra es la cantidad de canales (cables de bus) son necesarios para la transmisión, claro está que la paralela es mucho más rápida debido a que la velocidad de transferencia es igual que la del transmisor, sin embargo, el costo por la fabricación del hardware, así como el tamaño, aunque este tipo de comunicación se utiliza para transmisiones de corta distancia [15].

2.2.3.1 Comunicación Serial

Para efectos de este trabajo se profundiza en la comunicación serial, ya que es la que utiliza el protocolo de comunicación entre el microcontrolador con el equipo host. La comunicación serial es un protocolo de comunicación entre dispositivos que se incluye de manera estándar en prácticamente cualquier computadora [17]. Este concepto permite la transmisión y recepción de bit a bit de un byte completo.

Existen dos tipo de comunicación serial la síncrona y la asíncrona, donde en la primera el reto se presenta en mantener la sincronía entre el transmisor y receptor ya que usualmente se suele transmitir la señal de reloj (CLK) a la que funcionará; mientras que en la comunicación asíncrona usa un protocolo de bits de inicio y de parada para sincronizar al transmisor y receptor, esto presenta una ventaja en materia de costos pero este método es más lento que el primero debido a los bits de cabecera (start/stop).

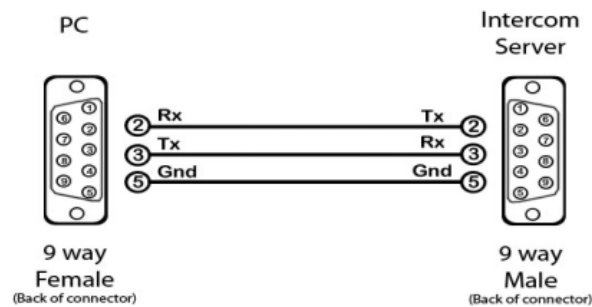


Ilustración 10 Comunicación serial asíncrona [15].

La comunicación asíncrona utiliza tres líneas de transmisión, tal como se muestra en la Ilustración 10: referencia (GND), transmisión (Tx) y recepción (Rx), debido a las características de esta, es posible enviar datos por una línea mientras se reciben datos por otra. Las características más importantes de la comunicación serial son la velocidad de transmisión (“baud rate”), el número de bits de datos, el número de bits de paro y si cuenta o no con el bit de paridad. A esta comunicación igual se le conoce como UART (transmisor-receptor asíncrono universal) dado que este presenta las mismas características que dicha comunicación; un ejemplo se expone en la Ilustración 11

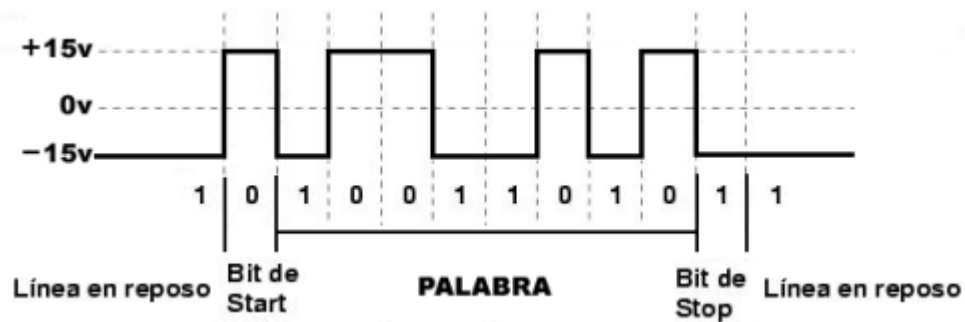


Ilustración 11 Paquetes de datos en la transmisión serial.

Para comprender mejor el concepto y el funcionamiento, se deben revisar los siguientes conceptos expuestos en [15], los cuales mencionan ciertas características importantes de este tipo de comunicación, como lo son los modos de operación, ya que con base en estos se construyen las líneas de transmisión:

- Modo Simplex: en este modo, la comunicación serial se lleva a cabo mediante la transmisión de datos en una sola dirección a la vez
- Modo Semidúplex: en este modo cada lado transmite, pero solo uno a la vez.
- Modo Duplex: en este modo, los datos pueden ser transmitidos y recibidos de ambos extremos de la línea de transmisión al mismo tiempo.
- Velocidad de transmisión: se le conoce como “baud rate” y este indica el número de bits por segundo que se transfieren, por ejemplo, algunos microcontroladores trabajan a 9600 o 115200.
- Tiempo de bit: es el tiempo requerido para transmitir o recibir un solo bit.
- Código de línea serial: es un mecanismo de codificación específica usada para transmitir y recibir información.

Los bits de datos hacen referencia a la cantidad de bits (palabra), en la Ilustración 11, en la transmisión y este número depende del tipo de información que se transfiere. Por ejemplo, la representación de caracteres ASCII ya que este puede tener un intervalo de valores que va de 0 a 127, que si se convierte a binario se tienen 7 bits [17]. Estos bits igual se le

conocen como “paquetes”, los cuales hacen referencia a la transferencia de un byte, el cual incluye los bits de inicio/paro, bits de datos y paridad.

Los bits de paro se utilizan para indicar el fin de la comunicación de un solo paquete y puede obtener valores de 1, 1.5 o 2 bits. La finalidad de estos bits, además de ofrecer información de cuando se termina de transmitir, es dar un margen de tolerancia para la diferencia en los relojes, ya que como es una comunicación asíncrona puede que los dos dispositivos estén en diferentes instancias de su propio reloj.

Otra característica de la comunicación serial presentada en [17] es el bit de paridad, el cual es el último bit después de los bits de datos. Esto permite verificar si hay o no errores en la transmisión de una forma sencilla. Existen cuatro tipos de bit de paridad: par, impar, marcada y espaciada; de igual forma se puede elegir no habilitar este bit. Usualmente, muchos entornos de programación (IDE) durante la creación del archivo aparecen las opciones referentes al tipo de implementación y es aquí donde se puede configurar el bit de paridad; sin embargo, se puede hacer por instrucciones dentro del código.

En el caso de habilitar el bit par o impar, el puerto serial fijará este a un valor para asegurarse que la transmisión tenga un número par o impar de bits en estado lógico alto. Por ejemplo, si la información que se transmite fuese 0111 y la paridad es par, el siguiente bit después de los datos estaría en estado lógico alto “1” para completar la paridad (cuatro unos). Caso contrario si se elige la paridad impar ya que como la información tiene un número impar de “1” este bit estaría en “0”. Los paquetes de datos se ejemplifican en la Ilustración 12.

Bit de inicio	Bits de datos	Bit de paridad	Bit de parada
---------------	---------------	----------------	---------------

Ilustración 12 Paquete de datos con bit de paridad.

En la paridad marcada y espaciada no verifica, en realidad, el estado de los bits de datos; simplemente fija el bit de paridad en estado lógico alto para la marcada y en estado bajo para la espaciada, lo cual representa una ventaja porque el equipo receptor buscará este bit al final de los datos para determinar si existe o no ruido en la transmisión [17].

3 Desarrollo del Prototipo

Los investigadores del LANRESC en el OCR Sisal requieren un sistema que automatice la recolección de datos del manto acuífero; dimensionado para soportar el peso de las sondas EXO 1 y EXO 2 de 5kg como máximo. Además, necesitan que se sumerja y emerja a una velocidad constante alrededor de los 5 cm/s, sin embargo, el usuario debe modificar esta variable, ya que de esta forma se establece el tiempo entre muestras durante el trayecto de la sonda. También debe permitir la configuración de las veces que se repetirá automáticamente el proceso de sumergir y emerger la sonda; así como el tiempo de espera entre repeticiones.

Tal como se menciona en el estado del arte en [8] la empresa YSI ofrece un producto que tiene las características antes mencionadas. Sin embargo, la solución de YSI requiere un dispositivo por cada cuerpo de agua a monitorear, lo que la convierte en una respuesta poco viable debido al alto costo de compra de estos equipos. Por lo tanto, este trabajo tiene como enfoque obtener un dispositivo que sea portátil ya que los pozos de interés tienen características de profundidad máxima y diámetro muy similares.

Este capítulo se divide en tres apartados, la estructura metálica, el hardware y el software. En el primero se definen los criterios de diseño de la estructura, sabiendo que el peso máximo de la sonda es de 5kg. También se elaboran los diagramas de cada pieza del sistema con la finalidad de tener un registro para futuras aplicaciones de interés.

Una vez definida la estructura, se establecen los criterios de diseño para las etapas de potencia y de control, las cuales son necesarias para el funcionamiento del motor a 12VDC, la tarjeta electrónica y los periféricos tanto de interacción con el usuario como de los elementos físicos para la retroalimentación del control por software.

En el último apartado se define el funcionamiento deseado por parte de los investigadores del LANRESC a fin de establecer un diagrama de flujo para la codificación de los algoritmos de operación del prototipo por parte del usuario. De igual forma, se define el método de control a utilizar y se establecen las bases para la comunicación serial entre la tarjeta de control y el operador.

3.1 Estructura metálica

Como parte de un prototipo funcional, es necesario diseñar y desarrollar una estructura capaz de integrar el software con el hardware.

El aluminio es el material propuesto para implementar gran parte de la estructura y dentro de esta existen elementos fabricados de madera, plástico y acrílico porque estos pueden soportar la salinidad, humedad y temperaturas elevadas superiores a los 40°C que son variables ambientales características de las zonas costeras.

De igual forma, se requiere el uso de poleas para mantener la fuerza y tensión que hace el motor, así como el fácil enrollado del cable en el carrete. También, estas se integran junto con el hardware para poder medir variables y eventos externos de interés para la operación del sistema, tales como la velocidad, la posición de la sonda y si su trayecto está bloqueado por algún obstáculo dentro del pozo. Las tres poleas utilizadas en el prototipo son de 5cm de diámetro exterior.

La implementación, el diseño de la estructura metálica y el funcionamiento de esta, es un trabajo en conjunto de la UNAM, el TecNM/ITMérida, la UADY y el I.E. Rubén Rosado. Los planos de cada elemento de la estructura se encuentran en el Anexo 1.

Para explicar la estructura se destacan las tres partes más relevantes para el funcionamiento las cuales son:

- El carrete
- La polea del codificador
- El arreglo de poleas tensoras

Estos elementos de la estructura se describen a continuación y su ubicación dentro de la estructura se presenta en la Ilustración 13. Se propone el diseño de prisma rectangular ya que el LANRESC así lo requiere para su fácil transporte. Las medidas se muestran en milímetros en la Ilustración 14.

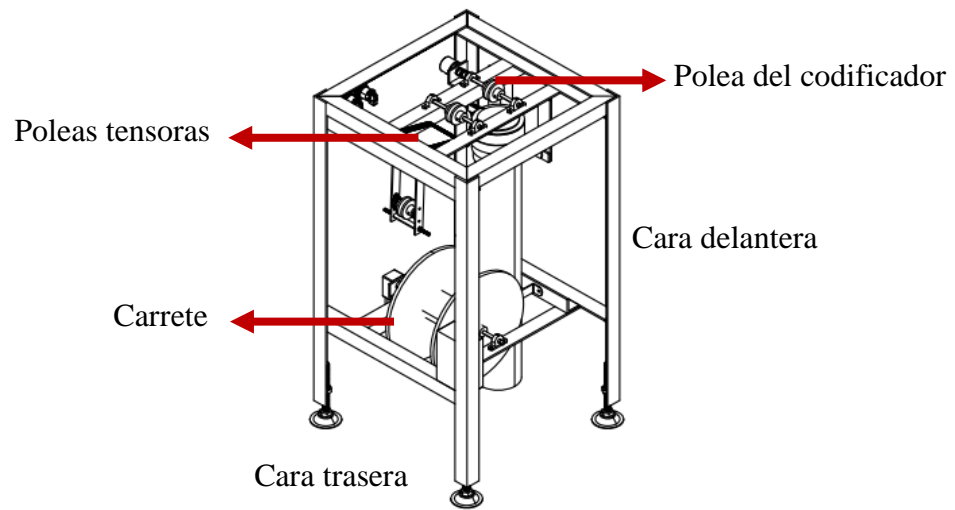


Ilustración 13 Elementos críticos para el funcionamiento del prototipo

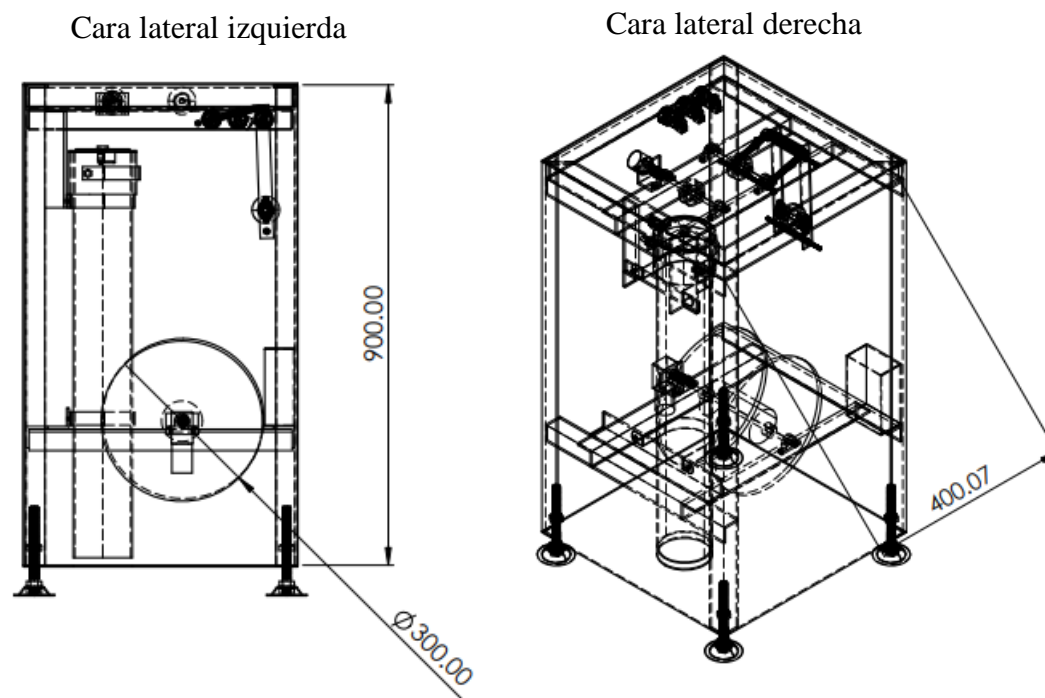


Ilustración 14 Vistas laterales y medidas en milímetros del prototipo

3.1.1 Arreglo de poleas de tensión

Al realizar las pruebas funcionales en laboratorio, se observa que puede ocurrir dos eventos críticos debido a la pérdida de tensión:

1. Al sumergir la sonda existe cierta flotación. Esto ocasiona un efecto de “rebote” en el eje de la polea; haciendo que gire momentáneamente en el sentido inverso al movimiento y los datos de pulsos contados se comprometan.
2. Al bloquear el camino de la sonda y/o al llegar al fondo. Si se pierde por completo la tensión, el cable tiende a salirse del cauce de la polea; ocasionando que el codificador deje de leer y/o que el cable se desenrolle al punto de amarrarse en la parte interna de la estructura.

La solución propuesta en esta tesis es diseñar e implementar un arreglo de dos poleas idénticas ubicadas en diferentes partes, tal cual se muestra en la Ilustración 15. La primera está en una estructura móvil de 25cm de largo con dos resortes de 10cm de longitud en cada lado y está ubicada (con referencia a la misma estructura) entre la polea del transductor y la segunda polea de este arreglo. La segunda estará fija sobre las barras superiores de aluminio localizada a 13.1cm de la polea del transductor.

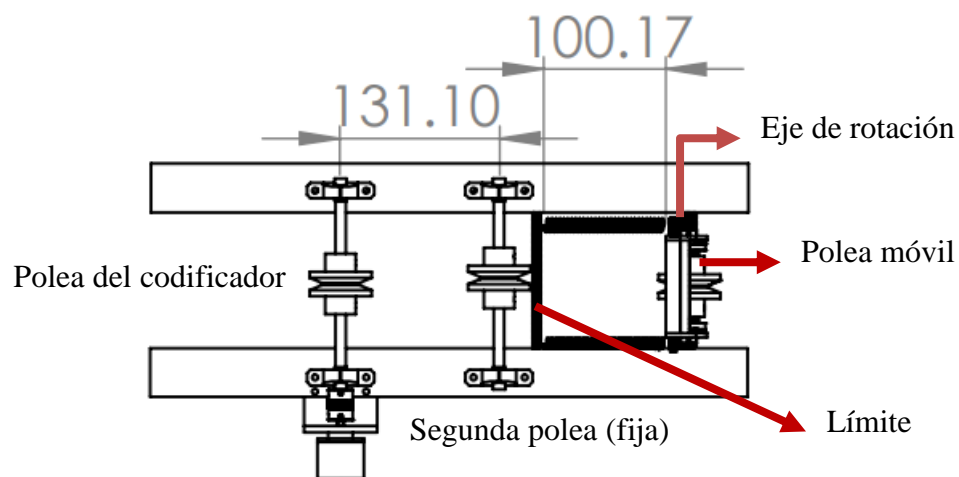


Ilustración 15 Vista superior del arreglo de poleas tensoras

La estructura móvil tendrá su eje a 10cm de la segunda polea para que cuando el cable tenga la máxima tensión la polea móvil se encuentre entre la del codificador y la fija. Cuando ocurra el evento de pérdida de tensión en el cable, la estructura móvil se desplazará hacia abajo haciendo que los resortes absorban dicha pérdida y la polea móvil se localice después de la polea fija, tal como se muestra en la Ilustración 16.

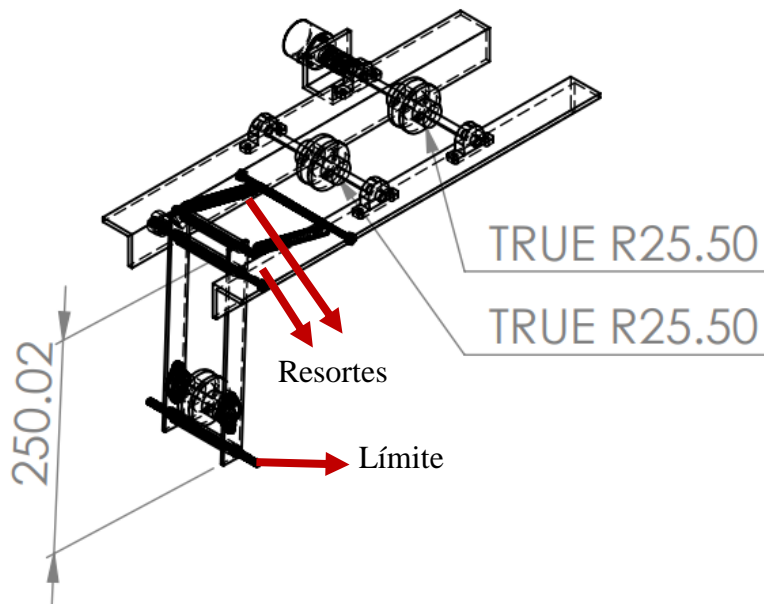


Ilustración 16 Estructura móvil absorbiendo la máxima pérdida de tensión del cable.

Durante las pruebas en laboratorio se observa que al absorber la máxima pérdida de tensión puede ocasionar que el cable se enrede entre el límite de la estructura móvil, por lo que se propone ajustar dicha estructura a la máxima absorción segura utilizando un interruptor localizado en el eje de rotación.

3.1.2 Carrete

La sonda estará conectada a un cable multi conductor de 8 hilos para que se puedan descargar los datos del monitoreo, sin embargo, para el enfoque de esta tesis se utiliza un cable de acero, mediante el cual se puede manipular la sonda dentro del pozo. Para esto, es necesario diseñar un carrete (Ilustración 17) que facilite el enrollado y desenrollado del cable durante la

operación del equipo. Para obtener el diámetro del carrete se considera que el cable multi conductor tiene un diámetro exterior de 8.48mm, por lo que se proponen los siguientes valores mínimos:

$$\varnothing_{carrete} = 10cm; \quad l = 20cm$$

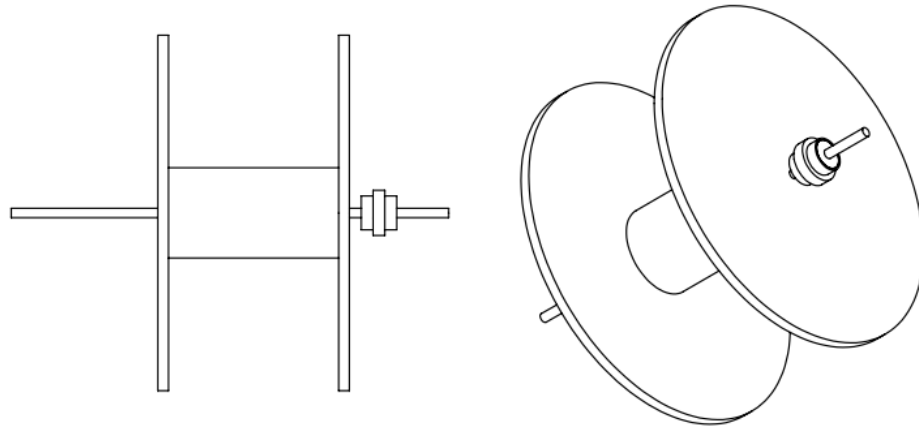


Ilustración 17 Diseño del carrete

Al enrollarse el cable en el carrete se irán formando “capas” al montarse las tiras del cable una encima de la otra, por lo tanto, se debe calcular el perímetro de los bordes del carrete para comprobar que la cantidad máxima de capas no supere los 10cm.

$$Perímetro = (10cm)(3.1416) \cong 31.4cm$$

$$Vueltas por capa = \frac{l_{carrete}}{\varnothing_{cable}}$$

$$Vueltas por capa = \frac{200mm}{8.48mm} \cong 24 vueltas$$

Se utiliza el perímetro de la capa más interna para realizar un estimado de la cantidad de cable que se puede almacenar en cada una, debido a que las capas superiores (al tener un diámetro más grande) pueden almacenar una longitud mayor. Sin embargo, no se espera que se acomoden perfectamente.

$$\text{Metros por cada capa} = (24 \text{ vueltas})(31.4\text{cm}) \cong 753.6\text{cm} \cong 7.5\text{m}$$

Para obtener el número de capas es necesario conocer la distancia máxima del cable a enrollar, para esto se considera que la profundidad de los pozos es de alrededor 20m y se considera un excedente de 10m de cable, haciendo así una longitud máxima de 30m.

$$\# \text{ de capas} = \frac{30\text{m}}{7.5\text{m}} \cong 4 \text{ capas}$$

$$r_{\text{capas}} = 4 \text{ capas} * 8.48\text{mm} = 33.92\text{mm} \cong 3.4\text{cm}$$

Para proponer un diámetro del carrete es necesario considerar que el radio máximo que se obtiene al acomodar las 4 capas de cable en toda la longitud de dicha pieza está compuesta al sumar el radio interno del carrete y el radio de las capas:

$$r_{\text{max}} = 5\text{cm} + 3.4\text{cm} = 8.4\text{cm}$$

En este trabajo se utiliza un carrete con un radio exterior de 10cm, ya que es suficiente considerando que el cable puede que no se acomode perfectamente en toda la longitud del carrete. También es relevante señalar que ambos extremos del eje están conectados mediante un juego de baleros idénticos a las poleas; sin embargo, el extremo izquierdo se conecta a un acople de tres partes que se conecta al eje del motor.

3.1.3 Polea del codificador

Dentro de la estructura se decide utilizar una polea que está situada sobre un tubo de PVC con diámetro de 15cm, el cual sirve como resguardo para la sonda cuando no esté dentro del pozo, tal como se muestra en la Ilustración 18. La finalidad que tiene esta polea es de mantener un ángulo de inclinación superior a los 90° con respecto al carrete para que la fuerza ejercida por el motor al hacer emerger la sonda sea lo más similar posible en cada instante.

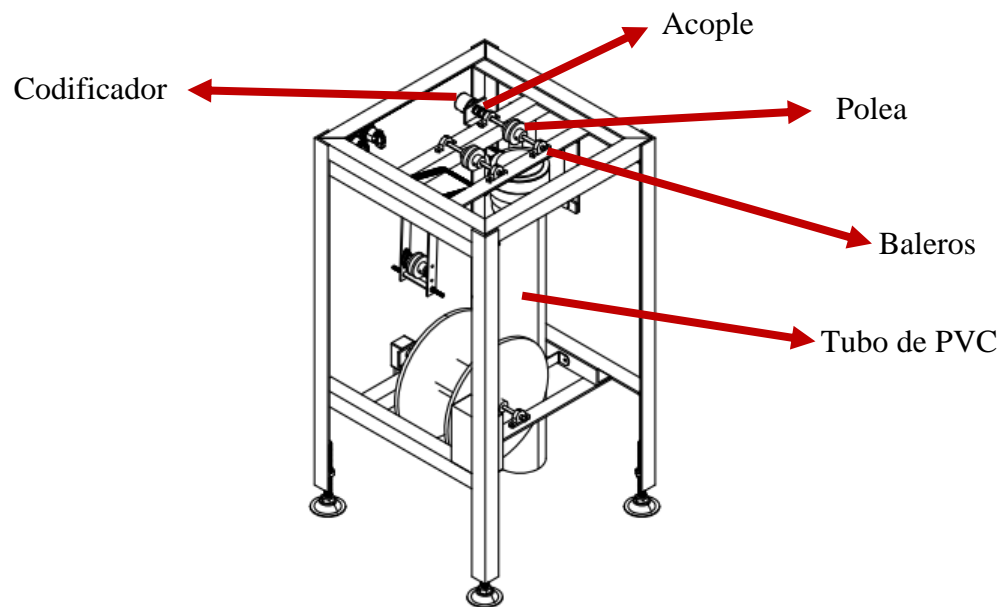


Ilustración 18 Localización de la polea del codificador respecto a los demás elementos de la estructura

Esta polea es de plástico poroso y la forma de su hendidura es cónica con diámetro interior de 3.6cm. Las características del material permiten que el cable no derrape mientras esté rotando.

Ambos extremos del eje de rotación se conectan a un juego de baleros con la finalidad de que durante el movimiento de giro no genere la fricción necesaria para detener el motor. Lo anterior se expone en la Ilustración 19. De igual forma, el eje del lado izquierdo se prolonga después del balero para conectarse mediante un acople flexible a un codificador, el cual sirve como retroalimentación de la cantidad de cable que se ha dispensado hacia el pozo o se ha extraído de él, por lo tanto es necesario que durante el giro de la polea no se le aplique la fuerza suficiente para dañar el eje del transductor rotativo. El acople es de aluminio con hendiduras en el centro y estas son las que absorben el exceso de fuerza.

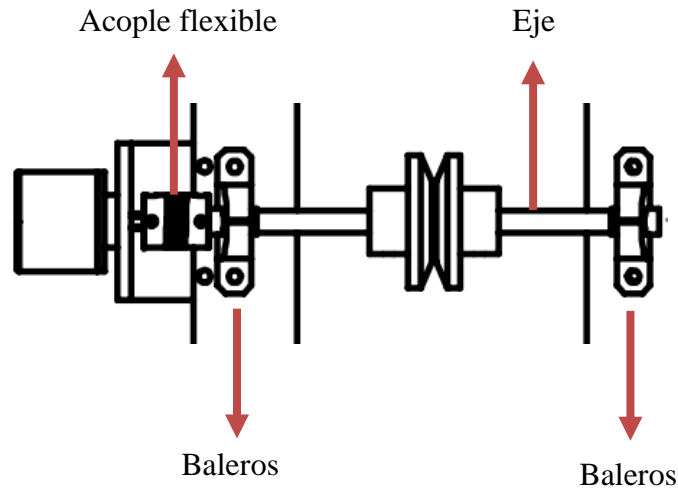


Ilustración 19 Vista superior de la estructura, enfocando el acople, los baleros y el eje.

3.2 Hardware

En esta sección del capítulo se revisan todos los elementos que conforman el hardware del sistema, para esto, es importante definir ciertas características tales como la conectividad, el método de control, el peso a mover por el motor, los sensores e interruptores que transforman los fenómenos y eventos en señales eléctricas para su análisis mediante software.

3.2.1 ESP32-DevKitC

El ESP32-DevKitC es un módulo de evaluación que cuenta con el chip ESP32-WROOM-32, el cual es un sistema de doble núcleo con dos CPUs “Harvard Architecture Xtensa LX6”. Incluye de forma nativa los controladores para los protocolos de comunicación Wi-Fi a 2.4GHZ y BLE. Además, su voltaje de operación lógico es 3.3V en todos los pines de propósito general y puede alimentarse a 5V en la entrada del regulador. Otra característica importante es que permite utilizar un entorno de programación amigable como el IDE de Arduino, sin embargo, al utilizar este no es posible utilizar los dos núcleos del chip.

Cuenta con dos canales ADC (convertidores analógicos a digital), sin embargo, el canal dos no puede ser utilizado mientras el módulo Wi-Fi esté encendido debido a que estos son compartidos y el módulo es de mayor prioridad durante la ejecución. Esta placa de desarrollo cuenta con 28 pines de entrada/salida de propósito general (GPIO), tal como se

muestra en la Ilustración 20 [18]; a todos se le puede asignar una interrupción y configurarse como salida de una señal de PWM.

ESP32-DevKitC

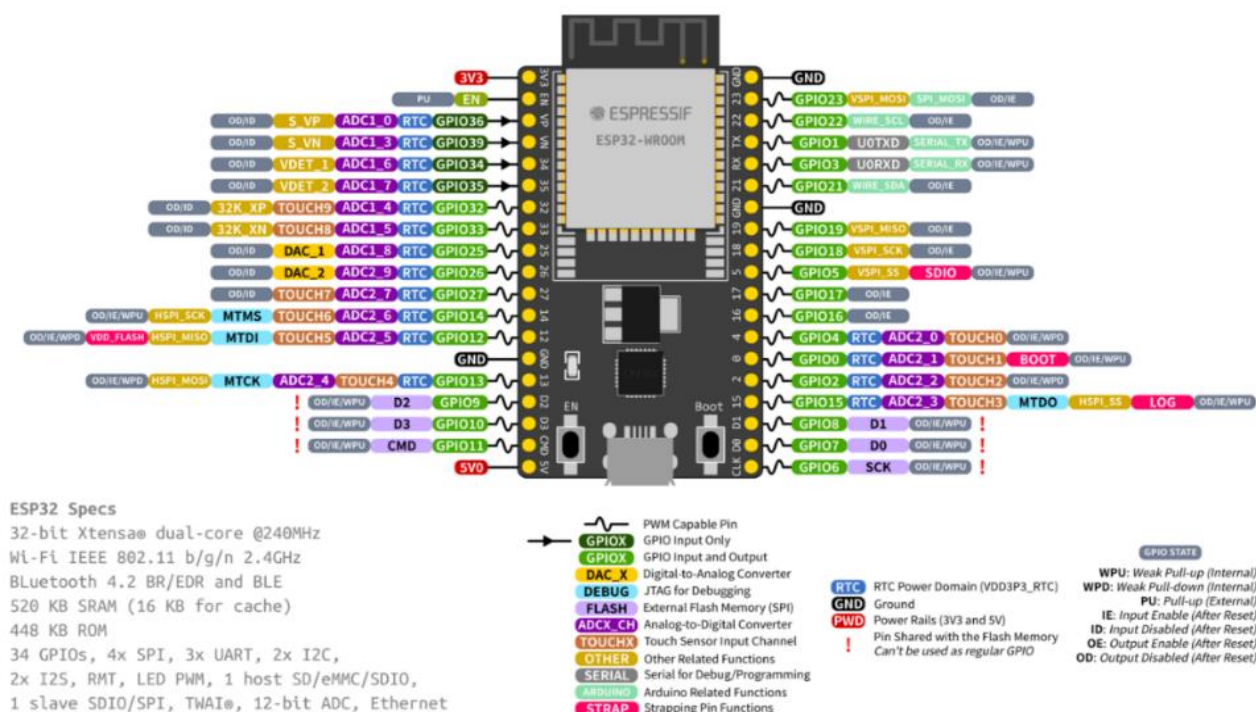


Ilustración 20 Mapa de ubicación y propósito de los pines del ESP32-DevKitC [9]

3.2.1.1 LEDC

Para la generación de las señales de PWM, este microcontrolador cuenta con dos opciones las cuales son: el controlador “LEDC” y el controlador “MCPWM” (Motor Control Pulse Width Modulation). Estos controladores configuran los periféricos del mismo nombre, siendo el periférico LEDC diseñado para controlar la intensidad de los LEDs conectados a la placa, sin embargo, puede generar señales de PWM para propósitos generales, según las especificaciones del fabricante [19]. Esta cuenta con 16 canales que pueden generar formas de onda diferentes entre ellos y se les puede asignar el mismo canal a más de una salida, permitiendo ser usados en simultáneo [19]

Los canales del controlador LEDC se dividen en dos grupos de ocho canales como se muestra en la Ilustración 21. Uno de ellos opera en el modo de alta velocidad, el cual está implementado en hardware, lo que permite que, si los valores máximos del contador o del divisor son modificados en el software, estos se actualizarán de forma automática y libre de errores. Mientras que los otros ocho operan en modo de baja velocidad y pueden reconfigurarse utilizando el controlador LEDC mediante software; sin embargo, estos se modificarán hasta que los registros se hayan reconfigurado.

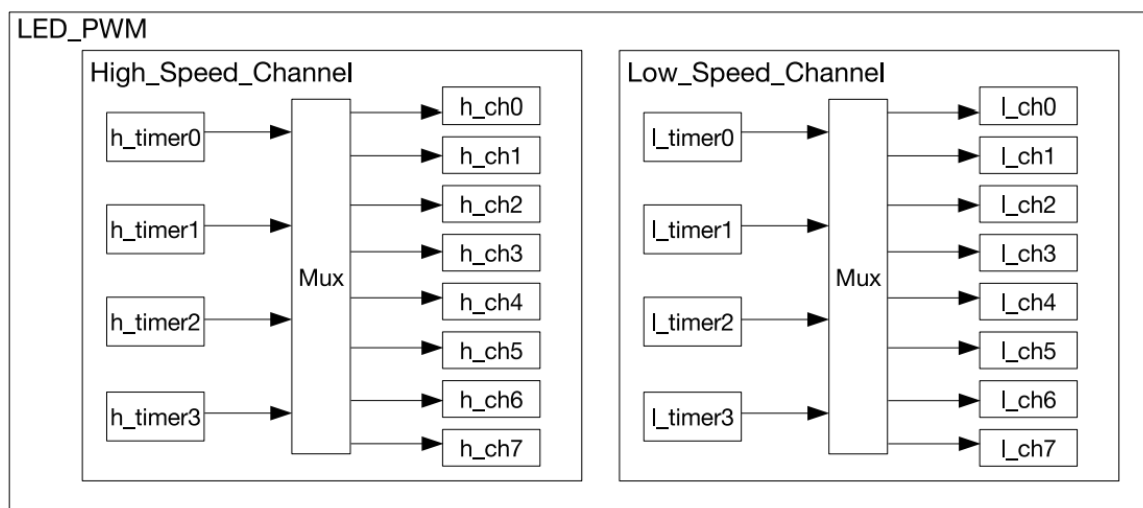


Ilustración 21 Arquitectura del controlador LED PWM del ESP32 [19]

La configuración propuesta por el fabricante para utilizar este controlador es la siguiente, tal como se muestra en la Ilustración 22:

- Configurar el contador especificando las resoluciones de frecuencia y del ciclo de trabajo de la señal de PWM.
- Configuración del canal asociando el contador a la salida GPIO deseada.
- Modificar la señal PWM de salida para cambiar el ciclo de trabajo. Esto puede realizarse mediante software o con las funciones de disminución automática del hardware.

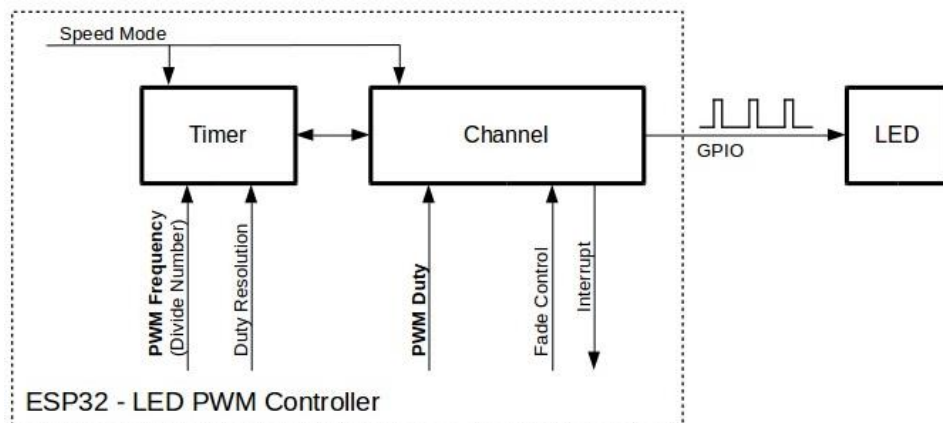


Ilustración 22 Diagrama de configuración del Controlador LED PWM [19]

En el Manual Técnico presentado por el fabricante se presentan los valores de frecuencias y resoluciones más usados, tal como se muestra en la Tabla 1:

Tabla 1 Frecuencias y resoluciones más usadas

Fuente del reloj LEDC	Frecuencia de salida LEDC	Máxima resolución
APB_CLK (80MHz)	1 kHz	1/65536 (16 bit)
APB_CLK (80MHz)	5 kHz	1/8192 (13 bit)
APB_CLK (80MHz)	10 kHz	1/4096 (12 bit)
RTC8M_CLK (8MHz)	1 kHz	1/4096 (12 bit)
RTC8M_CLK (8MHz)	8 kHz	1/512 (9 bit)
RTC8M_CLK (8MHz)	1 kHz	1/512 (9 bit)

De igual forma, el fabricante en [19] expone que, dependiendo de la frecuencia será la resolución máxima que se podrá configurar mediante software, siendo la frecuencia máxima 40 MHz con una resolución de 1 bit. Si se llega a configurar una resolución mayor a la capacidad del controlador arrojará el siguiente error:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try reducing
freq_hz or duty_resolution. div_param=128
```

Y si se configura una combinación de resolución/frecuencia no inferior a la soportada por el ESP32 la siguiente leyenda se mostrará en el monitor serie:

```
E (196) ledc: requested frequency and duty resolution cannot be achieved, try increasing  
freq_hz or duty_resolution. div_param=128000000
```

De la práctica, de las experiencias de diversos usuarios en diferentes aplicaciones del ESP32 y del manual técnico del fabricante; se conoce que la frecuencia del controlador va desde los 10 Hz hasta los 40 MHz, considerando que mientras mayor sea la frecuencia menor será la resolución, lo que lo hace menos preciso.

Para el trabajo de esta tesis se considera que al utilizar un motor controlado mediante una frecuencia de PWM en el rango de audición del ser humano, podría generar molestias a largo plazo, por lo que se utiliza una frecuencia de 20 kHz a una resolución de 8 bits, ya que esta será suficiente para el control oportuno de la velocidad mediante software a pesar de que no se encuentra dentro de los valores comúnmente usados según el fabricante.

3.2.1.2 Contador de alta resolución (ESP Timer)

Este chip provee contadores de alta resolución por software. Estos tienen dos limitantes, siendo la primera que la resolución máxima es la del periodo de conteo del RTOS (Real-Time Operative System) y la devolución de la solicitud del conteo es una tarea de baja prioridad. El hardware interno del *esp_timer* es de 64 bits y el valor de retorno de la cuenta está en microsegundos.

Al ser un evento de baja prioridad, pueden ocurrir retrasos en la obtención del dato. Esto se considera, debido a que el prototipo cuenta con tres interruptores físicos para la operación del sistema y están enlazados mediante software a la solicitud del conteo del tiempo actual del microcontrolador.

3.2.2 Motor

El LANRESC utiliza dos modelos de sondas para la medición de la calidad del agua, los cuales son la EXO 1 y EXO 2; por lo tanto, se conoce el peso máximo de la sonda que se

debe sumergir y hacer emerger, el cual es de 5kg. De igual forma, se considera el peso máximo del cable que debe cargar el motor al hacer emerger la sonda desde la base del pozo y la velocidad requerida de 5cm/s.

Teniendo estos criterios, se hace la elección del motor realizando los cálculos de fuerza en $kg \cdot cm$, así como la velocidad en RPM deseada por el usuario; lo que da como resultado los valores relativos que permiten elegir el motor. Por lo tanto, se considera que el cable tiene un diámetro exterior de 8.48mm con un peso por metro de, aproximadamente 55gr. Por lo tanto, se considera que la sonda pesa 5kg y que los 30m de cable pesan 1.5kg. Por lo que se propone un peso máximo de 6.5kg.

De la estructura diseñada, se obtiene que el radio máximo del cable enrollado es de 8.4cm (que sería el peor de los casos), entonces la fuerza se calcula:

$$Fuerza = (6.5kg)(8.4cm) = 54.6kg \cdot cm$$

Para que logre bajar la sonda a 0.05m/s la velocidad angular requerida es de:

$$Velocidad = \left(\frac{0.05m}{1s}\right) \left(\frac{60s}{1min}\right) = \left(\frac{3m}{1min}\right) \left(\frac{1 \text{ revolución}}{0.314m}\right) = 9.554rpm$$

Con estos cálculos se escoge un motor (Ilustración 23) con una fuerza de aproximadamente 54.6kg*cm y una velocidad mínima de 9.5 rpm; a causa de la accesibilidad y disponibilidad de motores con estas características se escoge el siguiente motor vendido a través de una tienda en línea:



Ilustración 23 Motor utilizado de 430rpm

Las especificaciones del motor según el fabricante se presentan en la Tabla 2

Tabla 2 Características del motor CHW-GW4058-555

Voltaje de alimentación	12V
Relación de engranes	1:36
Velocidad sin carga	215 rpm
Corriente sin carga	<1A
Eficiencia nominal a 12V	
Velocidad	105 rpm
Torque	20kg*cm
Corriente	<10A

Los cálculos obtenidos para la fuerza del motor son considerando el peso del cable multi conductor. Sin embargo, para el enfoque de esta tesis, se utiliza un cable de acero recubierto de plástico cuyo peso es mucho menor que los 1.5kg; además que la sonda con la que se prueba el sistema pesa alrededor de 2kg, por lo que este motor soluciona la necesidad para este trabajo.

3.2.3 Codificador rotativo incremental

El control es una de las características de un sistema automático y puede ser completamente de hardware o con la combinación del software y dispositivos que interpreten fenómenos físicos en señales eléctricas. Para este trabajo se propone el uso de un transductor rotativo óptico incremental de 400 pulsos por vuelta (Ilustración 24), el cual transformará el movimiento angular de un eje en una serie de pulsos digitales, con la finalidad de obtener el valor de la longitud de cable desenrollado del carrete.

La precisión obtenida por el codificador está dada por la siguiente fórmula:

$$Resolución\ del\ encoder = \frac{\varnothing \cdot \pi}{Pulsos}$$

Donde \emptyset es el diámetro de la polea que está conectada al eje del codificador y los “pulsos” es la cantidad de interrupciones que se genera en una vuelta completa. Sustituyendo el valor del diámetro propuesto de la polea y la cantidad de pulsos por vuelta del transductor se obtiene la resolución en centímetros.

$$\text{Resolución del encoder} = \frac{(3.6 \text{ cm})(\pi)}{400 \text{ ppv}} = 0.02827 \text{ cm}$$

El rango voltaje de alimentación del codificador utilizado en este trabajo es de 5 a 24VDC y sus salidas de la señal de interrupción utilizan tecnología “PUSH - PULL” por lo que no es necesario una interfaz para interpretar dichas señales.



Ilustración 24 Codificador rotativo incremental con resolución de 400ppr

Durante las pruebas de funcionamiento del codificador se obtuvo que el voltaje mínimo de operación es de $\sim 4.5V$, sin embargo, el voltaje máximo soportado por las entradas del ESP32 es de 3.3V, por lo que es necesario diseñar una interfaz utilizando divisores de tensión para evitar dañar el microcontrolador.

Basándose en el diagrama esquemático mostrado en la Ilustración 25 y en las igualdades siguientes se obtienen los valores de las resistencias del divisor resistivo.

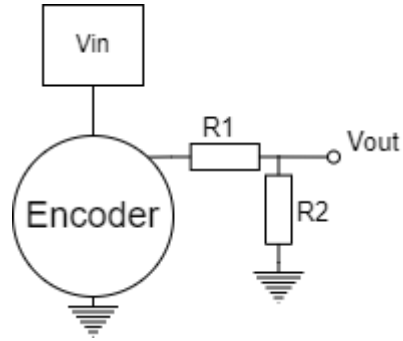


Ilustración 25 Circuito del divisor de tensión de 5V a 3.3V

Donde $V_{in} = 5V$, el cual es el voltaje de alimentación del codificador y $V_{out} = 3.3V$ sería el voltaje máximo de salida de la señal pulsante.

$$R_1 = \frac{V_{in} - V_{out}}{I}$$

$$R_1 = \frac{5V - 3.3V}{1mA} = 1.7k\Omega$$

$$R_2 = \frac{V_{out}}{I}$$

$$R_2 = \frac{3.3V}{1mA} = 3.3k\Omega$$

Los pines físicos del codificador son los siguientes:

Tabla 3 Funcionamiento de los pines físicos del codificador

Color del Pin	Funcionamiento
Rojo	Alimentación (VCC)
Negro	GND (0V)
Blanco	Fase A
Verde	Fase B

Este codificador al ser de tipo incremental cuenta con dos fases que permiten obtener la velocidad de giro y el sentido de giro. Si se analiza/procesa solo una se puede obtener la velocidad de giro mediante software, mientras que si se analizan/procesan ambas es posible obtener el sentido de giro. Sin embargo, este tipo de transductores presentan errores en la obtención de la información debido a las características mecánicas del funcionamiento (Ilustración 26), como pueden ser fallos en el foto-receptor, la calibración del disco óptico, ruido en la señal, entre otros.

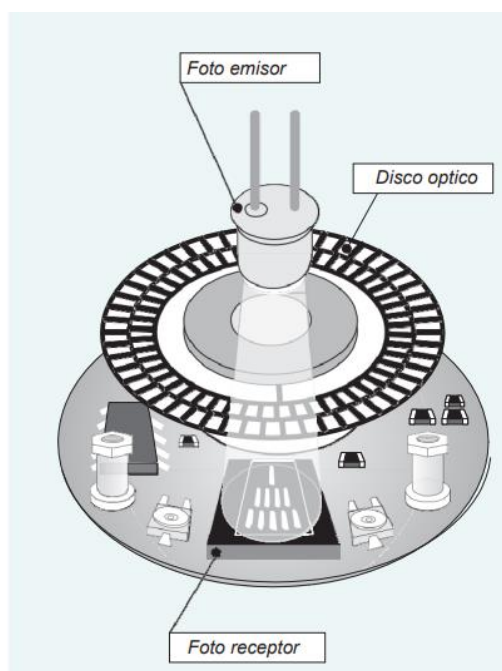


Ilustración 26 Principio de funcionamiento del transductor óptico incremental [20]

Por lo anterior, en este trabajo se utilizan ambas fases, donde una está conectada a una interrupción externa del microcontrolador y la otra a una entrada de uso general cuya función es evaluar el estado lógico del pin para conocer la dirección de giro.

3.2.4 Controlador de motor BTS7960

Debido a que se utiliza un motor en el proyecto, es necesario implementar un controlador que sirva de interfaz entre el microcontrolador y el motor seleccionado. Para esto, se propone utilizar el módulo de dos integrados BTS7960. En [21] se menciona que el

componente BTS7960 tiene un arreglo de dos MOSFET, uno canal N (bajo) y el otro canal P (alto) con un integrado controlador, formando así un medio puente “H” de alta corriente como se muestra en la Ilustración 28Ilustración 27. El BTS7960 se puede combinar con otro BTS7960 para formar un puente “H” completo.

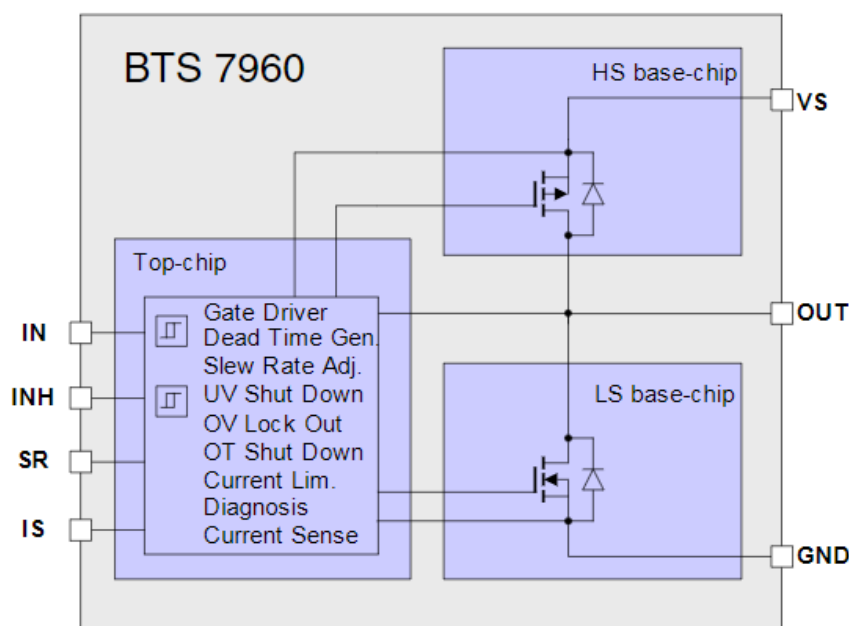


Ilustración 27 Diagrama de bloques funcionales del integrado BTS7960 [21]

Otra característica importante es que estos integrados están dimensionados para soportar un pico de corriente de 43A y un voltaje de salida al motor de 6 a 27VDC. De las especificaciones del fabricante se conoce que el voltaje típico del estado alto lógico es de 1.75V y que el máximo es de 2.15V; lo que permite interconectarlo con el ESP32. Además, la frecuencia máxima de PWM es de 25kHz, lo cual no representa una limitante para este trabajo, ya que se utiliza una frecuencia de 20kHz.

El módulo tiene 12 pines externos, de los cuales 6 son entradas de control, 2 son entradas de voltaje lógico, 2 son de la salida de la señal de PWM y 2 son de la entrada de alimentación del motor. El nombre, número y función de cada pin se describe en la Tabla 4:

Tabla 4 Distribución y descripción de pines físicos del módulo BTS7960

Pin	Función	Descripción
Señales de control		
1	RPWM	Giro hacia la derecha/Señal de PWM. Activo en HIGH
2	LPWM	Giro hacia la izquierda/Señal de PWM. Activo en HIGH
3	R_EN	Activación de giro hacia la derecha. ON = 1/OFF = 0
4	L_EN	Activación de giro hacia la izquierda. ON = 1/OFF = 0
5	R_IS	Salida de la alarma de corriente/Control hacia la derecha
6	L_IS	Salida de la alarma de corriente/Control hacia la izquierda
7	VCC	Alimentación lógica (3.3 a 5V) del microcontrolador
8	GND	Tierra de la fuente del microcontrolador
Señales de potencia		
1	B+	Alimentación positiva del motor (6 – 27VDC)
2	B-	Alimentación negativa del motor. Tierra.
3	M+	Salida del motor positiva.
4	M-	Salida del motor negativa.

[18] Los pines físicos del módulo se presentan en la Ilustración 28:



Ilustración 28 Distribución de los pines físicos del módulo BTS7960

Este módulo permite la velocidad de giro al cortocircuitar las terminales de R_EN y L_EN al pin de salida de la señal de PWM del microcontrolador. Mientras que las terminales de RPWM y LPWM se utilizan para controlar el giro del motor. Se propone esta estrategia debido a que esta configuración se utiliza un pin de control menos; ya que se prioriza la optimización de las terminales para permitir la conexión de más sensores para el monitoreo del propio sistema en etapas posteriores del proyecto. En la Ilustración 29 se muestra el diagrama de conexiones del módulo con los demás elementos del sistema.

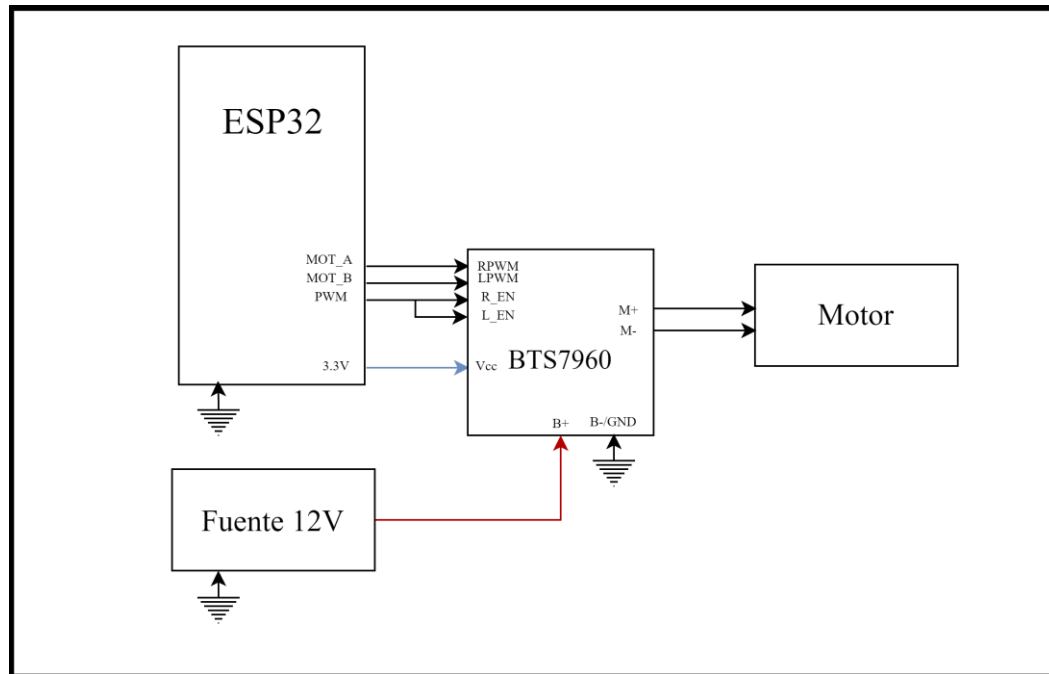


Ilustración 29 Diagrama de interconexión del módulo controlador BTS7960

3.2.5 Fuente de alimentación

Para la selección de la fuente se consideran los valores máximos del motor, donde $V_{max} = 12V$ e $I_{max} = 10A$, además se necesita una salida de $5V$ para alimentar al codificador rotativo y al ESP32. Se propone utilizar una fuente de computadora, la cual cuenta con estas características. Dentro de las ventajas más relevantes de esta clase de fuente están que ya cuenta con una salida de $5V$ para alimentar al microcontrolador y al codificador rotativo. Además cuenta con la electrónica necesaria para disminuir el voltaje suministrado cuando supera los $120W$; lo que representa una protección adicional cuando el motor consume más de $10A$.

3.2.6 Circuito sensor de corriente

Al realizar las pruebas en el pozo se obtiene que, durante la subida de la sonda, esta puede atorarse dentro del pozo debido a la diferencia en los diámetros de la perforación; haciendo que el motor logre detenerse aumentando el consumo de corriente a más de $10A$. Para medir estas situaciones y poder tomar acciones en el software, es necesario diseñar un

circuito sensor de corriente utilizando una resistencia de muy bajo valor, que permita disipar hasta 10W, y diseñar un filtro pasa bajos para eliminar el ruido generado por la frecuencia de 20kHz de la señal de PWM a la que estará funcionando el motor.

La resistencia R_{IS} está conectada en serie con la terminal negativa (GND) de dicha entrada del controlador BTS7960 y en paralelo con el filtro pasa bajos, tal como se muestra en la Ilustración 30. Para obtener el cálculo de la resistencia se toma en cuenta que el voltaje máximo que soporta el convertidor analógico digital (ADC) del ESP32 es de 3.3V y la corriente máxima que soporta el motor es de 10A. Por lo tanto, se diseña el circuito para que cuando ocurra el pico de 10A el voltaje en la entrada del ADC sea de 1V debido a dos razones importantes:

- El límite de voltaje en la entrada, ya que, si se tuviese un pico de mayor corriente, no dañe al ADC.
- La accesibilidad para conseguir una resistencia de características similares.

$$R = \frac{V}{I}$$

$$R = \frac{1V}{10A} = 0.1\Omega$$

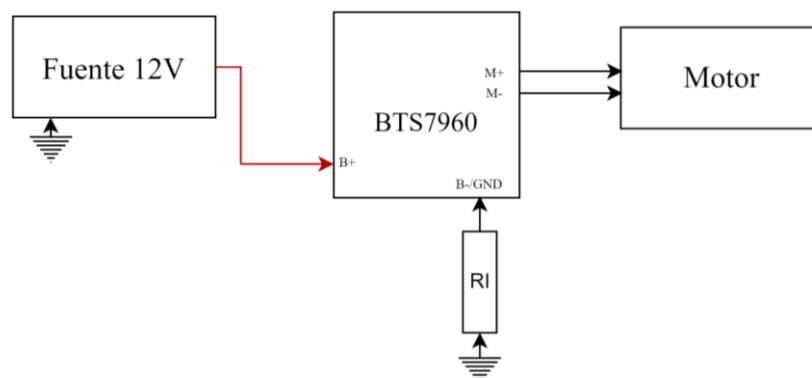


Ilustración 30 Diagrama de conexión de la resistencia sensor respecto al controlador de motores BTS7960

Al utilizar una resistencia de 0.1Ω , se cumplen los requerimientos antes mencionados, ya que este valor es fácil de conseguir en las tiendas en línea de componentes, además que el tamaño del encapsulado (50mm x 11mm x 10mm) no es lo suficientemente grande para que represente un problema en el diseño de la placa y el costo de este no es elevado debido a los 10W de potencia que disipará en el peor de los escenarios.

La otra parte de este circuito es el filtro pasa bajos, para esto se decide utilizar un filtro RC con una frecuencia de corte en 1kHz. Para definir el valor del capacitor es necesario realizar el análisis de las frecuencias de las bandas, para esto se considera que la frecuencia que se quiere rechazar es la del PWM que está en 20kHz.

Obteniendo así, el siguiente análisis:

$$\text{Si } f_c = \mathbf{1kHz} \therefore 20kHz \gg 10f_c$$

Donde f_c es la frecuencia de corte propuesta ya que, al multiplicar por 10 veces esta, se comprueba que los 20kHz del PWM sí están dentro de la banda de rechazo. Mientras que la banda de paso iniciará en 100Hz:

$$0.1f_c = 0.1(1000Hz) = \mathbf{100Hz} \therefore 10Hz \ll 0.1f_c$$

Con la igualdad anterior, se comprueba que la frecuencia propuesta para el inicio de la banda de paso es lo suficientemente pequeña para permitir pasar el paso de frecuencias de interés. Además, debido a la naturaleza mecánica del motor no ocurre movimiento alguno si el PWM tiene una frecuencia de 100Hz, por esto, se considera como el inicio de la banda de paso.

Conociendo el valor de f_c se pueden resolver las siguientes ecuaciones para obtener el valor de capacitancia con una resistencia propuesta de 100k Ω :

$$R = \frac{1}{2\pi fC} \therefore C = \frac{1}{2\pi fR} = \frac{1}{2\pi(1kHz)(100k\Omega)} \therefore C \approx 1.5nF$$

Comprobando que los valores sean funcionales para la f_c :

$$f_c = \frac{1}{2\pi RC} \therefore f_c = \frac{1}{2\pi(100k\Omega)(1.5nF)} = 1061.6Hz$$

Siendo entonces los valores para usar en el circuito:

$$R = 100k\Omega \quad C = 1.5nF \quad R_i = 0.1\Omega$$

Por lo tanto, en la Ilustración 31 se muestra el esquemático del circuito sensor de corriente respecto a la conexión de los demás componentes del proyecto.

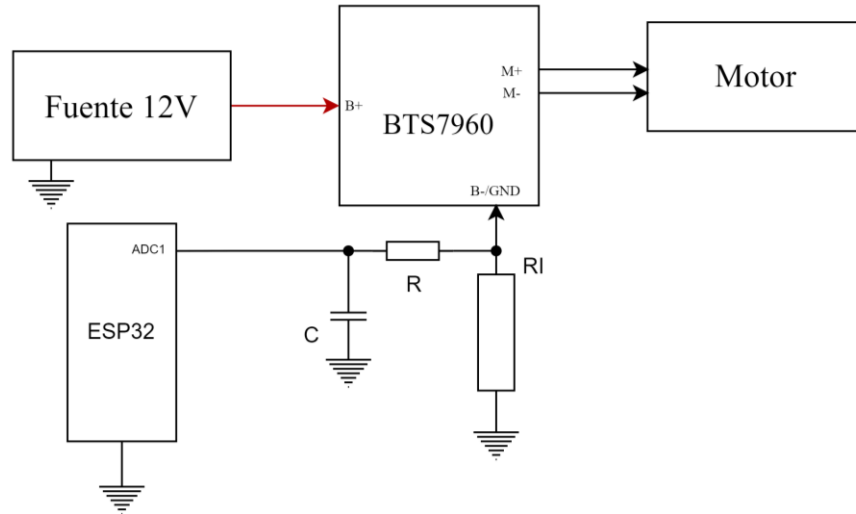


Ilustración 31 Circuito sensor de corriente completo conectado al ADC1 del ESP32

3.2.7 Switch fin de carrera

Como se menciona al inicio de este trabajo, se debe garantizar la acción de sumergir y hacer emerger la sonda en operación automática. Si bien ya se han establecido los medios por el cual conseguirlo, es necesario añadir dos interruptores de fin de carrera:

1. Montado en la tapa del contenedor donde se guarda la sonda al iniciar/finalizar su operación. Este sirve para conocer cuándo la sonda ha llegado a “casa” o

“inicio”, lo que permitirá detener el motor y reiniciar las variables principales del sistema en caso de un corte de energía en la placa.

2. Montado en el mecanismo de poleas (aparejos). Este sirve para conocer cuándo está en el límite de perder tensión en el cable, ya que si ocurre este evento se puede perder el valor de la distancia de cable desenrollado y puede ocasionar un malfuncionamiento del sistema.

Estos interruptores están conectados al ESP32 en pines configurados con la resistencia “PULLUP” interna del microcontrolador para evitar fallos en el caso que se pierda la conexión con el switch. De igual forma, estos pines se pueden configurar como entradas de interrupción externo debido a la importancia de responder en el menor tiempo posible a la perturbación ocasionada por los eventos de arribo a “casa” y pérdida de tensión.

3.2.8 Elementos de protección

Como todo circuito eléctrico, este no está excluido de errores humanos y por sobre corriente; para mitigar el impacto que puedan tener estas situaciones se consideran los siguientes elementos:

- Diodo 24V. Está localizado en la entrada positiva de voltaje (12VDC) de la placa para evitar conexiones inversas de voltaje.
- Fusible de 20A. La finalidad de este elemento no es proteger los componentes, si no evitar que los elementos se incendien cuando ocurra un pico de tal magnitud.
- Conectores de un solo sentido. Estos no solo permiten la sencilla y rápida conexión de los elementos a la placa, si no que solo se pueden conectar en el sentido del header macho.
- Caja de PVC. Dentro de esta se acomodan los cables y conectores al microcontrolador.
- Tornillos de acero. Estos fijan la placa a la caja donde se resguarda la tarjeta y aterrizan a tierra el circuito.

3.2.9 Diagrama a bloques completo del sistema

En los apartados anteriores se exponen los diferentes componentes y circuitos para desarrollar este proyecto, sin embargo, para poder diseñar una tarjeta de circuito impreso, se necesita obtener un diagrama de bloques completo de los elementos eléctricos/electrónicos del sistema, junto con la relación de pines-función que se utilizan en el ESP32. Por lo antes mencionado, se expone el diagrama a bloques en la Ilustración 32.

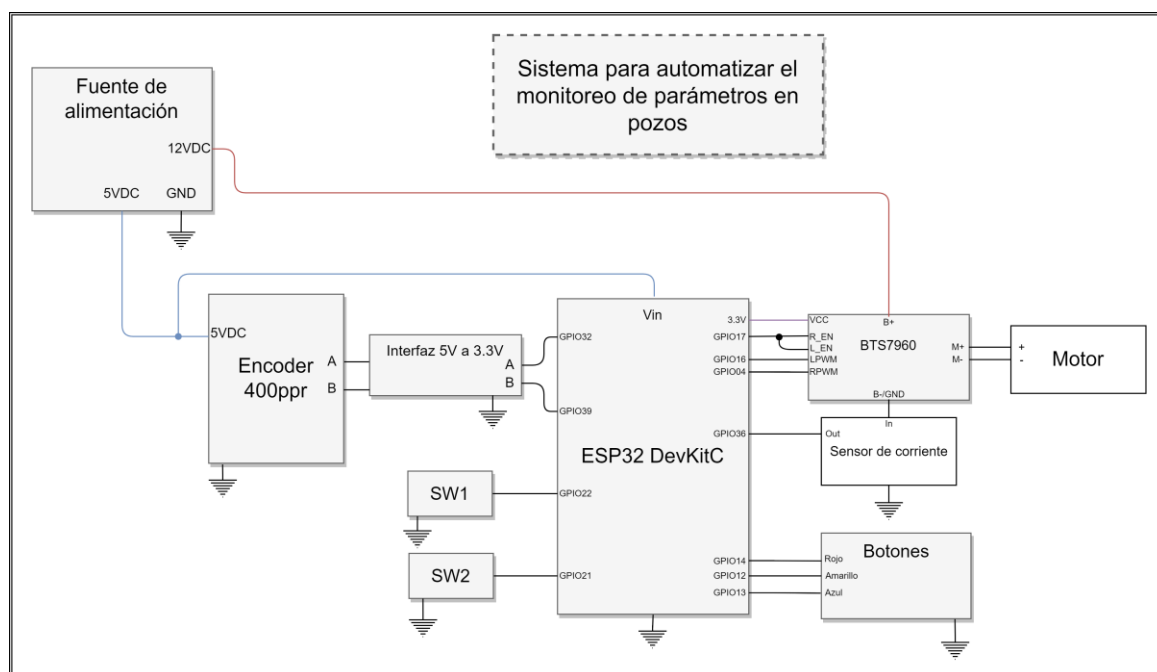


Ilustración 32 Diagrama a bloques de los componentes funcionales del sistema

En este diagrama se exponen los elementos eléctricos/electrónicos importantes para el funcionamiento del prototipo, sin embargo, se decide colocar conectores y borneras adicionales, los cuales están conectados a pines de interés para futuros trabajos. La relación de nombre-pin-modo-función conectados al ESP32 se expone en la Tabla 5.

Tabla 5 Relación de las conexiones al ESP32

Nombre	GPIO	Modo	Función
5V	VIN	Alimentación	Entrada de voltaje de alimentación ($5V_{CD}$)
3V3	3V3	Alimentación	Salida de $3.3V_{CD}$ de la fuente interna del microcontrolador.
GND	GND	Alimentación	Referencia a $0V_{CD}$.
AMARILLO	12	Entrada	Subir sonda manual
ROJO	14	Entrada	Bajar sonda manual
AZUL	13	Entrada	Operación automática
SW1	22	Entrada	Entrada de la interrupción del fin de carrera.
SW2	21	Entrada	Entrada de la interrupción de pérdida de tensión en el cable.
LPWM	16	Salida	Señal de control de giro de motor (izquierda)
RPWM	04	Salida	Señal de control de giro de motor (derecha)
R_EN/L_EN	17	Salida	Señal de PWM (PWM)
A	32	Entrada	Señal A del codificador
B	39	Entrada	Señal B del codificador
Out	36	Entrada	Señal filtrada del consumo de corriente del motor (ADC1)
Dedicado para trabajos futuros			
CS	05	Salida	Selección de chip
SCK	18	Salida	Señal de reloj
MISO	19	Entrada	Master In Slave Out
MOSI	23	Salida	Master Out Slave In
LED1	34	Salida	LED indicador
LED2	35	Salida	LED indicador
LED3	25	Salida	LED indicador

ADC1_5	33	A determinar	A determinar
GPIO26	26	A determinar	A determinar
GPIO27	27	A determinar	A determinar

3.2.10 Tarjeta de circuito impreso

Una vez diseñado el diagrama a bloques y la relación de pines utilizado, se realiza el diagrama de conexiones eléctricas utilizando el software CAD Eagle de Autodesk, obteniendo el esquemático presentado en la Ilustración 33 donde se pueden observar las disposiciones de los pines asignados para cada elemento del sistema expuestos en este mismo apartado. El criterio para asignar los pines fue la ruta más corta a las conexiones para evitar cualquier ruido que pueda generar la pista de cobre así como las funciones específicas de los pines del microcontrolador.

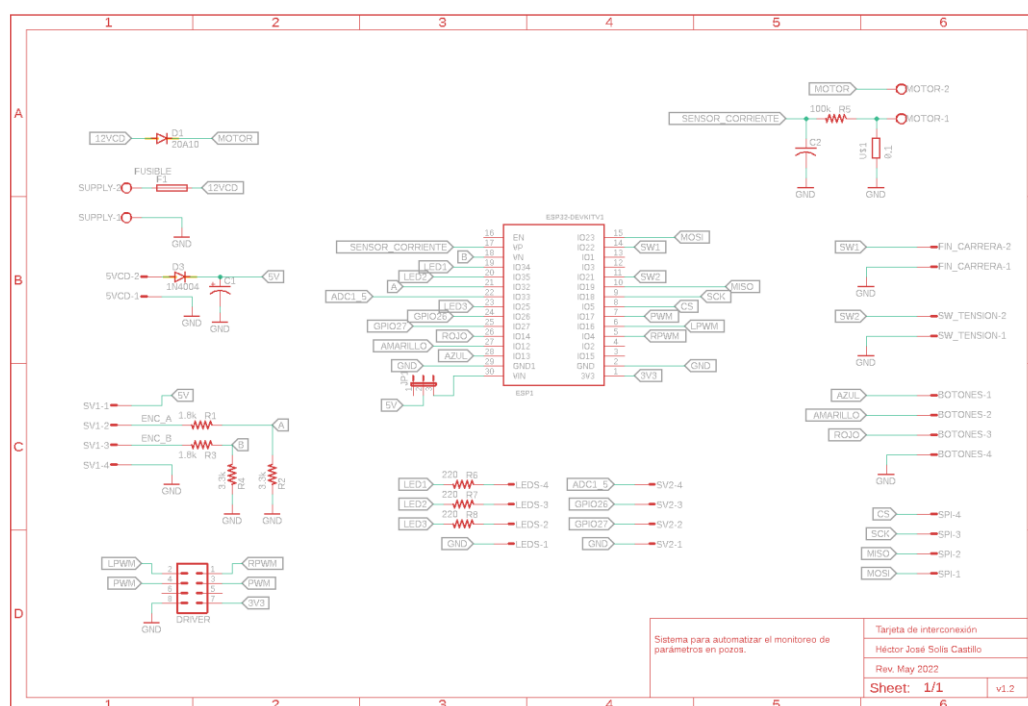


Ilustración 33 Diagrama esquemático de la tarjeta de interconexión

Para diseñar la placa, es necesario establecer el ancho de las pistas en milésimas de pulgada (mils); siendo estas clasificadas según la corriente que circulará por ellas. Por lo tanto,

las pistas de la etapa de potencia tienen un ancho de 180 mils, mientras que las de control tienen 40 mils.

Para calcular el ancho de las pistas de potencia se utilizó la herramienta de “Calculadora de ancho de trazas de PCB” que presenta DigiKey en su sitio web. Esta herramienta considera las siguientes ecuaciones [21]:

$$\text{Área}[\text{mils}^2] = \left(\frac{\text{Corriente}[\text{Amperios}]}{k * (\text{Aumento de temperatura } [^{\circ}\text{C}])^b} \right)^{\frac{1}{c}}$$

$$\text{Ancho}[\text{mils}] = \frac{\text{Área}[\text{mils}^2]}{\text{Espesor}[\text{oz}] * 1.378 \left[\frac{\text{mils}}{\text{oz}} \right]}$$

Sustituyendo, se obtiene que:

$$\text{Área}[\text{mils}^2] = \left(\frac{10\text{A}}{0.048 * (10^{\circ}\text{C})^{0.44}} \right)^{\frac{1}{0.725}} = 390.2035\text{mils}^2$$

$$\text{Ancho}[\text{mils}] = \frac{390.2935\text{mils}^2}{1\text{oz} * 1.378 \left(\frac{\text{mils}}{\text{oz}} \right)} = 283.231 \text{ mils}$$

Donde las constantes k, c y b para las capas externas al aire libre, según la norma IPC-2221 tienen los valores de: $k = 0.048$, $b = 0.44$ y $c = 0.725$. Sin embargo, por cuestiones de espacio se redujo el ancho mínimo a 180 milésimas, debido a que algunas pistas se superponían en los componentes pasivos de la placa.

En la Ilustración 34 se muestran las previsualizaciones de la placa de circuito impreso mediante la interfaz y herramientas del software EAGLE. La placa es de tipo “trough-hole” debido a las características de los elementos.

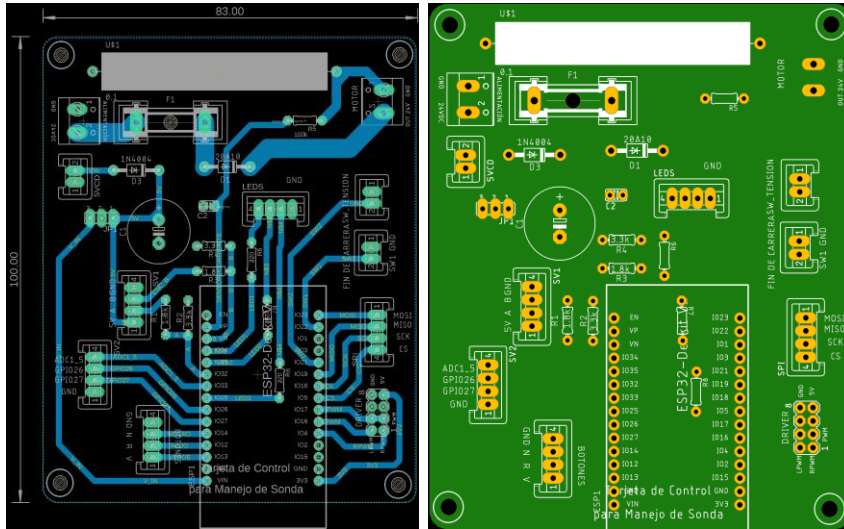


Ilustración 34 Previsualización del ruteo de las pistas y del solder mask de la placa

La resistencia de 0.1Ω no existe dentro de las librerías del software EAGLE con el empaquetado del componente, por lo que fue necesario crear uno para poder modificar el tamaño de la placa proporcionalmente.

3.3 Software

Esta sección es complemento de la anterior, debido a que se debe programar el funcionamiento del hardware para su implementación en la solución propuesta al LANRESC. El código completo funcional se encuentra en el Anexo #.

3.3.1 Función Principal

Antes de codificar el funcionamiento del sistema en el IDE de Arduino, es necesario describir los modos de operación que tendrá el dispositivo según los requerimientos del LANRESC. Los modos definidos se describen a continuación:

- MODO CASA (0): La sonda se encuentra en “CASA” cuando ocurre interrupción de SW y se reinician valores de variables de control.

- MODO AUTOMÁTICO (1): Funcionamiento automático de toma de muestras cuando el usuario ha agregado valores de velocidad, profundidad máxima, número de repeticiones del ciclo y el tiempo de espera en “CASA” previo a cada repetición.
- MODO DETENIDO (2): Motor detenido. Sonda no está en “CASA”.
- MODO BAJANDO (3): Bajar sonda manual a la velocidad preestablecida en el código de 0.05m/s.
- MODO SUBIENDO (4): Subir sonda manual a la velocidad preestablecida en el código de 0.05m/s.

Para ejemplificar mejor la interacción entre los modos de operación, se presenta el diagrama a flujo de la función “funcion_sistema()” donde se implementa la operación integral del dispositivo. En el diagrama (Ilustración 35) presentado se observa que el modo central es el “DETENIDO”, en el cual el sistema pueda funcionar dependiendo del botón que se haya presionado. Las cuatro variables que se leen y evalúan son de seguridad, las cuales son: “FIN_CARRERA”, “TENSION”, “corriente” y “distancia”.

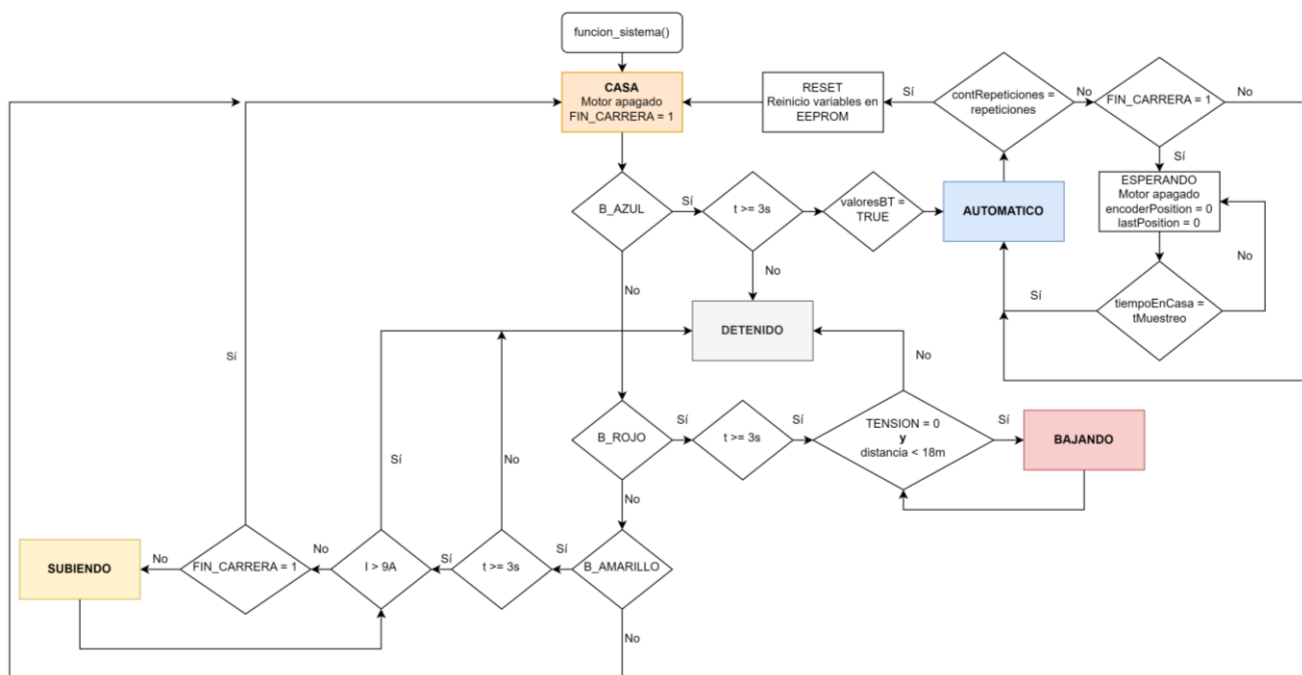


Ilustración 35 Diagrama de flujo del funcionamiento principal del prototipo

La primera hace referencia al interruptor de fin de carrera, el cual indica (mediante interrupción externa) que la sonda llegó a su altura máxima dentro de la estructura, haciendo así que se detenga el motor. La segunda es el interruptor ubicado en el subsistema de aparejos, el cual compensará la tensión y activará la interrupción externa cuando se haya llegado al límite de tensión compensada, para así detener el motor, evitar que se desenrolle el cable y se pierda la lectura del codificador.

La tercera es resultado de leer el ADC al cual está conectado el circuito sensor de corriente; si la medida es mayor que los 9A configurados, detendrá el motor para evitar daños en la fuente por el exceso de corriente. Esta variable solo se lee durante la rutina de subida, ya que es donde se presentan los mayores picos de corriente.

La última variable hace referencia a la distancia máxima que puede descender la sonda, la cual son 18m que es la profundidad máxima del pozo, sin embargo, esta puede ser configurada desde el código. Para evitar una confusión por parte del usuario al presionar cualquier botón, se programó con un máximo de tres segundos para así entrar al modo deseado.

En la función automática se define el uso de la EEPROM, sin embargo, el ESP32 ya no cuenta con esta tecnología, por lo que se utiliza la librería “Preferences” de espressif, la cual utiliza la memoria flash simulando el comportamiento de la EEPROM. Esto permite una lectura y escritura de la información más rápida. En esta memoria se guardan los valores definidos por el usuario (número de repeticiones, profundidad máxima, velocidad deseada y tiempo de espera), así como el modo de operación; esto en caso de que el dispositivo sea desconectado durante el funcionamiento automático, pueda reiniciar la posición y continuar normalmente.

3.3.2 Control

Para poder controlar el sistema, es necesario obtener el valor de la distancia que ha recorrido para conocer la velocidad en m/s de la sonda dentro del pozo. Para eso, se utiliza el codificador en la polea diseñada que se menciona en el apartado 3.1.3 y se ejecuta la siguiente operación en software:

$$distancia = \frac{(\phi_{internoPolea})(\pi)}{400ppv}$$

$$distancia = \frac{(0.036m)(3.1416)}{400ppv} = 0.000282m$$

Una vez obtenida la distancia de cable desenrollado por pulso, se procede con el cálculo de la velocidad. En la Ilustración 36 se presenta parte de las operaciones realizadas mediante código. En la primera operación “dif” ($dif = conteoActual - conteoAnterior$) se obtiene la diferencia entre el conteo actual y el anterior de las interrupciones externas que ha efectuado el codificador; después se actualiza el valor de dicho conteo en la variable “lastPosition” ($lastPosition = encoderPosition$) para el siguiente ciclo y se calcula la relación de la distancia; la cual está dada por la cantidad de pulsos desde el último ciclo por la distancia de cable desenrollado ($kDistance = distancia$).

```
dif = encoderPosition - lastPosition;
lastPosition = encoderPosition;
diferenciaM = (dif * kDistance);
VEL = diferenciaM / 0.1;
```

Ilustración 36 Algoritmo de cálculo de la velocidad.

Para conocer la velocidad, se utiliza la fórmula de la velocidad:

$$V = \frac{d}{t}$$

Donde los 100ms es el tiempo aproximado que dura un ciclo de ejecución de la función principal; es decir, cada 100ms se ejecuta función_sistema().

Durante las pruebas en el laboratorio, se comprueba que el sistema se comporta diferente al subir y al bajar la sonda; por lo que se propone dividir el control en *bajada* y en *subida*. Dentro de estas funciones se establecen diversas reglas dependiendo de la etapa en la

que estén las acciones de bajar o subir siguiendo la regla “si disminuye/aumenta drásticamente, incrementa/decrementa proporcionalmente y viceversa”:

3.3.2.1 control_subida()

Durante la subida hay dos momentos críticos:

- La sonda llegó a “CASA”
- Se perdió tensión

El primer momento está configurado desde la función “buttons()”, la cual se encarga de autorizar o no el funcionamiento dependiendo de las características y eventos del sistema. La segunda está resuelta al evaluar si el interruptor está oprimido (“0” lógico), lo cual significa que no se perdió la tensión, por lo tanto puede proceder con el aumento o decremento del PWM. Sin embargo, si se perdió la tensión (“1” lógico) se debe incrementar paulatinamente el PWM hasta que se detecte el interruptor y prosiga con el funcionamiento normal. Si se incrementa rápidamente el PWM cuando no hay tensión en el cable, este puede salirse de las poleas debido a la holgura que se genera.

El PWM se incrementa y decrementa de uno en uno (Ilustración 37), debido a que el sistema responde lento y durante la subida no se presente un problema de oscilación. Solo es necesario ajustar el PWM inicial en un valor de 80 (el PWM va de 0 a 255), ya que a partir de 100 es que comienza a notarse el movimiento.

```

if (!digitalRead(TENSION) || (esp_timer_get_time() - tiempoModo4) < tSubida) {
  if (error > 0) {
    if (error > 0.005) {
      PWM--;
      if (error > 0.02) {
        PWM--;
      }
      if (error > 0.06) {
        PWM--;
      }
    }
    if (PWM < 0) PWM = 0;
  }
  else {
    if (PWM < 80) PWM = 80;
    if (VEL == 0.00) {
      if (digitalRead(TENSION)) PWM = 175;
      else PWM = PWM + 5;
    }

    if (error < -0.005) {
      PWM++;
      if (error < -0.02) {
        PWM++;
      }
      if (error < -0.06) {
        PWM++;
      }
    }
    if (PWM > 255) PWM = 255;
  }
}
else {
  PWM = PWM - 10;
  if (PWM < 0) PWM = 0;
}
return PWM;
}

```

Ilustración 37 Función "control_subida()"

Esta función se encuentra dentro de “rutina_subida()” debido a que el “MODO AUTOMÁTICO” utiliza la misma lógica de control y la evaluación del sensor de corriente. Sin embargo, en el automático se guardan los valores del conteo actual en la EEPROM.

3.3.2.2 control_bajada()

Para controlar el sumergimiento de la sonda se necesita incrementar y decrementar en decimales el PWM (Ilustración 38), debido a que el sistema responde demasiado rápido a los

cambios, lo cual genera una oscilación en la polea haciendo muy complicado el control; por lo que se propone este método el cual redondea los valores de PWM haciendo los cambios más lentos.

```
if (!digitalRead(TENSION)) {
  if (error > 0) {
    if (error > 0.005) {
      pwmF = pwmF - 0.05;
      if (error > 0.02) {
        pwmF = pwmF - 0.1;
      }
      if (error > 0.06) {
        pwmF = pwmF - 0.5;
      }
      if (pwmF < 0)pwmF = 0;
    }
  } else {
    if (pwmF < 80)pwmF = 80;
    if (VEL == 0.00) pwmF = pwmF + 5;
    if (error < -0.005) {
      pwmF = pwmF + 0.05;
      if (error < -0.02) {
        pwmF = pwmF + 0.1;
      }
      if (error < -0.06) {
        pwmF = pwmF + 0.5;
      }
      if (pwmF > 255)pwmF = 255;
    }
  }
  } else {
    pwmF = pwmF - 10;
    if (pwmF < 0)pwmF = 0;
  }
  PWM = round(pwmF);
  return PWM;
}
```

Ilustración 38 Función "control_bajada()"

Al igual que en el control de la subida, esta función se encuentra dentro de “rutina_bajada()” debido a que comparten características de la subida en el modo automático. La diferencia de esta rutina es que no guarda datos en la EEPROM, no evalúa la corriente debido a que no demanda más de 9A durante la bajada y es necesario desactivar la interrupción del fin de carrera para evitar el ruido por el “rebote” del interruptor; de esta forma se evita que el sistema se detenga al detectar una doble interrupción de “llegada a casa”.

3.3.2.3 Función ledc

Como se menciona en el apartado 3.2.1.1 el ESP32 puede generar señales de PWM al configurar uno o más de los 16 canales de la función ledc. Para este trabajo, solo es necesario configurar un canal con las siguientes instrucciones:

```
#define PWM_OUT      17
const int freq = 20000;           //Frecuencia a 20kHz
const int ledChannel = 0;
const int resolution = 8;         //PWM de 0 a 255
```

Para configurar las funcionalidades del canal y agregarle la configuración al pin se utiliza “ledcAttachPin” donde se le ingresa el pin deseado y el **canal**:

```
ledcSetup(ledChannel, freq, resolution);
ledcAttachPin(PWM_OUT, ledChannel);
```

En el código se le asigna el PWM al motor en la función “control_motor()”, donde se le ingresa el sentido de giro “motorMovement” y el valor de PWM el cual se obtiene de “control_bajada()” y “control_subida()”. La función para asignar/reasignar un valor de PWM al pin es a través del canal, ya que esté ya está ligado al pin deseado:

```
ledcWrite(ledChannel, PWM);
```

3.3.3 Comunicación Serial

Uno de los requerimientos por parte del LANRESC es que la comunicación entre usuario-dispositivo fuese de manera sencilla, por lo que se propone utilizar una comunicación serial que permita en el futuro comunicarse por el protocolo Bluetooth, sin embargo, en los alcances de este trabajo la comunicación es por el puerto serie.

Por buenas costumbres se utiliza una función aparte dentro del código para llevar a cabo la obtención de los siguientes datos ingresados por el usuario:

- Distancia máxima (m)
- Velocidad (m/s)
- Número de repeticiones (entero positivo)
- Tiempo de espera entre repeticiones (μs)

Por lo tanto se necesita definir una estructura que contenga cuatro variables de diferentes tipos. Sin embargo, se deben convertir de tipo “String” a “int” y “float”, respectivamente; ya que ese es el tipo de variable que se obtiene de la comunicación serial. La lógica de funcionamiento se expone en la Ilustración 39

```
Valores comunicacion_serial() {
    //Variables para la comunicación serial
    bool request;           //Datos en el buffer
    int dataInit;
    int positionComma;
    String stringRead;      //Cadena leída
    String info[4];

    Valores setValues;

    request = Serial.readStringUntil('#');
    if (request == 1) {
        stringRead = Serial.readStringUntil ('\n');
        dataInit = 0;
        for (int i = 0; i < 4; i++) {
            positionComma = stringRead.indexOf(',', dataInit);
            info[i] = stringRead.substring(dataInit, positionComma);
            dataInit = positionComma + 1;
        }
        setValues = {info[0].toInt(), info[1].toFloat(), info[2].toFloat(), info[3].toFloat()};
    }
    return setValues;
}
```

Ilustración 39 Función "comunicación_serial()"

El tipo de variable “Valores” es la estructura definida que cuenta con un variable de tipo entera y tres de tipo flotante. Y los valores que se utilizan en el programa se obtiene del arreglo “info[4]” y se asignan a la variable “setValues” de tipo Valores.

3.3.4 Sensor de corriente

Para obtener el valor de la corriente que está consumiendo el motor, se utiliza el canal 1 del ADC, específicamente el pin 36. El ADC del ESP32 presenta una falla por diseño, la cual es que el ADC no es lineal; por lo que se debe ajustar para poder obtener un valor útil.

Para este caso específico se hizo un mapeo de los valores que obtiene el ADC durante los diferentes picos de corriente y promediándolos para reducir el ruido de las muestras, obteniendo así la función que se expone en la Ilustración 40:

```
void sensar_corriente() {  
    float valorADC;  
    float corriente;  
    valorADC = 0;  
    valorADC = analogRead(CORRIENTE);  
    corriente = 0;  
    for (int i = 4; i > 0; i--) {  
        valorCorriente[i] = valorCorriente[i-1];  
        corriente = corriente + valorCorriente[i];  
    }  
    corriente = corriente + valorADC;  
    valorCorriente[0] = corriente/5;  
    corriente = 2.63 + (0.011*valorCorriente[0]);  
    if(corriente > 9){modo = DETENIDO; sobreCorriente = true; }  
}
```

Ilustración 40 Función "sensar_corriente()"

Es en esta función donde se decide si se detiene el motor al detectar la sobre corriente y activa una bandera de exceso de corriente; lo que significa que el paso de la sonda está obstruido o la sonda se atoró entre alguna raíz que haya atravesado las paredes del pozo.

4 Resultados

En este capítulo se presentan los resultados que se han realizado en el pozo ubicado dentro de las instalaciones del Instituto de Ingeniería de la UNAM del Campus Sisal, Yucatán. Para esto, el prototipo se coloca sobre la boca del pozo, tal como se muestra en la Ilustración 41. Se prosigue a ejecutar las pruebas en el siguiente orden de eventos:

1. Bajar la sonda fuera del tubo de PVC.
2. Detener el ascenso a propósito simulando un atascamiento.
3. Realizar un ciclo completo de funcionamiento.
4. Cargar valores de usuario y ejecutar función automática.
5. Desconectar el dispositivo durante la función automática



Ilustración 41 Prototipo montado sobre el pozo a monitorear

Para las pruebas en el pozo se utiliza un tubo de PVC sellado con las dimensiones y características de las sondas YSI EXO 1 y 2. Durante todas las pruebas se observan los valores de corriente, velocidad y distancia recorrida mediante el puerto serie entre una computadora y el ESP32.

4.1 Primer y segundo evento

En el primer evento se comprueba el funcionamiento del inicio de la lógica de sumergimiento, pasados los tres segundos de haber presionado el botón ROJO, donde sucede la desactivación de la interrupción de “llegada a casa” (FIN_CARRERA) y el aumento momentáneo del PWM, el cual inicia en 80 y aumenta en 5 cada 100ms hasta que la velocidad sea superior a cero; esto para generar el impulso suficiente para iniciar el movimiento. Es necesario desactivar la interrupción FIN_CARRERA, ya que se genera un rebote haciendo que el motor se detenga inmediatamente sale del resguardo.

En el segundo evento se evalúa la lectura de corriente del ADC a través del puerto serie. Para esto, se acciona el botón AMARILLO durante tres segundos, lo que detiene el motor durante este tiempo para después comenzar la subida. Antes que la sonda reingrese al tubo de PVC, se detiene con la mano aumentando la tensión en el cable y generando un pico de corriente superior a los 9A, lo cual detiene el motor y entra en el modo “DETENIDO”.

Durante estas pruebas, se observa que durante la subida de la sonda, el motor no consume más de 9A; por lo que se comprueba que el límite de corriente propuesto en el laboratorio es el adecuado.

4.2 Tercer evento

Para la ejecución de esta prueba, es necesario que la sonda esté en el modo “CASA”. Se presiona el botón rojo durante tres segundos para que el motor haga bajar la sonda hasta que llegue a los 18m o hasta que el arreglo de poleas tensoras absorba la máxima pérdida de tensión configurada en el laboratorio; dicho arreglo se muestra en la Ilustración

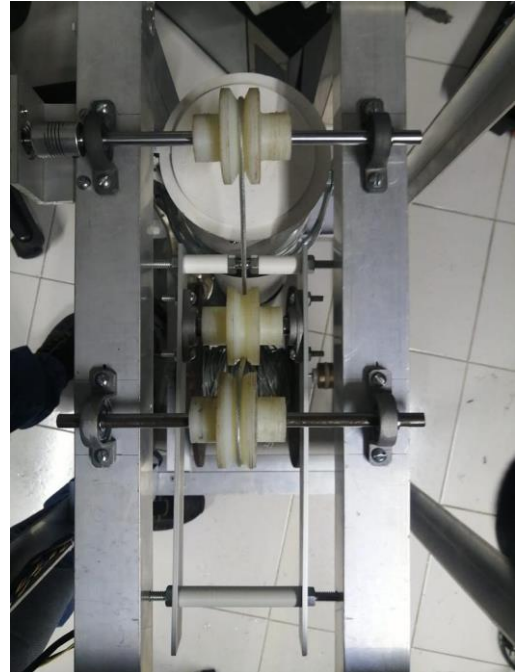
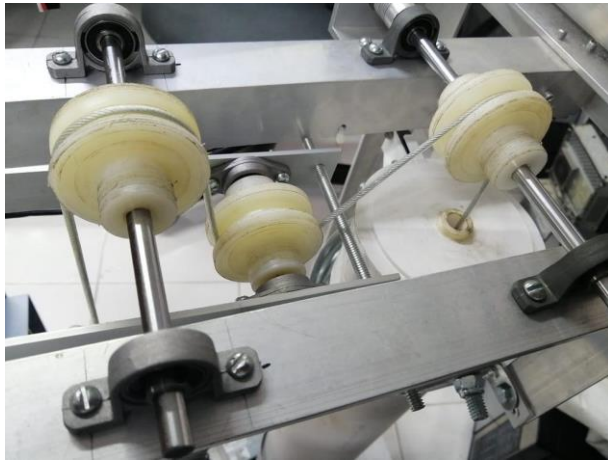


Ilustración 42 Arreglo de poleas tensoras

Los resultados que se obtienen demuestran que la calibración del interruptor a 210° , aproximadamente, respecto al eje de rotación de la estructura móvil; logra detectar a tiempo la pérdida de tensión, así como mitigarla y permitir que vuelva a obtener tensión el cable mientras la sonda sigue su descenso.

Una vez se desactiva la interrupción, el motor se detiene e inicia la segunda etapa de este evento. Se mantiene presionado el botón AMARILLO durante 3 segundos y se monitorean los valores de corriente, velocidad y distancia en el monitor serial hasta que se desactive la interrupción de FIN_CARRERA lo cual manda al sistema al modo “CASA”.

Se obtiene que el sistema sigue oscilando lo suficiente para generar errores en la bajada, los cuales ocasionan que el descenso sea interrumpido por lapsos de 3 a 5 segundos aproximadamente. Sin embargo, la velocidad de la subida es corregida a tiempo por el algoritmo de control, teniendo una velocidad media de 0.055m/s .

4.3 Cuarto y quinto evento

Con la sonda en modo “CASA” se envía la siguiente cadena de prueba:

#5,5,0.05,4000000

Donde el primer valor es el número de repeticiones que la sonda saldrá del resguardo, bajará hasta la distancia máxima (segundo valor) a la velocidad establecida (tercer valor) y regresará a la misma velocidad. Cuando esté en resguardo esperará a que el tiempo establecido (cuarto valor) se cumpla para repetir el proceso.

Durante dicha prueba se monitorea la velocidad, distancia, corriente en la subida, tiempo de espera y el conteo de repeticiones.

En el último evento se desconecta el equipo durante 5 segundos en tres etapas diferentes del funcionamiento:

- Bajando: la sonda comienza el ascenso para reiniciar las variables y restarle una repetición para continuar con la función automática.
- Subiendo: la sonda termina el ascenso, se reinician las variables pero no se resta el conteo de repeticiones. Prosigue con su funcionamiento normal.
- En resguardo: Se reinicia el tiempo de espera.

Los resultados que se obtienen son similares a los del tercer evento, debido a que el algoritmo utiliza las mismas funciones pero con la rutina automática. Sin embargo, al LANRESC le es de mayor utilidad que el usuario ingrese la profundidad mínima de la sonda, ya que esta debe ser el “set-point” en lugar del resguardo, debido a que el fabricante recomienda que las terminales de los sensores estén siempre en contacto con el agua.

5 Conclusiones y trabajos futuros

Esta tesis presenta una solución a la medida de los requerimientos del LANRESC para automatizar el monitoreo de los pozos, lo cual es necesario para realizar de manera más sencilla la obtención de los datos utilizados en las diferentes investigaciones que se trabajan en el Instituto de Ingeniería de la UNAM en Sisal, Yucatán.

En respuesta a los objetivos específicos, se obtiene dicha solución utilizando e implementando tecnologías existentes como lo son: el control de la velocidad mediante PWM, un método de control por retroalimentación de un codificador rotativo incremental con reglas definidas según el error en la velocidad, el diseño de un circuito resistivo con filtro pasa bajas para medir la corriente y una tarjeta de circuito impreso para la interconexión de los componentes.

De igual forma, el código que se implementa en el entorno de Arduino permite al usuario modificar las variables de velocidad, profundidad máxima, tiempo de operación y número de repeticiones; dejando así preparado para trabajos futuros, el ingreso de estos datos utilizando el protocolo Bluetooth. También se explora utilizar una parte de la memoria flash del ESP32 como memoria EEPROM a través de la librería “Preferences”, la cual permite la rápida lectura y escritura de la información, esto con la finalidad de que el sistema sea lo más autónomo posible.

Por último, se presenta una estructura de aluminio, lo que permite que sea ligera y resistente al clima costero. Dentro de esta estructura se diseña e implementa un arreglo de poleas para absorber diferentes pérdidas de tensión, el cual también sirve para detectar que la sonda ha llegado al fondo.

Basándose en los resultados presentados en el capítulo 4, se concluye que este prototipo responde adecuadamente a las necesidades actuales del LANRESC; sin embargo, se encuentran los siguientes puntos a mejorar en trabajos futuros:

- Aplicación y protocolo Bluetooth
- Agregar la variable de profundidad mínima al algoritmo de función automática.
- Mejorar la estructura con respecto a puntos como:
 - Anclaje para seguridad ante eventos climáticos

- Diseño de cubierta protectora para funcionamiento en intemperie
 - Rediseño general de acuerdo con resultados previos para mejorar características de potencia mecánica y consumo energético.
- Mejorar el control. Incluir un análisis del sistema, elegir el modelo de controlador óptimo y realizar el diseño e implementación de este.
- Realizar la caracterización y modelado energético para estimación parametrizada de consumo de potencia y autonomía.

6 Referencias

- [1] M. D. R. L. M. R. S. Behmel, «Water quality monitoring strategies - A review and future perspectives,» *Science of the Total Environment*, vol. 571, pp. 1312-1329, 2016.
- [2] S. T. d. L. S. D. H. d. S. Á. P. F. d. N. J. M. V. B. d. S. S. J. E. Á. J. a. L. M. G. G. Andouglas Goncalves da Silva Junior, «Towards a Real-Time Embedded System for Water Monitoring Installed in a Robotic Sailboat,» *sensors*, vol. 16, p. 19, 8 August 2016.
- [3] E. d. l. R. A. J. S. N. a. J. G. J. Fernando D. Von Borstel Luna, «Robotic System for Automation of Water Quality Monitoring and Feeding in Aquaculture Shadehouse,» *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 47, nº 7, pp. 1575-1589, July 2017.
- [4] M. B. N. C. B. M. S. P. K. P.-K. A. M. S. a. R. L. W. Dawn A. Shivaly, «Prototypic automated continuous recreational water quality monitoring of nine Chicago beaches,» *Journal of Environmental Management*, vol. 166, pp. 285-293, 15 January 2015.
- [5] D. H. Yiheng Chen, «Water quality monitoring in smart city: A pilot project,» *Automation in Construction*, vol. 89, pp. 307-316, May 2018.
- [6] A. M. D. S. W. K. C. K. F. W. Y. S. M. N. C. Kusmayanato Hadi Wibowo, «IoT-based Control and Monitoring for DC Motor Fed by Photovoltaic System,» *IOP Conference Series: Materials Science and Engineering*, vol. 588, 2019.
- [7] M. H. A. A. S. Shashi Bhushan Kumar, «Design and Simulation of Speed Control of DC Motor by Artificial Neuronal Network Technique,» *International Journal of Scientific and Research Publications*, vol. 4, nº 7, July 2014.
- [8] YSI, «6950 Fixed Vertical Profiler,» 2022. [En línea]. Available: <https://www.ysi.com/6950fixedverticalprofiler>. [Último acceso: 17 July 2022].
- [9] M.A. Tosini et al, «Metodologías de Diseño para Sistemas Embebidos,» *Workshop de Investigadores en ciencias de la comunicación*, vol. XV, pp. 717 - 737, 2013.
- [10] T. D. Morton, *Embedded Microcontroller*, Prentice Hall, 2000.
- [11] T. Wilmshurst, *An Introduction to the Design of Small-Scale Embedded Systems with examples from PIC, 80C51 and 68HC05/08 Microcontrollers*, Palgrave Foundations, 2003.

- [12] D. P. Abreu, *Sistemas Embebidos y Sistemas Operativos Embebidos*, Coimbra: Escuela de Computación - UCV, 2009.
- [13] F. V. Givargis, *Embedded System Design: A Unified Hardware/Software Approach*, Riverside: Department of Computer Science and Engineering - University of California, 1999.
- [14] C. Hallinan, *Embedded Linux Primer: A Practical, Real-World Approach*, Segunda ed., Prentice Hall, 2010.
- [15] D. J. P. Steven F. Barret, *Microcontrollers Fundamentals for Engineers and Scientists*, Suiza: Springer Charm, 2006, p. 115.
- [16] Technopedia, «Technopedia,» 16 junio 2017. [En línea]. Available: <https://www.techopedia.com/definition/8180/machine-cycle>. [Último acceso: 02 febrero 2022].
- [17] Rohde & Schwarz, «Rohde & Schwarz,» 27 enero 2020. [En línea]. Available: https://www.rohde-schwarz.com/lat/productos/prueba-y-medicion/osciloscopios/educational-content/entendiendo-el-uart_254524.html. [Último acceso: 02 febrero 2022].
- [18] espressif, «ESP-IDF Programming Guide,» 20 Julio 2022. [En línea]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/hw-reference/esp32/get-started-devkitc.html>. [Último acceso: Julio 2022].
- [19] espressif, «LED Control (LEDC),» 8 febrero 2022. [En línea]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/ledc.html#ledc-api-supported-range-frequency-duty-resolution>. [Último acceso: agosto 2022].
- [20] Guemisa, «Sensores e Instrumentación Guemisa,» 2018. [En línea]. Available: <https://www.guemisa.com/sicod/docus/ENCODER-TEC.pdf>. [Último acceso: Agosto 2022].
- [21] Infineon Technologies, «BTS 7960 High Current PN Half Bridge,» München, 2004.
- [22] DigiKey, «"Conversión de ancho de trazo PCB",» [En línea]. Available: <https://www.digikey.com.mx/es/resources/conversion-calculators/conversion-calculator-pcb-trace-width>. [Último acceso: 28 enero 2022].
- [23] espressif, «Motor Control Pulse Width Modulator (MCPWM) - ESP32 - — ESP-IDF Programming Guide latest documentation,» Espressif Systems, [En línea]. Available: <https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/mcpwm.html#fault-handler>. [Último acceso: 21 enero 2022].

7 Anexos

7.1 Planos de la estructura metálica

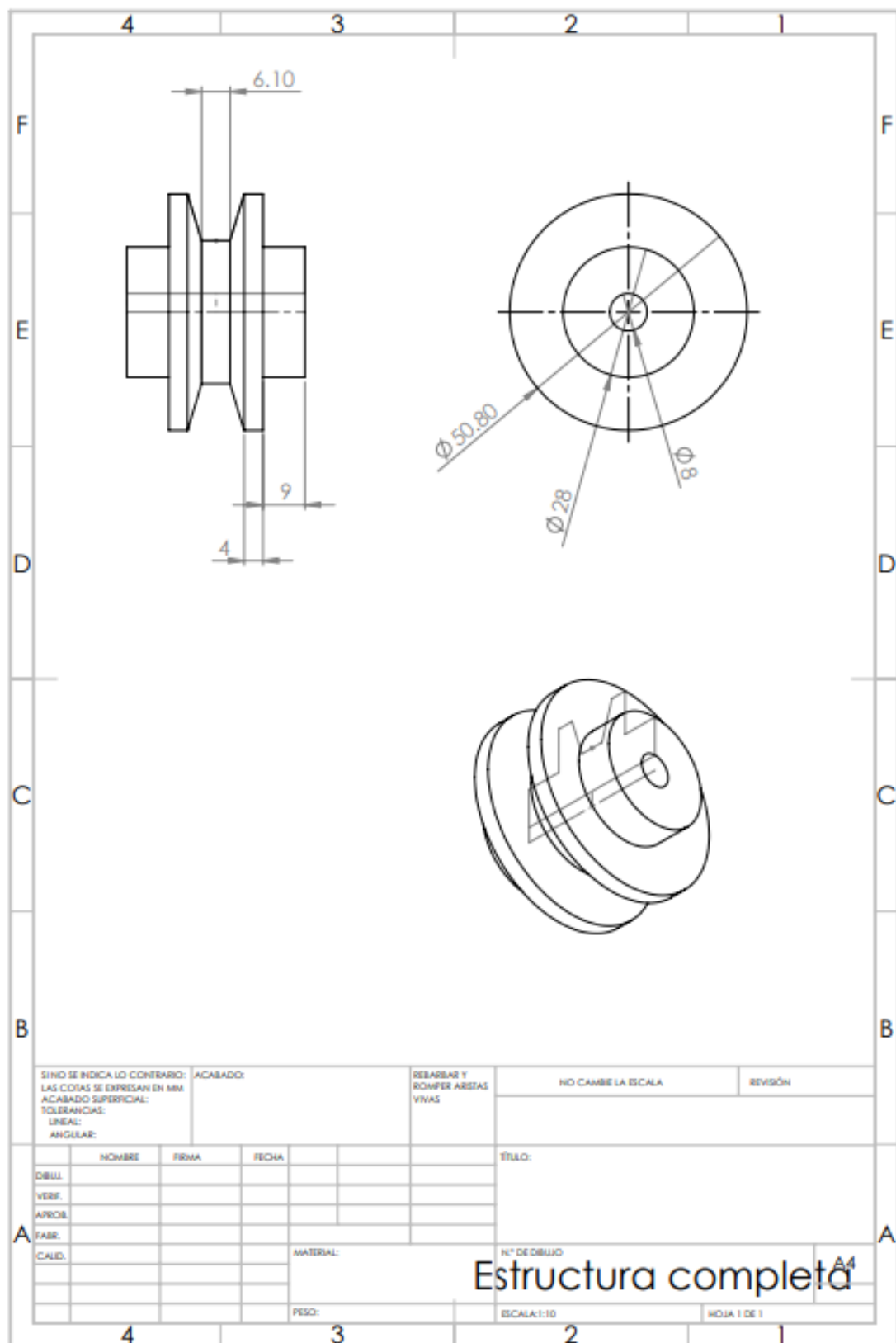
Technical drawing of a mechanical part. The top view shows a rectangular base with a central raised section. Dimensions: 90mm (left base), 150mm (central raised section), 130mm (right base). The side view shows a cylindrical shape with a diameter of 50.80mm and a central hole with a diameter of 8mm. The drawing is labeled "Estructura completa" and "A4".

4	3	2	1
F			F
E			E
D			D
C			C
B			B
<p>SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:</p>		<p>ACABADO:</p>	<p>REBAÑAR Y ROMPER ARISTAS VIVAS</p>
<p>NO CAMBE LA ESCALA</p>		<p>REVISIÓN</p>	
<p>TÍTULO:</p>		<p>N.º DE DIBUJO</p>	
<p>ESCALA: 1:10</p>		<p>HOJA 1 DE 1</p>	
4	3	2	1

Estructura completa

4	3	2	1																								
F			F																								
E			E																								
D			D																								
C			C																								
B			B																								
<p>SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:</p>		<p>ACABADO:</p>	<p>REBARBAR Y ROMPER ARISTAS VIVAS</p>																								
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 25%;">NOMBRE</td> <td style="width: 25%;">FIRMA</td> <td style="width: 25%;">FECHA</td> <td style="width: 25%;"></td> </tr> <tr> <td>DIBUJ.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>VERIF.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>APROB.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>FABR.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>CAUD.</td> <td></td> <td></td> <td></td> </tr> </table>		NOMBRE	FIRMA	FECHA		DIBUJ.				VERIF.				APROB.				FABR.				CAUD.				<p>NO CAMBIE LA ESCALA</p>	<p>REVISIÓN</p>
NOMBRE	FIRMA	FECHA																									
DIBUJ.																											
VERIF.																											
APROB.																											
FABR.																											
CAUD.																											
<p>TÍTULO:</p>		<p>Nº DE DIBUJO</p>																									
<p>MATERIAL:</p>		<p>ESCALA: 1:10</p>																									
<p>PESO:</p>		<p>HOJA 1 DE 1</p>																									
4	3	2	1																								

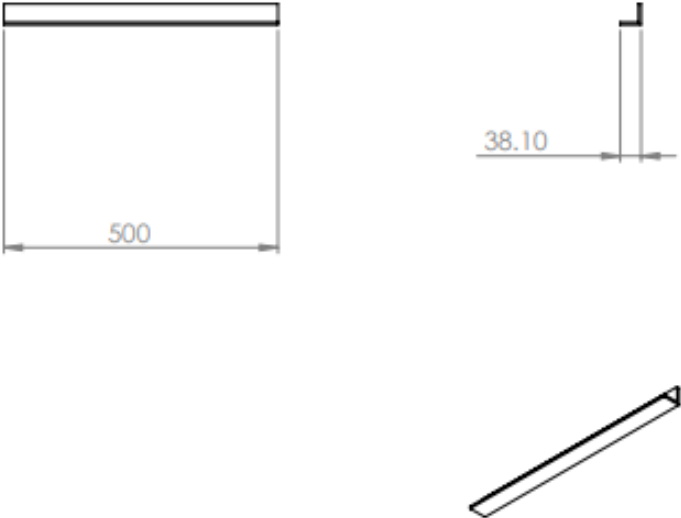
Estructura completa



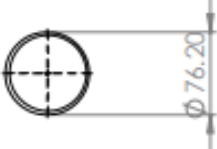


4	3	2	1																																				
F			F																																				
E			E																																				
D			D																																				
C			C																																				
B			B																																				
<p>SI NO SE INDICA LO CONTRARIO: LAS COSAS SE EXPRESAN EN MM. ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:</p>		<p>ACABADO:</p>	<p>REBARBAR Y ROMPER ARISTAS VIVAS</p>																																				
		NO CAMBE LA ESCALA	REVISIÓN																																				
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 10%;">NOMBRE</th> <th style="width: 10%;">FIRMA</th> <th style="width: 10%;">FECHA</th> <th style="width: 10%;"></th> <th style="width: 10%;"></th> <th style="width: 10%;"></th> </tr> <tr> <td>DEBIL.</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>VERIF.</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>APROB.</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>FABR.</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td>CAUD.</td> <td></td> <td></td> <td></td> <td></td> <td></td> </tr> </table>		NOMBRE	FIRMA	FECHA				DEBIL.						VERIF.						APROB.						FABR.						CAUD.						<p>TÍTULO:</p>	
NOMBRE	FIRMA	FECHA																																					
DEBIL.																																							
VERIF.																																							
APROB.																																							
FABR.																																							
CAUD.																																							
		<p>Nº DE DIBUJO</p> <h1 style="margin: 0;">Estructura completa</h1>																																					
		<p>ESCALA: 1:10</p>	<p>HOJA 1 DE 1</p>																																				
4	3	2	1																																				

4	3	2	1
F			F
E			E
D			D
C			C
B			B
SI NO SE INDICA LO CONTRARIO: LAS COSAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS
		NO CAMBIE LA ESCALA	
		REVISIÓN	
		TÍTULO:	
		N° DE DIBUJO	
		ESCALA: 1:10	
		HOJA 1 DE 1	
4	3	2	1
A			A

4		3		2		1	
F							F
E							E
D							D
C							C
B							B
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM. ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		ACABADO:		REBARBAR Y ROMPER ARISTAS VIVAS		NO CAMBE LA ESCALA	
						REVISIÓN	
						TÍTULO:	
NOMBRE FIRMA FECHA DIBU. VERIF. APROB. FABR. CALID.		MATERIAL:		PESO:		N° DE DIBUJO ESCALA: 1:10 HOJA 1 DE 1	
						Estructura completa A4	
4		3		2		1	

4	3	2	1																								
F			F																								
E			E																								
D			D																								
C			C																								
B			B																								
																											
SI NO SE INDICA LO CONTRARIO: LAS COSAS SE EXPRESAN EN MM. ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS																								
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">NOMBRE</td> <td style="width: 20%;">FIRMA</td> <td style="width: 20%;">FECHA</td> <td style="width: 50%;"></td> </tr> <tr> <td>DISEÑ.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>VERIF.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>APROB.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>FABR.</td> <td></td> <td></td> <td></td> </tr> <tr> <td>CALID.</td> <td></td> <td></td> <td></td> </tr> </table>		NOMBRE	FIRMA	FECHA		DISEÑ.				VERIF.				APROB.				FABR.				CALID.				NO CAMBIE LA ESCALA	REVISIÓN
NOMBRE	FIRMA	FECHA																									
DISEÑ.																											
VERIF.																											
APROB.																											
FABR.																											
CALID.																											
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <td style="width: 10%;">MATERIAL:</td> <td style="width: 90%;"></td> </tr> <tr> <td>PESO:</td> <td></td> </tr> </table>		MATERIAL:		PESO:		TÍTULO: <div style="text-align: right;">A4</div>																					
MATERIAL:																											
PESO:																											
Estructura completa		ESCALA: 1:10																									
4	3	2	1																								
A			A																								

4	3	2	1																					
F			F																					
E			E																					
D			D																					
C			C																					
B			B																					
SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:		ACABADO:	REBARBAR Y ROMPER ARISTAS VIVAS																					
		NO CAMBIE LA ESCALA	REVISIÓN																					
<table border="1" style="width: 100%; border-collapse: collapse;"> <tr> <th style="width: 10%;">NOMBRE</th> <th style="width: 10%;">FIRMA</th> <th style="width: 10%;">FECHA</th> </tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> <tr><td> </td><td> </td><td> </td></tr> </table>		NOMBRE	FIRMA	FECHA																			TÍTULO: 	
NOMBRE	FIRMA	FECHA																						
CALD. MATERIAL:		N° DE DIBUJO <div style="font-size: 24pt; font-weight: bold;">Estructura completa</div>																						
PESO:		ESCALA: 1:10																						
4		1																						

4				3				2				1							
F																		F	
E																		E	
D																		D	
C																		C	
B																		B	
A																		A	
4		3				2				1									

101.60

25.40

SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM. ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:				ACABADO:				REBARBAR Y ROMPER ARISTAS VIVAS				NO CAMBIE LA ESCALA				REVISIÓN															
DIBUJ.				NOMBRE				FIRMA				FECHA				TÍTULO:															
VERIF.																<div style="font-size: 1.5em; font-weight: bold;">Estructura completa</div> <div style="font-size: 0.8em;">A4</div>															
APROB.																															
FABR.																															
CALID.																															
																MATERIAL:				N° DE DIBUJO				ESCALA: 1:10				HOJA 1 DE 1			
																PESO:															

7.2 Algoritmo del funcionamiento del sistema

```
#include <Preferences.h>

#define IENC_A      32 //GPIO de interrupción del encoder
#define ENC_B      39
#define FIN_CARRERA 22 //Fin de carrera
#define G_BUTTON   13 //Azul físico
#define R_BUTTON   12 //Amarillo Físico
#define B_BUTTON   14 //Rojo físico
#define TENSION    21 //Switch sensor de tensión
#define CORRIENTE  36 //Resistencia sensora de corriente
#define PWM_OUT    17 //Salida de PWM. GPIO cortocircuitado
#define MOT_A      16 //LPWM
#define MOT_B      4  //RPWM

#define PARAR      0
#define BAJAR      1
#define SUBIR      2

// setting PWM properties
const int freq = 20000;
const int ledChannel = 0;
const int resolution = 8;

//Variables de tiempo
unsigned long ts = 100 * 1000;
uint64_t last_ts = 0.00;
unsigned long B_time = 3000 * 1000;
uint64_t last_timeB = 0;
uint64_t tiempoEnCasa;
unsigned long tSubida = 5000 * 1000;
```

```

uint64_t tiempoModo4;

//Variables del encoder
int lastStateENC_A = LOW;
int stateENC_A = LOW;
volatile long encoderPosition;
int Position = 0;
long diametter = 0.036;
const double kDistance = (0.036 * 3.1416) / 400; //Diámetro*pi/pasos

//Variables del control
long lastPosition = 0;
int PWM;
float valorCorriente[5];
long dif = 0;
float distancia = 0;
float VEL = 0.05; //setpoint
float error = 0;
int contRepeticiones;
float pwmF;

enum {
    CASA,
    AUTOMATICO,
    DETENIDO,
    BAJANDO,
    SUBIENDO
} modo;

enum {
    INICIO_A,

```

```

    ESPERANDO_A,
    BAJANDO_A,
    SUBIENDO_A,
    FIN_A,
    RESET_A
} fOperacionAuto;

bool valoresBT = false;
bool probeStatus = false;
unsigned long contCiclo;
bool funcionAuto;      //Error por falla en la energía.
bool intDesactivada;
bool sobreCorriente;

//variables para botones
bool B_Status;
bool lastG, lastR, lastB;
bool pulsoBotones;

//Variables de estado del motor
int motorMovement = 0;      //0 = stop; 1 = down; 2 = up
bool tope = LOW;

//Variables modificadas por el usuario, d = m, v = m/s, t = us.
float repeticiones;
float dMax;
float velDeseada;
float tMuestreo;

struct Valores {
    int repeticiones_;

```

```

float dMax_;
float velDeseada_;
float tMuestreo_;
};
Valores valoresUsuario;

Preferences datosGuardados;

//Interrupción del encoder
void IRAM_ATTR ai0() {
  if (digitalRead(ENC_B) == LOW) {
    encoderPosition++;
  } else {
    encoderPosition--;
  }
}

//Interrupción fin de carrera, Detecta cuando llega la sonda
void IRAM_ATTR int1() {
  if (digitalRead(FIN_CARRERA) == HIGH) {    //Switch NC = 0; al abrir da 1,
interrupción flanco de subida
    probeStatus = true;    //True = sonda guardada; False = sonda está en el pozo
  }
}

void setup() {
  ledcSetup(ledChannel, freq, resolution);
  ledcAttachPin(PWM_OUT, ledChannel);

  //Inicialización de variables
  B_Status = false;

```

```

lastG = false; lastR = false; lastB = false;
pulsoBotones = false;
contCiclo = 0;
encoderPosition = 0;
lastPosition = 0;
if (digitalRead(FIN_CARRERA) == HIGH) modo = CASA;
else modo = DETENIDO;
PWM = 0;
intDesactivada = false; //Si es false, la interrupción del fin de carrera está activada
contRepeticiones = 0;

Serial.begin(115200);
Serial.setTimeout(100);

pinMode(IENC_A, INPUT);
pinMode(ENC_B, INPUT);
pinMode(PWM_OUT, OUTPUT);
pinMode(MOT_A, OUTPUT);
pinMode(MOT_B, OUTPUT);
pinMode(G_BUTTON, INPUT_PULLUP);
pinMode(R_BUTTON, INPUT_PULLUP);
pinMode(B_BUTTON, INPUT_PULLUP);
pinMode(FIN_CARRERA, INPUT_PULLUP);
pinMode(TENSION, INPUT_PULLUP);

attachInterrupt(IENC_A, ai0, RISING);
attachInterrupt(FIN_CARRERA, int1, RISING);

datosGuardados.begin("grua_sonda", false);

if (digitalRead(FIN_CARRERA)) modo = CASA;

```



```

if (datosGuardados.getInt("modo", DETENIDO) == AUTOMATICO) {
    repeticiones = datosGuardados.getInt("repeticiones", 0);
    dMax = datosGuardados.getFloat("dmax", 0);
    velDeseada = datosGuardados.getFloat("veldeseada", 0);
    tMuestreo = datosGuardados.getFloat("tmuestreo", 0);
    contRepeticiones = datosGuardados.getInt("contador", -1);
    modo = AUTOMATICO;
    fOperacionAuto = RESET_A;
    valoresBT = true;
}

}

void loop() {
    if (Serial.available() > 0 && motorMovement == PARAR) {
        valoresUsuario = comunicacion_serial();
        repeticiones = valoresUsuario.repeticiones_;
        dMax = valoresUsuario.dMax_;
        velDeseada = valoresUsuario.velDeseada_;
        tMuestreo = valoresUsuario.tMuestreo_;
        valoresBT = true;
        datosGuardados.putInt("repeticiones", repeticiones);
        datosGuardados.putFloat("dmax", dMax);
        datosGuardados.putFloat("veldeseada", velDeseada);
        datosGuardados.putFloat("tmuestreo", tMuestreo);
    }
    if ((esp_timer_get_time() - last_ts) >= ts) {
        contCiclo = 0;
        funcion_sistema();
        last_ts = esp_timer_get_time();
    }
}

```

```

    contCiclo++;
}

void buttons() {
    bool GB, RB, BB;
    GB = !digitalRead(G_BUTTON);
    RB = !digitalRead(R_BUTTON);
    BB = !digitalRead(B_BUTTON);
    if ((GB && !lastG) || (RB && !lastR) || (BB && !lastB)) {
        if (modo != CASA) {
            modo = DETENIDO;
            datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
        }
        last_timeB = esp_timer_get_time();
    }
    if (GB && lastG && valoresBT) {
        if (esp_timer_get_time() - last_timeB >= B_time) {
            modo = AUTOMATICO;
            fOperacionAuto = INICIO_A;
            datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
            datosGuardados.putBool("funcionAuto", funcionAuto);
        }
    }
    if (RB && lastR) {
        if ((esp_timer_get_time() - last_timeB >= B_time) && modo != 0) {
            modo = SUBIENDO;
            tiempoModo4 = esp_timer_get_time();
            datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
        }
    }
    if (BB && lastB) {
        if (esp_timer_get_time() - last_timeB >= B_time) {

```

```

        modo = BAJANDO;
        datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
    }
}

lastG = GB;
lastR = RB;
lastB = BB;
}

void buttons() {
    bool GB, RB, BB;
    GB = !digitalRead(G_BUTTON);
    RB = !digitalRead(R_BUTTON);
    BB = !digitalRead(B_BUTTON);
    if ((GB && !lastG) || (RB && !lastR) || (BB && !lastB)) {
        if (modo != CASA) {
            modo = DETENIDO;
            datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
        }
        last_timeB = esp_timer_get_time();
    }
    if (GB && lastG && valoresBT) {
        if (esp_timer_get_time() - last_timeB >= B_time) {
            modo = AUTOMATICO;
            fOperacionAuto = INICIO_A;
            datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
            datosGuardados.putBool("funcionAuto", funcionAuto);
        }
    }
    if (RB && lastR) {

```

```

    if ((esp_timer_get_time() - last_timeB >= B_time) && modo != 0) {
        modo = SUBIENDO;
        tiempoModo4 = esp_timer_get_time();
        datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
    }
}

if (BB && lastB) {
    if (esp_timer_get_time() - last_timeB >= B_time) {
        modo = BAJANDO;
        datosGuardados.putInt("modo", modo); //Guardar valor en la EEPROM
    }
}

lastG = GB;
lastR = RB;
lastB = BB;
}

int control_bajada(float S_VEL) {
    double diferenciaM;
    double velAnterior = 0;
    bool velProm = false;

    dif = encoderPosition - lastPosition;
    lastPosition = encoderPosition;
    diferenciaM = (dif * kDistance);
    VEL = diferenciaM / 0.1;
    velAnterior = VEL;
    if (VEL < 0) VEL = -VEL;
    error = VEL - S_VEL;          //Error+ si VEL > U_VEL -> BAJAR VEL
                                //Error- si VEL < U_VEL -> SUBIR VEL

```

```

if (!digitalRead(TENSION)) {
  if (error > 0) {
    if (error > 0.005) {
      pwmF = pwmF - 0.05;
      if (error > 0.02) {
        pwmF = pwmF - 0.1;
      }
      if (error > 0.06) {
        pwmF = pwmF - 0.5;
      }
      if (pwmF < 0)pwmF = 0;
    }
  } else {
    if (pwmF < 80)pwmF = 80;
    if (VEL == 0.00) pwmF = pwmF + 5;
    if (error < -0.005) {
      pwmF = pwmF + 0.05;
      if (error < -0.02) {
        pwmF = pwmF + 0.1;
      }
      if (error < -0.06) {
        pwmF = pwmF + 0.5;
      }
      if (pwmF > 255)pwmF = 255;
    }
  }
} else {
  pwmF = pwmF - 10;
  if (pwmF < 0)pwmF = 0;
}
PWM = round(pwmF);

```

```

    return PWM;
}

void motorControl(int motorMovement, int PWM) { //0 stop; 1 abajo; 2 arriba
    switch (motorMovement) {
        case 0:
            digitalWrite(MOT_A, LOW);
            digitalWrite(MOT_B, LOW);
            ledcWrite(ledChannel, 0);
            break;
        case 1:
            digitalWrite(MOT_A, HIGH);
            digitalWrite(MOT_B, LOW);
            ledcWrite(ledChannel, PWM);
            break;
        case 2:
            if (!digitalRead(FIN_CARRERA)) {
                digitalWrite(MOT_A, LOW);
                digitalWrite(MOT_B, HIGH);
                ledcWrite(ledChannel, PWM);
            }
            break;
        default:
            digitalWrite(MOT_A, LOW);
            digitalWrite(MOT_B, LOW);
            ledcWrite(ledChannel, 0);
            break;
    }
}

int control_subida(float S_VEL) {

```

```

double diferenciaM;

dif = encoderPosition - lastPosition;
lastPosition = encoderPosition;
diferenciaM = (dif * kDistance);
VEL = diferenciaM / 0.1;
if (VEL < 0)VEL = -VEL;
error = VEL - S_VEL;          //Error+ si VEL > U_VEL -> BAJAR VEL
                               //Error- si VEL < U_VEL -> SUBIR VEL
if (!digitalRead(TENSION) || (esp_timer_get_time() - tiempoModo4) < tSubida) {
  if (error > 0) {
    if (error > 0.005) {
      PWM--;
      if (error > 0.02) {
        PWM--;
      }
      if (error > 0.06) {
        PWM--;
      }
      if (PWM < 0)PWM = 0;
    }
  } else {
    if (PWM < 80)PWM = 80;
    if (VEL == 0.00) {
      if (digitalRead(TENSION)) PWM = 175;
      else PWM = PWM + 5;
    }

    if (error < -0.005) {
      PWM++;
      if (error < -0.02) {

```

```

        PWM++;
    }
    if (error < -0.06) {
        PWM++;
    }
    if (PWM > 255) PWM = 255;
}
}
} else {
    PWM = PWM - 10;
    if (PWM < 0) PWM = 0;
}
return PWM;
}

```

```

void funcion_sistema() {
    float velocidad = 0.05;
    float disMax = 20;
    buttons();
    switch (modo) {
        case CASA:
            PWM = 0;
            motorControl(0, 0);    //Motor detenido. Cero velocidad.
            encoderPosition = 0;    //Reinicio valor de la posición.
            lastPosition = 0;
            break;
        case AUTOMATICO:
            switch (fOperacionAuto) {
                case INICIO_A:
                    contRepeticiones = 0;
                    datosGuardados.putInt("contador", contRepeticiones);

```



```

    fOperacionAuto = BAJANDO_A;
    break;
case ESPERANDO_A:
    PWM = 0;
    motorControl(PARAR, PWM);    //Motor detenido. Cero velocidad.
    encoderPosition = 0;        //Reinicio valor de la posición.
    lastPosition = 0;
    if ((esp_timer_get_time() - tiempoEnCasa) >= tMuestreo) fOperacionAuto =
BAJANDO_A;
    break;
case BAJANDO_A:
    rutina_bajada_automatica(dMax, velDeseada);
    break;
case SUBIENDO_A:
    rutina_subida_automatica(velDeseada);
    break;
case FIN_A:
    modo = CASA;
    funcionAuto = false;
    break;
case RESET_A:
    repeticiones = datosGuardados.getInt("repeticiones", 0);
    dMax = datosGuardados.getFloat("dmax", 0);
    velDeseada = datosGuardados.getFloat("veldeseada", 0);
    tMuestreo = datosGuardados.getFloat("tmuestreo", 0);
    contRepeticiones = datosGuardados.getInt("contador", -1);
    if (!probeStatus)rutina_subida_automatica(velocidad);
    else fOperacionAuto = ESPERANDO_A;
    break;
default:
    modo = CASA;

```

```

        break;
    }
    break;
case DETENIDO:
    PWM = 0;
    motorControl(PARAR, PWM);
    break;
case BAJANDO:
    funcionAuto = false;
    rutina_bajada(disMax, velocidad);
    break;
case SUBIENDO:
    funcionAuto = false;
    rutina_subida(velocidad);
    break;
default:
    modo = DETENIDO;
    break;
}
}

void rutina_bajada(float disMax, float velocidad) {
    if (digitalRead(FIN_CARRERA) == HIGH)probeStatus = true;
    if (probeStatus) {
        detachInterrupt(FIN_CARRERA);
        probeStatus = false;
        intDesactivada = true;
    }
    if (encoderPosition > 50 && intDesactivada) {
        attachInterrupt(FIN_CARRERA, int1, RISING);
        intDesactivada = false;
    }
}

```

```

    }
    distancia = encoderPosition * kDistance;
    if (distancia < disMax) {
        PWM = control_bajada(velocidad);
        motorControl(BAJAR, PWM);
    } else modo = DETENIDO;
}

void rutina_bajada_automatica(float disMax, float velocidad) {
    if (digitalRead(FIN_CARRERA) == HIGH) probeStatus = true;
    if (probeStatus) {
        detachInterrupt(FIN_CARRERA);
        probeStatus = false;
        intDesactivada = true;
    }
    if (encoderPosition > 50 && intDesactivada) {
        attachInterrupt(FIN_CARRERA, int1, RISING);
        intDesactivada = false;
    }
    distancia = encoderPosition * kDistance;
    if (distancia < disMax) {
        PWM = control_bajada(velocidad);
        motorControl(BAJAR, PWM);
    } else {
        PWM = 0;
        motorControl(PARAR, PWM);
        fOperacionAuto = SUBIENDO_A;
    }
}

void rutina_subida(float velocidad) {

```

```

    if (probeStatus) {
        modo = CASA;
        PWM = control_subida(0.05);
        motorControl(SUBIR, PWM);
        delay(1900);
    }
    PWM = control_subida(velocidad);
    motorControl(SUBIR, PWM);
    sensar_corriente();
}

void rutina_subida_automatica(float velocidad) {
    if (probeStatus) {
        contRepeticiones = datosGuardados.getInt("contador", 0);
        contRepeticiones++;
        datosGuardados.putInt("contador", contRepeticiones);
        if (contRepeticiones >= repeticiones) {
            fOperacionAuto = FIN_A;
        } else {
            fOperacionAuto = ESPERANDO_A;
            tiempoEnCasa = esp_timer_get_time();
        }
        PWM = control_subida(0.05);
        motorControl(SUBIR, PWM);
        delay(1900);
    }
    PWM = control_subida(velocidad);
    motorControl(SUBIR, PWM);
    sensar_corriente();
}

```

```

void sensor_corriente() {
    float valorADC;
    float corriente;
    valorADC = 0;
    valorADC = analogRead(CORRIENTE);
    corriente = 0;
    for (int i = 4; i > 0; i--) {
        valorCorriente[i] = valorCorriente[i-1];
        corriente = corriente + valorCorriente[i];
    }
    corriente = corriente + valorADC;
    valorCorriente[0] = corriente/5;
    corriente = 2.63 + (0.011*valorCorriente[0]);
    if(corriente > 9){modo = DETENIDO; sobreCorriente = true; }
}

```