

UNIVERSIDAD TÉCNICA FEDERICO SANTA MARÍA
SEDE DE CONCEPCIÓN - REY BALDUINO DE BÉLGICA

SIMULADOR MECATRONICO DE CONSOLA DE BUCEO PARA
ENTRENAMIENTO Y SEGURIDAD EN OPERACIONES INDUSTRIALES

Trabajo de Titulación para optar al Título
Profesional de Ingeniería de Ejecución en
CONTROL E INSTRUMENTACIÓN
INDUSTRIAL

Alumno:

Víctor Andrés Miranda Gallardo.

Profesor Guía:

Sr. Nelson Vásquez Concha



CONSTANCIA DE VALIDACIÓN Y CONFIDENCIALIDAD DE MONOGRAFÍA A REPOSITORIO ACADÉMICO

1.- IDENTIFICACIÓN DEL TRABAJO ACADÉMICO

Tipo de monografía (marcar una opción): ☒ Memoria o trabajo de título ☐ Tesis de Postgrado

Título del trabajo: **Simulador mecatrónico de consola de buceo para entrenamiento y seguridad en operaciones industriales.**

Nombre del candidato(a): **Víctor Andrés Miranda Gallardo.**

Carrera / Grado: **Ingeniería de ejecución en control e instrumentación industrial**

Campus: **Concepción**

Departamento: **Electrónica e Informática**

2.- VALIDACIÓN DEL PROFESOR GUÍA/DIRECTOR DE TESIS

Yo, Nelson Artemio Vásquez Concha, en mi calidad de profesor(a) guía/director(a) del trabajo académico mencionado anteriormente **DEJO CONSTANCIA** que:

- He revisado esta versión del documento y corresponde a la versión final aprobada del trabajo.
- El trabajo cumple con los requisitos académicos y de formato establecidos por la institución.

3.- EVALUACIÓN DE CONFIDENCIALIDAD POR PROPIEDAD INDUSTRIAL (marcar una opción)

☒ El trabajo **NO contiene** información que amerite confidencialidad y puede ser publicado de inmediato en repositorio con acceso abierto.

☐ El trabajo **CONTIENE** información con potenciales implicancias de propiedad industrial o intelectual y requiere un periodo de confidencialidad (**embargo**) por (**marcar una opción**):

☐ 6 meses ☐ 12 meses ☐ 2 años ☐ 3 años ☐ 5 años ☐ 10 años

Fundamentación de la necesidad de confidencialidad (obligatorio si se solicita embargo):

4.- FIRMAS

Profesor(a) guía o director(a) de memoria o tesis:

Fecha: 05-01-2026

Firma: _____

Estudiante o Candidato(a):

Fecha: 05-01-2026

Firma: _____

Este formulario debe ser insertado como página 2 de la memoria o tesis, completado y firmado por estudiante y profesor(a) antes de la entrega en portal PRISMA de Biblioteca USM.

RESUMEN

KEYWORDS: SIMULACIÓN MECATRÓNICA - CONSOLA DE BUCEO INDUSTRIAL - ENTRENAMIENTO DE COMPETENCIAS CRÍTICAS - SEGURIDAD HIPERBÁRICA

Las operaciones de buceo industrial son actividades de alto riesgo donde la seguridad de los buzos depende de la competencia del equipo de superficie (supervisor, hombre dato y operador de consola). Una gestión inadecuada de los parámetros vitales es causa directa de accidentes graves y, a menudo, fatales. La peligrosidad de esta situación se refleja en las estadísticas de la DIRECTEMAR, que, por ejemplo, reportaron 46 accidentes por enfermedad descompresiva con 14 fallecimientos solo en 2008, evidenciando una necesidad de fortalecer la capacitación práctica.

En respuesta a esta necesidad, este trabajo de titulación presenta el diseño y desarrollo de un simulador mecatrónico de una consola de buceo con suministro desde superficie. El objetivo principal es ofrecer una plataforma de entrenamiento segura y controlada que permita al personal perfeccionar sus habilidades en la gestión de operaciones de buceo, mitigando los riesgos inherentes a la formación en escenarios reales.

El sistema se materializa en un prototipo físico que integra componentes electromecánicos tales como motores paso a paso, potenciómetros e interruptores gobernados por microcontroladores ESP32 y ESP8266. Estos se comunican de forma inalámbrica con un software interactivo, emulando la funcionalidad y respuesta de una consola real. La arquitectura del sistema y los escenarios de simulación se basan rigurosamente en los protocolos y procedimientos operativos del US Navy Diving Manual, Revisión 7A, el estándar de referencia en la industria mundial.

El prototipo será validado mediante pruebas experimentales para alcanzar un Nivel de Madurez Tecnológica (TRL) 5. Se espera que esta herramienta contribuya significativamente a la reducción de incidentes críticos en el sector, al mitigar el factor de error humano causa principal de dichos eventos mediante el fortalecimiento de la capacitación, la experiencia y la toma de decisiones del equipo de superficie, mejorando así la seguridad y la eficiencia del buceo profesional en Chile.

ÍNDICE

RESUMEN 2

ÍNDICE.....3

FIGURAS 6

TABLAS 8

GRÁFICOS 9

DIAGRAMAS 10

SIGLA Y SIMBOLOGÍA.....11

SIGLA 11

SIMBOLOGÍA 13

INTRODUCCIÓN.....14

OBJETIVO GENERAL 16

OBJETIVOS ESPECÍFICOS 16

CAPÍTULO 1: MARCO OPERACIONAL Y FUNDAMENTOS DE SEGURIDAD EN EL BUCEO INDUSTRIAL.....17

1. MARCO OPERACIONAL Y NORMATIVO DEL BUCEO INDUSTRIAL.....18

1.1. FUNDAMENTOS FÍSICOS DEL BUCEO 18

1.1.1. Ley de Boyle.....18

1.1.2. Ley General de los Gases 19

1.1.3. Ley de Dalton 19

1.1.4. Ley de Henry 20

1.2. FISIOLÓGÍA Y AFECCIONES DEL BUCEO 21

1.2.1. Narcosis por Nitrógeno.....21

1.2.2. Toxicidad del Oxígeno.....22

1.2.3. Enfermedad por Descompresión (ED) 22

1.3. MARCO OPERACIONAL DEL BUCEO INDUSTRIAL 24

1.3.1. Regulaciones y Estándares 24

1.3.2. Fases de una Operación de Buceo Asistido 24

1.3.3. Límites Operacionales 25

1.3.4. Personal y Responsabilidades.....27

1.4. EQUIPAMIENTO MÍNIMO PARA UNA OPERACIÓN 30

1.4.1. Bancos de Aire.....31

1.4.2. Trasvasador.....31

1.4.3.	Consola De Buceo	32
1.4.4.	Umbilical	33
1.4.5.	Sistema de Comunicaciones y Monitoreo	34
1.4.6.	Equipamiento de Inmersión del Buzo	35
CAPÍTULO 2: ANÁLISIS FUNCIONAL Y DISEÑO DEL PROTOTIPO		37
2.	ANÁLISIS FUNCIONAL Y DISEÑO DEL PROTOTIPO DE CONSOLA	38
2.1.	DISEÑO DEL PANEL DE CONTROL INFERIOR	38
2.1.1.	Admisión y Monitoreo de Alta Presión (HP)	39
2.1.2.	Control y Visualización de la Presión Mínima Manométrica (MMP).....	40
2.1.3.	Integración de Suministros y Manifold de Distribución	41
2.1.4.	Distribución y Seccionamiento de Suministro a Buzos.....	42
2.2.	DISEÑO DEL PANEL DE CONTROL SUPERIOR	43
CAPITULO 3: INTEGRACIÓN DE LA ARQUITECTURA DE HARDWARE		45
3.	INTEGRACIÓN DE LA ARQUITECTURA DE HARDWARE.....	46
3.1.	MÓDULO DE CONTROL DE SUMINISTRO (CONSOLA INFERIOR) ..	47
3.1.1.	Controlador Principal (ESP32-S3)	47
3.1.2.	Coprocesador de Adquisición Analógica (ESP8266)	50
3.2.	MÓDULO DE CONTROL DEL PANEL SUPERIOR	52
3.2.1.	Actuadores para Manómetros	54
3.2.2.	Expansión de Salidas Digitales por I ² C (PCF8574)	55
CAPÍTULO 4: DESARROLLO DE FIRMWARE Y PROTOCOLO DE COMUNICACIÓN		59
4.	DESARROLLO DE FIRMWARE Y PROTOCOLO DE COMUNICACIÓN	60
4.1.	ARQUITECTURA DE COMUNICACIÓN DESACOPLADA CON MQTT..	61
4.1.1.	Principios Fundamentales y Calidad de Servicio (QoS)	62
4.1.2.	Mensajes Retenidos y Última Voluntad y Testamento (LWT)	62
4.1.3.	El Broker Central y Herramientas de Diagnóstico	63
4.1.4.	Estructura Jerárquica de Tópicos.....	63
4.2.	MODULARIDAD MEDIANTE (POO)	64
4.2.1.	Clase ComunicadorMQTT: El Núcleo de la Red	64
4.2.2.	Clase SistemaMotores: Control de Actuadores y Simulación Física	67
4.2.3.	Clase SistemaValvulas: Monitorización de Sensores Digitales.....	69

4.2.4.	Clases LectorPotenciometros / SistemaPotenciometros: Monitorización Analógica	70
4.3.	IMPLEMENTACIÓN DEL FIRMWARE POR MÓDULO	73
4.3.1.	Panel Inferior: Arquitectura Distribuida	74
4.3.2.	Panel Superior: Arquitectura Centralizada	79
CAPÍTULO 5: ARQUITECTURA DE SOFTWARE Y HMI.....		82
5.	ARQUITECTURA DE SOFTWARE DEL SIMULADOR	83
5.1.	ARQUITECTURA GENERAL Y FLUJO DE COMUNICACIÓN	83
5.2.	SECUENCIA DE INICIALIZACIÓN DEL SISTEMA.....	85
5.3.	ARQUITECTURA DETALLADA POR CAPAS	86
5.3.1.	Capa de Comunicación (Paquete gestor_mqtt)	86
5.3.2.	Capa de Lógica y Estado (Paquete modelo_estado).....	88
5.4.	DISEÑO DE LA INTERFAZ HOMBRE-MÁQUINA (HMI)	90
5.4.1.	Orquestación General (VentanaPrincipal)	90
5.4.2.	Componentes Visuales Reutilizables	92
5.4.3.	Control Panel Inferior	94
5.4.4.	Control Panel Superior	95
CONCLUSIONES Y RECOMENDACIONES		96
CONCLUSIONES.....		96
RECOMENDACIONES		97
BIBLIOGRAFÍA		100

FIGURAS

Figura 1-1 Ley de Boyle..... 19

Figura 1-2 Ley General de los Gases..... 19

Figura 1-3 Ley de Dalton.....20

Figura 1-4 Ley de Henry 21

Figura 1-5 Banco De Aire.....31

Figura 1-6 Trasvasadores.....32

Figura 1-7 Consola De Buceo 33

Figura 1-8 Umbilical 34

Figura 1-9 Equipo De Comunicaciones Y Monitoreo 35

Figura 1-10 Equipamiento de Inmersión del Buzo.....36

Figura 2-1 Vista Frontal del Panel Inferior.....39

Figura 2-2 Diagrama Funcional del Panel Inferior de Suministro de Aire (Vista Frontal) 39

Figura 2-3 Diseño 3D del conjunto de Válvula de Corte y Manómetro de Alta Presión para el prototipo de simulador.....40

Figura 2-4 Formula MMP 40

Figura 2-5 Tabla De Presión Sobre La Profundidad.....41

Figura 2-6 Diseño 3D del conjunto de control de la MMP, compuesto por la Válvula Reguladora y su Manómetro 41

Figura 2-7 Diseño 3D del montaje de una de las válvulas selectoras en el circuito de suministro del panel inferior.....42

Figura 2-8 Modelo CAD del Panel Inferior de la Consola de Buceo 43

Figura 2-9 Comparativa entre el diseño CAD del panel superior (izquierda) y el prototipo físico ensamblado (derecha) 44

Figura 3-1 Regulador De Voltaje Step Down Dc-dc Lm2596 Con voltímetro 46

Figura 3-2 Pintout ESP32 S3.....48

Figura 3-3 Motor 28BYJ-48.....49

Figura 3-4 Micro interruptor KW11 50

Figura 3-5 Microcontrolador ESP8266 50

Figura 3-6 ADS1115 de 16 bits 51

Figura 3-7 Potenciómetros multivuelta 3296W y Potenciómetro de 1K Ohms 51

Figura 3-8 Pintout ESP32-DevKitC 54

Figura 3-9 Módulo Interfaz PCF8574 I2C Expansor de Pines.....56

Figura 3-10 Módulo CD74HC4067.....58

Figura 4-1 Lógica de Resolución de Host con Fallback a mDNS.....65

Figura 4-2 Implementación del Método configurarLWT 66

Figura 4-3 Implementación de la Máquina de Estados en el Método 67

Figura 4-4 Interfaz Privada de la Clase SistemaMotores. 68

Figura 4-5 Implementación del Algoritmo Debounce y Sincronización Periódica.....70

Figura 4-6 Ecuación del Filtro de Suavizado Exponencial 71

Figura 4-7 Ecuación de Normalización por Escalado Mín-Máx 71

Figura 4-8 Implementación del Algoritmo de Filtrado y Detección de Cambios 72

Figura 4-9 Lógica de Lectura Multiplexada y Detección de Cambios.....73

Figura 5-1 Importación de Módulos Centrales de la Arquitectura 84

Figura 5-2 Flujo de Inicialización del Software 85

Figura 5-3 Implementación del Hilo de Conexión MQTT.....87

Figura 5-4 Métodos de Suscripción y Control de Motores 89

Figura 5-5 Método principal de renderizado del dibujante visual.....91

Figura 5-6 Ejemplos del Componente Manómetro Optimizado 92

Figura 5-7 Válvula Rotatoria Simulada.....93

Figura 5-8 Componente Indicador Digital.....93

Figura 5-9 Interfaz Humano-Máquina (HMI) del Simulador Panel Inferior.....94

Figura 5-10 Panel Superior de Monitoreo de Buzos y Telemetría 95

TABLAS

Tabla 1-1 Limites Normales Y Máximos Para Buceo Con Aire.....26

Tabla 1-2 Personal Mínimo Para Una Operación De Buceo Asistido 27

Tabla 3-1 GUÍA DE CONEXIONES (ESP32-S3) 48

Tabla 3-2 GUÍA DE CONEXIONES (ESP8266).....52

Tabla 3-3 GUÍA DE CONEXIONES ESP32-DevKitC.....53

Tabla 3-4 Parámetros del Bus I²C (PCF8574).....56

Tabla 3-5 Mapeo de Motores en Expansores PCF8574 56

Tabla 3-6 Mapeo de Módulo CD74HC4067 57

Tabla 4-1 NIVELES DE CALIDAD DE SERVICIO (QOS) EN MQTT 62

Tabla 4-2 DESGLOSE DE LA ESTRUCTURA DE TÓPICOS MQTT DEL PROYECTO 64

Tabla 4-3 TAREAS CONCURRENTES (FREERTOS) PARA LA GESTIÓN DE MOTORES
.....74

Tabla 4-4 PINOUT PARA MOTORES 75

Tabla 4-5 ARQUITECTURA DE TÓPICOS MQTT PARA EL MÓDULO DE MOTORES ..76

Tabla 4-6 TÓPICOS MQTT PARA EL MÓDULO DE VÁLVULAS 76

Tabla 4-7 TAREAS CONCURRENTES (FREERTOS) PARA LA GESTIÓN DE VÁLVULAS
.....77

Tabla 4-8 TÓPICOS MQTT PARA EL MÓDULO DE POTENCIÓMETROS 78

Tabla 4-9 SECUENCIA LÓGICA DEL BUCLE PRINCIPAL (ESP8266) 78

Tabla 4-10 TABLA DE CONFIGURACIÓN DEL MÓDULO.....79

Tabla 4-11 Módulo de Monitorización Analógica (Panel Superior).....80

Tabla 4-12 MÓDULO DE MONITORIZACIÓN ANALÓGICA.....81

GRÁFICOS

Gráfico 1-1 Accidentes de buceo por ED 23

DIAGRAMAS

Diagrama 4-1 Lógica Del Sistema De Simulación.....61

Diagrama 5-1 Arquitectura General Del Sistema 84

SIGLA Y SIMBOLOGÍA

SIGLA

AC	: Corriente Alterna (del inglés, Alternating Current)
ADCI	: Association of Diving Contractors International (Asociación Internacional de Contratistas de Buceo)
ADC	: Conversor Analógico-Digital (del inglés, Analog-to-Digital Converter)
API	: Interfaz de Programación de Aplicaciones (del inglés, Application Programming Interface)
ATA	: Atmósferas Absolutas (del inglés, Atmospheres Absolute)
CAD	: Diseño Asistido por Computadora (del inglés, Computer-Aided Design)
DC	: Corriente Continua (del inglés, Direct Current)
DIRECTEMAR	: Dirección General del Territorio Marítimo y de Marina Mercante
DSP	: Procesamiento Digital de Señales (del inglés, Digital Signal Processing)
DSWS	: Diving Supervisor Warfare Specialist (Especialista de Supervisión de Buceo de Combate)
E/S	: Entrada/Salida (del inglés, I/O - Input/Output)
ED	: Enfermedad por Descompresión
EGS	: Suministro de Gas de Emergencia (del inglés, Emergency Gas Supply)
EMI	: Interferencia Electromagnética (del inglés, Electromagnetic Interference)
FreeRTOS	: Sistema Operativo de Tiempo Real para Microcontroladores (del inglés, Real-Time Operating System for Microcontrollers)
fsw	: Pies de Agua de Mar (del inglés, Feet of Sea Water)
GPIO	: Entrada/Salida de Propósito General (del inglés, General-Purpose Input/Output)
HAL	: Capa de Abstracción de Hardware (del inglés, Hardware Abstraction Layer)
HMI	: Interfaz Humano-Máquina (del inglés, Human-Machine Interface)
HP	: Alta Presión (del inglés, High Pressure)

I ² C	: Circuito Inter-Integrado (del inglés, Inter-Integrated Circuit)
IoT	: Internet de las Cosas (del inglés, Internet of Things)
JSON	: Notación de Objetos de JavaScript (del inglés, JavaScript Object Notation)
LP	: Baja Presión (del inglés, Low Pressure)
LWT	: Última Voluntad y Testamento (del inglés, Last Will and Testament, en contexto MQTT)
mDNS	: DNS Multidifusión (del inglés, Multicast DNS)
MMP	: Presión Mínima Manométrica
MRO	: Manejo de Riesgos Operacionales
MQTT	: Message Queuing Telemetry Transport (Transporte de Telemetría por Cola de Mensajes)
MVC	: Modelo-Vista-Controlador (del inglés, Model-View-Controller)
PC	: Computador Personal (del inglés, Personal Computer)
PETG	: Tereftalato de Polietileno Glicolizado
PLA	: Ácido Poliláctico (del inglés, Polylactic Acid)
POO	: Programación Orientada a Objetos
QoS	: Calidad de Servicio (del inglés, Quality of Service)
RCP	: Reanimación Cardiopulmonar
RTOS	: Sistema Operativo de Tiempo Real (del inglés, Real-Time Operating System)
SCUBA	: Equipo de Respiración Subacuática Autónomo (del inglés, Self-Contained Underwater Breathing Apparatus)
SNC	: Sistema Nervioso Central
SoC	: Separación de Conceptos (del inglés, Separation of Concerns)
SRP	: Principio de Responsabilidad Única (del inglés, Single Responsibility Principle)
TCP/IP	: Protocolo de Control de Transmisión/Protocolo de Internet
TRL	: Nivel de Madurez Tecnológica (del inglés, Technology Readiness Level)
UI	: Interfaz de Usuario (del inglés, User Interface)
WLAN	: Red de Área Local Inalámbrica (del inglés, Wireless Local Area Network)

SIMBOLOGÍA

P	: Presión
V	: Volumen
T	: Temperatura
P _g	: Presión parcial de un gas sobre un líquido
P _{N2}	: Presión parcial del Nitrógeno
P _{O2}	: Presión parcial del Oxígeno
C	: Concentración de un gas disuelto
k	: Constante de solubilidad de un gas (Ley de Henry)
α	: Factor de suavizado (para filtro exponencial)
psi	: Libras por pulgada cuadrada (del inglés, Pounds per Square Inch)
N ₂	: Molécula de Nitrógeno
O ₂	: Molécula de Oxígeno

INTRODUCCIÓN

En las operaciones de buceo industrial, la intervención humana es indispensable para el desarrollo de sectores estratégicos para Chile, como la acuicultura y el mantenimiento de infraestructuras portuarias y mineras. Dada la extensa costa del país, con más de 4,000 kilómetros, el buceo industrial es una actividad de alta demanda y criticidad. Sin embargo, este entorno operativo es inherentemente hostil y expone al personal a severos riesgos fisiológicos, principalmente la enfermedad por descompresión (ED). La mitigación de estos riesgos se articula a través de protocolos rigurosos, donde el US Navy Diving Manual, Revisión 7A, se instaure como el estándar de facto a nivel internacional, definiendo los procedimientos para la ejecución segura de operaciones de buceo.

A pesar de los avances tecnológicos en equipos y sistemas de soporte vital, el factor humano permanece como la variable más crítica en la cadena de seguridad. La integridad del buzo depende directamente de la destreza y coordinación del equipo de superficie, compuesto por el Supervisor de Buceo, el Operador de Consola y el Hombre Datos. Este equipo es responsable de la gestión en tiempo real de parámetros vitales: el suministro y la presión del gas respirable, el monitoreo de la profundidad y el cumplimiento estricto de las tablas de descompresión.

No obstante, se ha detectado una discrepancia fundamental entre la asimilación teórica de los manuales y la capacidad del personal para ejecutar estos procedimientos bajo estrés o en contingencias. Esta deficiencia en la formación práctica incrementa el riesgo de error humano, con consecuencias fatales. Las estadísticas de la Dirección General del Territorio Marítimo y de Marina Mercante (DIRECTEMAR) validan esta problemática con datos alarmantes: en 2005 se registraron 116 accidentes por ED con 13 muertes, mientras que en 2008 la cifra fue de 46 accidentes con 14 fallecimientos como se muestra en el Gráfico 1-1. Estos indicadores subrayan la urgencia de adoptar soluciones tecnológicas que permitan un entrenamiento inmersivo, repetible y, fundamentalmente, seguro.

Como respuesta a esta necesidad, este trabajo de titulación aborda el diseño y la implementación de un simulador mecatrónico de una consola de buceo. La solución se materializa en una plataforma de entrenamiento integral que combina hardware y software. A nivel físico, un prototipo replica la interfaz y funcionalidad de una consola real, utilizando actuadores electromecánicos. Los motores paso a paso emulan la dinámica de los manómetros analógicos, potenciómetros simulan el ajuste de válvulas reguladoras y los interruptores operan las válvulas de control de flujo. El núcleo del sistema reside en una arquitectura de control distribuida, basada en microcontroladores ESP32 y ESP8266, que gestiona la lógica local y la comunicación de datos en tiempo real mediante el protocolo MQTT. Esta arquitectura se

interconecta inalámbricamente con un software de simulación en un PC, desde el cual un instructor puede diseñar y evaluar un amplio rango de escenarios, desde procedimientos estándar hasta emergencias complejas.

El propósito fundamental de este proyecto es crear un ecosistema de entrenamiento donde el personal de superficie pueda desarrollar y validar sus competencias críticas en un entorno controlado, sin exponer a ningún buzo a riesgos. La validación del sistema hasta un Nivel de Madurez Tecnológica (TRL) 5 permitirá entregar una herramienta tangible y eficaz, cuyo fin es contribuir directamente a la reducción de accidentes en la industria del buceo profesional en Chile, a través del refuerzo de la cultura de seguridad y la optimización de la toma de decisiones.

OBJETIVO GENERAL

Contribuir a la mitigación de la enfermedad descompresiva, un riesgo crítico en el buceo industrial mundial, mediante el diseño, implementación y validación de un sistema mecatrónico.

El propósito es crear una herramienta para el entrenamiento avanzado en seguridad operacional, que mejore las competencias del personal en la gestión de protocolos de descompresión y emergencias, alcanzando un Nivel de Madurez Tecnológica (TRL) 5.

OBJETIVOS ESPECÍFICOS

- Diseñar y construir el prototipo físico de la consola de buceo, asegurando que su ergonomía y disposición de componentes repliquen de manera funcional la interfaz de un equipo real.
- Integrar la arquitectura electrónica de control en el prototipo, implementando los microcontroladores, actuadores y sensores necesarios para emular los parámetros operacionales críticos (presión, profundidad y tiempo).
- Desarrollar el ecosistema de software del simulador, lo que abarca la programación del firmware de control en tiempo real (ESP32/ESP8266, protocolo MQTT) y la Interfaz Humano-Máquina (HMI) del instructor para la gestión de escenarios de entrenamiento (Python/PySide6).
- Validar la funcionalidad y efectividad del sistema integrado mediante pruebas experimentales con buzos profesionales, verificando el cumplimiento de los objetivos de entrenamiento en la correcta aplicación de procedimientos y en la toma de decisiones bajo simulación.

**CAPÍTULO 1: MARCO OPERACIONAL Y FUNDAMENTOS DE
SEGURIDAD EN EL BUCEO INDUSTRIAL**

1. MARCO OPERACIONAL Y NORMATIVO DEL BUCEO INDUSTRIAL

El buceo industrial constituye un pilar fundamental para el desarrollo de sectores estratégicos en Chile, aprovechando sus más de 4,000 kilómetros de costa. Sin embargo, esta actividad se desarrolla en un entorno adverso, donde la exposición del personal a presiones hiperbáricas impone severos riesgos fisiológicos. Entre ellos, la enfermedad por descompresión (ED) surge como la amenaza más crítica, siendo una de las principales causas de accidentes y fatalidades en la industria a nivel mundial.

En este contexto, la seguridad operacional constituye un pilar no negociable, Integrado a través de protocolos rigurosos y un conocimiento profundo de la interacción humana con el medio subacuático.

El propósito de este capítulo es establecer el marco teórico y práctico que da fundamento a este trabajo. Para ello, se analizarán los principios físicos y fisiológicos del buceo, se describirán los procedimientos y responsabilidades del personal, y se definirán los estándares de equipamiento, tomando como referencia normativa el US Navy Diving Manual, Revisión 7A.

La síntesis de estos elementos conformará el modelo fundamental sobre el cual se diseñará y validará el simulador, asegurando que esta herramienta de entrenamiento responda fidedignamente a las exigencias operacionales y a la necesidad imperativa de prevenir accidentes.

1.1. FUNDAMENTOS FÍSICOS DEL BUCEO

Para gestionar los riesgos inherentes al entorno hiperbárico, es indispensable comprender los principios físicos que gobiernan el comportamiento de los gases bajo presión. Esta sección detalla las leyes de Boyle, Dalton y Henry, cuya aplicación es fundamental para la prevención de barotraumas y la gestión de la enfermedad por descompresión.

1.1.1. Ley de Boyle

La Ley de Boyle es uno de los principios físicos importantes para la prevención de barotraumas en el buceo. Postula que, para una masa determinada de gas a temperatura constante, el volumen (V) varía de forma inversamente proporcional a la presión absoluta (P) que se ejerce sobre él. En términos prácticos, esto significa que a mayor presión absoluta el

volumen de un gas es menor, y a menor presión, su volumen aumenta. Esta relación fundamental, cuya implicancia directa es la expansión del aire en los espacios aéreos del cuerpo durante un ascenso, se expresa matemáticamente en la Figura 1-1.

$$P_1 \cdot V_1 = P_2 \cdot V_2$$

Figura 1-1 Ley de Boyle

Fuente: Adaptado de US Navy Diving Manual, Rev. 7A, Cap. 2.

1.1.2. Ley General de los Gases

Ley General de los Gases ofrece un modelo más completo al integrar la variable de la temperatura (T), combinando los principios de las leyes de Boyle y de Charles.

Esta ley es esencial para los cálculos de suministro y consumo de aire, ya que describe cómo la presión, el volumen y la temperatura de un gas se interrelacionan, como se muestra en la fórmula de la Figura 1-2.

Su aplicación práctica más crítica es la gestión de los bancos de aire: un cilindro cargado en un ambiente frío mostrará un aumento significativo de presión si se expone al calor en la cubierta de una embarcación. Ignorar esta variación térmica puede llevar a una estimación incorrecta de la reserva de gas disponible, afectando directamente la seguridad de la operación.

$$\frac{P_1 \cdot V_1}{T_1} = \frac{P_2 \cdot V_2}{T_2}$$

Figura 1-2 Ley General de los Gases

Fuente: Adaptado de US Navy Diving Manual, Rev. 7A, Cap. 2.

1.1.3. Ley de Dalton

La Ley de Dalton establece que la presión total ejercida por una mezcla de gases es igual a la suma de las presiones parciales que cada gas individual ejercería si ocupara el volumen total por sí solo, como se muestra en la fórmula de la Figura 1-3.

Su importancia en el buceo es crítica, ya que, al aumentar la presión ambiental durante un descenso, la presión parcial de cada componente del aire respirable se incrementa

proporcionalmente. Este fenómeno es la causa directa de varias afecciones fisiológicas que deben ser gestionadas desde la superficie.

A continuación, se analizan los dos riesgos principales derivados de esta ley para un buzo respirando aire (21% O₂ / 79% N₂) y su correspondiente gestión desde la consola de buceo. En primer lugar, aunque el oxígeno es vital, su presión parcial (PO₂) se vuelve tóxica para el Sistema Nervioso Central (SNC) cuando se aproxima a 1.6 ATA, un umbral de riesgo que se alcanza a una profundidad de aproximadamente 66 metros, siempre y cuando la mezcla del gas respirable sea aire.

El control primario de este peligro recae en el operador de consola, quien debe utilizar el neumofatometro para monitorear continuamente la profundidad y alertar de inmediato al Supervisor si la inmersión excede los límites operativos seguros.

Por otro lado, el nitrógeno ejerce un efecto narcótico significativo a partir de una presión parcial (PN₂) de 3.2 ATA, lo que para un buzo respirando aire equivale a una profundidad de 30 metros. En este caso, la gestión del riesgo es una tarea de equipo donde el operador de consola debe correlacionar la profundidad indicada en sus instrumentos con el comportamiento del buzo transmitido por las comunicaciones, detectando anomalías como confusión o euforia para reportarlas textualmente.

Por lo tanto, la gestión de estos riesgos dependientes de la profundidad es una responsabilidad primordial del equipo de superficie y un objetivo de entrenamiento clave para el simulador propuesto.

$$P_{total} = P_1 + P_2 + \dots + P_n$$

Figura 1-3 Ley de Dalton

Fuente: Adaptado de US Navy Diving Manual, Rev. 7A, Cap. 2.

1.1.4. Ley de Henry

La Ley de Henry es el principio que gobierna la disolución de gases en líquidos y constituye la base fisiológica para comprender la enfermedad por descompresión (ED). Postula que, a una temperatura constante, la cantidad de un gas que se disuelve en un líquido es directamente proporcional a la presión parcial que dicho gas ejerce sobre la superficie del líquido como se muestra en la fórmula de la Figura 1-4. Donde C es la concentración del gas disuelto, P_g es la presión parcial del gas sobre el líquido, y k es la constante de solubilidad.

Esta ley es fundamental, ya que explica el mecanismo de la enfermedad por descompresión. A medida que un buzo desciende, la presión ambiente aumenta y, según la Ley

de Dalton, también lo hace la presión parcial del nitrógeno (P_{N_2}) en el aire que respira. Este aumento del P_{N_2} en los alvéolos crea un gradiente de presión que impulsa al nitrógeno a disolverse en la sangre y, posteriormente, a acumularse en los distintos tejidos del cuerpo (proceso de "on-gassing").

El problema crítico surge durante el ascenso. Si el buzo asciende demasiado rápido, la presión ambiente disminuye abruptamente y el nitrógeno disuelto sale de la solución de forma violenta, formando burbujas de gas en los tejidos y el torrente sanguíneo.

Es la formación y acción de estas burbujas lo que causa el amplio espectro de síntomas conocidos como enfermedad por descompresión. Las tablas de descompresión del US Navy Diving Manual son, en esencia, un algoritmo de ascenso controlado, diseñado para permitir que el nitrógeno sea eliminado de forma segura a través de la respiración.

$$C = k \cdot P_g$$

Figura 1-4 Ley de Henry

Fuente: Adaptado de US Navy Diving Manual, Rev. 7A, Cap. 2.

1.2. FISIOLOGÍA Y AFECCIONES DEL BUCEO

La inmersión en un entorno hiperbárico introduce riesgos fisiológicos directos, derivados del comportamiento de los gases bajo presión, tal como se describió en las leyes físicas. La gestión de estas afecciones por parte del supervisor y el operador de consola es una competencia crítica basada en conocimientos fundamentales que a menudo no se aplican con la precisión necesaria, lo cual es clave para garantizar la seguridad de toda la operación subacuática. Las principales condiciones para gestionar se describen a continuación, basadas en los principios del US Navy Diving Manual, Revisión 7A, Capítulo 3.

1.2.1. Narcosis por Nitrógeno

A medida que aumenta la profundidad, la elevada presión parcial del nitrógeno (P_{N_2}) ejerce un efecto narcótico sobre el sistema nervioso central, análogo a la intoxicación alcohólica. Este estado, conocido como "borrachera de las profundidades", es reversible, pero compromete la seguridad de la operación al alterar el juicio, la coordinación y la capacidad de respuesta del buzo. Como se calculó previamente, sus efectos comienzan a ser significativos a partir de los 30 metros de profundidad.

1.2.2. Toxicidad del Oxígeno

Aunque es esencial para la vida, el oxígeno se vuelve tóxico cuando su presión parcial (P_{O_2}) supera los umbrales de seguridad establecidos. La sobreexposición puede desencadenar efectos graves en el Sistema Nervioso Central (SNC), siendo las convulsiones el evento más peligroso bajo el agua.

Como se detalló en la sección anterior, el límite máximo de exposición se alcanza a una P_{O_2} de 1.6 ATA, lo que corresponde a una profundidad de 66 metros respirando aire. La monitorización constante de la profundidad es la principal barrera de seguridad contra este riesgo

1.2.3. Enfermedad por Descompresión (ED)

La Enfermedad por Descompresión (ED) es uno de los riesgos fisiológicos más significativos en el buceo. Ocurre cuando el nitrógeno, disuelto en los tejidos corporales bajo presión, forma burbujas durante un ascenso inadecuado. Este ascenso inadecuado raramente es un hecho fortuito; suele ser la consecuencia directa de un error humano en la gestión de la operación, originado por una decisión incorrecta del Supervisor de Buceo, un mal cálculo del Hombre Datos o una gestión deficiente del Operador de Consola.

Estas burbujas no son inertes; actúan como agentes mecánicos lesivos que provocan un daño tisular directo, cuya severidad depende críticamente de su ubicación y del tiempo de respuesta para el tratamiento. A nivel articular, su presencia distiende los tejidos y terminaciones nerviosas, causando el dolor agudo conocido como "bends".

Sin embargo, el daño más catastrófico y a menudo irreversible ocurre a nivel neurológico. Si bien el espectro de lesiones es amplio y puede afectar a múltiples sistemas, los siguientes ejemplos ilustran el mecanismo de daño.

- A. Daño en el Sistema Nervioso Central (SNC): Si se alojan en la médula espinal, pueden comprimir, distorsionar o incluso desgarrar los delicados tractos nerviosos, interrumpiendo la señalización y provocando desde parestesias hasta una parálisis permanente.
- B. Daño en los Pares Craneales: Una burbuja que afecte a uno de los doce pares craneales puede tener consecuencias devastadoras, como ceguera (par craneal II

- óptico), vértigo severo (par craneal VIII - vestibulococlear) o parálisis facial (par craneal VII - facial).

C. Daño al Sistema Nervioso Periférico (SNP): Pueden generar una embolia gaseosa arterial, bloqueando el flujo sanguíneo (isquemia) y causando la muerte del tejido nervioso que depende de ese suministro.

El tratamiento definitivo para la ED es la recompresión en una cámara hiperbárica, que reduce el tamaño de las burbujas y facilita su reabsorción segura. Sin embargo, la prevención es el pilar de la seguridad, e incluye realizar ascensos a velocidades controladas y el uso estricto de las tablas de descompresión.

A pesar de esto, la tasa de accidentabilidad por ED en Chile ha sido un motivo de preocupación. Según estadísticas de la DIRECTEMAR, en 2008 se registraron 46 accidentes con 14 fallecimientos, como se visualiza en el Gráfico 1-1. Estas cifras, que se refieren exclusivamente a incidentes por descompresión, ponen de manifiesto la necesidad de mejorar la formación práctica del personal.

Tal como se ha establecido, la falta de entrenamiento en la gestión de la descompresión y en el manejo de equipos críticos como la consola de buceo, incrementa directamente el riesgo de que el equipo de superficie cometa un error humano. En este contexto, la implementación de simuladores mecatrónicos representa una estrategia clave para reducir esta tasa de accidentabilidad, al proporcionar un entorno seguro y controlado para mejorar la competencia del personal en la aplicación de protocolos de seguridad.

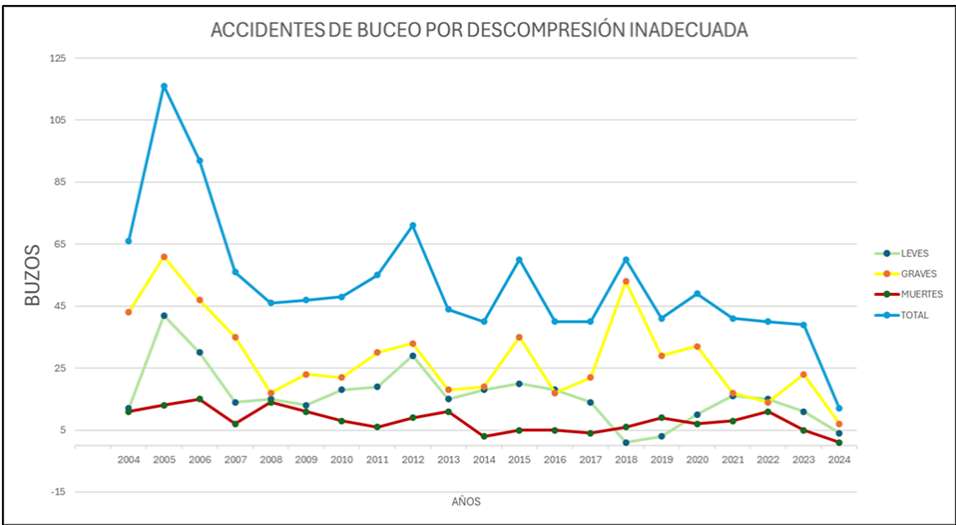


Gráfico 1-1 Accidentes de buceo por ED

Fuente: <https://www.directemar.cl/directemar/intereses-maritimos/buceo-profesional/estadisticas-de-accidentes-de-buceo-profesional/estadisticas-de-accidentes-de-buceo-profesional-2021>

1.3. MARCO OPERACIONAL DEL BUCEO INDUSTRIAL

El buceo profesional se rige por un estricto marco normativo diseñado para maximizar la seguridad en un entorno inherentemente riesgoso. Esta sección detalla dicho marco, describiendo las regulaciones, fases operativas, límites, roles del personal y el equipamiento mínimo indispensable para una operación de buceo con suministro desde superficie, tomando como referencia principal el US Navy Diving Manual, Rev. 7A.

1.3.1. Regulaciones y Estándares

A nivel global, el US Navy Diving Manual se considera el estándar de facto en la industria, especialmente sus tablas de descompresión. Organizaciones como la Association of Diving Contractors International (ADCI) basan sus certificaciones profesionales en las directrices de este manual.

En Chile, la autoridad competente es la Dirección General del Territorio Marítimo y de Marina Mercante (DIRECTEMAR). Aunque la normativa nacional se ha basado históricamente en las tablas de la Armada Española, la Autoridad Marítima chilena permite el uso de las tablas del US Navy Diving Manual previa autorización, facilitando la alineación con los más altos estándares internacionales.

Se observa una tendencia creciente en Chile hacia la adopción de estas normas, particularmente en industrias como la minería que exigen certificaciones ADCI.

Dentro de este marco, el buceo con suministro desde superficie (o buceo asistido) es la técnica por excelencia para operaciones industriales, garantizando un flujo de aire continuo, comunicación eficaz y mayor seguridad. Esta modalidad es ideal para inmersiones prolongadas a profundidades de hasta 190 pies (58 metros) y es el enfoque central del simulador propuesto.

1.3.2. Fases de una Operación de Buceo Asistido

Toda operación de buceo asistido se articula en tres fases secuenciales y rigurosas, diseñadas para mitigar los riesgos inherentes al entorno hiperbárico, tal como se describe en el Capítulo 6 ("Operational Planning and Risk Management") del manual de referencia.

La fase inicial comprende los procedimientos previos a la inmersión, dirigida por el Supervisor de Buceo, quien conduce un briefing de seguridad donde se realiza el análisis de riesgos operacionales (MRO) y se asignan responsabilidades.

Inmediatamente después, el Supervisor lidera una verificación exhaustiva del equipo de soporte vital, un proceso que incluye la inspección de la estanqueidad del sistema de respiración del buzo (cascos como el Kirby Morgan o máscaras oro-nasales), la revisión de la integridad estructural y conexiones del umbilical, la confirmación de la operatividad de la instrumentación de la consola, la prueba de los arneses y la validación de los sistemas de comunicación.

Una vez completados estos controles, se inician las operaciones de inmersión, durante las cuales el buzo comienza el descenso bajo el mando directo del Supervisor, quien mantiene una conciencia situacional completa.

En esta fase, el equipo de superficie opera en perfecta sincronía: el Operador de la Consola gestiona los parámetros críticos de profundidad y suministro de gas; el Tender se encarga del manejo físico del umbilical, asegurando la movilidad del buzo; y el Hombre Datos registra meticulosamente el tiempo de fondo y la profundidad, proveyendo al supervisor los cálculos de descompresión necesarios, todo mientras la comunicación bidireccional se mantiene ininterrumpidamente.

Finalmente, al concluir la operación de buceo, se ejecutan los procedimientos posteriores a la inmersión, que consisten en una evaluación médica del buzo para la detección precoz de cualquier signo de enfermedad, la finalización de la bitácora por parte del Supervisor, y los protocolos de mantenimiento del equipo por el resto del personal técnico, incluyendo la desalinización, inspección detallada y preparación para su siguiente despliegue.

1.3.3. Límites Operacionales

El buceo con aire ya sea con suministro desde superficie o mediante sistemas autónomos (SCUBA), está regulado por estrictos límites de profundidad y tiempo.

Estos parámetros de seguridad, establecidos para proteger a los buzos de riesgos graves como la enfermedad por descompresión (ED) y la toxicidad por oxígeno en el sistema nervioso central (SNC), se basan en el US Navy Diving Manual.

Este manual es la referencia técnica global tanto para operaciones militares como civiles, siendo adoptado por organizaciones como la Association of Diving Contractors International (ADCI) debido a su rigurosidad. Los límites normales y máximos para operaciones de buceo con aire se detallan a continuación, como se muestra en la Tabla 1-1.

Tabla 1-1
 Límites Normales Y Máximos Para Buceo Con Aire

LÍMITES NORMALES Y MAXIMOS PARA BUCEO CON AIRE		
Profundidad PAM/ (metros)	Límite por Equipo	Notas
60 (18)	Equipo de buceo MK 21 MOD 1(Mascara Full Face), KM-37(Casco), límite de trabajo máximo sin Suministro de Gas de Emergencia (SGE)	A
60 (18)	Equipo con suministro desde superficie MK 20 MOD 0 (Mascara Full Face)	A
60 (18)	Máxima profundidad para buzo scuba de respaldo usando cilindro sencillo con menos de 100 p3 de capacidad	
100 (30)	SCUBA de circuito abierto con cilindro de menos de 100 p3 de capacidad	B
130 (40)	SCUBA de circuito abierto, límite de trabajo normal	B
190 (58)	SCUBA de circuito abierto, límite de trabajo máximo con permiso del comandante u Oficial a Cargo,(solo buceo militar)	B, D
190 (58)	Equipo de buceo MK 21 MOD 1(Mascara Full Face), KM-37 (Casco) y EXO BR MS (Mascara Full Face) (aire) con SGE, límite de trabajo normal	C, D, E
285 (87)	Equipo de buceo MK 21 MOD 1(Mascara Full Face), KM-37 (Casco) y EXO BR MS (Mascara Full Face) (aire) con SGE, límite máximo de trabajo, exposición excepcional con autorización del Jefe de Operaciones Navales (N873),(solo buceo militar)	C, D, E
Notas de Operación General (Aplica a todos): 1. Estos límites están basados sobre una consideración práctica de tiempo de trabajo contra tiempo de descompresión y límites de tolerancia al oxígeno. Estos límites no deben ser excedidos excepto por autorización específica del Jefe de Operaciones Navales. 2. No exceder los límites para exposiciones excepcionales de la Tabla de Descompresión con Aire. 3. En una emergencia, cualquier cámara de recompresión operable puede ser usada para tratamiento si un Supervisor de Cámaras calificado DSWs juzga segura para usar. Notas específicas: A. Cuando se bucea en espacios confinados cada buzo debe usar un SGE.		

- B. Bajo circunstancias normales, no exceder los límites de la Tabla de No-Descompresión. Los buceos que requieren descompresión pueden ser hechos si se considera necesario con la aprobación del comandante u Oficial a Cargo del comando de buceo. El tiempo total de un buceo SCUBA (incluyendo descompresión) no excederá la duración del aparato en uso, sin importar cualquier reserva.
- C. Se requiere un Oficial Medico de Buceo en el sitio para todos los buceos más profundos que 190 pam, y para los buceos con exposiciones excepcionales.
- D. Todos los buceos de descompresión planeados a mayor profundidad que 130 pam, requieren una cámara de recompresión certificada en el sitio. Una cámara en el sitio es definida como una cámara lista y certificada accesible dentro de 30 minutos del sitio de buceo por la transportación disponible.
- E. Los buceos con Exposiciones Excepcionales tienen una significativamente muy alta probabilidad de EDD y toxicidad del SNC por oxígeno.
- Fuente: Adaptado de US Navy Diving Manual, Rev. 7A, Cap. 8.

1.3.4. Personal y Responsabilidades

Las operaciones de buceo con suministro desde superficie requieren un equipo especializado donde cada miembro desempeña funciones críticas para la seguridad y la eficacia de la inmersión.

A continuación, se detallan los requisitos mínimos de personal y las responsabilidades clave de cada rol, según lo establecido en el US Navy Diving Manual, Revisión 7A, Capítulo 8. El personal mínimo necesario se resume en la Tabla 1-2.

Tabla 1-2
Personal Mínimo Para Una Operación De Buceo Asistido

PERSONAL MÍNIMO PARA BUCEO CON SUMINISTRO DE AIRE DESDE SUPERFICIE	
SUPERVISOR DE BUCEO	1 (A)
HOMBRE DATOS	1 (A, B)
OPERADOR DE CONSOLA	1 (B)
BUZO:	1
BUZO DE RESPETO:	1 (B, C)
TENDER DE BUZO:	1 (D)
TENDER DEL BUZO DE RESPETO:	1
TOTAL: 6	(E, F)
ADVERTENCIA	

Estos son los mínimos requeridos de buzos calificados. El análisis de manejo de riesgo operacional (MRO) puede requerir aumentos en estos niveles para garantizar la seguridad de las operaciones de buceo.
NOTAS: (a) El Supervisor de Buceo puede encargarse de las Comunicaciones/Registros según sea necesario. (b) El Operador de Consola también puede encargarse de las Comunicaciones/Registros. (c) No se debe usar SCUBA para el buzo de reserva en operaciones de buceo con suministro de aire desde la superficie. (d) Se requiere un asistente por buzo. El Supervisor de Buceo puede elegir utilizar un asistente no buzo. El Supervisor de Buceo debe asegurarse de que los asistentes no buzos estén completamente instruidos en sus tareas. (e) Seis es el número mínimo de buzos calificados para operaciones de buceo con suministro de aire desde la superficie (configuraciones 1 y 2); sin embargo, se recomienda encarecidamente un equipo de siete o más, según los requisitos de la misión y el análisis de manejo de riesgo operacional (MRO). (f) Todos los buzos deben estar calificados en RCP (reanimación cardio pulmonar). Fuente: Adaptado de US Navy Diving Manual, Rev. 7A, Cap. 8.

A continuación, se detallan las funciones y responsabilidades esenciales de los miembros clave del equipo.

- A. El Supervisor de Buceo es el líder de la operación y tiene la máxima responsabilidad sobre la seguridad y el éxito de esta. Antes de la inmersión, debe realizar una inspección exhaustiva de la estación de buceo, lo que implica llevar a cabo y documentar el Manejo de Riesgos Operacionales (MRO), verificar las listas de chequeo, el kit médico, la cámara hiperbárica y el mantenimiento de los equipos. También es fundamental que compruebe las comunicaciones, revise el plan de contingencia y coordine con las embarcaciones y señales pertinentes.
- Además, el Supervisor de Buceo debe reunir al equipo y explicar detalladamente los objetivos general específicos y tareas a realizar en la operación de buceo, asignando responsabilidades específicas a cada miembro y confirmando su comprensión y disposición mediante un "OK". Durante la inmersión, mantiene una supervisión continua, monitoreando el cumplimiento de los procedimientos y protocolos establecidos. Es responsable de respaldar los cronómetros y controlar el tiempo,

especialmente en caso de despliegue del buzo de respeto. Debe tomar decisiones oportunas en situaciones de emergencia, priorizando siempre la seguridad de los buzos. En caso de emergencia, el Supervisor de Buceo se encarga personalmente del buzo afectado, coordinando su atención y traslado si es necesario. Su liderazgo y diligencia son cruciales para garantizar que todos los miembros del equipo comprendan sus responsabilidades y que los protocolos de seguridad se sigan al pie de la letra, asegurando así el éxito y la seguridad de las operaciones de buceo.

- B. El Operador de Consola es responsable del control de la consola de buceo y de mantener una comunicación eficaz con los buzos. Debe mantener el volumen del equipo alto y estar atento a ruidos inusuales. Es fundamental retransmitir las palabras del supervisor a los buzos y viceversa de forma textual, asegurándose de que los buzos den acuse de recibo. Durante el ascenso, debe recibir confirmación de "OK" de los buzos.

En cuanto al manejo de la consola, el Operador de Consola debe informar la profundidad a intervalos específicos: cada 10 pies (3 metros) durante el descenso, cada 5 pies (1.5 metros) durante el ascenso y cada 1 pie (0.3 metros) antes de la primera parada de descompresión. Debe mantener la presión mínima manométrica (MMP) 10 pies por delante de la profundidad actual de los buzos, ajustando la presión según sea necesario durante la inmersión y el ascenso.

En términos de monitoreo y seguridad, si los buzos se detienen, debe purgar el neumofatometro y comunicar la profundidad en voz alta. Es crucial informar al supervisor si la presión del regulador cae por debajo de la MMP. Además, debe estar atento a la mayor profundidad cuando los buzos estén en movimiento y permanecer alerta ante cualquier ruido inusual.

- C. El Hombre Datos es un buzo calificado responsable de llevar los registros y asegurar el seguimiento preciso de la inmersión. Antes de la operación, debe prepararse adecuadamente, teniendo a mano las Tablas de Descompresión de la Marina de los EE.UU., lápices y hojas de datos, asegurándose de contar con todo el material necesario para la operación.

Durante la inmersión, su labor consiste en marcar todas las paradas, indicando el motivo de cada una, y anotar el reporte del fondo y la mayor profundidad alcanzada. Debe registrar la profundidad, el tiempo en el fondo y cualquier evento importante, manteniendo al día las hojas de trabajo y el registro de buceo de la operación.

En cuanto a la comunicación con el Supervisor de Buceo, el Hombre Datos debe, antes del ascenso, recibir del supervisor la profundidad máxima y el tiempo en el fondo. Proporciona al supervisor la tabulación, la profundidad de la primera parada y los

tiempos de ascenso correspondientes. Informa al supervisor sobre el tiempo de ascenso necesario y los requisitos de descompresión. En la cámara hiperbárica, si es necesario, continúa desempeñándose como responsable de registros, asegurando que toda la información relevante quede adecuadamente documentada.

- D. Los buzos son responsables de confirmar su aptitud y deseo de bucear, informando sobre inmersiones previas o detalles médicos relevantes. Deben reportar cualquier condición de riesgo y comunicar cualquier cambio o malestar durante y después del buceo. Además, es su deber operar y mantener el equipo de buceo en óptimas condiciones, seguir las instrucciones del supervisor y utilizar las señales establecidas en caso de fallo de comunicaciones.
- E. El buzo de respeto debe estar atento durante toda la inmersión, escuchando el reporte de fondo y preparado para actuar en caso de emergencia. Si es necesario, debe realizar el descenso siguiendo el umbilical del buzo afectado, valorando la situación y buscando soluciones sin convertirse en parte del problema.
- F. El tender de buzo tiene la responsabilidad de asistir al buzo en la colocación y retiro del equipo, manteniendo un control positivo a través del umbilical y asegurando que este no tenga exceso de holgura o tensión. En cuanto a comunicación y seguridad, debe responder a todas las señales de tirón y mantener informado al supervisor sobre dichas señales y la cantidad de umbilical en uso. Debe estar alerta ante cualquier señal de emergencia, conocer primeros auxilios y RCP, y asistir en el traslado del buzo afectado a la superficie y a la cámara hiperbárica si es necesario.

1.4. EQUIPAMIENTO MÍNIMO PARA UNA OPERACIÓN

La ejecución segura y eficiente de una operación de buceo asistido depende directamente de la correcta selección, integración y funcionamiento de su equipamiento.

A continuación, se describen los componentes mínimos indispensables que, según normativas de referencia como el US Navy Diving Manual, Rev. 7A, conforman el sistema de soporte vital integral para llevar a cabo una operación de estas características.

1.4.1. Bancos de Aire

En el buceo con suministro desde superficie, los bancos de aire constituyen un componente fundamental del sistema de soporte vital, actuando como la fuente primaria de gas respirable para el buzo. Estos sistemas, compuestos por una batería de cilindros de alta presión interconectados, son cruciales para el sostenimiento de operaciones prolongadas o a profundidades considerables.

La gestión del suministro de gas es una responsabilidad crítica del Supervisor de Buceo, quien debe calcular rigurosamente el volumen de aire requerido para la operación, considerando el tiempo de fondo, los requisitos de descompresión y el número de buzos involucrados. Para garantizar la máxima seguridad, el US Navy Diving Manual estipula que el sistema debe operar bajo un principio de redundancia.

Esto implica configurar un banco de aire primario para la operación normal y un banco secundario de reserva, destinado a activarse ante cualquier contingencia, asegurando así un flujo de gas ininterrumpido al buzo, como se muestra en la Figura 1-5.



Figura 1-5 Banco De Aire

Fuente: <https://coltri.com/es/estaciones-de-acumulacion/>

1.4.2. Trasvasador

El trasvasador es un componente de interconexión de alta presión cuya función es acoplar neumáticamente los bancos de aire con la consola de buceo. Construido como un latiguillo flexible de alta resistencia, está diseñado para operar de forma segura con presiones de trabajo que exceden los 4500 psi.

Sus materiales son seleccionados para resistir la corrosión y la fatiga por presión, mientras que sus extremos cuentan con acoples estandarizados que garantizan una conexión

segura. De esta forma, se asegura una transferencia de gas estanca y controlada entre el sistema de almacenamiento y el panel de distribución.

Frecuentemente, se integra un manómetro en su configuración para la monitorización en tiempo real de la presión de línea durante la transferencia. Este instrumento es un elemento de seguridad vital, pues permite al operador verificar la presión del banco de aire antes de iniciar la carga. Su diseño combina la flexibilidad necesaria para la operación en campo con la robustez estructural indispensable para la fiabilidad y seguridad del sistema, como se muestra en la Figura 1-6.



Figura 1-6 Trasvasadores

Fuente: <https://www.tradeinn.com/diveinn/es/botellas-buceo-trasvasadores/11782/s>

1.4.3. Consola De Buceo

La Consola de Buceo es el centro de control neurálgico del sistema de soporte vital en superficie para operaciones de buceo asistido. Su diseño, compacto y robusto para soportar las condiciones del entorno marino, integra dos funciones primordiales e interdependientes: la gestión del suministro de gas respirable y el monitoreo en tiempo real de los parámetros críticos de la inmersión. Es el único punto de control que conecta directamente al equipo de superficie con el buzo, haciendo de su correcta operación un factor determinante para la seguridad.

En su función de gestión de gas, la consola recibe aire a alta presión (HP) desde los bancos y, a través de una válvula manorreductora de precisión, lo reduce a una presión de trabajo segura y constante, conocida como Presión Mínima Manométrica (MMP).

Posteriormente, lo distribuye de forma controlada a uno o varios buzos simultáneamente mediante un manifold con válvulas selectoras. La correcta calibración de los reguladores es esencial para entregar el aire al buzo a la presión exacta que su profundidad demanda.

Paralelamente, como sistema de monitoreo, utiliza un neumofatometro para medir con precisión la profundidad de cada buzo. Este se complementa con un conjunto de manómetros

de alta y baja presión (LP) que permiten supervisar las presiones en todo el circuito, desde los bancos de aire hasta la línea de suministro final. Esta unidad constituye la interfaz principal a través de la cual el Operador de Consola ejecuta las directrices del Supervisor. La pericia del operador para interpretar estas lecturas y actuar con rapidez es una habilidad que debe ser entrenada hasta volverse instintiva.

Dada su criticidad como enlace directo con la seguridad del buzo, la replicación funcional de la consola de buceo es el pilar fundamental del sistema de entrenamiento simulado que se desarrolla en este trabajo, como se muestra en la Figura 1-7.



Figura 1-7 Consola De Buceo

Fuente: <https://www.amronintl.com/air-control-and-depth-monitoring-system-for-3-diver-2-regulators-1-low-pressure-input>

1.4.4. Umbilical

El umbilical constituye la línea vital, el nexo físico y funcional entre el buzo y el sistema de soporte en superficie. Lejos de ser un simple conducto, este conjunto de líneas integradas está diseñado para proveer múltiples servicios críticos de manera simultánea: el suministro de gas respirable, la línea de comunicaciones de audio bidireccional, el cableado para la transmisión de video y energía para iluminación, y la manguera del neumofatometro para la telemetría precisa de la profundidad.

Con longitudes operativas que comúnmente oscilan entre 50 y 150 metros, su construcción helicoidal o trenzada está optimizada para ofrecer una alta resistencia a la tracción y a la abrasión, garantizando la integridad de los servicios vitales sin comprometer la flexibilidad necesaria para las maniobras del buzo, como se muestra en la Figura 1-8.



Figura 1-8 Umbilical
Fuente: Partida de Salvataje Valparaíso

1.4.5. Sistema de Comunicaciones y Monitoreo

El sistema de comunicaciones y monitoreo subacuático es el canal de mando y control que enlaza de forma bidireccional y en tiempo real al equipo de superficie con el buzo. Este sistema es fundamental, ya que la comunicación clara y constante es una de las barreras de seguridad más importantes en una operación. Integra la transmisión de audio para directrices verbales del Supervisor y reportes del buzo, pero también permite la supervisión acústica continua de la respiración. El ritmo y sonido de la respiración son indicadores clave del estado fisiológico y nivel de estrés del buzo.

Adicionalmente, los sistemas modernos incorporan la capacidad de transmitir video en vivo a través de una cámara montada en el casco. Esta funcionalidad es crucial, ya que proporciona al Supervisor una monitorización visual directa de las tareas y del entorno de trabajo, permitiéndole identificar peligros potenciales como riesgos de enredo o estructuras inestables. Este monitoreo combinado (audio y video) mejora drásticamente la conciencia situacional y la capacidad para tomar decisiones informadas y proactivas, facilitando una gestión más segura y eficiente de la operación, como se muestra en la Figura 1-9.

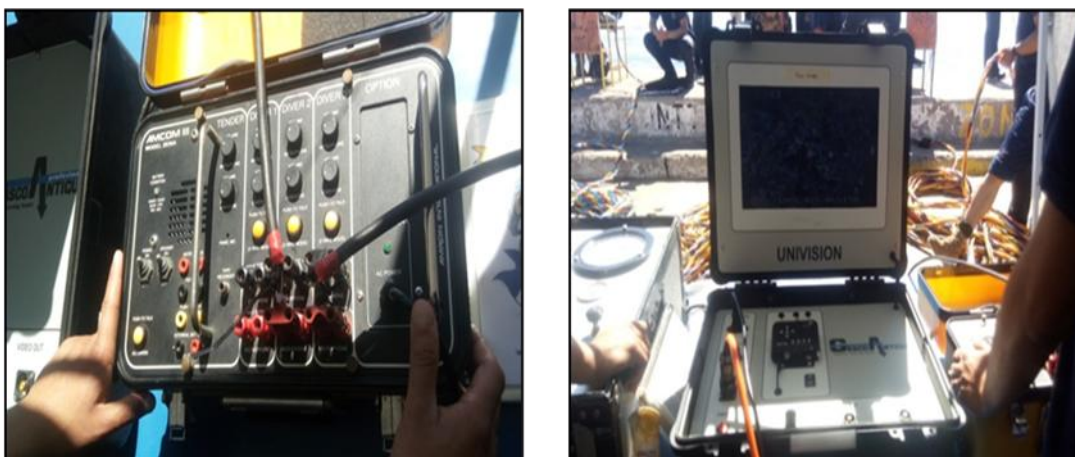


Figura 1-9 Equipo De Comunicaciones Y Monitoreo

Fuente: Partida de Salvataje Valparaíso

1.4.6. Equipamiento de Inmersión del Buzo

El equipamiento personal en el buceo con suministro de superficie constituye un sistema de soporte vital completamente integrado. Representa el punto de contacto directo del buzo con el entorno hostil y es la última barrera de seguridad. Su configuración y uso están estandarizados por normativas de referencia global, como el US Navy Diving Manual, para garantizar la máxima seguridad y eficiencia operativa durante la exposición hiperbárica.

El eje de este sistema es el mecanismo de respiración principal. Este se compone de un casco de buceo (ej. Kirby Morgan KM-37) o una máscara oro-nasal (ej. MK 20 MOD), como por ejemplo entre otros. Estos dispositivos encapsulan y protegen las vías respiratorias, asegurando un sello estanco contra la presión del agua, y albergan los sistemas de comunicación (auricular y micrófono) y la válvula de demanda que suministra el gas al buzo. Este componente se conecta físicamente al umbilical, que actúa como línea vital proveyendo un caudal constante de gas respirable y el enlace de comunicaciones.

Dada la dependencia de un suministro externo, es mandatorio un sistema de redundancia crítico: el Suministro de Gas de Emergencia (EGS). Se trata de un cilindro autónomo, montado en el arnés del buzo, que actúa como una salvaguarda inmediata en caso de un fallo catastrófico del suministro principal desde superficie (ej. rotura o pinzamiento del umbilical). Su porte es obligatorio según estándares internacionales en inmersiones que superan los 60 pies de agua de mar (fsw), proporcionando al buzo la autonomía de gas necesaria para iniciar un ascenso de emergencia controlado.

Para la operatividad y seguridad estructural, el buzo viste un arnés de seguridad robusto. Este componente es mucho más que un chaleco; sirve como la interfaz estructural que soporta la tensión mecánica del umbilical, protegiendo al casco y al cuerpo del buzo de tirones, y funcionando como el punto de izado principal para la recuperación del buzo. A esto se suma un

cinturón de lastre, que permite un control preciso de la flotabilidad para neutralizar el empuje positivo del traje y mantener una plataforma de trabajo estable en el fondo.

El equipamiento se completa con herramientas de seguridad y movilidad. Un cuchillo de buceo es una herramienta indispensable contra los riesgos de enredo con cabos, redes o estructuras sumergidas. El sistema de movilidad se adapta a la tarea: se emplean aletas para trabajos que requieren desplazamiento en la columna de agua, o botas con peso para operaciones que demandan una gran estabilidad en el fondo, como la construcción o el salvataje.

Finalmente, la protección ambiental es un factor fundamental para la seguridad del buzo, cuya configuración se adapta críticamente a las condiciones del medio donde se realiza la operación. El desafío más universal es la gestión térmica. Dado que el agua transfiere el calor corporal unas 25 veces más rápido que el aire, el riesgo de hipotermia es una amenaza constante, incluso en aguas consideradas templadas. Esta condición puede mermar rápidamente la capacidad cognitiva y física del buzo.

Paralelamente, el tipo de traje seleccionado debe responder a amenazas específicas del entorno. Para operaciones en aguas polares (como la Antártida), se requiere un traje seco con aislamiento térmico avanzado para prevenir la hipotermia severa. En contraste, en aguas tropicales (como el Caribe), se usan trajes húmedos ligeros principalmente para proteger contra abrasiones, mientras se gestiona el riesgo opuesto: la hipertermia por esfuerzo.

Un segundo nivel de riesgo, de criticidad creciente, es el trabajo en aguas contaminadas. En puertos, zonas industriales, emisarios submarinos o faenas con riesgo biológico, un traje húmedo (que permite la circulación de agua) es inaceptable. En estos casos, es mandatorio el uso de un traje seco de aislamiento total (tipo vulcanizado o Hazmat), que encapsula herméticamente al buzo para impedir cualquier contacto con agentes químicos o patógenos.

El conjunto sinérgico de todos estos elementos conforma la dotación estándar y el sistema de soporte vital personal para un buzo profesional, como se muestra en la Figura 1-10.



Figura 1-10 Equipamiento de Inmersión del Buzo

Fuente: <https://www.apolinav.cl/index.php/campus/campus-hyatt/>

CAPÍTULO 2: ANÁLISIS FUNCIONAL Y DISEÑO DEL PROTOTIPO

2. ANÁLISIS FUNCIONAL Y DISEÑO DEL PROTOTIPO DE CONSOLA

El desarrollo de un simulador exige una materialización física que replique de manera precisa no solo la funcionalidad, sino también la ergonomía y la interacción de una consola de buceo real. Este capítulo detalla el proceso de análisis, diseño y fabricación del prototipo físico que sirve como base mecatrónica para el sistema de entrenamiento.

El punto de partida fue un análisis funcional de consolas comerciales para tres buzos, equipadas con dos entradas de alta presión (HP) y una de baja presión (LP). El objetivo principal del diseño fue recrear fielmente la disposición, la escala y la interacción del operador, en lugar de buscar una réplica dimensional exacta. Este enfoque pragmático permitió adaptar el diseño a las tecnologías de fabricación disponibles, como el corte láser y la impresión 3D.

Para ello, se tradujo el análisis a un entorno digital mediante un flujo de trabajo con software de Diseño Asistido por Computador (CAD). Se empleó una metodología dual: SolidWorks para el diseño de componentes planos (paneles) optimizados para corte láser, y Autodesk Fusion 360 para el modelado de geometrías complejas orientadas a la manufactura aditiva (impresión 3D), como las carcasas de los manómetros y las perillas de las válvulas. Para la fabricación de estos componentes se seleccionó el material PETG (Tereftalato de Polietileno Glicolizado) por su superior resistencia térmica y mecánica en comparación con filamentos como el PLA, garantizando la durabilidad del prototipo.

Finalmente, para asegurar la portabilidad y protección del simulador, se construyó manualmente una caja de transporte de madera, integrando todos los elementos en un sistema robusto y funcional, cumpliendo así con los requerimientos para alcanzar un nivel de madurez tecnológica TRL 5.

2.1. DISEÑO DEL PANEL DE CONTROL INFERIOR

El panel inferior representa el núcleo funcional del sistema de soporte vital, especializándose en la gestión del suministro, regulación y distribución del gas respirable. Su arquitectura se fundamenta en principios de alta fiabilidad: redundancia en las fuentes, control preciso de la presión y seguridad pasiva. A continuación, se descompone su análisis en una serie de etapas funcionales que describen el recorrido del aire a través del circuito, como se observa en la Figura 2-1 y se esquematiza en el diagrama funcional de la Figura 2-2.



Figura 2-1 Vista Frontal del Panel Inferior
Fuente: Partida de Salvataje Talcahuano

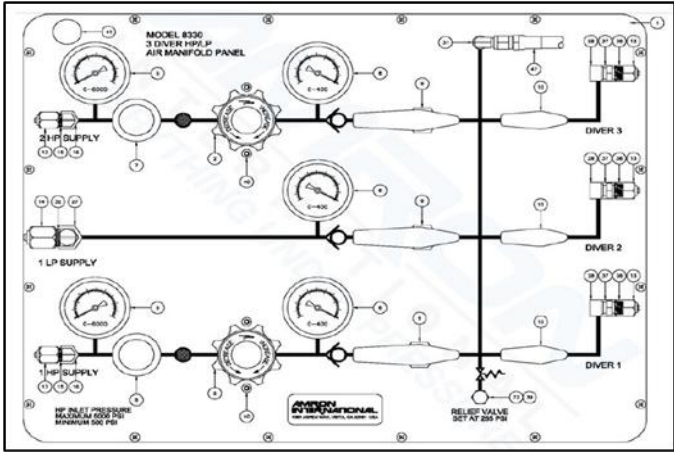


Figura 2-2 Diagrama Funcional del Panel Inferior de
Suministro de Aire (Vista Frontal)

Fuente: <https://www.amronintl.com/commercial-diving-equipment/air-systems/air-control-systems>

2.1.1. Admisión y Monitoreo de Alta Presión (HP)

La arquitectura del panel integra dos circuitos de admisión de alta presión (HP 1 y HP 2) idénticos e independientes, diseñados para conectarse a los bancos de aire primario y secundario, los cuales operan típicamente entre 3000 y 4500 psi.

Cada circuito es gobernado por una válvula de corte multivuelta (tipo aguja o globo) que permite una apertura y cierre gradual para evitar golpes de ariete neumáticos. Siguiendo el circuito del gas respirable, inmediatamente después de cada válvula, un manómetro de 0-6000 psi proporciona una indicación visual directa de la presión remanente en el banco conectado. Esta configuración de redundancia es fundamental, pues permite al operador una gestión precisa del gas y la capacidad de conmutar de forma segura al banco de respaldo para garantizar la continuidad operativa, como se muestra en el diseño 3D de la Figura 2-3.

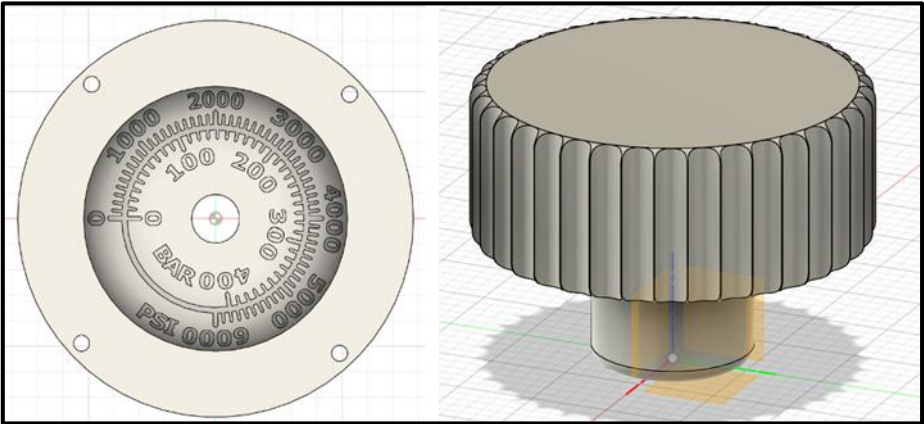


Figura 2-3 Diseño 3D del conjunto de Válvula de Corte y Manómetro de Alta

Presión para el prototipo de simulador
Fuente: Elaboración propia

2.1.2. Control y Visualización de la Presión Mínima Manométrica (MMP)

Siguiendo el circuito del gas, el aire de alta presión (HP) es dirigido a una válvula manorreductora de alta performance, componente central del sistema cuya función es reducir la presión del banco a un nivel de trabajo seguro y estable. Esta presión de salida es la Presión Mínima Manométrica (MMP), una de las variables operativas más críticas que el operador de consola debe gobernar.

Una MMP incorrecta tiene consecuencias fisiológicas directas. Según el US Navy Diving Manual, Rev. 7A, Cap. 8, una MMP insuficiente incrementa drásticamente el trabajo respiratorio del buzo, con riesgo de acumulación de dióxido de carbono (hipercapnia), mientras que una MMP excesiva puede provocar una fuga continua de aire en el regulador del casco (free-flow), consumiendo el suministro de gas aceleradamente.

El ajuste de la MMP no es arbitrario; debe calcularse para garantizar un esfuerzo respiratorio mínimo y la integridad del sistema. La fórmula para su cálculo se muestra en la Figura 2-4, donde D es la profundidad del buzo en pies (fsw), 0.445 es el factor de conversión de pies de agua de mar a psi, y P over-bottom es la "presión sobre el fondo", un valor que depende del equipo y la profundidad y que se obtiene de tablas estandarizadas como la que se muestra en la Figura 2-5.

$$MMP(\text{psig})=(D\times0.445)+P \text{ over-bottom}$$

Figura 2-4 Formula MMP
Fuente: US Navy Diving Manual, Rev. 7A, Cap. 8.

Table 8-1. KM-37 NS Overbottom Pressure Requirements.

Dive Depth	Pressure in psig		
	Minimum	Desired	Maximum
0-60 fsw	90*	135	165
61-130 fsw	135	135	165
131-190 fsw	165**	165	165

* Not approved for use with a double exhaust kit installed. Instead use a minimum of 135 psig.

** 135 psig is authorized for diver life support systems not capable of sustaining 165 psig over bottom due to system design limitations.

Figura 2-5 Tabla De Presión Sobre La Profundidad
Fuente: US Navy Diving Manual, Rev. 7A, Cap. 8.

El regulador de la consola permite un ajuste fino de la Presión Mínima Manométrica (MMP), actuando como el nexo de control crítico entre la fuente de alta presión y el gas que el buzo respira en su casco o máscara full face. Este componente, que ofrece un rango de ajuste (típicamente de 0 a 400 psi), tiene su salida monitoreada de forma continua por un manómetro dedicado. La correcta gestión de este parámetro, cuyo conjunto de control físico se modeló en 3D como se ilustra en la Figura 2-6, es fundamental para el sistema de soporte vital.

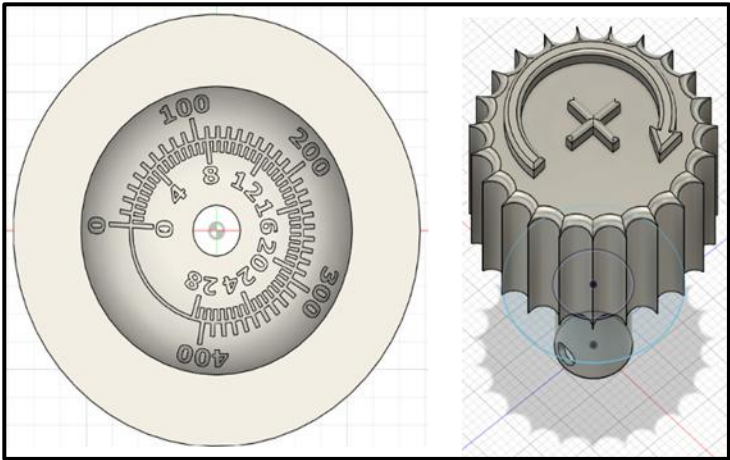


Figura 2-6 Diseño 3D del conjunto de control de la MMP,
compuesto por la Válvula Reguladora y su Manómetro

Fuente: Elaboración propia

2.1.3. Integración de Suministros y Manifold de Distribución

La arquitectura neumática del panel integra tres fuentes de aire: dos circuitos de alta presión (HP) ya regulados y una línea de baja presión (LP) proveniente de un compresor de

superficie. La gestión de estas fuentes es una decisión operativa explícita, dictada por el Supervisor y ejecutada por el Operador de Consola.

Mediante un conjunto de válvulas selectoras, el operador dirige deliberadamente el flujo de la fuente seleccionada banco primario, secundario o compresor hacia un manifold de suministro común. Este colector actúa como un pulmón de baja presión que, a su vez, alimenta los circuitos independientes de cada buzo.

La integridad del sistema se ve reforzada por válvulas antirretorno (check valves) instaladas en cada línea de entrada al manifold. Su función es prevenir de forma pasiva el reflujo de aire desde una línea de mayor presión hacia una de menor presión. Este es un mecanismo de seguridad crucial que evita la contaminación cruzada entre fuentes y la pérdida de gas, garantizando el aislamiento neumático del sistema, como se puede apreciar en el montaje de la Figura 2-7.

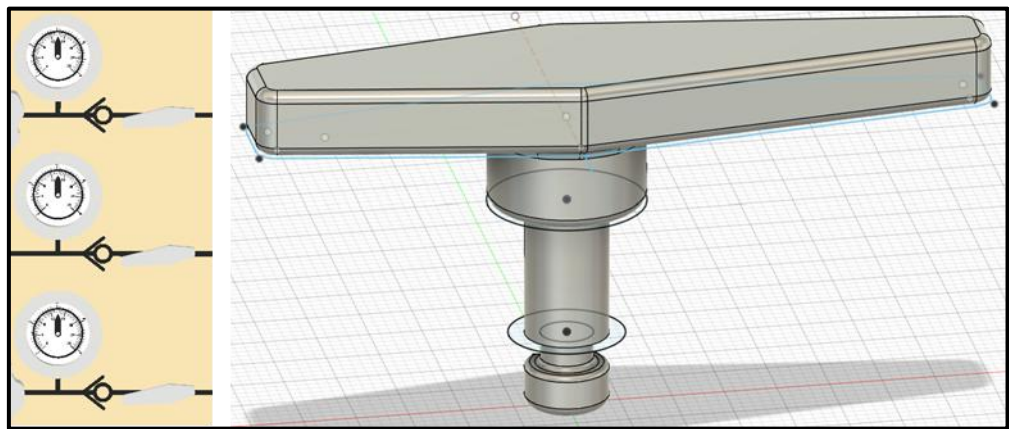


Figura 2-7 Diseño 3D del montaje de una de las válvulas selectoras en el circuito de suministro del panel inferior

Fuente: Elaboración propia

2.1.4. Distribución y Seccionamiento de Suministro a Buzos

A partir del manifold de baja presión, el flujo de gas se dirige a un subsistema de distribución configurado con tres circuitos paralelos e independientes, cada uno dedicado a un buzo. Esta arquitectura es fundamental, pues constituye el punto de control final y la salvaguarda individual para cada participante de la inmersión.

El control de cada línea se realiza mediante una válvula de bola de cuarto de vuelta. La selección de este tipo de válvula responde a criterios de seguridad operacional: su capacidad de actuación rápida (un giro de 90°) permite al operador habilitar o aislar el suministro de forma casi instantánea, y su diseño ofrece una indicación de estado visual inequívoca, ya que la posición de la palanca confirma de manera infalible si el circuito está abierto o cerrado.

Como último eslabón en la cadena de seguridad, cada circuito integra una válvula de sobrepresión (relief valve). Calibrada para activarse si la presión excede los 285 psi, su función es ventilar el exceso de presión, protegiendo al buzo y al equipamiento de una posible falla catastrófica del regulador principal. Esta capacidad de gestión individual es una herramienta de control esencial en la respuesta a contingencias, y su implementación física se puede observar en el modelo 3D completo del panel, como se muestra en la Figura 2-8.

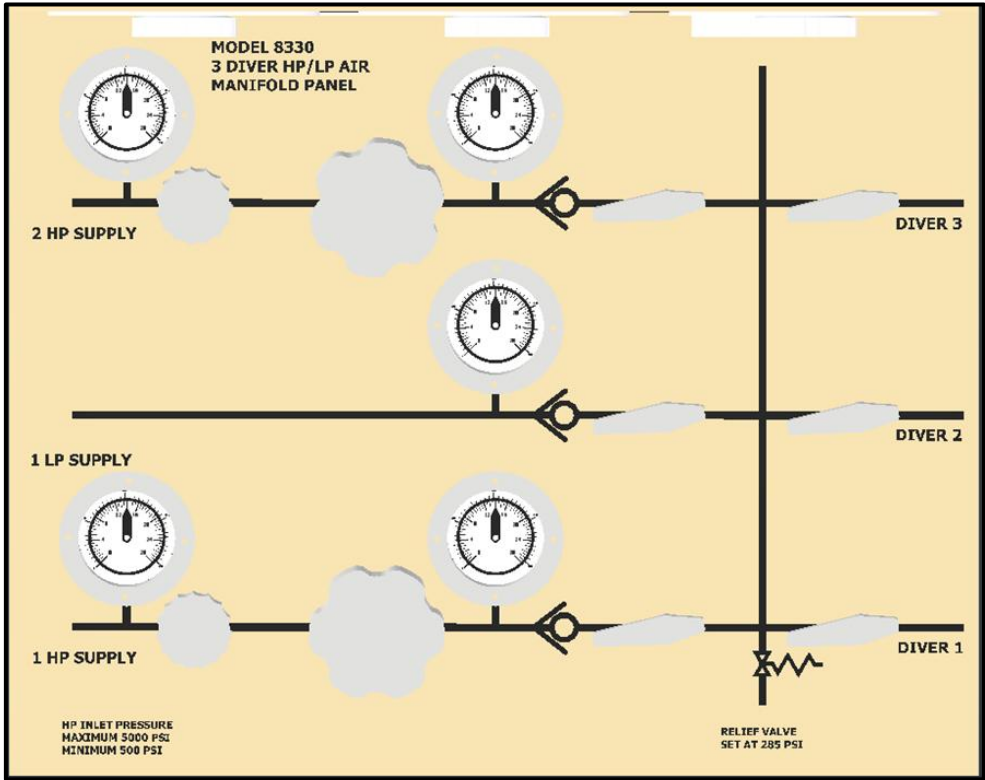


Figura 2-8 Modelo CAD del Panel Inferior de la Consola de Buceo

Fuente: Elaboración propia

2.2. DISEÑO DEL PANEL DE CONTROL SUPERIOR

El panel superior de la consola constituye el subsistema de telemetría y monitoreo, una interfaz cuya función primordial es proveer al equipo de superficie la data en tiempo real de la profundidad de cada buzo y un repetidor de la Presión Mínima Manométrica (MMP).

El núcleo de este panel es el sistema neumofatometro, un dispositivo de medición de profundidad que opera bajo el principio de equilibrio hidrostático. Su funcionamiento se basa en inyectar un flujo de aire a través de una manguera dedicada en el umbilical, cuyo extremo abierto se posiciona a la altura del pecho del buzo. Esta ubicación es fundamental, ya que establece la referencia fisiológica estándar para el cálculo de la presión ambiental. El flujo de aire purga el agua de la manguera hasta que la presión interna del gas iguala la presión hidrostática que rodea al buzo. Al interrumpir el flujo, la columna de agua externa comprime el

aire atrapado, estabilizando la presión interna para que coincida con precisión con la presión ambiente a la profundidad exacta del pecho del buzo. Esta presión es registrada por profundímetros de alta precisión en la consola, proporcionando una lectura directa y fiable de la profundidad. Dicha telemetría es, según el US Navy Diving Manual, indispensable para la gestión segura de los perfiles de ascenso y el cálculo de las paradas de descompresión.

Adicionalmente, este módulo está alimentado por una línea de baja presión (LP) que sirve a un manómetro repetidor de la MMP y a una alarma acústica de baja presión, calibrada para activarse por debajo de los 100 psi como alerta de una posible falla en el suministro. El diseño de este panel, con válvulas de seccionamiento para cada buzo, presenta un desafío de ingeniería: un accionamiento incorrecto puede provocar una comunicación cruzada entre circuitos, resultando en lecturas erróneas.

Para ilustrar la transición del diseño digital a la implementación física, se presenta a continuación una comparativa entre el modelo CAD y el prototipo físico ensamblado, como se muestra en la Figura 2-9.



Figura 2-9 Comparativa entre el diseño CAD del panel superior (izquierda) y el prototipo físico ensamblado (derecha)

Fuente: Elaboración propia

CAPITULO 3: INTEGRACIÓN DE LA ARQUITECTURA DE HARDWARE

3. **INTEGRACIÓN DE LA ARQUITECTURA DE HARDWARE**

Una vez definido el diseño funcional y la construcción física del prototipo en el capítulo anterior, este capítulo aborda la integración de la arquitectura de hardware que dota de inteligencia y capacidad de respuesta a esta estructura mecánica. El objetivo es instrumentar la réplica física con la electrónica de control necesaria para emular con precisión los parámetros de una operación de buceo, transformándola en un sistema interactivo.

La arquitectura se fundamenta en un enfoque de sistemas distribuidos con una alimentación centralizada. La energía para todo el sistema es suministrada por un adaptador AC/DC de 12V y 3A, cuya tensión es posteriormente regulada a 5V mediante un convertidor buck (LM2596), como se muestra en la Figura 3-1, para alimentar de forma estable toda la lógica de control. El control inteligente se distribuye en tres microcontroladores que operan sinérgicamente para garantizar la modularidad y el rendimiento en tiempo real: un ESP32-S3 para la consola inferior, encargado de la emulación de manómetros y el monitoreo de válvulas; un ESP8266 como coprocesador en la misma consola, dedicado a la lectura de alta resolución de las entradas analógicas; y un ESP32 para la consola superior, responsable de la simulación de profundímetros y del manómetro repetidor de MMP, gestionando sus actuadores y sensores a través de expansores de puertos y multiplexores.

Este capítulo describirá de forma modular la instrumentación de ambas consolas, detallando la justificación técnica de los componentes y las consideraciones de diseño críticas, con el fin de construir una plataforma física robusta capaz de cumplir con el Nivel de Madurez Tecnológica (TRL) 5 propuesto.



Figura 3-1 Regulador De Voltaje Step Down Dc-dc Lm2596
Con voltímetro

Fuente: <https://www.mercadolibre.cl/regulador-de-voltaje-step-down-dcdc-lm2596-con-voltimetro/up/MLCU21031373>

3.1. MÓDULO DE CONTROL DE SUMINISTRO (CONSOLA INFERIOR)

La instrumentación del panel inferior constituye la base mecatrónica del simulador, siendo responsable de emular las funciones críticas de gestión de gas. Para alcanzar la modularidad, el rendimiento en tiempo real y la fiabilidad requeridos, se implementó una arquitectura de control distribuida. Esta arquitectura se compone de dos microcontroladores que operan de forma sinérgica: un ESP32-S3 actúa como la unidad de control principal, gestionando los actuadores (motores) y sensores digitales (microinterruptores), mientras que un ESP8266 funciona como un coprocesador dedicado exclusivamente a la adquisición de datos analógicos de alta resolución desde los potenciómetros, aislando estas mediciones sensibles del ruido eléctrico del sistema.

3.1.1. Controlador Principal (ESP32-S3)

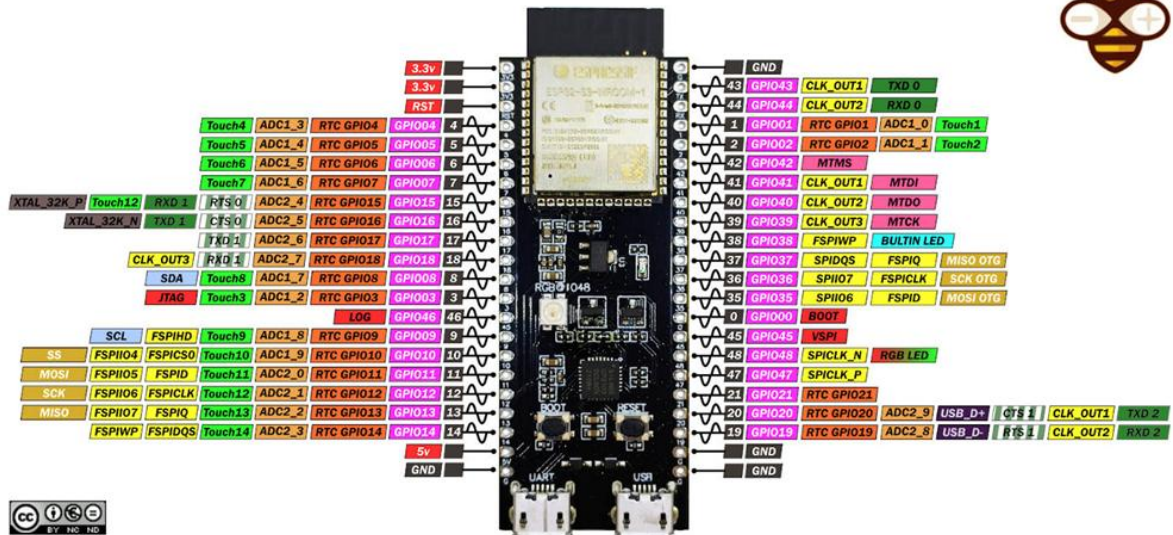
La unidad de control principal de este módulo se fundamenta en un microcontrolador ESP32-S3. La selección de este componente se basa en su potente procesador dual-core Xtensa® LX7 operando a 240 MHz, que proporciona la capacidad de cómputo necesaria para gestionar tareas de control en tiempo real y comunicaciones de red de forma concurrente.

Adicionalmente, su alta densidad de pines de Entrada/Salida de Propósito General (GPIO) es indispensable para gobernar la considerable cantidad de actuadores y sensores del sistema.

Finalmente, su módulo de radiofrecuencia integrado de 2.4 GHz, compatible con los estándares 802.11 b/g/n, provee la capacidad de comunicación inalámbrica Wi-Fi, un requisito fundamental del diseño.

Este microcontrolador actúa como el nexo central, orquestando la interacción entre los componentes de simulación. Su función es recibir comandos vía Wi-Fi, procesarlos y traducirlos en secuencias de pulsos digitales para los motores, mientras que, de forma concurrente, monitorea el estado de los microinterruptores para reportar las acciones del operador a la HMI.

El diagrama de pines completo del microcontrolador se presenta en la Figura 3-2, mientras que la Tabla 3-1 detalla la topología de interconexión específica implementada.



Fuente: <https://mischianti.org/esp32-s3-devkitc-1-high-resolution-pinout-and-specs/>

Tabla 3-1

GUÍA DE CONEXIONES (ESP32-S3)		
Componente y Función	Pin (GPIO)	Uso Específico
Motor 1 (Manómetro MMP 1)	4	Bobina IN1
	5	Bobina IN2
	6	Bobina IN3
	7	Bobina IN4
Motor 2 (Banco Presión Primario)	10	Bobina IN1
	11	Bobina IN2
	12	Bobina IN3
	13	Bobina IN4
Motor 3 (Banco Presión Secundario)	14	Bobina IN1
	15	Bobina IN2
	16	Bobina IN3
	17	Bobina IN4
Motor 4 (Manómetro MMP 2)	18	Bobina IN1
	21	Bobina IN2
	35	Bobina IN3
	36	Bobina IN4
Válvulas (Sensores)	41	Sensor "BUZO _ROJO"
	38	Sensor "BUZO _VERDE"
	39	Sensor "BUZO _AMARILLO"

	42	Sensor "BANCO PRIMARIO"
	40	Sensor "BAJA PRESION"
	47	Sensor "BANCO SECUNDARIO"
	1	Serial (TX)
Comunicación	3	Serial (RX)

Fuente: Elaboración propia

La emulación de la respuesta dinámica de los indicadores de presión se implementa mediante cuatro motores paso a paso modelo 28BYJ-48, cada uno gobernado por un controlador basado en el circuito integrado ULN2003. Estos actuadores se asignan a la simulación física de los manómetros de alta presión (HP) y de Presión Mínima Manométrica (MMP).

La selección de este componente se fundamenta en la resolución angular que proporciona su caja reductora interna. Al operarlo en modo de medio paso, es posible alcanzar 4096 micropasos por revolución. Dicha resolución es un requisito de diseño indispensable para reproducir con fidelidad el desplazamiento suave y preciso de las agujas en los manómetros analógicos, garantizando así una experiencia de simulación realista para el operador, como se muestra en la Figura 3-3.

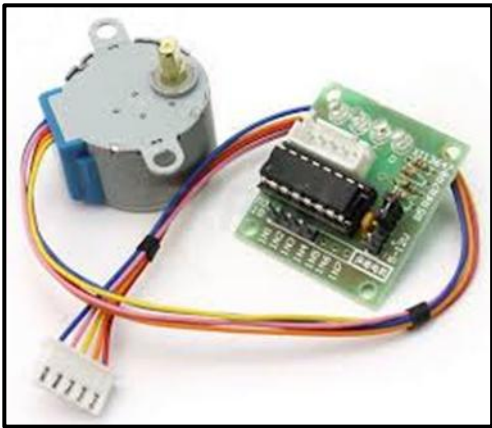


Figura 3-3 Motor 28BYJ-48

Fuente: <https://www.prometec.net/motor-28byj-48/>

La posición física de las válvulas de control de gas se captura mediante seis microinterruptores KW11. Estos componentes actúan como sensores electromecánicos, proporcionando una señal digital binaria (abierto/cerrado) al microcontrolador ESP32-S3 para indicar el estado de las válvulas selectoras.

La distribución de estos sensores es la siguiente: dos monitorean la selección del banco de alta presión (HP1, HP2), uno el del banco de baja presión (LP), y los tres restantes confirman

el seccionamiento del suministro a cada uno de los tres buzos. Su robustez mecánica y la confirmación táctil que ofrecen son ideales para una interfaz de entrenamiento de uso intensivo, como se muestra en la Figura 3-4.



Figura 3-4 Micro interruptor KW11

Fuente:https://es.aliexpress.com/item/1005006085607858.html?spm=a2g0o.order_list.order_list_main.335.1578194dyOu14J&gatewayAdapt=glo2esp

3.1.2. Coprocesador de Adquisición Analógica (ESP8266)

Para garantizar la máxima resolución en la lectura de las acciones de regulación fina del operador, se implementó un subsistema dedicado compuesto por un microcontrolador ESP8266, como el que se muestra en la Figura 3-5. La decisión de utilizar un procesador separado para esta tarea responde a la necesidad de aislar las lecturas analógicas sensibles del ruido eléctrico generado por los motores en el controlador principal, una decisión de diseño crítica para la fiabilidad de la señal.

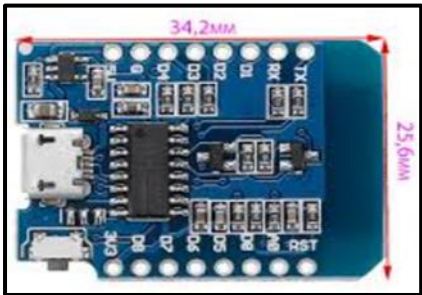


Figura 3-5 Microcontrolador ESP8266

Fuente: <https://www.electrocomponentes.es/>

Este módulo lee la posición de cuatro potenciómetros a través de un conversor analógico-digital (ADC) externo, el ADS1115, que se ilustra en la Figura 3-6. Este componente de 16 bits de resolución ofrece una estabilidad y precisión muy superiores al ADC interno del microcontrolador, lo que es fundamental para capturar con fidelidad los ajustes sutiles realizados por el operador.

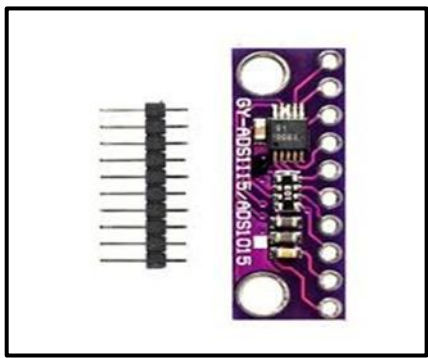


Figura 3-6 ADS1115 de 16 bits

Fuente: <https://www.mechatronicstore.cl/>

Para la simulación de los controles físicos, se emplearon dos tipos de potenciómetros, como se ven en la Figura 3-7. Dos potenciómetros multivuelta 3296W se utilizan para simular el ajuste fino de los reguladores de MMP, replicando la operación de múltiples giros de las válvulas manorreductoras reales. Complementariamente, otros dos potenciómetros de giro simple emulan el control de las válvulas de admisión de los bancos de alta presión.

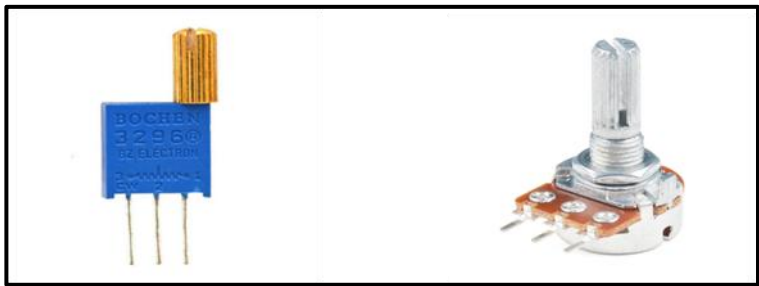


Figura 3-7 Potenciómetros multivuelta 3296W y Potenciómetro de 1K Ohms

Fuente: <https://laelectronica.com.gt/>

La interconexión entre el microcontrolador ESP8266 y el conversor ADC se establece mediante el protocolo de comunicación serial I²C. El ADC actúa como un front-end de adquisición, digitalizando con alta resolución las señales de los potenciómetros. La Tabla 3-2

detalla el esquema de conexiones físicas completo, incluyendo las líneas de comunicación I²C (SCL y SDA) y la alimentación.

Tabla 3-2
GUÍA DE CONEXIONES (ESP8266)

GUÍA DE CONEXIONES (ESP8266)			
Componente de Origen	Pin de Origen	Pin de Destino	Componente Destino
ESP8266 (NodeMCU)	GPIO 5 (D1)	SCL	ADS1115
ESP8266 (NodeMCU)	GPIO 4 (D2)	SDA	ADS1115
ESP8266 (NodeMCU)	3.3V	VDD	ADS1115
ESP8266 (NodeMCU)	GND	GND	ADS1115
ADS1115	A0	Pin central	Potenciómetro 1
ADS1115	A1	Pin central	Potenciómetro 2
ADS1115	A2	Pin central	Potenciómetro 3
ADS1115	A3	Pin central	Potenciómetro 4
Potenciómetros (x4)	Pin 1 (VCC)	3.3V	ESP8266
Potenciómetros (x4)	Pin 3 (GND)	GND	ESP8266

Fuente: Elaboración propia

3.2. **MÓDULO DE CONTROL DEL PANEL SUPERIOR**

La consola superior funciona como el subsistema de telemetría y monitoreo del simulador, emulando los cuatro indicadores analógicos cruciales: tres profundímetros y un manómetro repetidor de la Presión Mínima Manométrica (MMP). A diferencia del panel inferior, este módulo centraliza su control en un único microcontrolador ESP32 modelo DevKitC), cuyo diagrama de pines se muestra en la Figura 3-8.

La principal complejidad de ingeniería en este panel radica en la alta densidad de actuadores y sensores. Para gestionar eficientemente esta cantidad de Entradas/Salidas (E/S), se implementaron técnicas de expansión de puertos y multiplexación analógica, optimizando el uso de los pines del microcontrolador. La guía completa de conexiones se detalla en la Tabla 3-3.

Tabla 3-3
GUÍA DE CONEXIONES ESP32-DevKitC

GUÍA DE CONEXIONES ESP32-DevKitC		
Nombre	GPIO	Dispositivo Conectado
GND	-	Tierra
IO23	GPIO 23	Multiplexor (Pin S0)
IO22	GPIO 22	Expansores PCF8574 SCL
TX	GPIO 1	-
RX	GPIO 3	-
IO21	GPIO 21	Expansores PCF8574 SDA
GND	-	Tierra
IO19	GPIO 19	Multiplexor (Pin S1)
IO18	GPIO 18	-
IO5	GPIO 5	-
IO17	GPIO 17	-
IO16	GPIO 16	-
IO4	GPIO 4	-
IO0	GPIO 0	-
IO2	GPIO 2	-
IO15	GPIO 15	-
D1	GPIO 8	-
D0	GPIO 7	-
CLK	GPIO 6	-
3V3	-	Módulos (MUX, PCF)
EN	-	-
VP	GPIO 36	-
VN	GPIO 39	-
IO34	GPIO 34	Multiplexor (Pin SIG)
IO35	GPIO 35	-
IO32	GPIO 32	-
IO33	GPIO 33	-
IO25	GPIO 25	-
IO26	GPIO 26	Multiplexor (Pin S3)
IO27	GPIO 27	Multiplexor (Pin S2)
IO14	GPIO 14	-

IO12	GPIO 12	-
GND	-	Tierra
IO13	GPIO 13	-
D2	GPIO 9	-
D3	GPIO 10	-
CMD	GPIO 11	-
5V	-	-

Fuente: Elaboración propia

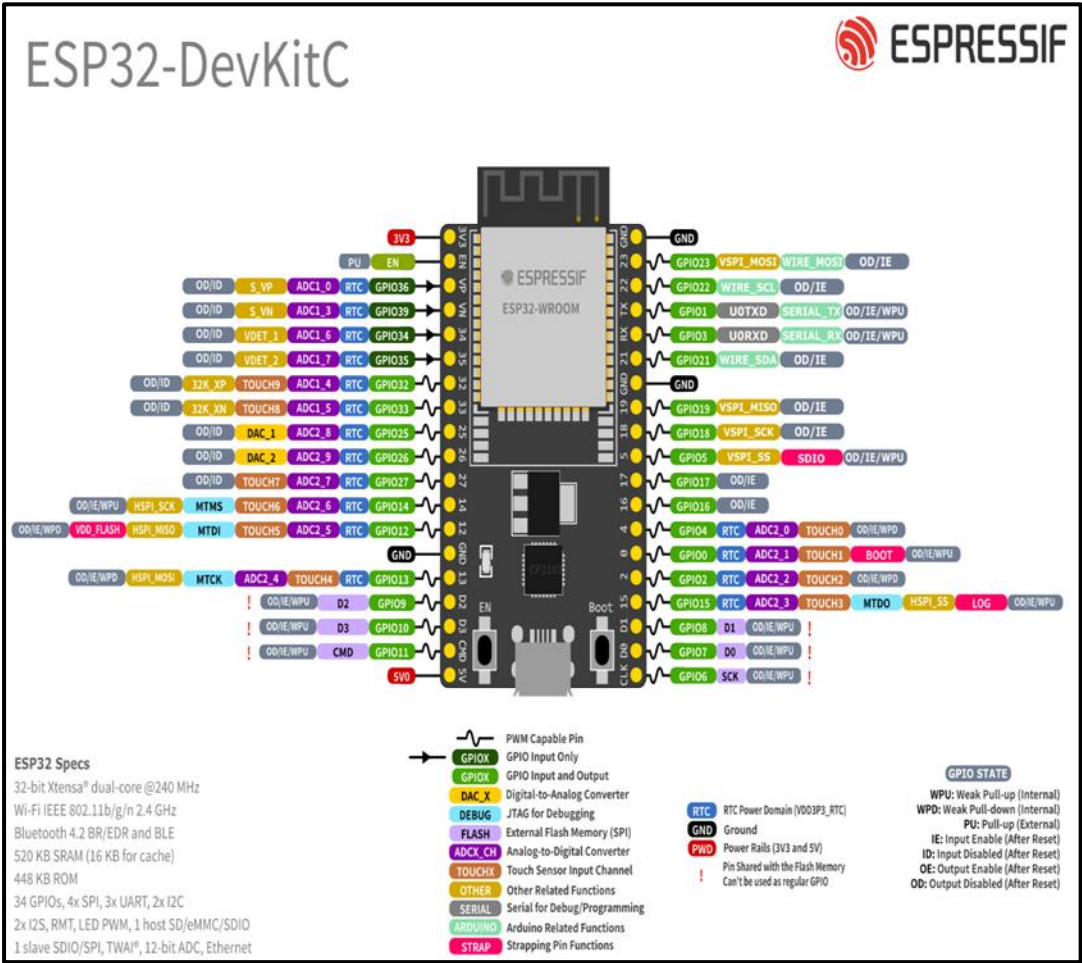


Figura 3-8 Pintout ESP32-DevKitC

Fuente: https://docs.espressif.com/projects/esp-dev-kits/en/latest/esp32/esp32-devkitc/user_guide.html#overview

3.2.1. Actuadores para Manómetros

Para la simulación física de los indicadores analógicos de la consola superior, se emplean cuatro motores paso a paso modelo 28BYJ-48, idénticos a los utilizados en el panel inferior. Su función en este panel es emular los tres profundímetros que muestran la profundidad de cada

buzo y el manómetro repetidor de la Presión Mínima Manométrica (MMP). Cada motor es gobernado por un controlador de basado en el circuito integrado ULN2003, que actúa como interfaz entre las señales de control y las bobinas del motor.

La justificación técnica para la selección de este actuador, ilustrado en la Figura 3-3, se fundamenta en su capacidad para lograr una emulación precisa del comportamiento de una aguja analógica. La clave reside en su caja reductora interna (con una relación aproximada de 64:1), que, al operar el motor en modo de medio paso (half-step), permite alcanzar una resolución angular de 4096 micropasos por revolución. Esta alta resolución es un requisito indispensable para reproducir un desplazamiento suave y sin saltos perceptibles (jitter), lo cual es crucial para que el operador en entrenamiento perciba el movimiento como el de un instrumento mecánico real.

3.2.2. Expansión de Salidas Digitales por I²C (PCF8574)

La arquitectura centralizada del panel superior presenta un desafío de ingeniería significativo: la alta densidad de Entradas/Salidas (E/S) requeridas. El sistema debe gobernar 16 líneas digitales para los cuatro motores paso a paso y, simultáneamente, leer 7 señales analógicas de los potenciómetros. Esta demanda de 23 pines supera la capacidad del microcontrolador ESP32, un problema que se agrava por una restricción de hardware crítica: la activación de la comunicación Wi-Fi, un requisito indispensable del proyecto, genera un conflicto con el segundo conversor analógico-digital (ADC2), volviendo sus pines inutilizables para lecturas analógicas fiables.

Esta limitación obligó a adoptar una estrategia de optimización de pines. Para las salidas digitales, se optó por la expansión de E/S utilizando dos módulos PCF8574, como el que se muestra en la Figura 3-9. Cada módulo es un expansor de 8 bits que opera sobre el bus de comunicación I²C, permitiendo gobernar las 16 líneas de los motores con solo dos pines del ESP32: SDA (Datos) y SCL (Reloj).

Aunque la comunicación serial I²C introduce una latencia teórica en comparación con el control directo por GPIO, la arquitectura es viable gracias a la capacidad de procesamiento del ESP32 (240 MHz) y la configuración del bus para operar en "Modo Rápido" a 400 kHz. Esto permite que las transacciones se completen en microsegundos, una velocidad suficiente para enviar las secuencias de pasos y lograr el movimiento suave requerido para la simulación. La configuración del bus y el mapeo de los motores a cada expansor (identificados por sus direcciones I²C 0x20 y 0x24) se detallan en la Tabla 3-4 y la Tabla 3-5.

No obstante, durante las pruebas de integración se corroboró experimentalmente que la carga computacional concurrente (gestión de Wi-Fi, MQTT y la HMI) introducía retardos (jitter)

que afectaban la fluidez de los actuadores. Este hallazgo práctico valida que una futura evolución del sistema se beneficiaría de una arquitectura aún más distribuida, con microcontroladores dedicados para las tareas de control en tiempo real estricto, separadas de la lógica de comunicaciones.

Tabla 3-4
Parámetros del Bus I²C (PCF8574)

Parámetros del Bus I²C (PCF8574)	
Parámetro	Valor / Conexión
Línea de Datos (SDA)	GPIO 21
Línea de Reloj (SCL)	GPIO 22
Velocidad de Operación	400 kHz

Fuente: Elaboración propia

Tabla 3-5
Mapeo de Motores en Expansores PCF8574

Mapeo de Motores en Expansores PCF8574		
Motor ID	Conectado a Expansor (Dirección I²C)	Pines del Expansor Asignados
Motor 1	PCF8574 (0x20)	0, 1, 2, 3
Motor 2	PCF8574 (0x20)	4, 5, 6, 7
Motor 3	PCF8574 (0x24)	0, 1, 2, 3
Motor 4	PCF8574 (0x24)	4, 5, 6, 7

Fuente: Elaboración propia

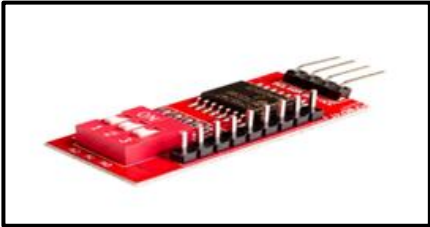


Figura 3-9 Módulo Interfaz PCF8574 I2C Expansor de Pines
Fuente: <https://maxelectronica.cl/>

3.2.3. Multiplexación de Entradas Analógicas (CD74HC4067)

La emulación de los siete controles analógicos del panel superior (válvulas de aguja del sistema neumofatometro) se implementa con potenciómetros, lo que presenta un desafío de hardware crítico en el ESP32: la interferencia del periférico Wi-Fi un requisito indispensable con el Conversor Analógico-Digital (ADC2), volviendo sus pines inutilizables para lecturas analógicas fiables.

Esta limitación restringe las entradas disponibles a las del ADC1, que eran insuficientes para conectar directamente los siete potenciómetros. Para superar esta restricción y garantizar tanto la integridad de las lecturas como la estabilidad de la conexión Wi-Fi, se implementó una solución de multiplexación analógica utilizando el circuito integrado CD74HC4067 (Figura 3-10).

Este componente es un multiplexor de 16 canales que funciona como un conmutador controlado digitalmente. Permite que un único pin del ADC1 (libre de conflictos) pueda leer secuencialmente los siete sensores. Para ello, el ESP32 utiliza cuatro pines de salida digital (GPIO 23, 19, 27 y 26) para enviar una dirección binaria de 4 bits (S0-S3), seleccionando qué canal se conecta a la salida común. Dicha salida común se conectó al GPIO 34 (el pin del ADC1).

Esta estrategia optimiza drásticamente el uso de los recursos del microcontrolador, permitiendo una lectura eficiente de múltiples controles analógicos con solo cinco pines del ESP32. El mapeo específico de cada función simulada al canal correspondiente del multiplexor se presenta en la Tabla 3-6.

Tabla 3-6
Mapeo de Módulo CD74HC4067

Módulo CD74HC4067	
Función Simulada	Canal MUX
Válvula del Neumofatometro - Buzo Rojo	0
Válvula del Neumofatometro - Buzo Verde	2
Válvula de Suministro de Aire a Buzo Verde	4
Válvula de Suministro Aire Neumo General	6
Válvula del Neumofatometro - Buzo Amarillo	8
Válvula de Suministro de Aire a Buzo Amarillo	10
Válvula de Suministro de Aire a Buzo Rojo	12

Fuente: Elaboración propia

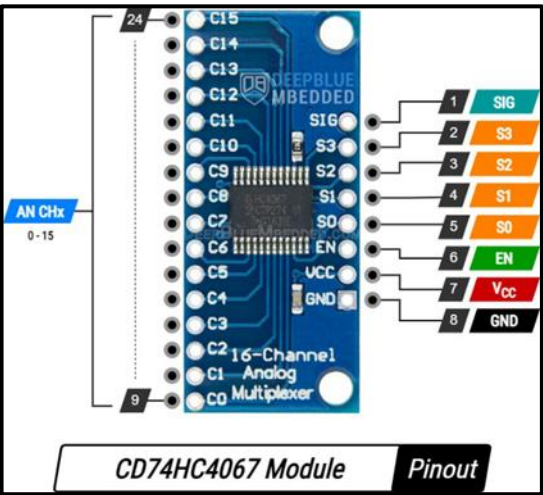


Figura 3-10 Módulo CD74HC4067

Fuente: <https://deepbluembedded.com/arduino-cd74hc4067-analog-multiplexer-library-code/>

CAPÍTULO 4: DESARROLLO DE FIRMWARE Y PROTOCOLO DE COMUNICACIÓN

4. DESARROLLO DE FIRMWARE Y PROTOCOLO DE COMUNICACIÓN

La transición de un prototipo físico inerte a un simulador de entrenamiento funcional se logra a través del firmware: el software embebido que constituye la inteligencia y la capacidad de respuesta del sistema.

Este capítulo abarca la arquitectura lógica que gobierna los microcontroladores, detallando cómo las interacciones físicas del usuario y los comandos remotos se traducen en respuestas dinámicas que emulan con precisión una operación de buceo.

El desarrollo, estandarizado sobre el ecosistema de Arduino/C++, se fundamenta en tres pilares de diseño estratégicos para dominar la complejidad de un sistema distribuido y en tiempo real.

El primero es la Modularidad y Abstracción mediante la Programación Orientada a Objetos, donde la lógica de cada subsistema (motores, válvulas, comunicación) se ha encapsulado en clases con archivos de cabecera (.h) e implementación (.cpp); este enfoque transforma el código principal de cada microcontrolador en un simple orquestador, mejorando drásticamente la mantenibilidad, legibilidad y reutilización del código.

El segundo pilar es la Comunicación Desacoplada y Escalable, lograda con el protocolo MQTT bajo un modelo de publicación/suscripción.

Este paradigma desacopla completamente los dispositivos (clientes) a través de un servidor central (Broker), para el cual se implementó Mosquitto dada su robustez y amplia adopción como solución de código abierto, facilitando además la validación y depuración de la arquitectura de comunicación mediante herramientas como MQTT Explorer que permiten visualizar en tiempo real los tópicos y mensajes intercambiados, lo que aporta una enorme flexibilidad para añadir o modificar componentes de hardware sin alterar la lógica de comunicación existente.

Finalmente, el tercer pilar es la gestión de la Concurrencia y Tiempo Real con FreeRTOS en los módulos basados en ESP32, un sistema operativo que permite dividir el programa en tareas concurrentes con prioridades asignadas y aprovechar la arquitectura de doble núcleo del procesador mediante afinidad de núcleo (core affinity), garantizando que las operaciones críticas como el control de motores nunca se vean afectadas por la latencia de la red.

A continuación, este capítulo desglosará la implementación del firmware específico para cada módulo, demostrando cómo estos tres pilares de diseño se aplican para dar vida al simulador.

4.1. ARQUITECTURA DE COMUNICACIÓN DESACOPLADA CON MQTT

El pilar fundamental que permite la operación coordinada y en tiempo real de los distintos módulos del simulador es el protocolo Message Queuing Telemetry Transport (MQTT). A diferencia de los modelos tradicionales cliente-servidor, MQTT se basa en un paradigma de publicación/suscripción (Pub/Sub), una elección de diseño estratégica para garantizar el desacoplamiento, la escalabilidad y la resiliencia del sistema.

En esta arquitectura, tal como se muestra en el diagrama 4-1 lógica del sistema de simulación, los dispositivos (clientes) no se comunican directamente. Toda la comunicación es mediada por una entidad central llamada Broker, que actúa como una central de correos inteligente. Los clientes asumen roles de publicadores (envían datos) o suscriptores (reciben datos), interactuando a través de canales llamados tópicos. El Broker recibe cada mensaje, lo clasifica por su tópico y lo distribuye instantáneamente a todos los clientes suscritos. Este modelo desacopla por completo a los participantes, lo que fue fundamental para facilitar el desarrollo modular del simulador.

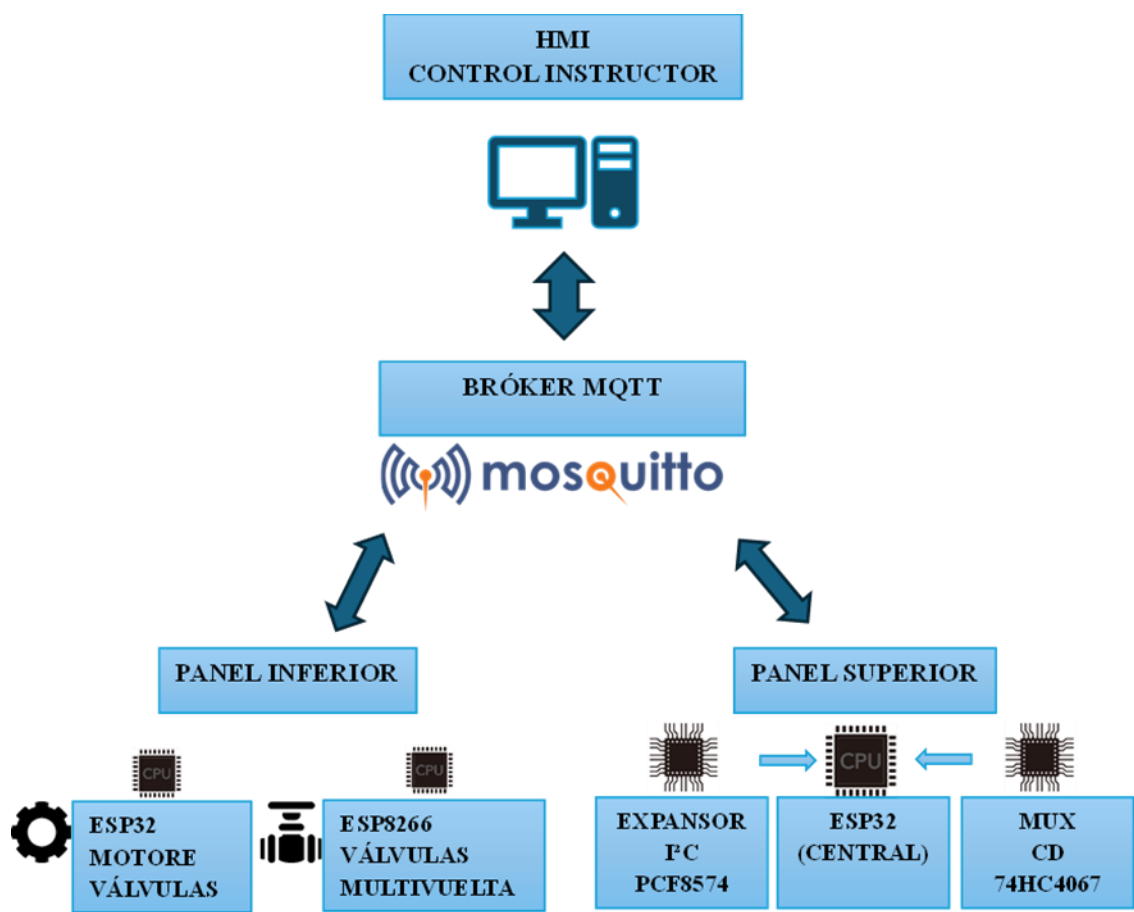


Diagrama 4-1 Lógica Del Sistema De Simulación

Fuente: Elaboración propia

4.1.1. Principios Fundamentales y Calidad de Servicio (QoS)

La fiabilidad de la comunicación en MQTT se puede ajustar según la criticidad del mensaje mediante los niveles de Calidad de Servicio (QoS). Esta característica permite balancear la garantía de entrega con la sobrecarga de la red. En este proyecto, se utilizaron estratégicamente dos de los tres niveles disponibles, como se detalla en la Tabla4-1.

Tabla 4-1
NIVELES DE CALIDAD DE SERVICIO (QOS) EN MQTT

NIVELES DE CALIDAD DE SERVICIO (QOS) EN MQTT Y SU APLICACIÓN EN EL PROYECTO			
Nivel	Garantía de Entrega	Mecanismo Técnico	Aplicación en el Proyecto
QoS 0	"A lo sumo una vez"	"Disparar y olvidar". El emisor envía el mensaje una vez sin solicitar confirmación. Es el método más rápido, pero no garantiza la entrega.	Utilizado para la telemetría de alta frecuencia de los potenciómetros. Dada la continua actualización, la pérdida ocasional de un paquete no es crítica.
QoS 1	"Al menos una vez"	Apretón de manos de 2 pasos (PUB/PUBACK). El emisor reintenta el envío hasta recibir una confirmación. Garantiza la entrega, aunque puede generar duplicados.	Utilizado para todos los comandos de motores y mensajes de estado críticos como el LWT y el estado de conexión de las válvulas, donde es imperativo asegurar la entrega de la orden.
QoS 2	"Exactamente una vez"	Apretón de manos de 4 pasos. Es el nivel más fiable y sin duplicados, pero con mayor latencia.	No se utilizó, ya que el nivel de fiabilidad de QoS 1 fue considerado suficiente para la operación en una red local controlada.

Fuente: Elaboración propia

4.1.2. Mensajes Retenidos y Última Voluntad y Testamento (LWT)

Para asegurar un sistema robusto, se implementaron dos características avanzadas de MQTT. La primera son los Mensajes Retenidos (Retained Messages), donde el Broker almacena el último mensaje publicado con esta bandera activada; así, cuando un nuevo cliente se suscribe, recibe inmediatamente el último estado conocido. Todos los mensajes de estado de motores y válvulas se publican como retenidos, garantizando que la HMI se sincronice al instante al iniciarse.

La segunda característica es la Última Voluntad y Testamento (LWT), que actúa como un "interruptor de hombre muerto" digital. Si un cliente se desconecta de forma abrupta, el Broker publica un mensaje predefinido en su nombre. La clase ComunicadorMQTT implementa esta funcionalidad de forma genérica, permitiendo que cualquier módulo la utilice para notificar fallos. Por ejemplo, mientras que el SistemaValvulas la usa para publicar {"estado":"desconectado"}, el módulo de potenciómetros podría, de igual manera, configurarla para reportar su propio fallo en un tópico diferente, notificando así al sistema central sobre la pérdida de un componente específico.

4.1.3. El Broker Central y Herramientas de Diagnóstico

Se seleccionó Eclipse Mosquitto como Broker central, por su ligereza y robustez. Se instaló en el PC de la HMI, convirtiéndolo en el sistema nervioso de la red. La instalación en Windows es directa: se descarga el instalador de mosquitto.org, se ejecuta y se configura como un servicio que se inicia automáticamente. Para este proyecto, se utilizó la configuración por defecto, que opera en el puerto 1883 y permite conexiones anónimas, una práctica segura dentro de una red local (WLAN) aislada. Durante el desarrollo, fue útil ejecutarlo en modo verboso para monitorear la actividad en tiempo real: mosquitto -v.

Para gestionar y depurar la comunicación que fluye a través de este broker, la aplicación MQTT Explorer fue un recurso indispensable. Esta herramienta gráfica se conecta al Broker y se suscribe a todos los tópicos (#), permitiendo visualizar el árbol completo de temas, monitorear el flujo de mensajes en tiempo real, validar los formatos de datos JSON y simular comandos manualmente para probar la respuesta del hardware de forma aislada.

4.1.4. Estructura Jerárquica de Tópicos

Se diseñó una estructura de tópicos jerárquica e intuitiva, donde todos los temas comienzan con el prefijo buceo/ para crear un espacio de nombres (namespace) único. La estructura, detallada en la Tabla 4-2, facilita la organización y el uso de comodines. Por ejemplo, el firmware se suscribe a buceo/comandos/consolasuperior/motores/+ para recibir comandos para cualquier motor.

Tabla 4-2
DESGLOSE DE LA ESTRUCTURA DE TÓPICOS MQTT DEL PROYECTO

DESGLOSE DE LA ESTRUCTURA DE TÓPICOS MQTT DEL PROYECTO		
Componente del Tópico	Descripción	Ejemplo
buceo/	Raíz del proyecto.	buceo/
{panel}	Sección física del simulador.	panel_inferior, panel_superior
{subsistema}	Componente funcional.	motores, valvulas, presion
{identificador}	Componente individual.	motor_1, BUZO_ROJO
{tipo}	Propósito del mensaje.	comando, estado

Fuente: Elaboración propia

4.2. **MODULARIDAD MEDIANTE (POO)**

Para gestionar la complejidad del sistema, el firmware se ha estructurado siguiendo el paradigma de Programación Orientada a Objetos (POO). Este enfoque organiza el código en clases que representan cada subsistema funcional (ej., SistemaMotores), actuando como "cajas negras" que encapsulan su lógica y exponen una interfaz simple.

Las clases principales desarrolladas son: ComunicadorMQTT, que abstrae la conectividad Wi-Fi/MQTT; SistemaMotores, que gestiona el control y movimiento de los actuadores; SistemaValvulas, que asegura la lectura fiable de sensores con filtros debounce; y LectorPotenciometros, que maneja la adquisición de datos analógicos.

Esta separación de responsabilidades permite que el archivo principal de cada microcontrolador (ej., CONSOLA_INFERIOR_MOTOR.ino) funcione como un simple orquestador, cuya única tarea es instanciar los objetos y coordinar la ejecución.

4.2.1. **Clase ComunicadorMQTT: El Núcleo de la Red**

La clase ComunicadorMQTT es el pilar de la comunicación en todo el proyecto. Diseñada como un componente transversal y reutilizable, su propósito es abstraer toda la complejidad de la gestión de una conexión de red inalámbrica robusta y fiable.

Para lograr la portabilidad, el firmware utiliza el protocolo mDNS (Multicast DNS) en lugar de una IP fija, permitiendo a los microcontroladores localizar el broker mediante un nombre consistente como victor.local.

Para asegurar la compatibilidad con Windows, se instaló el servicio Bonjour Print Services de Apple en el PC que aloja el broker. El firmware implementa una lógica de resolución

robusta que, como se muestra en la Figura 4-1, primero intenta una consulta DNS estándar y, si esta falla, recurre a una consulta mDNS para descubrir la IP del broker.

Una función crítica de la clase es la reconexión automática inteligente; si la conexión se pierde, se activa un mecanismo de exponential backoff que incrementa progresivamente el tiempo de espera entre intentos, evitando saturar la red.

Para un código reactivo, la clase utiliza un sistema de callbacks asíncronos que permite ejecutar funciones personalizadas ante eventos específicos como `alConectarMQTT()` o `alRecibirMensajeMQTT()`, eliminando la necesidad de sondeos constantes del estado de la conexión.

Además, asegura el manejo de suscripciones persistentes, almacenando internamente los tópicos de interés para re-suscribirse automáticamente tras una reconexión.

```
IPAddress ip_servidor;
// Intenta convertir la dirección directamente a una IP. Si falla, es un nombre de host.
if (!ip_servidor.fromString(broker_address)) {
    // Al no ser una IP, se asume que es un nombre de host (ej: "broker.local" o "broker.hivemq.com").
    Serial.printf("Resolviendo %s vía mDNS... ", broker_address);

    // Inicia el servicio mDNS para poder resolver nombres .local.
    if (!MDNS.begin(WiFi.getHostname())) {
        Serial.println(" mDNS no pudo iniciarse");
    }

    // Primero, intenta resolver usando DNS estándar, ideal para nombres de host de internet.
    if (!WiFi.hostByName(broker_address, ip_servidor)) {
        // Si el DNS falla, lo informa.
        Serial.println(" Fallo en la resolución DNS");

        // Como último recurso, intenta resolver usando mDNS, ideal para redes locales.
        IPAddress mdns_ip = MDNS.queryHost(broker_address);
        // Compara con una IP inválida (0.0.0.0) para saber si la resolución mDNS falló.
        if (mdns_ip == IPAddress(0,0,0,0)) {
            Serial.println(" Fallo también con mDNS");
            // Si ambos métodos (DNS y mDNS) fallan, la inicialización no puede continuar.
            return false;
        }
        // Si mDNS tuvo éxito, asigna la IP encontrada.
        ip_servidor = mdns_ip;
    }
    // Imprime la IP que se logró resolver.
    Serial.println(" OK " + ip_servidor.toString());
}
```

Figura 4-1 Lógica de Resolución de Host con Fallback a mDNS
Fuente: Elaboración propia

Finalmente, para una supervisión robusta del estado del sistema, la clase soporta la configuración de mensajes de Última Voluntad y Testamento (LWT), una funcionalidad avanzada del protocolo MQTT. El LWT es un mecanismo crucial que actúa como un "interruptor de hombre muerto" digital. Su configuración se realiza mediante el método `configurarLWT`, cuya implementación se detalla en la Figura 4-2. Este método permite al firmware definir un

mensaje (payload), un tópico, una Calidad de Servicio (QoS) y una bandera de retención, que se almacenan internamente en el objeto ComunicadorMQTT.

Es importante destacar que este mensaje no se envía en el momento de la configuración, sino que se registra en el broker MQTT como parte del paquete de conexión (CONNECT). Cuando el método reconectar() es invocado, verifica si un LWT ha sido configurado; en caso afirmativo, incluye estos parámetros en la solicitud de conexión. El broker, al recibir esta solicitud, almacena el mensaje LWT y lo asocia a la sesión del cliente. Si en algún momento la conexión TCP con el dispositivo se interrumpe de forma abrupta, es decir, sin que el cliente envíe un paquete DISCONNECT formal (por ejemplo, debido a un fallo de alimentación o una caída de la red), el broker publicará automáticamente el mensaje LWT almacenado en el tópico especificado. Esto permite que un sistema de monitoreo, suscrito a dicho tópico, sea notificado de manera inmediata y fiable sobre el fallo de un componente del simulador, mejorando significativamente la resiliencia y la capacidad de diagnóstico del sistema.

En resumen, la clase ComunicadorMQTT transforma la compleja tarea de gestionar la comunicación de red en un componente modular y fiable, permitiendo que el resto del firmware se concentre exclusivamente en su lógica de control y simulación.

```
bool ComunicadorMQTT::configurarLWT(const String& topico, const String& mensaje, uint8_t qos, bool retenido) {
    // Valida que los parámetros no estén vacíos, ya que son obligatorios para un LWT válido.
    if (topico.isEmpty() || mensaje.isEmpty()) {
        // Si el modo debug está activo, informa del error.
        if (debug_activo) {
            Serial.println(" Error: LWT requiere tópico y mensaje válidos");
        }
        // Retorna falso para indicar el fallo.
        return false;
    }

    // Guarda los parámetros del LWT en las variables miembro de la clase para usarlos en la conexión.
    lwt_topico = topico;      // Guarda el tópico del LWT.
    lwt_mensaje = mensaje;    // Guarda el mensaje del LWT.
    lwt_qos = qos;           // Guarda la calidad de servicio del LWT.
    lwt_retenido = retenido;  // Guarda la bandera de retención del LWT.
    lwt_configurado = true;   // Marca que el LWT ha sido configurado y está listo para usarse.

    // Si el modo debug está activo, imprime los detalles del LWT configurado.
    if (debug_activo) {
        Serial.printf(" LWT configurado: %s -> %s (QoS:%d, Retenido:%s)\n",
            topico.c_str(), mensaje.c_str(), qos, retenido ? "Sí" : "No");
    }

    // Retorna verdadero para indicar que la configuración fue exitosa.
    return true;
}
```

Figura 4-2 Implementación del Método configurarLWT

Fuente: Elaboración propia

4.2.2. Clase SistemaMotores: Control de Actuadores y Simulación Física

La clase SistemaMotores es el componente de software encargado de traducir los comandos digitales en movimientos físicos y realistas de las agujas de los manómetros, encapsulando toda la lógica de control desde la recepción de órdenes vía MQTT hasta la ejecución precisa de cada paso.

En el panel inferior, la clase utiliza la librería estándar AccelStepper para un control directo sobre los pines GPIO del ESP32, gestionando de forma autónoma los perfiles de aceleración y deceleración para emular la inercia de un instrumento mecánico real.

Una característica clave es la eficiencia energética, lograda mediante una máquina de estados con dos modos: DETENIDO y EN_MOVIMIENTO. Como se muestra en la Figura 4-3, el método actualizar() es el corazón de esta lógica; ejecutado continuamente por la tarea de alta prioridad tareaMotores, este bucle itera sobre cada motor y comprueba su estado. Si un motor está en estado EN_MOVIMIENTO, el código verifica si ha llegado a su destino con la función distanceToGo(). Si aún le quedan pasos por recorrer, se invoca el método run(), que calcula y ejecuta el siguiente paso del perfil de movimiento. Por el contrario, si ha llegado a su destino (distanceToGo() == 0), el estado se cambia a DETENIDO, se invoca inmediatamente la función desenergizarBobinas() para cortar la corriente del motor y se publica su estado final.

Esta lógica de control asegura que los motores solo consuman energía mientras se mueven, evitando el sobre calentamiento y reduciendo significativamente el consumo eléctrico del simulador durante la operación.

```
void SistemaMotores::actualizar() {
    for (uint8_t i = 0; i < NUM_MOTORES; i++) {
        if (_estados[i] == EN_MOVIMIENTO) {
            if (_motores[i]->distanceToGo() != 0) {
                _motores[i]->run();
            } else {
                _estados[i] = DETENIDO;
                desenergizarBobinas(i);
                publicarEstadoMotor(i);
            }
        }
    }
}
```

Figura 4-3 Implementación de la Máquina de Estados en el Método
Fuente: Elaboración propia

En contraste, en el panel superior, para resolver la escasez de pines GPIO en el único ESP32, el control se delega a dos expansores de puertos I2C (PCF8574). Esta solución requirió el desarrollo de una clase de driver especializada, AccelStepper_PCF8574, que actúa como un adaptador de software, traduciendo cada comando de paso en una transacción de datos a través

del bus I2C. Aunque la comunicación I2C introduce una latencia inherente, la velocidad alcanzable es muy superior a la necesaria para una simulación realista.

La Figura 4-4 detalla la interfaz privada de la clase SistemaMotores, revelando su estructura interna. En ella se define el enumerador EstadoMotor (DETENIDO, EN_MOVIMIENTO), que forma la base de su máquina de estados interna. También se declaran los miembros de datos, como el arreglo de punteros _motores para gestionar los objetos de cada motor, y los métodos privados que encapsulan las acciones de bajo nivel, tales como ordenarMovimientoAbsoluto() y desenergizarBobinas().

Ambas implementaciones son controladas a través de una interfaz de comandos MQTT común, donde el método procesarMensajeMQTT(topic, payload) analiza los mensajes JSON para identificar comandos de posición absoluta ("p"), movimiento relativo ("r") o calibración a cero ("z"), junto con parámetros opcionales de velocidad y aceleración. En ambos microcontroladores ESP32, toda la lógica de actualización se ejecuta dentro de una tarea de FreeRTOS dedicada, tareaMotores, a la que se le asigna la máxima prioridad y se fija su ejecución en el Núcleo 1 (APP_CPU). Esta afinidad de núcleo garantiza que el control de los motores, que requiere una temporización precisa para ser fluido, nunca sea interrumpido por las operaciones de red, que se ejecutan en el Núcleo 0.

```
51 // Método para forzar la publicación del estado de todos los motores.
52 void publicarTodosLosEstados(uint8_t qos = 1);
53
54 private: // Miembros y métodos privados.
55 // Define un enumerado para los posibles estados de un motor.
56 enum EstadoMotor { DETENIDO, EN_MOVIMIENTO };
57
58 // Referencia al objeto de comunicación MQTT.
59 ComunicadorMQTT& comunicador;
60 // Arreglo de punteros a los objetos de motor. Se usa punteros para poder crearlos dinámicamente.
61 AccelStepper_PCF8574* _motores[NUM_MOTORES];
62 // Arreglo para almacenar el estado actual (DETENIDO o EN_MOVIMIENTO) de cada motor.
63 EstadoMotor _estados[NUM_MOTORES];
64 // Para mantener el estado de salida de cada PCF8574 y permitir escrituras write8
65 uint8_t _pcf_output_states[NUM_EXPANDERS];
66
67 // --- MÉTODOS PRIVADOS ---
68 // Da la orden de mover un motor a una posición absoluta.
69 void ordenarMovimientoAbsoluto(uint8_t motorIndex, long posicion, float velocidad);
70 // Da la orden de mover un motor un número de pasos relativo.
71 void ordenarMovimientoRelativo(uint8_t motorIndex, long pasos, float velocidad);
72
73 // Establece la posición actual de un motor a cero.
74 void establecerPosicionCero(uint8_t motorIndex);
75 // Apaga todas las bobinas de un motor para ahorrar energía y evitar sobrecalentamiento.
76 void desenergizarBobinas(uint8_t motorIndex);
77 // Publica el estado (posición, velocidad, si se está moviendo) de un motor específico.
78 void publicarEstadoMotor(uint8_t motorIndex, uint8_t qos = 1);
79 // Encola un mensaje MQTT para ser enviado por una tarea en segundo plano (FreeRTOS).
80 void encolarPublicacionMQTT(const char* topico, const char* payload, bool retenido, uint8_t qos);
81 };
```

Figura 4-4 Interfaz Privada de la Clase SistemaMotores.

Fuente: Elaboración propia

4.2.3. Clase SistemaValvulas: Monitorización de Sensores Digitales

La clase SistemaValvulas es responsable de la adquisición y reporte fiable del estado de los seis sensores digitales (NUMERO_DE_VALVULAS), específicamente de los microinterruptores mapeados a los pines GPIO 41, 38, 39, 42, 40, 47, que indican si las válvulas físicas están abiertas o cerradas.

Su diseño se enfoca en la integridad de los datos y la eficiencia de la comunicación. Para garantizar la integridad, implementa un algoritmo de antirrebote (debounce) por software, una técnica indispensable para mitigar el problema de los rebotes eléctricos en los interruptores mecánicos.

El método actualizar(), como se detalla en la Figura 4-5, solo valida un cambio de estado si la señal del sensor permanece estable durante un periodo configurable de 25 milisegundos (DEBOUNCE_VALVULAS_MS), eliminando así las lecturas falsas causadas por fluctuaciones momentáneas.

Además, el firmware contempla la configuración eléctrica de cada sensor, aplicando una inversión lógica (LOGICA_INVERTIDA_VALVULAS) para aquellos sensores cuyo estado "abierto" corresponde a una señal LOW.

En cuanto a la eficiencia, la clase adopta una estrategia de publicación por excepción, enviando un mensaje MQTT únicamente cuando se detecta un cambio confirmado.

Para asegurar la sincronización con los clientes, esta estrategia se complementa con una publicación forzada del estado de todas las válvulas cada 30 segundos (INTERVALO_PUBLICACION_FORZADA_MS) y al momento de conectarse al broker.

Toda esta lógica se ejecuta de forma concurrente dentro de su propia tarea de FreeRTOS, tareaSistemaValvulas, a la cual se le asigna una prioridad intermedia y se fija su ejecución en el Núcleo 1 (APP_CPU).

Esta afinidad de núcleo asegura que la lectura de los sensores, que se realiza cada 25 ms, no se vea afectada por la latencia de la red, al mismo tiempo que cede prioridad a la tarea de control de motores, que es más crítica en términos de temporización.

```

95
96
97 void SistemaValvulas::actualizar() {
98     unsigned long tiempo_actual = millis(); // Obtiene el tiempo actual una vez por ciclo para eficiencia.
99
100     // Comprueba si ha pasado el tiempo para forzar una publicación de todos los estados.
101     if (tiempo_actual - ultima_publicacion_forzada >= INTERVALO_PUBLICACION_FORZADA_MS) {
102         publicarTodosLosEstados(); // Publica todos los estados para sincronización.
103         ultima_publicacion_forzada = tiempo_actual; // Reinicia el temporizador de publicación forzada.
104     }
105
106     // Procesa cada válvula para detectar cambios.
107     for (size_t i = 0; i < NUMERO_DE_VALVULAS; i++) {
108         bool lectura_actual = digitalRead(valvulas[i].pin); // Lee el estado digital del pin.
109
110         // Si la lectura actual es diferente de la anterior, es un posible cambio.
111         if (lectura_actual != valvulas[i].ultimo_estado_leido) {
112             valvulas[i].ultimo_tiempo_debounce = tiempo_actual; // Reinicia el temporizador de debounce.
113         }
114
115         // Si ha pasado suficiente tiempo desde el último cambio (el estado es estable).
116         if ((tiempo_actual - valvulas[i].ultimo_tiempo_debounce) > DEBOUNCE_VALVULAS_MS) {
117             // Y si el estado estable es diferente del último estado que se publicó.
118             if (lectura_actual != valvulas[i].estado_publicado) {
119                 valvulas[i].estado_publicado = lectura_actual; // Actualiza el estado publicado.
120                 publicarEstadoValvula(i); // Publica el nuevo estado.
121
122                 // Imprime un log del cambio en el monitor serie.
123                 // Aplica la lógica de inversión para mostrar el estado correcto (ABIERTA/CERRADA).
124                 bool estaAbierta = LOGICA_INVERTIDA_VALVULAS[i] ? !lectura_actual : lectura_actual;
125                 Serial.printf(" Actualizar Válvula %s cambió a: %s\n",
126                             NOMBRES_VALVULAS_JSON[i],
127                             estaAbierta ? "ABIERTA" : "CERRADA");
128             }
129         }
130
131         valvulas[i].ultimo_estado_leido = lectura_actual; // Guarda la lectura actual para la siguiente iteración.
132     }
133 }

```

Figura 4-5 Implementación del Algoritmo Debounce y Sincronización Periódica.

Fuente: Elaboración propia

4.2.4. Clases LectorPotenciometros / SistemaPotenciometros: Monitorización Analógica

La adquisición de datos analógicos provenientes de los potenciómetros es una tarea que se ha abordado con dos estrategias de hardware y software distintas, encapsuladas en las clases `LectorPotenciometros` y `SistemaPotenciometros`.

Para el panel inferior, donde la prioridad es la máxima fidelidad y aislamiento de la señal, la lectura de los cuatro potenciómetros se delega a un microcontrolador ESP8266 que actúa como un coprocesador especializado.

Esta arquitectura distribuida aísla las mediciones analógicas sensibles del ruido electromagnético (EMI) generado por los motores en el ESP32 principal, garantizando la fidelidad de la señal.

El firmware se basa en un bucle secuencial no bloqueante encapsulado en la clase LectorPotenciometros. Para garantizar una señal estable, se aplica un filtro de suavizado

exponencial, un método de procesamiento digital de señales (DSP) que no solo atenúa el ruido, sino que introduce una sutil inercia en la respuesta, emulando de forma más fidedigna el comportamiento de una válvula real.

La fórmula de este filtro, implementada directamente en el método actualizarLecturas() como se muestra en la Figura 4-8, se basa en la ecuación en diferencias de primer orden mostrada en la Figura 4-6, utilizando un factor de suavizado (α) de 0.4.

$$V_{suavizado} = (1 - \alpha)V_{suavizado(t-1)} + (\alpha \cdot V_{crudo(t)})$$

Figura 4-6 Ecuación del Filtro de Suavizado Exponencial

Fuente: Proakis, J. G., & Manolakis, D. G. (2007). Digital Signal Processing: Principles, Algorithms, and Applications (4th ed.). Pearson Prentice Hall.

Una vez que la señal ha sido estabilizada por el filtro, su valor crudo se normaliza a un rango porcentual para una visualización intuitiva en la HMI, un proceso conocido como escalado Mín-Máx.

Este se aplica con la fórmula mostrada en la Figura 4-7, donde Vactual es el valor estabilizado del ADC, y Vmin y Vmax son los límites del rango calibrado del sensor (ADC_VALOR_MIN_REAL y ADC_VALOR_MAX_REAL en el código).

$$P = \left(\frac{V_{actual} - V_{min}}{V_{max} - V_{min}} \right) \cdot 100$$

Figura 4-7 Ecuación de Normalización por Escalado Mín-Máx

Fuente: Hastie, T., Tibshirani, R., & Friedman, J. (2009). The Elements of Statistical Learning: Data Mining, Inference, and Prediction. Springer.

Finalmente, el firmware optimiza la comunicación mediante una detección de cambios por umbral, publicando un mensaje MQTT solo si la variación del valor supera un límite predefinido (UMBRAL_CAMBIO_REPORTE), y asegura la sincronización mediante un heartbeat periódico cada 30 segundos.

Al no usar un sistema operativo de tiempo real, su firmware se basa en un eficiente bucle principal no bloqueante (loop()).

```

// --- Lógica de Lectura y Filtrado del ADC ---
void LectorPotenciometros::actualizarLecturas(bool forzar) {
    // Si no se está forzando la lectura y no ha pasado el tiempo mínimo, no hace nada.
    if (!forzar && (millis() - _ultimoTiempoLectura < INTERVALO_LECTURA_ADC_MS)) {
        return; // Sale de la función para no saturar el bus I2C.
    }
    // Actualiza la marca de tiempo de la última lectura.
    _ultimoTiempoLectura = millis();

    // Recorre cada uno de los potenciómetros.
    for (int i = 0; i < NUMERO_POTENCIOMETROS; i++) {
        // Lee el valor crudo del canal correspondiente del ADC.
        int16_t valorCrudo = _ads.readADC_SingleEnded(CANALES_ADC[i]);
        // Aplica un filtro de suavizado exponencial para reducir el ruido de la lectura.
        _valoresSuavizados[i] = (1.0f - FACTOR_SUAIVIZADO) * _valoresSuavizados[i] + (FACTOR_SUAIVIZADO * valorCrudo);
        // Redondea el valor suavizado y lo limita a un rango válido (calibración).
        int16_t nuevoValor = constrain(round(_valoresSuavizados[i]), ADC_VALOR_MIN_REAL, ADC_VALOR_MAX_REAL);

        // Comprueba si la diferencia entre el nuevo valor y el último valor estable es mayor que el umbral.
        if (abs(nuevoValor - _valoresEstables[i]) > UMBRAL_CAMBIO_REPORTE) {
            // Si el cambio es significativo, actualiza el valor estable.
            _valoresEstables[i] = nuevoValor;
            // Activa la bandera para indicar que este valor debe ser publicado.
            _flagsCambio[i] = true;
        }
    }
}

```

Figura 4-8 Implementación del Algoritmo de Filtrado y Detección de Cambios

Fuente: Elaboración propia

En contraste con el panel inferior, la arquitectura del panel superior es centralizada en un único microcontrolador ESP32, lo que presenta un desafío de hardware conocido: la activación del periférico Wi-Fi genera un conflicto que vuelve inestables y poco fiables las lecturas en los pines del segundo conversor analógico-digital (ADC2).

Para superar esta restricción, la clase SistemaPotenciometros implementa una solución de hardware/firmware que gestiona la lectura de los siete potenciómetros a través de un multiplexor analógico (CD74HC4067).

El firmware utiliza cuatro pines de control digital (S0-S3) para seleccionar secuencialmente cada uno de los canales del multiplexor. De esta forma, la señal analógica de cada potenciómetro es dirigida, una a la vez, hacia un único y fiable pin de entrada del ADC1 (SIG_PIN = 34), el cual no se ve afectado por la actividad de la red inalámbrica.

El fragmento de código más importante que ilustra esta lógica es el método leerYPublicarCambios(), como se muestra en la Figura 4-9.

Como se observa en el código, el método itera sobre los canales predefinidos, selecciona cada uno en el multiplexor con selectChannel(), y tras una breve pausa de estabilización de 50 microsegundos, realiza la lectura. Al igual que en el panel inferior, se implementa una detección de cambios por umbral para optimizar la comunicación, publicando el nuevo valor solo si la variación supera un umbral de ruido.

Toda esta lógica se ejecuta de forma concurrente dentro de su propia tarea de FreeRTOS, tareaPotenciometros, que opera con prioridad media en el Núcleo 1 (APP_CPU), asegurando

que la lectura de las interacciones del usuario sea responsiva sin interferir con el control de los motores.

```
39 }
40
41 void SistemaPotenciometros::selectChannel(byte channel) {
42     for (int i = 0; i < 4; i++) {
43         digitalWrite(SELECT_PINS[i], bitRead(channel, i));
44     }
45 }
46
47 void SistemaPotenciometros::leerYPublicarCambios() {
48     for (int i = 0; i < NUM_CHANNELS; i++) {
49         byte canalActual = CHANNELS_TO_READ[i];
50         selectChannel(canalActual);
51         delayMicroseconds(50); // Pequeña espera para que la señal se estabilice
52         int valorActual = analogRead(SIG_PIN);
53
54         // Publicar solo si el cambio es significativo (mayor que el ruido)
55         if (abs(valorActual - ultimosValores[canalActual]) > 5) {
56
57             ultimosValores[canalActual] = valorActual;
58
59             MqttPublicacion mensaje;
60
61             // Construye el t pico: buceo/panel_superior/nombre_canal
62             String topicoCompleto = "buceo/panel_superior/" + CHANNEL_NAMES[i];
63             strncpy(mensaje.topico, topicoCompleto.c_str(), sizeof(mensaje.topico) - 1);
64             mensaje.topico[sizeof(mensaje.topico) - 1] = '\0'; // Asegurar terminaci n nula
65
66             // Convierte el valor a texto para el payload
67             snprintf(mensaje.payload, sizeof(mensaje.payload), "%d", valorActual);
68
69             mensaje.qos = 0;
70             mensaje.retenido = false;
71
72             // Env a el mensaje a la cola para que la tarea MQTT lo publique
73             if (xQueueSend(cola_publicaciones_mqtt, &mensaje, pdMS_TO_TICKS(10)) != pdTRUE) {
74                 Serial.printf(" Error: Cola MQTT llena, no se pudo encolar mensaje para canal %d\n", canalActual);
75             }
76         }
77     }
78 }
```

Figura 4-9 L gica de Lectura Multiplexada y Detecci n de Cambios

Fuente: Elaboraci n propia

4.3. IMPLEMENTACI N DEL FIRMWARE POR M DULO

La aplicaci n de los principios de dise o descritos se materializa de forma distinta en cada panel del simulador. Lejos de una soluci n monol tica, la arquitectura del firmware se ha adaptado a los requerimientos y limitaciones de hardware de cada m dulo, resultando en dos estrategias de implementaci n bien diferenciadas.

Una arquitectura distribuida para el panel inferior, donde las responsabilidades se reparten entre un ESP32 y un coprocesador ESP8266 para maximizar la fidelidad de la se al, y una arquitectura centralizada para el panel superior, donde un  nico ESP32 gestiona todas las funciones, requiriendo soluciones de firmware ingeniosas para superar las limitaciones de hardware.

Las siguientes secciones desglosarán en detalle cómo se implementó la lógica de control para cada uno de estos módulos, demostrando la flexibilidad y robustez de los pilares de diseño adoptados.

4.3.1. Panel Inferior: Arquitectura Distribuida

Para optimizar el rendimiento y la fiabilidad, el panel inferior implementa una arquitectura de control distribuida que opera con dos microcontroladores en sinergia.

El controlador principal, un ESP32, gobierna los actuadores y sensores digitales bajo el sistema operativo FreeRTOS. La gestión concurrente de su firmware se organiza en tareas con prioridades y afinidad de núcleo específicas, como se detalla en la Tabla 4-3, donde se observa que a la tarea tareaSistemaMotores se le asigna la máxima prioridad (6) y se fija su ejecución en el núcleo de aplicaciones (Core 1) para garantizar la fluidez de los actuadores.

Tabla 4-3
TAREAS CONCURRENTES (FREERTOS) PARA LA GESTIÓN DE MOTORES

TAREAS CONCURRENTES (FREERTOS) PARA LA GESTIÓN DE MOTORES				
Tarea	Prioridad	Stack bytes	Núcleo (core)	Responsabilidad principal en el contexto de los motores
Tareasistemamotores	6 (máxima)	4096	1 (core)	Actualizar los perfiles de movimiento de los motores en alta frecuencia (1 ms) para garantizar máxima fluidez.
Tareamqtt	3 (media-baja)	4096	0 (core)	Gestionar la conexión y procesar la cola de publicación, incluyendo los mensajes de estado de los motores.
Tareaestadosistema	2 (baja)	2048	0 (core)	Publicar periódicamente un reporte completo del estado de los motores para asegurar la sincronización.

Fuente: Elaboración propia

Su clase SistemaMotores utiliza la librería AccelStepper para gestionar perfiles de aceleración que emulan la inercia de una aguja analógica, implementando una máquina de estados que desenergiza las bobinas del motor en reposo para reducir el consumo y el calor. La

interconexión física entre el microcontrolador ESP32 y los cuatro drivers de los motores se detalla en la Tabla 4-4, que especifica la asignación de pines GPIO para cada una de las bobinas de los motores que simulan los manómetros del banco primario, secundario y sus respectivas MMP.

Tabla 4-4
PINOUT PARA MOTORES

TABLA DE ASIGNACIÓN DE PINES (PINOUT) DEL ESP32 PARA MOTORES					
Motor	Función	Pin in1 (driver)	Pin in2 (driver)	Pin in3 (driver)	Pin in4 (driver)
Motor 1	Manómetro Banco Primario	GPIO 4	GPIO 5	GPIO 6	GPIO 7
Motor 2	Manómetro MMP primario	GPIO 10	GPIO 11	GPIO 12	GPIO 13
Motor 3	Manómetro Banco Secundario	GPIO 14	GPIO 15	GPIO 16	GPIO 17
Motor 4	Manómetro MMP Secundario	GPIO 18	GPIO 21	GPIO 35	GPIO 36

Fuente: Elaboración propia

La interacción y el control de los motores se gestionan exclusivamente a través del protocolo MQTT, garantizando un desacoplamiento total entre el software de simulación y el hardware. Para ello, se ha definido una robusta estructura de comandos en formato JSON, como se detalla en la Tabla 4-5.

Esta estructura permite un control preciso sobre cada motor. Incluye comandos de movimiento como "p" para establecer una posición absoluta y "r" para realizar un ajuste relativo desde la posición actual. Además, se cuenta con un comando de calibración, "z", que define la posición actual del motor como el nuevo origen o cero.

Para un control dinámico más avanzado, las órdenes de movimiento pueden ir acompañadas de parámetros opcionales: "v" para ajustar la velocidad máxima del desplazamiento y "a" para modificar el perfil de aceleración en tiempo real.

Esta interfaz de comandos proporciona un control flexible y potente, fundamental para la arquitectura del simulador.

Tabla 4-5

ARQUITECTURA DE TÓPICOS MQTT PARA EL MÓDULO DE MOTORES

TABLA DE COMANDOS MQTT PARA MOTORES			
Clave json	Tipo dato	Función	Descripción de uso
"p"	Long	Posición absoluta	Mueve la aguja a un número de paso exacto del dial.
"r"	Long	Movimiento relativo	Ajusta la posición actual en una cantidad específica de pasos.
"z"	Bool	Puesta a cero	Calibra la posición del motor, definiendo la ubicación actual como el origen (0).
"v"	Float	Velocidad (opcional)	Ajusta la velocidad máxima del movimiento en pasos/segundo.
"a"	Float	Aceleración (opcional)	Ajusta la aceleración del movimiento en pasos/segundo².

Fuente: Elaboración propia

De forma concurrente, la clase SistemaValvulas asegura la lectura fiable de los sensores digitales. Para lograrlo, implementa un algoritmo de antirrebote (debounce) por software que valida un cambio de estado solo si la señal del sensor permanece estable durante 25 ms, optimizando la comunicación al publicar únicamente los cambios de estado confirmados. La arquitectura de comunicación de este módulo, como se detalla en la Tabla 4-6, utiliza tópicos MQTT específicos para reportar tanto el estado de conexión general del sistema como el estado individual (abierto/cerrado) de cada válvula.

Tabla 4-6

TÓPICOS MQTT PARA EL MÓDULO DE VÁLVULAS

TÓPICOS MQTT PARA EL MÓDULO DE VÁLVULAS				
Tipo de Tópico	Tópico	QoS	Dirección	Descripción
Estado	buceo/valvulas/sistema/estado	1	ESP32 → HMI	Publica el estado de conexión del subsistema de válvulas ("conectado")

				o "desconectado" vía LWT).
Estado	buceo/valvulas/{nombre}/estado	1	ESP32 → HMI	Publica el estado (abierto/cerrado) de una válvula específica. {nombre} es, por ejemplo, BUZO_ROJO.

Fuente: Elaboración propia

Toda esta lógica se ejecuta dentro de su propia tarea de FreeRTOS, tareaSistemaValvulas, la cual, como se muestra en la Tabla 4-7, opera con prioridad media (4) en el núcleo de aplicaciones (Core 1). Esta afinidad de núcleo garantiza que la lectura de los sensores no se vea afectada por la latencia de la red, mientras que las tareas de publicación (tareaMQTT) y sincronización periódica (tareaEstadoSistema) se gestionan en el núcleo de comunicaciones (Core 0) con prioridades más bajas.

Tabla 4-7

TAREAS CONCURRENTES (FREERTOS) PARA LA GESTIÓN DE VÁLVULAS

TAREAS CONCURRENTES (FREERTOS) PARA LA GESTIÓN DE VÁLVULAS				
Tarea	Prioridad	Stack (Bytes)	Núcleo (Core)	Responsabilidad Principal en el Contexto de las Válvulas
tareaSistemaValvulas	4 (Media)	4096	1 (Core)	Leer los sensores aplicando el filtro debounce y encolar los cambios de estado para su publicación.
tareaMQTT	3 (Media-Baja)	4096	0 (Core)	Procesar la cola de publicación, enviando los mensajes de estado de las válvulas a través de la red.
tareaEstadoSistema	2 (Baja)	2048	0 (Core)	Forzar la publicación periódica del estado de todas las válvulas para asegurar la sincronización.

Fuente: Elaboración propia

Para complementar al controlador principal, la arquitectura del panel inferior incorpora un coprocesador analógico (ESP8266), cuya única responsabilidad es la adquisición de datos de alta precisión de los potenciómetros, aislando estas mediciones sensibles del ruido electromagnético.

Toda la lógica de este módulo se encapsula en la clase LectorPotenciometros, la cual gestiona la lectura de los sensores a través de un ADC externo de 16 bits (ADS1115), aplica un filtro de suavizado exponencial para estabilizar la señal y emular la inercia de una válvula real,

y optimiza el tráfico de red mediante una detección de cambios por umbral y un heartbeat periódico para garantizar la sincronización.

La comunicación con la HMI se realiza a través de los tópicos MQTT individuales que se muestran en la Tabla 4-8, donde cada potenciómetro reporta su estado a un tópico específico, como buceo/presion/banco_principal.

Tabla 4-8
TÓPICOS MQTT PARA EL MÓDULO DE POTENCIÓMETROS

TÓPICOS MQTT PARA EL MÓDULO DE POTENCIÓMETROS				
Función Simulada	Tópico	QoS	Retenido	Dirección
Regulación Banco Primario	buceo/presion/banco_principal	0	Sí	ESP8266 → HMI
Regulación MMP Principal	buceo/presion/mmp_principal	0	Sí	ESP8266 → HMI
Regulación Banco Secundario	buceo/presion/banco_secundario	0	Sí	ESP8266 → HMI
Regulación MMP Secundaria	buceo/presion/mmp_secundario	0	Sí	ESP8266 → HMI

Fuente: Elaboración propia

Al no utilizar un sistema operativo de tiempo real, el firmware opera sobre un bucle principal no bloqueante y altamente eficiente. La secuencia lógica de este bucle, como se resume en la Tabla 4-9, prioriza en cada ciclo el mantenimiento de la conexión MQTT, seguido por la actualización de las lecturas de los sensores y, finalmente, una comprobación de tiempo para el envío del heartbeat de sincronización.

Tabla 4-9
SECUENCIA LÓGICA DEL BUCLE PRINCIPAL (ESP8266)

SECUENCIA LÓGICA DEL BUCLE PRINCIPAL (ESP8266)			
Paso	Función Invocada	Prioridad Lógica	Responsabilidad Principal
1	comunicador_mqtt.bucle()	Alta	Mantener la conexión Wi-Fi y el ciclo de vida del cliente MQTT.

2	lector_potenciometros.actualizar()	Media	Leer los 4 canales del ADC, aplicar filtro y marcar cambios significativos.
3	Comprobación de millis()	Baja	Verificar si han transcurrido los 30 segundos para el envío del "heartbeat".

Fuente: Elaboración propia

4.3.2. Panel Superior: Arquitectura Centralizada

A diferencia del panel inferior, el panel superior emplea una arquitectura centralizada en un único ESP32, resolviendo sus desafíos de hardware mediante soluciones de firmware ingeniosas y específicas para la gestión de recursos.

Para superar la escasez de pines GPIO, el control de los motores se delega a expansores de puertos I2C (PCF8574); esta decisión requirió el desarrollo de una clase driver especializada, AccelStepper_PCF8574, que traduce los comandos de paso en transacciones I2C, manteniendo una interfaz compatible con la librería estándar.

La Tabla 4-10 resume la configuración de este módulo, incluyendo las direcciones I2C de los expansores y la asignación de tareas en FreeRTOS.

Tabla 4-10
TABLA DE CONFIGURACIÓN DEL MÓDULO

TABLA DE CONFIGURACIÓN DEL MÓDULO			
Componente / tarea	Tipo	Detalle técnico	Descripción / responsabilidad
Motores 1-4	Comunicación	Tópico Comando: buceo/ comandos/ consolasuperior/ motores/{id}	Recibe órdenes de movimiento (posición, velocidad) desde la HMI.
	Comunicación	Tópico Estado: buceo/ consolasuperior/ motores/{id}/estado	Publica el estado actual del motor (posición, movimiento).
Tareamotores	Tarea RTOS	Prioridad: 3 (Alta), Núcleo: 1 (APP), Frecuencia: 1000 Hz	Ejecutar el método run() de cada motor para un movimiento suave.
Bus I2C	Hardware	SDA: GPIO 21, SCL: GPIO 22	Bus de comunicación para controlar los expansores de pines.
Expansor 1	Hardware	Dirección I2C: 0x20	Controla los Motores 1 y 2 (Profundímetros 1 y 2).

Expansor 2	Hardware	Dirección I2C: 0x24	Controla los Motores 3 y 4 (Profundímetro 3 y Manómetro MMP).
------------	----------	---------------------	---

Fuente: Elaboración propia

Para evitar el conocido conflicto de hardware entre el periférico Wi-Fi y el ADC2 del ESP32, la clase SistemaPotenciometros gestiona la lectura de siete potenciómetros a través de un multiplexor analógico (CD74HC4067).

El firmware selecciona secuencialmente cada canal y dirige su señal a través de un único y fiable pin de entrada del ADC1, que no se ve afectado por la actividad de la red inalámbrica. Como en los otros módulos, se utiliza una detección de cambios por umbral para optimizar la comunicación. La configuración de este sistema se presenta en la Tabla 4-11.

Tabla 4-11
Módulo de Monitorización Analógica (Panel Superior)

Módulo de Monitorización Analógica (Panel Superior)			
Componente Concepto	Tipo	Detalle Técnico	Descripción / Responsabilidad
Desafío de Hardware	ESP32	Conflicto Wi-Fi vs. ADC2	La activación del periférico Wi-Fi (requisito del proyecto) genera interferencia y vuelve inutilizables los pines del ADC2 para lecturas analógicas fiables.
Sensores a Leer	Hardware	7 potenciómetros	Simulan las válvulas de aguja del sistema neumofatometro.
Solución Implementada	Hardware Firmware	Multiplexor (MUX) CD74HC4067 + Clase SistemaPotenciometros	Se utiliza un multiplexor para dirigir las 7 señales analógicas, una por una, hacia un único pin fiable del ADC1.
Pin de Lectura (Señal)	Hardware (Pin)	GPIO 34 (ADC1)	Es el pin de entrada del ADC1 (libre de conflictos) que recibe la señal analógica del canal seleccionado por el MUX.
Pines de Control (Selección)	Hardware (Pines)	GPIO 23, 19, 27, 26	Cuatro pines digitales (S0-S3) que envían la dirección binaria (0-15) al MUX para seleccionar qué canal se conecta al pin de señal.

Optimización de Red	Firmware	Detección de cambios por umbral	El firmware solo encola un mensaje MQTT si la nueva lectura difiere de la anterior en un valor mayor al ruido, optimizando la comunicación.
---------------------	----------	---------------------------------	---

Fuente: Elaboración propia

La Tabla 4-12 detalla la configuración del módulo de monitorización analógica, responsable de gestionar la lectura de los siete potenciómetros del panel superior. Se especifican los componentes de hardware, los tópicos MQTT para la comunicación y las tareas de FreeRTOS que organizan su funcionamiento para evitar conflictos con el periférico Wi-Fi.

Tabla 4-12
MÓDULO DE MONITORIZACIÓN ANALÓGICA

MÓDULO DE MONITORIZACIÓN ANALÓGICA			
Componente / Tarea	Tipo	Detalle Técnico	Descripción / Responsabilidad
Potenciómetros (1-7)	Comunicación	Tópico Estado: buceo/ panel_superior/ {nombre canal}	Publica el valor numérico (0-4095) de un canal cuando se detecta un cambio.
tareaPotenciometros	Tarea RTOS	Prioridad: 2 (Media), Núcleo: 1 (APP), Frecuencia: 50 Hz	Tarea de prioridad media para leer los 7 canales del multiplexor y gestionar la publicación.
Multiplexor (Control)	Hardware	S0-S3: GPIO 23, 19, 27, 26	Pines de control digital que envían la dirección binaria del canal a leer.
Multiplexor (Señal)	Hardware	SIG: GPIO 34	Pin de entrada del ADC1 que recibe la señal analógica del canal seleccionado.
tareaMQTT	Tarea RTOS	Prioridad: 1 (Baja), Núcleo: 0 (PRO), Frecuencia: 100 Hz	Tarea de comunicaciones que encola y envía los mensajes MQTT generados por este módulo.

Fuente: Elaboración propia

CAPÍTULO 5: ARQUITECTURA DE SOFTWARE Y HMI

5. ARQUITECTURA DE SOFTWARE DEL SIMULADOR

La efectividad de un simulador reside en una arquitectura de software robusta, capaz de generar escenarios complejos, gestionar la comunicación en tiempo real y presentar una interfaz intuitiva y reactiva. Este capítulo detalla la arquitectura integral del sistema, un diseño modular y multihilo que garantiza la mantenibilidad, escalabilidad y un rendimiento óptimo. Se fundamenta en la simbiosis tecnológica de Python por su agilidad, y el framework PySide6 (Qt) por su motor de renderizado de alto rendimiento.

5.1. ARQUITECTURA GENERAL Y FLUJO DE COMUNICACIÓN

La arquitectura general del sistema se fundamenta en un diseño multihilo para asegurar que la interfaz gráfica (HMI) permanezca fluida y no sea bloqueada por las operaciones de red. Para lograrlo, el sistema opera sobre dos hilos de ejecución principales: un Hilo Principal (UI Thread), dedicado exclusivamente a la gestión del bucle de eventos de la interfaz de usuario, (renderizar la VentanaPrincipal, paneles, widgets y procesar interacciones); y un Hilo Secundario (MQTT Thread), que maneja toda la comunicación de red con el hardware (Arduino) a través del protocolo MQTT. Este aislamiento es crucial para que la latencia o las desconexiones de la red no afecten el rendimiento y la capacidad de respuesta de la HMI.

La comunicación y la estructura del sistema se apoyan en patrones de diseño de software probados para crear un sistema cohesivo y de bajo acoplamiento. Se utiliza el patrón Observer (Observador), implementado a través del mecanismo de señales y slots de PySide6, que permite a la HMI reaccionar asincrónicamente a los cambios de estado. La clase ModeloDeEstadoConsola actúa como un Mediator (Mediador) que centraliza la lógica de negocio, y tanto el modelo como el gestor de comunicación exponen una API simplificada bajo el patrón Facade (Fachada).

El patrón de Dependency Injection (Inyección de Dependencias) es fundamental en la inicialización. El punto de entrada de la aplicación (main.py) actúa como el orquestador principal de esta arquitectura, como se evidencia en la importación de sus componentes clave y se muestra en la Figura 5-1. Este script es responsable de la secuencia de inicialización: primero instancia el GestorMQTT (capa de comunicación), seguido por el ModeloDeEstadoConsola (el "cerebro" o capa lógica), y finalmente crea la VentanaPrincipal (la HMI). Es en este último paso donde se "inyecta" el modelo de estado en la ventana, conectando eficazmente la lógica del sistema con la interfaz de usuario.

```
46 import sys # Módulo para interactuar con el intérprete de Python y manejo de la aplicación
47 import logging # Sistema de logging para debugging y monitoreo del sistema
48 from PySide6.QtWidgets import QApplication # Clase principal para aplicaciones Qt
49
50 # ARQUITECTURA MODULARIZADA
51 # componente clave de la arquitectura
52 from comunicacion.gestor_mqtt.gestor_principal import GestorMQTT # Gestor principal de comunicación MQTT
53 from comunicacion.modelo_estado.modelo_principal import ModeloDeEstadoConsola # Modelo central de estado
54 from ventana_principal import VentanaPrincipal # Ventana principal que coordina todos los paneles
55
```

Figura 5-1 Importación de Módulos Centrales de la Arquitectura

Fuente: Elaboración propia

El script main.py evidencia la arquitectura modular, importando las tres clases que definen el sistema: GestorMQTT (capa de comunicación), ModeloDeEstadoConsola (capa de lógica y estado) y VentanaPrincipal (capa de presentación HMI).

Finalmente, el protocolo MQTT establece la comunicación bidireccional mediante el patrón Publisher/Subscriber (Publicador/Suscriptor), donde el hardware publica su estado en tópicos específicos, y el software se suscribe a ellos, e inversamente, el software publica comandos que el hardware recibe y ejecuta.

El Diagrama 5-1 muestra la vista general de la arquitectura del sistema, delineando los componentes principales y sus interacciones.

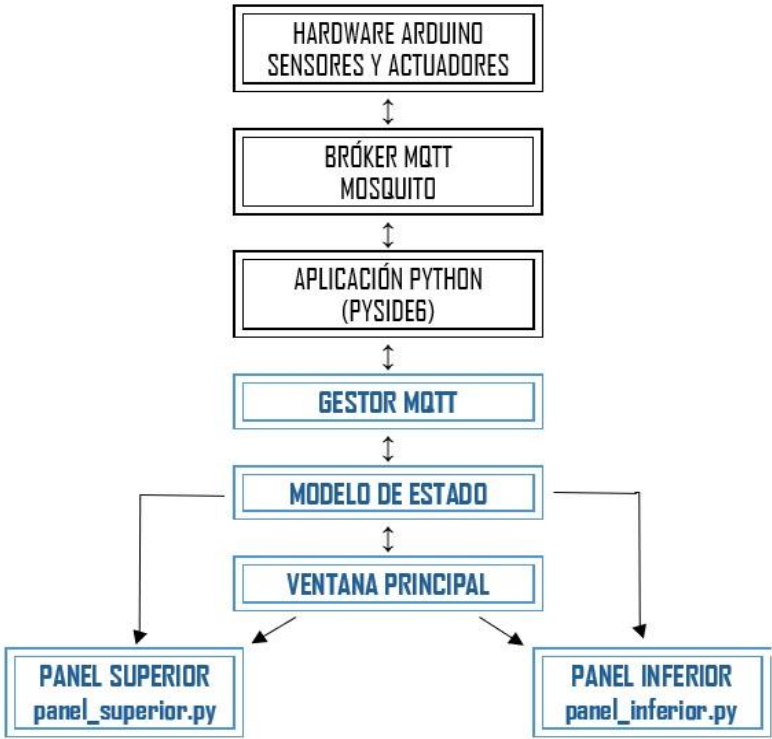


Diagrama 5-1 Arquitectura General Del Sistema

Fuente: Elaboración propia

5.2. SECUENCIA DE INICIALIZACIÓN DEL SISTEMA

El arranque de la aplicación, orquestado por el script main.py, sigue una secuencia ordenada y robusta para garantizar que todas las dependencias se resuelvan correctamente antes de la operación.

El proceso comienza con una fase de configuración y validación, donde se inicializa un sistema de logging detallado y se verifican las dependencias críticas del entorno, como la versión de Python y las bibliotecas PySide6 y paho-mqtt.

A continuación, se procede con la creación del GestorMQTT, componente que de inmediato genera y administra su propio hilo de comunicación para interactuar con el broker MQTT, aislando así las operaciones de red. Inmediatamente después, se instancia el modelo de estado, ModeloDeEstadoConsola (el "cerebro" del sistema), inyectándole el GestorMQTT como dependencia para que pueda enviar comandos al hardware.

Con la lógica ya establecida, se realiza la creación de la VentanaPrincipal (HMI), la cual recibe la instancia del ModeloDeEstadoConsola para que la vista pueda reflejar y modificar el estado del sistema. Esta secuencia de creación e inyección de dependencias se ilustra visualmente en la Figura 5-2.

```
200 > def crear_modelo_estado(gestor_mqtt):...
253
254
255 > def crear_ventana_principal(modelo_consola):...
308
309
310 > def establecer_referencias_cruzadas(modelo_consola, ventana):...
353
354
355 > def activar_comunicacion_mqtt(gestor_mqtt):...
399
400
401 > def configurar_gestion_cierre(app, gestor_mqtt):...
437
438
439 > def main():...
```

Figura 5-2 Flujo de Inicialización del Software
Fuente: Elaboración propia

Por ende, El Flujo de Inicialización del Software Se ilustra la secuencia de arranque orquestada por `main.py`, mostrando la instanciación del `GestorMQTT` (y su hilo), seguido por el `ModeloDeEstadoConsola` y la `VentanaPrincipal`, y destacando la inyección de dependencias entre los componentes.

Antes de lanzar la interfaz, se asegura un cierre limpio configurando una rutina de apagado seguro (`aboutToQuit`) para el hilo MQTT, y se activa la conexión de red, iniciando todo el flujo de datos.

Finalmente, se muestra la ventana y se da inicio al bucle de eventos, momento en el cual el sistema queda en un estado completamente operativo y listo para la interacción del usuario.

5.3. ARQUITECTURA DETALLADA POR CAPAS

Habiendo delineado la arquitectura general y la secuencia de inicialización del sistema, este apartado se sumerge en la estructura granular del software. El diseño se ha concebido como un entramado de capas lógicas bien definidas, donde cada una, encapsulada en un paquete de Python, ostenta responsabilidades claras y exclusivas.

Esta modularidad no solo facilita la comprensión y el mantenimiento del código, sino que también promueve la escalabilidad y la robustez de un sistema complejo de simulación. A continuación, se desglosa la función y los componentes clave de cada una de estas capas, revelando los patrones de diseño y las estrategias de implementación que rigen su funcionamiento.

5.3.1. Capa de Comunicación (Paquete `gestor_mqtt`)

Este módulo es el puente vital que abstrae y gestiona toda la interacción con el hardware físico del simulador. Su diseño multihilo y su resiliencia son fundamentales para garantizar la estabilidad operativa, incluso ante fluctuaciones en la red o desconexiones temporales. El paquete incluye el `GestorMQTT` (`gestor_principal.py`), que actúa como la Fachada del sistema de comunicación, ofreciendo una API simplificada (`conectar`, `publicar`, etc.) al resto de la aplicación y siendo responsable de inicializar y gestionar el ciclo de vida del `QThread` donde opera el `TrabajadorMQTT`.

Este `TrabajadorMQTT` (`trabajador_mqtt.py`) es el núcleo operativo que se ejecuta íntegramente en el hilo secundario, interactuando directamente con la biblioteca `paho-mqtt`, gestionando los callbacks para eventos clave como la conexión, desconexión y recepción de

mensajes, y delegando responsabilidades específicas a manejadores especializados para mayor robustez.

En la Figura 5-3, se observa el método `_ejecutar_intento_conexion` del `TrabajadorMQTT`. Este fragmento es crucial, ya que muestra la llamada a `cliente_mqtt.connect()` para establecer la conexión al broker y, fundamentalmente, la llamada a `cliente_mqtt.loop_start()`. Esta última función inicia el bucle de red de la biblioteca `paho-mqtt` en un hilo secundario propio, una acción que garantiza que la HMI (interfaz principal) no se bloquee por las operaciones de red.

```
121 def _ejecutar_intento_conexion(self): 2 usages
122     """
123     Ejecuta un intento de conexión al broker.
124     """
125     if self.conectado:
126         logger.debug("Ya conectado, ignorando intento de conexión")
127         return
128     try:
129         self.log_generado.emit(
130             f"Conectando a {self._config.mqtt_broker}:{self._config.mqtt_puerto}..."
131         )
132         self.cliente_mqtt.connect(
133             self._config.mqtt_broker,
134             self._config.mqtt_puerto,
135             self._config.timeout_conexion_segundos
136         )
137         # Iniciar el loop de paho-mqtt en su propio hilo
138         self.cliente_mqtt.loop_start()
139     except Exception as e:
140         error_msg = f"Error al conectar: {e}"
141         self.log_generado.emit(error_msg)
142         logger.error(f"Error de conexión: {e}")
143
```

Figura 5-3 Implementación del Hilo de Conexión MQTT
Fuente: Elaboración propia

Complementariamente, el `ManejadorReconexion` (`manejador_reconexion.py`) implementa una sofisticada estrategia de backoff exponencial que, ante una desconexión detectada, gestiona de forma autónoma los intentos de reconexión, duplicando progresivamente el intervalo de reintento entre cada intento para evitar la saturación de la red.

Por su parte, el `ManejadorSuscripciones` (`manejador_suscripciones.py`) tiene la única responsabilidad de mantener un registro persistente de todas las suscripciones a tópicos MQTT, asegurando que, tras una reconexión exitosa, todas las suscripciones se restauren automáticamente, garantizando la continuidad en la recepción de datos.

Finalmente, la `ConfiguracionMQTT` (`configuracion_mqtt.py`), definida como un `dataclass`, centraliza todos los parámetros de red esenciales (dirección del broker, puertos,

credenciales, Calidad de Servicio (QoS) por defecto), lo que facilita la modificación de la configuración del sistema sin necesidad de alterar el código fuente.

5.3.2. Capa de Lógica y Estado (Paquete modelo_estado)

Este paquete representa el cerebro del simulador, implementando una adaptación del paradigma Modelo-Vista-Controlador (MVC), donde la clase `ModeloDeEstadoConsola` encarna eficazmente tanto el Modelo (gestionando el estado global) como el Controlador (orquestrando la lógica de negocio). El núcleo de esta capa es el `ModeloDeEstadoConsola` (`modelo_principal.py`), que actúa como orquestador central y Mediador. Es el garante de la "fuente única de verdad" del estado completo del sistema, procesa las interacciones del usuario y coordina la operación de todos los demás componentes de esta capa.

La entrada para la telemetría se gestiona a través del `ProcesadorMQTT` (`procesador_mqtt.py`), que recibe los mensajes MQTT crudos del gestor de comunicación, se encarga de decodificar sus payloads JSON y traducirlos a las estructuras de datos fuertemente tipadas del sistema.

Para la interacción saliente, el `ControladorComandos` (`controlador_comandos_mejorado.py`) implementa el patrón Command, ofreciendo una API semántica de alto nivel para enviar órdenes al hardware (ej: `mover_motor_superior`), encapsulando la creación y serialización de los comandos JSON y enriqueciéndolos con metadatos como timestamps para facilitar la trazabilidad.

La comunicación interna y la notificación a otras capas se facilita con el `ManejadorSenales` (`manejador_senales.py`), que funciona como un Bus de Eventos (Event Bus), centralizando todas las señales del modelo (basadas en el sistema de señales y slots de la interfaz de usuario) y permitiendo que la capa de presentación se suscriba a cambios de estado específicos (ej: `presion_banco_1_actualizada`) de forma completamente desacoplada.

Para aportar realismo visual, el `SimuladorMovimiento` (`simulador_movimiento.py`) interviene en actuadores que no reportan su posición en tiempo real, interpolando su movimiento en la HMI utilizando un temporizador, lo que genera una animación fluida en lugar de cambios de estado instantáneos.

Finalmente, los módulos `estructuras_datos.py` y `configuracion_sistema.py` son esenciales para el soporte: el primero define el estado de los componentes (`EstadoMotor`, `EstadoSensorPresion`) mediante dataclasses, asegurando consistencia y claridad, mientras que el segundo centraliza parámetros operativos críticos como velocidades de motor y tópicos MQTT, permitiendo una gestión flexible.

Para ilustrar cómo esta capa opera en la práctica, analizamos los dos métodos específicos de `ModeloDeEstadoConsola`, como se muestra en el código de la figura 5-4.

```
366     def suscribirse_a_topicos_de_estado(self): 1 usage
367         """
368         Se suscribe a todos los tópicos necesarios para el estado del sistema.
369         """
370         topicos = [
371             "buceo/valvulas/#",
372             "buceo/presion/#",
373             "buceo/motores/#",
374             "buceo/panel_superior/#",
375             "buceo/consolasuperior/motores/#",
376             "buceo/consolasuperior/potenciometros/#"
377         ]
378
379         for topico in topicos:
380             self._gestor_mqtt.suscribir(topico, qos=1)
381
382         # === MÉTODOS DE CONTROL DE MOTORES INFERIORES ===
383
384     def mover_motor(self, motor_id: int, p: int = None, r: int = None, 2 usages
385                     v: float = -1, a: float = -1, s: bool = False, z: bool = False):
```

Figura 5-4 Métodos de Suscripción y Control de Motores

Fuente: Elaboración propia

El primer método, `def suscribirse_a_topicos_de_estado(self)`, es fundamental para la recepción de telemetría. Es la función que le indica al gestor MQTT qué datos debe escuchar y se ejecuta automáticamente cuando el sistema se conecta exitosamente al broker MQTT. Su mecanismo es simple pero efectivo: recorre una lista predefinida de "tópicos" (ej. "buceo/valvulas/#", "buceo/presion/#"). El uso del comodín # es una característica clave de MQTT que significa "suscribirse a todo lo que esté bajo esta ruta". Por ejemplo, al suscribirse a "buceo/valvulas/#", el sistema recibirá mensajes de "buceo/valvulas/rojo", "buceo/valvulas/verde", y cualquier otra sub-ruta que exista. El propósito de esta suscripción general es asegurar que el `ProcesadorMQTT` reciba toda la información de estado proveniente del hardware (válvulas, motores, presiones), permitiendo que el `ModeloDeEstadoConsola` mantenga la "fuente única de verdad" del sistema constantemente actualizada.

El segundo método, `def mover_motor(self, motor_id: int, p: int = None, ...)` es un ejemplo perfecto de la interacción saliente y el uso del patrón Command. Esta función provee una API simple y limpia para que otras partes del programa, como la interfaz de usuario, puedan ordenar el movimiento de un motor. Esto representa un alto nivel de abstracción, ya que la interfaz no necesita saber cómo construir un mensaje JSON complejo; simplemente llama a esta

función con parámetros claros (por ejemplo, `mover_motor(motor_id=1, p=5000)` para mover el motor 1 a la posición 5000, o `s=True` para enviar una orden de parada). Este método implementa el principio de delegación: no envía el comando MQTT directamente. Su única responsabilidad es tomar esos parámetros simples y delegar la tarea de comunicación al `self._controlador_comandos`. Es, por lo tanto, el `ControladorComandos` (mencionado en la descripción de la capa) el que realmente hace el trabajo de crear el payload JSON, añadir metadatos como timestamps y publicarlo en el tópico MQTT correcto. Este diseño mantiene al `ModeloDeEstadoConsola` enfocado puramente en la lógica de estado, y no en los detalles del protocolo de comunicación.

5.4. DISEÑO DE LA INTERFAZ HOMBRE-MÁQUINA (HMI)

La Interfaz Hombre-Máquina (HMI) se ha diseñado bajo una arquitectura modular y declarativa, promoviendo activamente la reutilización de componentes y una estricta separación entre el diseño visual y la lógica subyacente.

5.4.1. Orquestación General (VentanaPrincipal)

La clase `VentanaPrincipal` trasciende el rol de un simple contenedor visual para convertirse en un hub de coordinación que ensambla la interfaz, delegando tareas específicas a módulos especializados y manteniendo su propio código limpio y enfocado en la orquestación general. Para ello, cuenta con `ConfiguracionVentana`, un módulo clave que utiliza dataclasses para definir de forma completamente declarativa toda la estructura de la HMI: desde la disposición del layout principal y el posicionamiento de cada panel, hasta las complejas conexiones visuales que deben trazarse entre ellos.

Adicionalmente, el `CoordinadorPaneles` implementa los patrones Mediator y Registry, encargándose de instanciar los paneles específicos (como `PanelInferior` y `PanelSuperior`), inyectándoles la referencia al modelo de estado como dependencia fundamental.

Este coordinador también proporciona una API (`get_widget`) para que cualquier componente pueda acceder a otro por su nombre, sin acoplamiento directo. Finalmente, el `DibujanteVisual` tiene la responsabilidad de renderizar los elementos visuales que trascienden los límites de un único panel, como las tuberías de conexión entre diferentes secciones, utilizando la información de `ConfiguracionVentana` y el `CoordinadorPaneles` para obtener las coordenadas de los widgets de origen y destino, y dibujando las líneas directamente sobre el lienzo de la ventana principal.

Esta lógica se implementa directamente en el método `dibujar_conexiones_interpanel`, como se observa en la Figura 5-5. Este método actúa como el punto de entrada principal para el renderizado global, siendo llamado típicamente por el evento de pintado de la `VentanaPrincipal`. Su función demuestra perfectamente la orquestación descrita: primero, consulta a `self._config` (la instancia de `ConfiguracionVentana`) para obtener la lista declarativa de todas las conexiones que deben ser dibujadas (`obtener_configuraciones_conexiones()`); luego, prepara el `QPainter` con propiedades globales, como `Antialiasing` para suavizar los bordes de las líneas. Posteriormente, itera sobre cada conexión y, aplicando el principio de responsabilidad única, delega el trabajo complejo de dibujar cada línea a un método privado (`_dibujar_conexion_individual`). De manera crucial, a este método privado le pasa las dos dependencias que necesita para funcionar, tal como se describió: el pintor (lienzo) y el coordinador_paneles. Es precisamente este coordinador_paneles el que se usará internamente para resolver los nombres de los widgets (ej. "panel_superior", "widget_valvula_1") y obtener sus coordenadas exactas en la ventana. Esto permite al `DibujanteVisual` trazar la línea en el lienzo principal sin tener conocimiento previo de la disposición interna de cada panel, logrando un desacoplamiento total.

```
221 def dibujar_conexiones_interpanel(self, pintor: QPainter, coordinador_paneles): 1 usage
222 > """Dibuja todas las conexiones visuales entre paneles..."""
232 pintor.setRenderHint(QPainter.Antialiasing, on=True) # Activar suavizado
233 # Obtener configuraciones de conexiones
234 conexiones = self._config.obtener_configuraciones_conexiones()
235 # Inicializar estadísticas de renderizado
236 conexiones_exitosas = 0
237 total_conexiones = len(conexiones)
238 print(f"[DIBUJANTE] Iniciando renderizado de {total_conexiones} conexiones")
239 # Procesar cada conexión configurada
240 for i, config_conexion in enumerate(conexiones):
241     try:
242         if self._dibujar_conexion_individual(pintor, config_conexion, coordinador_paneles):
243             conexiones_exitosas += 1
244         else:
245             self._stats_renderizado['errores_renderizado'] += 1
246
247     except Exception as e:
248         print(f"[DIBUJANTE] Error dibujando conexión {i}: {e}")
249         self._stats_renderizado['errores_renderizado'] += 1
250 # Actualizar estadísticas
251 self._stats_renderizado['conexiones_dibujadas'] = conexiones_exitosas
252 self._stats_renderizado['ultimo_renderizado'] = True if conexiones_exitosas > 0 else False
253 print(f"[DIBUJANTE] Renderizado completado: {conexiones_exitosas}/{total_conexiones} conexiones")
```

Figura 5-5 Método principal de renderizado del dibujante visual

Fuente: Elaboración propia

5.4.2. Componentes Visuales Reutilizables

Para maximizar la cohesión del código y minimizar la duplicación, todos los instrumentos visuales clave del simulador heredan de una clase base común, BaseComponenteVisual. Esta clase centraliza funcionalidades esenciales como la gestión del escalado dinámico, la configuración de paletas de colores y optimizaciones de rendimiento, asegurando un comportamiento consistente y eficiente en toda la interfaz.

Uno de los componentes derivados más importantes es el Manómetro (ManómetroOptimizado), que emula un medidor analógico de presión o profundidad. Su característica más innovadora es un sistema de indicadores direccionales (▲/▼) que se activan automáticamente para mostrar la tendencia del cambio en el valor medido. Esta funcionalidad es crucial para el entrenamiento en la gestión de velocidades de ascenso y descenso, un factor directamente relacionado con la prevención de la enfermedad descompresiva como se muestra en la Figura 5-6.

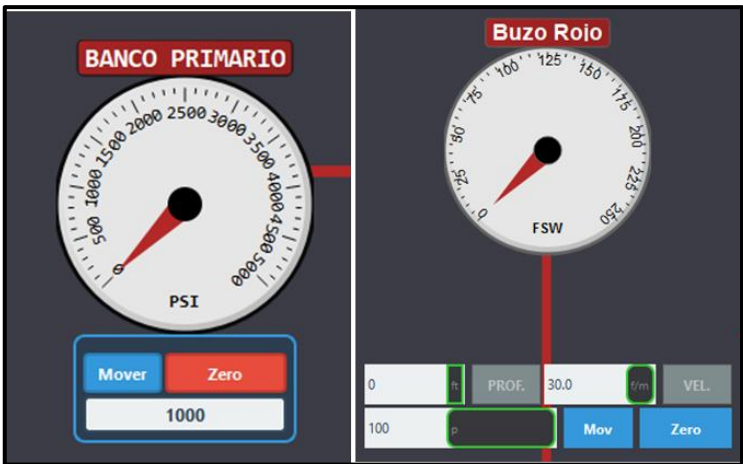


Figura 5-6 Ejemplos del Componente Manómetro Optimizado

Fuente: Elaboración propia

Otro componente clave de la interfaz es la Válvula Rotatoria (ValvulaRotatoria), diseñada para simular el comportamiento de una válvula de control de flujo mecánica, como se muestra en la Figura 5-7. A diferencia de un interruptor instantáneo, su estado se representa mediante un brazo giratorio que emula el accionamiento físico. Para lograr un movimiento realista, se implementó una animación de propiedad (QPropertyAnimation) con una curva de aceleración específica (QEasingCurve.OutBounce). Para eliminar cualquier ambigüedad, la válvula incluye un indicador de flujo que cambia de color y muestra un texto descriptivo AIRE ON, AIRE OFF o FLUJO PARCIAL correspondiendo precisamente con el estado actual del suministro de aire.

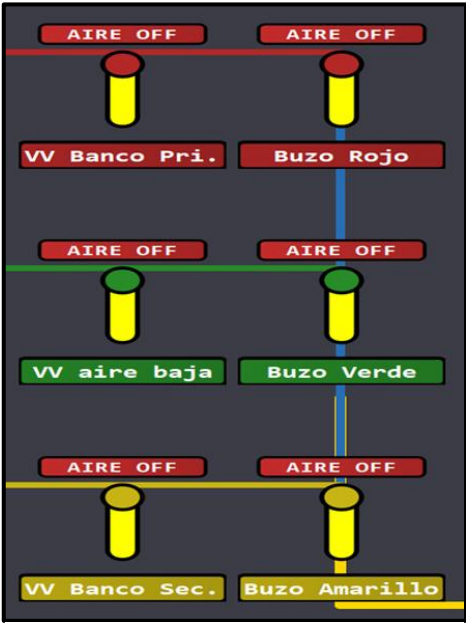


Figura 5-7 Válvula Rotatoria Simulada
Fuente: Elaboración propia

Finalmente, para la visualización de datos numéricos precisos, se desarrolló el Indicador Digital (IndicadorPresion), como se muestra en la Figura 5-8. Este componente ofrece lecturas de parámetros críticos, como la Presión Mínima Manométrica (MMP), en un formato compacto y de fácil lectura.

Su característica de diseño más notable es la doble funcionalidad de su texto superior. Por defecto, esta área muestra un dato operativo, como el porcentaje de apertura de una válvula. Sin embargo, puede ser anulada dinámicamente por el sistema para desplegar mensajes de alarma críticos (ej: "FALLA", "ALERTA"). Esta flexibilidad resulta vital para la simulación efectiva de escenarios de emergencia, garantizando que las advertencias más importantes tengan la máxima prioridad visual.

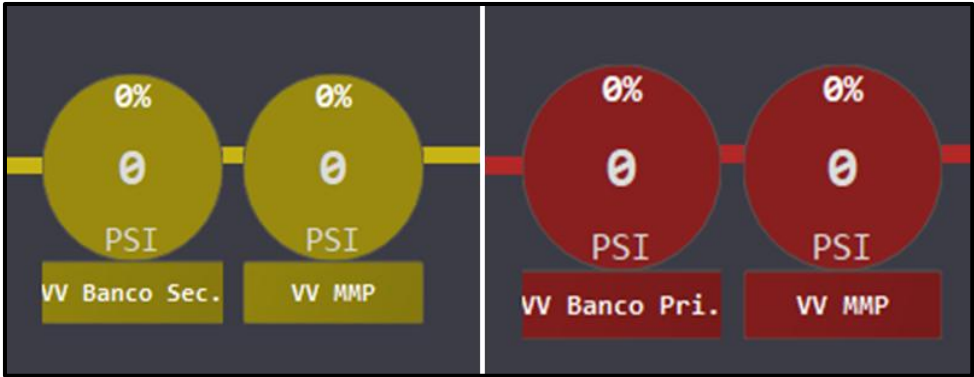


Figura 5-8 Componente Indicador Digital
Fuente: Elaboración propia

5.4.3. Control Panel Inferior

La interfaz del simulador está compuesta por dos paneles principales: Superior e Inferior. Ambos adoptan una arquitectura declarativa que asegura consistencia y modularidad en su diseño. La estructura visual de cada panel se define a través de diccionarios de configuración (CONFIG_PANEL_SUPERIOR y CONFIG_PANEL_INFERIOR), logrando un desacoplamiento efectivo entre el diseño de la interfaz y su lógica de control subyacente. Para garantizar una clara separación de responsabilidades, la lógica operativa específica de cada panel es administrada por un coordinador dedicado. Las figuras 5-9 y 5-10 ilustran la disposición y los componentes de los paneles Inferior y Superior, respectivamente.

El Panel Inferior, centrado en la gestión de suministro y presión, constituye el núcleo de la HMI para el soporte vital. Su lógica, implementada en la clase LogicaGestorComandos, introduce una optimización clave: una cola de comandos con throttling. Esta técnica previene la saturación de la red MQTT durante interacciones rápidas del usuario (p. ej., al ajustar una perilla de regulación), ya que agrupa las acciones y envía solo el comando más reciente para cada actuador a un ritmo controlado (ej. cada 100 ms), garantizando una operación fluida sin sobrecargar la red.

Complementariamente, la simulación de la dinámica de los bancos de aire se gestiona mediante la clase BancoDePresion. Este módulo encapsula una máquina de estados finitos con los estados REPOSO, AUMENTANDO, DESCENDIENDO y DESCENSO_AUTOMATICO que no solo modela los cambios de presión, sino que también incorpora un consumo de aire constante para representar la respiración de los buzos, logrando así un comportamiento de alta fidelidad.

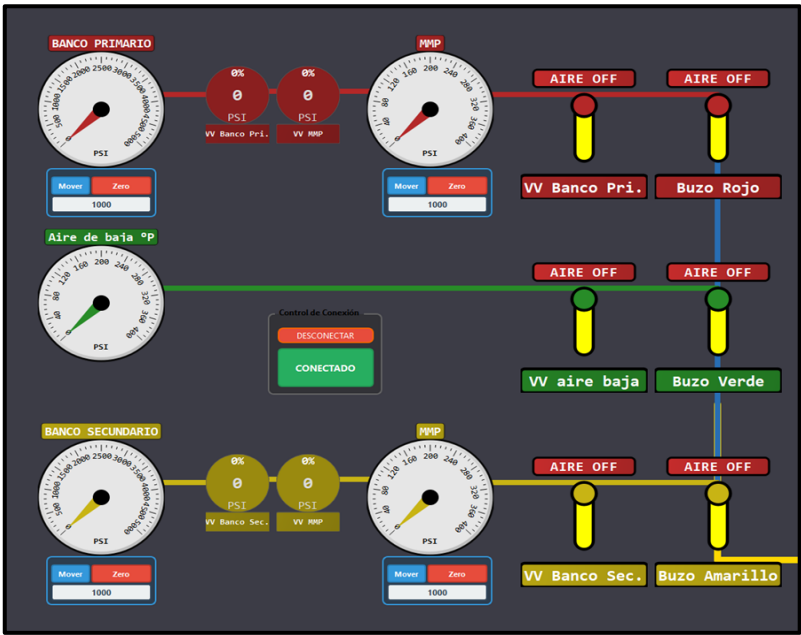


Figura 5-9 Interfaz Humano-Máquina (HMI) del Simulador Panel

Fuente: Elaboración propia

5.4.4. Control Panel Superior

El Panel Superior se enfoca en la visualización de la profundidad de cada buzo y la telemetría del sistema neumofatómetro, proporcionando los controles necesarios para simular perfiles de inmersión completos como se muestra en la Figura 5-10.

La característica más destacada de su lógica, implementada en la clase LogicaControlMMP, es la simulación del sistema repetidor de MMP (Presión Mínima Manométrica) con lógica cruzada, un comportamiento característico de las consolas de buceo reales. Este sistema monitorea activamente el estado de las válvulas de selección de banco (primario o secundario) del panel inferior para determinar qué fuente de presión MMP debe replicarse.

El valor seleccionado se actualiza de forma perfectamente sincronizada en tres componentes distintos: un indicador digital, un manómetro analógico y el motor físico correspondiente en el hardware. Esta implementación replica con precisión las particularidades operativas de una consola real, ofreciendo un escenario de entrenamiento de máxima fidelidad.

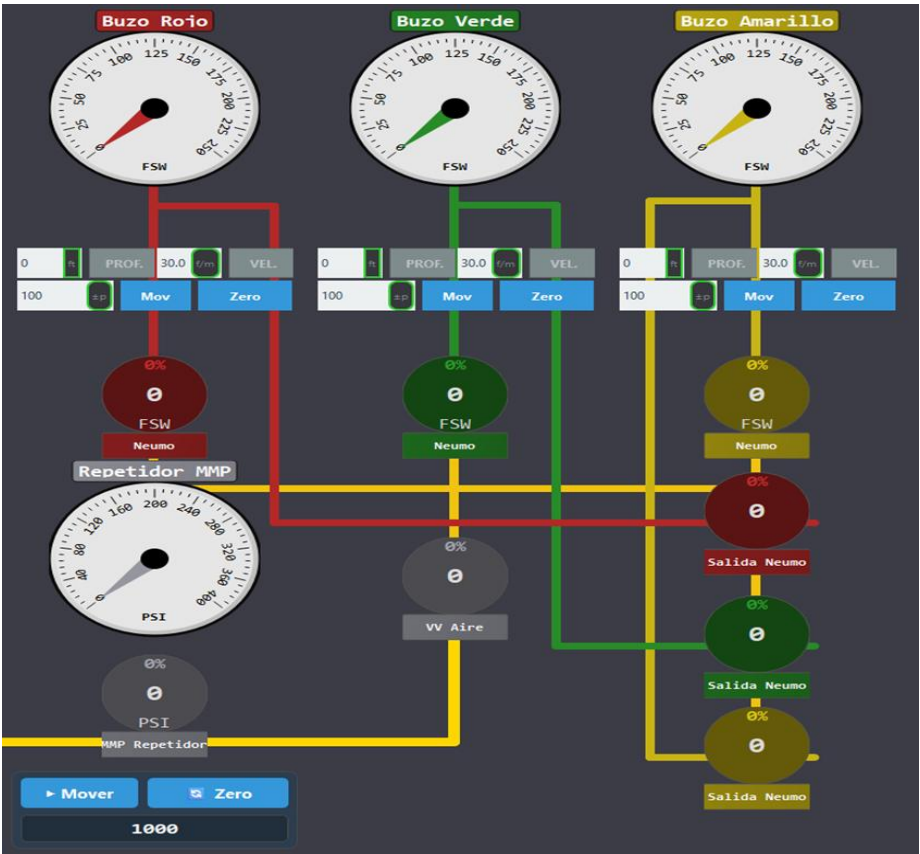


Figura 5-10 Panel Superior de Monitoreo de Buzos y Telemetría

Fuente: Elaboración propia

CONCLUSIONES Y RECOMENDACIONES

CONCLUSIONES

Al culminar las fases de diseño, implementación y validación del "Simulador Electrónico de Consola de Buceo", se ha alcanzado con éxito el objetivo general de desarrollar un sistema mecatrónico para el entrenamiento en seguridad operacional. El proyecto ha entregado un prototipo funcional que, tras ser sometido a pruebas con instructores y buzos profesionales, ha demostrado su aplicabilidad en un entorno relevante, logrando el Nivel de Madurez Tecnológica (TRL) 5 propuesto. La herramienta se ha validado como una plataforma robusta para la formación en procedimientos operativos, con un alto potencial para la mitigación de riesgos críticos en el buceo industrial. No obstante, se concluye que, para abordar escenarios avanzados como la descompresión en superficie, sería necesaria la integración de módulos adicionales, como un simulador de cámara hiperbárica.

El cumplimiento de los objetivos específicos se detalla a continuación.

1. Se materializó un prototipo físico que reproduce la interacción real del operador. Mediante un flujo de trabajo que integró diseño CAD y manufactura aditiva con material PETG, se construyó una réplica ergonómica y funcional de una consola de buceo. La plataforma resultante no solo replica la disposición visual de los controles, sino que también emula la respuesta táctil de su operación, sentando las bases para un entrenamiento inmersivo que fomenta la memoria muscular y procedimental, un factor clave en la respuesta a emergencias.
2. Se validó una arquitectura de hardware distribuida como la solución más capaz para la simulación en tiempo real. Las pruebas del sistema integrado revelaron que la arquitectura distribuida del panel inferior, donde microcontroladores dedicados gestionan subsistemas específicos, es significativamente más robusta que la arquitectura centralizada del panel superior. Si bien el uso de expansores de puertos I²C en el panel superior es funcional para operaciones a velocidad normal, la latencia inherente del bus I²C limita su rendimiento en escenarios de alta velocidad. Esta es una conclusión de diseño fundamental: futuras evoluciones del sistema deben priorizar una arquitectura completamente distribuida para maximizar la capacidad de respuesta y la fidelidad de la simulación.

3. Se implementó un ecosistema de software completo, modular y desacoplado. El firmware, basado en POO, MQTT y FreeRTOS, demostró ser una solución robusta para la inteligencia embebida. La HMI en Python/PySide6 se consolidó como un potente centro de control para el instructor. Sin embargo, se concluye que su dependencia de una red local es una limitación estratégica para su escalabilidad. Una migración de la HMI a una arquitectura web es esencial para desbloquear el potencial de entrenamiento remoto y posicionar el simulador en el mercado de formación internacional.
4. Se validó la funcionalidad del sistema integrado y su alineación con los estándares de la industria. Las pruebas experimentales confirmaron la correcta correlación entre las acciones en la HMI y las respuestas del prototipo físico, verificando la capacidad del simulador para replicar procedimientos operativos conforme a los protocolos del US Navy Diving Manual. El sistema, operado por un instructor, es una herramienta de capacitación efectiva para el personal de superficie, abordando directamente la brecha entre la formación teórica y la competencia práctica.

En síntesis, este trabajo ha entregado un sistema funcional y validado a un TRL 5, que demuestra la viabilidad de crear herramientas de entrenamiento de alto impacto con una arquitectura mecatrónica costo-efectiva.

RECOMENDACIONES

Basado en los hallazgos del proyecto y el análisis del panorama tecnológico y normativo de la industria, se proponen las siguientes líneas de trabajo futuro para maximizar el impacto y la viabilidad del simulador:

1. Capitalizar la Ventaja Competitiva de la Arquitectura Mecatrónica. El prototipo ha demostrado un equilibrio óptimo entre costo, funcionalidad y realismo. A diferencia de los simuladores puramente neumáticos (costosos) o las soluciones basadas en pantallas táctiles (que carecen de retroalimentación háptica), este diseño electromecánico ofrece una usabilidad táctil esencial para la memoria muscular con un costo de fabricación y consumo energético reducidos. Se recomienda profundizar en esta propuesta de valor, refinando la respuesta de los actuadores y los componentes mecánicos para consolidar el simulador como una solución accesible y competitiva en el mercado de la formación.

2. Priorizar la Migración a una Arquitectura Web para Entrenamiento Global. La principal limitación estratégica del sistema actual es su dependencia de una red local. Se recomienda priorizar la reingeniería de la HMI hacia una plataforma web (utilizando frameworks como React o Vue.js). Esta arquitectura permitiría un modelo de "formación como servicio", donde el hardware podría ser enviado a cualquier centro de buceo del mundo, mientras que el instructor y los sistemas de evaluación operarían desde la nube. Esto no solo habilitaría el entrenamiento remoto, sino que abriría un nuevo y escalable modelo de negocio.
3. Desarrollar un Motor de Escenarios Automatizados para Estandarizar la Evaluación:
Para evolucionar el simulador de una herramienta operada por un instructor a una plataforma de evaluación autónoma, es crucial desarrollar un "Motor de Escenarios". Este módulo de software permitiría programar secuencias de eventos y fallos que se ejecutan automáticamente, registrando las respuestas del operador (tiempos de reacción, acciones correctas/incorrectas) para generar un informe de desempeño objetivo. Esto estandarizaría la evaluación y aumentaría la escalabilidad de la herramienta de entrenamiento.
4. Expandir las Capacidades con Módulos de Entrenamiento Avanzado. Para que el simulador sea relevante en la formación de buzos profesionales de alto nivel, es necesario integrar módulos que cubran escenarios críticos. La recomendación principal es el diseño de un simulador de cámara hiperbárica. Este permitiría practicar los complejos procedimientos de descompresión en superficie, una de las operaciones de mayor riesgo que actualmente carece de herramientas de entrenamiento seguras y accesibles.
5. Iniciar un Proceso de Validación Formal y Acreditación Internacional. El objetivo estratégico a largo plazo debe ser la certificación del simulador por organismos reconocidos.
 - Validación Nacional: Presentar el prototipo a DIRECTEMAR para obtener retroalimentación formal y asegurar la alineación con la normativa chilena.
 - Acreditación Internacional (IMCA): El paso más significativo sería buscar la alineación con los estándares de la International Marine Contractors Association (IMCA). Según el esquema IMCA D013, el tiempo de entrenamiento en simuladores aprobados (como los de Clase B, digitales) puede ser acreditado formalmente como "horas de panel" para la certificación de Supervisores de

Buceo. Lograr que las horas en este simulador sean reconocidas oficialmente transformaría la herramienta de una ayuda pedagógica a una parte integral de la carrera profesional de un buzo.

- Certificación del Sistema (DNV): Explorar la certificación del propio sistema de simulación bajo estándares como DNV-ST-0033, que, valida el realismo, la fiabilidad y la eficacia pedagógica de la tecnología. Esto proporcionaría una garantía de calidad reconocida internacionalmente y aumentaría exponencialmente el valor comercial y la credibilidad del simulador en la industria de la formación profesional.

BIBLIOGRAFÍA

- A. Association of Diving Contractors International (ADCI). (2021). International consensus standards for commercial diving and underwater operations (Version 6.3).
- B. Banks, A., & Gupta, R. (2014). MQTT essentials: A lightweight IoT protocol. Packt Publishing.
- C. Barry, R. (2018). Mastering the FreeRTOS real time kernel: A hands-on tutorial guide. Real Time Engineers Ltd.
- D. Bennett, P. B., & Elliott, D. H. (Eds.). (2003). Bennett and Elliott's physiology and medicine of diving (5th ed.). Saunders.
- E. Blanchon, B. (2023). ArduinoJson [Computer software]. <https://arduinojson.org/>
- F. Dirección General del Territorio Marítimo y de Marina Mercante (DIRECTEMAR). (2019). Reglamento de buceo para buzos profesionales. Armada de Chile.
- G. Espressif Systems. (2022). ESP8266EX datasheet.
- H. Espressif Systems. (2023). ESP32-S3 series datasheet.
- I. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). Design patterns: Elements of reusable object-oriented software. Addison-Wesley.
- J. Hastie, T., Tibshirani, R., & Friedman, J. (2009). The elements of statistical learning: Data mining, inference, and prediction. Springer.
- K. Horowitz, P., & Hill, W. (2015). The art of electronics (3rd ed.). Cambridge University Press.
- L. Kolban, N. (2018). Kolban's book on ESP32. Leanpub.
- M. Lippmann, J., & Millar, S. (2018). Deeper into diving (2nd ed.). J.L. Publications.

- N. McCauley, M. (2021). AccelStepper: A C++ library for controlling stepper motors [Computer software]. <https://www.airspayce.com/mikem/arduino/AccelStepper/>
- O. Monk, S. (2017). Programming Arduino next steps: Going further with sketches (2nd ed.). McGraw-Hill Education.
- P. NXP Semiconductors. (2021). PCF8574/74A Remote 8-bit I/O expander for I²C-bus datasheet.
- Q. O'Leary, N. (2022). PubSubClient: A client library for MQTT [Computer software]. <https://pubsubclient.knolleary.net/>
- R. OASIS. (2019). MQTT Version 5.0. OASIS Standard.
- S. Proakis, J. G., & Manolakis, D. G. (2007). Digital signal processing: Principles, algorithms, and applications (4th ed.). Pearson Prentice Hall.
- T. Summerfield, M. (2021). Rapid GUI programming with Python and Qt (2nd ed.). Prentice Hall.
- U. Texas Instruments. (2019). ADS111x ultra-small, low-power, I²C-compatible, 860-SPS, 16-bit ADCs with internal reference datasheet.
- V. Texas Instruments. (2020). CD74HC4067 High-speed CMOS logic 16-channel analog multiplexer/demultiplexer datasheet.
- W. The Qt Company. (2023). Qt for Python (PySide6) [Computer software]. <https://www.qt.io/qt-for-python>
- X. U.S. Navy, Supervisor of Diving. (2016). U.S. Navy Diving Manual, Revision 7, Change A (Report No. SS521-AG-PRO-010). Commander, Naval Sea Systems Command.