UNIVERSIDAD POLITÉCNICA SALESIANA

SEDE GUAYAQUIL

CARRERA DE ELECTRÓNICA Y AUTOMATIZACIÓN

BIKER POINT – SISTEMA DE ESTACIONAMIENTO PARA BICICLETAS UTILIZANDO EL MICROCONTROLADOR PIC 18F

Trabajo de titulación previo a la obtención del

Título de Ingeniero Electrónico

AUTOR: MARCO VINICIO CAIN YUQUILEMA

TUTOR: Ing. ORLANDO BARCIA AYALA MSc.

Guayaquil – Ecuador

2024

CERTIFICADO DE RESPONSABILIDAD Y AUTORÍA DEL TRABAJO DE TITULACIÓN

Yo, Marco Vinicio Cain Yuquilema con documento de identificación N^{O} 0604678698 manifiesto que:

Soy el autor y responsable del presente trabajo; y, autorizo a que sin fines de lucro la Universidad Politécnica Salesiana pueda usar, difundir, reproducir, o publicar de manera total o parcial el presente trabajo de titulación.

Guayaquil, 22 de enero del año 2024

Atentamente,

Marco Vinicio Cain Yuquilema

0604678698

CERTIFICADO DE CESIÓN DE DERECHOS DE AUTOR DEL TRABAJO DE TITULACIÓN A LA UNIVERSIDAD POLITÉCNICA SALESIANA

Yo, Marco Vinicio Cain Yuquilema con documento de identificación N^{O} 0604678698, expreso mi voluntad y por medio del presente documento cedo a la Universidad Politécnica Salesiana la titularidad sobre los derechos patrimoniales en virtud de que soy autor del proyecto técnico: "Biker Point — Sistema de Estacionamiento Para Bicicletas Utilizando el Microcontrolador PICI 8F", el cual ha sido desarrollado para optar por el título de: Ingeniero Electrónico, en la Universidad Politécnica Salesiana, quedando la Universidad facultada para ejercer plenamente los derechos cedidos anteriormente.

En concordancia con lo manifestado, suscribo este documento en el momento que hago la entrega del trabajo final en formato digital a la Biblioteca de la Universidad Politécnica Salesiana.

Guayaquil, 22 de enero del año 2024

Atentamente,

Marco Vinicio Cain Yuquilema

0604678698

CERTIFICADO DE DIRECCIÓN DEL TRABAJO DE TITULACIÓN

Yo, Orlando Barcia Ayala con documento de identificación N^{O} 1309445714, docente de la Universidad Politécnica Salesiana, declaro que bajo mi tutoría fue desarrollado el trabajo de titulación: **BIKER POINT - SISTEMA DE ESTACIONAMIENTO PARA BICICLETAS UTILIZANDO EL MICROCONTROLADOR PIC18F**, realizado por: Marco Vinicio Cain Yuquilema con documento de identidad N^{O} 0604678698, obteniendo como resultado final el trabajo de titulación bajo la opción de proyecto técnico que cumple con todos los requisitos determinados por la Universidad Politécnica Salesiana.

Guayaquil, 22 de enero del año 2024

Atentamente,

Ing. Orlando Barcia Ayala MSc.

1309445714

DEDICATORIA

Con cariño para el ser que me sonríe y me cuida desde el cielo ¡Para ti papá!

Dedico también a mi madre, resaltando toda su entrega, amor y sacrificio.

A mis apreciados hermanos por brindar el apoyo incondicional en todo momento.

Me dedico a mí por concluir con éxito la carrera profesional que años atrás añoraba inquietantemente conocerla, vivirla y ejercerla.

Evidentemente este logro más que personal es de mi querida familia CAIN YUQUILEMA.

AGRADECIMIENTO

Me siento dichoso de llegar a este instante, de poderles compartir esta emoción de agradecimiento por ser partícipes en la culminación de uno de mis metas.

Agradezco a Dios por proveer todas las maravillas del mundo, pido sabiduría necesaria para apreciarla, empatía para convivir en armonía con la biodiversidad y conocimiento para contribuir con un granito de arena que tanto necesita el planeta de nosotros.

A mis padres por invertir parte de sus vidas en sus hijos, con el único anhelo de que aprovechemos las oportunidades que tenemos y cosechemos a futuro éxitos y felicidad en nuestras vidas, caminando siempre senderos de honradez y disciplina, a mis hermanos por ese apoyo incondicional y la maravillosa unidad que nos destaca como familia, a los docentes, amigos con los que compartí durante estos años y desde luego al honorable Ing. Orlando Barcia por esa predisposición de respaldar mi proyecto y dirigirla.

¡Mil gracias a todos!

Marco Vinicio

RESUMEN

El objetivo del presente proyecto es diseñar e implementar el prototipo BIKER POINT- SISTEMA DE ESTACIONAMIENTO PARA BICICLETAS UTILIZANDO EL MICROCONTROLADOR PIC18F, como una alternativa de solución al bajo nivel de seguridad que presentan los sitios de parqueos de bicicletas convencionales. BIKER POINT constituye un sistema mecánico de seguridad, diseñado para el bloqueo y desbloqueo de 2 cerraduras elaboradas para parquear bicicletas; también, está integrado de un sistema electrónico y control con interfaz de interacción para los usuarios del prototipo; asimismo, integra un elemento del sistema de firmware programado con lenguaje C, compilada y grabada en un microcontrolador PIC18F2550. El prototipo está desarrollado para brindar un parqueo seguro con capacidad para dos unidades; seguro 1 para la bicicleta 1 y seguro 2 para la bicicleta 2. El funcionamiento de este módulo es a base de un identificador biométrico que consiste en un sensor de huella dactilar, mediante el cual los usuarios se registrarán a través del sensor dactilar al momento de aparcar su unidad, luego cuando el usuario requiera retirar su unidad, el sistema solicitará su identificación dactilar verificando su registro; si la huella dactilar coincide con el registro, el sistema abrirá el seguro correspondiente, caso contrario, le negará el retiro de la unidad. De la misma forma, se incorporó un display LCD 20x4 para facilitar el uso de este prototipo. La metodología de la implementación está enfocada en el modelo DESIGN THINKING por lo que se desarrolló la siguiente secuencia: empatizar la problemática, definir las características del prototipo, idear el boceto del sistema, prototipar la elaboración del módulo y evaluar su operatividad. En conclusión, el sistema BIKER POINT es óptimo y listo para su funcionamiento.

Palabras claves: Sistemas embebidos, Prototipo, Reconocimiento Biométrico, Pensamiento de diseño, Estacionamiento de Bicicletas.

ABSTRACT

The objective of this project is to design and implement the BIKER POINT - PARKING SYSTEM FOR BICYCLES USING THE PIC18F MICROCONTROLLER, as an alternative solution to the low level of security presented by conventional bicycle parking sites. BIKER POINT constitutes a mechanical security system, designed for locking and unlocking 2 locks made for parking bicycles; Also, it is integrated with an electronic and control system with an interaction interface for the prototype users; Likewise, it integrates an element of the firmware system programmed with the C language, compiled and recorded on a PIC18F2550 microcontroller. The prototype is developed to provide safe parking with capacity for two units; insurance 1 for bicycle 1 and insurance 2 for bicycle 2. The operation of this module is based on a biometric identifier that consists of a fingerprint sensor, through which users will register through the fingerprint sensor at the time of park your unit, then when the user requires to remove their unit, the system will request their fingerprint identification verifying their registration; If the fingerprint matches the registration, the system will open the corresponding lock, otherwise it will deny you the removal of the unit. In the same way, a 20x4 LCD display was incorporated to facilitate the use of this prototype. The implementation methodology is focused on the DESIGN THINKING model, so the following sequence was developed: empathize with the problem, define the characteristics of the prototype, devise the sketch of the system, prototype the development of the module and evaluate its operability. In conclusion, the BIKER POINT system is optimal and ready for operation.

Keywords: Embedded Systems, Prototype, Biometric Recognition, Design Thinking, Bicycle Parking.

ÍNDICE GENERAL

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

Ante el bajo nivel de seguridad en los sistemas de parqueos de bicicletas convencionales, se plantea el diseño y la implementación del prototipo BIKER POINT con la finalidad de brindar un sistema de parqueo fiable y seguro en cualquier lugar que fuese instalado.

BIKER POINT es un módulo de parqueo de bicicletas con capacidad para dos unidades que cuenta con dos seguros, un seguro para cada unidad. La estructura del módulo es de material de acero inoxidable que puede ser fijado en el piso en cualquier espacio, cuyas dimensiones son: 25 cm x 40 cm de base y 90 cm de alto, con un panel frontal intuitivo para guiar a los usuarios a través del uso de los botones.

La operatividad del prototipo se da a través de la conexión de los tres subsistemas:

El sistema mecánico. - Este sistema ejecuta el bloqueo y desbloqueo de los dos seguros del módulo basado en el principio de propulsión por cadena de eslabones y piñones que ejecuta el movimiento rectilíneo de un vástago que funciona de forma independiente para cada seguro, dependiendo del sentido de giro de los motores DC para su operación.

El sistema electrónico y de control. - La función de este sistema es integrar y controlar la operatividad de todo el prototipo en conjunto el cual consta de una tarjeta de control, un panel de operación con LCD 20x4, sensor dactilar AS608, Botón PARO DE EMERGENCIA, botoneras NA, LEDS indicadores y una tarjeta de potencia donde se conectan la salida a 2 motores 12VDC. Además, contiene un microcontrolador PIC18F2550 el cual ejecuta el código programado en lenguaje C.

El algoritmo de operación (firmware). - El código fuente del algoritmo se programó en lenguaje C en el software MPLAB, luego con la herramienta de compilación es traducida al lenguaje máquina y finalmente grabada en el microcontrolador PIC18F2550 que cumple la función de CPU de BIKER POINT.

La operatividad externa de este módulo es a base de un identificador biométrico que funciona a base de un sensor de huella dactilar, mediante el cual los usuarios se registrarán a través del sensor dactilar para aparcar su bicicleta, de la misma forma, para retirar su unidad, el sistema solicitará su identificación dactilar y verificará su registro; si la huella dactilar es validado por el sistema, automáticamente se BIKER POINT procederá a la apertura del seguro correspondiente, caso contrario, será negado el retiro. De esta forma, este prototipo garantiza la seguridad óptima de parqueo seguro a los propietarios de las bicicletas.

PROBLEMA

En la actualidad, los usuarios que optan por utilizar un medio de transporte ecológico y económico como las bicicletas no encuentran un sistema de parqueo seguro en espacios regulares o espacios públicos. Esta causa genera diferentes incidentes para el propietario tales como: el fácil hurto de la bicicleta ya que la unidad no tiene ningún tipo de seguridad, muchas veces los usuarios solo dejan un seguro manual improvisado con una cadena y candado en una de las llantas. Cabe mencionar que estas cadenas pueden ser cortadas fácilmente para sustraerlo. Otras veces, estas llaves son extraviadas por propietarios quienes deben enfrentar un sin número de acciones para desbloquearlo perdiendo su tiempo y creando incidentes. Además, la falta de parqueos seguros para este medio transporte, es un limitante para la población el uso de bicicletas para movilizarse a cualquier lugar ya que temen no encontrar un sitio confiable para aparcar su unidad por lo tanto prefieren usar otro medio de transporte.

Antecedentes:

A partir de la pandemia del covid-19, existe un incremento considerable de la demanda y uso de bicicletas por parte de las personas de todas las edades para las diferentes actividades ya sean laborales o recreativas. No obstante, los sitios de parqueo son escazas o no presentan una seguridad confiable al usuario, Derivándose en el aumento del delito de robos de bicicletas que va en aumento, su incremento mes a mes es el 7,5% (expreso, 2022). A continuación, en la Figura 1 se muestra la forma más común de hurto de bicicletas de los parqueos convencionales.

Figura 1

Hurto común de bicicletas

Nota: hurto común de bicicletas de parqueos convencionales. Tomado de (REALINCE FOUNDRY, 2016)

La seguridad que posee los mecanismos de parqueo convencionales existentes actualmente en los lugares de parqueo para este tipo de vehículos presenta gran nivel de vulnerabilidad.

Mediante este prototipo y la cooperación de instituciones públicas y privadas, BIKER POINT se proyecta a crear puntos de parqueo seguros y prácticos de poder utilizar para este tipo de transporte, en los puntos estratégicos y beneficiar a muchas personas que se suman a la movilidad ecológica.

Importancia y alcances:

En la actualidad se maneja mucho el tema de sostenibilidad, y las energías verdes. El prototipo BIKER POINT aporta directamente al fortalecimiento de estos conceptos en nuestro entorno, otro de los importantes aportes a la población es ofrecer sitios de parqueos más seguros que los sistemas convencionales que ya existen, para de esta manera fomentar mayor uso de bicicletas en las tareas

cotidianas. Esta solución está diseñada para ser instalado en diferentes áreas, comercios, instituciones, parques, gimnasios, cines, peluquerías o en cualquier lugar donde se lo requiera.

Delimitación:

Delimitación temporal:

El sistema BIKER POINT se desarrolló en el domicilio del autor del proyecto ubicado en la provincia del Guayas, cantón Durán, Cdla. El Recreo MZ. 239, V. 40, en un lapso aproximado de un periodo académico.

Delimitación Espacial:

El proyecto es de interés social, estará a disposición de cualquier ciudadano, empresas o instituciones públicas y privadas que deseen implementar el sistema BIKER POINT en sus instalaciones, pensando en el beneficio propio, así también en el de sus clientes y colaboradores.

Delimitación Académica:

La propuesta planteada se desarrolló cumpliendo todos los requerimientos solicitados para la titulación por la institución académica Universidad Politécnica Salesiana, bajo el modelo de proyecto técnico. Se integra los conocimientos teóricos, técnicos adquiridos en el transcurso de la carrera: electrónica digital, sistemas micro controlados, metodología de la investigación, programación en lenguaje C, electiva, física, automatización y control, además se contó con herramientas tecnológicas para el diseño, implementación y simulación del circuito de control de BIKER POINT.

OBJETIVOS

Objetivo principal:

- Diseñar e implementar el prototipo BIKER POINT sistema de estacionamiento para bicicletas utilizando el microcontrolador PIC18F.

Objetivos específicos:

- Diseñar la estructura metálica del prototipo con su respectivo mecanismo de seguridad.
- Implementar en el microcontrolador PIC18F2550 el algoritmo de operación del sistema en lenguaje C.
- Armar el panel de mando con el sensor de huella activado con botones, de igual manera el circuito eléctrico y electrónico del sistema.
- Realizar pruebas y ajustar parámetros para el correcto funcionamiento de BIKER POINT.

FUNDAMENTOS TEÓRICOS

El SISTEMA DE SEGURIDAD Y REGISTRO POR MEDIO DE ACCESO RFID PARA MÓDULO DE PARQUEADERO DE BICICLETAS implementado por estudiantes de la Universidad Distrital San José de Caldas comparte el objetivo en común con BIKER POINT, el diseño y la implementación de un sistema para el parqueo de bicicletas, como cerebro del sistema de control se encuentra un microcontrolador PIC18f4550. A diferencia de BIKER POINT que funciona mediante identificación biométrica con el uso del sensor AS608 este sistema trabaja con tarjetas RFID que son leídas por el módulo RC522, el sistema cuenta con su LCD para facilitar la interacción con el usuario. El mecanismo que sujeta y asegura la bicicleta es un solenoide ubicado en la parte superior del sistema. Cuenta con una aplicación móvil para el monitoreo en tiempo del estado del sistema o incidencia en el sitio (Garzón Felipe, 2017). La desventaja de este sistema es que es de dimensiones muy grandes y solo posee capacidad para una bicicleta, inclusive los propios autores sugieren de mejoras en este aspecto.

Figura 2

SISTEMA DE SEGURIDAD DE BICICLETAS

Nota: Sistema de parqueo en operación, Tomado de (Garzón Felipe, 2017).

También se analizó una solución interesante que diseñan en el proyecto BIKER-CAMP & CLOUD COMPUTING. Es un modelo de propuesta para el desarrollo de parqueos inteligentes en la ciudad

de Bogotá, desarrollado por: Camilo Guerrero y Álvaro Reyes. Proponen el uso de una red de comunicación inalámbrica para el control de bici parqueos especialmente bajo el protocolo 802.11, además la integración a una plataforma cloud mediante una arquitectura flexible (Guerrero Camilo, 2020).

También sugieren la integración de herramientas tecnológicas NTIC, que posibilitan y proyectan la mejora de espacios seguros de bici parqueos, de igual manera la compatibilidad y la interconexión a servicios cloud computing e IoT.

Reconocimiento Biométrico

Los datos biométricos han sido usados para identificar y diferenciar a un individuo de manera única, los rasgos usados por civilizaciones antiguas eran pocas y simples como la voz y el rostro(Rojas Portilla & Rueda, 2018). En la actualidad los sistemas inteligentes de identificación utilizan múltiples parámetros físicos tales como: Iris de los ojos, asimetría de, reconocimiento facial, etc. En la figura 3 se muestra el dato biométrico más utilizado en diferentes aplicaciones y soluciones de identificación que es la: La huella dactilar que está formado por dos parámetros morfológicos las crestas y los valles.

Figura 3

Huella dactilar

Nota: Digitalización de la huella dactilar. Tomado de (Welivesegurity, 2018).

Sensor de huella dactilar A608S

Dispositivo electrónico mostrada en la figura 4, diseñada para capturar los patrones de la huella dactilar de una persona y convertir esos patrones en datos informáticos. Las principales características son:

Alimentación 3.3V – 5.0 VDC, capacidad de buffer 162 plantillas, protocolo de comunicación UART, maneja velocidad de 9600 a 57600 Baudios (Adafruit Industries, 2024).

Figura 4

Sensor óptico A608S

Nota: Sensor de huella dactilar con interfaz UART. Tomado de (Rajguru Electronics (I) Pvt. Ltd., 2018)

Algoritmo de operación sensor dactilar óptico

Mediante un algoritmo los sensores basan su funcionamiento en la siguiente secuencia.

1. Captar la imagen de la huella dactilar
2. Comparar el patrón de los valles y crestas de la imagen capturada
3. Mediante el protocolo de comunicación UART esta información es enviada a un microcontrolador en forma de paquete.

Tabla 1

Paquete de Transmisión

cabecera	Dirección	Identificador de paquete	Longitud de paquete	Contenido (información, parámetro)	Check um

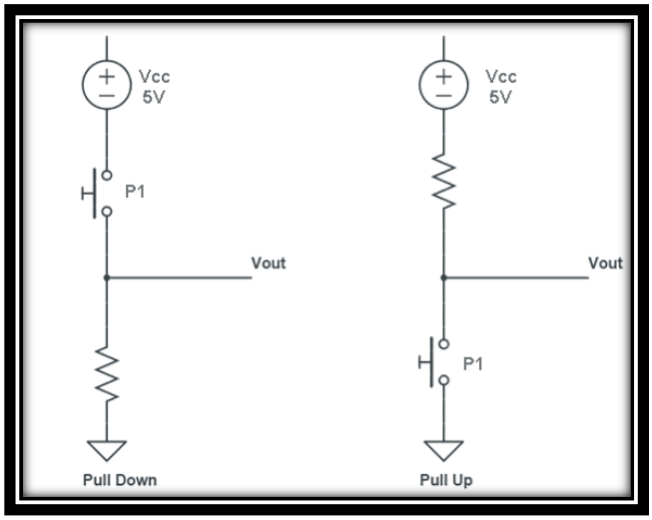
Nota: Paquete de datos enviado por el sensor dactilar al microcontrolador. (Autoría propia, 2024).

Electrónica Digital

A diferencia de la electrónica analógica donde todas las magnitudes son continuas y varían de amplitud con respecto al tiempo, en cambio la electrónica digital consta de dos valores discretos el 0 y el 1, es una manera de representar la presencia o la ausencia de magnitud de variables o datos. En electrónica digital el cero representa el nivel lógico bajo y el valor uno o alto representa la presencia de datos o magnitud de variables, pude ser de corriente, voltaje, pulsos (Tokheim, 2021).

Figura 5

Resistencia PULL DOWN Y PULL UP



Nota: Diagrama de configuración de resistencias en PULL DOWN y PULL UP. Tomado de (Francisco Ernesto Cortez, 2021)

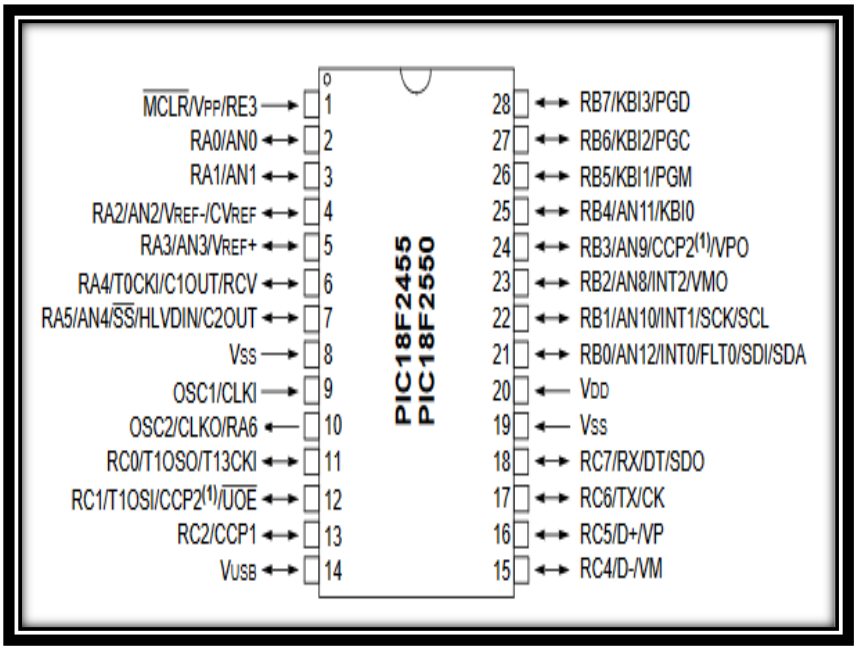
La Figura 5 muestra la configuración de resistencias PULL DOWN y PULL UP garantizan que las señales recibidas no sean producto de fluctuaciones en la señal de entrada, en PULL DOWN el valor que se asume cuando no ningún emisor de señal externo está enviando una equivalente a el estado bajo, lo contrario en PULL UP que en su diseño original siempre percibe un estado alto hasta que ocurra un cambio de estado mediante periféricos externos (Artero, 2019).

MICROCONTROLADOR PIC18F2550

Es un dispositivo programable que consta de una computadora digital, una unidad de almacenamiento de datos, una unidad de almacenamiento de programas y puertos de entrada/salida en un circuito integrado mostrado en la figura 6, actúa como un controlador periférico en un sistema mínimo (Microchip technollogy Inc., 2006).

Figura 6

Pines de PIC18F2550



Nota: Detalle de disposición de pines y puertos del microcontrolador PIC18F2550. Tomado de (Microchip Technollogy Inc., 2023)

- Las entradas Vdd y Vss del microcontrolador dependen de una fuente de alimentación de al menos 5V y 0V respectivamente para funcionar.
- Necesita una señal de reloj que indique la frecuencia de funcionamiento, que introducimos a través de un oscilador de cristal de cuarzo en los pines OSC1 y OSC2.
- El MCLR que es el pin de reinicio que activa el microcontrolador (Omar Enrique Barra Zapata, 2018).

El funcionamiento de un microcontrolador está determinado por un programa almacenado en su memoria flash ROM y se puede programar varias veces para cambiar su estado y comportamiento, lo que convierte a los microcontroladores en una parte esencial del rápido desarrollo de las aplicaciones electrónicas. Procede de la familia de los PIC18F (Lucechetti, 2021).

Memoria EEPROM

Tambien lllamada memoria no volatil de datos, posee la capacidad de ser programadas o reprogramadas por la CPU de un microcontrolador con esto se garantiza que en situaciones de corte de suministro o fallas similarea a ello los valores o variables criticas se mantengan redundantes incluso si se apaga el microcontrolador, dichos valores o variables estaran disponibles al momento de volver a encender el PIC (Mosquera, 2018).

Tabla 2

Funciones memoria EEPROM

FUNCIÓN	DESCRIPCIÓN
EEPROM_Read(dirección)	Retoma un byte de la dirección específica
EEPROM_Write(dirección, dato)	Escribe un dato en la direccipon especifica

Nota: Es de gran utilidad estas funciones, ya que nos facilita la lectura y la escritura en la memoria EEPROM. (Autoría propia, 2024).

PWM

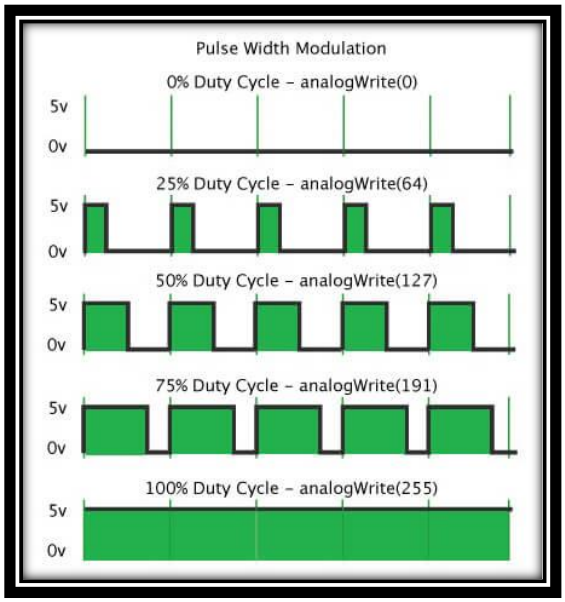
El PWM (Modulación de ancho de pulso) es el método básico usado en el control electrónico,

La duración o ancho de pulso varía de manera proporcional a la amplitud de la señal moduladora, técnica usada en variedad de aplicaciones para controlar la velocidad de motores, intensidad de luminosidad, nivel de temperatura, etc.

La frecuencia es constante lo que varia y con ello la señal resultante es el tiempo de la variable en estado alto y la misma variable en estado bajo tal como se ilustra en la figura 7 (Artal-Sevil, 2018).

Figura 7

PWM ciclos de trabajo de variables en estado alto y bajo



Nota: Ejemplos de duración de ciclos de trabajo de PWM. Tomado de (Circuits Geeks, 2022).

Protocolo de comunicación UART

El UART es empleado para la trasmisión asíncrono de datos entre el DTE (Equipo de comunicación de datos) carece de sincronización de datos (Luis Gil Sánchez, 2018). Esta comunicación puede ser:

- Simplex. Único sentido de trasmisión y recepción

- Full Dúplex. Se puede manejar el envío y recepción en las dos direcciones y de manera simultánea para aquello se debe configurar lo siguiente

La unidad de medida para la velocidad de trasmisión de datos es bits por segundo.

Protocolo de comunicación I2C

Este protocolo diseñado por Philips a principios de los 80s facilita la comunicación entre microcontroladores, memorias y algunos dispositivos con cierto nivel de inteligencia. Para su implementación solo requiere de dos líneas y una común conectado a masa, la velocidad de trasmisión mediante esta tecnología es de 100Kbits/s. se basa en la comunicación serie y síncrona (Ortega, 2020).

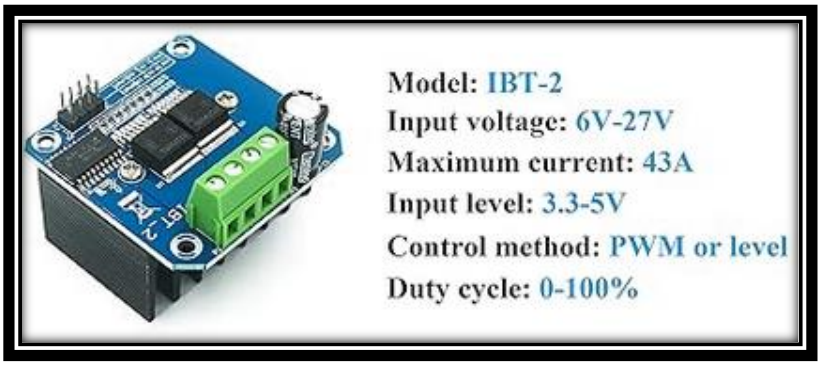
Módulo IBT_2

Módulo de manejo para motores DC hasta 43A conformada en chip BTN7960B/BTS7960B. posee la capacidad de manejar unidades de la potencia y elementos de solenoides frecuencia de operación máxima es 25KHz, compatible con Arduino, pic y AVR (Universidad Politécnica de Madrid, 2018).

A continuación en la Figura 8 se muestra el driver de potencia para motor DC y sus características.

Figura 8

Módulo de potencia IBT_2



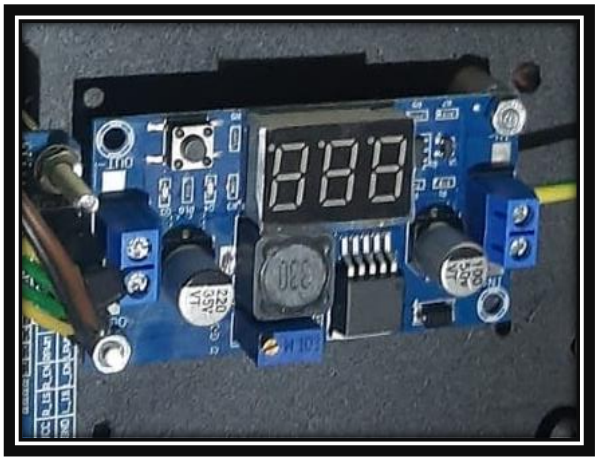
Nota: El modulo IBT_2 tolera niveles de corriente hasta 43ª. Tomado de (Mvtronic, 2016).

Modulo regulador DC-DC 2A

Regulador de voltaje monolítica. Tiene la capacidad de regulador de conmutación con una carga de hasta 3A, se alimenta con 12VDC y su salida es voltaje DC regulable (Quintero, 2019). En la figura 9 se muestra un ejemplar del módulo LM2596.

Figura 9

Módulo LM2596



Nota: Módulo LM2596 con display con señal de salida regulable. (Autoría propia, 2024).

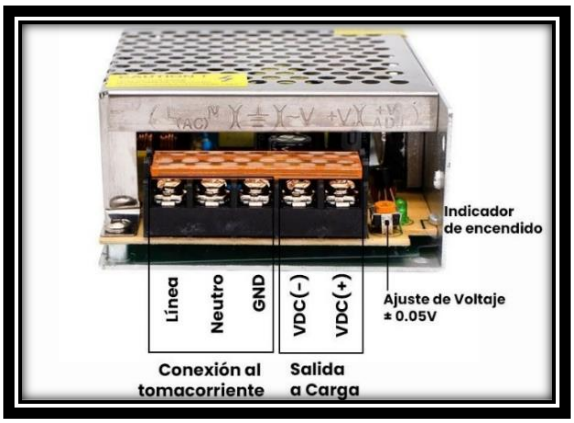
Fuente de alimentación conmutada 12VDC – 5A

Elemento de fuente de alimentación AC/DC, El modelo mostrado en la figura 10 posee las siguientes características técnicas.

- Voltaje de salida: 12V DC
- Corriente Max.: 5A DC
- Potencia Nominal: 60W
- Tiempo de activación: 20 ms/230VAC
- Voltaje de entrada: AC 100-120 V, AC 200-240 V, 50Hz/60Hz
- Eficiencia (Típico):> 70%
- Temperatura de trabajo: -10 ~ + 60 ° C

Figura 10

Fuente conmutada



Nota: Fuente conmutada con alimentación a 110 o 120 VAC. Tomado de (DigitalPaper, 2016)

LCD 20X4

Para optimizar los recursos del PIC es posible manejar la LCD con 2 líneas de comunicación, utilizando el módulo PCF8574. El módulo está diseñado para conectarse de manera directa al LCD, cada uno de sus pines coinciden perfectamente entre los dos elementos (WINSTAR Display Co.,Ltd.). A continuación, se muestra el display LCD 20X4 en la figura 11.

Figura 11

Módulo LCD 20X4

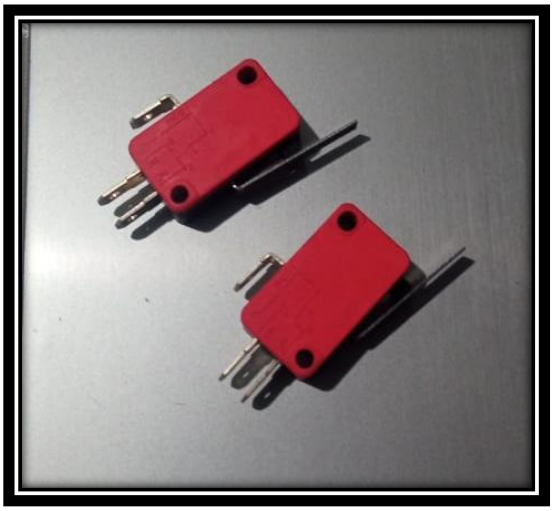
Nota: Pantalla LCD con módulo I2C compacto. (Fuente propia, 2024).

Interruptor de fin de carrera

Se denomina final de carrera o también sensor de contacto, Ejemplares de finales de carrera mostrados en la figura 12. Consiste en un interruptor se coloca en una posición estratégica de la carrera de los elementos móviles para saber la posición de dicho elemento. El contacto NA hace la función de conducir señal eléctrica al momento de ser oprimido (Ingeniería Mecafenix , 2021)

Figura 12

Sensor final de carrera



Nota: Sensor interruptor de señal eléctrica. (Autoría propia, 2024).

Motor 12vdc con reductor

Son motores eléctricos que convierten la energía eléctrica en energía mecánica y se utilizan en robótica, gracias a una desaceleración adecuada, se puede lograr un movimiento seguro y eficiente. Los motores con reductores ofrecen una serie de ventajas, como una perfecta regularidad de velocidad, potencia de transmisión, peso ligero y tamaño pequeño (Loreto, 2015).

En la Figura 13 se muestra el motor reductor elevavidrios universal.

Figura 13

Motor 12VDC - 5A

Nota: Motor reductor 12VDC – 5A. Tomado de (BETAFIX S.A, 2023)

Pulsadores NA

Botón o pulsador eléctrico Figura 14, es un componente eléctrico que permite o impide el paso de la corriente cuando esta se aprieta o pulsa. Existen dos tipos de pulsadores NA y NC estas se abren o se cierran dependiendo del tipo (Ocampo C., 2020).

Figura 14

Pulsadores NA

Nota: Pulsadores de contacto normalmente abierto. (Fuente propia, 2024)

Diodo leds

El diodo emisor de luz o LED (light-emitting diode) es un elemento electrónico que emite fotones al momento de recibir corriente eléctrica de muy baja intensidad. Únicamente conduce corriente al ser polarizado directamente, es decir en sentido cátodo o terminal negativa hacia el ánodo o terminal de conexión positiva (Mecatrónicalatam.com, 2021).

Figura 15

Diodos leds

Nota: Diodos emisores de luz utilizados como indicadores. (Fuente propia, 2024).

El diodo emisor de luz o LED (light-emitting diode) es un elemento electrónico que emite fotones al momento de recibir corriente eléctrica de muy baja intensidad. Únicamente conduce corriente al ser polarizado directamente es decir en sentido cátodo o terminal negativa hacia el ánodo o terminal de conexión positiva (mecatrónicalatam.com, 2021).

Pulsador de parada de emergencia

El objetivo y la función principal de este dispositivo dentro de un equipo o circuito será de interrumpir en caso de peligro el suministro de fuentes de energía y detener la operación de un sistema o máquina lo más pronto posible. Consiste en un interruptor de tipo NC situado en la línea de alimentación, debe colocarse en un lugar de fácil acceso y visible para el operario (Cosaar, 1984).

A continuación, se muestra un ejemplar de un modelo de botón de PARO DE EMERGENCIA en la Figura 16.

Figura 16

Botón de paro de emergencia

Nota: Dispositivo de seguridad Pulsador de parada de emergencia. Tomado de (ABB, 2019)

Lenguaje de programación C

El lenguaje de programación C se caracteriza por ser un lenguaje de medio nivel, de propósito general. Se basa en estructuras de programación tipo control-flujo (Buriticá, 2017).

Permitiendo la implementación de algoritmos de las siguientes clases:

- De toma de decisiones (if-else)
- Conjunto de casos posibles (switch)
- Términos de ciclos con criterio (while)

Variables y Registros

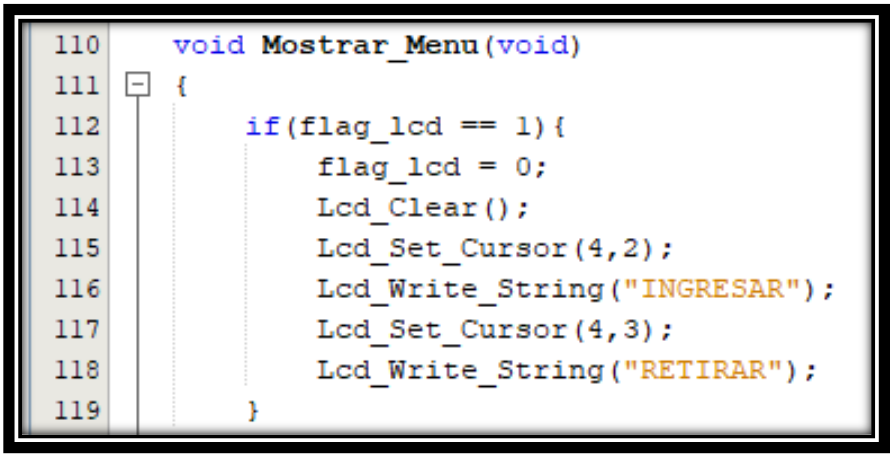
Las variables son espacios en la memoria que se utilizan para almacenar datos en las diferentes etapas del programa, Variables de entrada, variables de salida y variables intermedios, Arreglos una matriz de variables en bloque puede almacenar conjunto de valores de un mismo tipo (Sixto Reinoso, 2018).

Funciones

En Lenguaje C se maneja 2 tipo de funciones. Las funciones principales y las funciones auxiliares. La función principal está diseñada para el código principal de un algoritmo, en cambio las funciones auxiliares realizan tareas puntuales (Chávez, 2019).

Figura 17

Función auxiliar Mostrar_Menu



```
110   void Mostrar_Menu(void)
111   {
112       if(flag_lcd == 1){
113           flag_lcd = 0;
114           Lcd_Clear();
115           Lcd_Set_Cursor(4,2);
116           Lcd_Write_String("INGRESAR");
117           Lcd_Set_Cursor(4,3);
118           Lcd_Write_String("RETIRAR");
119       }
```

Nota: Líneas de código de la función Mostrar_Menu. (fuente propia, 2024).

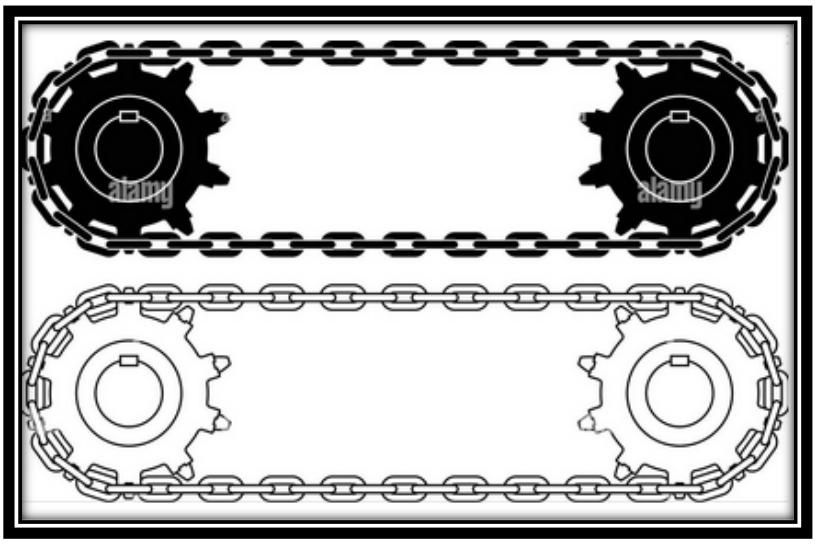
Propulsión por cadena – piñón

Es un sistema de transmisión de movimiento similar a la carrea dentada y bandas, la diferencia es que este sistema ocupa una cadena de eslabones y rodillos que interconectan piñones y coronas dentadas, trabaja de manera sincronizada y ofrece una gran trasmisión de potencia. Es un sistema básico pero muy robusto el cual permite el uso en un sin número de aplicaciones de nivel doméstico e industrial (Soler, 2020).

La configuración de un sistema de propulsión por cadena de eslabones y piñón se evidencia en la Figura 18.

Figura 18

Sistema de propulsión por cadena-piñón



Nota: Representación general de un sistema de propulsión por cadena de eslabones y piñones.

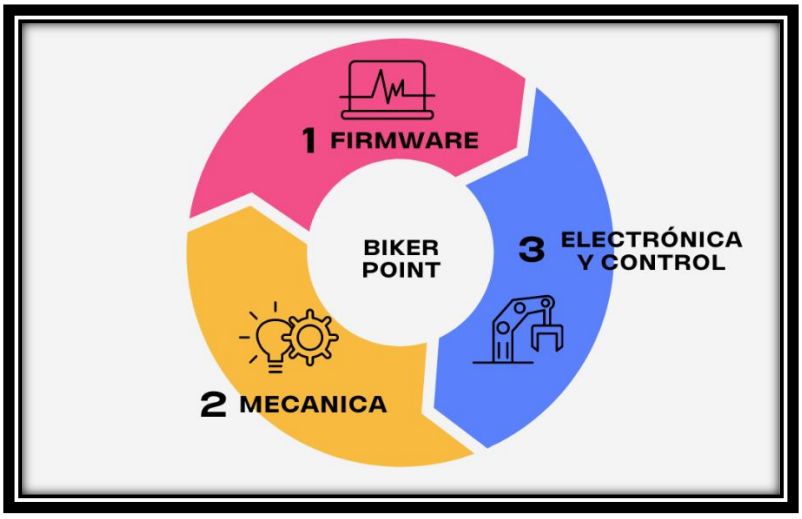
Tomado de (Alamy, s f)

MARCO METODOLÓGICO

BIKER POINT prototipo conformado por 3 subsistemas: Sistema Mecánico, Sistema electrónico de control y el algoritmo de operación (firmware) diseñado para el parqueo de bicicletas. A demás cuenta con identificación biométrica para el registro de los usuarios. En la figura 19 se muestra la integración de los 3 sistemas.

Figura 19

Subsistemas de BIKER POINT



Nota: Representación de los 3 subsistemas que conforman BIKER POINT. (Autoría propia, 2024).

La investigación, diseño e implementación se llevó bajo el modelo DESIGN THINKING.

Es una Metodología de resolución de problemas o planteamientos a través de un proceso ordenado y sencillo, el comienzo del proceso marca el comprender problemas, analizar y generar ideas que permite resolver problemas a través de la creación rápida de prototipos. A pesar de ser creado para el sector educativo, es una herramienta muy poderosa, aplicable en cualquier ambiente o campo (ITMadrid Digital School, 2020).

Los pasos de esta metodología se ilustra en la Figura 20.

Figura 20

Pasos de DESIGN THINKING

Nota: Pasos de desarrollo de BIKER - POINT de acuerdo con el modelo DESIGN THINKING.

(Fuente propia, 2024).

Paso 1: EMPATIZAR

El Objetivo principal del creador de BIKER POINT es brindar solución a la falta de sistemas confiables e intuitivos para el parqueo de las bicicletas en lugares públicos. A partir de la pandemia del covid-19 la demanda y el uso de este medio de transporte se incrementó de manera exponencial. Se vio la necesidad de muchas personas que hacen uso de estos medios de transportación ecológicos.

Paso 2: DEFINIR

A partir del paso uno nació la idea de crear un sistema de parqueo que tenga las siguientes características:

- Intuitivo
- Fiable
- Robusto

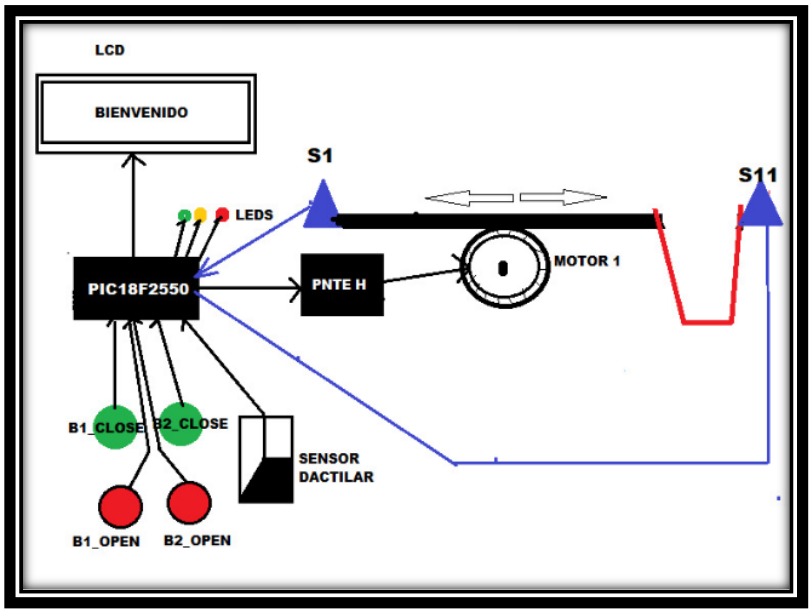
Estos requisitos fueron indispensables para buscar ideas y proyectar soluciones, puesto que el sistema desarrollado se instalará en un ambiente externo donde estará bajo condiciones de radiación, humedad, polvo.

Paso 3: IDEAR

Una vez reunido estas cualidades se formuló un boceto y se investigó el presupuesto requerido para este prototipo.

Figura 21

Boceto del Sistema BIKER POINT



Nota: Representación general de uno de los mecanismos de seguridad. (Fuente propia, 2024).

Además del boceto se planteó las siguientes interrogantes para analizar el proyecto a implementar:

- **¿Qué conceptos de la electrónica y la ingeniería se va a aplicar?**

Sistemas embebidos, Tecnología TTL, Lenguaje de programación C, Comunicación serial, Identificación biométrica, Diseño de circuitos electrónicos.

- ¿**Nombre del sistema?**

BIKER POINT

- **¿Qué material se va a usar para la cubierta del sistema?**

Es necesaria que la cubierta sea apta para condiciones de operación en un ambiente externo, por ende, el acero inoxidable es el más adecuado.

- **¿Qué mecanismo de seguridad implementar?**

Cerraduras de metal conectados mediante un Mecanismo de movimiento rectilíneo, el sentido de giro de los motores determina si la cerradura se abre o se cierra, adicional a aquello se implementará sensores finales de carrera que indicaran al microcontrolador en qué estado se encuentra la cerradura.

- **¿Cuál será el dispositivo que actuará como el cerebro del sistema BIKER POINT?**

De acuerdo con los requerimientos existentes para el sistema, el microcontrolador a utilizar es el PIC18F2550, el mismo que ofrece la velocidad de CPU hasta 48 MHz, 24 pines E/S, memoria programable de 32KB, memoria RAM 2KB, rango de alimentación 4.2V – 5.5V, con una retención de datos hasta 40 años.

- **¿Cuál será la interfaz para que el usuario pueda utilizar sin dificultad?**

Se implementará una LCD que indica el menú y las instrucciones a seguir al usuario durante la operación de BIKER POINT.

- **¿Qué recursos tecnológicos se va a requerir?**

Para el desarrollo de BIKER POINT se requiera un ordenador con capacidades adecuadas para el diseño y simulación de circuitos electrónicos. También para el diseño de la PCB de la tarjeta de control. La programación y compilación del algoritmo de funcionamiento del sistema también se realizará con softwares especializadas y la PC.

En base a estas interrogantes, sumada de conocimientos de electrónica, herramientas tecnológicas de diseño, simulación y programación adquiridos durante la carrera así también se investigó la

disponibilidad de los materiales necesarios en el mercado y los recursos monetarios que incluye. Finalmente se definió desarrollar el sistema BIKER POINT - SISTEMA DE ESTACIONAMIENTO PARA BICICLETAS UTILIZANDO EL MICROCONTROLADOR PIC18F.

Paso 4: PROTOTIPAR

En esta instancia, contando con todos los elementos físicos, softwares, materiales, se procede a implementar cada uno de los subsistemas del prototipo BIKER POINT.

Se inicio con la implementación del sistema mecánico ya que de esta depende las dimensiones del equipo.

Implementación de sistema mecánico

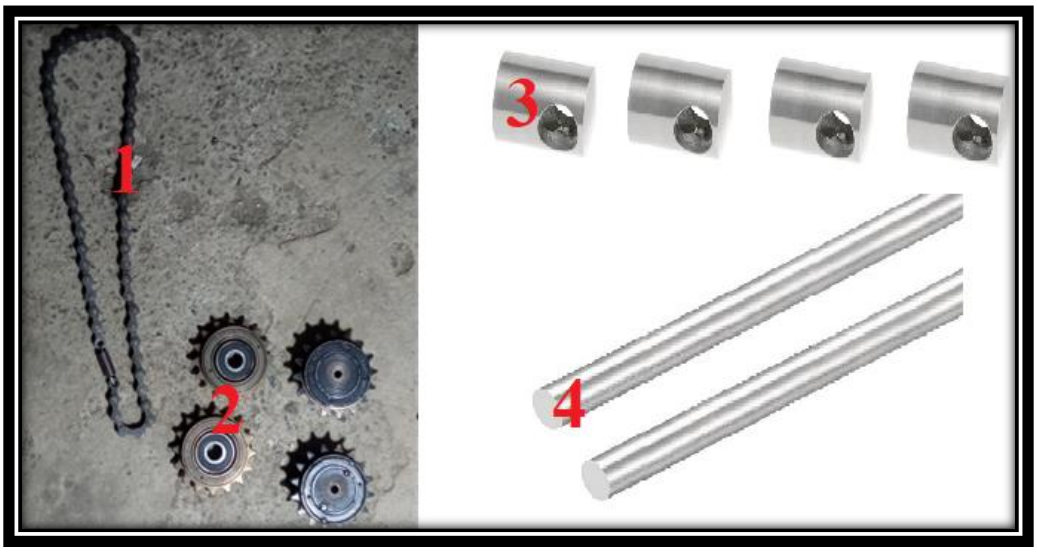
Consiste en un sistema de trasmisión de engranaje y cadenas. El sistema mecánico de BIKER POINT fue el primero en implementarse, ya que aquello me daba las pautas para diseñar los subsistemas que restaban, ejemplo las dimensiones del gabinete, la ubicación de los elementos eléctricos y electrónicos, etc.

Pasos de desarrollo del sistema mecánico de BIKER POINT

1. Enlistar los respectivos elementos del sistema que se muestra en la figura 22.
2. Diseño de placa de soporte del sistema mecánico.
3. Montaje de motores en la placa de soporte.
4. Montaje del sistema de propulsión por cadenas y piñones
5. Readecuación del diseño de la cubierta del sistema
6. Instalación del vástago y cerraduras

Figura 22

Elementos del sistema mecánico



Nota: El sistema mecánico esta basado en propulsion por cadenas y engranajes que se unen a un vástago movil, mediante un conectro, el cierre y apertura de la cerradura depende del sentido de giro de los motores. (Autoría propia, 2024).

Tabla 3

Elementos de sistema mecánico

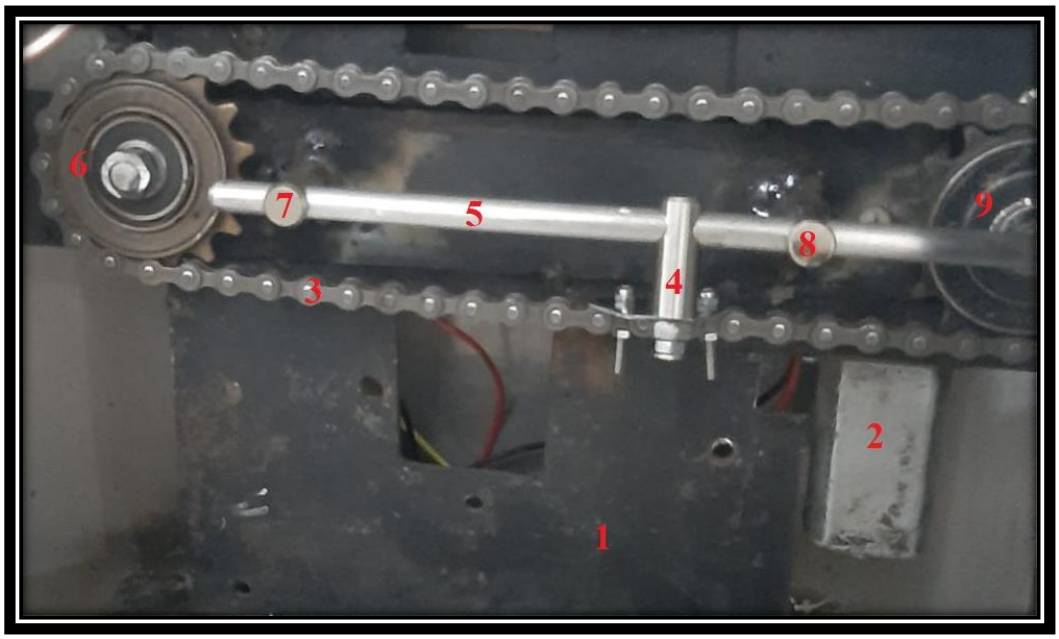
ITEM	DETALLE
1	Cadena de propulsión (Cadena de bicicleta)
2	Engranajes piñon (rache estándar de bicicletas)
3	Soporte de acero inoxcidable 8 mm (Guias)
4	Varilla de acero inoxcidable 8mm x 500mm (Vástago)

Nota: Piezas mecánicas que forman parte del sistema mecánico de BIKER POINT. (Fuente propia, 2024).

En la figura 23 se muestra el sistema de propulsión por cadena de eslabones y sus elemntos mecánicos.

Figura 23

Configuración sistema mecánico



Nota: Sistema de propulsión mediante cadena y engranajes. (Autoría propia, 2024).

Tabla 4

Sistema de propulsión CADENA - PIÑON

ITEM	DETALLE
1	Placa metálica 20x25 cm y 4mm de espesor.
2	Motor 12VDC
3	Cadena
4	Conector (Vástago y cadena de propulsión)
5	Vástago (pieza de cierre y apertura del seguro)
6 y 9	Engranajes (uno de ellos se une al mecanismo de rotación del motor)
7 y 8	Guías del vástago

Nota: Elementos del sistema de propulsión mediante cadena y engranajes. (Autoría propia, 2024).

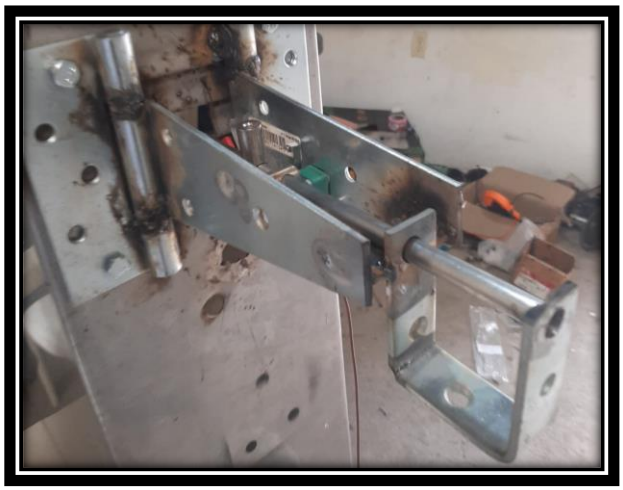
Figura 24

Ensamblaje de gabinete

Nota: Fabricación de gabinete en acero inoxidable. (Autoría propia, 2024).

Figura 25

Seguro mecánico



Nota: cerradura de seguridad y parte del mecanismo externo. (Autoría propia, 2024).

Posteriormente se procede a implementar parte del mecanismo de seguridad de la parte externa del gabinete, al igual que la cerradura de seguridad, los sensores de carrera se colocan en el soporte externo del mecanismo como se observa en la Figura 25.

El vástago de las cerraduras al estar directamente conectado a la cadena de propulsión su movimiento lineal de cierre y apertura depende directamente del sentido del giro de los motores.

Implementación del sistema eléctrico y control

1. **Asignación de entradas, salidas, pines de configuración y alimentación.**

Tabla 5

Asignación de pines del microcontrolador PIC18F2550

PIN	DETALLE	ENTRADA/SALIDA
1	MSCLR	Entrada
2	B1 (B1 OPEN)	Entrada
3	B1 (CLOSE)	Entrada
4	B2 (OPEN)	Entrada
5	B2 (CLOSE)	Entrada
6	LEDV	Salida
7	LEDR	Salida
8	VSS (GROUND)	-
9	OSCILADOR EXTERNO	-
10	OSCILADOR EXTERNO	-
11	M2-L (MOTOR2 LEFT)	Salida
12	M2-R (MOTOR2 RIGHT)	Salida
13	-	-

14	-	-
15	-	-
16	-	-
17	TX (LECTOR DE HUELLA)	-
18	RX (LECTOR DE HUELLA)	-
19	VSS (GROUND)	-
20	VDD (POWER)	-
21	SDA (LCD)	-
22	SCL (LCD)	-
23	S1 (S1 OPEN)	Entrada
24	S1 (S1 CLOSE)	Entrada
25	S2 (S2 OPEN)	Entrada
26	S2 (S2 CLOSE)	Entrada
27	M1-L (MOTOR1 LEFT)	Salida
28	M1-R (MOTOR1 RIGHT)	Salida

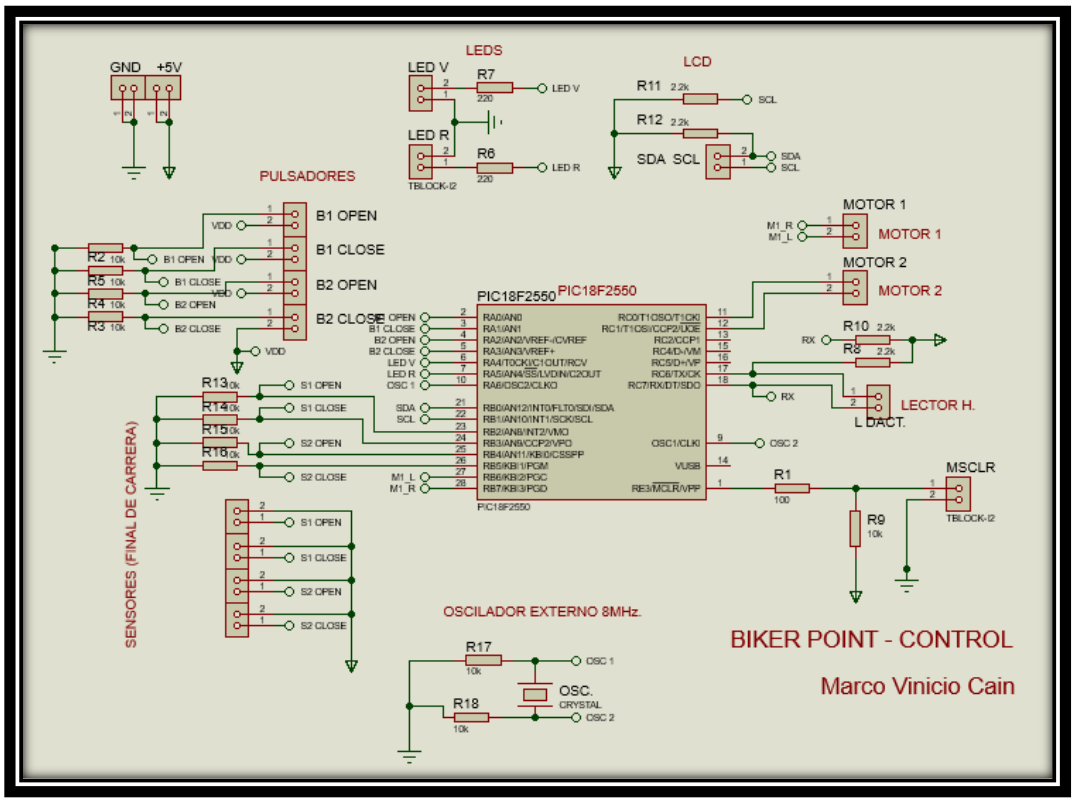
Nota: Detalle de la conexión de elementos en cada uno de los pines del microcontrolador.

(Fuente propia, 2024).

2. Diseño de tarjeta de control de BIKER POINT

Figura 26

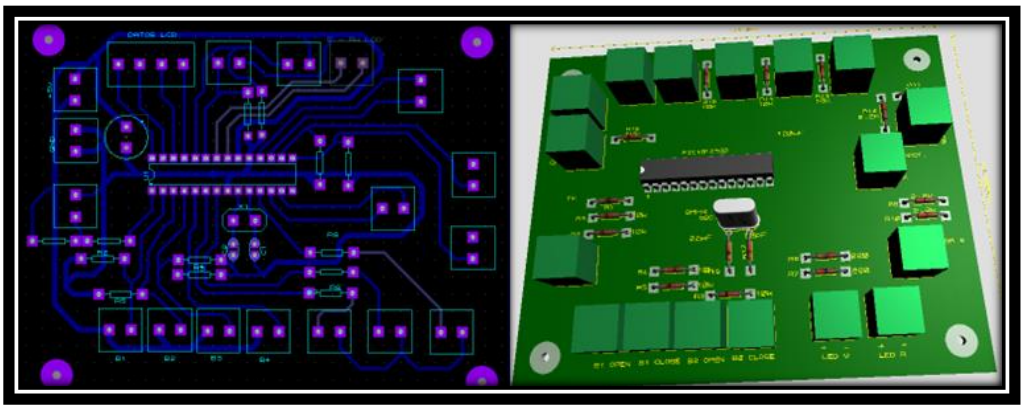
Diseño circuito PCB



Nota: Circuito esquemático del diseño de tarjeta de control. (Autoría propia, 2024).

Figura 27

Diseño de tarjeta PCB

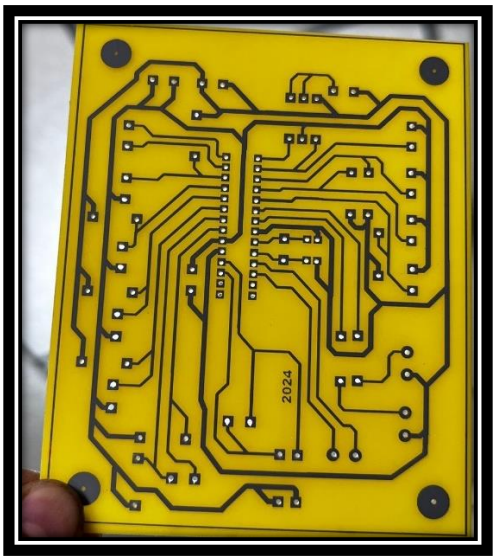


Nota: Tarjeta PCB. (Fuente propia, 2024).

3. Impresión de la tarjeta PCB.

Figura 28

PCB Tarjeta de control

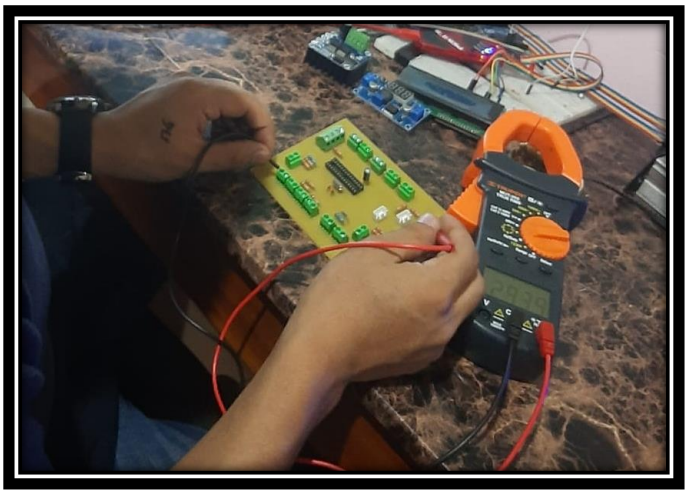


Nota: Tarjeta PCB lista para el montaje de la tarjeta de control. (Fuente propia, 2024).

4. Montaje electrónico y pruebas de la tarjeta PCB.

Figura 29

Montaje tarjeta PCB



Nota: Montaje de elementos electrónicos de la tarjeta de control. (Fuente propia, 2024).

5. **Instalación del circuito de potencia.**
6. **Implementación del panel de control.**
7. **Interconexión de sensores, motores, fuente de alimentación, panel de control y todos los elementos eléctricos del sistema.**

Programación algoritmo de operación (FIRMWARE)

Implementación del algoritmo de funcionamiento de BIKER - POINT en lenguaje C

1. **Definición de la operación del sistema y condiciones de programación.**

Se desea implementar el algoritmo de operación para el ingreso y retiro de bicicletas de BIKER POINT. El control consiste en la apertura y cierre de SEGURO 1 y SEGURO 2 del sistema. El registro y la identificación del usuario será a través del sensor dactilar AS608, además el sistema posee un LCD donde se mostrará el menú principal, instrucciones de operación y mensajes de advertencia al usuario. También posee un led verde que se enciende si el registro o identificación es correcta y un led rojo que indicara que existe un problema de identificación de usuario. Se aplicará señal PWM a los motores.

1. **Definición de pasos para el ingreso de bicicletas (cierre de seguros)**

- Solicitud de registro de huella dactilar “Coloque su huella”.
- Captura de la huella dactilar del usuario.
- Registro de la huella dactilar del usuario.
- Cierre del SEGURO elegido por el usuario.
- Encendido led verde y mensaje “Usuario Registrado”.

2. **Definición de pasos para el retiro de bicicletas (apertura de seguros).**

- Solicitud de validación del registro del usuario “Coloque su huella”
- Búsqueda del usuario en el registro.

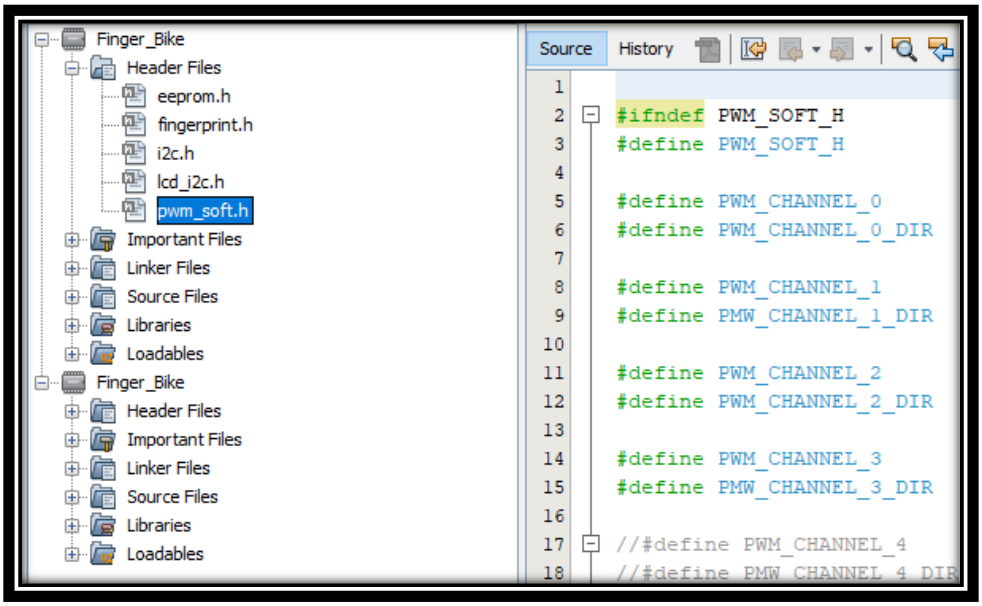
- Apertura del SEGURO si el usuario registrado es correcto, caso contrario “Usuario no identificado”, encendido led rojo y paso 4.
- Búsqueda por segunda ocasión del usuario en el registro.
- Apertura del SEGURO si el usuario registrado es correcto, caso contrario “Usuario no identificado”, encendido led rojo y menú principal.

3. Creación de librerías para los periféricos y protocolos de comunicación

Creación de librerías que facilita la funcionalidad y la comunicación de los periféricos con el microcontrolador. Cada uno de los periféricos, protocolos de comunicación y programación específica requiere de librerías propias como se muestra en la Figura 30 el ejemplo de la librería de programación PWM por software.

Figura 30

Librería pwm_soft.h



Nota: Parte de líneas de código de la programación del PWM. (Fuente propia, 2024).

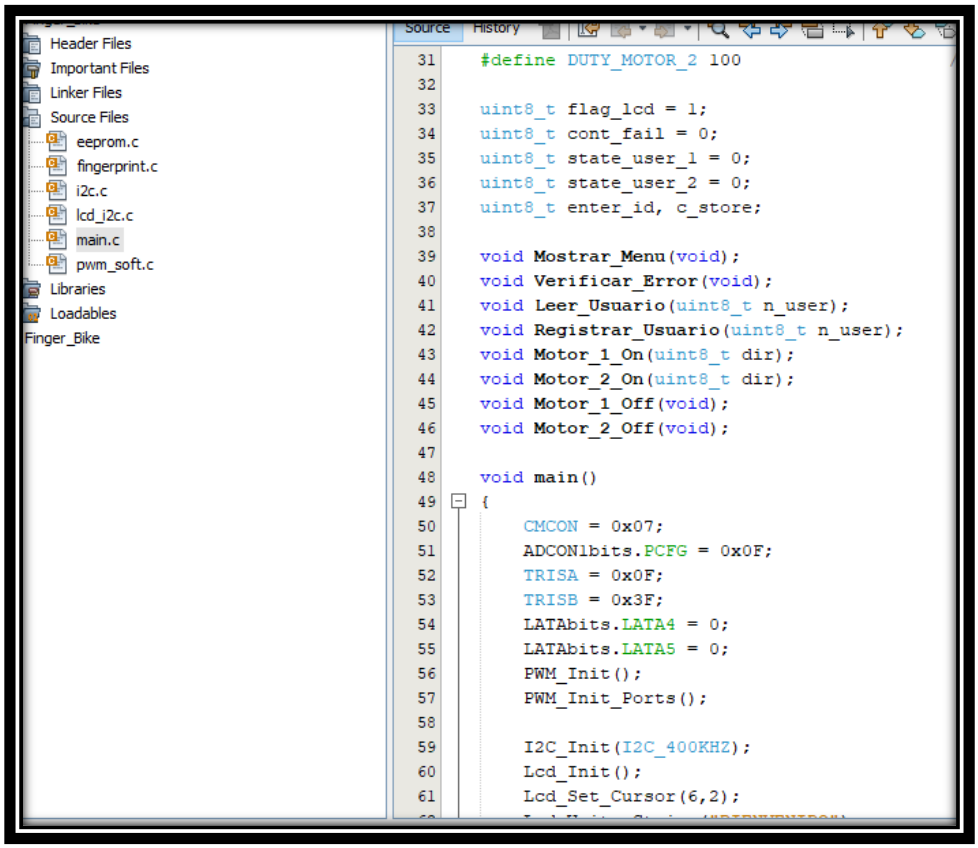
4. Creación del código fuente en el lenguaje C con el software MPLAB

Tomando en cuenta la secuencia de pasos del prototipo en operación, los datos de entrada, salida y condicionales, se elabora un diseño de algoritmo en lenguaje C en el software MPLAB. Se elaboró una función para cada tarea especifica de los elementos del sistema con ello se obtiene un mejor contraste en el manejo de la información, reduciendo el número de líneas de código en el programa principal. Y facilidad de corrección de errores.

Parte de la estructura de un algoritmo en lenguaje C es la descripción de bibliotecas y las configuraciones generales. También conocido como encabezado. La función main de la Figura 31 define el algoritmo de funcionamiento del sistema BIKER POINT.

Figura 31

Función main



Nota: Parte del código de la función principal main. (Fuente propia, 2024).

5. Compilación y grabado del algoritmo desarrollado en el PIC18F2550

El grabado del algoritmo de operación (firmware) en el microcontrolador se realizó mediante la herramienta PICKIT3 tal como se demuestra en la Figura 32.

Figura 32

Grabado del algoritmo de operación del sistema

Nota: El grabado se realizó con la herramienta pickit 3.

Paso 5: EVALUAR

Una vez terminado el ensamblaje de BIKER POINT se procedió a evaluar el correcto funcionamiento del sistema en conjunto. Además, en este proceso de evaluación se identificó algunos inconvenientes en el desempeño del módulo, los cuales están citados en el apartado de resultados específicamente en realización de pruebas. los problemas menores presentados a lo largo del desarrollo de cada subsistema de BIKER POINT se solucionaron a lo largo del desarrollo de los mismos para hacer posible el avance del proyecto.

RESULTADOS

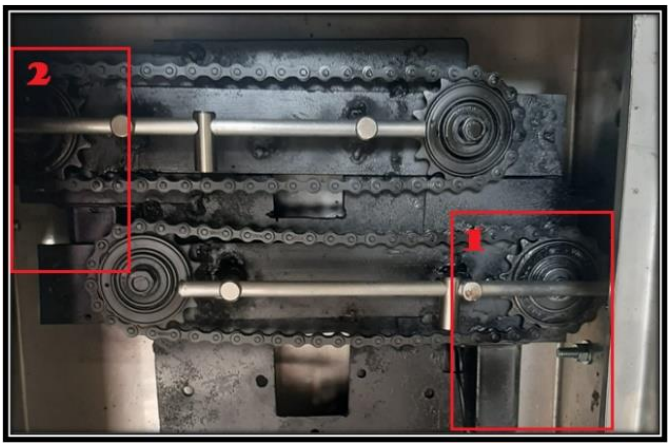
En el presente proyecto se planteó el siguiente objetivo general que es: diseñar e implementar el prototipo BIKER POINT sistema de estacionamiento para bicicletas utilizando el microcontrolador PIC18F. Al respecto, se puede constatar el cumplimiento de este tal como se planteó. Además, cabe indicar que para el cumplimiento del objetivo general se ha trazado una secuencia de actividades los cuales son los denominados como objetivos específicos que se han ido desarrollando de forma ordenada y progresiva. Consecutivamente, se detallan los resultados obtenidos por cada una de estas etapas.

Diseño de la estructura metálica del con su respectivo mecanismo de seguridad

En esta etapa se diseñó la cubierta metálica en acero inoxidable y el sistema mecánico de seguridad. En la Figura 33 se muestra el sistema mecánico de propulsión implementado que funciona mediante cadena de eslabones y piñones. Además, el montaje final en el interior de la estructura prototipo. Conjuntamente de los vástagos que son los elementos de cierre y apertura de cada uno de los seguros.

Figura 33

Sistema mecánico de BIKER POINT

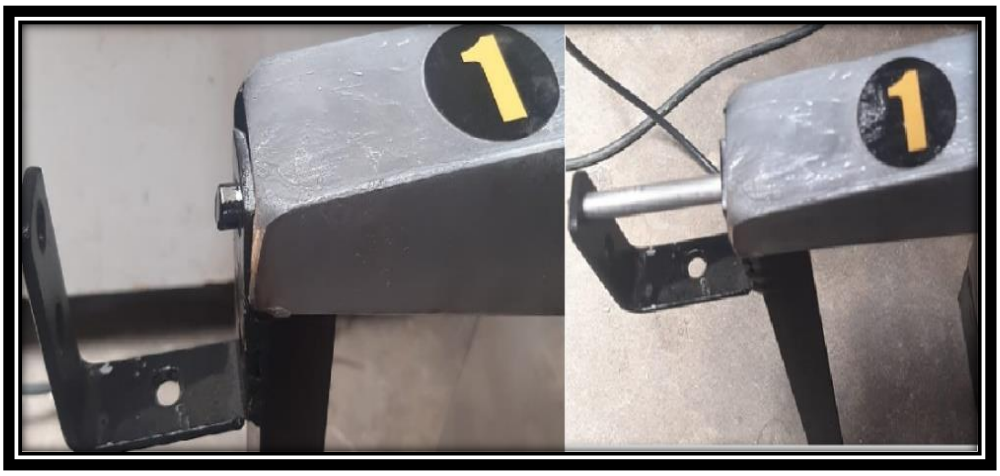


Nota: Sistema de propulsión mecánico montado en el módulo. (Autoría propia, 2024).

El vástago de las cerraduras, al estar directamente conectado a la cadena de propulsión, su movimiento lineal de cierre y apertura depende directamente del sentido del giro de los motores. En la figura 34 se puede apreciar los dos posibles estados del SEGURO 1.

Figura 34

Estados de operación del SEGURO 1



Nota: Estados de operación de SEGURO1/2 son idénticas. (Autoría propia, 2024).

Diseño del panel de mando y de circuito electrónico

En esta etapa se implementó el panel de mando con el sensor de huella activado con botones, de la misma forma el circuito eléctrico y electrónico del sistema. A continuación, en la figura 36 se refleja el panel con todos los componentes: Botón PARO DE EMERGENCIA, LCD 20x4, LEDS, SENSOR AS608, botoneras NA que funciona como una interfaz del usuario.

Figura 35

Panel de control del sistema BIKER POINT

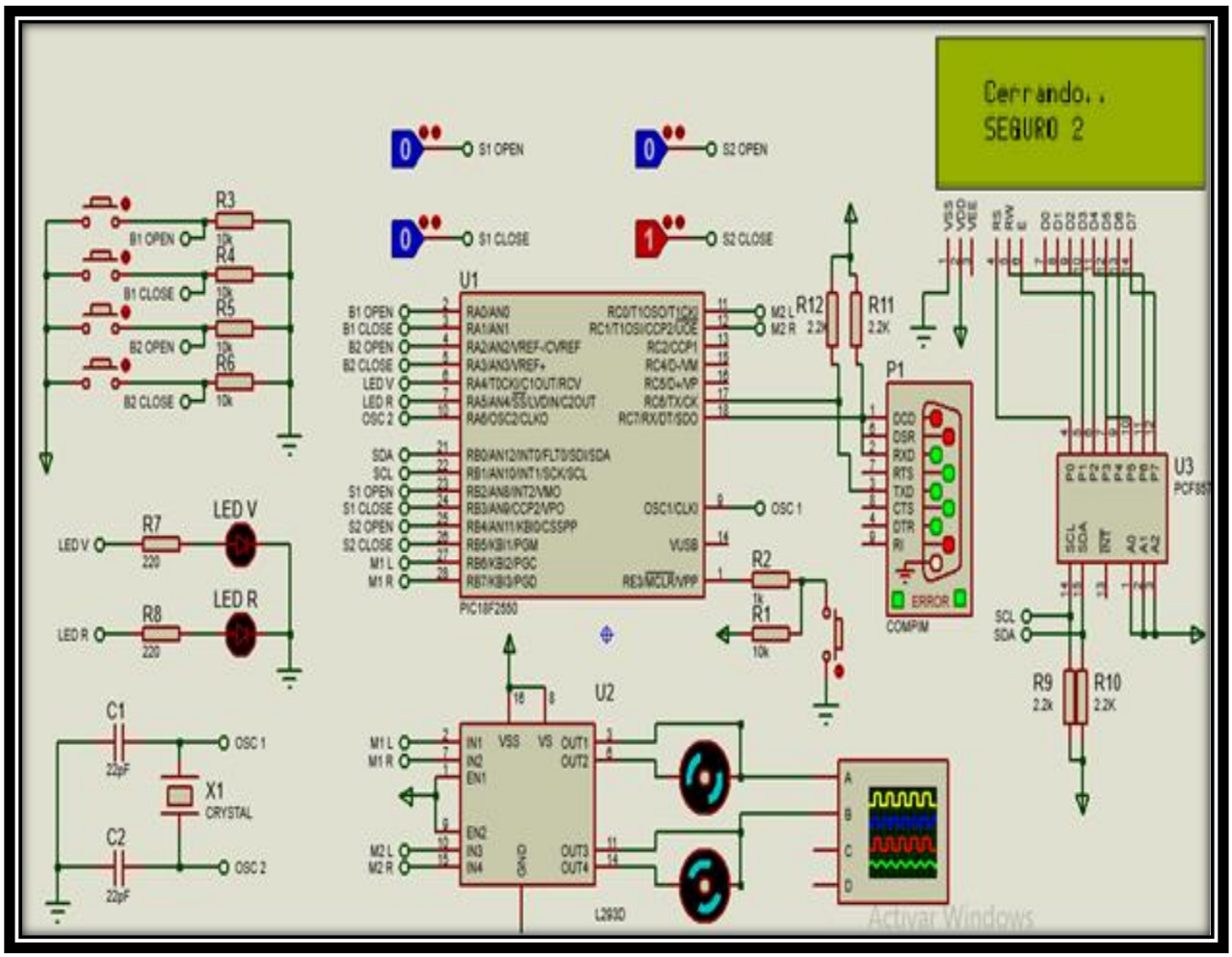
Nota: El panel de control cuenta con su pantalla LCD, botoneras de operación, botón de parada de emergencia, sensor dactilar AS608, leds indicadores y sus respectivos etiquetas de las funciones de cada elemento del panel. (Autoría propia, 2024).

El panel de mando con los periféricos del sistema está debidamente rotulado para facilitar al usuario el uso del sistema BIKER POINT.

La simulación y pruebas del sistema electrónico y la programación se llevó a cabo mediante el software Proteus. También se hizo el uso del Osciloscopio virtual para ver las grafica de las señales PWM tal como se muestra en la Figura 36.

Figura 36

Simulación del sistema electrónico

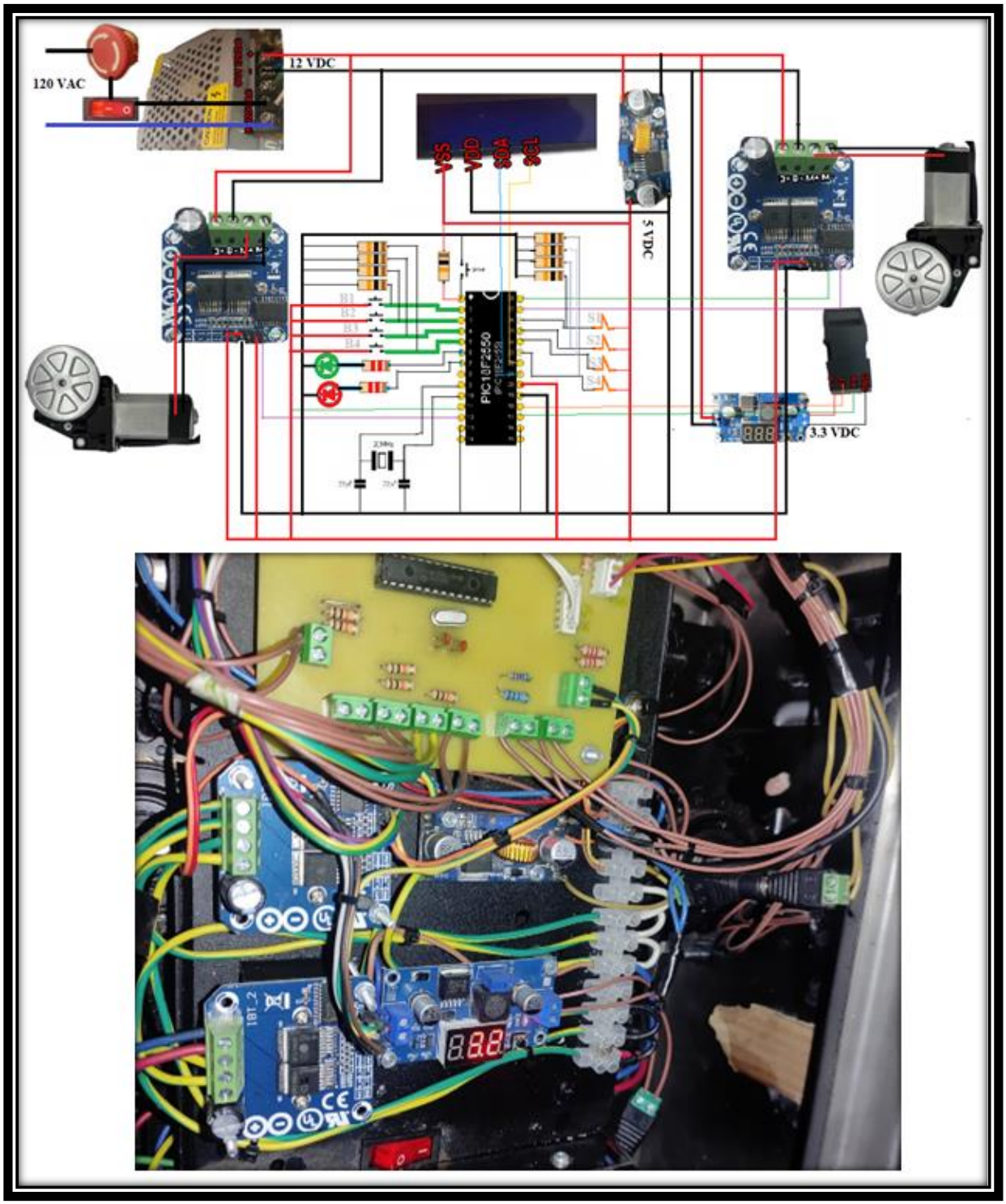


Nota: El sensor dactilar AS608 se conecta de manera externa mediante un módulo serial USB-TTL. (Autoría propia, 2024).

En la Figura 37 se muestra el resultado de la implementación final y la conexión de todos los elementos que conforman el sistema eléctrico, electrónico y control de BIKER POINT y su respectivo diagrama.

Figura 37

Diagrama de interconexión de BIKER POINT



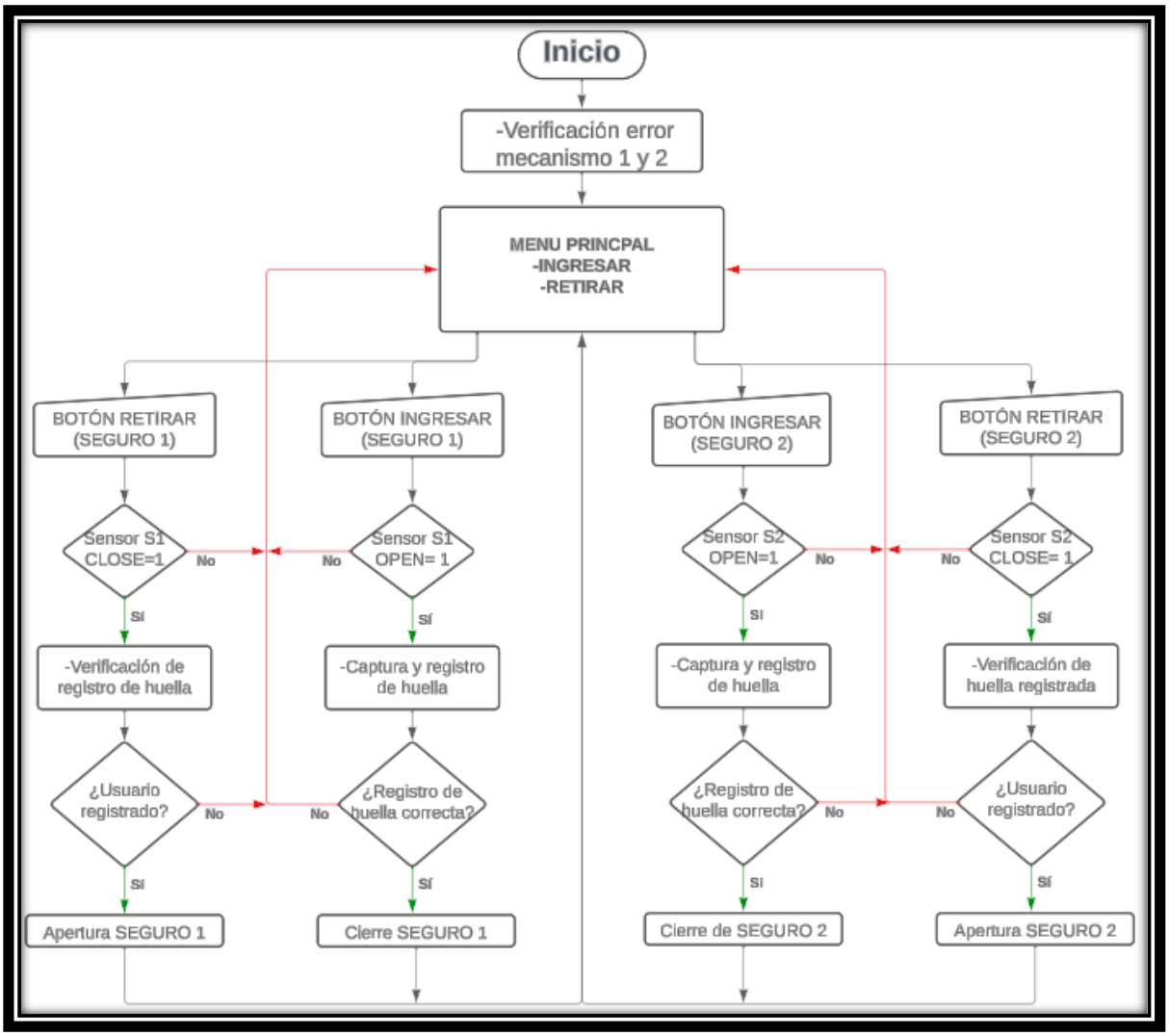
(Autoría propia, 2024)

Implementación del algoritmo de operación

El algoritmo de operación está estructurado e implementado adecuadamente, mostrando resultados eficientes en la operación del sistema BIKER POINT. La secuencia y condiciones de funcionamiento se muestra en la Figura 38.

Figura 38

Algoritmo de operación de BIKER POINT



Nota: La lógica de funcionamiento del sistema BIKER POINT cumple la secuencia y condiciones de este diagrama de flujo. (Autoría propia, 2024).

A continuación de detalla parte del código del algoritmo de operación del sistema BIKER POINT.

La Constante __EEPROM_DATA(1,1,0,0,0xFF,0xFF,0xFF,0xFF) asigna direcciones de la EEPROM para guardar datos.

Se adjunta las librerías de cada uno de los periféricos y programación definidas. En las siguientes líneas se detalla las librerías usadas en el algoritmo de BIKER POINT.

```
#include "i2c.h"                    // Librería del protocolo i2c
#include "eeprom.h"                 // librería de la memoria EEPROM interna
#include "pwm_soft.h"               // librería para generar PWM por software
#include "lcd_i2c.h"                // librería para el control de la pantalla lcd por i2c
#include "fingerprint.h"            // librería estándar de operación con el sensor AS608
```

Definición de variables y constantes a utilizar en el algoritmo.

```
#define MOTOR_DIR_IZQ 0x01      // Constante para el giro de motor hacia la izquierda
#define MOTOR_DIR_DER 0x00      // Constante para el giro de motor hacia la derecha
uint8_t flag_lcd = 1;           // Variable de estado para mostrar el menú de la pantalla lcd
uint8_t cont_fail = 0;          // Contador de intentos para apertura
uint8_t state_user_1 = 0;       // Variable de estado para el mecanismo 1
uint8_t state_user_2 = 0;       // Variable de estado para el mecanismo 2
uint8_t enter_id, c_store;      // Almacena id del lector de huella y estado del lector
```

Con las siguientes líneas se aplica el PWM del 40% de su velocidad nominal a los dos motores, sabiendo que el 100% del ciclo de trabajo equivale a 255.

```
#define DUTY_MOTOR_1 100    // Ciclo de trabajo (100=40%)  para el PWM del motor 1
#define DUTY_MOTOR_2 100    // Ciclo de trabajo (100=40%)  para el PWM del motor 2
```

Declaración y programación de funciones auxiliares.

```
void Mostrar_Menu(void);        // Muestra el menú principal en la LCD
void Verificar_Error(void);     // Verifica si hubo error en la apertura o cierre
```

El algoritmo ocupa de esta función void **Registrar_Usuario**(uint8_t n_user); para capturar la información de la huella dactilar a través del sensor AS608.

```
Lcd_Set_Cursor(4,2);
Lcd_Write_String("Coloque huella");          // solicita al usuario su huella dactilar
Fingerprint_GetImage();                      // Lee la huella detectada
Fingerprint_Image2Tz(1);                     // Genera un archivo digital de la huella
c_store = Fingerprint_StoreModel(n_user);    // Almacena la huella en el arreglo automáticamente
```

Con la siguiente secuencia verifica si la información de la huella dactilar fue almacenada correctamente.

```
if(c_store != FINGERPRINT_OK)
   {
      Lcd_Clear();
      Lcd_Set_Cursor(2,2);                   // Si hubo un error, muestra mensaje en la lcd
      Lcd_Write_String("Error al registrar");
      Lcd_Set_Cursor(2,3);
      Lcd_Write_String("huella en el sistema");
   }
 else                                        // Si el registro fue exitoso “Huella almacenada”
   {
      Lcd_Clear();
      Lcd_Set_Cursor(2,1);
      Lcd_Write_String("Huella almacenada");
      Lcd_Set_Cursor(2,2);
      Lcd_Write_String("correctamente !");
      Lcd_Set_Cursor(2,4);
      Lcd_Write_String("Por favor espere...");
   }
```

La función principal del programa (main) tiene la siguiente secuencia, para la apertura de los mecanismos el algoritmo solicitar identificarse al usuario con su respectiva huella dactilar, toma la lectura y compara con las que se encuentran registradas, si la huella coincide con alguna registrada anteriormente hace la apertura del mecanismo correspondiente caso contrario no procede. En este comando switch(n_user) el argumento es para identificar el número de usuario, en caso de ser el usuario del seguro 1 el algoritmo procede de la siguiente manera en el caso de ser el usuario 2 la misma secuencia, pero se opera el MOTOR 2.

```
switch(n_user)                          // Compara el usuario correspondiente
  {
    case 1:                             // Verifica si es el usuario 1
     LATAbits.LATA4 = 1;                // Enciende el led verde
    Motor_1_On(MOTOR_DIR_DER);          // Motor 1 gira a la derecha
    EEPROM_Write(2, 0x01);
    while(PORTBbits.RB3 != 1);          // Espera a que el sensor 1 "close" detecte
    EEPROM_Write(2, 0x00);
    Motor_1_Off();                      // Apaga el motor 1
    LATAbits.LATA4 = 0;                 // Apaga el led verde
    state_user_1 = 0;
    EEPROM_Write(0, state_user_1);      // Guarda el estado actual del mecanismo 1
    break;
  }
```

Código de la función principal (MAIN) del algoritmo del sistema BIKER POINT.

```
CMCON = 0x07;                           // Apaga los comparadores internos
ADCON1bits.PCFG = 0x0F;                 // Configura todos los pines como digitales
TRISA = 0x0F;                           // RA0 a RA3 entradas, RA4 y RA5 salidas
TRISB = 0x3F;                           // RB0 a RB5 entradas, RB6 y RB7 como salidas
LATAbits.LATA4 = 0;                     // Pin RA4 - led verde
LATAbits.LATA5 = 0;                     // Pin RA5 - led rojo
PWM_Init();                             // Inicializa el PWM por software
PWM_Init_Ports();                       // Inicializa los puertos PWM
I2C_Init(I2C_400KHZ);                   // Inicializa el protocolo i2c
Lcd_Init();                             // Inicializa la pantalla lcd
Lcd_Set_Cursor(6,2);
Lcd_Write_String("BIENVENIDO");         // Muestra un mensaje de bienvenida al encender
Lcd_Set_Cursor(6,3);
Lcd_Write_String("BIKER POINT");
Fingerprint_Init(57600);                // Inicializa el lector de huella dactilar
Fingerprint_Read_Parameters();          // Lee los parámetros del lector de huella dactilar
__delay_ms(3000);
Lcd_Clear();
state_user_1 = EEPROM_Read(0);          // Lee el ultimo estado guardado del mecanismo 1
state_user_2 = EEPROM_Read(1);          // Lee el ultimo estado guardado del mecanismo 2
Verificar_Error();                      // Verifica si hubo un error de estado del sistema
```

El while (true) o while (1) es una secuencia que repite de manera indefinida, parte del algoritmo que se encuentra en ella estará en un bucle siempre y cuando el microcontrolador con el algoritmo grabado se encuentre energizado.

```
while(1)
   {
     Mostrar_Menu();                          // muestra el menú de opciones
   }
```

- Si se presiona el botón “INGRESAR” (SEGURO 1) cierra el mecanismo 1, pero antes registra la huella dactilar del usuario con la función Registrar_Usuario(1);

```
   if(PORTAbits.RA1 == 1 && state_user_1 == 1)
   {
    Registrar_Usuario(1);                     // Llama a la función para registrar el usuario 1
   }
```

- Si se presiona el botón “RETIRAR” (SEGURO 1) Procede con la programación de la función Leer_Usuario(1);

Si ocurre algún inconveniente en la toma de la lectura de la huella dactilar el sistema ofrece una segunda oportunidad al usuario, luego de esto el sistema procede a regresar al menú principal y retoma el algoritmo del MAIN.

```
if(PORTAbits.RA0 == 1 && state_user_1 == 0
   {
     while(cont_fail != 2){          // Verifica si intento acceder 2 veces con la huella incorrecta
     Leer_Usuario(1);                // Lee la huella del usuario 1 para abrir el mecanismo 1
   }
     cont_fail = 0;                  // Reinicia el contador de intentos de acceso
```

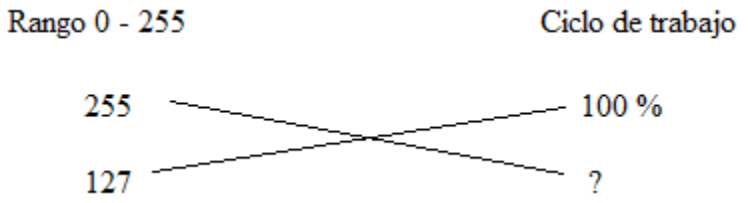
Realización de pruebas de operación del prototipo.

Una vez armado el módulo BIKER POINT fue necesario realizar pruebas de funcionalidad del prototipo con la finalidad de garantizar un funcionamiento óptimo. En esta etapa se encontró los siguientes inconvenientes:

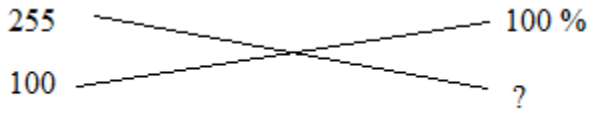
- **Exceso de velocidad de apertura y cierre de SEGURO 1/2, comprometiendo la integridad de los elementos mecánicos y presentando un peligro para los usuarios.**

Dicho problema se corrigió mediante software reduciendo el ciclo de trabajo de las señales PWM en DUTY_MOTOR 1/2 que están relacionadas directamente con la velocidad de los motores. Inicialmente se trabajó con el valor 127 que resulto inapropiado. Se procedió al ensayo con valores menores a ello concluyendo que 100 es el valor óptimo.

Al estar trabajando con señales de 8 bits nuestro rango de valores es de 0 a 255, siendo 0 el equivalente a 0% y 255 al 100% del ciclo de trabajo (Duty cycle).



$$Ct = 127 * \frac{100}{255} = 49{,}8\% \qquad (\text{Ec. } 1)$$

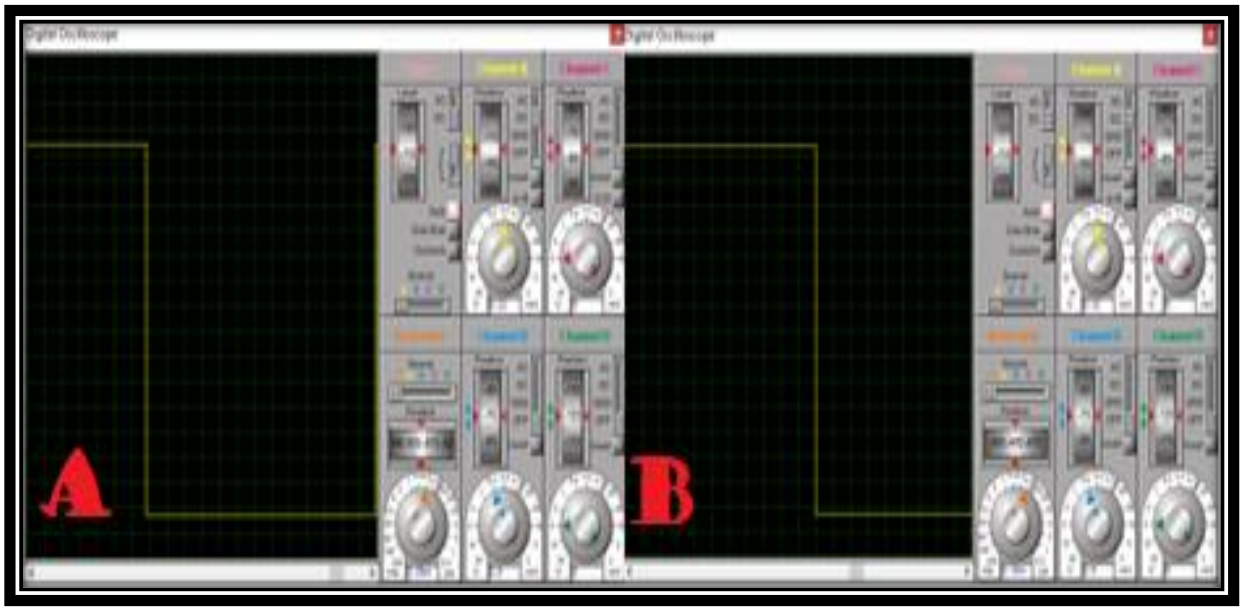


$$Ct = 100 * \frac{100}{255} = 39{,}2\% \qquad (\text{Ec. } 2)$$

```
#define DUTY_MOTOR_1 100          // Ciclo de trabajo para el PWM del motor 1
#define DUTY_MOTOR_2 100          // Ciclo de trabajo para el PWM del motor 2
```

Figura 39

Gráfica ciclos de trabajo



Nota: Literal A ciclo de trabajo del 40% y B ciclo de trabajo al 50%, resultados de la EC. 1 Y EC.2. (Autoría propia, 2024)

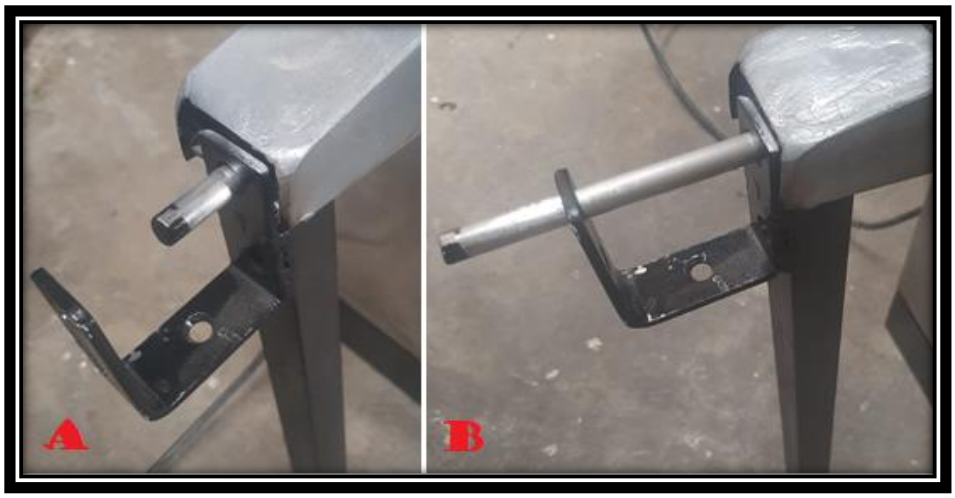
Al estar trabajando con señales de 8 bits nuestro rango de valores es de 0 a 255, siendo 0 el equivalente a 0% y 255 al 100% de la variable a considerar en este caso la velocidad del motor.

```
void Motor_1_On(uint8_t dir)                    // Funcion para mover el motor 1
{
  if(dir == 1){                                 // Motor 1 gira a la izquierda
    PWM_CH0_Duty(DUTY_MOTOR_1);
    PWM_CH1_Duty(0);
  }else{                                        // Motor 1 gira a la derecha
    PWM_CH0_Duty(0);
    PWM_CH1_Duty(DUTY_MOTOR_1);
  }
}
```

- **El desplazamiento de los vástagos excedía los límites de cierre o no se abría por completo. Este problema se evidencia en la Figura 40.**

Figura 40

Cierre y apertura de seguros Incorrecto

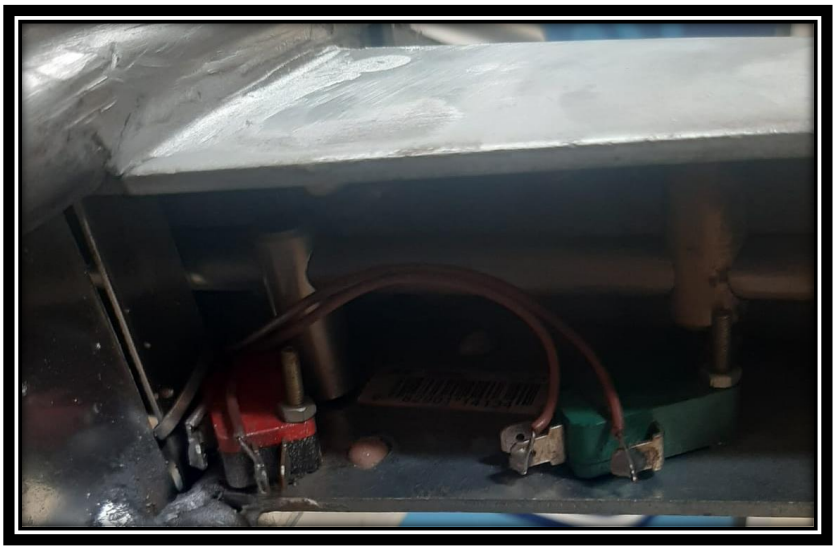


Nota: Al no activarse correctamente los sensores el desplazamiento de los seguros ocasiona este problema. (Autoría propia, 2024)

Se calibro el recorrido de la pieza metálica que activa los sensores finales de carrera para una activación oportuna al momento de cierre y apertura de los seguros. También se reubico los sensores para tener un rango de desplazamiento necesario de los vástagos. La ubicación final se muestra en la Figura 41. Obteniendo como respuesta una excelente operación mecánica de los seguros, esto se demuestra en la Figura 42.

Figura 41

Sensores cierre y apertura de los seguros



Nota: Sensores que indican si el seguro esta en estado abierto o cerrado.(Autoría propia, 2024)

Figura 42

Estado de operación de seguros

Nota: Problema de cierre y apertura solucionado. (Autoría propia, 2024).

El sistema mecánico se quedaba inoperativo al encender BIKER POINT luego de un apagado imprevisto por corte de energía. Este problema se presentaba si el sistema se apagó justo en el transcurso de cierre o apertura de los seguros.

En la Figura 43 se evidencia el problema mencionado.

Figura 43

Seguro 2 a medio cerrar

Nota: La maquina encendida con el sistema mecanico sin posibilidad de operación. (Autoría propia, 2024)

Se implementó la función **"Verificar_Error"** que supervisa errores en el proceso del cierre o apertura de los seguros. La información del último estado de cada seguro se almacena en la memoria EEPROM.

- Si el ERROR ocurre a medio cerrar SEGURO 1/2, el ultimo estado guardado en la memoria EEPROM fue SEGURO 1/2 abierto. Por lo cual al encender BIKER POINT SEGURO 1/2 regresa al estado abierto y se reinicia el registro de la huella.

- Si el ERROR ocurre en el proceso de apertura de SEGURO 1/2, el ultimo estado guardado en la memoria EEPROM fue SEGURO 1/2 cerrado. Por lo cual al encender BIKER POINT SEGURO 1/2 regresa al estado cerrado y el registro de la huella se mantiene.

Cabe recalcar que la lógica de programación y condiciones de operación son equivalentes para el SEGURO 2 y el SEGURO 1. A continuación se detalla el código de la función encargada de garantizar la operatividad del sistema mecánico a nivel de software.

```
void Verificar_Error(void)
{
  if(EEPROM_Read(2) == 0x01){                  // Verifica si hubo error al cerrar SEGURO 1
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("Hubo un error al");
    Lcd_Set_Cursor(1,2);
    Lcd_Write_String("cerrar mecanismo 1");
    Lcd_Set_Cursor(1,4);
    Lcd_Write_String("Por favor espere...");
    Motor_1_On(MOTOR_DIR_IZQ);          // Motor 1 gira a la izquierda
    while(PORTBbits.RB2 != 1);                 // Espera a que el sensor 1 "open" detecte
    Motor_1_Off();                             // Apaga el motor 1
    EEPROM_Write(2, 0x00);                     // Elimina el error del mecanismo 1
    Lcd_Clear();
    __delay_ms(500);
  }
  else if(EEPROM_Read(2) == 0x02){          // Verifica si hubo abrir SEGURO 1
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("Hubo un error al");
    Lcd_Set_Cursor(1,2);
    Lcd_Write_String("abrir mecanismo 1");
    Lcd_Set_Cursor(1,4);
    Lcd_Write_String("Por favor espere...");
    Motor_1_On(MOTOR_DIR_DER);                 // Motor 1 gira a la derecha
    while(PORTBbits.RB3 != 1);                 // Espera a que el sensor 1 "close" detecte
    Motor_1_Off();                             // Apaga el motor 1
```

```
        EEPROM_Write(2, 0x00);                      // Elimina el error del mecanismo 1
        Lcd_Clear();
        __delay_ms(500);
    }
else if(EEPROM_Read(3) == 0x01){
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Hubo un error al");
        Lcd_Set_Cursor(1,2);
        Lcd_Write_String("cerrar mecanismo 2");
        Lcd_Set_Cursor(1,4);
        Lcd_Write_String("Por favor espere...");
        Motor_2_On(MOTOR_DIR_IZQ);              // Motor 2 gira a la izquierda
        while(PORTBbits.RB4 != 1);              // Espera a que el sensor 2 "open" detecte
        Motor_2_Off();                          // Apaga el motor 2
        EEPROM_Write(3, 0x00);                  // Elimina el error del mecanismo 2
        Lcd_Clear();
        __delay_ms(500);
    }
    else if(EEPROM_Read(3) == 0x02){
        Lcd_Clear();
        Lcd_Set_Cursor(1,1);
        Lcd_Write_String("Hubo un error al");
        Lcd_Set_Cursor(1,2);
        Lcd_Write_String("abrir mecanismo 2");
        Lcd_Set_Cursor(1,4);
        Lcd_Write_String("Por favor espere...");
        Motor_2_On(MOTOR_DIR_DER);              // Motor 2 gira a la derecha
        while(PORTBbits.RB5 != 1);              // Espera a que el sensor 2 "close" detecte
        Motor_2_Off();                          // Apaga el motor 2
        EEPROM_Write(3, 0x00);                  // Elimina el error del mecanismo 2
        Lcd_Clear();
        __delay_ms(500);
    }
```

Además, el sistema dispone de un botón PARO DE EMERGENCIA ante cualquier eventualidad que podría ocurrir mediante el uso de BIKER POINT por posibles descuidos o negligencia de los usuarios como el mostrado en la Figura 44. No debe obstruir la línea de desplazamiento de los vástagos y prestar atención si hay niños cerca de la maquina al momento de su operación.

Figura 44

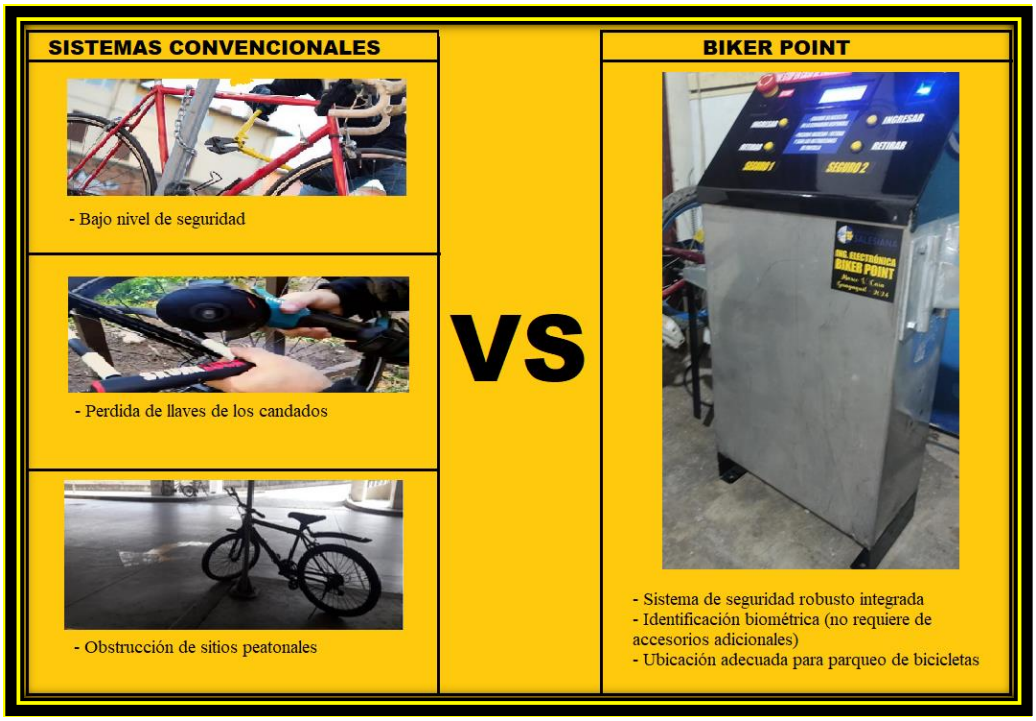
Problema de seguridad

Nota: No se debe obstruir el sistema de esta manera ni algun tipo de objeto, tener precaucion si va acompañado de niños. (Autoría propia, 2024)

En la Figura 45 se muestra las ventajas alcanzadas frente a los problemas que presentan los sistemas de aseguramiento de bicicletas convencionales.

Figura 45

Ventajas del sistema BIKER POINT



Nota: Se menciona las principales diferentes entre los sistemas de parqueo de bicicletas. (Autoría propia, 2024)

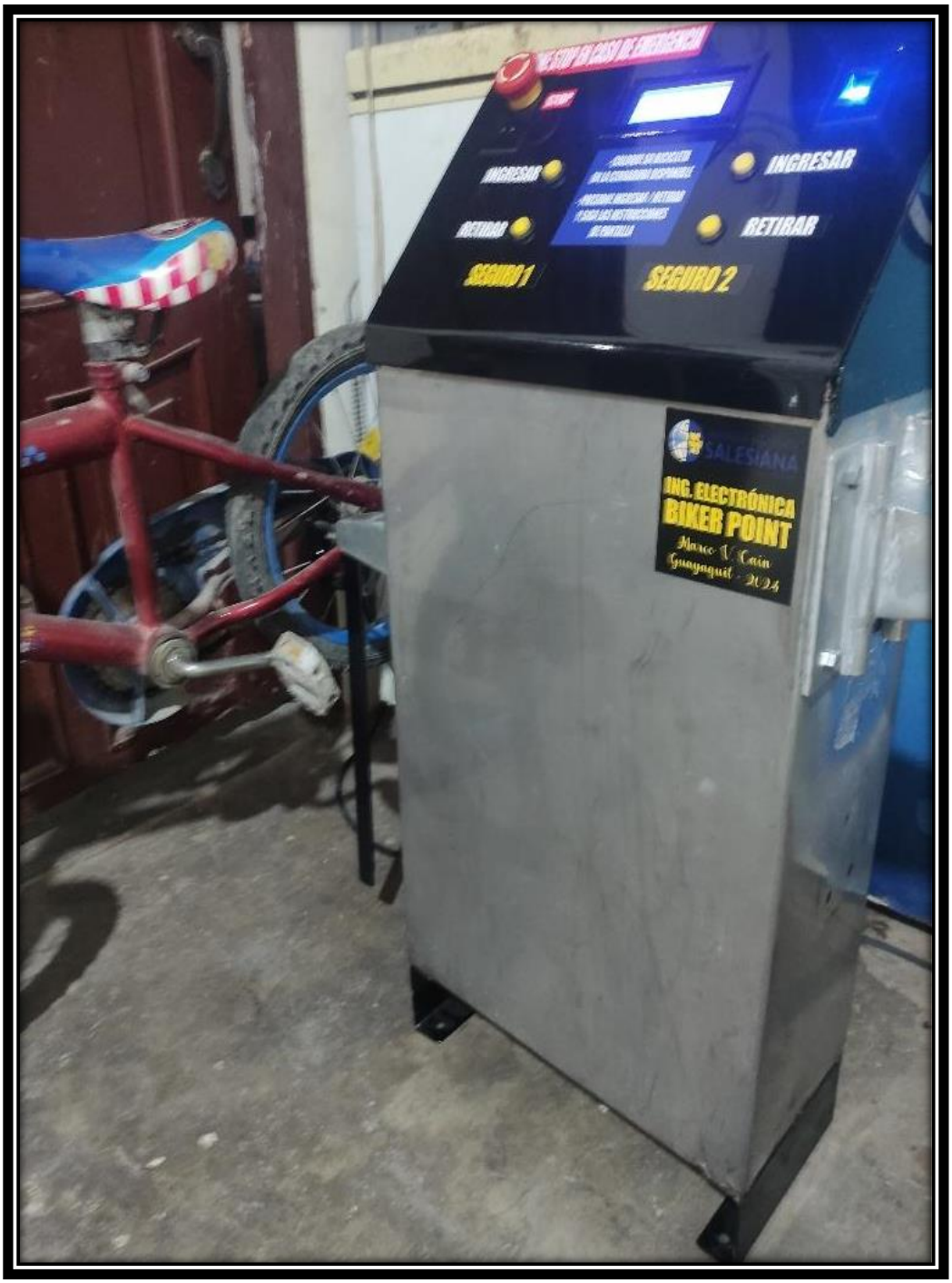
Operatividad de BIKER POINT

Todos los inconvenientes encontrados fueron corregidos exitosamente, hasta obtener el funcionamiento optimo del sistema. El resultado de estas evaluaciones permitió la mejora y la optimización de la funcionalidad del sistema y también proveer seguridad al usuario. Resultado final BIKER POINT 100% funcional, listo para poner al servicio de la población.

El resultado final se muestra en la Figura 46.

Figura 46

BIKER POINT



Nota: Módulo de BIKER POINT en estado de operación. (Autoría propia, 2024).

A continuación, se detallan los pasos para la utilización del BIKER POINT por parte de los usuarios:

- **Procedimiento ingreso bicicleta**

1. Colocar la Bicicleta en uno de los seguros de la forma indicada en la Figura 47
2. Presionar botón “INGRESAR” (SEGURO 1/2)
3. Seguir instrucciones de pantalla, cuya secuencia se muestra en la Figura 48

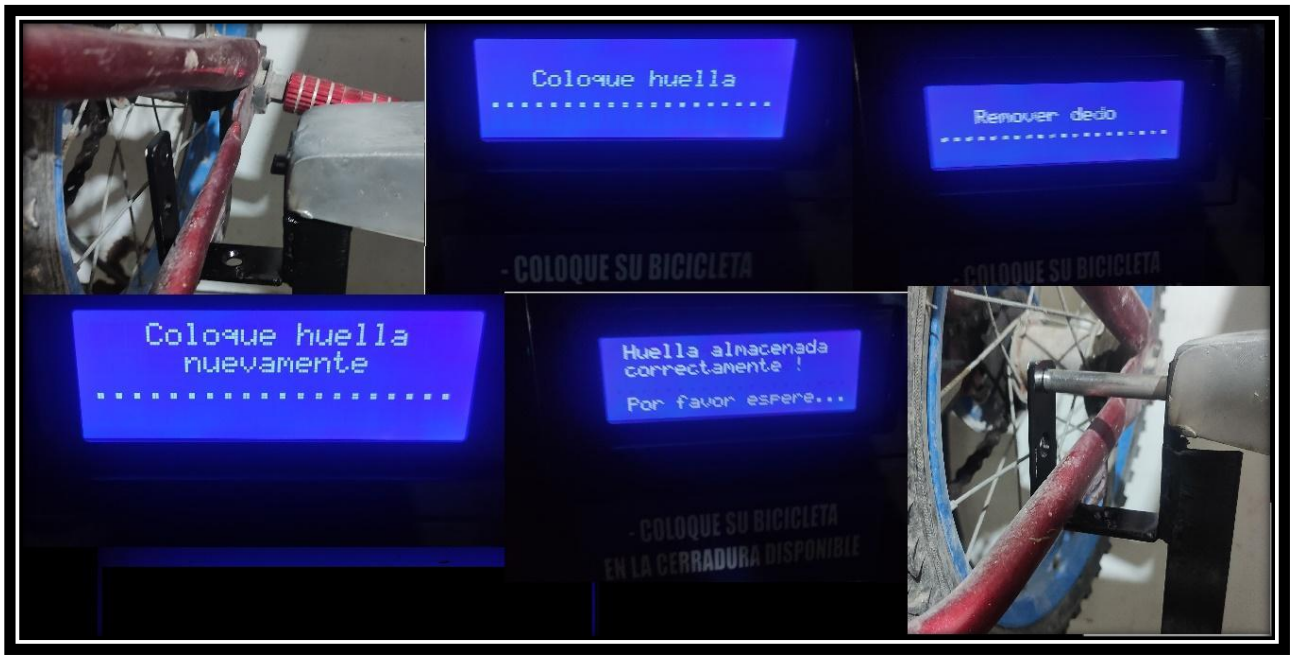
Figura 47

Colocación de la Bicicleta

Nota: Forma correcta de ubicar la unidad en el seguro. (Autoría propia, 2024)

Figura 48

Secuencia Ingreso de bicicleta



Nota: Siga los pasos indicados en la pantalla LCD para el ingreso correcto de bicicleta en el sistema BIKER POINT. (Autoría propia, 2024).

Luego de tomar el registro correspondiente y grabarlo en una dirección del registro, el mecanismo se cierra automáticamente. En caso de ocurrir algún accidente con las cerraduras del sistema, el usuario tiene el acceso al botón de emergencia en el panel “STOP”.

- **Procedimiento retirar bicicleta**

1. Presione botón “RETIRAR” (SEGURO 1/2) del seguro correspondiente en la que se encuentra su bicicleta,
2. El sistema pedirá verificar su huella digital registrada.
    - Si su huella dactilar es la registrada, el mecanismo hace su apertura de manera automática.

- Si el sistema no lee correctamente la huella o si la misma no está registrada, el sistema pide verificar la huella por segunda ocasión.
- Si en la segunda lectura el sistema verifica que su huella digital registrada es correcta. El mecanismo libera su bicicleta. Caso contrario, el sistema da una advertencia "Usuario no identificado" e impide el retiro de la bicicleta como se muestra en la Figura 49.

Figura 49

Validación de huella registrada

Nota: Indicación de una validación de huella correcta. (Autoría propia, 2024).

Figura 50

Retiro de bicicleta denegado

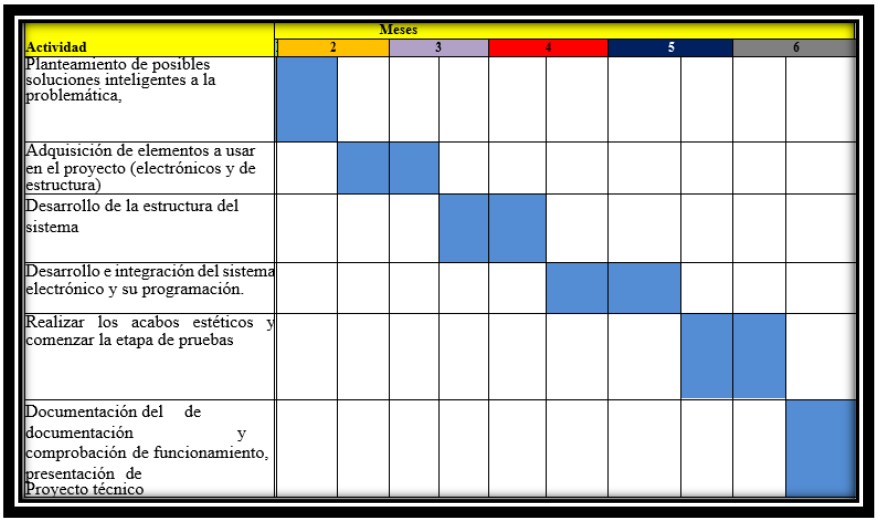
Nota: El sistema deniega siempre y cuando intente retirar con una identificación biométrica falsa. (Autoría propia, 2024).

El sistema restringe el retiro de la bicicleta si la huella dactilar que registra no está ingresada tal como se evidencia en la Figura 50, evitando así un acto de urto de la unidad.

CRONOGRAMA

Figura 51

Cronograma de desarrollo de BIKER POINT



PRESUPUESTO

Figura 52

Presupuesto desarrollo de BIKER POINT

Descripción	Cantidad	Costo unitario	Costo final
PIC18F4550	1	$ 10.00	$ 10.00
Motores	2	$ 20,00	$ 40,00
LCD		$ 15,00	$ 15,00
Driver de potencia para motor	2	$ 10,00	$ 10,00
Rodamientos	2	$ 10,00	$ 20,00
Barra acero	2	$ 5,00	$ 10,00
Estructura metálica	1	$ 150,00	$ 150,00
botones	1	$ 0,50	$ 2,00
Lector de huella	1	$ 25,00	$ 25,00
Fuente de voltaje	1	$ 15,00	$ 15,00
leds	2	$ 0,25	$ 0,50
Mano de obra		$ 200,00	$ 200,00
Rueda dentada	2	$ 5,00	$ 10,00
Placa PCB	1	$ 15,00	$ 15,00
Quemador de pic	1	$ 20,00	$ 20,00
Total			**$732,50**

CONCLUSIONES

El desarrollo de este prototipo BIKER POINT-SISTEMA DE ESTACIONAMIENTO PARA BICICLETAS UTILIZANDO EL MICROCONTROLADOR PIC18F. En base a los objetivos planteados al principio del proyecto, se llega a la conclusión de haber desarrollado con éxito el sistema, mejorando o replanteando cada etapa de su desarrollo.

- La operación del sistema es adecuada, pero, las dimensiones son un poco mayores de lo previsto, aumentado el volumen de la máquina.
- En el diseño electrónico y la parte de control se desempeña de manera adecuada cumpliendo con los objetivos y las expectativas planteadas al comiendo del proyecto.
- El microcontrolador PIC18F2550 se eligió por las características técnicas brindadas, además, se ajustó perfectamente a la cantidad de recursos que se requirieron para nuestro desarrolló.
- La estructura o cubierta del sistema se construyó en material acero inoxidable para brindar protección adecuada los elementos electrónicos que se encuentran en su interior, además, para evitar el deterioro del prototipo por humedad y corrosión.
- Se realizó las pruebas y ensayos necesarios del circuito electrónico con la programación en circuitos de pruebas con el objetivo de reajustar diseños y parámetros, así garantizar correcta operación del prototipo.

RECOMENDACIONES

BIKER POINT al ser un sistema desarrollado en primera instancia por ende está sujeto a realizar mejoras y complementos para brindar a la sociedad un sistema mucho más fiable de menores dimensiones y costos.

- No colocar elementos, obstáculos que obstruyan el libre recorrido de los vástagos del sistema mecánico. El uso del sistema por niños debe ser supervisado por un adulto.
- Se recomienda mantenimiento preventivo con regularidad del sistema eléctrico, electrónico y mecánico del sistema (2 vez por año).
- Mantener limpia el panel de control de manera especialmente la ventana de lectura del sensor AS608.
- No forjar la máquina por ningún caso, no golpear, seguir las instrucciones indicadas por el algoritmo de operación mediante la LCD.
- Si no se va a usar por un largo periodo desconectar el sistema (por las noches)
- La alimentación adecuada para el sistema es de 110-120VAC (no alimentar con 240VAC).
- En caso de observar daños de algún tipo en el sistema dar aviso al personal responsable.
- Tener los dedos libres de humedad y grasas para no tener inconvenientes con el uso de BIKER POINT.
- Revisar el correcto funcionamiento del sensor, caso contrario el sistema es operativo.

REFERENCIAS BIBLIOGRÁFICAS

ABB. (2019). Emergency stop.

Adafruit Industries. (2024). Adafruit Optical Fingerprint Sensor.

Alamy. (s f). Transmisión de cadena de piñón Vectores y arte vectorial [Imagen]. Obtenido de https://www.alamy.es/rueda-dentada-con-cadenas-de-eslabones-redondos-y-rueda-dentada-ilustracion-vectorial-image401792897.html?imageid=6CE08A44-6AEA-44DC-AEAE-1C1D97354603&p=464219&pn=1&searchId=63b28fb7bd7f953633932883216a9047&searchtype=0

Artal-Sevil, R. D.-L.-A. (2018). Analysis of different PWM Modulation Techniques: Comparison and Design.

Artero, Ó. T. (2019). Arduinno Curso práctico de formación. España: RC Libros.

BETAFIX S.A. (2023). Motor Elevavidrios Universal 7 Dientes 3 Patas 12v [Imagen].

Buriticá, O. I. (2017). PROGRAMACIÓN IMPERATIVA CON LENGUAJE C. Mexico.

Chávez, J. D. (2019). Fundamentos de Programación en Lenguaje C.

Circuits Geeks. (21 de 1 de 2022). Pulse Width Modulation [Imagen]. Obtenido de https://www.circuitgeeks.com/arduino-pwm/

Cosaar, R. C. (1984). NTP 86: Dispositivos de parada de emergencia.

DigitalPaper. (2016). Fuente Conmutada 12V 5A [Imagen]. Obtenido de https://ja-bots.com/producto/fuente-conmutada-12v-5a/

expreso. (2022). El robo de bicis, un delito que está al alza. Obtenido de https://www.expreso.ec/actualidad/robo-bicis-delito-alza-94154.html

Francisco Ernesto Cortez, E. A. (2021). PROPUESTA DE DISEÑO DE SISTEMA INALÁMBRICO PARA EL CONTROL DE DISPOSITIVOS ELÉCTRICOS Y ELECTRÓNICOS DEL AUTOMÓVIL UTILIZANDO TECNOLOGÍA BLUETOOTH [Imagen].

García, H. (2015). Firma electrónica. Ecuador.

Garzón Felipe, R. C. (2017). SISTEMA DE SEGURIDAD Y REGISTRO POR MEDIO DE ACCESO RFID PARA. Bogotá.

Guerrero Camilo, R. Á. (2020). BIKER-CAMP & CLOUD COMPUTING, UNA PROPUESTA DE DESARROLLO DE. Colombia.

Ingeniería Mecafenix . (26 de 05 de 2021). limit switch. Obtenido de https://www.ingmecafenix.com/automatizacion/sensores/final-de-carrera/

ITMadrid Digital School. (06 de 02 de 2020). Qué es y para qué sirve Design Thinking. Obtenido de https://www.itmadrid.com/que-es-y-para-que-sirve-design-thinking/

Loreto, A. (2015). Sensores y Actuadores.

Lucechetti, S. (2021). Sistemas Embebidos y sus características.

Luis Gil Sánchez, J. I. (2018). PROBLEMAS DE ELECTRÓNICA DIGITAL. Valencia.

mecatrónicalatam.com. (23 de 04 de 2021). Diodo LED. Obtenido de https://www.mecatronicalatam.com/es/tutoriales/electronica/componentes-electronicos/diodo/diodo-led/

Microchip technollogy Inc. (2006). PIC18F2455/2550/4455/4550.

Microchip Technollogy Inc. (2023). PIC18F2550 [Imagen]. Obtenido de https://www.microchip.com/en-us/product/pic18f2550

Mosquera, R. B. (2018). Arduino Prototipado y programación avanzada. Marcombo.

Mvtronic. (2016). 43A High Power Motor Driver [Imagen]. Obtenido de https://www.mpja.com/download/37225MD%20SchA.pdf

Ocampo C., G. L. (2020). Automatismos eléctricos. Colombia.

Omar Enrique Barra Zapata, F. B. (2018). Microcontroladores PIC con programación PBP.

Ortega, S. (10 de 09 de 2020). Diseño de módulos entrada y salida para uso industrial basado en comunicación I2C. España.

Quintero, N. B. (11 de 2019). Integración HMI-PLC para un filtro de agua con aplicación hidropónica. Obtenido de Texas instruments: https://www.ti.com/lit/ds/symlink/lm2596.pdf?ts=1706165305129

Rajguru Electronics (I) Pvt. Ltd. (2018). AS608 Fingerprint Reader Sensor Module With Cable [Imagen].

REALINCE FOUNDRY. (2016). Robo de Bicicletas: Contraatacando estrategias policiales y organizacionales hacia un ciclismo más seguro [Imagen]. Obtenido de https://www.reliance-foundry.com/blog/vigilacia-policial-robo-bicicletas-es

Sixto Reinoso, L. M. (2018). PROGRAMACIÓN DE MICROCONTROLADORES PIC CON LENGUAJE C. Zaragoza.

Soler, C. S. (2020). DISEÑO MECÁNICO PARA LA ADAPTACIÓN A PROPULSIÓN ELÉCTRICA DE UN KART-CROSS. Castellon.

Tokheim, R. L. (2021). ELECTRÓNICA DIGITAL.

Universidad Politécnica de Madrid. (2018). IBT-2 HIGH CURRENT 30A DC MOTOR DRIVER. Madrid. Obtenido de https://www.amazon.com/WWZMDiB-BTS7960-Driver-Arduino-Current/dp/B0BV925J5W/ref=sr_1_2?crid=I0NH1JV0GDJ6&keywords=MODULO+IBT_2&qid=1706198044&sprefix=modulo+ibt_2%2Caps%2C153&sr=8-2

Welivesegurity. (16 de 11 de 2018). Seguridad Digital [Imagen]. Obtenido de https://www.welivesecurity.com/la-es/2018/11/16/demuestran-que-es-posible-vulnerar-lectores-de-huella-digital-mediante-huellas-maestras/

WINSTAR Display Co.,Ltd. (s.f.). MODULE NO.: WH2004D.

ANEXOS

Programas utilizados

- Proteus 8 Profesional: Diseño de tarjeta PCB y simulación de circuito electrónico.
- MPLAB X IDE V6.15: Programación del algoritmo de operación en Lenguaje C.
- XC8 COMPILER: Compilación del código fuente a lenguaje máquina.
- PICKIT3: Grabado del firmware en el microcontrolador.
- WORD: Redacción de la memoria del proyecto.

Trabajos complementarios

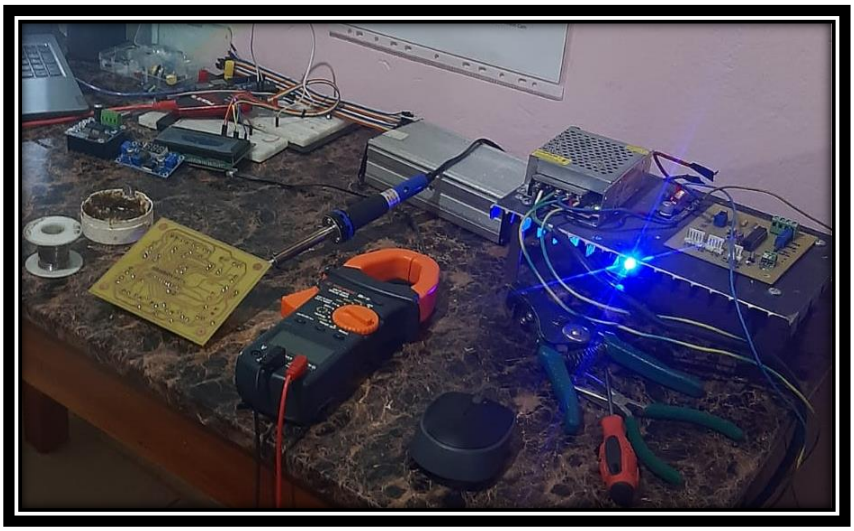
Figura 53

Soldadura estructura del gabinete

(Autoría propia, 2024).

Figura 54

Uso de herramientas de emsamblaje y pruebas



(Autoría propia, 2024).

Código Fuente del sistema BIKER POINT

CÓDIGO DE LIBRERÍAS

- **Librería EEPROM (eeprom.h)**

```
#ifndef EEPROM_H
#define EEPROM_H
void EEPROM_Write(uint8_t dir, uint8_t data);
uint8_t EEPROM_Read(uint8_t dir);
#endif
```

- **Librería del sensor dactilar AS608 (fingerprinth.h)**

```
#ifndef FINGERPRINT_H
#define FINGERPRINT_H
#define FINGERPRINT_OK 0x00
#define FINGERPRINT_PACKETRECIEVEERR 0x01
#define FINGERPRINT_IMAGEMESS 0x06
#define FINGERPRINT_NOTFOUND 0x09
#define FINGERPRINT_ENROLLMISMATCH 0x0A
#define FINGERPRINT_BADLOCATION 0x0B
#define FINGERPRINT_DELETEFAIL 0x10
#define FINGERPRINT_DBCLEARFAIL 0x11
#define FINGERPRINT_FLASHERR 0x18
void Fingerprint_Init(unsigned long baud);
```

```
void Fingerprint_Send_Buffer(uint8_t* buf, uint8_t size);
void Fingerprint_Read_Parameters(void);
void Fingerprint_GetImage(void);
void Fingerprint_Image2Tz(uint8_t n);
void Fingerprint_Search(void);
void Fingerprint_GetModel(void);
uint8_t Fingerprint_StoreModel(uint8_t id);
uint8_t Fingerprint_DeleteModel(uint8_t id);
uint8_t Fingerprint_GetTemplateCount(void);
uint8_t Fingerprint_EmptyDatabase(void);
uint16_t Fingerprint_Get_ID(void);
#endif
```

- **Librería de comunicación I2C (i2c.h)**

```
#ifndef I2C_H
#define          I2C_H
#define TRIS_SCL TRISBbits.RB1
#define TRIS_SDA TRISBbits.RB0
#define I2C_100KHZ 0x80
#define I2C_400KHZ 0x00
void I2C_Init(uint8_t sp_i2c);
void I2C_Start(void);
void I2C_Stop(void);
void I2C_Restart(void);
void I2C_Ack(void);
void I2C_Nack(void);
uint8_t I2C_Read(void);
uint8_t I2C_Write(uint8_t data);
#endif
```

- **Librería PWM por software (pwm.h)**

```
#ifndef PWM_SOFT_H
#define PWM_SOFT_H
#define PWM_CHANNEL_0  LATBbits.LATB6
#define PWM_CHANNEL_0_DIR   TRISBbits.TRISB6
#define PWM_CHANNEL_1  LATBbits.LATB7
#define PMW_CHANNEL_1_DIR   TRISBbits.TRISB7
#define PWM_CHANNEL_2  LATCbits.LATC0
#define PWM_CHANNEL_2_DIR   TRISCbits.TRISC0
#define PWM_CHANNEL_3 LATCbits.LATC1
#define PMW_CHANNEL_3_DIR   TRISCbits.TRISC1
#define PWM_SOFTWARE
void PWM_Init(void);
void PWM_Init_Ports(void);
#ifdef PWM_CHANNEL_0
void PWM_CH0_Duty(int duty);
```

```
#endif
#ifdef PWM_CHANNEL_1
void PWM_CH1_Duty(int duty);
#endif

#ifdef PWM_CHANNEL_2
void PWM_CH2_Duty(int duty);
#endif
#ifdef PWM_CHANNEL_3
void PWM_CH3_Duty(int duty);
#endif
#ifdef PWM_CHANNEL_4
void PWM_CH4_Duty(int duty);
#endif
#ifdef PWM_PORTB
void PWM_PORTB_Duty(int duty);
#endif
#ifdef PWM_PORTD
void PWM_PORTD_Duty(int duty);
#endif
```

- **Librería de LCD 20X4 (lcd_i2c.h)**

```
#ifndef LCD_I2C_H
#define LCD_I2C_H
#define ADDRESS_LCD 0x4E
void Lcd_Init(void);
void Lcd_Cmd(unsigned char cmd);
void Lcd_Set_Cursor(char col, char row);
void Lcd_Write_Char(char c);
void Lcd_Write_String(const char *str);
void Lcd_Clear(void);
void Lcd_Shift_Right(void);
void Lcd_Shift_Left(void);
void Lcd_Blink(void);
void Lcd_NoBlink(void);
void Lcd_CGRAM_WriteChar(char n);
void Lcd_CGRAM_CreateChar(char pos, const char* new_char);
#endif
```

CÓDIGO DE FUNCIONES AUXILIARES

- **Función EEPROM (eeprom.c)**

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <stdint.h>
```

```
#include "eeprom.h"

void EEPROM_Write(uint8_t dir, uint8_t data)
{
   EEADR = dir;
   EEDATA = data;
   EECON1bits.EEPGD = 0;
   EECON1bits.CFGS = 0;
   EECON1bits.WREN = 1;
   INTCONbits.GIE = 0;
   EECON2 = 0x55;
   EECON2 = 0x0AA;
   EECON1bits.WR = 1;
   INTCONbits.GIE = 1;
   while(!PIR2bits.EEIF);
   PIR2bits.EEIF = 0;
   EECON1bits.WREN = 0;
}
uint8_t EEPROM_Read(uint8_t dir)
{
   EEADR = dir;
   EECON1bits.EEPGD = 0;
   EECON1bits.CFGS = 0;
   EECON1bits.RD = 1;
   return EEDATA;
}
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <stdint.h>
#include "eeprom.h"
void EEPROM_Write(uint8_t dir, uint8_t data)
{
   EEADR = dir;
   EEDATA = data;
   EECON1bits.EEPGD = 0;
   EECON1bits.CFGS = 0;
   EECON1bits.WREN = 1;
   INTCONbits.GIE = 0;
   EECON2 = 0x55;
   EECON2 = 0x0AA;
   EECON1bits.WR = 1;
   INTCONbits.GIE = 1;
   while(!PIR2bits.EEIF);
   PIR2bits.EEIF = 0;
   EECON1bits.WREN = 0;
}
```

```
uint8_t EEPROM_Read(uint8_t dir)
{
   EEADR = dir;
   EECON1bits.EEPGD = 0;
   EECON1bits.CFGS = 0;
   EECON1bits.RD = 1;
   return EEDATA;
}
```

- **Función de operación del sensor (fingerprinth.c)**

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <stdint.h>
#include "fingerprint.h"
uint8_t cont_rx = 0;
uint8_t request_fingerprint[28];
uint8_t finger_size_l;
uint16_t checksum;
const uint8_t finger_read_data[12] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x03,0x0F,0x00,0x13};
const uint8_t finger_detect[12] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x03,0x01,0x00,0x05};
const uint8_t finger_img2tz_1[13] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x04,0x02,0x01,0x00,0x08};
const uint8_t finger_img2tz_2[13] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x04,0x02,0x02,0x00,0x09};
const uint8_t finger_get_model[12] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x03,0x05,0x00,0x09};
const uint8_t finger_template[12] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x03,0x1D,0x00,0x21};
const uint8_t finger_delete_all[12] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x03,0x0D,0x00,0x11};
uint8_t finger_sh[17] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x08,0x04,0x01,0x00,0x00,0x00,0x00,0x00,0x00
};
uint8_t finger_store_model[15] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x06,0x06,0x01,0x00,0x00,0x00,0x00};
uint8_t finger_delete[16] =
{0xEF,0x01,0xFF,0xFF,0xFF,0xFF,0x01,0x00,0x07,0x0C,0x00,0x00,0x00,0x01,0x00,0x00};
void Fingerprint_Init(unsigned long baud)
{
 unsigned int vx;
   TRISCbits.RC6 = 0;
    TRISCbits.RC7 = 1;
    TXSTA = 0x24;
    RCSTA = 0x90;
```

```
    BAUDCON = 0x00;
    BAUDCONbits.BRG16 = 1;
    vx = (unsigned int)(_XTAL_FREQ/(baud*4))-1;
    SPBRG = vx & 0x00FF;
    SPBRGH = vx >> 8;
    PIE1bits.RCIE = 1;
    PIR1bits.RCIF = 0;
    INTCONbits.PEIE = 1;
    INTCONbits.GIE = 1;
    __delay_ms(1000);
}
void Fingerprint_Send_Buffer(uint8_t* buf, uint8_t size)
{
    for(uint8_t i=0; i<size; i++)
    {
        while(TXSTAbits.TRMT == 0);
        TXREG = *buf++;
    }
}
void Fingerprint_Read_Parameters(void)
{
    Fingerprint_Send_Buffer((uint8_t*)finger_read_data, 12);
    cont_rx = 0;
    while(cont_rx != 27);
    __delay_us(300);
    finger_size_l = request_fingerprint[15];
    while(request_fingerprint[9] != FINGERPRINT_OK);
}
void Fingerprint_GetImage(void)
{
    do{
        Fingerprint_Send_Buffer((uint8_t*)finger_detect, 12);
        cont_rx = 0;
        while(cont_rx != 11);
        __delay_us(300);
    }while(request_fingerprint[9] != FINGERPRINT_OK);
}
void Fingerprint_Image2Tz(uint8_t n)
{
    switch(n)
    {
        case 1:
            Fingerprint_Send_Buffer((uint8_t*)finger_img2tz_1, 13);
            break;

        case 2:
```

```
            Fingerprint_Send_Buffer((uint8_t*)finger_img2tz_2, 13);
            break;
    }
    cont_rx = 0;
    while(cont_rx != 11);
    __delay_us(300);
    while(request_fingerprint[9] != FINGERPRINT_OK);
}

void Fingerprint_Search(void)
{
    checksum = 0;
    finger_sh[14] = finger_size_l;
    checksum = (uint16_t)(14 + finger_sh[14]);
    finger_sh[15] = (uint8_t)(checksum >> 8);
    finger_sh[16] = (uint8_t)(checksum & 0xFF);
    Fingerprint_Send_Buffer((uint8_t*)finger_sh, 17);
    cont_rx = 0;
    while(cont_rx != 15);
    __delay_us(300);
}

void Fingerprint_GetModel(void)
{
    Fingerprint_Send_Buffer((uint8_t*)finger_get_model, 12);
    cont_rx = 0;
    while(cont_rx != 11);
    __delay_us(300);
    while(request_fingerprint[9] != FINGERPRINT_OK);
}

uint8_t Fingerprint_StoreModel(uint8_t id)
{
    checksum = 0;
    finger_store_model[12] = id;
    checksum = (uint16_t)(14 + finger_store_model[12]);
    finger_store_model[13] = (uint8_t)(checksum >> 8);
    finger_store_model[14] = (uint8_t)(checksum & 0xFF);
    Fingerprint_Send_Buffer((uint8_t*)finger_store_model, 15);
    cont_rx = 0;
    while(cont_rx != 11);
    __delay_us(300);
    while(request_fingerprint[9] != FINGERPRINT_OK);
    return request_fingerprint[9];
}
```

```
uint8_t Fingerprint_DeleteModel(uint8_t id)
{
   checksum = 0;
   finger_delete[11] = id;
   checksum = (uint16_t)(21 + finger_delete[11]);
   finger_delete[14] = (uint8_t)(checksum >> 8);
   finger_delete[15] = (uint8_t)(checksum & 0xFF);
   Fingerprint_Send_Buffer((uint8_t*)finger_delete, 16);
   cont_rx = 0;
   while(cont_rx != 11);
   __delay_us(300);
   while(request_fingerprint[9] != FINGERPRINT_OK);
   return request_fingerprint[9];
}

uint8_t Fingerprint_GetTemplateCount(void)
{
   uint8_t usr = 0;
   Fingerprint_Send_Buffer((uint8_t*)finger_template, 12);
   cont_rx = 0;
   while(cont_rx != 13);
   __delay_us(300);
   while(request_fingerprint[9] != FINGERPRINT_OK);

   switch(request_fingerprint[9])
   {
      case FINGERPRINT_OK:
         usr = request_fingerprint[11];
         break;

      case FINGERPRINT_PACKETRECIEVEERR:
         return 0;
         break;
   }
   return usr;
}

uint8_t Fingerprint_EmptyDatabase(void)
{
   Fingerprint_Send_Buffer((uint8_t*)finger_delete_all, 12);
   cont_rx = 0;
   while(cont_rx != 11);
   __delay_us(300);
   while(request_fingerprint[9] != FINGERPRINT_OK);
   return request_fingerprint[9];
}
```

```
uint16_t Fingerprint_Get_ID(void)
{
   uint16_t page_id, sc;

   Fingerprint_GetImage();
   Fingerprint_Image2Tz(1);
   Fingerprint_Search();

   switch(request_fingerprint[9])
   {
      case FINGERPRINT_OK:
         page_id = (uint16_t)((request_fingerprint[10]<<8) | request_fingerprint[11]);
         sc = (uint16_t)((request_fingerprint[12]<<8) | request_fingerprint[13]);
         break;

      case FINGERPRINT_NOTFOUND:
         page_id = 0;
         break;
   }
   return page_id;
}
```

- **Función de la comunicación I2C (i2c.c)**

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <stdint.h>
#include "i2c.h"
void I2C_Init(uint8_t sp_i2c)
{
   TRIS_SCL = 1;
   TRIS_SDA = 1;
   SSPSTAT = sp_i2c;
         SSPCON1 = 0x28;
   SSPCON2 = 0x00;
   if(sp_i2c == I2C_100KHZ){
      SSPADD = 19;
   }
   else if(sp_i2c == I2C_400KHZ){
      SSPADD = 4;
   }
}
void I2C_Start(void)
{
   SSPCON2bits.SEN = 1;
   while(SSPCON2bits.SEN == 1);
```

```
}
void I2C_Stop(void)
{
   SSPCON2bits.PEN = 1;
   while(SSPCON2bits.PEN == 1);
}
void I2C_Restart(void)
{
   SSPCON2bits.RSEN = 1;
   while(SSPCON2bits.RSEN == 1);
}
void I2C_Ack(void)
{
   PIR1bits.SSPIF = 0;
   SSPCON2bits.ACKDT = 0;
         SSPCON2bits.ACKEN = 1;
         while(PIR1bits.SSPIF == 0);
}
void I2C_Nack(void)
{
   PIR1bits.SSPIF = 0;
   SSPCON2bits.ACKDT = 1;
         SSPCON2bits.ACKEN = 1;
         while(PIR1bits.SSPIF == 0);
}
uint8_t I2C_Write(uint8_t data)
{
   PIR1bits.SSPIF = 0;
   SSPBUF = data;
   while(PIR1bits.SSPIF == 0);
   return (uint8_t)SSPCON2bits.ACKSTAT;
}

uint8_t I2C_Read(void)
{
   PIR1bits.SSPIF = 0;
   SSPCON2bits.RCEN = 1;
   while(PIR1bits.SSPIF == 0);
   return SSPBUF;
}
```

- **Función de operación LCD por I2C (lcd_i2c.c)**

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include <stdint.h>
#include "i2c.h"
```

```
#include "lcd_i2c.h"
void Lcd_Init(void)
{
   __delay_ms(20);
   Lcd_Cmd(0x33);
   Lcd_Cmd(0x32);
   Lcd_Cmd(0x28);
   Lcd_Cmd(0x0C);
   Lcd_Cmd(0x06);
         Lcd_Cmd(0x01);
   __delay_ms(3);
}
void Lcd_Cmd(unsigned char cmd)
{
   char data_u, data_l;
         data_u = (cmd & 0xF0);
         data_l = ((cmd<<4) & 0xF0);
   I2C_Start();
   I2C_Write(ADDRESS_LCD);
   I2C_Write(data_u|0x0C);
   I2C_Write(data_u|0x08);
   I2C_Write(data_l|0x0C);
   I2C_Write(data_l|0x08);
   I2C_Stop();
}
void Lcd_Write_Char(char c)
{
   char data_u, data_l;
         data_u = (c & 0xF0);
         data_l = ((c<<4) & 0xF0);
   I2C_Start();
   I2C_Write(ADDRESS_LCD);
   I2C_Write(data_u|0x0D);
   I2C_Write(data_u|0x09);
   I2C_Write(data_l|0x0D);
   I2C_Write(data_l|0x09);
   I2C_Stop();
}
void Lcd_Set_Cursor(char col, char row)
{
   unsigned char address;
   switch(row)
   {
      case 1:
         address = 0x00;
         break;
```

```
        case 2:
           address = 0x40;
           break;
        case 3:
           address = 0x14;
           break;
        case 4:
           address = 0x54;
           break;
    }
    address += col - 1;
          Lcd_Cmd(0x80 | address);
}

void Lcd_Write_String(const char *str)
{
    while(*str != '\0')
    {
       Lcd_Write_Char(*str++);
    }
}

void Lcd_Clear(void)
{
    Lcd_Cmd(0x01);
    __delay_ms(2);
}
void Lcd_Shift_Right(void)
{
          Lcd_Cmd(0x1C);
}
void Lcd_Shift_Left(void)
{
          Lcd_Cmd(0x18);
}
void Lcd_Blink(void)
{
    Lcd_Cmd(0x0F);
}
void Lcd_NoBlink(void)
{
    Lcd_Cmd(0x0C);
}
void Lcd_CGRAM_WriteChar(char n)
{
   Lcd_Write_Char(n);
```

```
}
void Lcd_CGRAM_CreateChar(char pos, const char* new_char)
{
   if(pos < 8)
   {
      Lcd_Cmd(0x40 + (pos*8));
      for(char i=0; i<8; i++)
      {
         Lcd_Write_Char(new_char[i]);
      }
   }
}
```

- **Función PWM (pwm_soft.c)**

```
#define _XTAL_FREQ 8000000
#include <xc.h>
#include "pwm_soft.h"
unsigned int contador;
int pwm_value[10];
extern uint8_t cont_rx;
extern uint8_t request_fingerprint[28];

#ifdef PWM_SOFTWARE
void __interrupt() INT_TMR0()
{
   if(INTCONbits.TMR0IF)
   {
      contador+=5;
      if(contador > 256){
         contador = 0;
      }
      #ifdef PWM_CHANNEL_0
      if(contador < pwm_value[0]){PWM_CHANNEL_0 = 1;} else{PWM_CHANNEL_0 = 0;}
      #endif

      #ifdef PWM_CHANNEL_1
      if(contador < pwm_value[1]){PWM_CHANNEL_1 = 1;} else{PWM_CHANNEL_1 = 0;}
      #endif

      #ifdef PWM_CHANNEL_2
      if(contador < pwm_value[2]){PWM_CHANNEL_2 = 1;} else{PWM_CHANNEL_2 = 0;}
      #endif

      #ifdef PWM_CHANNEL_3
      if(contador < pwm_value[3]){PWM_CHANNEL_3 = 1;} else{PWM_CHANNEL_3 = 0;}
      #endif
```

```
      #ifdef PWM_CHANNEL_4
      if(contador < pwm_value[4]){PWM_CHANNEL_4 = 1;} else{PWM_CHANNEL_4 = 0;}
      #endif

      #ifdef PWM_CHANNEL_5
      if(contador < pwm_value[5]){PWM_CHANNEL_5 = 1;} else{PWM_CHANNEL_5 = 0;}
      #endif

      #ifdef PWM_CHANNEL_6
      if(contador < pwm_value[6]){PWM_CHANNEL_6 = 1;} else{PWM_CHANNEL_6 = 0;}
      #endif

      #ifdef PWM_CHANNEL_7
      if(contador < pwm_value[7]){PWM_CHANNEL_7 = 1;} else{PWM_CHANNEL_7 = 0;}
      #endif

      #ifdef PWM_CHANNEL_8
      if(contador < pwm_value[8]){PWM_CHANNEL_8 = 1;} else{PWM_CHANNEL_8 = 0;}
      #endif

      #ifdef PWM_CHANNEL_9
      if(contador < pwm_value[9]){PWM_CHANNEL_9 = 1;} else{PWM_CHANNEL_9 = 0;}
      #endif
      TMR0 = 0;
      INTCONbits.TMR0IF = 0;
   }
   else if(PIR1bits.RCIF == 1)
   {
      request_fingerprint[cont_rx++] = RCREG;
      PIR1bits.RCIF = 0;
   }
}
#endif
void PWM_Init(void)
{
   T0CON = 0x48;              // configuración del Timer 0
   INTCONbits.GIE = 1;        // Habilita las interrupciones globales
   INTCONbits.TMR0IF = 0;     // Flag del Timer 0 apagada
   INTCONbits.TMR0IE = 1;     // Habilita la interrupción del Timer 0
   T0CONbits.TMR0ON = 1;      // Enciende el Timer 0
   TMR0 = 0;                  // Carga 0 al Timer 0
}
void PWM_Init_Ports(void)
{
   #ifdef PWM_CHANNEL_0
```

```
    PWM_CHANNEL_0_DIR = 0;
    PWM_CHANNEL_0 = 0;
    #endif
    #ifdef PWM_CHANNEL_1
    PMW_CHANNEL_1_DIR = 0;
    PWM_CHANNEL_1 = 0;
    #endif
    #ifdef PWM_CHANNEL_2
    PWM_CHANNEL_2_DIR = 0;
    PWM_CHANNEL_2 = 0;
    #endif
    #ifdef PWM_CHANNEL_3
    PMW_CHANNEL_3_DIR = 0;
    PWM_CHANNEL_3 = 0;
    #endif
    #ifdef PWM_CHANNEL_4
    PMW_CHANNEL_4_DIR = 0;
    PWM_CHANNEL_4 = 0;
    #endif
    #ifdef PWM_CHANNEL_5
    PMW_CHANNEL_5_DIR = 0;
    PWM_CHANNEL_5 = 0;
    #endif
    #ifdef PWM_CHANNEL_6
    PMW_CHANNEL_6_DIR = 0;
    PWM_CHANNEL_6 = 0;
    #endif
    #ifdef PWM_CHANNEL_7
    PMW_CHANNEL_7_DIR = 0;
    PWM_CHANNEL_7 = 0;
    #endif
    #ifdef PWM_CHANNEL_8
    PMW_CHANNEL_8_DIR = 0;
    PWM_CHANNEL_8 = 0;
    #endif
    #ifdef PWM_CHANNEL_9
    PMW_CHANNEL_9_DIR = 0;
    PWM_CHANNEL_9 = 0;
    #endif
}

#ifdef PWM_CHANNEL_0
void PWM_CH0_Duty(int duty)
{
  pwm_value[0] = duty;
}
```

```
#endif
#ifdef PWM_CHANNEL_1
void PWM_CH1_Duty(int duty)
{
  pwm_value[1] = duty;
}
#endif
#ifdef PWM_CHANNEL_2
void PWM_CH2_Duty(int duty)
{
  pwm_value[2] = duty;
}
#endif
#ifdef PWM_CHANNEL_3
void PWM_CH3_Duty(int duty)
{
  pwm_value[3] = duty;
}
#endif
#ifdef PWM_CHANNEL_4
void PWM_CH4_Duty(int duty)
{
  pwm_value[4] = duty;
}
#endif
#ifdef PWM_CHANNEL_5
void PWM_CH5_Duty(int duty)
{
  pwm_value[5] = duty;
}
#endif
#ifdef PWM_CHANNEL_6
void PWM_CH6_Duty(int duty)
{
  pwm_value[6] = duty;
}
#endif
#ifdef PWM_CHANNEL_7
void PWM_CH7_Duty(int duty)
{
  pwm_value[7] = duty;
}
#endif
#ifdef PWM_CHANNEL_8
void PWM_CH8_Duty(int duty)
{
```

```
  pwm_value[8] = duty;
}
#endif
#ifdef PWM_CHANNEL_9
void PWM_CH9_Duty(int duty)
{
  pwm_value[9] = duty;
}
#endif
```

FUNCIÓN PRINCIPAL MAIN.C

```
#pragma config PLLDIV = 1, CPUDIV = OSC1_PLL2, USBDIV = 1
#pragma config FOSC = HS, FCMEN = OFF, IESO = OFF
#pragma config PWRT = OFF, BOR = OFF, BORV = 3, VREGEN = OFF
#pragma config WDT = OFF
#pragma config WDTPS = 32768
#pragma config CCP2MX = ON, PBADEN = OFF, LPT1OSC = OFF, MCLRE = ON
#pragma config STVREN = ON, LVP = OFF, XINST = OFF
#pragma config CP0 = OFF, CP1 = OFF, CP2 = OFF, CP3 = OFF
#pragma config CPB = OFF, CPD = OFF
#pragma config WRT0 = OFF, WRT1 = OFF, WRT2 = OFF, WRT3 = OFF
#pragma config WRTC = OFF, WRTB = OFF, WRTD = OFF
#pragma config EBTR0 = OFF, EBTR1 = OFF, EBTR2 = OFF, EBTR3 = OFF
#pragma config EBTRB = OFF

#define _XTAL_FREQ 8000000
#include <xc.h>

// Constante para asignar direcciones de la EEPROM para guardar datos
__EEPROM_DATA(1,1,0,0,0xFF,0xFF,0xFF,0xFF);

#include "i2c.h"                    // librería del protocolo i2c
#include "eeprom.h"                 // librería de la memoria eeprom interna
#include "pwm_soft.h"               // librería para generar PWM por software
#include "lcd_i2c.h"                // librería para el control de la pantalla lcd por i2c
#include "fingerprint.h"            // librería del lector de huella dactilar

#define MOTOR_DIR_IZQ 0x01          // Constante para el giro de motor hacia la izquierda
#define MOTOR_DIR_DER 0x00          // Constante para el giro de motor hacia la derecha

#define DUTY_MOTOR_1 100            // Ciclo de trabajo para el PWM del motor 1
#define DUTY_MOTOR_2 100            // Ciclo de trabajo para el PWM del motor 2

uint8_t flag_lcd = 1;               // Variable de estado para mostrar el menú de la pantalla lcd
uint8_t cont_fail = 0;              // Contador de intentos para apertura
uint8_t state_user_1 = 0;           // Variable de estado para el mecanismo 1
```

```
uint8_t state_user_2 = 0;                // Variable de estado para el mecanismo 2
uint8_t enter_id, c_store;               // Almacena id del lector de huella y estado del lector

void Mostrar_Menu(void);                 // declaración de prototipos de funciones a utilizar
void Verificar_Error(void);
void Leer_Usuario(uint8_t n_user);
void Registrar_Usuario(uint8_t n_user);
void Motor_1_On(uint8_t dir);
void Motor_2_On(uint8_t dir);
void Motor_1_Off(void);
void Motor_2_Off(void);

void main()
{
   CMCON = 0x07;                         // Apaga los comparadores internos
   ADCON1bits.PCFG = 0x0F;               // Configura todos los pines como digitales
   TRISA = 0x0F;                         // Pines RA0 a RA3 como entradas, pines RA4 y RA5 como
salidas
   TRISB = 0x3F;                         // Pines RB0 a RB5 como entradas, pines RB6 y RB7 como
salidas
   LATAbits.LATA4 = 0;                   // Pin RA4 - led verde
   LATAbits.LATA5 = 0;                   // Pin RA5 - led rojo
   PWM_Init();                           // Inicializa el PWM por software
   PWM_Init_Ports();                     // Inicializa los puertos PWM

   I2C_Init(I2C_400KHZ);                 // Inicializa el protocolo i2c
   Lcd_Init();                           // Inicializa la pantalla lcd
   Lcd_Set_Cursor(6,2);
   Lcd_Write_String("BIENVENIDO");       // Muestra un mensaje de bienvenida al iniciar el
sistema
   Lcd_Set_Cursor(6,3);
   Lcd_Write_String("BIKER POINT");
   Fingerprint_Init(57600);              // Inicializa el lector de huella dactilar
   Fingerprint_Read_Parameters();        // Lee los parámetros iniciales del lector de huella
dactilar
   __delay_ms(3000);
   Lcd_Clear();
   state_user_1 = EEPROM_Read(0);        // Lee el ultimo estado guardado del mecanismo 1
   state_user_2 = EEPROM_Read(1);        // Lee el ultimo estado guardado del mecanismo 2
   Verificar_Error();                    // Verifica si hubo un error de estado del sistema

   while(1)
   {
      Mostrar_Menu();                    // Inicialmente muestra el menu de opciones

      // Si se presiona el botón 1 "open" abre el macanismo 1
```

```
      if(PORTAbits.RA0 == 1 && state_user_1 == 0)
      {
         while(cont_fail != 2){         // Verifica si intento accesar 2 veces con la huella incorrecta
            Leer_Usuario(1);           // Lee el la huella del usuario 1 para abrir el mecanismo 1
         }
         cont_fail = 0;                // Reinicia el contador de intentos de acceso
      }

       // Si se presiona el botón 1 "close" cierra el mecanismo 1
      if(PORTAbits.RA1 == 1 && state_user_1 == 1)
      {
         Registrar_Usuario(1);         // Llama a la función para registrar el usuario 1
      }

      // Si se presiona el botón 2 "open" abre el mecanismo 2
      if(PORTAbits.RA2 == 1 && state_user_2 == 0)
      {
         while(cont_fail != 2)         // Verifica si intento el acceso 2 veces con la huella incorrecta
         {
            Leer_Usuario(2);           // Lee el la huella del usuario 2 para abrir el mecanismo 2
         }
         cont_fail = 0;                // Reinicia el contador de intentos de acceso
      }

      // Si se presiona el botón 2 "close" cierra el mecanismo 2
      if(PORTAbits.RA3 == 1 && state_user_2 == 1)
      {
         Registrar_Usuario(2);         // Llama a la función para registrar el usuario 2
      }
   }
}

void Mostrar_Menu(void)                  // función para mostrar las opciones del menú principal en
la lcd
{
   if(flag_lcd == 1){
      flag_lcd = 0;
      Lcd_Clear();
      Lcd_Set_Cursor(4,2);
      Lcd_Write_String("INGRESAR");      // Muestra opción de "ingresar"
      Lcd_Set_Cursor(4,3);
      Lcd_Write_String("RETIRAR");       // Muestra opción de "retirar"
   }
}

void Leer_Usuario(uint8_t n_user)        // función para leer el usuario correspondiente
```

```
{
   Lcd_Clear();
   Lcd_Set_Cursor(4,2);
   Lcd_Write_String("Coloque huella");     // Solicita que coloque huella
   Lcd_Set_Cursor(1,3);
   Lcd_Write_String(".....");
   uint16_t id = Fingerprint_Get_ID();     // Lee y verifica si la huella esta almacenada en el
lector

   if(n_user == 1 && id == 1)              // Verifica si es el usuario 1
   {
      LATAbits.LATA4 = 1;                 // Enciende el led verde
      Lcd_Clear();
      Lcd_Set_Cursor(4,2);                // Muestra mensaje de acceso correcto en la lcd
      Lcd_Write_String("Acceso Correcto");
      Lcd_Set_Cursor(4,3);
      Lcd_Write_String("Adelante !!!");
      Motor_1_On(MOTOR_DIR_IZQ);          // Motor 1 gira a la izquierda
      EEPROM_Write(2, 0x02);
      while(PORTBbits.RB2 != 1);          // Espera a que el sensor 1 "open" detecte
      EEPROM_Write(2, 0x00);
      Motor_1_Off();                      // Apaga el motor 1
      LATAbits.LATA4 = 0;                 // Apaga el led verde
      state_user_1 = 1;
      cont_fail = 2;
      EEPROM_Write(0, state_user_1);      // Guarda el estado actual del mecanismo 1
      Fingerprint_DeleteModel(n_user);    // Elimina la huella correspondiente al usuario 1
   }
   else if(n_user == 2 && id == 2)        // Verifica si es el usuario 1
   {
      LATAbits.LATA4 = 1;                 // Enciende el led verde
      Lcd_Clear();
      Lcd_Set_Cursor(4,2);                // Muestra mensaje de acceso correcto en la lcd
      Lcd_Write_String("Acceso Correcto");
      Lcd_Set_Cursor(4,3);
      Lcd_Write_String("Adelante !!!");
      Motor_2_On(MOTOR_DIR_IZQ);          // Motor 2 gira a la izquierda
      EEPROM_Write(3, 0x02);
      while(PORTBbits.RB4 != 1);          // Espera a que el sensor 2 "open" detecte
      EEPROM_Write(3, 0x00);
      Motor_2_Off();                      // Apaga el motor 2
      LATAbits.LATA4 = 0;                 // Apaga el led verde
      state_user_2 = 1;
      cont_fail = 2;
      EEPROM_Write(1, state_user_2);      // Guarda el estado actual del mecanismo 2
      Fingerprint_DeleteModel(n_user);    // Elimina la huella correspondiente al usuario 2
```

```
    }
    else                                    // Verifica si no detecto la huella correcta
    {
      LATAbits.LATA5 = 1;                   // Enciende el led rojo
      Lcd_Clear();
      Lcd_Set_Cursor(7,2);
      Lcd_Write_String("Usuario");          // Muestra mensaje de error en la lcd
      Lcd_Set_Cursor(4,3);
      Lcd_Write_String("No Identificado");
      __delay_ms(1700);
      LATAbits.LATA5 = 0;               // Apaga el led rojo
      cont_fail++;                      // Incrementa el contador de intentos
    }
    Lcd_Clear();
    __delay_ms(300);
    flag_lcd = 1;
}

void Registrar_Usuario(uint8_t n_user)      // función para registrar una huella
{
    Lcd_Clear();
    Lcd_Set_Cursor(4,2);
    Lcd_Write_String("Coloque huella");     // Solicita que coloque huella
    Lcd_Set_Cursor(1,3);
    Lcd_Write_String(".....................");
    Fingerprint_GetImage();                 // Lee le huella detectada
    __delay_ms(200);
    Fingerprint_Image2Tz(1);                // Genera un archivo con la información de la huella
presente
    Lcd_Clear();
    Lcd_Set_Cursor(4,2);
    Lcd_Write_String("Remover dedo");       // Solicita remover el dedo
    Lcd_Set_Cursor(1,3);
    Lcd_Write_String(".....................");
    __delay_ms(1500);
    Lcd_Clear();
    Lcd_Set_Cursor(4,1);
    Lcd_Write_String("Coloque huella");     // Solicita que coloque huella nuevamente
    Lcd_Set_Cursor(6,2);
    Lcd_Write_String("nuevamente");
    Lcd_Set_Cursor(1,3);
    Lcd_Write_String(".....................");
    Fingerprint_GetImage();                 // Lee le huella detectada
    Fingerprint_Image2Tz(2);                // Verifica y compara ambas lecturas de la huella presente
    Fingerprint_GetModel();                 // Genera un archivo con la información de la huella
    c_store = Fingerprint_StoreModel(n_user);   // Almacena la huella en la posición indicada
```

```
if(c_store != FINGERPRINT_OK)          // Verifica si la huella fue almacenada correctamente
{
  Lcd_Clear();
  Lcd_Set_Cursor(2,2);              // Si hubo un error, muestra mensaje de alerta en la lcd
  Lcd_Write_String("Error al registrar");
  Lcd_Set_Cursor(2,3);
  Lcd_Write_String("huella en el sistema");
}
else                                // Sino hubo error, muestra mensaje de huella almacenada
{
  Lcd_Clear();
  Lcd_Set_Cursor(2,1);
  Lcd_Write_String("Huella almacenada");
  Lcd_Set_Cursor(2,2);
  Lcd_Write_String("correctamente !");
  Lcd_Set_Cursor(2,4);
  Lcd_Write_String("Por favor espere...");

  switch(n_user)                    // Compara el usuario correspondiente
  {
    case 1:                         // Verifica si es el usuario 1
      LATAbits.LATA4 = 1;         // Enciende el led verde
      Motor_1_On(MOTOR_DIR_DER);  // Motor 1 gira a la derecha
      EEPROM_Write(2, 0x01);
      while(PORTBbits.RB3 != 1);  // Espera a que el sensor 1 "close" detecte
      EEPROM_Write(2, 0x00);
      Motor_1_Off();              // Apaga el motor 1
      LATAbits.LATA4 = 0;         // Apaga el led verde
      state_user_1 = 0;
      EEPROM_Write(0, state_user_1);  // Guarda el estado actual del mecanismo 1
      break;

    case 2:                         // Verifica si es el usuario 2
      LATAbits.LATA4 = 1;         // Enciende el led verde
      Motor_2_On(MOTOR_DIR_DER);  // Motor 2 gira a la derecha
      EEPROM_Write(3, 0x01);
      while(PORTBbits.RB5 != 1);  // Espera a que el sensor 2 "close" detecte
      EEPROM_Write(3, 0x00);
      Motor_2_Off();              // Apaga el motor 2
      LATAbits.LATA4 = 0;         // Apaga el led verde
      state_user_2 = 0;
      EEPROM_Write(1, state_user_2);  // Guarda el estado actual del mecanismo 2
      break;
  }
}
```

```
   Lcd_Clear();
   __delay_ms(300);
   flag_lcd = 1;
}

void Motor_1_On(uint8_t dir)              // función para mover el motor 1
{
   if(dir == 1){                          // Motor 1 gira a la izquierda
      PWM_CH0_Duty(DUTY_MOTOR_1);
      PWM_CH1_Duty(0);
   }else{                                 // Motor 1 gira a la derecha
      PWM_CH0_Duty(0);
      PWM_CH1_Duty(DUTY_MOTOR_1);
   }
}

void Motor_2_On(uint8_t dir)              // función para mover el motor 2
{
   if(dir == 1){                          // Motor 2 gira a la izquierda
      PWM_CH2_Duty(DUTY_MOTOR_2);
      PWM_CH3_Duty(0);
   }else{                                 // Motor 2 gira a la derecha
      PWM_CH2_Duty(0);
      PWM_CH3_Duty(DUTY_MOTOR_2);
   }
}

void Motor_1_Off(void)                    // función para apagar el motor 1
{
   PWM_CH0_Duty(0);
   PWM_CH1_Duty(0);
}

void Motor_2_Off(void)                    // función para apagar el motor 2
{
   PWM_CH2_Duty(0);
   PWM_CH3_Duty(0);
}

void Verificar_Error(void)                // función para verificar si hubo un error en el sistema
{
   if(EEPROM_Read(2) == 0x01){            // Verifica si hubo error el cerrar mecanismo 1
      Lcd_Clear();
      Lcd_Set_Cursor(1,1);
      Lcd_Write_String("Hubo un error al");
      Lcd_Set_Cursor(1,2);
```

```
    Lcd_Write_String("cerrar mecanismo 1");
    Lcd_Set_Cursor(1,4);
    Lcd_Write_String("Por favor espere...");
    Motor_1_On(MOTOR_DIR_IZQ);          // Motor 1 gira a la izquierda
    while(PORTBbits.RB2 != 1);          // Espera a que el sensor 1 "open" detecte
    Motor_1_Off();                      // Apaga el motor 1
    EEPROM_Write(2, 0x00);              // Elimina el error del mecanismo 1
    Lcd_Clear();
    __delay_ms(500);
}
else if(EEPROM_Read(2) == 0x02){        // Verifica si hubo abrir el cerrar mecanismo 1
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("Hubo un error al");
    Lcd_Set_Cursor(1,2);
    Lcd_Write_String("abrir mecanismo 1");
    Lcd_Set_Cursor(1,4);
    Lcd_Write_String("Por favor espere...");
    Motor_1_On(MOTOR_DIR_DER);          // Motor 1 gira a la derecha
    while(PORTBbits.RB3 != 1);          // Espera a que el sensor 1 "close" detecte
    Motor_1_Off();                      // Apaga el motor 1
    EEPROM_Write(2, 0x00);              // Elimina el error del mecanismo 1
    Lcd_Clear();
    __delay_ms(500);
}
else if(EEPROM_Read(3) == 0x01){
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("Hubo un error al");
    Lcd_Set_Cursor(1,2);
    Lcd_Write_String("cerrar mecanismo 2");
    Lcd_Set_Cursor(1,4);
    Lcd_Write_String("Por favor espere...");
    Motor_2_On(MOTOR_DIR_IZQ);          // Motor 2 gira a la izquierda
    while(PORTBbits.RB4 != 1);          // Espera a que el sensor 2 "open" detecte
    Motor_2_Off();                      // Apaga el motor 2
    EEPROM_Write(3, 0x00);              // Elimina el error del mecanismo 2
    Lcd_Clear();
    __delay_ms(500);
}
else if(EEPROM_Read(3) == 0x02){
    Lcd_Clear();
    Lcd_Set_Cursor(1,1);
    Lcd_Write_String("Hubo un error al");
    Lcd_Set_Cursor(1,2);
    Lcd_Write_String("abrir mecanismo 2");
```

```
        Lcd_Set_Cursor(1,4);
        Lcd_Write_String("Por favor espere...");
        Motor_2_On(MOTOR_DIR_DER);          // Motor 2 gira a la derecha
        while(PORTBbits.RB5 != 1);          // Espera a que el sensor 2 "close" detecte
        Motor_2_Off();                      // Apaga el motor 2
        EEPROM_Write(3, 0x00);              // Elimina el error del mecanismo 2
        Lcd_Clear();
        __delay_ms(500);
    }
}
```