Sistema Bici-Smart

Informe sobre IA

Leandro Raul Mallia, Julián Ezequiel Naspleda, Alejandro Maudet, Leonel De Luca
43520743, 44391303, 43407685, 42588356
Martes noche, M1

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina

Resumen.

Este informe presenta el proceso y los resultados de rehacer el trabajo práctico de la bicicleta inteligente con ESP32 utilizando VibeCoding, comparándolo con el desarrollo manual. Se detallarán las herramientas empleadas, el modelo de IA seleccionado y los prompts utilizados para generar el código. Además, se documentará el tiempo requerido para lograr una versión funcional mediante VibeCoding total y se analizarán métricas de rendimiento (uso de CPU y memoria) en ambos casos de prueba: código manual vs. código generado por IA. Finalmente, se incluirá la publicación del proyecto en Git con dos directorios (con y sin VibeCoding) y una evaluación según rúbricas de funcionalidad, legibilidad y mantenibilidad, destacando ventajas, limitaciones y aprendizajes obtenidos.

Herramienta Utilizada: ChatGPT

Modelo Utilizado: Vibe Coding GPT 5.1 Thinking (Por Luca Amadori)

Prompt Final Utilizado:

LENGUAJE DE PROGRAMACIÓN:

- Utilizar lenguaje Wiring / C++ para Arduino IDE.
- El código debe ser compilable para ESP32 sin dependencias externas.

PLATAFORMA:

- ESP32 DevKit v1 (Arduino IDE).
- Enfoque IoT: integración con Ubidots vía WiFi + MQTT.
- Simulación posterior en Wokwi (requiere poder comentar WiFi/MQTT).

FUNCIONALIDAD A IMPLEMENTAR:

- Desarrollar una "bicicleta inteligente" que mida velocidad, distancia frontal y nivel de luz.
- Mostrar velocidad en LCD, encender luz automática por LDR, y activar freno + buzzer según distancia.
- Reportar a Ubidots el contador de vueltas de rueda durante viajes activos.
- Implementar una máquina de estados finitos con: OFF, STOPPED, RIDING, BRAKING.

REQUISITOS ESPECÍFICOS:

1. Hardware y pines obligatorios
    - Ultrasonido HC-SR04:
        - TRIG en GPIO 5
        - ECHO en GPIO 18
    - Hall effect (vueltas rueda):
        - GPIO 4 con interrupción + antirrebote
    - LDR:
        - ADC GPIO 32
    - Switch ON/OFF físico:
        - GPIO 26 con INPUT_PULLUP
    - Actuadores:
        - Servo freno GPIO 23
        - LED luz GPIO 12
        - Buzzer PWM GPIO 19
        - LCD RGB I2C 16×2 (rgb_lcd)
2. Parámetros y umbrales

- Diámetro rueda: 0,71 m.
- Distancia (cm):
  - 30 lejos
  - 20–30 medio
  - 10–20 cerca
  - <10 muy cerca
- Luz (ADC):
  - <1500 poca luz ⇒ prender LED
  - 2000 mucha luz ⇒ apagar LED
- Buzzer:
  - 0 Hz lejos
  - 500 Hz medio
  - 1000 Hz cerca
  - 2000 Hz muy cerca
- Servo freno:
  - 0° freno liberado
  - hasta 180° freno máximo
  - interpolación lineal entre 20 cm y 10 cm
- Timeout de velocidad:
  - si no hay pulsos Hall por 5 s, velocidad pasa a 0 km/h.

3. Conectividad Ubidots (WiFi/MQTT)
   - Conectarse a WIFI_SSID / WIFI_PASSWORD.
   - Broker MQTT: industrial.api.ubidots.com, puerto 1883.
   - Autenticación: USER + PASS (token Ubidots).
   - Suscribirse a:
     - TOPIC_CONTROL “/v1.6/devices/bici/control/lv”
       - “1.0” encender sistema, “0.0” apagar
     - TOPIC_TRIPS “/v1.6/devices/bici/trips/lv”
       - “1.0” iniciar viaje, “0.0” terminar viaje
   - Publicar en:
     - TOPIC_DATA “/v1.6/devices/bici”
     - JSON: {"data": <wheel_turn_count>} cada 3 s mientras viaje esté activo.
   - Si se corta MQTT, reconectar periódicamente.
4. Lógica por estados (FSM)

- OFF: todo apagado (LCD sin display, LED off, buzzer off, freno liberado, ultrasonido deshabilitado).
- STOPPED: al encender (switch físico o MQTT) inicializa sensores y LCD con “Velocidad: 0.0”. Mantiene control de luz con LDR.
- RIDING: al detectar una vuelta (Hall) pasa a RIDING y habilita ultrasonido. Actualiza velocidad en LCD, buzzer según distancia y freno si cerca/muy cerca.
- BRAKING: aplica freno proporcional y buzzer según distancia; si vuelve a medio/lejos libera freno y regresa a RIDING.
- Timeout de movimiento: desde RIDING/BRAKING vuelve a STOPPED.
- Apagado global: desde cualquier estado, si llega apagado (switch o MQTT), vuelve a OFF.

5. Arquitectura solicitada
    - Usar FreeRTOS con tareas separadas:
        - lectura de sensores
        - lectura de switch
        - comunicaciones WiFi/MQTT
        - tarea FSM que consume eventos desde una cola
    - Proteger contador de vueltas con sección crítica / mutex.
    - Incluir comentarios claros en el código.

RESTRICCIONES:

- La FSM debe seguir patrón de switch anidados:
    - un switch principal por estado y dentro un switch por evento.
- Los eventos solo se generan en las tareas de sensores/switch/MQTT y se envían por cola.
- La FSM no debe leer sensores directamente (solo consume eventos).
- Evitar delays bloqueantes dentro de FSM; el tiempo se gestiona en tareas.
- El código debe poder adaptarse a Wokwi comentando toda la parte WiFi/MQTT sin romper la compilación.

Aclaración posterior solicitada:

Luego de obtener el código inicial con conectividad WiFi y MQTT (según el prompt anterior), se requirió una adaptación específica para su ejecución en el simulador Wokwi. Debido a que Wokwi no incluye por defecto las librerías de red (por ejemplo PubSubClient.h) y a diferencias en las APIs PWM del entorno, se pidió comentar y deshabilitar toda la lógica relacionada con WiFi y MQTT, manteniendo intacta la lógica de sensores, actuadores y la máquina de estados. En consecuencia, se eliminó del proceso de compilación la inicialización de WiFi/MQTT, la tarea de comunicaciones y los eventos asociados, ajustando el control PWM del buzzer al método compatible con Wokwi. Esta modificación permitió validar el comportamiento funcional del sistema en simulación sin conectividad externa.

Tiempo necesario en hacer que funcione: Aproximado a 6 horas.

Caso de prueba 1 (Idle)

Se llama a initStats al final del setup, y se finaliza el muestreo luego de 10 segundos en ambos casos.

Métricas código manual:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     10(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  98.64% | Ocupado:   3.39% | Libre (IDLE):  95.25%
Core 1 -> Total:  99.92% | Ocupado:  60.99% | Libre (IDLE):  38.93%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 328616 bytes
Heap libre : 201463 bytes
Heap usado : 127152 bytes
Uso        : 38.69 %

Muestreo de Metricas finalizado
```

Métricas código con vibecoding:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     10(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  99.54% | Ocupado:   6.22% | Libre (IDLE):  93.32%
Core 1 -> Total:  99.91% | Ocupado:   4.85% | Libre (IDLE):  95.05%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 329432 bytes
Heap libre : 206363 bytes
Heap usado : 123068 bytes
Uso        : 37.36 %

Muestreo de Metricas finalizado
```

Caso de prueba 2 (Viaje)

Se llama a initStats al iniciar un viaje, y se finaliza el muestreo luego de 30 segundos.

Métricas código manual:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     30(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  99.40% | Ocupado:   2.02% | Libre (IDLE):  97.39%
Core 1 -> Total:  99.98% | Ocupado:  89.78% | Libre (IDLE):  10.19%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 328616 bytes
Heap libre : 201027 bytes
Heap usado : 127588 bytes
Uso        : 38.83 %
```

Métricas código con vibecoding:

```
=== Contribución Promedio al total del sistema ===
=== Tiempo de muestreado:     30(segundos) ===

================================================
====== Estado Promedio del Uso de CPU en ESP32 ===
================================================
Core 0 -> Total:  99.92% | Ocupado:   3.99% | Libre (IDLE):  95.93%
Core 1 -> Total:  99.99% | Ocupado:   3.34% | Libre (IDLE):  96.65%

================================================
====== Estado Promedio de la memoria en ESP32 ====
============= Memoria interna (Heap) =============
================================================
Heap total : 329432 bytes
Heap libre : 204711 bytes
Heap usado : 124720 bytes
Uso        : 37.86 %
```