UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola Politècnica Superior d'Enginyeria de Manresa

Trabajo de final de grado

Planta piloto Laboratorio de minas

Grado en Ingeniería de Sistemas TIC
Curso 17/18

Autor: Pavel Alfredo Macutela Villarreal

Director: Teresa Escobet Canal y Josep Oliva Moncunill

Fecha:04-07-2018

Localidad: Manresa

Resumen

En este proyecto en el cual he trabajado como becario de la UPC durante un año, busca la monitorización y control de diferentes procesos de la planta piloto en el laboratorio de minas, que no estaba adaptado para poder obtener información. Para obtener la monitorización de los procesos se ha estudiado diferentes tipos de sensores, motores, módulos; con el fin de poder instalar estos elementos en los procesos, para obtener la información o controlar distintos aspectos del experimento que se realice. Otro caso de estudio es la forma de instalación de los sensores en el proceso y obtener una correcta lectura o actuar de una manera correcta en los diferentes procesos. Como resultado de este trabajo de final de carrera tenemos el programa el cual le pasamos unos parámetros iniciales, que recoge toda la información de los diferentes procesos a analizar y se pude ver cómo va evolucionando el experimento realizado.

Resume

In this project, in which I have worked as a UPC scholarship holder for a year, I am looking for the monitoring and control of different processes of the pilot plant in the mining laboratory, which was not adapted to obtain information. To monitor the processes, different types of sensors, motors and modules have been studied in order to be able to install these elements in the processes, to obtain the information or to control different aspects of the experiment being carried out. Another case study is how to install the sensors in the process and obtain a correct reading or act correctly in the different processes. As a result of this end-of-studies work we have the program which we pass to it some initial parameters, which gathers all the information of the different processes to be analyzed and we can see how the experiment is evolving.

Contenido

Indice de Ilustraciones

1 Introducción

1.1 Contextualización del trabajo

Este trabajo se ha centrado en el desarrollo de sensores y actuadores inteligentes utilizando como basa una plataforma Arduino, para ser instalados en diferentes máquinas de la planta piloto instalada en el laboratorio de minas de la EPSEM. El trabajo se ha realizado en el marco del proyecto europeo OptimOre, concretamente he contribuido básicamente en el Work Packages 8: Intelligent Control and Design durante 1 año como becario.

En esta parte del proyecto tratamos de capturar los valores que son importantes para poder obtener la mayor información de los procesos por lo que atraviesan los minerales y saber o ver si el experimento o prueba va por buen camino o sí hay mucha desviación con lo esperado.

También hemos analizado cuales son los sensores que mejor se adaptan a lo que necesitamos y que, además, sea compatible con la plataforma Arduino, que es el entorno en el que trabajaremos.

En este trabajo se presenta el desarrollo de los sensores y actuadores destinados a la máquina destinada a tareas de separación por gravedad (gravity separation).

1.2 OptimOre

El principal objetivo de Optimore es optimizar las tecnologías de procesamiento de trituración, molienda y separación para el procesamiento de minerales especialmente tungsteno y tántalo mediante un mejor control de proceso de producción de ajuste fino rápido y flexible basado en nuevos modelos de software, detección avanzada y estudio físico de proceso más profundo para aumentar el rendimiento en un 7-12% en los mejores procesos actuales de producción, aumentando el ahorro de energía en un 5% en comparación con las mejores técnicas de procesamiento.

La economía moderna depende en gran medida de materia primas específicas, y se prevé que esta dependencia aumentará en el futuro cercano. La mayoría de ellos son escasos en la UE y de poca pureza, y se mezclan en agregados complejos y de bajo grado que deben procesarse mediante un proceso de separación que consume grandes cantidades energía y de agua. En algunos casos esto hace que la explotación sea inviable debido a los gastos de producción son excesivos. Siendo la UE dependiente de algunos de estos materiales, según lo identificado por la iniciativa EIP (European Investment Practitioner), esta sociedad exige procesos de extracción más eficientes para contribuir a una mayor independencia europea de estas materias primas *Critical Raw Materials* (CRM).

Este proyecto busca aplicar tecnologías inteligentes para un procesamiento más eficiente y flexible de minerales como el tántalo y el tungsteno desde el proceso de trituración hasta el proceso de separación.

En un mercado actualmente dominado por la producción de China y Rusia, en Europa la producción de tungsteno (limitada) se concentra en Reino Unido, España y Portugal. Por el contrario, el tántalo es un elemento clave en los componentes electrónicos con una clara dependencia de la producción externa de la UE, ya que es naturalmente escaso en Europa (solo el 1% de la producción mundial se concentra en UE). En este contexto el proyecto Optimore propone la investigación y el desarrollo de tecnologías de modelado y control, utilizando detección avanzada y control industrial avanzado.

El proyecto Optimore consta de 10 Work Packages, en la ilustración 1 se muestra un esquema de cómo se desarrollará los diferentes paquetes de trabajo durante el proyecto; a continuación, los citare y explicar brevemente en qué consistía cada proceso:

- WP1 Project Management and Scientific Coordination

Configurar el marco organizacional para ejecutar el proyecto como se prevé en la Descripción del trabajo. Asegurar el logro de objetivos y tareas. Coordine los informes técnicos y administrativos comunes y administrar la garantía de calidad general.

- WP2 Ore Production Documents and System Analysis

Comparar el punto de partida del proyecto con las tecnologías más avanzadas disponibles, incluido un SoA (Arquitectura orientada a servicios) constante y vigilancia del mercado para asegurar que el desarrollo del proyecto esté siempre por delante de las técnicas actuales y proporcione un claro avance.
Además, este WP incluirá la selección de las tecnologías más adecuadas para la ejecución del proyecto, así como estudios sobre sus requisitos.

- WP3 Crushing

El objetivo principal es obtener modelos de trituración para maquinaria de trituración y para el procesamiento de tungsteno y tántalo. Dichos modelos se basan en el tipo de equipo, el modo operacional y el mineral. Se encontrarán y caracterizarán los parámetros específicos para un mejor rendimiento preciso, que incluyen:
    - previsión de consumo de energía.
    - pronóstico sobre la distribución del tamaño.
    - pronosticar la liberación de minerales.

- WP4 Milling

El objetivo de trituración es reducir el tamaño de los minerales por debajo de 1 mm con métodos húmedos o secos. El objetivo del paquete de trabajo abarca todos los modelos de molienda que operan con los parámetros de los equipos de molienda y con los procesos aplicados a los minerales de tántalo y tungsteno. Habrá pruebas de laboratorio y un proceso a escala real por validación y refinamiento del modelo y sus parámetros. También se estudiará el consumo de energía y la liberación de mineral para cada parámetro característico del mineral procesado.

- WP5 Gravity Separation

El objetivo principal es desarrollar modelos mejorados para equipos de separación por gravedad comúnmente utilizados en el procesamiento de tungsteno y tántalo. El rendimiento de la separación estará vinculado a la información mineralógica cuantitativa producida por QEMSCAN[1]. Los modelos desarrollados se utilizarán para auditar varias operaciones de procesamiento de minerales comparando el rendimiento esperado de la mineralogía con el rendimiento real.
El trabajo de modelado se utilizará para resaltar las fortalezas y debilidades de las técnicas de separación existentes e indicar la mejor manera de combinar procesos para el tratamiento de minerales de baja ley.

- WP6 Magnetic Separation

El objetivo principal es desarrollar un nuevo modelo de equipos de separación magnética comúnmente utilizado para el procesamiento de tungsteno y tántalo. La identificación adecuada de la variedad de mineral como Hübnerite, Wolframite o Ferberite en el caso de los minerales de tungsteno es importante para el éxito del rendimiento de separación magnética. Por lo tanto, el modelo estará vinculado al análisis mineralógico cuantitativo producido por microscopía estereoscópica en combinación con *Mineral Liberation Analysis* (MLA).

El trabajo de modelado se usará para resaltar las fortalezas y debilidades de las técnicas de separación existentes e indicar las áreas donde la separación magnética de los minerales de baja ley molidos es una opción viable.

- WP7 Froth Flotation

La flotación con espuma representa uno de los procesos más ampliamente utilizados para la separación de minerales. Aunque sus principios básicos se comprenden bastante bien, es difícil predecir cuantitativamente el resultado del proceso, debido a la complejidad de los diferentes micro procesos que se producen, que pueden interactuar de forma sinérgica o antagónica. Por lo tanto, el objetivo de este WP es el desarrollo y la evaluación de un enfoque predictivo para los equipos de flotación por espuma y los regímenes de reactivos utilizados comúnmente para el beneficio de minerales típicos de tungsteno y tántalo.

[1] QEMSCAN: Quantitative Evaluation of Minerals by Scaning electron microscopy

- WP8 Intelligent Control and Design

Los modelos proporcionan una comprensión profunda de los principios del proceso y funcionamiento operativo, pero para un control adecuado es necesario incluir diferentes herramientas de decisión de parámetros que operen bajo supuestos de dichos modelos y que puedan decidir el mejor conjunto de valores de parámetros, incluida la adaptación paramétrica específica para cada entorno dado y basado en los resultados de entrada pasados y experiencias pasadas (participación de razonamiento basado en casos).

Con todo este entendimiento y control de subprocesos diferentes, se generará un modelo de simulación completo que posibilita la evaluación de estrategias de decisión, analizando su viabilidad en función de los sensores disponibles y generando las mejores decisiones de proceso teniendo en cuenta el rendimiento, la calidad del mineral, consumo de energía y agua.

- WP9 System Integration and Validation

El objetivo es integrar los elementos de hardware y software para la validación y la validación adicional llevada a cabo en el campo. La validación se llevará a cabo en tres etapas:

  - Entorno virtual simulado utilizando software y experiencia propios.
  - Pruebas de planta piloto en el laboratorio de UPC.
  - Tantos sitios operativos de minas existentes como sea posible.
  - Soporte para el Proyecto Penouta Tantalum en desarrollo por Strategic Minerals Spain.

- WP10 Dissemination and Exploitation

Este paquete de trabajo coordinará las estrategias de explotación y las actividades de capacitación, comunicación y diseminación de todo el proyecto para asegurar la concientización del proyecto, el intercambio de conocimientos y la diseminación de los resultados del proyecto a la producción de maquinaria minera. El paquete de trabajo utilizará una variedad de medios de comunicación para interactuar e influenciar a los grupos destinatarios, comprender y abordar sus necesidades y ofrecer la comunicación adecuada en los momentos apropiados. Se buscará la comunicación con los actores relevantes dentro del campo de los sistemas de producción minera, mientras tanto se desarrollará el acceso a ellos a través de asociaciones y actores relevantes. Se espera que los resultados del proyecto se proporcionen como acceso libre a los conocimientos a través de publicaciones.

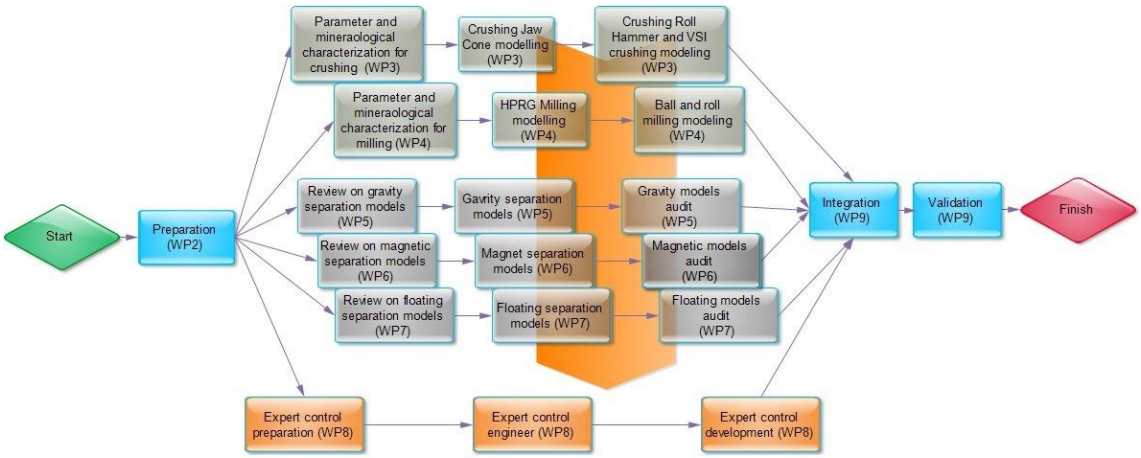


Ilustración 1: Esquema de seguimiento del proyecto

1.3 Tareas realizadas durante el proyecto

Por lo tanto, la instalación y la decisión de que sensores y como utilizarlos era la tarea que tendría que hacer yo durante duración del proyecto.

A lo largo del proyecto, se pretendía trabajar en diferente maquinas como son:

1. Analizador en línea de la scheelita basado en la luz violeta, con el que se busca determinar la cantidad de material en porcentaje que se extrae de una serie de muestras.
2. Monitorización de características de los flujos de entrada y salida, consumos, y control del flujo de entrada.
3. Shaking table (tabla clasificadora), se propuso el control de los flujos de entrada de agua e inclinación de la misma en los dos ejes, X e Y.
4. Del sistema de flotación se pretendía monitorizar el pH y controlar distintos flujos: productos químicos y la entrada de aire en las diferentes celdas de flotación.

Estos son los principales procesos que tiene que atravesar los minerales y los cuales tenemos que monitorizar y controlar.

Para poder hacer el desarrollo de estas aplicaciones necesitamos de un “cerebro” o plataforma de control con cierta inteligencia, por ello utilizaremos el famoso microcontrolador de la plataforma Arduino, en todo el proyecto utilizaremos Arduino Yun y Arduino Yun Mini ya que gracias a su microprocesador nos permite comunicarnos fácilmente con el mediante la red wifi.

Cada proceso tiene conectado diferentes sensores, los cuales nos dan una información, esta información es enviada al servidor, (ordenador central) el cual recoge todos los datos de todos los dispositivos (Arduinos Yun).

Esta información recogida por el ordenador central, el cual tiene el programa experto, este programa analizando los valores que le llega enviará información a los actuadores para que modifiquen ya sea el caudal de entrada, inclinación, más velocidad en los motores (variadores), con tal de cumplir el objetivo de reducir el consumo energético y de agua.

1.4 Objetivo

El objetivo de este trabajo final de grado es monitorizar la planta piloto, utilizando los sensores que más se adapten a las características que nos interesa conocer, obtener valores representativos para poder monitorizar y así poder hacer un mejor control de la planta piloto.

Por otra parte, tenemos la parte de control y actuación sobre estos procesos que ayudaran a mejorar la recuperación del material que interese. El acoplamiento de

Para ello debemos estudiar los diferentes procesos y determinar cuál es la información que se necesita para poder extraer los valores más representativos para poder monitorizar estos procesos.

Tanto el monitoreo como control necesitan acoplarse de manera efectiva para que todos los procesos funcionen de la manera correcta y que se pueda obtener la mayor y mejor control de los procesos, por lo tanto, el acoplamiento del hardware con el software es un factor importante a la hora de determinar el sensor que utilizaremos para la planta piloto.

Este es otro objetivo claro de nuestro trabajo, poder controlar los diferentes procesos desde un ordenador central.

Los objetivos principales de este trabajo serán:

- Conocer y estudiar cada proceso de los materiales, para poder obtener la mayor información que interesa.
- Estudio de la plataforma Arduino Yun para sacar el mayor provecho a la aplicación.
- Estudio de los diferentes sensores utilizados en cada uno de los procesos.
- Implantación del hardware en los diferentes procesos y establecer comunicación entre los diferentes dispositivos.
- Obtención de los valores leídos por los sensores.
- Preparación y utilización del código del Arduino.
- Monitorización de los diferentes procesos de la planta piloto.
- Control de parámetros requeridos de la planta piloto.

2 Arduino Yun

Este dispositivo es el que utilizamos para poder comunicarnos con los diferentes procesos, en este apartado comentaremos como funciona y algunas características del mismo.

Para la captación de la información requerida y el control de los diferentes procesos necesitábamos un hardware que se pudiera mover, ya que las maquina cambiaban de posición durante los experimentos, preferíamos que no tuvieran cables conectados, por ese motivo elegimos Arduino Yun el cual tiene una interfaz wi-fi que simplifica la comunicación con nuestro servidor.

2.1 Descripción del Arduino Yun

Arduino yun (Ilustración 2) es un microcontrolador como el que hemos utilizado a lo largo de la carrera, pero con unas características que nos ayudan y facilitan la comunicación con varios dispositivos, en nuestro caso nos facilita la comunicación con nuestro servidor.

Este dispositivo se distingue de otra placa Arduino por la capacidad que tiene de comunicarse con la distribución de Linux que tiene incorporado, ofreciendo un poderoso ordenador en red con la facilidad de un Arduino. Además, posee comandas de Linux con la que puedes escribir tus propios scripts de Shell y Python para lograr unas interacciones más robustas.

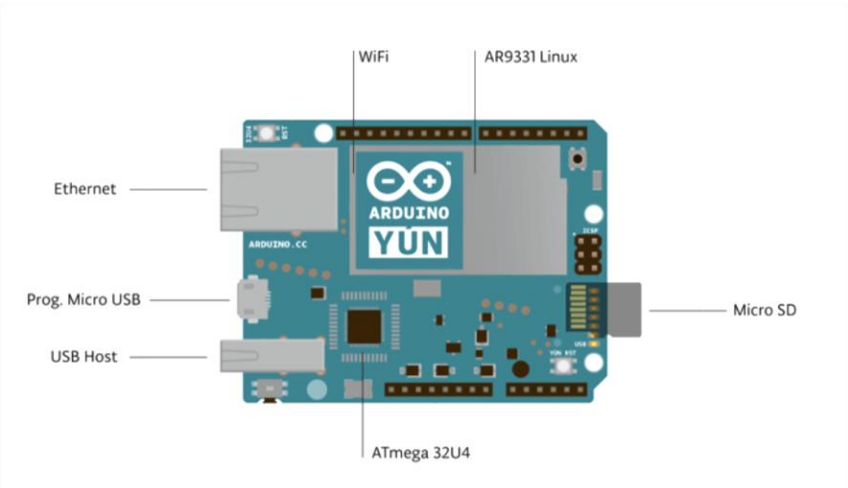
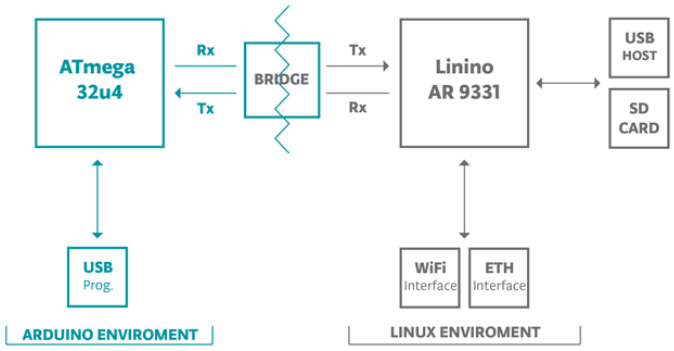


Ilustración 2: Arduino Yun

Esta placa, tiene como característica principal, que está compuesto por un microcontrolador basado en el ATmega32u4, que se encarga de ejecutar las aplicaciones de Arduino; y un microprocesador basado en el Atheros AR9331 (ilustración 3), los cuales se comunican con el protocolo Serial TTL.

El procesador Atheros admite una distribución de Linux basada en el OpenWrt que se llama Linino OS. Este procesador (AR9331) tiene un soporte Ethernet y wi-fi incorporado, lo que facilita la comunicación y la generación de una red de dispositivos; un puerto USB HOST (pendrive, teclado, ratón), una ranura para tarjetas micro-SD (almacenamiento de datos).

Por otra parte, el microcontrolador ATmega32u4 tiene conectada la parte de pines digitales de entrada y salida, entradas analógicas, un cristal de cuarzo de 16MHz, conexión micro USB (para alimentarlo y poder cargar los programas), un encabezado ICSP.



lustración 3: Comunicación entre microprocesador(Linux) y microcontrolador(ATmega32u4).

Otras características de esta placa se describen en las siguientes tablas.

Tabla 1: Características del microcontrolador

Arduino Yun Microcontrolador	
Microcontrolador	ATmega32u4
Operación de voltaje	5V
Pines Digitales E/S	20
Pines Analógicos	12
Corriente DC por pin E/S	40 mA
Corriente DC de 3.3V pin	50mA
Memoria Flash	32KB (4 KB para el Bootloader)
SRAM	2.5KB
EEPROM	1KB
Velocidad de reloj	16MHz

Tabla 2: Características del microprocesador(Linux)

Arduino Yun Microprocesador	
Procesador	Atheros AR9331
Arquitectura	MIPS
Voltaje de operación	3.3
Ethernet	802.11b/g/n 2.4GHz
Lector de tarjetas	Micro-SD
RAM	64 MB DDR2
Flash Memory	16 MB
EEPROM	1 KB
Velocidad de reloj	400 MHz

2.1.1 Alimentación

Esta placa no dispone de un regulador de tensión como tienen otros modelos Arduino, por lo tanto, debe ser alimentado exclusivamente a 5V en corriente continua, en cualquier caso, si lo alimentamos a más de este voltaje corre el riesgo de dañar la placa.

La alimentación a la placa se puede obtener, bien a través de la conexión micro-USB que incorpora nuestro Arduino, o bien a través del pin “Vin”

2.1.2 Entradas y Salidas

Los pines no son accesibles desde el microprocesador Atheros AR9331. Para poder acceder a los pines de entrada y salida lo tendremos que hacer a partir del microcontrolador 32U4.

Como ya se comentó anteriormente, tiene 20 pines digitales entrada/salida, los cuales operan a 5V;7 canales PWM y 12 canales de entrada analógica. cada pin puede proporcionar o recibir como máximo 40mA y tienen una resistencia interna pull-up de entre 20 y 50 kOhm desconectada por defecto.

En la ilustración 6 mostramos un esquema de la descripción de funciona de cada pin, a continuación, explicaremos estas funciones.

TWI:

En nuestra placa corresponde a los pines 2 (SDA) y 3(SCL), aunque tiene un pin externo, así si necesitamos utilizar los 2 y 3 para una interrupción, nos facilita la comunicación con el dispositivo.

Interrupciones Externas

Tiene hasta un máximo de 5 interrupciones externas, aunque no podemos utilizarlas todas a la vez. En el pin 3 (interrupción 0), 2 (interrupción 1), 0 (interrupción 2), 1

(interrupción 3) y 7 (interrupción 4). Se recomienda no utilizar las interrupciones 2 y 3 (pin 0 y 1 respectivamente) ya que son los pines de comunicaciones entre el microprocesador y el microcontrolador. Se tiene que tener cuidado si se quiere utilizar más de dos interrupciones externas.

PWM (Power wave modulation)

En los pines 3, 5, 6, 9, 10, 11 y 13; los cuales proporcionan una salida PWM de 8 bits.

SPI:

En el conector ICSP. Estos terminales soportan la comunicación SPI a través de la librería SPI. Estos son los pines que admiten comunicación SPI utilizando la librería SPI de Arduino. Estos pines no están conectados de manera física a ningún pin de Entrada/Salida así que, si queremos comunicar dos dispositivos mediante el protocolo SPI, el otro tiene que tener los 6 pines de ICSP, de lo contrario no funcionara la comunicación. Estos pines también están conectados a los pines GPIO del AR9331. Esto significa que el ATmega32U4 y el AR9331 se pueden comunicar mediante este protocolo.

Entradas Analógicas:

Tiene 12 entradas analógicas: A0-A5 y A6-A11, se pueden utilizar como Entrada/Salida digital. Cada entrada analógica proporciona 10 bits de resolución (1024) valores diferentes. Las entradas analógicas por defecto, miden de 0 a 5V, aunque es posible cambiar el rango del extremo superior usando el pin AREF

2.1.3 Memoria

La memoria en el AR9331 no está integrada dentro del procesador. La memoria RAM y la memoria de almacenamiento están conectadas externamente. El Yun tiene 64 MB de RAM DDR2 y 16 MB de memoria flash. La memoria flash está precargada en fábrica con una distribución Linux basada en OpenWrt llamada Linino OS. Puede cambiar el contenido de la imagen de fábrica, como cuando instala un programa o cuando cambia un archivo de configuración.

La instalación de Linino OS ocupa alrededor de 9 MB de los 16 MB disponibles de la memoria flash interna. Puede usar una tarjeta micro SD si necesita más espacio en disco para instalar aplicaciones.

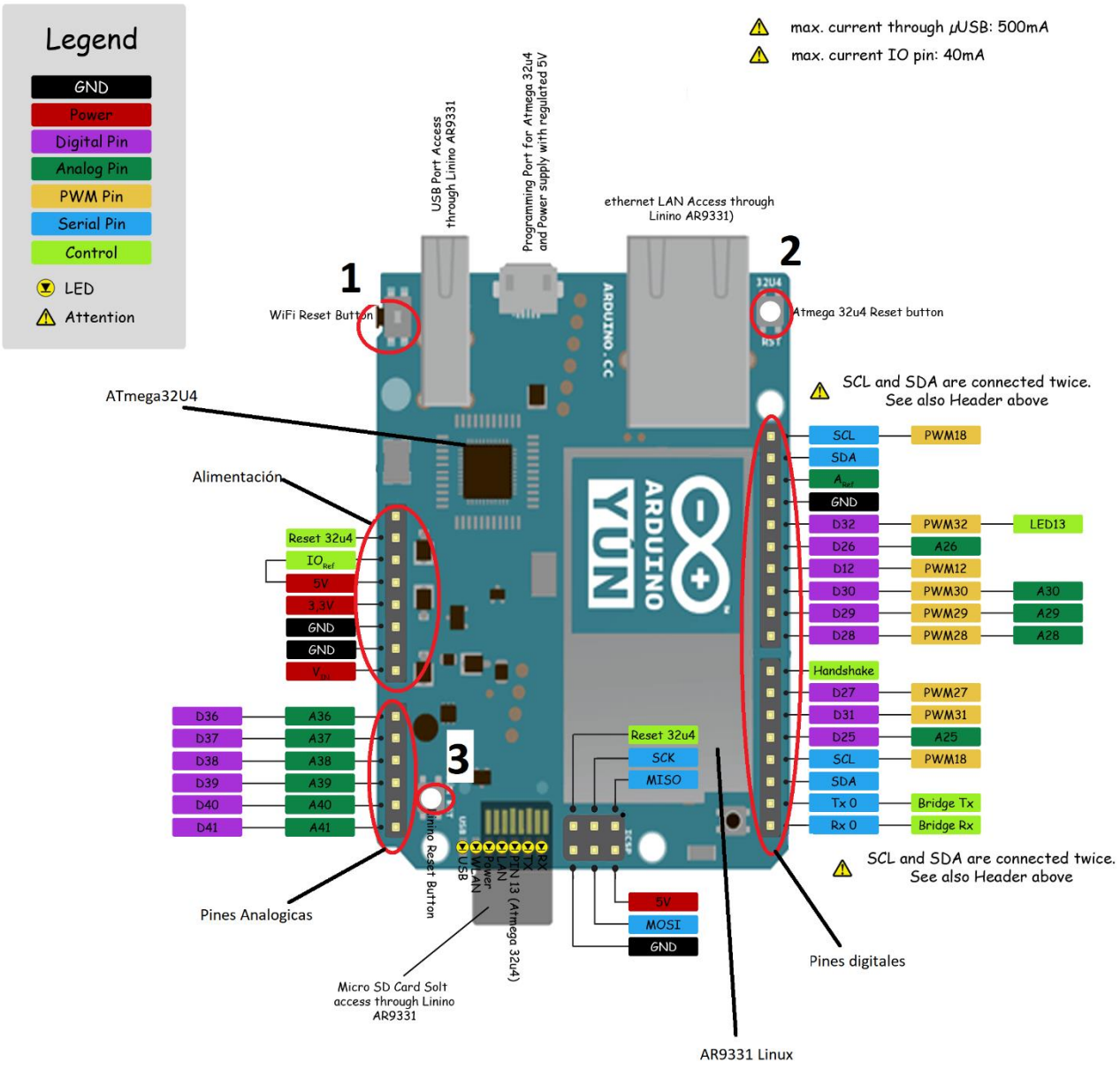


Ilustración 4: Distribución Arduino Yun

2.1.4 Comunicación

El Arduino Yun tiene varias facilidades para la comunicación con un ordenador, con otro Arduino, con otros microcontroladores. El ATmega32U4 tiene una comunicación serial UART TTL (5V) dedicada, también permite la comunicación en serie CDC a través de USB, aparece como un puerto virtual para el software de un ordenador. También actúa como un dispositivo USB 2.0 de velocidad completa, utilizando controladores COM USB estándar.

Los pines digitales 0 y 1 son usados para la comunicación Serie entre el microprocesador AR9331 y el microcontrolador ATmega32U4 por eso no utilizaremos estos pines ya sea como interrupciones o como para comunicación serie con otros Arduinos.

Una biblioteca SoftwareSerial permite la comunicación serial en cualquiera de los pines digitales de Yun. Los pines 0 y 1 deben evitarse ya que son utilizados por la biblioteca Bridge. ATmega32U4 también es compatible con I2C (TWI) y comunicación SPI. El software Arduino incluye una biblioteca Wire para simplificar el uso del bus I2C; para la comunicación SPI, usamos la biblioteca SPI.

Las interfaces Ethernet y WiFi incorporadas están expuestas directamente al procesador AR9331. Para enviar y recibir datos a través de ellos, usamos la biblioteca Bridge.

El yun también tiene capacidades de host USB a través de Linino OS. Puede conectar periféricos como dispositivos flash USB para almacenamiento adicional, teclados o cámaras web. Es posible que deba descargar e instalar software adicional para que estos dispositivos funcionen.

2.1.5 Botones

Como vemos en la ilustración 4 existen tres botones en la placa Arduino cada una de ellas con una función diferente.

1. **Botón RST WLAN:**
   Este botón tiene doble función, la principal función es restaurar la configuración Wi-fi de fábrica. Esta configuración activa un punto de acceso Wi-fi cuyo nombre de la red Wi-fi aparecerá con el nombre SSID “Arduino Yun-XXXXXXXXXXXX” donde las X son los dígitos de la dirección MAC de cada dispositivo. Una vez nos conectamos a esta red podemos conectarla a nuestra red local de casa y asignarle una dirección IP fija. Para poder conectarlo a nuestra red local nos conectamos a través del navegador con la dirección: 192.168.240.1. Cuando restablecemos la configuración Wi-fi también se reiniciará el entorno Linux.
   La segunda función de del botón WLAN RESET es restaurar la imagen de Linux a la imagen de fábrica predeterminada. Para restaurarlo se debe presionar este botón durante 30 seg. La restauración a la imagen de fábrica hace que pierda todos los datos guardados y los softwares instalados en la memoria flash incorporada conectada al AR9331.
2. **Botón RST 32U4:**
   Reinicia el microcontrolador ATmega32u4. Se usa normalmente para reinicia el programa que se está ejecutando cuando se bloquea. No se pierde el programa ya que se encuentran en memoria flash.
3. **Botón RST Yun:**
   Al presionar el botón de reset del AR9331, provoca el reinicio de sistemas Linux. Todos los datos almacenados en la RAM se perderán y los programas que están en ejecución se finalizarán.

3 Sensores

En este apartado explicaremos en detalle cuales son los sensores que hemos utilizado y algunas de sus características y por qué lo escogimos.

Para recoger la información de los diferentes procesos utilizamos diferentes sensores, los cuales como pueden ser temperatura, pH, nivel, etc. Buscamos que estos sensores sean de la plataforma Arduino, para que sea más fácil la comunicación y la obtención de los valores que nos interesa.

3.1. Sensor de temperatura MLX90614

El MLX90614 es un sensor de temperatura infrarrojo sin contacto, es posible conectar estos sensores con un autómata o procesador como Arduino para medir la temperatura de un objeto a distancia.

Existen distintos modelos del MLX90614 cada uno con un sufijo de tres letras. Los diferentes sensores difieren en el voltaje de operación, el número de sensores infrarrojos, y la posición del filtro.

La comunicación se realiza a través de SMBus, un subconjunto de bus I2C, por lo que resulta sencilla su lectura, y es posible conectar más de un sensor de forma simultáneamente o a través de la salida PWM(pin SDA) del sensor.

Frecuentemente se encuentran termómetros MLX90614 integrados en módulos como la GY-906 (ilustración 5) que incorporan la electrónica necesaria para conectarla de forma sencilla a un Arduino. En la mayoría de los módulos, esto incluye un regulador de voltaje que permite alimentar directamente a 5V.

Este tipo de termómetros infrarrojos tienen un gran número de aplicaciones, incluyendo sistemas de control de temperatura en instalaciones térmicas en edificios, control industrial de temperatura, detección de movimiento, y aplicaciones de salud.

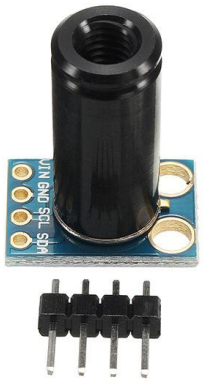


Ilustración 5: Modulo de temperatura GY-906

3.1.1 Descripción de pines

En la tabla 3 vemos la descripción de los pines del sensor MLX90614 que utilizamos para calcular la temperatura de entrada y salida del molino.

Tabla 3:Descripcion de los pines del sensor MLX90614

Nombre del pin	Función
SCL/VZ	Entrada de reloj en serie para protocolo de comunicación a 2 hilos. El zener de 5.7V está disponible en esta clavija para la conexión de transistores bipolares externos
SDA/PWM	Entrada/salida digital. En modo normal, la temperatura del objeto medido está disponible en esta patilla de ancho de pulso modulado. En el modo compatible con SMBus el pin es configurado automáticamente como NMOS de drenaje abierto.
VDD	Fuente alimentación externa
Vss	Tierra

3.1.2 Funcionamiento del sensor

El MLX90614 está construido a partir de 2 chips desarrollados y fabricados por Melexis:

1. Detector de termopila infrarroja MLX81101.
2. El acondicionador de señal ASSP MLX90302, especialmente diseñado para procesar la salida del sensor IR.

El conjunto incluye un amplificador de bajo ruido, un conversor ADC de 17 bits, un DSP (procesador digital de señal) y compensación de la temperatura ambiente (ilustración 6).

Gracias al amplificador de bajo ruido, ADC de 17 bits de alta resolución y la potente unidad DSP de MLX90302 se consigue una alta precisión y resolución del termómetro. Las temperaturas de objeto y ambiente calculadas son guardadas en la RAM del MLX90302 con resolución de 0,01 °C. Estos valores son accesibles por SMBus serie de 2 hilos (resolución de 0,02°C) o a través de la salida PWM (modulación por ancho de pulso) de 10 bits del dispositivo.

El MLX90614 viene calibrado de fábrica en un amplio rango de temperaturas: -40 a 125°C para la temperatura ambiente y -70 a 382 °C para la temperatura de objetos. La precisión estándar es de 0.5 °C referente a la temperatura ambiente, aunque existen

versiones médicas que ofrecen una resolución de 0.1ºC en temperaturas ambiente entre 35-38ºC.

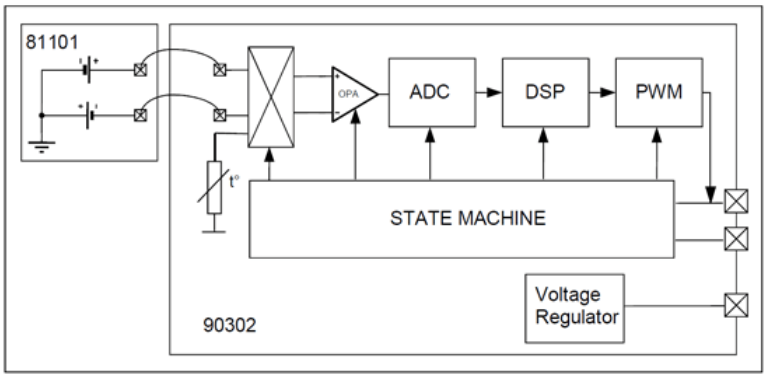


Ilustración 6: Diagrama ML90614

El MLX90614 dispone de dos modos de salida. La estándar es SMBus, un conjunto del I2C, con una resolución de 0.02ºC. También puede emplear una salida PWM de 10 bits para mediciones continuas, aunque con menor resolución 0.14ºC.

Es importante tener en cuenta la lectura del sensor solo es estable cuando el sensor se encuentra en equilibrio térmico con el ambiente. También puede afectarle la suciedad en la ventana del sensor.

También es importante entender que el MLX90614 es sensible a todos los objetos ubicados en su campo de visión. El ángulo de visión "FOV" (*Field Of View*) depende del modelo, y varía desde 5º a 80ºC. En el ángulo más amplio de 80º, el área de medición a 0.5 tiene un diámetro de 0.83 metros.

Es decir, los modelos de menos ángulo son apropiados para medidas puntuales en frente del sensor. Los sensores de ángulo amplio están diseñados para detectar incrementos de temperatura en una gran zona, por ejemplo, para detección de fallas en maquinaria.

La ilustración 7 nos muestra el campo de visión del sensor que utilizamos, en la cual vemos que puede llegar a medir objetos hasta un metro de distancia.

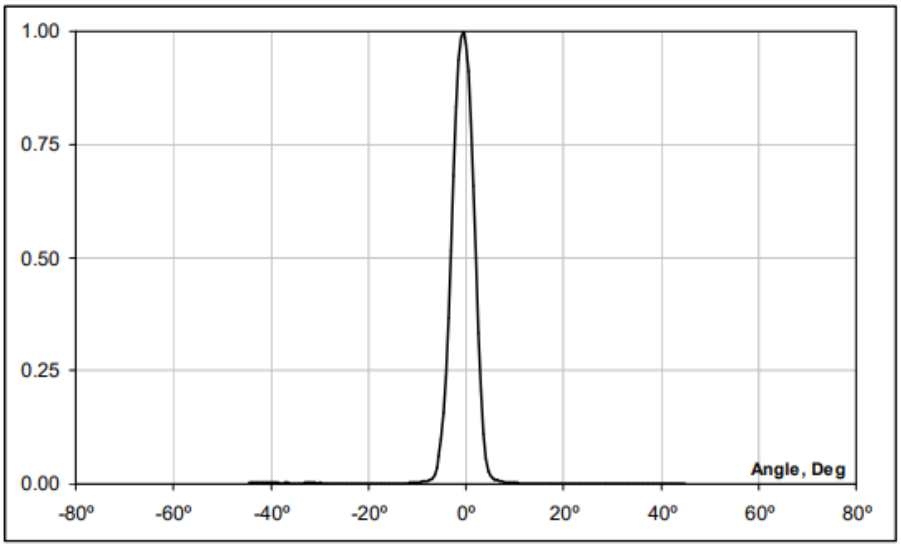


Ilustración 7: FOV típico del MLX90614xCI

3.1.2 Comunicación con Arduino

Como la familia de termómetros sin contacto MLX90614 incorpora todo lo necesario para conectarse al sistema que lo explota, su implementación hardware es muy sencilla, solamente necesita los componentes pasivos que acompañen al bus que se utilice.

Básicamente hay dos tipos de montajes con los que se pueden utilizar los sensores de temperatura MLX90614 controlándolos desde un microcontrolador: accediendo cuando sea necesario a sus registros de temperatura por SMBus / bus I2C o realizando una lectura continua de la temperatura por PWM.

En el ámbito de este proyecto utilizamos la comunicación SMBus para obtener la temperatura del objeto al que apuntamos.

Para obtener los valores que nos interesan, temperatura del objeto al que apunta y temperatura ambiente; utilizamos una librería (Adafuit, 2016) que nos facilitaba la obtención de los valores que necesitamos.

En la ilustración 8 podemos ver el esquema de conexión con el Arduino los pines SDA y SCL corresponden a los pines 2 y 3 respectivamente de la placa. Pero si estamos utilizando estos pines, podemos conectarlo a los pines externo SDA y SCL de la misma placa.

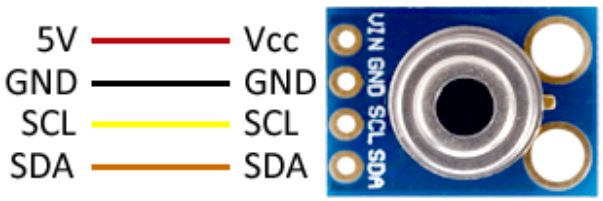


Ilustración 8: Esquema conexión MLX90614 –Arduino

3.2 Sensor de temperatura DS18B20

El DS18B20 es un sensor de temperaturas fabricado por la compañía Maxim Integrated. Proporciona la salida mediante un bus de comunicación digital que puede ser leído con las entradas digitales de Arduino.

Una de las ventajas del DS18B20 es que se comercializa tanto en un integrado TO-92 como en forma de sonda impermeable, lo que permite realizar mediciones de temperatura en líquidos y gases (ilustración 9).

El termómetro digital DS18B20 proporciona una precisión de 9 a 12 bits (12 bits por defecto), los cuales podemos cambiar la precisión. Este termómetro hace la medición en grados Celsius. Una de las mayores características de este sensor es que se comunica a través de un bus de 1 cable, que por definición solo utiliza una línea de datos para la comunicación con un microprocesador. Además, el DD18B20 puede derivar energía directamente de la línea de datos ("potencia parasita"), eliminando así la necesidad de una fuerte de alimentación externa. Cada DS18B20 tiene una serie única de 64 bits, lo que cual permite que varios de estos sensores funcionen con el mismo bus. Por lo tanto, es simple usar un microcontrolador que controle muchos de estos sensores distribuidos en un área grande. Son varias las aplicaciones que se pueden beneficiar de esta característica como pueden ser: controles ambientales HVAC, sistemas de control dentro de edificios, equipos o maquinaria industrial, sistemas de monitoreo y control de procesos.

Ilustración 9: Sensor DS18B20

3.2.1Beneficios y características

- La interfaz única de 1 cable requiere solo 1 pin para la comunicación.
- Deriva la potencia de la línea de datos ("potencia parasito"), es decir, solo requiere de dos pines para su operación (DQ y GND) y no requiere una fuente de alimentación local.
- Se puede programar la resolución desde 9 bits a 12 bits de precisión.
- La capacidad de multipunto simplifica las aplicaciones de detección de temperaturas distribuida.
- No requiere componentes externos.
- Convierte la temperatura a un valor digital de 12 bits en 750 ms (max).

- Configuración de alarma flexible no volátil que puede definir el usuario con el comando de búsqueda de alarma que identifica dispositivos con temperaturas fuera de los limites programados.

3.2.3 Descripción de pines

En la Ilustración 10 vemos las diferentes formas que se fabrica el integrado DS18B20, en este proyecto se utilizara el estándar TO-92 que es el que está integrado en la Ilustración 9.

En la tabla 4 se muestra la descripción y utilización de los pines del sensor.

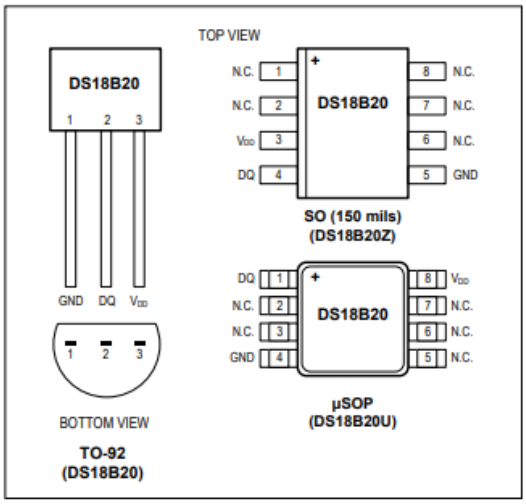


Ilustración 10: Configuración de pines del DS18B20

Tabla 4: Descripción de los pines del DS18B20

Pin			Nombre	Función
S0	µSOP	TO-92		
1,2,6,7,8	2,3,5,6,7	-	N. C.	No conectados
3	8	3	Vdd	Opcional Vcc. Vcc conectado a GND si lo queremos en modo parasito
4	1	2	Dq	Datos E/S de la interface de 1-Wire. También proporciona alimentación en modo “alimentación parasito”
5	4	1	GND	Tierra

3.2.4 Funcionamiento del DS18B20

La Ilustración 11 muestra el diagrama de bloques del DS18B20. La ROM de 64 bits almacena el código de serie único del dispositivo.

El scratchpad contiene el registro de temperatura de 2 bytes que almacena la salida digital del sensor de temperatura. Además, el scratchpad proporciona acceso a los registros de disparo de alarma superior e inferior (TH y TL). El registro de configuración permite al usuario establecer la resolución de la conversión de temperatura digital en 9, 10, 11 y 12 bits.

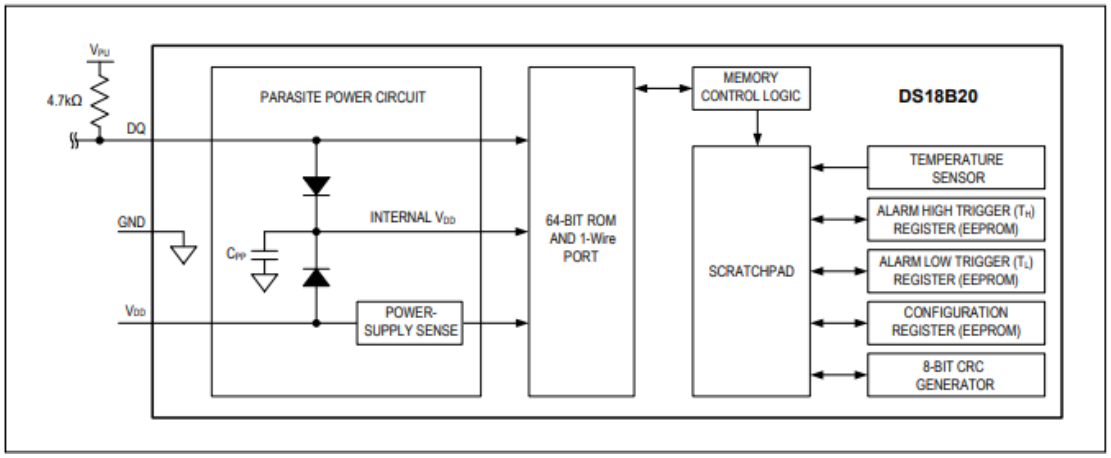


Ilustración 11: Diagrama de bloque del DS18B20

Los registros TH y TL no son registros volátiles, con lo que conservaran sus valores cuando el dispositivo se apague. El DS18B20 utiliza el protocolo exclusivo 1-Wire que implementa la comunicación utilizando una señal de control. Esta línea de control necesita de una resistencia pull-up ya que todos los dispositivos están vinculados al bus a través de un puerto de tres estados (el pin DQ en el DS18B20). En este sistema de bus el microprocesador (dispositivo maestro), identifica y de dirige a los dispositivos en el bus utilizando el código único de 64 bits. La cantidad de dispositivos que se pueden conectar a la misma línea de bus es virtualmente ilimitada.

Como hemos dicho, internamente el sensor DS18B20 es más complicado de lo que en principio podríamos creer. Está formado por un procesador con múltiples módulos, que se encargan de controlar la comunicación, medir la temperatura, y gestionar el sistema de alarmas.

Una de las principales ventajas de DS18B20 es su bus de comunicación 1-Wire que le permite realizar la transmisión empleando únicamente un cable de datos. Para ello, 1-Wire está basado en un complejo sistema de timings en la señal, entre el dispositivo emisor y el receptor (Integrate, Maxin, s.f.).

La señal de comunicación empleada en 1-Wire, así como sus timings, son realmente largas y complejas. La mayor desventaja del sistema 1-Wire es que requiere un código complejo, lo que a su vez supone una alta carga del procesador para consultar el estado de los sensores. El tiempo de adquisición total de una medición de 750ms.

En cualquier caso, el bus 1-Wire requiere una resistencia de pull-up de 4,7KOhm entre Vdd y GND para que funcione correctamente.

Que el DS18B20 disponga de una resolución por defecto de 0.0625ºC, no significa que esa sea su precisión. Sin embargo, el DS18B20 es considerablemente preciso en todo el rango –10°C de +85°C. Podemos consultar las desviaciones medias a 3 sigma en la ilustración 12.

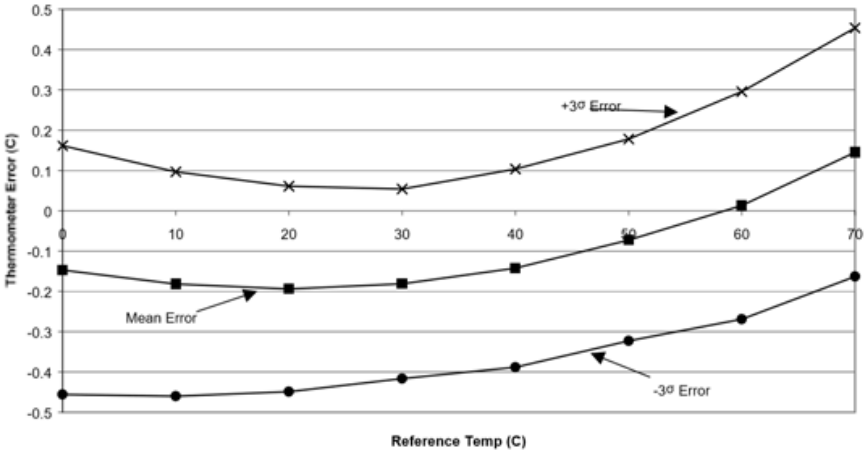


Ilustración 12: Curva de rendimiento tipica

3.2.5 Comunicación con Arduino

Para la comunicación con el Arduino y obtener la temperatura del agua al proceso, lo hacemos mediante la utilización de dos librerías: 1-Wire para poder comunicarnos a través de un solo cable con el sensor (PaulStoffregen, 2012) y la librería del sensor DS18B20 (Burton, 2015).

Con estas librerías podemos obtener la temperatura del sensor conectándolo al Arduino. En este proyecto no utilizamos la "alimentación parasito" conectamos 3 cables con nuestro Arduino.

En la Ilustración 13 se muestra el esquema de conexión con nuestro Arduino.

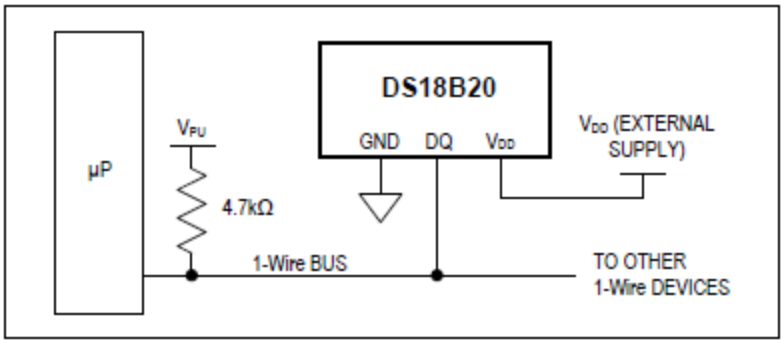


Ilustración 13: Esquema de conexión con Arduino

3.3. Sensor de PH

El sensor de pH es utilizado para medir la acidez o la alcalinidad de una solución. La medición de este valor se utiliza en laboratorio químicos, industrias alimenticias y de las bebidas, plantas de tratamiento de agua.

El pH es una medida de acidez o alcalinidad de una disolución, la escala de pH varía de 0 a 14. El valor del pH de determinada sustancia está directamente relacionado a la proporción de las concentraciones de los iones de hidrógeno [H+] e hidroxilo [OH-]. Si la concentración de H+ es mayor que la de OH-, el material es ácido; el valor del pH es menor que 7. Si la concentración de OH- es mayor que la de H+, el material es básico, con un pH con valor mayor que 7. Si las cantidades de H+ y de OH- son las mismas, el material es neutral y su pH es 7. Ácidos y bases tienen, respectivamente, iones de hidrógeno y de hidroxilo libres. La relación entre los iones de hidrógeno y de hidroxilo en determinada solución es constante para un dado conjunto de condiciones y cada uno puede ser determinado desde que se conozca el valor del otro.

El medidor de pH que utilizaremos en este proyecto es el de la ilustración 14, como se ve en la imagen consta de dos partes:

1. Electrodo (la parte que se introduce en el líquido).
2. Circuito integrado.

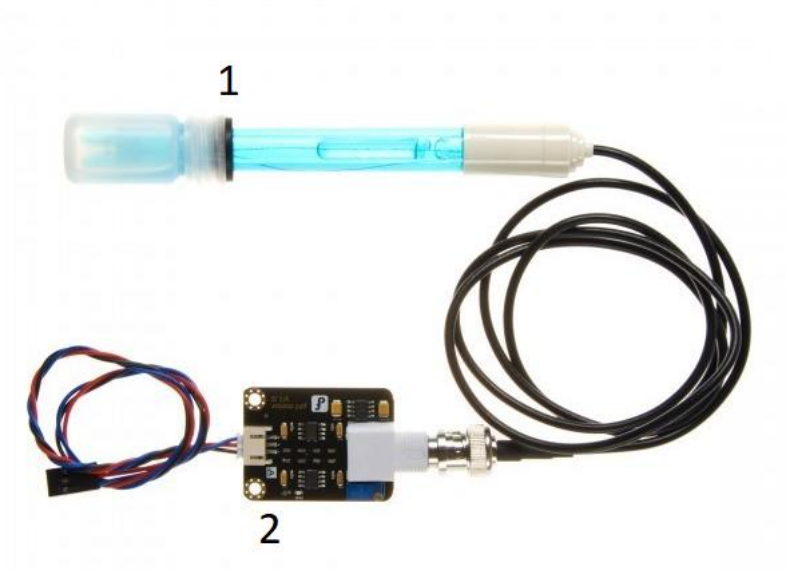


Ilustración 14: Sensor pH

3.3.1 Funcionamiento del sensor de pH.

El electrodo de pH puede ser considerado como si fuera una batería, con una tensión que varía conforme el pH de la solución medida.

Se puede cuantificar de forma precisa mediante un sensor que mide la diferencia de potencial entre dos electrodos: un electrodo de referencia (de plata/cloruro de plata) y

un electrodo de vidrio que es sensible al ion de hidrógeno. Esto es lo que formará la sonda. Además, hay que utilizar un circuito electrónico (ilustración 15) para acondicionar la señal adecuadamente y que podamos usar este sensor con un microcontrolador, como por ejemplo con Arduino.

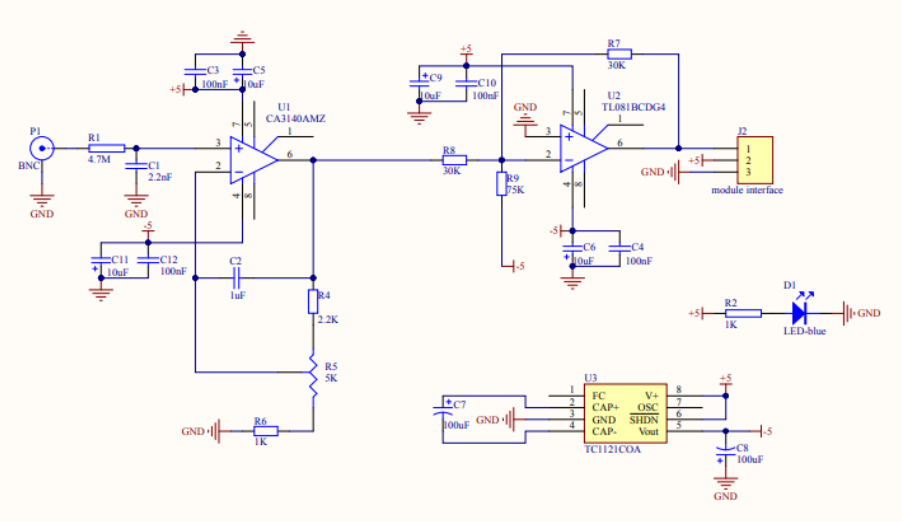


Ilustración 15: Circuito integrado del medidor pH

Asimismo, una sonda de pH que mide la actividad de los iones de hidrógeno mediante un proceso en el que genera una pequeña cantidad de tensión entre el sensor y el electrodo de referencia. La tensión se mide como una corriente eléctrica y el voltaje generado en esta corriente se convierte en un valor de pH mediante un algoritmo. El Arduino el cual recibe un valor en mili voltios por el pin analógico conectado al circuito integrado. Este valor tendremos que regularlo gracias al potenciómetro para obtener una medición correcta.

3.3.2 Calibrar el sensor

Para obtener valores del sensor que sean correctos en nuestro sistema, el primer paso que habrá que hacer es calibrar el sensor.

La ilustración 16 muestra la relación que hay entre los valores obtenidos y el valor del pH que corresponde.

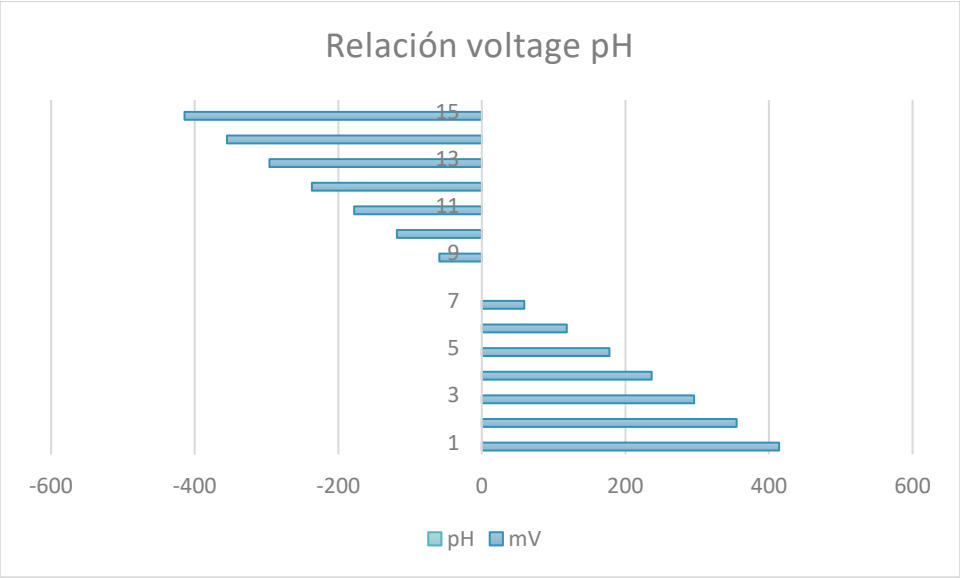


Ilustración 16: Relación voltaje -pH

Para obtener una mayor precisión al obtener el pH de una solución, tenemos que calibrar el sensor para ello seguimos los pasos del fabricante (DFRobot, 2017):

1. Conectar el electrodo al circuito integrado y los cables que van al Arduino como indica la Ilustración 17.
2. Subir el código de muestra al microcontrolador (Ejemplo, s.f.).
3. Colocar el electrodo de pH en una solución estándar que conozcamos el valor 7 p.e. Visualizamos el valor que captura el sensor por el monitor serie, comparamos con el valor real, en este caso 7, y la diferencia será nuestro Offset (una variable global en el código de muestra).
4. Para una solución Acida, introducimos el electrodo en una solución cuyo valor pH será 4.00. Ajustamos el potenciómetro hasta que la lectura del electrodo sea la esperada.
5. Para una solución básica o Alcalina, igual que el paso 4 pero con una solución de valor 9.

3.3.3 Comunicación con el Arduino

Este sensor se conecta las salidas del circuito integrado (ilustración 18) al Arduino una de estas salidas ira a una entrada analógica de nuestro microcontrolador, por la entrada analógica recibiremos una cantidad de voltaje, el cual con un cálculo algorítmico obtendremos el valor de pH de la solución que introducimos el electrodo. No necesitamos ninguna librería externa.

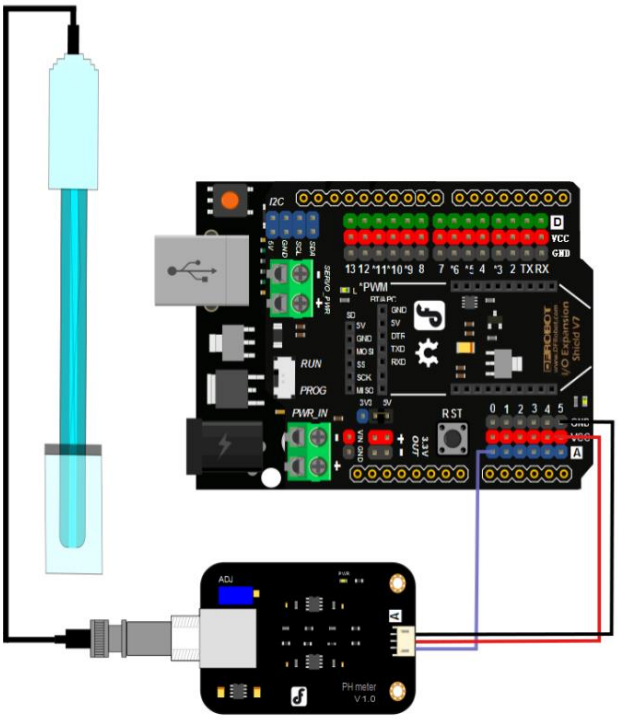


Ilustración 17: Esquema conexión sonda pH-Arduino

3.4. Sensor de nivel de agua

El sensor que utilizaremos será como el de la ilustración 18, en la tabla 6 vemos la descripción de cada uno de los pines de este sensor.

Ilustración 18: Sensor de nivel

Tabla 5: Descripción de pines del sensor de nivel

Nombre pin	Función
+	5V conectados al Arduino
-	Tierra
S	Señal de salida del sensor

Sensor de nivel de agua fácil de usar. Es un sensor de bajo costo y compacto que mide por medio de las trazas estañadas expuestas que tanta agua está en contacto con el sensor. Mide los valores analógicos que van desde 0 a 1023. Este artículo tiene bajo consumo de energía y alta sensibilidad, que son las mayores características de este módulo.

Es un sensor de bajo consumo de energía y con mucha sensibilidad, lo que nos es muy útil para el proceso que queremos monitorear.

Algunas características de este sensor proporcionadas por el fabricante las podemos observar en la tabla 7.

Tabla 6:Características sensor de nivel

ESPECIFICACIONES	
Modelo:	K-0135
Voltaje de operacion	5V
Corriente	Hasta 20mA
Salida	Analógica
Área de detección	40 x 16 mm
Material del PCB	FR4 HASL
Temperatura de trabajo	10C a 30C
Humedad recomendada	10% a 90%
Peso	3g
Dimensiones	65 x 20 x 8 mm

3.4.2 Funcionamiento del sensor de nivel

Funciona directamente con cualquier tarjeta de desarrollo con entrada analógica, así como microcontroladores que tengan canales de ADC como AVRs o PICs. Esta placa consta de principalmente tres partes: conectores, una resistencia de 1MOhm y varias líneas de cables conductores.

Mide la conductividad que hay en las líneas expuestas a tierra (GND). Las líneas de este sensor tienen una resistencia pull-up de 1MOhm. Esta resistencia aumenta hasta que caiga una gota de agua o este sumergido en un recipiente, este valor es enviado al microcontrolador a través del pin analógico.

Este sensor es capaz de calcular el nivel de agua que esta introducido el sensor, a partir de una serie de experimentos. También podemos conectarlo a un pin de entrada digital poniendo un lindar

3.4.3 Comunicación con el Arduino

Este sensor mide 4 cm de largo, que es el máximo de cantidad de agua, líquido que puede medir, lo cual repartimos en franjas de 5mm para poder saber cuántos milímetros está el sensor tocando con el agua.

Teniendo en cuenta esto, para obtener cuantos milímetros está sumergido en el líquido hicimos pruebas introduciéndolo en líquido y obteniendo que valor de lectura obtenemos en cada franja, en cada franja del sensor, la lectura del sensor es un valor diferente con lo que podemos saber a qué nivel se encuentra el líquido.

La ilustración 19 podemos ver el esquema de conexión con nuestra placa, este sensor tiene tres salidas como detallamos en la tabla 5.

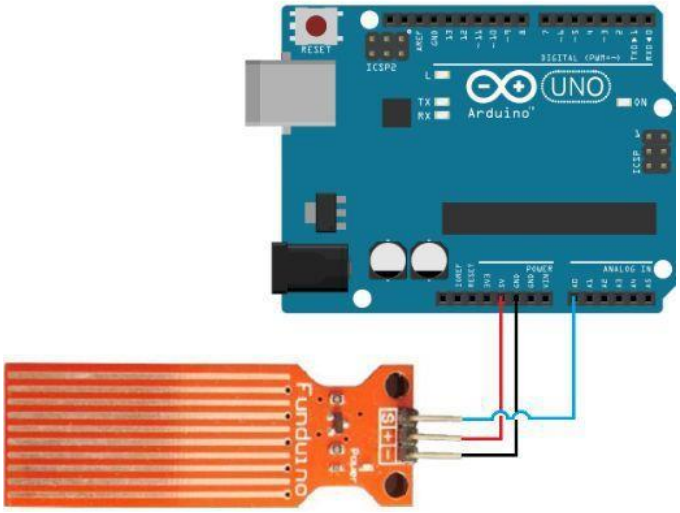


Ilustración 19: Esquema conexión del sensor de nivel con el Arduino

3.5. Sensor de Flujo

Un caudalímetro es un sensor que permite medir la cantidad de agua o gas que atraviesa una tubería. Este caudal depende de varios factores, principalmente de la sección de tubería y de la presión de suministro.

Las unidades en el sistema, internacional para medir este flujo son m^3/s, aunque también se puede medir en l/min, en el ámbito del proyecto utilizaremos l/min.

Hay diferentes tipos de medidores de flujo como pueden ser: YF-S401 (ilustración20), YF-S201 (ilustración 21), FS300A, FS400A. Cada uno de ellos tiene unas características específicas en cuanto al cálculo del flujo. En el proyecto utilizamos los sensores YF-S401 (1/8”) y YF-S201 (1/2”). Los cuales cada uno tiene una ecuación diferente de calcular el caudal que atraviesa por cada uno de ellos.

Ilustración 20: Sensor YF-S401

Ilustración 21: Sensor YF-S201

En la tabla 7 podemos ver la descripción de cables de estos sensores, la primera diferencia que vemos es que no tiene pines de salida, sino que son cables, por lo general todos los fabricantes de este sensor utilizan estos colores para poder conectarlo a la aplicación.

Tabla 7: Descripción cables sensor de flujo

Color	Función
Rojo	Alimentación 5V
Negro	Tierra (GND)
Amarillo	Señal , conectado a un pin de interrupción

3.5.1 Funcionamiento del sensor de flujo

El sensor internamente tiene un rotor cuyas paletas tiene un imán, la cámara en donde se encuentra el rotor es totalmente aislado evitando fugas de agua, externamente a la cámara tiene un sensor de efecto hall (Wikipedia, s.f.)que detecta el campo magnético del imán de las paletas y con esto el movimiento del rotor, el sensor de efecto hall envía los pulsos por uno de los cables del sensor, los pulsos deberán ser convertidos

posteriormente a flujo pero esto ya es tarea del Arduino o controlador que se desee usar.

Todos los modelos tienen tres cables para su conexión, rojo y negro para la alimentación y amarillo para la salida de los pulsos.

La salida de pulsos es una onda cuadrada cuya frecuencia es proporcional al caudal. El factor de conversión de frecuencia (Hz) a caudal (L/min) varía entre modelos y depende de la presión, densidad e incluso del mismo caudal.

Para el caso del sensor de ½" el factor de conversión promedio proporcionado por el fabrícate es:

$$f(Hz) = 7.5\ x\ Q(L/min)$$

En el caso del sensor de 1/8" el factor de conversión proporcionada por el fabricante es:

$$f(Hz) = 98\ x\ Q(L/min)$$

Llamaremos nosotros K al factor de conversión, siendo K=7.5 para el sensor de ½", K=5.5 para el sensor de ¾", K=98 para el sensor de 1/8" y 3.5 para el sensor de 1"; trabajar con dichos valores no nos garantiza precisión (precisión del 5 %), pero nos pueden servir para aplicaciones simples, si necesitamos mayor exactitud necesitamos calibrar y calcular dicho factor.

3.5.2 Comunicación con Arduino

Para poder obtener información de este sensor y poder controlar los flujos de agua de entrada y salida del sistema, debemos conectarlo con indica la tabla 7.

Además, estos sensores tienen una dirección establecida como muestra en la ilustración 22. Es importante realizar el montaje en la dirección que nos indica el fabricante, ya que si lo realizamos de manera incorrecta podemos obtener valores erróneos.

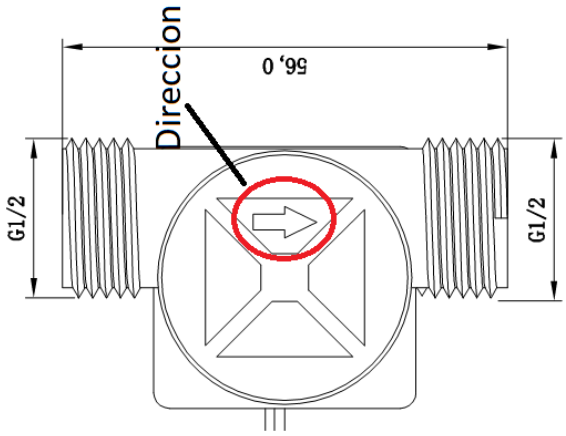


Ilustración 22: Dirección del sensor de flujo

Para calcular el flujo que circula por el sensor lo conectaremos a un pin del Arduino. Para poder detectar cuantos pulsos se producen, este pin tendría que estar configurado como una interrupción externa, como hemos comentado en la descripción del Arduino preferiblemente en los pines 3(interrupción 0) y 2(interrupción 1).

Para calcular el caudal que circula por el caudalímetro, tenemos que tener las interrupciones que se producen durante un 1 segundo, con este valor aplicamos la fórmula que nos proporciona el fabricante.

4 Actuadores y su control

4.1 Motores

Los motores son los principales actuadores en todo nuestro sistema. Por sus cualidades y por la manera de poder controlar y saber su posición en todo momento.

El tipo de motor utilizado es un motor paso a paso, es un dispositivo electromecánico que convierte una serie de pulsos eléctricos en desplazamientos angulares, lo que significa que es capaz de girar una cantidad de grados (paso o medio paso) dependiendo de sus entradas de control.

Los motores paso a paso son ideales para la construcción de mecanismos en donde se requieren movimientos muy precisos. La característica principal de estos motores es el hecho de poder moverlos un paso a la vez por cada pulso que se le aplique. Este paso puede variar desde 90° hasta pequeños movimientos de 1.8°, Es por eso que ese tipo de motores son muy utilizados, ya que pueden moverse a deseo del usuario según la secuencia que se les indique a través de un microcontrolador.

Estos motores poseen la habilidad de quedar enclavados en una posición si una o más de sus bobinas está energizada o bien total mente libres de corriente. Aunque son poco eficientes ya que derrochan bastante energía comparados con otros motores eléctricos convencionales porque consumen más amperios que los motores normales.

Consumen el máximo de corriente cuando están parados y por este motivo suelen calentarse bastante hasta llegar al punto que dejen de funcionar.

Los motores paso a paso que utilizamos son bipolares (podemos ver su esquema en la ilustración 23), internamente, son más sencillos ya que únicamente tienen 2 bobinas.

La complejidad de estos motores está en el driver, porque no sólo tiene que permitir pasar la corriente por la bobina, sino que tiene que cambiar la polaridad de la corriente.

De esta forma, en un primer momento, la corriente viaja en una dirección, creando un campo magnético norte/sur, y en el siguiente momento la dirección de la corriente se invierte, creando un campo magnético sur/norte.

Precisamente por eso se llaman bipolares, porque las bobinas están pensadas para cambiar de polaridad durante el movimiento.

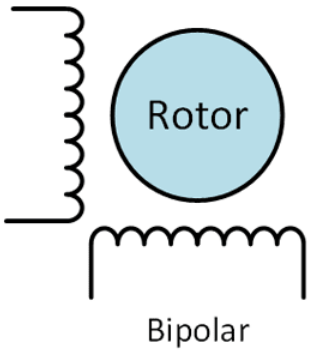


Ilustración 23:Esquema motor paso a paso bipolar de 4 cables

Existen multitud de drivers para motores paso a paso. Los más conocidos en el mundo de la impresión 3D y la robótica están basados en el chip Allegro A4988, el Texas Instruments DRV8825.

Para los motores más grandes, que necesitan más potencia, se usan drivers más potentes. Los más conocidos (y caros) son los Gecko. Este tipo de drivers los encuentras en las máquinas CNC.

4.1.1. Nema 17

En la ilustración 24 vemos el motor Nema 17 que utilizamos en la gran mayoría de control de procesos. Este motor enganchado a un mecanismo de engranajes hará el accionamiento de abrir o cerrar las válvulas que permiten el paso de agua o levantar un eje de la mesa de sacudidas.

Ilustración 24: Motor paso a paso Nema 17

En la tabla 8 vemos algunas de las especificaciones del motor que utilizamos en ente proyecto.

Tabla 8: Características del Motor Nema 17

Características del motor Nema 17	
Angulo de paso	1.8 º
Par de retención	0.59 N.m (84 oz.in)
Corriente nominal / fase	1.7 A
Numero de fases	2
Resistencia de fase	1.8 ohms
Voltaje nominal	3.06 V
Inductancia	3.8 mH +- 20%
Inercia del rotor	82 g cm^2

La ilustración 25 muestra la curva de rendimiento del motor Nema 17, en nuestro caso corresponde a la línea trazada de color negro ya que alimentamos.

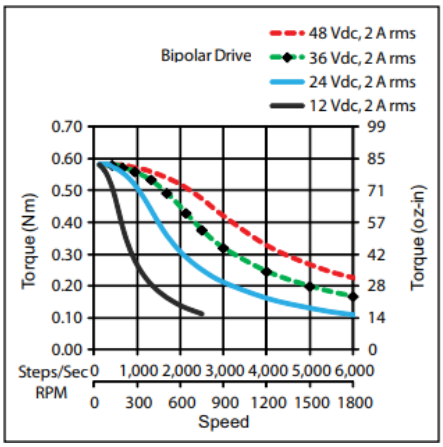


Ilustración 25: Curva de rendimiento del Nema 17

4.2. Controladores

4.2.1 DRV8825

El DRV8825 son controladores (drivers) que simplifican el manejo de motores paso a paso desde un autómata o procesador como Arduino, en la ilustración 26 vemos el driver que utilizamos.

Estos controladores nos permiten manejar los altos voltajes e intensidades que requieren estos motores, limitar la corriente que circula por el motor, y proporcionan las protecciones para evitar que la electrónica pueda resultar dañada.

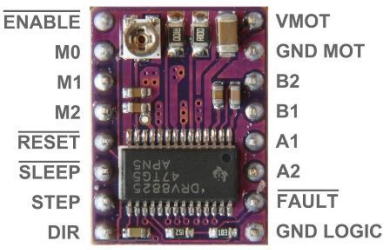


Ilustración 26:DRV8825

Para su control únicamente requieren dos salidas digitales, una para indicar el sentido de giro y otra para comunicar que queremos que el motor avance un paso. Además, permiten realizar microstepping, una técnica para conseguir precisiones superiores al paso nominal del motor.

El DRV8825 permite trabajar con tensiones altas máx. 45V, e intensidades de hasta 2.5A. Además, como vemos en las características de la tabla 9, permite un nuevo modo de microstepping (1/32).

Tabla 9: Características del controlador DRV8825

Modelo	DRV8825
Intensidad máxima	2.5 A
Tensión máxima	45 V
Microsteps	32
Rs típico	0.1
Formulas	I_max = Vref / (5 * Rs)
	Vref = I_max * 5 * Rs

Este controlador puede alcanzar altas temperaturas durante su funcionamiento y es necesario disipar el calor para que el dispositivo no se dañe. Para intensidades superiores a 1.5A es necesario añadir un sistema de disipación de calor, e incluso ventilación forzada. En el ámbito de la planta piloto no utilizamos ventilación forzada ya que disponemos de un relé que habilita o deshabilita el paso de la corriente para evitar que el driver se caliente. Esta práctica no sería útil si el motor tiene que soportar una fuerza, ya que perdería toda la fuerza.

Dispone de protecciones contra sobreintensidad, cortocircuito, sobretensión y sobretemperatura. En general, es un dispositivo robusto y fiable siempre que realicemos la conexión correctamente, e incorporemos disipación de calor si es necesario.

En la tabla 10 mostramos la descripción de los pines que se muestran en la ilustración 26.

Tabla 10: Descripción de pines del driver DRV8825

Nombre del pin	Función
ENABLE	Permite que el driver pueda enviar corriente al motor
M0	configuración de los micro pasos
M1	
M2	
RESET	Normalmente conectado a Sleep para que el driver funcione
SLEEP	Normalmente conectado a Reset para que el driver funcione
VMOT	Alimentación de los motores 8-45V, también alimenta a parte digital
GND MOT	Grand del voltaje de entrada
B2	Bobina B
B1	Bobina B
A1	Bobina A
A2	Bobina A
FAULT	Envía un pulso si se ha perdido un paso
GND LOGIC	Ground del voltaje de entrada
STEP	El Arduino manda un pulso para que el motor avance un paso
DIR	Indica la dirección de giro del motor

El DRV8825 son muy empleados en una gran variedad de proyectos que requieren el uso de motores paso a paso, como máquinas de CNC, plotters, robots que dibujan, impresoras 3D, y escáneres 3D.

Como en la mayoría de los controladores de motores el componente fundamental es un puente-H. El DRV8825, destinados a controlar motores paso a paso, se dispone de dos puentes-H (uno por canal) constituidos por transistores MOSFET, en la ilustración 27, se puede ver claramente estos dos componentes.

Sin embargo, a diferencia de controladores más simples como el L298N o el TB6612FNG, que presenta una electrónica relativamente simple, el DRV8825 tienen una electrónica considerablemente más compleja.

Uno de los motivos para esta complejidad es que únicamente requieren dos señales digitales de control para hacer funcionar el motor, y que incorporan las protecciones necesarias para su manejo.

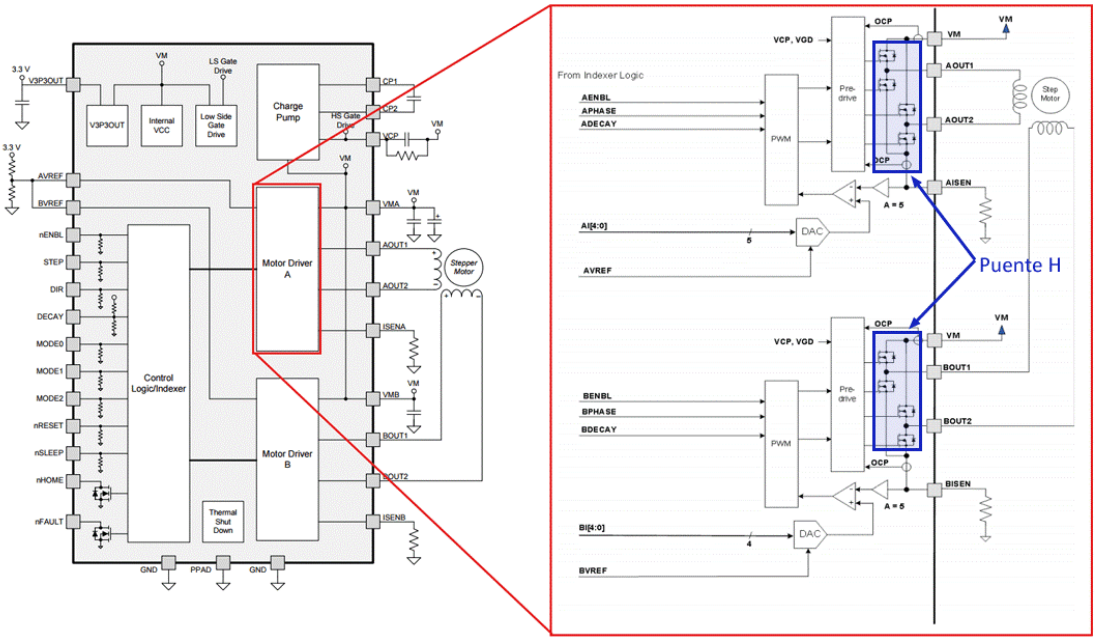


Ilustración 27: Esquema del driver DRV8825

El otro motivo de la complejidad de su electrónica es que incorporan funciones especialmente diseñadas para el control de motores paso a paso, como son el regulador de intensidad y el Microstepping.

Este controlador dispone de reguladores de intensidad incorporado. El motivo es que los motores paso a paso de cierto tamaño y potencia, como por ejemplo los NEMA 17 o NEMA 23, necesitan tensiones superiores a las que podrían soportar las bobinas por su corriente nominal.

El limitador interrumpe la señal proporcionando una señal pulsada PWM de forma que el valor promedio de la intensidad que atraviesa la bobina es la intensidad nominal del motor.

Para regular la intensidad que proporcionara el limitador y ajustarlo al valor del motor que vayamos a emplear la placa disponen de un potenciómetro que regula la intensidad del limitador.

4.2.2 Microstepping

Como hemos dicho, el microstepping es una técnica que permite obtener pasos inferiores al paso nominal del motor paso a paso que vamos a controlar.

Cuando vimos las secuencias típicas de encendido de un paso a paso, vimos que aplicando un control todo a nada a las bobinas teníamos varias posibles combinaciones,

de las cuales vimos las tres más habituales (1-fase, 2-fases, media-fase). Pero nadie ha dicho que tengamos que encender o apagar por completo las bobinas.

El microstepping hace variar la corriente aplicada a cada bobina emulando un valor analógico. Si pusiéramos a ambas bobinas dos señales eléctricas sinusoidales perfectas desfasadas 90º conseguiríamos un campo magnético rotatorio perfecto en el interior del motor.

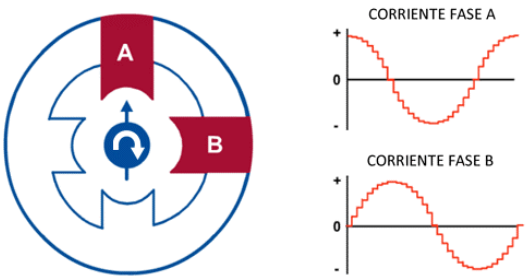


Ilustración 28: Microstepping

Por supuesto el controlador digital no genera valores analógicos perfectos, si no valores discretizados (“a saltos”), por lo que la señal eléctrica que aplica es igualmente una función sinusoidal discretizada como el de la ilustración 28.

El resultado es un campo magnético un campo magnético rotativo con un paso inferior al paso nominal, que depende del número de niveles de discretos que podemos emplear en las señales de excitación de la bobina.

Cuando funcionamos sin microstepping (Modo full step), los controladores aplican una secuencia de 2-fases, por lo que aplican de forma permanente el 71% de la corriente del limitador a cada bobina. Únicamente varían el sentido en el que la corriente circula por la bobina.

Sin embargo, si aplicamos microstepping en cualquier de sus modos de funcionamiento, el controlador llega a aplicar el 100% de la corriente a una de las bobinas en un determinado paso. La cantidad de corriente concreta aplicada a cada bobina varía con cada paso.

La resolución con la que queremos que funcione el controlador se controla aplicando tensión a los Pines M0, M1 y M2. Estos pines están puestos a tierra mediante resistencias de Pull-Up, por lo que si no conectamos nada estarán a Low, y sólo deberemos forzar los pines en High.

En la tabla 11 muestra las conexiones que debemos hacer si queremos obtener el microstepping.

Tabla 11: Obtención del microstepping

Resolución	M0	M1	M0
Full step	Low	Low	Low
1/2 step	High	Low	Low
1/4 step	Low	High	Low
1/8 step	High	High	Low
1/16 step	Low	Low	High
1/32 step	High	Low	High
1/32 step	Low	High	High
1/32 step	High	High	High

4.2.2 Módulo RS485

El RS485 es un estándar de comunicaciones ampliamente empleado en industria que podemos emplear en procesadores como Arduino para leer o escribir en otros dispositivos. Muchos dispositivos sensores y actuadores lo han adoptado como forma de comunicación, siendo frecuente en el ámbito industrial.

Una de las ventajas del RS485 es la larga distancia de transmisión. El alcance depende de la velocidad, siendo posible conseguir 35 Mbps en distancias inferiores a 10 metros, y hasta 100 Kbps en distancias hasta 1200 metros.

El RS485 es un protocolo de capa física según el modelo OSI. Es decir, no pone normas ni restricciones sobre el contenido, forma, o codificación de los mensajes enviados. Por tanto, podemos emplearlo para enviar cualquier tipo de señal, como por ejemplo una sea digital, PWM, puerto serie o bus I2C.

Un bus RS485 dispone de dos conductores denominados A y B (inversora). Para ellos se suele emplear un cable de par trenzado para aumentar la inmunidad al ruido. Es posible acceder hasta 32, 128 o 254 estaciones empleando un único par trenzado.

El protocolo funciona, simplemente, invirtiendo la tensión entre A (no inversora) y B (inversora):

- Cuando A+ y B- se considera estado LOW.
- Cuando A- y B+ se considera estado HIGH.

Con RS485 podemos establecer comunicación simplex, half-duplex y full-duplex. Sin embargo, para la comunicación full-duplex tendremos que establecer dos canales distintos, y disponer de un receptor y emisor en cada uno de los terminales.

El RS485 se emplea frecuentemente en combinación con UARTs, para enviar las señales a largas distancias. También es habitual encontrarlo como capa física en una implantación de protocolo ModBus.

Proyectos con RS485 incluyen automatización de plantas industriales. También se emplea en ámbitos de automoción e incluso en aviones para la conexión de dispositivos. Otros ejemplos de uso incluyen automatización de edificios, monitorización de sistemas

fotovoltaicos, o control grandes sistemas de iluminación o sonido como en conciertos de música.

4.2.2.1 Comunicación con Arduino

La conexión de los módulos con MAX485 es sencilla. En primer lugar, alimentamos el módulo conectando Vcc y Gnd, respectivamente, a 5V y Gnd de Arduino.

Por otro lado, tendremos que conectar los conductores A y B del par trenzado que constituyen el propio bus RS485, y al que estarán conectados el variador de frecuencia.

Ahora, deberemos configurar el módulo como emisor o receptor, para lo cual empleamos los pines RE (receiver enable) y DE (driver enable). Si conectamos estos pines a Vcc el módulo actuará como emisor, y si los conectamos a Gnd como receptor.

Finalmente, tendremos que conectar, respectivamente la entrada de datos al módulo DI (drive input) en el caso de que actué como emisor, o la salida de datos del módulo RO (receiver output) en el caso que actué receptor.

Si queremos que durante la conexión el conversor RS485 pueda cambiar su papel de emisor a receptor (conexión half duplex) simplemente tenemos que conectar los pines RE y DE a una salida digital para poder cambiar su tensión de Gnd a Vcc (ilustración 29).

Para la comunicación con el variador de frecuencia utilizamos la librería ModbusMaster485 (Git-Hub, 2016).

Ilustración 29: Conexión del MAX485 con el Arduino

5 Diseño, Montaje y Programación.

5.2. Ball Mill

El principio básico del molino de bolas, es tener el material en un medio saturado de bolas de acero, de una distribución de medidas heterogénea, así logra una mayor capacidad de reducción.

Hay factores que afectan el producto de molienda y que pueden ser monitorizados y controlados. Estos son dos:

- Velocidad de giros del molino sobre su propio eje. Para esto, se define la velocidad critica, y es la cual por encima de ésta se produce el efecto centrifuga, por lo tanto, la molienda pasa de ser ineficiente a inexistente, y luego, por debajo de la velocidad crítica, no se produce el efecto cascada en el interior del reactor, teniendo como resultado también la nula eficiencia.
- Caudal de sólidos: es uno de los factores que más afectan los productos de molienda, ya que del caudal depende el tiempo real que el material estará dentro del reactor. Obviamente, a mayor tiempo, mayor es la reducción, y viceversa.
- La temperatura también es monitorizada, pero con fines de balance energético, para determinar las pérdidas de calor durante la acción de fragmentación, e implementar posibles medidas de aprovechamiento de la energía disipada.

5.2.1. Temperaturas

Para este proceso es importante las temperaturas tanto de entrada como de salida del material, para ver como evoluciona y poder calcular cuanta energía que se utiliza en el proceso.

Para la obtención de estas temperaturas utilizamos dos sensores de temperatura:

1. El sensor de temperatura MLX90614, para obtener la temperatura a la que entra y sale el material.
2. El sensor de temperatura DS18B20, con el obtenemos la temperatura a la que entra el agua al sistema.

En la ilustración 30 podemos observar la instalación del sensor de temperatura de entrada del material, como vemos está situada a aproximadamente unos 15 cm, para detectar mejor la temperatura del objeto al que se apunta. La temperatura de salida está instalada de la misma manera que la de entrada, pero enfocada a la parte de la salida del material.

Ilustración 30: Montaje del sensor de temperatura de entrada.

Por una parte, tenemos las temperaturas de entrada, con un mismo Arduino capturamos dos valores, la temperatura de entrada del material, que lo obtiene con el sensor MLX90614 y la temperatura de entrada del agua, que se obtiene con el sensor DS18B20. Estos dos valores se obtienen a partir de un solo Arduino. Cada uno de estos sensores necesita una librería especial para poder utilizar estos sensores. La librería para utilizarlos son: OneWire (Stoffregen, 2014) para poder (Burton, 2015) para utilizar el DS18B20 y la librería Adafruit MLX90614 (Adafuit, 2016) para poder utilizar el MLX90614. El código para obtener estos valores está en el anexo

Y por otra parte necesitamos la temperatura de salida, para ello utilizamos el sensor de MLX90614 a la salida del proceso de molienda del material junto con el agua. Este sensor está conectado a un solo Arduino dedicado a la obtención de este valor.

Para obtener este valor utilizamos la librería Adafruit MLX90614 (Adafuit, 2016). El código para este sensor está en el anexo

5.2.2. Velocidad de giro

Para este proceso utilizamos el módulo RS485 para poder establecer comunicación con el variador de frecuencia. Entramos a los registros del variador y cambiamos la frecuencia.

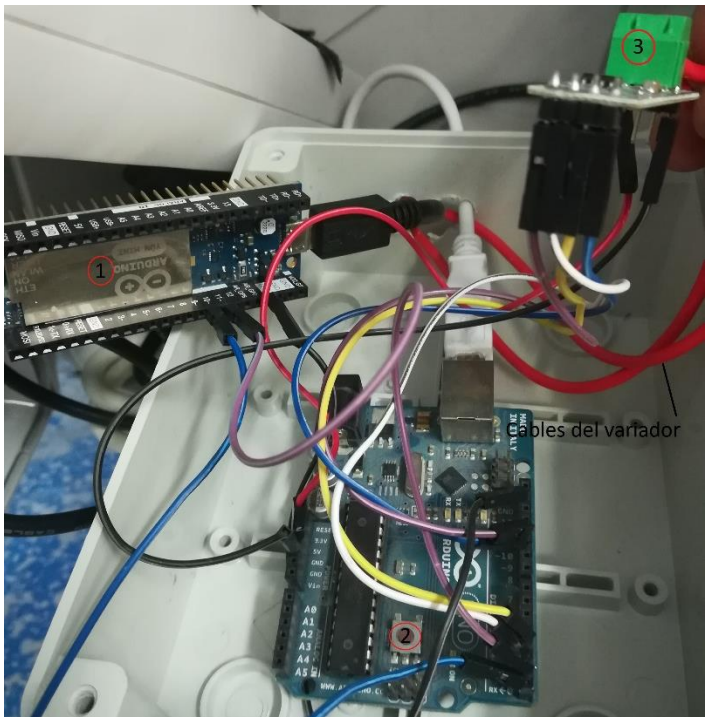


Ilustración 31: Conexión de los Arduinos con el Variador de frecuencia.

Para cambiar la frecuencia y obtener el valor del consumo de energía, utilizamos un Arduino Yun (nº 1 en la ilustración 31) y un Arduino UNO (nº 2 en la ilustración 31) junto con el módulo RS485 (nº 3 en la ilustración 31). El Arduino Yun el que recibirá las peticiones del servidor, se lo comunicará al Arduino uno vía SoftSerial que a su vez lo enviará al variador de frecuencia para que actué. El Arduino recibe la comunicación del Arduino Yun por los pines RX y TX. En la ilustración 33 se ve la conexión que se ha realizado para comunicar con el variador, los cables del variador (2 cables rojos) irán conectados al variador para establecer comunicación con él, en la ilustración 32 vemos los cables que van del módulo RS485 al variador.

Ilustración 32: Conexión del variador con el Arduino

5.2.3. Control liquido entrada

Otro proceso importante en la ball mil es del control del agua, los procesos anteriores captan los valores del sensor sin tener ninguna acción sobre algún actuador. En este caso tenemos que actuar sobre un motor paso a paso.

Para lograr el control del agua entrante en el proceso de trituración utilizamos el sensor de flujo YF-S201, lo utilizaremos en rangos más pequeños que van de 0l/min hasta 5l/min. Calculamos el caudal cada segundo mediante las interrupciones que obtenemos en ese tiempo. Una vez el Arduino reciba la petición de control de caudal, actuara sobre el motor abriendo o cerrando la válvula, ajustando así el caudal que se requiera.

En la ilustración 33 se ve el sistema de engranajes que conecta el motor (nº 1) con la válvula (nº 2). Este sistema permite el paso del agua hacia la ball mil, en la ilustración 34 vemos el sensor de flujo que mide el caudal de entrada a la ball mil, este sensor se encuentra después del sistema de engranajes para obtener un mejor control del parámetro.

Ilustración 33: Sistema de engranajes para mover la válvula

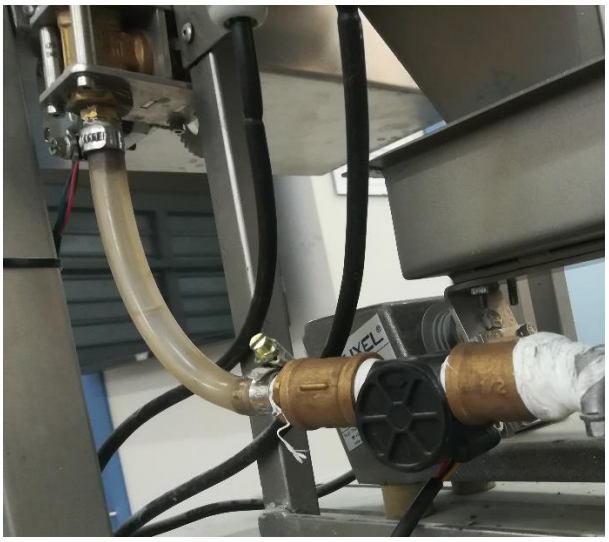


Ilustración 34: sensor de flujo del caudal de entrada a la ball mill

5.3 Shaking Table

Es la parte del proceso en la que nos centraremos, ya que es la parte sobre la cual se necesita tener más control. En este proceso se producirá la "*gravity separation*", este fenómeno es el que más representativo de la planta piloto.

En esta etapa, se recupera el mineral que interesa analizarse, como puede ser el tántalo o el tungsteno, que son los minerales que con los que se ha trabajado en Optimore.

Para poder hacer este proceso de recuperación, se tiene que tener en cuenta como se comentó anteriormente estos tres factores como son:

1. Caudal de entrada, tenemos dos flujos de entrada como se ve en la ilustración 35, el caudal principal, que reparte el agua por toda la mesa; y el caudal de alimentación que reparte el material que viene de otros procesos por toda la mesa.
2. La inclinación de la mesa, es otro factor importante tanto del eje x, estos movimientos de regulación de ángulo de la mesa de sacudidas se hacen gracias a motores paso a paso y un sistema de engranajes.
3. La velocidad de movimiento de la mesa.

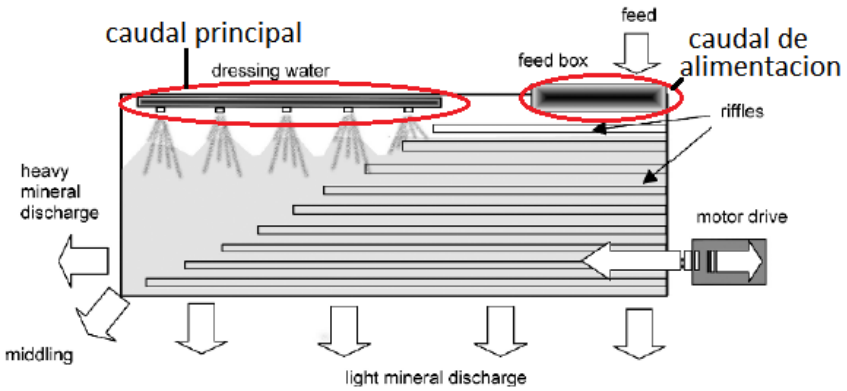


Ilustración 35: Esquema de la shaking table.

5.3.1 Caudales

Como se comentó anteriormente a la mesa de sacudidas le entra dos flujos de agua, estos flujos vienen de un canal central el cual dividimos para las dos entradas.

Para hacer la lectura del caudal de cada flujo utilizamos los sensores YF-S201. Según la lectura de este sensor, el Arduino moverá el motor que está conectado a un sistema de engranajes, el cual abrirá o cerrará la válvula para ajustar el caudal a que se le indicará desde el programa que controla la planta piloto.

Desde el programa se indicará el caudal que se necesita para la mesa de sacudidas tanto de un flujo como del otro, el tendrá que mantener estos parámetros en todo momento.

Ilustración 36: Montaje de los dispositivos de control del flujo.

En la ilustración 36 se ve el sistema está formado por un Arduino Yun que recibe las peticiones desde el ordenador central y se encarga del cálculo de los caudales; y un Arduino UNO, que se encarga del movimiento de los motores paso a paso (Nema 17), esta placa tiene conectada una CNC shield con el DRV8825 que controla los motores. La comunicación entre estos Arduinos se hace por la comunicación serie, en el caso del Arduino Yun envía información al Arduino Uno por el SoftSerial, esta librería permite habilitar dos pines cualesquiera como si fueran pines RX y TX, es una librería de Arduino; utilizamos esta librería porque como comentamos en la descripción del Arduino Yun, los pines RX (pin 0) y TX (pin 1) son utilizados para la comunicación entre el microcontrolador y el microprocesador. El Arduino Uno recibe estos mensajes por los pines RX y TX habituales del Arduino.

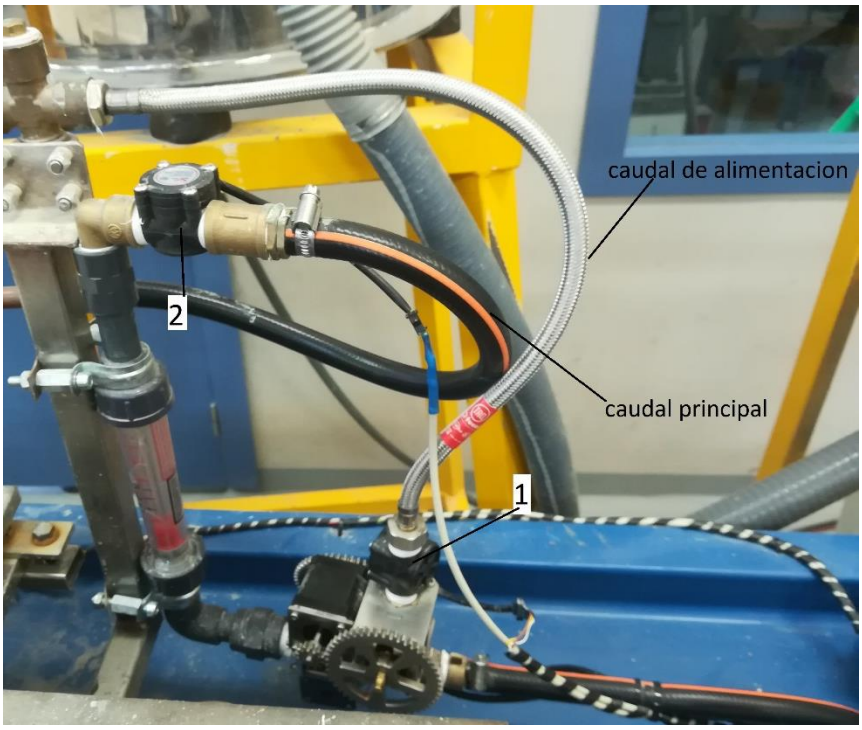


Ilustración 37: Montaje del control del flujo.

El control de estos flujos es constante, es decir, intentara mantener constantemente el flujo que se le indique. Por la mecánica de fluidos al cambiar un flujo de entrada a la shaking table, el otro flujo se verá afectado por lo que tendrá que el Arduino tendrá que regular el flujo y mantenerlo al caudal que se le ha indicado.

En la ilustración 37, podemos ver el montaje que se ha realizado en la shaking table para el control del caudal, el caudalímetro del caudal principal (nº 2 en la ilustración 37), reparte el agua por toda la mesa y el caudalímetro de la alimentación (nº 1 en la ilustración 37) que ayuda a repartir el material por toda la mesa. Debajo del caudalímetro de la alimentación, se encuentran los motores con el sistema de engranaje que harán mover las válvulas que permiten el paso del agua.

5.3.2. Inclinación de la mesa de sacudidas

Este factor es importante para el resultado que se buscaba en Optimore, ya que con el control de este factor se conseguirá una mayor recuperación de residuos y optimizar el proceso.

Para este proceso utilizamos un Arduino Yun con una CNC shield, como se ve en la ilustración 38. Utilizamos un motor para poder realizar el control de la inclinación.

Utilizamos el motor de paso a paso Nema 17 con el DRV8825 ya que no necesitamos tanta fuerza para poder mover el eje y utilizamos un sistema de engranajes con un final de carrera para detectar el tope mínimo, y haciendo varias pruebas, con un DXL360S (un sensor Dual Axis digital) con el que veremos el ángulo en el que se encuentra la mesa.

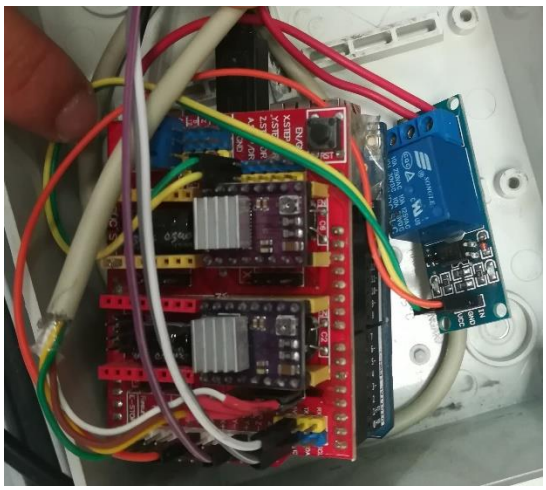


Ilustración 38: Montaje del Arduino con la shield CNC

Este sensor no nos es útil ya que la mesa al estar en movimiento, los valores que obtenemos de este sensor van variando demasiado, y la lectura no sería la correcta. Pero lo utilizamos para medir la variación de la mesa por cada movimiento del motor, con este valor sabemos cuántas vueltas tiene que dar nuestro motor para poder ponerlo a la posición que se desea.

El otro eje de la mesa de sacudidas utilizamos otro motor, ya que los que hemos utilizado hasta el momento no tienen suficiente fuerza como para levantar toda la mesa. En este caso hemos utilizado un motor Nema 34 por las características mencionadas en la descripción de dicho motor.

5.3.2.1 Funcionamiento del control

Este proceso funciona de la siguiente manera:

Para el eje Y utilizamos el motor Nema 17 juntamente con un sistema de engranajes que hace posible el movimiento de la mesa en este eje.

Al encender la máquina, el Arduino indica al motor que vaya hasta posición mínima donde se encuentra un final de carrera, cuando la mesa llega hasta esa posición se para, en la ilustración 39 podemos ver el montaje realizado para poder realizar el control.

A partir de allí, el Arduino pone la mesa automáticamente en la posición horizontal Y=0º, sabemos que es la posición porque se han hecho hechos pruebas anteriores con el sensor DXL360S.

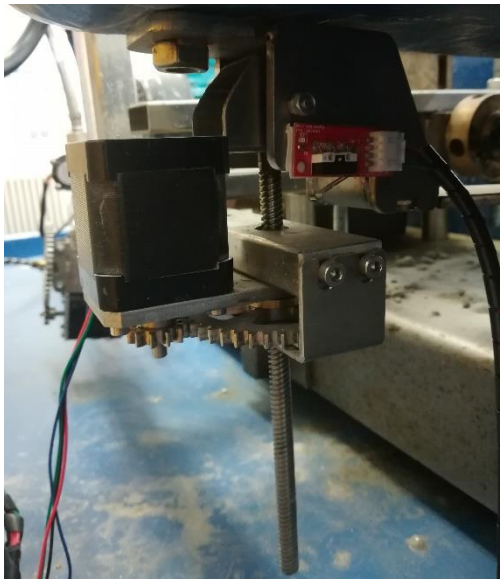


Ilustración 39: Montaje inclinación eje Y

Gracias a las pruebas realizadas con el DXL360S cada vuelta del motor (200 pasos) la mesa incrementa/decrementa su ángulo en 0.5 º gracias a eso sabemos la posición en cada momento de la mesa de sacudidas. Al apretar el botón de Reset del Arduino la mesa volverá a la posición horizontal.

Para el eje X, tenemos el mismo funcionamiento, cada vuelta del motor (200 pasos) la mesa se aumentará o decrementará 0.5º. Este eje por lo general se mantendrá fijo, pero al momento de moverlo se conocerá el ángulo del eje en todo momento, porque guardaremos la última posición que se le enviará desde el ordenador central.

5.3.3 Velocidad de movimiento

Este factor es importante para la recuperación del mineral que nos interesa. Para controlarlo utilizamos el módulo RS485 de la misma manera que hicimos con la ball mil. Ya que tenemos el mismo variador de frecuencia en todas las maquinas que se necesita controlar la velocidad.

5.4. Flotation Cell

Previas a las celdas, se encuentran dos tanques donde se agregan ciertos químicos condicionantes. Estos harán que las partículas de interés sean flotadas, aumentando sus capacidades hidrofóbicas, y las partículas de material estéril sedimentarán, ya que, con la ayuda de los químicos, adquirirán una cualidad hidrofílica. El principio de flotación se basa en generar una cama de burbujas con la ayuda de un agitador y un caudal de aire, que ayudaran a la flotación del material. En la ilustración 40 se observa la estructura previa a las celdas de flotación.

Los aspectos importantes a tener en cuenta en este proceso son:

- Correcta dosificación de reactivos de flotación
- Controlar el caudal de aire dentro de las celdas.
- Obtener información sobre el nivel de espuma creada.

Ilustración 40: Instalación del sensor de pH

5.4.1 Análisis del pH

En la entrada del proceso con el fin de obtener la lectura del pH del líquido entrante al proceso, este líquido junto con el material y químicos determinan el pH del material a analizar.

Para obtener el valor del pH utilizamos el sensor de pH descrito anteriormente, el sensor está situado en el círculo rojo de la ilustración 40 el sensor mide la cantidad de pH del depósito. Por lo que se refiere a la conexión del Arduino con el sensor lo podemos ver en la ilustración 41 conectamos tres cables que van a la shield del sensor de pH.

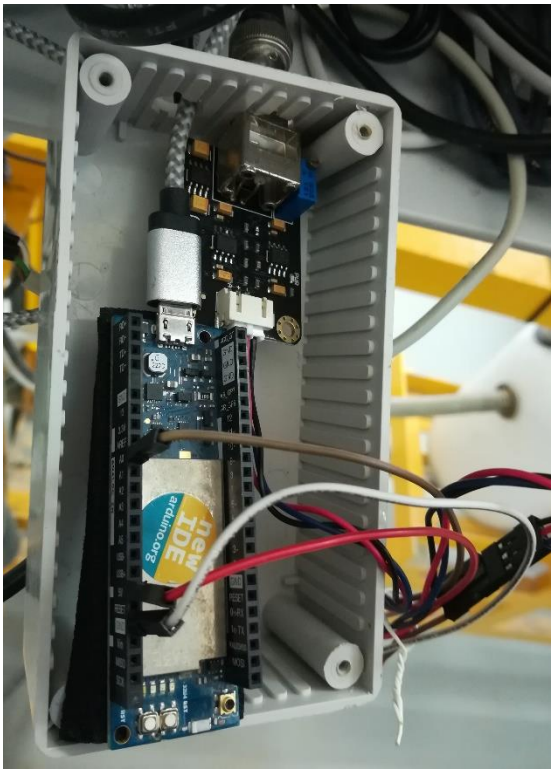


Ilustración 41: Conexión Arduino con el sensor de pH

En la entrada del proceso, tenemos nuestro lector de PH, el cual esta introducido en el primer depósito de la celda de flotación, si se considera que el PH se sales de las métricas, con las bombas, las cuales bombean material químico, el cual hace regular el PH del primer depósito, el segundo deposito también se le introduce productos.

Pasan a las celdas de flotación donde con el aire y los químicos hacen que se forme una espuma, lo que nos interesa es el nivel de espuma en cm, según la lectura de este espesor, modificaremos algunos parámetros de nuestro proceso.

6.4.2 Dosificación de reactivos

En este apartado tenemos 4 depósitos (los cuales se ven en la ilustración 43) con componentes químicos preparados para aumentar las capacidades del material para obtener mejor resultado.

Esta dosificación la conseguimos con bombas peristálticas, las cuales controlamos con un Arduino Yun. Este Arduino controla 4 motores DC alimentados 5V con una fuente externa de 5V y una circuitería hecha para poder controlar los motores como se ve en la ilustración 42, dos bombas van hacia el primer depósito y las otras dos hacia el segundo depósito.

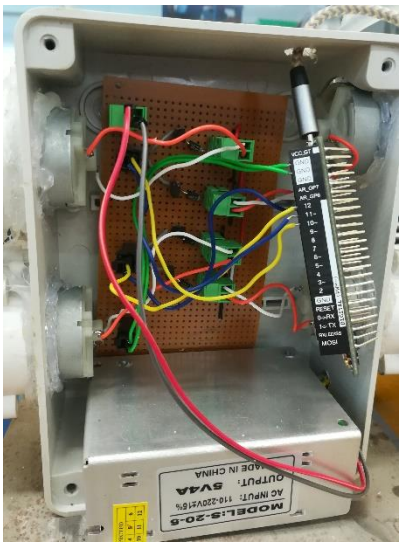


Ilustración 42:Circuito para controlar los motores de dosificación

Para controlar la cantidad bombeada a los depósitos, se hicieron unas pruebas anteriormente para poder inyectar la cantidad exacta al proceso.

Ilustración 43: Dosificación de material químico

Desde el ordenador central indicamos la cantidad de material químico que queremos introducir en cada deposito durante un tiempo determinado, el programa instalado en el Arduino reparte equitativamente el material total indicado en el tiempo fijado.

En la ilustración 45 observamos cómo esta conectados los depósitos con material químico a nuestro dispositivo creado para repartirlo en los depósitos situados en la parte inferior.

5.4.3 Nivel de espuma

Para obtener el nivel de espuma utilizamos el sensor de nivel descrito anteriormente. La máquina tiene 4 celdas, estas celdas se van llenando secuencialmente.

Estos sensores están enganchados en las paredes de las celdas para medir el nivel de espuma que crea el sistema. En cada celda tenemos 3 tres sensores con la finalidad de obtener mayor precisión en la captura de los valores.

5.4.4 control de caudal aire

Para poder realizar el control de este, lo hacemos con el sensor YF-S401 descrito anteriormente.

La entrada de aire viene de un caudal central, este caudal se reparte de manera equivalente hacia las 4 celdas como se ve en la ilustración 44. Por esa razón decidimos calcular la entrada total y dividirla por 4 para obtener el caudal que le llega a cada celda.

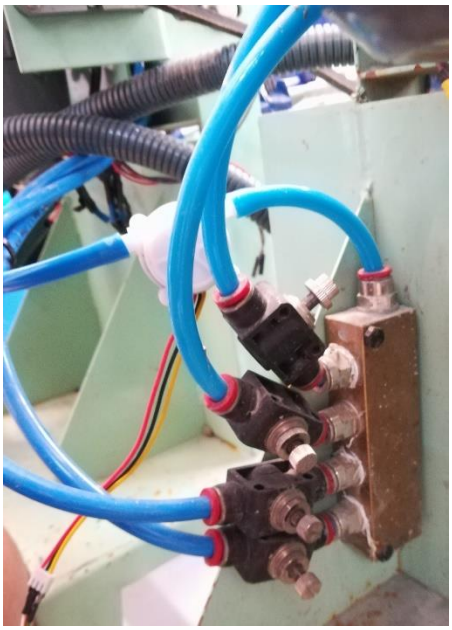


Ilustración 44: Repartición de caudales en las celdas de flotación

Se tiene que tener un control del caudal de aire de entrada, el sensor lee el flujo de aire que entra a las celdas, calcula el caudal que pasa cada segundo. Desde el programa de control se le indica el caudal que se necesita para las celdas, pudiendo ajustarse en cada momento. El Arduino una vez calcula el caudal que circula por el sensor

El programa calcula cada segundo el flujo que circula, y cada 5 segundos el Arduino indica al motor conectado al sistema de engranajes para abrir o cerrar la válvula que habilita el paso del flujo de aire hasta que el caudal del aire se sitúe en torno a un +/- 5% de error con el caudal que se le indica desde el programa de control.

El sistema de cálculo del flujo lo hace mediante un pin, configurado como interrupción externa, que cuenta cuantas interrupciones se han producido en un segundo y aplica la formula detallada anteriormente en la descripción del sensor.

El código utilizado para obtener el valor lo encontramos en el anexo

Ilustración 45: Mecanismo de engranajes para rotar la válvula del sistema de aire

En la ilustración 45 podemos ver el sistema de engranajes hecho para poder abrir o cerrar la válvula del flujo de aire.

5.5. Caja UV

Este proceso es importante, una vez el material ha pasado por los diferentes procesos y se ha extraído el material que nos interesa pasara por este proceso.

Se encarga de analizar el material restante, que contiene la mayor parte del mineral que interesa. Este material pasa a este proceso una vez está seco, ya que los experimentos por lo general se hacen en húmedo. El resultado se obtiene del análisis de fotografías, en el caso del tungsteno y el tántalo, tienen la característica de que al enfocarlo con luz ultra violeta brilla y se puede reconocer fácilmente.

Los elementos que tenemos que controlar son varias para poder realizar todo el proceso completo:

1. Un pequeño vibrador, que utilizaremos para esparcir el material sobre una cinta de manera homogénea. Podemos modular la frecuencia del vibrador, ya que lo conectamos a un pin PWM del Arduino
2. Una luz back light, el cual encendemos y apagamos, gracias a un relé conectado al Arduino.
3. La luz ultravioleta, de igual manera que la luz back light, tenemos 4 luces las cuales se controla cada una con un relé.
4. Un motor paso a paso, el cual hace mover la cinta que transporta el material. La velocidad de este motor es continua y lenta para evitar tirar el material que cae encima de la cinta.

Para la realización de este apartado utilizamos un Arduino Uno. Este dispositivo está conectado a un ordenador el cual tiene un programa instalado para controlar los diferentes elementos. Además, este ordenador tiene conectada una cámara, la cual

hace fotos al material concentrado para analizar y ver el porcentaje de material que existe en una muestra significativa.

6 Entorno de Programación

6.1 SiteWhere

Plataforma que nos permite recibir todos los datos obtenido de los sensores. La mayor parte de la funcionalidad principal proporcionada por las API de SiteWhere se puede acceder de forma externa a través de los servicios REST. Al usar los servicios, una entidad externa puede crear, ver, actualizar o eliminar entidades en el sistema. Los servicios también pueden interactuar con subsistemas, como la gestión de activos. Todas las llamadas REST están sujetas a autenticación y usan el modelo de permisos para verificar que el usuario autenticado está autorizado para la operación.

SiteWhere Server incluye una versión operativa de Swagger que agrega una interfaz de usuario alrededor de los servicios REST. Utilizando la interfaz de Swagger, los usuarios pueden ejecutar de manera interactiva llamadas REST contra una instancia de SiteWhere en ejecución y ver las respuestas de JSON.

En la ilustración 46 podemos ver cómo está estructurada esta plataforma. SiteWhere está compuesta por muchos componentes diferentes que están conectados entre sí, para proporcionar la plataforma central un programa completo.

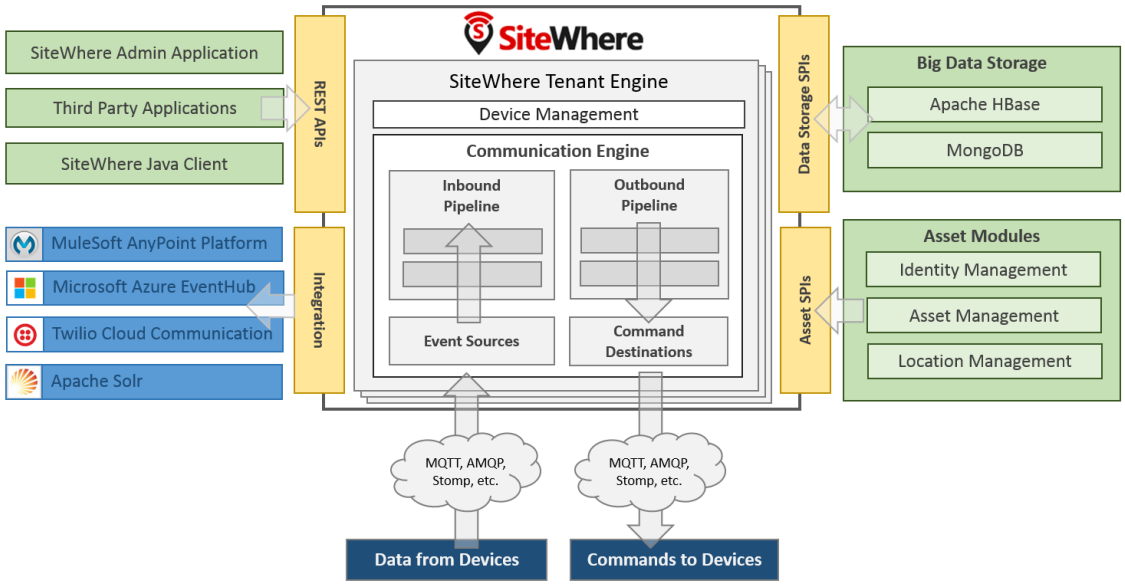


Ilustración 46: Esquema funcionamiento con el mundo exterior SiteWhere

6.1.2. Funcionamiento del SiteWhere

El programa del ordenador central envía peticiones REST a los Arduinos para que estos comiencen a capturar y a poder enviar información a la aplicación. En la petición Rest que se envía a lo Arduinos se envía también la localidad como los “tenants” o los “assigments” y la frecuencia con la que deben enviar los datos al servidor.

Gracias a los tenants y los assigments el SiteWhere sabe el lugar donde guardar el valor recibido desde los end devices.

6.2. Mongo DB

Es una manera de crear una base de datos no relacional, es decir, que cada miembro no hace falta que tenga una estructura determinada, cada información que nos llega puede ser de una forma distinta.

Es una base de datos no relacional orientada a documentos. Esto quiere decir que los datos no se guardan en registros, sino que se guarda en documentos. Estos documentos son almacenados en un BSON (representación binaria de un JSON).

Una de las diferencias más importante en relación a las bases de datos relacionales, es que no es necesario seguir un esquema. Los documentos de una misma colección (concepto similar a una tabla, en una base de datos relacional), puede tener esquemas diferentes.

Se decidió usar MongoDB por ser un proyecto de código abierto orientado a IoT, es decir, no ser una base de datos relacional (NoSQL), sino orientado a documentos estructurados. Eso en este proyecto era clave puesto que el objetivo del proyecto estaba orientado a que diferentes partners aportaran datos de sus plantas, cosa que a su vez ayudaría para la construcción del sistema experto. A continuación, la ilustración 47 muestra las capas de acceso y protocolo como se ve a continuación.

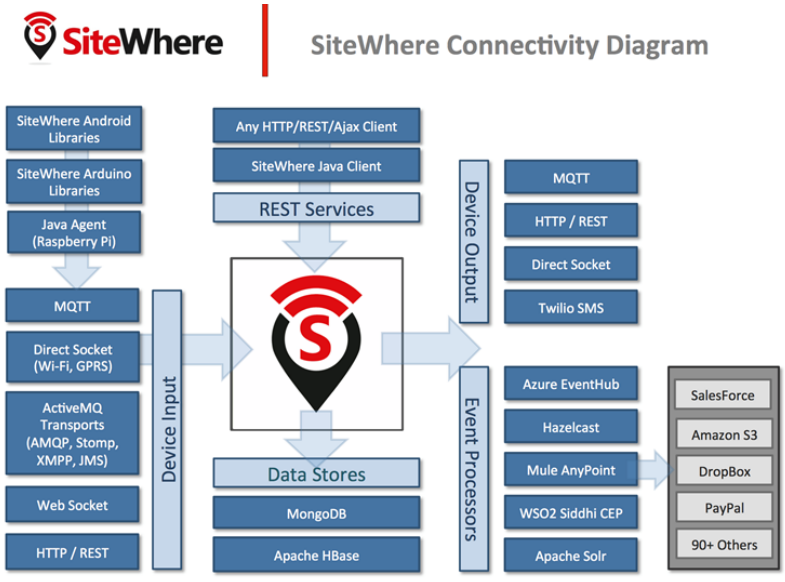


Ilustración 47:Protocolo de acceso y de protocolo por capas.

Al no ser relacional, guardar gran cantidad de datos es más fácil y escalable. Se estudió el paralelismo en esta plataforma y tenía una cola auto gestionada por si llegaban gran cantidad de datos y con el servidor que teníamos y nuestra red, concluimos que era capaz de entrar un máximo de 10 valores cada 100 milisegundos, para la necesidad de del proyecto era más que suficiente.

6.3 Programa de control

Es la interfaz gráfica que permite el control de todos los procesos. Este programa está instalado en el ordenador central.

Consta de tres partes:

1. La parte de registro de experimento. En él se muestran los datos relevantes de cada experimento que se vaya a hacer, como el tipo de material, el porcentaje en volumen de masa, el periodo de tiempo en los que el Arduino enviara los valores al servidor, etc. La ilustración 48 muestra imagen de cómo es el programa el registro de experimento.

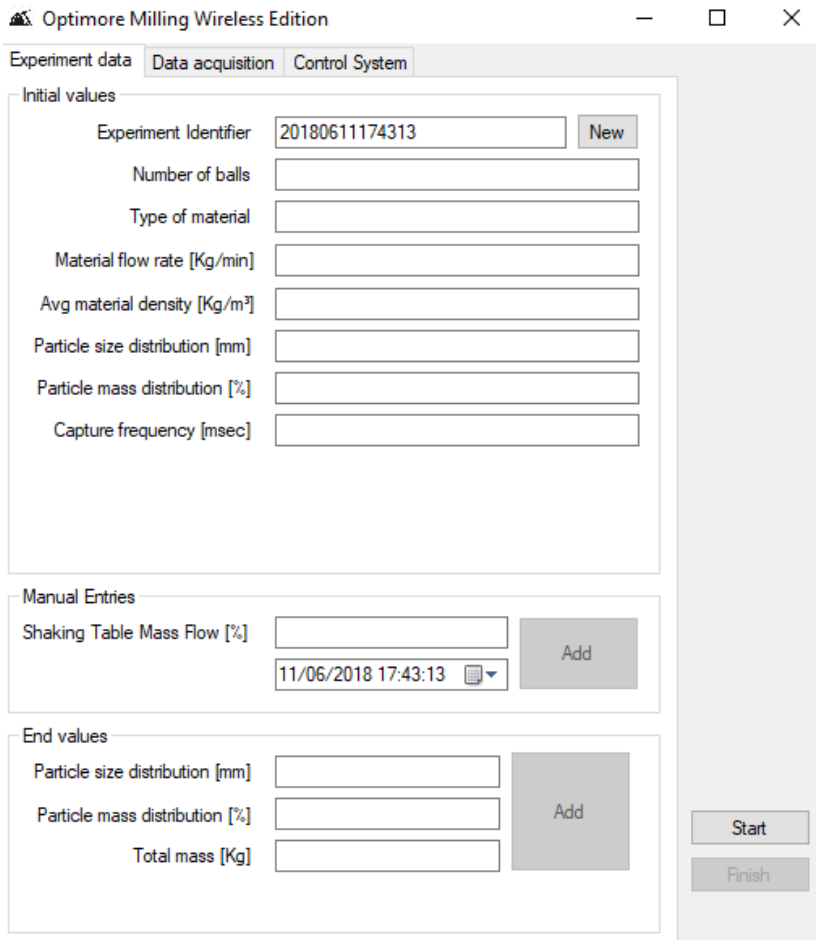


Ilustración 48: Programa de control (registro de Experimento)

2. La parte de adquisicion de datos. La ilustracion 49 muestra la aplicación de escritorio para la adquisiscion de datos. En ella vemos Las de color rojo, son los sensores que no estan activados, y los de color verde son los que estan activos y de los cuales podemos obtener respuesta. En este apartado solo indicamos cuales son los valores que necesitamos captar.

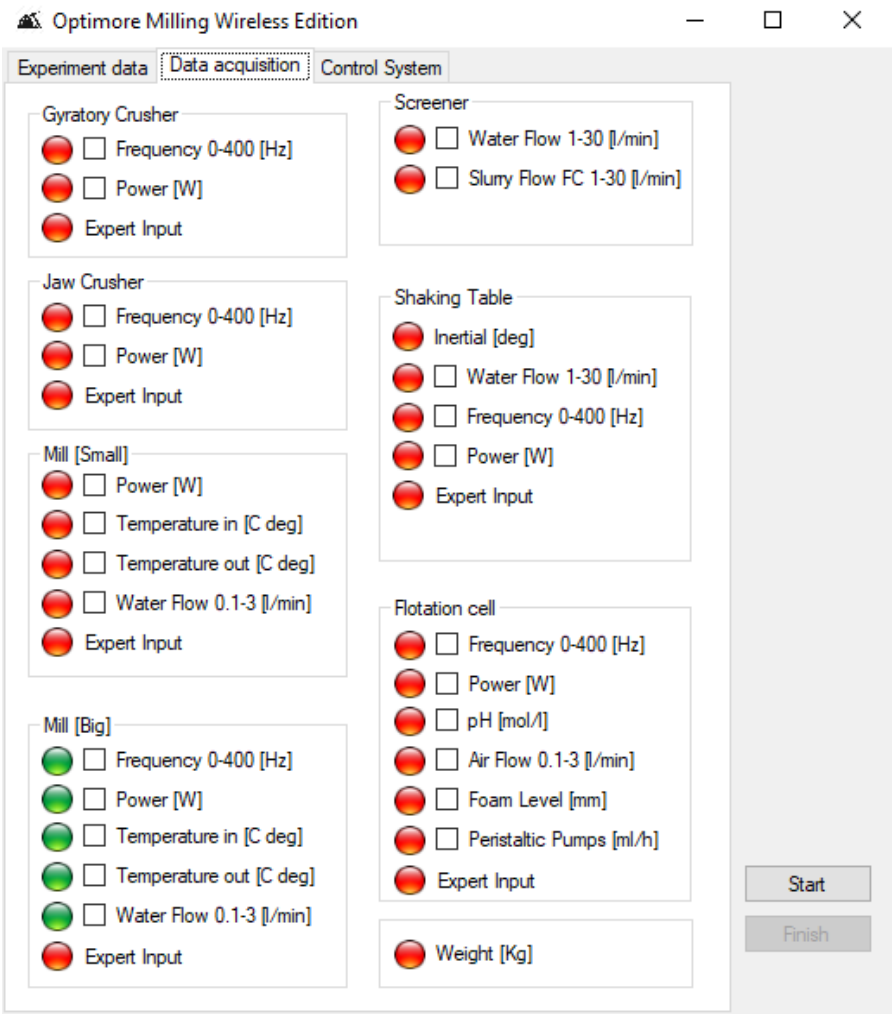


Ilustración 49:Programa de control Adquisición de datos

3. La parte de control se ve reflejada en la ilustración 50, en este apartado se controla los procesos de la planta piloto, en ello se indican los parámetros a los que se quiere ajustar como la frecuencia de los variadores, el flujo de los caudales, ángulo de la mesa de sacudidas, dosificación de químicos, etc.

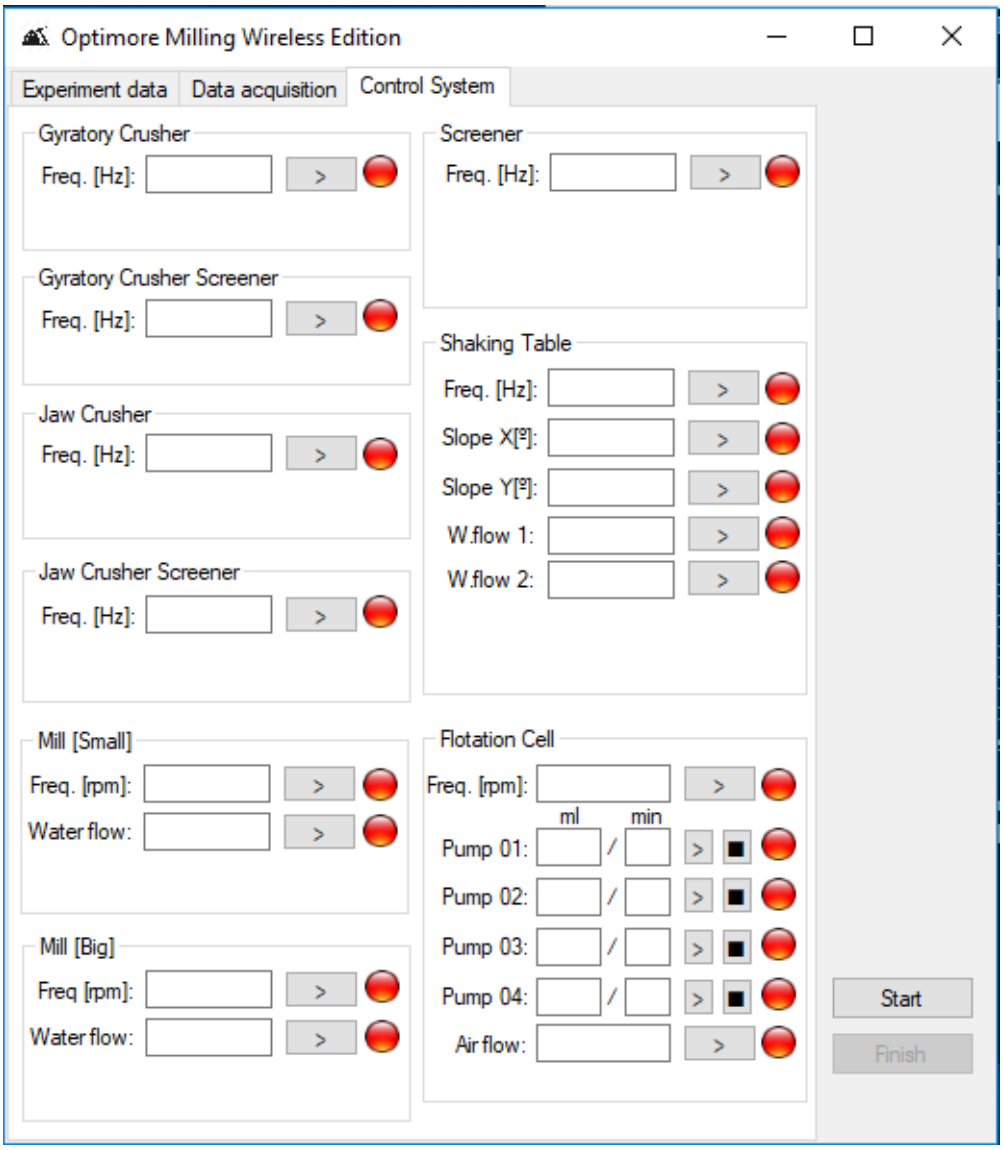


Ilustración 50: Programa de control (Control de procesos)

7 Resultados experimentales

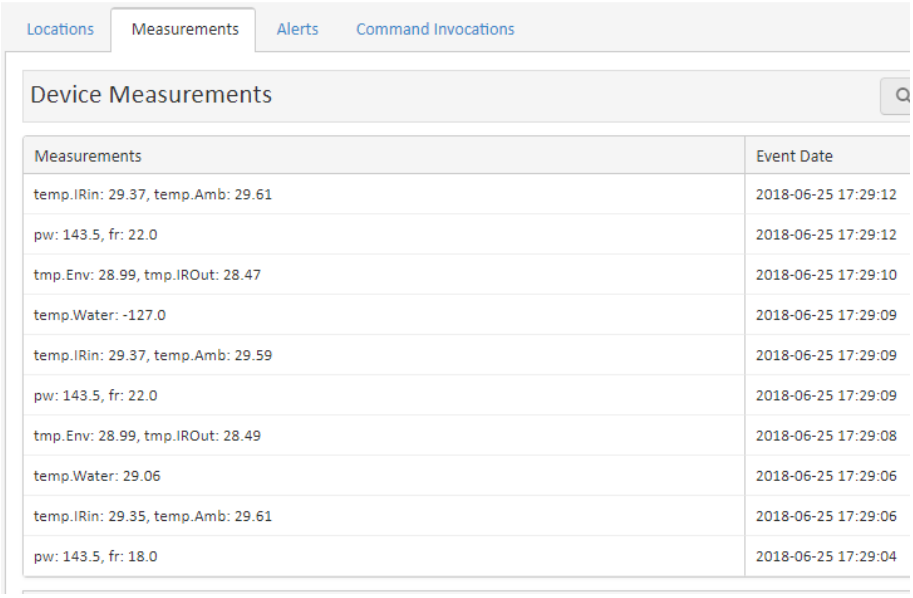
En este apartado veremos los resultados obtenidos de los sensores en los diferentes procesos de la planta piloto del laboratorio de minas, con el fin de poder validar los diferentes resultados obtenidos.

7.1 Ball Mill

Para comprobar el correcto funcionamiento del proceso de la ball mil, hacemos la prueba como muestra la ilustración 51, en la cual ponemos el caudal de entrada a 0 y la frecuencia del variador a 22 rpm.

Ilustración 51: Primera prueba de control ball mill.

En la ilustración 52 podemos ver los resultados obtenidos de la realización de la prueba, podemos ver la frecuencia a la que está el rotor y concuerda con la frecuencia indicada en el apartado de control como indica la ilustración 51.



Measurements	Event Date
temp.IRin: 29.37, temp.Amb: 29.61	2018-06-25 17:29:12
pw: 143.5, fr: 22.0	2018-06-25 17:29:12
tmp.Env: 28.99, tmp.IROut: 28.47	2018-06-25 17:29:10
temp.Water: -127.0	2018-06-25 17:29:09
temp.IRin: 29.37, temp.Amb: 29.59	2018-06-25 17:29:09
pw: 143.5, fr: 22.0	2018-06-25 17:29:09
tmp.Env: 28.99, tmp.IROut: 28.49	2018-06-25 17:29:08
temp.Water: 29.06	2018-06-25 17:29:06
temp.IRin: 29.35, temp.Amb: 29.61	2018-06-25 17:29:06
pw: 143.5, fr: 18.0	2018-06-25 17:29:04

Ilustración 52:Resultados de la primera prueba

La segunda prueba realizada se ha realizado para ver que se puede controlar continuamente los diferentes aspectos de cada proceso. En la ilustración 53 vemos que en el apartado de control hemos realizado cambios, para ver el funcionamiento del control se ve los resultados en la ilustración 54.

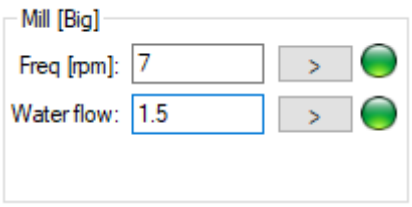


Ilustración 53: Segunda prueba control de Ball Mill

En la ilustración 53 vemos que hemos cambiado la frecuencia hemos pasado de 22 rpm a 7 rpm y el caudal de entrada de 0 l/min a 1.5 l/min, en la ilustración 54 podemos ver que el cambio de se ha realizado correctamente, vemos que el caudal de entrada no es exactamente 1.5 sino que es de 1.33 pero para lo que los que utilizan de la planta ya es suficiente, no es un error muy grande y lo podemos dar por bueno.

Locations | Measurements | Alerts | Command Invocations

Device Measurements

Measurements	Event Date
wf1: 1.33	2018-06-25 17:44:18
pw: 143.6, fr: 7.0	2018-06-25 17:44:18
temp.Water: 25.63	2018-06-25 17:44:18
tmp.Env: 29.13, tmp.IROut: 28.81	2018-06-25 17:44:18
temp.IRin: 29.33, temp.Amb: 29.55	2018-06-25 17:44:18
pw: 143.6, fr: 7.0	2018-06-25 17:44:15
wf1: 1.33	2018-06-25 17:44:15
temp.Water: 25.63	2018-06-25 17:44:15
tmp.Env: 29.13, tmp.IROut: 28.81	2018-06-25 17:44:15
temp.IRin: 29.33, temp.Amb: 29.55	2018-06-25 17:44:15

Ilustración 54: Resultados segunda prueba control Ball Mill

Estas variables como son la frecuencia y el caudal de entrada, son los aspectos a controlar dentro del proceso de la ball mill. En la ilustración 54 también podemos observar otras variables como son la temperatura del agua que entra, la temperatura del material a la entrada (temp. IRin), la temperatura ambiente (temp.Env), y la temperatura del material a la salida(temp.IROut).

7.2 Flotation Cell

La Flotation Cell es otro proceso a controlar, las variables a controlar en este ciclo son: la repartición de material químico en los diferentes depósitos y el control del caudal de entrada de aire que hace posible que se produzca la espuma en los depósitos finales, en los cuales se mide el nivel de espuma.

En la ilustración 55 vemos los parámetros que podemos modificar en este proceso, lo pump 01, 02, 03, 04 corresponde a la repartición de material químico en los depósitos del proceso. El Air flow corresponde al caudal de aire de entrada a las celdas, aquí indicamos el caudal que se quiere que reparta para cada celda.

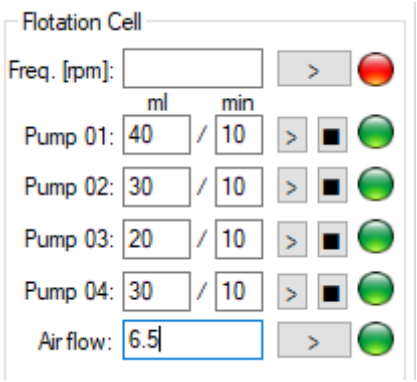


Ilustración 55: Primera prueba de control de la Flotation Cell

En la ilustración 56 vemos los resultados obtenidos de los sensores y de la parte de control de estas variables de la primera prueba realizada. Las variables Af1, Af2, Af3, Af4 corresponden al caudal de aire de que circula por cada una de las 4 celdas; el ChFlow corresponde a la cantidad de material químico que reparte el dosificador en cada minuto, también podemos observar la lectura que realiza el sensor de pH que corresponde a la a variable pH.

En estos resultados podemos ver que el flujo del aire es de 6.6 en comparación con el caudal indicado en el apartado de control 6.5 (ilustración 55), es un error bastante pequeño, alrededor del 1.5 %, teniendo en cuenta que el error del sensor tiene un 10% de error, podemos considerar que se ha producido un buen ajuste en el control del flujo de entrada de aire.

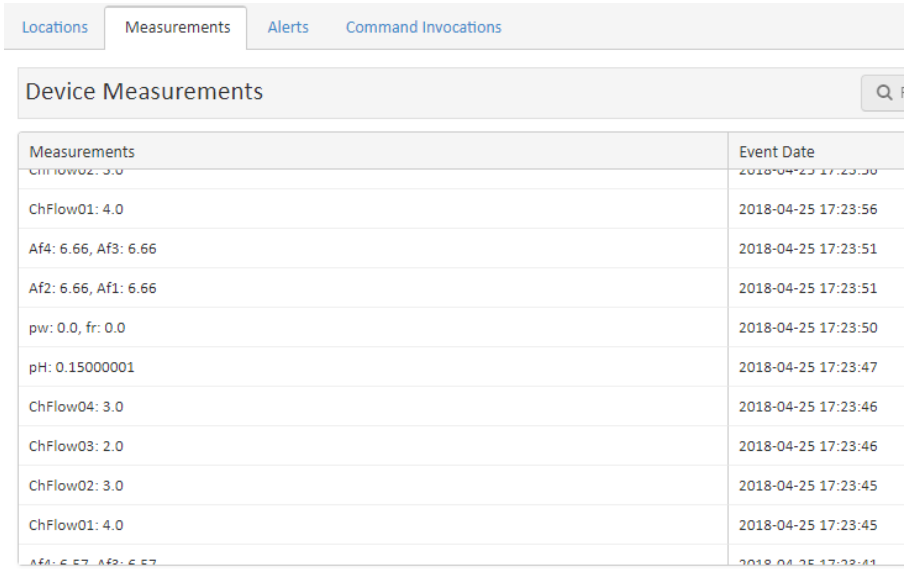


Ilustración 56:Resultados primera prueba de control flotation cell

La segunda prueba realizada en el proceso de la flotation cell lo podemos observar en la ilustración 57, en la que vemos que se cambia la variable del Air flow (flujo de aire) de 6 l/min a 0 l/min sin cambiar la dosificación de las bombas peristálticas.

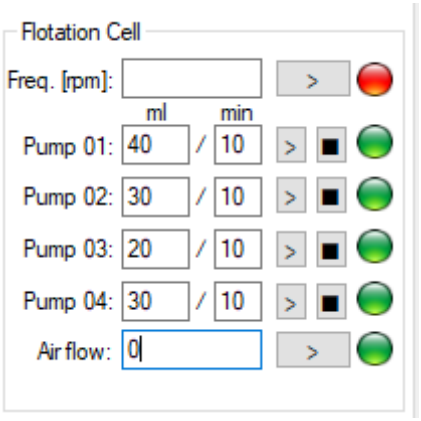


Ilustración 57: Segunda prueba control flotation cell

En la ilustración 58 podemos ver los resultados obtenidos al ejecutar la prueba de la ilustración 59, en estos resultados podemos observar que las variables ChFlow de la dosificación de están a 0, esto es debido a que ya acabo de repartir el material que le indicamos en la prueba anterior, comenzara la dosificación de nuevo cuando le demos al botón ">". También podemos observar que la variable del aire ha cambiado correctamente, en estos resultados también podemos observar como la lectura del pH ha aumentado notablemente, esto es debido a los productos químicos introducidos en el deposito donde se produce la lectura de este factor. La variable fl1, fl2, fl3, fl4 corresponden al nivel de espuma de cada una de las celdas de flotación, en este caso obtenemos que es cero porque la prueba se realizó sin material y no se produjo la espuma.

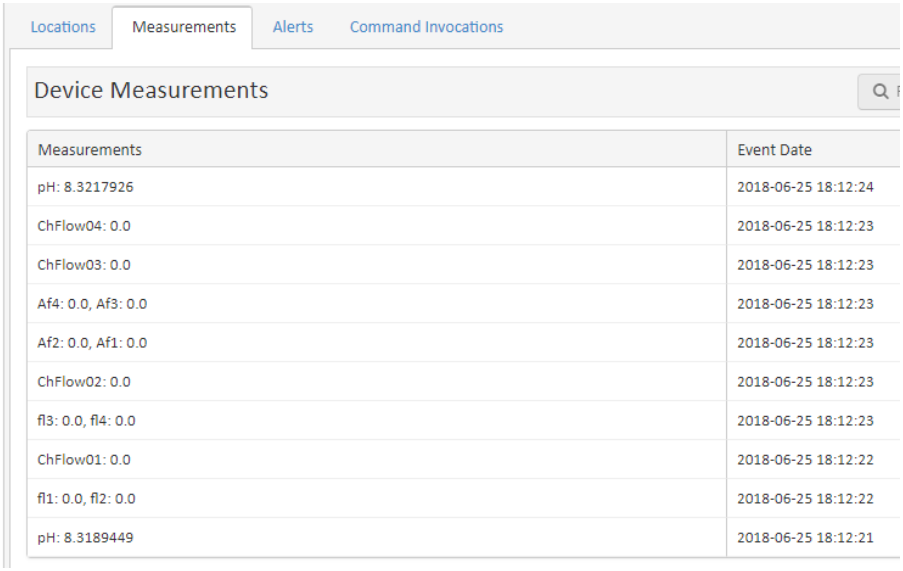


Ilustración 58:Resultados de la segunda prueba de control de la flotation cell

7.3 Shaking Table

La primera prueba de control realizada en la shaking table, la realizamos para comprobar que el control de velocidad de motor en la ball mil funciona de igual manera que este caso. En esta prueba modificamos las variables que se muestran en la ilustración 59, en la cual ponemos la frecuencia de oscilación de la mesa a 60 rpm, esta velocidad es la que se utilizara en casi todos los experimentos realizados, ya que a esta velocidad en la shaking table se dibuja correctamente la curva representativa de la cantidad de material y el flujo de los dos caudales los pondremos a 0, que no circule agua.

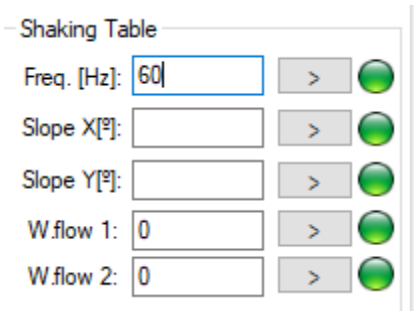


Ilustración 59: Primera prueba de control realizada de la shaking

En la ilustración 60 podemos ver los resultados obtenidos en la primera prueba, como podemos comprobar el cambio se ha ejecutado correctamente, la velocidad que indica los resultados es 60 rpm, igual que la frecuencia de oscilación que indicamos en la ilustración 59. De igual manera que sucede con el flujo de los caudales, se adaptan al flujo que le indicamos en la primera prueba como se ve en la ilustración anterior.

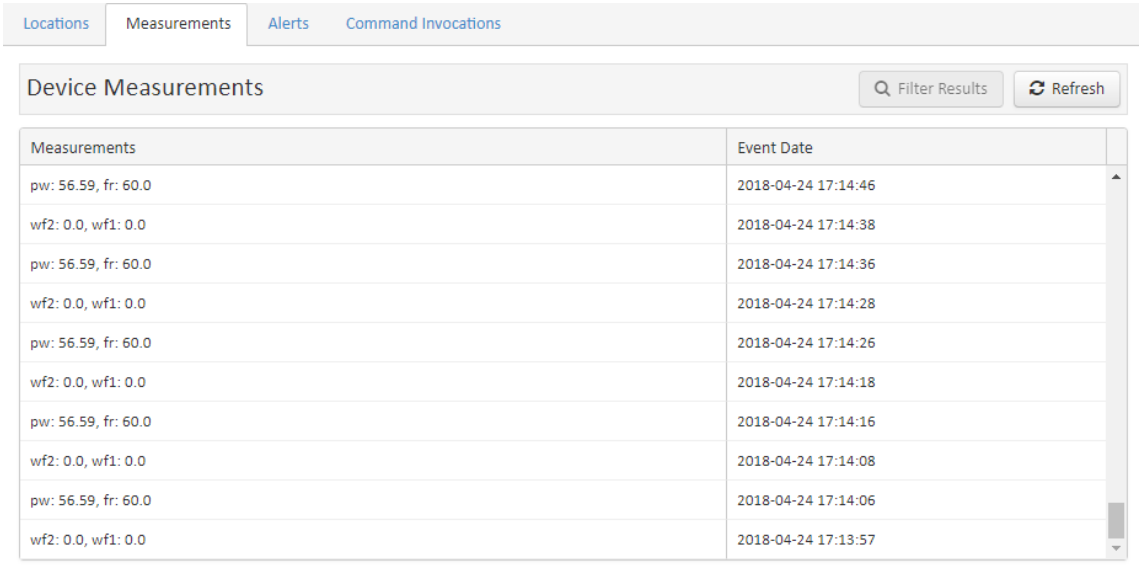


Ilustración 60: Resultados obtenidos primera prueba sobre la shaking table

La segunda prueba de control de las variables de la shaking table la podemos observar en la ilustración 61, en ella ponemos los caudales a 7.5 el caudal principal (wf1) y el caudal que ayuda a repartir el material por encima de la mesa (wf2) lo ponemos a 1.

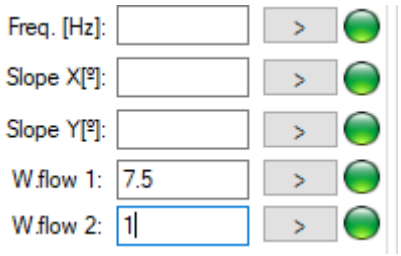


Ilustración 61: Segunda prueba realizada, control de caudal Shaking table

Los resultados de la prueba anterior los podemos observar en la ilustración 62 en ella se ve que el valor que circula por los flujos se ajusta al caudal que le indicamos en la prueba de la ilustración 61.

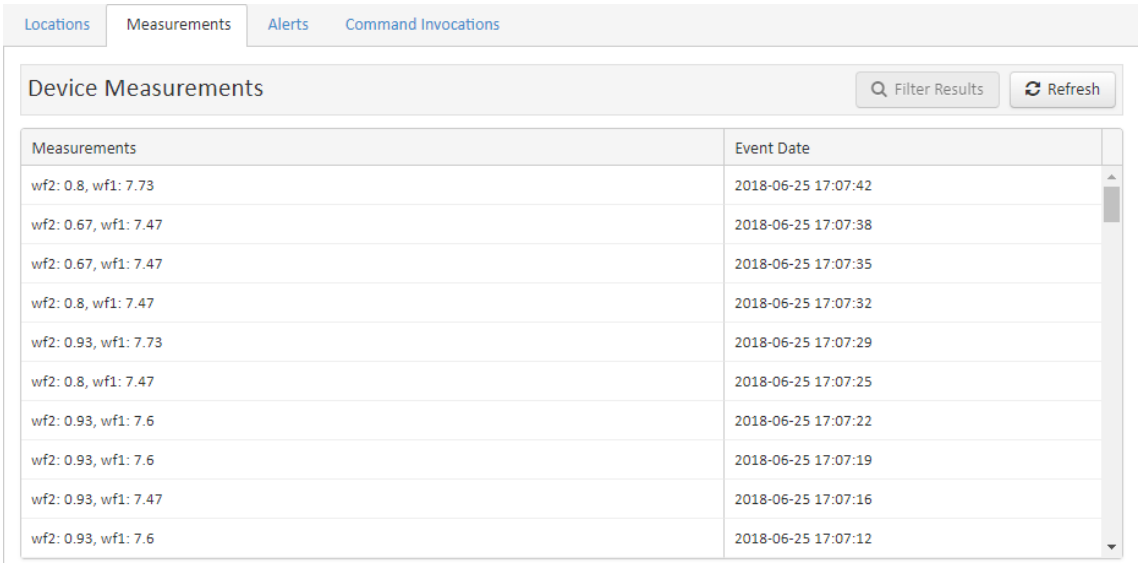


Ilustración 62: Resultados obtenidos en la segunda prueba

En la ilustración 63 realizamos otra prueba de control del caudal que entra en la shaking table, para esta prueba realizamos un pequeño cambio para observar el control del flujo principal, disminuiremos el caudal a 5 l/min.

En la ilustración 64 se observan los resultados obtenidos de pequeño cambio que se hizo en el caudal, en estos resultados se puede observar que el cambio realizado se realiza correctamente en un periodo de tiempo de aproximadamente 5 min. En esta ilustración vemos que hay un pequeño error de precisión en los caudales, este error se produce porque el sensor tiene un error del 10%, el error que hay en los resultados es menor que el que tiene por defecto el sensor, por lo tanto, podemos dar por bueno este ajuste.

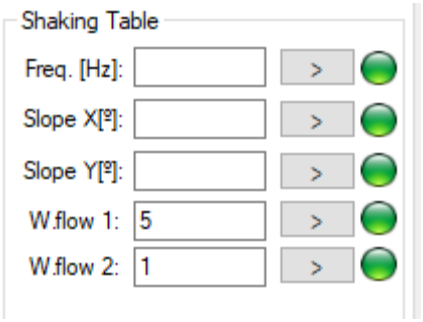


Ilustración 63:Tercera prueba de control shaking table

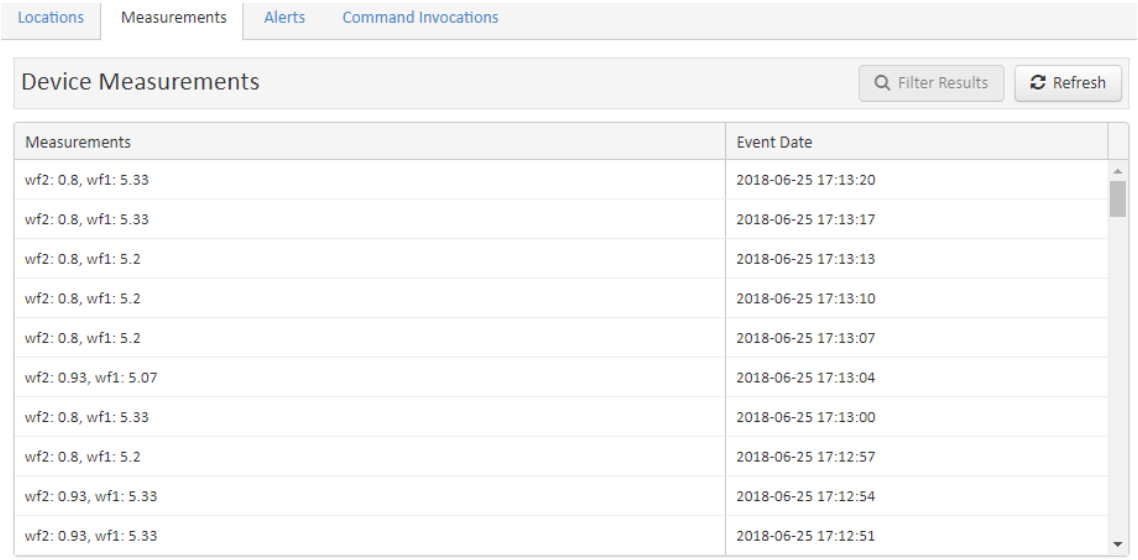


Ilustración 64: Resultados obtenidos de la tercera prueba

La siguiente prueba que realizamos se ve en la ilustración 65. En ella podemos observar que se hacen cambios en ambos flujos, aumentaremos estos flujos para comprobar que el ajuste se hace correctamente. Además, añadimos el control del ángulo de la mesa, en la variable Slope Y indicaremos el ángulo al que queremos que se posicione la mesa de sacudidas, en este ejemplo lo pondremos a 1º.

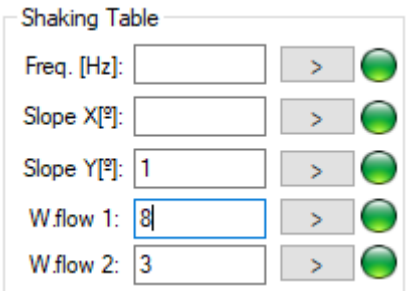


Ilustración 65: Cuarta prueba de control realizada sobre la shaking table.

En la ilustración 66 podemos ver los resultados que se obtienen de la lectura de los sensores, en las respuestas obtenidas podemos observar que el ajuste de los flujos y el de la mesa se realiza en apenas 3 min, un tiempo más reducido que en el de la prueba anterior. También podemos observar la posición a la que hemos situado la mesa de sacudidas, en este caso solo hemos cambiado el eje y, por lo tanto, podemos considerarla como correcta. El error que tenemos en la lectura es menor que el error que tiene por defecto el sensor, y para la precisión que se necesita este ámbito es más que suficiente, por lo tanto, se puede considerar como un ajuste correcto.

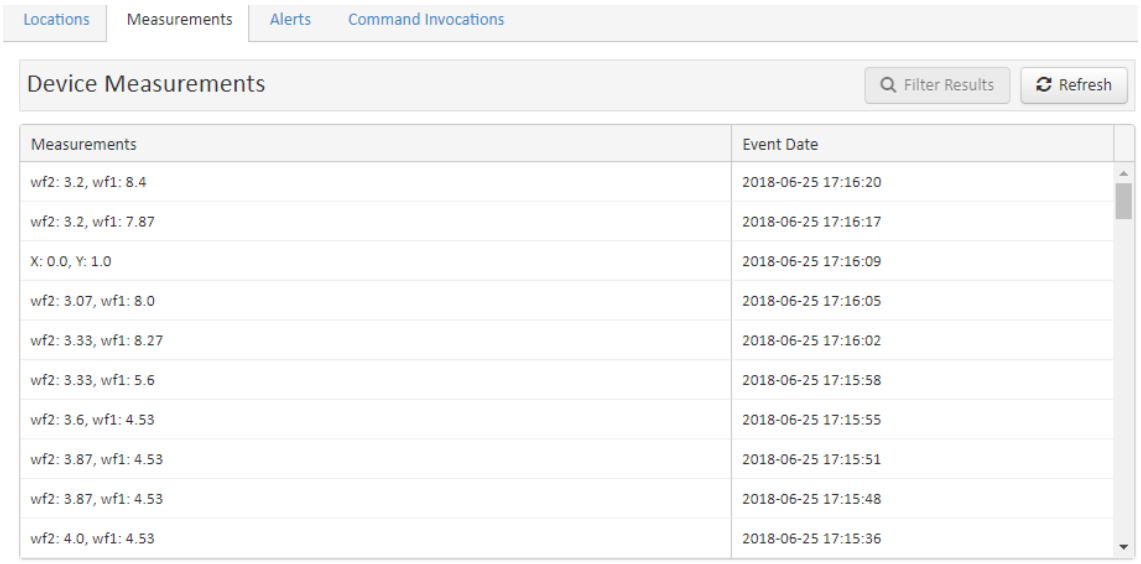


Ilustración 66: Resultados de la cuarta prueba de la shaking table

La última prueba que realizaremos, será cambiar el ángulo de inclinación de la mesa de sacudidas, y disminuiremos el caudal secundario de 3 l/min a 2.5l/min para comprobar que realiza cambios con más precisión. Los cambios realizados los podemos observar en la ilustración 67.

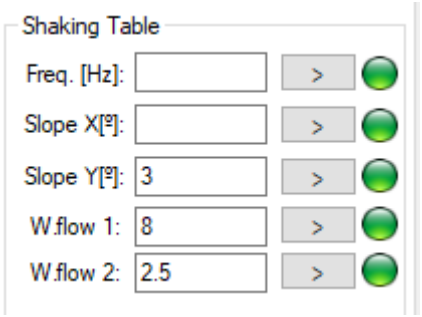


Ilustración 67: Quinta prueba de control realizada sobre la shaking table

En la ilustración 68 podemos ver los resultados obtenidos de los cambios realizados en esta última prueba. Podemos ver como se ha ajustado el caudal del segundo flujo, y el ángulo de inclinación de la mesa de sacudidas como vemos se ha aumentado a la posición que le indicamos en la ilustración 67.

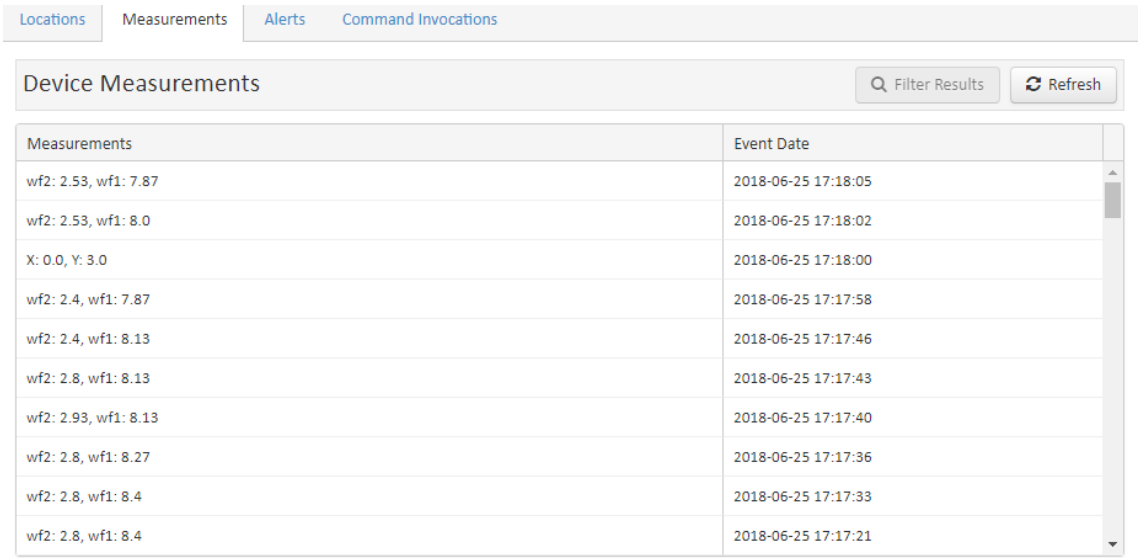


Ilustración 68: Resultados obtenidos de la quinta prueba de control de la shaking table

Estas pruebas realizadas se han hecho para comprobar el control de ciertos parámetros de los diferentes procesos por separado. El día 25 de abril se realizó la prueba final del funcionamiento de toda la planta piloto, es esta presentación estuvieron representantes de los diferentes partners del proyecto Optimore. Para validar el correcto funcionamiento del trabajo realizado, tanto del control como de los resultados obtenidos por parte de todas las partes del proyecto.

8 Conclusiones y trabajos futuros

En este último capítulo, resumiremos la labor realizada a lo largo del trabajo de fin de grado. Numeraremos los objetivos cumplidos, analizaremos las principales aportaciones y los posibles trabajos futuros que puedan mejorar el presente proyecto.

8.1 Objetivos cumplidos

Durante el desarrollo de este proyecto se han estudiado diferentes aspectos para la construcción y puesta a punto de la planta piloto en el laboratorio de minas.

Desde el comienzo del proyecto, fueron necesarios unos requisitos, los cuales debíamos alcanzar para lograr el éxito de este proyecto. Los objetivos logrados han sido:

- ✓ Conocer y estudiar cada proceso de los materiales, para poder obtener la mayor información que interesa.
- ✓ Estudio de la plataforma Arduino Yun para sacar el mayor provecho a la aplicación.
- ✓ Estudio de los diferentes sensores utilizados en cada uno de los procesos.
- ✓ Implantación del hardware en los diferentes procesos y establecer comunicación entre los diferentes dispositivos.
- ✓ Obtención de los valores leídos por los sensores.
- ✓ Preparación y utilización del código del Arduino.
- ✓ Monitorización de los diferentes procesos de la planta piloto.
- ✓ Control de parámetros requeridos de la planta piloto.

Con este desarrollo hemos alcanzado todos los objetivos fijados el inicio de este proyecto. La implantación de del hardware en los procesos resultó la parte más compleja, debido a los pocos conocimientos sobre mecánica o mecánica de fluidos, que hizo retrasar algunas fases de implantación del hardware en los procesos.

Respecto al estudio de los diferentes sensores, nos ha ayudado a sacar un mejor provecho de estos sensores y obtener un mejor resultado de ellos.

La monitorización y en control de todos los sensores se producen correctamente, los Arduinos responden correctamente a las peticiones REST que se realizan desde el programa del ordenador central. Al momento de probar todos los dispositivos, nos dimos cuenta de que no podemos tener más de 16 dispositivos conectados a la red vía wi-fi, la cual cosa limita el número máximo de Arduino Yun que tenemos en la planta piloto.

8.2 Mejoras y trabajos futuros

Una vez realizado este proyecto, en este proyecto describiremos algunos detalles de mejora que se podrían aplicar a este trabajo.

En primer lugar, la primera mejora a realizar seria hacer la conexión de los dispositivos en de las cajas más compacto. Para poder lograrlo se realizaría un diseño de PCB para cada dispositivo, con esta mejora lograríamos sustituir los cables por algo más compacto, evitando así riesgo de desconexión de los cables.

El segundo trabajo sobre el que podríamos tratar, sería poder observar los resultados obtenidos de los sensores instantáneamente dentro del programa, para evitar el tener que observar los resultados dentro de la web.

La ultima mejora que se podría introducir en el proyecto, sería que el servidor fuese capaz de aceptar más de 16 dispositivos conectados a la vez, sin que se produzca ninguna pérdida de valores pudiendo así añadir nuevos dispositivos a controlar o a monitorizar.

9 Referencias

Adafuit. (05 de 2016). *Git-Hub*. Obtenido de https://github.com/adafruit/Adafruit-MLX90614-Library

Arduino. (s.f.). *Datasheet Arduino*. Obtenido de https://www.arduino.cc/en/uploads/Main/YUN-V04(20150114).pdf

Boltzmann, L. (s.f.). *https://es.wikipedia.org/wiki/Ley_de_Stefan-Boltzmann*.

Burton, M. (2015). *Git-Hub*. Obtenido de https://github.com/milesburton/Arduino-Temperature-Control-Library

Datasheet sensor DS18B20. (Enero de 2018). Obtenido de https://cdn.sparkfun.com/datasheets/Sensors/Temp/DS18B20.pdf

DFRobot. (2017). *wiki pH sensor*. Obtenido de https://media.digikey.com/pdf/Data%20Sheets/DFRobot%20PDFs/SEN0161_SEN0169_Web.pdf

Ejemplo, C. (s.f.). *DFRobot*. Obtenido de http://www.dfrobot.com/image/data/SEN0161/phMeterSample.zip

Falconer, A. (2003). Gravity separation: old technique/new methods. *Physical Separation in Science and Engineering 12.1*, 31-48.

Git-Hub. (2016). *ModbusMaster*. Obtenido de https://github.com/4-20ma/ModbusMaster

Integrate, Maxin. (s.f.). *Dataheet DS18B20*. Obtenido de https://datasheets.maximintegrated.com/en/ds/DS18B20.pdf

Optimore. (15 de Enero de 2018). Obtenido de https://www.optimore-news.eu

PaulStoffregen. (03 de 2012). *Git-Hub*. Obtenido de https://github.com/PaulStoffregen/OneWire

SiteWhere. (Marzo de 2018). *Documentacion SiteWhere*. Obtenido de http://documentation.sitewhere.io/architecture.html

SMbus-git. (s.f.). Obtenido de https://github.com/sparkfun/SparkFun_MLX90614_Arduino_Library

Sparfunk. (2018). *Datasheet MLX60164*. Obtenido de https://www.sparkfun.com/datasheets/Sensors/Temperature/MLX90614_rev001.pdf

Stoffregen, P. (2014). *Git-Hub*. Obtenido de https://github.com/PaulStoffregen/OneWire

Wikipedia. (s.f.). *Hall effect sensor*. Obtenido de https://en.wikipedia.org/wiki/Hall_effect_sensor

10 Anexos

En este apartado tenemos los códigos utilizados en los diferentes dispositivos Arduino para la captación de los valores requeridos.

10.1 Ball mil

10.1.1 Temperatura de entrada

```
#include <OneWire.h>
#include <DallasTemperature.h>
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
#include <Wire.h>
#include <Adafruit_MLX90614.h>
#define ONE_WIRE_BUS 9

BridgeServer server;
long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String message;          //message to be sent
String ip,port,assetId,tenantId;
bool captureP;
String command;
Process avviso;          //process to use microprocessor linux in Arduino yun
//Adafruit_MLX90614 mlx;
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
float tempIR, tempAmbient;
void setup(void){
  sensors.begin();
  Bridge.begin();
  Serial.begin(9600); //This is the setup function where the serial port is initialised,
  mlx.begin();
  delay( 1000 );
  captureP=false;
  server.listenOnLocalhost();
  server.begin();
}

void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if(captureP==true){
      captureSensorValue();
  }
  else{
    delay(100); // Poll every 100ms
  }
}

void process(BridgeClient client) {
  command = client.readStringUntil('/');
  if (command == "start") {
    startCommand(client);
    Serial.println("Start");
  }
  if (command == "stop") {
    captureP=false;
    Serial.println("Stop");
  }
  client.print("OK");
}
```

```
void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP=true;
}

void captureSensorValue(){
  start=millis();
  tempAmbient = mlx.readAmbientTempC();    // Aqui queda emmagatzemada la temperatura
Ambient mesurada.
  tempIR = mlx.readObjectTempC();          // Aqui queda emmagatzemada la temperatura de
l'objecte per IR.
  float tempWater = Sensor_Temp_Water();
  saveDB("temp.IRin", String(tempIR,2),"temp.Amb",String(tempAmbient,2));
  saveDB("temp.Water",String(tempWater,2));
  endt=millis();
  timedelayRest=timedelay-(endt-start);
  if(timedelayRest>0){
    Serial.println(message);
    message="";
    delay(timedelayRest);}
}

float Sensor_Temp_Water(void){
  sensors.requestTemperatures(); // Send the command to get temperatures
  return sensors.getTempCByIndex(0);
}

void saveDB(String id,String value){
  //message max 4 values in wireless
  message = "\""+id+"\":"+value;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {"+message+"}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: "+tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");

avviso.addParameter("http://"+ip+":"+port+"/sitewhere/api/assignments/"+assetId+"/measur
ements");
  avviso.run();
}
void saveDB(String id1,String value1,String id2,String value2){
  //message max 4 values in wireless
  message = "\""+id1+"\":"+value1+","+"\""+id2+"\":"+value2;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {"+message+"}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: "+tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");

avviso.addParameter("http://"+ip+":"+port+"/sitewhere/api/assignments/"+assetId+"/measur
ements");
  avviso.run();
}
```

10.1.2 Temperatura de salida.

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
#include <Wire.h>
#include <Adafruit_MLX90614.h>

BridgeServer server;
long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String message;          //message to be sent
String ip, port, assetId, tenantId;
bool captureP;
String command;
Process avviso;          //process to use microprocessor linux in Arduino yun
float tempAmbientFinal;
float tempIRFinal;
Adafruit_MLX90614 mlx = Adafruit_MLX90614();
float tempIR, tempAmbient;

void setup() {
  Bridge.begin();
  Serial.begin(9600); //This is the setup function where the serial port is initialised,
  mlx.begin();
  tempAmbientFinal = 0;
  tempIRFinal = 0;
  delay( 1000 );
  captureP = false;
  server.listenOnLocalhost();
  server.begin();
}

void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if (captureP == true) {
    captureSensorValue();
  }
  else{
    delay(100); // Poll every 100ms
  }
}

void process(BridgeClient client) {
  command = client.readStringUntil('/');
  if (command == "start") {
    startCommand(client);
  }
  if (command == "stop") {
    captureP = false;
  }
  client.print("OK");
}

void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP = true;
}

void captureSensorValue() {
```

```
  start = millis();
  tempAmbient = 100.1;
  tempAmbient = mlx.readAmbientTempC();    // Aqui queda emmagatzemada la temperatura
Ambient mesurada.
  tempIR = 100.1;
  tempIR = mlx.readObjectTempC();          // Aqui queda emmagatzemada la temperatura de
l'objecte per IR.
  tempAmbientFinal = tempAmbient;
  tempIRFinal = tempIR;
  saveDB("tmp.IROut", String(tempIRFinal, 2), "tmp.Env", String(tempAmbientFinal, 2));
  //wait the time of delay
  endt = millis();
  timedelayRest = timedelay - (endt - start);
  if (timedelayRest > 0) {
    delay(timedelayRest);
  }
}

void saveDB(String id1, String value1, String id2, String value2) {
  //message max 4 values in wireless
  message = "\"" + id1 + "\":" + value1 + "," + "\"" + id2 + "\":" + value2;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
}
```

10.1.3 Control de velocidad del variador de frecuencia.

Como hemos comentado anteriormente para que se produzca el control hemos necesitado un Arduino Yun junto con un Arduino Uno con el módulo RS485

10.1.3.1 Codigo del Arduino Uno

```
#include <SoftwareSerial.h>
#include <ModbusMaster.h>
#define MAX485_DE      3
#define MAX485_RE_NEG  2
//instantiate ModbusMaster object
ModbusMaster node;
String modbusString;
const byte rxPin = 10;
const byte txPin = 9;
SoftwareSerial myserial(rxPin, txPin);
bool state = true;
String frequency = "";
String power = "";
String messageSent = "";

struct tram
{
  int  id;
  float freq;
  float ener;
};

struct tram trama;

void sendStructure(byte *structurePointer, int structureLength){
  Serial.write(structurePointer, structureLength);
}
void recieveStructure(byte *structurePointer, int structureLength){
  if (Serial.available() < sizeof(trama)) return;
```

```
  Serial.readBytes(structurePointer, structureLength);
}

void preTransmission(){
  digitalWrite(MAX485_RE_NEG, 1);
  digitalWrite(MAX485_DE, 1);
}

void postTransmission(){
  digitalWrite(MAX485_RE_NEG, 0);
  digitalWrite(MAX485_DE, 0);
}

void setup(){
  pinMode(MAX485_RE_NEG, OUTPUT);
  pinMode(MAX485_DE, OUTPUT);
  // Init in receive mode
  digitalWrite(MAX485_RE_NEG, 0);
  digitalWrite(MAX485_DE, 0);
  // Modbus communication runs at 115200 baud
  Serial.begin(9600, SERIAL_8N1);
  myserial.begin(19200);
  // Modbus slave ID 1
  node.begin(1, myserial);
  // Callbacks allow us to configure the RS485 transceiver correctly
  node.preTransmission(preTransmission);
  node.postTransmission(postTransmission);
}

void loop()
{
  TractamentMsgYUN();
}

void escriu() {
  uint8_t result;
  uint16_t data[6];
  // primer argument es direccio de memoria, segon el numero de registres a llegir
  int writevalue = (trama.freq) * 100;
  result = node.writeSingleRegister(0x0001, writevalue); //escriu al registre 0 el valor
  ////////////////////////ENVIAMENT DE TRAMA
ESCRIU/////////////////////////////////////////////////
  if (result == 0) {
    trama.id = 0; //primer numero=0 perque no hi ha hagut error
    trama.freq = (writevalue / 100.0);
    sendStructure((byte*)&trama, sizeof(trama));
  }
  else {
    trama.id = 1; //primer numero=0 perque  hi ha hagut error
    trama.freq = (writevalue / 100.0);
    sendStructure((byte*)&trama, sizeof(trama));
  }
}

bool esascii(int valor) {
  if ((valor >= 48) && (valor <= 57)) {
    return true;
  }
  else {
    return false;
  }
}

void TractamentMsgYUN() {
  bool pasa = false;
  while (Serial.available()) {
    recieveStructure((byte*)&trama, sizeof(trama));
    pasa = true;
  }

  if (pasa) {
    if (trama.id == 0) {
      sendDataYun();
    }
    else if (trama.id == 1) {
      escriu();
    }
```

```
  }
}

void sendDataYun() {

  messageSent = "";
  frequency = getFrecuency();
  frequency = removeRightZeros(frequency);
  trama.freq = frequency.toFloat();
  power = getPower();
  power = removeRightZeros(power);
  trama.ener = power.toFloat();
  sendStructure((byte*)&trama, sizeof(trama));
  delay(500);
}

String removeRightZeros(String value) {
  String result = value.substring(0, 5);
  return result;
}

String getFrecuency() {
  uint8_t result;
  uint16_t data[6];
  float resultado = 0.0;
  // primer argument es direccio de memoria, segon el numero de registres a llegir
  result = node.readHoldingRegisters(0x0001, 1);
  if (result == node.ku8MBSuccess)  {
    resultado = node.getResponseBuffer(0) / 100.000;
  }
  modbusString = String(resultado, DEC);
  return modbusString;
}

String getPower() {
  uint8_t result;
  uint16_t data[6];
  float resultado = 0.0;
  // primer argument es direccio de memoria, segon el numero de registres a llegir
  result = node.readHoldingRegisters(0x1013, 1);
  //Serial.println("vatios/hora");
  if (result == node.ku8MBSuccess)  {
    resultado = node.getResponseBuffer(0) / 10.000;
  }
  modbusString = String(resultado, DEC);
  return modbusString;
}

void estado() {
  uint8_t result;
  uint16_t data[6];
  // primer argument es direccio de memoria, segon el numero de registres a llegir
  result = node.readHoldingRegisters(0x0003, 1);
  Serial.println("estado");
  if (result == node.ku8MBSuccess)  {
    Serial.println (node.getResponseBuffer(0));
  }
}
```

10.1.3.2 Código Arduino Yun

```
#include <Bridge.h>
#include <Process.h>
#include <YunClient.h>
#include <YunServer.h>
#include <SoftwareSerial.h>

YunServer server;
long start;              //start time operation
long endt;               //end time operation
```

```
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String message;          //message to be sent
String ip, port, assetId, tenantId;
bool captureP;
String command;
Process avviso;          //process to use microprocessor linux in Arduino yun
const byte rxPin = 11;
const byte txPin = 10;
SoftwareSerial myserial(rxPin, txPin);
bool stringcomplete = false;
struct tram {
  int  id;
  float freq;
  float ener;
};
struct tram trama;
void sendStructure(byte *structurePointer, int structureLength){
  myserial.write(structurePointer, structureLength);
}
void recieveStructure(byte *structurePointer, int structureLength){
  if (myserial.available() < sizeof(trama)) return;
  myserial.readBytes(structurePointer, structureLength);
}

void setup() {
  Bridge.begin();
  myserial.begin(9600);
  Serial.begin(9600);
  timedelay = 10000;
  delay( 1000 );
  captureP = false;
  server.listenOnLocalhost();
  server.begin();
}

void loop() {
  YunClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if (captureP == true) {
    captureSensorValue();
  }
  else{
    delay(100); // Poll every 100ms
  }
}

void controla(YunClient client){
  softSerialFlush(myserial);
  trama.id = 1;
  String comand = client.readStringUntil('/');
  if (comand == "frequencia") {
    String freq = client.readStringUntil('/');
    trama.freq = (freq.toFloat());
    Serial.print("LA frequencia que volem posar és: ");
    Serial.println(String(trama.freq));
    delay(50);
    softSerialFlush(myserial);
    sendStructure((byte*)&trama, sizeof(trama));
    delay(200);
    myserial.listen();
    while (myserial.available()) {
      recieveStructure((byte*)&trama, sizeof(trama));
      softSerialFlush(myserial);
    }
  }
  delay(100);
}

void process(YunClient client) {
  // read the command
  command = client.readStringUntil('/');

  // is "start" command?
```

```
  if (command == "start") {
    startCommand(client);
  }
  if (command == "stop") {
    captureP = false;
  }
  if (command == "control") {
    controla(client);
  }
  client.print("OK");
}

void startCommand(YunClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP = true;
}

void captureSensorValue() {
  String valor1, valor2;
  trama.id = 0;
  trama.freq = 0;
  trama.ener = 0;
  sendStructure((byte*)&trama, sizeof(trama));
  start = millis();
  myserial.listen();
  stringcomplete = false;
  delay(100);
  // recibimos String del arduino uno.
  while (myserial.available()) {
    recieveStructure((byte*)&trama, sizeof(trama));
    softSerialFlush(myserial);
    stringcomplete = true;
  }
  //analizamos lo que nos llega del arduino uno
  if (stringcomplete) {
    valor1 = trama.freq;
    valor2 = trama.ener;
  }
  stringcomplete = false;
  //message max 4 values in wireless
  message = "\"fr\":" + valor1 + "," + "\"pw\":" + valor2;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
  endt = millis();
  timedelayRest = timedelay - (endt - start);
  if (timedelayRest > 0) {
    delay(timedelayRest);
  }
}

void softSerialFlush(SoftwareSerial xat_arduino) {
  xat_arduino.listen();
  while (xat_arduino.available()) {
    xat_arduino.read();
  }
```

```
}
```

10.1.4 Control de caudal de entrada a la ball mil

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>

#define tiempo_espera 5000
#define LimiteError 7
#define NumMaxControl 3

#define StepPin 6
#define DirPin 7
#define  RELE 5
#define  en 3

#define max_caudal 5
const int PASOS_PERIODO = 800;u7h


BridgeServer server;

long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
String message;          //message to be sent
String ip, port, assetId, tenantId;
bool captureP;
String command;
Process avviso;          //process to use microprocessor linux in Arduino yun
unsigned long startAdjust;
unsigned long endAdjust;
unsigned int pulseCounter;
int CaudalObjetivo;
float  CaudalActual;
volatile unsigned long NbTopsFan = 0;
const byte hallsensor = 2;

void rpm() {
  NbTopsFan++;
}

void setup() {
  // Bridge startup
  Bridge.begin();
  Serial.begin(9600); //This is the setup function where the serial port is initialise
  captureP = false;
  server.listenOnLocalhost();
  server.begin();
  NbTopsFan = 0;      //Set NbTops to 0 ready for calculations
  pinMode(StepPin, OUTPUT);
  pinMode(DirPin, OUTPUT);
  pinMode(RELE, OUTPUT);
  digitalWrite(RELE, HIGH);
  pinMode(en, OUTPUT);
  start = millis();
  startAdjust = millis();
  CaudalObjetivo = -1;
  pulseCounter = captacionValores();
  CaudalActual = CalculaCaudal(pulseCounter);
}

void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if (captureP == true) {
    captureSensorValue(client);
  }
  pulseCounter = captacionValores();
```

```
  CaudalActual = CalculaCaudal(pulseCounter);
  adjustFlow();
}

void adjustFlow(void) {
  long tiempo;
  int error;
  endAdjust = millis();
  tiempo =  endAdjust - startAdjust;
  if (CaudalObjetivo != -1) {
    if (tiempo > tiempo_espera) {
      if (CaudalObjetivo < 0.5) {
        if (CaudalActual > 0.5) {
            ControlCaudal(CaudalObjetivo);
        }
        error = 0;
      }
      else if (CaudalObjetivo >= 0.5) {
        error = ((abs(CaudalObjetivo - CaudalActual) / CaudalObjetivo) * 100);
      }
      if (error > LimiteError) {
          ControlCaudal(CaudalObjetivo);

      }
      startAdjust = millis();
    }
  }
}

void process(BridgeClient client) {
  command = client.readStringUntil('/');
  if (command == "start") {
    startCommand(client);
  }
  if (command == "stop") {
    captureP = false;
  }
  if (command == "control") {
    controlCommand(client);
  }
  client.print("OK");
}

void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP = true;
}

void controlCommand(BridgeClient client) {
  CaudalObjetivo = client.parseFloat();
}

float CalculaCaudal(int pulsos) {
  if (pulsos > 0 ) {
    CaudalActual = pulsos / 7.5;
  }
  else {
    CaudalActual = 0;
  }
  return CaudalActual;
}
void captureSensorValue(BridgeClient client) {
  long resta;
  endt = millis();
  resta = endt - start;
  if (resta > timedelay) {
    saveDB("wf1", String(CaudalActual, 2));
    start = millis();
```

```
  }
}

int captacionValores(void) {
  int pulsos;
  NbTopsFan = 0;
  attachInterrupt(digitalPinToInterrupt(hallsensor), rpm, RISING);
  delay(1000);
  detachInterrupt(digitalPinToInterrupt(hallsensor));
  pulsos = NbTopsFan;
  return pulsos;
}

void saveDB(String id1, String value1) {
  //message max 4 values in wireless
  message = "\"" + id1 + "\":" + value1;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
}

void ControlCaudal(float caudalObjetivo) {

  int resta;//con los pasos que tenemos que dar pasa obtener el caudal que se requiera
  if (caudalObjetivo <= max_caudal) {
    resta = CaudalMintoPasos(caudalObjetivo) - CaudalMintoPasos(CaudalActual);
  }
  else {
    resta = CaudalMintoPasos(max_caudal) - CaudalMintoPasos(CaudalActual);
  }
  if (resta == 0) {
    Serial.println("es el mismo caudal");
  }
  else if (resta > 0) { //el caudal actual sobrepasa el caudal objetivo, cerramos
valvula
    digitalWrite(DirPin, HIGH);
    move_motor(resta, false);
  }
  else { //si resta<0
    digitalWrite(DirPin, LOW);
    move_motor(abs(resta), true);
  }
}

void move_motor(int pasos , bool sentido) {
  digitalWrite(RELE, LOW);
  delay(200);
  for (int i = 0; i < pasos; i++) {
    digitalWrite(StepPin, HIGH);
    delayMicroseconds(PASOS_PERIODO);
    digitalWrite(StepPin, LOW);
    delayMicroseconds(PASOS_PERIODO);
  }
  delay(100);
  digitalWrite(RELE, HIGH);
}

int CaudalMintoPasos(int caudal) {
  int resultado;
  resultado = caudal / (max_caudal / 100.0);
  return resultado * 4;
}
```

10.2 Flotation cell:

10.2.1 Repartición de los materiales químicos

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
BridgeServer server;
long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String message;          //message to be sent
String ip, port, assetId, tenantId; //input data on start
bool captureP;           //bool to save data on the DB or not
String command;          //incomming base command (start/stop/control)
Process avviso;          //process to use microprocessor linux in Arduino yun

//CONTROL VARIABLES
const int pumpPIN_01 = 10;    //pin conrol pump 01
const int pumpPIN_02 = 11;    //pin conrol pump 02
const int pumpPIN_03 = 12;    //pin conrol pump 03
const int pumpPIN_04 = 13;    //pin conrol pump 04
const long DoseTime = 60000;  //flowing liquid every minute 1m = 60s = 60000ms
String command_pumpID;        //incomming pump id command
//incomming ml(command_ml_0X) per minute(command_duringTime_0X)
String command_ml_01, command_duringTime_01;
String command_ml_02, command_duringTime_02;
String command_ml_03, command_duringTime_03;
String command_ml_04, command_duringTime_04;
//mlPerMinute_0X how many ml must flow per minute, mlAccum_0X how many ml has already
flowed thru the pump
long mlPerMinute_01, mlAccum_01;
long mlPerMinute_02, mlAccum_02;
long mlPerMinute_03, mlAccum_03;
long mlPerMinute_04, mlAccum_04;
//status of the pumps, if is taking place a control action or not
bool controlAction_pump01, controlAction_pump02, controlAction_pump03,
controlAction_pump04;
//temp variable to control the time left to dosify
long doseTimeLeft;
//dosisTime_0X is the period to have enabled the pump to dispose the ml required to
acomplish the total of ml (command_ml_0X) ...
// when the time ended (command_duringTime_0X)
//dose_end_0X and dose_start_0X are both for saving the timestamp and count the time
remaining to start the pump
long dosisTime_01, dose_end_01, dose_start_01;
long dosisTime_02, dose_end_02, dose_start_02;
long dosisTime_03, dose_end_03, dose_start_03;
long dosisTime_04, dose_end_04, dose_start_04;

void setup() {
  // Bridge startup
  Bridge.begin();
  ip = "192.168.7.101"; //local
  port = "8080";
  assetId = "b1944afe-d1b6-4834-8d19-0a55709f1749";
  tenantId = "EDMAiot1nn0v4";
  timedelay = 1000;

  pinMode(pumpPIN_01, OUTPUT);  //define pin as OUTPUT
  pinMode(pumpPIN_02, OUTPUT);  //define pin as OUTPUT
  pinMode(pumpPIN_03, OUTPUT);  //define pin as OUTPUT
  pinMode(pumpPIN_04, OUTPUT);  //define pin as OUTPUT
  //Initialize variables
  command_ml_01 = "0"; command_duringTime_01 = "0";
  command_ml_02 = "0"; command_duringTime_02 = "0";
  command_ml_03 = "0"; command_duringTime_03 = "0";
  command_ml_04 = "0"; command_duringTime_04 = "0";
  mlPerMinute_01 = 0;
  mlPerMinute_02 = 0;
  mlPerMinute_03 = 0;
```

```
  mlPerMinute_04 = 0;
  controlAction_pump01 = false;
  controlAction_pump02 = false;
  controlAction_pump03 = false;
  controlAction_pump04 = false;
  /* Initialize communication. */
  Serial.begin(9600); //This is the setup function where the serial port is initialised,
  delay( 1000 );
  captureP = false;
  //server.listenOnLocalhost();
  server.begin();
}

void loop() {
  // Get clients coming from server
  BridgeClient client = server.accept();
  // Handle client petitions
  if (client) {
    process(client);// Process request
    client.stop();// Close connection and free resources.
  }
  // Handle capture Sensor Values
  if (captureP == true) {
    captureSensorValue();
  }
  // Handle Control Actions
  if (controlAction_pump01 == true) {
    controlPump01();
  }
  if (controlAction_pump02 == true) {
    controlPump02();
  }
  if (controlAction_pump03 == true) {
    controlPump03();
  }
  if (controlAction_pump04 == true) {
    controlPump04();
  }
}

void process(BridgeClient client) {
  // read the command
  command = client.readStringUntil('/');
  // is "start" command
  if (command == "start") {
    startCommand(client);
    Serial.println("Start");
  }
  // is "stop" command
  if (command == "stop") {
    captureP = false;
    Serial.println("Stop");
  }
  // is "control" command
  if (command == "control") {
    controlCommand(client);
    Serial.println("Control");
  }
  // Send feedback to client
  client.print("OK");
}

void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP = true;
}

void controlCommand(BridgeClient client) {
```

```
  command = client.readStringUntil('/');
  if (command == "stop") {
    //  /COMMAND/PUMPID/ml/minutes
    //  /stop/1/
    controlCommandStop(client);
  }
  else if (command == "start") {
    //  /COMMAND/PUMPID/ml/minutes
    //  /start/1/100/20/
    //  pump 1 dispose 100 ml during 20 minutes = 5ml/min = 0.083ml/s
    controlCommandStart(client);
  }
}

void controlCommandStop(BridgeClient client) {
  command_pumpID = client.readStringUntil('/');
  if (command_pumpID == "1") controlAction_pump01 = false;
  if (command_pumpID == "2") controlAction_pump02 = false;
  if (command_pumpID == "3") controlAction_pump03 = false;
  if (command_pumpID == "4") controlAction_pump04 = false;
}

void controlCommandStart(BridgeClient client) {
  command_pumpID = client.readStringUntil('/');
  Serial.println(command_pumpID);
  if (command_pumpID == "1") {
    command_ml_01 = client.readStringUntil('/');
    Serial.println(command_ml_01);
    command_duringTime_01 = client.readStringUntil('/');
    Serial.println(command_duringTime_01);
    mlPerMinute_01 = (command_ml_01.toInt()) / (command_duringTime_01.toInt());
    dosisTime_01 = mlPerMinute_01 * 492;
    controlAction_pump01 = true;
    dose_start_01 = 0;
  }

  if (command_pumpID == "2") {
    command_ml_02 = client.readStringUntil('/');
    Serial.println(command_ml_02);
    command_duringTime_02 = client.readStringUntil('/');
    Serial.println(command_duringTime_02);
    mlPerMinute_02 = (command_ml_02.toInt()) / (command_duringTime_02.toInt());
    dosisTime_02 = mlPerMinute_02 * 545;
    controlAction_pump02 = true;
    dose_start_02 = 0;
  }

  if (command_pumpID == "3") {
    command_ml_03 = client.readStringUntil('/');
    Serial.println(command_ml_03);
    command_duringTime_03 = client.readStringUntil('/');
    Serial.println(command_duringTime_03);
    mlPerMinute_03 = (command_ml_03.toInt()) / (command_duringTime_03.toInt());
    dosisTime_03 = mlPerMinute_03 * 535;
    controlAction_pump03 = true;
    dose_start_03 = 0;
  }

  if (command_pumpID == "4") {
    command_ml_04 = client.readStringUntil('/');
    Serial.println(command_ml_04);
    command_duringTime_04 = client.readStringUntil('/');
    Serial.println(command_duringTime_04);
    mlPerMinute_04 = (command_ml_04.toInt()) / (command_duringTime_04.toInt());
    dosisTime_04 = mlPerMinute_04 * 545;
    controlAction_pump04 = true;
    dose_start_04 = 0;
  }
}

void captureSensorValue() {
  start = millis();
  saveDB("ChFlow01", String(mlPerMinute_01, DEC)); //ml/minute
  saveDB("ChFlow02", String(mlPerMinute_02, DEC)); //ml/minute
  saveDB("ChFlow03", String(mlPerMinute_03, DEC)); //ml/minute
  saveDB("ChFlow04", String(mlPerMinute_04, DEC)); //ml/minute
  //wait the time of delay
```

```
  endt = millis();
  timedelayRest = timedelay - (endt - start);
  //Serial.println(timedelayRest);
  //Serial.println(timedelay);
  if (timedelayRest > 0) {
    delay(timedelayRest);
  }
}

void saveDB(String id, String value) {
  //message max 4 values in wireless
  message = "\"" + id + "\":" + value;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
}

void controlPump01 ()     //This is the function that the interupt calls
{
  dose_end_01 = millis();
  doseTimeLeft = (dose_end_01 - dose_start_01) - DoseTime;
  if (doseTimeLeft >= 0) {
    digitalWrite(pumpPIN_01, HIGH);   // poner el Pin en HIGH
    delay(dosisTime_01);               // esperar un segundo
    digitalWrite(pumpPIN_01, LOW);    // poner el Pin en LOW
    mlAccum_01 = mlAccum_01 + (dosisTime_01 * 0.001); //0.001ml/ms
    if (mlAccum_01 >= (command_ml_01.toInt())) {
      mlPerMinute_01 = 0;
      dosisTime_01 = 0;
      mlAccum_01 = 0;
      controlAction_pump01 = false;
    }
    dose_start_01 = millis();
  }
}

void controlPump02 (){     //This is the function that the interupt calls
  dose_end_02 = millis();
  doseTimeLeft = (dose_end_02 - dose_start_02) - DoseTime;
  if (doseTimeLeft >= 0) {
    digitalWrite(pumpPIN_02, HIGH);   // poner el Pin en HIGH
    delay(dosisTime_02);               // esperar un segundo
    digitalWrite(pumpPIN_02, LOW);    // poner el Pin en LOW
    mlAccum_02 = mlAccum_02 + (dosisTime_02 * 0.001); //0.001ml/ms
    if (mlAccum_02 >= (command_ml_02.toInt())) {
      mlPerMinute_02 = 0;
      dosisTime_02 = 0;
      mlAccum_02 = 0;
      controlAction_pump02 = false;
    }
    dose_start_02 = millis();
  }
}

void controlPump03 (){     //This is the function that the interupt calls
  dose_end_03 = millis();
  doseTimeLeft = (dose_end_03 - dose_start_03) - DoseTime;
  if (doseTimeLeft >= 0) {
    digitalWrite(pumpPIN_03, HIGH);   // poner el Pin en HIGH
    delay(dosisTime_03);               // esperar un segundo
    digitalWrite(pumpPIN_03, LOW);    // poner el Pin en LOW
    mlAccum_03 = mlAccum_03 + (dosisTime_03 * 0.001); //0.001ml/ms
    if (mlAccum_03 >= (command_ml_03.toInt())) {
      mlPerMinute_03 = 0;
      dosisTime_03 = 0;
      mlAccum_03 = 0;
```

```
      controlAction_pump03 = false;
    }
    dose_start_03 = millis();
  }
}

void controlPump04 (){     //This is the function that the interupt calls
  dose_end_04 = millis();
  doseTimeLeft = (dose_end_04 - dose_start_04) - DoseTime;
  if (doseTimeLeft >= 0) {
    digitalWrite(pumpPIN_04, HIGH);   // poner el Pin en HIGH
    delay(dosisTime_04);              // esperar un segundo
    digitalWrite(pumpPIN_04, LOW);    // poner el Pin en LOW
    mlAccum_04 = mlAccum_04 + (dosisTime_04 * 0.001); //0.001ml/ms
    if (mlAccum_04 >= (command_ml_04.toInt())) {
      mlPerMinute_04 = 0;
      dosisTime_04 = 0;
      mlAccum_04 = 0;
      controlAction_pump04 = false;
    }
    dose_start_04 = millis();
  }
}
```

10.2.2 Control del caudal del aire

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
#define StepPin 12//3
#define DirPin 13//6
#define RELE 11 //4
#define en 8
#define max_caudal 42
const int PASOS_RPM = 100;
const int PASOS_PERIODO = 800;
//----------VARAIABLES DE LA PETICION-------------
long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String ip, port, assetId, tenantId, accion;
bool captureP;
String command;
Process avviso;
String message;
BridgeServer server;
struct Values {
  unsigned long pulseCounter1;
};

volatile unsigned long NbTopsFan = 0;
const byte hallsensor1 = 2;    //The pin location of the sensor

void rpm() {
  NbTopsFan++;
}
void setup() {
  // put your setup code here, to run once:
  Bridge.begin();
  Serial.begin(9600); //This is the setup function where the serial port is initialised,
  captureP = false;
  server.listenOnLocalhost();
  server.begin();
  //-----------ACTIVAMOS LASINTERRUPCIONES POR LOS PINES, QUE LLAMAN A LA FUNCION
  attachInterrupt(digitalPinToInterrupt(hallsensor1), rpm, RISING);
  NbTopsFan = 0;      //Set NbTops to 0 ready for calculations
  pinMode (DirPin, OUTPUT);
  pinMode(StepPin, OUTPUT);
  pinMode(RELE, OUTPUT);
  pinMode(en, OUTPUT);
  digitalWrite(RELE, HIGH);
}
```

```
void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if (captureP == true) {
    captureSensorValue(client);
  }
  else {
    delay(1000); // Poll every 100ms
  }
}

void process( BridgeClient client) {
  command = client.readStringUntil('/');
  if (command == "start") {
    startCommand(client);
  }
  if (command == "stop") {
    captureP = false;
  }
  if (command == "control") {
    controlCommand(client);
  }
  client.print("OK");
}
void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP = true;
}

void controlCommand(BridgeClient client) {
  int CaudalObjetivo = client.parseInt();
  ControlCaudal(CaudalObjetivo);
}

float CalculaCaudal(Values sensores) {
  float Calc;
  float caudal = 0;
  float caudal_individual;
  if (sensores.pulseCounter1 > 0 ) {
    caudal=sensores.pulseCounter1/98.0;
  }
  else {
    Calc = 0;
    caudal = 0;
  }
  caudal_individual = caudal / 4;
  return caudal_individual;
}
void captureSensorValue(BridgeClient client) {
  float caudal_individual;
  Values sensores;
  String message = "";
  start = millis();
  sensores = captacionValores();
  caudal_individual = CalculaCaudal(sensores);
  saveDB("wf1", String(caudal_individual, 2), "wf2", String(caudal_individual, 2));
  saveDB("wf3", String(caudal_individual, 2), "wf4", String(caudal_individual, 2));
  Serial.print("caudal_individual:");
  Serial.println(caudal_individual);
  endt = millis();
  timedelayRest = timedelay - (endt - start);
  if (timedelayRest > 0) {
    delay(timedelayRest);
  }
```

```
}

Values captacionValores(void) {
  Values capturas;
  NbTopsFan = 0;
  attachInterrupt(digitalPinToInterrupt(hallsensor1), rpm, RISING);
  delay(1000);
  detachInterrupt(digitalPinToInterrupt(hallsensor1));
  capturas.pulseCounter1 = NbTopsFan;
  NbTopsFan = 0;
  return capturas;
}

void saveDB(String id1, String value1, String id2, String value2) {
  //message max 4 values in wireless
  message = "\"" + id1 + "\":" + value1 + "," + "\"" + id2 + "\":" + value2;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
}

void ControlCaudal(float caudalObjetivo) {
  Values sensores;
  float caudal_actual;
  int resta, pasos;
  sensores = captacionValores();
  caudal_actual = CalculaCaudal(sensores);
  if (caudalObjetivo <= max_caudal) {
    resta = CaudalMintoPasos(caudalObjetivo) - CaudalMintoPasos(caudal_actual);
  }
  else {
    resta = CaudalMintoPasos(max_caudal) - CaudalMintoPasos(caudal_actual);
  }
  if (resta == 0) {
    Serial.println("es el mismo caudal");
  }
  else if (resta > 0) { //el caudal actual sobrepasa el caudal objetivo, cerramos
valvula
    digitalWrite(DirPin, HIGH);
    move_motor(resta, false);
  }
  else { //si resta<0
    digitalWrite(DirPin, LOW);
    move_motor(abs(resta), true);
  }
}

void move_motor(int pasos , bool sentido) {
  digitalWrite(RELE, LOW);
  delay(200);
  for (int i = 0; i < pasos; i++) {
    digitalWrite(StepPin, HIGH);
    delayMicroseconds(PASOS_PERIODO);
    digitalWrite(StepPin, LOW);
    delayMicroseconds(PASOS_PERIODO);
  }
  delay(100);
  digitalWrite(RELE, HIGH);
}

int CaudalMintoPasos(int caudal) {
  int resultado;
  resultado = caudal / (max_caudal / 100.0); //42 litros por minuto el maximo/ 100 pasos
  return resultado;
}
```

10.2.3 Codigo para la obtención de nivel en la flotation cell

En este caso utilizamos dos dispositivos, los cuales se comunican via Serial por un lado tenemos el Arduino Yun y otro el Arduino Uno.

10.2.3.1 Codigo Arduino Yun:

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
#include <SoftwareSerial.h>
#define celda1a A0
#define celda1b A1
#define celda1c A2
#define celda2a A3
#define celda2b A4
#define celda2c A5
long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String message;
String ip, port, assetId, tenantId; //input data on start
Process avviso;          //process to use microprocessor linux in Arduino yun
BridgeServer server;
bool captureP;
String command, accion;
SoftwareSerial mySerial(8, 9); // RX, TX
struct Values {
  float celda1;
  float celda2;
  float celda3;
  float celda4;
};

struct tram {
  bool enviado;
  float mensaje1;
  float mensaje2;
};
struct Values Resultados;
struct tram trama;
bool stringcomplete = false;

void recieveStructure(byte *structurePointer, int structureLength){
  if (mySerial.available() < sizeof(trama)) return;
  mySerial.readBytes(structurePointer, structureLength);
}
void sendStructure(byte * structurePointer, int structureLength){
  mySerial.write(structurePointer, structureLength);
}
void softSerialFlush(SoftwareSerial xat_arduino){
  xat_arduino.listen();
  while (xat_arduino.available()) {
    xat_arduino.read();
  }
}

void enviar(tram trama) {
  trama.enviado = true;
  sendStructure((byte*)&trama, sizeof(trama));
}

void saveDB(String id1, String value1, String id2, String value2) {
  //message max 4 values in wireless
  message = "\"" + id1 + "\":" + value1 + "," + "\"" + id2 + "\":" + value2;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
```

```
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
}

void setup() {
  // put your setup code here, to run once:
  Bridge.begin();
  captureP = false;
  Serial.begin(9600); //This is the setup function where the serial port is initialised
  mySerial.begin(9600);
  timedelay = 5000;
  delay( 1000 );
  server.listenOnLocalhost();
  server.begin();
}

void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if (captureP == true) {
    captureSensorValue(client);
  }
  else {
    delay(100); // Poll every 100ms
  }
}

//-------------PROCESS-------
void process( BridgeClient client) {
  command = client.readStringUntil('/');
  Serial.println(command);
  if (command == "start") {
    startCommand(client);
  }
  if (command == "stop") {
    captureP = false;
  }
  client.print("OK");
}
void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  Serial.println(timedelay);
  captureP = true;
}

void captureSensorValue(BridgeClient client) {
  start = millis();
  float value1, value2, value3, value4, value5, value6;
  float finalvalor1, finalvalor2, finalvalor3, finalvalor4, finalvalor5, finalvalor6;
  float resultCelda1, resultCelda2;
  float distancia;
  Serial.println("nueva prueba");
  value1 = analogRead(celda1a);
  value2 = analogRead(celda1b);
  value4 = analogRead(celda2a);
  value6 = analogRead(celda2c);
  finalvalor1 = medir_distancia(value1);
  finalvalor2 = medir_distancia(value2);
  finalvalor4 = medir_distancia(value4);
```

```
  finalvalor6 = medir_distancia(value6);
  resultCelda1 = CalcularMedia(finalvalor1, finalvalor2,finalvalor2);
  resultCelda2 = CalcularMedia(finalvalor4, finalvalor6,finalvalor6);
  Resultados.celda1 = resultCelda1;
  Resultados.celda2 = resultCelda2;
  //envio la trama para recibir
  enviar(trama);
  trama = recibido();
  if (stringcomplete) {
    Resultados.celda3 = trama.mensaje1;
    Resultados.celda4 = trama.mensaje2;
    stringcomplete = false;
  }
  //wait the time of delay
  saveDB("fl1", String(Resultados.celda1, 2), "fl2", String(Resultados.celda2, 2));
//ml/minute
  saveDB("fl3", String(Resultados.celda3, 2), "fl4", String(Resultados.celda4, 2));
  endt = millis();
  timedelayRest = timedelay - (endt - start);
  //Serial.println(timedelayRest);
  //Serial.println(timedelay);
  if (timedelayRest > 0) {
    Serial.println(timedelayRest);
    Serial.println(Resultados.celda1);
    Serial.println(Resultados.celda2);
    Serial.println(Resultados.celda3);
    Serial.println(Resultados.celda4);
    Serial.println("echo");
    delay(timedelayRest);
  }
}

float medir_distancia(int value) {
  int distancia;
  if (value < 480)
    distancia = 0;
  else if (value>480 && value<=540)
    distancia = 5;
  else if (value>540 && value<=580 )
    distancia = 10;
  else if  (value>580 && value<=610)
    distancia = 15;
  else if (value>610 && value<=680)
    distancia = 20;
  else if (value>680 && value<=710)
    distancia = 25;
  else if (value>710 && value<=740)
    distancia = 30;
  else if (value>740 && value<=760)
    distancia = 35;
  else if (value>480 && value<=530)
    distancia = 40;
  return distancia;
}

tram recibido() {
  while ((mySerial.available()) && (!stringcomplete)) {
    recieveStructure((byte*)&trama, sizeof(trama));
    if (trama.enviado == false) {
    }
    stringcomplete = true;
  }
  return trama;
}

float CalcularMedia(float val1, float val2,float val3) {
  float result;
  result = (val1 + val2 + val3) / 2;
  return result;
}
```

10.2.3.2 Código del Arduino Uno

```
#define celda1a A0
#define celda1b A1
```

```
#define celda1c A2
#define celda2a A3
#define celda2b A4
#define celda2c A5

struct tram {
  bool enviado;
  float mensaje1;
  float mensaje2;
};

struct tram trama;
//SoftwareSerial mySerial(10, 11); // RX, TX
bool stringcomplete = false;
bool envio = false;
String inputString = "";

void sendStructure(byte * structurePointer, int structureLength){
  Serial.write(structurePointer, structureLength);
}

void recieveStructure(byte *structurePointer, int structureLength){
  if (Serial.available() < sizeof(trama)) return;
  Serial.readBytes(structurePointer, structureLength);
}
void enviar(tram mensaje) {
  //Serial.flush();
  sendStructure((byte*)&mensaje, sizeof(mensaje));
  delay(200);
}

void setup() {
  pinMode(13, OUTPUT);
  Serial.begin(9600);
  trama.enviado = false;
  trama.mensaje1 = random(0, 50);
  trama.mensaje2 = random(0, 50);
}

void loop() {
  tram trama;
  while ((Serial.available()) && (!stringcomplete)) {
    recieveStructure((byte*)&trama, sizeof(trama));
    if (trama.enviado == true) {
      trama = captureSensorValue();
      stringcomplete = true;
    }
  }
  if (stringcomplete) {
    digitalWrite(13, HIGH);
    enviar(trama);
    delay(1000);
    digitalWrite(13, LOW);
    stringcomplete = false;
  }
}

tram captureSensorValue() {
  float value1, value2, value3, value4, value5, value6;
  float finalvalor1, finalvalor2, finalvalor3, finalvalor4, finalvalor5, finalvalor6;
  float resultCelda3, resultCelda4;
  value1 = analogRead(celda1a);
  value3 = analogRead(celda1c);
  value4 = analogRead(celda2a);
  value6 = analogRead(celda2c);
  finalvalor1 = medir_distancia(value1);
  finalvalor3 = medir_distancia(value3);
  finalvalor4 = medir_distancia(value4);
  finalvalor6 = medir_distancia(value6);
  resultCelda3 = CalcularMedia(finalvalor1, finalvalor1, finalvalor3);
  resultCelda4 = CalcularMedia(finalvalor4, finalvalor6, finalvalor6);
  trama.mensaje1 = resultCelda3;
  trama.mensaje2 = resultCelda4;
  trama.enviado = false;
  return trama;
}
float medir_distancia(int value) {
```

```
  int distancia;
  if (value < 480)
    distancia = 0;
  else if (value > 480 && value <= 540)
    distancia = 5;
  else if (value > 540 && value <= 580)
    distancia = 10;
  else if  (value > 580 && value <= 610)
    distancia = 15;
  else if (value > 610 && value <= 680)
    distancia = 20;
  else if (value > 680 && value <= 710)
    distancia = 25;
  else if (value > 710 && value <= 740)
    distancia = 30;
  else if (value > 740 && value <= 760)
    distancia = 35;
  else if (value > 480 && value <= 530)
    distancia = 40;
  return distancia;
}

float CalcularMedia(float val1, float val2, float val3) {
  float result;
  result = (val1 + val2 + val3) / 3;
  return result;
}
```

10.3 Shaking table

10.3.1 Inclinacion de la mesa

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
#define StepPinX 2
#define DirPinX 5
#define RELE 12
#define en 8
#define StepPinY 3
#define DirPinY 6
#define EndStopX 9
#define EndStopY 10
long start;               //start time operation
long endt;                //end time operation
long timedelay;           //time of delay at the end of the process
long timedelayRest;       //real time to delay
String message;
String ip, port, assetId, tenantId; //input data on start
Process avviso;           //process to use microprocessor linux in Arduino yun
BridgeServer server;
bool captureP;
String command, accion;
String angulo;
const int PASOS_VUELTA = 200;
const int PASOS_RPM = 100;
const int PASOS_PERIODO = 2000;
float angulo_actualx = 0;
float angulo_anteriorx = 0;
float angulo_a_mover = 0;
float angulo_actualy = 0;
float angulo_anteriory = 0;
bool StopX = true;
bool StopY = true;
int estat = 0;

int angulo_a_vueltas(float angulo) {
  int vueltas;
  int anguloMaximo = 7;
  if (angulo == 0) vueltas = 0;
  else if (angulo > anguloMaximo) vueltas = (anguloMaximo / 0.5);
  else vueltas = (angulo / 0.5);
```

```
  return vueltas;
}

void ir_posicion_inicial() {
  bool StopY = true;
  while (StopY) {
    if (!digitalRead(EndStopY)) {
      StopY = false;
    }
    else {
      digitalWrite(RELE, LOW);
      for (int i = 0; i < PASOS_VUELTA; i++) {
        digitalWrite(StepPinY, HIGH);
        delayMicroseconds(PASOS_PERIODO);
        digitalWrite(StepPinY, LOW);
        delayMicroseconds (PASOS_PERIODO);
      }
      digitalWrite(RELE, HIGH);
    }
  }
  double angleZero = 8.0;
  angulo_anteriory = 0;
  definir_sentido(angleZero);
  move_motor((float)angleZero, StepPinY);
}

void setup() {
  Serial.begin(9600);
  pinMode(DirPinX, OUTPUT);
  pinMode(StepPinX, OUTPUT);
  pinMode(DirPinY, OUTPUT);
  pinMode(StepPinY, OUTPUT);
  pinMode(en, OUTPUT);
  pinMode(RELE, OUTPUT);
  digitalWrite(RELE, HIGH);
  digitalWrite(DirPinX, HIGH);
  digitalWrite(DirPinY, HIGH);
  Bridge.begin();
  captureP = false;
  Serial.begin(9600);
  timedelay = 5000;
  delay( 1000 );
  server.begin();
  ir_posicion_inicial();
}

void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  else {
    delay(100); // Poll every 100ms
  }
}

void process( BridgeClient client) {
  command = client.readStringUntil('/');
  if (command == "control") {
    controlCommand(client);
  }
  client.print("OK");
}

void controlCommand(BridgeClient client) {
  String eje_a_mover = client.readStringUntil('/');
  float angulo_destino = client.parseFloat();
  if (eje_a_mover == "X") {
    estat = 1;
  }
  else if (eje_a_mover == "Y") {
    estat = 2;
  }
  int vueltasADar = angulo_a_vueltas(angulo_destino) - angulo_a_vueltas(angulo_actualy);
  definir_sentido(vueltasADar);
  if (angulo_destino == 0) {
```

```
    ir_posicion_inicial();
    angulo_actualy = 0;
  }
  else {
    move_motor(vueltasADar, StepPinY);
    angulo_actualy = angulo_destino;
  }
}

void definir_sentido(float angulo_destino) {
  if (angulo_destino >= 0) digitalWrite(DirPinY, LOW);
  else digitalWrite(DirPinY, HIGH);
}

void move_motor(float angulo , int stepper_pin) {
  int finalY;
  int vueltas = abs(angulo);
  digitalWrite(RELE, LOW);
  delay(100);
  for (; vueltas > 0; vueltas--) {
    for (int i = 0; i < PASOS_VUELTA; i++) {
      digitalWrite(stepper_pin, HIGH);
      delayMicroseconds (PASOS_PERIODO);
      digitalWrite(stepper_pin, LOW);
      delayMicroseconds(PASOS_PERIODO);
    }
  }
  delay(100);
  digitalWrite(RELE, HIGH);
}

float funcion_calculo(float angulo_a_mover) {
  float angulo_a_mover_r = angulo_a_mover;
  switch (estat) {
    case 1: //eje X
      if (angulo_a_mover_r < angulo_maximox) {
        angulo_actualx = angulo_a_mover_r;
        if (angulo_actualx > angulo_anteriorx) {
          digitalWrite(DirPinX, LOW);
          angulo_a_mover_r = angulo_actualx - angulo_anteriorx;
        }
        else if (angulo_anteriorx > angulo_actualx) {
          digitalWrite(DirPinX, HIGH);
          angulo_a_mover_r = angulo_anteriorx - angulo_actualx;
        }
        move_motor(angulo_a_mover_r, StepPinX);
        angulo_anteriorx = angulo_actualx ;
      }
      estat = 0;
      break;
    case 2: //eje Y
      if (angulo_a_mover_r < angulo_maximoy) {
        angulo_actualy = angulo_a_mover_r;
        if (angulo_actualy > angulo_anteriory) {
          digitalWrite(DirPinY, LOW);
          angulo_a_mover_r = angulo_actualy - angulo_anteriory;
        }
        else if (angulo_anteriory > angulo_actualy) {
          digitalWrite(DirPinY, HIGH);
          angulo_a_mover_r = angulo_anteriory - angulo_actualy;
        }
        move_motor(angulo_a_mover_r, StepPinY);
        angulo_anteriory = angulo_actualy ;
      }
      estat = 0;
      break;
  }
  return angulo_a_mover_r;
}
```

10.3.2 Control caudal de entrada de la shaking table

En este caso utilizamos dos microcontroladores, un Arduino Yun la lectura de las peticiones del ordenador central y el lee los caudalimetros.

10.3.2.1 Código Arduino Yun:

```
#include <Bridge.h>
#include <Process.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
#include <SoftwareSerial.h>
#define tiempo_espera 5000
#define LimiteError 5
#define NumMaxControl1 3
#define NumMaxControl2 3

long start;              //start time operation
long endt;               //end time operation
long timedelay;          //time of delay at the end of the process
long timedelayRest;      //real time to delay
String ip, port, assetId, tenantId, accion;
bool captureP;
String command;
Process avviso;
String message;
BridgeServer server;
SoftwareSerial mySerial(8, 9); // RX, TX
//--variables globales maximas veces ajuste
unsigned long startAdjust;
unsigned long endAdjust;

struct Values {
  unsigned long pulseCounter1;
  unsigned long pulseCounter2;
};
Values sensores;
struct Caudales {
  float caudalimetro1;
  float caudalimetro2;
};
Caudales caudal_actual;
struct tram {
  bool enviado;
  int eje;
  bool abrir_cerrar;
  int  vueltas;
};
struct tram trama;

volatile unsigned long NbTopsFan1 = 0;
volatile unsigned long NbTopsFan2 = 0;
const byte hallsensor1 = 2;
const byte hallsensor2 = 3;

float CaudalObjetivo1 = -1;
float CaudalObjetivo2 = -1;
int eje;

void rpm1() {
  NbTopsFan1++;
}
void rpm2() {
  NbTopsFan2++;
}

void recieveStructure(byte *structurePointer, int structureLength) {
  if (mySerial.available() < sizeof(trama)) return;
  mySerial.readBytes(structurePointer, structureLength);
}
void sendStructure(byte * structurePointer, int structureLength) {
  mySerial.write(structurePointer, structureLength);
}
void softSerialFlush(SoftwareSerial xat_arduino) {
  xat_arduino.listen();
  while (xat_arduino.available()) {
    xat_arduino.read();
  }
}

void enviar(tram trama) {
  trama.enviado = true;
```

```
  sendStructure((byte*)&trama, sizeof(trama));
}

void setup() {
  Bridge.begin();
  mySerial.begin(9600);
  Serial.begin(9600); //This is the setup function where the serial port is initialised,
  captureP = false;
  server.listenOnLocalhost();
  server.begin();
  CaudalObjetivo1 = -1;
  CaudalObjetivo2 = -1;
  startAdjust = millis();
  start = millis();
  sensores = captacionValores();
  caudal_actual = calculaCaudal(sensores);
}

void adjustFlow() {
  //comprobar que el caudalobjetivo diferente de -1;
  long tiempo;
  int error1, error2;
  endAdjust = millis();
  tiempo =  endAdjust - startAdjust;
  if (CaudalObjetivo1 != -1) {
    if (tiempo > tiempo_espera) {
      if (CaudalObjetivo1 < 0.5) {
        if (caudal_actual.caudalimetro1 > 0.5) {
          controlCaudal(CaudalObjetivo1, 1);
        }
        error1 = 0;
      }
      else if (CaudalObjetivo1 >= 0.5) {
        error1 = ((abs(CaudalObjetivo1 - caudal_actual.caudalimetro1) / CaudalObjetivo1)
* 100);
      }
      if (error1 > LimiteError) {
        controlCaudal(CaudalObjetivo1, 1);
      }
      startAdjust = millis();
    }
  }
  if (CaudalObjetivo2 != -1) {
    if (tiempo > tiempo_espera) {
      if (CaudalObjetivo2 < 0.5) {
        if (caudal_actual.caudalimetro2 > 0.5) {
          controlCaudal(CaudalObjetivo2, 2);
        }
        error2 = 0;
      }
      else if (CaudalObjetivo2 >= 0.5) {
        error2 = ((abs(CaudalObjetivo2 - caudal_actual.caudalimetro2) / CaudalObjetivo2)
* 100);
      }
      if (error2 > LimiteError) {
          controlCaudal(CaudalObjetivo2, 2);
      }
      startAdjust = millis();
    }
  }
}

void loop() {
  BridgeClient client = server.accept();
  if (client) {
    process(client);
    client.stop();
  }
  if (captureP == true) {
    captureSensorValue(client);
  }
  sensores = captacionValores();
  caudal_actual = calculaCaudal(sensores);
  adjustFlow();
  Serial.flush();
}
void process( BridgeClient client) {
```

```
  command = client.readStringUntil('/');
  if (command == "start") {
    startCommand(client);
  }
  if (command == "stop") {
    captureP = false;
  }
  if (command == "control") {
    controlCommand(client);
  }
  client.print("OK");
}
void controlCommand(BridgeClient client) {
  eje = (client.readStringUntil('/').toInt());
  if (eje == 1) {
    CaudalObjetivo1 = client.parseFloat();
  }
  else if (eje == 2) {
    CaudalObjetivo2 = client.parseFloat();
  }
}

void startCommand(BridgeClient client) {
  ip = client.readStringUntil('/');
  Serial.println(ip);
  port = client.readStringUntil('/');
  Serial.println(port);
  assetId = client.readStringUntil('/');
  Serial.println(assetId);
  tenantId = client.readStringUntil('/');
  Serial.println(tenantId);
  timedelay = client.parseInt();
  //myserial.println(); //enviamos el tiempo de respuesta
  Serial.println(timedelay);
  captureP = true;
}

Caudales calculaCaudal(Values sensores) {
  float caudal1, caudal2;
  caudal1 = 0.0;
  caudal2 = 0.0;
  if (sensores.pulseCounter1 > 0 ) {
    caudal1 = (sensores.pulseCounter1 / 7.5);
  }
  else {
    caudal1 = 0;
  }
  if (sensores.pulseCounter2 > 0) {
    caudal2 = (sensores.pulseCounter2 / 7.5);
  }
  else {
    caudal2 = 0;
  }
  caudal_actual.caudalimetro1 = caudal1;
  caudal_actual.caudalimetro2 = caudal2;
  return caudal_actual;
}

void captureSensorValue(BridgeClient client) {
  long resta;
  endt = millis();
  resta = endt - start;
  if (resta > timedelay) {
    saveDB("wf1", String(caudal_actual.caudalimetro1, 2), "wf2",
String(caudal_actual.caudalimetro2, 2));
    start = millis();
  }
}
Values captacionValores(void) {
  NbTopsFan1 = 0;
  NbTopsFan2 = 0;
  attachInterrupt(digitalPinToInterrupt(hallsensor1), rpm1, RISING);
  attachInterrupt(digitalPinToInterrupt(hallsensor2), rpm2, RISING);
  delay(1000);
  detachInterrupt(digitalPinToInterrupt(hallsensor1));
  detachInterrupt(digitalPinToInterrupt(hallsensor2));
  sensores.pulseCounter1 = NbTopsFan1;
```

```
  sensores.pulseCounter2 = NbTopsFan2;
  return sensores;
}

void saveDB(String id1, String value1, String id2, String value2) {
  //message max 4 values in wireless
  message = "\"" + id1 + "\":" + value1 + "," + "\"" + id2 + "\":" + value2;
  avviso.begin("curl");
  avviso.addParameter("-k");
  avviso.addParameter("POST");
  avviso.addParameter("--data");
  avviso.addParameter("{\"measurements\": {" + message + "}}");
  avviso.addParameter("--header");
  avviso.addParameter("Authorization: Basic YWRtaW46cGFzc3dvcmQ=");
  avviso.addParameter("--header");
  avviso.addParameter("X-SiteWhere-Tenant: " + tenantId);
  avviso.addParameter("--header");
  avviso.addParameter("Content-Type: application/json");
  avviso.addParameter("http://" + ip + ":" + port + "/sitewhere/api/assignments/" +
assetId + "/measurements");
  avviso.run();
}

void controlCaudal(float caudalObjetivo, int eje) {
  int resta;
  int pasos;
  if (eje == 1) {
    trama.eje = 1;
    resta = LitrosMintoPasos1(caudalObjetivo) -
LitrosMintoPasos1(caudal_actual.caudalimetro1);
    Serial.print("resta:");
    Serial.println(resta);
    if (resta < 400) {
      if (resta > 0) {
        Serial.print("pasos:");
        Serial.println(resta);
        trama.vueltas = resta;
        trama.abrir_cerrar = false ;
      }
      else if (resta < 0) {
        Serial.print("pasos:");
        Serial.println(abs(resta));
        trama.vueltas = abs(resta);
        trama.abrir_cerrar = true;
      }
      enviar(trama);
      Serial.println("enviado");
    }
  }
  else if (eje == 2) {
    trama.eje = 2;
    resta = LitrosMintoPasos2(caudalObjetivo) -
LitrosMintoPasos2(caudal_actual.caudalimetro2);
    Serial.print("resta:");
    Serial.println(resta);
    if (resta > 0) {
      Serial.print("pasos:");
      Serial.println(resta);
      trama.vueltas = resta;
      trama.abrir_cerrar = false;
    }
    else if (resta < 0) {
      Serial.print("pasos:");
      Serial.println(abs(resta));
      trama.vueltas = abs(resta);
      trama.abrir_cerrar = true;
    }
    enviar(trama);
  }
}
int LitrosMintoPasos2(float caudal ) {
  float max_valor_1_quarto_vuelta_2 = 5.85;
  float max_valor_2_quarto_vuelta_2 = 9.15;
  float max_valor_3_quarto_vuelta_2 = 13.1;
  float max_valor_4_quarto_vuelta_2 = 14.85;
  float media_1_quarto_vuelta_2 = 5.59;
  float media_2_quarto_vuelta_2 = 9.162;
```

```
  float media_3_quarto_vuelta_2 = 12.76153846;
  float media_4_quarto_vuelta_2 = 14.46;
  int resultado;
  if (caudal < max_valor_1_quarto_vuelta_2) {
    resultado = map(caudal, 0,  max_valor_1_quarto_vuelta_2, 0,
media_1_quarto_vuelta_2);
    resultado = map(resultado, 0, media_1_quarto_vuelta_2, 0, 200);
  }
  else if (caudal < max_valor_2_quarto_vuelta_2) {
    resultado = map(caudal, max_valor_1_quarto_vuelta_2,  max_valor_2_quarto_vuelta_2,
media_1_quarto_vuelta_2, media_2_quarto_vuelta_2);
    resultado = map(resultado, media_1_quarto_vuelta_2, media_2_quarto_vuelta_2, 200 ,
400);
  }
  else if (caudal < max_valor_3_quarto_vuelta_2) {
    resultado = map(caudal, max_valor_2_quarto_vuelta_2, max_valor_3_quarto_vuelta_2,
media_2_quarto_vuelta_2, media_3_quarto_vuelta_2);
    resultado = map(resultado, media_2_quarto_vuelta_2, media_3_quarto_vuelta_2,  400,
600);
  }
  else if (caudal < max_valor_4_quarto_vuelta_2) {
    resultado = map(caudal, max_valor_3_quarto_vuelta_2, max_valor_4_quarto_vuelta_2,
media_3_quarto_vuelta_2, media_4_quarto_vuelta_2);
    resultado = map(resultado, media_3_quarto_vuelta_2, media_4_quarto_vuelta_2,  600,
800);
  }
  else {
    resultado = caudal - media_4_quarto_vuelta_2;
    resultado = resultado / 0.0316;
    resultado = resultado + 800;
  }
  return resultado;
}

int LitrosMintoPasos1(float caudal) {
  float max_valor_1_quarto_vuelta_1 = 4.65;
  float max_valor_2_quarto_vuelta_1 = 12.15;
  float max_valor_3_quarto_vuelta_1 = 17.85;
  float max_valor_4_quarto_vuelta_1 = 21.15;
  float media_1_quarto_vuelta_1 = 4.527272727;
  float media_2_quarto_vuelta_1 = 12.16666667;
  float media_3_quarto_vuelta_1 = 17.64375;
  float media_4_quarto_vuelta_1 = 21;
  int resultado;
  if (caudal < max_valor_1_quarto_vuelta_1) {
    resultado = map(caudal, 0,  max_valor_1_quarto_vuelta_1, 0,
media_1_quarto_vuelta_1);
    resultado = map(resultado, 0, media_1_quarto_vuelta_1, 0, 200);
  }
  else if (caudal < max_valor_2_quarto_vuelta_1) {
    resultado = map(caudal, max_valor_1_quarto_vuelta_1,  max_valor_2_quarto_vuelta_1,
media_1_quarto_vuelta_1, media_2_quarto_vuelta_1);
    resultado = map(resultado, media_1_quarto_vuelta_1, media_2_quarto_vuelta_1,  200,
400);
  }
  else if (caudal < max_valor_3_quarto_vuelta_1) {
    resultado = map(caudal, max_valor_2_quarto_vuelta_1, max_valor_3_quarto_vuelta_1,
media_2_quarto_vuelta_1, media_3_quarto_vuelta_1);
    resultado = map(resultado, media_2_quarto_vuelta_1, media_3_quarto_vuelta_1,  400,
600);
  }
  else if (caudal < max_valor_4_quarto_vuelta_1) {
    resultado = map(caudal, max_valor_3_quarto_vuelta_1, max_valor_4_quarto_vuelta_1,
media_3_quarto_vuelta_1, media_4_quarto_vuelta_1);
    resultado = map(resultado, media_3_quarto_vuelta_1, media_4_quarto_vuelta_1,  600,
800);
  }
  else {
    resultado = caudal - media_4_quarto_vuelta_1;
    resultado = resultado / 0.0316;
    resultado = resultado + 800;
  }
  return resultado;
}
```

10.3.2.2 Código Arduino Uno

```
#define StepPinX 2
#define DirPinX 5
#define  RELE 12
#define  en 8
#define StepPinY 3
#define DirPinY 6
const int PASOS_PERIODO = 1000;
int DarVueltas;
struct tram {
  bool enviado;
  int eje;
  bool abrir_cerrar;
  int vueltas;
};
struct tram trama;

void sendStructure(byte * structurePointer, int structureLength) {
  Serial.write(structurePointer, structureLength);
}

void recieveStructure(byte *structurePointer, int structureLength) {
  if (Serial.available() < sizeof(trama)) return;
  Serial.readBytes(structurePointer, structureLength);
}

void enviar(tram mensaje) {
  sendStructure((byte*)&mensaje, sizeof(mensaje));
}
void setup() {
  Serial.begin(9600);
  pinMode(DirPinX, OUTPUT);
  pinMode(StepPinX, OUTPUT);
  pinMode(DirPinY, OUTPUT);
  pinMode(StepPinY, OUTPUT);
  pinMode(en, OUTPUT);
  pinMode(RELE, OUTPUT);
  digitalWrite(RELE, HIGH);
}

void loop() {
  tram trama;
  while ((Serial.available())) {
    recieveStructure((byte*)&trama, sizeof(trama));
    if (trama.enviado == true) {
      trama.enviado=false;
      if (trama.eje == 1) {
        DarVueltas = trama.vueltas;
        ejecutar(DarVueltas, trama.eje, trama.abrir_cerrar);
      }
      else if (trama.eje == 2) {
        DarVueltas = trama.vueltas;
        ejecutar(DarVueltas, trama.eje, trama.abrir_cerrar);
      }
      break;
    }
  }
}
void move_motor(int pasos, int pin) {
  digitalWrite(RELE, LOW);
  for (int i = 0; i < pasos ; i++) {
    digitalWrite(pin, HIGH);
    delayMicroseconds(PASOS_PERIODO);
    digitalWrite(pin, LOW);
    delayMicroseconds(PASOS_PERIODO);
  }
  digitalWrite(RELE, HIGH);
}

void ejecutar(int pasos, int eje, bool sentidoGiro) {
  switch (eje) {
    case 1:
      if (sentidoGiro) {
        digitalWrite(DirPinX, HIGH);
        move_motor(pasos, StepPinX);
      }
```

```
      else {
        digitalWrite(DirPinX, LOW);
        move_motor(pasos, StepPinX);
      }
      break;
    case 2:
      if (sentidoGiro) {
        digitalWrite(DirPinY, HIGH);
        move_motor(pasos, StepPinY);
      }
      else {
        digitalWrite(DirPinY, LOW);
        move_motor(pasos, StepPinY);
      }
      break;
    default: break;
  }
}
```

10.4 Caja UV

```
#define DirPin 5
#define StepPin  2
#define LUZ_BACK_LIGHT 9
#define UV1  11    // se definen los pines donde se conectaran los reles
#define UV2  10
#define VIBRA 3 //conectar el motor en la y-step
#define en 8
#define ReleMotor 12  //SpinEn

const int PASOS_VUELTA = 200;
const int PASOS_RPM=60;
const int PASOS_PERIODO=10000*(PASOS_RPM/60)/PASOS_VUELTA;

String orden1="UV-ON";
String orden2="UV-OFF";
String orden3="BACKLIGHT-ON";
String orden4="BACKLIGHT-OFF";
String orden5="MOTOR-ON";
String orden6="MOTOR-OFF";
String orden7="VIBRA"; //MAXIMO VALOR 255
String orden8="VIBRA-OFF";
boolean flag=false;
boolean move_motor=false;

String incoming;

void setup() {
  // put your setup code here, to run once:
  pinMode(UV1,OUTPUT);
  pinMode(UV2,OUTPUT);
  pinMode(LUZ_BACK_LIGHT,OUTPUT);
  pinMode(DirPin,OUTPUT);
  pinMode(StepPin,OUTPUT);
  pinMode(VIBRA,OUTPUT);
  pinMode(en,OUTPUT);
  pinMode(ReleMotor,OUTPUT);

  Serial.begin(115200);
  digitalWrite(DirPin,HIGH); //ponemos el motor en direccion DirPin
  digitalWrite(UV1,HIGH);
  digitalWrite(UV2,HIGH);
  digitalWrite(LUZ_BACK_LIGHT,HIGH);
  analogWrite(VIBRA,0);
  digitalWrite(ReleMotor,HIGH);
}

void loop() {
  char c;
  delay(50);
  //while(!Serial);
```

```
  while (Serial.available()){
   c=Serial.read();
   if (c!='#'){
    incoming=incoming+c;
   }
   else{//CUANDO LLEGA EL CARACTER #
    flag=true;
    break;
    }
  }
  if ((flag) && incoming != ""){
    flag=false;
    Serial.println(incoming);
    funcion1(incoming);
  }
  incoming="";
  if(move_motor){
    digitalWrite(StepPin,HIGH);
    delayMicroseconds(PASOS_PERIODO);
    digitalWrite(StepPin,LOW);
    delayMicroseconds(PASOS_PERIODO);
  }
}
void funcion1(String data){
    int pwd;
    String pot;
    if (!orden1.compareTo(data)){
      digitalWrite(UV1,LOW);
      digitalWrite(UV2,LOW);
      Serial.println("orden1");
      Serial.flush();
      }
    else if (!orden2.compareTo(data)){
      digitalWrite(UV1,HIGH);
      digitalWrite(UV2,HIGH);
      Serial.println("orden2");Serial.flush();}
    else if (!orden3.compareTo(data)){
      digitalWrite(LUZ_BACK_LIGHT,LOW);
      Serial.println("orden3");Serial.flush();}
    else if (!orden4.compareTo(data)){
      digitalWrite(LUZ_BACK_LIGHT,HIGH);
      Serial.println("orden4");Serial.flush();}
    else if (!orden5.compareTo(data)){
      move_motor=true;
      digitalWrite(ReleMotor,LOW);
      Serial.println("move_motor");Serial.flush();}
    else if (!orden6.compareTo(data)){
      move_motor=false;
      digitalWrite(ReleMotor,HIGH);}
    else if (data.startsWith(orden7)){
      pot=data.substring(5);
      pwd=pot.toInt();
      analogWrite(VIBRA,pwd);
      Serial.println(pwd);
      Serial.flush();
      }
    else if (!orden8.compareTo(data)){
      digitalWrite(VIBRA,LOW);
      Serial.flush();
      }
    else{
      Serial.println("mal");
      Serial.flush();
      }
}
```