Smart Roller

Albanesi Matias, Arias Gabriel, Collazo Ignacio, Panigazzi Agustin, Rios Cristian
39770399, 39849565, 41537099, 43744593, 40015557
Martes, 2

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

Resumen. Smart Roller es un sistema embebido el cual permite controlar el ascenso y descenso de una cortina. El sistema posee dos modos de uso: Automático y Manual. El automático sube o baja la cortina comparando la luminosidad captada por un sensor de luz, con un umbral pre-configurado. Mientras que el modo manual, permite ingresar los comandos por consola/app “Abrir”, “Cerrar”, “Pausar”, “Cm auto” y “Cm manual”. Para manejar la cortina, utiliza un motor conectado a corriente continua y un dispositivo Puente H L298 encargado de indicar el sentido de giro, además de otros componentes electrónicos..

Palabras claves: Cortina, Luz, Automático, Manual.

Introducción

El presente proyecto consiste en el desarrollo de un sistema embebido implementado en **ESP32**, complementado con una aplicación móvil **Android**. El sistema tiene como finalidad controlar remotamente una **cortina roller** motorizada, integrando sensores de luz y dos finales de carrera para el control preciso de apertura y cierre.

La aplicación Android se comunica vía protocolo **MQTT** con el sistema embebido, permitiendo al usuario seleccionar entre un modo automático (apertura/cierre basado en el nivel de luz ambiente) y un modo manual (control directo de apertura/cierre), además de poder establecer programación de horarios para abrir o cerrar la cortina en los intervalos establecidos.

Este desarrollo apunta a ser una solución de automatización para el hogar, fomentando el ahorro energético y la comodidad del usuario.

Estado Inicial

Al encender el sistema, se inicia en **modo manual**, con el estado del motor en **"STOPPED"** (detenido). A partir de allí, el usuario podrá interactuar mediante comandos ingresados por consola/app.

Comandos Disponibles (Modo Manual)

Comando	Accion
Abrir	Abre la cortina
Cerrar	Cierra la cortina
Pausar	Detiene el motor que maneja la cortina
Cm manual	Cambia el sistema al **modo manual**.
Cm auto	Cambia el sistema al **modo automático**.

Modo Automático

En este modo, el sistema controla la cortina en base a la **cantidad de luz medida por un sensor**. El motor se acciona automáticamente según el valor de luz en lux:

- Si la luz **supera los 950 lux**, el motor pasa a **forwarding** (sube la cortina).
- Si la luz **es menor a 950 lux**, el motor pasa a **backwarding** (baja la cortina).

Importante:

- El sensor de luz tiene un **delay de 5 segundos**, ya que no es crítico.
- Si se alcanza un **final de carrera** mientras el motor está en movimiento, el motor se detendrá.
- Si el final de carrera no se mantiene presionado, y el sensor de luz sigue detectando condiciones para moverse, la cortina **volverá a moverse automáticamente**.

Finales de Carrera (FC)

Los **finales de carrera** actúan como **límites físicos de movimiento** de la cortina:

Final de carrera	Asociado a	Accion
FC_START	Backwarding (bajar)	Detiene el motor si está bajando y se alcanza el tope inferior.
FC_END	Forwarding (subir)	Detiene el motor si está subiendo y se alcanza el tope superior.

Consideraciones Finales

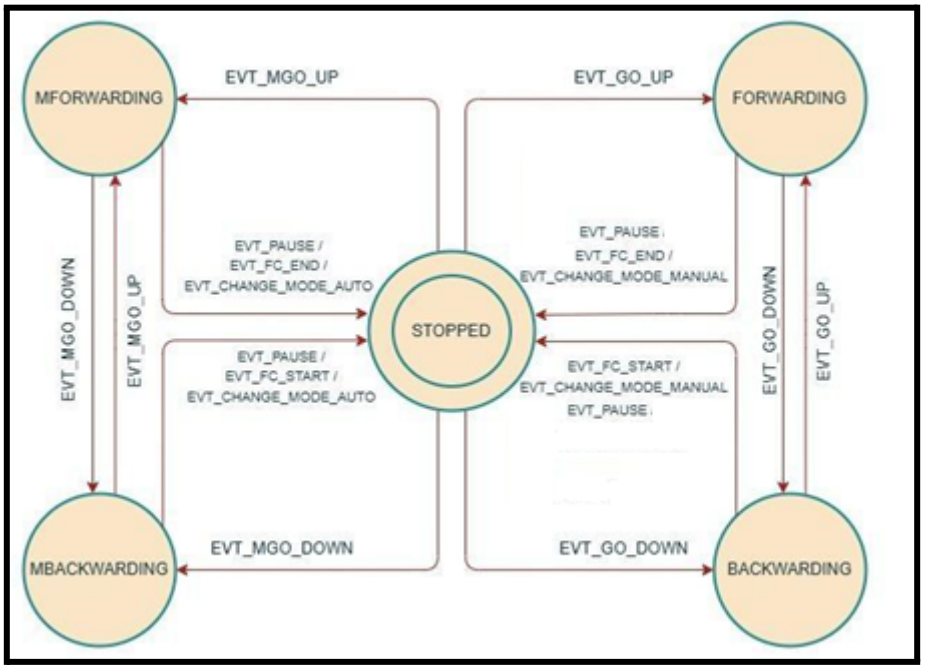
- El sistema **puede cambiar entre modos manual y automático** en cualquier momento.
- En modo automático, el usuario no necesita intervenir salvo para cambiar el modo o apagar el sistema.
- Los finales de carrera son mecanismos de seguridad para evitar daños físicoss.

Desarrollo

URL al proyecto de Wokwi:
https://www.wokwi.com/projects/430781745282961409

URL al repositorio del Proyecto:
https://www.github.com/UNLAM-SOA/2025_SOA_Q1_M2

Diagrama de estados



Conexiones del Circuito de Wokwi:

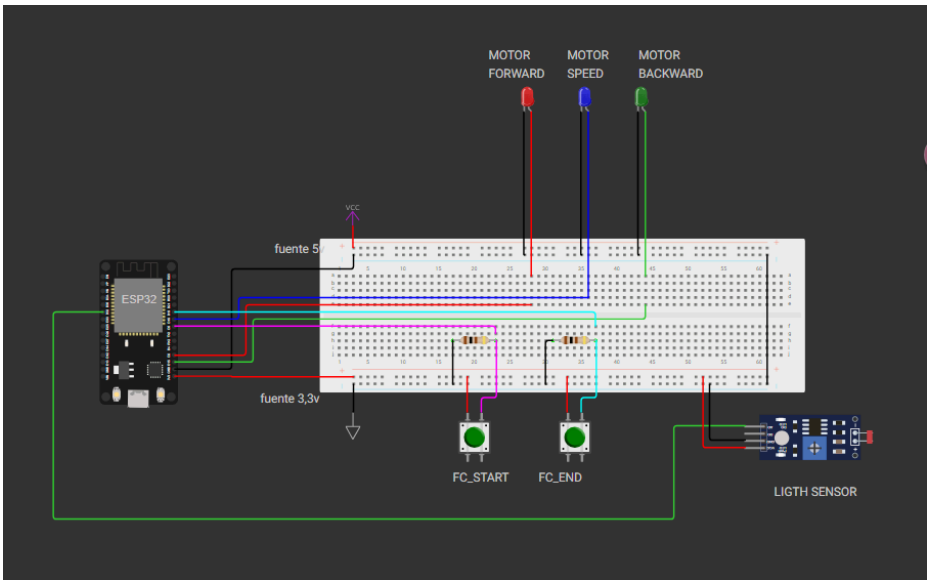
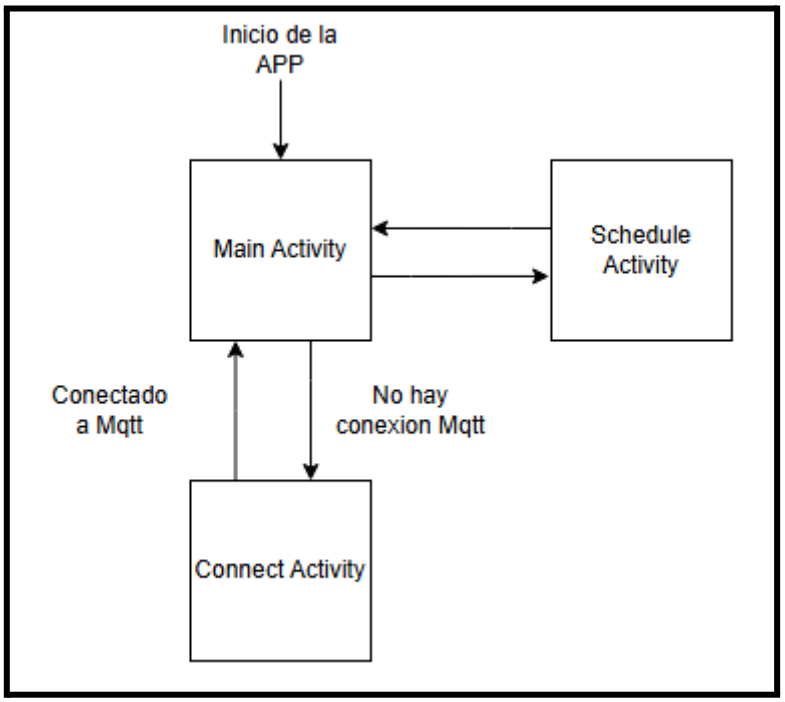


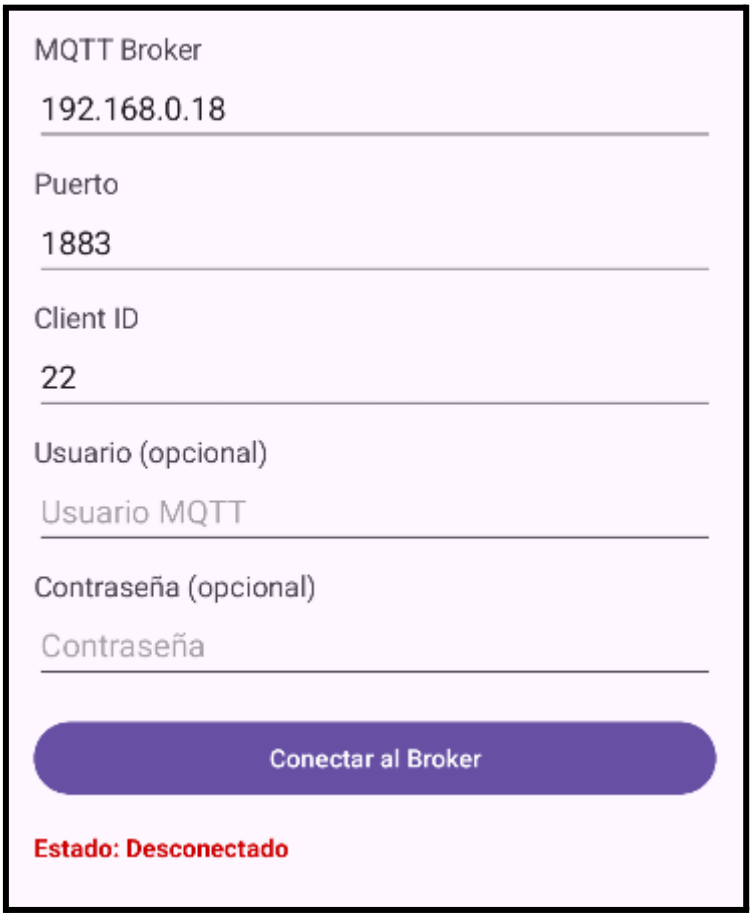
Diagrama functional de la APP



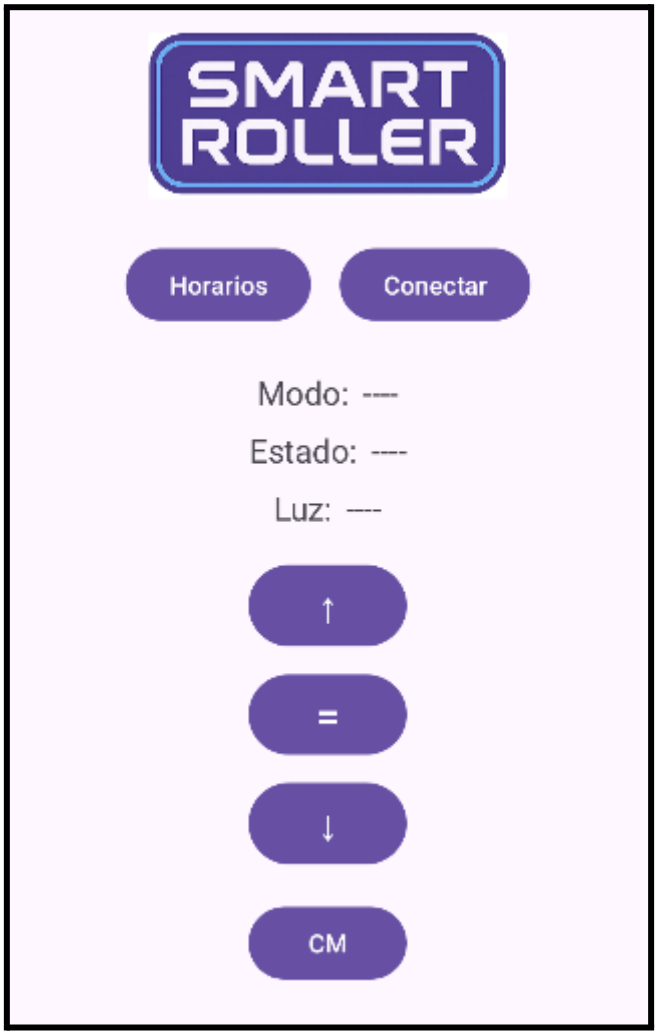
Manual de Usuario

Para poder usar la app deberá de seguir los siguientes pasos

1) Conectarse en con el broker de mqtt
   Ejemplo de conexión, como se indica el usuario y contraseña son opcionales. En caso de que el broker requiera usuario y contraseña deberá completarlos para poder efectuar la conexión correctamente.

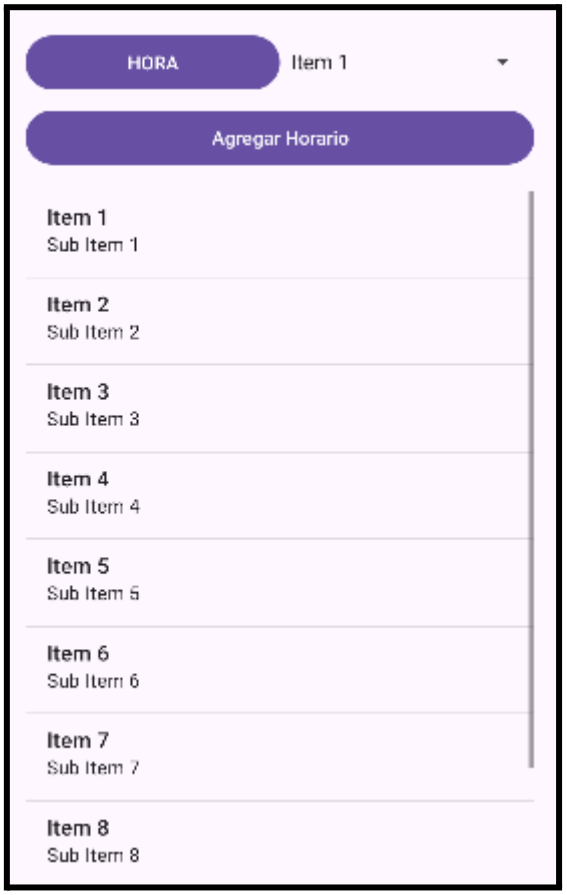


2) Pantalla principal



- Flecha hacia arriba: Envia el comando ‘Abrir’ a la cortina.
- Signo =: Envia el comando ‘Pausar’ a la cortina.
- Flecha hacia abajo: Envia el comando ‘Cerrar’ a la cortina.
- Boton CM: Alterna el cambio de modo de la cortina.
- Shake (agitar celular): Envia un ‘ping’ para actualizar los datos de ‘Modo’, ‘Estado’ y ‘Luz’.
- Conectar: Envia a la pantalla de conexión.
- Horarios: Envia a la pantalla donde se programan horarios.

3) Programar horarios de apertura y cierre



- Hora: Permite seleccionar la hora a programar.
- Item 1: Permite elegir entre ‘Abrir’ y ’Cerrar’.
- Agregar horario: Añade el horario a la lista.
- Para eliminar un horario mantenerlo presionado.

Conclusiones

- **Problemas encontrados y resueltos**

  Prototipo:

  - **Problema: La apertura/cierre de la cortina no lograba presionar los finales de carrera**
    Solución: Se modificó el diseño físico de la maqueta para integrar unas guías laterales que permiten el correcto funcionamiento de la apertura y cierre de la cortina.
  - **Problema: Fuente continua original planteada no tenía potencia para mover el motor**.
    Solución: Se lo reemplazó por una fuente regulada de voltaje conectada a la línea.
  - **Problema: Al cerrar la persiana no presionaba el final de carrera inferior.**
    Solución: Se reemplazó el final de carrera por 2 contactos que al cerrar la persiana, la cual contaba con un conductor, cerraba el circuito y enviaba la señal de corte.

  Software:

  - **Problema: La conectividad de wifi / MQTT en el embebido**
    Solución: Chequear la conectividad a wifi antes de tratar de realizar las conexiones al broker MQTT.

  - **Problema: El broker local (Mosquitto) no acepta conexión desde Android**
    Solución: Se verificó la IP local del broker y se configuró en la app la conexión, con la dirección, puerto y usuarios correctos.

  - **Problema: Dependencia deprecada de paho (MQTT), presenta problemas de compatibilidad con Android 12+**
    Solución: Asegurarse de usar dependencias que estén en vigencia para no tener problemas a futuro.

- **Lecciones aprendidas**
  - La simulación en Wokwi es útil pero no reemplaza el testeo en hardware real, especialmente para temas de red.
  - La interacción entre un sistema embebido físico y una aplicación Android presenta múltiples desafíos, especialmente en cuanto a conectividad y sincronización.
  - El protocolo MQTT resultó ideal por su ligereza y velocidad, pero requiere una buena gestión de reconexión y control de errores.

- El diseño modular de la app con múltiples Activities permitió escalar y probar funcionalidades de forma más ordenada.
- El hilo de UI no debe usarse para lógica de red porque algunos errores pueden surgir si se bloquea la interfaz esperando respuestas del ESP32.
- El ciclo de vida de Android puede interrumpir la conexión. Se debe controlar el ciclo de vida (onPause, onResume) para reconectar o conservar estado.
- Verificar la compatibilidad de las dependencia a usar.

Referencias

1. https://wokwi.com/projects/375932751433618433
2. https://wokwi.com/projects/375257142494987265
3. https://esp32io.com/tutorials/esp32-dc-motor
4. https://www.freertos.org/Documentation/00-Overview
5. https://www.soa-unlam.com.ar/wiki/index.php/PUBLICO:Tiempo_Real
6. https://www.soa-unlam.com.ar/wiki/index.php/PUBLICO:Android
7. https://arduinojson.org/
8. https://github.com/eclipse-paho/paho.mqtt.android
9. https://github.com/hivemq/hivemq-mqtt-client