Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

ESCUELA DE INGENIERIAS INDUSTRIALES

Grado en Ingeniería Electrónica Industrial y Automática

Sistema de reconocimiento de actividades para el cuidado de personas ancianas independientes en su entorno domiciliario

Autor:

Álvarez Asensio, Juan

Tutor:

Zalama Casanova, Eduardo

Departamento de Ingeniería de Sistemas y Automática

Valladolid, julio 2021.

Resumen:

El número de personas mayores de 65 años que viven solas en sus casas ha aumentado en los últimos años. Esto, en muchas ocasiones, puede derivar en sensación de vulnerabilidad debido al deterioro de la salud que se experimenta con el envejecimiento. Sin embargo, a través del seguimiento de la rutina de estas personas, con el fin de añadir a sus hábitos las recomendaciones del personal social y sanitario, es posible mejorar su calidad de vida y fomentar su autonomía.

Por ello, en este trabajo se pretende diseñar un sistema de reconocimiento de actividades domésticas que facilite al personal social y sanitario la realización de sus labores y así poder mejorar los servicios y atenciones que prestan a la población mayor. Se desea también diseñar un algoritmo de reconocimiento de actividades basándose en la rutina del usuario, de forma que se puedan detectar las actividades básicas de la vida diaria con el menor número posible de sensores.

Palabras clave:

Personas mayores, Sensores, Reconocimiento de Actividades, Rutina Diaria, Personal Social/Sanitario.

Abstract:

Nowadays, situations involving people who are older than 65 and live alone in their homes are getting more frequent. The main problem is that the independence of this people gets more difficult as they grow older because of the healthy changes that are used to break through at these ages. Nevertheless, it is possible to improve their life quality and encourage their autonomy by following their daily living routine and including new activities in it that are recommended by social and health workers.

The main objective of this project is to design a home activity recognition system to help social and health workers in their tasks of following the daily living routines of the elderly, so it could be easier for them to improve the cares and services they give. Also, it is pursued to design an activity recognition algorithm by studying the user's daily routine, so basic activities of daily living can be identified with the lower amount possible of sensors.

Keywords:

Elderly, Sensors, Activity Recognition, Daily Living Routine, Social/Health Workers

Índice

1. Introducción y objetivos

1.1. Marco del proyecto

El número de personas mayores de 65 años que viven solas en España ha ido aumentando en los últimos años (ver Figura 1). Es frecuente que, a medida que pasa el tiempo, estas personas comiencen a necesitar algún tipo de cuidados que requiera la atención del personal médico o de ayudas sociales. Según el Instituto Nacional de Estadística (INE) [9], el grupo de edad que más vive estas situaciones es de 85 años o más, como se muestra en la Tabla 1.

A partir de cierta edad, el estado salud de las personas se va deteriorando debido a numerosos aspectos, dando lugar a problemas cada vez más comunes como problemas sensoriales (pérdida auditiva, pérdida de visión, etc.), de nutrición y alimentación, de sueño, de incontinencia, reducción progresiva de la motricidad, conflictos mentales y transtornos de conducta, entre otros aspectos, como refleja un artículo de la página de Cuidania [4], una agencia de Servicio Doméstico en Madrid, especializada en el cuidado de personas ancianas. Es por esta razón por la que son necesarias con más frecuencia las revisiones médicas para gente que se encuentra por encima de los 65 años.

Muchas de estas personas viven de forma dependiente junto con familiares o en residencias especializadas, mientras que otras viven solas. Estas personas que viven de forma independiente están más expuestas a que, en caso de que les surja algún tipo de problema, no tengan a nadie cerca que les pueda ayudar o socorrer.

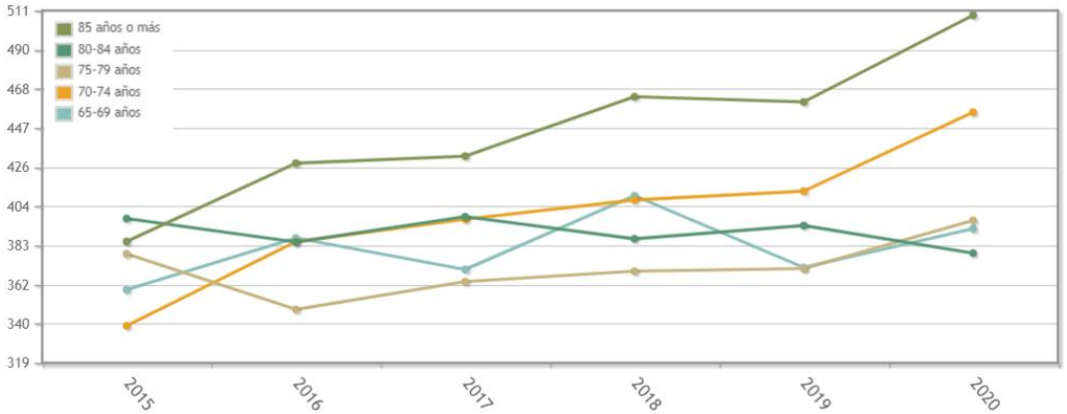


Figura 1. Instituto Nacional de Estadística

	Ambos sexos				
	65-69 años	70-74 años	75-79 años	80-84 años	85 años o más
2020					
Personas que viven solas	392.2	455.5	396.7	378.9	508.2
2019					
Personas que viven solas	371.1	412.6	370.5	393.9	461.1
2018					
Personas que viven solas	410.1	407.8	369.1	386.7	463.9
2017					
Personas que viven solas	370.0	397.3	363.4	398.7	431.6
2016					
Personas que viven solas	387.1	385.1	348.3	384.9	427.8
2015					
Personas que viven solas	359.0	339.4	378.5	397.7	385.2

Fuente:
Instituto Nacional de Estadística

Tabla 1. Evolución del número de personas que viven solas en España

También existen riesgos de padecer problemas de salud relacionados con el hábito de la persona anciana como lo son, por ejemplo, la diabetes, la osteoporosis o la artritis, los cuales en muchos casos son inevitables, pero igualmente se pueden mantener controladas de forma que se pueda desarrollar una vida normal con ellas. Para ello, mantener un cierto control sobre los hábitos de estas personas podría ser conveniente para prevenir estas enfermedades, sino para tenerlas bajo control. Entre estos hábitos influye, entre otros factores, la alimentación, la higiene, las horas de sueño y el ejercicio diario. Por ejemplo, una persona anciana que mantenga un hábito de pasear todos los días y una dieta adecuada puede ver reducida las posibilidades de padecer diabetes, enfermedades de tipo cardiovascular e incluso el riesgo de mortalidad, como así lo refleja un artículo de la revista digital Geriatric Area [7].

Figura 2. Medalla de Teleasistencia de la Cruz Roja

El hecho de que las personas ancianas necesiten cada vez más cuidados dificulta la vida diaria de aquellas que viven de forma autónoma. Este tipo de cuidados dependerán de cada persona: qué hábitos tiene, en qué entorno vive, cuáles son sus limitaciones y su estado de salud, etc. Actualmente, existen muchos sistemas pensados para mejorar la calidad de vida de las personas mayores y fomentar su autonomía.

Por ejemplo, es frecuente entre estos colectivos que se disponga de asistencia doméstica por parte de la Cruz Roja o de alguna agencia independiente. En el caso de la Cruz Roja [4], existe una medalla que permite comunicar a la persona que la lleva con un centro de teleasistencia. Como se muestra en la Figura 2, esta medalla consta de un botón y un terminal. Al pulsarse el botón, se activa el manos libres del terminal para poder hablar, desde cualquier habitación de la casa, con un profesional que le asistirá en caso de que sea necesario. Esta medalla debe ser portada por el usuario colgada del cuello o en la muñeca, convenientemente siempre que esté en casa, incluso en la ducha (el dispositivo es impermeable). De esta forma, en caso de que al usuario le surja una necesidad o una emergencia, tan solo tendrá que pulsar el botón para comunicarse con el centro de teleasistencia donde le atenderá un profesional, y desde donde se movilizarán todos los recursos necesarios.

Este tipo de sistemas presentan muchas ventajas en cuanto al cuidado de personas ancianas, permitiéndoles mantener su vida de forma independiente. Sin embargo, puede presentar algunos inconvenientes en caso de que se requiera asistencia durante una emergencia en la que el usuario esté inconsciente o no tenga capacidad para accionar el pulsador.

Una alternativa a este tipo de sistemas, que dependen del accionamiento del usuario para su activación, son los sistemas autónomos inteligentes, destinados al cuidado de personas mayores. Hay estudios [6] en los que se aplican este tipo de sistemas para planear el tiempo que trabajan las enfermeras para poder ofrecer un cuidado efectivo a los pacientes de enfermedades neurodegenerativas como el Alzheimer.

Por otro lado, hay sistemas inteligentes destinados al cuidado de personas ancianas dentro del entorno doméstico basados en el reconocimiento de actividades. Este tipo de sistemas son cada vez más frecuentes. Algunos estudios anteriores sobre el reconocimiento de actividades domésticas utilizan sistemas de visión. Sin embargo, estos sistemas no son adecuados para el cuidado de personas mayores, dados los problemas de privacidad que puede ocasionar el uso de cámaras y/o micrófonos.

En este proyecto se va a desarrollar un sistema inteligente que, aplicando la tecnología actual y utilizando sensores no invasivos que no comprometen la

privacidad del usuario captando imágenes o sonido, será capaz de reconocer qué actividad se está realizando en el domicilio en cada momento. De esta forma, será más fácil, tanto para el usuario que viva solo en casa como para el personal sanitario o social que así lo requiera, llevar un control de sus hábitos.

Por ejemplo, sería de gran utilidad para el propio usuario saber cuándo se ha tomado la medicación o cuándo se ha duchado, sobre todo si presenta problemas de memoria; pero también para las revisiones médicas puede ser conveniente saber las horas de sueño que tiene el usuario o si sale a pasear o a realizar algún tipo de actividad física con frecuencia.

También podría detectar situaciones de emergencia en las que la persona no está consciente, que es uno de los inconvenientes que tiene la medalla de la Cruz Roja. Por ejemplo, si se lleva mucho tiempo sin detectar ninguna actividad en la casa y se sabe que la persona sigue dentro, podría ser un indicio de que el usuario está en peligro y de esta forma se podría actuar con rapidez y eficacia para evitar posibles accidentes o incluso la pérdida de la vida del usuario.

La idea es crear una red de sensores comerciales distribuidos por todo el domicilio, como sensores de detección de vibraciones, de contacto para puertas y ventanas, sensores de humedad y temperatura, de luminosidad, de movimiento, entre otros. En trabajos previos, los sensores han sido diseñados para ser dispositivos "*tape on and forget*" (pegar y olvidarse), lo que permite una instalación rápida y sencilla en entornos domésticos.

En trabajos anteriores, cada dispositivo consiste en un pequeño sensor que recoge la información y una tarjeta de adquisición de datos. En la actualidad, muchos sensores comerciales se comunican de manera remota a través de protocolos de comunicación como por ejemplo el protocolo de bajo consumo *Zigbee,* con un controlador que depende de la marca de los sensores, es decir, cada marca tiene su propio controlador. Por ejemplo, para poder conectar los dispositivos de la marca *Xiaomi* será necesario el controlador *MiHub* de esta marca. Sin embargo, hay alternativas que permiten comunicar los sensores utilizando una *Raspberry pi* como controlador.

En cuanto al tratamiento de los datos, algunos trabajos en los que se basa este proyecto hacen un estudio estadístico sobre la activación de los sensores con el fin de poder determinar qué actividad se está realizando a partir de la activación de determinados sensores, otros aplican algún algoritmo de reconocimiento de actividades. En ambos casos, es necesario una fase de experimentación previa: el etiquetado. En esta fase se recogen datos de los sensores durante varias semanas y se asigna a cada combinación de medidas una actividad.

1.2. Objetivos

El objetivo principal de este proyecto es desarrollar un sistema autónomo inteligente de reconocimiento de actividades que realiza un usuario en su domicilio.

Este objetivo se debe cumplir siguiendo ciertas especificaciones:

- En primer lugar, se pretende desplegar una red de sensores inalámbricos dentro de un domicilio. Los sensores deben estar distribuidos por todas las habitaciones de la casa para que con los datos recogidos se puedan detectar un mayor número de actividades.

- El algoritmo de reconocimiento de actividades debe ser capaz de reconocer, al menos, las siguientes actividades: ducharse, ir al baño, cocinar, trabajar con el ordenador, dormir, despertarse, vestirse, entrar en casa, salir de casa, ver la televisión. Estas actividades son consideradas como actividades básicas de la vida diaria y son imprescindibles en cualquier rutina.

- Se busca que el sistema desarrollado no comprometa la privacidad del usuario, en ningún caso se utilizarán dispositivos capaces de captar o interpretar imágenes o sonido. La red de sensores debe ser discreta, es decir, se debe buscar una colocación de los sensores de forma que se vean lo menos posible.

- Se persigue que la red sea económica y flexible, que permita la conexión de múltiples dispositivos de distintos tipos.

- La red de sensores debe proporcionar datos constantemente y éstos se deben recoger en un fichero de texto para su posterior tratamiento. Para ello se debe incluir, a parte de la medida de cada sensor, la fecha y la hora en la que se ha recogido la medida. Cada dato recogido debe identificar al sensor del que procede.

- Para la interpretación de los datos se debe diseñar un algoritmo que sea capaz de identificar la actividad que se ha realizado a partir del fichero de datos recogidos.

- Para complementar la red de sensores, se pretende diseñar un programa que haga de algoritmo de reconocimiento de actividades. A través de la información suministrada por el sistema, genere como salida el nombre de la actividad que se está realizando.

- Existen ciertas actividades para las que será conveniente contar con información adicional, una información que los sensores comerciales no pueden recoger. Es por ello por lo que, para la actividad de dormir, es necesario saber si el usuario se encuentra en el dormitorio y si está tumbado en la cama. Sin embargo, los sensores convencionales no podrían aportar la información adecuada.

En resumen, los objetivos se centrarán en desarrollar el sistema de reconocimiento de actividades, aplicando para ello la domótica doméstica y protegiendo la privacidad del usuario; y los subobjetivos consistirán en aportar información adicional al sistema para mejorar la identificación de las actividades y obtener un sistema más preciso.

1.3. Descripción de la memoria

A continuación, se va a describir cómo se organiza la memoria, explicando los contenidos de cada capítulo.

En el capítulo 2 se realizará un estudio sobre otros sistemas destinados al cuidado de personas ancianas independientes en su entorno domiciliario. Se contrastarán trabajos previos relacionados analizando aquellos que sean compatibles con los objetivos de este proyecto.

En el capítulo 3 se hablará sobre las distintas herramientas de las que se hará uso para lograr los objetivos marcados. Todos los elementos de los que se hará uso, partiendo desde los protocolos de comunicación con los que se diseña la red de sensores y acabando por los dispositivos que servirán para el tratamiento de los datos.

En el capítulo 4 se hará un recorrido sobre los elementos de software que se han utilizado para el correcto funcionamiento del sistema de reconocimiento de actividades, así como los programas diseñados o adaptados para la resolución del problema, explicando las funciones de cada elemento dentro del proyecto. Se explicará detalladamente la estructura del algoritmo utilizado para el reconocimiento de actividades.

En el capítulo 5 se analizarán los resultados de las pruebas realizadas durante la elaboración del proyecto. Se contrastarán los resultados con los objetivos planteados.

En el capítulo 6 se elaborará un estudio de los costes directos e indirectos que supone la realización del proyecto. Se estudiará la viabilidad económica del sistema para un posible caso real.

En el capítulo 7 se expondrán las conclusiones del trabajo, en las que se analizarán los resultados obtenidos, los objetivos cumplidos y se propondrán posibles vías futuras de desarrollo de este proyecto.

En la bibliografía aparecerán todos aquellos artículos, trabajos, páginas web, etc. en los que se basa este proyecto y que de alguna forma han aportado información durante su desarrollo.

En los anexos figurarán esquemas, planos y todos los programas desarrollados durante la realización de este proyecto.

2. Estado de la tecnología

Durante muchos años en España se han estudiado numerosas ayudas para promover la autonomía y apoyar a aquellas personas mayores de 65 años que viven solas. Muchas ciudades y comunidades autónomas han diseñado planes para proteger a estas personas y así se sientan menos vulnerables. También, existen empresas y asociaciones que proponen actividades y ponen todo tipo de medios a disposición de aquellos que lo necesiten, como servicios de asistencia presencial y telemática.

Los trabajos realizados hasta la actualidad se podrían dividir en dos grupos: aquellos cuyo objetivo es mejorar el bienestar y facilitar la vida diaria de las personas mayores sin considerar alertas por posibles situaciones de emergencia; y aquellos que se centran en alertar a miembros de la familia o centros de salud cercanos sobre posibles situaciones de emergencia que puedan estar teniendo lugar en el domicilio del usuario.

A continuación, se hará un recorrido sobre los distintos trabajos o estudios enfocados en mejorar la calidad de vida de las personas mayores. Se mostrarán planes de actuación y medidas para proteger la situación de estas personas, con el objetivo de hacerlas sentir menos vulnerables, fomentar su autonomía y mejorar su calidad de vida y, también, se centrará la atención en los usos de la domótica para este cometido.

2.1. Sistemas de atención a personas ancianas no dependientes

Al ser un problema cada vez más frecuente en España, muchas comunidades autónomas cuentan con estrategias de atención a personas mayores. Es el caso, por ejemplo, de Aragón [14] que, a través de los servicios sociales generales, ofrece unos servicios y prestaciones para personas por encima de 65 años, o 60 años jubiladas. El Instituto Aragonés de Servicios Sociales (IASS) ofrece a esta población numerosas prestaciones, servicios y programas adaptados a las circunstancias y necesidades de cada persona.

En primer lugar, el Servicio de información y orientación social se encarga de mantener informada a esta población de las prestaciones del Sistema Público de Servicios Sociales y de otros sistemas públicos con el mismo cometido, y de esta forma poder estudiar cada caso concreto pudiendo ofrecer un mejor servicio.

La prestación del Servicio de ayuda a domicilio (SAD) proporciona un conjunto de atenciones orientadas principalmente a facilitar un entorno de convivencia adecuado. Sin embargo, está centrado en facilitar el desarrollo de la vida diaria a personas que tienen limitaciones de autonomía personal con el fin de evitar o retrasar el ingreso en centros residenciales. En el año 2016 se atendieron a 14.360 personas en la comunidad autónoma de Aragón.

Por otro lado, el Servicio de teleasistencia básica ofrece la posibilidad de acceder inmediatamente a los servicios oportunos en situaciones de emergencia, aislamiento, inseguridad o soledad. Según un estudio, en 2016 alrededor de 15.760 usuarios solicitaron estos servicios. Por un lado, para la población general esta es una prestación complementaria, pero por otro lado se convierte en algo esencial cuando es solicitado por aquellas personas en situación de dependencia.

Entre los programas que desarrollan estos servicios se encuentra el Programa de Envejecimiento activo. La OMS define el envejecimiento activo como "el proceso por el cual se optimizan las oportunidades de bienestar físico, social y mental durante toda la vida, con el objetivo de ampliar la esperanza de vida saludable, la productividad y la calidad de vida en la vejez". Este programa ofrece distintas acciones formativas y actividades siguiendo las directrices de la OMS. Algunas de las áreas en las que desarrollan estas actividades son:

- Promoción y mantenimiento de las capacidades físicas.
- Promoción y mantenimiento de las capacidades cognitivas.
- Crecimiento personal y envejecimiento saludable.
- Promoción y uso de las nuevas tecnologías.

Esta sería una forma de velar por la salud mental y física de las personas mayores, y de que ellas mismas se sientan seguras y menos vulnerables.

Por otro lado, en el mismo plan de atención a personas mayores de Aragón se incluye el Programa de Promoción de la Autonomía Personal (PAP), el cual resulta más interesante para este proyecto. Se desarrolla por medio de los Servicios de prevención y promoción de la autonomía personal y el objetivo de este programa es el de mantener la capacidad personal de afrontar, controlar y tomar decisiones sobre cómo vivir siguiendo las preferencias personales y facilitar la ejecución de las actividades del día a día. A continuación, se citan algunos servicios que se ofrecen: habilitación y terapia ocupacional, estimulación cognitiva, atención social y formación en tecnologías de apoyo y adaptaciones del hogar.

Mediante actividades de este tipo, se pretende prevenir la situación de dependencia de las personas mayores y mejorar su calidad de vida, así como aumentar la sensación de seguridad.

Por otro lado, existe un sistema desarrollado por la empresa *Just Checking* [10] en el que se pretende monitorizar al usuario en el entorno de su casa utilizando sensores no intrusivos, es decir, sin utilizar cámaras. Sin embargo, puede añadirse el uso de micrófonos para poder configurar comandos por voz y, así, poder comunicarse con algún familiar o alertar en caso de emergencia. La red de sensores se extiende por todo el domicilio, y está compuesta principalmente por sensores de detección de presencia (PIR) y sensores de puertas y ventanas. Los sensores mandan datos a un controlador que se encarga de procesarlos y de generar mensajes que serán enviados a un familiar, avisando de cualquier tipo de información relevante como la hora en la que el usuario se ha ido a dormir, si se ha tomado la medicación, etc.

Este sistema no incluye el reconocimiento de actividades domésticas, y tampoco ese puede considerar como un sistema inteligente, ya que no puede interpretar la información recibida por los sensores para poder estudiar los patrones de comportamiento del usuario. Las situaciones de emergencia o cualquier tipo de cambio en la rutina deben ser interpretados por el familiar o responsable del cuidado del usuario.

Anteriormente se ha hecho referencia al uso de micrófonos para la detección de comandos por voz. Un trabajo llevado a cabo por Emanuele Principi [18] se centra en la asistencia de personas en su entorno domiciliario mediante la detección de señales acústicas. El sistema está basado en el reconocimiento de voz para detectar situaciones de emergencia. En él, también se detectan anomalías en el entorno acústico del domicilio para alertar al usuario y que éste pueda prevenir este tipo de situaciones.

El proyecto mencionado consta de dos modalidades de funcionamiento del sistema. En primer lugar, un Detector de Actividad por Voz (*Voice Activity Detector*) captura segmentos de voz de las señales de audio de forma que se puedan distinguir comandos concretos y poder realizar llamadas de emergencia. Por otro lado, el modo proactivo, se emplea un detector acústico para reconocer sonidos desconocidos o anormales para poder prevenir situaciones de emergencia. El sistema es capaz de comunicarse con familiares o con un centro de salud cercano si es preciso.

El principal inconveniente de este sistema es el uso de micrófonos y procesadores de sonido, que pueden ser considerados intrusivos, algo que en este proyecto se pretende evitar.

2.2. Sistemas inteligentes de reconocimiento de actividades

Cada vez son más los trabajos que hablan de implementar la tecnología domótica para desarrollar sistemas de "Casas Inteligentes" para el reconocimiento de las actividades desarrolladas por el usuario. Estos sistemas, en general, se basan en identificar patrones en la rutina del usuario para poder advertir situaciones de emergencia o prevenir problemas de salud física o mental al detectar cambios en esos patrones de comportamiento. Este tipo de sistemas son los que más se aproximan a los objetivos que persigue este proyecto.

2.2.1. Sistemas sin aviso a familiares o personal médico

En un estudio realizado en 2019 [13] se presenta un sistema, llamado *LaPlace*, que es un sistema inteligente que aprende patrones de comportamiento a través del contexto en el que se mueve el usuario. Utiliza un algoritmo online de *adaptative learning* llamado *TIMe*, creado para aprender estos patrones. Gracias a este algoritmo, se pueden observar cambios en la rutina habitual del usuario, permitiendo detectar a tiempo posibles problemas médicos de carácter físico y/o mental.

A pesar de que en este estudio se habla de la implementación de una "casa inteligente", no se especifican los distintos sensores que se utilizan, por lo que no se descarta el uso de sensores intrusivos. Por otro lado, tampoco se genera ninguna acción que lleve a cabo el sistema para proteger al usuario ante alguna posible emergencia, por ejemplo, alertando a la familia o a algún centro de salud cercano.

En otro estudio [17], se habla de un sistema de recordatorios que se adaptan y personalizan en función del usuario. Los sistemas de este tipo están basados en una red de sensores distribuidos en sitios concretos de la casa y un procesamiento de los datos recogidos por éstos se lleva a cabo mediante herramientas de Inteligencia Artificial. Mediante el desarrollo de un sistema de reconocimiento de actividades que permita identificar la actividad que está siendo realizada por el usuario, y así poder crear recordatorios o mensajes que puedan ser de utilidad para el usuario a la hora de realizar la tarea que está realizando o que vaya a realizar a continuación.

Este tipo de sistemas supone una ayuda para el paciente, simplificando la planificación de su rutina. Sin embargo, la mayor desventaja que supone es la ausencia de alertas que advierten si las actividades se están realizando de manera adecuada, o si se han visto interrumpidas por algún problema que haya surgido, que pueda suponer un riesgo para el usuario y que pueda necesitar ayuda de algún miembro de la familia o incluso atención médica.

2.2.2. Sistemas con aviso a familiares o personal médico

Algunos de estos estudios se centran en tratar de detectar situaciones concretas de peligro que se puedan dar en el entorno domiciliario del usuario. Si el sistema está entrenado para detectar este tipo de situaciones y es capaz de alertar al personal médico o social que proceda, lo que podría permitir una actuación rápida y poder proteger la vida del usuario.

En un artículo publicado por el IEEE [16], se propone un sistema capaz de detectar caídas en tiempo real a partir de un dispositivo portable: una pulsera de actividad en el tobillo del usuario. Las caídas en personas mayores pueden provocar lesiones y efectos indeseables a nivel físico y psicológico, por eso es necesario atender a estas personas cuanto antes, y así prevenir consecuencias peores en su estado de salud. El dispositivo incluye acelerómetro, giroscopio y magnetómetro que permite obtener la orientación tridimensional en tiempo real, de forma que se puede detectar una caída a partir de un cambio brusco de esta orientación y gracias al algoritmo desarrollado en este estudio.

Como no es posible prever la naturaleza de la caída y considerando los casos más extremos en los que el usuario queda inconsciente, se integra un módulo Bluetooth que se comunica con el teléfono móvil del usuario. En caso de caída, se genera una señal, a partir de una aplicación afín al estudio, que alerta a familiares y a cualquier teléfono de contacto que el usuario haya configurado previamente como contactos de emergencia.

Del mismo tipo es un estudio llevado a cabo en 2012 [1] cuyo objetivo es detectar situaciones de caídas accidentales mediante una red de sensores situados en lugares concretos en el domicilio del usuario. Este sistema también incluye un procesamiento de datos por Inteligencia Artificial, permitiendo detectar cambios en los patrones de comportamiento del usuario. Los resultados son menos acertados si el usuario tiene mascota o cuando recibe visitas en su casa, pudiendo detectar falsas caídas.

3. Métodos y herramientas

3.1. ZigBee

En este trabajo, se utilizarán sensores comerciales inalámbricos que se comunicarán con un controlador, al que irán enviando datos cada vez que se activen. Las condiciones de activación dependerán del tipo de sensor y la información que vaya a transmitir. Existen numerosos protocolos de comunicación a partir de los cuales se puede establecer las conexiones entre ellos, sin embargo, en el mundo de la domótica doméstica es muy frecuente que los dispositivos se comuniquen mediante ZigBee.

El origen de este conjunto de protocolos viene de la llamada "ZigBee Alliance" [8], creada en 1997 por 8 empresas, y cuyo objetivo era el de permitir una red de comunicaciones que sea fiable, de bajo consumo energético, económica e inalámbrica para poder controlar y monitorizar productos, y basada en un estándar global. Actualmente, la ZigBee Alliance cuenta con más de 200 empresas, muchas de ellas se encuentran en la cabeza de sus respectivos mercados, se pueden destacar empresas como Motorola, Schneider Electric, Siemens, Philips o Texas Instruments,

ZigBee es una tecnología que consiste en un conjunto de protocolos de comunicación inalámbrica de corto alcance y bajo consumo cuyo uso ha ido aumentando en los últimos años en el campo de la domótica en general y la automatización industrial. Está basada en el estándar IEEE 802.15.4 [15] (junio de 2007) de redes inalámbricas de área personal (*WPAN*).

En condiciones normales, es frecuente que en este protocolo se gestione el control de acceso a los recursos mediante CSMA/CA (*Carrier Sense Multiple Access with Collision Avoidance*), siendo posible también usar TDMA (*Time Division Multiple Access*), pero esta opción sólo es recomendable para aplicaciones de baja latencia.

La principal ventaja que presenta esta tecnología (que es, a su vez, la razón por la que su uso es cada vez más extendido) es el bajo consumo energético que supone, permitiendo que los dispositivos que se comunican según este protocolo puedan aumentar su autonomía, pudiendo aguantar varios años sin necesidad de cambiar las baterías. Como inconveniente, la velocidad de transmisión de datos alcanzada no es muy alta.

De cara al ahorro energético, estos sensores permanecen la mayor parte del tiempo en un estado latente, esto es, dormidos hasta que se produce una activación. En la Figura 3 se muestra el espectro de ocupación en las bandas del protocolo IEEE 802.

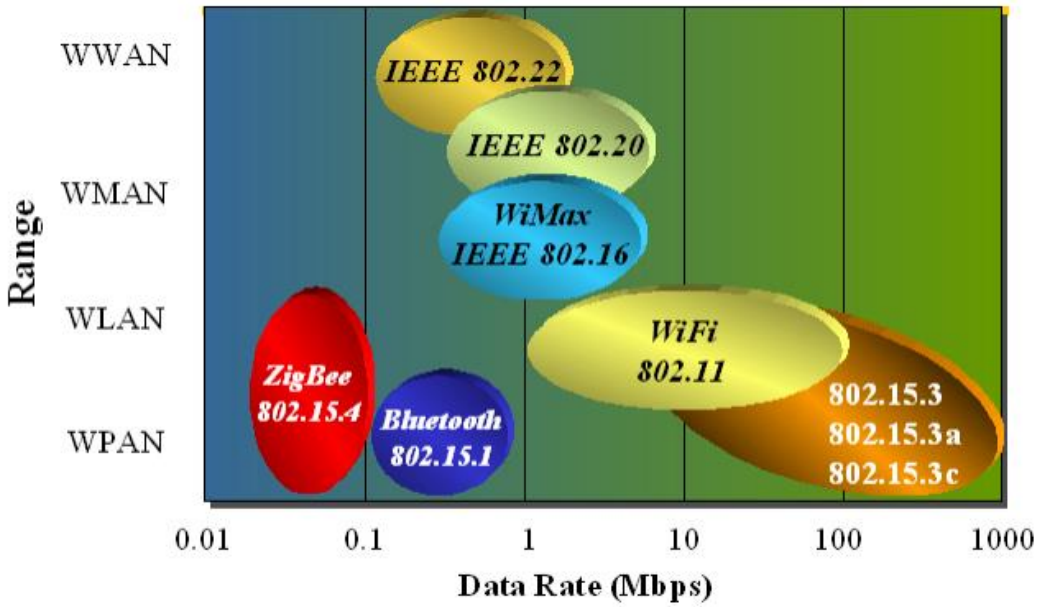


Figura 3. Espectro de ocupación. [14]

En cuanto a la topología de las redes, ésta puede ser variada. En una red ZigBee puede haber hasta 254 nodos que pueden disponerse en estrella, maya, o en grupos de árboles. Cabe destacar la robustez de esta tecnología ya que, en caso de caída de alguno de los nodos, la red trata de encontrar un camino alternativo por el que enrutar el mensaje de forma que llegue a su destino correctamente. Es necesario tener en cuenta que existen tres tipos de entidades en ZigBee (Figura 4):

- Coordinador: es un dispositivo único por cada red. Es el dispositivo con el que se inicia la formación de la red, y el que se conecta a la red Internet.
- Router: se encarga del enrutamiento de los mensajes. Puede conectarse al coordinador de la red, o a otro router ZigBee. Permiten ampliar el rango de la red.
- Dispositivo final: sensor o actuador. Es el objetivo final de la comunicación, no realiza tareas de enrutamiento.

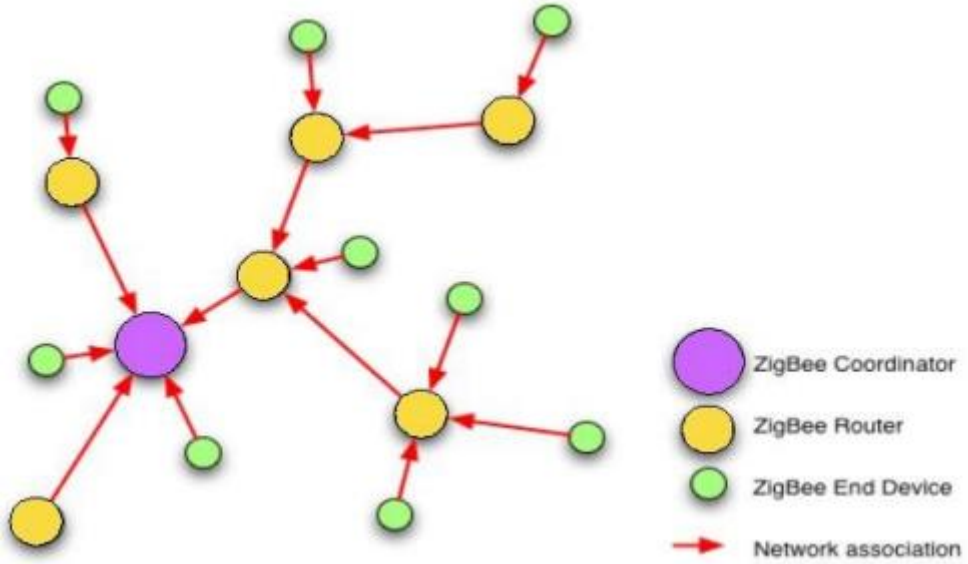


Figura 4. Ejemplo de red ZigBee. [14] p.6

Uno de los principales usos de este protocolo es el de la automatización y la domótica doméstica. La razón por la que es tan utilizado en este campo es debido a que permite conectar muchos dispositivos de manera inalámbrica y de tal forma que tan solo uno de los nodos, el coordinador, esté conectado a Internet. Esto supone muchas ventajas respecto a los sensores conectados mediante Wifi, ya que evita problemas con la saturación del router Internet cuando se están conectando demasiados dispositivos, ya que cada uno de ellos ocupa una dirección IP, por lo que es más probable que se produzcan colisiones durante el acceso al medio derivando en problemas de comunicación.

El coordinador es el elemento central de la red ZigBee, a la que se conectan todos los dispositivos finales (y el router ZigBee en algún caso). Este dispositivo es el que se encarga de comunicarse con los distintos dispositivos finales, conectando toda la red a Internet y ocupando tan solo una dirección IP.

3.2. Protocolo MQTT

El protocolo MQTT (*Message Queue Telemetry Transport*), es un protocolo de código abierto desarrollado y optimizado por IBM para la comunicación M2M (*Machine To Machine*) utilizando el modelo publicador-suscriptor [3].

En la comunicación intervienen tres entidades: publicador, bróker y suscriptor (Figura 5):

- En primer lugar, uno o varios publicadores publican un mensaje dentro de un *topic*. Un *topic* es la categoría del mensaje que se está enviando, es decir, un tema o un asunto. Los suscriptores serán aquellos dispositivos que se suscriben a un determinado *topic* y reciben todos los mensajes que se publican en él.

- El *bróker* es un elemento que actúa de intermediario, todos los mensajes enviados por los publicadores se publican con un *topic* asociado en un *bróker*. El *bróker* se encarga de filtrar los mensajes según su *topic*, para después redireccionarlos a los suscriptores correspondientes. Por lo general, los *bróker* no almacenan los mensajes que reciben. Las conexiones con el bróker se realizan mediante el protocolo TCP/IP. Mosquitto es uno de los *bróker* más conocidos.

- Los suscriptores no tienen direcciones fijas asociadas, por lo tanto, los mensajes no se envían directamente a ellos ya que no hay una conexión directa. Un cliente se debe suscribir a uno a varios *topic* dentro del mismo *bróker* en el que se publican los mensajes para poder recibirlos.

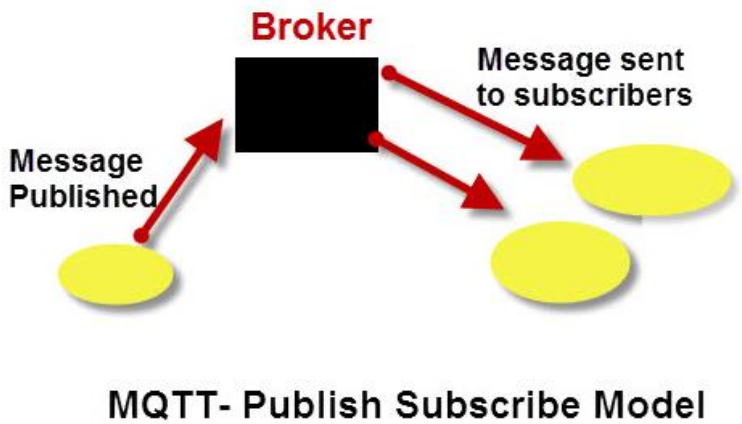


Figura 5. Modelo publicador suscriptor MQTT. [3]

La razón de que el uso de este protocolo sea tan extendido en el mundo del IoT es que el envío de datos requiere poco ancho de banda, pocos recursos computacionales y tienen un bajo consumo. La topología de este tipo de redes es siempre de tipo estrella, en la que el nodo central es el bróker al que se podrán conectar un gran número de clientes. Cada cliente debe tener un nombre o ID único para poder publicar o suscribirse a un *topic*. Existen librerías que permiten crear clientes en programas informáticos para poder publicar o suscribirse a un *topic* utilizando MQTT.

La estructura de los *topic* en MQTT es jerárquica, parecida a la que se utiliza con los directorios de carpetas y ficheros. Puede existir un *topic* padre, del que partan varios *topic* hijos, algunos de esos *topic* hijos podrán ser a su vez padres de otros *topic* (Figura 6). Los *topic* son datos de tipo cadena de caracteres, y deben tener un tamaño mínimo de un caracter. Para referirse a un *topic*, es necesario nombrar todos los padres de ese *topic* separados por el caracter "/". Algunos posibles ejemplos de posibles *topic* siguiendo el esquema de la Figura 6 son:

- "*casa/habitaciones/hab1/presencia*"
- "*casa/habitaciones/hab1/+*" (se utiliza para suscribirse a todos los *topic* hijos de hab1)
- "*casa/habitaciones/#*" (se utiliza para suscribirse a todos los *topic* descendientes de "habitaciones")
- "#" (se utiliza para suscribirse a todos los *topic*)

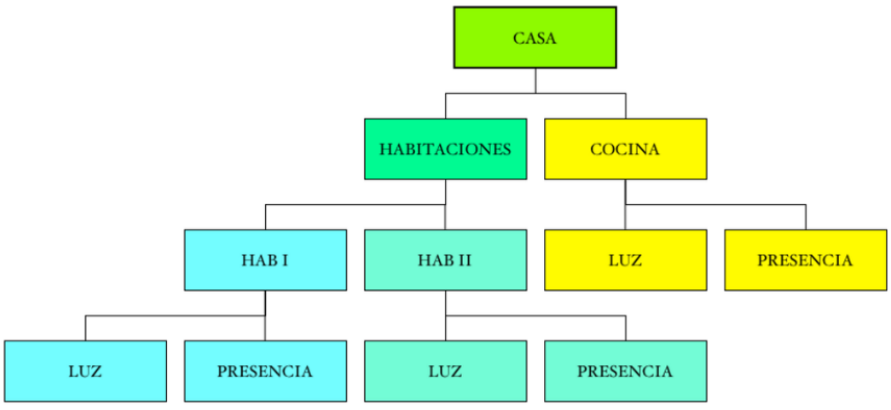


Figura 6. Ejemplo de estructura de *topic*

3.3. Sniffer CC2531

La mayoría de los sensores comerciales que se comunican usando ZigBee tienen una limitación, y es que cada empresa que fabrica los sensores tiene un coordinador determinado para controlar todos los sensores de su marca. Es decir, que para que una red de sensores comerciales funcione correctamente es necesario que todos los elementos pertenecientes a la red sean del mismo fabricante. Esto implica que los sensores pertenecientes a la red deben encontrarse dentro del catálogo de venta de la empresa que produce el coordinador, y en caso de que se quiera añadir un sensor con distintas características o de un tipo que no esté disponible en el catálogo de la marca del coordinador, sería necesario comprar un segundo controlador de la marca del sensor que se quiere añadir.

Existe una alternativa con la cual poder coordinar un gran número de sensores de distintas marcas con un único dispositivo: un *Network Sniffer* (en adelante, *sniffer*). Un *sniffer* es un dispositivo que captura las tramas de la red a la que se conecta, lo que significa que un dispositivo de estos dentro de una red ZigBee como la que se pretende desplegar en este trabajo, sería capaz de capturar todos los mensajes enviados por los sensores para poder realizar un tratamiento posterior de esos mensajes.

Estos dispositivos presentan algunas ventajas con respecto a utilizar un coordinador comercial para la red:

- No tiene límite de dispositivos conectados, ya que simplemente se encarga de capturar las tramas que lanzan los sensores. Sin embargo, un exceso de tráfico podría dar problemas en la comunicación.

- Por otro lado, no tiene limitación de conexión en función del fabricante de los sensores. En otras palabras, da igual si los sensores son de *Xiaomi* o de *Philips*, siempre que funcionen con el protocolo ZigBee. En primera instancia, todos los sensores seleccionados pertenecen a Xiaomi por lo que con el coordinador de *Aqara* no debería haber problema. Sin embargo, de cara a una ampliación de la gama de sensores para poder detectar mayor número de actividades, la opción de utilizar un *sniffer* ZigBee es la más adecuada.

Al ser su uso tan extendido, existen muchas empresas que ofrecen distintos modelos y que se pueden adquirir a un precio asequible. En la Figura 7 (izquierda) se muestra el *sniffer* CC2531 cuyo vendedor es la empresa Texas Instruments, y que será el que se utilizará en este proyecto para capturar las tramas que envían los sensores y, de esta forma, poder garantizar la flexibilidad del sistema, pudiendo añadir sensores de distintas marcas utilizando un único coordinador.

Figura 7. Izquierda: USB sniffer CC2531 de Texas Instruments. Derecha: CC Debugger

El CC2531 es un dispositivo diseñado y distribuido por la empresa Texas Instruments. El límite de dispositivos que se pueden conectar al sistema está entorno a los 40 dispositivos, sin embargo, este rango se puede ampliar utilizando otro CC2531 o un CC2540 (otro modelo de *sniffer* de la misma marca) de forma que se pueda utilizar a modo de router ZigBee. Esta solución se puede adaptar también en el caso de que la señal de los sensores sea débil y sea necesario aumentar el rango de las señales.

Para poder utilizar este dispositivo, en primer lugar, es necesario programarlo e instalar el firmware de Texas Instruments que permitirá que el *sniffer* funcione correctamente. Esta acción no se podría llevar a cabo sin el "*CC Debugger*" (Figura 7 derecha), que no es más que un depurador necesario para la instalación del firmware. Es frecuente que tanto el *sniffer* como el depurador se puedan adquirir juntos, pero un mismo depurador bastaría para instalar el firmware en tantos *sniffer* como se desee (sin que éstos deban ser concretamente del mismo modelo CC2531, también sirve para otros modelos de *sniffer* de Texas Instruments).

Una vez instalado el firmware, el dispositivo ya está listo para ser usado. Para poner en funcionamiento el *sniffer*, se debe conectar a uno de los puertos USB de la Raspberry Pi y terminar la configuración a partir de este elemento que se explicará en el apartado 3.4.

3.4. Raspberry Pi

Una Raspberry Pi es un ordenador de dimensiones reducidas y bajo precio creada en 2012 por la "Raspberry Pi Foundation", en Reino Unido. Es frecuente instalar en la Raspberry Pi el sistema operativo Raspbian, que es un sistema operativo de tipo Linux que permite explorar el mundo de la computación y desarrollar programas informáticos en lenguajes como, por ejemplo, Python que es con lo que se va a programar el algoritmo que se va a diseñar en este proyecto.

Figura 8. Raspberry Pi 3 Modelo B+.

Su uso es muy extendido en el IoT debido a su versatilidad y a las innumerables posibilidades que ofrece para crear proyectos de todo tipo, entre ellos, proyectos de domótica. Al igual que en cualquier ordenador, es capaz de realizar tareas como navegar en internet, reproducir vídeos, reproducir juegos,

entre otras muchas cosas. Existen varios modelos, pero en este proyecto se utilizará la Raspberry Pi 3 B+ (Figura 8).

Este modelo cuenta con 4 puertos USB, en uno de los cuales irá conectado el CC2531 para capturar los mensajes de los sensores. Hay dos opciones para poder acceder al escritorio de Raspbian: se podría conectar un monitor a través del puerto HDMI, y usar dos de los puertos USB para conectar un ratón y un teclado para controlarlo; y la otra opción sería conectarse desde otro ordenador a través de un escritorio remoto, para lo que será necesario conocer la dirección IP de la Raspberry Pi, es decir, la dirección del router a la que se asocia cuando se conecta a Internet a través de Wifi o mediante conexión Ethernet.

Otras características de este modelo que pueden ser de interés para el proyecto son que la CPU tiene 4 núcleos, 1GB de memoria RAM y dispone tanto de Wifi como de BLE (Bluetooth de baja energía). La Raspberry Pi utiliza una fuente de alimentación micro USB de 5 V.

El papel de este dispositivo dentro del proyecto será el de coordinar las señales que reciba el CC2531 de los sensores y, a través de un programa de Python que se diseñará, recoger esos datos de forma ordenada en un fichero de texto, siguiendo un formato determinado, que se utilizará posteriormente para el tratamiento e interpretación de los datos mediante algoritmo que se diseñará para poder identificar las actividades que realice el usuario en cada momento.

Como se ha mencionado anteriormente, los sensores comerciales que se utilizarán en este trabajo se comunican con el CC2531 usando el protocolo ZigBee. Sin embargo, la Raspberry Pi utilizará el protocolo MQTT para el procesamiento de los mensajes, lo que implica que será necesario instalar un bróker MQTT, como se ha mencionado en el apartado 3.2, para llevar a cabo la comunicación. El bróker que se utilizará para este trabajo será el Mosquitto, debido a la versatilidad y a la popularidad que lo hacen uno de los bróker más utilizados en el mundo de la domótica.

Por otro lado, el hecho de que se utilicen dos protocolos distintos para la adquisición de datos implica la necesidad de traducir la información de ZigBee, que es la forma en la que el CC2531 captura la información de los sensores, a MQTT, que es el protocolo que utilizará la Raspberry Pi para el tratamiento de esa información. Para esta tarea, existe un software de código abierto que permite realizar esta traducción entre los dos protocolos, que se explicará en el apartado 3.5: ZigBee2MQTT.

3.5. ZigBee2MQTT

Zigbee2MQTT es un software de código abierto desarrollado por Koen Kanters [12] para poder controlar múltiples dispositivos ZigBee de distintos fabricantes vía MQTT.

La arquitectura interna está compuesta por tres módulos:

- En primer lugar, el módulo "*zigbee-herdsman*" parte del adaptador que captura la señal ZigBee (CC2531 en el caso de este proyecto) y crea un Interfaz de Programación de Aplicaciones (API) para los niveles superiores del protocolo. En el caso de CC2531, este módulo utilizaría el "*TI zStack monitoring and test API*" desarrollado por Texas Instruments para comunicarse con el adaptador y manejar la comunicación central ZigBee.

- El siguiente módulo es el "*zigbee-herdsman-converters*", que se encarga de mapear las señales de cada dispositivo a los clúster de ZigBee admitidos por este software. Los clúster ZigBee son las capas superiores del protocolo ZigBee que definen la comunicación de los sensores y otros dispositivos dentro de la red.

- El último módulo es el Zigbee2MQTT, que se encarga del mapeo de los mensajes capturados por el CC2531 en el protocolo ZigBee a los mensajes en el protocolo MQTT.

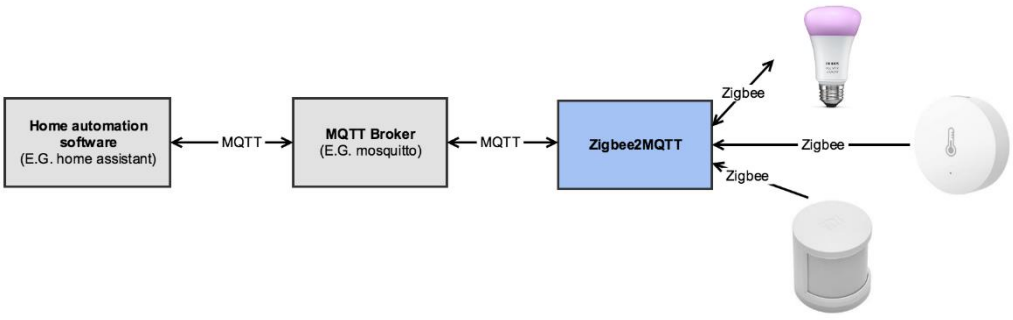


Figura 9. Diagrama de bloques de un sistema domótico utilizando zigbee2mqtt. [12]

Una vez convertidos los mensajes al protocolo MQTT, se utilizará el Mosquitto como bróker para publicar los mensajes con la información proporcionada por cada sensor y el sistema estará preparado para que un cliente se suscriba a los *topic* y comenzar el tratamiento de los datos. El diagrama de bloques de la Figura 9, muestra cómo es la comunicación de cada etapa de la automatización o domotización de una casa inteligente utilizando *zigbee2mqtt*.

Para poder suscribirse a los mensajes publicados por el Zigbee2MQTT, será necesario conocer los *topic* más relevantes para su posterior tratamiento. Existen dos *topic* que serán de mayor interés para este trabajo: aquél que indica al sistema que se ha añadido un nuevo sensor a la red y aquél que publica toda la información proporcionada por cada sensor. Respectivamente, estos *topic* son:

- "*zigbee2mqtt/bridge/log*": en el mensaje la información de interés es "*interview_successful*" que indica que la conexión se ha realizado convenientemente y que el sensor se ha integrado a la red; "*friendly_name*" seguido de un número único de 16 dígitos en formato hexadecimal que es identificador de cada sensor conectado; "*model*" muestra el nombre del sensor del que se trata, es el nombre que le da el fabricante.

- "*zigbee2mqtt/+*": el *topic* padre es "zigbee2mqtt" y todos los hijos de éste se nombran con los identificadores hexadecimales que se mencionaron anteriormente, por lo tanto, se puede saber fácilmente de qué sensor se está obteniendo el mensaje. Los datos de interés de las medidas de los sensores dependerán del tipo de sensor que envíe el mensaje, es por eso por lo que es necesario conocer el identificador y el tipo de cada uno de los dispositivos que se encuentran en la red.

3.6. Sensores comerciales

A partir de las actividades que, según los objetivos, se quieren identificar en este trabajo, se pretende concretar qué sensores serán necesarios para la toma de datos. En la tabla 2 se muestran las actividades clasificadas en función de la parte de la casa en la que se realizan y, para cada actividad, los sensores que se necesitarán y la ubicación de éstos, siguiendo la distribución de una casa genérica.

LUGAR DE LA CASA	ACTIVIDAD	SENSORES	UBICACIÓN
Dormitorio	Dormir	PIR, luz	techo, paredes
		sensor de presión	cama
	Despertarse	PIR, luz	techo, paredes
		sensor de presión	cama
	Vestirse	PIR	techo
		contacto, vibración	cajones, armario
Baño	Ir al baño	luz	paredes
		contacto	puerta
	Ducharse	luz	paredes
		contacto	puerta
		temperatura	paredes
Cocina	Cocinar	contacto	puerta
		temperatura	campana de extracción
Sala de estar	Ver la televisión	sensor de presión	sofá/sillón
		consumo	televisión
Entrada	Volver a casa	contacto	puerta de entrada
		PIR	techo
	Salir de casa	contacto	puerta de entrada
		PIR	techo
Despacho	Trabajar con el ordenador	PIR	techo
		vibración	silla de trabajo

Tabla 2. Descripción de tareas y sensores asociados

Por lo tanto, los tipos de sensores que se van a necesitar para este proyecto son:

- Sensores de detección de presencia (PIR): "Aqara human body movement and illuminance sensor" (Figura 10.a).
- Sensores de intensidad luminosa: "MiJia Light Intensity Sensor" (Figura 10.b).
- Sensores de puertas y ventanas: "Aqara Door and Window Contact Sensor" (Figura 10.c).
- Sensores de vibración: "Aqara Smart Vibration Sensor" (Figura 10.d).

- Sensores de temperatura y humedad: “Aqara temperature, humidity and pressure sensor” (Figura 10.e).
- Sensores de consumo: “Xiaomi zigbee smart socket” (Figura 10.f).

a. Sensor PIR

b. Sensor de luz

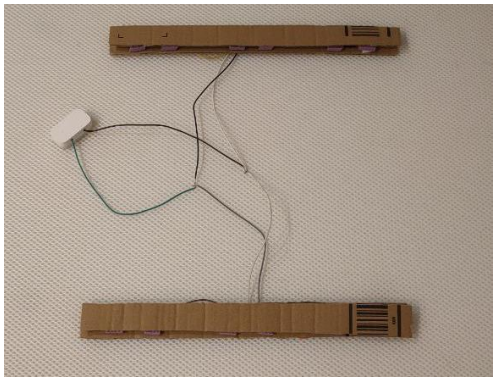
c. Sensor de contacto

d. Sensor de vibración

e. Sensor de temperatura y humedad

f. Sensor de consumo

g. Botón Zigbee



h. Sensor de presión

Figura 10. Sensores utilizados en el proyecto

Además de estos sensores, para la etapa de etiquetado, se necesitarán unos botones para que, durante las pruebas, el usuario los pulse antes de realizar una actividad y así poder determinar el grado de fiabilidad del sistema de reconocimiento de actividades:

- Botones: “Aqara wirless switch” (Figura 10.g).

Por otro lado, para mejorar la precisión de la detección de actividades como dormir y despertarse, se va a modificar uno de los sensores de contacto “Aqara Door and Window Contact Sensor” de forma que se pueda utilizar como un sensor de presión para detectar presencia en una cama.

- Sensor de presión: “Aqara Door and Window Contact Sensor” modificado (Figura 10.h).

La elección de estos sensores se ha hecho teniendo en cuenta los objetivos propuestos en el apartado 1.2, entre los que se encuentra priorizar la privacidad del usuario evitando cámaras y micrófonos. También, los sensores escogidos para este estudio garantizan una autonomía de más de un año y todos se comunican utilizando ZigBee. Además, se ha procurado elegir los sensores de forma que el sistema final no tenga un precio excesivamente elevado.

4. Desarrollo del sistema de reconocimiento de actividades

Una vez explicadas las herramientas que serán necesarias para este trabajo, se explicará cómo será el método que se va a seguir para desarrollar el sistema de reconocimiento de actividades domésticas. El método comienza con la fase de etiquetado tras un estudio primario de los datos tomados las primeras semanas y, a medida que avance el estudio, se irán puliendo las premisas para ir mejorando la precisión del sistema.

En primer lugar, la fase de etiquetado consistirá en desplegar la red de sensores en el domicilio de un usuario del que se estudiará su rutina diaria y realizar una toma de datos primaria durante las primeras semanas. Posteriormente, con la red activa, se analizarán esos datos con el fin de estudiar la rutina que sigue el usuario y así poder concretar las premisas que indicarán la realización de cada actividad. Con estas premisas se diseñará el algoritmo de reconocimiento de actividades del que se partirá para continuar el estudio de la segunda fase.

La siguiente fase será la del estudio de fiabilidad, en la que se determinará la precisión con la que el sistema es capaz de reconocer las actividades propuestas. En esta fase, el sistema estará en continuo funcionamiento, pues el objetivo de esta fase es la de mejorar el sistema en base a las necesidades del usuario. Esta fase consiste en un proceso que se puede repetir tantas veces como se desee. El proceso comenzará comprobando la fiabilidad del sistema, y añadiendo las modificaciones al algoritmo que se estimen necesarias, con cada modificación, se comprobarán los resultados obtenidos por el sistema. De esta forma se podrá ir mejorando la identificación de las actividades en base a la rutina del usuario.

En la Figura 11 se puede observar el diagrama UML de actividad que describe el método que se ha explicado. En cada una de las fases, se pueden ver las etapas que se van a llevar a cabo durante el desarrollo de este proyecto.

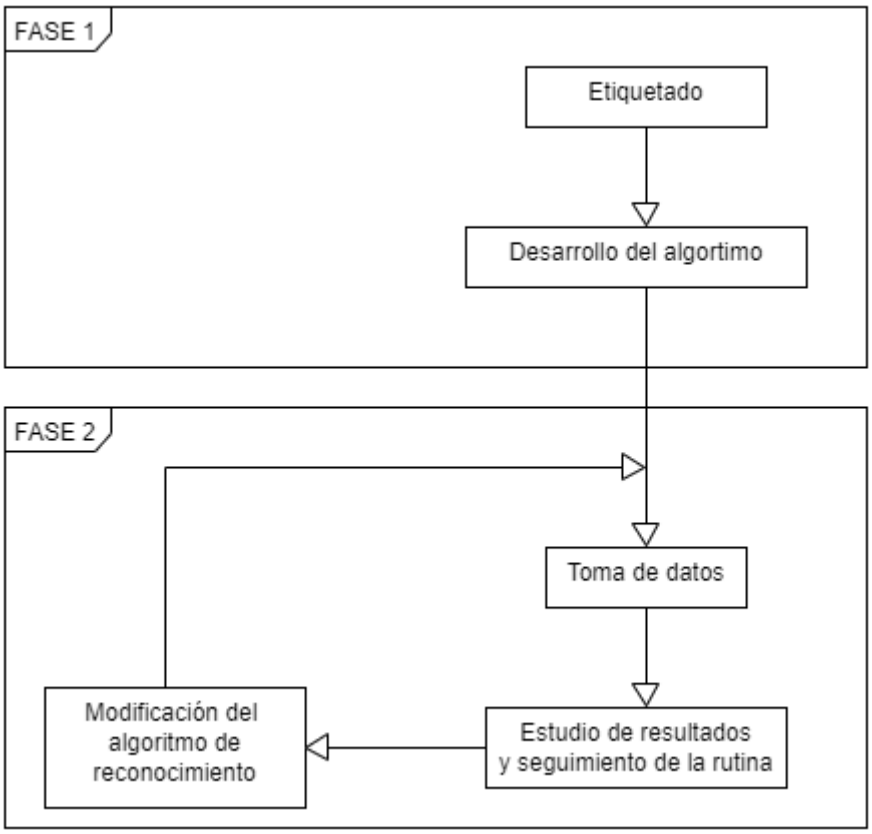


Figura 11. Diagrama de actividad del método utilizado para el desarrollo del proyecto

4.1. Hardware

El primer paso para el desarrollo de este proyecto es la instalación y conexión de la red de sensores y la Raspberry Pi para comenzar la toma de datos. En primer lugar, se debe tener en cuenta que la instalación y emplazamiento de la Raspberry Pi es determinante, ya que todos los sensores deben estar dentro de su rango. Por ello, para el despliegue de la red, se va a colocar este dispositivo en un punto central del domicilio. Durante todas las etapas del proyecto, el CC2531 debe ir conectado a uno de los puertos de la Raspberry Pi, de lo contrario, no se podrán recibir los mensajes correctamente.

En la Tabla 3 se puede observar cómo se ha llevado a cabo la distribución de los sensores en un domicilio para la toma de datos. Los sensores son adhesivos y fáciles de instalar. Para crear la red tan solo habrá que ejecutar el programa de la Raspberry Pi, que se suscribirá a los *topic* deseados y, una vez esté el programa a la espera de la publicación de nuevos mensajes, habrá que mantener pulsado el botón de encendido de cada sensor durante un par de segundos.

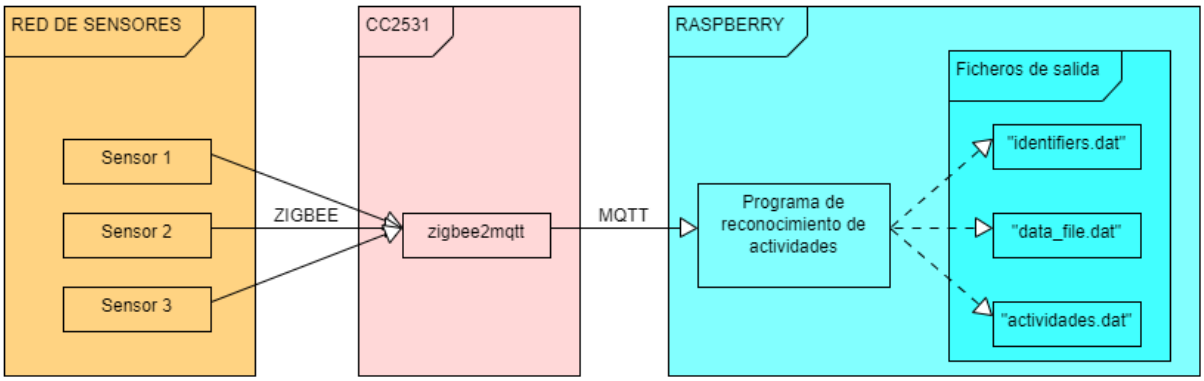


Figura 12. Diagrama UML de componentes de la red de sensores

En la Figura 12, se puede observar el diagrama de bloques que resume el funcionamiento de la red de sensores. En él se reflejan las tres principales etapas que van a recorrer los mensajes desde el punto de partida, que sería la red de sensores, donde se genera toda la información y se transmitirá usando ZigBee hasta el *sniffer* CC2531. Este dispositivo estará conectado a la Raspberry Pi y se encargará de capturar los mensajes ZigBee y publicarlos en el bróker de Mosquitto instalado en la Raspberry Pi siguiendo el protocolo MQTT. En la última etapa, el programa de reconocimiento de actividades diseñado se encarga del tratamiento de los datos y genera los tres ficheros de salida que se muestran.

El funcionamiento del programa de Python, que se explicará más adelante, se encargará de recoger la información que cada uno de los sensores genera en el momento de su conexión a la red y los introducirá en un fichero de texto "*identifiers.dat*". El programa generará un código único para cada sensor con el fin de facilitar la identificación del tipo y el sensor que manda la información. Si se desea cambiar el código que se asigna automáticamente a los sensores, tan solo habría que abrir el fichero de texto y cambiarlos manualmente, el programa seguirá identificando al mismo sensor, pero se referirá a él de manera distinta, algo que no afecta en absoluto al correcto funcionamiento del sistema.

En el caso de este proyecto, se ha asignado un valor numérico a cada estancia de la casa en la que hay instalado al menos un sensor, de forma que el índice de cada código coincida con el número asignado a la parte de la casa en la que se encuentra (ver Tabla 3):

- Entrada = 1
- Dormitorio = 2
- Baño = 3
- Cocina = 4
- Sala de estar/salón = 5
- Escritorio/despacho = 6

ESTANCIA DE LA CASA	SENSORES	CÓDIGO
Entrada	Contacto	C1
Dormitorio	Contacto	C2
	Vibración	V2
	Presencia en cama	B2
	PIR	M2
	Luz	L2
Baño	Contacto	C3
	Temperatura	T3
	Humedad	H3
	Luz	L3
Cocina	Contacto	C4
	Temperatura	T4
	Humedad	H4
	Vibración	V4
Sala de estar	Consumo	P5
Despacho	Vibración	V6
	PIR	M6

Tabla 3. Distribución de la red de sensores

La distribución de los botones ZigBee será de uno por cada estancia en la que esté instalado, al menos, un sensor. De esta forma, el usuario podrá marcar qué actividad está realizando en cada momento para poder relacionar los datos con la activación de una determinada actividad.

4.2. Software

Una vez desplegada la red de sensores dentro del domicilio del usuario, es necesario conectar cada uno de los elementos a la Raspberry Pi para poder recolectar los datos en un fichero de texto para su posterior tratamiento. Para ello, se ha desarrollado un programa de Python. Se ha utilizado Python porque es un lenguaje multiparadigma muy versátil, interpretado de código muy legible y ampliamente utilizado en aplicaciones de IoT.

Como se ha mencionado anteriormente, los sensores se comunican utilizando ZigBee, de forma que el *sniffer* captura los mensajes que contienen toda la información que se intercambian los sensores. A continuación, gracias al software zigbee2mqtt, la Raspberry se conecta al bróker de Mosquitto que se ha instalado en este dispositivo. Zigbee2mqtt traduce la información que contienen los mensajes y hace la función de publicador, utilizando el bróker de Mosquitto. La información intercambiada depende del tipo de sensor que la envía, por ello, es imprescindible poder identificar a los sensores a partir del mensaje que envían.

Zigbee2mqtt es un sistema muy complejo que utiliza diferentes *topic* para publicar información de distintos tipos. Para este trabajo sólo serán necesarios dos tipos de *topic*: aquéllos que anuncian que la inclusión de un nuevo sensor a la red se ha realizado de manera satisfactoria, los cuáles solo son enviados una vez por cada sensor nuevo en la red; y aquéllos que contienen la información proporcionada por los sensores.

El principal problema ha sido el de poder identificar los sensores y poder clasificarlos según su tipo. A continuación, se muestra un ejemplo de los mensajes que se publican cada vez que se añade un sensor de tipo PIR a la red:

- Topic: '*zigbee2mqtt/bridge/log*'
- *{"message":"interview_successful","meta":{"description":"Aqara human body movement and illuminance sensor", "friendly_name":"0x00158d00069d78ef","model":"RTCGQ11LM", "supported":true,"vendor":"Xiaomi"},"type":"pairing"}*

El *topic* es independiente del sensor que se comunique. En cuanto al mensaje, se puede observar que aparece información que es de especial interés para la identificación. En concreto, son tres los campos más relevantes:

- El campo '*message*': viene seguido de un mensaje que informa del estado de la conexión en el momento en el que se añade el sensor.

Cuando la conexión se ha realizado satisfactoriamente, el mensaje será el de: "*interview_successful*", en cambio si algo ha ido mal el mensaje será el de "*interview_failed*", y se tendrá que volver a intentar la conexión.

- El campo '*description'*: en este campo aparece el nombre comercial del sensor que se está tratando de añadir. Este nombre es uno de los que se mencionan en el apartado 3.6, correspondiente a cada sensor.

- El campo '*friendly_name*': este campo contiene el identificador de un sensor, que es un dato único para cada sensor. Consiste en un número de 16 cifras en formato hexadecimal. No existen dos sensores con el mismo identificador, con lo que se puede utilizar para identificar un sensor a partir del mensaje que captura la Raspberry, pues los *topic* donde se publican los mensajes que contienen las medidas de los sensores incluye este identificador.

El programa que se ha diseñado utiliza estos tres campos para poder registrar en un fichero de texto todos los sensores que integran la red. Para cada sensor se registrará la fecha y la hora en la que se incorporaron a la red, seguido del identificador y, en la última columna, un código compuesto por una letra, que dependerá del tipo de sensor que sea, y un número para distinguir los sensores del mismo tipo. Los sensores de luz se identificarán con la letra "L", los de contacto con la letra "C", los de movimiento (PIR) con la "M", los de vibración con la "V", los de consumo con la "P", los botones con la "S" y, por último, los de temperatura y humedad se registrarán como dos sensores distintos, pero con el mismo identificador, las medidas de temperatura aparecerán como procedentes de un sensor identificado con la letra "T", y las medidas de humedad con la identificación "H".

El fichero será guardado como '*identifiers.dat*'. La primera acción que realiza el programa antes de empezar a capturar los mensajes es leer este fichero para comprobar si existen sensores dentro de la red, por lo que será posible modificar este fichero. Por ejemplo, se podría añadir una columna al final que indique en qué sala se encuentra cada sensor (baño, entrada, dormitorio…). También será posible modificar los códigos de los sensores, siempre y cuando el campo del identificador permanezca intacto, pues este es el dato que utilizará el programa para identificar al sensor del que procede la información.

4.2.1. Funcionamiento del programa

En la Figura 13 se puede ver un diagrama de actividades que ilustra el tratamiento que va a realizar el programa con los mensajes recibidos de los sensores. Este tratamiento comienza con la llegada de un nuevo mensaje que puede ser de dos tipos: el primero indica que un nuevo sensor se ha añadido a la red y el segundo es un mensaje que contiene la medida recogida por un sensor que ya pertenece a la red. En cada caso, el tratamiento de la información es distinto. Para saber de qué tipo es cada mensaje, se pueden discriminar según su *topic*, como se ha comentado anteriormente.

El programa trabaja con varias listas: un conjunto de listas donde aparecen todos los identificadores para cada tipo de sensor (switch_list, light_list...) y otro conjunto de listas donde aparecen en los índices de los códigos de cada sensor en el mismo orden en que aparecen en la lista de identificadores (switch_index, light_index...). Esto significa que, si en el fichero de identificadores aparecen registrados tres sensores PIR, los códigos para estos sensores no tienen por qué ser necesariamente M1, M2 y M3, sino que el índice puede ser cualquiera elegido por el usuario, como M9, M3 y M5. Por lo tanto, la lista de índices en este caso sería [9, 3, 5].

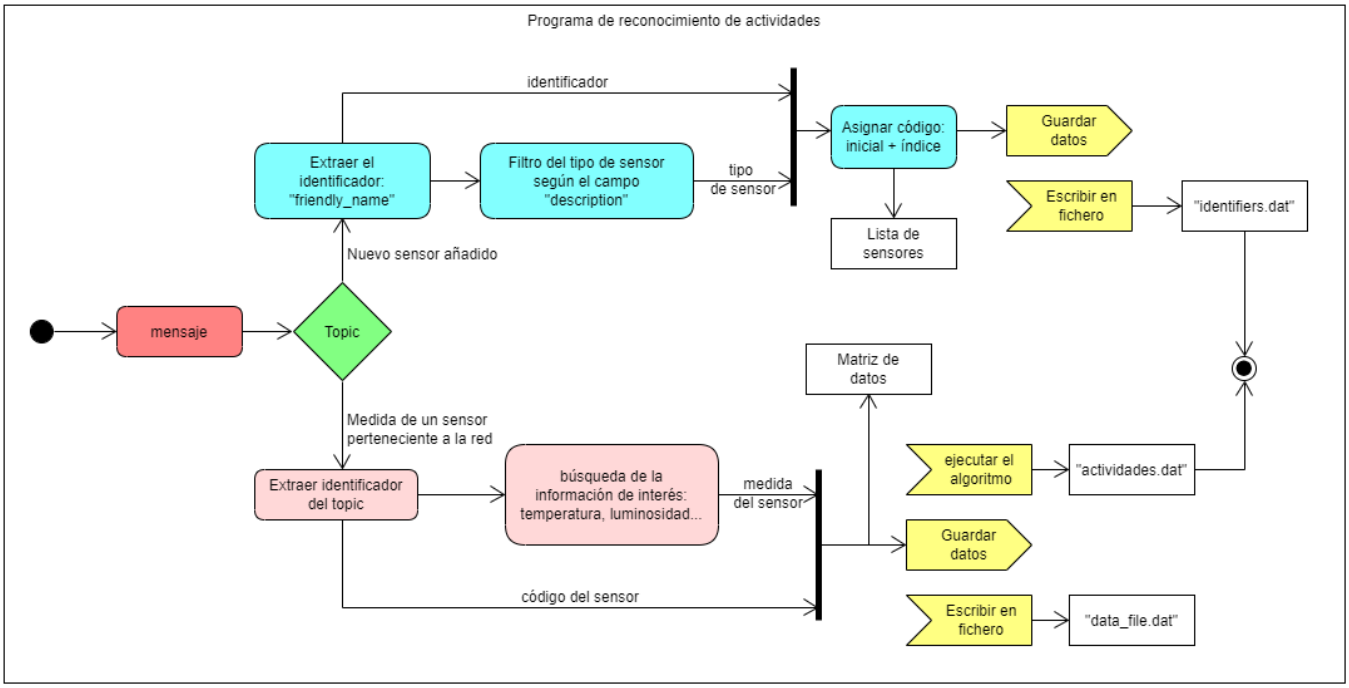


Figura 13. Diagrama UML de actividad del funcionamiento del programa

Una vez se han registrado todos los sensores que van a formar parte de la red (tarea que se realiza llamando a la función '*read_identifiers_file()*'), se crea una matriz de datos para cada tipo de sensor (light_data, temperatura_data...). Estas matrices tendrán tantas filas como sensores de ese tipo existan en la red: si hay tres sensores PIR, la matriz de datos PIR tendrá tres filas. El número de columnas de cada matriz dependerá del tipo de sensor: en el caso de los sensores de temperatura y humedad es interesante guardar, al menos, las últimas diez medidas, ya que más adelante será necesario realizar operaciones con ellas; en los demás casos, bastará con guarda las últimas cinco medidas.

A continuación, el programa leerá, si existe, un fichero con datos registrados anteriormente, a través de la función '*read_data_file()*' que se explicará más adelante. Esto es porque, cada dato recibido de los sensores por el programa no solo se guarda en la matriz de datos que se ha mencionado con anterioridad, ya que esta matriz solo guarda, como mucho, los últimos diez datos y los demás son descartados. Por ello, también se llevará un registro en un fichero de texto llamado '*data_file.dat*' donde se guardarán todos los datos enviados por los sensores mientras se esté ejecutando el programa.

Tras haber definido cada uno de los sensores que van a formar parte de la red y de registrar los todos los datos anteriores, se inicia la creación del cliente que se suscribirá a los mensajes del *zigbee2mqtt* y se establecerá la conexión con el bróker de Mosquitto utilizando una librería '*paho.mqtt.client*'. Se crea el cliente y se conecta a través de la función *on_connect()* que se suscribirá a los *topic* que se ha mencionado en el apartado 3.5. Por último, se llama a la función '*on_message*' que captura los mensajes publicados por *zigbee2mqtt* y se establece la conexión con el bróker, para lo que será necesario indicar el nombre del *host* y el número del puerto, que por defecto será el 1883. Para el *host* es frecuente utilizar el servidor local *localhost*, ya que la instalación del bróker de Mosquitto se suele realizar en la propia Raspberry Pi, sin embargo, también se puede acceder a un bróker externo introduciendo la dirección IP del servidor. En este trabajo no será necesario, pues el servidor está instalado en la Raspberry Pi.

Para asegurar el funcionamiento continuo del programa, se llama a la función *loop_forever()*, que ejecuta un bucle infinito en el que se mantiene la conexión con el bróker, dejando que el programa pueda recibir todos los mensajes enviados por los sensores. Solo se detendrá el programa ante una interrupción.

A continuación, se explicará detalladamente cómo está compuesta cada parte del programa y el papel que tiene para el correcto funcionamiento del sistema.

4.2.2. Clase *client*

Forma parte de la librería *paho.mqtt.client* de la que se hace uso en este programa. En ella se definen todas las funciones necesarias para que un cliente pueda conectarse al bróker MQTT, publicar mensajes, suscribirse a un topic y recibir mensajes.

El constructor de la clase puede tener cuatro parámetros opcionales, cuyos valores por defecto se muestran en la Figura 14.

```
Client(client_id="", clean_session=True, userdata=None, protocol=MQTTv311, transport="tcp")
```

Figura 14. Parámetros de la clase Client [3]

El *client_id* es el nombre que utiliza el bróker para identificar al cliente. Para crear un objeto de la clase Client, bastaría con definir un id que sea único por cada cliente. En este programa el nombre que se crea para el cliente es *Cliente1*.

Las funciones *on_connect*() y *on_message*() son las funciones que se han implementado para tratar algunas llamadas generadas por miembros de la clase durante la ejecución del programa. Ambas necesitan de un bucle infinito para procesar las llamadas que se producen cuando se recibe un mensaje.

El programa tendrá la siguiente estructura:

- Primero la creación del cliente.
- Luego la creación de la llamada a la función *on_connect*().
- Conexión de la llamada de reconocimiento del servidor con la llamada a la función *on_connect*().
- Conexión establecida con el bróker.
- Comienzo del bucle infinito.

4.2.3. Función on_connect()

Esta función se ha diseñado para asegurarse de que la conexión con el bróker se ha realizado con éxito. Esto es debido a que, al realizar la llamada a una función miembro de la clase Client, *client.connect*(), que tiene como parámetros de entrada, entre otros de menor interés, el nombre del host y el puerto donde se encuentra el bróker, durante el tiempo en que se realiza la conexión, el programa se bloquea. El resultado es que el bróker recibe un

intento de conexión que, en caso de ser aceptada, debe corresponder con una llamada de reconocimiento que es la que será tratada en esta función.

La llamada de reconocimiento debería recibir cuatro parámetros: el cliente que se ha conectado, los datos asociados a ese cliente, las 'banderas' que envía el bróker a modo de respuesta y un código de retorno que se utiliza para comprobar que se ha establecido la conexión satisfactoriamente, en cuyo caso el valor de este parámetro será 0. Los parámetros de interés son generalmente el nombre del cliente y el código de retorno.

La función *on_connect*() comprueba que el código de retorno y se suscribe a los *topic* de interés para el programa.

4.2.4. Función *on_message*()

Una vez se ha suscrito el cliente de Python a los *topic* de interés, comienza la espera del cliente a la publicación de nuevos mensajes en el bróker de Mosquitto publicados por *zigbee2mqtt*. Se ha añadido un retardo de 0.1 segundos debido a la naturaleza asíncrona de la llamada a esta función y, de esta forma, poder evitar problemas en la recepción de los mensajes.

Cada mensaje recibido será tratado por esta función. El mensaje que recibe está compuesto por el *topic* en el que se ha publicado, y la información que contiene el propio mensaje.

Se va a aplicar un filtro mediante sentencias *if* para distinguir entre los mensajes publicado en el *topic* en el que se anuncia la adhesión de un nuevo sensor a la red, y los mensajes con la información aportada por los sensores.

En el primer caso, sólo interesa que el sensor se haya conectado a la red convenientemente. Si es así, se hace una llamada a otra función que se ha diseñado para tratar la información que contiene el mensaje: *initialize_sensor_list().*

En el segundo caso, los mensajes contienen toda la información que aportan las medidas de los sensores (temperatura, vibración, movimiento, luminosidad…), por ello se ha creado otra función que tratará esa información identificando primeramente de qué tipo de sensor se trata, para saber qué información es la que se está recibiendo: *process_message()*.

4.2.5. Función *initialize_sensor_list*()

La función *initialize_sensor_list*() es la función que se encarga del tratamiento de los mensajes que anuncian que se ha añadido un nuevo sensor a la red.

En primer lugar, extrae el identificador del sensor (el campo '*friendly_name*' que se mencionó en el apartado 4.2), ya que este dato es único para cada sensor y por lo tanto es identificativo del mismo.

A continuación, mediante sentencias *if- else if*, se filtra el tipo del sensor que se está conectando en función del nombre comercial que tiene el sensor (el campo '*description*' que se mencionó en el apartado 4.2). Se puede observar en la Tabla 4, qué nombre es el que corresponde a cada tipo de sensor y qué clave se le va a asociar a todos los sensores del mismo tipo. La clave está compuesta por una letra y, a continuación, un índice numérico que será asignado automáticamente en esta función, teniendo en cuenta que dos sensores nunca pueden tener el mismo índice si son del mismo tipo.

Una vez asignados el índice y el identificador del sensor, se guarda cada dato en una lista. Existen dos listas para cada tipo de sensor, la lista de índices y la lista de identificadores. Como la forma de guardar los datos en estas listas es en forma de pila, la posición en la que se encuentra un identificador en la lista de identificadores es la misma en la que se encuentra el código asociado a ese sensor en la lista de índices.

Una vez se han guardado estos datos en sus listas correspondientes, se escriben en un fichero de texto: 'identifiers.dat'. De esta forma, se puede conservar el registro de los sensores que pertenecen a la red, sin tener que conectarlos cada vez que se arranque el sistema.

Cuando la Raspberry Pi se apaga o cuando se interrumpe la ejecución del programa, los sensores siguen conectados al *sniffer* y se siguen publicando los mensajes con el *zigbee2mqtt*, pero si no se registran los identificadores de los sensores, no sería posible suscribirse a los mensajes que se publican. Es lo que hace necesaria la implementación de esta función, ya que los mensajes de conexión a la red únicamente se publican una vez en el momento que se establece la conexión.

Otra forma de añadir un sensor a la red en caso de que se haya conectado antes de ejecutar el programa y no se tenga registro de él, es escribir manualmente en el fichero de texto con el formato [fecha, hora, identificador, código].

NOMBRE COMERCIAL	TIPO DE SENSOR	CÓDIGO
Aqara wireless switch	Botón	S
MiJia light intensity sensor	Luminosidad	L
Aqara temperatura, humidity and pressure sensor	Temperatura y humedad	T H
Aqara door & window contact sensor	Puertas y ventanas	C
Aqara human body movement and illuminance sensor	PIR	M
Xiaomi zigbee smart socket	Consumo de potencia	P
Aqara vibration sensor	Vibración	V
Aqara door & window contact sensor	Presencia en cama	B

Tabla 4. Tabla de nombre comerciales y códigos de los sensores

4.2.6. Función *removeprefix*()

Se trata de una función sencilla para eliminar un prefijo de un mensaje. La razón de que esta función sea necesaria es que, como se ha mencionado en apartados anteriores, los *topic* en los que se publican los mensajes enviados por los sensores siempre empiezan por "*zigbee2mqtt/*", seguidos del identificador del sensor que está enviando la información. Esto significa que se puede saber qué sensor está comunicándose a partir del *topic* del mensaje.

Por lo tanto, el diseño de una función para eliminar el prefijo "*zigbee2mqtt/*" del *topic* será de mucha utilidad, ya que se llamará a esta función cada vez que se reciba un mensaje de un sensor perteneciente a la red.

Esta función recibe como argumentos el mensaje que se va a tratar, y el prefijo que se quiere eliminar. La salida será el mensaje con el prefijo eliminado.

4.2.7. Función *process_message*()

En esta función se tratarán todos los mensajes con información procedente de los sensores que pertenecen a la red. Cada tipo de sensor aporta una información relevante que será la que se quiere guardar para su posterior tratamiento. Por ejemplo, a pesar de que los mensajes publicados por un sensor de tipo PIR incluyen varios campos a parte de la detección de presencia, como temperatura del dispositivo, luminosidad, voltaje, entre otros; el sensor sólo se activa cuando existe un cambio en la detección de presencia y, por lo tanto, es el único dato que interesa recoger. No se podrían utilizar los sensores PIR como sensores de luminosidad, ya que un cambio en la iluminación no activa el sensor, solo se activa cuando detecta presencia. Esto es un factor relevante a la hora de desplegar la red de sensores por el domicilio.

Sin embargo, los sensores de detección de temperatura y humedad, sí que se activan cuando existe un cambio en cualquiera de esas características, es por eso por lo que este tipo de sensores se desglosan como si fueran dos sensores independientes.

La función obtiene como argumentos el *topic* en el que se publica el mensaje y el mensaje en sí mismo. Primero, se obtiene el identificador partiendo del *topic* del mensaje con la función *removeprefix*(). Una vez obtenido el identificador, se pasa por un filtro de sentencias *if- else if* en el que se busca si el identificador del *topic* se encuentra en alguna de las listas de identificadores de los distintos tipos de sensores.

Cuando se obtiene una coincidencia dentro de algunas de las listas, obtenemos a su vez el tipo de sensor que está mandando la información, por lo tanto, se busca dentro de la lista de índices del tipo de sensor aquel elemento que se encuentre en la misma posición que el identificador en la lista de identificadores. Con este procedimiento se puede obtener el código del sensor a partir del identificador.

Cada tipo de sensor tiene un formato de mensaje distinto, por lo que es necesario conocerlo para poder obtener la información que se necesita. En la Tabla 5 se puede observar para cada tipo de sensor, el nombre del campo dentro del mensaje que contiene la información de interés y los posibles valores que se pueden encontrar. Con la función "*busca_info*", que se explicará más adelante, se puede obtener la información que se busca introduciendo el nombre del campo en el que aparece.

Existen dos tipos de sensores, aquellos cuya información aportada puede ser una opción entre un número finito de valores (por ejemplo: verdadero/falso) y aquellos cuya información se encuentra dentro de un rango infinito de valores (por ejemplo, la temperatura se encuentra entre -20 y +50 ºC). Para poder

manejar mejor la información aportada por los sensores de valores finitos, se transformarán los datos que aportan a un formato numérico según se muestra en el Tabla 5. En este formato, será más sencillo poder visualizar los resultados para el posterior diseño del algoritmo.

TIPO DE SENSOR	CAMPO	VALORES EN EL MENSAJE	VALORES EN EL REGISTRO
Botón	"action"	single	1
		double	2
		hold	3
		release	0
Luminosidad	"illuminance"	[0, 83000] lux	[0, 83000] lux
PIR	"occupancy"	false	0
		true	1
Temperatura	"temperature"	[-20, 50]ºC	[-20, +50]ºC
Humedad	"humidity"	[0, 100] %	[0, 100] %
Vibración	"action"	null	0
		vibration	1
		tilt	2
		drop	3
Consumo de potencia	"power"	[0, 3000] W	[0, 3000] W
Puertas y ventanas	"contact"	false	0
		true	1

Tabla 5. Medidas de cada tipo de sensor

4.2.8. Función *read_identifiers_file*()

Esta función es la primera a la que se llama en el momento en el que se inicia el programa para comprobar si existen sensores actualmente dentro de la red. Se encargará de leer, si existe, el fichero con los datos de los sensores que pertenecen a la red. Para ello, se deberá tener en cuenta el formato de este fichero en el que cada fila de texto representa un único sensor, y cada columna contiene la información relacionada con ese sensor. De izquierda a derecha, los datos que aparecen para cada sensor son: fecha de la primera conexión, hora de la primera conexión, identificador del sensor y, por último, el código del sensor. A mayores, se podrá añadir una columna en la que indicar la posición del sensor dentro de la casa, por ejemplo: puerta de entrada, silla del escritorio, etc.

La función abre el fichero en modo de sólo lectura e irá leyendo línea por línea. Cada línea se dividirá según las distintas columnas, quedando como resultado una lista en la que cada posición corresponde con una columna de información sobre el sensor: en la primera posición se almacenará la fecha, en la segunda, la hora, en la tercera el identificador y en la cuarta el código. Como el objetivo de esta función es inicializar las listas de los sensores, primero es necesario saber de qué tipo será el sensor del que se leerán los datos, por lo que se pasará por un filtro de sentencias *if- else if* analizando el código del sensor, ya que será el campo que indica de qué tipo será.

Una vez identificado el tipo de sensor del que se trata, se añadirá el identificador a la lista de identificadores del tipo de sensores que corresponda y el índice del código en la lista de índices que corresponda.

Tras leer todos los datos y almacenarlos en las listas correspondientes, se podrá continuar con la ejecución del programa, ya que será necesario conocer los identificadores de los sensores que pertenecen a la red para suscribirse a los mensajes que envían. Esta función permite recordar los sensores que se han conectado cada vez que se inicie el programa para poder determinar las dimensiones de las matrices de datos de cada tipo de sensor.

4.2.9. Función *algorithm*()

El diseño de esta función es posterior a la etapa de etiquetado, ya que es necesario conocer la rutina del usuario y poder establecer patrones que permitan determinar las premisas que indican qué actividad se está realizando en cada momento.

El objetivo de esta función es de interpretar cada línea de datos que se obtiene de los sensores, tratando de relacionar las medidas de los diferentes sensores para obtener como salida la actividad que se está realizando.

Para ello se crean distintas variables de tipo verdadero/falso para guardar el estado de una actividad, es decir, si esa actividad se está realizando o no. Las diez actividades que se pretende identificar son:

- Dormirse
- Despertarse
- Vestirse
- Ducharse

- Ir al baño
- Cocinar
- Entrar en casa
- Salir de casa
- Ver la televisión
- Trabajar con el ordenador

Algunas de estas actividades son instantáneas, es decir, se realizan en un momento determinado y no tienen duración como, por ejemplo, entrar en casa, salir de casa, irse a dormir y despertarse. Cuando el algoritmo se active debido a alguna de estas actividades, se mostrarán en el fichero de texto como "detectadas". Las demás actividades tendrán una fecha y hora en la que se han iniciado; y una fecha y hora de finalización.

Se deberá conocer qué sensor es el último que se ha activado y en qué habitación se encuentra para poder identificar las actividades que se pueden llevar a cabo en esa habitación. Mediante sentencias *else- if else* se filtran los sensores de los que previamente se conoce su posición y se establecen unas reglas o condiciones que indican el comienzo o el final de cada actividad. Cada actividad detectada, iniciada o finalizada se escribirá en un fichero de texto llamado "*actividades.dat*" junto con la fecha y la hora a la que se ha detectado.

Antes de comenzar con el diseño del algoritmo, se debe desplegar la red de sensores en el domicilio del usuario y realizar la recogida de datos durante un tiempo que no debe ser menor de un mes. Estos datos deberán ser estudiados para el etiquetado de las actividades y el diseño de este algoritmo.

Una vez diseñado el algoritmo, se debe integrar en el programa para comprobar su veracidad durante las siguientes semanas.

4.2.10. Función *read_data_file*()

Esta función se encargará de leer, si existe, un fichero con los datos registrados en el fichero de texto "*data_file.dat*". El objetivo de esta función es el de almacenar los datos registrados en anteriores ejecuciones del programa dentro de las matrices de datos de cada tipo de sensor.

El procedimiento comienza abriendo el fichero de texto en modo sólo lectura. Se recuerda que, en este fichero, los datos se escriben en líneas, y cada línea almacena la información que se ha extraído de un mensaje recibido de uno de los sensores. Dentro de cada línea, existen varias columnas con distintos datos relativos a la medida, que de izquierda derecha son: fecha en la que se ha recogido cada medida, hora en la que se ha recogido la medida, código del sensor que ha recogido la medida y, por último, medida que ha recogido el sensor según el formato de la Tabla 5 del apartado 4.2.7.

De cada línea de texto, se buscará el tipo de sensor que realiza la medida para saber en qué matriz se deberán guardar los datos. Estas matrices tienen tantas filas como sensores de ese tipo existan en la red y tantas columnas como datos se quieran guardar. Por lo tanto, se deberá buscar el índice del sensor dentro de la línea de texto para saber en qué fila de la matriz se debe guardar el dato. La forma de almacenar los datos que envía un sensor determinado dentro de la matriz es introduciendo el nuevo dato en la posición 0 y eliminando el dato en la última posición en forma de cola FIFO: el primero en entrar es el primero en salir.

Tras haber guardado el dato en la matriz correspondiente, se hace una llamada al algoritmo de reconocimiento de actividades, que buscará la relación entre la medida que se acaba de leer con las medidas anteriores de los distintos sensores registradas en el resto de las matrices.

4.2.11. Función *write_activity_file()*

Esta función será llamada cada vez que el algoritmo de reconocimiento de actividades haya detectado alguna activación. Se escribirá en el fichero "*actividades.dat*" siguiendo el siguiente formato: fecha y hora a la que se inició o se finalizó la actividad, estado de la actividad (iniciada/finalizada/detectada) y, por último, el nombre de la actividad que ha provocado la activación.

Este fichero muestra los resultados obtenidos por el algoritmo y es el que se utilizará para comprobar la veracidad y la precisión del sistema.

4.2.12. Función *write_data_file*()

El objetivo de esta función es el de llevar un registro en ficheros de texto de toda la información aportada por los mensajes que son enviados por los sensores y capturados en la Raspberry Pi. Esta función se llamará cuando existan datos para registrar en alguno de los siguientes ficheros:

- "*identifiers.dat*": la llamada a esta función se realizará cuando se haya recibido un mensaje por parte de un sensor que se acaba de añadir con éxito a la red. El formato en el que se escribe en este fichero se muestra en la Tabla 6.

FECHA DE CONEXIÓN	HORA DE CONEXIÓN	IDENTIFICADOR DEL SENSOR	CÓDIGO	(opcional) LUGAR
2021-04-18	00:01:48.329410	0x00158d00034fba5b	C3	entrada

Tabla 6. Formato del fichero de identificadores

FECHA	HORA	CÓDIGO DEL SENSOR	MEDIDA EN FORMATO NUMÉRICO
2021-04-20	20:35:11.896539	T3	27.18
2021-04-20	21:02:08.285760	M2	1
2021-04-20	21:05:46.643784	V6	0

Tabla 7. Formato del fichero de datos

- "*data_file.dat*": este fichero contiene todos los datos recogidos por los sensores durante la ejecución de todo el programa. Cada vez que se recibe un dato, se registra en el fichero siguiendo el formato de la Tabla 7. La fecha y la hora en la que se llama a esta función es la misma que se escribe en el fichero de texto, por lo que una vez se ha registrado esta medida en el fichero, también se guardan los datos en las matrices destinadas para ello, de igual forma que se hace con la función de *read_data_file*() explicada en el apartado 4.2.10.

  Una vez se han guardado los datos tanto en el fichero de texto como en la matriz de datos del tipo de sensor adecuado, se hace una llamada al algoritmo de reconocimiento de actividades que buscará una relación entre la medida que se acaba de registrar y las medidas ya guardadas en cada una de las matrices de datos de los sensores.

4.2.13. Función *busca_info*()

Esta es una pequeña función para extraer de los mensajes la información incluida en el campo que se señale. Los argumentos son el mensaje que contiene la información y el nombre del campo que se desea extraer.

La función busca dentro del mensaje el nombre del campo y se queda con la posición de la cadena de caracteres del mensaje donde comienza el dato. La posición final será donde se encuentre el carácter ',' ya que los campos están separados entre sí por comas.

La llamada a esta función se realizará desde la función *process_message*() explicada en el apartado 4.2.7 una vez se haya identificado el tipo de sensor que envía el mensaje y se conozca el dato que se quiere procesar.

4.3. Desarrollo del sensor de presencia en cama

Este tipo de sensores no son muy frecuentes en el mundo de la domótica, sin embargo, su uso en este proyecto es determinante para poder identificar actividades como irse a dormir y despertarse.

El objetivo es desarrollar un sensor de presencia en cama a partir de un sensor de contacto ZigBee que se active cuando el usuario se tumbe en la cama. Este sensor es el que se utiliza normalmente para detectar cuando se abre y se cierra una puerta o ventana, y consta de dos piezas: un pequeño imán y una pieza electrónica donde se encuentra el circuito que permite el funcionamiento de este dispositivo. Para el desarrollo del sensor de presencia en cama, la parte del imán no se utilizará.

- El primer paso será soldar un par de cables en los extremos del interruptor de lengüeta del circuito como se muestra en la Figura 15, donde en negro se señala el interrupto de lengüeta y en rojo la posición donde se van a soldar los cables, y se harán dos taladros en la caja de plástico del sensor para que los cables sobresalgan. Este interruptor es activado por un campo magnético, pero soldando los dos cables a los extremos se puede modificar para que se active cuando estos cables hagan contacto.

- Una vez conectados los cables, el sensor mandaría una señal a la Raspberry Pi cuando los cables hagan contacto creando un puente. Esta señal sería equivalente a la que se envía el sensor de puerta cuando detecta que está en contacto con el imán. Para poder regular la señal que envía este sensor y limitarla para que se accione únicamente cuando detecte presión, se crearán dos tiras de cartón en las que se instalarán y soldarán botones en serie (ver Figura 16b).

Figura 15. Circuito electrónico del sensor de puerta donde se soldarán los cables [10]

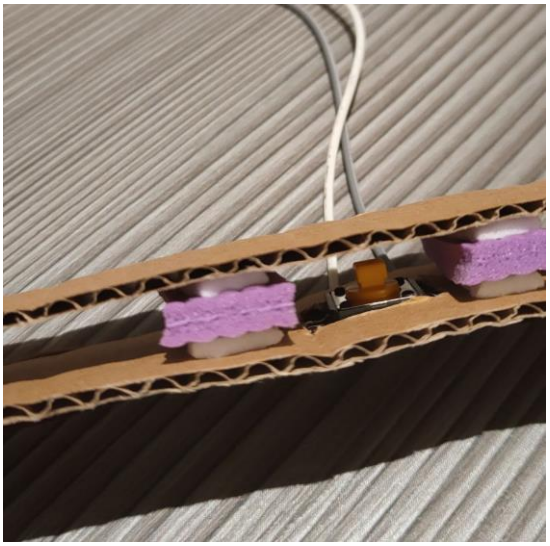


Figura 16a. Botón dentro de las tiras de cartón

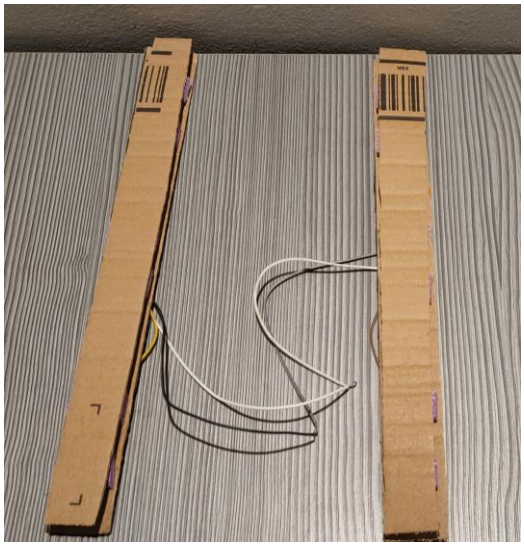


Figura 16b. Tiras de cartón con botones en serie.

- Cada uno de los cables que salen del sensor de puerta se soldará a uno de los extremos de cada fila de botones. De esta forma cada vez que, al menos, uno de los botones se accione, se cerrará el circuito que puenteaba el interruptor magnético, provocando el envío de la señal del sensor de contacto.

- Por último, para poder adaptar el accionamiento de los botones para que no se provoque una activación debida al propio peso del colchón, se han añadido unas almohadillas en los extremos de cada uno de los botones en cada tira de cartón (Figura 16a). Estas almohadillas consiguen aguantar una fracción del peso del colchón lo suficiente para evitar que se activen accidentalmente.

- La instalación de este sensor se realizará colocando las tiras de cartón paralelas al lado de la cama en el que se coloca la almohada, a una distancia más o menos central.

- Es necesario tener en cuenta que cuando se conecte este sensor a la red, el programa lo identificará como un sensor de puertas y ventanas, asignándole el código que comienza en “C” (ver Tabla 4 del apartado 4.2.5). Sin embargo, se ha creado una categoría que incluye el sensor de cama identificándolo, según los códigos establecidos, con una “B”. Para que el programa lo identifique como tal, se debe modificar el fichero de texto “*identifiers.dat*”, localizando el identificador que define el sensor que se ha modificado y cambiando manualmente la “C” del código que el programa le ha asignado automáticamente por una “B”.

5. Resultados obtenidos

5.1. Análisis de la rutina del usuario

A continuación, se van a mostrar la evolución de las medidas de ciertos sensores de cara al algoritmo de reconocimiento de actividades. Se va a estudiar esta evolución contrastando los datos con las actividades que el usuario ha marcado que estaba realizando a través de los botones para cada una de las actividades. La comparación en este apartado de las actividades detectadas con las actividades que el usuario ha indicado que ha realizado se explicará mostrando un ejemplo de un día cualquiera, pero la búsqueda del patrón para cada actividad es fruto del estudio de varias semanas de toma de datos.

5.1.1. Dormir y despertarse

A continuación, se puede observar un ejemplo de la evolución de los parámetros más significativos durante la realización de estas tareas a lo largo de 24 horas. Algunas observaciones sobre la evolución de estos parámetros, sabiendo que el día 30 de junio de 2021 el usuario se fue a dormir a las 00:45 y se despertó a las 8:30, son las siguientes:

- La luz de la habitación es generalmente 0 mientras el usuario está durmiendo (Figura 17).
- El sensor de presencia en cama está activo mientras el usuario está dormido (Figura 18).
- El sensor PIR detecta que el usuario está en el dormitorio (Figura 19).
- La puerta está cerrada mientras el usuario está dormido (Figura 20).
- Al despertarse, el usuario enciende la luz y abre la puerta (Figura 17 y Figura 20).
- El sensor de presencia en cama se desactiva cuando el usuario se despierta (Figura 18).

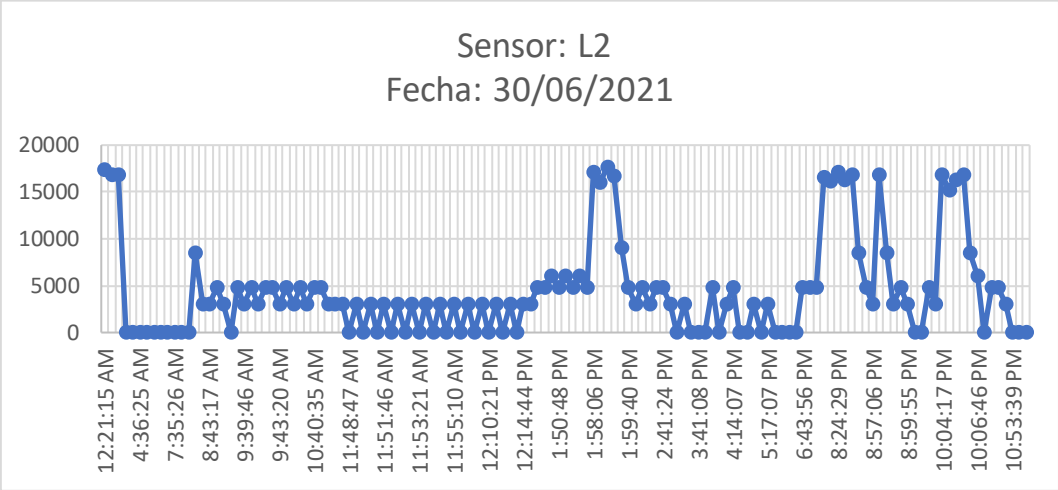


Figura 17. Evolución de la luminosidad en el dormitorio

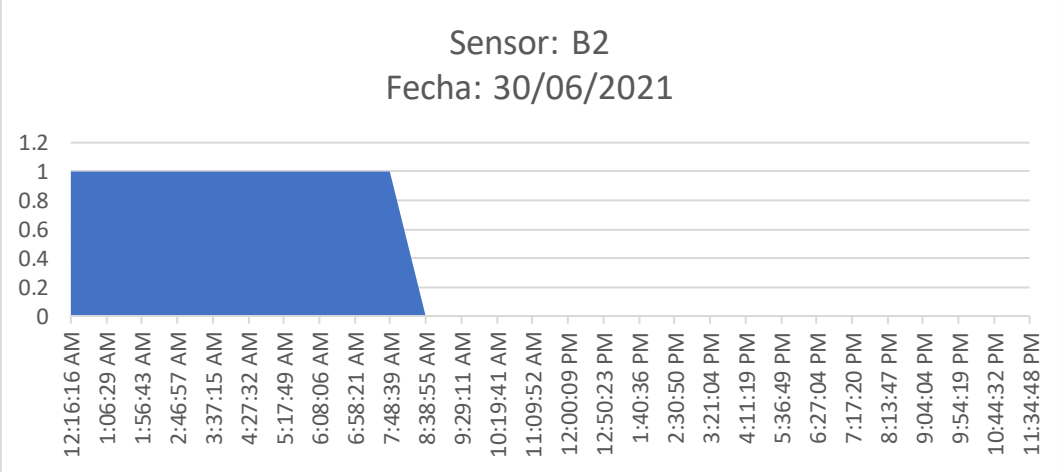
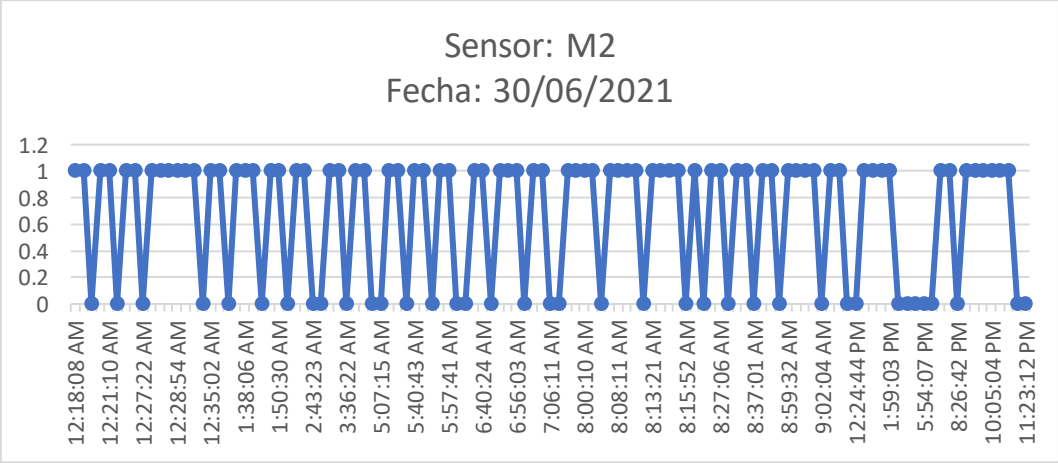


Figura 18. Evolución del sensor de presencia en cama



Nota: obsérvese que las horas de activación del sensor no están distribuidas igualitariamente

Figura 19. Evolución del sensor PIR del dormitorio

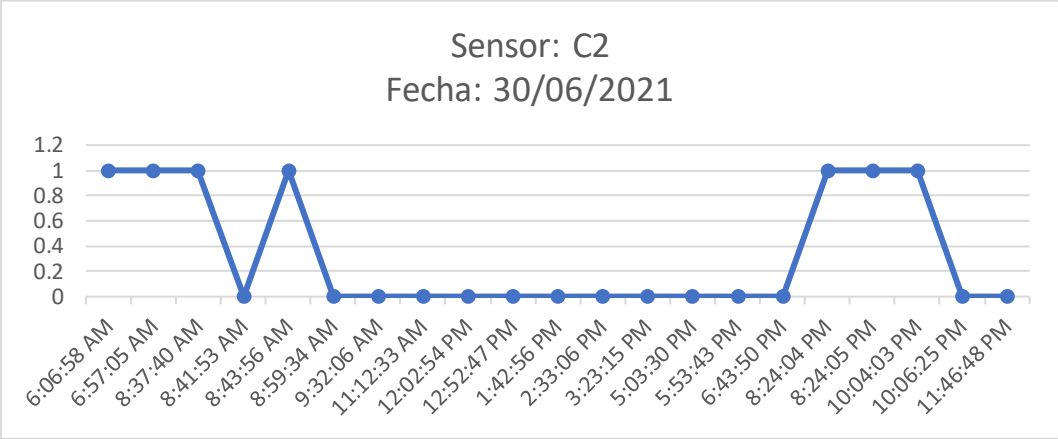


Figura 20. Evolución del sensor de la puerta del dormitorio

5.1.2. Vestirse

En este caso, la información aportada por el sensor de vibración situado en uno de los cajones del armario ofrece uno de los datos más relevante cuando se está realizando esta actividad. Tras estudiar durante varias semanas la recogida de datos se pueden obtener las siguientes observaciones a partir de los datos de un día cualquiera, como el 1 de julio de 2021, en el que el usuario registró que se cambió de ropa desde las 18:16 a las 18:20:

- Cuando el usuario se cambia de ropa, se activa el sensor de puerta (Figura 21).
- También se activa el sensor de luz, observando medidas de luminosidad de hasta 17000 lux (Figura 22).
- El sensor de movimiento del dormitorio se activa (Figura 23), detectando la presencia del usuario dentro de esta habitación.
- El sensor de vibración del armario se activa (Figura 24).

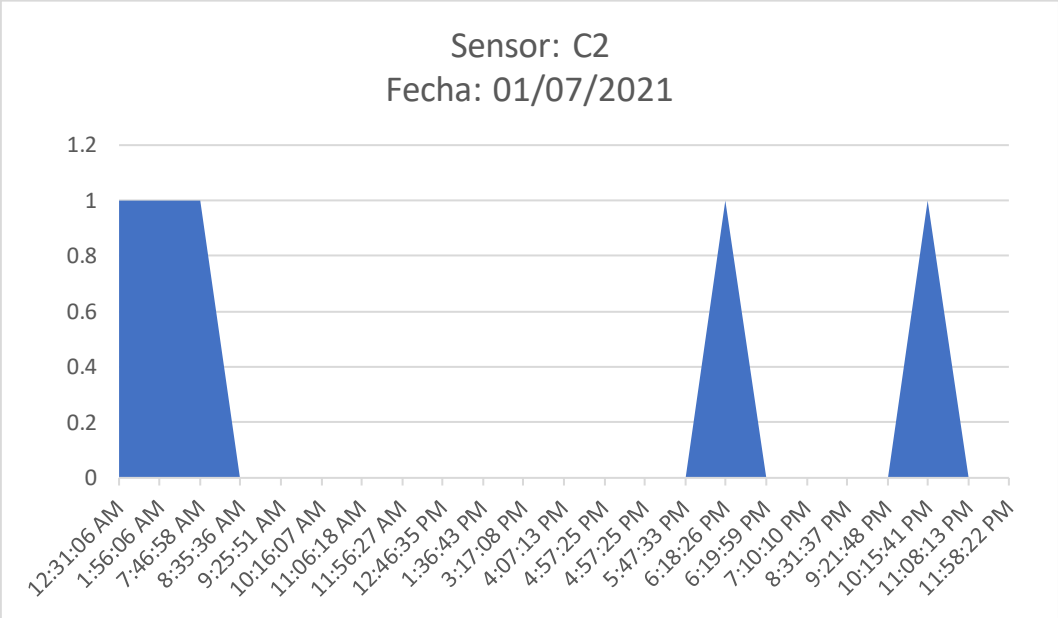


Figura 21. Evolución del sensor de puerta del dormitorio

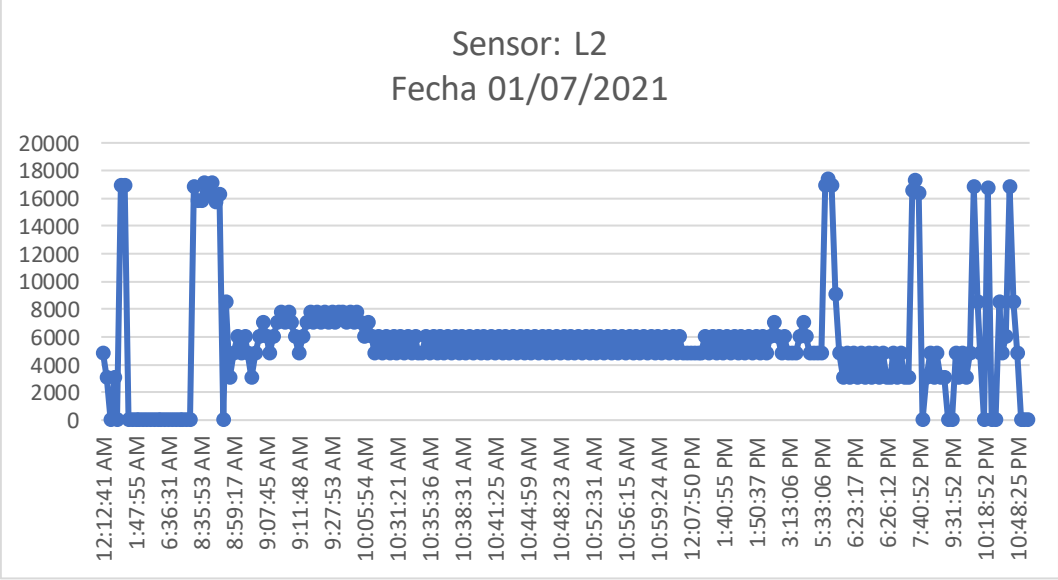


Figura 22. Evolución de la actividad del sensor de luz del dormitorio

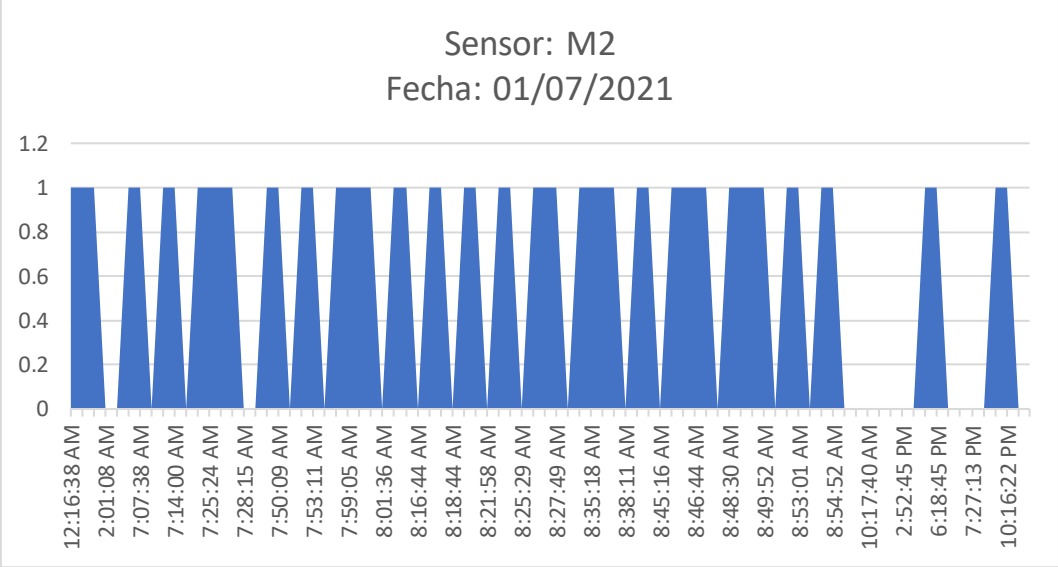


Figura 23. Evolución de la actividad del sensor PIR del dormitorio

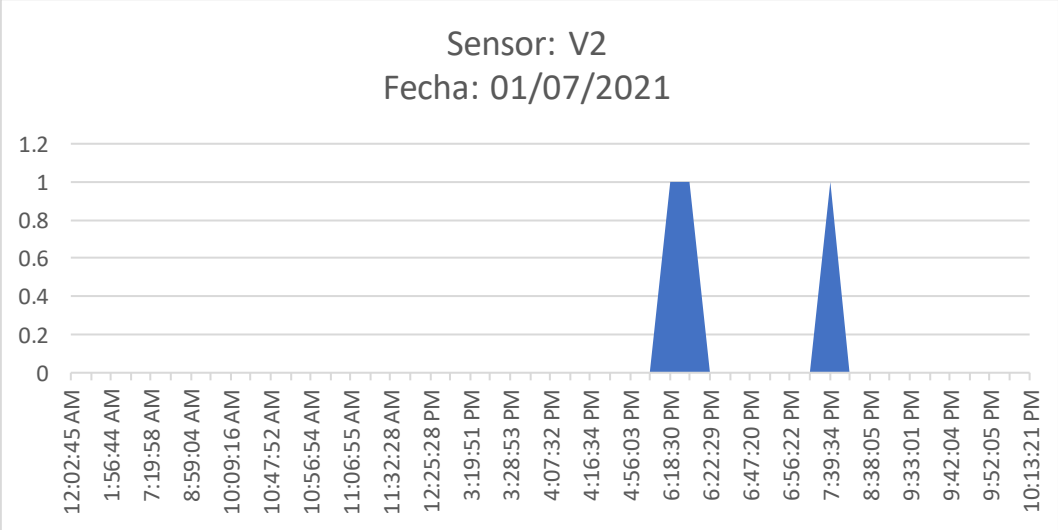


Figura 24. Evolución de la actividad del sensor de vibración del dormitorio

5.1.3. Ducharse o ir al baño

Estas dos acciones tienen lugar en el baño, por lo que los sensores más relevantes serán aquéllos con índice 3. Las premisas sobre la realización de estas actividades son muy parecidas entre sí, el único factor que cambia es la humedad de la habitación. Se pueden destacar las siguientes observaciones de la rutina del usuario, que se explicarán basándose en un día cualquiera, como el 1 de julio de 2021, en el que el usuario se duchó a entre las 22.08 y las 22.15 y fue al baño varias veces entre las 14.00 y las 23.59 (por ejemplo, a las 16.31 y 19.07):

- Tanto cuando se va a duchar como cuando va al baño, el sensor de puerta se activa (Figura 25), indicando que está cerrada, mientras que cuando el baño está desocupado, este sensor permanece a 0 (puerta abierta).
- El sensor de luz del baño marca normalmente 0, salvo cuando el usuario indica que va a realizar alguna actividad dentro, en cuyo caso este sensor marca una luminosidad superior a los 4000 lux (Figura 26).
- El factor más determinante es la humedad (Figura 27): en los casos en los que el usuario indica que se va a duchar, la humedad de la habitación aumenta considerablemente, mientras que cuando el usuario solo va al baño, la humedad que marca el sensor apenas se inmuta.

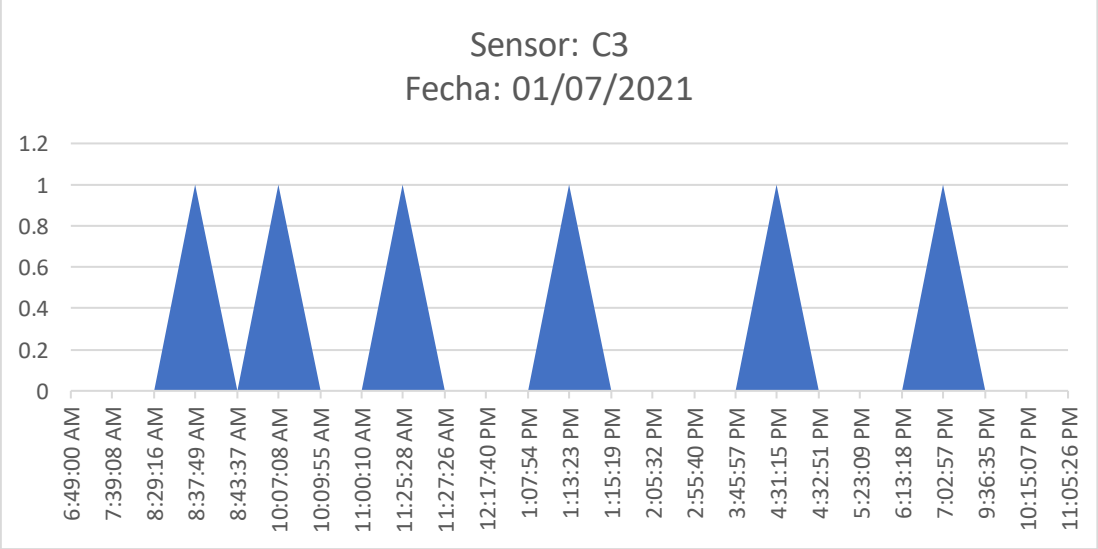


Figura 25. Evolución de la actividad del sensor de puerta del baño

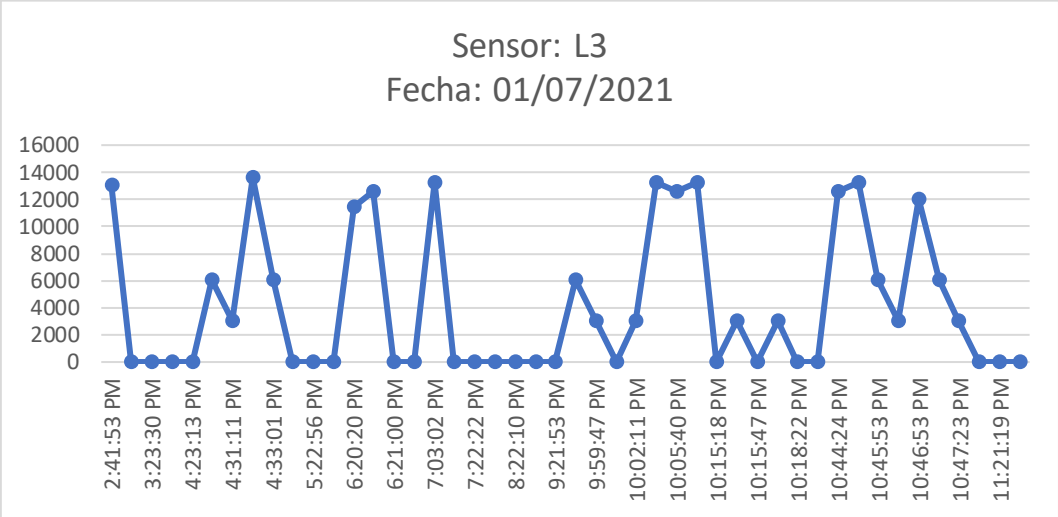


Figura 26. Evolución de la actividad del sensor de luminosidad del baño

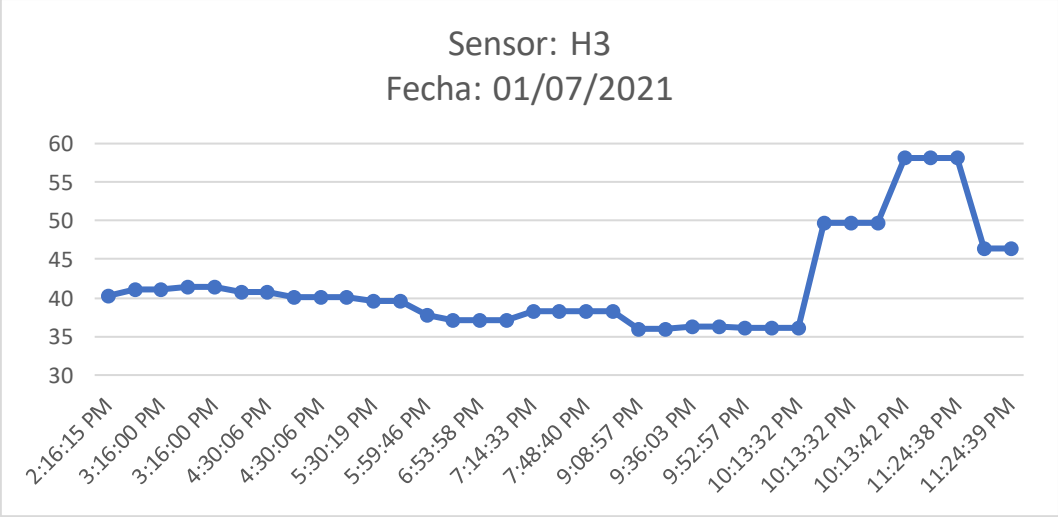


Figura 27. Evolución de la actividad del sensor de humedad del baño

5.1.4. Cocinar

Tras varias semanas de estudio, se ha observado el comportamiento de los sensores colocados en la cocina para tratar de identificar un patrón cuando el usuario realiza la acción de cocinar. Como se observa en la gráfica de la Figura 28, que representa la temperatura de la cocina a lo largo del día 30 de junio de 2021, en el que el usuario indicó que realizo la actividad de cocinar a las

14.10, a las 17.02, a las 21.47 y a las 23.05, la temperatura aumenta cuando se realiza la actividad, produciendo esos picos que se muestran en la imagen.

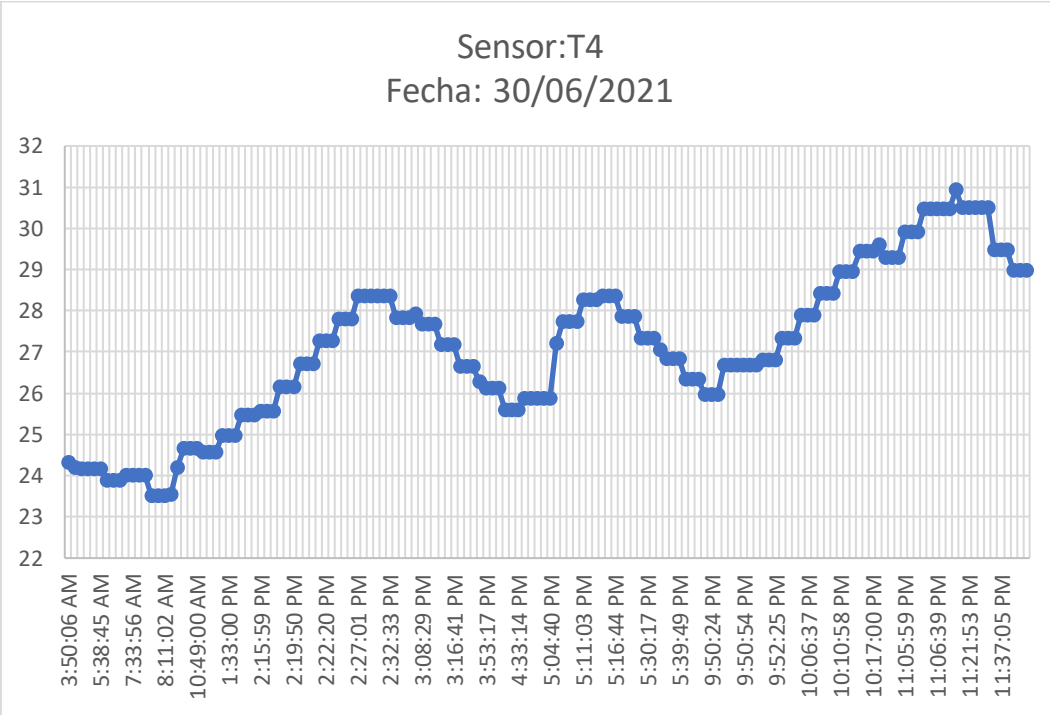


Figura 28. Evolución de la actividad del sensor de temperatura de la cocina

5.1.5. Entrar y salir de casa

Para esta actividad, es más difícil encontrar un patrón, ya que el único sensor que hay colocado en la entrada es el de la puerta. El sensor de puerta permanece a 1 la mayor parte del tiempo (ver Figura 29, datos tomados el día 1 de julio de 2021 de 14.00 a 22.00, el usuario indicó que **salió** de casa a las 14.42, a las 17.33, a las 18.20 y a las 19.40; y que **entró** en casa a las 16.30, a las 17.37, a las 19.15 y a las 21.58), pero no siempre que el sensor se pone a 0 significa que el usuario ha salido de casa, ya que pueden existir otras razones por las que el usuario abra la puerta de casa.

Sin embargo, se puede discriminar cuantas de las veces que se abre la puerta son para salir de casa si se observa la activación del resto de actividades. Si el usuario abre y cierra la puerta y no se detecta que se esté realizando ninguna otra actividad, significa que el usuario ha salido de casa, pero si al cerrarse de nuevo la puerta, se detecta alguna actividad o activación de los PIR, implica que la persona sigue dentro de la casa.

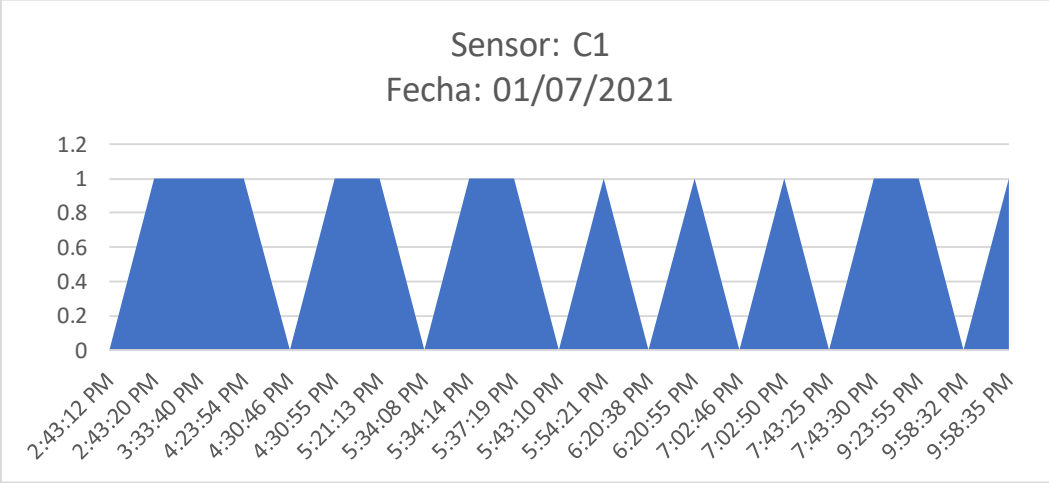


Figura 29. Evolución de la actividad del sensor de la puerta de entrada

5.1.6. Ver la televisión

En la gráfica de la Figura 30 se observa la toma de datos recogida por el sensor de detección de consumo el día 2 de julio de 2021, en el que el usuario indicó que estuvo viendo la televisión entre las 15.00 y las 15.25 y desde las 20.29 a las 20.54. Se puede observar claramente los picos de potencia que se producen mientras el usuario está realizando esta actividad, mientras que el resto del tiempo el valor de que registra este sensor es muy próximo a 0.

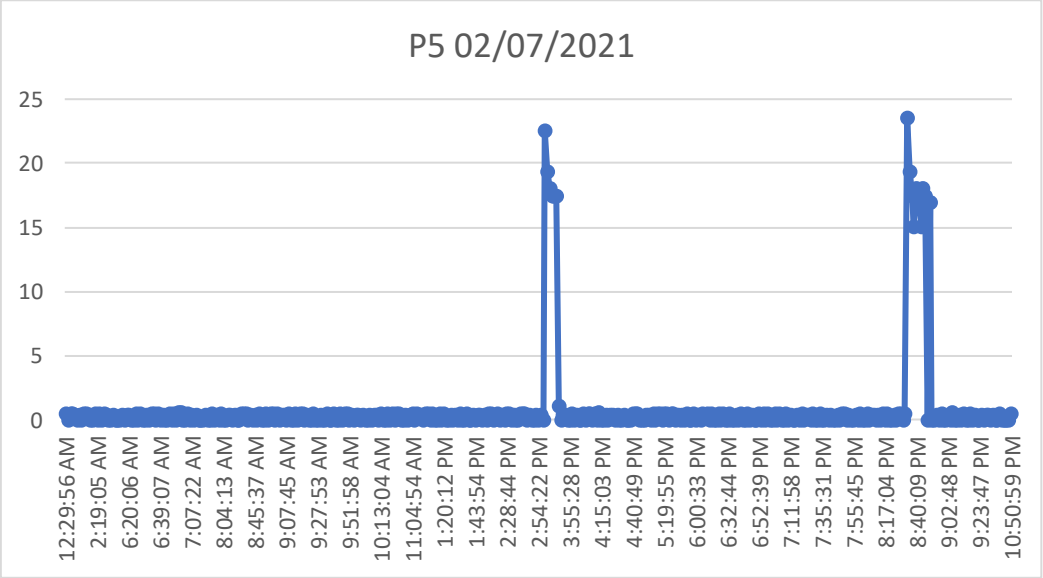


Figura 30. Evolución de la actividad del sensor de consumo del salón

5.1.7. Trabajar en el ordenador

En este caso, al igual que con las actividades de salir y entrar en casa, la detección de un patrón es más complicada, ya que el despacho es una habitación muy concurrida, como se observa en la gráfica de la Figura 31, que muestra las medidas del sensor PIR a lo largo del día 1 de julio de 2021 desde las 10.00 a las 12.00, el sensor M6 se activa muy frecuentemente. Sin embargo, como se muestra en la Figura 32, que muestra las activaciones del sensor de vibración colocado en la silla del despacho durante el mismo periodo de tiempo.

Estas medidas en conjunto con las obtenidas por el sensor de movimiento, y sabiendo que el usuario indicó que estuvo trabajando con el ordenador entre las 10.10 y las 11.24 de la mañana, puede observarse que la actividad comienza cuando el sensor de vibración se activa y el finaliza cuando se vuelve a activar, pero entre ambas activaciones, de inicio y de final, existe presencia detectada por el sensor PIR.

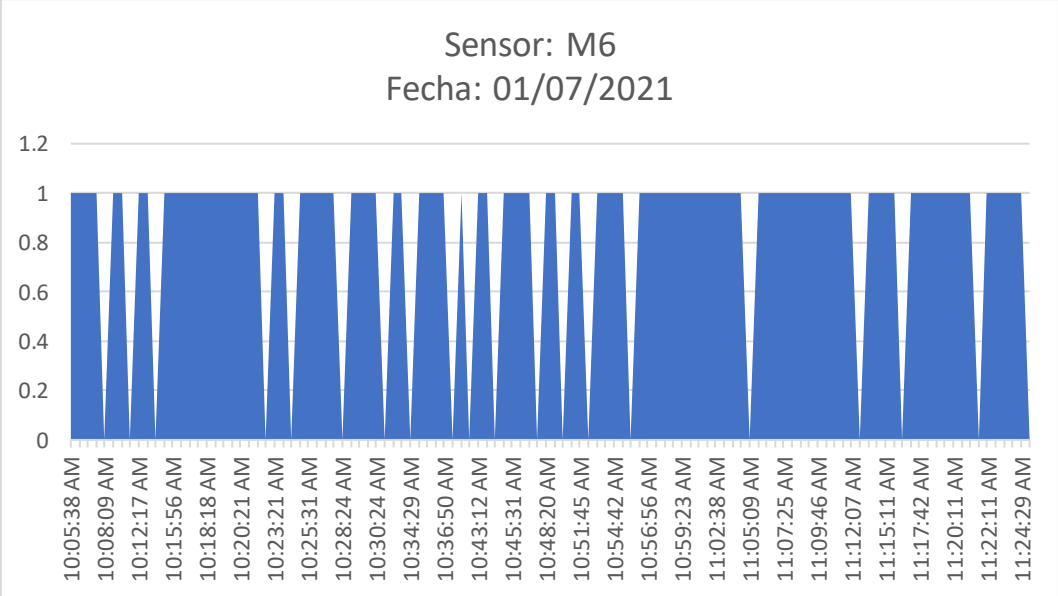


Figura 31. Evolución de la actividad del sensor PIR del despacho

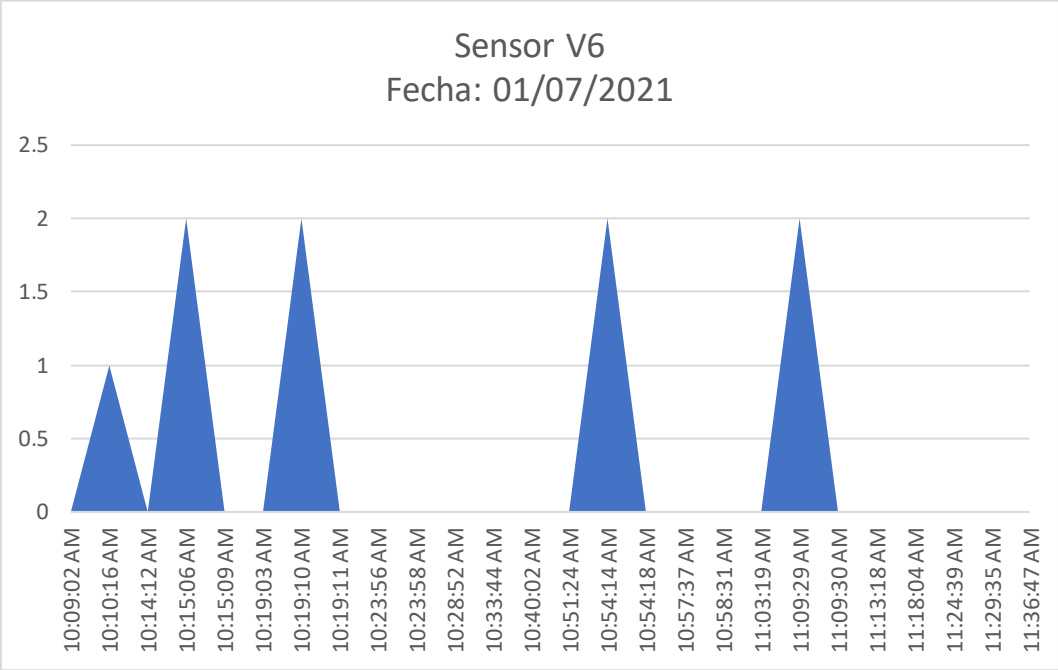


Figura 32. Evolución de la actividad del sensor de vibración del despacho

5.2. Resultados del algoritmo de reconocimiento de actividades

El etiquetado de las actividades se ha realizado teniendo en cuenta el análisis de la rutina del usuario. Con cada actividad, como se ha mencionado anteriormente, se activan ciertos sensores y se cumplen unos patrones. Con estos patrones, los cuales se han analizado en el apartado 5.1, se crean las premisas que utilizará el algoritmo de detección de actividades para la identificación del inicio y el final de cada una de las actividades mencionadas.

Una vez configurado el algoritmo en base al estudio previo, se procede a comprobar la fiabilidad del sistema para cada una de las actividades. Esta fiabilidad se mide comparando las veces que el usuario ha indicado el comienzo de la actividad de estudio con las veces que el algoritmo ha detectado que se estaba realizando.

Se pueden distinguir varios tipos de resultados:

- En primer lugar, cuando la actividad que se ha realizado a una hora se ha identificado correctamente a la misma hora (con una tolerancia de 15 minutos). En cuyo caso los resultados son óptimos y el sistema ha respondido correctamente.

- Luego están las actividades que se han realizado a una hora y el sistema las ha identificado bien, pero la hora a registra que se han realizado difiere más de 15 minutos, pero menor de 2 horas. En este caso los resultados son válidos pero mejorables.
- A continuación, estarían los resultados en los que el sistema confunde dos actividades parecidas que se realizan en la misma sala. Por ejemplo, podría pasar que el sistema confunda la acción de salir de casa con la acción de entrar en casa, o que confunda la actividad de ir al baño con la de ducharse. Estos resultados no son válidos.
- Por último, estarían los resultados en los que el sistema no detecta una actividad que se ha realizado, o cuando el sistema detecta una actividad cuando ésta no se ha realizado. Estos resultados no son válidos.

Recopilando los datos recogidos durante una semana, se pueden observar los resultados obtenidos en la Tabla 8. Como se puede ver, la mayoría de las actividades son detectadas correctamente en más de la mitad de los casos, salvo en excepciones como pueden ser las actividades “salir de casa” y “entrar en casa”, en estos casos se observa que son muy frecuentes las detecciones confundidas, es decir, que se está detectando una actividad similar a la que en realidad se está realizando. Los mismo sucede si se observan los resultados de las actividades “ducharse” e “ir al baño”.

Para poder observar este fenómeno mejor, se ha creado la tabla de confusión que se observa en la Tabla 9.

NOMBRE DE ACTIVIDAD	NO DETECTADAS	CONFUNDIDAS	FALSAS DETECCIONES	CORRECTAS	NUMERO DE VECES
Dormirse	2	1	0	4	7
Despertarse	0	1	0	6	7
Vestirse	5	0	0	6	11
Entrar en casa	5	8	1	11	25
Salir de casa	2	12	5	16	35
Ducharse	4	6	0	2	12
Ir al baño	11	7	3	30	51
Cocinar	0	3	2	23	28
Ver la televisión	0	0	2	24	26
Trabajar con el ordenador	3	9	20	28	60
TOTAL	**32**	**47**	**33**	**150**	**262**

Tabla 8. Resultados obtenidos del sistema de reconocimiento de actividades

ACTIVIDADES REALES \ ACTIVIDADES DETECTADAS	Dormirse	Despertarse	Vestirse	Entrar en casa	Salir de casa	Ducharse	Ir al baño	Cocinar	Ver la televisión	Trabajar con el ordenador	otras actividades
Dormirse	0.7	0	0	0	0	0	0	0	0	0	0.3
Despertarse	0	1	0	0	0	0	0	0	0	0	0
Vestirse	0	0	0.5	0	0	0	0	0	0	0	0.5
Entrar en casa	0	0	0	0.4	0.4	0	0	0	0	0	0.2
Salir de casa	0	0	0	0.2	0.7	0	0	0	0	0	0.1
Ducharse	0	0	0	0	0	0.5	0.2	0	0	0	0.3
Ir al baño	0	0	0	0	0	0	0.8	0	0	0	0.2
Cocinar	0	0	0	0	0	0	0	1	0	0	0
Ver la televisión	0	0	0	0	0	0	0	0	1	0	0
Trabajar con el ordenador	0	0	0	0	0	0	0	0	0	0.9	0.1
otras actividades	0	0	0	0	0.1	0	0.1	0.1	0	0.5	0

Tabla 9. Tabla de confusión de las actividades detectadas

Una tabla de confusión es una tabla en la que se contrastan las actividades detectadas por el algoritmo con las actividades reales realizadas por el usuario. Cada fila de la tabla representa el porcentaje de veces que una actividad ha sido realizada por el usuario, y cada columna representa el porcentaje de veces que una actividad se ha detectado. Por lo que cada casilla es un cálculo de las veces que se ha detectado la actividad de la columna a la que pertenece la casilla, mientras se estaba realizando la actividad de la fila correspondiente, dividido entre las veces que se ha realizado la actividad.

Se han tenido en cuenta para la elaboración de esta tabla las actividades detectadas fuera de la tolerancia de 15 minutos que se había establecido, y se han considerado como actividades válidas, a pesar de haberse detectado con un retardo mayor del deseado, debido a que el uso de esta tabla es interesante comprobar qué actividades son las que se confunden, y no tanto la precisión del espacio temporal en el que se detectan.

Por otro lado, se han considerado también las falsas detecciones como actividades que se han activado en lugar de la actividad real. Se reflejan en la tabla de confusión como “otras actividades”.

Se puede comprobar que, dentro del conjunto de actividades consideradas, es frecuente que las actividades de entrar y salir de casa se confundan entre sí, por esta razón es conveniente repasar las premisas y realizar ciertas modificaciones en el algoritmo.

En cuanto a las actividades “ducharse” e “ir al baño”, también es común que el algoritmo las confunda, pero este fenómeno se asocia a las medidas que se reciben del sensor de humedad. En este caso la modificación que se va a llevar a cabo es la de reducir ligeramente la sensibilidad del algoritmo a los cambios de humedad del baño, de forma que sea más fácil detectar un aumento de la humedad debido a que el usuario se está duchando.

Mientras se realizan las modificaciones y se examina el algoritmo de reconocimiento de actividades con las nuevas premisas, el sistema instalado en el domicilio del usuario sigue recogiendo datos y detectando actividades.

En la siguiente etapa, se vuelven a estudiar los resultados obtenidos por el algoritmo tras varias semanas y, si fuera necesario, se volverían a realizar modificaciones en las premisas. Este proceso se repite tantas veces como se estime necesario, ya que la rutina del usuario puede variar dependiendo de varios factores externos, como por ejemplo la estación del año o factores internos como el estado de salud.

En las semanas siguientes, se vuelven a estudiar los resultados del algoritmo y, de nuevo, se realiza la tabla de confusión que contrasta las actividades realizadas por el usuario con las actividades detectadas (Tabla 10).

Se puede observar que las modificaciones realizadas en el algoritmo han provocado un aumento en la precisión de las actividades de “entrar en casa” y “salir de casa”, aunque estos resultados todavía son mejorables.

Por otro lado, también se cambió la sensibilidad del algoritmo frente a los cambios de humedad en el baño, por lo que se observa una menor confusión entre las actividades “ducharse” e “ir al baño”, lo que ha provocado un aumento considerable en los rangos de acierto del sistema de reconocimiento para ambas actividades.

En el ámbito global del proyecto, estos resultados podrían considerarse válidos, pudiendo mantener el sistema en funcionamiento y realizando ciertas revisiones periódicas para comprobar que la rutina del usuario no varía demasiado, y la detección de actividades sigue funcionando correctamente. Sin embargo, en caso de que se quiera mejorar la identificación de algunas actividades que todavía tienen margen de mejora, como la actividad “vestirse”, tan solo habría que seguir tomando datos y así poder estudiar más a fondo la

rutina del usuario mientras se está realizando esta actividad. De nuevo se propondrían nuevas modificaciones del algoritmo y se comprobarían las mejoras experimentadas en los resultados.

ACTIVIDADES REALES \ ACTIVIDADES DETECTADAS	Dormirse	Despertarse	Vestirse	Entrar en casa	Salir de casa	Ducharse	Ir al baño	Cocinar	Ver la televisión	Trabajar con el ordenador	otras actividades
Dormirse	0.83	0	0	0	0	0	0	0	0	0	0.17
Despertarse	0	1	0	0	0	0	0	0	0	0	0
Vestirse	0	0	0.55	0	0	0	0	0	0	0	0.45
Entrar en casa	0	0	0	0.55	0.29	0	0	0	0	0	0.16
Salir de casa	0	0	0	0.18	0.72	0	0	0	0	0	0.11
Ducharse	0	0	0	0	0	0.67	0.14	0	0	0	0.19
Ir al baño	0	0	0	0	0	0	0.87	0	0	0	0.13
Cocinar	0	0	0	0	0	0	0	0.98	0	0	0.02
Ver la televisión	0	0	0	0	0	0	0	0	1	0	0
Trabajar con el ordenador	0	0	0	0	0	0	0	0	0	0.93	0.07
otras actividades	0.02	0.02	0.02	0.02	0.16	0.04	0.08	0.1	0.04	0.51	0

Tabla 10. Tabla de confusión de las actividades detectadas tras las modificaciones del algoritmo

6. Estudio económico

6.1. Recursos empleados

El resumen de los materiales utilizados durante la realización de este proyecto se pueden dividir dos clases: recursos de software y recursos de hardware. También se debe plantear la amortización del material utilizado con el objetivo de calcular el coste real y comprobar la viabilidad del proyecto.

En cuanto a los recursos de software:

- Sistema Operativo Raspbian
- Sistema Operativo Windows 10
- Spyder
- Zigbee2mqtt

En cuanto a los recursos de hardware:

- Raspberry Pi 3 modelo B+
- Sniffer CC2531
- Sensores de presencia
- Sensores de intensidad luminosa
- Sensores de vibración
- Botones ZigBee
- Sensores de humedad y temperatura
- Sensores de puertas y ventanas
- Sensores de consumo de potencia
- Botones
- Cable

6.2. Costes directos

En este apartado se estudiarán los costes de tipo amortizables de programas y equipo, costes de materiales y dispositivos directos y los costes de personal.

6.2.1. Costes amortizables

De cara al cálculo de este tipo de costes, hay que tener en cuenta la inversión total de los equipos para poder calcular la amortización lineal que corresponda, tal y como lo estipula la ley.

Para calcular el coste de amortización de los materiales utilizados en este proyecto, se debe considerar un tiempo de amortización de los equipos, que en este caso se estipulará en 5 años, que es el tiempo que se estima que tardarán en quedarse obsoletos.

Los costes de algunos elementos de software como el sistema operativo Raspbian o Spyder para la edición de programas en Python, no suponen ningún gasto de amortización, debido a que son gratuitos.

Por tanto, los únicos gastos que quedan por amortizar son los que se muestran en la Tabla 11.

MATERIAL	VALOR (APROX.) (€)	AMORTIZACIÓN (33%) (€)
Sistema operativo Windows 10	145	47.85
Ordenador Asus VivoBook S14	690	227.7
TOTAL	835	275.55

Tabla 11. Costes de amortización del proyecto

El gasto económico que suponen los equipos por cada hora de uso se puede calcular a partir de la siguiente fórmula:

$$Coste\ por\ hora = \frac{CosteTotal}{horas\ de\ uso\ anuales} = \frac{€}{h}$$

En el caso de este proyecto, se obtiene el siguiente resultado:

$$Coste\ por\ hora = \frac{275.55}{1762.12} = 0.1563 \frac{€}{h}$$

Teniendo en cuenta que el gasto total ocasionado durante la realización de este proyecto, estimando la cantidad de horas aplicada a este proyecto como la que se muestra en la Tabla 12.

ACTIVIDAD	HORAS DEDICADAS
Estudio y preparación previa	100
Desarrollo del proyecto	200
Elaboración de la memoria	100
TOTAL (horas)	400

Tabla 12. Desglose de horas dedicadas al proyecto

De esta forma, se ha calculado los costes de amortización totales como:

$$\boldsymbol{Coste\ de\ amortización\ = 0.1563 \frac{€}{h} * 400\ h = 62.52\ €}$$

6.2.2. Coste de materiales directos

Se incluirán los costes de todos los materiales que se han adquirido a lo largo de la realización del proyecto. Para ello, es necesario destacar la lista de todos esos elementos, que son los que se observan en la Tabla 13.

COMPONENTES	CANTIDAD	PRECIO UNITARIO (€)	PRECIO TOTAL
Raspberry Pi 3 modelo B+	1	39.34 €	39.34 €
Aqara Wireless switch	6	7.01 €	42.06 €
Aqara human body movement and illuminance sensor	2	12.99 €	25.98 €
Aqara door and window sensor	5	8.22 €	41.10 €
Aqara Smart Vibration Sensor	3	9.06 €	27.18 €
Aqara Temperature and humidity sensor	2	8.60 €	17.20 €
MiJia smart light sensor	2	6.70 €	13.40 €
Xiaomi ZigBee smart Socket	1	15.17 €	15.17 €
Sniffer CC2531	1	66.79 €	66.79 €
CC Debugger	1	41.52 €	41.52 €
Botón pulsador	6	0.78 €	4.69 €
Cables	20	0.34 €	6.70 €
		TOTAL	**341.13 €**

Tabla 13. Tabla de materiales utilizados

6.2.3. Coste del personal

Se estudiarán los costes del personal durante la realización del proyecto. Este personal está compuesto por un ingeniero electrónico de especificación en el ámbito de la mecatrónica, que ha sido el encargado de realizar las tareas de estudio previo y recopilación de bibliografía, búsqueda de soluciones y desarrollo del proyecto, desarrollo del software y despliegue de la red de sensores, instalación de dispositivos y la recopilación y el estudio de los datos.

El coste anual de un ingeniero electrónico se puede dividir en dos clases: el sueldo que cobra el ingeniero a lo largo de un año (sueldo en bruto), y el coste de cotización en la Seguridad Social (que según la ley es el 35% del sueldo en bruto). En la Tabla 14, se pueden observar estos costes divididos en estas categorías, lo que hace un total de 38.475 € de gastos anuales por el trabajo de un ingeniero electrónico.

Sueldo en bruto	28.500 €
Seguridad Social (35 %)	9.975 €
TOTAL	**38.475 €**

Tabla 14. Desglose del coste anual de un ingeniero electrónico

Al estar esta cifra referida a un año de trabajo, es necesario desglosar el número de horas laborales que componen un año. Esto se calcula contando el número de días que tiene un año y restando días no laborales, como días festivos, fines de semana, vacaciones, etc. Esto hace un total de 236 días.

Si una jornada completa de trabajo está definida por 8 horas de trabajo diarias (40 horas semanales), hace un total de 1.888 horas laborales en un año.

Si un ingeniero electrónico trabaja 1.888 horas en un año, y cuesta un total de 38.475 €:

$$Coste\ por\ hora = \frac{3.8475 \frac{€}{año}}{1.888 \frac{horas}{año}} = 20{,}379 \frac{€}{hora}$$

Según la Tabla 12, este proyecto ha supuesto un tiempo total de aproximadamente 400 horas, lo que supone un total de:

$$\boldsymbol{Coste\ de\ personal} = 20{,}379 \frac{€}{hora} * 400\ horas = \mathbf{8.151.48}\ €$$

6.2.4. Costes directos totales

$$\boldsymbol{Costes\ directos\ totales} = \mathbf{62.52}\ € + \mathbf{341.13}\ € + \mathbf{8.151.48}\ € = \mathbf{8.555{,}23}\ €$$

6.3. Costes indirectos

Este tipo de costes incluyen todos aquellos costes generales necesarios para la realización del proyecto, y cuyo cálculo se realiza como una estimación de los gastos administrativos, energéticos, directivos, etc. En la Tabla 15 se pueden observar los costes indirectos que ha supuesto la realización de este proyecto.

COSTE ADMINISTRATIVO	100 €
COSTE DE DESPLAZAMIENTO	320 €
COSTE ENERGÉTICO	220 €
TOTAL	**640 €**

Tabla 15. Costes indirectos totales

6.4. Costes totales

El resultado del coste total del proyecto, el cual se calcula como la suma de los resultados de los subapartados anteriores, como se observa en la Tabla 16, asciende a la cifra de.

COSTES DIRECTOS	8.555,23 €
COSTES INDIRECTOS	640 €
TOTAL	**9.195.23 €**

Tabla 16. Costes totales del proyecto

7. Conclusiones

7.1. Introducción

Una vez finalizado el desarrollo del proyecto, es necesario destacar las conclusiones que se han obtenido a lo largo de la realización del mismo, y contrastar los resultados con los objetivos propuestos al principio para así poder proponer posibles mejoras y vías futuras de desarrollo.

Si bien este trabajo puede suponer una mejora para vida de algunas personas, permitiendo al personal sanitario y social llevar un cierto control sobre la rutina del usuario, pero sin violar su privacidad. Ha nivel de desarrollo, ha supuesto un aumento de los conocimientos sobre distintos protocolos de comunicación muy populares, tanto a nivel doméstico como a nivel industrial. Esto ha sido posible gracias a una buena base en programación y el campo de Comunicaciones Industriales.

7.2. Conclusión y objetivos cumplidos

Tras haberse estudiado los resultados de la elaboración del proyecto, es necesario volver la vista atrás y contrastar, según el apartado 1.2, si este trabajo satisface los objetivos planteados al principio. A continuación, se hará una mención a cada uno de los objetivos cumplidos durante la realización de este trabajo, así como el valor añadido en alguno de ellos:

- Desplegar una red de sensores inalámbricos dentro de un domicilio de forma que estén distribuidos por toda la casa para poder reconocer el mayor número de actividades con el menor número de sensores.

  En la experimentación realizado con un usuario, se ha desplegado una red que incluye 15 sensores distribuidos entre las seis habitaciones de la casa. Estos sensores se comunican de forma inalámbrica utilizando ZigBee, que es un conjunto de protocolos de comunicación.

- El algoritmo de reconocimiento diseñado es capaz de identificar adecuadamente diez actividades: dormir, despertarse, ir al baño, ducharse, vestirse, entrar en casa, salir de casa, ver la televisión, trabajar con el ordenador y cocinar.

El algoritmo diseñado es capaz de indicar la fecha y hora de comienzo y de finalización de cada actividad, con una precisión de 15 minutos.

- Que el sistema desarrollado no comprometa la privacidad del usuario, evitando a toda costa aquellos dispositivos capaces de capturar o procesar imágenes o sonido. De la misma forma, los sensores deben ser discretos, procurando su instalación en lugares poco visibles y que no interrumpan o condicionen las actividades del usuario.

Esto se consigue gracias a los tipos de sensores utilizados en este trabajo, que son de detección de presencia PIR, de temperatura y de humedad, de vibración, de contacto para puertas y ventanas, de intensidad luminosa y de consumo de potencia.

Debido al tamaño reducido y a la versatilidad de estos sensores se ha podido garantizar que la instalación de los sensores dentro del domicilio sea discreta y poco visible.

- Que la red de sensores sea económica y flexible, permitiendo la conexión de múltiples dispositivos de distintos fabricantes.

Gracias al uso de la Raspberry Pi con el *sniffer* CC2531 como coordinador, es posible conectar distintos dispositivos de marcas variadas, permitiendo ampliar el rango de la red, si se desea, empleando otro CC2531 como repetidor.

- Que la red de sensores proporcione datos constantemente, recogiéndolos en un fichero de texto para su posterior tratamiento. El dato recogido por el coordinador debe identificar al sensor del que procede.

Con el software de zigbee2mqtt, por cada mensaje enviado por los sensores se publica un mensaje en el protocolo MQTT cuyo *topic* es el identificador del sensor del que procede.

Como valor añadido, el programa diseñado en Python traduce automáticamente el identificador de cada sensor por un código cada de distinguir también el tipo de sensor emisor del mensaje.

Todos estos datos sobre cada sensor incluido en la red se guardan en el fichero de texto "identifiers.dat", así como todos los datos recogidos por los sensores mientras el programa diseñado se está ejecutando se guardan en el fichero "data_file.dat", en un formato que facilita el tratamiento de estos datos.

- Que se diseñe un algoritmo de reconocimiento de actividades capaz de identificar las actividades realizadas por un usuario, una vez estudiada su rutina diaria, a partir de los datos recogidos por los sensores y registrados en el fichero de texto.

El algoritmo diseñado también recoge las actividades detectadas en tiempo real, guardando la información de cada actividad identificada en un fichero de texto junto con la fecha y hora de comienzo y de finalización: "actividades.dat". También existe la opción de leer los datos recogidos previamente y realizar una identificación *a posteriori.*

El comportamiento del algoritmo, tras el estudio previo y el etiquetado, muestra un porcentaje de reconocimiento global del 75%, que es el grado de acierto que tiene el sistema. El porcentaje de confusión global es del 2.6%, este número representa el porcentaje de veces que el sistema detecta una actividad cuando el usuario está realizando otra. Por último, está el porcentaje global de actividades sin clasificar, que son actividades que se han detectado, pero que no pertenecen a las actividades consideradas para este estudio. Representa el 22.4% de las detecciones realizadas por el sistema.

En cambio, tras haber realizado las modificaciones en el algoritmo en base al estudio posterior de los datos, se han obtenido un grado de reconocimiento global del 80.9%, un grado de confusión global del 2% y un grado global de actividades sin clasificar del 17.1%.

Se puede comprobar que, tras una etapa de estudio posterior y modificación de las premisas, el sistema ha mejorado el reconocimiento de

las actividades, disminuyendo el grado de confusión y de actividades no clasificadas.

- Aumentar la información proporcionada por la red para los casos en las que la detección de una actividad requiera información muy concreta. Como, por ejemplo, dormir y despertarse.

En este proyecto se ha modificado uno de los sensores de contacto para puertas y ventanas utilizados en la instalación de la red para obtener un sensor de presencia en cama. La información aportada por este sensor es determinante, ya que permite saber si la persona está tumbada o no en la cama y, en conjunto con los datos proporcionados por los demás sensores, puede identificarse las actividades de dormir y despertarse con mayor precisión.

7.3. Posibles vías futuras de desarrollo

De cara a mejorar este trabajo e investigar posibles líneas futuras de desarrollo, se proponen las siguientes mejoras:

- Sustitución del algoritmo de reconocimiento de actividades por una red neuronal: gracias al etiquetado de actividades realizado durante las primeras semanas de recogida de datos, que servirían para entrenar a la red, sería una implantación que mejoraría la precisión del sistema, pudiendo adaptarse a rutinas más diversas.

- Implementación de nuevas actividades más específicas: en este trabajo solo se tratan actividades básicas de la vida diaria, que están presentes en todo tipo de rutinas, sin embargo, con la adición de nuevos tipos de sensores a la red se podrían estudiar actividades como lavarse los dientes, realizar tareas de limpieza, por ejemplo.

- Adición de un sistema de posicionamiento doméstico a la red: se propone desarrollar un sistema que permita calcular la posición del usuario dentro de su domicilio mediante la detección de la señal de bluetooth dispositivos como móviles o pulseras de actividad. Así se

podría situar al usuario dentro de su domicilio y utilizar esa información adicional para abrir el camino hacia la detección de nuevas actividades.

- Integrar el sistema con Home Assistant: de esta forma, se podría abrir las puertas hacia nuevas posibilidades. Crear recordatorios periódicos y personalizados a la rutina de cada usuario podrían ser útiles, por ejemplo, para recordar tomarse las medicinas o proponer algún tipo de ejercicio diario como salir a pasear y otras medidas que pueden mejorar el estado de salud de la persona.

8. Bibliografía

[1] BOTIA, Juan A.; VILLA, Ana; PALMA, Jose. Ambient assisted living system for in-home monitoring of healthy independent elders. *Expert Systems with Applications*, 2012, vol. 39, no 9, p. 8136-8148.

[2] CHERNBUMROONG, Saisakul; CANG, Shuang; YU, Hongnian. A practical multi-sensor activity recognition system for home-based care. *decision support systems*, 2014, vol. 66, p. 61-70

[3] COPE, Steve, 2021. Beginners Guide To The MQTT Protocol. *Steve's Internet Guide* [online]. Disponible en: http://www.steves-internet-guide.com/mqtt-works/ [consulta: mayo 2021].

[4] CRUZ ROJA. Teleasistencia en casa [online]. Disponible en: https://www.cruzroja.es/principal/web/teleasistencia/teleasistencia-en-casa [consulta: mayo 2021]

[5] CUIDANIA 2018. Los 10 grandes problemas de las personas mayores [online] Disponible en: https://www.cuidania.com/problemas-de-las-personas-mayores/ [consulta: mayo 2021]

[6] GALLO ESTRADA, Julia; MOLINA MULA, Jesús; NOVAJRA, Alexandre Miquel y TALTAVULL APARICIO, Joana Maria. Estrategias de cuidados de las familias con las personas mayores que viven solas. *Index Enferm* [online]. Disponible en: http://scielo.isciii.es/scielo.php?script=sci_arttext&pid=S1132-12962013000100005&lng=es&nrm=iso ISSN 1699-5988 https://dx.doi.org/10.4321/S1132-12962013000100005. 2013, vol.22, n.1-2 [consulta: julio 2021], pp.20-24.

[7] GERIATRICAREA. 2017. Pautas para prevenir la aparición de enfermedades crónicas en personas mayores [online] Disponible en: https://www.geriatricarea.com/2017/08/02/pautas-para-prevenir-las-enfermedades-cronicas-en-personas-mayores/ [consulta: mayo 2021]

[8] GISLASON, Drew. *Zigbee wireless networking*. Newnes, 2008.

[9] Instituto Nacional de Estadística (INE). Disponible en: https://www.ine.es/ [consulta: junio 2021]

[10] Instructables Circuits 2021. *Zigbee Bed Presence Detector* [online] Disponible en: https://www.instructables.com/Zigbee-Bed-Presence-Detector/ [consulta: mayo 2021]

[11] JUST CHECKING [online] Disponible en: https://justchecking.co.uk/about-us [consulta: mayo 2021]

[12] KANTERS, Koen. *Zigbee2MQTT*. [online] Disponible en: https://www.zigbee2mqtt.io/ [consulta: abril 2021]

[13] LAGO, Paula; RONCANCIO, Claudia; JIMÉNEZ-GUARÍN, Claudia. Learning and managing context enriched behavior patterns in smart homes. *Future Generation Computer Systems*, 2019, vol. 91, p. 191-205.

[14] LUESIA, Fernando; MOREL, Miguel Ángel. *Estrategia de atención y protección social para las personas mayores en Aragon.* 2018

[15] MARTÍN MORENO, Javier, et al. ZigBee (IEEE 802.15. 4). 2007.

[16] PIERLEONI, Paola, et al. A versatile ankle-mounted fall detection device based on attitude heading systems. En *2014 IEEE Biomedical Circuits and Systems Conference (BioCAS) Proceedings*. IEEE, 2014. p. 153-156.

[17] POLLACK, Martha E., et al. Autominder: An intelligent cognitive orthotic system for people with memory impairment. *Robotics and autonomous systems*, 2003, vol. 44, no 3-4, p. 273-282.

[18] PRINCIPI, Emanuele, et al. An integrated system for voice command recognition and emergency detection based on audio signals. *Expert Systems with Applications*, 2015, vol. 42, no 13, p. 5668-5683.

[19] TAPIA, Emmanuel Munguia; INTILLE, Stephen S.; LARSON, Kent. Activity recognition in the home using simple and ubiquitous sensors. En *International conference on pervasive computing*. Springer, Berlin, Heidelberg, 2004. p. 158-175.

[20] Wan, J., O'grady, M. J., & O'Hare, G. M. (2015). Dynamic sensor event segmentation for real-time activity recognition in a smart home context. *Personal and Ubiquitous Computing*, 19(2), 287-301.

9. Anexos

9.1. Programa de reconocimiento de actividades

subscribe_write.py

```
"""
Escuela de Ingenierias Industriales
Universidad de Valladolid

TFG:
@author: Juan Alvarez Asensio
Sistema de reconocimiento de actividades para el cuidado de
personas en su entorno domiciliario

Julio 2021

Algoritmo de reconocimiento de actividades:

A cada parte de la casa se le asigna un numero:
    Entrada = 1 (C1)
    Dormitorio = 2 (C2, L2, M2, V2, B2)
    BaÃ±o1 = 3 (C3, T3, H3, L3)
    Cocina = 4 (C4, T4, H4, V4)
    SalÃ³n = 5 (P5)
    Despacho/Sala de trabajo = 6 (V6, M6)

En el fichero de identificacion de los sensores, aparecera una
columna que indicara en que zona de la casa se encuentra el sensor
para poder hacer el reconocimiento

existira una matriz mxn por cada tipo de sensor donde:
    m = numero de sensores de ese tipo
    n = numero de medidas que se quiere estudiar

los datos tomados por un mismo sensor se iran guardando dentro de
la misma fila de la matriz correspondiente a su tipo. Solo se
registraran las 5 ultimas medidas leidas por cada sensor.
Los sensores de temperatura y humedad regitraran las 10 ultimas
medidas.

El registro tendra estructura de cola, cuando llega un dato nuevo
se guarda en
la posicion 0 y el dato en la ultima posicion se elimina.
"""

import paho.mqtt.client as paho
import time
import re
from datetime import datetime

broker = "localhost"
broker_port = 1883
topic1 = "zigbee2mqtt/bridge/log"    # NEW SENSOR ACTIVATED
```

```
MESSAGES
topic2 = "zigbee2mqtt/+"                    # NEW ACTION MESSAGES

data_file = "/home/pi/Desktop/data_file.dat"
identifiers_file = "/home/pi/Desktop/identifiers.dat"
activity_file = "/home/pi/Desktop/actividades.dat"

# GLOBAL DATA
identifier_length = 18

#To read identifiers file
line = []

# SENSOR LISTS
switch_list = []
light_list = []
temperature_list = []
contact_list = []
pir_list = []
powerconsumption_list = []
vibration_list = []
bed_list = []

# Indices de cada tipo de sensor. El indice es el numero que
acompaÃ±a al sensor para su identificacion. C1
switch_index = []
light_index = []
temperature_index = []
contact_index = []
pir_index = []
powerconsumption_index = []
vibration_index = []
bed_index = []

# matriz de datos para cada tipo de sensor, que analizara el
algoritmo para detectar actividades
switch_data = []
light_data = []
temperature_data = []
humidity_data = []
contact_data = []
pir_data = []
powerconsumption_data = []
vibration_data = []
bed_data = []


# variables booleanas para detectar las actividades activas en
cada momento
dormir = False
vestirse = False
cocinar = False
television = False
trabajar = False
estar_dentro = True
ducharse = False
ir_al_bano = False

def on_connect(client, userdata, flags, rc):
```

```
        print("Connected with result code " + str(rc))
        print("UserData= " + str(userdata))
        print("flags= " + str(flags))
        print("")
        client.subscribe(topic1)
        client.subscribe(topic2)
        print("Suscrito: "+ topic1)
        print("Suscrito: "+ topic2)

    def on_message(client, userdata, message):
        time.sleep(0.1)
        if topic1 in message.topic: #this topic implies the loggin of
    a new sensor in the system
            if "interview_successful" in
    str(message.payload.decode("utf-8")):

    initialize_sensor_list(str(message.payload.decode("utf-8")))
        else:
            process_message(message.topic,
    str(message.payload.decode("utf-8")))

    def initialize_sensor_list(message): #extracts the identifier of
    the sensor and classifies it regarding the sensor type

        ident_index = message.index("friendly_name") +
    len("\"friendly_name\":")
        identifier = message[ident_index:ident_index +
    identifier_length]


        #creates a new definition of the sensors:
        """Aqara wirless switch --> S
        Mijia light intensity sensor --> L
        Aqara temperature, humidity and pressure sensor --> T
    (temperature data) H (humidity data)
        Aqara door and window contact sensor --> C
        Aqara human body movement and illuminance sensor --> M
        Xiaomi zigbee smart socket --> P
        Aqara smart vibration sensor --> V"""


        sensor_index = 1


        if "Aqara wireless switch" in message:
            if identifier not in switch_list:
                switch_list.append(identifier)
                for i in range(1, len(switch_list)+1):
                    if i not in switch_index:
                        sensor_index = i
                        break;
                switch_index.append(sensor_index)
                switch_data.append(['','','','',''])
                write_data_file(identifiers_file, identifier + "\tS" +
    str(sensor_index))

        elif "MiJia light intensity sensor" in message:
            if identifier not in light_list:
                light_list.append(identifier)
```

```
                for i in range(1, len(light_list)+1):
                    if i not in light_index:
                        sensor_index = i
                        break;
                light_index.append(sensor_index)
                light_data.append(['','','','',''])
                write_data_file(identifiers_file, identifier + "\tL" +
str(sensor_index))

    elif "Aqara temperature, humidity and pressure sensor" in
message:
        if identifier not in temperature_list:
            temperature_list.append(identifier)
            for i in range(1, len(temperature_list)+1):
                if i not in temperature_index:
                    sensor_index = i
                    break;
            temperature_index.append(sensor_index)

temperature_data.append(['','','','','','','','','',''])
            humidity_data.append(['','','','','','','','','',''])
            write_data_file(identifiers_file, identifier + "\tT" +
str(sensor_index))
            write_data_file(identifiers_file, identifier + "\tH" +
str(sensor_index))

    elif "Aqara door & window contact sensor" in message:
        if identifier not in contact_list:
            contact_list.append(identifier)
            for i in range(1, len(contact_list)+1):
                if i not in contact_index:
                    sensor_index = i
                    break;
            contact_index.append(sensor_index)
            contact_data.append(['','','','',''])
            write_data_file(identifiers_file, identifier + "\tC" +
str(sensor_index))

    elif "Aqara human body movement and illuminance sensor" in
message:
        if identifier not in pir_list:
            pir_list.append(identifier)
            for i in range(1, len(pir_list)+1):
                if i not in pir_index:
                    sensor_index = i
                    break;
            pir_index.append(sensor_index)
            pir_data.append(['','','','',''])
            write_data_file(identifiers_file, identifier + "\tM" +
str(sensor_index))

    elif "Xiaomi zigbee smart socket" in message:
        if identifier not in powerconsumption_list:
            powerconsumption_list.append(identifier)
            for i in range(1, len(powerconsumption_list)+1):
                if i not in powerconsumption_index:
                    sensor_index = i
                    break;
            powerconsumption_index.append(sensor_index)
```

```
            powerconsumption_data.append(['','','','',''])
            write_data_file(identifiers_file, identifier + "\tP" +
str(sensor_index))

    elif "Aqara vibration sensor" in message:
        if identifier not in vibration_list:
            vibration_list.append(identifier)
            for i in range(1, len(vibration_list)+1):
                if i not in vibration_index:
                    sensor_index = i
                    break;
            vibration_index.append(sensor_index)
            vibration_data.append(['','','','',''])
            write_data_file(identifiers_file, identifier + "\tV" +
str(sensor_index))


def removeprefix(text, prefix):
    if text.startswith(prefix):
        return text[len(prefix):]
    return text

def process_message(topic, message):
     #extract sensor identifier from the topic
     identifier = removeprefix(topic, "zigbee2mqtt/")
     identifier = identifier [0:identifier_length] #every
identifier has the exactly same length
     sensor_name = ""

     if identifier in switch_list:        #switch data format:
{action: 1(single)/2(double)/3(hold)/0(release)}
         sensor_name = "S" +
str(switch_index[switch_list.index(identifier)])
         info = busca_info(message, "action")
         if "single" in info:
             info = "1"
         elif "double" in info:
             info = "2"
         elif "hold" in info:
             info = "3"
         elif "release" in info:
             info = "0"
         else:
             info = ""

         if info != "":
             write_data_file(data_file, "\t" + sensor_name + "\t"
+ info)

     elif identifier in light_list:   #light data format:
{illuminance: }
         sensor_name = "L" +
str(light_index[light_list.index(identifier)])
         info = busca_info(message, "illuminance")
         if "null" not in info:
            write_data_file(data_file, "\t" + sensor_name + "\t" +
info)

     elif identifier in temperature_list:       #temperature data
```

```
format: {temperature: }, {humidity:}
            #temperature data
            info = busca_info(message, "temperature")
            sensor_name = "T" +
str(temperature_index[temperature_list.index(identifier)])
            write_data_file(data_file, "\t" + sensor_name + "\t" +
info)

            #humidity data
            info = busca_info(message, "humidity")
            sensor_name = "H" +
str(temperature_index[temperature_list.index(identifier)])
            write_data_file(data_file, "\t" + sensor_name + "\t" +
info)

        elif identifier in contact_list:            #contact data
format: {contact: 0/1}
            sensor_name = "C" +
str(contact_index[contact_list.index(identifier)])
            info = busca_info(message, "contact")
            contact = "0"
            if "true" in info:
                contact = "1"
            write_data_file(data_file, "\t" + sensor_name + "\t" +
contact)

        elif identifier in pir_list:                # pir data format:
{occupancy: 0/1}
            sensor_name = "M" +
str(pir_index[pir_list.index(identifier)])
            info = busca_info(message, "occupancy")
            occupancy = "0"
            if "true" in info:
                occupancy = "1"
            info = occupancy
            write_data_file(data_file, "\t" + sensor_name + "\t" +
info)

        elif identifier in powerconsumption_list:        # power data
format: {power: }
            sensor_name = "P" +
str(powerconsumption_index[powerconsumption_list.index(identifier)
])
            info = busca_info(message, "power")
            write_data_file(data_file, "\t" + sensor_name + "\t" +
info)

        elif identifier in vibration_list:        #vibration data
format: {action: 0 (null), 1 (vibration), 2 (tilt), 3 (drop)}
            sensor_name = "V" +
str(vibration_index[vibration_list.index(identifier)])
            info = busca_info(message, "action")
            if "vibration" in info:
                info = "1"
            elif "tilt" in info:
                info = "2"
            elif "drop" in info:
                info = "3"
            elif "null" in info:
```

```
                info = "0"
            else:
                info = ""

            if info != "":
                write_data_file(data_file, "\t" + sensor_name + "\t"
+ info)

        elif identifier in bed_list:            #contact data format:
{contact: 0/1}
            sensor_name = "B" +
str(bed_index[bed_list.index(identifier)])
            info = busca_info(message, "contact")
            contact = "0"
            if "true" in info:
                contact = "1"
            write_data_file(data_file, "\t" + sensor_name + "\t" +
contact)


def read_identifiers_file(file_name):
    with open(file_name, 'r') as file:
        for line in file:
            field = line.split()
            pos = int(field[3][1])

            if field[3][0] == 'S':
                switch_list.append(field[2])
                switch_index.append(pos)

            elif field[3][0] == 'L':
                light_list.append(field[2])
                light_index.append(pos)

            elif field[3][0] == 'T':
                temperature_list.append(field[2])
                temperature_index.append(pos)

            elif field[3][0] == 'C':
                contact_list.append(field[2])
                contact_index.append(pos)

            elif field[3][0] == 'M':
                pir_list.append(field[2])
                pir_index.append(pos)

            elif field[3][0] == 'P':
                powerconsumption_list.append(field[2])
                powerconsumption_index.append(pos)

            elif field[3][0] == 'V':
                vibration_list.append(field[2])
                vibration_index.append(pos)

            elif field[3][0] == 'B':
                bed_list.append(field[2])
                bed_index.append(pos)
```

```
def algorithm(last_sensor):
    global trabajar
    global dormir
    global vestirse
    global television
    global cocinar
    global estar_dentro
    global ducharse
    global ir_al_bano

    fecha = ''
    estado = ''
    actividad = ''

#================================================================
========
    # ACTIVIDAD: DORMIR Y DESPERTARSE

#================================================================
========
    if last_sensor == "L2" or last_sensor == "M2" or last_sensor
== "C2" or last_sensor == "B2":

        medidas_l2 = light_data[light_index.index(2)]
        ultima_l2 = medidas_l2[0].split()

        medidas_m2 = pir_data[pir_index.index(2)]
        ultima_m2 = medidas_m2[0].split()

        medidas_c2 = contact_data[contact_index.index(2)]
        ultima_c2 = medidas_c2[0].split()

        medidas_b2 = bed_data[bed_index.index(2)]
        ultima_b2 = medidas_b2[0].split()

        # tiene que haber al menos una medida de cada sensor
        if len(ultima_l2) == 0 or len(ultima_m2) == 0 or
len(ultima_c2) == 0 or len(ultima_b2) == 0:
            pass
        else:
            if ((not dormir) and ultima_l2[-1] == '0' and
ultima_m2[-1] == '1'
                and ultima_c2[-1] == '1' and ultima_b2[-1] == 1):
                dormir = True
                estar_dentro = True

                fecha = ultima_l2[0] + "\t" + ultima_l2[1] + "\t"
                estado = "actividad detectada"
                actividad = "dormir"


            elif (dormir and ((ultima_l2[-1] != '0' and
ultima_m2[-1] == '1')
                or (ultima_c2[-1] == '0' and ultima_b2[-1] == 0
))):
                dormir = False

                fecha = ultima_l2[0] + "\t" + ultima_l2[1] + "\t"
```

```
                estado = "actividad detectada"
                actividad = "despertarse"


#================================================================
========
    # ACTIVIDAD: VESTIRSE

#================================================================
========
    elif last_sensor == "V2" or last_sensor == "M2" or last_sensor
== "C2":

        medidas_v2 = vibration_data[vibration_index.index(2)]
        ultima_v2 = medidas_v2[0].split()

        medidas_m2 = pir_data[pir_index.index(2)]
        ultima_m2 = medidas_m2[0].split()

        medidas_c2 = contact_data[contact_index.index(2)]
        ultima_c2 = medidas_c2[0].split()

        # tiene que haber al menos una medida de cada sensor
        if len(ultima_v2) == 0 or len(ultima_m2) == 0 or
len(ultima_c2) == 0:
            pass

        else:
            if ((not vestirse) and ultima_v2[-1] != '0' and
ultima_m2[-1] == '1'
                and ultima_c2[-1] == '1'):
                vestirse = True
                estar_dentro = True


                fecha = ultima_c2[0] + "\t" + ultima_c2[1] + "\t"
                estado = "actividad iniciada"
                actividad = "vestirse"

            elif vestirse and (ultima_m2[-1] == '0' or ultima_c2[-
1] == '0'):
                vestirse = False


                fecha = ultima_c2[0] + "\t" + ultima_c2[1] + "\t"
                estado = "actividad finalizada"
                actividad = "vestirse"



#================================================================
========
    # ACTIVIDAD: DUCHARSE O IR AL BANO

#================================================================
========
    elif last_sensor == "C3"  or last_sensor == "H3" or
last_sensor == "L3":
```

```
            medidas_c3 = contact_data[contact_index.index(3)]
            ultima_c3 = medidas_c3[0].split()
            anterior_c3 = medidas_c3[1].split()

            medidas_l3 = light_data[light_index.index(3)]
            ultima_l3 = medidas_l3[0].split()

            medidas_h3 = humidity_data[temperature_index.index(3)]
            ultima_h3 = medidas_h3[0].split()

            # tiene que haber al menos una medida de cada sensor
            if len(ultima_h3) == 0 or len(ultima_c3) == 0 or
len(anterior_c3) == 0 or len(ultima_l3) == 0:
                pass
            else:
                # lista de los 9 ultimos valores de humedad sin contar
el primero
                humedad = []
                # valor medio de los datos de la lista
                media_hum = 0

                # se introducen los datos de humedad MENOS LA ULTIMA
MEDIDA
                for i in range(1, len(medidas_h3)):
                        aux = medidas_h3[i].split()
                        if len(aux) != 0:
                            humedad.append(float(aux[-1]))

                # se calcula la media sin tener en cuenta la humedad
actual
                if len(humedad) - 1 > 0:
                        media_hum = sum(humedad)/len(humedad)

                # se comprueban las condiciones para que se cumpla la
accion
                if len(humedad) == 9:
                    if (float(ultima_h3[-1]) - media_hum) > 5 and
ultima_c3[-1] == '1' and not ducharse:
                        ducharse = True

                        fecha = ultima_h3[0] + "\t" + ultima_h3[1] +
"\t"
                        estado = "actividad iniciada"
                        actividad = "ducharse"


                    elif ducharse and ultima_c3[-1] == '0':
                        ducharse = False

                        fecha = ultima_c3[0] + "\t" + ultima_c3[1] +
"\t"
                        estado = "actividad finalizada"
                        actividad = "ducharse"


                    elif (not (ducharse or ir_al_bano)) and
ultima_c3[-1] == '0' and anterior_c3[-1] == '1' and
(float(ultima_h3[-1]) - media_hum) < 5:
                        ir_al_bano = True
```

```
                    fecha = anterior_c3[0] + "\t" + anterior_c3[1]
+ "\t"
                    estado = "actividad iniciada"
                    actividad = "ir al bano"


                elif ir_al_bano and ultima_c3[-1] == '0' and
anterior_c3[-1] == '0':
                    ir_al_bano = False

                    fecha = anterior_c3[0] + "\t" + anterior_c3[1]
+ "\t"
                    estado = "actividad finalizada"
                    actividad = "ir al bano"



#=================================================================
========
    # ACTIVIDAD: COCINAR

#=================================================================
========
    elif last_sensor == "T4":
        medidas_t4 = temperature_data[temperature_index.index(4)]
        ultima_t4 = medidas_t4[0].split()

        # tiene que haber al menos una medida de cada sensor
        if len(ultima_t4) == 0:
            pass

        else:
            # lista de los 9 ultimos valores de temperaturas sin
contar el primero
            temperaturas = []
            # valor medio de las temperaturas de la lista
            media_temp = 0

            # se introducen los datos de temperatura MENOS LA
ULTIMA MEDIDA
            for i in range(1, len(medidas_t4)):
                aux = medidas_t4[i].split()
                if len(aux) != 0:
                    temperaturas.append(float(aux[-1]))

            # se calcula la media sin tener en cuenta la
temperatura actual
            if len(temperaturas) - 1 > 0:
                media_temp = sum(temperaturas)/len(temperaturas)

            # se comprueban las condiciones para que se cumpla la
accion
            if len(temperaturas) > 7:
                if (not cocinar) and float(ultima_t4[-1]) -
media_temp > 1:
                    cocinar = True

                    fecha = ultima_t4[0] + "\t" + ultima_t4[1] +
```

```
"\t"
                    estado = "actividad iniciada"
                    actividad = "cocinar"

                elif cocinar and float(ultima_t4[-1]) -
temperaturas[0] < -0.5:
                    cocinar = False

                    fecha = ultima_t4[0] + "\t" + ultima_t4[1] +
"\t"
                    estado = "actividad finalizada"
                    actividad = "cocinar"



#================================================================
========
    # ACTIVIDAD: ENTRAR O SALIR DE CASA

#================================================================
========
    elif last_sensor == "C1":
        medidas_c1 = contact_data[contact_index.index(1)]

        # es necesario saber las dos ultimas medidas
        ultima_c1 = medidas_c1[0].split()
        anterior_c1 = medidas_c1[1].split()

        # tiene que haber al menos una medida de cada sensor
        if len(ultima_c1) == 0 or len(anterior_c1) == 0:
            pass

        else:

            # tiene que abrirse la puerta (0) y luego cerrarse (1)
            if (ultima_c1[-1] == '1' and anterior_c1[-1] == '0'
                and (not estar_dentro)):
                estar_dentro = True


                fecha = anterior_c1[0] + "\t" + anterior_c1[1] +
"\t"
                estado = "actividad detectada"
                actividad = "entrar en casa"


            elif (ultima_c1[-1] == '1' and anterior_c1[-1] == '0'
                and estar_dentro):
                estar_dentro = False

                fecha = anterior_c1[0] + "\t" + anterior_c1[1] +
"\t"
                estado = "actividad detectada"
                actividad = "salir de casa"



#================================================================
========
```

```
        # ACTIVIDAD: VER LA TELEVISION

#================================================================
========
    elif last_sensor == "P5":
        medidas_p5 =
powerconsumption_data[powerconsumption_index.index(5)]
        ultima_p5 = medidas_p5[0].split()

        # tiene que haber al menos una medida de cada sensor
        if len(ultima_p5) == 0:
            pass

        else:
            # si el consumo detectado es mayor que un umbral (5)
            if (not television) and float(ultima_p5[-1]) > 5:
                television = True

                fecha = ultima_p5[0] + "\t" + ultima_p5[1] + "\t"
                estado = "actividad iniciada"
                actividad = "ver la television"

            elif television and float(ultima_p5[-1]) < 5:
                television = False

                fecha = ultima_p5[0] + "\t" + ultima_p5[1] + "\t"
                estado = "actividad finalizada"
                actividad = "ver la television"



#================================================================
========
    # ACTIVIDAD: TRABAJAR CON EL ORDENADOR

#================================================================
========
    elif last_sensor == "M6" or last_sensor == "V6":

        medidas_m6 = pir_data[pir_index.index(6)]
        ultima_m6 = medidas_m6[0].split()
        anterior_m6 = medidas_m6[1].split()

        medidas_v6 = vibration_data[vibration_index.index(6)]
        ultima_v6 = medidas_v6[0].split()

        # tiene que haber al menos una medida de cada sensor
        if len(ultima_m6) == 0 or len(ultima_v6) == 0:
            pass

        else:
            # Dos activaciones del PIR consecutivas a 0 para
garantizar que se
            # ha finalizado la actividad
            if (trabajar and ultima_m6[-1] == '0' and
anterior_m6[-1] == '0'
                and ultima_v6[-1] == '0'):
                trabajar = False
```

```
                    # Se conserva la fecha y hora del final como la de
la PRIMERA
                    # activacion

                    fecha = anterior_m6[0] + "\t" + anterior_m6[1] +
"\t"
                    estado = "actividad finalizada"
                    actividad = "trabajar con el ordenador"

                # Dos activaciones del PIR consecutivas a 1 para
evitar
                # activaciones accidentales del sensor
                elif ((not trabajar) and ((ultima_m6[-1] == '1'
                    and anterior_m6 == '1') or ultima_v6[-1] != '0')):
                    trabajar = True
                    estar_dentro = True

                    # Se conserva la fecha y hora del inicio como la
de la PRIMERA
                    # activacion
                    fecha = anterior_m6[0] + "\t" + anterior_m6[1] +
"\t"
                    estado = "actividad iniciada"
                    actividad = "trabajar con el ordenador"


    if actividad != '':
        write_activity_file(activity_file, fecha, estado,
actividad)


def read_data_file(file_name):
    with open(file_name, 'r') as file:
        for line in file:
            sensor = ''
            if "L" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("L") + 1])
                sensor = "L" + str(i)
                i = light_index.index(i)
                light_data[i].insert(0, line)
                light_data[i].pop()

            elif "T" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("T") + 1])
                sensor = "T" + str(i)
                i = temperature_index.index(i)
                temperature_data[i].insert(0, line)
                temperature_data[i].pop()

            elif "H" in line:
                # i --> fila dentro de la matriz donde se
```

```
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("H") + 1])
                sensor = "H" + str(i)
                i = temperature_index.index(i)
                humidity_data[i].insert(0, line)
                humidity_data[i].pop()

            elif "C" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("C") + 1])
                sensor = "C" + str(i)
                i = contact_index.index(i)
                contact_data[i].insert(0, line)
                contact_data[i].pop()

            elif "M" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("M") + 1])
                sensor = "M" + str(i)
                i = pir_index.index(i)
                pir_data[i].insert(0, line)
                pir_data[i].pop()

            elif "P" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("P") + 1])
                sensor = "P" + str(i)
                i = powerconsumption_index.index(i)
                powerconsumption_data[i].insert(0, line)
                powerconsumption_data[i].pop()

            elif "V" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("V") + 1])
                sensor = "V" + str(i)
                i = vibration_index.index(i)
                vibration_data[i].insert(0, line)
                vibration_data[i].pop()

            elif "B" in line:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(line[line.index("B") + 1])
```

```
                    sensor = "B" + str(i)
                    i = bed_index.index(i)
                    bed_data[i].insert(0, line)
                    bed_data[i].pop()


                if sensor != '':
                    algorithm(sensor)


def write_activity_file(name, fecha, estado, actividad):
    try:
        with open(name, 'a') as file:
            file.write(fecha + "\t" + estado + "\t" + actividad +
"\n")
    except FileNotFoundError as error:
        raise FileNotFoundError(error)


def write_data_file(name, message):
    now = datetime.now()
    try:
        with open(name, 'a') as file:
            file.write(str(now) + "\t" + message + '\n')

            # se guardan los datos en la matriz correspondiente y
se ejecuta el algoritmo
            sensor = ''
            if "L" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("L") + 1])
                sensor = "L" + str(i)
                i = light_index.index(i)
                light_data[i].insert(0, str(now) + "\t" + message
+ '\n')
                light_data[i].pop()

            elif "T" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("T") + 1])
                sensor = "T" + str(i)
                i = temperature_index.index(i)
                temperature_data[i].insert(0, str(now) + "\t" +
message + '\n')
                temperature_data[i].pop()

            elif "H" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("H") + 1])
                sensor = "H" + str(i)
```

```
                i = temperature_index.index(i)
                humidity_data[i].insert(0, str(now) + "\t" +
message + '\n')
                humidity_data[i].pop()

            elif "C" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("C") + 1])
                sensor = "C" + str(i)
                i = contact_index.index(i)
                contact_data[i].insert(0, str(now) + "\t" +
message + '\n')
                contact_data[i].pop()

            elif "M" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("M") + 1])
                sensor = "M" + str(i)
                i = pir_index.index(i)
                pir_data[i].insert(0, str(now) + "\t" + message +
'\n')
                pir_data[i].pop()

            elif "P" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("P") + 1])
                sensor = "P" + str(i)
                i = powerconsumption_index.index(i)
                powerconsumption_data[i].insert(0, str(now) + "\t"
+ message + '\n')
                powerconsumption_data[i].pop()

            elif "V" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("V") + 1])
                sensor = "V" + str(i)
                i = vibration_index.index(i)
                vibration_data[i].insert(0, str(now) + "\t" +
message + '\n')
                vibration_data[i].pop()

            elif "B" in message:
                # i --> fila dentro de la matriz donde se
encuentran las
                # medidas del sensor correspondiente a la medida
actual
                i = int(message[message.index("B") + 1])
```

```
                    sensor = "B" + str(i)
                    i = bed_index.index(i)
                    bed_data[i].insert(0, str(now) + "\t" + message +
'\n')
                    bed_data[i].pop()


                if sensor != '':
                    algorithm(sensor)

        except FileNotFoundError as error:
            raise FileNotFoundError(error)


def busca_info(message, value):
    pos1 = message.find(value) + len(value) + 2
    pos2 = message.find(",", pos1)

    return message[pos1:pos2]



print("reading identifiers...")
read_identifiers_file(identifiers_file)

# Se crea una matriz para cada tipo de sensor con los Ã°ltimos
datos registrados
for i in range(len(light_index)):
    light_data.append(['', '', '', '', ''])

# los sensores de temperatura y humedad guardarÃ¡n los Ã°ltimos 10
datos
for i in range(len(temperature_index)):
    temperature_data.append(['', '', '', '', '','','','','','',''])

for i in range(len(temperature_index)):
    humidity_data.append(['', '', '', '', '','','','','','',''])

for i in range(len(contact_index)):
    contact_data.append(['', '', '', '', ''])

for i in range(len(pir_index)):
    pir_data.append(['', '', '', '', ''])

for i in range(len(powerconsumption_index)):
    powerconsumption_data.append(['', '', '', '', ''])

for i in range(len(vibration_index)):
    vibration_data.append(['', '', '', '', ''])

for i in range(len(bed_index)):
    bed_data.append(['', '', '', '', ''])

print("reading data...")
read_data_file(data_file)

client = paho.Client('Cliente1', userdata = "UsuarioPrueba")
client.on_connect = on_connect
```

```
client.on_message = on_message

client.connect(broker, broker_port, 60)

print("loop started successfully")
#start loop to process received messages
client.loop_forever()

client.disconnect()
client.loop_stop()
```