Benemérita Universidad Autónoma de Puebla Móvil, Proyecto Final Mecatrónica Avanzada

Iván Palacios Mirón, Joel Eduardo Lugo Chávez, Cruz Hazai Solís Tlachi, Yair Tezmol Juárez, Noe Tlachi Cuamani

Resumen— el siguiente documento muestra el desarrollo y creación de un vehículo móvil terrestre controlado por una interfaz bluetooth y un microcontrolador.

I. INTRODUCCIÓN

Este proyecto se sitúa en la intersección de la mecánica, la electrónica y la informática, con el objetivo de diseñar y construir un vehículo capaz de desplazarse de manera autónoma, controlado de forma remota a través de una aplicación por bluetooth, busca no solo la construcción de un vehículo capaz de moverse con agilidad y precisión, sino también la implementación de sistemas de control que permitan una comunicación eficiente y segura a través de la aplicación controlada por bluetooth.

La integración de sensores, actuadores y sistemas de control constituirá la base fundamental para lograr un funcionamiento óptimo y una respuesta adecuada a las instrucciones enviadas desde el control remoto.

A lo largo de este informe, se detallará el proceso de diseño, fabricación, implementación y pruebas del vehículo terrestre controlado por bluetooth, destacando los desafíos encontrados, las soluciones aplicadas y los resultados obtenidos. Este proyecto representa una oportunidad para explorar el potencial de la mecatrónica en la creación de soluciones innovadoras y funcionales en el campo de la movilidad controlada a distancia.

II. DESARROLLO

Diseño del modelo cilíndrico móvil en Solidworks, carcasa del cilindro, tapa del cilindro, rin, neumático.

Para la carcasa del cilindro se diseñó el modelo, con un diámetro de 12 mm y una longitud de 18 mm, en la parte de abajo se tomó en cuenta el tamaño de las pilas de 9v, el tamaño del puente H y en la parte de arriba, se hizo el espacio para una mini protoboard donde se colocará la tarjeta esp32, y el pequeño circuito de leds. Véase Ilustración 1

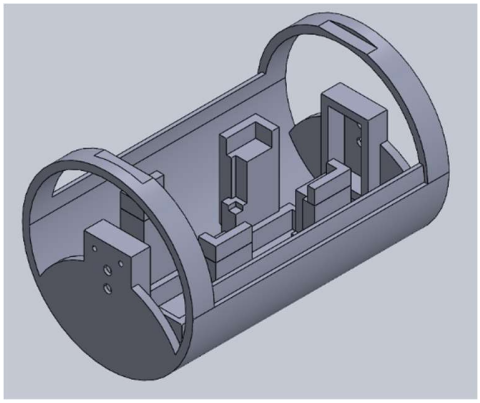


Ilustración 1. Carcasa de Cilindro.

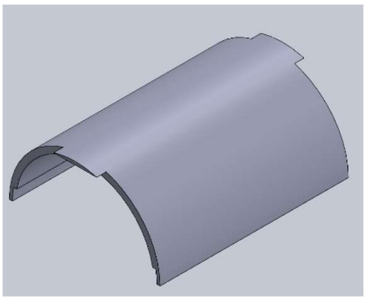


Ilustración 2. Tapa para la Carcasa de Cilindro.

Diseño de las Llantas

Para el modelo de las llantas del cilindro, se decidió un modelo de rin que fuera lo más sencillo, pensando en ahorrar tiempo en el momento de la impresión 3d, con un diámetro de 140 mm y un ancho de 36 mm,

también se diseñó un tipo neumático que abrazara el rin y al momento de ponerlo a girar en físico, pudiera tener fricción con el piso y se pueda mover sin problemas. Véase Ilustración 3.

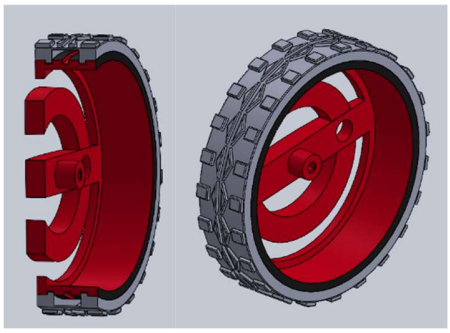


Ilustración 3. Neumático del Cilindro

Por último, uniendo las dos partes, se realizó un ensamble final de las dos partes (carcasa y llantas) para poder previsualizar como quedaría impreso todo nuestro modelo cilíndrico ya unido en físico. Véase Ilustración 4.

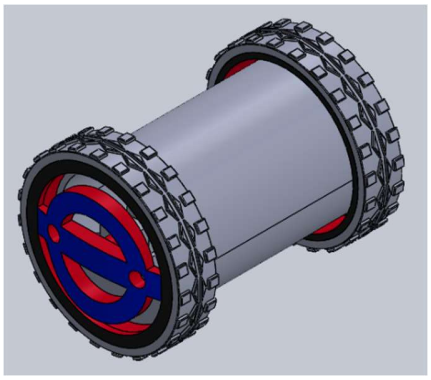


Ilustración 4. Modelo Final del Móvil Cilíndrico.

----Para realizar la comunicación por Bluetooth entre la interfaz de usuario (telefono movil) y el SoC (System on chip) ESP32. Se realizó la combinación de software y firmware.

Para el firmware del SoC ESP32 se utilizó la interfaz de Arduino, la cual permitió desarrollar el código con ayuda de las librerías de Bluetooth de Arduino (Ilustración 5), esto a su vez permitió, que, al ser enviado un número de parte del teléfono móvil, este mandara señales hacia un puente H, controlando la velocidad por medio de PWM y finalmente consiguiendo el movimiento definido por el usuario a través de UI de la aplicación dentro del teléfono móvil.

Ilustración 5 Código para firmware de tarjeta SoC ESP32

Para realizar la parte de software se utilizó el IDE (Entorno de desarrollo integrado) Xcode (Ilustración 6).

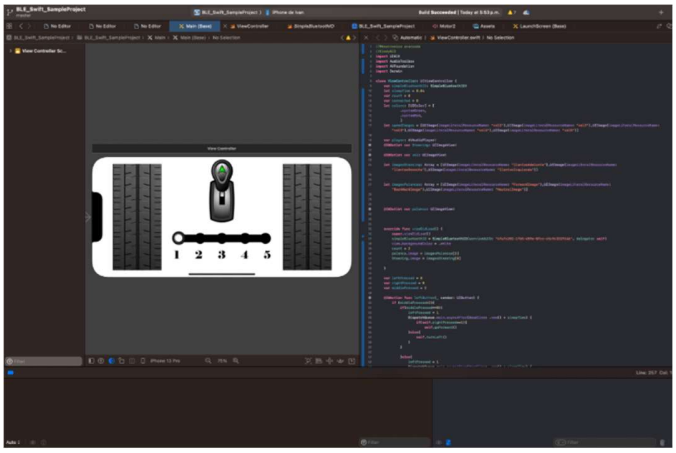
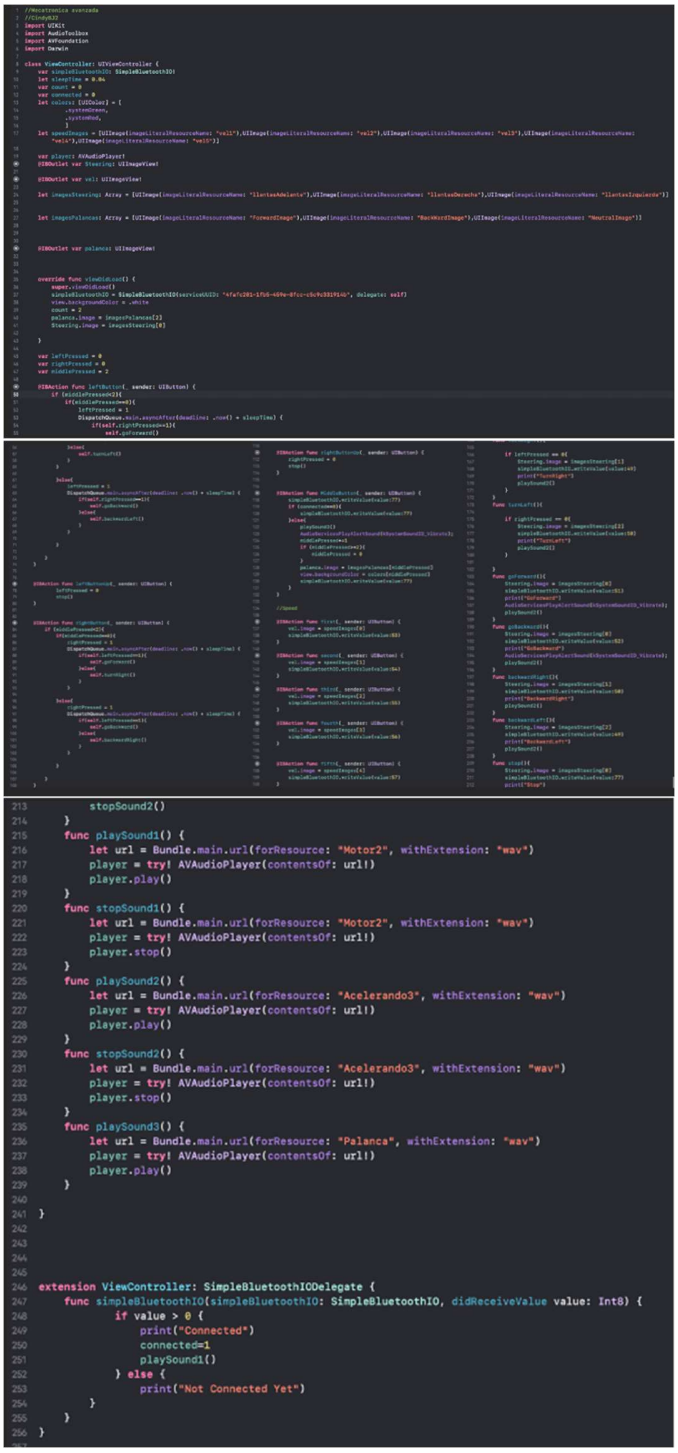


Ilustración 6. Xcode IDE para interfaz de usuario CindyBJ2.

En donde se escribió el código, lógica e interfaz de usuario para el control del proyecto Cindy BJ2 (Ilustración 7).



```
        stopSound2()
    }
    func playSound1() {
        let url = Bundle.main.url(forResource: "Motor2", withExtension: "wav")
        player = try! AVAudioPlayer(contentsOf: url!)
        player.play()
    }
    func stopSound1() {
        let url = Bundle.main.url(forResource: "Motor2", withExtension: "wav")
        player = try! AVAudioPlayer(contentsOf: url!)
        player.stop()
    }
    func playSound2() {
        let url = Bundle.main.url(forResource: "Acelerando3", withExtension: "wav")
        player = try! AVAudioPlayer(contentsOf: url!)
        player.play()
    }
    func stopSound2() {
        let url = Bundle.main.url(forResource: "Acelerando3", withExtension: "wav")
        player = try! AVAudioPlayer(contentsOf: url!)
        player.stop()
    }
    func playSound3() {
        let url = Bundle.main.url(forResource: "Palanca", withExtension: "wav")
        player = try! AVAudioPlayer(contentsOf: url!)
        player.play()
    }

}



extension ViewController: SimpleBluetoothIODelegate {
    func simpleBluetoothIO(simpleBluetoothIO: SimpleBluetoothIO, didReceiveValue value: Int8) {
            if value > 0 {
                print("Connected")
                connected=1
                playSound1()
            } else {
                print("Not Connected Yet")
            }
    }
}
```

Ilustración 7. Código principal para software de UI para Cindy BJ2.

Se utilizó las librerías de Xcode (CoreBluetooth) para hacer la conexión Bluetooth entre el telefono movil y la SoC (Ilustración 8).

```
import CoreBluetooth

protocol SimpleBluetoothIODelegate: AnyObject {
    func simpleBluetoothIO(simpleBluetoothIO: SimpleBluetoothIO, didReceiveValue value: Int8)
}

class SimpleBluetoothIO: NSObject {
    let serviceUUID: String
    weak var delegate: SimpleBluetoothIODelegate?

    var centralManager: CBCentralManager!
    var connectedPeripheral: CBPeripheral?
    var targetService: CBService?
    var writableCharacteristic: CBCharacteristic?

    init(serviceUUID: String, delegate: SimpleBluetoothIODelegate?) {
        self.serviceUUID = serviceUUID
        self.delegate = delegate

        super.init()

        centralManager = CBCentralManager(delegate: self, queue: nil)

    }

    func writeValue(value: Int8) {
        guard let peripheral = connectedPeripheral, let characteristic = writableCharacteristic else {
            return
        }

        let data = Data.dateWithValue(value: value)
        peripheral.writeValue(data, for: characteristic, type: .withResponse)
    }

}

extension SimpleBluetoothIO: CBCentralManagerDelegate {
    func centralManager(_ central: CBCentralManager, didConnect peripheral: CBPeripheral) {
        peripheral.discoverServices(nil)
        if let name = peripheral.name {
            print("Connected! With \(name)")
        }
    }

    func centralManager(_ central: CBCentralManager, didDiscover peripheral: CBPeripheral, advertisementData: [String : Any], rssi RSSI: NSNumber) {
        connectedPeripheral = peripheral
        if let name = peripheral.name {
            print("Discovered \(name)")
            if peripheral.name == "BLE_DEVICE" {
                if let connectedPeripheral = connectedPeripheral {
                    connectedPeripheral.delegate = self
                    centralManager.connect(connectedPeripheral, options: nil)
                }
            }
            centralManager.stopScan()
        }
```

```
    }

    func centralManagerDidUpdateState(_ central: CBCentralManager) {
        if central.state == .poweredOn {
//          centralManager.scanForPeripherals(withServices: [CBUUID(string: serviceUUID)], options: nil)
            centralManager.scanForPeripherals(withServices: nil, options: nil)
        }
    }
}

extension SimpleBluetoothIO: CBPeripheralDelegate {
    func peripheral(_ peripheral: CBPeripheral, didDiscoverServices error: Error?) {
        guard let services = peripheral.services else {
            return
        }

        targetService = services.first
        if let service = services.first {
            targetService = service
            peripheral.discoverCharacteristics(nil, for: service)
        }
    }

    func peripheral(_ peripheral: CBPeripheral, didDiscoverCharacteristicsFor service: CBService, error: Error?) {
        guard let characteristics = service.characteristics else {
            return
        }

        for characteristic in characteristics {
            if characteristic.properties.contains(.write) || characteristic.properties.contains(.writeWithoutResponse) {
                writableCharacteristic = characteristic
            }
            peripheral.setNotifyValue(true, for: characteristic)
        }
    }

    func peripheral(_ peripheral: CBPeripheral, didUpdateValueFor characteristic: CBCharacteristic, error: Error?) {
        guard let data = characteristic.value, let delegate = delegate else {
            return
        }

        delegate.simpleBluetoothIO(simpleBluetoothIO: self, didReceiveValue: data.int8Value())
    }
}
```

Ilustración 8. Módulo (librería) de bluetooth de XCode.

Para la creación de imágenes se utilizó la ayuda del Software de animación de Adobe, Animate y se utilizaron como apoyo visual para la interfaz de usuario como Assets (Ilustración 9).

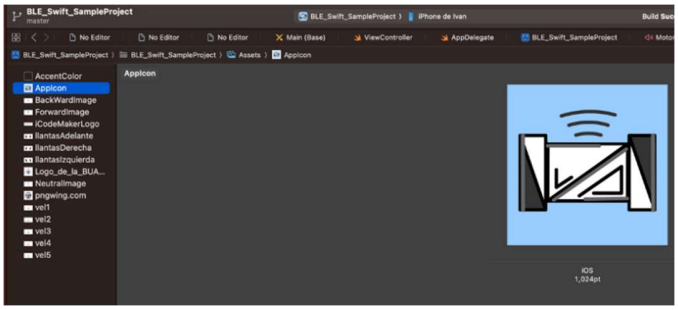


Ilustración 9. Software de animación de Adobe, Animate.

Etapa de potencia

Para ofrecer el voltaje adecuado a los motores utilizados se hizo uso de 2 pilas de 9 V y 400 mah en una configuración en paralelo con el objetivo de proporcionar 9 V al puente H utilizado.

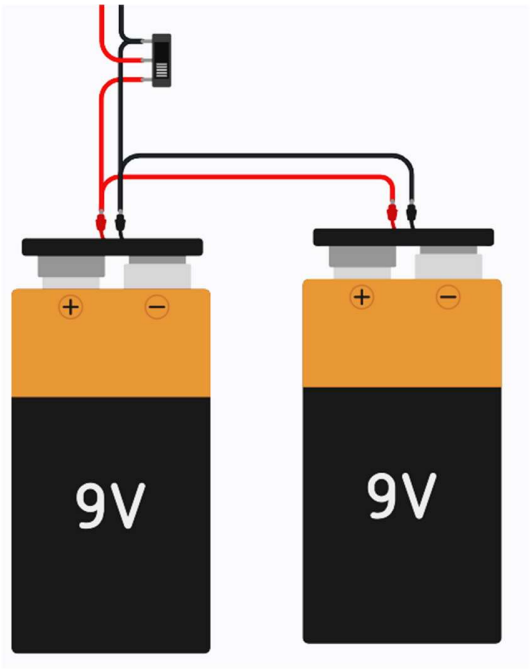


Ilustración 10. Configuración en paralelo para la alimentación.

También se añadió un interruptor para cortar la alimentación manualmente y evitar drenar la energía innecesariamente.

Se realizo esta conexión a nuestro puente h el cual íbamos a implementar para controlar el sentido de giro de nuestros dos motores.

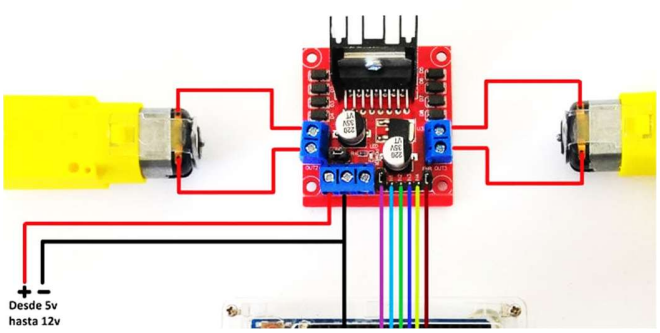


Ilustración 11. Modulo Puente H L298N, conexión a motores a implementar del dispositivo móvil.

Se hizo uso de un Módulo de puente h L298N con doble control de motor el cual nos servirá para controlar los dos motores que se colocaron para el uso de las llantas de nuestro dispositivo móvil de dos ruedas.

Para las conexiones de los motores, se colocaron respecto a su polo, cada uno a su respectiva entrada, ambos motores debían de coincidir ya que sino el sentido de giro de los motores se invertía y nos daban sentidos contrario y conflicto al momento de que nuestro dispositivo móvil quisiera avanzar.

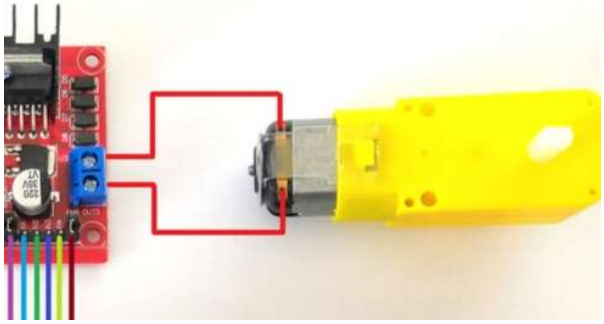


Ilustración 12. Conexión del módulo del puente h a los motores con su respectiva polaridad.

Una vez realizada la conexión de nuestros motores y de nuestras pilas para darle el voltaje a nuestro puente h, se hizo la conexión a nuestra tarjeta SoC ESP32 a sus respectivos pines de entrada, los cuales ya estaban asignados conforme al código presentado anteriormente en la explicación de la interfaz.

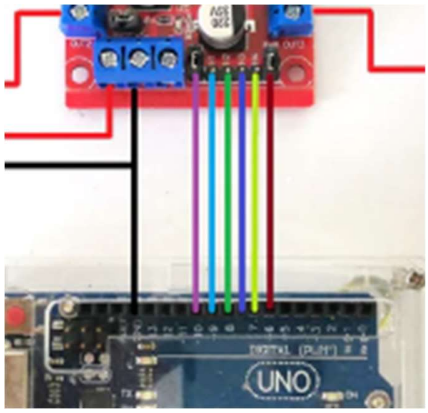


Ilustración 13. Conexión de las entradas del puente h a las entradas de la tarjeta programable.

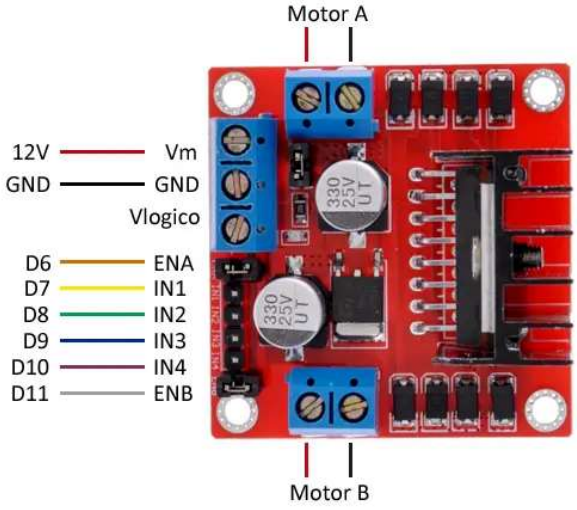


Ilustración 14. Conexión de las entradas del puente h a las entradas de la tarjeta programable.

Como podemos observar en la ilustración 14, tenemos lo que son nuestras entadas ENA, IN1, IN2, IN3, IN4 y ENB, las cuales serán conectadas a nuestras salidas de nuestra tarjeta SoC ESP23, que vendrían a ser nuestros D6, D7, D8, D9, D10, D11.

Nuestras entradas ENA y ENB, serán nuestro PWM, los cuales van a ser aquellos que nos regularan el giro del motor, estos determinan a qué velocidad estará girando nuestras llantas respecto al código que se implementó anteriormente.

En las demás entradas corresponden las conexiones de lo que son nuestros motores, IN1, IN2, estarán conectados a nuestro D7, D8 y serán los que estarán conectados a nuestro motor A. Las entradas IN3, IN4, van a estar conectadas a D9, D10 y serán los controladores de nuestro motor B.

III. RESULTADOS

Se imprimieron todas las piezas en 3d de material PLA y cuando ya estaban listas, se lijaron las partes imperfectas del modelo, para que las piezas se pudieran ajustar de una mejor manera o para quitar la rebaba de la impresión, se colocaron las pilas de 9v, que es la alimentación de todo el circuito de potencia, se colocaron los motores y se conectaron al puente H, se colocó en la parte de arriba el mini protoboard donde va la tarjeta esp32 y todas las conexiones pertinentes del puente H con la tarjeta. Se colocaron las dos llantas a los motorreductores y se colocó un peso en el cilindro, para que tenga un centro de gravedad pesado y no gire la carcasa del modelo.

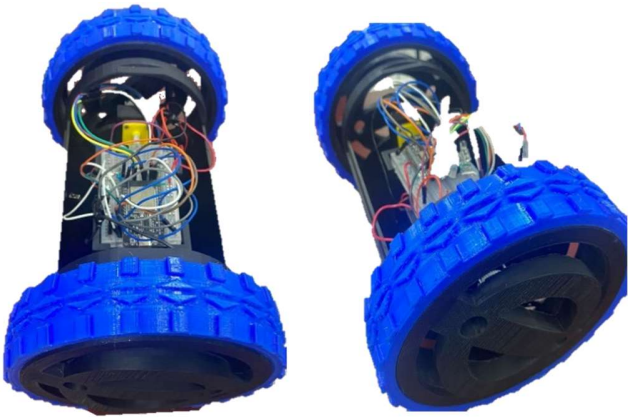


Ilustración 15. Montaje de componentes en la carcasa.

Ilustración 16. Producto final del móvil cilíndrico.

IV. CONCLUSIONES

Durante la realización de este proyecto pudimos contemplar el diseño y El desarrollo de un vehículo terrestre controlado por control remoto, esto ha representado un viaje emocionante a través de la integración de disciplinas clave en la mecatrónica.

Desde la concepción del diseño hasta las pruebas finales, este proyecto ha sido un testimonio de la colaboración entre la mecánica, la electrónica y la informática para alcanzar un objetivo común crear un vehículo autónomo y ágil que responda de manera precisa a comandos remotos a través de conectividad bluetooth.

Durante este proceso, hemos enfrentado desafíos significativos, desde la selección de componentes óptimos hasta la implementación de algoritmos de control eficientes. La sincronización entre los sistemas mecánicos y electrónicos, junto con la programación meticulosa, ha sido crucial para lograr un funcionamiento fluido y una respuesta ágil a las instrucciones recibidas desde el control remoto, así como la actuación de motores y sistemas de dirección y gestión eficiente de la comunicación por bluetooth, ha sido el corazón de este proyecto, obteniendo que los resultados obtenidos sean satisfactorios.

Este proyecto no solo ha sido un ejercicio técnico, sino también una oportunidad para aprender y aplicar conceptos teóricos en un entorno práctico. Además, ha resaltado la importancia de la colaboración interdisciplinaria, el trabajo en equipo y la resolución de problemas, así como la perseverancia en la búsqueda de soluciones innovadoras.

REFERENCIAS

[1] S. M. Metev and V. P. Veiko, Laser Assisted Microtechnologv, 2nd ed., R. M. Osgood, Jr., Ed. Berlin, Germany: Springer-Verlag, 1998.

[2] J. Breckling, Ed., The Analysis of Directional Time Series: Applications to Wind Speed and Direction, ser. Lecture Notes in Statistics. Berlin, Germany: Springer, 1989, vol. 61.