Universidad Zaragoza
1542

Trabajo Fin de Máster

Sistema automatizado para el control de existencias dentro de una cámara frigorífica

Automated system for the control of stock in a refrigerator

Autor

Víctor Xiang Zheng

Director

José Ramón Gállego Martínez

Máster Universitario en Ingeniería de Telecomunicación

Escuela de Ingeniería y Arquitectura

2016

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Víctor Xiang Zheng ,

con nº de DNI X2181766D en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Máster , (Título del Trabajo) Sistema automatizado para el control de existencias dentro de una cámara frigorífica ,

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 22 de Noviembre de 2016

Fdo: Víctor Xiang Zheng

Sistema automatizado para el control de existencias dentro de una cámara frigorífica

Resumen

En este Trabajo de Fin de Máster se ha realizado un sistema automatizado para el control de existencias dentro de una cámara frigorífica, enfocando a su aplicación a negocios del sector de la restauración. Para ello, se ha desarrollado una serie de sensores que son capaces de medir y registrar de forma automática la cantidad de comida que hay dentro de la cámara frigorífica y enviar estos datos a un servidor para procesarlos.

El trabajo está enmarcado como un proyecto de IoT (Internet of Things), cuyo principal objetivo es la interconexión digital de objetos cotidianos utilizando dispositivos electrónicos embebidos, sensores y proveyéndoles de conectividad con el mundo externo. De esta forma el sistema provee a una cámara frigorífica la capacidad de saber la cantidad de comida que contiene dentro y enviar esta información a una entidad externa.

El sensor se ha implementado en un Arduino Mini Pro que controlará el sensado de las medidas. Para el envío de los datos generados al servidor se ha utilizado ZigBee, un protocolo de comunicaciones inalámbrico de bajo consumo energético.

Con los datos generados por los sensores se ha implementado un sistema de gestión que permitirá llevar un control eficiente sobre la tasa a la que se gasta la comida y el tiempo que éste lleva dentro de las cámaras. Se ha implementado también una interfaz web gráfica a través de la cual se puede visualizar estos datos. Con esta información podrá realizar pedidos de forma automática a los proveedores calculando la cantidad que podría ser necesaria durante los próximos días.

De este modo, en este Trabajo de Fin de Máster se ha desarrollado un módulo de sensado que generará datos y medidas, un módulo de servidor que procesará estos datos para obtener y mostrar información útil y un módulo de comunicaciones que conecten estos dos elementos de forma inalámbrica.

Automated system for the control of stock in a refrigerator

Abstract

In this MSc dissertation, an automated system for the control of stock in a refrigerator has been developed with its application aimed at catering trades. A sensor has been designed to automatically measure the quantity of food in the refrigerator and to send this data to a server to be processed.

This dissertation is framed in a project of IoT (Internet of Things). Its main objective is the internetworking of physical devices and ordinary items embedded with electronic, sensors and network connectivity. In this way, the system provides a common object like a refrigerator the ability to know the quantity of food it has inside and send this information to an external entity.

The sensor has been implemented in an Arduino Mini Pro that will control the sensing of the measurements. ZigBee has been used as the communication protocol to send the data to the server. It is a wireless and low power consumption protocol.

With all the data generated from these sensors, a management system has been developed. It will allow to keep control efficiently of the food spending rate and the time it has been kept in the refrigerator. A graphical web interface has also been developed to show this data. With this information, the system will also be able to make orders automatically to the suppliers, calculating the quantity of food needed for the following few days.

As a result, this dissertation has developed a sensing module that will generate data, a server module to process this data to extract and show valuable information and a communications module to connect these two elements wirelessly.

Índice de contenidos

Índice de figuras

Índice de tablas

Índice de ecuaciones

Acrónimos

IoT	Internet of Things
OEM	Original Equipment Manufacturer
FIFO	First In First Out
IDE	Integrated Development Environment
EOQ	Economic Order Quantity
USB	Universal Serial Bus
LED	Light-emitting diode
ADC	Analog-to-digital converter
LCD	Liquid-crystal-display
SIPO	Serial Input Parallel Output
TPV	Terminal Punto de Venta
XAMPP	Cross-Platform Apache, MariaDB, PHP and Perl
PHP	PHP: Hypertext Preprocessor
HTML	Hypertext Transfer Protocol
CSS	Cascading Style Sheets
MCU	Microcontroller Unit
SOA	Safety Operating Area
SMCR	Sleep Mode Control Register
MCUCR	MCU Control Register
PRR	Power Reduction Register
EEPROM	Electrically erasable programmable read-only memory
MSB	Most Significative Bit
MAC	Medium Access Control
UART	Universal Asynchronus Receiver / Transmitter
CTS	Clear to Send
IP	Internet Protocol
CMA	Cumulative Moving Average

1 Introducción y objetivos

1.1 Introducción a la problemática

El objetivo de este proyecto es realizar un sistema de control automático de los alimentos enfocado a negocios del sector de la restauración tales como hoteles, restaurantes o bares.

Una de las tareas más importantes de los encargados de este tipo de negocios es la gestión de los alimentos. Esta tarea puede parecer sencilla en un principio ya que todos estamos acostumbrados a controlar la comida que hay en casa.

Sin embargo en estos negocios el volumen manejado es mucho mayor y una gestión inapropiada puede generar problemas económicos, sanitarios e incluso sociales de forma indirecta. Un problema podría ser por ejemplo no tener material suficiente para la demanda del momento, con su consecuente pérdida económica y una mala reputación frente al cliente. También puede ocurrir el problema contrario; demasiado material encargado, que puede llegar a estropearse generando también pérdidas económicas al desecharlo o lo que es peor aún, no darse cuenta de que un producto se ha estropeado ya que en el caso de servirlo se estaría incumpliendo las normativas sanitarias establecidas. Es necesario remarcar que se está gestionando un producto cuyo tiempo de almacenamiento es muy corto al contrario que en otros negocios minoristas como una librería o una tienda de ropa donde el producto puede estar meses guardado en el almacén sin sufrir ningún desperfecto.

El despilfarro de comida es un importante problema de la actualidad. Diversos estudios [1] indican que hasta un tercio de toda la comida producida en el mundo acaba en la basura. Este hecho puede parecer sorprendente si tenemos en cuenta que más de un 10% de la población mundial no tiene suficientes alimentos para subsistir [2]. Tampoco se debe olvidar que cada vez que se tira comida, toda la energía, agua, tiempo, dinero y en general, todos los recursos que se han utilizado para producirlo, prepararlo, almacenarlo, envasarlo y transportarlo también está siendo malgastados. Por lo tanto, el despilfarro de comida es un gran problema social que aún está lejos de solucionarse.

Volviendo al caso de los restaurantes, el uso de un sistema de control del stock ineficiente, un incorrecto mantenimiento de sistemas de refrigeración o una inadecuada predicción de la demanda son algunas de las principales causas de desperdicio en negocios del sector de la restauración. La mayor parte de este despilfarro se podría evitar si se aplicaran las técnicas de gestión adecuadas.

Por lo tanto en este proyecto se ha propuesto la realización de un sistema de gestión que evite todos estos problemas citados anteriormente.

Este proyecto se ha realizado en colaboración con un restaurante de Zaragoza, que ha facilitado todos los datos necesarios para su desarrollo.

En la operativa habitual de este restaurante, el primer paso es procesar la comida que trae un proveedor. Una vez es procesada, la comida se guarda en contenedores metálicos o de plástico, se etiquetan con su fecha de envasado y se almacenan en un congelador. A medida que

la comida de las cámaras frigoríficas se va gastando, se va sacando los contenedores guardados en el congelador y se deja descongelar para su uso próximo.

La técnica de congelado y descongelado es muy importante para mantener una buena calidad en estos alimentos. La industria alimentaria posee el material y las condiciones adecuadas para realizar la congelación óptima de los alimentos. Sin embargo los restaurante no disponen de estas condiciones y la comida congelada difícilmente tendrá el mismo sabor y la misma textura que un producto fresco no congelado.

Este Trabajo de Fin de Máster, a partir de ahora TFM, propone también utilizar exclusivamente productos frescos prescindiendo de la congelación, ofreciendo una mayor calidad en el producto final y ahorrando también los costes de manutención de un congelador.

Este sistema será capaz de medir la cantidad de comida que hay dentro de las cámaras frigoríficas de un restaurante. Controlará también cada vez que se introduce comida cuando lo traiga un proveedor y cada vez que se extraiga una ración cuando se necesite elaborar un plato. El tiempo que la comida lleva almacenada será un indicativo de la calidad de ésta.

Estos datos son recolectados y enviados a una base de datos. De estos datos se extraerá información útil como la tasa a la que se gasta la comida y el sistema será capaz de pedir automáticamente a los proveedores la comida necesaria para los próximos días sin necesidad de congelarla y evitando quedarse sin stock ante la demanda de un cliente y evitando también que se produzcan despilfarros por exceso de material.

1.2 Antecedentes

Este proyecto está enmarcado como un proyecto de Internet of Things (IoT), cuyo principal objetivo es la interconexión digital de objetos cotidianos como dispositivos, vehículos, edificios, etc, utilizando dispositivos electrónicos embebidos, sensores y proveyéndoles de conectividad con el mundo externo. El sistema propuesto cuenta con una serie de sensores que capta datos dentro de una cámara frigorífica y los envía a una base de datos donde un servidor los procesa y extrae información para realizar un pedido si es necesario.

Por lo tanto, este sistema provee de inteligencia a un objeto cotidiano como es una cámara frigorífica. Sin embargo su labor de detección de comida disponible y sus otras características lo hace distinto de las neveras inteligentes fabricadas por grandes OEMs (Original Equipment Manufacturer) como Samsung o Siemens. En la actualidad, la función de estos aparatos es más la de proporcionar entretenimiento en la cocina para el ámbito familiar, proporcionando una pantalla que te permita interactuar con la nevera, ver lo que hay en el interior sin necesidad de abrirlo, proporcionar apps de recetarios, música para cocinar, etc. En el futuro sí es posible que estos electrodomésticos sean capaces de identificar los objetos que haya dentro o detectar su fecha de caducidad por ejemplo.

Figura 1. Nevera inteligente

El mayor valor proporcionado por los proyectos de IoT es la información que se puede obtener de los datos generados. En este proyecto se generan datos sobre el consumo de carne en un restaurante, pero el encargado de éste puede no ser el único interesado en conocer la información que se puede obtener de estos datos. Los clientes por ejemplo suelen mostrar también interés en conocer la frescura del alimento que se le ofrece. Los proveedores también podrían estar interesados en conocer esta información, ya que de ello dependen sus ganancias. Y más en general toda la cadena de abastecimiento también podría querer información sobre el consumo de carne por ejemplo de todos los restaurantes de una ciudad o de la población en general.

1.3 Motivación

Este sistema de control automático de la comida de una cámara frigorífica se ha diseñado desde el inicio tras observar la problemática comentada anteriormente, con los objetivos generales de proporcionar una mayor eficiencia durante la gestión de los alimentos para que no se produzcan despilfarros, ofrecer productos más frescos a los clientes y realizar pedidos a los proveedores de forma automática.

Tal y como se ha comentado antes, este proyecto se ha realizado en colaboración con los datos y la operativa de un restaurante en concreto, pero este sistema se podría generalizar para otros negocios, especialmente cadenas de restaurantes, donde la gestión es aún más complicada debido al mayor volumen de material necesario.

1.4 Objetivos

Para cumplir con estos objetivos generales se han definido una serie de objetivos específicos que se muestran a continuación:

- Análisis de la problemática existente en la gestión de alimentos.

- Análisis de los requisitos que se deben cumplir en relación a la operativa de un restaurante.
- Diseño de sistema que solucione la problemática abordada. Este sistema es capaz de monitorizar y controlar la calidad de la comida. Para ello se utilizarán unos sensores que detecten la cantidad de comida que hay, la fecha desde la que está guardada y la fecha en la que se gasta. Tras este diseño inicial, el sistema se ha dividido en tres módulos: el módulo sensor que irá dentro de la cámara frigorífica, el módulo servidor que almacenará y procesará los datos y un módulo de comunicaciones.
- Diseño y desarrollo de módulo de sensado que generará los datos necesarios.
- Implementación de un sistema de comunicaciones inalámbrico entre el sensor y el servidor.
- Diseño y desarrollo de un servidor capaz de recolectar los datos enviados por el sensor, almacenarlo en la base de datos y procesarlo.
- Diseño y desarrollo de un sistema de predicción y pedidos basado en los datos almacenados en la base de datos

1.5 Estructura de la memoria

La memoria de este TFM está dividida en dos partes. En la primera parte reside el grueso de este trabajo, donde se explica el análisis y el desarrollo de los objetivos descritos anteriormente. La segunda parte contiene el anexo con los ficheros de código más importantes creados durante el desarrollo del sistema.

La primera parte de la memoria está dividida en los siguientes capítulos:

- Introducción y objetivos:
  En esta primera parte se explica la problemática, la motivación que ha llevado a la elaboración de este TFM y los objetivos que pretende abordar.
- Análisis y diseño del sistema:
  Se analiza los requisitos a seguir impuestos por el restaurante para ajustar el uso del sistema dentro del funcionamiento diario del negocio. Posteriormente se plantea una solución para la problemática abordada anteriormente y se analiza el diseño, los materiales y las tecnologías a utilizar justificando el camino seguido para la elaboración del sistema.
- Desarrollo e implementación:
  Describe de forma detallada todo el trabajo realizado para la implementación de cada uno de los tres módulos en los que se ha dividido el sistema.
- Pruebas y resultados:
  Una vez finalizado el desarrollo se comenta la metodología seguida para probar el funcionamiento del sistema y se explican los resultados obtenidos.
- Conclusión y líneas futuras:
  Se resumen los conceptos aprendidos durante la elaboración de este proyecto y se presenta una serie de líneas futuras que podrían mejorar el sistema y proporcionarle mayor versatilidad.

2 Análisis y diseño del sistema

2.1 Análisis de requisitos

El primer paso a seguir tras observar la problemática es realizar un análisis de requisitos sobre el cual el sistema propuesto se va a basar. En este análisis se comparte todos los datos necesarios entre las partes interesadas, en este caso el encargado de restaurante, el desarrollador y los usuarios que será el personal del restaurante. Para estudiar los requisitos se ha utilizado el siguiente guion:

Análisis de los requisitos:

1 *Requisitos de la empresa*
2 *Funcionamiento del sistema actual*
3 *Funcionamiento del sistema propuesto*

Los requisitos de la empresa forman en marco contextual en el que se va a implantar el sistema. Algunos de los requisitos que se han fijado en relación al proyecto son:

- En relación al horario:
    - El restaurante funciona 7 días a la semana en dos medias jornadas: de 12:00 a 16:00 y de 20:00 a 00:00.
    - Los proveedores solo entregan pedidos de lunes a viernes en horario para restaurantes de 12:00 a 14:00.
    - El plazo de entrega de los proveedores es menor de un día. Se debe realizar el pedido la noche anterior para tenerlo el día siguiente. El pedido se puede hacer mediante una llamada a un contestador automático o mediante un correo electrónico.
- En relación al higiene:
    - Los contendedores deben poder moverse libremente. Se debe permitir que un contenedor se pueda sacar de la cámara frigorífica para rellenarlo o para lavarlo por ejemplo.
    - Distintos tipos de comida van en distintos contenedores.
    - Comida del mismo tipo pero de distinta fecha deben ir en contenedores separados. Además la regulación actual exige que se etiquete la fecha de envasado.

Estos datos condicionarán el diseño del sistema.

En cuanto al funcionamiento actual de la operativa del restaurante se ha proporcionado las siguientes claves:

- Se encarga comida fresca, no congelada, ya que esto permite el procesamiento de la comida para dejarla preparada según los usos del restaurante, es decir, para poder cortarla en su tamaño adecuado y añadir ciertos ingredientes a modo de marinado. Posteriormente se envasa en contenedores de plástico, se etiqueta con la fecha de envasado y se almacena en el congelador a una temperatura menor que -18ºC para una

óptima conservación. Cada contenedor contiene aproximadamente 2 kilogramos de comida.

- A medida que se va utilizando y se va gastando la comida, se sacan contenedores del congelador y se dejan a descongelar en la cámara de refrigeración. El proceso de descongelado dura aproximadamente un día.
- Dentro del refrigerador, los contenedores tienen un lugar habitual aunque a veces se pueden mover si se decide reorganizar su distribución interna.
- La tasa de uso para carnes es de aproximadamente 20-25 kilogramos por semana. Una ración contiene alrededor de 200 gramos de carne.
- La carne fresca como carne de vacuno o porcino se conserva adecuadamente entre 4 y 5 días en un refrigerador a menos de 4ºC.
- Los pedidos se realizan una vez por semana. El plazo de entrega es menor que un día. Por ejemplo, para que traigan el producto el lunes hay que hacer el pedido antes de domingo por la noche. Normalmente el encargado realiza una llamada a un contestador automático por la noche entre las 23:00 y la 1:00 indicando la cantidad necesaria.

Figura 2. Modelo de cámara frigorífica (a la izquierda) y congelador (a la derecha) utilizado en el restaurante

Con los datos proporcionados por los requisitos de la empresa y observando su funcionamiento actual, se ha diseñado una solución que permita llevar la operativa actual comentada y además evitar la problemática explicada en el apartado anterior.

El sistema propuesto se basa en el uso de unos sensores colocados en el interior de la cámara frigorífica que miden el peso de cada tipo de comida (ternera, cerdo, pollo, etc) que hay dentro.

Los sensores están formados por una célula de carga que es la que realiza el sensado, y dos placas de cualquier material que se pueda guardar dentro de la cámara. Estas placas forman la base y la superficie sobre la que se colocan los contenedores con la comida.

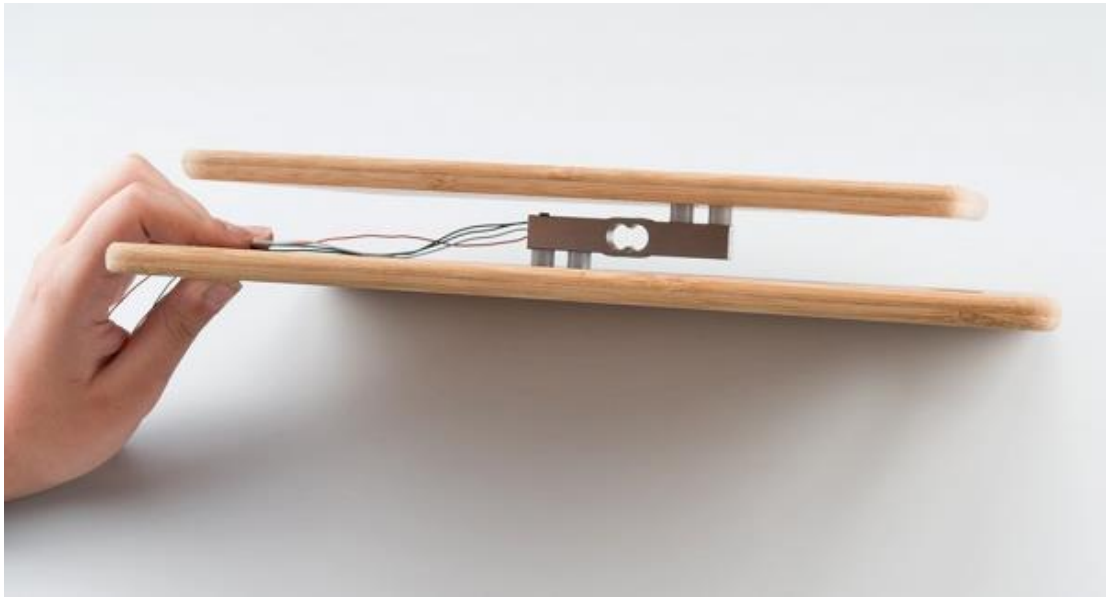

Figura 3. Montaje de uno de los sensores utilizados para medir el peso

Como el sistema es capaz de medir el peso, cuando se detecte que la última medida es mucho mayor que la anterior, por ejemplo antes quedaba medio kilogramo de ternera y ahora en la última medida se ha detectado que hay diez kilogramos, significa que se ha recargado el material. La fecha en la que se ha generado este dato se guarda en la base de datos para llevar el control del tiempo que el nuevo material va a estar guardado en la cámara y garantizar así su frescura.

Esta forma de trabajar implica que hay que gastar el material usando una técnica FIFO (First In First Out). Se gasta primero ese medio kilogramo que había antes de empezar a utilizar el nuevo material.

Los datos de medidas generados por los sensores se envían a un servidor remoto, que la almacena en una base de datos. Al contrario que los sensores, el servidor está situado en algún lugar fuera de la cámara frigorífica. Por ello es necesario que la comunicación entre sensor y servidor sea de forma inalámbrica ya que no se permite pasar ningún cable entre el interior y el exterior del refrigerador.

Una vez que el servidor tiene los datos, los almacena en una base de datos y a la vez realiza una serie de cálculos para realizar los pedidos cuando sea necesario.

Un sistema de control de inventario [3] eficaz tiene como objetivo reducir el inventario para reducir costes pero a la vez asegurando que no se produzca una rotura de stock. Para ello se centra básicamente en dos cuestiones: cuándo pedir y cuánto pedir. En la siguiente gráfica se muestra un ejemplo de uso de inventario a lo largo del tiempo.

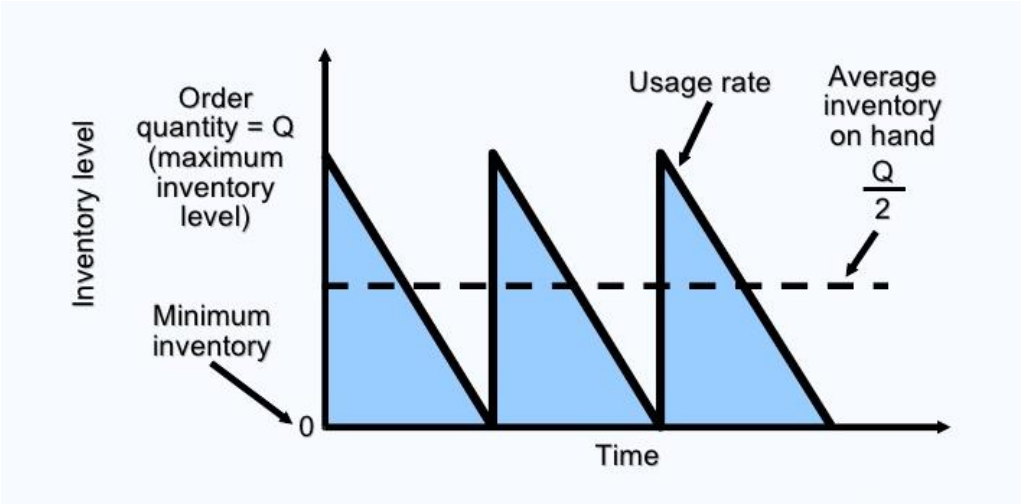

Figura 4. Gráfica de uso de stock según el modelo EOQ

Este modelo de uso de inventario es el Economic Order Quantity (EOQ). Para aplicar este modelo, se realizan ciertas suposiciones que se ajustan bastante bien al gasto en el restaurante:

- La demanda es conocida y más o menos constante
- El plazo de entrega de los proveedores es conocido
- La recepción de un pedido es instantánea y completa, es decir, se recibe todo el material en una única entrega

El sistema propuesto debe diseñarse para que pida la cantidad adecuada en el momento adecuado teniendo en cuenta los requisitos que se han impuesto. Los que más limitan el diseño son que los proveedores solo traen material de lunes a viernes y el hecho de que se está trabajando con un producto cuya vida útil es muy corta, manteniéndose en buen estado un máximo de 5 días. Esto obliga a realizar dos pedidos a la semana de forma que cada ciclo dura entre 3 y 4 días. Teniendo en cuenta que los fines de semana se gastan más cantidad que entre semana, se ha fijado que el material sea recibido los lunes y los viernes, por lo que los pedidos se realizarían los domingos y los jueves por la noche. Así el material del lunes se utilizaría desde este día hasta el jueves y los que lleguen el viernes se utilizarían el viernes y a lo largo del fin de semana.

Una vez se sabe cuándo hay que pedir, hay que calcular cuánto hay que pedir. Esto viene dado por los datos generados por los sensores en los periodos de tiempo anteriores. La metodología propuesta es separar los datos generados según sean del primer ciclo (de lunes a jueves) o segundo ciclo (viernes y fin de semana) ya que las cantidades son diferentes entre estos dos ciclos. Para cada ciclo, la cantidad a pedir se ha propuesto que sea el promedio de todos los periodos anteriores del mismo ciclo, menos la cantidad que aún queda sin gastar, más un margen de seguridad para evitar quedarse sin stock en caso de que la demanda sea ligeramente superior a la habitual. La comida que no se gasta en un ciclo tiene un día más para gastarse si es del primer ciclo (hay que gastarlo el viernes si se ha traído el lunes) o dos días si es del segundo ciclo (hay que gastarlo entre el lunes y el martes si se ha traído el viernes de la semana anterior).

Una vez conocido el funcionamiento del sistema propuesto, se ha dividido el sistema en tres módulos para facilitar el análisis y desarrollo de cada uno de ellos:

- Módulo de sensor
- Módulo de comunicaciones
- Módulo de servidor

2.2 Análisis del módulo de sensor

El objetivo del sensor es tomar medidas y generar datos y está situado dentro de la cámara frigorífica. Para realizar el desarrollo del sistema se ha utilizado tres tipos de comida que serán carne de vacuno, porcino y de ave, por lo que se necesitarán tres sensores. Cada sensor está compuesto por una celda de carga montada como si fuera una báscula.

Una celda de carga [4] es un transductor que crea una señal eléctrica cuya magnitud es directamente proporcional a la fuerza aplicada sobre ella. Se basa en el uso de galgas extensiométricas con efecto piezorresistivo, es decir, que el valor nominal de su resistencia cambia cuando al material conductor que lo forma se le someten fuerzas de tensión o compresión, tal y como muestra la siguiente imagen:

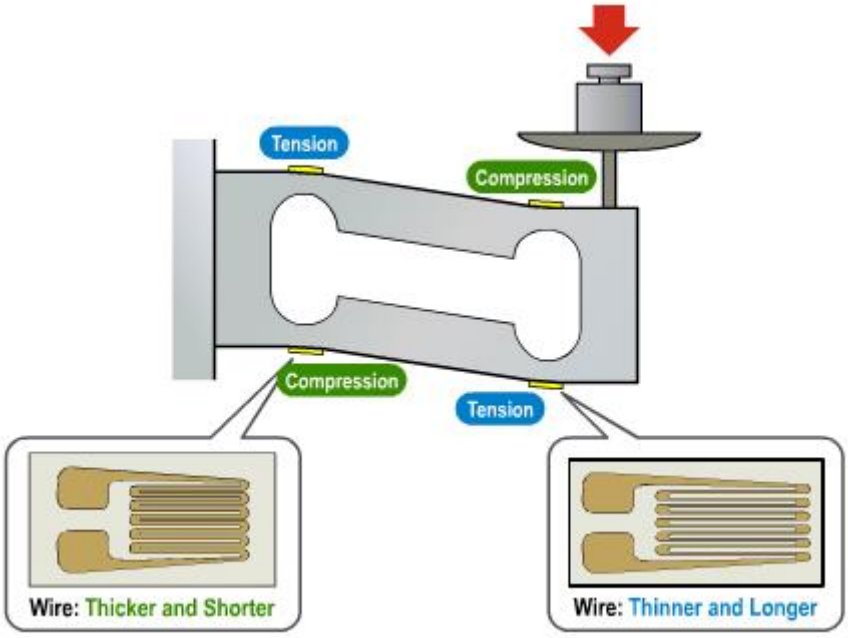

Figura 5. Gráfica del efecto de la tensión y la compresión en una celda de carga

Existen multitud de tipos de celdas de carga. Los más comunes son los de un solo punto, tipo S o de compresión:

- Las celdas de un solo punto se utilizan en pesajes de pequeña escala con una capacidad de carga entre 0.1 y 50 kg. Es la que se usa en básculas de cocina, de joyería, etc.

- Las células de compresión están diseñadas para que la presión se ejerza únicamente sobre un punto. Se pueden encontrar frecuentemente en las básculas de baño en las que se suele montar 2 o 4 celdas de este tipo en las esquinas. Tienen mayor capacidad de carga que las celdas de un solo punto.
- Las celdas de carga tipo S se emplean para pesajes que involucren compresión o tensión a gran escala. Se utiliza entornos industriales.

Figura 6. Tipos de celdas de carga. A la izquierda, una celda de un solo punto; en medio, una celda de compresión y a la derecha, una celda de tipo S

Para el sistema propuesto se ha decidido utilizar la celda de un solo punto ya que ofrece una capacidad adecuada para nuestro uso. La carga máxima se producen los días que el proveedor trae la comida. Se ha comentado que el gasto estimado es de 20-25 kg por semana, lo que supone una carga máxima aproximada de 10-13 kg para cada ciclo. Este tipo de celda también es más fácil de instalar respecto a las de compresión, aparte de que para este último tipo sería necesario adquirir 4 unidades para cada báscula.

Para controlar a los tres sensores se ha utilizado un microcontrolador. Para este proyecto se ha decidido utilizar Arduino [5] porque esta plataforma ofrece el hardware necesario y el IDE (Integrated Development Environment) para programarlo sin necesidad de ningún tipo de tarjeta de programación. Además su bajo precio y la gran comunidad de desarrolladores que trabaja con él lo hacen adecuado para realizar cualquier tipo de prototipo.

Otra de sus ventajas es la gran variedad de tipos de Arduinos existentes lo que ofrece una mayor flexibilidad a la hora de elegir la placa que mejor se adapta a las necesidades de cada proyecto. El Arduino más comúnmente utilizado es el Arduino UNO, que incorpora un microcontrolador ATmega328P [6]. Es el modelo más versátil porque ofrece muchas opciones para alimentarlo, ya sea por USB, o utilizando una fuente de 5V y alimentarlo directamente por este pin o utilizando una fuente externa de entre 6 y 18V por el conector que incorpora en la esquina inferior izquierda. Utilizando este conector, el voltaje es regulado a 5V para alimentar el microcontrolador.

Figura 7. Arduino UNO

Los sensores, al estar ubicados dentro de la cámara frigorífica, no tienen acceso a ninguna fuente de alimentación externa por lo que es necesario alimentarlo con una batería o unas pilas. Sin embargo el Arduino UNO no fue diseñado para utilizarlo en proyectos sin alimentación externa y el regulador LM7805 que lleva incorporado no es de alta eficiencia. En reposo y sólo con el LED de power encendido el Arduino UNO consume 46mA. Esto se traduce en que utilizando 4 pilas AA en serie de 1.5V (consiguiendo los 6V necesarios) y 2100mAh tendríamos una duración de 2100/46 = 46.65 horas, es decir, que duraría menos de dos días sin incluir siquiera el consumo de los otros dispositivos que lleva el sensor.

La placa Arduino más indicada para este tipo de proyectos es el Arduino Pro Mini. Este Arduino contiene la misma cantidad de pines que el Arduino UNO, 13 pines digitales y 6 pines analógicos, que pueden ser utilizados como pines digitales si ninguno de ellos es utilizado para leer un sensor analógico, y su tamaño es una quinta parte de éste. También utiliza el microprocesador ATmega328P y cuenta con un botón de reset. Para este proyecto se ha utilizado un Arduino Mini Pro compatible que tiene disponible dos pines analógicos más, y como no se utiliza ninguna lectura de un sensor analógico, se tiene disponible en total 21 pines. Los pines analógicos, al utilizarse como pines digitales, se enumeran del 14 al 21.

Existen dos versiones del Arduino Mini Pro, uno de 5V/16MHz y otro de 3.3V/8MHz. Se ha decidido utilizar el Arduino de 3.3V por compatibilidad con el resto de dispositivos que conforman el conjunto del sensor. Además el microcontrolador consume menos con una frecuencia de reloj más baja y para la versión de 5V su regulador necesita más de 6V, a los que no se llegaría a medida que se va gastando si utilizamos 4 pilas de 1.5V.

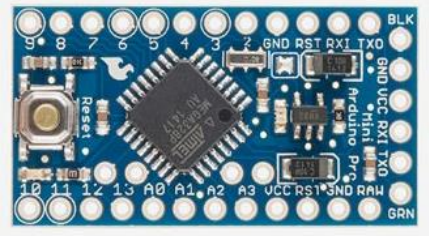

Figura 8. Arduino Mini Pro

Otra de las diferencias entre el Arduino UNO y el Mini Pro es que este último no cuenta con puerto USB para programarlo. Para ello es necesario utilizar un conversor serie-USB. Los seis pines de la derecha que se pueden observar en la imagen inferior se conectan con los seis pines laterales que tiene el Arduino Mini Pro.

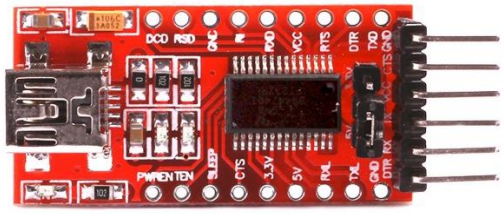

Figura 9. Conversor Serie-USB para programar el Arduino Mini Pro.

El Arduino Pro Mini de 3.3V/8MHz utiliza un regulador MIC5205 y consume 4.7mA en reposo, es decir que duraría unos 18 días con 4 pilas de 2100mAh. Este tampoco parece una duración suficiente para un sistema cómodo de utilizar. Sin embargo se pueden aplicar técnicas para bajar el consumo como ponerlo en modo dormido o bajar la frecuencia de reloj. Esto se explicará más en profundamente en los sucesivos apartados.

La ratio de salida típica de una celda de carga es de 1mV por cada voltio de entrada utilizando la máxima carga admisible por la celda. Por lo tanto si se excitara con 3.3V una celda con capacidad máxima de 20kg y colocamos 10kg sobre ella, la salida de la celda será de (10/20)*5*1mV = 1.65mV. Para tener una precisión de 10 gramos, la salida obtenida sería de 1.64835mV, con una diferencia de solo 1.65uV. El Arduino utiliza un ADC (Analog-to-Digital Converter) de 10 bits, proporcionando 1024 niveles digitales, lo que supone una precisión de medición de +-1.61mV. Esta precisión está lejos de la necesaria para medir los cambios de voltaje de la celda de carga. Por ello es necesario amplificar la señal de la celda de carga antes del Arduino.

Lo más común para utilizar con células de carga es el ADC HX711 [7] que incluye una etapa amplificadora seguido por un ADC ΣΔ de 24 bits. Por lo tanto, ofrece directamente una salida digital para el Arduino. Este ADC utiliza como entrada una señal de reloj externa y su máxima tasa de salida es de 80 lecturas por segundo.

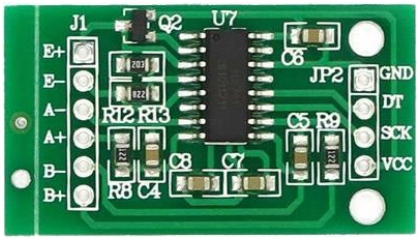

Figura 10. ADC HX711

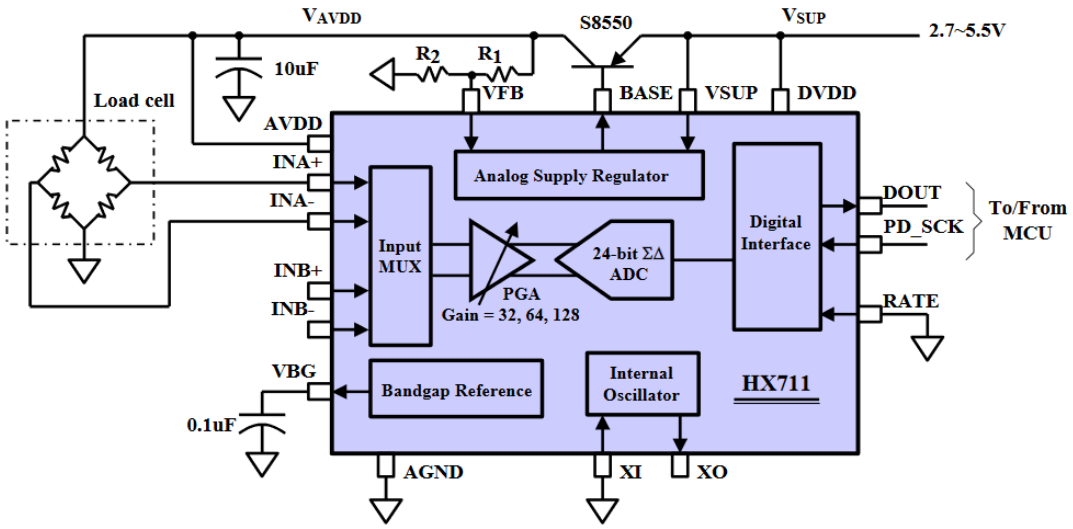

Figura 11. Diagrama de bloques del ADC HX711

Se ha comentado antes que es necesario buscar una maximización en la duración de la batería. El Arduino no tiene necesidad de estar midiendo el peso en todo momento. Aunque el HX711 pueda dar 80 mediciones por segundo, el sistema solo necesita una lectura cuando se produzca un cambio en el peso. Por lo tanto el resto del tiempo el Arduino puede estar en modo dormido, lo que proporcionaría un consumo de 0.9mA, 5 veces menor que en modo activo. Utilizando el modo dormido, con 4 pilas dando una capacidad de 2100mAh obtendríamos una duración de más de tres meses (97 días). Hay que tener en cuenta que en el Arduino no es posible deshabilitar el power LED y éste permanece siempre encendido con un consumo de 0.85mA. La única forma de ganar ese consumo es desoldando el LED o cortando la traza que lo conecta con el microcontrolador. Tras eliminar este LED, el Arduino Mini Pro en modo dormido tendría una duración superior a 4 años. En este proyecto no se ha realizado este proceso porque poniendo el Arduino en modo dormido y con el power LED activo ya ofrece suficiente autonomía para probar el prototipo.

Hay que destacar que el consumo de 0.9mA es únicamente de un Arduino Mini Pro en modo dormido. A estos 0.9mA se le deberán añadir el consumo del resto de los dispositivos del sensor, como por ejemplo el ADC HX711 comentado anteriormente. También hay que recalcar que el Arduino no está continuamente en modo dormido ya que cuando deba realizar una medida saldrá de este modo el tiempo que necesite. Cuanto más se minimice el tiempo para tomar una medida, mayor será la duración total del sistema. Por lo tanto, la duración obtenida para el prototipo no es tan amplia como los 97 días que se ha calculado. Se realizará un análisis detallado del consumo final en el capítulo de pruebas y resultados.

Para el prototipo del sistema se ha elegido Arduino, pero una vez superada la fase de pruebas, para la fabricación y comercialización podría ser más conveniente el uso de sensores específicamente diseñados para para proyectos IoT de bajo consumo energético como la gama Waspmote [8] de Libelium. Estos sensores pueden gastar tan poco como 7 µA en modo dormido, un consumo 100 veces menor que el del Arduino Mini Pro. En el apartado de líneas futuras se darán más detalles de estos sensores.

Cuando el Arduino entra en modo dormido, es necesario despertarlo con algún tipo de evento. No es posible utilizar el cambio de peso como evento ya que el Arduino detecta el evento como el cambio de un nivel en uno de sus pines digitales, pero el HX711 es un ADC cuya salida es un stream de datos de 24bits. Además esto impediría poner en modo dormido también el HX711 y el consumo requerido sería notablemente mayor.

Por este motivo se ha utilizado un reed switch cuya finalidad es detectar la apertura y el cierre de la puerta de la cámara frigorífica ya que mientras ésta esté cerrada es imposible que el peso del material que hay dentro pueda cambiar. Un reed switch es un interruptor que cambia de estado (abierto o cerrado) en presencia de un campo magnético. Está formado por un par de contactos ferrosos que suelen estar abiertos y cuando se acerca un campo magnético (un imán por ejemplo) se cierran obteniendo de esta forma un interruptor cerrado. El reed switch se instalaría en el marco de la puerta y el imán se instalaría en la puerta de forma que entren en contacto cuando la puerta esté cerrada. Se considera que entran en contacto cuando las dos partes estén a una distancia menor a unos dos centímetros.

Figura 12. Reed switch magnético

Lo que se desea es que el Arduino tome la medida una vez el cocinero termine de sacar una ración. Por lo tanto sería incorrecto tomar la medida justo después de abrir la puerta, antes de que el cocinero saque nada, y también sería incorrecto tomar la medida mientras el cocinero esté sacando la comida porque podría estar detectando el peso que ejerce la presión de la mano. Para evitar estas lecturas erróneas, se ha fijado que el sistema tome la medida cuando la puerta se cierre, lo que significa que el cocinero ya ha terminado de sacar lo que necesitaba. Por lo tanto el Arduino se despierta cuando la puerta de la cámara frigorífica se cierra, no cuando se abre.

Finalmente, el conjunto sensor incorpora también un display para que el personal de la cocina pueda ver el peso de la comida que hay en cada báscula. No es tarea fácil coger una porción adecuada para cada ración cuando se deben preparar muchas raciones a la vez, por lo que el display es especialmente útil para ver si la cantidad que se ha cogido se corresponde efectivamente a la cantidad que se debe emplear. Para habilitar el display, el Arduino no puede

estar en modo dormido. Pero el Arduino sólo se despierta con el reed switch cuando la puerta se cierra y esto imposibilitaría poder ver el display. Por lo tanto se ha habilitado también un pulsador para activar manualmente el sensor y poder mostrar el peso en el visualizador cuando el cocinero lo crea oportuno.

Existen muchos tipos de displays en el mercado, de los cuales se han planteado dos tipos para utilizar en este proyecto: una pantalla LCD monocroma y un visualizador de 7 segmentos. Ambos son adecuados para el sistema planteado ya que sólo se desea mostrar el peso que hay en cada báscula, para lo cual solo se necesita imprimir números.

Una pantalla LCD es un componente que lleva integrado un controlador de la matriz de cristal líquido que compone el visualizador. Sin embargo la mayoría de ellos requiere 5V para un correcto funcionamiento y habría sido necesario utilizar un conversor de DC si se utilizara el Arduino de 3.3V.

Figura 13. Pantalla LCD de 2 filas de 16 caracteres

Por ello finalmente se ha decidido utilizar el visualizador de 7 segmentos. Un visualizador de 7 segmentos está compuesto internamente por LEDs, cada uno de los cuales ilumina uno de los siete segmentos de cada dígito. Por ejemplo para representar el número 1, se iluminarían los dos segmentos verticales de la derecha. En el sensor del sistema se ha utilizado un visualizador de un dígito y otro de cuatro, tal y como se muestra en las siguientes imágenes. El visualizador de cuatro dígitos mostrará el peso que hay en el número de báscula mostrado en el de un dígito. El peso representado será con dos cifras decimales ya que la comida que hay en cada báscula no va a pasar de los 99 kilogramos.

Figura 14. Visualizadores de 7 segmentos. A la izquierda de un dígito y a la derecha de 4 dígitos.

Para conectar estos dos componentes al Arduino haría falta un pin por cada LED, incluyendo también el LED de punto decimal, y además 5 pines, cada uno para controlar cuándo se enciende cada dígito, por lo que serían necesarios 13 pines. Sin embargo, tras conectar los

componentes necesarios, no se dispone de tantos pines libres. Por ello se ha utilizado un registro de desplazamientos serie-paralelo (SIPO: Serial Input, Parallel Output). Concretamente se ha utilizado el 74HC595 [9], de 8 bits, que es justo el número de bits que necesita cada dígito. De esta forma en vez de los 8 pines que se necesitarían para cada dígito, sólo se necesitarían 3; dos para los dos relojes que necesita y uno para la entrada de datos.

Finalmente el conjunto del sensor incorpora también un botón para establecer la puesta a cero del peso que hay sobre la báscula ya que se quiere medir únicamente el peso de la comida que hay, no el peso de los contenedores.

Todos estos elementos se han montado sobre dos placas de pruebas estándar de 840 puntos. La sección de desarrollo e implementación del sensor incluye un diagrama con la distribución y conexión de todos los componentes mencionados.

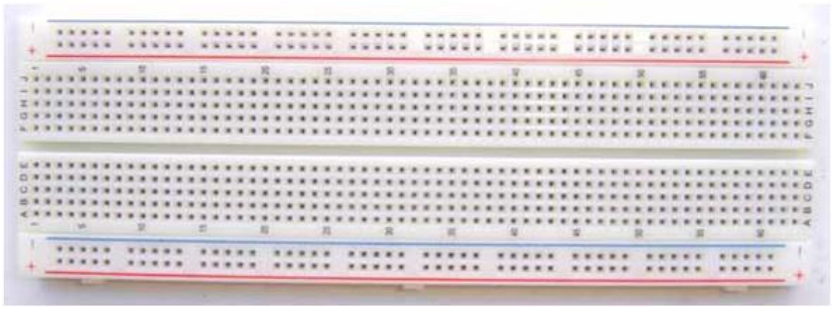

Figura 15. Placa de pruebas

2.3 Análisis del módulo de comunicaciones

Como el sensor está dentro de la cámara y el servidor no, es necesaria una comunicación inalámbrica para enviar los datos del sensor. La comunicación necesaria es unidireccional ya que el sensor solo toma medidas y las envía. No necesita ninguna información proveniente del servidor.

Para implementar la comunicación se ha decidido utilizar la tecnología ZigBee [10]. Otras opciones que también se han analizado han sido WiFi [11] y Bluetooth 4.0 [12]. Las tres ofrecen módulos fácilmente integrables con Arduino e incluyen el modo dormido en el que los módulos tienen un consumo de pocos microamperios (6 microamperios para WiFi, 60 para Bluetooth 4.0 y 3.5 para ZigBee). Finalmente se ha elegido ZigBee porque es una tecnología ya conocida y utilizada previamente y porque fue diseñada específicamente para este tipo de aplicaciones: envío de pocos datos (sin requerimientos de una alta tasa de transmisión) a una distancia no muy lejana (decenas o pocas centenas de metros) y permanecer la mayor parte del tiempo en modo dormido para minimizar el consumo energético y despertándose exclusivamente para sensar un dato y enviarlo.

Los módulos ZigBee más utilizados son comercializados por Digi [13] y se denominan XBee.

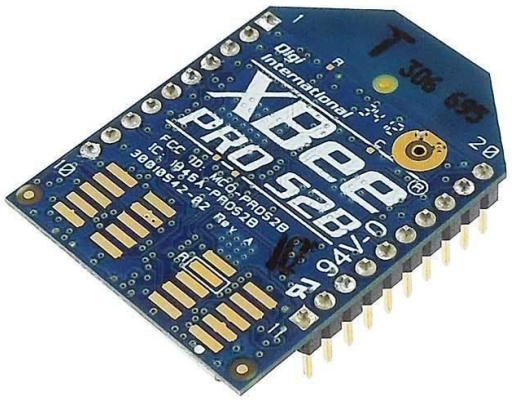

Figura 16. XBee PRO S2B

Para que la comunicación inalámbrica sea posible, se debe disponer al menos de dos dispositivos ZigBee. Este protocolo define tres tipos de dispositivos: ZigBee Coordinator, ZigBee Router y ZigBee End Device. En toda red ZigBee debe existir un Coordinator que establece los parámetros de la comunicación. Los End Device se pueden comunicar con otros dispositivos de la red ZigBee pero toda la comunicación relega en su padre, que puede ser un Router o un Coordinator. También puede entrar en modo dormido para reducir su consumo a tan solo 3µA, por lo que el rol de End Device es el más adecuado para el sensor.

Para establecer la comunicación, el Coordinator irá conectado al TPV (Terminal Punto de Venta) del restaurante. Para ello es necesario una placa que permita conectar un módulo XBee a un ordenador. En concreto se ha utilizado un XBee Explorer USB sobre que el que se monta un módulo XBee y ofrece una conexión microUSB. Lleva montado un conversor serie-USB.

Figura 17. XBee Explorer USB

En el sensor, la conexión entre el XBee y el Arduino Pro Mini se ha realizado con un XBee Breakout Board. Esta placa solo adapta la separación de las patas del módulo XBee (2.0 mm) a la placa de pruebas (2.54 mm).

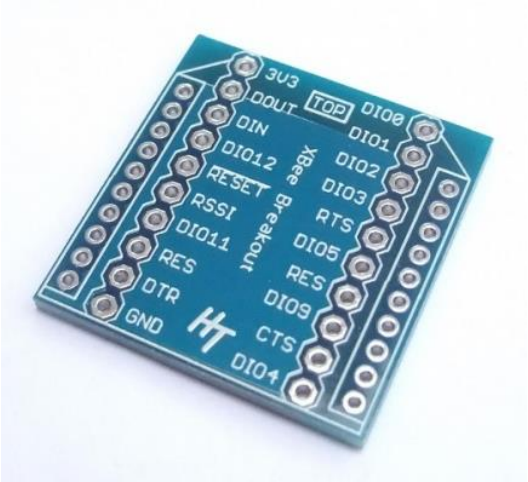

Figura 18. XBee Breakout Board

Una vez finalizado el desarrollo del sistema, la comunicación inalámbrica que se desea realizar es entre el sensor y el ordenador TPV del restaurante situado en la barra de la entrada. La distancia entre ambos es de aproximadamente 80 metros. Para esta distancia la comunicación de podría funcionar sin problemas solo con el Coordinator y el End Device, pero se ha añadido también un Router a la topología por si la distancia pudiera ser mayor y para probar su correcto funcionamiento en el caso de que se añadiera un Router, estableciendo una topología multisalto. Si la distancia lo exigiese se podrían añadir más Routers.

Tanto el Coordinator como el Router son módulos que no tienen restricciones energéticas ya que están fuera de la cámara frigorífica. El Router está formado por un Arduino UNO y el módulo XBee va montado sobre él con un Arduino XBee Shield. Las shields de Arduino están diseñados para usarlos muy fácilmente con el Arduino UNO ya que solo hay que montarlo encima.

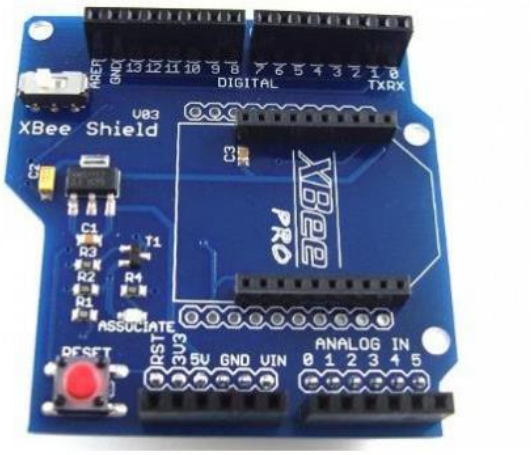

Figura 19. Arduino XBee Shield

El software utilizado para programar los XBee y probar su correcto funcionamiento es el XCTU de Digi. Este permite definir el rol de los XBee, cambiar sus parámetros de configuración y probar su funcionamiento mediante un monitor serie de los puertos COM a los que van conectados los XBee.

Finalmente cabe destacar que los XBee funcionan con un voltaje de 3.3V, razón por la cual se ha elegido la versión de 3.3V del Arduino Mini Pro. Si el Arduino elegido fuera de 5V sería necesario utilizar un level shifter entre ellos para que el Arduino no interprete como 0 un 1 lógico de XBee y para que el XBee no sea dañado cuando el Arduino envíe un 1.

2.4 Análisis del módulo de servidor

El servidor tiene como objetivo las siguientes funcionalidades:

- Guardar los datos enviados por el sensor en una base de datos.
- Procesar los datos para realizar los pedidos según el ciclo que corresponda y con la cantidad correcta para poder satisfacer la demanda prevista y evitar que se generen despilfarros.
- Ofrecer una interfaz a través de la cual el encargado puede controlar y supervisar la cantidad de material que hay en la cámara.

Para cumplir con estos objetivos se han utilizado diversas tecnologías que se verán en este apartado.

El servidor elegido es XAMPP [14] por ser una plataforma de software libre que integra el servidor web Apache, un sistema de gestión de base de datos MySQL y los intérpretes de lenguajes de scripting PHP y Perl. Sus capacidades multisistema, simbolizado por la X al inicio del acrónimo, funcionando en sistemas Windows, GNU/Linux y Mac OS, y su facilidad de instalación y uso lo hacen adecuado para este tipo de proyectos.

En el subapartado anterior se ha explicado que los datos que se deben almacenar se mandan desde un sensor utilizando XBee y el equipo con el servidor cuenta con un receptor XBee conectado a uno de sus puertos USB. Por lo tanto la recepción de datos se produce desde un puerto USB y el objetivo es leer datos entrantes por este puerto y guardarlo en la base de datos. Para ello se ha hecho un script de Python [15] (addData.py) que está siempre en ejecución y cuando lee un dato entrante por el puerto USB correspondiente, lo guarda en la base de datos MySQL.

El procesamiento de datos para realizar pedidos también se ha realizado con un script de Python (stockOrder.py) que se ejecuta automáticamente los días en los que se realizan los pedidos, calcula la cantidad a pedir y envía un correo al proveedor adecuado.

Para la el desarrollo y la prueba del sistema el servidor XAMPP se ha instalado en el equipo de desarrollo, en localhost, pero el sistema funcionaría igualmente si estuviera instalado en cualquier otro lugar con un servicio de hosting. Tan solo sería necesario proporcionar el dominio y el puerto de conexión a los scripts de Python. Sin embargo, estos scripts sí que deben ejecutarse en el equipo con el receptor XBee ya que leen los datos procedentes del puerto USB.

Finalmente se ha realizado también una interfaz web para que el encargado pueda ver los datos generados utilizando cualquier navegador web. Para esto se han utilizado las tecnologías web más populares:

- HTML (Hypertext Transfer Protocol) para mostrar el contenido
- CSS (Cascading Style Sheets) para realizar el styling o el diseño gráfico de la página
- PHP (PHP: Hypertext Preprocessor) para realizar cierto tipo de procesado necesario en las páginas

Para mostrar los datos de forma gráfica se ha integrado HighCharts [16], que es una librería escrita en JavaScript y permite dibujar todo tipo de gráficas e interactuar con ellas.

Finalmente se muestra un esquema de la arquitectura y las tecnologías y materiales utilizados en el sistema diseñado.

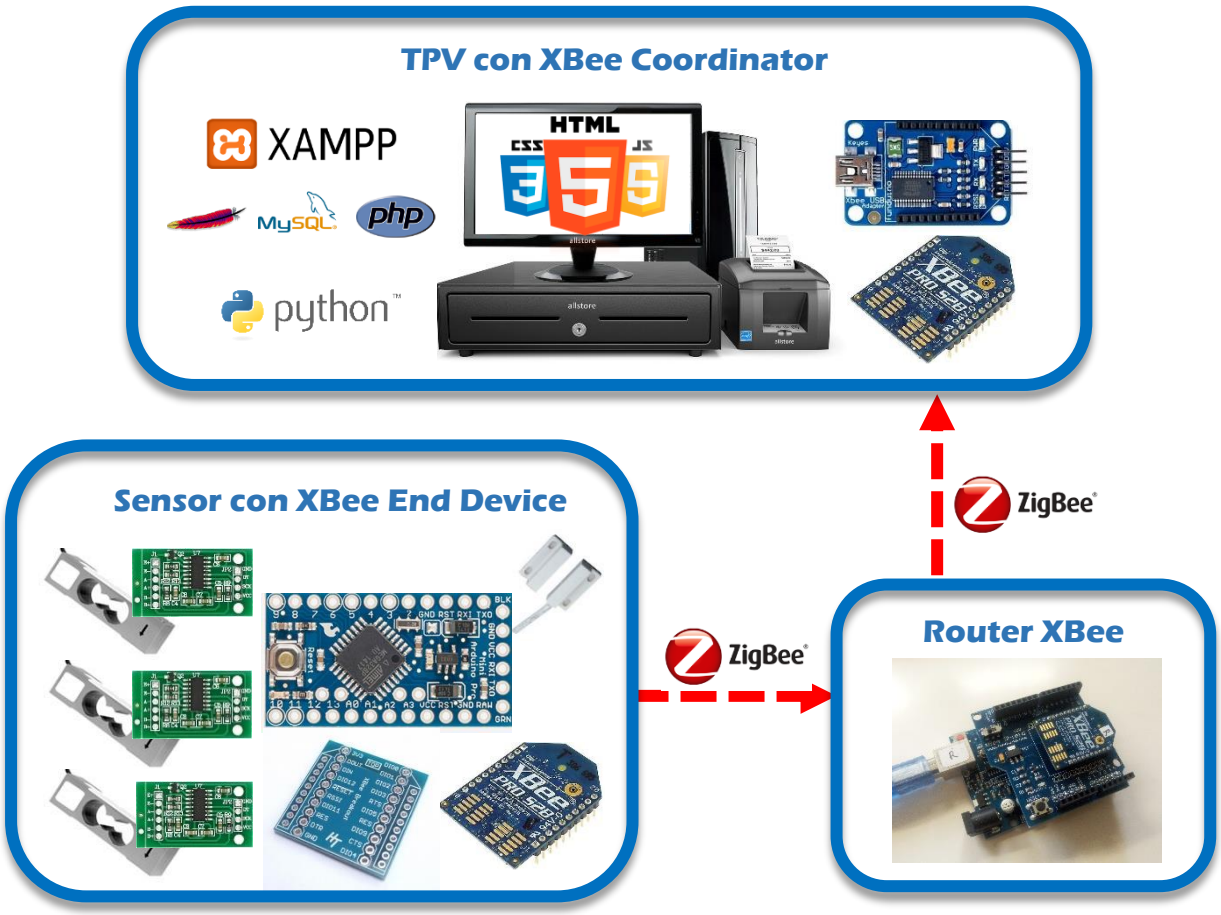

Figura 20. Esquema de la arquitectura, materiales y tecnologías utilizadas

3 Desarrollo e implementación del sensor

En este y los sucesivos apartados se describirá el desarrollo e implementación de cada uno de los tres módulos en los que se ha dividido el sistema.

El elemento principal del sensor es el Arduino Mini Pro, que controla el funcionamiento del sensor.

Un Arduino no es más que una placa que lleva integrado un MCU (microcontroller unit), en el caso del Mini Pro un ATmega328P, junto con los componentes necesarios para funcionar. Lo que lo hace distinto a un chip ATmega328P es que lleva el bootloader de Arduino grabado en la memoria. Este bootloader es el que permite utilizar el IDE que ofrece la plataforma. El lenguaje de programación utilizado es C.

El Arduino Mini Pro de 3.3V utiliza una frecuencia de reloj de 8MHz y no de 16MHz porque así lo especifica el SOA (Safe Operating Area) en el datasheet del microcontrolador.

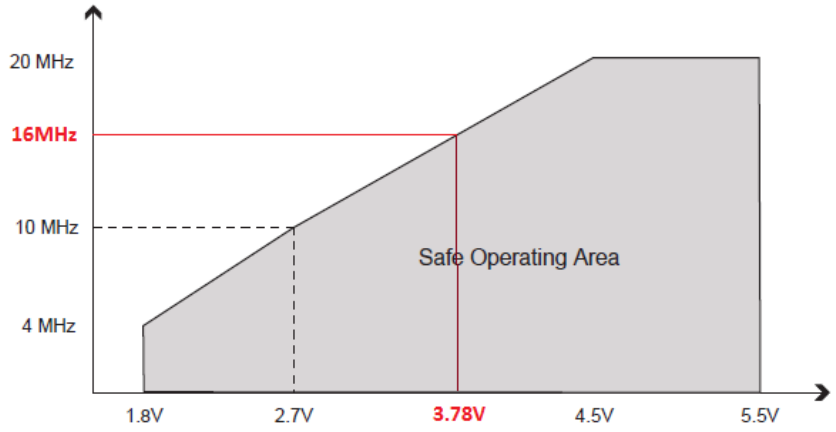

Figura 21. Safe Operating Area del microcontrolador ATmega328P

Un programa de Arduino se denomina sketch y éste contiene siempre al menos las siguientes dos funciones que son necesarias para hacer funcionar un Arduino:

- void setup()

  Esta función se ejecuta únicamente cuando se inicializa el sketch. Aquí se definen los pines que van a ser utilizados y su modo (INPUT u OUTPUT) y se inicializan las variables. Esta función se ejecuta una única vez al encenderse o resetear el Arduino.

- void loop()

  Esta función se ejecuta en bucle y contiene el código que se ha de ejecutar en cada iteración.

En la siguiente imagen podemos ver el montaje del sensor sobre la placa de pruebas.

Arriba del todo y sin estar fijados a la placa de pruebas están los HX711. A la izquierda podemos ver el botón utilizado para poder despertar el sensor para mostrar el peso sobre las básculas cuando se abre la cámara frigorífica. Después podemos ver un botón que será utilizado para la puesta a cero de las celdas de carga. A la derecha de los botones se encuentra el Arduino Mini Pro. Después a la derecha tenemos el módulo XBee End Device. En la parte inferior, a la izquierda podemos ver los dos visualizadores de 7 segmentos junto con las resistencias de protección para los LEDs y el Arduino y también el registro de desplazamientos para controlar el los segmentos que se encienden para mostrar un número. Podemos observar que para alimentar el sensor se utilizan 4 pilas AA de 1.5V cada una, con una capacidad total de 2100mAh.

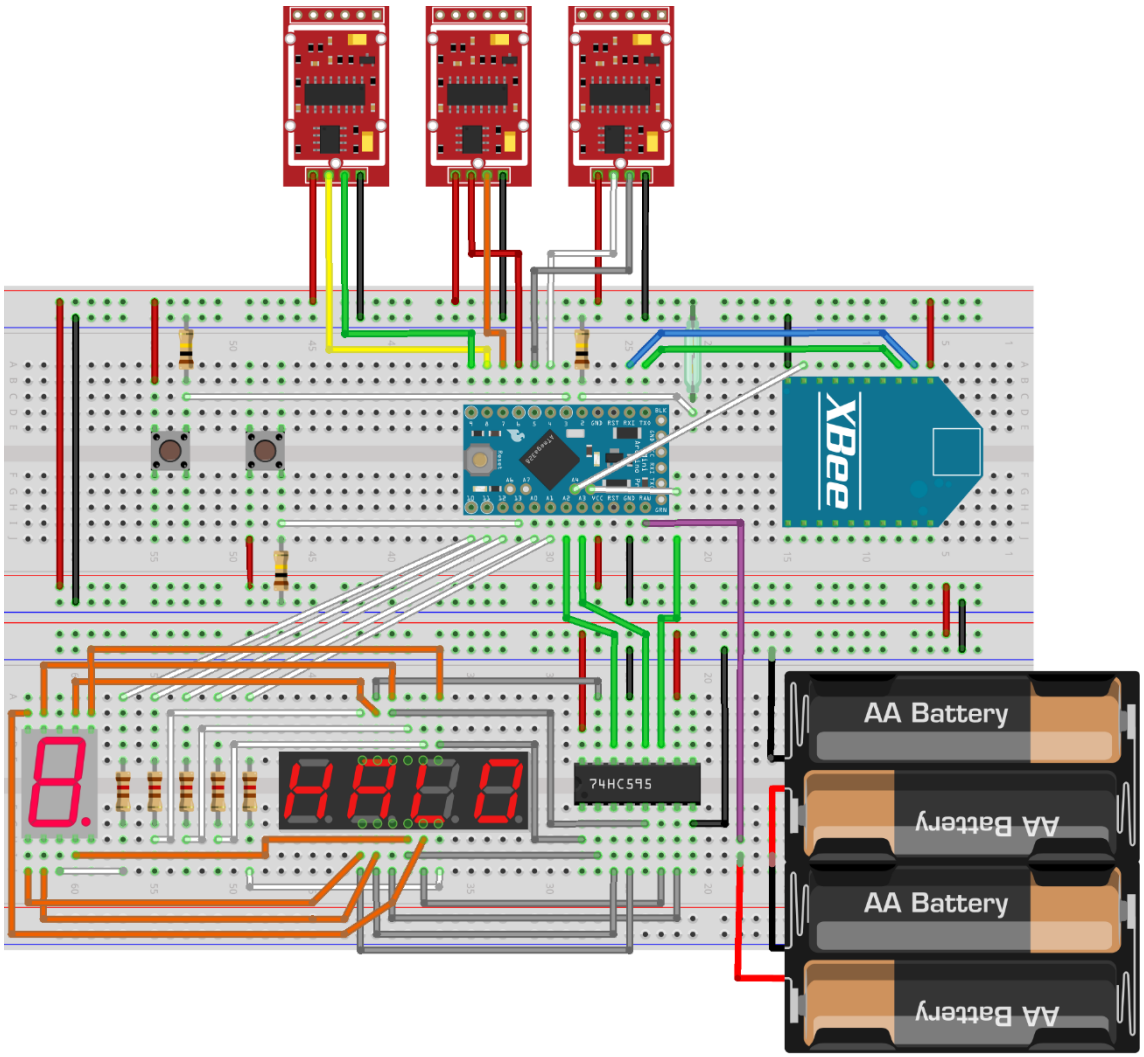

Figura 22. Diagrama del montaje del sensor sobre la placa de pruebas

El módulo del sensor se divide en dos bloques principales que son el ahorro del consumo energético con el modo dormido y el sensado del peso con las celdas de carga. También se explica brevemente la implementación del display de 7 segmentos.

3.1 Modo dormido

Como el sensor va dentro de la cámara frigorífica y utiliza pilas para su alimentación, es fundamental optimizar el consumo de éste. Por lo tanto el sistema está desarrollado de forma que tanto el Arduino, el módulo de XBee, los ADC HX711 y los visualizadores de 7 segmentos drenen la mínima corriente posible. La forma de conseguir esto es implementando el modo dormido que incluyen los dispositivos utilizados. Tanto el Arduino, como el XBee y el HX711 disponen de un modo de bajo consumo. El visualizador se puede apagar simplemente apagando los leds que lo componen. En este apartado se explicará cómo es el modo dormido de cada dispositivo y cuándo entran en funcionamiento dentro del sistema. En el apartado de pruebas y resultados se comentarán los consumos obtenidos tras implementar estos métodos.

El microcontrolador del Arduino Mini Pro, el ATmega328P, tiene varios modos dormido para apagar los módulos que no sean utilizados y ahorrar energía. En su datasheet vienen explicados los distintos modos de baja energía que tiene, cómo activarlos y cómo despertarlo de estos modos. En la siguiente tabla se muestra todos los modos disponibles y los módulos que quedan habilitados en cada uno de ellos.

Tabla 1. Modos de bajo consumo del ATmega328P

	Active Clock Domains					**Oscillators**		**Wake-up Sources**						
Sleep Mode	clk_{CPU}	clk_{FLASH}	clk_{IO}	clk_{ADC}	clk_{ASY}	Main Clock Source Enabled	Timer Oscillator Enabled	INT1, INT0 and Pin Change	TWI Address Match	Timer2	SPM/EEPROM Ready	ADC	WDT	Other/O
Idle			X	X	X	X	X(2)	X	X	X	X	X	X	X
ADC Noise Reduction				X	X	X	X(2)	X(3)	X	X(2)	X	X	X	
Power-down								X(3)	X				X	
Power-save					X		X(2)	X(3)	X	X			X	
Standby(1)						X		X(3)	X				X	

Notes: 1. Only recommended with external crystal or resonator selected as clock source.
2. If Timer/Counter2 is running in asynchronous mode.
3. For INT1 and INT0, only level interrupt.

El modo que tiene un menor consumo es Power-down. En este solo permanecen activos la detección de eventos en los pines de interrupción (INT0 e INT1), en el TWI (Two-Wire Serial Interface) y el WDT (Watch Dog Timer). Este es el modo que hay que utilizar si mientras el Arduino permanece dormido no tiene que hacer nada y su único objetivo es ahorrar la máxima energía posible. Por lo tanto este es el modo que se ha utilizado para el sistema. Para salir del modo dormido se han utilizado ambos pines de interrupción (INT0 e INT1). Estos pines se corresponden con los pines 2 y 3 respectivamente del Arduino Mini Pro.

Con el Arduino, para habilitar el modo dormido y configurar las interrupciones, se puede escribir directamente en los registros del microcontrolador, concretamente en los registros SMCR (Sleep Mode Control Register), MCUCR (MCU Control Register) y PRR (Power Reduction Register); o utilizar las librerías que implementa Arduino [17]. En este caso se han utilizado las librerías.

El XBee también tiene varios tipos de modo dormido. El modo dormido es uno de los componentes más importantes de XBee ya que el protocolo Zigbee fue diseñado para enviar datos sólo cuando hay que enviarlos y el resto del tiempo permanecer en modo dormido para consumir la menor cantidad de batería posible. El modo dormido se configura con el configurador XCTU de Digi. En el apartado de desarrollo del módulo de comunicaciones se explicará más a fondo los tipos de modo dormido que dispone y cómo se ha configurado para su funcionamiento. Para este sistema, en la que se dispone de un MCU para controlar la ejecución del sensor, es más conveniente que los modos dormidos sean controlados por éste en vez de por el XBee. Por lo tanto, el XBee se duerme y se despierta según un pin que controla el Arduino.

El HX711 también tiene un modo de bajo consumo. Tal y como se explica en su datasheet, el modo dormido se activa con el pin SCK. Cuando este pin cambia de LOW a HIGH y permanece 60µs en este estado, el HX711 entra en modo dormido. Se despierta cuando el pin SCK vuelve a LOW.

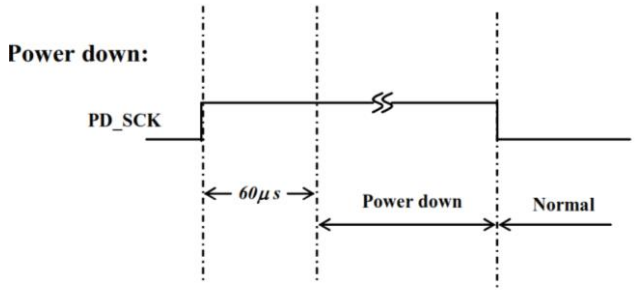

Figura 23. Activación del modo dormido del HX711

El Arduino controla y activa el modo dormido tanto del resto de los componentes como el del suyo propio. En el apartado de análisis se ha explicado que el evento utilizado para despertar el Arduino es el cierre de las puertas de la cámara frigorífica, que se detecta con un switch magnético. Este reed switch va conectado al pin 2 del Arduino, correspondiente a la interrupción INT0.

Si el personal de la cocina desea ver el peso de cada báscula o realizar una calibración, el Arduino deberá estar también despierto, lógicamente, sin que la puerta esté cerrada. Para ello, el sensor cuenta con un botón para despertar el Arduino manualmente tras presionarlo. Este botón está conectado al pin 3 del Arduino, que es la interrupción INT1.

Tanto el reed switch como el botón se conectan al Arduino con una resistencia del pull-down de 100kΩ. La resistencia de pull-down se utiliza para que mientras el interruptor esté abierto, la entrada del Arduino no esté en estado de alta impedancia, sin determinar si el estado

es alto o bajo. Por lo tanto cuando el interruptor esté abierto, el Arduino verá un ‘0’ y cuando se cierre, verá un ‘1’.

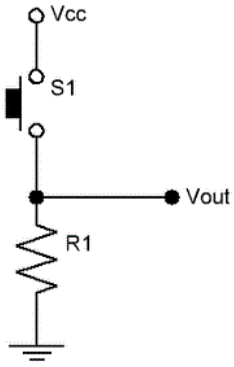

Figura 24. Resistencia de pull-down

El interruptor abierto se corresponde con la puerta de la cámara abierta o con el botón sin presionar. Por lo tanto los eventos que despertarán al Arduino serán un flanco de subida en INT0 o INT1, que se producirá en el instante en que se cierra la puerta o se presiona el botón.

Tras detectar un evento de interrupción, el Arduino sale del modo dormido y acto seguido despierta a los tres ADC HX711, ya que independientemente de por qué se ha despertado se van a tomar medidas de peso. Después evalúa si se ha despertado por haberse cerrado la puerta o por haberlo despertado manualmente con el pulsador. Si se ha activado por haberse cerrado la puerta, entonces toma la medida del peso de cada una de las tres básculas y las prepara para el envío de datos. En este momento el Arduino despierta el XBee End Device y se envían los datos. Tras enviar los datos, el Arduino duerme los HX711, luego duerme el XBee y finalmente se duerme a sí mismo.

Los componentes sólo permanecen despiertos el tiempo necesario para tomar la medida y realizar el envío de datos, ya que en este modo el consumo energético es bastante elevado. Se ha fijado un tiempo de 500ms para cada envío.

Si el Arduino se ha despertado por el botón entonces se enciende el display de 7 segmentos y se habilita la posibilidad de calibrar las tres básculas. Posteriormente cuando se cierra la puerta, el Arduino prosigue con la toma de medidas y el envío de datos.

A continuación se puede ver un diagrama de flujo del funcionamiento del sensor.

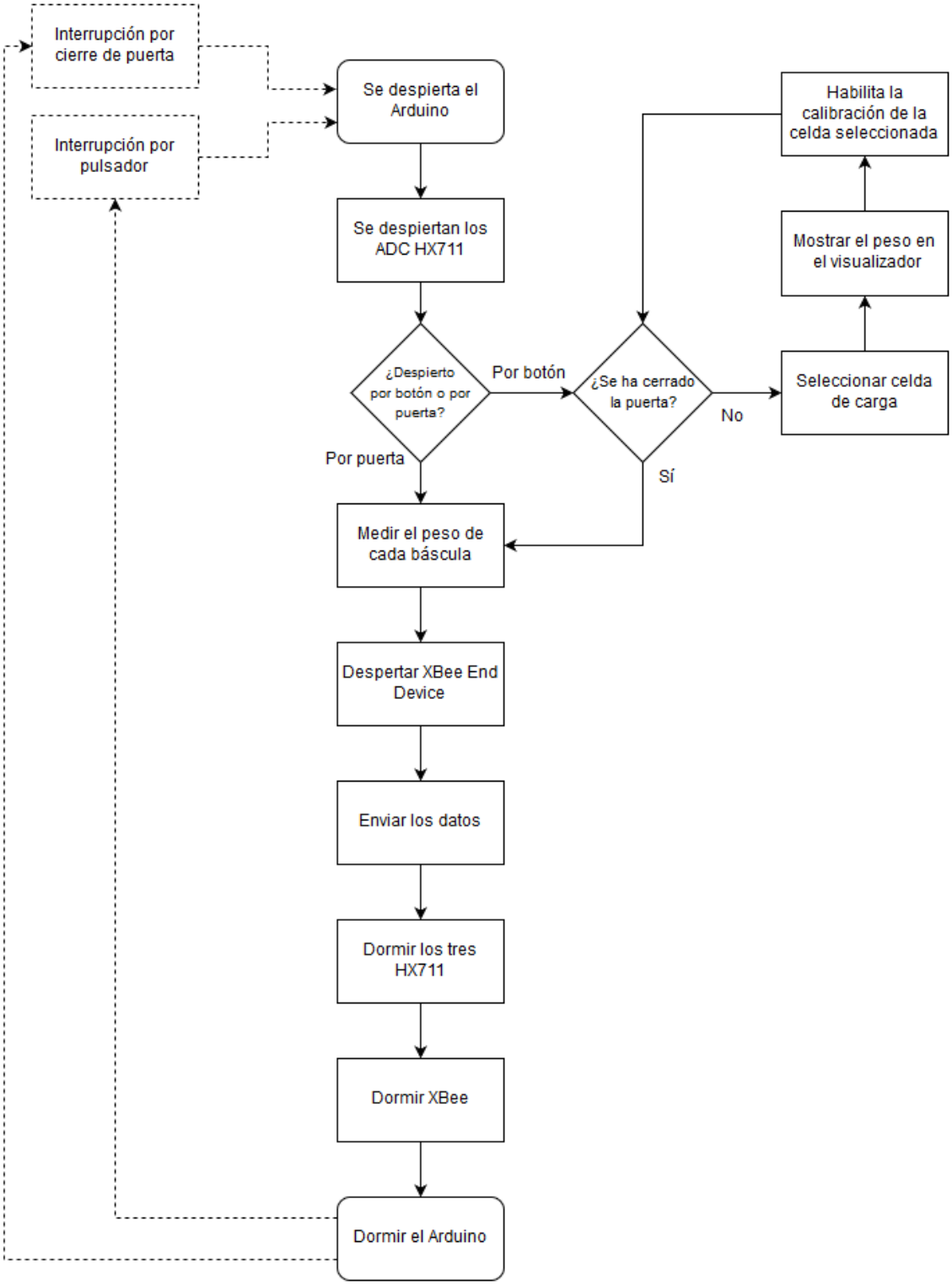

Figura 25. Diagrama de flujo del funcionamiento del sensor

Para mostrar el peso sobre las básculas se han utilizado dos displays de 7 segmentos; uno de un dígito para mostrar el número de la báscula seleccionada (1, 2 o 3) y uno de 4 dígitos que muestra el peso sobre esa báscula, con dos cifras decimales.

Tal y como se puede observar en el diagrama de flujo, cuando el Arduino se despierta manualmente con el botón, entra en un bucle hasta que se cierre la puerta. Mientras la puerta siga abierta, por un lado se muestra en el visualizador de 7 segmentos el peso de la báscula seleccionada y por el otro, si se pulsa el botón de recalibración, el HX711 de la báscula indicada en el visualizador fija el nuevo peso como cero kilogramos. Esto se explicará más en profundidad en el siguiente subapartado.

La conexión de los elementos necesarios para disminuir el consumo energético se muestra en la siguiente tabla:

Tabla 2. Conexión de los elementos para disminuir el consumo energético

Elemento	Pin de Arduino	Tipo de pin desde el punto de vista del Arduino	Color de cable utilizado
Reed switch que despierta el Arduino cuando se cierra la puerta	2	INPUT	Blanco
Botón para despertar el Arduino manualmente	3	INPUT	Blanco
Conexión al pin del XBee que lo duerme	18	OUTPUT	Blanco

3.2 Sensado de peso

La medida del peso se obtiene de la celda de carga y el ADC HX711. Una celda de carga tiene 4 cables de salida, dos correspondientes a la alimentación del circuito (VCC y GND) y los otros dos a la salida balanceada. Anteriormente se ha comentado que una celda de carga funciona a través de un conjunto de resistencias cuyo valor cambia de forma proporcional a la presión aplicada. Estas resistencias están dispuestas con una estructura de puente de Wheatstone, tal y como se muestra en la siguiente imagen.

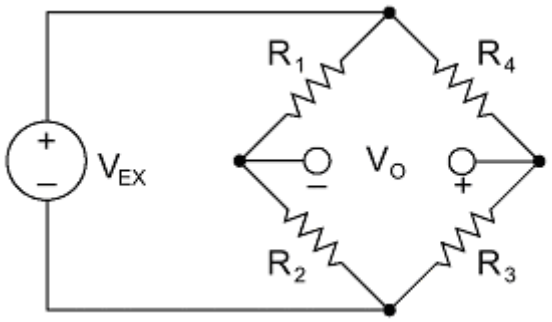

Figura 26. Puente de Wheatstone representando una celda de carga

En este circuito las 4 resistencias tienen el mismo valor. Por lo tanto, cuando no se aplica ninguna presión, ambas salidas, que son dos divisores de tensión, tienen el mismo valor. Al aplicar una presión, tal y como se ha mostrado en la figura 5, las resistencias que sufren

compresión (R2 y R4) disminuyen su valor y las que sufren tensión (R1 y R3) aumentan su valor. Como consecuencia de esto la salida Output+ aumenta su valor y la Output- disminuye su valor. Para que la tensión y la compresión sean óptimas es importante que la presión se ejerza sobre el extremo de la celda de carga de un solo punto.

La diferencia entre estas dos salidas, del orden de µV, es la que amplifica el ADC HX711. Por defecto tiene una ganancia de 128, aunque se puede configurar para tener menos amplificación. El ADC proporciona la salida al Arduino mediante dos cables: DOUT y PD_SCK.

Desde el punto de vista del Arduino, DOUT es la entrada por donde entra el flujo de 24 bits y PD_SCK es la salida que proporciona al HX711 el reloj necesario para saber cuándo enviar un dato disponible.

El protocolo de comunicación viene detallado en el datasheet el HX711. En el Arduino, este protocolo estaba ya implementado y se ha importado mediante una librería [18]. La librería incluye funciones para leer datos, calibrar el peso medido y poner el HX711 en modo dormido.

Antes de usar la báscula es necesario calibrarlo para que el peso medido sea el correcto. En cualquier báscula hay dos variables que se debe calibrar: el offset y la escala. Para realizar la calibración se ha creado un sketch de Arduino cuyo único objetivo es obtener estos dos valores para cada una de las celdas de carga.

En este sketch primero se obtiene el offset correcto y después se calcula la escala. El offset es el valor que devuelve el HX711 cuando sobre la báscula no hay ningún peso colocado. Por lo tanto, primero se ejecuta el sketch sin colocar ningún peso encima y el valor devuelto es el offset. La siguiente tabla muestra el offset de cada báscula.

Tabla 3. Offset de las tres básculas

Báscula	Offset
1	8.471.350
2	8.514.030
3	8.590.390

Una vez se tiene el offset, se procede a calcular la escala, que es el valor por el cual hay que dividir el valor devuelto para que el peso medido sea el correcto. Para ello se necesita un objeto cuyo peso sea conocido. Para este proyecto se ha utilizado un peso de 0.150 kilogramos. Por lo tanto la escala, para kilogramos, es el valor leído, restándole el offset y posteriormente dividiendo por el peso conocido. Por ejemplo para la báscula 2 el valor leído con el peso encima es 8531196. Por lo tanto la escala es:

$$\frac{8531196 - offset}{0.150} = 114440$$

Ecuación 1. Cálculo de la escala

La siguiente tabla muestra la escala de cada báscula.

Tabla 4. Escalas de las tres básculas

Báscula	Escala
1	119530
2	114440
3	508600

Tal y como se ha explicado anteriormente, en el sketch principal del sensor se permite también redefinir el offset ya que los valores mostrados en la tabla 3 se corresponden al offset de la báscula sin ningún objeto encima. Sin embargo para la aplicación del sistema el offset deseado es el peso de los contenedores vacíos para poder medir de esta forma únicamente el peso del material que contienen éstos.

Para ello se ha añadido el botón de calibración a la izquierda del Arduino que, una vez pulsado, fija el offset de forma que el valor que está leyendo en ese instante sea considerado como cero kilogramos. Antes de presionar este botón el personal de la cocina debe despertar primero el Arduino manualmente con el botón conectado al pin 3.

Este nuevo valor de offset es importante tenerlo disponible siempre, aunque el Arduino se resetee o se apague, por ejemplo para cambiarle las pilas. No es deseable que cada vez que se apague el sistema, se deba volver a realizar la calibración del offset con los contenedores vacíos. Para ello se ha guardado la variable del nuevo offset en la memoria EEPROM (electrically erasable programmable read-only memory) del Arduino. El ATmega328P del Arduino Mini Pro tiene una EEPROM de 1kB. Arduino incluye una librería para leer, escribir y realizar otras acciones sobre la EEPROM [19]. De esta forma en el sketch principal, en el setup() se lee el valor offset de cada celda de carga guardado en la EEPROM antes de establecer el offset. La escala se define al principio del sketch y no es necesario guardarlo en la EEPROM ya que no es susceptible de ser cambiado.

Cada variable de nuevo offset es un long de 4 bytes (todas las escalas tienen un valor de offset del orden de pocos millones) por lo que ocupa 4 bytes de los 1024 disponibles. Las direcciones utilizadas en este sistema son 0-3 para el offset de la primera báscula, 4-7 para el de la segunda y 8-11 para el de la tercera. A la hora de escribir o leer de la EEPROM hay que tener cuidado de hacer los desplazamientos correctos. El byte almacenado en la dirección más baja corresponde al MSB (Most Significative Bit).

La conexión de los elementos mencionados con el Arduino Mini Pro se ha realizado tal y como se muestra en la siguiente tabla:

Tabla 5. Conexión de los elementos necesarios para medir el peso

Dispositivo/Componente	Pin de Arduino	Tipo de pin desde el punto de vista del Arduino	Colores de cable utilizado
HX711 DOUT1	9	INPUT	Verde
HX711 SCK1	8	OUTPUT	Amarillo
HX711 DOUT2	7	INPUT	Naranja
HX711 SCK2	6	OUTPUT	Rojo
HX711 DOUT3	5	INPUT	Gris
HX711 SCK3	4	OUTPUT	Blanco
Botón para despertar el Arduino manualmente	3	INPUT	Blanco
Botón para cambiar el offset de la báscula seleccionada	13	INPUT	Blanco

Se han utilizado los 6 pines de la fila superior porque los HX711 no están conectados a la placa de pruebas sino que se conectan al Arduino mediante cables y los pines de la fila superior son más accesibles.

El botón para cambiar el offset de las básculas se ha conectado al pin 13 porque va conectado al LED integrado que tienen los Arduinos. Como no se utiliza este LED, lo más recomendable es utilizar este pin como un pin de entrada.

Finalmente se muestran unas imágenes del montaje del sensor:

Figura 27. Fotografía del montaje de las básculas y el HX711

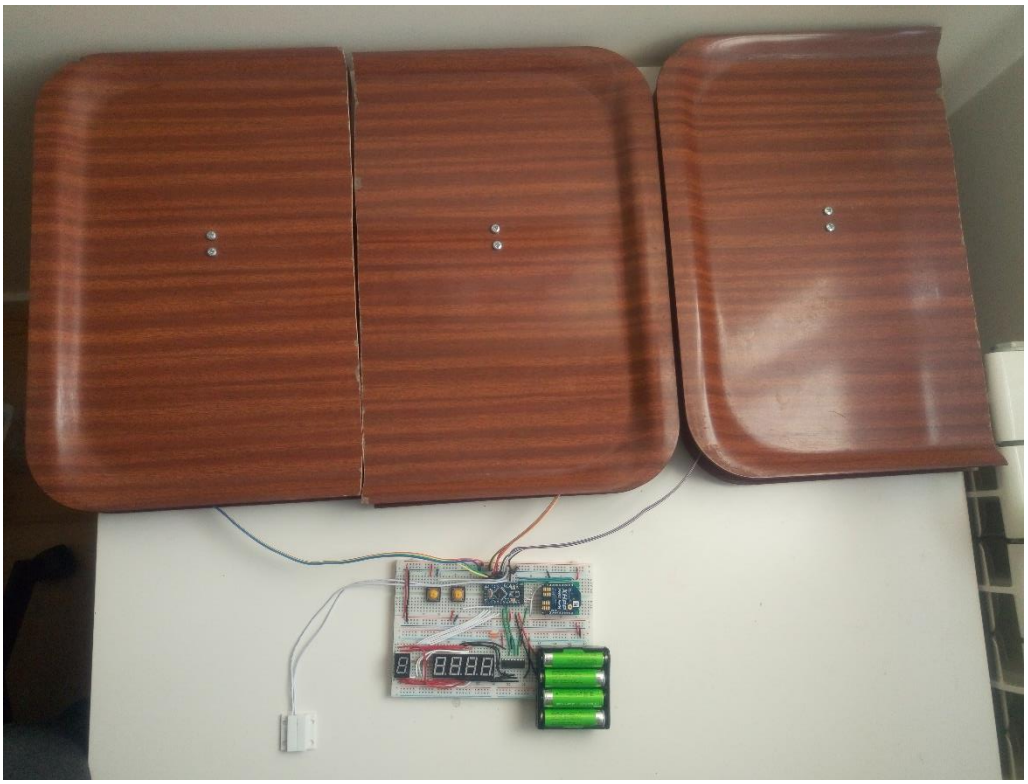

Figura 28. Fotografía del montaje del sensor completo

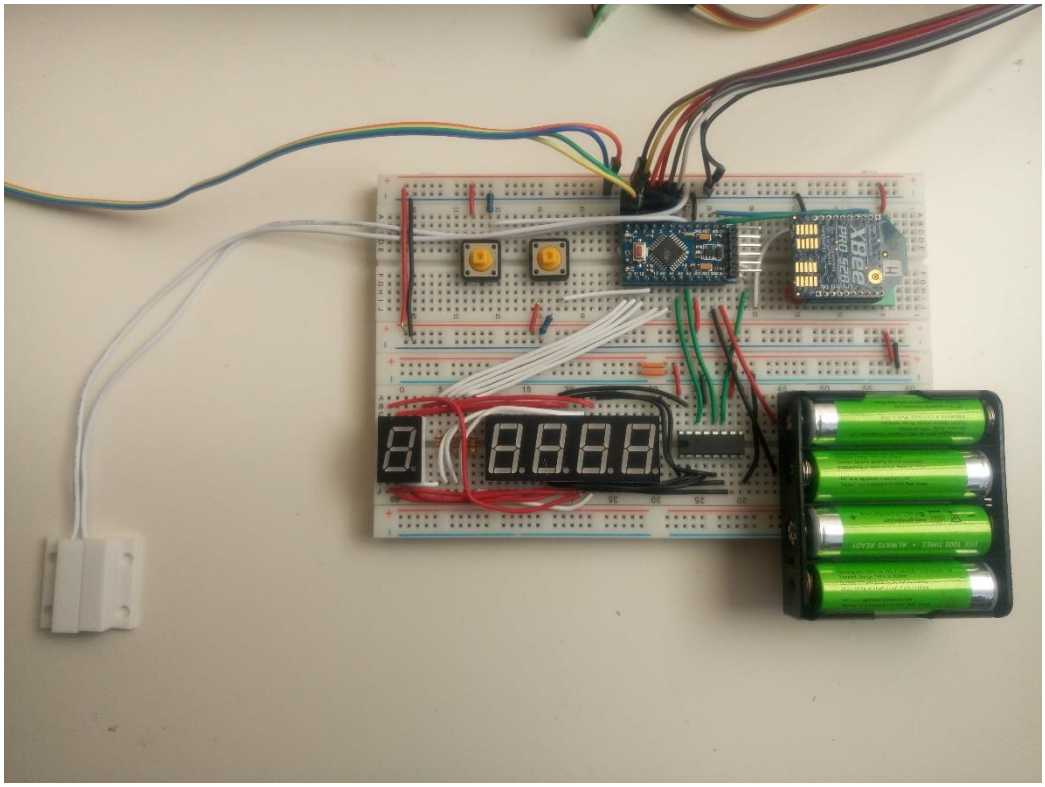

Figura 29. Fotografía de las conexiones de los componentes al Arduino

3.3 Visualizador de 7 segmentos

Tal y como se ha comentado anteriormente, para ver el peso de la báscula seleccionada, se despierta el Arduino con el botón del pin 3 y entra en un bucle hasta que se cierre la puerta. Dentro del bucle se selecciona la báscula para la cual se quiere ver el peso. La selección de la báscula se hace presionando este mismo botón. Cada vez que se pulsa se avanza de forma cíclica a la siguiente báscula. Para ello se ha implementado una función para evitar rebotes en el pulsador y que cada vez que se presiona se detecte como una única pulsación. Esto se hace añadiendo un retardo para realizar una segunda lectura del estado del conector cuando se detecta un primer cambio de estado. El retardo utilizado es de 10ms.

La báscula seleccionada se muestra en el display de 7 segmentos de un único dígito y su peso, en el de 4. Ambos displays utilizados son de cátodo común. En el apartado de análisis se había explicado la necesidad de utilizar un registro de desplazamientos 74HC595 para ahorrar el número de pines necesarios para controlar los ánodos que se encienden, pasando de requerir 8 pines a 3.

En los displays de 7 segmentos de varios dígitos se utiliza la multiplexación para mostrar un número. No todos los números se encienden a la vez sino que se van alternando lo suficientemente rápido para que el ojo humano interpole y lo vea como si todos los dígitos estuvieran encendidos. Según las pruebas realizadas una frecuencia de refresco de 100 Hz es adecuada ya que evita el parpadeo y además hace que los leds se vean de forma nítida. Esto significa que cada LED permanece encendido 2ms. El encendido y apagado de un dígito se controla con el cátodo de los LEDs. Los 5 cátodos de los 5 dígitos a controlar van conectados directamente al Arduino aunque si no se dispusiera de suficientes pines libres, se podría utilizar un segundo registro de desplazamientos.

Una vez se tiene el peso de la báscula que se quiere mostrar, se separan todas las cifras y se guardan en un vector de 5 elementos, siendo el primero el número de báscula y los otros 4, las cifras del número a mostrar. La función se inicia con todos los dígitos apagados (todos los cátodos en alto).

Tal y como se muestra en el datasheet del 74HC595 y en el siguiente diagrama de tiempos, antes de proporcionarle la entrada en serie del byte de datos, el pin ST_CP, debe estar en nivel bajo. Posteriormente se utiliza la función shiftOut() de Arduino que escribe el byte de datos (los ánodos de los LEDs que se quieren iluminar) en serie de bit en bit mientras proporciona un flanco de subida en SH_CP cuando el bit está disponible. Cuando todos los registros hayan guardado el bit correspondiente se pone ST_CP en alto. Este flanco de subida es el que proporciona la salida en paralelo del byte. Cuando ya se dispone de todos los segmentos a iluminar, se enciende el dígito poniendo el cátodo correspondiente en bajo, se deja 2ms encendido y después se apaga para volver a iniciar el proceso con el siguiente dígito.

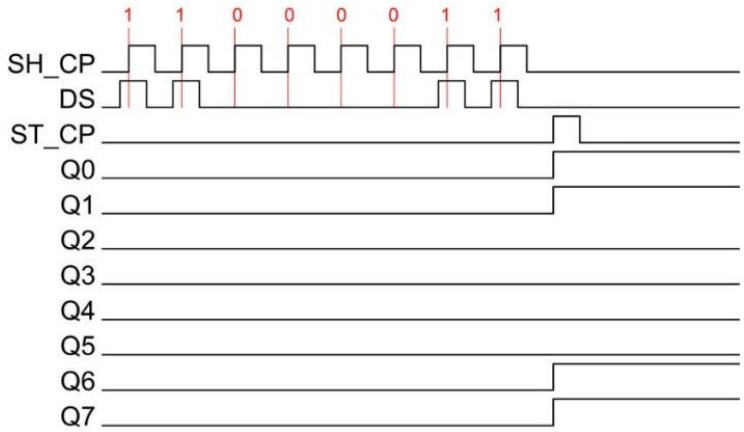

Figura 30. Diagrama de tiempos del 74HC595

La conexión de los elementos necesarios para el visualizador de 7 segmentos se muestra en la siguiente tabla:

Tabla 6. Conexión de los visualizadores de 7 segmentos al Arduino

Elemento	Pin de Arduino	Tipo de pin desde el punto de vista del Arduino	Colores de cable utilizados
DS (Serial Data Input)	16	OUTPUT	Verde
SH_CP (Shift Register Clock Input)	17	OUTPUT	Verde
ST_CP (Storage Register Clock Input)	19	OUTPUT	Verde
Cátodo del display de un dígito	10	OUTPUT	Blanco
Cátodos del display de cuatro dígitos	11, 12, 14, 15	OUTPUT	Blanco

Aparte de estas conexiones también se tienen que conectar los ánodos de los segmentos al 74HC595. Los segmentos A-G van conectados a los pines Q0-Q6 del registro de desplazamientos. El segmento DP (decimal point) va conectado al pin Q7. Los colores utilizados son el negro para conectarlo entre el 74HC595 y el display de 4 dígitos y rojo para conectarlo al de un dígito.

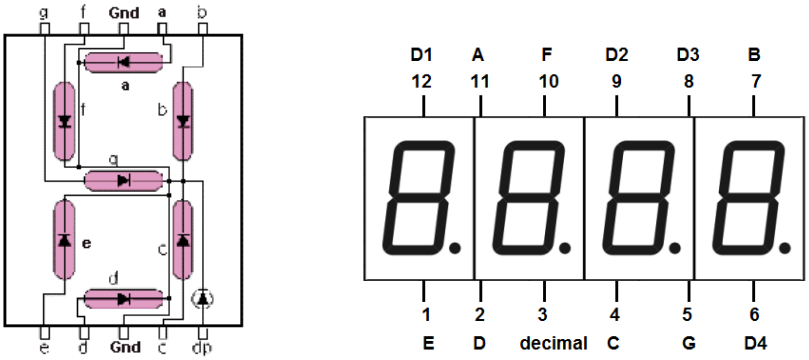

Figura 31. Diagrama de pines de los dos visualizadores de siete segmentos

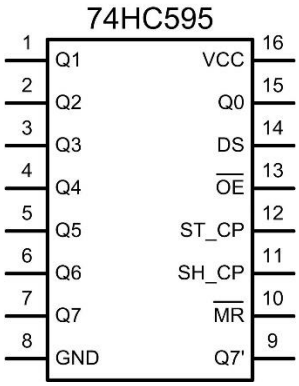

Figura 32. Diagrama de pines del registro de desplazamiento 74HC595

4 Desarrollo e implementación del módulo de comunicaciones

Para que el sensor y el servidor se puedan comunicar de forma inalámbrica se han utilizado los módulos XBee de Digi.

Los XBee se clasifican en serie1 y serie2. La serie1 es simplemente 804.15.4 que define la capa física y la capa MAC que utiliza Zigbee. Por lo tanto sólo permite comunicaciones punto a punto y punto a multipunto. No existen Routers; sólo Coordinators y End Devices. La serie2 es la que sería realmente Zigbee, en concreto Zigbee PRO, e implementa toda la pila de protocolos definidas por la Zigbee Alliance. Los XBee serie1 y XBee serie2 no son interoperables.

Dentro de la serie2 caben dos categorías: la XBee regular y la XBee PRO. Ambos son muy parecidos pero la versión PRO ofrece un mayor alcance a cambio de un mayor consumo energético en transmisión y además es más caro.

Para el desarrollo de este proyecto se han adquirido tres XBee PRO para trabajar con cada uno de los roles definidos por Zigbee: un End Device (a partir de ahora ED) que irá conectado con el Arduino formando parte del sensor; un Coordinator, que irá conectado al servidor y será el que establece los parámetros de la comunicación y un Router, que será el padre del End Device y le proporciona un mayor alcance.

Para configurar un XBee se utiliza XCTU, software proporcionado por Digi. Las herramientas fundamentales que incluye son un buscador de módulos XBee conectados al equipo, una sección para configurar tanto el rol como los parámetros del dispositivo y algunas herramientas para testear la comunicación como un monitor serie o un generador de tramas.

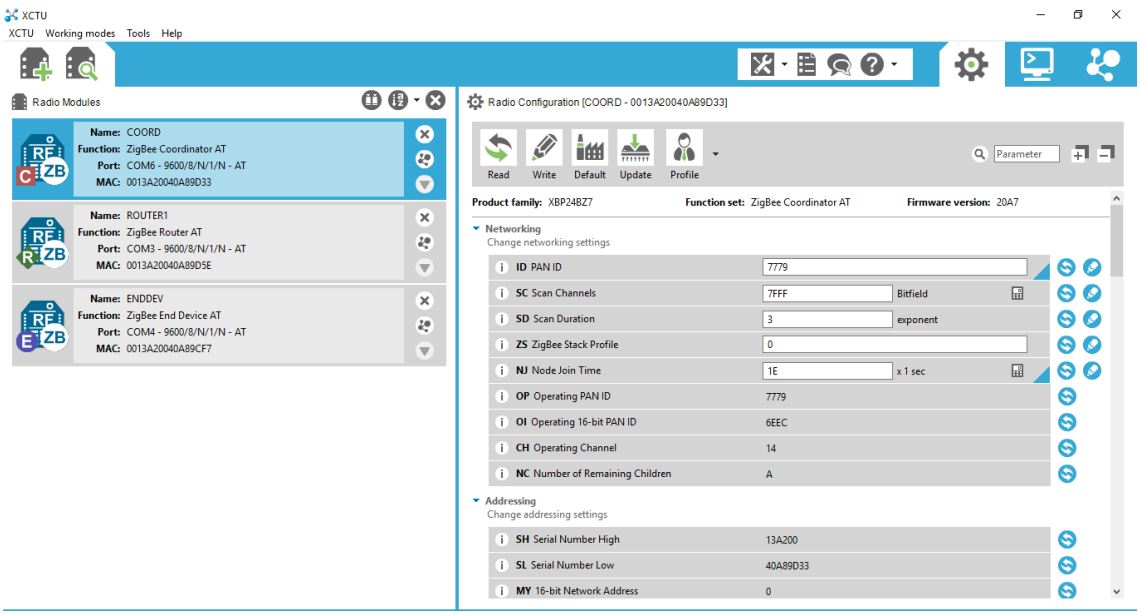

Figura 33. Captura de pantalla del configurador de parámetros de XCTU mostrando los tres módulos conectados al equipo en la sección izquierda

4.1 Comunicación entre XBee y el host

En el segundo capítulo del manual de usuario de XBee [20] (XBee ZB RF Module operation) viene detallado cómo es la comunicación entre el módulo XBee y cualquier otro dispositivo con una interfaz UART (Universal Asynchronus Receiver / Transmitter), también llamado puerto serie.

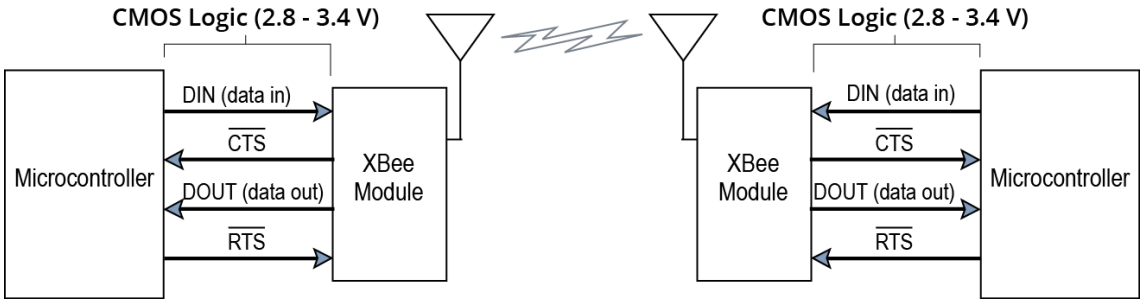

Figura 34. Diagrama de flujo de datos entre XBee y el equipo al que se conecta

Tal y como se muestra en la figura anterior para que la comunicación inalámbrica funcione el pin TX del Arduino tiene que estar conectado al pin DIN del XBee y el RX del Arduino al DOUT del XBee. Este conexionado se denomina modo XBee ya que el módulo transmite por la interfaz RF los datos recibidos desde el pin TX del microcontrolador. Sin embargo si el Arduino está conectado también al puerto USB del ordenador con el programador FTDI, por ejemplo para programar el Arduino o configurar el XBee, la comunicación entre el puerto USB y el Arduino no funcionará porque el pin TX y RX del Arduino ya está realizando una comunicación con el XBee. Por lo tanto la programación/configuración fallará.

Para esto es necesario emplear el modo USB cambiando el conexionado entre el Arduino y el XBee. Por lo tanto el pin DIN del XBee va conectado al pin RX del Arduino, que a su vez va conectado al pin TX del FTDI. De esta forma se permite programar el Arduino y también configurar el XBee si se mantiene apretado el reset del Arduino para que no interfiera en la comunicación entre XBee y FTDI. Sin embargo en este modo el XBee no transmite nada por su interfaz RF ya que el pin TX del Arduino va al pin DOUT del XBee en vez de al DIN.

Con el Arduino Breakout Board, que es el que se emplea con el ED, el cambio de conexionado se hace directamente moviendo los cables. El Router utiliza el Arduino XBee Shield que lleva incorporado un jumper para ponerlo en modo XBee o modo USB. Para que el Router funcione, en realidad da igual el modo en que se pone ya que en ningún momento realiza ninguna comunicación con el Arduino. Éste solo sirve de soporte para mantener la alimentación del XBee. Lo que sí se tiene que cumplir es que mientras se configura el XBee, el jumper debe estar en modo USB y se debe mantener presionado el reset del Arduino mientras se lee o se escribe sobre el XBee. El Arduino del Router ejecuta un código vacío. El forwarding de los paquetes lo realiza el procesador del XBee.

Figura 35. El jumper para cambiar entre modo XBee y modo USB se puede observar en la esquina inferior izquierda

El manual de XBee también define dos tipos de protocolos para la interfaz serie: el modo transparente (también llamado AT) y el modo API.

En el modo AT, la interfaz RF de XBee es como una continuación de su interfaz serie. Todos los datos recibidos por el pin DIN se encolan para enviarlo directamente por la interfaz RF y lo mismo ocurre con la recepción de datos. No se realiza ningún entramado y es el modo más sencillo de utilizar. Pero tiene la desventaja de que los datos sólo se puede transmitir a un dispositivo cuya dirección está configurada en el XBee (la dirección de destino).

Por otro lado en el modo API se realiza un entramado de los paquetes según a qué destino se quiere enviar y se aprovechan las capacidades de encaminamiento que tiene el protocolo.

Para este proyecto se ha utilizado el modo transparente porque sólo se utiliza una ruta, en la que el sensor (End Device) envía datos al servidor (Coordinator).

4.2 Configuración de parámetros de un XBee

Antes de configurar los distintos parámetros de un XBee es necesario definir el rol que va a tener (Coordinator, Router o End Device) y el modo a utilizar (AT o API). Posteriormente se cambian los parámetros necesarios según la topología de la red. Hay dos formas de configurar los parámetros: utilizando directamente el configurador del XCTU o utilizando el modo comando que consiste en escribir comandos AT en la consola de XCTU. Se ha utilizado mayoritariamente el configurador porque es más sencillo ya que ofrece una interfaz gráfica.

El parámetro de red más importante es el PAN ID que es el identificador de la red ZigBee. Este parámetro se define en el Coordinator, que es el dispositivo que establece la red, y el resto de los dispositivos deben configurarse con este ID para unirse a ésta. En toda red ZigBee debe existir al menos un Coordinator. El PAN ID utilizado es 7779.

En cuanto al direccionamiento, los parámetros más importantes son:

- SH (32 bits - Serial Number High). Parámetro solo de lectura asignado durante la fabricación del módulo.
- SL (32 bits - Serial Number Low). Parámetro solo de lectura asignado durante la fabricación del módulo.
- MY (16 bits - Network Address). Parámetro solo de lectura generado aleatoriamente cuando un módulo se conecta a la PAN. Esta es la dirección que se utiliza cuando se envía datos. Para descubrir la dirección MY del resto de dispositivos de la red, ZigBee utiliza un protocolo de descubrimiento de direcciones que crea una tabla que mapea la dirección de 64 bits y la de 16 bits.
- DH (32 bits – Destination Address High)
- DL (32 bits – Destination Address Low)

DH y DL es la dirección SH, SL del dispositivo al que se quiere enviar los datos. 0x000000000000FFFF es la dirección de Broadcast de la PAN. 0x0000000000000000 es la dirección a utilizar para enviar al Coordinator de la PAN.

Las direcciones de los dispositivos se han configurado de la siguiente manera:

Tabla 7. Tabla de direccionamiento de los módulos XBee

	Coordinator	Router	End Device
SH	0x0013A200	0x0013A200	0x0013A200
SL	0x40A89D33	0x40A89D5E	0x40A89CF7
DH	0x0013A200	0x00000000	0x0013A200
DL	0x40A89CF7	0x00000000	0x40A89D33

Como se puede observar, el Coordinator utiliza como dirección de destino la del ED y viceversa. El ED también habría podido utilizar la dirección 0x0000000000000000. Si se hubiera utilizado la dirección de Broadcast, la comunicación también habría funcionado correctamente pero esta solución no es escalable y daría problemas si la red tuviera más dispositivos, por lo que se recomienda evitar el uso del Broadcast a no ser que sea estrictamente necesario. El Router utiliza la dirección de Unicast del coordinador de la PAN. Hay que remarcar que aunque la dirección de destino del Router se configure con la dirección del End Device, la comunicación entre End Device y Coordinator funcionaría igualmente. La dirección de destino solo indica la dirección a la que se envían los paquetes generados desde ese dispositivo.

4.3 Configuración de parámetros para dormir el End Device

Un ED puede configurarse para entrar en modo dormido para ahorrar consumo cuando no tiene que transmitir o recibir ningún dato.

Cuando un ED se conecta a la red, siempre lo hace a través de un Router o el Coordinator directamente y éste se convierte en el padre del ED. Toda comunicación de la red entrante o saliente del ED relega su padre. Éste es el responsable de hacerle llegar los datos.

En este sistema, para probar el correcto funcionamiento del Router, se ha forzado a éste a ser el padre del ED. Para ello en el Coordinator se ha configurado el parámetro NJ (Node Join Time) a 0x1E, que son 30 segundos. Esto significa que tras 30 segundos, el Coordinator no va a permitir a más dispositivos conectarse a él como hijo. Si se enciende el ED 30 segundos después de conectar el Coordinator, el ED estará forzado a elegir al Router como padre. En un sistema final esto no debería implementarse así, ya que la red es la que se encarga de elegir el dispositivo más adecuado para que sea el padre. Acortar el NJ del Coordinator es sólo para probar el correcto funcionamiento del sistema con un Router intermedio.

Cuando el ED no está en modo dormido, realiza un polling continuo (por defecto cada 100ms) al padre para ver si necesita recibir datos. Si no, el ED entra en idle mode (distinto al modo dormido) donde apaga el receptor de radiofrecuencia y ahorra batería. Si se prevé que el ED no va a recibir o enviar datos en los próximos segundos, éste se puede poner en modo dormido. Hay dos modos dormido distintos: dormido por pin o dormido cíclico.

En el modo dormido por pin el XBee se duerme o se despierta con una entrada externa conectada a su pin 9 (SleepRQ). El XBee está despierto cuando SleepRQ está en estado bajo y permanece dormido cuando permanece alto.

El modo dormido por pin es el utilizado en el sistema. Este pin 9 del XBee es el que va conectado al Arduino, concretamente a su pin 18 que desde su punto de vista es un OUTPUT. Cuando el Arduino tiene los datos listos para transmitir, pone su pin 18 en bajo, envía el dato a través de una comunicación serie (XBee en modo transparente), lo mantiene despierto durante 500 ms y luego lo duerme poniendo SleepRQ en nivel alto. Durante los 500 ms que permanece despierto el XBee hace pollings al padre cada 100 ms.

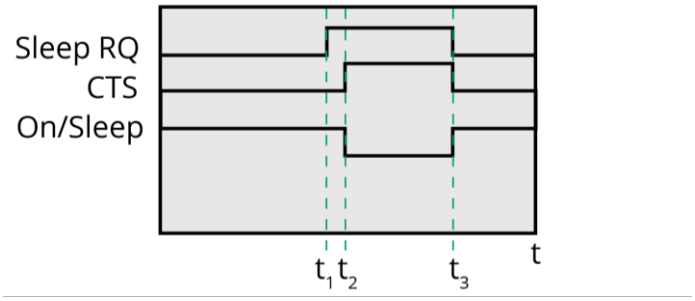

Figura 36. Diagrama de tiempos del modo dormido por pin

Cuando el XBee se duerme, su pin 13 (On/Sleep) está en estado bajo y cuando está despierto este pin está en nivel alto. Es un indicador del estado en el que está el XBee. Este pin fue utilizado como entrada al Arduino durante el desarrollo pero una vez finalizado, se ha quitado ya que no se necesitaba y así dejaba un pin libre para otros usos.

En el diagrama de tiempos, t1 es el instante en el que se indica al ED que se tiene que dormir y t2 es el instante en que realmente entra en modo dormido. El tiempo entre t1 y t2 depende del estado del ED pero en el peor escenario podría ser de unos pocos segundos. Lo importante es que en t3, tan pronto como se indique que se tiene que despertar, el ED se despierta sin ningún retardo. Cuando el ED está dormido el pin CTS (Clear to Send), que

implementa el control de flujo en los buffers internos, indica que está dormido y que no le deben llegar más datos.

El modo cíclico se utiliza en aplicaciones en las que los intervalos donde el ED permanece despierto y dormido son conocidos y constantes. Existe un modo cíclico corto en la que se despierta como máximo cada 28 segundos, y un modo extendido donde puede permanecer dormido varios días.

La configuración del modo dormido por pin se realiza de la siguiente manera:

Tabla 8. Parámetros de configuración para el modo dormido por pin

	Padre	End Device
SM (Sleep Mode)	No sleep (Router) [0]	Pin Hibernate [1]
SP (Cyclic Sleep Period)	0x7D0 (x10ms)	No utilizado
SN (Number of Cyclic Sleep Periods)	0x10E0	No utilizado

En el ED solo es necesario indicarle que se va a dormir por entrada externa. El resto de los parámetros son para configurar el dormido cíclico. En el padre se ha fijado SP y SN para crear el Child Poll Timeout. Este timeout se utiliza para borrar al ED de la tabla de hijos si éste no hace un polling dentro de este timeout. Según el manual, SP y SN se debe fijar de forma que SP*SN sea el tiempo más largo para el cual se espera que el ED pueda permanecer dormido. El timeout vale 3*SP*SN. Por lo tanto con los valores fijados son 20000ms para SP y 4320 para SN. El tiempo esperado en el que permanece despierto se ha fijado en 4320*20000ms = 24 horas y por lo tanto el padre borrará al ED de la lista de hijos tras tres días.

No debería pasar más de tres días sin abrirse la puerta de la cámara frigorífica pero si esto ocurriera, los datos que mandaran el ED y sus pollings serán rechazados por el padre. En estos casos lo que se debería hacer es resetear el sensor ya que en el setup() del sketch de Arduino se despierta el ED durante 3 segundos para volver a realizar la conexión con el padre.

5 Desarrollo e implementación del módulo del servidor

En el apartado del análisis se comentaron los objetivos específicos del servidor, que se enumeran a continuación:

- Guardar los datos enviados por el sensor en una base de datos.
- Procesar estos datos para realizar los pedidos según el ciclo que corresponda y con la cantidad correcta para poder satisfacer la demanda prevista y evitar que se generen despilfarros.
- Ofrecer una interfaz a través de la cual el encargado puede controlar y supervisar la cantidad de material que hay en la cámara.

En los siguientes subapartados se explicarán los elementos necesarios y el desarrollo realizado para cumplir con cada uno de estos objetivos.

5.1 Diseño de la base de datos

La base de datos es uno de los elementos fundamentales del sistema ya que almacenará todos los datos que se generen desde el sensor. El sistema de gestión de base de datos utilizado es MySQL, que es el que incluye XAMPP. Para administrarla y configurarla se ha utilizado phpMyAdmin. Esta herramienta utiliza una interfaz gráfica para gestionar la base de datos, aparte de permitir configurarlo a través de sentencias SQL.

Primero se ha creado una base de datos llamada restdb que contendrá todas las tablas utilizadas en el sistema. Las tablas utilizadas y las relaciones entre ellas se muestran en el siguiente diagrama:

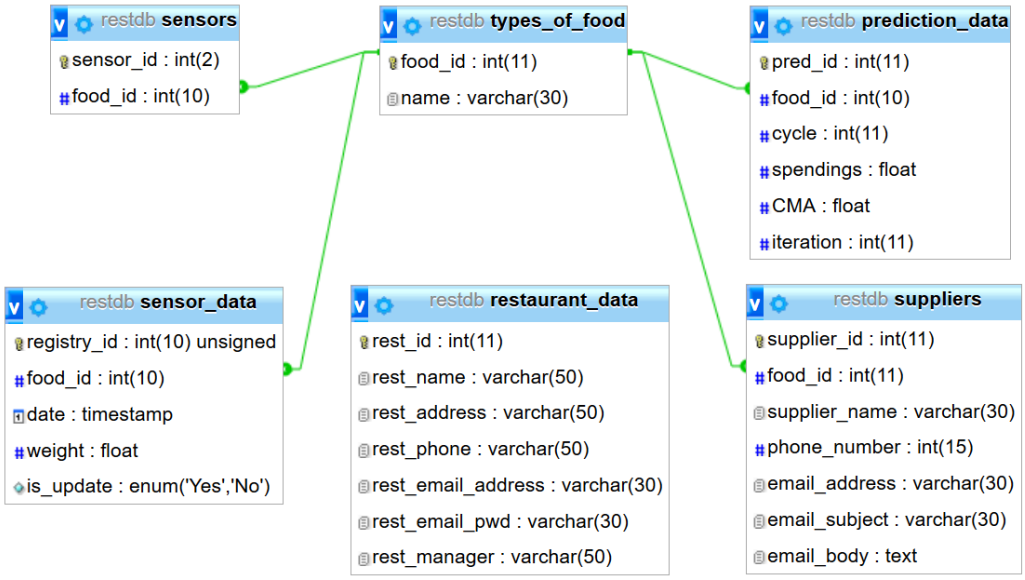

Figura 37. Diagrama representativo de las tablas de la base de datos y sus relaciones

Todas las tablas tienen una clave primaria, simbolizado con una llave amarilla a la izquierda, que identifica de forma única cada una de las filas que contiene una tabla. Este identificador no puede ser nulo y se autoincrementa con cada nuevo registro.

La tabla types_of_food representa los tipos de comida con los que se va a trabajar y le proporciona un identificador a cada uno de ellos. Este identificador (food_id) va a ser utilizado por el resto de las tablas, excepto restaurant_data, como clave foránea. Esto significa que las entradas en estas tablas deben tener un valor de food_id que exista en la tabla types_of_food.

La tabla sensors representa las tres básculas utilizadas en el sensor y el tipo de comida que contiene. Las tres identificadores van de 1 – 3 pero estos números son independientes del número de tipo de comida. Por ejemplo la ternera que tiene un identificador food_id = 1 en la tabla types_of_food podría ir en el sensor número 3.

La tabla sensor_data contiene los datos generados por el sensor. Estos datos representan la cantidad que queda de cada tipo de comida. Para cada dato, se define el tipo de comida que es, la cantidad que queda, la fecha en la que se produce este dato y si se corresponde o no con una recarga de material que han traído los proveedores. Es importante destacar que para cada registro lo que se desea conocer es el tipo de comida que es y no se tiene en cuenta el sensor que ha generado este dato. Esto es debido a que una báscula no conoce la comida que tiene encima, sólo sabe el peso. Por lo tanto antes de introducir los datos en esta tabla, se debe chequear qué tipo de comida hay en cada báscula. Esto se hace en el script de inserción de datos que se explicará posteriormente. Esto se ha diseñado de esta forma porque uno de los requisitos del restaurante es que la comida dentro de la nevera pueda ser ordenada si fuera necesario. Así, si en una báscula había por ejemplo contenedores de pollo, después de ordenarlo se podría colocar ternera. La comida que hay en cada báscula se puede configurar desde la interfaz web.

La tabla prediction_data contiene los datos necesarios para realizar los cálculos de cuánta cantidad de comida hay que pedir a los proveedores en los dos días semanales que se han definido para realizar los encargos.

La tabla suppliers contiene los datos de los distintos proveedores del material utilizado en el restaurante.

Finalmente la tabla restaurant_data contiene los datos del restaurante. Contiene una única fila que representa al restaurante en cuestión. Si el sistema se extendiera a más de un restaurante gestionado por el mismo gerente es más conveniente crear otra base de datos distinta a restdb.

5.2 Inserción de datos en la base de datos

El módulo del servidor está separado en dos elementos lógicos. Por un lado está el servidor XAMPP que utiliza tecnologías web para almacenar y mostrar los datos y por otro, está el receptor XBee (Coordinator) conectado a un puerto USB en el ordenador TPV del restaurante y que recibe los datos enviados desde el sensor, pero que no utiliza tecnologías web sino ZigBee. Por lo tanto es necesario una interfaz que conecte ambos elementos. Para esto se ha desarrollado un script con Python (addData.py) que lee los datos entrantes por el puerto USB y

los almacena en la base de datos MySQL que integra XAMPP. Python es un lenguaje de programación interpretado (no necesita compilar el código fuente para su ejecución) y de propósito general. El IDE utilizado es PyCharm ya que proporciona todas las herramientas necesarias para el desarrollo y la prueba del script.

Este script se ejecuta de forma permanente en el equipo al que el Coordinator XBee va conectado. Para leer estos datos entrantes primero es necesario abrir el puerto serie, haciendo uso de la librería Serial que facilita Python, e indicar el puerto por el que se conecta el Coordinator.

Posteriormente se abre una conexión a la base de datos MySQL, indicando la dirección IP, la base de datos concreta a la que se conecta y el usuario y su contraseña.

Lo primero que hace el script es procesar los datos entrantes por el puerto serie. Para que la comunicación funcione correctamente el Arduino y el script de Python deben utilizar la misma trama de datos. Esta trama tiene el siguiente formato:

[Número de básculas utilizadas, identificador de la primera báscula, peso sobre la primera báscula, identificador de la segunda báscula, peso sobre la segunda báscula, etc]

Cada elemento de la trama está separado por espacios. Tras separar los datos de esta trama y antes de añadirlos a la base de datos, se hace previamente un procesamiento para determinar cómo se ha de hacer la inserción. Las básculas del sensor no pueden determinar el tipo de comida que tienen encima, sólo saben el peso. Por lo tanto primero se realiza un chequeo para saber qué comida hay en cada báscula. Esto se hace leyendo los datos de la tabla sensors, que indica el tipo de comida que hay en cada sensor. La sentencia SQL para leer una tabla es SELECT.

Además antes de introducirlos a la base de datos, se comprueba qué pesos han cambiado desde la última vez que se ha recibido una trama. De esta forma se crea una nueva fila en la tabla sensor_data solo con la comida cuyo peso ha cambiado. Esto evita que la tabla se llene de datos que no aportan ningún valor.

Para desarrollar esto se han hecho uso de los diccionarios de Python [21]. Un diccionario es una lista de pares clave valor. Se han utilizado cuatro diccionarios:

- Sensors: la clave indica el identificador de la báscula y el valor es el identificador de tipo de comida.
- Serial_input: la clave es el identificador de la báscula y el valor es el peso actual de la comida.
- Previous_data: la clave indica el identificador de comida y el valor es el peso de la trama anterior.
- Current_data: la clave es el identificador de comida y el valor es el peso actual de la comida.

Current_data son los datos que se desean almacenar en la base de datos y se crea a partir de sensors, que se obtiene haciendo un SELECT a la base de datos, y serial_input que se obtiene de la entrada serie. Una vez se crea current_data, se compara con previous_data y los datos que sean iguales no se insertan. Si se observa que la cantidad actual es significativamente mayor que

la anterior, quiere decir que se ha encargado comida nueva y se marca en la columna is_update de la tabla sensor_data. Finalmente se insertan estos valores a la tabla con la sentencia SQL INSERT. Una vez insertado, vuelve al inicio del bucle esperando la entrada de una nueva trama.

5.3 Procesamiento de datos para realizar pedidos

La realización de encargos nuevos también se ha hecho con un script de Python. También se podría haber utilizado PHP pero esto implicaba tener la página web de la interfaz gráfica abierta para que PHP se pueda ejecutar.

Este script (stockOrder.py) también se ejecuta permanentemente en el equipo y el bucle ejecuta un ciclo en los días de la semana y en la hora prefijada, que según se ha explicado anteriormente, serían los lunes y los viernes a la 1:00. Al hacer los pedidos en estos tiempos, la comida llegaría el mismo día sobre las 12:00.

Los datos necesarios para calcular la cantidad a pedir están guardados en la tabla prediction_data. Las columnas que se han definido son las siguientes:

- pred_id: el identificador para cada uno de los registros creados en esta tabla.
- cycle: el ciclo al que se refiere la fila. 1 para el primer ciclo (lunes a jueves) y 2 para el segundo (viernes a domingo).
- food_id: el tipo de comida de esta fila.
- spendings: la cantidad gastada de este tipo de comida en desde la última recarga.
- CMA: cumulative moving average. Es el promedio del gasto de todos los ciclos del mismo tipo, incluyendo este último gasto.
- iteration: iteración para calcular CMA.

Cuando se ejecuta el bucle, se crea una fila indicando lo que se ha gastado este último ciclo. Primero lee de la tabla sensor_data el último registro que fue un update. Esto indica la cantidad de comida que había al empezar el ciclo. Posteriormente se lee el último registro de esta misma tabla, que indica la cantidad que hay ahora mismo y se resta a la cantidad que había, obteniendo lo que se ha gastado en este ciclo.

Después se calcula el promedio de los gastos de este tipo de ciclo (primero o segundo, según toque), incluyendo este nuevo gasto. Este promedio será la cantidad que se quiere tener disponible (más un pequeño margen de seguridad), la próxima vez que se inicie este ciclo.

El promedio acumulativo se calcula de la siguiente forma:

$$CMAi = \frac{(i-1) * CMA_{i-1} + x_i}{i}$$

Ecuación 2. Cálculo de la media móvil acumulada

Para calcular este nuevo promedio se necesita el promedio anterior y su iteración.

Una vez realizado el resumen de los gastos de este ciclo finalizado, se calcula la cantidad a gastar para este nuevo ciclo que empieza. La cantidad a pedir es el promedio de gastos del ciclo correspondiente, restándole la cantidad que aún está disponible y añadiéndole un margen de seguridad, por ejemplo de un kilogramo.

Una vez calculado, se pide esta cantidad enviando un correo al proveedor correspondiente. Para ello se ha utilizado la librería smtplib de Python. Se necesitan los datos de contacto del restaurante (teléfono, dirección, etc), su dirección de correo electrónico con la contraseña y los datos del proveedor (tipo de comida que provee, su dirección de correo y una plantilla del correo a enviar). Todos estos datos se obtienen haciendo un SELECT de las tablas restaurant_data y supplier.

5.4 Interfaz gráfica para el visualizar los datos

Para facilitar la gestión del material al encargado se ha realizado una interfaz web gráfica desde la que puede observar cómo se ha producido el gasto de la comida. Se ha decidido utilizar una interfaz gráfica en vez de una aplicación de escritorio debido a la integración que trae XAMPP y además de esta forma está disponible desde cualquier equipo que quiera utilizar el encargado ya que solo necesita un navegador web para acceder a esta información.

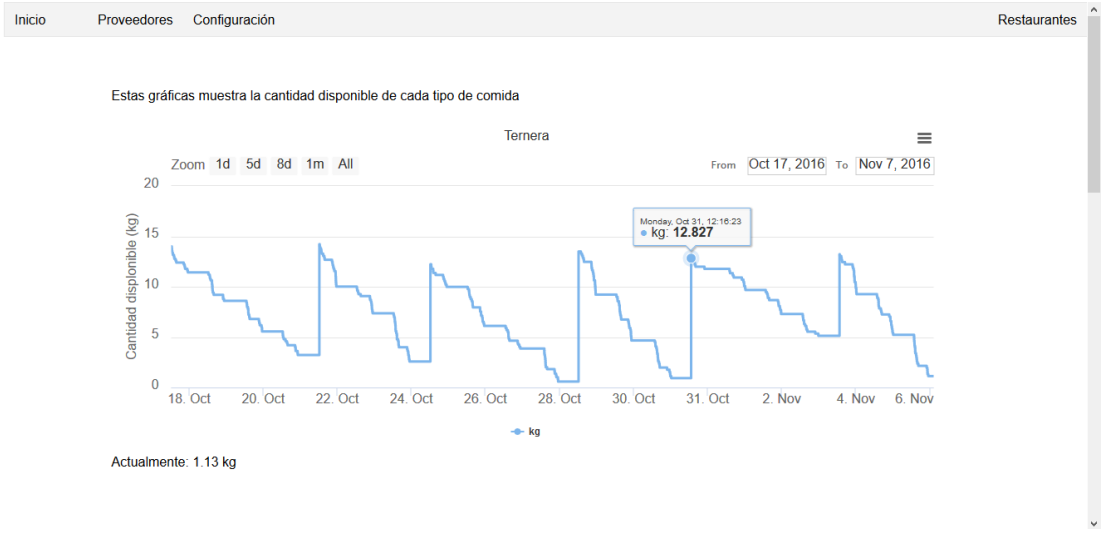

Figura 38. Captura de pantalla de la interfaz gráfica

Esta interfaz web se ha desarrollado con las tecnologías web clásicas: HTML para fijar el contenido, CSS para el diseño gráfico y PHP para el procesamiento de datos.

También hay que destacar el uso de Highcharts, que es una librería escrita en JavaScript, para dibujar las gráficas a partir de los datos de la tabla sensor_data. JavaScript es un lenguaje de programación interpretado, utilizado comúnmente para realizar procesado en el lado del cliente (navegador web),

La página principal index.php es la base de la página. Dentro de la etiqueta head se importa el fichero de estilos style.css y las librerías necesarias para Highcharts. En la cabecera

body, donde se escribe el contenido, se ha elaborado primero una barra de navegación horizontal con algunas de las opciones que podría tener una interfaz de un sistema de gestión. Debajo de ésta se ha colocado la sección que incluye la lista de gráficas. En la imagen de la interfaz gráfica sólo se puede observar una gráfica pero haciendo scroll hacia abajo aparecen las otras dos gráficas correspondientes a los tres tipos de comida utilizados para el sistema.

En style.css se define el diseño gráfico, como por ejemplo la barra tiene que ser horizontal, con color gris de fondo y cambiar el color de fondo cuando se pasa el ratón por encima de un botón. También define que la sección de gráficas esté centrada y con una anchura de 800 píxeles.

Finalmente, index.php incluye la ejecución de los scripts de Highcharts que dibuja una gráfica en un contenedor especificado. Antes de pintar las gráficas se ejecuta un bloque de código PHP que se conecta a la base de datos, realiza un SELECT según el tipo de comida que va a pintar en cada gráfico y almacena estos datos en dos arrays. Posteriormente se crea un objeto Highcharts al que se le proporcionan estos datos con un formato correcto, además de configurar ciertos parámetros de las gráficas, como tipos de ejes, el título, la posibilidad de realizar zoom sobre la gráfica o habilitar dos cuadros para introducir las fechas de las que se quiere ver las gráficas.

6 Pruebas y resultados

6.1 Pruebas durante el desarrollo

A lo largo del desarrollo, se han utilizado las diversas herramientas proporcionados por los IDE para probar el correcto funcionamiento, inicialmente de cada uno de los bloques por separado y finalmente del sistema completo.

Para las pruebas del sensor, se ha colocado el prototipo con las básculas dentro de la cámara frigorífica para probar su funcionamiento. Tras calibrarlo dentro de la cámara frigorífica, se ha probado que efectivamente el peso medido era correcto con los pesos conocidos. Previamente, durante la búsqueda de componentes electrónicos se ha comprobado que en todos ellos, el rango de temperatura de operación incluye el rango de 0-4º con el que funciona la cámara.

Para las pruebas del módulo de comunicaciones se han colocado en la cámara frigorífica el XBee End Device con un Arduino cargado con un sketch que envía cada segundo el tiempo transcurrido desde que se habían encendido los dispositivos. El Router y el Coordinator se han conectado a un ordenador portátil con XCTU para monitorizar la recepción del mensaje enviado por el End Device. Los resultados obtenidos son totalmente satisfactorios con el Router colocado dentro de la cocina pero fuera de ella, en el comedor, el Router no se podía alejar más de tres, cuatro metros de la pared que separa ambas salas. Por lo tanto, finalmente se ha concluido que la utilización de un Router es necesario si el equipo TPV al que iría conectado el Coordinator no está situado cerca de la cocina.

Finalmente para las pruebas del funcionamiento del servidor se ha utilizado la consola de PyCharm y phpMyAdmin para comprobar el funcionamiento de los scripts y verificar que efectivamente se han introducido los datos en la base de datos. Por último se ha comprobado la compatibilidad de la interfaz web desarrollada con los navegadores web Google Chrome, Mozilla Firefox y Microsoft Edge.

6.2 Prueba del script de pedidos

Para probar el script de pedidos es necesario primero tener una base de datos donde la tabla sensor_data contenga al menos los gastos que se han producido en una semana para poder calcular sobre él la cantidad que había al principio y al final.

Para generar una serie de datos relevantes y con sentido se ha elaborado otro script de Python (dataGeneration.py) que escribe en un fichero de texto una sentencia SQL insertando una simulación de gastos a lo largo de cada ciclo de pedido. Para el script se han tenido en cuenta los siguientes detalles para representar el consumo de la forma más realista posible:

- Las fechas utilizadas van del 17 de Octubre al 6 de Noviembre (tres semanas).
- Las horas en las que se producen gastos van de 13:00 – 16:00 y 20:30 – 23:45 y se distribuyen de forma uniforme.

- El peso utilizado para cada ración es una variable uniforme entre 185 y 215 gramos.
- En el turno de mañanas el número de veces que se consume cada tipo de comida es una variable uniforme entre 7 y 10. En el turno de noches es entre 5 y 7. De viernes por la noche a domingos por la tarde este número se multiplica por dos ya que en los fines de semana hay mayor demanda.

De esta forma en el primer ciclo (lunes a jueves) el número de raciones servidas para cada tipo de comida es aproximadamente 60, y para el segundo ciclo (viernes a domingo), alrededor de 75.

Después de generar los datos de consumo de un ciclo, se ha ejecuta una vez el bucle de stockOrder.py y se comprueba que tanto los gastos como el promedio generados para ese ciclo son correctos. Posteriormente se vuelve a ejecutar dataGeneration.py para introducir datos del siguiente ciclo y volver a comprobar que stockOrder.py funciona bien.

A lo largo de las tres semanas simuladas estos son los datos generados en la tabla prediction_data:

Tabla 9. Prediction_data generado tras tres semanas

Cycle	Food_id	Spendins (kg)	CMA (kg)	Iteration
1	1	10.778	10.778	1
1	2	11.901	11.901	1
1	3	12.708	12.708	1
2	1	11.646	11.646	1
2	2	11.743	11.743	1
2	3	12.068	12.068	1
1	1	11.642	11.21	2
1	2	12.424	12.1625	2
1	3	12.301	12.5045	2
2	1	12.574	12.11	2
2	2	11.238	11.4905	2
2	3	12.308	12.188	2
1	1	7.697	10.039	3
1	2	13.482	12.6023	3
1	3	12.021	12.3433	3
2	1	13.084	12.4347	3
2	2	9.883	10.9547	3
2	3	10.716	11.6973	3

6.3 Pruebas de consumo energético

En el apartado del análisis se mencionó que el Arduino Mini Pro de 3.3V tiene un consumo de 4.7mA en modo activo y que activando el modo dormido el consumo pasa a ser de 0.9mA, de la cual 0.85mA son consumidos por el LED de power. Esto se traduciría a una duración de 18

días en modo activo y 97 días en modo dormido sin desactivar el LED. Sin embargo esto es solo la corriente que drena el Arduino.

En este apartado todas las duraciones se calculan a partir de la corriente utilizada con 4 pilas AA de 1.5V en serie dando una capacidad total de 2100mAh. Hay que tener en cuenta que en realidad el consumo eléctrico se mide en vatios, pero el voltaje utilizado es el mismo para todos los dispositivos, 3,3V.

En la siguiente tabla se han anotado el consumo del sistema en función de si un dispositivo está en modo activo o modo dormido. Tal y como se muestra en la siguiente tabla, si añadiéramos el consumo de todos los elementos en modo activo la corriente drenada sería de 54.2mA.

Tabla 10. Corriente drenado por el sensor en función de los dispositivos activos

Arduino	HX711	XBee	Display 7 Seg.	Corriente drenada (mA)
Activo	Activo	Activo	Encendido	54.2
Dormido	Dormido	Dormido	Apagado	1.63
Activo	Activo	Activo	Apagado	46.7
Activo	Activo	Dormido	Encendido	34.7
Activo	Activo	Dormido	Apagado	27.0
Dormido	Activo	Dormido	Apagado	23.4

Se puede observar también que en el modo dormido la corriente que se drena es de 1.63mA, que se traduce a una duración de 53 días. Cuando se tiene que enviar una medida el sensor drena 46.7mA pero solo durante el tiempo que permanece despierto tras cerrarse la puerta, que es de 0.5 segundos. Si suponemos que al día la cámara frigorífica se abre 20 veces, el drenaje promedio es de:

$$\frac{46.7 * (20 * 0.5) + 1.63 * (86400 - 20 * 0.5)}{86400} = 1.635mA$$

Ecuación 3. Cálculo de la corriente drenada promedio

Podemos ver que el drenaje correspondiente al sensor despierto es aproximadamente un 0.30% del gasto total. Por lo tanto la duración también sería de 53 días.

7 Conclusiones y líneas futuras

7.1 Conclusiones

En este proyecto se ha propuesto, analizado y desarrollado un sistema de gestión de la comida en un restaurante. Para ello se han utilizado diversos conocimientos adquiridos durante el máster, especialmente de asignaturas de la rama de electrónica y telemática.

Para el sensor, se ha realizado el montaje completo, buscando y adquiriendo todos los materiales necesarios. Se han utilizado tres celdas de carga para crear tres básculas que miden pesos de forma independiente. Además el sensor incluye botones para poder recalibrarlos según las necesidades del personal. También se ha añadido un display para que en caso de que sea necesario se pueda conocer el peso directamente desde el sensor, sin tener que acceder a la interfaz web creada para ello. Finalmente se ha incluido un portapilas para cuatro pilas AA que proporciona el voltaje suficiente para el funcionamiento del sensor.

Respecto al problema del consumo energético al final se ha conseguido una duración teórica de 53 días. El consumo conseguido podría ser del orden de cientos de veces menor si se hubiera utilizado un Waspmote, por ejemplo que consume pocos µA en modo dormido, pero también hay que tener en cuenta que el precio de un Arduino Mini es muchísimo más accesible (alrededor de tres euros). Esto lo hace apropiado probar determinados prototipos como este sistema, logrando una duración correcta para realizar estas pruebas.

Una vez verificado que el sensor es capaz de generar los datos necesarios de forma automática, se han configurado los XBee para proporcionar una comunicación inalámbrica entre sensor y servidor consumiendo la mínima cantidad de energía posible.

Finalmente se ha implementado un servidor que recoge estos datos y los muestra en una interfaz web tal que el usuario pueda interactuar con ella. También se realiza un procesado de estos datos para pedir la comida de forma automática a los proveedores.

Por lo tanto se ha conseguido un sistema que es capaz de monitorizar el consumo de comida en un restaurante y pedir la cantidad necesaria en el momento adecuado, todo de forma automática.

Con esto se consiguen diversas ventajas, como un mejor control del stock disponible permitiendo al encargado ver fácilmente el material necesario cada semana para hacer el inventario, evitar los despilfarros por pedir comida de más, ofrecer comida más fresca, no congelada, a la clientela y ahorrar en gastos de congelador, que son equipos cuyo consumo energético es muy alto.

Por otro lado el sistema propuesto también tiene ciertas desventajas como el hecho de que el restaurante se puede quedar sin stock o sobrar comida si la demanda es imprevisiblemente alta o baja. Esto se podría evitar mediante un mejor procesado estadístico de los datos. Este sistema también supone que se necesita un mayor espacio en la cámara frigorífica, aunque no será mucho más que en la operativa actual, ya que en esta se suele guardar la comida necesaria para unos dos o tres días.

7.2 Líneas futuras

Una vez desarrollada y probada la base del sistema, este se podría mejorar en diversos aspectos.

En relación al sensor, tal y como se acaba de comentar se podría probar su funcionamiento en otros dispositivos como el Waspmote. También se podría probar directamente con un MCU ATmega utilizando por ejemplo un convertidor Buck para proporcionar el voltaje necesario de forma eficiente. Esto daría paso a un posterior prototipado. También se podría agregar otros sensores aparte de las celdas de carga como por ejemplo sensores de temperatura y humedad, que pueden aportar datos relevantes respecto al funcionamiento de una cámara frigorífica.

Por otro lado en el servidor, el sistema debería también incorporar todas las soluciones de seguridad que se consideren necesarias. Además de utilizar una interfaz web, también se podría utilizar una aplicación de escritorio, que es una solución más habitual comercialmente para este tipo de sistemas.

Por otro lado su interconexión con otros sistemas podría brindarle una mayor funcionalidad. Por ejemplo, si el sistema tuviera los datos de las comandas que se introducen en el sistema TPV, el sistema podría utilizarse por ejemplo para poder detectar un uso inapropiado del material del restaurante. En un paso más allá, con esta información también se podría poner raciones automáticamente para que el cocinero proceda directamente a la elaboración del plato en vez de que el cocinero tenga que sacar la comida de la nevera.

En cuanto al procesado de datos para pedir, el sistema podría tener en cuenta más parámetros estadísticos para realizar una mejor predicción del material que se va a gastar en el ciclo siguiente. Por ejemplo podría tenerse en cuenta que en determinadas épocas que coinciden con festividades la demanda es significativamente superior a la habitual.

Finalmente en el futuro su unión con otros sistemas de IoT permitiría que sus datos puedan ser compartidos pudiendo generar información valiosa por ejemplo para conocer tendencias de consumo en la población.

8 Bibliografía y referencias

[1] United Nations Environment Programme. (Septiembre, 2013). *Food waste harms climate, water, land and biodiversity - new FAO report.* [En línea]. http://www.unep.org/newscentre/Default.aspx?DocumentID=2726&ArticleID=9611

[2] World Food Programme (2015). *Hunger Statistics*. [En línea]. http://www.wfp.org/hunger/stats

[3] S. Anil Kumar, N. Suresh (2009). Operations Management. 1.ed. New age international publishers.

[4] Tutoriales 5 Hertz Electrónica. *Celdas de carga*. [En línea]. http://www.electrrio.com/Anuncios/SensoresyModulos/Peso/sensordepeso.pdf

[5] Arduino (2016). *Arduino*. [En línea]. https://www.arduino.cc/

[6] Atmel. *ATmega328/P Datasheet complete*. [En línea] http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_Datasheet.pdf

[7] Avia Semiconductor (2016). 24-Bit Analog-to-Digital Converter (ADC) for Weigh Scales. [En línea]. http://www.aviaic.com/UpLoadFile/hx711_brief_en.pdf

[8] Libelium (2016). *Waspmote*. [En línea]. http://www.libelium.com/products/waspmote/

[9] NXP Semiconductors N.V. (2016). *74HC595; 74HCT595 Product datasheet*. [En línea]. http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

[10] ZigBee Alliance (2016). *ZigBee.* [En línea]. http://www.zigbee.org/

[11] Wi-Fi Alliance (2016). *WiFi*. [En línea]. http://www.wi-fi.org/

[12] Bluetooth Special Interest Group (2016). Bluetooth. [En línea]. https://www.bluetooth.com/

[13] Digi (2016). *XBee*. [En línea]. https://www.digi.com/lp/xbee

[14] Apache Friends. *XAMPP*. [En línea]. https://www.apachefriends.org/es/index.html

[15] Python (2016). *Python*. [En línea]. https://www.python.org/

[16] Highsoft. Highcharts. [En línea]. http://www.highcharts.com/

[17] AVR. *avr/sleep.h* [En línea]. http://www.nongnu.org/avr-libc/user-manual/group__avr__sleep.html

[18] Bogde. *HX711*. [En línea]. https://github.com/bogde/HX711

[19] Arduino. *Arduino EEPROM*. [En línea]. https://www.arduino.cc/en/Reference/EEPROM

[20] Digi (Diciembre 2015). *ZigBee RF Modules XBEE2, XBEEPRO2, PRO S2B User Guide*.

[21] Python. *Python dictionaries*. [En línea]. https://docs.python.org/2/tutorial/datastructures.html#dictionaries

[22] Charles Bell (2013). *Beginning Sensor Networks with Arduino and Raspberry Pi*. Apress.

[23] Michael Glass, Yann Le Scourarnec, Elisabeth Naramore, Garey Mailer, Jeremy Stolz, Jason Gerner (2004). *Desarrollo Web con PHP, Apache y MySQL*. Anaya multimedia.

Anexos

A. Código

Este anexo incluye los ficheros de código más relevantes del sistema.

- sensor.ino es el fichero del código del sensor Arduino
- addData.py y stockOrder.py son los dos scripts de Python
- index.php y drawchart.php son los ficheros del código de la página principal de la interfaz de usuarios

A.1 sensor.ino

```
#include "HX711.h"
#include <avr/sleep.h>
#include <EEPROM.h>


//scale_factor obtained from calibration sketch
#define scale_factor1 119530.0
#define scale_factor2 114440.0
#define scale_factor3 508600.0

//VARIABLES INITIALIZATION

//Load cells
HX711 cell1(9, 8);
long new_offset1 = 0;
float kilograms1 = 0;
HX711 cell2(7, 6);
long new_offset2 = 0;
float kilograms2 = 0;
HX711 cell3(5, 4);
long new_offset3 = 0;
float kilograms3 = 0;
int newOffsetPin = 13;

//Sleep
int wakePin0 = 2;                 // reed switch
int wakePin1 = 3;                 // wake up button
int sleepRQPin = 18;              // sleep request pin for XBee
int sleepStatus = 0;              // variable to store a request for sleep
boolean buttonState = LOW;

//7 Segment display common cathode
#define digit_on LOW
#define digit_off HIGH
int selectedCell = 1;
int delayTime = 2000;              // microseconds for each digit
int cathodePins[5] = {10,11,12,14,15};  //5 common cathode pins of the display
int clockPin = 19;                //74HC595 pin 11
int latchPin = 17;                //74HC595 pin 12
int dataPin = 16;                 //74HC595 pin 14
```

```
int digit[10] = {
  63,     //0
  6,      //1
  91,     //2
  79,     //3
  102,    //4
  109,    //5
  125,    //6
  7,      //7
  127,    //8
  103     //9
};
int displayNumber[5] = {0, 0, 0, 0, 0};


void setup() {
  // Set offset, scale and pin mode of load cells
  Serial.begin(9600);
  delay(1000);
  Serial.println("Setting up");
  long offset1 = EEPROMReadlong(0);
  long offset2 = EEPROMReadlong(4);
  long offset3 = EEPROMReadlong(8);
  cell1.set_offset(offset1);
  cell1.set_scale(scale_factor1);
  cell2.set_offset(offset2);
  cell2.set_scale(scale_factor2);
  cell3.set_offset(offset3);
  cell3.set_scale(scale_factor3);
  pinMode(newOffsetPin, INPUT);

  // Set sleeping mode pinModes
  pinMode(wakePin0, INPUT);
  pinMode(wakePin1, INPUT);
  pinMode(sleepRQPin, OUTPUT);

  // Attach sleeping interrputs
  attachInterrupt(0, wakeUpNow, RISING); // use interrupt 0 (pin 2) and run
function wakeUpNow when pin 2 gets HIGH
  attachInterrupt(1, wakeUpNow, RISING); // use interrupt 1 (pin 3) and run
function wakeUpNow when pin 2 gets HIGH

  // Time delay for XBee connection
  digitalWrite(sleepRQPin, LOW);        // wake up and get connection with parent
  delay(3000);

  //7 segment display pinModes
  for(int i=0;i<5;i++) {
    pinMode(cathodePins[i],OUTPUT);
    digitalWrite(cathodePins[i], HIGH);  // off by default
  }
  pinMode(latchPin, OUTPUT);
  pinMode(clockPin, OUTPUT);
  pinMode(dataPin, OUTPUT);

  Serial.println("Setup ready");
}


void loop() {
  //wake up load cell
```

```
  delay(50);
  cell1.power_up();
  cell2.power_up();
  cell3.power_up();

  // button interruption
  if(debounceButton(buttonState) == HIGH && buttonState == LOW) // button pressed
  {
    Serial.println("Despierto por boton");
    buttonState = HIGH;

    // keep awake until door closes
    while(digitalRead(wakePin0) == LOW){

      if(debounceButton(buttonState) == LOW && buttonState == HIGH) // button
released
      {
        buttonState = LOW;
      }

      // switch to next load cell when button pressed
      if(debounceButton(buttonState) == HIGH && buttonState == LOW)
      {
        buttonState = HIGH;
        selectedCell++;
        if (selectedCell == 4) {
          selectedCell = 1;
        }
      }

      // show values in the display
      switch (selectedCell) {
        case 1:
          kilograms1 = cell1.get_units();
          for (int i = 0; i <15; i++) {
            updateDisp(1, kilograms1);
          }
          break;
        case 2:
          kilograms2 = cell2.get_units();
          for (int i = 0; i <15; i++) {
            updateDisp(2, kilograms2);
          }
          break;
        case 3:
          kilograms3 = cell3.get_units();
          for (int i = 0; i <15; i++) {
            updateDisp(3, kilograms3);
          }
          break;
      }
      calib_offset(selectedCell);
    }
  }


  // closed door interruption
  kilograms1 = cell1.get_units();       //.get_units() returns a float
  kilograms2 = cell2.get_units();
  kilograms3 = cell3.get_units();
```

```
  // wake up XBee and send data
  digitalWrite(sleepRQPin, LOW);
  delay(50);
  Serial.print("3");                // number of sensors
  Serial.print(" 1 ");              //Id
  Serial.print(kilograms1, 3);
  Serial.print(" 2 ");              //Id
  Serial.print(kilograms2, 3);
  Serial.print(" 3 ");              //Id
  Serial.println(kilograms3, 3);
  delay(500); // hald a second is enough to send the data

  // put HX711, XBee and Arduino in sleep mode
  cell1.power_down();
  cell2.power_down();
  cell3.power_down();
  digitalWrite(sleepRQPin, HIGH);
  delay(100);     // this delay is needed, the sleep function will provoke a
Serial error otherwise
  sleepNow();     // sleep function called here
}


void wakeUpNow() {   // handle interruption after wakeup. After this handle, it
continues from where it stopped when going to sleep
}


void sleepNow() {    // put the arduino to sleep
  // sleep mode is set here
  set_sleep_mode(SLEEP_MODE_PWR_DOWN);
  // enable the sleep bit in the mcucr register
  sleep_enable();
  // use interrupt 0 (pin 2) and run function wakeUpNow when pin 2 gets LOW
  attachInterrupt(0,wakeUpNow, RISING);
  // use interrupt 1 (pin 3) and run function wakeUpNow when pin 3 gets LOW
  attachInterrupt(1,wakeUpNow, RISING);
  // enter into sleep mode
  sleep_mode();

  // THE PROGRAM CONTINUES FROM HERE AFTER WAKING UP
  // first thing after waking from sleep: disable sleep...
  sleep_disable();
  // disable interrupts so the wakeUpNow code will not be executed
  detachInterrupt(0);
  detachInterrupt(1);

  // during normal running time.
}


void allDigitOff() {
  digitalWrite(cathodePins[0], digit_off);
  digitalWrite(cathodePins[1], digit_off);
  digitalWrite(cathodePins[2], digit_off);
  digitalWrite(cathodePins[3], digit_off);
  digitalWrite(cathodePins[4], digit_off);
}


void updateDisp(int selectedCell, float kilograms){
```

```
  kilograms = int(kilograms*100);
  displayNumber[0] = selectedCell;
  displayNumber[1] = int(kilograms)/1000;
  displayNumber[2] = (int(kilograms)%1000)/100;
  displayNumber[3] = (int(kilograms)%100)/10;
  displayNumber[4] = (int(kilograms)%100)%10;

  //0
  digitalWrite(latchPin, LOW); // take the latch pin low so the LEDs don't change
while you're sending in bits:
  shiftOut(dataPin, clockPin, MSBFIRST, digit[displayNumber[0]]); // shift out
the bits
  digitalWrite(latchPin, HIGH); // take the latch pin high so the LEDs will light
up
  digitalWrite(cathodePins[0], digit_on);
  delayMicroseconds(delayTime);
  allDigitOff();

  //1
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, digit[displayNumber[1]]);
  digitalWrite(latchPin, HIGH);
  digitalWrite(cathodePins[1], digit_on);
  delayMicroseconds(delayTime);
  allDigitOff();

  //2
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, digit[displayNumber[2]] + 128); //+128 is
the decimal point
  digitalWrite(latchPin, HIGH);
  digitalWrite(cathodePins[2], digit_on);
  delayMicroseconds(delayTime);
  allDigitOff();

  //3
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, digit[displayNumber[3]]);
  digitalWrite(latchPin, HIGH);
  digitalWrite(cathodePins[3], digit_on);
  delayMicroseconds(delayTime);
  allDigitOff();

  //4
  digitalWrite(latchPin, LOW);
  shiftOut(dataPin, clockPin, MSBFIRST, digit[displayNumber[4]]);
  digitalWrite(latchPin, HIGH);
  digitalWrite(cathodePins[4], digit_on);
  delayMicroseconds(delayTime);
  allDigitOff();
}


void calib_offset(int selectedCell) {
  if (digitalRead(newOffsetPin) == HIGH){
    Serial.print("Setting new offset ");
    Serial.println(selectedCell);
    switch (selectedCell) {
        case 1:
          new_offset1 = cell1.read_average();
          cell1.set_offset(new_offset1);
```

```
          EEPROMWritelong(0, new_offset1);           //save it in the EEPROM
          break;
        case 2:
          new_offset2 = cell2.read_average();
          cell2.set_offset(new_offset2);
          EEPROMWritelong(4, new_offset2);           //save it in the EEPROM
          break;
        case 3:
          new_offset3 = cell3.read_average();
          cell3.set_offset(new_offset3);
          EEPROMWritelong(8, new_offset3);           //save it in the EEPROM
          break;
    }
  }
}


boolean debounceButton(boolean state) {
  boolean stateNow = digitalRead(wakePin1); //read button value
  if(state!=stateNow)
  {
    delay(10);
    stateNow = digitalRead(wakePin1); //delay to confirm that the button has been
pressed
  }
  return stateNow;
}


void EEPROMWritelong(int address, long value) {
  //decomposition from a long to 4 bytes by using bitshift.
  //One = Most significant -> Four = Least significant byte
  byte four = (value & 0xFF);
  byte three = ((value >> 8) & 0xFF);
  byte two = ((value >> 16) & 0xFF);
  byte one = ((value >> 24) & 0xFF);

  //Write the 4 bytes into the eeprom memory.
  EEPROM.write(address, four);
  EEPROM.write(address + 1, three);
  EEPROM.write(address + 2, two);
  EEPROM.write(address + 3, one);
}


long EEPROMReadlong(long address) {
  //Read the 4 bytes from the eeprom memory.
  long four = EEPROM.read(address);
  long three = EEPROM.read(address + 1);
  long two = EEPROM.read(address + 2);
  long one = EEPROM.read(address + 3);

  //Return the recomposed long by using bitshift.
  return ((four << 0) & 0xFF) + ((three << 8) & 0xFFFF) + ((two << 16) &
0xFFFFFF) + ((one << 24) & 0xFFFFFFFF);
}
```

A.2 addData.py

```
#!/usr/bin/python
import MySQLdb
import time
import datetime
import serial


print "Executing add data script"

# serial port configuration
def open_serial_port():
    ser = serial.Serial(
        port='COM6',
        baudrate=9600,
        parity=serial.PARITY_NONE,
        stopbits=serial.STOPBITS_ONE
    )
    if not ser.isOpen:
        ser.open()
    print "Is port open? " + str(ser.isOpen())


#MAIN
open_serial_port()

#Sensor initialization
# Open database connection and create cursor
db = MySQLdb.connect("localhost", "restsensor", "", "restdb")
cursor = db.cursor()

# 1 Sensors dictionary initialization
sensors = {}
cursor.execute("SELECT * FROM sensors")
db.commit()
prevsensors = cursor.fetchall()
for i in range(len(prevsensors)):
    sensors[prevsensors[i][0]] = prevsensors[i][1]
print "SENSORS", sensors

# 2 Prevdada dictionary initialization
prevdata = {}
sensorsvals = sensors.values()
for i in range(len(sensors)):
    prevdata[sensorsvals[i]] = float(0)
print "PREVDATA", prevdata

#Main loop
while 1:

    # 1 Read data from sensor
    input = {}
    serialinput = ser.readline()  #readline read until carriage return
    # Loop stopped here until a data is received
    inputSplit = serialinput.split()
    Nsensors = inputSplit[0]
    if (Nsensors == str(3)) and (len(inputSplit) == 7):
        for i in range(1, 2*int(Nsensors), 2):
            id = inputSplit[i]
            weight = inputSplit[i+1]
```

```
            input[id] = weight
        print "INPUT", input

        # 2 Create sensors dictionary
        cursor.execute("SELECT * FROM sensors")
        db.commit()
        prevsensors = cursor.fetchall()
        for i in range(len(prevsensors)):
            sensors[prevsensors[i][0]] = prevsensors[i][1]
        print "SENSORS", sensors


        # 3 Create currentdata dictionary.
        # key = value of sensors
        # value = value of input
        currentdata = {}
        for i in range(len(input)):
            k = sensors.get(int(input.keys()[i]))
            v = input.values()[i]
            currentdata[k] = float(v)
        print "CURRENTDATA", currentdata


        # 4 Check changed values
        insert = False  #insert false: no insert needed, no data change
        update = "No"
        values = []
        ts = time.time()
        st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
        for i in range(len(currentdata)):
            currentk = currentdata.keys()[i]
            currentv = currentdata.get(currentk)
            prevv = prevdata.get(currentk)
            if (currentv < prevv - 0.010) or (currentv > prevv + 0.010) or
(currentv > 0): # weight changed in more than 10 grams
                if (currentv > prevv + 5.0): # There is more weight than before
                    update = "Yes"
                else:
                    update = "No"
                insert = True

                # 5 Prepare insert statement
                params = (currentk, st, currentv, update)
                values.append(params)
                # Values is a list of tuple. Each tuple is one type of food

                # 6 Update prevdata for next iteration
                prevdata[currentk] = currentv

        print "VALUES", values

        sql = """INSERT INTO sensor_data(food_id, date, weight, is_update)
                 VALUES (%s, %s, %s, %s)"""

        try:
            # Execute the SQL command
            cursor.executemany(sql, values)

            # Commit your changes in the database
            # This method sends a COMMIT statement to the MySQL server,
committing the current transaction.
```

```
            # Since by default Connector/Python does not autocommit, it is
important to call this method after every
            # transaction that modifies data for tables that use transactional
storage engines.
            db.commit()
        except:
            print "Ha ocurrido un fallo en el insert"
            # Rollback in case there is any error
            db.rollback()

            # disconnect from server
            db.close()

    else:
        print "Input no concuerda con la trama de datos"
```

A.3 stockOrder.py

```
#!/usr/bin/python
from __future__ import division
import MySQLdb
import time
import datetime
import math


def send_email(user, pwd, recipient, subject, body):
    import smtplib

    gmail_user = user
    gmail_pwd = pwd
    FROM = user
    TO = recipient if type(recipient) is list else [recipient]
    SUBJECT = subject
    TEXT = body

    # Prepare actual message
    message = """From: %s\nTo: %s\nSubject: %s\n\n%s
    """ % (FROM, ", ".join(TO), SUBJECT, TEXT)
    try:
        server = smtplib.SMTP("smtp.gmail.com", 587)
        server.ehlo()
        server.starttls()
        server.login(gmail_user, gmail_pwd)
        server.sendmail(FROM, TO, message)
        server.close()
        print 'successfully sent the mail'
    except:
        print "failed to send mail"


# Main
# Open database connection
db = MySQLdb.connect("localhost", "restsensor", "", "restdb")
cursor = db.cursor()

# 1 Day of the week
today = datetime.datetime.today().isoweekday()
ts = time.time()
st = datetime.datetime.fromtimestamp(ts).strftime('%H')
print today, st

while True:
    #if it's Monday or Friday at 1 o'clock
    if ((today == 1 or today == 5) and st == "1"):
        print "hay que pedir"
        values = []
        ss = 1 # kg safety margin

        if (today == 1):
            #cycle is the cycle that has just finished
            cycle = 2
            order_cycle = 1
        elif (today == 5):
            cycle = 1
            order_cycle = 2
```

```
        #Restaurant data
        cursor.execute("SELECT * FROM restaurant_data")
        rest_result = cursor.fetchall()[0]
        print rest_result
        rest_name = rest_result[0]
        rest_address = rest_result[1]
        rest_phone = rest_result[2]
        user = rest_result[3]
        pwd = rest_result[4]

        # 2 Create new row in prediction_data
        cursor.execute("SELECT * FROM `sensors` JOIN types_of_food ON
types_of_food.food_id = sensors.food_id")
        food_list = cursor.fetchall()
        for i in range(len(food_list)):

            #variables initialization for each type of food
            update_quantity = 0
            current_quantity = 0
            spending = 0
            current_i = 1
            prev_CMA = 0
            params = ()

            food_id = food_list[i][1]
            type_of_food = food_list[i][3]
            print ""
            print food_id, type_of_food

            # 3 Calculate the spendings of this passed cycle
            cursor.execute("SELECT weight FROM sensor_data WHERE food_id = '" +
str(food_id) + "' and is_update = 'Yes' ORDER BY registry_id DESC LIMIT 1")
            update_quantity = cursor.fetchall()[0][0]
            cursor.execute("SELECT weight FROM sensor_data WHERE food_id = '" +
str(food_id) + "' and is_update = 'NO' ORDER BY registry_id DESC LIMIT 1")
            current_quantity = cursor.fetchall()[0][0]
            spending = update_quantity - current_quantity
            print "Spending: ", update_quantity, current_quantity, spending

            # 4 Read CMA and iteration of the last row of this same cycle
            cursor.execute("SELECT CMA, iteration FROM prediction_data WHERE
food_id = '" + str(food_id) + "' and cycle = " + str(cycle) + " ORDER BY pred_id
DESC LIMIT 1")
            result = cursor.fetchall()
            if (result != ()):
                current_i = int(result[0][1] + 1)
                prev_CMA = result[0][0]

            # 5 Calculate CMA and iteration of the new row and prepare parameters
for insert
            new_CMA = (((current_i - 1) / current_i) * prev_CMA) +
(spending/current_i)
            params = (cycle, food_id, spending, new_CMA, current_i)
            print "Params", params
            values.append(params)


            # 6 Finished registring this passed cycle. Now calculate the order
quantity. Use the CMA of the other cycle as it's the cycle that it's going to
begin.
```

```
            cursor.execute("SELECT CMA FROM prediction_data WHERE food_id = '" +
str(food_id) + "' and cycle = " + str(order_cycle) + " ORDER BY pred_id DESC
LIMIT 1")
            result = cursor.fetchall()
            if (result != ()):
                prev_order_CMA = result[0][0]
            else:
                prev_order_CMA = update_quantity
            order_quantity = math.ceil(prev_order_CMA - current_quantity + ss)
            print "Order quantity", order_quantity

            # 7 Generate email
            #Supplier data
            cursor.execute("SELECT * FROM suppliers WHERE food_id = '" +
str(food_id) + "'")
            supplier_result = cursor.fetchall()[0]

            recipient = supplier_result[2]
            subject = supplier_result[3]
            text = supplier_result[4]

            text = text.replace("[rest_name]", rest_name)
            text = text.replace("[rest_address]", rest_address)
            text = text.replace("[rest_phone]", rest_phone)
            text = text.replace("[necessary_food_quantity]", str(order_quantity))
            text = text.replace("[type_of_food]", type_of_food)

            print user, pwd, recipient, subject
            print text

            send_email(user, pwd, recipient, subject, text)


        #End of for

        print values
        sql = """INSERT INTO prediction_data(cycle, food_id, spendings, CMA,
iteration)
                VALUES (%s, %s, %s, %s, %s)"""
        try:
            # Execute the SQL command
            cursor.executemany(sql, values)

            # Commit your changes in the database
            # This method sends a COMMIT statement to the MySQL server,
committing the current transaction.
            # Since by default Connector/Python does not autocommit, it is
important to call this method after every
            # transaction that modifies data for tables that use transactional
storage engines.
            db.commit()
        except:
            print "Ha ocurrido un fallo en el insert"
            # Rollback in case there is any error
            db.rollback()
            # disconnect from server
            db.close()

        time.sleep(3600) #sleep after making the order
    else:
        time.sleep(1800)
```

A.4 index.php

```
<html>
    <head>
        <meta charset="UTF-8">
        <link href="css/reset.css" rel="stylesheet" >
        <link href="css/style.css" rel="stylesheet">
        <script type="text/javascript"
src="https://ajax.googleapis.com/ajax/libs/jquery/1.8.2/jquery.min.js"></script>
        <script src="https://code.jquery.com/jquery.js"></script>
        <script src="http://code.highcharts.com/stock/highstock.js"></script>
        <script src="http://code.highcharts.com/modules/exporting.js"></script>
        <title>Sistema de control</title>
    </head>

    <body>

        <ul class="nav_bar">
            <li class="inicio"><a href="index.php">Inicio</a></li>
            <li><a href="">Proveedores</a></li>
            <li><a href="">Configuración</a>
                <ul>
                    <li><a href="sensors.php">Configurador de básculas</a></li>
                    <li><a href="">Datos del restaurante</a></li>
                </ul>
            </li>
            <li class="restaurantes"><a href="">Restaurantes</a>
                <ul>
                    <li><a href="">Restaurante1</a></li>
                    <li><a href="">Restaurante2</a></li>
                    <li><a href="">Restaurante3</a></li>
                </ul>
            </li>
        </ul>

        <section class="graph_list">
            <p>Estas gráficas muestra la cantidad disponible de cada tipo de
comida </p>
            <br>
            <ul>
                <li>
                    <figure>
                        <div id="container1"></div>
                        <?php require_once("php-includes/drawchart1.inc.php"); ?>
                    </figure>
                </li>
                </br></br></br></br></br>
                <li>
                    <figure>
                        <div id="container2"></div>
                        <?php require_once("php-includes/drawchart2.inc.php"); ?>
                    </figure>
                </li>
                </br></br></br></br></br>
                <li>
                    <figure>
                        <div id="container3"></div>
                        <?php require_once("php-includes/drawchart3.inc.php"); ?>
                    </figure>
                </li>
            </ul>
```

```
        </section>

        <footer>
            <p>Proyecto de control automático de existencias</a></p>
        </footer>

    </body>
</html>
```

A.5 drawchart.inc.php

```
<?php
require_once("db.func.php");

//create an instance of SensorDataTable. This class contains all the functions to
connect, disconnect, select, insert and update data to the database
$dataTable = new SensorDataTable();

//name of the type of food
$rawname = $dataTable->getName1();
$name = $rawname[0][1];

//from sensor_data table
$rawdata = $dataTable->getSensor1();

//two arrays for values and times
$values;
$timeArray;

//get the values and timestamps
for($i = 0 ;$i<count($rawdata);$i++){
    $ values[$i]= $rawdata[$i][3];
    $time= $rawdata[$i][2];
    $date = new DateTime($time);
    $timeArray[$i] = $date->getTimestamp()*1000;
}
?>

<script>
Highcharts.setOptions({
            global: {
                useUTC: false
            }
        });

chart = new Highcharts.Chart({
    chart: {
        renderTo: 'container1',
        zoomType: 'x'
    },
    rangeSelector : {
        enabled: true,
        buttons: [{
            type: 'day',
            count: 1,
            text: '1d'
        }, {
            type: 'day',
            count: 3,
            text: '5d'
        }, {
            type: 'day',
            count: 8,
            text: '8d'
        }, {
            type: 'month',
            count: 1,
            text: '1m'
        }, {
            type: 'all',
```

```
            text: 'All'
        }]
    },
    title: {
        text: '<?php echo $name;?>'
    },
    xAxis: {
        type: 'datetime'
    },
    yAxis: {
        minPadding: 0.2,
        maxPadding: 0.2,
        title: {
            text: 'Cantidad displonible (kg)',
            margin: 10
        }
    },
    series: [{
        data: (function() {
                var data = [];
                <?php
                    for($i = 0 ;$i<count($rawdata);$i++){
                ?>
                data.push([<?php echo $timeArray[$i];?>,<?php echo
$ values[$i];?>]);
                <?php } ?>
                return data;
            })(),
        step: 'left',
        name: 'kg'
    }],
    credits: {
            enabled: false
    }
});

</script>
```