Trabajo Práctico Integrador

Sistemas Embebidos, Licenciatura en Ciencias de la Computación

Facultad de Ingeniería, UNCuyo

Integrantes:

Arrieta, Nahuel

Del Longo, Micaela

Maglione, Andrés

Docente: Dr. Godoy, Pablo

Año 2025

Introducción

Este informe describe la arquitectura y el funcionamiento de un sistema embebido diseñado para el monitoreo y control automatizado de un jardín. El sistema se compone de tres partes principales: el firmware del microcontrolador Arduino, una aplicación de *backend* desarrollada en Python y una interfaz de usuario *frontend* basada en tecnologías web.

Arquitectura del Sistema

Firmware del Microcontrolador Arduino (`sys_controller.ino`)

El Arduino está programado en C++ y utiliza el sistema operativo en tiempo real FreeRTOS para gestionar múltiples tareas concurrentemente. Las funcionalidades principales del firmware incluyen la lectura de sensores, el control de actuadores, el registro de eventos en la memoria EEPROM y la comunicación serial con un sistema anfitrión.

La inicialización del sistema, que ocurre dentro de la función `setup()`, configura los pines de entrada/salida para los sensores y actuadores, inicializa la comunicación serial y prepara la memoria EEPROM. Se recupera la última hora guardada de la EEPROM para mantener la continuidad temporal en caso de reinicio.

De manera crucial, se determina la próxima dirección disponible en la EEPROM para el registro de eventos mediante la función `findNextEEPROMAddress()`. Esta función escanea el área de registro buscando la primera ranura vacía, marcada por un *timestamp* de `0xFFFFFFFF`. Si toda el área de registro está llena, la función devuelve la dirección de inicio del área de registro, implementando así un mecanismo de sobreescritura circular para los logs.

Finalmente, se crean y se inician varias tareas de FreeRTOS: `readSensorsTask` para la adquisición de datos de sensores, `autoControlTask` para la lógica de control autónomo,

`serialComTask` para la comunicación y `updateTimeTask` para la gestión del tiempo. Los semáforos (*mutex*) `eepromMutex`, `serialMutex` y `sensorMutex` se crean para proteger el acceso a recursos compartidos como la EEPROM, el puerto serial y las variables globales de los sensores, respectivamente, previniendo condiciones de carrera.

La tarea `readSensorsTask` es responsable de leer periódicamente los valores de los sensores de humedad del suelo y de luz ambiental. Estas lecturas se almacenan en variables globales, protegidas por el `sensorMutex`. Después de cada lectura, los valores se envían a través del puerto serial para su procesamiento por el *backend*. Además, esta tarea registra periódicamente los valores de los sensores en la EEPROM utilizando la función `logEvent()`.

La tarea `autoControlTask` implementa la lógica de control autónomo del sistema. Basándose en los umbrales de humedad y luz configurados y las lecturas actuales de los sensores (obtenidas de forma segura usando `sensorMutex`), esta tarea decide si activar o desactivar el sistema de riego y ajustar la intensidad de las luces. El riego se activa si la humedad del suelo es superior al umbral (indicando baja humedad) y se desactiva si es inferior o igual. Las luces se ajustan utilizando una función de mapeo que relaciona inversamente la lectura del sensor de luz con el brillo de las luces LED; a menor luz ambiental, mayor brillo de las luces artificiales. Los cambios de estado significativos, como el encendido/apagado del riego o cambios en el estado de luz baja, se registran en la EEPROM.

El registro de eventos en la EEPROM se gestiona mediante la función `logEvent()`. Cada evento se almacena como una estructura `Event`, que contiene un *timestamp* Unix, un tipo de evento (definido en el enumerado `EventType`, como `AUTO_IRRIGATION`, `MEASURE_MOISTURE`, etc.) y un valor asociado al evento. Antes de escribir, se verifica si hay suficiente espacio utilizable en la EEPROM. Si el espacio es nulo, se envía un error por serial y no se registra nada. Si hay espacio, se toma el `eepromMutex`, se escribe el nuevo evento en la dirección `nextAddr`, y luego `nextAddr` se incrementa. Si `nextAddr` alcanza el final del

área de registro utilizable, se reinicia a `LOG_START_ADDRESS`, implementando así la sobreescritura de los registros más antiguos cuando la memoria se llena.

La función `printLogs()` lee todos los registros válidos de la EEPROM y los envía por el puerto serial, mientras que `clearLogs()` borra todos los registros escribiendo `0xFFFFFFFF` en sus *timestamps* y reiniciando `nextAddr`.

La comunicación con el sistema anfitrión se maneja en la tarea `serialComTask`. Esta tarea lee continuamente los datos entrantes del puerto serial, ensamblando comandos terminados por un carácter de nueva línea o retorno de carro en un búfer (`cmd_buffer`). Una vez que se recibe un comando completo, se pasa a la función `handleCommand()`. Esta función interpreta los comandos recibidos: `'T'` para establecer la hora actual, `'G'` para solicitar los registros, `'D'` para borrar los registros, `'X'` y `'Z'` para solicitar los umbrales de humedad y luz respectivamente, y `'L'` y `'M'` para establecer dichos umbrales. Los comandos desconocidos generan un mensaje de error. Todas las escrituras al puerto serial están protegidas por `serialMutex`.

La tarea `updateTimeTask` se encarga de mantener actualizada la variable `currentTime`, que representa el tiempo Unix actual. Incrementa `currentTime` cada segundo. Además, a intervalos regulares definidos por `timeSaveInterval` (15 minutos), guarda el valor de `currentTime` en la dirección `EEPROM_ADDR_CURRENT_TIME` de la EEPROM, asegurando la persistencia del tiempo entre reinicios. Esta operación también está protegida por `eepromMutex`.

Aplicación Backend (`web_server.py`)

La aplicación *backend*, desarrollada en Python utilizando el *framework* Flask y la extensión Flask-SocketIO, actúa como un puente entre el microcontrolador Arduino y la interfaz de usuario web. Sus responsabilidades incluyen la comunicación serial bidireccional con el Arduino, la sincronización horaria, el procesamiento de datos y la transmisión en tiempo real de información al *frontend* mediante WebSockets.

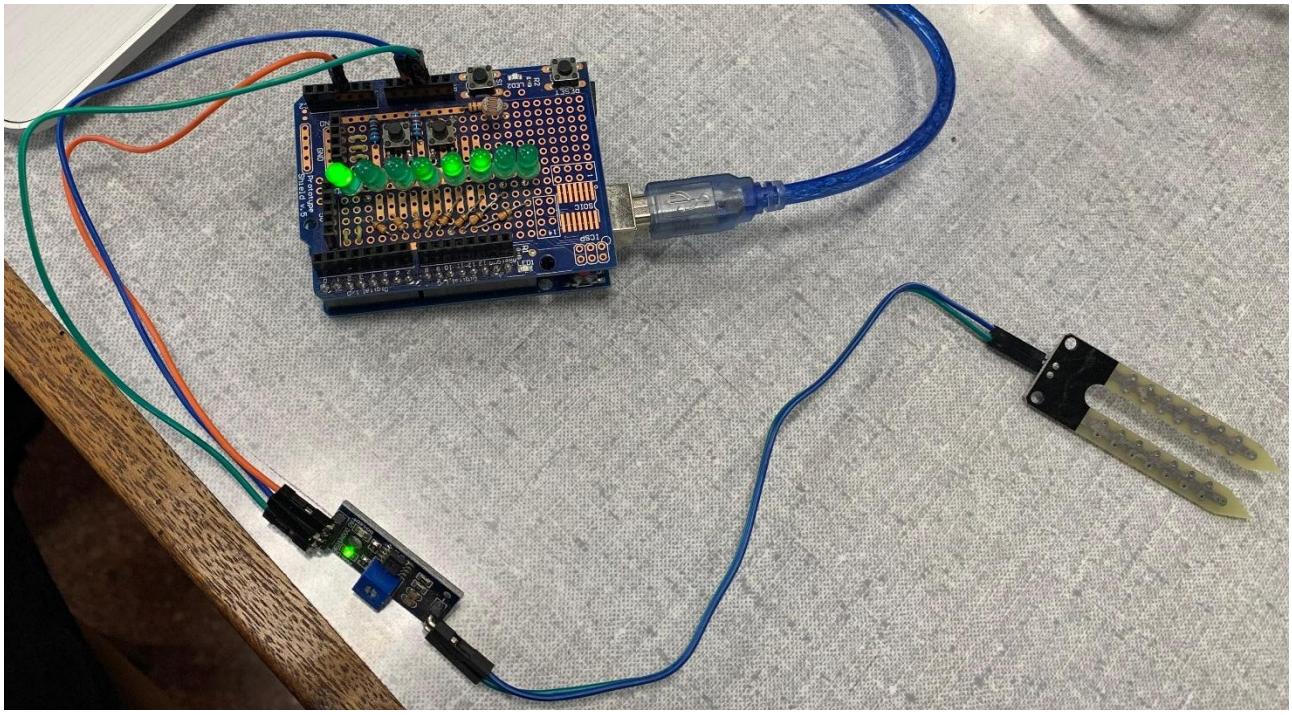

Figura 1: Foto del Arduino en funcionamiento con los sensores de humedad y luz.

Al iniciarse, `web_server.py` intenta establecer una conexión serial con el Arduino utilizando los parámetros definidos en `config.py`. Si la conexión es exitosa, se realizan varias acciones iniciales: se resetea el búfer de entrada del Arduino para eliminar datos obsoletos, se intenta una sincronización horaria inicial y se solicitan los umbrales de humedad y luz actuales del Arduino.

La sincronización horaria se realiza obteniendo la hora de un servidor NTP mediante la librería `ntplib`, ajustándola según `TIME_ZONE_OFFSET` para obtener un *timestamp* Unix, y enviándola al Arduino con el comando `'T'`. Esta sincronización se repite periódicamente en un hilo separado (`sync_time`) para mantener la precisión horaria del Arduino. Otro hilo, `serial_reader`, se dedica a leer continuamente los datos provenientes del Arduino.

La función `serial_reader` se encarga de la recepción de datos. Lee líneas de texto del puerto serial y las procesa. Utiliza expresiones regulares para identificar diferentes tipos de mensajes: datos de sensores (formato `m{humedad}l{luz}`), respuestas a solicitudes de umbrales (formato `X{umbralHumedad}` o `Z{umbralLuz}`), encabezados de logs (`TS,T,V`),

datos de logs individuales (formato `timestamp,tipo,valor`) y el marcador de fin de logs (`E`).

Cuando se reciben datos de sensores, se emiten al *frontend* a través de un evento SocketIO llamado `sensor_update`. Las actualizaciones de umbrales recibidas del Arduino se emiten como `threshold_update`. Cuando se detecta el encabezado de logs, el *backend* entra en un modo de lectura de logs, acumulando cada entrada de log en una lista. Al recibir el marcador de fin de logs, la lista completa de logs se envía al *frontend* mediante el evento `log_data`. Los mensajes de error o no reconocidos por el Arduino (prefijo `'U'`) también se retransmiten al *frontend*. Un Lock de threading (`lock`) se utiliza para sincronizar el acceso de escritura al puerto serial desde diferentes hilos o manejadores de eventos de SocketIO, evitando así la corrupción de los comandos enviados al Arduino.

La aplicación Flask define una ruta principal (`'/'`) que sirve el archivo `index.html`. Las interacciones en tiempo real se manejan mediante eventos de SocketIO. Cuando el *frontend* emite un evento `threshold_update`, el *backend* recibe el nuevo valor del umbral, lo valida (asegurándose de que esté dentro del rango 0-1023) y envía el comando correspondiente (`'L'` o `'M'` seguido del valor) al Arduino. Si el comando se envía con éxito, el *backend* actualiza su copia local del umbral y retransmite la actualización a todos los clientes conectados para mantener la sincronización. Las solicitudes de logs (`logs_request`) y de borrado de logs (`clear_logs_request`) desde el *frontend* se traducen en los comandos `'G'` y `'D'` respectivamente, que se envían al Arduino. Al conectarse un nuevo cliente, el *backend* le envía los valores actuales de los umbrales almacenados.

Interfaz de Usuario Frontend (`index.html`, `styles.css` y `script.js`)

La interfaz de usuario *frontend* proporciona una visualización interactiva de los datos del jardín y permite al usuario controlar ciertos aspectos del sistema. Está construida con HTML para la estructura, CSS para la presentación visual y JavaScript para la lógica y la comunicación en tiempo real.

El archivo `index.html` define la estructura de la página web. Incluye áreas para mostrar gráficos de los sensores de humedad y luz, valores numéricos actuales de estos sensores, controles deslizantes (*sliders*) para ajustar los umbrales de humedad y luz, y una sección para visualizar los registros de eventos. Se utilizan librerías externas como jQuery para simplificar la manipulación del DOM, Socket.IO para la comunicación WebSocket con el *backend* y Chart.js para la representación gráfica de los datos de los sensores. Se incluye un componente tipo *snackbar* para mostrar notificaciones no intrusivas.

El archivo `script.js` contiene la lógica del lado del cliente. Al cargar la página, se establece una conexión Socket.IO con el servidor *backend* y se inicializan los gráficos de Chart.js. El script maneja varios eventos de Socket.IO provenientes del *backend*. Cuando se recibe un evento `sensor_update`, la función `updateDisplay` actualiza los valores numéricos de humedad y luz en la página y añade los nuevos datos a los gráficos. Un evento `threshold_update` del *backend* actualiza la posición de los sliders de umbral y sus valores mostrados, asegurando que la interfaz refleje el estado actual configurado en el Arduino.

Cuando el usuario interactúa con los controles deslizantes de umbral, la función `updateThresholdDisplay` actualiza el valor numérico mostrado junto al slider en tiempo real. Al soltar el slider, la función `sendThresholdUpdate` (envuelta en una función `debounce` para evitar el envío excesivo de eventos) emite un evento `threshold_update` al *backend* con el nuevo valor del umbral. Los botones "Get Logs" y "Clear Logs" emiten los eventos `logs_request` y `clear_logs_request` al *backend*, respectivamente.

Figura 2: Captura de la interfaz de usuario. Se observa los gráficos de los sensores de humedad y luz, y los valores numéricos actuales de estos sensores.

La recepción de datos de logs (`log_data`) desencadena la actualización de la tabla de logs en la interfaz. Cada entrada de log se formatea y se añade como una fila en la tabla, mostrando el *timestamp* (convertido a formato legible), el tipo de evento y el valor.

Para la retroalimentación al usuario, se implementó la función `showSnackbar(message, type)`. Esta función crea dinámicamente un elemento `div` para el *snackbar* y lo muestra con una animación. El *snackbar* desaparece automáticamente después de unos segundos. Esta función se utiliza para mostrar mensajes informativos (como "Requesting logs...") y errores (como los recibidos del *backend* a través del evento error de Socket.IO).

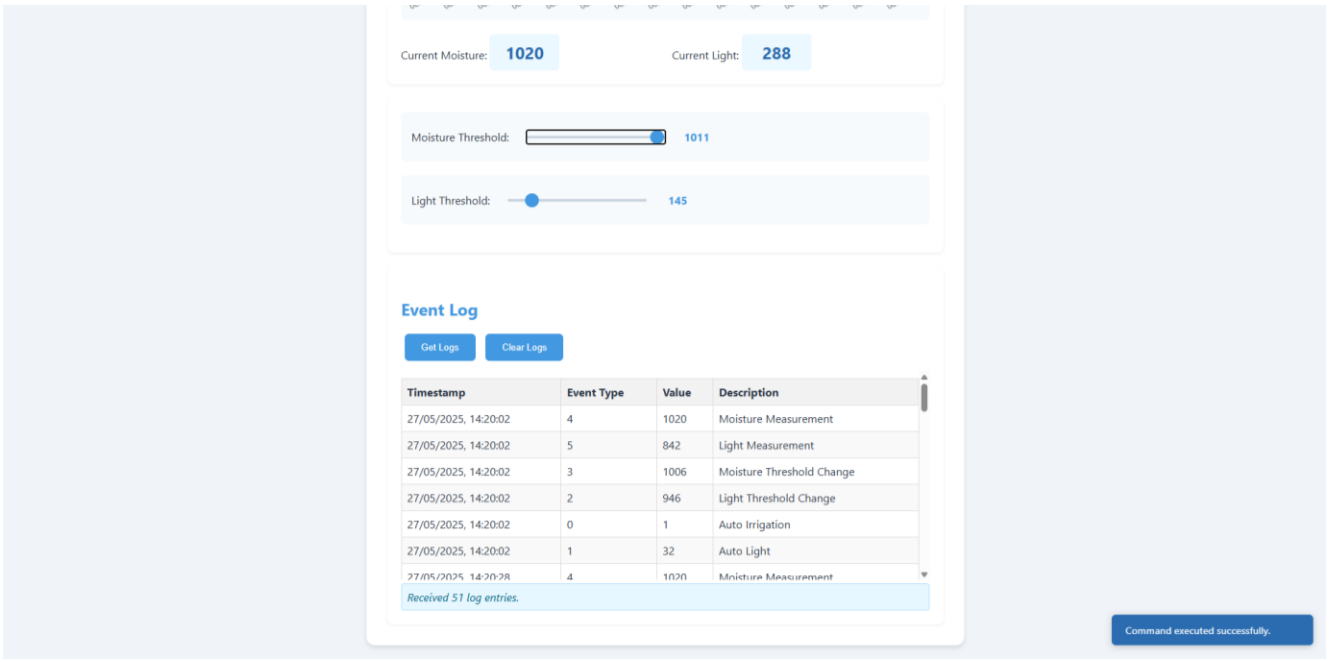

Figura 3: Captura de la interfaz de usuario. Se muestra los controles deslizantes (sliders) para ajustar los umbrales de humedad y luz, y una sección para visualizar los registros de eventos. También se visualiza en la esquina inferior derecha una notificación mediante el componente "snackbar".