PROYECTO SISTEMA SCADA

Ingeniería en Computación I

27 DE NOVIEMBRE DE 2024

CHIABO FRANCO, KÜHN LAUTARO, PALOMEQUE MATEO
Profesores: Javier Fornari, Milton Pozzo, Hernán Bettello, Fabio Rocchi

Índice:

Introducción:

En la materia Ingeniería en Computación 1, se planteó la realización de un proyecto integrador que aplicara los conocimientos adquiridos durante el cursado. Este proyecto debía incorporar un dispositivo digital conectado a componentes electrónicos, comunicándose mediante un programa desarrollado en Arduino. Además, se requería que el sistema recibiera órdenes y valores desde un dispositivo móvil, mediante conexión Bluetooth o WiFi. La finalidad del trabajo era poner en práctica la teoría aprendida y los conceptos de asignaturas como Física, Química y Análisis Matemático.

Nuestro grupo decidió trabajar con fluidos, implementando un sistema SCADA (Supervisión, Control y Adquisición de Datos), una tecnología fundamental en la industria 4.0, que optimiza la supervisión y el control de procesos a distancia. Nos ubicamos en un contexto hipotético en el que nuestra "empresa" recibe el encargo de diseñar y automatizar una planta de producción de jugos.

La maqueta diseñada para representar la planta incluye:

- Dos depósitos de almacenamiento para diferentes líquidos.
- Un bombo de mezcla, donde convergen los fluidos provenientes de los depósitos, en proporciones ajustables según la receta.
- Un depósito de entrada, que simula un camión transportador.
- Un depósito de salida, que representa el contenedor final, listo para la línea de ensamblaje o despacho.

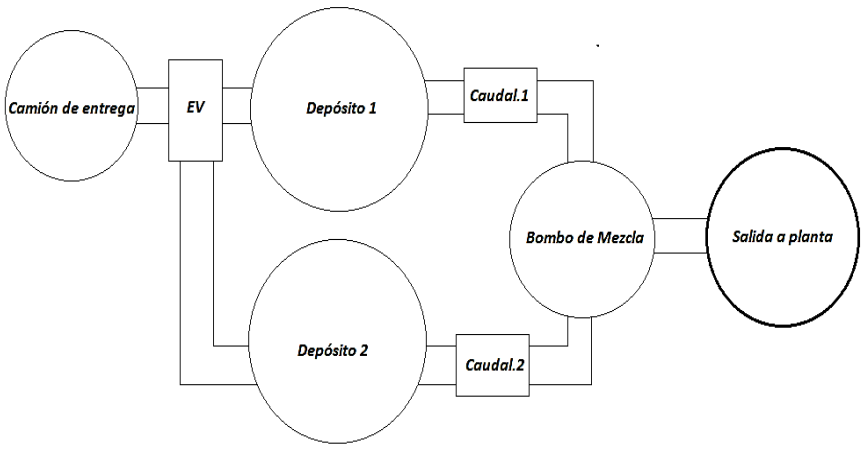


Figura 1: Boceto inicial del sistema.

Para automatizar la planta, se incorporaron diferentes sensores y actuadores:

- Caudalímetros, para medir el flujo de los líquidos.
- Sensores ultrasónicos, para registrar el nivel de llenado de los depósitos.

- Bombas de agua, un motor de corriente continua, relés y electroválvulas, que permiten realizar las acciones requeridas.

El procesamiento de datos y la comunicación a distancia se logran mediante un Arduino Mega 2560 y un módulo Bluetooth, integrados con un programa en Arduino y una aplicación móvil diseñada para maximizar las funcionalidades del sistema.

Todos estos componentes conforman el hardware del sistema, complementado por el desarrollo del software necesario. Para facilitar el montaje y la experimentación, se utilizaron experimentadores y cables unipolares y multipolares, evitando la fabricación de placas electrónicas. Esto permitió construir una maqueta funcional para visualizar y analizar los procesos implementados.

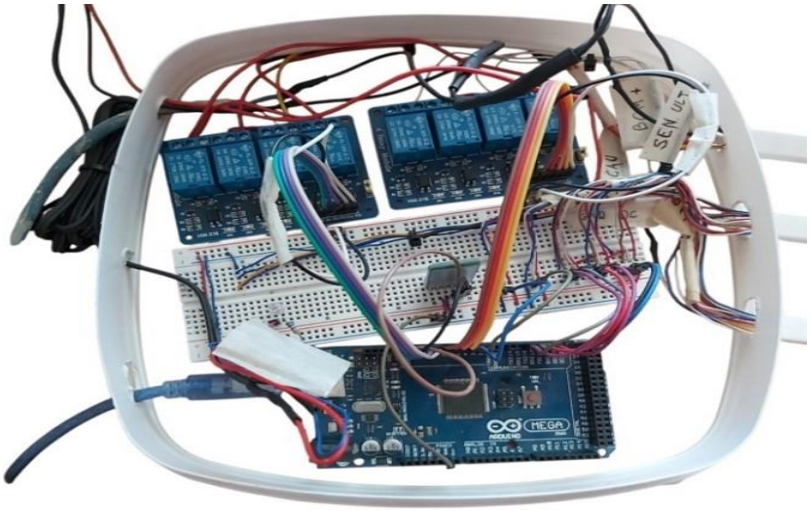


Figura 2: Centro de control internamente.

Figura 3: Sistema SCADA completo.

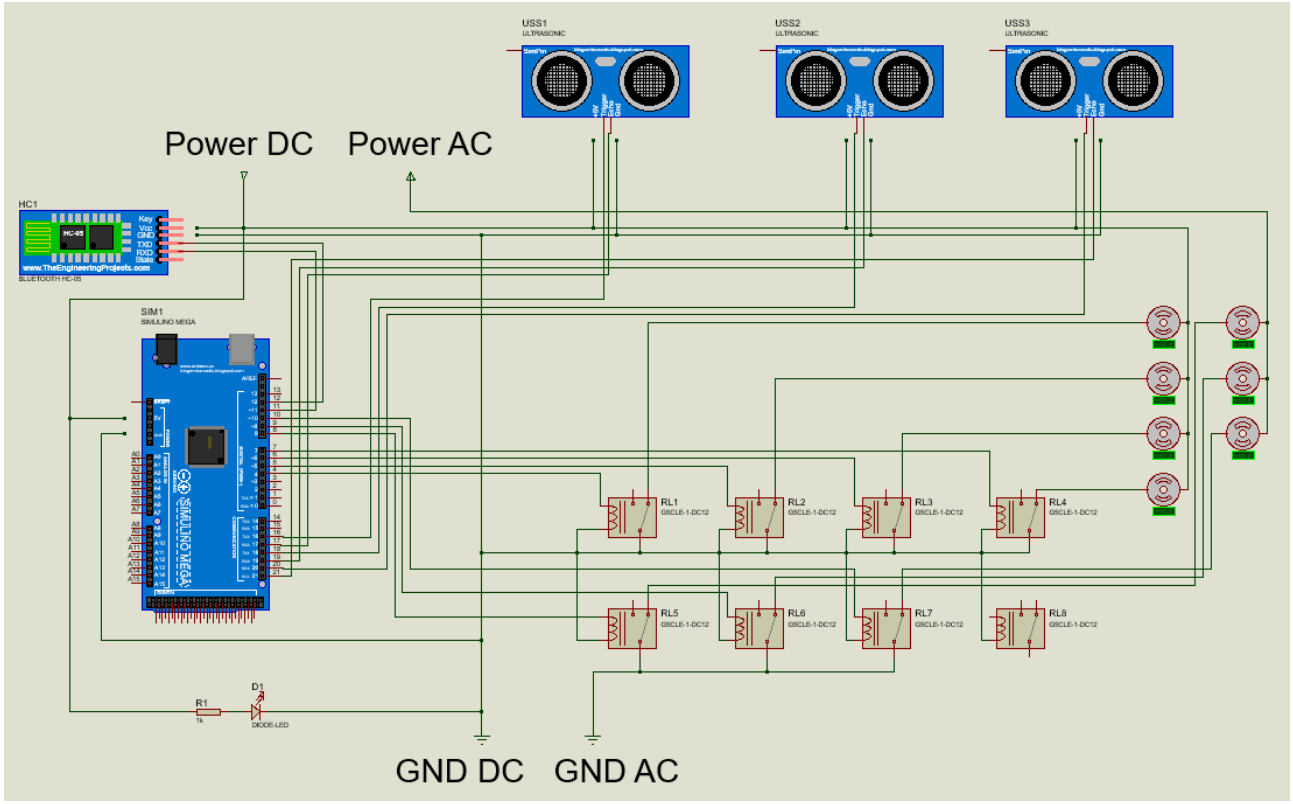


Figura 4: Esquema del sistema elaborado en Proteus 8.

Conceptos:

Para empezar, nos gustaría explicar o detallar algunos conceptos y características de los componentes, dispositivos e incluso metodologías que se utilizaron.

Sistema SCADA:

SCADA (Supervisory Control and Data Acquisition) es un sistema de control y supervisión utilizado en la industria para monitorear, recopilar y procesar datos en tiempo real, así como para controlar remotamente equipos y dispositivos de manera automática. Estos sistemas son fundamentales en entornos donde se requiere un alto grado de automatización y control eficiente de procesos. Permite a los operadores tener una visión completa y detallada de lo que sucede en la planta o sistema que están supervisando, además de la capacidad de intervenir cuando sea necesario para ajustar parámetros o responder a alarmas.

Los sistemas SCADA se utilizan principalmente para mejorar la eficiencia, seguridad y control de procesos industriales y de infraestructuras críticas. Algunas de las principales finalidades son:

- **Monitoreo y Supervisión en Tiempo Real:** Proporciona una vista en tiempo real de los procesos industriales, permitiendo a los operadores detectar rápidamente cualquier anomalía o condición anormal en el sistema. Esta supervisión constante ayuda a mejorar la eficiencia operativa y a reducir el tiempo de respuesta ante problemas.

- **Control Remoto de Equipos:** Facilita el control de dispositivos distribuidos geográficamente, lo que es esencial en sectores como el de agua y saneamiento, distribución de energía, petróleo y gas. Por ejemplo, en nuestro caso, el sistema SCADA puede activar o desactivar bombas de agua en estaciones remotas desde una ubicación central, reduciendo la necesidad de enviar personal al sitio.
- **Optimización de Procesos:** La recopilación de datos y la capacidad de analizar el rendimiento a lo largo del tiempo permite optimizar los procesos y ajustar parámetros para reducir costos operativos y mejorar la eficiencia. Permite identificar cuellos de botella o áreas de mejora en el sistema, ayudando a maximizar el rendimiento de la planta o red.
- **Gestión de Alarmas y Seguridad:** SCADA permite configurar alarmas que se activan cuando una variable excede los límites predefinidos, alertando a los operadores sobre problemas potenciales. También puede ser configurado para realizar acciones automáticas cuando se detectan condiciones críticas, como la desactivación de las bombas de agua en caso de superar la capacidad de los bombos, lo que mejora la seguridad del sistema.
- **Reducción de Costos de Operación:** Al permitir el control remoto y reducir la necesidad de supervisión física constante, un sistema SCADA puede disminuir los costos de mantenimiento y operación. Además, su capacidad para optimizar procesos reduce el consumo de energía y recursos, lo cual es especialmente relevante en industrias con alto consumo energético.
- **Registro Histórico y Generación de Informes:** Los sistemas SCADA almacenan datos históricos que son útiles para la generación de informes de producción, mantenimiento, eficiencia, y otros indicadores clave de desempeño. Esto facilita el cumplimiento normativo y la auditoría, especialmente en sectores regulados como el agua, la energía y la industria farmacéutica.

Este tipo de sistemas utiliza protocolos de comunicación, formatos de transmisión y dispositivos especializados, diseñados para ser inmunes al ruido del ambiente industrial, robustos mecánicamente y simples en cuanto a software, con el fin de simplificar y garantizar la comunicación. Estas características especializadas llevan a que los dispositivos utilizados tengan un alto coste, pero a cambio garantizan fiabilidad y desempeño en entornos adversos.

Generalmente, como unidades de procesamiento se utilizan PLC (Controladores Lógicos Programables). No se emplean comunicaciones inalámbricas en grandes áreas debido al ruido electromagnético característico del ambiente industrial; en su lugar, se utilizan comunicaciones cableadas que implementan protocolos como Modbus RTU, OPC UA o Profinet. Estos protocolos se montan sobre plataformas de comunicación físicas como RS-485, RS-232 o Ethernet industrial.

En el área de producción, estas configuraciones garantizan una transmisión de datos segura y estable. Cuando los datos deben escalarse hacia otras áreas o conectarse al internet, se emplean

tecnologías como MQTT u otras plataformas diseñadas para la transmisión eficiente y confiable de datos a nivel empresarial o en la nube.

Microcontrolador:

Un microcontrolador es un circuito integrado que actúa como una pequeña computadora programable. Está diseñado para ejecutar tareas específicas de manera eficiente y con bajo consumo de energía en sistemas embebidos y aplicaciones de automatización. Un microcontrolador normalmente contiene:

- **Unidad Central de Procesamiento (CPU):** Su función es interpretar y ejecuta las instrucciones del programa mediante la realización de las operaciones básicas aritméticas, lógicas y externas.
- **Memoria:** Incluye memoria de programa (ROM/Flash) y memoria de datos (RAM) para almacenar tanto el código como los datos temporales.
- **Periféricos de Entrada/Salida (I/O):** Permiten interactuar con otros componentes y dispositivos, como sensores y actuadores.
- **Temporizadores y Contadores:** Controlan el tiempo y eventos en el microcontrolador.
- **Convertidores A/D y D/A:** Permiten la conversión de señales analógicas a digitales y viceversa para trabajar con sensores y otros dispositivos.

Componentes

Arduino MEGA 2560:

Luego de explicar que es un microcontrolador, será más sencillo entender que es una placa de Arduino Mega 2560 y porque fue este el modelo elegido para el desarrollo de este proyecto.

Figura 5: Placa Arduino MEGA 2560.

Este dispositivo es una placa de desarrollo basada en el microcontrolador ATmega2560, diseñada para aplicaciones más complejas que requieren más pines de entrada/salida (I/O), memoria y capacidad de procesamiento que otras placas de la familia Arduino, como el Arduino Uno que fue la principal sugerencia para trabajar al inicio de este curso. Sus principales características son:

- **Microcontrolador:** ATmega2560, con una arquitectura de 8 bits.
- **Voltaje de operación:** 5V, aunque algunos pines soportan 3.3V para compatibilidad con sensores.

- **Voltaje de entrada (recomendado):** 7-12V, a través de un conector de alimentación externa o mediante un puerto USB.
- **Memoria:**
  - **Memoria Flash:** 256 KB para almacenar el programa, de los cuales 8 KB son utilizados por el bootloader.
  - **SRAM:** 8 KB para el almacenamiento de variables temporales durante la ejecución del programa.
  - **EEPROM:** 4 KB, que permite guardar datos de manera persistente.
- **Pines de Entrada/Salida (I/O):**
  - **54 pines digitales:** De los cuales 15 pueden ser utilizados como salidas PWM (Modulación por Ancho de Pulso).
  - **16 pines analógicos:** Para lectura de señales analógicas mediante un convertidor A/D de 10 bits de resolución.
  - **4 puertos UART**: Permiten la comunicación serial (RX/TX) con otros dispositivos.
- **Frecuencia de reloj:** 16 MHz, lo que permite un buen equilibrio entre velocidad y consumo de energía.
- **Conectividad:** Incluye un puerto USB tipo B para programación y comunicación con una computadora.
- **Conector ICSP (In-Circuit Serial Programming):** Para la programación directa del microcontrolador y cargar el bootloader si es necesario.
- **Tamaño:** 101.52 mm x 53.3 mm.

Como se explica y da a entender mediante sus características, esta placa fue seleccionada por encima de las demás debido a su capacidad de procesamiento, el tamaño de su memoria ya que requeríamos de realizar un código de una dimensión mayor a lo habitual, y también por la cantidad de pines que nos permite trabajar con muchos sensores, actuadores y módulos de comunicación.

Módulo Bluetooth:

Para la comunicación inalámbrica con el dispositivo móvil hemos decidido utilizar un módulo bluetooth llamada Serial HC-05 que nos facilita la comunicación a distancia. Este componente se puede utilizar en modo maestro o esclavo. El HC-05 tiene 6 pines, status, RXD, TXD, GND, VCC y EN. Los pines RXD y TDX se conectan a los pines digitales para realizar la comunicación con el Arduino, el pin VCC a 5 voltios y el pin GND a la tierra. Es muy usado en el para dar conectividad inalámbrica a través de

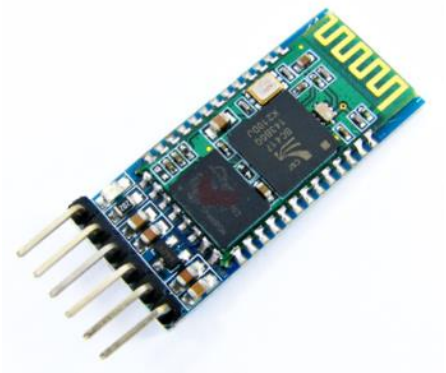


Figura 6: Módulo Bluetooth HC-05.

una interfaz serial TTL entre Microcontroladores y otros dispositivos como PC, laptops o celulares Smartphone.

Las principales características que nos brinda este módulo son:

- **Protocolo:** Bluetooth especificación V2.0+EDR
- **Protocolo comunicación:** UART
- **Tensión de comunicación:** 3,3V
- **Tensión de Alimentación:** 5V
- **Frecuencia:** 2.4Ghz Banda ISM
- **Velocidad de transmisión en baudios ajustable:** 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
- **Configuración por Defecto:** 9600 baud rate, N, 8,1, contraseña 1234
- **Distancia Cobertura Bluetooth:** Aproximadamente 10 metros
- **Tamaño compacto:** 4.3cm x 1.6cm x 0.7cm

Comunicación UART:

Como se aclara en las características de HC-05, este componente utiliza un protocolo de comunicación UART. Un UART (Universal Asynchronous ReceiverTransmitter) es un tipo de circuito integrado que se usa para enviar y recibir datos a través de un puerto serie en un equipo o dispositivo periférico. Los UART son ampliamente utilizados y conocidos por su sencillez. Sin embargo, a diferencia de SPI e I2C, los UART no admiten múltiples dispositivos subordinados.

La comunicación UART no requiere una señal de reloj compartida. En su lugar, utiliza bits de inicio y parada para definir los límites de cada paquete de datos. Los datos se transmiten bit a bit. Un paquete típico incluye un bit de inicio que indica el comienzo de un paquete, los bits de datos que generalmente son entre 5 y 9 bits por paquete, un bit de paridad (opcional) para detección de errores y por último un bit de parada que pueden ser uno o dos bits que indican el final del paquete. La velocidad de transmisión, medida en baudios (bits por segundo), debe ser la misma para el transmisor y el receptor. Los UARTs suelen permitir comunicación Full Duplex, lo que significa que pueden enviar y recibir datos simultáneamente a través de dos líneas diferentes.

Módulo Relé de 4 Canales 5V:

Figura 7: Módulo Relé de 4 canales 5V.

Un módulo relé de 4 canales es un componente electrónico diseñado para controlar dispositivos de alta potencia mediante una señal de baja potencia, esto permite encender o apagar dispositivos más grandes desde un microcontrolador. En este caso, hablamos de un módulo con 4 relés que operan a 5V y pueden manejar cargas de hasta 10A cada uno.

Características Principales del Módulo Relé de 4 Canales 5V

- **Voltaje de Operación**: Funciona con una alimentación de **5V** en el lado de la bobina, lo que lo hace compatible con la mayoría de los microcontroladores.
- **Corriente Máxima por Canal**: Cada uno de los 4 relés puede manejar cargas de hasta **10A** a **250V AC** o **30V DC**. Esto significa que pueden controlar dispositivos de alta potencia.
- **Número de Canales**: Tiene **4 canales**, lo que permite controlar 4 dispositivos de manera independiente desde un solo módulo. Cada relé puede ser controlado individualmente por un microcontrolador, lo que ofrece flexibilidad en la gestión de los dispositivos conectados.
- **Interfaz de Control**:
    - Las entradas de control del módulo suelen estar etiquetadas como **IN1, IN2, IN3, IN4**, cada una de las cuales activa el relé correspondiente.
    - Las entradas de control son de bajo voltaje (5V), por lo que se pueden conectar directamente a las salidas digitales de un microcontrolador.
    - Algunas versiones tienen un **pin de señal de activación** con **lógica activa baja**, lo que significa que el relé se activa cuando la señal de control es baja (0V) y se desactiva cuando es alta (5V).
- **Aislamiento**: El módulo de relé incluye **optoacopladores**, que son dispositivos que utilizan luz para transferir la señal eléctrica entre dos circuitos aislados. Esto proporciona un aislamiento seguro entre el lado de control (microcontrolador) y el lado de alta potencia, protegiendo el microcontrolador de picos de voltaje o interferencias que puedan provenir de los dispositivos conectados.
- **Indicadores LED**: Cada canal tiene un LED indicador que se enciende cuando el relé está activado, lo que facilita la identificación del estado de cada relé.
- **Bornes de Conexión**: Cada relé cuenta con un bloque de terminales de 3 pines: **COM** (común), **NO** (normalmente abierto), y **NC** (normalmente cerrado).
    - **COM**: Es el punto común del relé.
    - **NO (Normally Open)**: El circuito está abierto cuando el relé está desactivado y se cierra cuando el relé se activa.

- **NC (Normally Closed)**: El circuito está cerrado cuando el relé está desactivado y se abre cuando el relé se activa.

Mini Bomba de Agua Motor Sumergible DC 3-5V 70-120L/h:

Una mini bomba de agua sumergible DC 3-5V es un pequeño dispositivo diseñado para mover agua de un lugar a otro de manera eficiente. Funciona mediante un motor de corriente continua (DC) que impulsa un rotor o impulsor, el cual genera el flujo de agua.

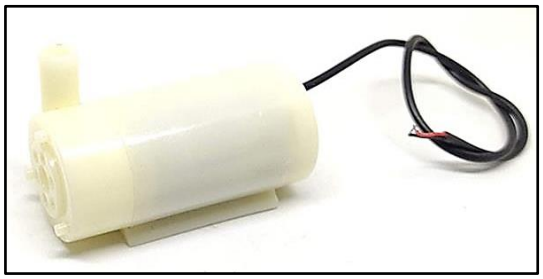


Figura 8: Mini Bomba de Agua 3-5V.

La bomba se conecta a una fuente de alimentación de 3 a 5V DC lo que hace que el motor interno de la bomba comienza a girar, por lo que el impulsor (una pequeña hélice) dentro de la bomba también gire. El giro crea una presión diferencial que succiona agua a través de la entrada de la bomba y la expulsa por la salida. Esto permite que el agua se mueva a través de la bomba a una velocidad determinada.

Características Principales de la Mini Bomba de Agua Sumergible DC 3-5V 70-120L/h:

- **Voltaje de Operación:** Funciona con un rango de 3 a 5V DC. Esto la hace compatible con una amplia variedad de fuentes de energía de bajo voltaje.
- **Consumo de Corriente:** Normalmente consume entre 100 a 200 mA de corriente, lo que la hace eficiente en cuanto al consumo energético.
- **Caudal de Agua:** La bomba tiene un caudal de 70 a 120 litros por hora (L/h), lo que significa que puede mover esta cantidad de agua dependiendo del voltaje y la altura de elevación. A mayor voltaje, el caudal aumenta, mientras que a menor voltaje disminuye.
- **Altura Máxima de Elevación:** La bomba puede elevar el agua a una altura de entre 0.4 a 1.1 metros, dependiendo de la potencia aplicada. Esto es suficiente para pequeñas aplicaciones como fuentes, sistemas de enfriamiento, y riego de plantas.
- **Dimensiones:** Generalmente, estas bombas son compactas, con dimensiones aproximadas de 45 mm x 24 mm x 30 mm, lo que facilita su integración en espacios reducidos.
- **Peso:** Al ser una bomba pequeña, suele pesar alrededor de 30 a 50 gramos, lo que la hace fácil de transportar e instalar.
- **Material y Diseño:** Están construidas con materiales plásticos resistentes al agua (como ABS) para evitar la corrosión y asegurar la durabilidad cuando se sumergen. El diseño está pensado para permitir la circulación del agua de forma eficiente sin fugas.

Bomba De Agua Sobo Wp 4000 - 2000 L/h:

Similar a las mini bombas de agua esta es una versión mejorada de estas, esta tiene un mayor tiraje de agua y una mayor presión que las anteriores. A cambio de obtener en la cantidad y presión de agua que puede manejar esta utiliza 220V de AC, a diferencia de las anteriores que usan 5V de DC.

Figura 9: Bomba de Agua 4000 – 2000 L/h.

Este tipo de bomba se usa en peceras y en fuentes decorativas de patio, nosotros la usamos para que realice el vaciado del “camión” y envía el liquido a los bombos de almacenamiento. Era necesario utilizar una bomba mas potente para esta etapa del sistema ya que entre los bombos hay una electroválvula que hace necesario tener mas fuerza de empuje del agua para que pase por la que esta abierta, con las primeras bombas el agua no podía pasar por la electroválvula ya que no tienen fuerza suficiente.

Electroválvula Doble:

Una electroválvula doble es un dispositivo electromecánico utilizado para controlar el flujo de líquidos en sistemas de tuberías mediante la acción de una señal eléctrica. El contenido entra por un único conducto, y la válvula decide a cuál de las dos salidas dirigir el flujo. Este tipo de válvula es útil cuando se quiere dividir el flujo o redirigirlo a dos lugares distintos de manera alternada o selectiva dependiendo de cuál de los solenoides esté activado, el fluido se redirige a una de las dos salidas. Si se activa el solenoide 1, el fluido fluye hacia la salida 1 y si se activa el solenoide 2, el fluido fluye hacia la salida 2. Es posible, en algunos diseños, que ambas salidas estén cerradas simultáneamente si ambos solenoides están desactivados.

Figura 10: Electroválvula Doble.

Características Principales de la Electroválvula Doble:

- **Tensión de Alimentación:** Las electroválvulas suelen operar con voltajes de 12V, 24V, 110V, o 220V en corriente continua (DC) o corriente alterna (AC). La elección del voltaje depende del sistema en el que se va a integrar la válvula.
- **Tipo de Fluido Compatible:**
    - **Agua:** La mayoría de las electroválvulas dobles están diseñadas para controlar el flujo de agua.

  - **Gases:** Algunas electroválvulas están diseñadas para manejar gases como el aire comprimido, dependiendo de su construcción y materiales de sellado.
- **Materiales de Fabricación:**
  - **Cuerpo de la Válvula:** Puede estar fabricado en latón, acero inoxidable, o plástico de alta resistencia. El material se selecciona en función del tipo de fluido y la presión de trabajo.
  - **Juntas de Sellado:** Usualmente hechas de materiales como nitrilo, silicona, o EPDM, dependiendo del tipo de fluido que manejen y la temperatura de operación.
- **Tamaño de las Conexiones:** Las conexiones estándar pueden variar, siendo comunes las de 1/2 pulgada, 3/4 pulgada, y 1 pulgada en roscas BSP o NPT. El tamaño de las conexiones determina el caudal máximo que puede manejar la válvula.
- **Presión de Trabajo:** Las electroválvulas dobles están diseñadas para operar dentro de un rango de presiones, por ejemplo, de 0.1 a 10 bares, dependiendo del modelo. Esto es importante para garantizar un funcionamiento adecuado en sistemas de alta o baja presión.
- **Dos Vías de Control:** Cada una de las dos válvulas de la electroválvula doble puede ser controlada de forma independiente mediante una señal eléctrica, lo que permite:
  - Controlar dos líneas de entrada para mezclar líquidos.
  - Desviar el flujo de una sola entrada hacia dos salidas diferentes.
  - Controlar el paso de fluido en dos circuitos separados, lo que aumenta la versatilidad en la gestión del flujo.

Sensor de Caudal (Caudalímetro) 0.3-6L/min ¼:

Un sensor de caudal o caudalímetro es un dispositivo diseñado para medir la cantidad de líquido que fluye a través de una tubería en un periodo de tiempo específico. En este caso, el sensor tiene un rango de medición de 0.3 a 6 litros por minuto (L/min) y una conexión de 1/4 de pulgada, lo cual se refiere al diámetro del conector de la tubería.

Figura 11: Caudalímetro ¼".

Un caudalímetro de tipo hall-effect (efecto Hall), que es el más común para estos rangos de caudal, funciona mediante el líquido que fluye a través del cuerpo del sensor, lo cual hace girar un pequeño rotor o turbina interna. Esta turbina tiene aletas que giran proporcionalmente a la velocidad del flujo del líquido. Dentro del sensor, hay un imán incrustado en la turbina, y un sensor de efecto Hall que detecta cada vez que el imán pasa frente a él. Cada giro del imán genera un pulso eléctrico que es capturado por el sensor. Estos

pulsos son enviados a un microcontrolador, que los cuenta y los convierte en una lectura de caudal. La frecuencia de los pulsos generados es directamente proporcional al caudal de líquido que pasa a través del sensor. Cuanto mayor es el flujo, más rápida es la rotación de la turbina y, por lo tanto, más alta es la frecuencia de los pulsos. Con una calibración previa, el microcontrolador puede convertir la cantidad de pulsos en una lectura de litros por minuto (L/min).

Características Principales del Sensor de Caudal (Caudalímetro) 0.3-6L/min ¼:

- **Rango de Medición:** Mide flujos de 0.3 a 6 litros por minuto (L/min). Esto lo hace adecuado para aplicaciones donde se manejan flujos de baja a moderada cantidad de agua.
- **Conexión de 1/4 de Pulgada:** La rosca de 1/4 de pulgada es estándar para tuberías pequeñas. Esto facilita la conexión del sensor a mangueras o tuberías de este diámetro.
- **Tensión de Operación:** Normalmente, este tipo de sensores opera a un voltaje de 5V DC, lo que los hace compatibles con microcontroladores y otros sistemas de control electrónico.
- **Salida de Señal:** Proporciona una salida de pulsos digitales que representan la velocidad de flujo del líquido. Cada pulso corresponde a una cierta cantidad de agua que pasa a través del sensor.
- **Precisión:** La precisión del sensor depende del modelo, pero generalmente puede tener un margen de error de entre 1% y 5% en la medición del flujo, lo que es adecuado para la mayoría de las aplicaciones no industriales de alta precisión.
- **Materiales y Construcción:** El cuerpo del sensor suele estar fabricado en plástico de alta resistencia o nylon para garantizar la durabilidad y resistencia a la corrosión. El diseño interno asegura que el flujo de agua no se vea significativamente afectado por la presencia del sensor, manteniendo una pérdida mínima de presión.

Sensor Ultrasónico HC-SR04:

El HC-SR04 es un sensor ultrasónico ampliamente utilizado para medir distancias de manera precisa y sin contacto. Funciona mediante la emisión y recepción de ondas ultrasónicas, lo que permite detectar la distancia a la que se encuentran objetos en su entorno.

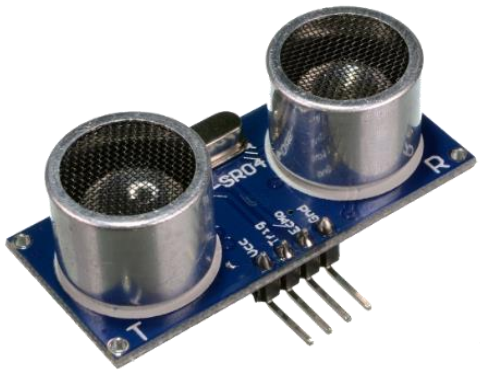


Figura 12: Sensor Ultrasónico HC-SR04.

El funcionamiento del HC-SR04 se basa en el principio del eco de las ondas ultrasónicas. El sensor cuenta con dos componentes principales: el transmisor (ultrasonic transmitter) y el receptor (ultrasonic receiver).

Cuando el sensor es activado, el transmisor emite una onda ultrasónica de 40 kHz (fuera del rango audible para los humanos). Esta onda viaja a través del aire hasta que choca con un objeto

y se refleja de vuelta hacia el sensor. El receptor del HC-SR04 capta la onda reflejada y mide el tiempo que tarda en regresar desde el objeto hasta el sensor. Con la *fórmula Distancia = (Tiempo x Velocidad del sonido) / 2*, se calcula la distancia al objeto. La velocidad del sonido en el aire es de aproximadamente 343 metros por segundo a 20°C. La división por 2 se debe a que el tiempo medido corresponde al recorrido de ida y vuelta de la onda. La distancia calculada se da en centímetros (cm) o milímetros (mm) según el procesamiento de la señal por el microcontrolador.

Características Principales del Sensor Ultrasónico HC-SR04:

- **Rango de Medición:** Mide distancias desde 2 cm hasta 400 cm (4 metros). Esto lo hace adecuado para detectar objetos en un rango relativamente amplio en aplicaciones de corto a medio alcance.
- **Precisión:** La precisión del HC-SR04 es de aproximadamente 3 mm, lo que permite obtener mediciones bastante exactas para la mayoría de las aplicaciones de detección de proximidad.
- **Ángulo de Detección:** Tiene un ángulo de detección de aproximadamente 15 grados, lo que significa que es más efectivo cuando se apunta directamente hacia el objeto a medir. Esto lo hace ideal para aplicaciones donde se requiere un haz de detección enfocado.
- **Tensión de Alimentación:** Funciona con una tensión de 5V DC, lo que lo hace compatible con microcontroladores y otros sistemas de control de 5V.
- **Interfaz de Conexión:** Tiene cuatro pines:
    - **VCC:** Conexión de 5V para alimentar el sensor.
    - **GND:** Conexión a tierra.
    - **Trigger (TRIG):** Pin de entrada que recibe la señal para iniciar la emisión del pulso ultrasónico.
    - **Echo (ECHO):** Pin de salida que emite una señal de alta (HIGH) durante el tiempo que la onda tarda en regresar. La duración de esta señal se utiliza para calcular la distancia.

Mini Motor DC:

Un mini motor DC (motor de corriente continua) es un motor eléctrico que convierte la energía eléctrica en energía mecánica en forma de rotación, utilizando una fuente de alimentación de corriente continua (DC). Son motores compactos y ligeros. Su tamaño reducido y facilidad de control los hacen ideales para proyectos que requieren un movimiento preciso y de bajo consumo.

Figura 13: Mini Motor DC.

Componentes Internos Básicos:

- **Rotor (Armature):** Es la parte móvil del motor que rota cuando se aplica corriente. Contiene bobinados de cobre.
- **Estator:** Parte fija que genera un campo magnético. En mini motores, el estator suele estar compuesto por imanes permanentes.
- **Colector (Commutator):** Conjunto de segmentos de cobre que ayuda a cambiar la dirección de la corriente en el rotor, permitiendo la rotación continua.
- **Escobillas (Brushes):** Son piezas de contacto que conectan la fuente de alimentación al rotor a través del colector, asegurando la transmisión de corriente.

Características Principales del Mini Motor DC:

- **Tamaño Compacto:** Su diseño pequeño y liviano los hace ideales para proyectos donde el espacio es limitado y se requiere un motor de bajo peso.
- **Voltaje de Operación:** Los mini motores DC suelen funcionar a voltajes bajos, típicamente entre 1.5V y 12V. Esto los hace compatibles con baterías y fuentes de alimentación de baja potencia como las que se utilizan en proyectos de electrónica. Un voltaje más alto generalmente proporciona una velocidad de rotación mayor, mientras que un voltaje más bajo reduce la velocidad.
- **Corriente de Operación:** El consumo de corriente varía según la carga y el tamaño del motor, pero en general, los mini motores tienen un consumo bajo (desde unos pocos miliamperios hasta 1A o 2A en condiciones de carga máxima).
- **Velocidad de Rotación:** Su velocidad de rotación se mide en revoluciones por minuto (RPM), y puede variar ampliamente dependiendo del modelo y el voltaje aplicado. Algunos mini motores pueden alcanzar entre 3000 RPM y 15000 RPM. La velocidad se puede controlar mediante la modificación del voltaje de entrada o utilizando un controlador de velocidad (controlador PWM).
- **Par Motor (Torque):** Aunque los mini motores DC son rápidos, su par motor (la fuerza de giro que pueden generar) es limitado debido a su tamaño compacto. En aplicaciones que requieren un par mayor, es común acoplar estos motores a cajas de engranajes para reducir la velocidad y aumentar el torque.
- **Durabilidad y Mantenimiento:** Las escobillas y el colector son piezas que están en constante contacto, lo que puede provocar un desgaste con el tiempo. Los mini motores DC requieren mantenimiento si se usan de forma prolongada. Existen versiones de mini motores sin escobillas (brushless) que son más duraderas, aunque son más complejas y requieren controladores especializados.

Fuente de Alimentación 5V 3A:

Una fuente de alimentación 5V 3A es un dispositivo que convierte la corriente alterna (AC) de la red eléctrica en corriente continua (DC) con el voltaje adecuado. Esta fuente está diseñada para proveer una alimentación estable y segura a los dispositivos, asegurando su funcionamiento continuo y sin interrupciones.

Figura 14: Fuente de Alimentación de 5V.

La referencia a 3A indica que la fuente de alimentación puede proporcionar un máximo de 3 amperios de corriente mientras que el 5V hace referencia a la tensión que entrega a la salida, a su vez el diseño en forma de cargador de celular ayuda a disminuir el tamaño del sistema en comparación de si usáramos una fuente de alimentación en base a transformadores y cuadro de diodos.

La fuente de alimentación se conecta a la corriente alterna (AC) de la red eléctrica, generalmente de 220V. Mediante un transformador switching, la fuente convierte el voltaje de entrada de la red a un voltaje más bajo adecuado, según lo indicado 5V de DC. El voltaje reducido pasa por un circuito que mediante la conmutación de un circuito convierte la corriente alterna en corriente continua (DC). Luego, se realiza un proceso de filtrado mediante capacitores para eliminar cualquier tipo de ripple o ruido en la señal de salida, proporcionando una corriente continua estable y adecuada para equipos electrónicos sensibles. La fuente de alimentación tiene una capacidad de 3 amperios. Esto significa que puede alimentar múltiples dispositivos simultáneamente, siempre y cuando la suma de los consumos de los dispositivos no exceda los 3A.

Desarrollo:

El funcionamiento de nuestro proyecto consiste que una vez que llega el camión cisterna a descargar la materia prima, el operario a través de la aplicación instalada en un dispositivo móvil y mediante la característica bluetooth podrá comunicarse con el sistema para decidir a qué deposito desea enviar la carga. Una vez seleccionado comenzara a funcionar una bomba que, en conjunto con una electroválvula, se encargara de que la materia prima llegue al destino seleccionado. En caso de que el depósito este lleno o se llene rápidamente aun habiendo materia prima dentro de camión, el proceso se detendrá y se le alertara al operario de la situación.

Por otro lado, gracias a los sensores ultrasónicos además de poder frenar el proceso cuando ocurre la situación mencionada anteriormente, también se puede avisar al operario cuando el nivel de cualquier depósito disminuye de un cierto valor. Permitiendo que se haga un pedido de más materia prima antes de que se termine la reserva y se frene la producción.

En cuanto a la parte de elaboración del producto, con la aplicación en conjunto con los sensores y actuadores tenemos diferentes opciones en cuanto a cómo configurar el proceso de elaboración.

El operario podrá elegir la cantidad de que quiere que venga desde cada deposito, esto es posible gracias a los caudalímetros que nos indican la cantidad de fluido que paso por los tubos de alimentación.

A su vez, mediante la medición obtenida con el sensor ultrasónico colocado en el mezclador, podremos indicarle al operario si vació o no el mezclador una vez terminado el proceso o si la cantidad introducida de líquido supera la capacidad de este.

También con un motor que cuenta con una paleta en la punta podremos realizar una acción de mezclado para revolver el fluido y que este se mezcle y quede homogéneo. El operario atreves de la aplicación podrá establecer cuanto debe durar este proceso o detenerlo por si hay alguna emergencia.

Finalmente, cuando el operario lo disponga, podrá a través de la app, activar el mecanismo de vaciado, que consistirá en que se ponga en marcha un motor que enviara la producción a otro deposito, a una planta de envasado o a otro camión cisterna que se encargara de transportarlo a otro destino.

Explicación del código Arduino Mega 2560:

A lo largo del escrito se mencionó la idea del proyecto, como se iba a desempeñar, los componentes utilizados y cuál era su finalidad, pero para poder culminar con dicho trabajo se requiere de una programación para que el dispositivo digital, el Arduino Mega 2560, sea capaz de realizar las acciones, mediciones y recolección de datos de forma automática. Estos códigos estarán a su vez relacionados con la aplicación móvil diseñada para la comunicación de datos y ejecución de las actividades. Cabe aclarar que se hará mención de esta aplicación, pero será explicada, tanto el diseño como su programación, más detalladamente en la posterioridad del informe.

La programación se realiza en el entorno Arduino IDE que es una plataforma de desarrollo de código abierto utilizada para programar microcontroladores Arduino. Este entorno facilita la escritura, compilación y carga de código en la placa Arduino, permitiendo crear una gran variedad de proyectos electrónicos y de automatización. El siguiente programa ha sido desarrollado en el lenguaje de programación C/C++ propio del Arduino IDE y está diseñado para automatizar el control de niveles de líquido en depósitos, la activación de bombas y la mezcla de diferentes líquidos. Utiliza sensores de nivel (ultrasónicos) para medir los niveles de los depósitos y caudalímetros para medir la cantidad de líquido que pasa a través de las tuberías. La comunicación Bluetooth facilita la interacción con una aplicación móvil para supervisar y controlar el sistema.

Para empezar, se incluyen dos librerías, la primera llamada *TimerFive* se utiliza para configurar el Timer 5 del microcontrolador, permitiendo ejecutar funciones a intervalos de tiempo específicos, en nuestro caso es para poder leer datos provenientes de la comunicación Bluetooth de forma periódica. La segunda, *SoftwareSerial*, permite crear una comunicación serial en pines digitales diferentes del puerto serial por defecto, para mantener disponible la conexión serial principal para la comunicación con la PC.

```
#include <TimerFive.h>
#include <SoftwareSerial.h>
```

Figura 15: Librerías utilizadas.

Esta librería nos permite definir la comunicación Bluetooth mediante los pines 11 y 12, que luego dentro de la función *SoftwareSerial BT()*, definimos al pin 11 para transmitir (Tx) y al pin 12 para recibir (Rx) datos del módulo Bluetooth.

```
int BT_Rx = 12;                /
int BT_Tx = 11;

SoftwareSerial BT(BT_Rx, BT_Tx);
```

Figura 16: Comunicación Bluetooth

A continuación, se realiza la declaración de múltiples variables, las cuales son de diferentes tipos y con finalidades distintas.

Se dispone de dos variables de tipo *int* para la función del caudal, cada variable corresponde a uno de los caudalímetros.

```
// Definir variables para los caudales
int caudal_1 = 0;
int caudal_2 = 0;
```

Figura 17: Variables para caudales.

También contamos con dos variables booleanas para las bombas, al igual que para el caudal, cada variable pertenece a una bomba en específico.

```
// Definir variables para el estado de las bombas
bool bomba_1_encendida = false;
bool bomba_2_encendida = false;
```

Figura 18: Variables para bombas.

Para el proceso de mezcla se definen variables de tipo *unsigned long* lo que significa que la variable puede representar un número entero sin signo de tamaño largo, lo cual ocupa 4 bytes de memoria, y es recomendado para cuando se trabaja con mediciones de tiempo que pueden superar los 32,767 milisegundos, 32 segundos aproximadamente. También está presente una variable booleana.

```
// Definir variables para el proceso de mezcla
long inicio_mezcla = 0;
long duracion_mezcla_ms = 0;
long tiempo_pausado = 0;
bool mezcla_en_progreso = false;
```

Figura 19: Variables para mezcla.

Luego en la función de control se encuentran variables de tipo *int* que nos permitirán iniciar el proceso, pausar el proceso, vaciar al finalizar la mezcla y la unificación de la variable continuar.

```
// Variables de control
int empezar = 0;
int parar = 0;
int vaciar = 0;
int continuar = 0;
```

Figura 20: Variables de control.

También se declaran variables para el tiempo de las tareas que miden la duración en horas, minutos, el total en milisegundos, el momento en que inicia la tarea, el tiempo transcurrido y el tiempo restante en horas, minutos y segundos que falta para finalizar. Estas variables son del tipo *long*.

```
// Variables de tiempo de la tarea
long duracion_horas = 0;         // Duración en horas
long duracion_minutos = 0;       // Duración en minutos
long duracion_horas_ms = 0;
long duracion_min_ms = 0;
long duracion_total_ms = 0;       // Duración total en milisegundos
long tiempo_inicio = 0;           // Momento de inicio de la tarea
long tiempo_transcurrido = 0;     // Tiempo transcurrido
long tiempo_restante = 0;         // Tiempo restante
long horas_restantes = 0;         // Horas restantes
long minutos_restantes = 0;       // Minutos restantes
long segundos_restantes = 0;      // Segundos restantes
```

Figura 21: Variables de tiempo para tareas.

```
// Variables temporales del loop
long tiempoEnvio = 0;
long tiempoMonitoreo = 0;
long TInicioMezclado = 0;
long previousMillis = 0;
long TiempoMotorOn = 5000;
long TiempoMotorOff = 3000;
long currentMillis = 0;
long resto = 0;
long TiempoHor = 0;
long TiempoMin = 0;
long TiempoHorUso = 0;
long TiempoMinUso = TiempoMin * 60000;
byte MotorOn = 1;
byte MotorOff = 0;
```

Figura 22: Variables de tiempo para bucle principal.

Se definen variables con la intención de controlar la transmisión Bluetooth con la aplicación móvil, en esta instancia, tendremos la variable que nos brindará un dato, en este caso una letra, que determinará la acción a realizar, también la variable que dependiendo del valor que contenga almacenará dicho registro recibido en una variable tipo *String*.

```
// Variables de control de transmisión Bluetooth
int g = 0;
char valor = 'F';
String estado;
byte flagTransmicion = 1;
```

Figura 23: Variables transmisión Bluetooth.

Le siguen las variables pertenecientes a la recepción del Bluetooth, estas se inicializarán en 0 y almacenaran el número correspondiente para indicar la cantidad a reponer en los bombos y a cuál estará dirigido.

```
// Variables de recepción Bluetooth
int convinacion = 0;
int bomboSeleccionado = 0;
int valorMaxReposicion = 0;
byte activarMezcla = 0;
byte listo = 0;
byte ejecucion = 1;
char c = 0;
```

Figura 24: Variables recepción Bluetooth.

En la instancia para el estado de los motores se definen variables para controlar el parado de emergencia de la reposición, detectar si los niveles que se envían son para reposición y detecta si los niveles que se envían son para mezcla. Estas variables mayormente son de tipo *byte*, estas contienen un dato que ocupa 1 byte (8 bits) de memoria y está diseñado para almacenar números enteros sin signo.

```
// Variables estados de motores
byte flagParadaR = 0;
byte flagR = 0;
byte flagM = 0;
byte EMezclador = 0;
byte EBomba1 = 0;
byte EBomba2 = 0;
byte EBombaM = 0;
byte EBombaR = 0;
byte EValvula1 = 0;
byte EValvula2 = 0;
byte EProceso = 0;
byte EBomboM = 0;
long horaRest = 0;
long minRest = 0;
int error = 0;
byte desechar = 0;
byte arranque2 = 0;
byte detener = 0;
```

Figura 25: Variables estado de motores.

Para el control del nivel de los tanques se asignaron pines para el sensor ultrasónico y para su funcionamiento óptimo, por eso se añadió una variable que permite el suavizado y entre 0 y 1, reduce la oscilación del sensor.

```
// Variables control de nivel
int i = 0;
int x = 25;
int banderal = 0;
float alpha = 0.5;

// Pines de control de los sensores
int trig = 16;
int eco = 17;
```

Figura 26: Variables de control sensores ultrasónicos.

En la medición de las distancias se contemplan las variables de duración que almacena el tiempo que tarda la onda ultrasónica en viajar desde el sensor hasta el líquido en el bombo y regresar. La variable de distancia, que se repite por cada bombo, calcula y almacena la distancia resultante a partir del tiempo obtenido en la variable de duración. Las variables de tipo *byte* almacenan el porcentaje de nivel calculado para los diferentes depósitos y usan ese tipo de dato porque los porcentajes siempre estarán en el rango de 0 a 100, que cabe perfectamente en un solo byte. Además de las ya mencionadas variables, se presentan variables que representan las distancias filtradas, los porcentajes crudos se procesan para filtrar valores inestables y para suavizar posibles fluctuaciones antes de usarlos en el control del sistema.

```
// Variables de medición de distancia
float duracion;
float distancia;
float distancia1;
float distancia2;
float distancia3;
byte constrainedPorcentaje1 = 0;
byte constrainedPorcentaje2 = 0;
byte constrainedPorcentaje3 = 0;

// Mediciones de distancias filtradas
byte Fporcentaje1 = 0;
byte Fporcentaje2 = 0;
byte Fporcentaje3 = 0;
```

Figura 27: Variables para calcular distancias.

Las variables pertenecientes a la sección de estadística son de utilidad para los filtros aplicados en las mediciones de los diferentes sensores utilizados en el proyecto. Definimos una constante con el valor 15. Esto significa que se realizarán 15 lecturas consecutivas para calcular el promedio y al usar #define, el valor 15 será reemplazado directamente en el código por el preprocesador antes de la compilación. Los sensores ultrasónicos pueden generar valores fluctuantes debido a interferencias o ruido por lo que promediar múltiples lecturas ayuda a suavizar estos valores, proporcionando mediciones más estables y confiables. Las variables de lectura declaran un arreglo de tipo *float* con 15 posiciones para almacenar las lecturas de los sensores. Cada posición del arreglo guarda un valor individual de las últimas 15 mediciones. Estos arreglos almacenan las últimas mediciones de cada sensor. Posteriormente, se calcula el promedio de los valores almacenados para obtener una medida más precisa.

```
// Estadísticas
#define NUM_READINGS 15
float readings1[NUM_READINGS];
float readings2[NUM_READINGS];
float readings3[NUM_READINGS];
int readIndex = 0;
float total1 = 0;
float total2 = 0;
float total3 = 0;
float average1 = 0;
float average2 = 0;
float average3 = 0;
```

Figura 28: Variables para estadísticas.

Y para finalizar, las variables definidas que corresponden al control de los caudalímetros, las dos primeras son de tipo *volatile double* e indica que estas variables pueden ser modificadas en una interrupción, por lo que el compilador no optimiza su acceso y garantiza que siempre se lea el valor actualizado. Las variables almacenan el valor del flujo del líquido medido por el caudalímetro conectado a los dos canales. Las demás variables permiten almacenar la cantidad de líquido que se depositará y el nivel en el que se encuentran los bombos. También se controla la finalización o no del llenado en los depósitos.

```
// Variables de control de caudal
volatile double waterFlow1;
volatile double waterFlow2;
double Ingrediente1 = 0;
double Ingrediente2 = 0;
double nivel_liquido_1 = 0;
double nivel_liquido_2 = 0;
double liquido1 = 0;
double liquido2 = 0;
int bandera_c = 1;
byte terminoLlenadoLiquido1 = 0;
byte terminoLlenadoLiquido2 = 0;
```

Figura 29: Variables de control caudal.

Cabe aclarar que, tanto a estas variables como a otras, se les dará una explicación más detallada mediante vayan apareciendo a lo largo del programa.

En *void Setup()* se realiza la configuración inicial del programa, en esta instancia se preparan los pines, sensores e interrupciones que se necesitan para el funcionamiento del sistema.

Primero configuramos el puerto serie para mandar señales de control al monitor de la PC. Mediante la sentencia *Serial.begin*(9600) se inicia la comunicación con una velocidad de 9600 baudios para enviar y recibir datos. Y en cuanto a *Serial.print*("Listo"), envía un mensaje al monitor serial indicando que el sistema está listo.

```
void setup() {
  //configuracion puerto serie para mandar señales
  Serial.begin(9600);
  Serial.print("Listo");
```

Figura 30: Configuración puerto serie.

Continuamos con la configuración de los pines de los sensores ultrasónicos. Se define como INPUT a los pines que reciben las señales de rebote (eco) y como OUTPUT a los que emiten el pulso ultrasónico (trig).

```
//pines de sensores ultrasonicos, nivel
pinMode(17, INPUT);
pinMode(19, INPUT);
pinMode(21, INPUT);
pinMode(16, OUTPUT);
pinMode(18, OUTPUT);
pinMode(20, OUTPUT);
```

Figura 31: Configuración pines ultrasónico.

Definimos los pines como salidas para controlar las bombas (DC y AC), el motor de mezcla DC y las electroválvulas que regulan el flujo de líquido. Luego establecemos los estados iniciales de los dispositivos, mediante HIGH las bombas están apagadas y las válvulas cerradas.

```
//pines de bombas
//Bombas dc y motor de mezcla dc
pinMode(4, OUTPUT);
pinMode(5, OUTPUT);
pinMode(6, OUTPUT);
pinMode(7, OUTPUT);
//Bomba ac y electrovalvulas ac
pinMode(8, OUTPUT);
pinMode(9, OUTPUT);
pinMode(10, OUTPUT);
pinMode(13, OUTPUT);

//Bombas dc y motor de mezcla dc
digitalWrite(4, HIGH);
digitalWrite(5, HIGH);
digitalWrite(6, HIGH);
digitalWrite(7, HIGH);
//Bomba ac y electrovalvulas ac
digitalWrite(8, HIGH);
digitalWrite(9, HIGH);
digitalWrite(10, HIGH);
```

Figura 32: Configuración pines bombas y electroválvulas.

Configuramos interrupciones externas en los pines 2 y 3 para detectar los pulsos generados por los caudalímetros. En la función se incluye pulse1 y pulse2 que manejan los eventos para actualizar los datos de flujo en las variables *waterFlow1* y *waterFlow2*. Con RISING determinamos que disparará la interrupción cuando haya un flanco de subida en la señal, es decir, al inicio de un pulso.

```
//configuracion de caudal
waterFlow1 = 0;
waterFlow2 = 0;

attachInterrupt(digitalPinToInterrupt(2), pulse1, RISING);
attachInterrupt(digitalPinToInterrupt(3), pulse2, RISING);
```

Figura 33: Interrupciones de caudalímetros.

Con la sentencia *BT.begin*(9600) iniciamos la comunicación Bluetooth con una velocidad de 9600 baudios. Esto nos permite enviar y recibir datos desde el dispositivo móvil donde se encuentra la aplicación.

```
//configuracion bluethooot
BT.begin(9600);
```

Figura 34: Configuración puerto Bluetooth.

Utilizamos un bucle *for* en el que su función es encargarse de poner todas las posiciones de las cadenas que se usaran para hacer los cálculos estadísticos. Esto asegura que los cálculos sean correctos desde el inicio.

```
for (int K = 0; K < NUM_READINGS; K++) {
  readings1[K] = 0;
  readings2[K] = 0;
  readings3[K] = 0;
}
```

Figura 35: Bucle para cálculos estadísticos.

Y por último configuramos el timer 5 para que cada 250 ms lea los datos que vienen del celular mediante la función lectura, la configuración se debe de escribir en micro segundos. La función *attachInterrupt* se encarga de recibir variables tipo texto desde el módulo, a su vez también transforma esos datos en variables numéricas aptas para su trabajo.

```
Timer5.initialize(250000);

Timer5.attachInterrupt(lectura);
```

Figura 36: Configuración del Timer5.

En el bucle principal, *void loop*(), se encuentra el código que permitirá la funcionalidad de programa y que se ejecuta de forma continua, en él están las funciones que realizan las acciones de, envió y recepción de datos, mediciones y cálculos mediante los sensores y dispositivos utilizados.

```
void loop() {
  //Funcion de monitoreo a travez del PC
  if ((tiempoMonitoreo + 2000) <= millis()) {
    monitoreo();
    tiempoMonitoreo = millis();
  }

  //control de nivel de depositos
  nivel();
  filtrado();

  //funciones relacionadas con Bluetooth
  if ((tiempoEnvio + 1000) <= millis()) {
    enviarValores();
    tiempoEnvio = millis();
  }


  //control de bomba a reposicoin de bombos
  llamadaRepo();

  //control de bombas a bombo de mezcla y mezclador
  llamadaProduccion();

  //control de caudal
  caudal();


}
```

Figura 37: Bucle principal.

Empezamos con una función que nos permitirá monitorear y controlar el proceso desde la PC. Recibiremos datos cada 2 segundos al monitor serial y se llama a la función *monitoreo()* que imprimirá los valores actuales de las variables *liquido1* y *liquido2*, que representan la cantidad de

líquido en los depósitos correspondientes, indicará la duración total del proceso configurada en horas, en minutos y en milisegundos, también se presenciará el tiempo restante y el estado de la variable empezar, que mostrará “1” si el proceso ha iniciado o “0” si está detenido. Se mostrarán los valores del tiempo que el proceso estuvo en pausa, el transcurrido desde el inicio y el faltante para finalizar. Al final se visualizará el estado en el que se encuentra la mezcla, si está en curso será “1” o si no “0”. Su finalidad será facilitar la identificación de problemas mostrando todos los valores clave en tiempo real.

```
//Funcion de monitoreo a travez del PC
if ((tiempoMonitoreo + 2000) <= millis()) {
  monitoreo();
  tiempoMonitoreo = millis();
}
```

Figura 38: Monitoreo por PC.

```
void monitoreo() {
  Serial.print("liquido 1:");
  Serial.print(liquido1);
  Serial.print("  liquido 2:");
  Serial.print(liquido2);
  Serial.print(" duracion hora:");
  Serial.print(duracion_horas);
  Serial.print(" duracion min:");
  Serial.println(duracion_minutos);
  Serial.print("duracion_total_ms:");
  Serial.print(duracion_total_ms);
  Serial.print("  horas_restantes:");
  Serial.print(horas_restantes);
  Serial.print("  minutos_restantes:");
  Serial.print(minutos_restantes);
  Serial.print("  empezar:");
  Serial.println(empezar);
  Serial.print("tiempo pausado:");
  Serial.print(tiempo_pausado);
  Serial.print("  tiempor rest (ms):");
  Serial.print(tiempo_restante);
  Serial.print("  tiempor trasnc (ms):");
  Serial.print(tiempo_transcurrido);
  Serial.print("  tiempor inicio (ms):");
  Serial.println(tiempo_inicio);
  Serial.print("duracion_total_ms:");
  Serial.print(duracion_total_ms);
  Serial.print("  mezcla:");
  Serial.println(mezcla_en_progreso);
}
```

Figura 39: Función Monitoreo.

En el control de los depósitos se hace un llamado a las funciones *nivel()* y *filtrado()*. Con la primera función nos encargamos de obtener la distancia del contenido con los sensores ultrasónicos. La función recopila datos de cada sensor secuencialmente y los almacena en sus variables específicas. Al inicio se emplea un bucle *while* con la condición de que tiene que ser menor que 3 para asegurarse de iterar esa cantidad de veces, una por cada sensor. Se establece que cada 5 microsegundos el sensor se activa enviando un pulso por el pin *trig*. Para medir el tiempo de rebote se emplea la función *pulseIn* que mide el tiempo, en microsegundos, durante el cual el pin eco permanece en alto. Este tiempo representa el viaje de ida y vuelta de la señal ultrasónica. Una

vez obtenida la duración, se calcula la distancia mediante una operación de división con la constante 58.2, derivada de las especificaciones del sensor, basada en la velocidad del sonido en aire a 20 °C. Luego, dependiendo el índice en el que se encuentre el bucle, se almacena la distancia en una variable específica de cada sensor, y para esto se incrementa en 2 los valores de *eco* y *trig* para apuntar al siguiente par de pines asignados a los otros sensores, también se le suma 1 al índice para pasar a otra iteración. Por último, se restablecen los valores de los pines y vuelven a sus iniciales, los pines del primer sensor, y el índice se reinicia para prepararse para la siguiente ejecución de la función.

```
void nivel() {
  while (i <= 3) {
    digitalWrite(trig, HIGH);
    delayMicroseconds(5);
    digitalWrite(trig, LOW);
    duracion = pulseIn(eco, HIGH);
    distancia = duracion / 58.2;
    if (i == 0) {
      distancia2 = distancia;
    }

    if (i == 1) {
      distancia3 = distancia;
    }

    if (i == 2) {
      distancia1 = distancia;
    }

    trig = trig + 2;
    eco = eco + 2;
    i = i + 1;
  }

  i = 0;
  trig = 16;
  eco = 17;
}
```

Figura 40: Función Nivel.

La función *filtrado()* se utiliza con el fin de reducir el error de las mediciones de los sensores, como usamos sensores ultrasónicos para medir el nivel de llenado de los bombos el ruido al usar ultrasonido puede generar errores en la medición, puede ser que una misma onda llegue dos veces y por eso esta función toma 10 mediciones de cada sensor y genera una salida promedio con el fin de suavizar los errores y el ruido.

```
void filtrado() {
  // Restar la lectura más antigua de la suma total
  total1 = total1 - readings1[readIndex];
  total2 = total2 - readings2[readIndex];
  total3 = total3 - readings3[readIndex];

  // guardar los valores de distancia
  readings1[readIndex] = distancia1;
  readings2[readIndex] = distancia2;
  readings3[readIndex] = distancia3;

  // Añadir la nueva lectura a la suma total
  total1 = total1 + readings1[readIndex];
  total2 = total2 + readings2[readIndex];
  total3 = total3 + readings3[readIndex];

  // Avanzar al próximo índice
  readIndex = readIndex + 1;

  // Si llegamos al final del arreglo, volver al inicio
  if (readIndex >= NUM_READINGS) {
    readIndex = 0;
  }

  // Calcular el promedio
  average1 = total1 / NUM_READINGS;
  average2 = total2 / NUM_READINGS;
  average3 = total3 / NUM_READINGS;

  //obtiene el porcentaje de llenado de cada uno de los bombos
  Fporcentaje1 = map(average1, 27, 6, 0, 100);
  Fporcentaje2 = map(average2, 27, 6, 0, 100);
  Fporcentaje3 = map(average3, 27, 6, 0, 100);

  constrainedPorcentaje1 = constrain(Fporcentaje1, 0, 100);
  constrainedPorcentaje2 = constrain(Fporcentaje2, 0, 100);
  constrainedPorcentaje3 = constrain(Fporcentaje3, 0, 100);

}
```

Figura 41: Función Filtrado.

Al inicio se restará la lectura más antigua de la suma total eliminando ese valor antes de agregar el nuevo. Después se almacenan las nuevas distancias en las variables y se actualiza la suma total añadiendo las nuevas lecturas (NUM_READINGS).

Se incrementa el índice para apuntar al siguiente elemento en el arreglo y si el índice supera el tamaño del arreglo (NUM_READINGS), se reinicia a 0, haciendo que el arreglo actúe como un buffer circular.

Se calcula el promedio dividiendo la suma total entre el número de lecturas (NUM_READINGS). Este promedio representa una medida filtrada, eliminando el ruido y las mediciones espurias. Siguiente, con la función *map* se deben ajustar las características físicas del sistema, convirtiendo el rango de distancias medidas, de 27 cm a 6 cm, en un rango de 0% a 100%. 27 representa el nivel vacío, y 6 el nivel lleno de un recipiente. Utilizando *constrain* limitamos los valores calculados para asegurarse de que no sean menores a 0 ni mayores a 100. Esto evita errores en las mediciones que puedan dar porcentajes fuera del rango esperado.

Para realizar la comunicación Bluetooth y sus acciones relacionadas con la aplicación móvil. Se establece un condicional que al transcurrir un segundo desde la última tarea ejecutada se accede a la función *enviarValores()*. Esta se encarga de enviar datos mediante el módulo Bluetooth a la

aplicación del dispositivo externo. Luego de completar la emisión de la información se actualiza la variable *tiempoEnvio* con el tiempo actual. Esto reinicia el contador para que el intervalo de un segundo vuelva a comenzar.

```
//funciones relacionadas con Bluetooth
if ((tiempoEnvio + 1000) <= millis()) {
  enviarValores();
  tiempoEnvio = millis();
}
```

Figura 42: Funciones para envió por Bluetooth.

La función *enviarValores()* al cumplir con su condicional envía los valores leídos a la aplicación a través de Bluetooth. Se ejecuta cada vez que se recibe el carácter que indica una nueva conexión, en este caso definimos 'E'. La transmisión utiliza el comando *BT.print()* donde se añade la variable con el dato que queremos imprimir a través de los puertos Rx y Tx de conexión establecidos anteriormente. Luego para se repite la sentencia, pero esta vez utilizará “|” para delimitar cada valor y asignarle a cada uno una posición en una lista que podrá ser interpretada como número, de realiza de esta forma ya que la aplicación Android lo recibirá todo como texto y eso nos ayudará para mostrar o visualizar todos los datos que queramos. Estas interacciones con la aplicación y su comportamiento se detallarán cuando se explique su conformación y programación en la posterioridad.

```
void enviarValores() {
  if (flagTransmicion == 1) {
    BT.print(average1);
    BT.print("|");
    BT.print(constrainedPorcentaje1);
    BT.print("|");
    BT.print(average2);
    BT.print("|");
    BT.print(constrainedPorcentaje2);
    BT.print("|");
    BT.print(average3);
    BT.print("|");
    BT.print(constrainedPorcentaje3);
    BT.print("|");
    BT.print(nivel_liquido_1);
    BT.print("|");
    BT.print(nivel_liquido_2);
    BT.print("|");
    BT.print(EBomba1);
    BT.print("|");
    BT.print(EBomba2);
    BT.print("|");
    BT.print(EBombaM);
    BT.print("|");
    BT.print(EMezclador);
    BT.print("|");
    BT.print(EBombaR);
    BT.print("|");
    BT.print(error);
    BT.print("|");
    BT.print(horas_restantes);
    BT.print("|");
    BT.print(minutos_restantes);
    BT.print("|");
    BT.print(EProceso);
    BT.print("|");
    BT.print(EBomboM);
    BT.print("|");
    BT.print(segundos_restantes);
    BT.print("\n"); // Fin de linea. Importante.
  }
}
```

Figura 43: Función EnviarValores().

Continuamos con el llamado a la función *llamadaRepo()* que se encarga de la comunicación para controlar el llenado de los bombos y la mezcla dependiendo de la combinación obtenida.

Con el uso de dos condiciones definimos a cuáles de los bombos queremos reponer y la cantidad de líquido a depositar. Si la variable combinación está entre 1000 y 1100, el bombo 1 es seleccionado y la cantidad a reponer es la diferencia con 1000. En caso de que sea entre 2000 y 2200, se selecciona el bombo 2 y la cantidad se obtiene restándole 2000.

```
void llamadaRepo() {
  //se decodifica que bombo y cuanto se debe de reponer
  if (combinacion >= 1000 and combinacion <= 1100) {
    bomboSeleccionado = 1;
    valorMaxReposicion = combinacion - 1000;
  }

  if (combinacion >= 2000 and combinacion <= 2200) {
    bomboSeleccionado = 2;
    valorMaxReposicion = combinacion - 2000;
  }
```

Figura 44: Función llamadaRepo.

Dentro de un condicional se empleará la variable *flagParadaR* que, al estar en 0, el sistema estará activo y podrá realizar la reposición. En el caso de que se presione parar por algún motivo u ocurra un error, se pondrá en 1 el valor de la variable perteneciente al condicional y la reposición se detiene, se desactivarán las siguientes ejecuciones y se deberán cargar los valores nuevamente si se quiere reanudar.

```
if (flagParadaR == 0) {
```

Figura 45: Condicional para reposición.

Cuando se accede al bombo 1 para poder depositar el líquido en él, se presenta un condicional con la función de verificar errores, en caso de que el nivel actual del contenido ya supera o iguala la cantidad deseada y la bomba no está activa, se genera un error con el código 722. Este código es enviado a la aplicación y al detectar este valor, dará una alerta indicando lo sucedido.

```
if (bomboSeleccionado == 1) {
  if (valorMaxReposicion <= Fporcentajel and EBombaR == 0) {
    error = 722;
  }
```

Figura 46: Condicional para acceder al bombo 1.

Posteriormente se establece otra condición para iniciar la reposición. Si el nivel es menor que el objetivo y el bombo no está lleno, se activa la bomba y la electroválvula para llevar a cabo el proceso de llenado.

```
if (valorMaxReposicion > Fporcentajel and Fporcentajel < 100) {
  EBombaR = 1;
  //encendido electrovalvula
  digitalWrite(10, LOW);
  //encendido bomba repo
  digitalWrite(9, LOW);
}
```

Figura 47: Condicional para iniciar reposición.

Y para finalizar con la reposición al alcanzar el nivel deseado, la bomba y la electroválvula se apagan, y el sistema se prepara para una nueva operación.

```
if (valorMaxReposicion <= Fporcentaje1) {
  EBombaR = 0;
  flagParadaR = 1;
  valorMaxReposicion = 0;
  //apagado bomba repo
  digitalWrite(9, HIGH);
  bomboSeleccionado = 0;
  //apagado electrovalvula
  digitalWrite(10, HIGH);
}
```

Figura 48: Condicional para finalizar reposición.

Al acceder al bombo 2 el procedimiento es el mismo pero las acciones afectan a sus variables específicas y usan los pines correspondientes, electroválvula en pin 8 y bomba en pin 9.

```
if (bomboSeleccionado == 2) {
  if (valorMaxReposicion <= Fporcentaje2 and EBombaR == 0) {
    error = 722;
  }

  if (valorMaxReposicion > Fporcentaje2 and Fporcentaje2 < 100) {
    EBombaR = 1;
    //encendido bomba repo
    digitalWrite(9, LOW);
    //encendido electrovalvula
    digitalWrite(8, LOW);
  }

  if (valorMaxReposicion <= Fporcentaje2 or Fporcentaje2 == 100) {
    EBombaR = 0;
    flagParadaR = 1;
    valorMaxReposicion = 0;
    //apagado bomba repo
    digitalWrite(9, HIGH);
    //apagado electrovalvula
    digitalWrite(8, HIGH);
    bomboSeleccionado = 0;
  }
 }
}
```

Figura 49: Condicional para acceder al bombo 2, inicializar y finalizar reposición.

A continuación en el bucle principal, se contempla la función *llamadaProduccion()*, esta es responsable de gestionar la producción de una mezcla en función de los ingredientes especificados, la duración del proceso y las acciones requeridas, como pausar, continuar o desechar el proceso. En primera instancia, convierte las horas y minutos especificados a milisegundos para controlar el tiempo de mezcla. Luego calcula la cantidad de líquido que hay en ambos bombos.

```
void llamadaProduccion() {
  duracion_total_ms = ((duracion_horas * 60) + duracion_minutos) * 60 * 1000;

  liquido1 = (Ingrediente1 - 10000) / 1000;
  liquido2 = (Ingrediente2 - 20000) / 1000;
```

Figura 50: Función llamadaProduccion.

Dada la orden de desechar el contenido en el bombo de mezcla hace que este se vacíe y reinicia todas las variables relacionadas con la producción ya que se descarta el proceso que se estaba realizando. En el caso de que el nivel del depósito de mezcla es menor al 10%, se detiene el vaciado automáticamente.

```
if (desechar == 1) {
  digitalWrite(4, LOW);
  EBombaM = 1;
  liquido1 = 0;
  nivel_liquido_1 = 0;
  liquido2 = 0;
  nivel_liquido_2 = 0;
  Ingrediente1 = 0;
  Ingrediente2 = 0;
  EProceso = 4;
  mezcla_en_progreso = false;
  duracion_total_ms = 0;
  duracion_horas = 0;
  duracion_minutos = 0;
}

  if (constrainedPorcentaje3 < 10) {
    digitalWrite(4, HIGH);
    vaciar = 0;
    EBombaM = 0;
    desechar = 0;
    EProceso = 0;
  }
```

Figura 51: Condicionales para desechar contenido.

Después se especifica el llenado del depósito de mezcla. Inicialmente se activa la bomba de llenado para el primer líquido si el nivel actual es menor al requerido, y este se detiene cuando se alcanza el nivel deseado. Luego se realiza el llenado con el segundo líquido una vez que el primero ha terminado.

```
// Control del llenado de líquidos
if (empezar == 1) {
  // Llenar con líquido 1
  if (nivel_liquido_1 <= liquido1 && bomba_1_encendida == false) {
    digitalWrite(5, LOW);
    bomba_1_encendida = true;
    Serial.println("Bomba 1 encendida: llenando líquido 1");
  } else if (nivel_liquido_1 >= liquido1 && bomba_1_encendida == true) {
    digitalWrite(5, HIGH);
    bomba_1_encendida = false;
    Serial.println("Llenado de líquido 1 completado.");
    terminoLlenadoLiquido1 = 1;
  }

  // Llenar con líquido 2
  if (nivel_liquido_2 <= liquido2 && bomba_2_encendida == false && bomba_1_encendida == false) {
    digitalWrite(6, LOW);
    bomba_2_encendida = true;
    Serial.println("Bomba 2 encendida: llenando líquido 2");
  } else if (nivel_liquido_2 >= liquido2 && bomba_2_encendida == true) {
    digitalWrite(6, HIGH);
    bomba_2_encendida = false;
    Serial.println("Llenado de líquido 2 completado.");
  }
```

Figura 52: Control de llenado de bombo 1 y bombo 2.

Cuando ambas bombas han terminado de llenar los líquidos, se inicia la mezcla, registrando el tiempo de inicio.

```
  // Iniciar mezcla si ambas bombas han terminado
  if (bomba_1_encendida == false && bomba_2_encendida == false && mezcla_en_progreso == false) {
    mezcla_en_progreso = true;
    tiempo_inicio = millis();
    tiempo_pausado = 0;
    Serial.println("Iniciando mezcla.");
    terminoLlenadoLiquido1 = 0;
    Ingrediente1 = 0;
    Ingrediente2 = 0 ;
  }
}
```

Figura 53: Inicialización de mezclado.

Este proceso se puede alterar de forma manual. El usuario de la aplicación puede indicar que la producción se detenga mediante una pausa y reanudarla cuando él disponga. Si se cumple con la condición de pausa, el sistema detiene las bombas y el mezclador, registrando el tiempo pausado. Y se reanuda el proceso donde había quedado una vez que se recibe la acción de continuar.

```
//Si se solicita pausar
if (parar == 1) {
  //para las bombas
  mezcla_en_progreso = false;
  digitalWrite(4, HIGH);
  digitalWrite(5, HIGH);
  digitalWrite(6, HIGH);
  tiempo_pausado += millis() - tiempo_inicio;
  Serial.println("Proceso pausado");

  // Reiniciar variable de control
  parar = 0;
  desechar = 0;
  vaciar = 0;
  EProceso = 3;
}

//Si se solicita continuar
if (continuar == 1 && mezcla_en_progreso == false) {
  mezcla_en_progreso = true;
  tiempo_inicio = millis();
  Serial.println("Proceso reanudado");
  continuar = 0;
}
```

Figura 54: Solicitud de pausar y continuar.

El proceso de mezclado cumple con un ciclo de tiempo en el que el motor se mantiene encendido por 5 segundo y luego se apaga por los próximos 2 segundos. Esta acción se repite hasta que el tiempo total de mezclado se cumpla.

```
// Proceso de mezcla
if (mezcla_en_progreso == true) {
  empezar = 0;

  tiempo_transcurrido = millis() - tiempo_inicio + tiempo_pausado;
  tiempo_restante = duracion_total_ms - tiempo_transcurrido;

  horas_restantes = tiempo_restante / 3600000;
  minutos_restantes = (tiempo_restante / 60000) - (horas_restantes * 60);
  segundos_restantes = (tiempo_restante / 1000) - (horas_restantes * 3600) - (minutos_restantes * 60);

  // Mostrar tiempo restante por Serial
  Serial.print("Tiempo restante: ");
  Serial.print(horas_restantes);
  Serial.print("h ");
  Serial.print(minutos_restantes);
  Serial.println("m");
  Serial.print(segundos_restantes);
  Serial.println("s");

  // Ciclo del motor (5 segundos encendido, 2 segundos apagado)
  if (tiempo_transcurrido % 7000 < 5000) {
    digitalWrite(13, LOW);
    Serial.println("Motor encendido");
    EMezclador = 1;
    EProceso = 1;
  } else {
    digitalWrite(13, HIGH);
    Serial.println("Motor apagado");
    EMezclador = 0;
    EProceso = 1;
  }

  // Finalizar mezcla cuando el tiempo se acabe
  if (tiempo_restante <= 0) {
    mezcla_en_progreso = false;
    Serial.println("Mezcla finalizada. Esperando vaciado.");
    EProceso = 2;
  }
}
```

Figura 55: Proceso de mezclado.

Una vez terminado el proceso de mezclado se activa una señal de vaciado. Culminando así con la producción de unificar los dos líquidos provenientes de los bombos 1 y 2.

```
// Activar señal de vaciado si el tiempo de mezcla ha terminado
if (mezcla_en_progreso == false && vaciar == 1) {
  digitalWrite(4, LOW);
  Serial.println("Pin de vaciado activado.");
  vaciar = 0; // Reiniciar variable de control
  EProceso = 4;
}
```

Figura 56: Tiempo de mezcla para finalizar.

Como última función presente en el bucle principal, *caudal()*, se encargan de medir el flujo de agua mediante los caudalímetros y calcular el nivel de líquido agregado al sistema.

```
void caudal() {
  nivel_liquido_1 = waterFlow1;
  nivel_liquido_2 = waterFlow2;
}
```

Figura 57: Función caudal.

Anteriormente se definen dos funciones que solo se ejecutan por interrupciones externas, definidas previamente, de forma independiente generadas por los caudalímetros cuando detectan un flujo de agua. Cada vez que ocurre una interrupción, se actualiza el flujo acumulado para el líquido y se almacena su valor en variables de carácter global que luego se usarán para controlar los procesos del sistema. La primera función mide la cantidad de flujo del caudalímetro 1 y cuenta con la condición de contabilizar el flujo en caso de que el llenado del primer líquido este en progreso. Si se cumple se ejecuta la sentencia *waterFlow1* += 1.0 / 450 que corresponde a la cantidad de flujo medida en litros por cada pulso generado por el caudalímetro de 1/2 pulgada. Nosotros contamos con 1/4 pulgada, lo que le correspondería un valor de 5880 pulsos por litro, este cálculo se debería ajustar si fuera necesario.

```
void pulse1()
  if (terminoLlenadoLiquido1 == 0) {
    waterFlow1 += 1.0 / 450;
  }
}
```

Figura 58: Función pulse1.

```
void pulse2()
{
  if (terminoLlenadoLiquido1 == 1) {
    waterFlow2 += 1.0 / 450;
  }
}
```

Figura 59: Función pulse2.

La segunda función mide el flujo del segundo caudalímetro y es similar a la anterior pero solo se ejecuta cuando el primer líquido ya se ha llenado y el sistema está trabajando en el llenado del segundo líquido.

Para finalizar, nos quedaron por explicar funciones importantes que son implementadas para la comunicación Bluetooth con la aplicación móvil. La función *lectura()* se utiliza para leer que es lo que llega desde la aplicación del celular, decodifica los datos entrantes desde el módulo Bluetooth y según el carácter recibido, realiza acciones específicas o almacena valores para su uso posterior. Primero comprueba que haya datos disponibles en el puerto serie Bluetooth mediante el comando *BT.available()*, en caso de que la condición se cumpla, lee el valor recibido.

```
void lectura() {

  if (BT.available()) {
    valor = BT.read();
```

Figura 60: Función lectura.

Dependiendo del carácter que obtuvimos, se realizará una acción en específico. Para el control de reposición y mezclado se utilizan:

- 'F': Detiene la reposición.
- 'R': Indica cantidad máxima para reposición.

- 'G': Selecciona el bombo de reposición.
- 'S': Señala que se han configurado valores para la reposición.
- 'C' / 'c': Indica el valor de ingredientes para el mezclado y comienza el proceso, ‘c’ también establece la variable empezar en 1.
- 'D': Detiene el proceso de mezclado.
- 'A': Reanuda el proceso de mezclado.
- 'V': Activa el vaciado del bombo de mezcla.
- ‘H’: Indica las horas de duración del mezclado.
- ‘h’: Indica los minutos de duración del mezclado.
- ‘X’: Indica la orden de desechar el contenido del bombo de mezcla.

```
if (valor == 'F') {      //Si el dato entrante es una F, se para la reposicion
  g = 1;
  frenadoReposicion();
  flagParadaR = 1;
}

if (valor == 'R') {      //Si el dato entrante es una R, indica cantidad/max de reposicion
  g = 2;
  obtencionEntero();     //Llama la función que controla el valor a guradar el valor
  flagR = 1;
  flagM = 0;
  flagParadaR = 0;
}

if (valor == 'G') {      //Si el dato entrante es una G, indica Bombo Reposicion
  g = 3;
  obtencionEntero();     //Llama la función que controla el valor a guradar el valor
}

if (valor == 'S') {      //Si el dato entrante es una S, indica que se realizo el seteo de los valores de reposicion en el celular
  g = 4;
  obtencionEntero();     //Llama la función que controla el valor a guradar el valor
}

if (valor == 'C') {      //Si el dato entrante es una C indica el valor de Ingrediente y comenzar
  g = 5;
  obtencionEntero();     //Llama la función que controla el valor a guradar el valor
}

if (valor == 'c') {      //Si el dato entrante es una c indica el valor de Ingrediente2 y comenzar
  g = 6;
  obtencionEntero();     //Llama la función que controla el valor a guradar el valor
  empezar = 1;
}

if (valor == 'V') {      //Si el dato entrante es una V, llega una señal de vaciar. Activa la bomba del Bombo de Mezcla
  g = 7;
  vaciar = 1;
}
```

Figura 61: Acciones por caracter.

```
if (valor == 'D') {      //Si el dato entrante es una D se detiene el proceso de mezclado
  g = 8;
  parar = 1;
}

if (valor == 'A') {      //Si el dato entrante es una A se continua el mezclado
  g = 10;
  continuar = 1;
}

if (valor == 'T') {      //Si el dato entrante es una T guarda un falga de estado
  flagTransmicion = 1;
}

if (valor == 'H') {      //Si el dato entrante es una H indica que le esta llegando un valor a referencia de Horas
  g = 11;
  obtencionEntero();     //Llama la función que controla el valor a guradar
}

if (valor == 'h') {      //Si el dato entrante es una H indica que le esta llegando un valor a referencia de minutos
  g = 12;
  obtencionEntero();     //Llama la función que controla el valor a guradar
}

if (valor == 'X') {      //Si el dato entrante es una X indica que le esta llegando una orden de desechar
  desechar = 1;          //la variable encargada de controlar el desecho del contenido del bombo de mezcla
}
```

Figura 62: Acciones por carácter.

En la ejecución de la acción en base al carácter ‘F’, se llama a la función *frenadoReposicion()* que tiene como objetivo detener el proceso de reposición en el sistema, desactivando las bombas y las electroválvulas relacionadas con los bombos de reposición. Es una función sencilla pero crítica para garantizar un control seguro y ordenado del sistema en situaciones donde se requiere detener el proceso de llenado.

```
void frenadoReposicion() {
  flagParadaR = 1;
  EBombaR = 0;
  //apagado bomba reposicion
  digitalWrite(9, HIGH);
  bomboSeleccionado = 0;
  //apagado electrovalvula
  digitalWrite(10, HIGH);
  //apagado electrovalvula
  digitalWrite(8, HIGH);
}
```

Figura 63: Función frenadoReposicion.

Por último, la función a la cual la mayoría de los caracteres llaman en la función anterior, es *obtencionEntero()*. Se encarga de recibir y convertir los datos enviados desde Bluetooth, posteriores al carácter principal, en valores enteros. Estos valores se almacenan en las variables correspondientes dependiendo del contexto “g”. Captura los datos de entrada mediante un bucle *while* y los guarda en una variable tipo *char* para crear un *string*.

```
void obtencionEntero() {
  delay(30);
  while (BT.available()) {
    char c = BT.read();
    estado += c;
  }
```

Figura 64: Función obtencionEntero.

Si la cadena tiene datos, se convierte a entero con *toInt()* y se almacena en la variable correspondiente dependiendo del valor de “g”.

```
if (estado.length() > 0) {          //Se verifica que la cadena tipo String tenga un largo mayor a cero

  if (g == 1) {                     //dependiendo del valor de g
                                    //Guarda en un registro el dato en forma de entero (int)
  }

  if (g == 2) {
    convinacion = estado.toInt();   //guarda el valor que se usara para saber hasta donde llenar los bombo 1 y 2
  }

  if (g == 3) {
    bomboSeleccionado = estado.toInt(); //guarda que bombo se tiene que reponer
  }

  if (g == 5) {
    Ingrediente1 = estado.toInt();      //cantidad de ingrediente1 que se va a utilizar en la mezcla
  }

  if (g == 6) {
    Ingrediente2 = estado.toInt();      //cantidad de ingrediente2 que se va a utilizar en la mezcla
  }

  if (g == 11) {
    duracion_horas = estado.toInt();    //guarda cuantas horas debe de durar el proceso de mezclado
  }

  if (g == 12) {
    duracion_minutos = estado.toInt();  //guarda cuantos minutos debe de durar el proceso de mezclado
  }
```

Figura 65: Conversión a valor entero.

Una vez finalizado vuelve la variable “g” a 0 y limpia la variable para poder leer posteriormente nuevos datos.

```
g = 0;
estado = "";
```

Figura 66: Variables limpias.

Aplicación del celular:

Introducción:

Según los requisitos de del proyecto el sistema debe de tener una funcionalidad inalámbrica, dentro de las opciones que barajamos estaban los módulos bluetooth o los módulos wifi.

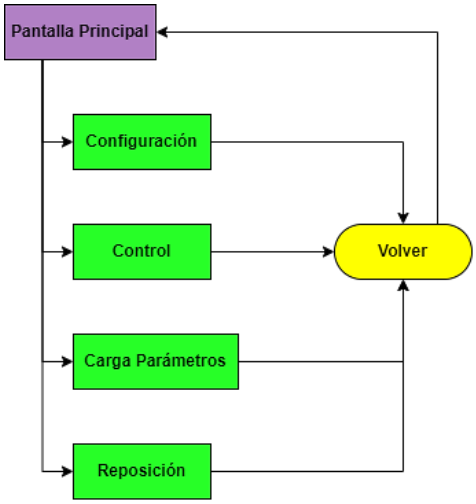
Nosotros elegimos el modulo bluetooth para establecer comunicación con dispositivos móviles donde estos podrán controlar y monitorear el funcionamiento del sistema.

Creamos una aplicación usando “MIT APP INVENTOR”, que es un entorno de programación web de acceso gratuito y que permite crear aplicaciones totalmente funcionales para teléfonos Android, iPhones y tabletas Android/iOS basándose en la programación en bloques lo que lo hace mucho más simple que métodos tradicionales de programación.

Esquema de navegación de la aplicación:

Nuestra aplicación tiene diferentes pantallas con el fin de controlar varios aspectos del sistema de control y monitorear los valores de los sensores.

Como se observa en el diagrama de flujo, todas las pantallas dependen de una pantalla principal, la cual da acceso a las demás. Ninguna de estas pantallas tiene subpantallas dependientes, por lo que todas incluyen un botón "Volver" que permite al usuario regresar a la pantalla principal.



Detalles de cada pantalla:

Pantalla Principal:

- **Objetivo o función:** pantalla principal donde se ven las diferentes secciones del sistema.
- **Componentes clave:**
  Tiene cuatro botones, uno por cada pantalla secundaria a la que se puede acceder, estos son “Configuración”, “Control”, “Carga Parámetros” y “Reposición”. En adición hay una etiqueta donde se indica el estado del conexionado bluetooth, esta muestra “Estado: (estado conexión)” y se mantiene visible en todas las secciones.

Figura 67: Pantalla de inicio.

- **Flujo de interacción:**
  El usuario cuando inicia la aplicación empieza desde esta pantalla, en esta puede visualizar el estado de la conexión Bluetooth y las diferentes secciones que tiene la app, dependiendo del botón del que se presione el usuario será desplazado a una u otra página.

Configuración:

Figura 68: Pantalla de Configuración

- **Objetivo o función:** seleccionar el dispositivo bluetooth al que se quiere conectar y mostrar la información sobre la conexión.
- **Componentes clave:**
  Cuenta con 3 etiquetas que muestran información de la conexión junto a tres botones que permiten realizar acciones sobre la conexión y la página. El botón “Seleccionar BT” despliega una lista con los dispositivos bluetooth disponibles mientras que el botón “Desconectar” desvincula el dispositivo móvil del módulo bluetooth conectado. Finalmente, el botón “Volver” al ser presionado cierra la ventana y vuelve a la “Pantalla Principal”.
- **Flujo de interacción:**
  El usuario presiona el botón de “Configuración” lo que lo dirigirá desde la “Página Principal” a la de “Configuración” donde una vez en esta sección puede pulsar el botón “Seleccionar BT” donde se desplegará una lista con los dispositivos disponibles detectados, el usuario selecciona el que le sea conveniente. Una vez realizado esto se le mostrará la información del dispositivo en las etiquetas que aparecen y a su vez la etiqueta superior que se mantiene en todas las ventanas se actualizará a “Conectado” si realiza exitosamente la conexión. Para retomar a las ventanas anteriores el usuario debe de presionar el botón de “Volver” mientras tantos si se busaca desvincular los dispositivos el usuario puede utilizar el botón de “Desconectar”.

Control:

- **Objetivo o función:** esta sección mostrará información sobre los sensores y además se puede controlar el proceso de mezcla.
- **Componentes clave:**

  La pantalla está dividida en cuatro partes principales: “Nivel de Bombos”, “Bombo de Mezcla”, “Tiempos y Estados” y una sección de botones sin un nombre específico, aunque cada uno de ellos cumple una función diferente.

  En el apartado “Nivel de Bombos” se presenta información específica sobre los bombos utilizados para el almacenaje. Aquí se muestra la distancia medida desde la parte superior del bombo y el porcentaje de llenado correspondiente.

  En el apartado “Bombo de Mezcla”, además de los datos sobre la distancia desde la parte superior del bombo y el porcentaje de llenado, se detalla la cantidad de líquido de cada tipo introducido en el bombo, lo cual es posible gracias al uso de los caudalímetros.

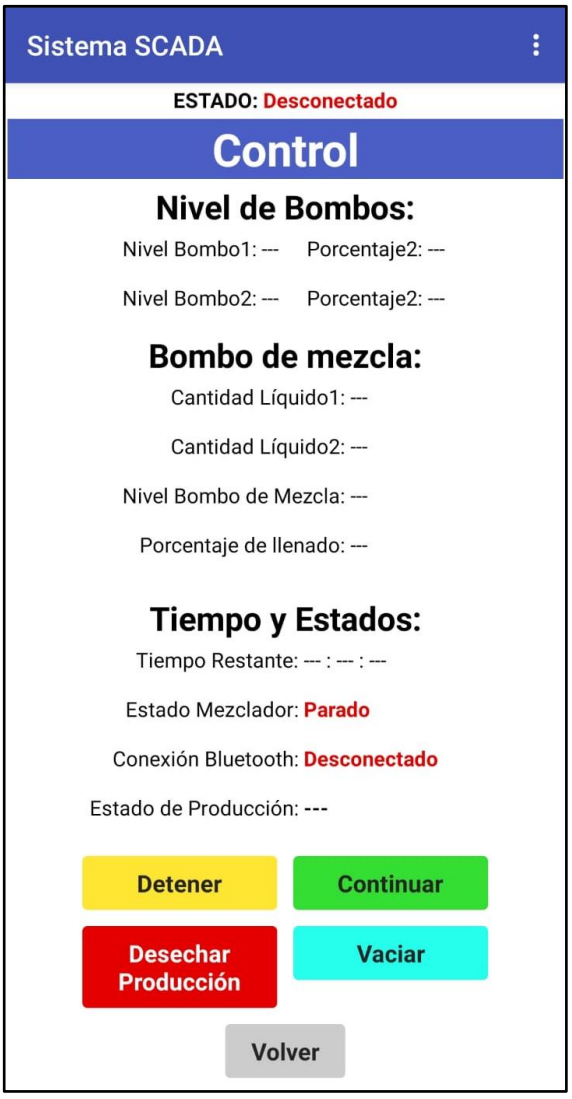


Figura 69: Pantalla de Control.

La sección “Tiempos y Estados” muestra información clave como el tiempo restante para que termine el proceso, el estado del mezclador, el estado de la conexión y el estado de la producción. Estos datos permiten al usuario identificar en qué etapa del proceso se encuentra la producción y tomar las medidas necesarias.

Finalmente, la sección de botones incluye cuatro controles que gestionan el funcionamiento del bombo de mezcla, junto con el botón “Volver”, que permite al usuario regresar a la Página Principal.

- **Flujo de interacción:**

  El usuario puede presionar el botón “Control” desde la Página Principal, lo que lo dirigirá a la sección de Control, donde podrá visualizar el estado general del sistema. En esta sección, además de monitorear los datos del sistema, el usuario puede intervenir en el proceso de mezclado:

- Si desea detener el proceso, puede utilizar el botón “Parar”.
- Para reanudar el proceso desde el punto en que se detuvo, se utiliza el botón “Continuar”.
- En caso de que la mezcla deba desecharse debido a un producto defectuoso, puede presionar el botón “Desechar Producción”. Al hacerlo, se desplegará una notificación de advertencia que informa al usuario sobre las posibles consecuencias de esta acción. Si el usuario presiona “Cancelar”, no se realizará ninguna acción; en cambio, si decide continuar, deberá confirmar presionando “Continuar”, lo que activará el proceso de descarte.

Por otro lado, si no se interviene en el proceso y este finaliza correctamente, el usuario podrá elegir cuándo vaciar el producto presionando el botón “Vaciar”. Este botón solo estará habilitado cuando el proceso haya concluido completamente, y al activarlo, se iniciará el vaciado del bombo de mezcla hasta dejarlo vacío.

Carga Parámetros:

- **Objetivo o función:** pantalla donde se configuran y envían los parámetros de la mezcla a realizar.
- **Componentes clave:**
  Tiene cuatro boxes de texto donde se introducen la cantidad de mililitros de cada litro que se debe de cargar y las horas y minutos que debe de durar el proceso de mezclado.
  En adición hay tres botones, estos son “Setear”, “Comenzar” y “Volver”. “Setear” guarda los valores dentro de la aplicación y envía el tiempo al Arduino mientras que “Comenzar” envía la cantidad de los líquidos y la señal de comienzo del proceso. Finalmente “Volver” se encarga de cerrar esta pantalla y cargar la “Página Principal”.

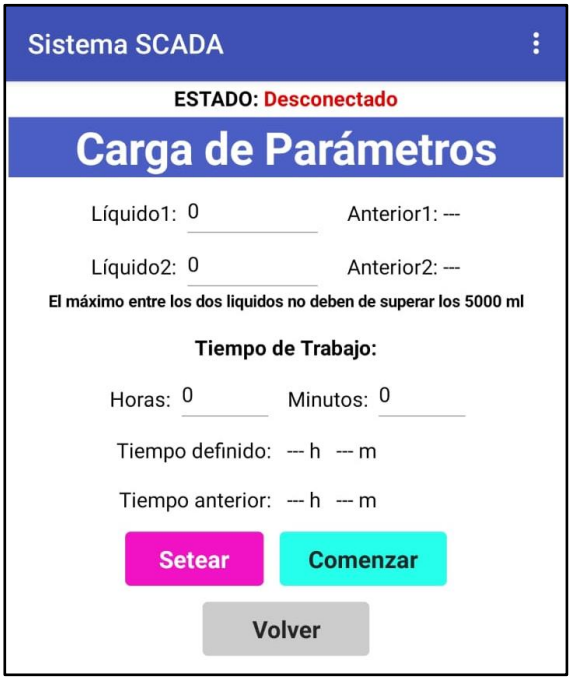


Figura 70: Pantalla de Carga de Parámetros.

- **Flujo de interacción:**
  El usuario debe de presionar el botón de “Carga Parámetros” en la “Página Principal”, una vez dentro de esta sección el usuario debe de introducir la cantidad de líquido que va a conformar la mezcla y el tiempo que debe de durar el proceso, una vez que los datos se introdujeron se debe de presionar el botón de “Setear” para fijar los parámetros y luego presionar el botón de “Comenzar” para que comience el proceso, una vez realizado estos

pasos si el usuario desea salir de esta sección puede presionar el botón “Volver” que lo guiara devuelta a “Página Principal”.

Reposición:

- **Objetivo o función:** esta sección muestra información sobre los bombos y estado de trabajo en el ámbito de la reposición de los líquidos a utilizar.
- **Componentes clave:**
  La redacción es comprensible, pero se pueden realizar algunos ajustes para mejorar la claridad, corregir errores ortográficos y optimizar la estructura. Aquí tienes una versión revisada:
  Las etiquetas proporcionan información sobre los niveles de los bombos, el estado de trabajo de la bomba de reposición y los bombos seleccionados, así como el porcentaje máximo establecido.

Figura 71: Pantalla de Reposición.

  La interfaz incluye un botón que despliega una lista de los bombos disponibles, donde el usuario debe seleccionar uno. Posteriormente, en el cuadro de texto, se introduce el porcentaje hasta el cual debe llenarse el bombo previamente seleccionado.
  El botón “Setear” guarda los datos en la aplicación, mientras que el botón “Iniciar” envía los datos a través de la conexión Bluetooth junto con la señal para comenzar la reposición. Si se necesita detener el proceso por algún inconveniente, se utiliza el botón “Parar”.
  Finalmente, el botón “Volver” permite al usuario regresar a la Página Principal.
- **Flujo de interacción:**
  El usuario presiona el botón “Reposición”, que lo lleva desde la “Página Principal” a la ventana de “Reposición”. En esta ventana puede visualizar los parámetros de los bombos y, para configurarlos, presiona el botón “Seleccionar Bombo”, que despliega una lista con los bombos disponibles. Ahí selecciona el bombo necesario e introduce en el cuadro de texto el porcentaje al cual debe llenarse.
  Luego, presiona el botón “Setear” para guardar los valores en la aplicación. Para enviarlos por Bluetooth y comenzar el proceso de reposición, debe presionar el botón “Continuar”. Si el usuario desea detener el proceso por algún motivo, puede presionar el botón “Parar”. De lo contrario, el proceso se detiene automáticamente al alcanzar el nivel deseado.

Cuando el usuario quiera volver a la “Página Principal” para cambiar a otra sección, debe presionar el botón “Volver”.

Funcionamiento combinado:

El usuario inicia la aplicación móvil y energiza el circuito del sistema. Una luz roja en la caja de señalización confirma que el sistema está alimentado y listo para operar. A continuación, el usuario vincula el dispositivo móvil al módulo Bluetooth del sistema mediante el apartado de configuración de la aplicación.

La comunicación entre la aplicación y el Arduino se realiza mediante el envío de comandos formados por caracteres y valores numéricos:

- Los caracteres definen las acciones a ejecutar.
- Los valores numéricos asociados establecen los parámetros requeridos para dichas acciones.
- Algunas funciones solo requieren caracteres sin valores adicionales.

Estos comandos se transmiten a través de Bluetooth hacia el módulo receptor, que los envía al Arduino mediante comunicación serial. Al recibir los datos, el Arduino los interpreta y ejecuta las acciones necesarias sobre los actuadores del sistema.

Simultáneamente, el Arduino supervisa constantemente las variables de control del sistema, como niveles y flujos. Estos datos se organizan en una trama de información que el Arduino envía de regreso a la aplicación móvil, siguiendo el flujo:

- El Arduino transmite los datos al módulo Bluetooth por comunicación serial.
- El módulo retransmite la información al dispositivo móvil.

La aplicación recibe la trama, procesa los datos mediante bloques de código, actualiza las variables globales y muestra los valores en tiempo real en las pantallas correspondientes.

La aplicación incorpora una verificación periódica de conexión. En intervalos definidos, envía un carácter de prueba al módulo Bluetooth. Si no se recibe respuesta, el estado de la aplicación cambia a "desconectado", se desvincula del módulo, y el usuario debe restablecer la conexión desde el menú de configuración.

El sistema desarrollado simula un entorno SCADA funcional, replicando los procesos básicos de supervisión y control sin incluir los dispositivos y protocolos específicos que comúnmente se emplean en aplicaciones industriales reales. Esto permite enfocarse en la lógica operativa y la interacción entre hardware y software, ofreciendo una aproximación simplificada pero efectiva para comprender el funcionamiento de sistemas de automatización industrial.

Problemáticas:

Problemas en el desarrollo del código y la aplicación móvil para el proyecto:

- Como bien se sabe un requerimiento del proyecto era que este debía de contar con la implementación de un método de comunicación inalámbrica. Nuestro grupo opto por utilizar un módulo Bluetooth para cumplir con este requerimiento, pero a la hora de crear una aplicación con una interface sencilla para móviles que se comunique con dicho módulo nos topamos con un problema. Virtuino.IoT, la aplicación escogida para desarrollar dicha aplicación móvil constaba de una versión gratuita y esta nos limitaba la cantidad de información, botones y grafico que podíamos incorporar en la pantalla. No nos percatamos de ello hasta que ya habíamos escrito casi todo el programa que vincularía el Microcontrolador de Arduino, el módulo Bluetooth y la app de Android. En consecuencia, tuvimos que realizar una investigación si había alguna forma de repararlo si tener que invertir más recursos económicos en el proyecto. Al concluir habíamos encontrado una aplicación similar llamada MIT App Inventor donde no contábamos con las otras limitaciones y que además no era necesario reescribir demasiado el código con el que ya disponíamos. Por ende, determinamos migrar a dicha aplicación.

Problemas en la construcción y ensamblaje de la Planta de ensayo:

- Rugosidad de los depósitos: Inicialmente, utilizamos bidones de agua de 6 litros como depósitos. Sin embargo, la rugosidad de sus paredes generaba ruido en las ondas ultrasónicas que rebotaban hacia el sensor, afectando la precisión de las mediciones.
- Alimentación de las bombas y del circuito completo: En un principio, planeábamos alimentar todo el circuito mediante una fuente creada a partir de un transformador, un puente de diodos y reguladores de voltaje, elaborada por uno de los integrantes del grupo en años anteriores. Sin embargo, esta fuente terminó quemándose cuando conectamos dos bombas de agua al mismo tiempo junto con los sensores y módulos del sistema.

  Nos resultó extraño, ya que el consumo combinado de las dos bombas debía ser de 120 mA más el consumo del resto del circuito. Anteriormente habíamos probado con una sola bomba junto con el sistema, y todo había funcionado correctamente. Decidimos medir la corriente que consumía una sola bomba con un multímetro, y descubrimos que el consumo era más de tres veces superior al especificado por el proveedor. Según este, las bombas no debían superar los 60 mA; sin embargo, nuestras mediciones indicaban un consumo de 195 mA por bomba.

  Realizando cálculos, las dos bombas consumían juntas 390 mA, y sumando el consumo del sistema completo, se superaban los 500 mA. El regulador de la fuente soportaba 500

mA por defecto, ampliándose hasta 1 A si contaba con un buen disipador. No obstante, como el regulador solo tenía el disipador básico, terminó quemándose por sobrecalentamiento.

Como solución, conseguimos una fuente de alimentación en formato de cargador que soportaba con creces el consumo de todo el sistema, resolviendo así el problema.

- Envío de datos demasiado rápido por el módulo Bluetooth, impidiendo la lectura correcta de datos: Enviábamos datos a la aplicación cada vez que se ejecutaba el ciclo del void loop, lo que generaba una superposición de datos de entrada y salida en el módulo Bluetooth. Esto ocasionaba pérdida de información tanto en el microcontrolador como en la aplicación. Para solucionarlo, implementamos un temporizador (timer) que envía los datos periódicamente a la aplicación sin detener la ejecución del programa, permitiendo que los datos sigan llegando desde la aplicación a través del módulo Bluetooth.
- Ruido en las mediciones del sensor ultrasónico: Inicialmente, recibíamos el dato de la distancia, calculábamos el porcentaje de llenado en base a esta y enviábamos ambos datos a la aplicación. Estos valores también se utilizaban para el control interno del sistema. Sin embargo, ocasionalmente, ocurrían errores o fluctuaciones en las mediciones que afectaban el proceso en curso. Para solucionar esto, cambiamos la forma en que se procesaban los datos. En lugar de usar el valor instantáneo, implementamos un sistema de promediado basado en las últimas mediciones. Esto redujo significativamente los errores. Esta solución se detalla en la sección "Explicación del código Arduino Mega 2560".
- Acumulación de sarro en los motores: Durante épocas de calor, el agua utilizada en las pruebas se evaporaba parcialmente, dejando impurezas que se acumulaban dentro de las bombas. Esto provocaba obstrucciones en las paletas internas, impidiendo su correcto funcionamiento. La solución fue desarmar las bombas y limpiarlas manualmente. No encontramos una solución alternativa en esta etapa, pero sugerimos implementar una función de gestión de mantenimiento en el futuro.

Proyecciones de Mejora:

En este apartado pensamos posibles mejoras de nuestro sistema, esto con el fin de ir mejorando y escalando este proyecto en los próximos años donde el proyecto alcance un nivel parecido a los de la industria.

Mejoras:

- Detector de metales: incluir un detector de metales a la salida del mezclador en caso de que alguna parte del sistema se rompa y caiga dentro de la producción.

- Registro de producción: configuraciones predefinidas que agilicen el proceso a su vez de agregar un calendario/cronograma para saber que procesos se realizaron anterior mente.
- Control de temperatura: agregar sensores de temperatura para monitorizar este apartado y agregar algún actuador que nos permita enfriar o calentar los fruido según se considere necesario.
- Agregar una plataforma mas potente para el muestreo de datos y control del sistema. Ejemplo: Raspberry Pi con sistema de gestión online.
- Fuerza del motor de mezclado: cambiar el motor DC que tenemos por uno con un mayor torque ya que cuando hay muchos líquidos en la producción a este le cuesta arrancar por la oposición que se le genera.
- Función de mantenimiento: agregar una función dentro de la aplicación o sistema que notifique el tiempo transcurrido desde el último mantenimiento o alertar cuando las horas de trabajo garantizadas hayan sido excedidas, indicando la necesidad de realizar mantenimiento preventivo.

Conclusiones:

El desarrollo de este sistema SCADA aplicado a una planta de producción que manipula líquidos nos permitió integrar conocimientos teóricos y prácticos en un proyecto de ingeniería. Mediante el uso de tecnologías como Arduino Mega 2560, sensores ultrasónicos, caudalímetros y módulos de comunicación Bluetooth, logramos automatizar y supervisar los procesos críticos de almacenamiento y mezcla de fluidos.

La implementación de este sistema refleja la importancia de la supervisión remota y la adquisición de datos en tiempo real, características esenciales en la Industria 4.0. Adicionalmente, el diseño modular y adaptable de nuestro sistema facilita futuras mejoras, como el control de temperatura o la integración de herramientas de registro y planificación.

Los desafíos enfrentados, desde la limitación de las aplicaciones móviles hasta problemas mecánicos en el ensamblaje, resaltaron la importancia de la planificación, la flexibilidad y el trabajo en equipo. Gracias a ello, no solo resolvimos los problemas, sino que también identificamos áreas clave para innovaciones futuras, como el registro automatizado de producción y el monitoreo de variables adicionales.

Este proyecto no solo cumplió con los objetivos iniciales, sino que también demostró cómo las herramientas tecnológicas modernas pueden transformar procesos industriales en sistemas más inteligentes y sostenibles.

Bibliografía:

Naylamp Mechatronics. (2017, 09 de abril). Tutorial de Arduino y sensor ultrasónico HC-SR04. https://naylampmechatronics.com/blog/10_tutorial-de-arduino-y-sensor-ultrasonico-hc-sr04.html

DFRobot. (2016, 13 de marzo). Water Flow Sensor - 1/8" (SKU: SEN0216). https://wiki.dfrobot.com/Water_Flow_Sensor_-_1_8__SKU__SEN0216

MIT App Inventor. (2015, 01 de julio). Documentation. Recuperado el [fecha de consulta], de https://appinventor.mit.edu/explore/library

Random Arduino Projects. (2018, 16 junio). Sensor de caudal con Arduino (Water Flow Sensor) [Video]. YouTube. Recuperado de https://www.youtube.com/watch?v=Jjha1pyXrZc

Random Arduino Projects. (2022, 18 de mayo). Arduino Projects: Sensor Applications [Lista de reproducción]. YouTube. Recuperado de https://www.youtube.com/playlist?list=PLCTD_CpMeEKS15LJHtAIocRCnHSLFuktA

Kio4. (2020, 01 de mayo). Proteus con Arduino. Recuperado el [fecha de consulta], de http://kio4.com/arduino/68_Proteus.htm

Anexos:

EquipoPI. IC1. GitHub. https://github.com/equipoPI/IC1