Universidad Nacional de Río Negro, Sede Andina

Carrera:	**Materia:**	**Com.:**
Ingeniería electrónica	**ARQUITECTURA DE COMPUTADORAS Y SISTEMAS EMBEBIDOS**	**1**
Título: **VEHICULO CONTROLADO POR BLUETOOTH**		

Preparado por	**Revisado por**	**Aprobado por**
Nombre Ezequiel Müller	Nombre -	Nombre Leonardo Castillo
Firma	Firma	Firma
Fecha 21/06/2025	Fecha	Fecha

OBSERVACIONES:

Propiedad de título en “Archivo/propiedades/propiedades avanzadas”. Para actualizar los campos automáticos. Presionar “Ctrl + E” y luego “F9”. Hacer doble clic en el pie de página, presionar “Ctrl + E” y luego “F9”. Hacer doble clic fuera del pie de página. Hacer doble clic en encabezado, presionar “Ctrl + E” y luego “F9”. El texto resaltado en amarillo es de orientación borrarlo antes de enviar la versión final. El texto en verde actualizar manualmente.

Nota:

REVISIONES

REV	**Fecha**	**Página Número / Cambios**	**AUTOR**	**REVISÓ**	**APROBÓ**
A	dd/mm/aaaa	Original	Alumno 1, N.	Alumno 2, N.	Profesor, N.

ÍNDICE

1 Objetivo

El objetivo de este proyecto fue construir un auto a control remoto con materiales reciclados y controlado vía bluetooth desde un teléfono. Para ello, se utilizó un microcontrolador ESP32 y se programó en MicroPython, lo que permitió una rápida iteración del código y una estructura modular del sistema.

Se buscó que este auto realizara los movimientos básicos: avanzar, retroceder y girar. Para el desplazamiento se utilizó un único motor de CC y para el giro se diseñó un mecanismo con un servomotor, además, se simularon focos delanteros con LEDS. El proyecto se enfocó no sólo en el aspecto técnico y funcional, sino que, en fomentar el aprovechamiento de materiales reutilizables, principalmente recortes de madera y piezas electrónicas recicladas, con el fin de reducir costos y promover la conciencia ambiental.

Cabe mencionar que la idea original del proyecto incluía funcionalidades más avanzadas, como control de velocidad variable, sensores de proximidad, control de iluminación, imposibilidad de iniciar el vehículo sin tener las luces encendidas, guiñe, modos automáticos como activación obligatoria de luces si la iluminación ambiente era baja, controles PID, entre otros, además de montar todas las conexiones sobre placas de cobre fabricadas de forma artesanal. Sin embargo, por limitaciones de tiempo y recursos, se decidió acotar el desarrollo a una versión más sencilla pero funcional y con conexiones mediante protoboard.

2 Alcance

El proyecto se limitó a controlar un motor de CC para manejar el avance y retroceso del carro y un servomotor para controlar la dirección. También, se incluyeron dos leds que simulan los focos, todo esto activable desde la interfaz de control.

El proyecto abarcó:

- La reutilización de materiales reciclados para la estructura y elementos móviles del vehículo.

- La implementación de un sistema de control inalámbrico por bluetooth, a través de una aplicación móvil compatible (Bluefruit Connect)
- La programación del ESP32 en MicroPython para manejar tanto la lógica de control como las respuestas a los comandos recibidos.
- El diseño y construcción de un sistema de dirección funcional con servomotor.
- La integración de indicadores visuales (LEDs) simulando iluminación vehicular.

El proyecto no abarcó:

- Sistemas de control autónomo o navegación por sensores.
- Transmisión de video, cámaras o sensores avanzados (ultrasonido, infrarrojo, etc.).
- Carcasa estética o terminaciones de diseño profesional.
- Control de velocidad variable (se usó control ON/OFF simple).
- Baterías recargables o administración energética optimizada (se utilizó alimentación básica).

3 MATERIALES

- Microcontrolador ESP32.
- Cables de tipo Arduino y cables extraídos de placas electrónicas.
- 2 LEDS de 5mm color blanco.
- Módulo Puente H L298N.
- Motor de CC de 5V – 12V.
- Servomotor SG90.
- Protoboard.
- Madera reciclada de placas de melamina y recortes varios.
- Tornillos, clavos y tubos metálicos.
- Silicona y pegamentos.
- Engranajes.
- Cadena de caucho

- 2 baterías de 12V.
- Conversor DC-DC Step-Down LM2596.
- Resistencia de 1k x ¼ W.
- Teléfono celular.

4 DESARROLLO

4.1 PASO 1: DISEÑO MECÁNICO

El diseño mecánico fue, sin duda, una de las partes más desafiantes y trabajosas del proyecto. Dado que se buscó trabajar exclusivamente con materiales reciclados, muchas decisiones debieron tomarse en función de los recursos disponibles más que de criterios ideales de ingeniería. Esto implicó rediseños frecuentes, ajustes manuales y numerosas pruebas hasta lograr una estructura funcional y resistente. Debido a estas restricciones, el diseño no fue completamente definido desde el inicio, sino que se desarrolló de manera iterativa: se partió de ideas preliminares que luego se modificaron sobre la marcha a medida que surgían dificultades prácticas. En el apéndice **Diseño Mecánico,** se presenta uno de los últimos esquemas, que refleja de forma aproximada la versión final implementada en el prototipo. También se incluye un diseño alternativo que se intentó implementar en las primeras fases, pero que fue descartado por problemas de potencia y estabilidad estructural

4.2 PASO 2: CODIFICACIÓN

Una vez resuelto el diseño mecánico básico, se procedió al desarrollo del código de control. Inicialmente se realizaron pruebas individuales para cada subsistema (motor, servomotor, LEDs), y se verificó que todas las partes respondieran correctamente a los comandos esperados. Luego de validar el funcionamiento de cada lazo, se unificaron los módulos en un archivo principal (main.py). Con el objetivo de mantener un estilo limpio, reutilizable y escalable, se optó por una estructura basada en programación orientada a objetos, lo que permitirá incorporar nuevas funcionalidades en el futuro sin reescribir el código base.
El programa está dividido en las siguientes clases:

- ble.py (archivo no propio que permite funcionalidades bluetooth y del que sólo se utiliza una clase llamada BLEUART).
- config.py (archivo que contiene definiciones de constantes).
- led_controller.py (archivo que controla los focos).
- motor_controller.py (archivo que controla la marcha del auto).
- servo_controller.py (archivo que controla la dirección del auto).
- main.py (archivo principal).

El código completo se puede ver en el apéndice **Código**

4.3 Paso 3: Montaje eléctrico

El último paso fue el montaje e integración eléctrica de todos los componentes. Para garantizar una mayor seguridad mecánica y estabilidad eléctrica, se decidió realizar conexiones mediante soldadura, con el fin de minimizar la cantidad de conexiones en la protoboard. Además, se requirió alimentar el ESP32 con una fuente externa, ya que la alimentación por USB no es práctica ni adecuada para un vehículo en movimiento. En el apéndice **Montaje Eléctrico**, se presenta el diagrama de conexiones elaborado en la aplicación Fritzing. Cabe aclarar que el modelo exacto del ESP32 utilizado no se encontraba en la biblioteca de Fritzing, por lo que se documentó cuidadosamente la correspondencia real de cada pin.

5 Resultados

Afortunadamente, se logró construir un vehículo completamente funcional, capaz de recibir comandos vía Bluetooth desde un teléfono celular. El sistema permite el movimiento hacia adelante y atrás mediante un motor de corriente continua, así como el giro gracias a un sistema de dirección controlado por un servomotor. Además, se implementó exitosamente el encendido y apagado de LEDs frontales. No obstante, uno de los principales desafíos técnicos fue la alimentación del sistema. Durante las pruebas se observó que el ESP32 no podía ser alimentado eficientemente a través del puerto USB una vez montado en el vehículo, debido a la necesidad de una fuente de energía autónoma y estable. Inicialmente, se intentó utilizar una batería de 9V, pero esta no logró siquiera encender un LED, lo cual evidenció su insuficiente capacidad de corriente.

Como solución, se optó por utilizar dos baterías de 12V con una capacidad superior a 500 mAh, combinada con un convertidor step-down (reductor de tensión) que permitió obtener 5V estables con mayor corriente disponible. Este ajuste fue crucial para evitar reinicios aleatorios del microcontrolador y garantizar el funcionamiento simultáneo de todos los módulos. Además, fue necesario tener en cuenta el consumo total del sistema, en especial durante el funcionamiento conjunto del motor, el servomotor y los LEDs. El diseño debió equilibrar cuidadosamente la demanda de corriente para evitar caídas de tensión que comprometieran la estabilidad del circuito. A pesar de estas dificultades, se logró alcanzar un equilibrio funcional entre rendimiento y simplicidad, cumpliendo con los objetivos propuestos para esta versión del prototipo.

6 Conclusiones

Este proyecto permitió dar una segunda vida a componentes electrónicos y materiales considerados basura, lo que fomenta la reutilización consciente de recursos en un contexto práctico. A través del diseño y la construcción del vehículo, se abordaron aspectos clave de la electrónica, la programación y la mecánica, integrando conocimientos de distintas áreas de forma aplicada. Una de las principales lecciones aprendidas fue la importancia de una correcta planificación del sistema de alimentación, especialmente en proyectos móviles donde la estabilidad eléctrica es crítica para el funcionamiento confiable de los módulos. Además, se evidenció el valor de diseñar el software de forma modular y orientada a objetos, ya que esto facilitó la depuración y dejó la puerta abierta a futuras ampliaciones del sistema. Si bien se descartaron varias funcionalidades ambiciosas por motivos de tiempo y recursos, el proyecto logró su propósito fundamental: demostrar que es posible construir un sistema embebido funcional, económico y educativo reutilizando materiales reciclados y con herramientas accesibles.

7 Bibliografía

- Espressif Systems. (s.f.). *ESP32-WROOM-32 Datasheet* [Archivo PDF]. Espressif.

https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf

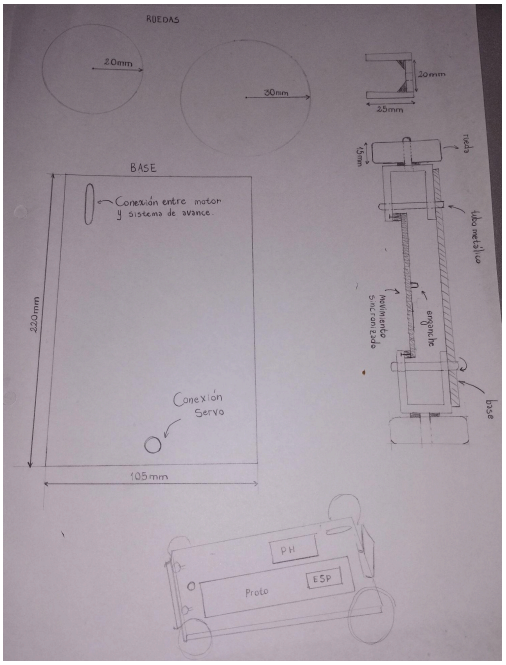
- Random Nerd Tutorials. (2021). *Getting Started with MicroPython on ESP32 and ESP8266* https://randomnerdtutorials.com/getting-started-micropython-esp32-esp8266/
- DroneBot Workshop. (2021, 8 de mayo). *Control DC Motors with L298N Dual H-Bridge and Arduino* [Video]. YouTube https://www.youtube.com/watch?v=MGaJrUe-Zqw

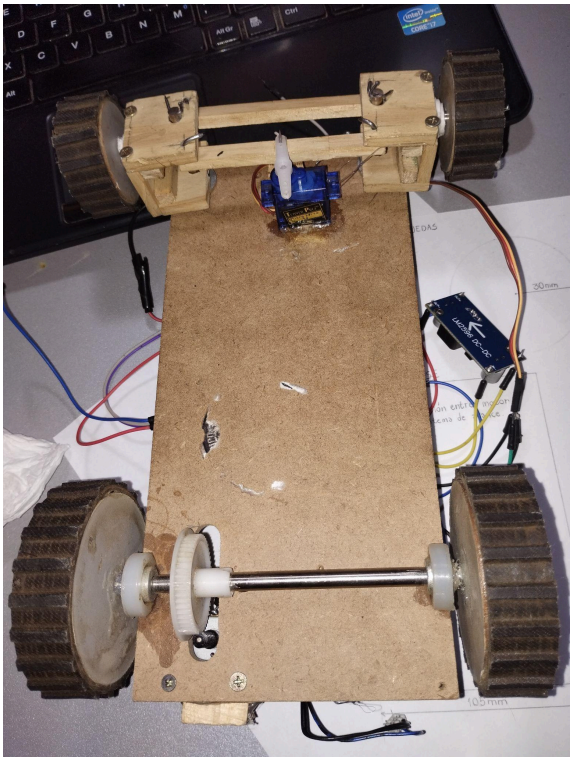
- Makerguides.com. (s.f.). *How to use SG90 Servo Motor with ESP32 or Arduino*. https://www.makerguides.com/sg90-servo-esp32-arduino/

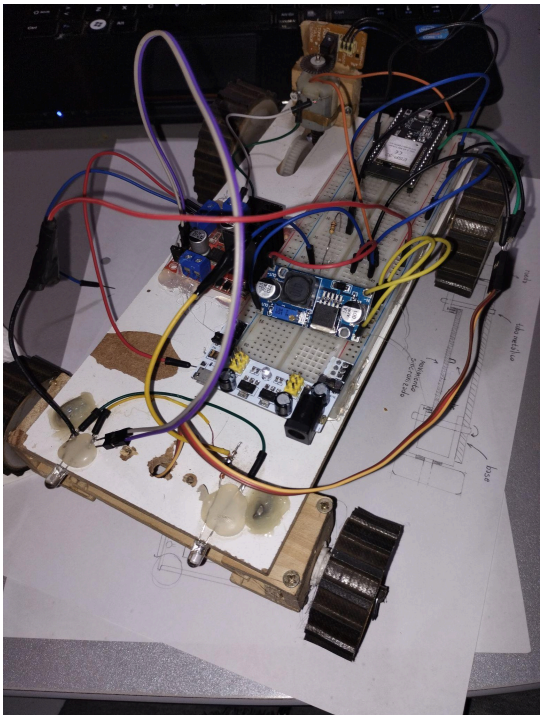
8 Apéndices

8.1 Diagrama mecánico

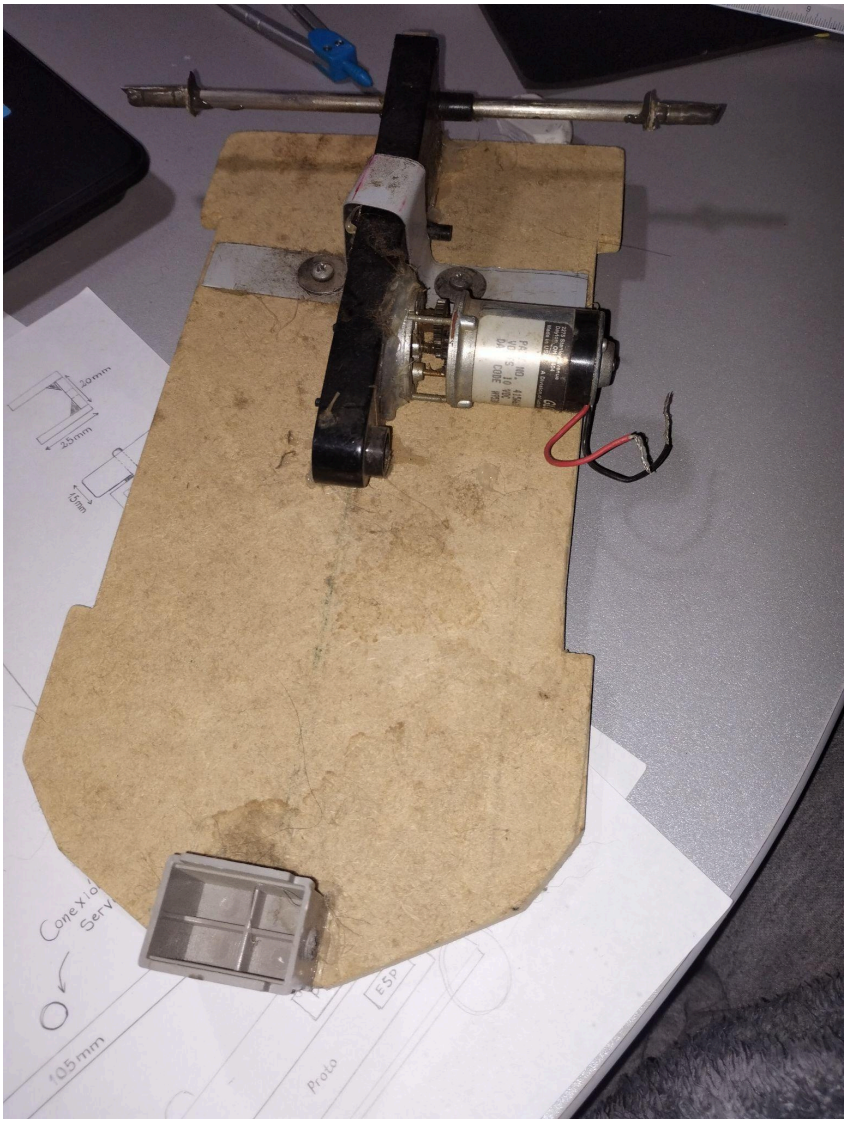
Idea utilizada de diseño mecánico



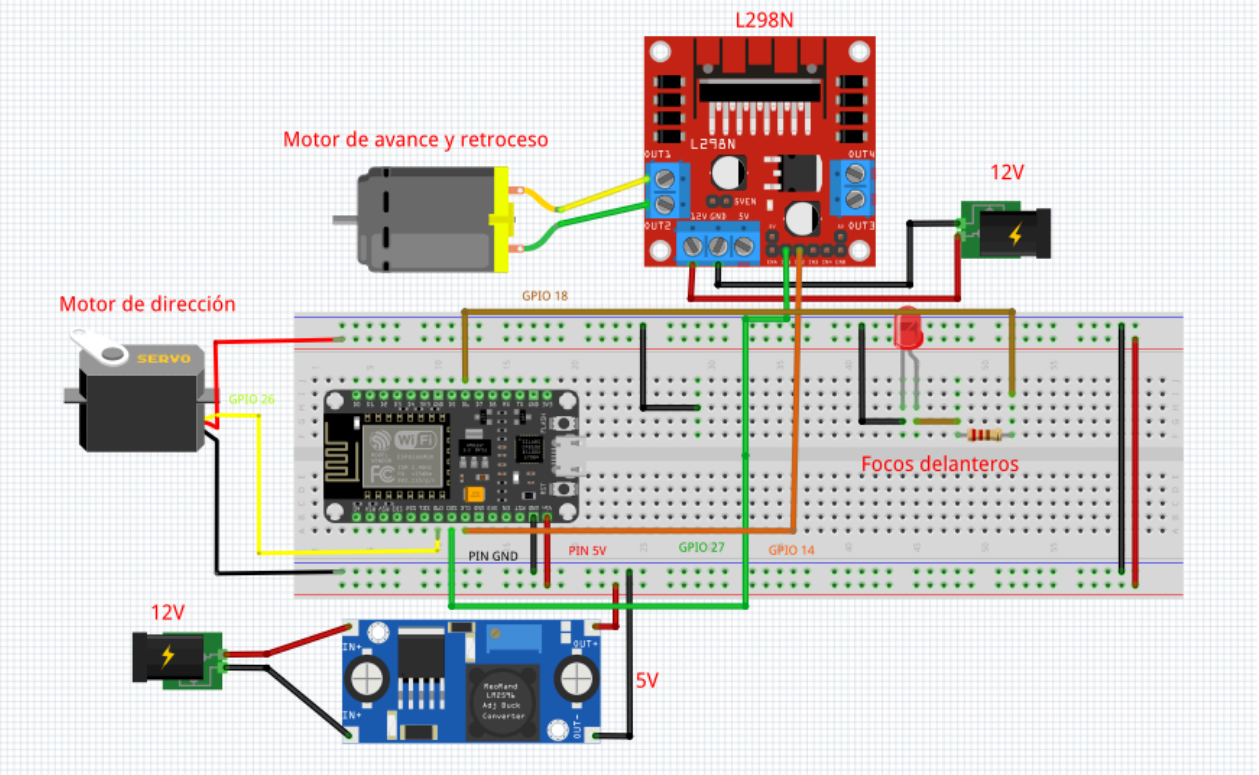




Idea no utilizada de diseño mecánico



8.2 Diagrama eléctrico



8.3 Código

```
class BLEUART:
    def __init__(self, ble, name, rxbuf=1000):
        self._ble = ble
        self._ble.active(True)
        self._ble.irq(self._irq)
        ((self._tx_handle, self._rx_handle),
         ) = self._ble.gatts_register_services((_UART_SERVICE,))
        # Increase the size of the rx buffer and enable append mode.
        self._ble.gatts_set_buffer(self._rx_handle, rxbuf, True)
        self._connections = set()
        self._rx_buffer = bytearray()
        self._handler = None
        # Optionally add services=[_UART_UUID], but this is likely to make the payload too large.
        self._payload = advertising_payload(
            name=name, appearance=_ADV_APPEARANCE_GENERIC_COMPUTER)
        self._advertise()
```

```
# Configuración BLE
BLE_NAME = "Banano"

# Pines
SERVO_PIN = 26
LED = 18

# Posiciones del servo (en grados)
RIGHT = 30
LEFT = 150
NEUTRAL = 90

# Direccion del motor
IN1 = 27
IN2 = 14
```

Universidad Nacional de Río Negro, Sede Andina

Vehículo controlado por bluetooth

```python
# LedController.py
from machine import Pin

class LedController:
    def __init__(self, pin):
        self.led = Pin(pin, Pin.OUT)
        self.off()  # Estado inicial apagado

    def toggle(self):
        self.led.value(not self.led.value())

    def on(self):
        self.led.value(1)

    def off(self):
        self.led.value(0)
```

```python
# motor_controller.py
from machine import Pin

class MotorController:
    def __init__(self, in1_pin, in2_pin):
        self.in1 = Pin(in1_pin, Pin.OUT)
        self.in2 = Pin(in2_pin, Pin.OUT)
        self.stop()  # Estado inicial

    def adelante(self):
        self.in1.on()
        self.in2.off()

    def atras(self):
        self.in1.off()
        self.in2.on()

    def stop(self):
        self.in1.off()
        self.in2.off()
```

```python
from machine import Pin, PWM
from config import *

class ServoController:
    def __init__(self, pin):
        self.servo = PWM(Pin(pin), freq=50)
        self.move(NEUTRAL)  # Inicia en neutro

    def _angle_to_duty(self, angle):
        """Convierte ángulo (0-180) a duty cycle"""
        return int((angle * (125 - 25)) / 180 + 25)

    def move(self, angle):
        """Mueve el servo al ángulo especificado"""
        self.servo.duty(self._angle_to_duty(angle))
```

```python
from machine import Pin
import bluetooth
from ble import BLEUART
from servo_controller import ServoController
from led_controller import LedController
from motor_controller import MotorController
import config

def main():
    # Hardware
    servo = ServoController(config.SERVO_PIN)
    led = LedController(config.LED)
    motor = MotorController(config.IN1, config.IN2)

    # Configuración BLE
    BLE_NAME = "Banano"
    print(BLE_NAME, " Conectado")
    ble = bluetooth.BLE()
    uart = BLEUART(ble, BLE_NAME)

    def on_ble_command():
        cmd = uart.read().decode().strip()
        print("Comando:", cmd)
```

```
        if cmd == "!B507":
            motor.adelante()

        if cmd == "!B606":
            motor.atras()

        if cmd == "!B219":
            motor.stop()

        if cmd == "!B804":
            servo.move(config.RIGHT)

        if cmd == "!B705":
            servo.move(config.LEFT)

        if cmd == "!B11:":
            servo.move(config.NEUTRAL)

        if cmd == "!B408":
            led.on()
        if cmd == "!B309":
            led.off()

    uart.irq(handler=on_ble_command)
    print(f"{BLE_NAME} listo!")

if __name__ == "__main__":
    main()
```