Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Interconexión, mediante base de datos, de célula de fabricación con su gemelo digital

Autor: José María Azcutia Rodríguez

Tutor: Juan Manuel Escaño González

Cotutor: Javier Gómez Jiménez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Ingeniería de Tecnologías Industriales

Interconexión, mediante base de datos, de célula de fabricación con su gemelo digital

Autor:

José María Azcutia Rodríguez

Tutores:

Juan Manuel Escaño González
Javier Gómez Jiménez

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Interconexión, mediante base de datos, de célula de fabricación con su gemelo digital

Autor: José María Azcutia Rodríguez
Tutores: Juan Manuel Escaño González y
Javier Gómez Jiménez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mis abuelos, que en paz descansen

Agradecimientos

En primer lugar, agradecer a mis padres todo el apoyo que he recibido en esta etapa y a lo largo de mi vida, animándome y confiando en mí cuando más lo necesitaba.

Agradecer a mis tutores Juan Manuel Escaño y Javier Gómez por su tiempo y su confianza desde el primer momento. Especialmente a Javier, por su paciencia y ayudarme con sus conocimientos.

Agradecer a María por ser un apoyo incondicional, comprenderme y soportarme durante este proyecto y estos años.

Finalmente, doy las gracias a mi familia y mis amigos por acompañarme cada día y dejarme acompañarles.

José María Azcutia Rodríguez

Sevilla, 2022

Resumen

El marco general de este Trabajo Fin de Grado es la interconexión de datos de determinados dispositivos situados en una célula de fabricación flexible a un Gemelo Digital.
Para llevar a cabo dicha interconexión es necesario el diseño de una base de datos que realice de intermediario y almacene los datos que recibe de la planta real. Es por ello que en este trabajo se desarrolla una interconexión entre la célula de fabricación y la base de datos y otra interconexión entre la base de datos y el Gemelo Digital, implementado en Unity3D. Así como se desarrolla un aprendizaje del Gemelo Digital, a partir de los datos obtenidos, aplicado a casos concretos.

Abstract

The general objective of this Final Degree Project is the interconnection of data from certain devices located in a flexible manufacturing cell to a Digital Twin.
In order to carry out this interconnection, it is necessary to design a database that acts as an intermediary and stores the data received from the real plant. That is why in this work an interconnection is developed between the manufacturing cell and the database and another interconnection between the database and the Digital Twin, implemented in Unity3D. It also develops a learning process for the Digital Twin, based on the data obtained, applied to specific cases.

Índice

Índice de Figuras

1 Introducción

El presente proyecto forma parte del proyecto europeo DENiM (Digital Intelligence for collaborative for Energy managemente in Manufacturing). En el proyecto DENiM se desarrolla una cadena de herramientas integrada para prestar servicios digitales avanzados, mediante el uso de Internet de las Cosas (IoT), el análisis de datos, gemelos digitales, el modelado de energía y la automatización; con el objetivo final de supervisar y optimizar la gestión de la energía en la fabricación mediante inteligencia digital.[3]

El Gemelo Digital (Digital Twin) es una réplica virtual y dinámica de un determinado sistema que, a través de la información obtenida de sensores o automatismos, modela su comportamiento. De este modo, se posibilita su monitorización y análisis inteligente, así como la realización de simulaciones con el objetivo de optimizar el sistema, mejorando así el rendimiento del sistema real.[4]
El término "Gemelo Digital" comenzó a aplicarse en la industria a partir de 2003, cuando el Dr. Michael Grieves lo hizo durante una presentación sobre la gestión del ciclo de vida de un producto [5], a pesar de que este concepto estuviese presente en trabajos realizados por la NASA a finales de los 80. Como parte de su programa de investigación en la Universidad de Michigan, quería recrear representaciones digitales de sistemas físicos que tuvieran entidad por sí mismas. Esa información digital sería un "gemelo" vinculado al sistema físico durante todo su ciclo de vida.[6]

En la actualidad, los Gemelos Digitales tienen un papel fundamental en la industria. Según el informe 'Digital Twins: Adding Intelligence to the Real World'del Capgemini Research Institute [7], el 60 % de las organizaciones de los principales sectores se apoyan en los Gemelos Digitales, no solo para optimizar procesos; sino también, para cumplir con su agenda de sostenibilidad. Al poder simular el mundo físico, los Gemelos Digitales pueden ayudar a las organizaciones a utilizar mejor los recursos, reducir las emisiones de carbono, optimizar las redes de suministro y transporte; así como, aumentar la seguridad de los empleados. Entre las funciones de un Gemelo Digital se encuentran [8]:

- Detección de comportamientos con antelación.
- Reducción de los tiempos de inactividad.
- Reducción de costes.
- Identificación de oportunidades de negocio: debido al análisis masivo de datos, es posible comparar la utilidad con los datos empresariales e identificar mejor posibles oportunidades.

En cuanto a sus aplicaciones, destaca sobretodo en el sector aeroespacial, energético y automotriz.

El informe de Capgemini Research Institute comentado anteriormente[7], también revela que las implementaciones de gemelos digitales aumentarán en promedio un 36 % durante los próximos cinco años. Además, según la empresa de investigación de mercado "Markets and Markets", se estima que el mercado relacionado con la generación de Gemelos Digitales crecerá hasta los 73.500 millones de dólares en 2027, con una tasa de crecimiento anual compuesto (CAGR) del 60.6 % [9]. Por lo que la tecnología del Gemelo Digital seguirá en desarrollo durante los próximos años.

1.1 Objetivos

El objetivo principal de este Trabajo de Fin de Grado (TFG) es realizar la interconexión inalámbrica (a través del estándar WiFi) entre determinados dispositivos o sensores y un Gemelo Digital. Este modo de comunicación es una alternativa para que el Gemelo Digital obtenga determinados datos reales obtenidos sobre una célula de fabricación flexible, de modo que el Gemelo Digital realice un aprendizaje a partir de dichos datos.

El objetivo de realizar el aprendizaje del Gemelo Digital, tal y como se comenta más adelante, es poder recrear operaciones reales para que así sean lo más parecidas posibles a la realidad, de modo que sea posible realizar pruebas y estudios para optimizar procesos.

Particularmente, para demostrar la utilidad de la interconexión realizada, se instalan sobre la célula de fabricación flexible un acelerómetro y un sensor de corriente, cuyas características se comentan posteriormente.

El objetivo inicial de utilizar el acelerómetro en este proyecto es situarlos sobre bandejas que se mueven por las cintas transportadoras de la célula, para así obtener valores de velocidad de las cintas y transmitirlas al Gemelo Digital. Por otro lado, el objetivo de utilizar el sensor de corriente en este proyecto es situarlo sobre los cables de los motores de las cintas transportadoras, para así obtener la potencia que consumen y transmitirla al Gemelo Digital.

De este modo, mediante la utilización de un microcontrolador Arduino, el cual también se comenta más adelante, se obtienen medidas de los dispositivos comentados y se realiza un procesamiento para así transmitir los datos finales comentados previamente hacia el Gemelo Digital.

Particularmente, se pretenden realizan dos tipos de aprendizaje del Gemelo Digital: aprendizaje de potencia (o energía) y aprendizaje de la velocidad de las cintas transportadoras de la célula de fabricación flexible.

La comunicación no es directa, consta de dos partes:

- Comunicación inalámbrica entre los dispositivos situados en la célula de fabricación y una base de datos, la cual almacena los datos ya procesados por el microcontrolador.
- Comunicación bidireccional entre la base de datos y el Gemelo Digital implementado en Unity 3D.

En la siguiente imagen se describe visualmente el objetivo de este proyecto:

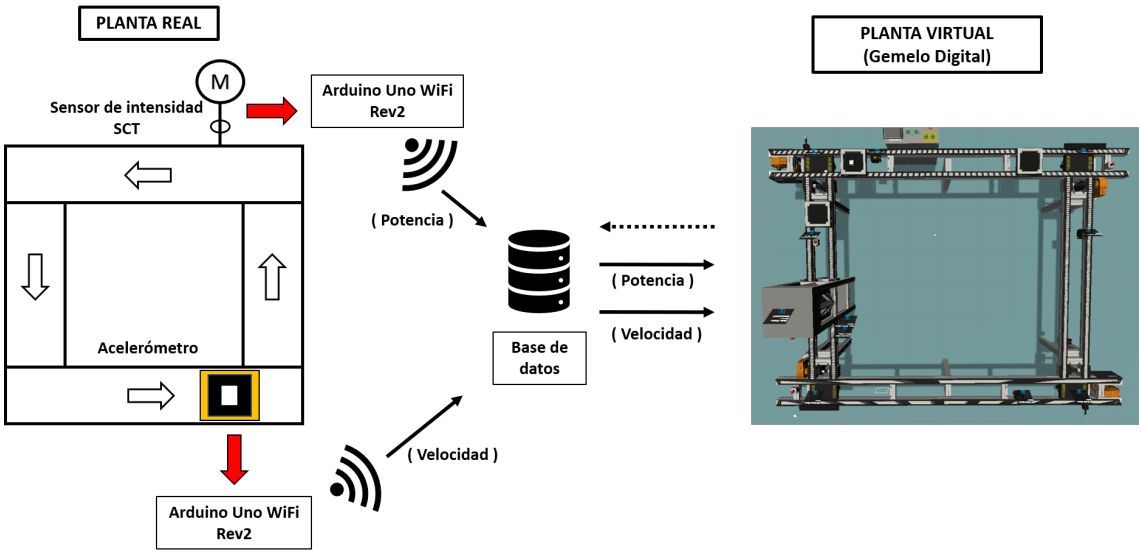


Figura 1.1 Esquema del objetivo del proyecto.

1.2 Célula de fabricación flexible

Los datos que recibe el Gemelo Digital se obtienen a partir los dispositivos instalados sobre la célula de fabricación flexible situada en la planta baja del laboratorio L1 de la Escuela Técnica Superior de Ingeniería

de la Universidad de Sevilla. Dicha célula de fabricación es el sistema real, y por tanto, el sistema que el Gemelo Digital pretende recrear a partir de la comunicación establecida entre ambos.

Una célula de fabricación flexible consiste en un grupo de estaciones de trabajo automatizadas e interconectadas mediante un sistema de transporte de materiales también automatizado, permitiendo así la producción de gran cantidad de productos diferenciados [10]. Los sistemas de fabricación flexible tratan de cubrir el vacío entre la producción con automatización fija (secuencias de procesado fijas y grandes cantidades de producción) y la producción con automatización programable (producción en lotes y bajas tasas de producción).[11]

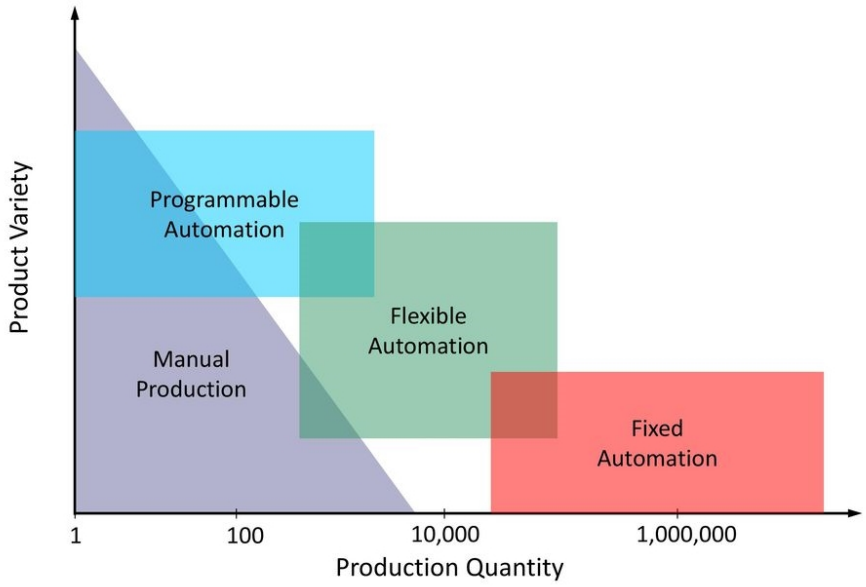


Figura 1.2 Diferencias entre la variación del producto y la cantidad para los distintos tipos de automatización.[1].

La célula de fabricación presente en el laboratorio comentado está conformada por los elementos indicados en el siguiente esquema:

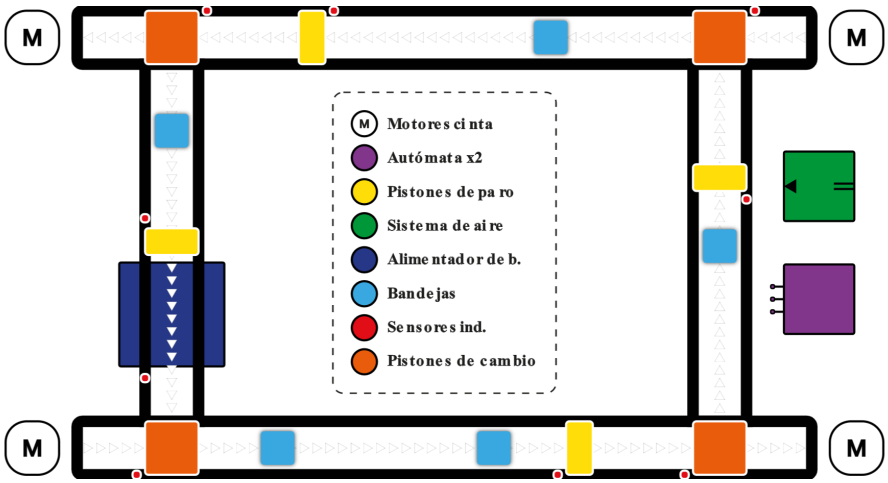


Figura 1.3 Elementos de la célula de fabricación flexible.

De todos ellos, para este proyecto son destacables las cintas transportadoras y los motores de las mismas. Sobre las cintas se desplazan las bandejas, sobre las cuales se instala un acelerómetro, para así obtener la velocidad de las cintas y transmitirla al Gemelo Digital. Por otro lado, en los cables de los motores de las

cintas se instala un sensor de corriente cuyo objetivo es medir la potencia (o energía) consumida por dichas cintas, para así transmitirla al Gemelo Digital.

Figura 1.4 Fotografía de la célula de fabricación flexible de la Escuela Técnica Superior de Ingeniería.

1.3 Metodología y estructura del documento

Este documento se estructura en siete capítulos, siendo este el primero llamado "Introducción". En el segundo, se presentan los dispositivos empleados y sus características y conexionado. Los programas necesarios para el desarrollo de este proyecto se presentan en el capítulo tres, así como también se presenta la plataforma de almacenamiento elegida y el diseño de las tablas que forman parte de la misma. En el capítulo cuatro se presenta la interconexión de datos entre la célula de fabricación y el gemelo digital, describiendo así las dos partes que conforman dicha interconexión. En el quinto capítulo se describe el tratamiento de datos realizado a los sensores, el cual es necesario ya que es la forma de obtener los resultados reales y con sentido según las necesidades para cumplir los objetivos del proyecto. En el capítulo seis se describen los dos tipos de aprendizaje que realiza el Gemelo Digital a partir de los datos reales enviados al mismo. Finalmente, en el último capítulo se presentan las conclusiones y posibles trabajos futuros.

Además, este documento cuenta con dos anexos, en los cuales se adjuntan los códigos utilizados para el desarrollo de este proyecto.

2 Descripción del hardware

En este capítulo se describen los dispositivos expuestos en 1.1. Concretamente, los dispositivos descritos son el microcontrolador Arduino UNO WiFi Rev2, el acelerómetro ADXL335 y el sensor de corriente SCT-013-030. La utilización de estos dispositivos se debe a aplicar la interconexión, que es el objetivo principal de este proyecto, a un ejemplo concreto. Este ejemplo consiste en instalar el sensor de corriente SCT junto con el acelerómetro en la planta real y conectados a microcontroladores Arduino. Posteriormente, el microcontrolador Arduino, gracias a su característica de conexión a WiFi, envía los datos al Gemelo Digital tras su procesamiento (procesamiento descrito en el capítulo 5 e interconexión con Gemelo Digital descrito en el capítulo 4).

2.1 Arduino UNO WiFi Rev2

El microcontrolador Arduino UNO WiFi Rev2 está indicado para su aplicación con Internet de las Cosas (IoT). En el presente proyecto, el microcontrolador se encarga de leer los datos de distintos sensores, realizar las operaciones pertinentes y enviar los resultados a la base de datos debido a su conexión a una red WiFi. Dentro de la familia de microcontroladores Arduino, destaca por tener módulo WiFi integrado, el cual es un SoC (System on a Chip) autónomo con protocolo TCP/IP integrado que puede proporcionar acceso a una red WiFi. Sus especificaciones técnicas son:

Tabla 2.1 Especificaciones técnicas del Arduino UNO Wifi Rev 2 [12].

Tablero	**Nombre**	Arduino UNO WiFi Rev 2
	SKU	ABX00021
Microcontrolador	ATmega4809	
Conector USB	USB-B	
Pines	**Pin LED incorporado**	25
	Pines de E/S digitales	14
	Pines de entrada analógica	6
	Pines PWM	5
Conectividad	**Bluetooth**	Módulo Nina W102 uBlox
	WiFi	Módulo Nina W102 uBlox
	Elemento seguro	ATECC608A
Sensores	**IMU**	LSM6DS3TR
Comunicación	**UART**	Sí
	I2C	Sí
	SPI	Sí
Energía	**Voltaje de E/S**	5 V
	Voltaje de entrada (nominal)	7 - 12 V
	Corriente CC por pin de E/S	20 mA
Velocidad de reloj	**Procesador**	ATmega4809 16 MHz
Memoria	**ATmega4809**	SRAM de 6 KB, flash de 48 KB, EEPROM de 256 bytes
	Módulo Nina W102 uBlox	ROM de 448 KB, SRAM de 520 KB, Flash de 2 MB

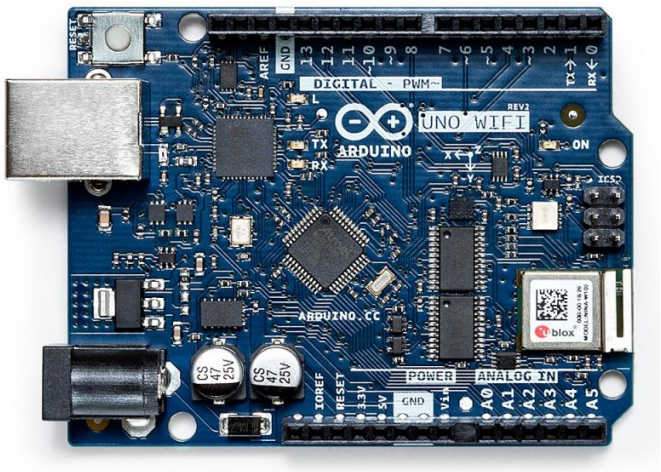


Figura 2.1 Arduino UNO Wifi Rev 2.

2.2 Sensor de corriente SCT-013-030

El SCT-013 es un sensor de corriente alterna no invasivo. Trabaja como un transformador con núcleo de ferrita, ya que la corriente que circula por el cable que se desea medir actúa como devanado primario de una espira, mientras que el propio sensor tiene internamente un devanado secundario de 1800 espiras. Es de fácil instalación debido a su forma de pinza, la cuál se abre, se introduce un único cable y se vuelve a cerrar.

Figura 2.2 Sensor de corriente SCT-013-030.

Este proyecto utiliza el modelo SCT-013-030, el cual se caracteriza por permitir realizar medidas de una carga de hasta 30 A. Tiene una relación nominal de 30A/1V, lo que significa que para una entrada de 30 A genera una salida de 1 V tal y como refleja la Figura 2.3 [13]. El sensor tiene una resistencia de carga incorporada (R_L) de 62 Ω.

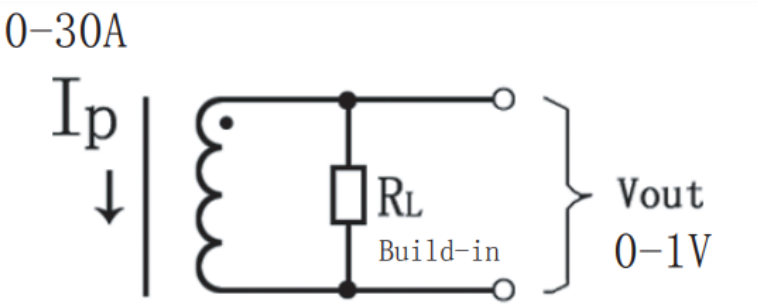


Figura 2.3 Relación 30A/1V del sensor de corriente SCT-013-030.

La utilidad que tiene este dispositivo en el presente proyecto es la de obtener la corriente que circula por los motores que alimentan las cintas transportadoras de la célula de fabricación. Posteriormente, se calcula su potencia (o energía) y será transmitida al Gemelo Digital.

2.2.1 Acondicionamiento de señal

Al ser señal alterna, se debe acondicionar la señal de modo que el microcontrolador Arduino reciba una tensión de entrada analógica entre 0 y +5V; ya que no admite tensiones negativas y podría dañar el Arduino. El acondicionamiento de la señal de este dispositivo se realiza mediante la rectificación de la entrada de modo que se trabaja con la parte positiva asumiendo la simetría de la señal.

Se utiliza un amplificador operacional LM358 (Figura 2.5) configurado como seguidor de tensión como se observa en la Figura 2.4. Esta decisión se debe a que no se puede rectificar con diodos ya que la caída de tensión en el diodo es muy grande en comparación con la tensión de la señal.[2]

Alimentar el LM358 con una tensión de 5V implica que sature a 3.5V aproximadamente, por tanto no es posible amplificar hasta 5V. Es por ello que se decide trabajar con la referencia analógica interna del microcontrolador Arduino, que en caso del Arduino WiFi Uno Rev2 se trata de 0.55V.

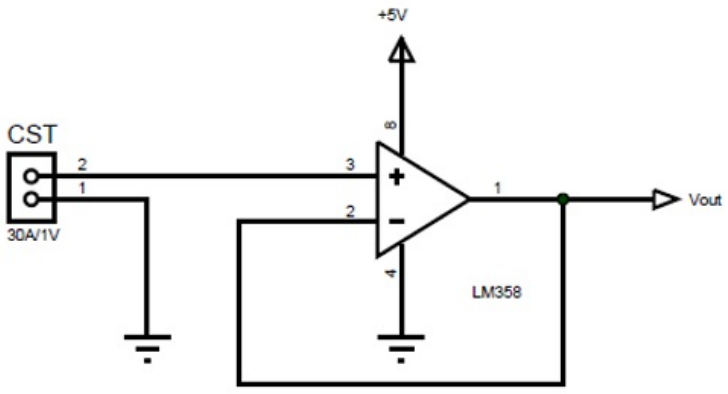
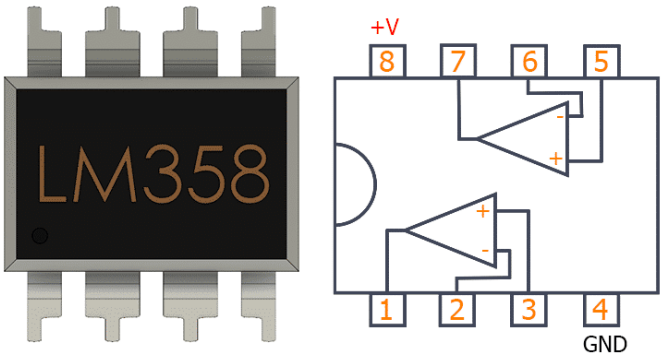


Figura 2.4 Circuito de acondicionamiento de señal. [2].



PINOUT LM358	
1. OUTPUT A	5. NO-INVERTING INPUT B
2. INVERTING INPUT A	6. INVERTING INPUT B
3. NO-INVERTING INPUT A	7. OUTPUT B
4. GND	8. +V

Figura 2.5 Amplificador operacional LM358.

2.2.2 Cálculo de corriente eficaz y potencia

Al realizar un tratamiento de los valores que lee el SCT, el cual se comenta posteriormente, se puede obtener la corriente en cada instante.
A partir de la corriente en cada instante se puede calcular la corriente eficaz (Irms) y la potencia.

El valor eficaz de una corriente alterna es el valor que tendría una corriente continua que produjera la misma potencia que dicha corriente alterna al aplicarla sobre una misma resistencia. Esta surge de la necesidad de medir la efectividad de una fuente de tensión o corriente, al suministrar potencia a una carga resistiva.

El cálculo de la corriente eficaz es:

$$i_{RMS} = \sqrt{\frac{1}{T}\int_0^T i^2 \mathrm{d}t} \tag{2.1}$$

Mientras que el cálculo de la corriente eficaz en tiempo discreto es:

$$I_{RMS} = \sqrt{\frac{1}{N}\sum_{n=0}^{N} i_n^2} \tag{2.2}$$

Siendo N el número de muestras en un periodo determinado.

2.2.3 Conexionado

En la siguiente figura se pueden observar las conexiones entre el sensor SCT-013-030 y el Arduino Uno WiFi Rev2, pasando por el amplificador operacional LM358:

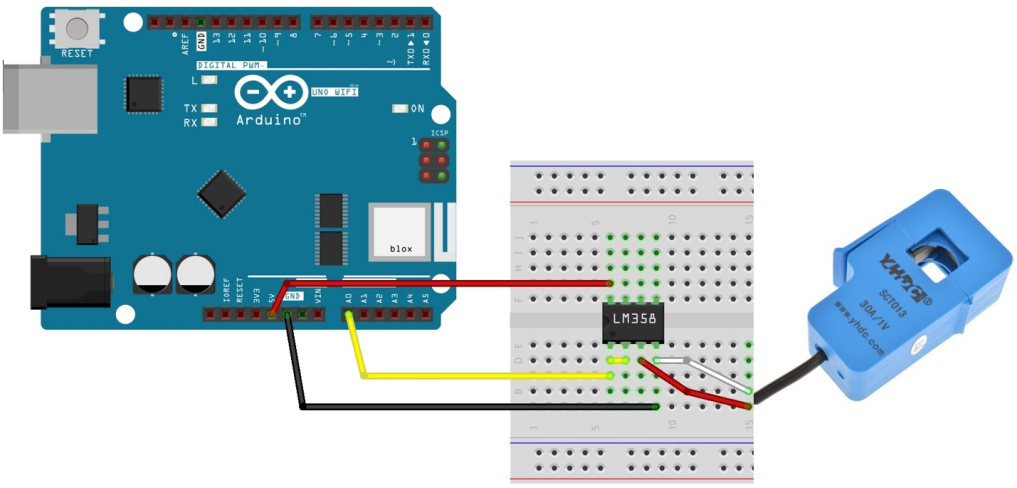


Figura 2.6 Conexionado del SCT-013-030 al Arduino Uno WiFi Rev2.

2.3 Acelerómetro ADXL335

Un acelerómetro consiste en un dispositivo electromecánico que mide la fuerza de aceleración. En este proyecto, el modelo de acelerómetro utilizado es el ADXL335.
El ADXL335 es un acelerómetro con interfaz analógica, por lo que entrega un voltaje proporcional a la aceleración en cada uno de sus 3 ejes (X, Y, Z).

Este acelerómetro mide la aceleración con un rango de $\pm$ 3g y opera con un nivel de tensión de 3.3 V. Puede medir la aceleración estática de la gravedad así como la aceleración dinámica resultante del movimiento, choque o vibración.
Tiene una salida de datos máxima de 550 Hz sin elementos externos para los tres ejes y podría aumentarse hasta los 1600 Hz para los ejes X e Y al incorporar capacitores al circuito [14].

Entre otros aspectos, cuenta con buena estabilidad frente a los cambios de temperatura con un rango de operación entre -55^{o}C y 125^{o}C. Además, posee buena resistencia física pudiendo soportar fuertes impactos.

Figura 2.7 Acelerómetro ADXL335.

La utilidad que tiene este dispositivo en el presente proyecto es la de obtener la aceleración de su movimiento, con el objetivo de calcular su velocidad. Esta velocidad es la que experimentan las bandejas que circulan por las cintas transportadoras de la célula de fabricación, que tras un proceso de tratamiento y filtrado son transmitidas al Gemelo Digital.

Para la utilización del acelerómetro se ha decidido utilizar la librería de Arduino IDE denominada "Accelerometer ADXL335" [15]. Antes de su utilización, hay que realizar una calibración con el objetivo de

definir varios parámetros con los que se realiza el cálculo de la aceleración. Dicha calibración consiste en cargar el fichero de ejemplo de esta librería denominado "Calibration.ino".

Al observar el monitor serie de Arduino IDE, indica que se coloque el acelerómetro de manera que la dirección del eje Z apunte hacia arriba y a continuación presione una tecla. Luego, hay que realizar lo mismo pero apuntando la dirección del eje X hacia arriba. Una vez realizado esto, el código devuelve tres valores denominados "$ZERO_X$", "$ZERO_Y$" y "$ZERO_Z$", los cuales representan la tensión cuando la aceleración es nula en cada eje.

Esos valores deben ser próximos a 1.65 V, debido a que al estar el acelerómetro sin movimiento (0g) la salida analógica residirá en la mitad de la tensión de alimentación, por lo que al ser alimentado por 3.3 V, da ese valor de 1.65 V aproximadamente. Además, el código de calibración también devuelve un valor denominado "SENSITIVITY", el cual indica la sensibilidad en los ejes X, Y y Z medido en V/g. Estos cuatro parámetros deben definirse como macros en el fichero "ADXL335.h" de la librería comentada anteriormente. Una vez finalizado esto, queda completada la calibración inicial del acelerómetro.

En el presente proyecto se va a utilizar la librería comentada para el cálculo de la aceleración en cada instante, para así posteriormente poder calcular velocidades. Dicha librería realiza el cálculo de la aceleración del siguiente modo:

Calcula la tensión (V) que representa el eje X:

$$x_{voltage} = \frac{x \cdot ADC_{Ref}}{ADC_{Amplitude}} \tag{2.3}$$

Siendo:

x: Valor de lectura analógica del acelerómetro en el eje x.

ADC_{Ref}: Valor del voltaje de referencia para el convertidor analógico digital del Arduino Uno WiFi Rev2. En este caso valdrá 5 V.

$ADC_{Amplitude}$: Valor de la resolución del convertidor analógico digital del Arduino Uno WiFi Rev2. En este caso valdrá 1024 al ser de 10 bits.

Finalmente, calcula la aceleración (g) en el eje X:

$$a_x = \frac{x_{voltage} - ZERO_X}{Sensitivity} \tag{2.4}$$

Siendo:
$x_{voltage}$: Valor de la tensión (V) que representa el eje X.

$ZERO_X$: Valor de tensión (V) cuando la aceleración del eje X es 0g.

Sensitivity: Valor de la sensibilidad (V/g) del acelerómetro.

Con los ejes Y y Z se actuaría de la misma forma.

2.3.1 Cálculo de la velocidad a partir de la aceleración

Como ya es conocido, la velocidad se puede calcular a partir de la aceleración mediante la integración de esta respecto al tiempo.

En este proyecto, la integral de la aceleración se ha realizado mediante el método del trapecio:

$$v(t) = \int_{t_1}^{t_2} a(t)\mathrm{d}t \approx \frac{a(t_1) + a(t_2)}{2}(t_2 - t_1) \tag{2.5}$$

Que en términos discretos se puede obtener mediante:

$$,v[k] = v[k-1] + \frac{a[k] + a[k-1]}{2}\Delta t \tag{2.6}$$

En este caso, para mejorar la fiabilidad de las medidas de diferencias de tiempo, se realiza la media de la diferencia de tiempo entre medidas correspondiente y la del instante anterior. Siendo $\Delta t = \frac{\Delta t_k + \Delta t_{k-1}}{2}$

2.3.2 Conexionado

El conexionado entre el acelerómetro ADXL335 y el Arduino Uno WiFi Rev2 se puede observar en la Figura 2.8.

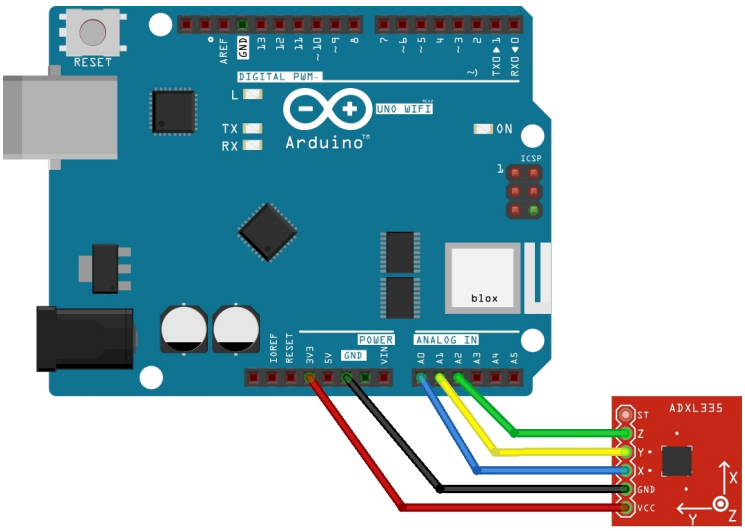


Figura 2.8 Conexionado del acelerómetro ADXL335 al Arduino Uno WiFi Rev2.

3 Descripción del software

En este capítulo se presentan los distintos programas utilizados para llevar a cabo el proyecto. Por un lado, es necesario el uso de Arduino IDE, para así poder programar y cargar el código correspondiente a la placa Arduino Uno WiFi Rev2, así como también para visualización de resultados. Por otro lado, también es necesario el uso de Unity 3D, ya que es el software en el que se encuentra implementado el Gemelo Digital perteneciente al proyecto DENiM. En Unity3D se toman los datos previamente obtenidos con Arduino y así poder implementar el aprendizaje del Gemelo Digital a partir de dichos datos.

El software utilizado para implementar la base de datos se encuentra explicado en el capítulo 3.3.

3.1 Arduino IDE

Un entorno de desarrollo integrado, llamado IDE (siglas en inglés de "Integrated Development Environment), consiste en un programa de aplicación basado en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica (GUI) [16].

En el caso de Arduino IDE, es el entorno de programación proporcionado por la empresa Arduino y sirve para escribir y cargar los códigos a la placa mediante el puerto serie. También, ofrece el "Monitor Serie", útil para mostrar por pantalla mensajes o valores de variables, entre otros; y el "Serial Plotter", útil para mostrar gráficas en tiempo real. Este es un software gratuito disponible en la web de Arduino. Permite la instalación de multitud de librerías ya creadas, las cuales serán muy útiles para conseguir mayor simplicidad en los códigos.

Figura 3.1 Arduino IDE.

3.2 Unity 3D

Unity es un software de desarrollo de videojuegos, visualizaciones y animaciones en tiempo real, tanto en 2D como en 3D, creado por la empresa Unity Technologies. Que sea un software de desarrollo de videojuegos se conoce como motor de videojuego, lo que hace referencia a una herramienta que tiene una serie de rutinas de programación que permiten la creación, diseño y funcionamiento del entorno interactivo.

El Gemelo Digital desarrollado en el proyecto DENiM se ha realizado con Unity 3D, donde se puede visualizar la recreación virtual de la célula de fabricación y simular los distintos comportamientos de la misma a partir de datos reales. El lenguaje de programación utilizado es C#.

Entre sus características cabe destacar su motor físico (Nvidia y PhysX), el cual permite gran cantidad de funcionalidades y herramientas de navegación NavMesh para inteligencia artificial o soporte de Realidad Virtual. Además de su interfaz intuitiva, alta calidad de gráficos, software multiplataforma y la ejecución paralela de todos los módulos implementados.

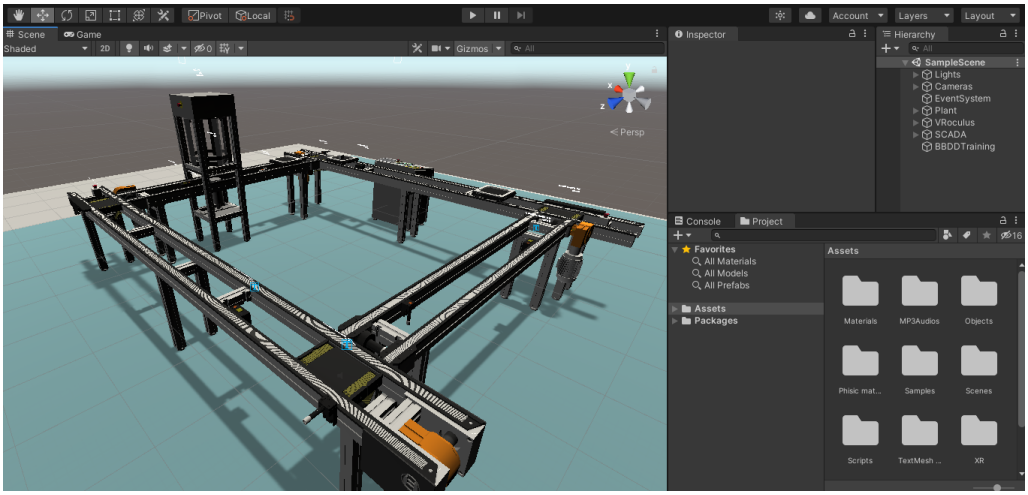


Figura 3.2 Interfaz de Unity 3D.

3.3 Bases de datos

Este proyecto requiere de la utilización de una base de datos para almacenar todos los datos recogidos por los dispositivos mediante el microcontrolador Arduino WiFi Uno Rev2. La base de datos es el nexo de unión entre los dispositivos Arduino instalados en la célula y el Gemelo Digital implementado en Unity3D. También, se diseña para almacenar datos enviados desde Unity3D.

3.3.1 PostgreSQL

Es un sistema gestor de base de datos relacional de código abierto y orientado a objetos.Su origen fue en 1986 como parte del proyecto POSTGRES de la Universidad de California en Berkeley.

Las características principales de este sistema son [17]:

- Disponibilidad para la mayoría de sistemas operativos.
- Facilidad de uso.
- Robustez.
- Modelo MVCC (Multiversion Concurrency Control). Significa que permite acceder de manera concurrente a la base de datos, de modo que permite ejecutar consultas que tocan las mismas filas simultáneamente, mientras mantiene esas consultas aisladas entre sí.
- Base de datos ACID, es decir, garantiza que las transacciones de la base de datos se realizan de manera fiable. ACID son las siglas de Atomicidad, Consistencia, Aislamiento y Durabilidad.

Además, PostgreSQL cuenta con una licencia libre, por lo que puede ser utilizado, modificado y distribuido por cualquier persona de forma gratuita para cualquier propósito; ya sea comercial o académico.

Por todo ello, se ha decidido utilizar el gestor PostgreSQL para el desarrollo de este proyecto.

PgAdmin4

Es una herramienta de código abierto necesaria para gestionar y administrar PostgreSQL. Esta permite acceder a las funcionalidades de la base de datos y desde ella se puede realizar gráficamente todo lo que PostgreSQL permite, a través de comandos.

Está escrita en Python y jQuery, y es compatible con todas las características de PostgreSQL. Cabe destacar la herramienta "Query Tool", mediante la cual se ejecutan los comandos SQL necesarios para la creación de las tablas e inserción de datos en las mismas.

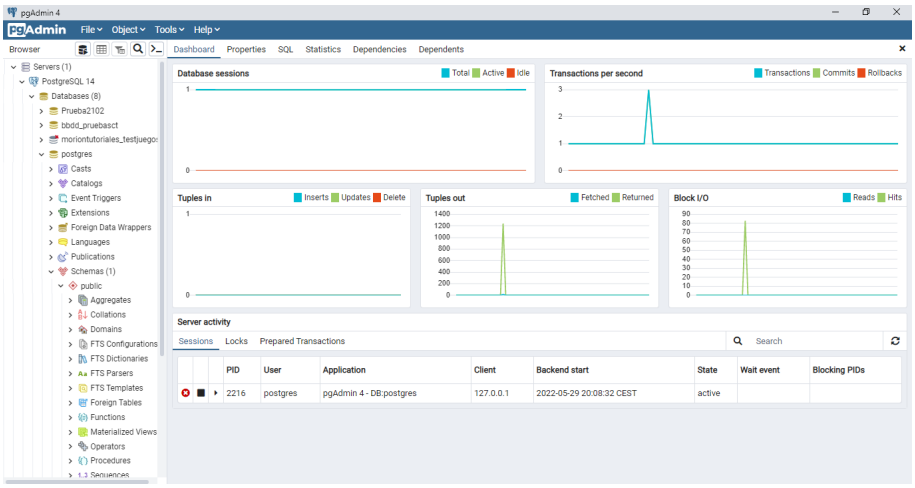


Figura 3.3 Interfaz gráfica de pgAdmin4.

SQL

El lenguaje de consulta estructurado o SQL (Structured Query Language) es un lenguaje declarativo de acceso a bases de datos relacionales que permite realizar diversos tipos de operaciones en ellas: creación de bases de datos, de tablas, modificar o eliminar las mismas.

Una de sus características es el manejo del álgebra y el cálculo relacional, que permite efectuar consultas con el objetivo de obtener de forma sencilla información de interés de bases de datos, así como hacer cambios en ella [18].

Diseño y creación de la base de datos

Como paso previo a la creación de la base de datos, en la cuál se va a almacenar la información, está el diseño de la misma.La base de datos que se va a crear consta de dos tipos de tablas.

En primer lugar, hay dos tablas denominadas "de datos", donde hay un campo que indica la cantidad de dispositivos (acelerómetros y SCTs) que se van a utilizar en el proyecto, junto con un nombre que lo defina y un comentario. Por ejemplo, para conocer qué dispositivo es y dónde se va a situar.

Las tablas son idénticas para ambos tipos de dispositivos salvo el identificador, el cuál es "ID_Accel" en el caso del acelerómetro y "ID_SCT" en el del SCT.

Campo	Tipo
ID	SERIAL PRIMARY KEY
Nombre	CHAR (30)
Comentario	CHAR (50)

Tabla 3.1 Campos de la tabla de datos.

Por otro lado, hay dos tabla denominadas "de valores", donde se almacenan los datos que envía el Arduino a la base de datos. Se distingue ahora entre la tabla de valores del acelerómetro y la tabla de valores del SCT, para medidas del acelerómetro y para medidas del SCT, respectivamente.

- **Tabla de valores del acelerómetro:**
  En esta tabla hay un identificador autoincremental "ID" y un identificador dependiente del identificador de la tabla de datos para así evitar errores "ID_Accel". Además, un valor de velocidad en eje 'x', en eje 'y'y en eje 'z', junto con la marca temporal en cual se sube el dato a la tabla.

Campo	Tipo
ID	SERIAL PRIMARY KEY
ID_Accel	INT
SpeedX	FLOAT
SpeedY	FLOAT
SpeedZ	FLOAT
Time	TIMESTAMP

Tabla 3.2 Campos de la tabla de valores del acelerómetro.

- **Tabla de valores del SCT:**
  En esta tabla hay un identificador autoincremental "ID" y un identificador dependiente del identificador de la tabla de datos para así evitar errores "ID_SCT". Además, un valor de intensidad, de potencia y de diferencial de tiempo, junto con la marca temporal en cual se sube el dato a la tabla.

Campo	Tipo
ID	SERIAL PRIMARY KEY
ID_SCT	INT
Current	FLOAT
Power	FLOAT
DeltaTime	FLOAT
Time	TIMESTAMP

Tabla 3.3 Campos de la tabla de valores del SCT.

Los campos ID_Accel y ID_SCT tienen clave foránea, ya que sus valores corresponden a los valores de la clave primaria de la tabla de datos. Para poder añadir una fila con un valor de clave foránea específico, debe existir una fila en la tabla relacionada con el mismo valor de clave primaria.

Por ejemplo, si en la tabla de datos hay introducidas filas cuyo campo ID sea '1', '2'y '3'y se intenta introducir en la tabla de valores una fila cuyo campo ID sea '4', éste no se introducirá y saltará mensaje de error.

En estas tablas se ha suprimido la declaración de clave foránea por ser extensa, en los códigos SQL que se adjuntan a continuación se ve en detalle la creación de las tablas comentadas.
La creación de las tablas se realiza mediante los siguientes códigos en lenguaje SQL.

- Creación de tabla de datos del acelerómetro:

```
CREATE TABLE tabla_datos_acelv1(
ID_Acel SERIAL PRIMARY KEY,
Nombre char(30),
Comentarios char(50)
);
```

- Creación de tabla de valores del acelerómetro:

```
CREATE TABLE tabla_valores_acelv1(
ID_Valor SERIAL PRIMARY KEY,
ID_Acel INT,
CONSTRAINT fk_Acel FOREIGN KEY (ID_Acel) REFERENCES
    tabla_datos_acelv1(ID_Acel),
```

```
    Vel_x FLOAT,
    Vel_y FLOAT,
    Vel_z FLOAT,
    Tiempo TIMESTAMP
    );
```

- Creación de tabla de datos del SCT:

```
    CREATE TABLE tabla_datos_sctv1(
    ID_Sensor SERIAL PRIMARY KEY,
    Nombre char(30),
    Comentarios char(50)
    );
```

- Creación de tabla de valores del SCT:

```
    CREATE TABLE tabla_valores_sctv1(
    ID_Valor SERIAL PRIMARY KEY,
    ID_Sensor INT,
    CONSTRAINT fk_Sensor FOREIGN KEY (ID_Sensor) REFERENCES
        tabla_datos_sctv1(ID_Sensor),
    Corriente FLOAT,
    Potencia FLOAT,
    difTiempo FLOAT,
    Tiempo TIMESTAMP
    );
```

4 Interconexión

Como se explicó en 1.1, la interconexión de datos entre la célula de fabricación y el Gemelo Digital no es directa. Requiere en primer lugar de una interconexión entre el Arduino Uno WiFi Rev2 instalado sobre la célula y la base de datos, que es en lo que se centra este capítulo. En segundo lugar, requiere de una interconexión entre la base de datos y el Gemelo Digital implementado en Unity3D.

Por tanto, en este capítulo se explican las dos fases comentadas de interconexión que se deben realizar para conseguir el objetivo del proyecto.

4.1 Interconexión microcontrolador Arduino - Base de datos PostgreSQL

Este subcapítulo se centra en la interconexión entre el microcontrolador Arduino Uno WiFi Rev2 instalado sobre la célula y la base de datos. Para realizar la interconexión entre Arduino Uno WiFi Rev2 y la base de datos implementada en PostgreSQL, se realiza desde Arduino una solicitud HTTP que ejecuta un fichero PHP. Este fichero PHP se encarga de conectarse a la base de datos y de introducir mediante consultas SQL los datos enviados en la tabla de valores deseada.

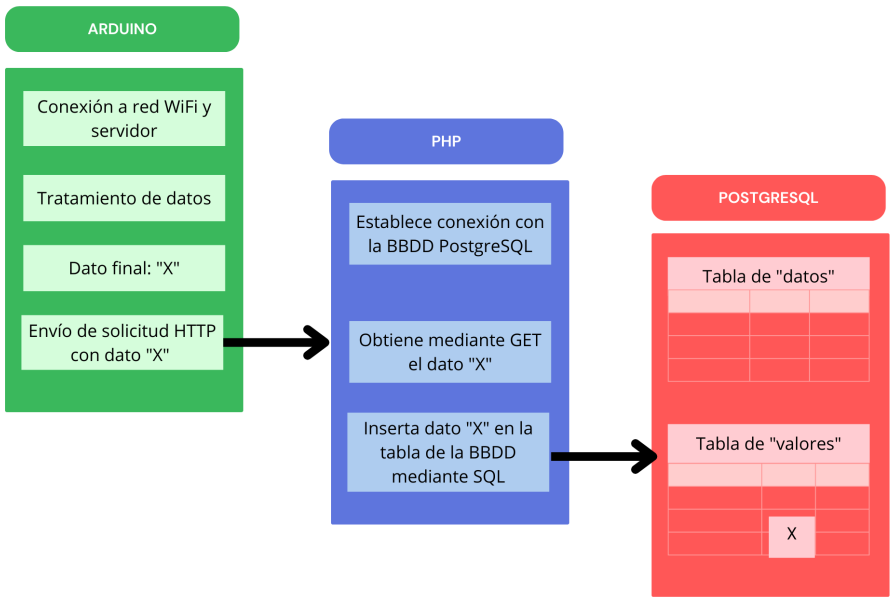


Figura 4.1 Esquema del procedimiento para la interconexión de datos Arduino - Base de datos.

De forma más detallada, se realiza del siguiente modo:

1. En primer lugar, es necesario que el microcontrolador Arduino Uno WiFi Rev2 se encuentre conectado a una red WiFi para así poder realizar el envío. Además de introducir las librerías y funciones explicadas

en 4.1.1.

2. Se realiza la solicitud HTTP que ejecute el fichero PHP necesario, se deben incluir las siguientes líneas de código en Arduino. Por ejemplo, en caso del SCT se desean enviar los datos de corriente eficaz Irms, potencia y diferencia de tiempo. Para ello, se debe ejecutar lo siguiente:

```
//// SENDING TO DATABASE ////

if ( aux_status == WL_CONNECTED)
{
  String Irms_string= String(Irms,6);
  String P_string= String(P,6);
  String ID_Sensor_string= String(ID_Sensor);
  String difTiempo_string = String(TimeDifference,6);

  String httpRequestData =
      "/php2pgsql/php_prueba_linkeada_sctv1.php?ID_Sensor="+ID_Sensor_string+
      "&Corriente="+Irms_string+"&Potencia="+P_string+"&difTiempo="+difTiempo_string;

  Serial.println("Making get request");

  client.sendHeader(HTTP_HEADER_CONTENT_TYPE,"application/x-www-form-urlencoded");

  client.get(httpRequestData);

  int statusCode = client.responseStatusCode();
  Serial.print("Status code: ");
  Serial.println(statusCode);

  String response = client.responseBody();
  Serial.print("Response: ");
  Serial.println(response);
}
```

3. El fichero PHP al que se llama, se encarga de conectarse a la base de datos y si tiene éxito, introducir los datos recibidos en la tabla deseada de la base datos mediante consultas SQL. Por ejemplo, en caso del SCT, el fichero PHP que introduce los datos sería el siguiente:

Código 4.1 Código PHP para introducir datos del SCT en la BBDD.

```
<?php

$conexion = pg_connect("host=localhost port=5432 dbname=bbdd_pruebasct
    user=postgres password=josemariaazcutia");

if($conexion)
{
  echo "Se conectó correctamente";
}
else
{
  echo "Ha ocurrido un problema";
}
```

```
    $sql="INSERT INTO tabla_valores_sctv1 (ID_Sensor, Corriente, Potencia,
        difTiempo, Tiempo) VALUES ('" . $_GET['ID_Sensor'] . "','" .
        $_GET['Corriente'] . "','" . $_GET['Potencia'] . "','" .
        $_GET['difTiempo'] . "',CURRENT_TIMESTAMP)";

    $consulta=pg_query($conexion, $sql) or die ("No se envió");

    pg_close( $conexion );

    ?>
```

Como puede observarse, para realizar la conexión con la base de datos es necesario el uso de la orden *pg_connect*, la cual abre una conexión a PostgreSQL. Su sintaxis se explica en 4.2.1.

También se hace uso de la orden *pg_query* la cual ejecuta una consulta. Dicha consulta será la orden SQL. Su sintaxis se explica en 4.2.1.

Finalmente, cierra la conexión PostgreSQL mediante la orden *pg_close*. Su sintaxis también se explica en 4.2.1.

4. De este modo, queda almacenado el dato enviado en la columna y tabla deseadas, como se describe en 3.3.1.

En caso de los datos del acelerómetro, la manera de proceder es análoga.

En ese caso, el fichero PHP encargado de introducir los datos en la base de datos es:

Código 4.2 Código PHP para introducir datos del acelerómetro en la BBDD.

```
<?php

$conexion = pg_connect("host=localhost port=5432 dbname=bbdd_pruebasct
    user=postgres password=josemariaazcutia");

if($conexion){
  echo "Successfully connected";

}
else{
  echo "A problem has occurred";
}


$sql="INSERT INTO tabla_valores_acelv1 (ID_Acel, vel_x, vel_y, vel_z, Tiempo)
    VALUES ('" . $_GET['ID_Acel'] . "','" . $_GET['vel_x'] . "','" .
    $_GET['vel_y'] . "','" . $_GET['vel_z'] . "',CURRENT_TIMESTAMP)";

$consulta=pg_query($conexion, $sql) or die ("Not sent");

pg_close( $conexion );

?>
```

4.1.1 Librerías y funciones necesarias para el envío de datos a la base de datos

Para lograr el correcto envío de los datos a la base de datos, en el código de Arduino, es necesario utilizar las librerías "**WiFiNINA.h**", "**SPI.h**" y "**ArduinoHttp-Client.h**" en Arduino IDE.

- "**WiFiNINA.h**" [19]:
  La librería "WiFiNINA" permite utilizar las capacidades WiFi del dispositivo Arduino UNO WiFi Rev2, entre otros. Se utilizan las llamadas:

  - **WiFiClient**: Crea un cliente que puede conectarse a una dirección IP de Internet especificada.
  - **WiFi.begin(ssid,pass)**: Inicializa la configuración de red de la librería WiFiNINA y proporciona el estado actual. "**ssid**" es el nombre de la red WiFi a la que se desea conectar y "**pass**" es la contraseña WPA de dicha red.
    Devuelve WL_CONNECTED cuando está conectado a una red y WL_IDLE_STATUS cuando no está conectado a una red pero está encendido.
- "**SPI.h**":
  La librería "SPI" es utilizada por la librería "WiFiNINA", por lo que también debe ser cargada.
- "**ArduinoHttp-Client.h**" [20]:
  La librería "ArduinoHttpClient" permite facilitar la interacción con los servidores web mediante HTTP. Se utilizan las llamadas:

  - **HttpClient**: Proporciona una clase para enviar solicitudes HTTP.
  - **HttpClient(wifi, serverName, port)**: Permite realizar la conexión al servidor para realizar las solicitudes HTTP. "**wifi**" es la clase declarada con WiFiClient (WiFiNINA), "**servername**" es la dirección IP del servidor y "**port**" es el puerto, el cual será 80 debido a que es el que se usa para HTTP.
  - **client.sendHeader(HTTP_HEADER_CONTENT_TYPE, "application/x-www-form-urlencoded")**: Proporciona información sobre el tipo de archivo para la solicitud HTTP. En este caso los valores son codificados en tuplas llave-valor separadas por "&", con un "=" entre la llave y el valor, por tanto se añade "application/x-www-form-urlencoded" [21].
  - **client.get()**: Realiza la solicitud GET. Se le pasa como parámetro la URL donde se realiza la solicitud HTTP.

Destacar que para la conexión con PostgreSQL el puerto determinado es el 5432.

4.2 Interconexión Base de Datos - Gemelo Digital

Este subcapítulo se centra en la segunda parte de la interconexión de datos entre los dispositivos instalados en la célula y el Gemelo Digital, es decir, describe la comunicación de la base de datos en PostgreSQL con el Gemelo Digital implementado en Unity 3D. Así como las operaciones a realizar en Unity 3D para la obtención de datos independiente de cada dispositivo. Dentro de un mismo tipo de dispositivo también se obtienen diversos parámetros.

4.2.1 Interconexión Base de Datos - Unity 3D

La comunicación entre la base de datos PostgreSQL y Unity3D se realiza mediante órdenes en lenguaje PHP que ejecutan solicitudes SQL determinadas. De manera similar a lo comentado al inicio de 4.1, Unity3D se conecta con la base de datos, mediante PHP, y obtiene los datos deseados de la base de datos en Unity3D para su posterior uso.

En primer lugar, se describen las órdenes a ejecutar en el fichero PHP para conseguir el objetivo de este capítulo.

- Para realizar la conexión con la base de datos, es necesario el uso de la orden *pg_connect*, la cual abre una conexión a PostgreSQL.
  Su sintaxis es: $\$connection = pg\_connect(\text{"}host\text{"},\text{"}port\text{"},\text{"}options\text{"},\text{"}tty\text{"},\text{"}dbname\text{"})$. En su argumento se define el servidor, puerto, nombre de la base de datos, usuario y contraseña de PostgreSQL.
- Mediante la orden *pg_query* se ejecuta una consulta.
  Su sintaxis es: $\$pg\_query(\$connection, \$query)$ y ejecuta la consulta dada por $\$query$ en la conexión a la base de datos especificada por $\$connection$. Dicha consulta será la orden SQL.

- La orden *pg_num_rows* devuelve el número de filas en un resultado.
  Su sintaxis es: *pg_num_rows*($*result*) siendo $*result* la respuesta a la orden *pg_query*.
  El uso de esta orden se debe a conocer si una tabla tiene alguna fila con valores almacenados. Es necesario ya que si su valor es 0 es incoherente realizar ninguna operación.
- La orden *pg_fetch_assoc* obtiene una fila como una matriz asociativa.
  Su sintaxis es: $*pg_fetch_assoc*($*result*) siendo $*result* la respuesta a la orden *pg_query*.
  El uso de esta orden se realiza para que mientras exista una fila de datos, coloca esa fila en $*row* para mostrarla según indica *echo*.

El código PHP para realizar únicamente la conexión con la base de datos es:

Código 4.3 Código PHP que permite la conexión a la BBDD.

```php
<?php

$conexion = pg_connect("host=localhost port=5432 dbname=bbdd_pruebasct
    user=postgres password=josemariaazcutia");

if($conexion)
{
  echo "Se conectó correctamente";
}
else
{
  echo "Ha ocurrido un problema";
}

?>
```

Por otro lado, los códigos PHP para obtener todos los datos almacenados en la base de datos PostgreSQL son:

- En caso del acelerómetro:

Código 4.4 Código PHP que devuelve todos los datos del acelerómetro almacenados en la BBDD.

```php
<?php

include('connection_bbdd_unity_sct.php');

$sql = "select id_valor,id_acel,vel_x,vel_y,vel_z,tiempo from
    tabla_valores_acelv1";

$result = pg_query($conexion,$sql);

if(pg_num_rows($result)>0)
{
  while($row = pg_fetch_assoc($result))
  {
    echo "id_valor:".$row['id_valor']."|id_acel:".$row['id_acel']."|vel_x:"
        .$row['vel_x']."|vel_y:".$row['vel_y']."|vel_z:".$row['vel_z'].
        "|tiempo:".$row['tiempo'].";";
  }
}

?>
```

- En caso del SCT:

Código 4.5 Código PHP que devuelve todos los datos del SCT almacenados en la BBDD.

```
<?php

include('connection_bbdd_unity_sct.php');

$sql = "select id_valor,id_sensor,corriente,potencia,diftiempo,tiempo from
    tabla_valores_sctv1";

$result = pg_query($conexion,$sql);

if(pg_num_rows($result)>0)
{
  while($row = pg_fetch_assoc($result))
  {
    echo
        "id_valor:".$row['id_valor']."|id_sensor:".$row['id_sensor']."|corriente:"
        .$row['corriente']."|potencia:".$row['potencia']."|diftiempo:".
        $row['diftiempo']."|tiempo:".$row['tiempo'].";";
  }
}



?>
```

De este modo, como se explica posteriormente, en Unity 3D se tiene el mensaje en el formato emitido mediante *echo*. Es decir, en Unity 3D se tienen todos los valores almacenados de un determinado dispositivo. Los campos están separados por "|" mientras que las filas están separados por ";". Así, se pueden obtener los valores de cada campo independientemente.

El esquema que define el procedimiento de interconexión entre la base de datos y Unity3D es:

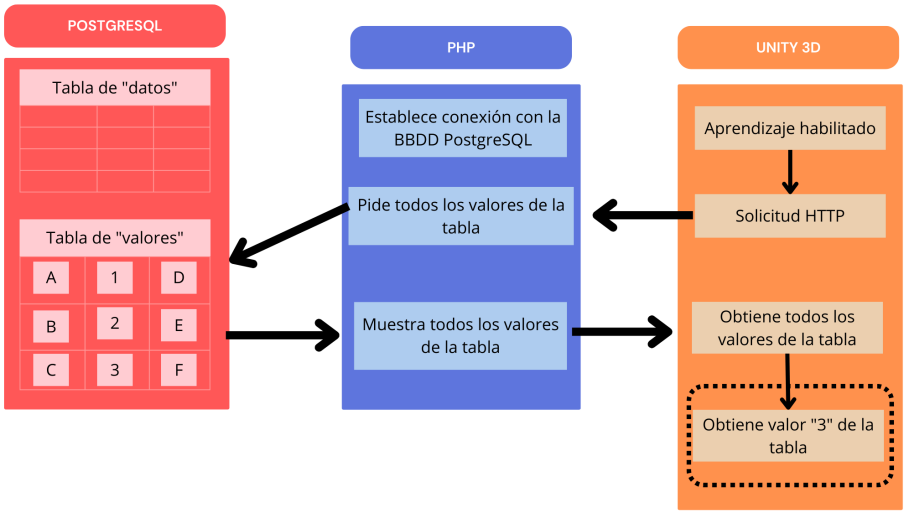


Figura 4.2 Esquema del procedimiento para la interconexión de Base de datos - Unity3D.

La obtención del dato rodeado en línea discontinua representa lo descrito en 4.2.2.

4.2.2 Obtención de datos en Unity 3D

En Unity 3D, partiendo de lo comentado en la sección anterior, se pueden leer todos los valores almacenados en una tabla de la base de datos. Ahora hay que obtener cada valor independientemente, para así poder entrenar el Gemelo Digital en función de dichos datos.

Para ello, se hace uso de lenguaje C#. En el fichero ***CommunicationSupport.cs*** se crea la clase ***SQLParameters***. Ahí se definen distintos parámetros de identificación y conexión que dan "apoyo" al fichero de obtención de datos de la base de datos, denominado *SQL.cs*.

Concretamente, en ***SQLParameters*** se define el número de acelerómetros que habrá (considerando que hay un único SCT), las direcciones de los ficheros PHP de los cuáles se leen los datos almacenados en la base de datos, los distintos tipos de dispositivos (en este caso, sensor de corriente y acelerómetro) y las componentes de las velocidades en cada eje para el acelerómetro.

Código 4.6 Código C# que define la clase SQLParameters incluida en el fichero CommunicationSupport.cs.

```
public class SQLParameters : MonoBehaviour
{
  // IDENTIFICATION AND CONNECTION PARAMETERS
  public const int NumAccelerometer = 1;

  public static string[] PHPUrl =
      {"http://localhost/bbdd_sct_unity_v1/selectData_bbdd_unity_sct.php",
    "http://localhost/bbdd_sct_unity_v1/selectData_bbdd_unity_acel.php"};

  public static string[] SensorType = {"Current", "Accelerometer"};

  public static string[] ComponentsSpeed = {"vel_x:", "vel_y:", "vel_z:"};

}
```

En ***CommunicationSupport.cs*** también se crea la clase ***SQLMethods***, en la cual se definen las estructuras que van a almacenar los datos obtenidos de la base de datos, así como también se define el método mediante el cuál descarga los datos almacenados en la base de datos (***GetValueData***)

Código 4.7 Código C# que define la clase SQLMethods incluida en el fichero CommunicationSupport.cs.

```
public class SQLMethods : MonoBehaviour
{
  // Main structure of the data

  public struct DDBBData
  {
    public bool Valid;

    public StructAccelerometerSensorData[] Accelerometers;

    public StructCurrentSensorData CurrentSensor;
  }

  // Structures defining the types of data to be collected from the DDBB
  public struct StructCurrentSensorData

  {
    public float Power;
```

```
    public float DeltaTime;

  }

  public struct StructAccelerometerSensorData
  {
    public int Id;

    public float[] Speed;

    public string UpdateTime;

  }


  // Method to download the data from the DataBase

  public static string GetValueData(string data, string index)
  {
    string value = data.Substring (data.IndexOf(index) + index.Length);

    if(value.Contains("|"))
    {
      value = value.Remove (value.IndexOf("|"));
    }

    return value;
  }
}
```

Finalmente, se crea el fichero ***SQL.cs*** con el objetivo de conseguir los últimos valores almacenados en la base de datos tanto del acelerómetro como del SCT.

Para ello, se define una estructura tipo *SQLMethods.DDBBData* denominada ***LastData*** y se crea una función tipo "IEnumerator" denominada ***DownloadAllData()***.

La función ***DownloadAllData()*** realiza lo siguiente:

- Envía la solicitud al fichero PHP correspondiente para extraer todos los datos en *DownloadedData* mediante el uso de *UnityWebRequest*.
- En la cadena denominada *UpdateData* se almacenan las filas de *DownloadedData*, ya que toma los valores hasta ";", la cual delimita el final de cada fila.
  Estos dos puntos se implementan en el código de la siguiente manera.

```
    // Send the request to the PHP server to extract the data

    UnityWebRequest DownloadedData =
        UnityWebRequest.Get(SQLParameters.PHPUrl[type]);

    yield return DownloadedData.SendWebRequest();

    // Data processing

    string[] UpdateData = DownloadedData.downloadHandler.text.Split(';');
```

- Luego, se distingue según del dispositivo que se trate, pero el modo de funcionamiento es análogo. Por ejemplo, en el caso del SCT, los datos que interesan obtener son los últimos valores de potencia y diferencia de tiempo almacenados. Mientras que en la cadena *UpdateData* se encuentran todos los datos almacenados en la tabla del SCT.
  Por ejemplo, en *UpdateData* se tiene algo similar a esto, suponiendo que existen los 592 valores anteriores no mostrados y que únicamente se quiere el valor de "potencia":

  id_valor:593|id_sensor:2|corriente:0.068298|potencia:15.708579|diftiempo:0.552|tiempo:2022-09-09 01:13:22.662861;
  id_valor:594|id_sensor:2|corriente:0.069395|potencia:15.960889|diftiempo:0.552|tiempo:2022-09-09 01:13:24.255944;
  id_valor:595|id_sensor:2|corriente:0.068298|potencia:15.708648|diftiempo:0.553|tiempo:2022-09-09 01:13:25.811652;

  El objetivo es conseguir únicamente el valor "15.708648" correspondiente a la potencia. Entonces, se ejecuta, entre otras cosas, una llamada a la función *GetValueData* perteneciente a la clase *SQLMethods*, pasándole como argumentos la última componente de la cadena *UpdateData* y la cadena "potencia:". La función *GetValueData* es la siguiente:

```
public static string GetValueData(string data, string index)
{
  string value = data.Substring (data.IndexOf(index) + index.Length);
  if(value.Contains("|"))
  {
    value = value.Remove (value.IndexOf("|"));
  }
  return value;
}
```

  Se encarga de extraer únicamente el valor que se encuentra a continuación del último caracter indicado en la cadena del segundo argumento. Es decir, en el caso del ejemplo, en *value* se queda con el tramo de la última fila de la cadena que se encuentra a partir de "potencia:". En este caso, en *value* se queda lo siguiente:

  15.708648|diftiempo:0.553|tiempo:2022-09-09 01:13:25.811652;

  Luego, se encarga de eliminar el resto de la fila a partir del caracter "|", por lo que finalmente consigue su objetivo de extraer el valor "15.708648".

- Una vez conseguida la extracción del valor deseado, se convierte al formato correspondiente y se asigna a la correspondiente estructura de ***LastData***. Por ejemplo, en el caso del SCT, se realiza del siguiente modo:

```
    case "Current":
    LastData.CurrentSensor.Power =
        Convert.ToSingle(SQLMethods.GetValueData(UpdateData[UpdateData.Length
        - 2],"potencia:").Replace(".",","));

    LastData.CurrentSensor.DeltaTime =
        Convert.ToSingle(SQLMethods.GetValueData(UpdateData[UpdateData.Length
        - 2],"diftiempo:").Replace(".",","));

    break;
```

- En el ejemplo mostrado, si se desea hacer uso de los valores obtenidos por el SCT, el último valor de potencia que se encuentre almacenado en la base de datos se asigna a la variable *LastData.CurrentSensor.Power*

De este modo, se puede conseguir extraer en Unity 3D el valor almacenado en la base de datos deseado tanto del acelerómetro como del SCT.

El código completo del fichero ***SQL.cs*** se encuentra adjunto en el Anexo II (9)

4.2.3 Interconexión Unity 3D - Base de datos PostgreSQL

En este proyecto también se ha realizado la comunicación de datos desde Unity 3D hacia la base de datos PostgreSQL, aunque no se ha llegado a dar utilidad aplicada al proyecto. Sólo se ha diseñado para datos del SCT, aunque para datos del acelerómetro el diseño es análogo.

En primer lugar, se crea una tabla en la base de datos que solo se dedique a almacenar los datos recibidos, de modo que no se mezclen con los emitidos. Dicha tabla se crea mediante el siguiente código SQL:

```
CREATE TABLE tabla_receptora_unity_sctv1(
ID_Valor_unity SERIAL PRIMARY KEY,
ID_Sensor_unity INT,
CONSTRAINT fk_Sensor FOREIGN KEY (ID_Sensor_unity) REFERENCES
    tabla_datos_sctv1(ID_Sensor),
Corriente_unity FLOAT,
Potencia_unity FLOAT,
Tiempo_unity TIMESTAMP
);
```

Se desarrolla un ejemplo de insertar los campos *ID_Sensor*, *Corriente* y *Potencia* en la tabla denominada receptora en la base de datos.

Para ello, mediante el uso de las herramientas de Unity tipo botón y tipo campo de entrada de texto se ha creado la siguiente escena en Unity 3D, de modo que manualmente se introduce texto en los campos comentados.

Figura 4.3 Ejemplo de escena para introducir datos desde Unity3D a PostgreSQL.

El texto que se introduce, posteriormente se convierte al tipo que se haya indicado en la tabla creada en la base de datos, es decir, se convierte a entero en caso del campo *ID_Sensor* y a flotante en caso de los campos *Corriente* y *Potencia*. Además, en caso del *ID_Sensor* se ha creado de modo que cumpla con la llave foránea referenciada a la tabla de datos del SCT siguiendo la misma idea comentada en 3.3.1, de modo que si se intenta introducir un *ID_Sensor* no definido en su tabla de datos correspondiente, saltará un mensaje de error y no realizará dicha operación.

Para realizar el ingreso de datos en la base de datos se crea el código denominado *UnityToPostgreSQL.cs* asociado a un objeto vacío en Unity 3D. El código completo se encuentra adjunto en el Anexo II (Capítulo 9).

Los datos son introducidos en la base de datos mediante la ejecución de un fichero PHP, del mismo modo que se realiza para la comunicación Arduino-BBDD.

El código PHP es:

Código 4.8 Código PHP para la inserción de datos de Unity3D en la base de datos PostgreSQL.

```
<?php

include('connection_bbdd_unity_sct.php');

$id_sensor = $_GET['id_sensor'];
$corriente = $_GET['corriente'];
$potencia = $_GET['potencia'];

if(!$conexion){
  echo "400"; //Error de conexion con base de datos
}else{

  $sql = "INSERT INTO tabla_receptora_unity_sctv1 (ID_Sensor_unity,
      Corriente_unity, Potencia_unity, Tiempo_unity) VALUES
      ('$id_sensor','$corriente','$potencia',CURRENT_TIMESTAMP)";

  $resultado = pg_query($conexion, $sql);

  echo "200"; //Valores introducidos correctamente en la base de datos.
}

?>
```

5 Tratamiento de datos

En este capítulo se presenta el tratamiento de datos que se ha realizado para obtener los datos necesarios para el Gemelo Digital. El procesamiento se ha realizado en los microcontroladores para restar procesamiento al Gemelo Digital. Es necesario distinguir entre los dos tipos de sensores (acelerómetro y SCT) con los que cuenta el presente proyecto.

Además, también se presentan las pruebas realizadas y análisis de los resultados obtenidos en el proceso de tratamiento de datos.

5.1 Tratamiento de datos del acelerómetro

Inicialmente, tras realizar varias pruebas y observar sus resultados, se decidió realizar un filtrado de las medidas de aceleraciones. Debido a que proporcionaba valores claramente erróneos para la situación en que se realizaron las pruebas. Por tanto, tras el filtrado se realizaba el cálculo de las velocidad a partir de las aceleraciones filtradas y las diferencias de tiempo entre medidas.

Más adelante, tras observar que la dinámica no funcionaba como era esperado, se decidió modificar el algoritmo, de modo que filtrase velocidades en vez de aceleraciones, consiguiendo así mejores resultados.

Una vez se alcanza dicha conclusión, se realiza un nuevo algoritmo que se comenta posteriormente. Antes se definen los parámetros a obtener, en cada eje, para conseguir un correcto funcionamiento. Estos son:

- **PermissibleDeviationRange**: Factor que multiplicado por la desviación media indica el valor máximo de desviación admisible. Se utiliza en el filtrado. Valores de desviación mayores o igual a dicho valor máximo provocan el descarte de su correspondiente valor de velocidad. Cuanto más próximo a '1'(superior a '1') sea, más estricto será el filtrado.
- **OffsetSpeedDifference**: Parámetro encargado de realizar una compensación de la diferencia de velocidad, calculada a partir de la aceleración y el tiempo, de modo que sea lo más próximo posible a cero. Es necesario crear este parámetro ya que al mantener quieto el acelerómetro se observa que no salen valores próximos a cero como debería y suelen estar siempre en torno al mismo valor, por lo que aplicando un "offset" se soluciona dicho problema.
- **MovementRangeSpeedDifference**: Parámetro que indica el valor de diferencia de velocidad a partir del cual se considera que el acelerómetro está en movimiento. Si el valor de diferencia de velocidad es mayor que este parámetro, se realiza el cálculo de la velocidad a partir de la velocidad en el instante anterior, por lo que la velocidad será distinta de cero. Es necesario crear este parámetro para que cuando el acelerómetro no esté en movimiento, no se realice el cálculo de la velocidad, ya que se incrementaría cuando no debe.
- **DeltaCinta**: Parámetro que indica el valor a partir del que se considera que ha habido un cambio brusco de velocidad. Esto ocurrirá cuando el acelerómetro cambie de cinta, ya que experimenta una detención y posteriormente continúa el movimiento.

El procedimiento a seguir para seleccionar los valores de estos parámetros se explicará más adelante aplicado a las gráficas resultantes.

También cabe definir la estructura creada denominada "DataStructure". La función que realiza el filtrado recibe esta estructura con medidas de velocidades y las filtra devolviendo como resultado una estructura del mismo tipo con valores filtrados.

Se crea un vector de estructuras "DataStructure", de dimensión igual a '3', denominado "Data". La posición '0'del vector corresponde al eje 'x'del acelerómetro, mientras que las posiciones '1'y '2'corresponden a los ejes 'y'y 'z'respectivamente.

Esta estructura está compuesta de los siguientes campos:

Código 5.1 Declaración de estructura tipo DataStructure.

```
struct DataStructure
{
  float Accelerations[MaxNumMeasures];   // ( m/s^2 )
  float TimeDifferences[MaxNumMeasures]; // ( s )
  bool Valids[MaxNumMeasures-1];         // Valid: '1'. Not valid: '0'.
  int NumberValids;                      // Number of valid speed values.
  float Speeds[MaxNumMeasures-1];        // (m/s)
};
```

- **MaxNumMeasures**: Número de medidas de aceleraciones que se recogen para realizar los cálculos.
- **Accelerations**: Vector flotante de MaxNumMeasures componentes, en el cual se van a almacenar las medidas de aceleración obtenidas a partir del acelerómetro, en m/s^2.
- **TimeDifferences**: Vector flotante de MaxNumMeasures componentes, en el cual se van a almacenar las diferencias de tiempos entre medidas de aceleración, en segundos.
- **Valids**: Vector booleano de MaxNumMeasures-1 componentes, en el cual se van a almacenar '1'si el valor de velocidad correspondiente es "válido" y '0én caso de que sea "no válido". El procedimiento para conocer si un valor de velocidad es válido o no se explicará más adelante.
- **NumberValids**: Entero que almacena el número de valores de velocidades válidas. Por tanto, $NumberValids$ será igual a tantos '1'como haya en el vector $Vaslids$.
- **Speeds**: Vector flotante de MaxNumMeasures-1 componentes, en el cual se van a almacenar las velocidades.

A continuación, se definen los subprocesos que conforman el procedimiento a seguir para el tratamiento de datos en el acelerómetro. Cabe destacar que cada subproceso se realiza para los 3 ejes del acelerómetro.

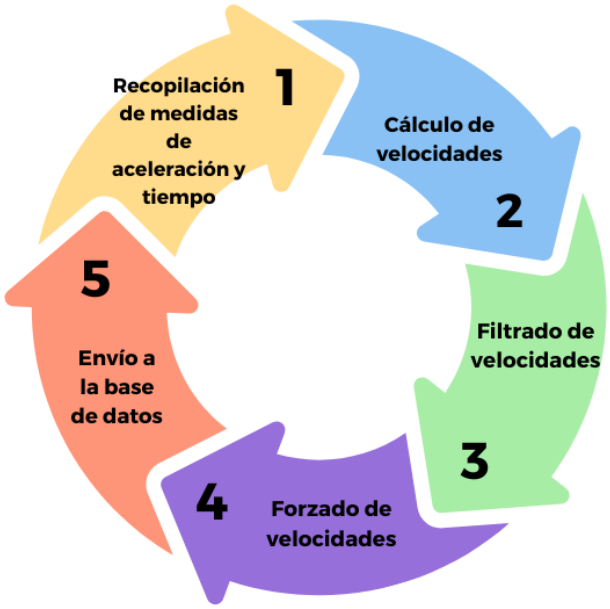


Figura 5.1 Proceso del tratamiento de datos acelerómetro.

1. **Recopilación de medidas de aceleración y tiempo**
   Cada "Tm" milisegundos, realiza una medida de aceleración y de diferencia de tiempo entre medidas. Ambas medidas son almacenadas en la estructura "Data". El proceso de recopilación finaliza cuando se alcanzan las "MaxNumMeasures" medidas.

2. **Cálculo de velocidades**
   Una vez finalizado el proceso de recopilación, se calculan las diferencias de velocidades a partir de los valores de aceleración y de tiempo recopilados como se indica en (2.6).
   En caso de que se considere que el acelerómetro se encuentra en movimiento, se calculan los valores de velocidad a partir de la diferencia de velocidad calculada y de la velocidad en el instante anterior.
   Si el acelerómetro no está en movimiento, los valores de velocidad calculados pueden ser nulos o constantes en el último valor que calculó cuando estaba en movimiento.
   En un caso u otro, el valor de velocidad calculado en cada instante es almacenado en la estructura "Data". Habrá "MaxNumMeasures-1" valores de velocidad.

3. **Filtrado de velocidades**
   El funcionamiento y procedimiento del filtro se explica en detalle en 5.1.2. Una vez salga del filtro, recorre únicamente las medidas válidas de velocidad para posteriormente calcular su media. A partir de este punto, se trabaja con la media de las velocidades filtradas.

4. **Forzado de velocidades**
   Inicialmente, no se contemplaba la realización de este subproceso, ya que el objetivo era obtener el valor de velocidad del acelerómetro en cada instante. Al realizar pruebas, se detectaron que los valores de velocidad no eran reales, y por tanto, no eran valores fiables. Es por ello que se decidió crear este subproceso, con el objetivo de detectar únicamente si el acelerómetro se encuentra en movimiento y en qué cinta transportadora se encuentra.
   De este modo, el resultado final de todo el proceso de tratamiento de datos será un valor distinto de cero si el acelerómetro está en movimiento y un valor igual a cero en caso de que no esté en movimiento, todo ello aplicado a cada eje como se explica en 5.2.

5. **Envío a la base de datos**
   El envío a la base de datos se realiza con la ayuda de PHP como se describe en 4.1. Para ello, es necesario que el microcontrolador Arduino se encuentre conectado a una red WiFi, ya que se realiza el envío mediante solicitudes HTTP.

El valor que se envía a la base de datos es el valor de velocidad forzado que se comenta en el punto anterior.
Una vez finalizado este subproceso, vuelve al primer punto, iniciando la recopilación de datos hasta alcanzar "MaxNumMeasures" nuevamente.

El código completo y el diagrama de flujo se encuentran adjuntos en el Anexo I (8)

5.1.1 Explicación detallada del código de tratamiento de datos del acelerómetro

Para la obtención de datos del acelerómetro ADXL335 se ha considerado el uso de la librería de Arduino denominada "ADXL335.h" [15] para así facilitar el código, como se ha comentado anteriormente. Previamente hay que realizar una calibración del dispositivo siguiendo los pasos descritos en dicha librería, tal y como fue descrito en el punto 2.3 del presente proyecto. Tras realizar la calibración, la aceleración en cada eje (g) se obtiene con la orden $accelerometer.getAcceleration(\&ax, \&ay, \&az)$; siendo "$accelerometer$" una instancia creada de clase $ADXL335$ y siendo ax, ay y az flotantes previamente declarados donde se guardan los valores de aceleraciones determinados.

En primer lugar, sucede el subproceso de recopilación de medidas, en el que cada "Tm" milisegundos realiza lo siguiente:

1. Obtiene una medida de aceleración (en g) del acelerómetro para cada eje y la almacena en el vector "**a**". Siendo "**a**" un vector de 3 componentes, uno para cada eje.
2. Para cada eje, convierte el valor de aceleración correspondiente en m/s^2 y lo almacena en el vector "**Accelerations**" perteneciente a la estructura **Data** comentada previamente.
3. También, para cada eje, en el vector "**TimeDifferences**" de la estructura "**Data**" almacena la diferencia de tiempo entre medidas de aceleraciones en segundos.
4. Además, para cada eje, escribe un $'1'$ en el vector "**Valids**" de la estructura "**Data**". Esto se realiza ya que, inicialmente, todas las medidas son válidas y será posteriormente cuando se descarten o no siguiendo el criterio explicado en 5.1.2.
5. Incrementa el número de medidas de aceleraciones obtenidas, definido en el código por la variable "**NumMeasures**".

Código 5.2 Subproceso de recopilación de medidas.

```
float ax, ay, az;

if (millis() > vtime + Tm)
{
  // Get acceleration measures

  accelerometer.getAcceleration(&ax, &ay, &az);
  float a[3] = { ax, ay, az};

  for ( int k = 0; k < 1; k++ )
  {
    a[k] = a[k] * 9.81;                    // ( m/s^2 )

    // Store acceleration measures

    Data[k].Accelerations[NumMeasures] = a[k];

    // Get time measures

    if(NumMeasures == 0)
    {
```

```
      Data[k].TimeDifferences[NumMeasures] = 2.00/1000.0; [s]
    }
    else
    {
      Data[k].TimeDifferences[NumMeasures] = (millis() - vtime)/1000.0; // [s]

    }


    // Initially, all values are valid
    if(NumMeasures < (MaxNumMeasures - 1))
    {
      Data[k].Valids[NumMeasures] = true;
    }

  }

  NumMeasures++;

  vtime = millis();

}
```

Cuando se dé el caso de que "**NumMeasures**" es igual a "**MaxNumMeasures**", comienzan los subprocesos de cálculo de velocidades, filtrado, forzado y envío de datos a la base de datos.

Se realizan las siguiente operaciones para cada eje:

La variable "**NumberValids**", perteneciente a la estructura tipo "**DataStructure**", se define igual al número de medidas, ya que inicialmente todas las medidas son válidas y posteriormente al realizar un filtrado se irán descartando medidas no válidas.

```
    // All measures are collected

if (NumMeasures == MaxNumMeasures)
{
  DataStructure FinalsData[3];

  // Calculate the axis speed. | k=0 : "X" axis | k=1 : "Y" axis | k=2 : "Z" axis |

  for (int k = 0; k < 2; k++)
  {
    //Initially all measures are valids
    Data[k].NumberValids = MaxNumMeasures;

    ...
    ...
  }
  ...
  ...
}
```

Recorriendo todas las medidas de aceleraciones y diferencias de tiempo obtenidas previamente, se calculan las diferencias de velocidades a partir de la integral en términos discretos de las aceleraciones respecto al tiempo, tal como indica (2.6). Cada diferencia de velocidad se tiene en la variable local "**SpeedDifference**".

El vector de velocidades tiene MaxNumMeasures-1 componentes mientras que los de aceleraciones y tiempos tienen MaxNumMeasures componentes, ya que cada velocidad se calcula a partir de dos aceleraciones.

Al término "**SpeedDifference**" se le suma una compensación para conseguir que sea lo más próximo posible a '0'. La compensación es la denominada "**OffsetSpeedDifference**", explicada anteriormente.

Si el valor absoluto de "**SpeedDifference**" es mayor al parámetro "**MovementRangeSpeedDifference**" explicado previamente, significa que el acelerómetro ha experimentado una aceleración, es decir, ha comenzado un movimiento.

En ese caso, calcula el valor de la velocidad ("**Speedk**") a partir de la velocidad en el instante anterior y el valor de "**SpeedDifference**" comentado.

La primera vez que esté quieto, es decir, antes de arrancar las cintas transportadoras, dicha condición no se dará si está correctamente calibrado, por tanto, la velocidad será nula.

Tampoco se dará dicha condición cuando el acelerómetro detenga el movimiento, por tanto, el valor de velocidad se quedará fijo en un valor.

Los valores de velocidad se almacenan en el vector "**Speeds**" perteneciente a la estructura "**Data**".

Código 5.3 Subproceso de cálculo de velocidades.

```
...
...
// Declaration of vector where the speed calculations are to be stored

float SpeedDifference = 0;

// Do the integral calculation to obtain the speed
for (int i = 1; i < Data[k].NumberValids; i++)
{

  SpeedDifference = ((Data[k].Accelerations[i] + Data[k].Accelerations[i -
      1]) / 2) * (Data[k].TimeDifferences[i] + Data[k].TimeDifferences[i -
      1])/2;

  SpeedDifference = OffsetSpeedDifference[k] + SpeedDifference;
      //SpeedDifference must be displayed (where Offset = 0) to calculate the
      required offset.

  if (fabs(SpeedDifference) >  MovementRangeSpeedDifference[k])
  {

    Speedk[k] = Speedk1[k] + SpeedDifference;

  }

  Data[k].Speeds[i-1] = Speedk[k];

  Speedk1[k] = Speedk[k];

}
...
...
```

Cuando se hayan calculado todas las velocidades, se realiza el filtrado de las mismas, obteniendo una estructura de tipo "**DataStructure**" denominada "**FinalsData**". Dicha estructura contendrá las velocidades que han pasado el filtro y por tanto son admitidas.

Las velocidades ya filtradas, es decir, las almacenadas en el vector "**Speeds**" perteneciente a la nueva estructura llamada "**FinalsData**", se pasan al vector denominado "**FinalsSpeeds**" para así conseguir más simplicidad a la hora de trabajar con el vector.

Posteriormente, se obtiene la media de dichas velocidades filtradas, para así trabajar con un único valor. Dicha media se guarda en la variable "**MeanFinalsSpeedk**", mientras que la media de velocidades filtradas en la ejecución anterior se encuentra en la variable "**MeanFinalsSpeedsk1**".

Código 5.4 Subproceso de filtrado y cálculo de la media de velocidades filtradas.

```
...
...
float FinalsSpeedsSum = 0;

//The recursive function filter is called

FinalsData[k] = RecursiveFilter(Data[k], k);

float FinalsSpeeds[FinalsData[k].NumberValids];

int contValidsSpeeds = 0;

//The velocities that have passed the filter are stored in the vector
    "FinalsSpeeds"

for (int i = 0; i < (MaxNumMeasures-1); i++)
{
  if((FinalsData[k].Valids[i] == 1) && (contValidsSpeeds <
      FinalsData[k].NumberValids))
  {
    FinalsSpeeds[contValidsSpeeds] = FinalsData[k].Speeds[i];
    FinalsSpeedsSum += FinalsSpeeds[contValidsSpeeds];
    contValidsSpeeds = contValidsSpeeds + 1;

  }

}

//The average of the speeds that have passed the filter is averaged.

MeanFinalsSpeedk[k] = FinalsSpeedsSum/FinalsData[k].NumberValids;
...
...
```

Si el valor absoluto de la diferencia de medias de velocidades filtradas en la ejecución actual y la ejecución anterior es mayor al parámetro "**DeltaCinta**" comentado anteriormente, significa que se considera que el acelerómetro ha experimentado un incremento o decremento importante de velocidad. Esto ocurre cuando el acelerómetro finaliza su recorrido en una cinta o inicia el recorrido en otra cinta.

Según está implementado el código, cuando el acelerómetro se detiene, los valores de velocidad no "bajan" a cero, si no que se mantiene en el último valor de velocidad que tenía. Es por ello que se implementa un subproceso de forzado de velocidades, tanto para el inicio del movimiento como para el final, ya que como se ha comentado anteriormente, el objetivo es conocer si la cinta está en movimiento o no y en qué cinta se encuentra. El forzado de velocidad es el que permite cumplir ese objetivo.

En caso de que ocurra la situación anterior, hay que distinguir entre incremento y decremento de velocidad. Para ello hay dos casos:

- Si la media de velocidades filtradas en la ejecución anterior (**MeanFinalsSpeedsk1**) es mayor que la media de velocidades filtradas en la ejecución actual (**MeanFinalsSpeedk**) significa que el acelerómetro experimenta un decremento de velocidad. Por tanto, se fuerza la media de velocidades (**MeanFinalsSpeedk**), que al ser forzada pasa a llamarse **MeanFinalsSpeedkForced,** al valor **Vcero**, ya que se considera que el acelerómetro se ha detenido.
  Además, la variable **FlagForced** toma el valor '0'. Su uso se comentará posteriormente.
- En caso contrario, es decir, si la media de velocidades filtradas en la ejecución anterior (**MeanFinalsSpeedsk1**) es menor que la media de velocidades filtradas en la ejecución actual (**MeanFinalsSpeedsk**) significa que el acelerómetro experimenta un incremento de velocidad.
  Por tanto, la velocidad de la cinta (**ConveyorBeltSpeed**) será la calculada anteriormente como **MeanFinalsSpeedsk**. El valor de velocidad forzado (**MeanFinalsSpeedkForced**) también toma este valor.
  Además, la variable **FlagForced** toma el valor '1'. Su uso se comentará posteriormente.

En caso que no se de la situación comentada anteriormente, es decir, que no se considere que el acelerómetro ha incrementado o decrementado su velocidad considerablemente, ocurre lo siguiente:

- Si la variable **FlagForced** tiene valor '1', el valor final de velocidad (**MeanFinalsSpeedkForced**) se fuerza al valor **ConveyorBeltSpeed**, el cual a su vez es la media de velocidades filtradas calculada anteriormente. Por tanto, en caso de que **FlagForced** sea '1', la velocidad final será la calculada antes en **MeanFinalsSpeedk**.
- Si la variable **FlagForced** tiene valor '0', el valor final de velocidad (**MeanFinalsSpeedkForced**) se fuerza al valor **Vcero**.

Código 5.5 Subproceso de forzado de velocidades finales.

```
...
...
  //"DeltaCinta" is obtained by observing the difference of the two values  when
      the accelerometer starts or ends the movement.
  //"DeltaCinta" should be slightly less than this difference.

  if (fabs(MeanFinalsSpeedk[k] - MeanFinalsSpeedk1[k]) > DeltaCinta[k])
  {

    //If the movement is starting
    if(MeanFinalsSpeedk1[k] > MeanFinalsSpeedk[k])
    {
      MeanFinalsSpeedkForced[k] = Vcero;

      FlagForced[k] = 0;

    }
    else //If the movement is ending
    {

      ConveyorBeltSpeed[k] = MeanFinalsSpeedk[k];

      MeanFinalsSpeedkForced[k] = MeanFinalsSpeedk[k];

      FlagForced[k] = 1;

    }

  }
  else
  {
```

```
      if(FlagForced[k])
      {
        MeanFinalsSpeedkForced[k] = ConveyorBeltSpeed[k];
      }
      else
      {
        MeanFinalsSpeedkForced[k] = Vcero;
      }

    }

    MeanFinalsSpeedk1[k] = MeanFinalsSpeedk[k];

  }
  ...
  ...
```

Finalmente, la velocidad en cada eje será la que indique **MeanFinalsSpeedkForced** y se realiza el envío de dicho valor a la base de datos.
El envío a la base de datos se detalla en el capítulo 4.1.

5.1.2 Filtro de velocidades

El filtrado de las velocidades se realiza al llamar a la función "*RecursiveFilter*", la cual tiene como argumentos una estructura de entrada de tipo "*DataStructure*" El filtrado de las velocidades se realiza mediante el uso de una función recursiva, la cual recibe una estructura de datos de entrada y devuelve una estructura de salida del mismo tipo pero con valores filtrados, y así sucesivamente hasta cumplir determinadas condiciones de fin de recursividad que se comentan más adelante.

Se realiza de este modo para conseguir un mejor filtrado de las medidas de velocidad, ya que así se consiguen eliminar la mayoría de valores irreales que afectan al funcionamiento del algoritmo.

El uso de la recursividad también presenta inconvenientes como se verá al analizar los resultados más adelante.

En cada ejecución del filtro se realizan las siguientes operaciones:

1. Realiza la media de las medidas de velocidades válidas. Es decir, aquellas cuyo valor correspondiente del vector "*Valids*" sea '1'. En la primera ejecución, todas las medidas serán válidas y según avance el filtrado se irán descartando medidas.

2. Realiza la media de la desviación de los mismos valores de velocidades válidas del punto anterior.

3. Recorre todas las medidas de velocidades y si el valor absoluto de la desviación de cada valor es mayor a "*PermissibleDeviationRange*" veces la media de la desviación y su correspondiente valor en "*Valids*" es '1', ese valor en "*Valids*" pasa a ser '0'.
   De modo que esa medida queda descartada. De esta forma, se decrementa el número de medidas válidas, denominado como el campo "*NumberValids*", tantas unidades como medidas hayan sido descartadas. Por tanto en "*NumberValids*" se tiene el número de medidas válidas que hay en la estructura de salida.

4. Finalmente, la función recursiva vuelve a ejecutarse si el número de valores de velocidades válidas es distinto al número de valores de velocidades válidas de la ejecución anterior. Además, la función recursiva no se ejecutará de nuevo si no hay más de un valor válido de aceleración.

El esquema que define el funcionamiento del filtro es:

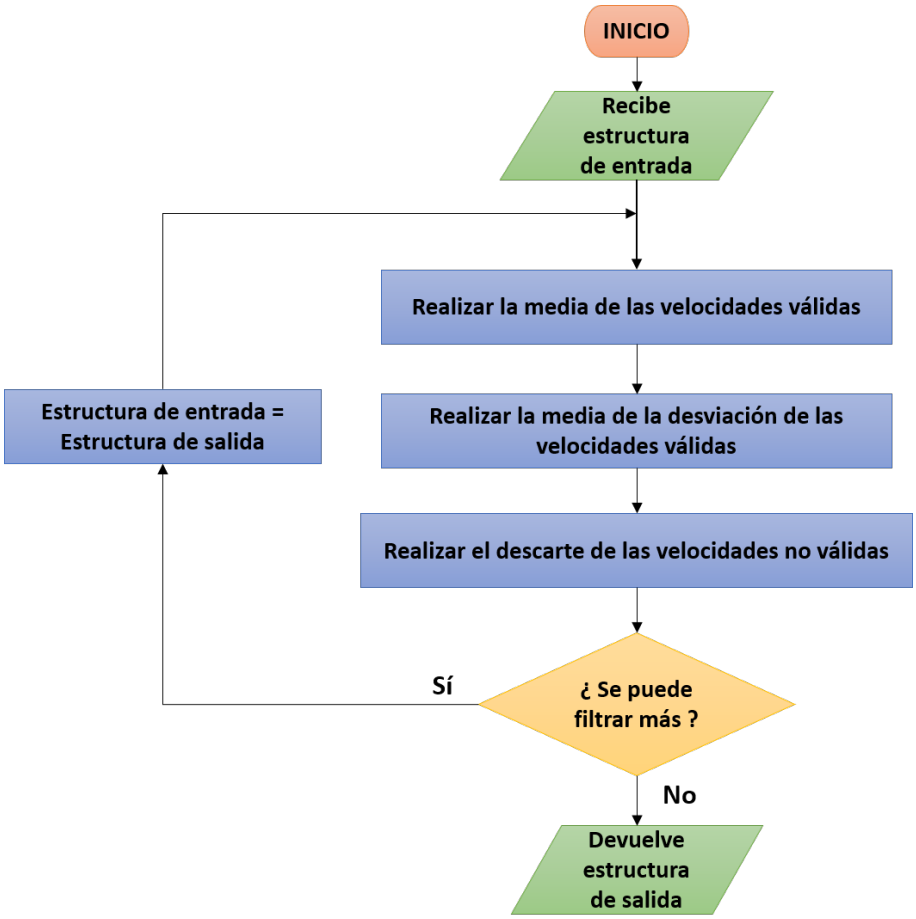


Figura 5.2 Esquema del filtro para tratamiento de datos del acelerómetro.

Ejemplo de filtrado de velocidades

Para comprender mejor el funcionamiento del filtro se expone el siguiente ejemplo práctico. Supóngase que el número máximo de medidas para ejecutar el filtrado son 12 medidas. También se supone que el parámetro "*PermissibleDeviationRange*" toma el valor 1.5.

- Se suponen las medidas de velocidad. Como se ha comentado anteriormente, en la primera ejecución todos los valores son válidos, por tanto, todos los valores del vector "*Valids*" serán '1'. Por lo que "*NumberValids*" es 12.

$$Speeds = [2.33, 4.81, 2.41, 2.14, 2.41, 4.89, 7.94, 2.78, 2.55, 2.41, 1.95, 2.41]$$
$$Valids = [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]$$
$$NumberValids = 12$$

- Se realiza la media de aquellas medidas cuyo correspondiente valor en "*Valids*" sea '1'. En este caso, se realiza la media de todas las medidas. La media es: 3.2525.
- Se realiza la media de la desviación de dichos valores, esta es: 1.3137.
- Las medidas cuya desviación (en valor absoluto) sea superior al valor 1.9706 (producto de "*PermissibleDeviationRange*" y la media de la desviación) y cuya componente en "*Valids*" sea '1', quedan descartadas. Por tanto, observando los valores que toman las componentes de "*Valids*", la medida de velocidad correspondiente al índice 7 queda descartada. La medida descartada será 7.94. "*NumberValids*" ahora valdrá 11. Al haber cambiado el número de medidas validas respecto a la

ejecución anterior (12), se ejecuta de nuevo la función siendo ahora el número de medidas válidas igual a 11 y el vector "*Valids*" el comentado anteriormente.

$$Speeds = [2.33, 4.81, 2.41, 2.14, 2.41, 4.89, 7.94, 2.78, 2.55, 2.41, 1.95, 2.41]$$
$$Valids = [1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1]$$
$$NumberValids = 11$$

- Realizando lo mismo cada iteración, el número de medidas válidas va disminuyendo. Toma los valores 9, 6 y 4. Siendo 4 el que provoca la salida del filtro, ya que no se pueden filtrar más los valores. Finalmente, cuando acabe el filtro se tendrán los siguientes resultados:

$$Speeds = [2.33, 4.81, 2.41, 2.14, 2.41, 4.89, 7.94, 2.78, 2.55, 2.41, 1.95, 2.41]$$
$$Valids = [0, 0, 1, 0, 1, 0, 0, 0, 0, 1, 0, 1]$$
$$NumberValids = 4$$

  Por tanto, las medidas válidas son: $[2.41, 2.41, 2.41, 2.41]$

5.2 Pruebas realizadas y análisis de resultados del tratamiento de datos del acelerómetro

Se han realizado diversas pruebas sobre la célula de fabricación situada en el laboratorio de la Escuela Técnica Superior de Ingeniería (Universidad de Sevilla).

El objetivo de estas pruebas es comprobar si el funcionamiento del sistema es el esperado o no y analizar los resultados obtenidos. Los resultados que se analizan en este punto son aquellos correspondientes al tratamiento de datos del acelerómetro, comentado en el punto anterior.

Para realizar estas pruebas, el acelerómetro ADXL335 se sitúa sobre una bandeja que recorre las cintas transportadoras.

La célula de fabricación está conformada por cuatro cintas transportadoras: dos cintas largas y dos cintas cortas. El acelerómetro se va a situar sobre la bandeja de modo que cuando recorre una cinta larga, se mueve en el eje "X" del acelerómetro, por lo que los valores de velocidad correspondientes al eje "Y" deberían ser nulos. En cambio, cuando recorre una cinta corta, se mueve en el eje "Y" del acelerómetro, por lo que los valores de velocidad correspondientes al eje "X" deberían ser nulos.

Para comprender mejor lo que se explica, se presenta la Figura 5.3. Debido a limitaciones de movimiento, la zona de realización de las pruebas es la rodeada en rojo. Para el resto de la planta es análogo.

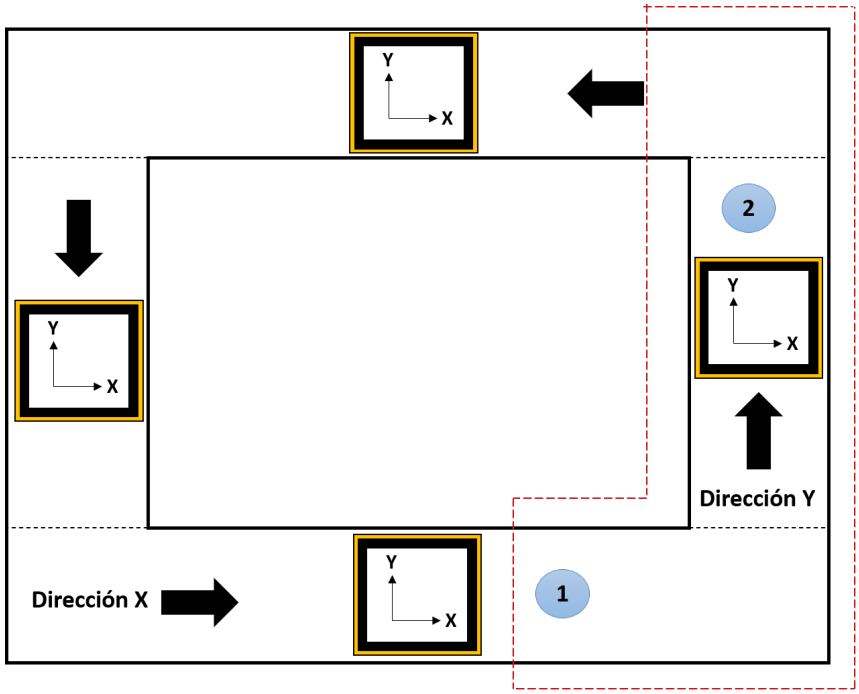


Figura 5.3 Orientación del acelerómetro sobre las cintas transportadoras.

Debido a diversas limitaciones que presenta el microcontrolador Arduino (se detallan en el capítulo 7), se decide recopilar datos crudos de aceleraciones, tanto en la cinta larga como en la corta, para visualizar las gráficas y analizar el comportamiento con la ayuda de Matlab.

El tratamiento en Matlab de los datos recopilados es idéntico al realizado en Arduino.

Todas las pruebas se han realizado para **MaxNumMeasures** igual a 20 muestras.

El valor del parámetro PermissibleDeviationRange para las pruebas realizadas es de 1.5. Cuanto más próximo sea el valor a 1 (por encima), más estricto será el filtro.

Por tanto, aplicando los subprocesos descritos en la Figura 5.1, se tienen las siguientes gráficas aplicadas a la **dirección X**, es decir, cuando el acelerómetro recorre la **cinta larga** (marcada con un "1" en la Figura 5.3):

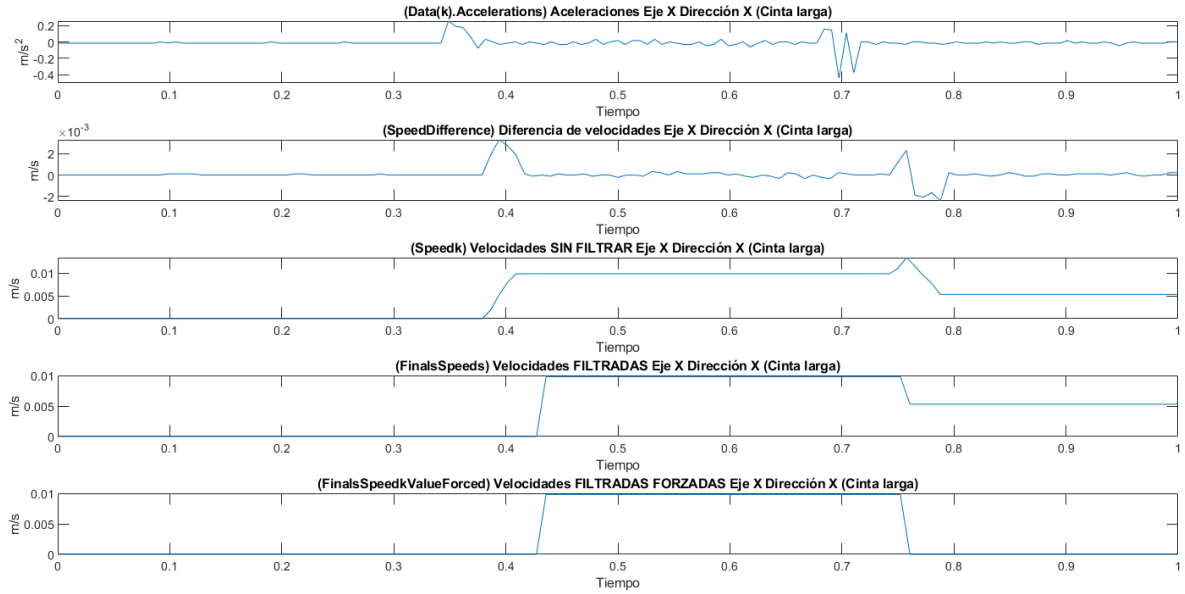


Figura 5.4 Gráficas del tratamiento de datos del acelerómetro en dirección "X" eje "X".

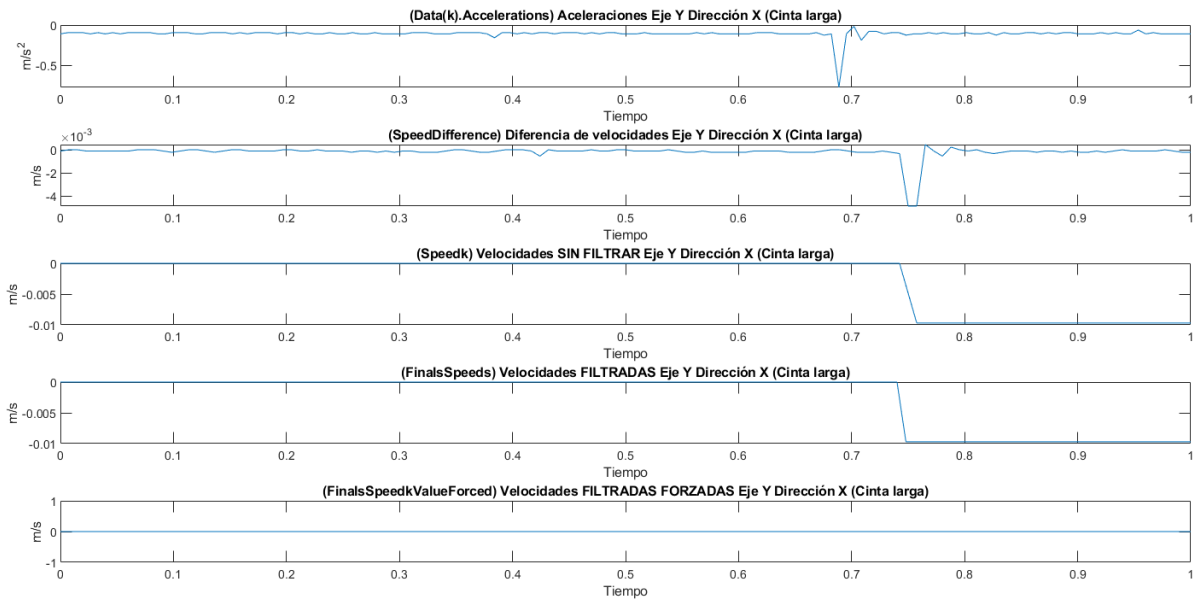


Figura 5.5 Gráficas del tratamiento de datos del acelerómetro en dirección "X" eje "Y".

Como se puede observar, para la dirección "X", es decir, cinta larga, el tratamiento de datos se realiza correctamente, ya que las velocidades finales forzadas en el eje "X" son distintas a cero cuando el acelerómetro está en movimiento. Mientras que las velocidades finales forzadas en el eje "Y" son iguales a cero durante todo el recorrido por la cinta larga.

Aplicando los subprocesos descritos en la Figura 5.1, se tienen las siguientes gráficas aplicadas a la **dirección Y**, es decir, cuando el acelerómetro recorre la **cinta corta** (marcada con un "2" en la Figura 5.3):

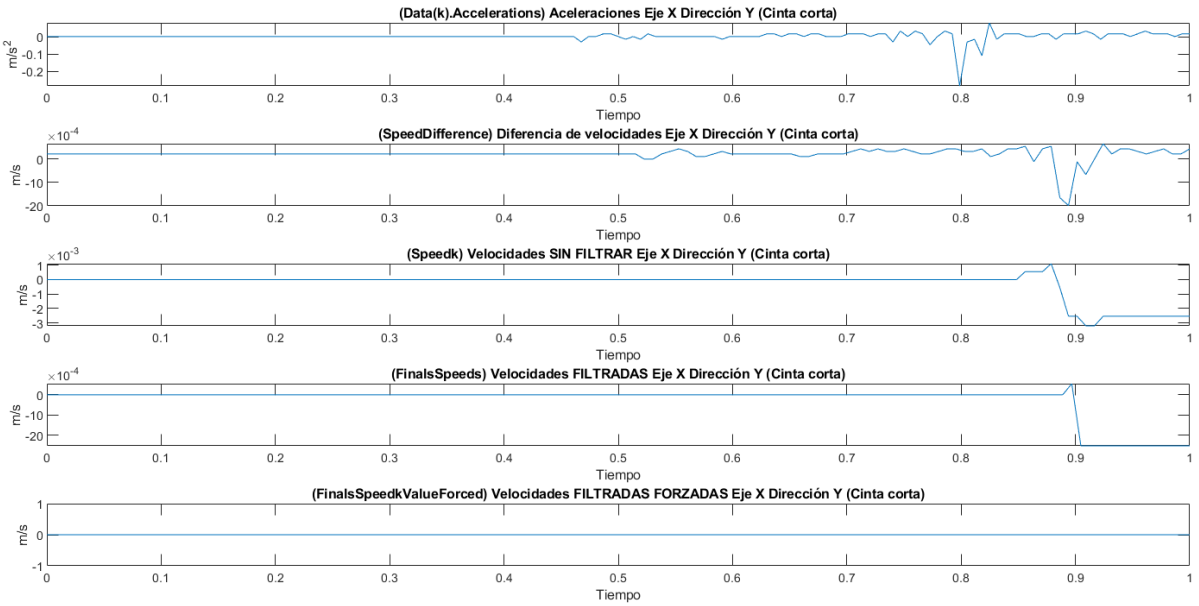


Figura 5.6 Gráficas del tratamiento de datos del acelerómetro en dirección "Y" eje "X".

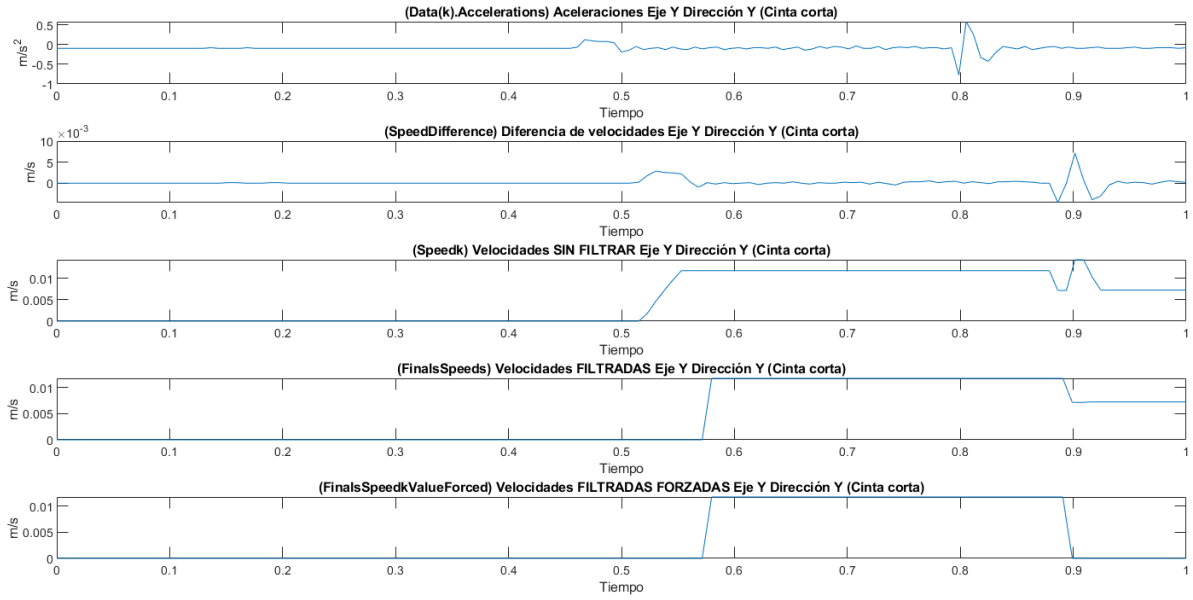


Figura 5.7 Gráficas del tratamiento de datos del acelerómetro en dirección "Y" eje "Y".

Como se puede observar, para la dirección "Y", es decir, cinta corta, el tratamiento de datos se realiza correctamente, ya que las velocidades finales forzadas en el eje "Y" son distintas a cero cuando el acelerómetro está en movimiento. Mientras que las velocidades finales forzadas en el eje "X" son iguales a cero durante todo el recorrido por la cinta corta.

Los valores de los parámetros creados, para el **eje "X"** (tanto para dirección "X" como dirección "Y"), son:

- **OffsetSpeedDifferenceX = 0.0002028719**

  La elección de ese valor se realiza del siguiente modo:

  – Inicialmente, todos los parámetros son cero. Al observar los valores de "SpeedDifference" cuando el acelerómetro está detenido, se observa que el valor más probable es "-0.0002028719", mientras que si fuese ideal debería ser cero.

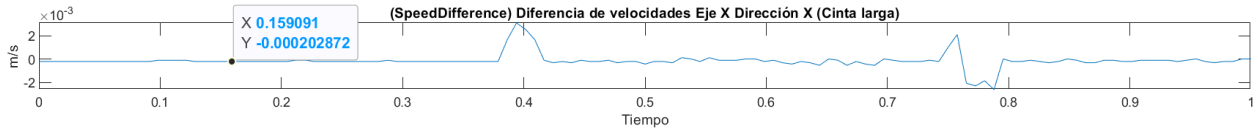


Figura 5.8 Gráfica de SpeedDifference cuando OffsetSpeedDifferenceX = 0.

  – Por tanto, al tomar "OffsetSpeedDifferenceX" el valor "0.0002028719", el valor de "SpeedDifference" cuando el acelerómetro está detenido pasa a ser prácticamente cero.

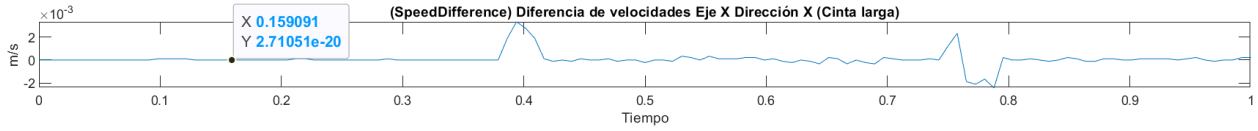


Figura 5.9 Gráfica de SpeedDifference cuando OffsetSpeedDifferenceX = 0.0002028719.

- **MovementRangeSpeedDifference = 0.0005**

  La elección de ese valor se realiza del siguiente modo:

– Una vez definido el valor de "OffsetSpeedDifferenceX", hay que definir el valor de "MovementRangeSpeedDifferenceX". Al observar los valores de "Speedk", se detecta que dicho valor aumenta cuando el acelerómetro está detenido, algo que no debería ocurrir.

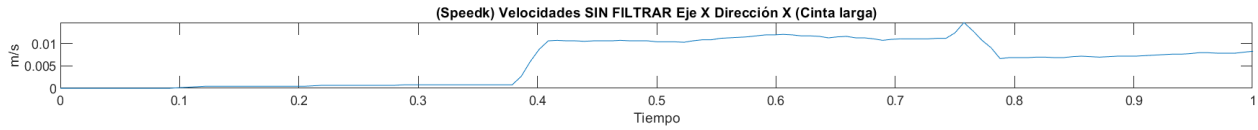


Figura 5.10 Gráfica de Speedk cuando MovementRangeSpeedDifferenceX = 0.

– Por lo tanto, hay que definir el valor de "MovementRangeSpeedDifferenceX" de modo que indique cuando el acelerómetro ha comenzado el movimiento para así comenzar a calcular velocidades. Para ello, hay que observar la gráfica de "SpeedDifference" en la zona de mayor ruido cuando el acelerómetro está detenido, es decir, antes y después de los picos de aceleración. Se busca uno de los valores más altos en esa zona de ruido y se encuentra el valor "0.000220514".

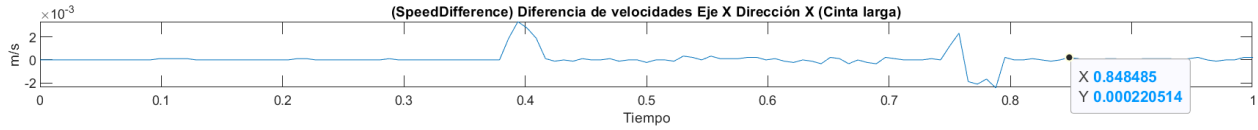


Figura 5.11 Gráfica de SpeedDifference para obtener el valor de MovementRangeSpeedDifferenceX.

– Entonces, "MovementRangeSpeedDifferenceX" deberá ser mayor que dicho valor, para que no considere que el acelerómetro está en movimiento. Si se define "MovementRangeSpeedDifferenceX" con el valor "0.0003" por ejemplo, funcionaría correctamente, pero tras realizar pruebas en ambas cintas, en la dirección "Y" (cinta corta) no sería suficiente ese valor, mientras que con el valor "0.0005" sí sería suficiente en ambos casos.

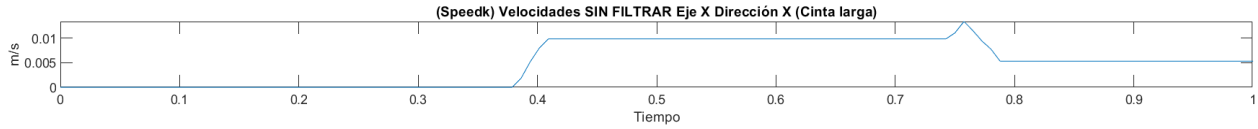


Figura 5.12 Gráfica de Speedk cuando MovementRangeSpeedDifferenceX = 0.0005.

- **DeltaCinta = 0.003**

  La elección de ese valor se realiza del siguiente modo:

  – Una vez definidos los valores de "OffsetSpeedDifferenceX" y "MovementRangeSpeedDifferenceX" hay que definir el valor de "DeltaCintaX". En este caso concreto en la dirección "X", si "DeltaCintaX" fuese cero, seguiría funcionando bien, pero no siempre ocurre esto tal y como se explicará posteriormente. En cambio en la dirección "Y" esto no es así, por lo que se define el valor de modo que cumpla para ambas direcciones.
  El valor de DeltaCinta se define observando la gráfica de "FinalsSpeeds" (velocidades filtradas). Para ello, hay que observar tanto la subida como la bajada de velocidad. En este caso, la de bajada tiene menor diferencia de altura, por lo que hay que centrarse en esa situación.

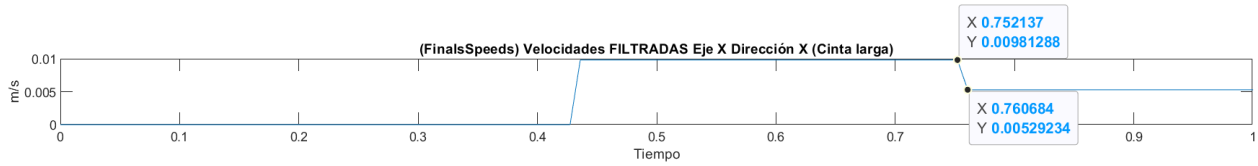


Figura 5.13 Gráfica de FinalsSpeeds para obtener el valor de DeltaCintaX.

El valor superior es "0.00981288", mientras que el inferior es "0.00529234". La diferencia de ambos es "0.00452054". El valor de "DeltaCintaX" debe ser menor que dicha diferencia, ya que cuando la diferencia sea mayor que "DeltaCintaX", fuerza el valor de la velocidad a cero (en este caso de disminución de velocidad). Si cumple para la zona de menor escalón, también cumplirá para la zona de mayor escalón, que en este caso es la zona en que aumenta la velocidad.
Aplicando un valor de DeltaCintaX igual a "0.003" se consigue la siguiente gráfica de velocidades finales forzadas:

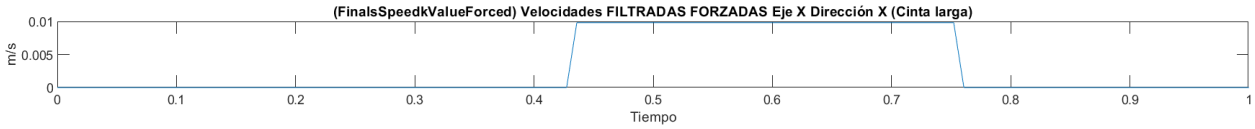


Figura 5.14 Gráfica de FinalsSpeedsForced cuando DeltaCintaX = 0.003.

Del mismo modo que se ha comentado, los valores de los parámetros creados, para el **eje "Y"** (tanto para dirección "X" como dirección "Y"), son:

- **OffsetSpeedDifferenceY = 0.00131602674**
- **MovementRangeSpeedDifferenceY = 0.0007**
- **DeltaCintaY = 0.003**

El caso que se ha expuesto en este apartado ha sido aquel que se comportaba según lo esperado. En cambio, se recopilaron datos crudos de aceleraciones en cinco ocasiones en total, siendo dos de ellas satisfactorias y tres de ellas insatisfactorias. Que una prueba sea insatisfactoria quiere decir que las medidas no se pueden predecir y al ser medidas aleatorias no cumplen con lo diseñado en el tratamiento de datos. Dicho problema surge cuando la bandeja que contiene el acelerómetro choca con el tope de final de cinta, lo que provoca un impacto y vibraciones que alteran las medidas del acelerómetro en esos instantes. A continuación se muestran las gráficas de un caso insatisfactorio.

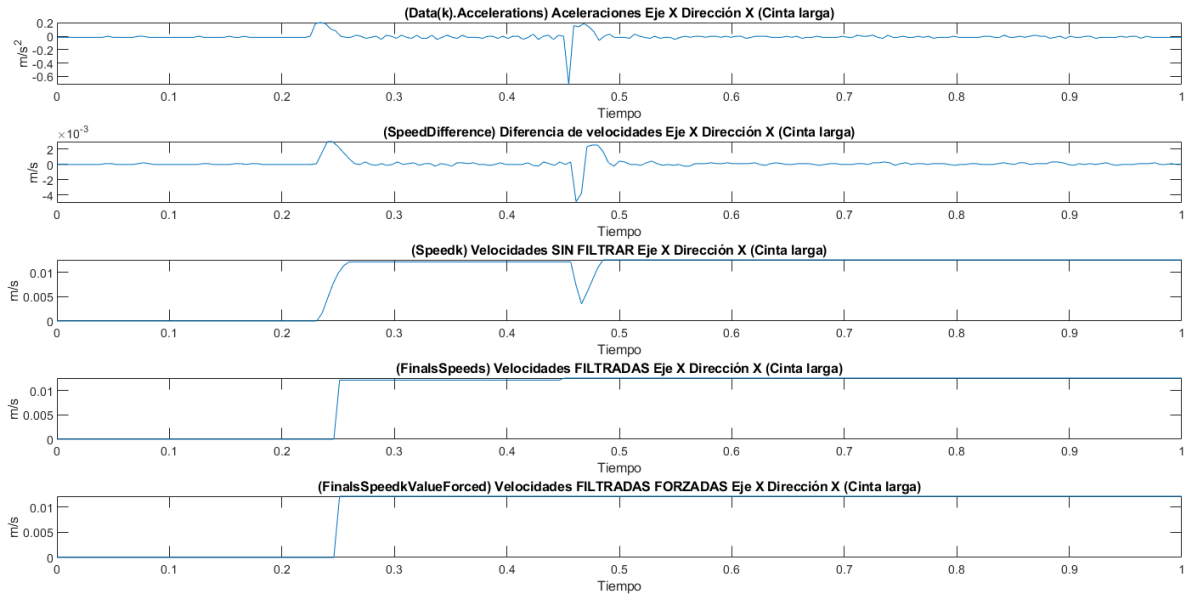


Figura 5.15 Gráficas del tratamiento insatisfactorio de datos del acelerómetro en dirección "X" eje "X".

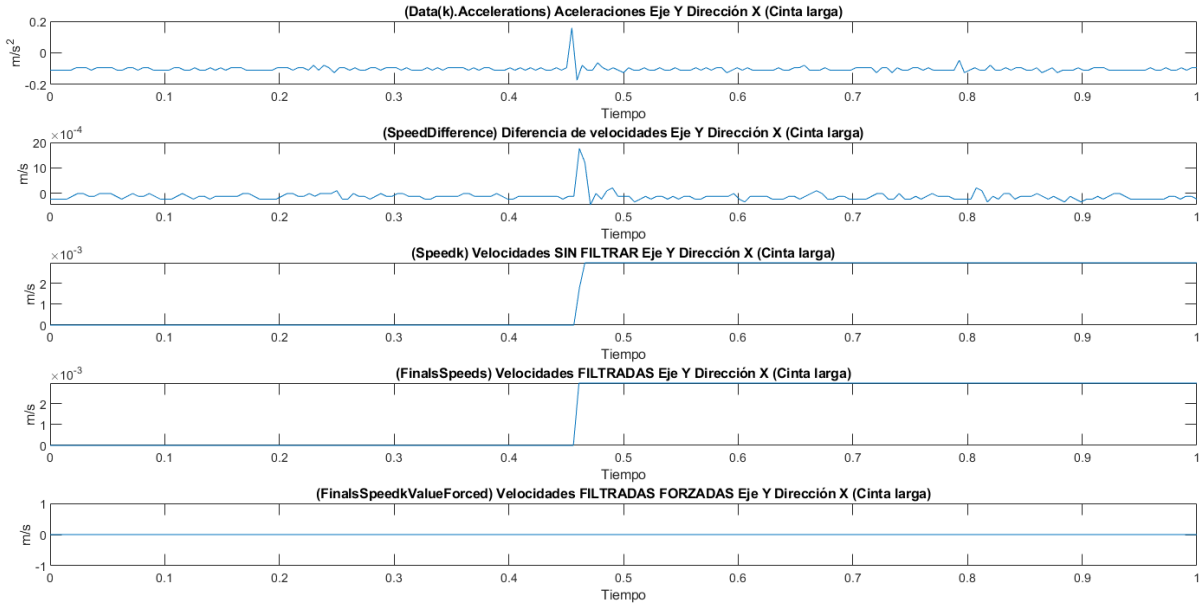


Figura 5.16 Gráficas del tratamiento insatisfactorio de datos del acelerómetro en dirección "X" eje "Y".

Como puede observarse, en caso del eje "X", la gráfica de aceleraciones tiene un comportamiento inesperado cuando detiene el movimiento debido al impacto contra el final de la cinta, afectando así al resto del tratamiento, ya que considera que el acelerómetro está en movimiento. No obstante, en caso del eje "Y" no hay incidencias en este caso.

5.3 Tratamiento de datos del SCT

El sensor de corriente SCT-013-030 proporciona valores de tensión, a partir de una lectura de valores analógicos. Por tanto, hay que realizar un tratamiento de datos para conseguir valores de intensidad y así poder calcular la potencia que consume el dispositivo al cual se le instale el SCT.

El esquema que define el tratamiento de datos con el SCT es:

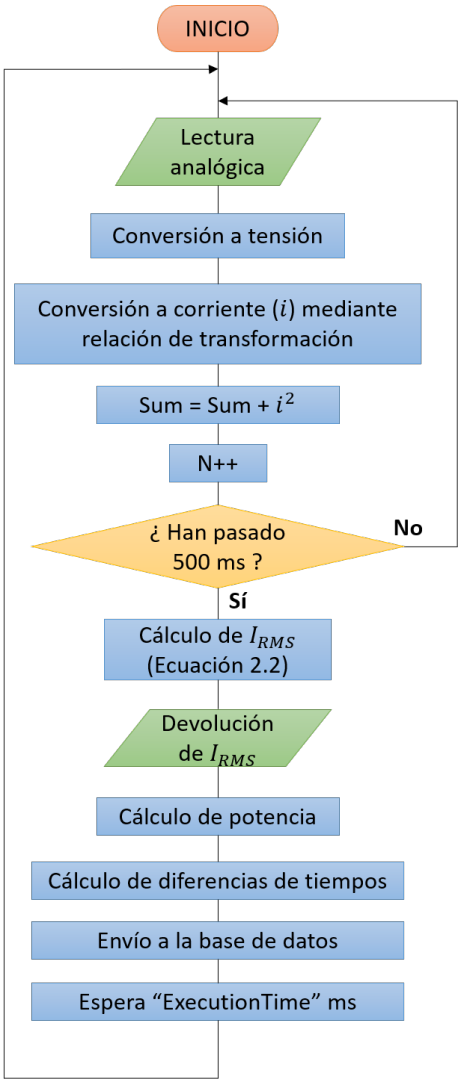


Figura 5.17 Esquema del tratamiento de datos del SCT.

El código completo se encuentra adjunto en el Anexo I (Capítulo 8)

De manera más detallada, para obtener valores de intensidad y potencia el procedimiento a seguir es el siguiente:

1. Durante 500 ms (25 ciclos) realiza lo siguiente:
   - **a)** Realizar la lectura de valores analógicos del pin de entrada del Arduino Uno WiFi Rev2. Será un valor entero entre 0 y 1023. Cabe destacar el uso de la función "$analogReference(INTERNAL)$", cambiando así la tensión tomada como referencia por el ADC al usar "*analogRead*". En el caso del Arduino Uno WiFi Rev2, ese valor de referencia interna de tensión es de 0.55 V.
   - **b)** Por tanto, para obtener el valor de la tensión (denominado en el código "*SensorVoltage*") correspondiente a la medida realizada por el SCT, es necesario multiplicar el valor de lectura analógica (entero entre 0 y 1023) por el valor de referencia interna de tensión y dividirlo por 1023.
   - **c)** El valor de la intensidad que circula por el interior del SCT se obtiene a partir de la relación 30A/1V característica del modelo SCT-013-030. Por tanto, la intensidad (A) será 30 veces el valor de tensión "*SensorVoltage*" (V).
   - **d)** Una vez calculado el valor de la intensidad, acumula en un sumatorio el valor de dicha intensidad al cuadrado y contabiliza el número de medidas mediante "N".
2. Cuando hayan pasado 500 ms recopilando valores de intensidad, multiplica el valor acumulado en el sumatorio por dos, para así compensar el semiciclo negativo que se anula en la rectificación de media

onda. El sumatorio lo divide entre "N" para así tener la media y le hace la raíz cuadrada, tal y como fue descrito en 2.2. El código que describe el subproceso de calcular la corriente eficaz I_{RMS} es:

Código 5.6 Código del tratamiento de datos previo a obtener la corriente eficaz Irms.

```
float GetCurrent()
{
  float SensorVoltage;
  float Current=0;
  float Sum=0;
  long SampleDataTime=millis();
  int N=0;

  while(millis()-SampleDataTime<500)
  {

    SensorVoltage = analogRead(A0) * (0.55 / 1023.0); // 0.55 because is
        the value of the internal reference voltage

    Current=SensorVoltage*30.0;         // Current=SensorVoltage*(30A/1V)
    Sum=Sum+sq(Current);                // Sum of squares
    N=N+1;
    delay(1);

  }
  Sum=Sum*2;   // The value of the summation is doubled to compensate
      for the negative half-cycle that was cancelled in the half-wave
      rectification.

  Current=sqrt((Sum)/N);  // RMS equation

  return(Current);
}
```

3. Una vez realizado todo lo anterior, se tiene el valor de corriente eficaz I_{RMS}.
4. Siguiendo lo comentado en 2.2.2, al multiplicar la corriente eficaz I_{RMS} por 230 V (P=V·I) se obtiene la potencia (W).
5. También, se recogen datos de la diferencia de tiempo entre medidas para así poder obtener la energía consumida en dicho periodo de tiempo. La diferencia de tiempo se envía a la base de datos, para así poder obtenerla en Unity3D y poder realizar aprendizaje tanto de potencia como de energía, según se desee. El código de esta parte del proceso es:

Código 5.7 Código del tratamiento de datos para calcular potencia y diferencias de tiempos.

```
// Call to function for get current measures

float Irms=GetCurrent(); // Get RMS current (A)

// Calculation of power

float P=Irms*230.0;           // P=IV (W)

// Calculation of time differences (to calculate the energy later)

float TimeDifference = (millis()-ExecutionTime)/1000.0; // Time
    differential between measurements
```

6. Finalmente, se envían a la base de datos los valores de corriente eficaz, potencia y diferencias de tiempos. Tras una espera de "ExecutionTime" ms, el proceso comienza de nuevo.

5.4 Pruebas realizadas y análisis de resultados del tratamiento de datos del SCT

Se han realizado diversas pruebas con el objetivo de analizar los resultados del tratamiento de datos diseñado.

En caso del SCT, no se han realizado pruebas sobre los motores de las cintas transportadoras de la célula de fabricación, pero las pruebas realizadas son análogas a si se probase sobre la planta.

El modo de empleo del SCT para realizar las pruebas es:

- Abrir la pinza por la pestaña indicada.
- Hacer pasar un único hilo del dispositivo del que se desea calcular el consumo.
- Cerrar la pinza.

Al tener que pasar por el interior del SCT un único hilo, para realizar las pruebas se hace uso de una regleta de alimentación. Por el interior de la cubierta de una regleta de alimentación circulan 3 hilos: fase, neutro y toma de tierra. Por tanto, al retirar la cubierta, se instala el SCT sobre el hilo de fase, para así poder calcular el consumo del dispositivo que se desee enchufar a la regleta de alimentación.

Figura 5.18 Instalación del SCT para la realización de pruebas.

Para comprender el resultado del tratamiento de datos del SCT, se realiza una prueba con un ejemplo aplicado a estimar el consumo de potencia de un ventilador.

Se observa como al estar el ventilador apagado da valores de potencia próximos a cero, mientras que al encenderlo sube hasta los 30 W aproximadamente.

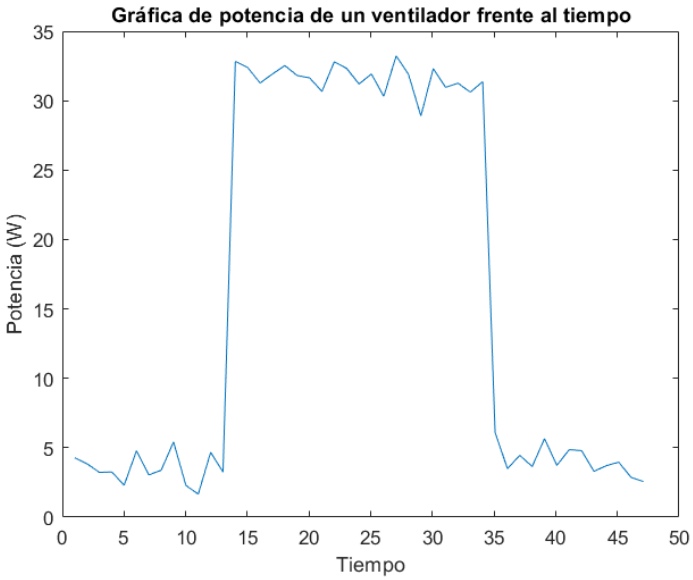


Figura 5.19 Gráfica de consumo de potencia de un ventilador.

6 Aprendizaje del Gemelo Digital

En este capítulo se describe el aprendizaje que realiza el Gemelo Digital a partir de los datos obtenidos mediante los procedimientos y los dispositivos comentados anteriormente. Los modelos implementados en el Gemelo Digital están completamente parametrizados y el entrenamiento va a consistir en adaptar esos parámetros al sistema real con los datos obtenidos de los sensores.

Se realizan dos tipos de aprendizaje: la energía que consumen los motores que alimentan las cintas transportadoras, y la velocidad de las cintas transportadoras.

Se presenta el fichero *TrainingBBDD.cs*, el cual es el que ordena el aprendizaje en función del parámetro *TrainingBBDDParameters.EnableTrainingBBDD* definido en el fichero *TrainingBBDDSupport.cs*.

El fichero *TrainingBBDDSupport.cs* (se adjunta completo en Anexo II (Capítulo 9) sirve para "dar apoyo" al fichero principal *TrainingBBDD.cs*, ya que define parámetros y métodos de modo que el fichero principal sólo necesita llamarlos. Para ejecutar el aprendizaje, además de *TrainingBBDDParameters.EnableTrainingBBDD* también debe de estar activa la variable *SQL.LastData.Valid*, que se crea para que el aprendizaje se ejecute cuando haya terminado de obtener los últimos datos de la base de datos.

Por tanto, el código del fichero principal ***TrainingBBDD.cs*** es:

```
using System.Collections.Generic;
using System.Collections;
using UnityEngine;
using System;

public class TrainingBBDD : MonoBehaviour
{

  void Start()
  {
    if (TrainingBBDDParameters.EnableTrainingBBDD)
    {
      StartCoroutine(SQL.DownloadAllData());
    }
  }

  void Update()
  {
    if (TrainingBBDDParameters.EnableTrainingBBDD && SQL.LastData.Valid == true)
    {
      TrainingBBDDMethods.TrainingBeltMotors();

      TrainingBBDDMethods.TrainingBeltConveyor();

      SQL.LastData.Valid = false;
    }
  }
```

```
}
```

Por tanto, como se puede observar en el código, si se dan las condiciones comentadas anteriormente para realizar el aprendizaje se ejecutan las llamadas a *TrainingBeltMotors*() y *TrainingBeltConveyor*(), contenidas en la clase *TrainingBBDDMethods* del fichero *TrainingBBDDSupport.cs*.

Esas llamadas a *TrainingBeltMotors*() y *TrainingBeltConveyor*() provocan el comienzo del aprendizaje del Gemelo Digital en función de los valores obtenidos de la base de datos y, anteriormente, del Arduino.

6.1 Energía consumida por los motores de las cintas transportadoras (TrainingBeltMotors())

La llamada ***TrainingBeltMotors()*** realiza el aprendizaje sobre la potencia (o energía) que consumen los motores de las cintas transportadoras.

Dicho aprendizaje consiste en asignar el valor de potencia consumida (W), obtenido a partir de los datos de potencia contenidos en la variable *SQL.LastData.CurrentSensor.Power*, al parámetro *BeltMotorsPower* definido en la clase *BeltMotorsEnergyModel* del fichero *ConsumptionModels.cs*.

También se puede realizar el aprendizaje sobre la energía que consumen los motores de las cintas transportadoras. Es por ello que también se obtienen valores de diferencia de tiempo de medidas de potencia en la variable *SQL.LastData.CurrentSensor.DeltaTime* asignado al parámetro *TimeMeasuresPower* definido en la clase *BeltMotorsEnergyModel* del fichero *ConsumptionModels.cs*, para así poder multiplicarlo por la potencia y obtener la energía consumida (kWh). El código en C# de la función *TrainingBeltMotors*() es:

Código 6.1 Código C# de la función de aprendizaje "TrainingBeltMotors()".

```
//Training methods

public static void TrainingBeltMotors()
{
  // Read of the times measures
  BeltMotorsEnergyModel.TimeMeasuresPower =
      SQL.LastData.CurrentSensor.DeltaTime;; // In case of energy training

  // Change the instantaneous power value that is present right now
  BeltMotorsEnergyModel.BeltMotorsPower = SQL.LastData.CurrentSensor.Power;

}
```

De este modo, el gemelo aprenderá con esos datos cuando se llame a la función *UpdateEnergy*() (en código de *ConsumptionModels.cs*), la cual todavía no está siendo usada en el proyecto DENiM del que forma parte este proyecto.

Mientras que los códigos de aprendizaje de energía son:

- ***ConsumptionModels.cs*** (solo la parte utilizada en este proyecto):

```
    // Model that realize the consumption dynamics

public class BeltMotorsEnergyModel : MonoBehaviour
{
  public static float BeltMotorsPower = 0.01f;
  public static float TimeMeasuresPower = 0.01f;
  public static void UpdateEnergy()
  {
```

```
    Consumption.BeltMotors.Power = BeltMotorsPower;
    Consumption.BeltMotors.Energy += BeltMotorsPower*TimeMeasuresPower;
  }
}
```

- ***PowerConsumption.cs*** (solo la parte utilizada en este proyecto):

```
// Variables that store de energy consumed

public class PowerConsumption : MonoBehaviour
{
  public static double BeltMotors;
}
```

6.2 Velocidad de las cintas transportadoras (TrainingBeltConveyor())

La llamada ***TrainingBeltConveyor()*** realiza el aprendizaje sobre la velocidad de las cintas transportadoras, así como también conocer a partir de su velocidad si la bandeja que contiene el acelerómetro se encuentra en una cinta corta o larga. Dicho aprendizaje consiste en asignar el valor de velocidad (m/s), contenido en $SQL.LastData.Accelerometers[numAccelerometer].Speed[i]$, al parámetro *Speed* definido en la estructura *Parameters*.

Como se ha comentado anteriormente, tras realizar pruebas, los valores de velocidades obtenidos tras el tratamiento de datos en Arduino no son valores fiables. Por lo que se decide realizar aprendizaje sobre conocer en qué cinta se encuentra el acelerómetro.

Esta estructura se encuentra en la clase correspondiente según cinta corta (*ShortBeltConveyorModel*) o larga (*LongBeltConveyorModel*) del fichero *ModelsSupport.cs*. Siendo *i* un entero entre 0 (eje x), 1 (eje y), 2 (eje z) y siendo *numAccelerometer* un entero entre 0 y *SQLParameters.NumAccelerometer*, para cuando haya varios acelerómetros en funcionamiento.

Las pruebas se realizan considerando un único acelerómetro en funcionamiento, pero el código queda diseñado para el caso general en que haya varios acelerómetros. En ese caso, el parámetro de velocidad almacenado en *ShortBeltConveyorModel.Parameters.Speed* o *LongBeltConveyorModel.Parameters.Speed* será la media en cada cinta de los distintos dispositivos que están circulando por las cintas.

El código que representa la función *TrainingBeltConveyor()* es:

```
public static void TrainingBeltConveyor()
{
  // Variables declaration
  float[,] SpeedReaded = new float[3, SQLParameters.NumAccelerometer];

  // Reading function
  for(int numAccelerometer = 0; numAccelerometer <
      SQLParameters.NumAccelerometer; numAccelerometer++)
  {
    for (int axis = 0; axis < 3; axis++)
    {
      SpeedReaded[axis, numAccelerometer] =
          SQL.LastData.Accelerometers[numAccelerometer].Speed[axis];
    }

  }

  float SumSpeedReadedLong;
  float SumSpeedReadedShort;
```

```
  while(SpeedReaded[0,0] != 0.0f) //Long conveyor considering Axis X = Long
      Conveyor
  {
    for(int numAccelerometer = 0; numAccelerometer <
        SQLParameters.NumAccelerometer; numAccelerometer++)
    {
      SumSpeedReadedLong += SpeedReaded[0, numAccelerometer];
    }

    LongBeltConveyorModel.Parameters.Speed = SumSpeedReadedLong /
        SQLParameters.NumAccelerometer;

  }

  while(SpeedReaded[1,0] != 0.0f) //Short conveyor considering Axis Y = Short
      Conveyor
  {
    for(int numAccelerometer = 0; numAccelerometer <
        SQLParameters.NumAccelerometer; numAccelerometer++)
    {
      SumSpeedReadedShort += SpeedReaded[1, numAccelerometer];
    }

    ShortBeltConveyorModel.Parameters.Speed = SumSpeedReadedShort /
        SQLParameters.NumAccelerometer;

  }

}
```

7 Conclusiones y trabajos futuros

En este proyecto, se ha expuesto un método de interconexión de una célula de fabricación flexible con un Gemelo Digital mediante una base de datos.

Tras la explicación de todos los ámbitos de este proyecto, a continuación se realiza un resumen de las conclusiones y objetivos conseguidos.

Entre las ideas llevadas a cabo en este proyecto, junto con algunas conclusiones, destacan:

- Se ha realizado la interconexión de datos entre los dispositivos instalados sobre la célula y una base de datos implementada en PostgreSQL. Cabe destacar que, tras realizar diversas pruebas, se concluye que por muy pequeño que sea el intervalo de tiempo entre envíos de datos a la BBDD, el envío se va a realizar cada 1,5 - 2 segundos aproximadamente.
- Se ha creado y diseñado una base de datos en PostgreSQL, formada por diversos tipos de tablas que almacenan los datos tratados de los dispositivos instalados en la célula.
- Se ha realizado la interconexión de datos entre la BBDD y el Gemelo Digital implementado en Unity3D, así como se ha diseñado un procedimiento de operaciones para obtener datos deseados según el tipo de dispositivo, a partir de las tablas completas de datos, en Unity3D. Además del diseño de una comunicación desde Unity3D hacia la BBDD.
- Se ha realizado un tratamiento de datos en Arduino tanto al acelerómetro como al SCT para conseguir los valores deseados:
  - En el caso del SCT, ha sido necesario realizar el tratamiento de datos de cara a conseguir medidas de potencia, a partir del cálculo de la corriente eficaz. En cuanto a las pruebas realizadas no hay limitaciones destacables, por lo que se cumple lo previsto inicialmente.
  - En el caso del acelerómetro, ha sido necesario realizar el tratamiento de datos para obtener medidas de velocidades "forzadas" válidas que indiquen cuando las bandejas estén en movimiento o no sobre las cintas transportadoras. Como se comentó en 5.2, esta no era la idea inicial, pero debido a las adversidades debidas a limitaciones del microcontrolador junto con las vibraciones e impactos de las bandejas, se decidió cambiar de objetivo en esta parte del proyecto.
    Pese a mejorar los resultados con este cambio, sigue sin ser suficiente, ya que el impacto de las bandejas sobre el final de las cintas provoca medidas impredecibles (tal y como fue descrito en 5.2), además de la persistencia del problema de la limitación del Arduino, debido a la recursividad.
    En cuanto a las limitaciones del microcontrolador, se deben al uso de la recursividad en el filtrado de velocidades. Esto provoca que, en ocasiones, el programa se bloquee debido al desbordamiento de pila, ya que en los Arduino, la memoria suele ser escasa y la recursividad provoca desconocer la memoria que va a ocupar la ejecución del programa.
- Se ha realizado un aprendizaje del Gemelo Digital, a partir de los datos obtenidos de la planta real, aplicado a dos ejemplos concretos.

Finalmente, se plantea una idea relacionada con el proyecto para posibles ampliaciones a implementar en trabajos futuros.

Basándose en el modo de interconexión expuesto en este proyecto, o diseñando otro modo de interconexión, sería interesante ampliarlo utilizando otro tipo de sensores o dispositivos que también puedan aportar al

aprendizaje del Gemelo Digital y así adaptarse a nuevas necesidades, con el objetivo de poder recrear operaciones reales realizadas sobre la planta real.

La competencia dentro de la industria hace que sea fundamental una ejecución optimizada a nivel de procesos, así como la flexibilidad de operaciones. Es por ello, que profundizar la investigación tanto en este tema, como en la tecnología de los gemelos digitales en general, puede aportar mucho a la industria.

8 Anexo I

Este anexo está formado por los diagramas de flujo y los códigos de Arduino necesarios para la utilización del acelerómetro ADXL335 y el sensor de corriente SCT en el desarrollo del presente proyecto.

Código 8.1 Código completo de tratamiento de datos para el acelerómetro ADXL335 (Arduino).

```
/*
This code consists of measuring accelerations with an ADXL335 accelerometer
and calculating the velocity of its axes from the integral of the
acceleration with respect to time.
Filtering is also performed using recursion.
*/


//////////////////////////////////////////////////
/////////// DECLARATION OF PARAMETERS ////////////
//////////////////////////////////////////////////

#define MaxNumMeasures 20       // Maximum number of acceleration measures

#define DeltaCintaX 0.00       //  0.001   Hay que determinar que valor se
    considera para detectar cambio brusco de velocidad ( incremento Y decremento )
#define DeltaCintaY 0.00
#define DeltaCintaZ 0.00

#define Vcero 0.00

#define Tm 1UL                //   Sample time ( ms )

// Range to admit a new value for the acceleration
#define PermissibleDeviationRangeX 1.5
#define PermissibleDeviationRangeY 1.5
#define PermissibleDeviationRangeZ 1.5

// Speed offset. Speed difference offset to get it as close as possible to '0'.
#define OffsetSpeedDifferenceX 0.00
#define OffsetSpeedDifferenceY 0.00
#define OffsetSpeedDifferenceZ 0.00

// Speed range. Range from which the accelerometer is considered to be moving.
#define MovementRangeSpeedDifferenceX 0.00
#define MovementRangeSpeedDifferenceY 0.00
#define MovementRangeSpeedDifferenceZ 0.00
```

```
// Accuracy of the Arduino
#define accuracy 0.0001

//////////////////////////////////////////////////

float ConveyorBeltSpeed[3] = {0.006, 0.0015, 0.00};
float VoidAxisZ = 0.00;

//Component 0: Length of long conveyor belt | Component 1: Length of short
    conveyor belt
float ConveyorBeltLength[2] = {6.0, 4.5};

float PermissibleDeviationRange[3] = {PermissibleDeviationRangeX,
    PermissibleDeviationRangeY, PermissibleDeviationRangeZ};
float OffsetSpeedDifference[3] = {OffsetSpeedDifferenceX, OffsetSpeedDifferenceY,
    OffsetSpeedDifferenceZ};
float MovementRangeSpeedDifference[3] = {MovementRangeSpeedDifferenceX,
    MovementRangeSpeedDifferenceY, MovementRangeSpeedDifferenceZ};
float DeltaCinta[3] = {DeltaCintaX, DeltaCintaY, DeltaCintaZ};

//////////////////////////////////////////////////////
//// DECLARATION OF GLOBAL VARIABLES ( ACCELERATOR ) ///
//////////////////////////////////////////////////////

// Accelerometer declaration

#include "ADXL335.h"
ADXL335 accelerometer;

int i, NumMeasures, N;

// Structure for storing data

struct DataStructure
{
  float Accelerations[MaxNumMeasures];   // ( m/s^2 )
  float TimeDifferences[MaxNumMeasures]; // ( s )
  bool Valids[MaxNumMeasures-1];         // Valid: '1'. Not valid: '0'.
  int NumberValids;                      // Number of valid speed values.
  float Speeds[MaxNumMeasures-1];        // (m/s)
};

// Create a vector of structures. One for each axis.| 0: axis x | 1: axis y | 2:
    axis z

DataStructure Data[3];

// Global variables for speed and time measures

int FlagForced[3] = {0, 0, 0};

float Speedk[3] = {0, 0, 0};
float Speedk1[3] = {0, 0, 0}; // Speedk1 = v(k-1)

float MeanFinalsSpeedk[3] = {0, 0, 0};
float MeanFinalsSpeedk1[3] = {0, 0, 0};
```

```
float MeanFinalsSpeedkForced[3] = {0, 0, 0};

unsigned long vtime = 0;

unsigned long StartTime[3] = {0, 0, 0};
unsigned long EndTime[3] = {0, 0, 0};

//////////////////////////////////////////////////////
///// DECLARATION OF GLOBAL VARIABLES ( DATABASE ) ////
//////////////////////////////////////////////////////

// Libraries needed to connect to the Internet and communicate with the database

#include <WiFiNINA.h>
#include <SPI.h>
#include <ArduinoHttpClient.h>

// Server IP Address and port

const char serverName[] = "--INSERT IP ADDRESS--"; //For example 192.168.0.1
int port = 80;

// WiFi credentials

char ssid[] = "--INSERT WIFI SSID--";
char pass[] = "--INSERT WIFI PASS--";

// Connection to the server

WiFiClient wifi;
HttpClient client = HttpClient(wifi, serverName, port);

// ID of the Accelerometer. Needs to be previously declared in the database

int ID_Accel = 1; //ID of the Accelerometer
int aux_status;


//////////////////////////////////////////////////

//////////////////////////////////////////////////
////////////////////// SETUP /////////////////////
//////////////////////////////////////////////////

void setup()
{
  Serial.begin(115200);
  accelerometer.begin();

  // Begin WiFi section

  int status = WiFi.begin(ssid, pass);
  aux_status=status;
  if ( status != WL_CONNECTED)
  {
    Serial.println("Couldn't get a wifi connection");
    while(true);
```

```
  }
  // print out info about the connection:
  else
  {
    Serial.println("Connected to network");
    IPAddress ip = WiFi.localIP();
    Serial.print("My IP address is: ");
    Serial.println(ip);
  }
  // End WiFi section

  for (int k = 0; k < 3; k++)
  Speedk[k] = Speedk1[k];
}

////////////////////////////////////////////////

////////////////////////////////////////////////
//////////////////// LOOP //////////////////////
////////////////////////////////////////////////

void loop()
{
  float ax, ay, az;

  if (millis() > vtime + Tm)
  {
    // Get acceleration measures

    accelerometer.getAcceleration(&ax, &ay, &az);
    float a[3] = { ax, ay, az};

    for ( int k = 0; k < 1; k++ )
    {
      a[k] = a[k] * 9.81;                    // ( m/s^2 )

      // Store acceleration measures

      Data[k].Accelerations[NumMeasures] = a[k];

      // Get time measures

      if(NumMeasures == 0)
      {
        Data[k].TimeDifferences[NumMeasures] = 2.00/1000.0; [s]
      }
      else
      {
        Data[k].TimeDifferences[NumMeasures] = (millis() - vtime)/1000.0; // [s]

      }


      // Initially, all values are valid
      if(NumMeasures < (MaxNumMeasures - 1))
      {
        Data[k].Valids[NumMeasures] = true;
```

```
    }

  }

  NumMeasures++;

  vtime = millis();

}

// All measures are collected

if (NumMeasures == MaxNumMeasures)
{
  DataStructure FinalsData[3];

  // Calculate the axis speed. | k=0 : "X" axis | k=1 : "Y" axis | k=2 : "Z"
      axis |

  for (int k = 0; k < 2; k++)
  {
    //Initially all measures are valids
    Data[k].NumberValids = MaxNumMeasures;

    // Declaration of vector where the speed calculations are to be stored

    float SpeedDifference = 0;

    // Do the integral calculation to obtain the speed
    for (int i = 1; i < Data[k].NumberValids; i++)
    {

      SpeedDifference = ((Data[k].Accelerations[i] + Data[k].Accelerations[i -
          1]) / 2) * (Data[k].TimeDifferences[i] + Data[k].TimeDifferences[i -
          1])/2;

      SpeedDifference = OffsetSpeedDifference[k] + SpeedDifference;
          //SpeedDifference must be displayed (where Offset = 0) to calculate
          the required offset.

      if (fabs(SpeedDifference) > MovementRangeSpeedDifference[k])
      {

        Speedk[k] = Speedk1[k] + SpeedDifference;

      }

      Data[k].Speeds[i-1] = Speedk[k];

      Speedk1[k] = Speedk[k];

    }

    float FinalsSpeedsSum = 0;

    //The recursive function filter is called
```

```
      FinalsData[k] = RecursiveFilter(Data[k], k);

      float FinalsSpeeds[FinalsData[k].NumberValids];

      int contValidsSpeeds = 0;

      //The velocities that have passed the filter are stored in the vector
          "FinalsSpeeds"

      for (int i = 0; i < (MaxNumMeasures-1); i++)
      {
        if((FinalsData[k].Valids[i] == 1) && (contValidsSpeeds <
            FinalsData[k].NumberValids))
        {
          FinalsSpeeds[contValidsSpeeds] = FinalsData[k].Speeds[i];
          FinalsSpeedsSum += FinalsSpeeds[contValidsSpeeds];
          contValidsSpeeds = contValidsSpeeds + 1;

        }

      }

      //The average of the speeds that have passed the filter is averaged.

      MeanFinalsSpeedk[k] = FinalsSpeedsSum/FinalsData[k].NumberValids;

      //"DeltaCinta" is obtained by observing the difference of the two values
          when the accelerometer starts or ends the movement.
      //"DeltaCinta" should be slightly less than this difference.

      if (fabs(MeanFinalsSpeedk[k] - MeanFinalsSpeedk1[k]) > DeltaCinta[k])
      {

        //If the movement is starting
        if(MeanFinalsSpeedk1[k] > MeanFinalsSpeedk[k])
        {
          MeanFinalsSpeedkForced[k] = Vcero;

          FlagForced[k] = 0;

        }
        else //If the movement is ending
        {

          ConveyorBeltSpeed[k] = MeanFinalsSpeedk[k];

          MeanFinalsSpeedkForced[k] = MeanFinalsSpeedk[k];

          FlagForced[k] = 1;

        }

      }
      else
      {
        if(FlagForced[k])
        {
```

```
          MeanFinalsSpeedkForced[k] = ConveyorBeltSpeed[k];
        }
        else
        {
          MeanFinalsSpeedkForced[k] = Vcero;
        }

      }

      MeanFinalsSpeedk1[k] = MeanFinalsSpeedk[k];

    }

    ///// SENDING TO DATABASE ////

    if ( aux_status == WL_CONNECTED )
    {
      String ID_Accel_string= String(ID_Accel);
      String vk_x_string= String(MeanFinalsSpeedkForced[0],6);
      String vk_y_string= String(MeanFinalsSpeedkForced[1],6);
      String vk_z_string= String(VoidAxisZ,6);
      String httpRequestData =
          "/php2pgsql/php_prueba_linkeada_acelv1.php?ID_Acel="+ID_Accel_string+"&vel_x="+vk_x_string+"&ve
      Serial.println("Making get request");
      client.sendHeader(HTTP_HEADER_CONTENT_TYPE,
          "application/x-www-form-urlencoded");
      client.get(httpRequestData);
      int statusCode = client.responseStatusCode();
      Serial.print("Status code: ");
      Serial.println(statusCode);
      String response = client.responseBody();
      Serial.print("Response: ");
      Serial.println(response);
    }

    NumMeasures = 0;

  }

}


////////////////////////////////////////////////
/////////////// RECURSIVE FUNCTION ////////////////
////////////////////////////////////////////////

/*
The proccedure of the recursive function is:
1) Do the mean of valid speed measures
2) Do the mean of the deviation of valid speed measures
3) If the deviation is greater than PermissibleDeviationRange * MeanDeviation and
    if the measure is apparently valid, the measure is not valid.
4) Finally, if the quantity of valid measures are not the same and there are more
    than one valid measure, the recursive function is executed again to achieve
    better filtering.
*/
```

```
DataStructure RecursiveFilter(DataStructure InputStructure, int axis)
{

  DataStructure OutputData = InputStructure;
  float contValids = 0;
  float Mean = 0;
  float Sum = 0;
  float MeanDeviation = 0;
  float SumDeviation = 0;

  for (int i = 0; i < (MaxNumMeasures-1); i++)
  {
    if (OutputData.Valids[i] == true)
    {
      Sum = Sum + OutputData.Speeds[i];
      contValids++;
    }
  }
  OutputData.NumberValids = contValids;
  Mean = Sum / contValids;

  for (int i = 0; i < (MaxNumMeasures-1); i++)
  {
    if (OutputData.Valids[i] == true)
    {
      SumDeviation = SumDeviation + fabs(OutputData.Speeds[i] - Mean);
    }
  }

  MeanDeviation = SumDeviation / contValids;

  for (int i = 0; i < (MaxNumMeasures-1); i++)
  {
    if ((MeanDeviation > accuracy) && (fabs(OutputData.Speeds[i] - Mean) >=
        PermissibleDeviationRange[axis] * MeanDeviation) && (OutputData.Valids[i]
        == true))
    {
      OutputData.Valids[i] = false;
      OutputData.NumberValids = OutputData.NumberValids - 1;
    }
  }

  // Call to recursive function again if the condition is fulfilled

  if (!((InputStructure.NumberValids == OutputData.NumberValids) ||
      (OutputData.NumberValids < 2)))
  {
    OutputData = RecursiveFilter(OutputData, axis);
  }

  // End of recursive function

  return OutputData;

}
```

Código 8.2 Código completo de tratamiento de datos para el sensor de corriente SCT-013-030 (Arduino).

```
////////////////////////////////////////////////
/////////// DECLARATION OF PARAMETERS ////////////
////////////////////////////////////////////////

#define Tm 3000          // Sample time ( ms )


// ID of the SCT. Needs to be previously declared in the database

int ID_Sensor = 2;       //ID of the SCT

////////////////////////////////////////////////

//////////////////////////////////////////////////////////
///// DECLARATION OF GLOBAL VARIABLES ( DATABASE ) ////
//////////////////////////////////////////////////////////

// Libraries needed to connect to the Internet and communicate with the database

#include <WiFiNINA.h>
#include <SPI.h>
#include <ArduinoHttpClient.h>

// Server IP Address and port

const char serverName[] = "--INSERT IP ADDRESS--"; //For example 192.168.0.1
int port = 80;

// WiFi credentials

char ssid[] = "--INSERT WIFI SSID--";
char pass[] = "--INSERT WIFI PASS--";

// Connection to the server

WiFiClient wifi;
HttpClient client = HttpClient(wifi, serverName, port);

int aux_status;

// Global variables for time measures

unsigned long SampleDataTime;  // Time you are sampling data for IRMS calculation
unsigned long ExecutionTime; // How often does everything run

////////////////////////////////////////////////

////////////////////////////////////////////////
////////////////////// SETUP ///////////////////////
////////////////////////////////////////////////

void setup()
{
  Serial.begin(115200);
```

```
  //Reference voltage for analogue measurements

  analogReference(INTERNAL);   // 0.55 V on Arduino UNO WiFi Rev2

  // Begin WiFi section

  int status = WiFi.begin(ssid, pass);
  aux_status=status;
  if ( status != WL_CONNECTED)
  {
    Serial.println("Couldn't get a wifi connection");
    while(true);
  }
  // print out info about the connection:
  else
  {
    Serial.println("Connected to network");
    IPAddress ip = WiFi.localIP();
    Serial.print("My IP address is: ");
    Serial.println(ip);
  }
  // End WiFi section

}

////////////////////////////////////////////////

////////////////////////////////////////////////
//////////////////// LOOP //////////////////////
////////////////////////////////////////////////

void loop()
{
  if (millis() > ExecutionTime + Tm)
  {
    // Call to function for get current measures

    float Irms=GetCurrent(); // Get RMS current (A)

    // Calculation of power

    float P=Irms*230.0;           // P=IV (W)

    // Calculation of time differences (to calculate the energy later)

    float TimeDifference = (millis()-ExecutionTime)/1000.0; // Time differential
        between measurements

    ///// SENDING TO DATABASE ////

    if ( aux_status == WL_CONNECTED)
    {
      String Irms_string= String(Irms,6);
      String P_string= String(P,6);
      String ID_Sensor_string= String(ID_Sensor);
      String difTiempo_string = String(TimeDifference,6);
```

```
      String httpRequestData =
          "/php2pgsql/php_prueba_linkeada_sctv1.php?ID_Sensor="+ID_Sensor_string+"&Corriente="+Irms_stri
      Serial.println("Making get request");
      client.sendHeader(HTTP_HEADER_CONTENT_TYPE,
          "application/x-www-form-urlencoded");
      client.get(httpRequestData);

      int statusCode = client.responseStatusCode();
      Serial.print("Status code: ");
      Serial.println(statusCode);
      String response = client.responseBody();
      Serial.print("Response: ");
      Serial.println(response);
    }

    ExecutionTime = millis();
  }
}


////////////////////////////////////////////////
/////////// FUNCTION TO GET RMS CURRENT //////////
////////////////////////////////////////////////

/*
The proccedure of the recursive function is:
1) Read analogue input values
2) That analogue reading is an integer between 0 and 1023. So it is converted
to voltage by multiplying by the reference voltage and dividing by 1023.
3) That resulting voltage is multiplied by the ratio of the SCT to convert it
    into current.
4) Then, the RMS equation is applied.
*/

float GetCurrent()
{

  float SensorVoltage;
  float Current=0;
  float Sum=0;
  long SampleDataTime=millis();
  int N=0;
  while(millis()-SampleDataTime<500)
  {

    SensorVoltage = analogRead(A0) * (0.55 / 1023.0); // 0.55 because is the
        value of the internal reference voltage
    Current=SensorVoltage*30.0;                    //
        Current=SensorVoltage*(30A/1V)
    Sum=Sum+sq(Current);                           // Sum of squares
    N=N+1;
    delay(1);

  }
```

```
    Sum=Sum*2;              // The value of the summation is doubled to compensate
        for the negative half-cycle that was cancelled in the half-wave
        rectification.
    Current=sqrt((Sum)/N);  // RMS equation

    return(Current);
  }
```

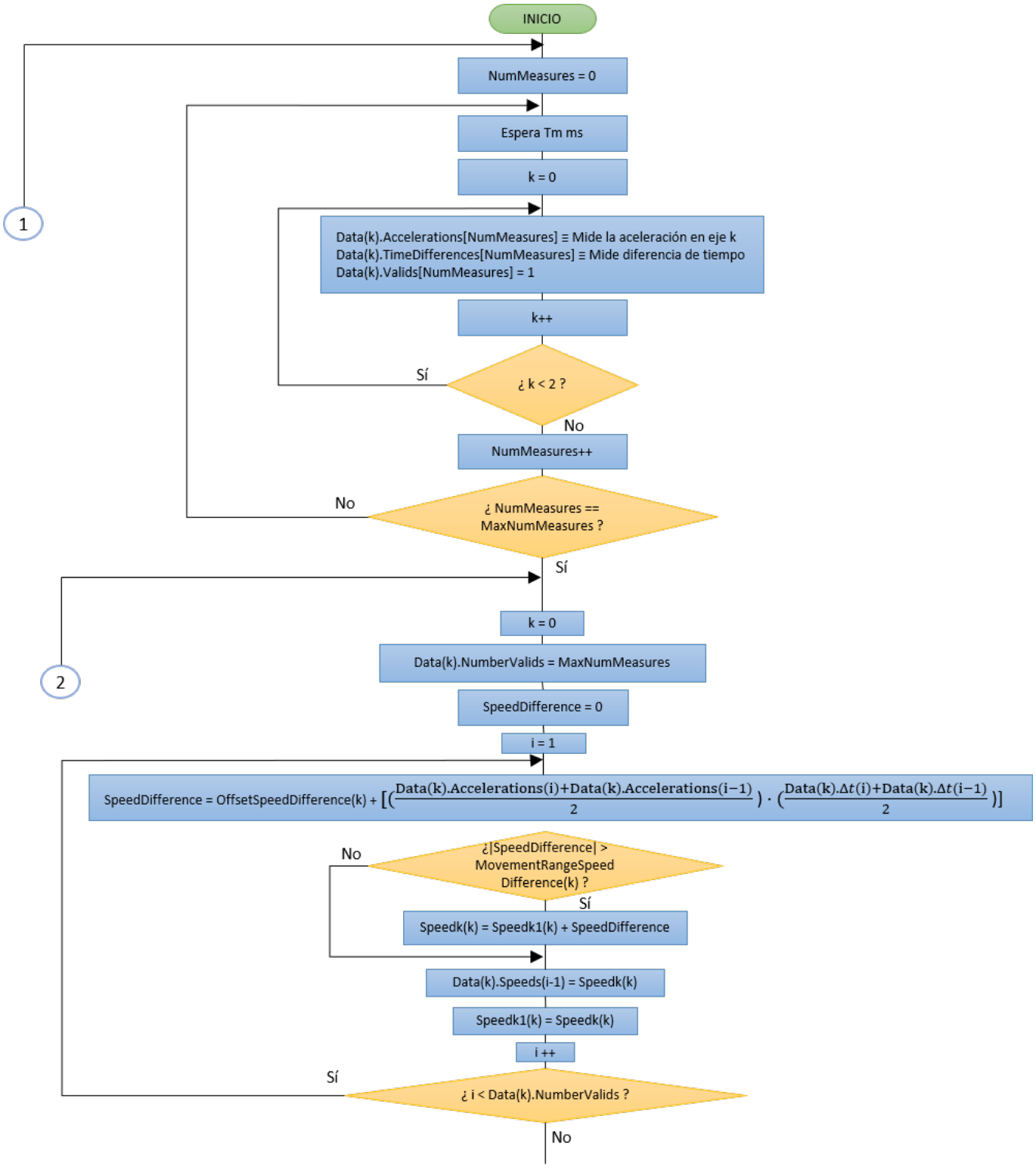


Figura 8.1 (1 de 2) Diagrama de flujo del código del acelerómetro en Arduino.

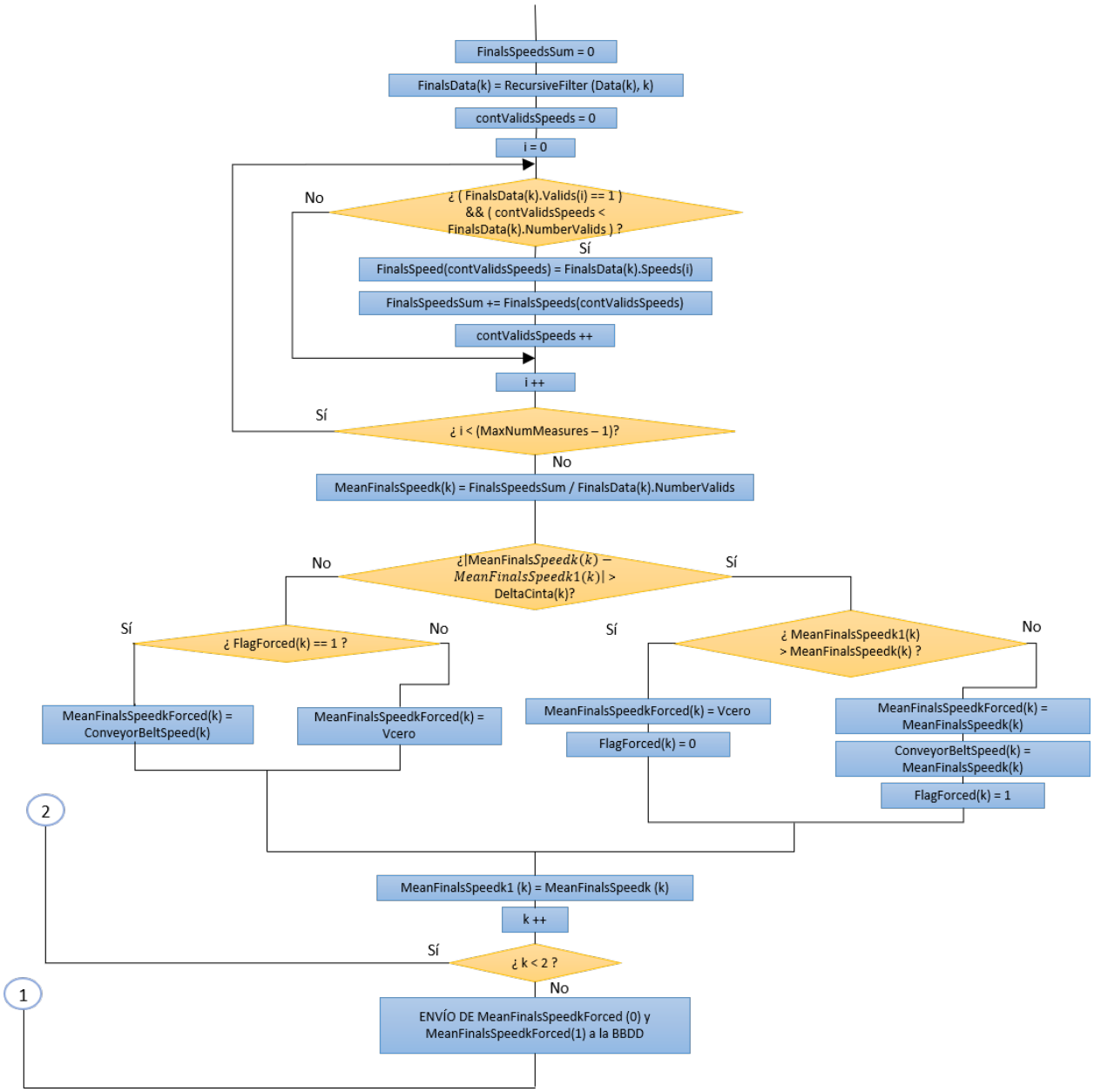


Figura 8.2 (2 de 2) Diagrama de flujo del código del acelerómetro en Arduino.

9 Anexo II

Este anexo está formado por los códigos completo de C# implementados en Unity3D necesarios para el aprendizaje del Gemelo Digital a partir de los datos obtenidos en el desarrollo del presente proyecto.

Código 9.1 Código "SQL.cs" de C# para la obtención de datos en Unity3D.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System;
using UnityEngine.Networking;

public class SQL : MonoBehaviour
{
  public static SQLMethods.DDBBData LastData;

  // Method that returns all the latest data

  public static IEnumerator DownloadAllData()
  {
    for (int type = 0; type < 2; type++)
    {
      // Send the request to the PHP server to extract the data
      UnityWebRequest DownloadedData =
          UnityWebRequest.Get(SQLParameters.PHPUrl[type]);
      yield return DownloadedData.SendWebRequest();

      // Data processing
      string[] UpdateData = DownloadedData.downloadHandler.text.Split(';');
      switch(SQLParameters.SensorType[type])
      {
        case "Current":

        LastData.CurrentSensor.Power =
            Convert.ToSingle(SQLMethods.GetValueData(UpdateData[UpdateData.Length
            - 2],"potencia:").Replace(".",","));
        LastData.CurrentSensor.DeltaTime =
            Convert.ToSingle(SQLMethods.GetValueData(UpdateData[UpdateData.Length
            - 2],"diftiempo:").Replace(".",","));

        break;
```

```
        case "Accelerometer":

        LastData.Accelerometers = new
            SQLMethods.StructAccelerometerSensorData[SQLParameters.NumAccelerometer];

        for(int numAccelerometer = 0; numAccelerometer <
            SQLParameters.NumAccelerometer; numAccelerometer++)
        {
          LastData.Accelerometers[numAccelerometer].Id =
              Convert.ToInt32(SQLMethods.GetValueData(UpdateData[UpdateData.Length
              - 2],"id_acel:").Replace(".",","));
          LastData.Accelerometers[numAccelerometer].UpdateTime =
              SQLMethods.GetValueData(UpdateData[UpdateData.Length -
              2],"tiempo:").Replace(".",",");
          LastData.Accelerometers[numAccelerometer].Speed = new float[3];
          for (int i = 0; i < 3; i++)
          {
            LastData.Accelerometers[numAccelerometer].Speed[i] =
                Convert.ToSingle(SQLMethods.GetValueData(UpdateData[UpdateData.Length
                - 2],SQLParameters.ComponentsSpeed[i]).Replace(".",","));

          }
        }

        break;

        default:
        Debug.Log("Houston, we have a problem");
        break;
      }
    }
    LastData.Valid = true;
  }
}
```

Código 9.2 Código C# denominado *UnityToPostgreSQL.cs* para enviar datos de Unity3D a la BBDD mediante la ejecución del fichero PHP indicado.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

using UnityEngine.Networking; //Introducido segun UnityWebRequest

public class DatosToPostgreSQL : MonoBehaviour
{
  public InputField txtID_Sensor;
  public InputField txtCorriente;
  public InputField txtPotencia;
```

```
  public int ID_Sensor_SCT;
  public float Corriente_SCT;
  public float Potencia_SCT;

  //string URLenviar =
      "http://localhost/bbdd_sct_unity_v1/enviodatosapostgresql_unity_sct.php";


  public void EnviarDatosDesdeUnity(){ //Asociado a boton "Enviar datos desde
      Unity hacia postgresql"
    StartCoroutine (enviardatos());
  }

  IEnumerator enviardatos(){

    WWW conexion = new
        WWW("http://localhost/bbdd_sct_unity_v1/enviodatosapostgresql_unity_sct.php?
        id_sensor="+ txtID_Sensor.text + "&corriente=" + txtCorriente.text +
        "&potencia=" + txtPotencia.text );//+ txtID_Sensor + "&corriente=" +
        txtCorriente + "&potencia=" + txtPotencia);

    yield return(conexion);

    if(conexion.text == "200"){

      ID_Sensor_SCT = int.Parse(txtID_Sensor.text);
      Corriente_SCT = float.Parse(txtCorriente.text);
      Potencia_SCT = float.Parse(txtPotencia.text);
      print("Valores introducidos correctamente");

    }else{

      Debug.Log(conexion.text); //Devolverá "400" lo cual representa fallo en
          conexion con bbdd
    }

  }
```

Código 9.3 Código "TrainingBBDDSupport.cs" de C# que define parámetros a partir de los datos obtenidos.

```
using System.Net.Sockets;
using System.Threading;
using UnityEngine;
using TrainingLibrary;
using System;

public class TrainingBBDDParameters : MonoBehaviour
{
  //Training parameters
  public const bool EnableTrainingBBDD = true;
```

```
}


public class TrainingBBDDMethods : MonoBehaviour
{
  //Training methods
  public static void TrainingBeltMotors()
  {
    // Read of the times measures
    //float TimeRead = SQL.LastData.CurrentSensor.DeltaTime; // In case of
        energy training
    BeltMotorsEnergyModel.TimeMeasuresPower =
        SQL.LastData.CurrentSensor.DeltaTime; // In case of energy training

    // Change the instantaneous power value that is present right now
    BeltMotorsEnergyModel.BeltMotorsPower = SQL.LastData.CurrentSensor.Power;

    Debug.Log(TimeRead);
    Debug.Log(BeltMotorsEnergyModel.BeltMotorsPower);

  }

  public static void TrainingBeltConveyor()
  {
    // Variables declaration
    float[,] SpeedReaded = new float[3, SQLParameters.NumAccelerometer];

    // Reading function
    for(int numAccelerometer = 0; numAccelerometer <
        SQLParameters.NumAccelerometer; numAccelerometer++)
    {
      for (int axis = 0; axis < 3; axis++)
      {
        SpeedReaded[axis, numAccelerometer] =
            SQL.LastData.Accelerometers[numAccelerometer].Speed[axis];
      }

    }

    float SumSpeedReadedLong;
    float SumSpeedReadedShort;

    while(SpeedReaded[0,0] != 0.0f) //Long conveyor considering Axis X = Long
        Conveyor
    {
      for(int numAccelerometer = 0; numAccelerometer <
          SQLParameters.NumAccelerometer; numAccelerometer++)
      {
        SumSpeedReadedLong += SpeedReaded[0, numAccelerometer];
      }

      LongBeltConveyorModel.Parameters.Speed = SumSpeedReadedLong /
          SQLParameters.NumAccelerometer;

    }
```

```
      while(SpeedReaded[1,0] != 0.0f) //Short conveyor considering Axis Y = Short
          Conveyor
      {
        for(int numAccelerometer = 0; numAccelerometer <
            SQLParameters.NumAccelerometer; numAccelerometer++)
        {
          SumSpeedReadedShort += SpeedReaded[1, numAccelerometer];
        }

        ShortBeltConveyorModel.Parameters.Speed = SumSpeedReadedShort /
            SQLParameters.NumAccelerometer;

      }

    }

  }
```

Bibliografía

[1] IE321 Industrial Production Systems Garry Woods. Información sobre sistemas de célula de fabricación. *https://slideplayer.com/slide/13093576/*, 2017. Online. Fecha de consulta: 8 de septiembre de 2022.

[2] Naylamp Mechatronics. Guía de uso sobre el sct-013-030. *https://naylampmechatronics.com/blog/51_tutorial-sensor-de-corriente-ac-no-invasivo-sct-013.html*. Online. Fecha de consulta: 12 de julio de 2022.

[3] "DENiM Project". *https://cordis.europa.eu/project/id/958339*. Online. Fecha de consulta: 12 de julio de 2022.

[4] Centro Tecnológico CTIC. Tecnología de los gemelos digitales. *https://www.fundacionctic.org/es/actualidad/gemelos-digitales*. Online. Fecha de consulta: 12 de julio de 2022.

[5] Nobbot. 'digital twin': los objetos físicos buscan a su gemelo digital. *https://www.nobbot.com/negocios/digital-twin-los-objetos-fisicos-buscan-a-su-gemelo-digital/*. Online. Fecha de consulta: 12 de julio de 2022.

[6] decide. Información sobre gemelos digitales. *https://decidesoluciones.es/gemelos-digitales-cadena-de-suministro/*. Online. Fecha de consulta: 12 de julio de 2022.

[7] Capgemini Research Institute. *Digital Twins: Adding Intelligence to the Real World*. 2022. Fecha de consulta: 12 de julio de 2022.

[8] ViewNext. *https://www.viewnext.com/que-son-los-gemelos-digitales/*. Online. Fecha de consulta: 12 de julio de 2022.

[9] Markets and Markets. Mercado de los gemelos digitales. *https://www.marketsandmarkets.com/PressReleases/digital-twin.asp*, 2022. Online. Fecha de consulta: 12 de julio de 2022.

[10] Dpto. Ingeniería Mecánica Universidad del País Vasco. Sistemas de fabricación flexible. *https://www.ehu.eus/manufacturing/docencia/1151_ca.pdf*. Online. Fecha de consulta: 12 de julio de 2022.

[11] TFG Universidad de Sevilla R. Iglesias Bayo. Integración de equipos en célula de fabricación flexible. *http://hdl.handle.net/11441/43587*, 2016. Online.

[12] Arduino. Página oficial del arduino uno wifi rev2. *https://docs.arduino.cc/hardware/uno-wifi-rev2*. Online. Fecha de consulta: 12 de julio de 2022.

[13] YHDC. *Split Core Current Transformer, Model: SCT013-030. Patent No. ZL 2015 3 0060067.X.*

[14] Analog Devices. Accelerometer adxl335. *https://www.analog.com/media/en/technical-documentation/data-sheets/adxl335.pdf*. Online. Fecha de consulta: 12 de julio de 2022.

[15] Seeed-Studio. Repositorio de la librería de arduino accelerometer_adxl335. *https://github.com/Seeed-Studio/Accelerometer_ADXL335*. Online. Fecha de consulta: 12 de julio de 2022.

[16] AprendiendoArduino. Ide arduino. *https://aprendiendoarduino.wordpress.com/2016/12/11/ide-arduino/*, 2016. Online. Fecha de consulta: 12 de julio de 2022.

[17] HostingPedia. Definición y características de postgresql. *https://hostingpedia.net/postgresql.html*, 2019. Online. Fecha de consulta: 12 de julio de 2022.

[18] Geotalleres. Conceptos básicos de sql. *https://geotalleres.readthedocs.io/es/latest/conceptos-sql/conceptos_sql.html*. Online. Fecha de consulta: 12 de julio de 2022.

[19] Arduino. Página oficial de arduino: Librería wifinina. *https://www.arduino.cc/reference/en/libraries/wifinina/*. Online. Fecha de consulta: 12 de julio de 2022.

[20] Adrian McEwen. Repositorio de la librería de arduinohttpclient. *https://github.com/arduino-libraries/ArduinoHttpClient*. Online. Fecha de consulta: 12 de julio de 2022.

[21] MDN. Solicitud http. *https://developer.mozilla.org/es/docs/Web/HTTP/Methods/POST*. Online. Fecha de consulta: 12 de julio de 2022.