Daniil Nikiforov

MONITORIZACIÓN DE PERSONAS DE TERCERA EDAD EN SU DOMICILIO

TRABAJO DE FIN DE GRADO

dirigido por Maria dels Àngels Moncusí Mercadé

Grado de Ingeniería Informática

UNIVERSITAT ROVIRA I VIRGILI

TARRAGONA

2024

Resumen.

Este proyecto aborda la necesidad de mejorar la calidad de vida y autonomía de las personas en su domicilio.Nos centraremos en el estudio económico con diferentes puntos de abordaje para implementar el sistema completo, tanto sensores, placas y diferentes herramientas, con el objetivo de obtener datos y analizarlos más adelante.

En cuanto al factor técnico, podremos obtener inicialmente algunos patrones de movilidad de los usuarios en diferentes horas y condiciones climatológicas, lo que nos permitiría detectar posibles problemas de seguridad o salud. Esto se realizará mediante una investigación sobre diferentes tipos de sensores en base a las necesidades ocasionadas, además, se ha desarrollado una simulación con la herramienta de Unity para poder visualizar el comportamiento de el sistema.

Finalmente, el proyecto incluye un pequeño análisis de costes de los diferentes sensores, placas y diferentes herramientas a utilizar, así como la evaluación de los protocolos de comunicación que se consideren más adecuados, donde se tendrán en cuenta algunos puntos del tratamientos de datos personales como la seguridad del entorno a desarrollar.

Resum.

Aquest projecte està adreçat a la necessitat de millorar la qualitat de vida i autonomia de les persones al seu domicili. Ens centrarem en l'estudi econòmic amb diferents punts d'abordatge per implementar el sistema complet, tant sensors, plaques com diferents eines, amb l'objectiu d'obtenir dades i poder analitzar-les més endavant.

Des d'un punt de vista tècnic, podrem identificar patrons de mobilitat dels usuaris en diferents hores i condicions climàtiques, cosa que ens permetra detectar possibles problemes de seguretat o salut. Això es realitzarà mitjançant una recerca dels diferents tipus de sensors per a cobrir les necessitats dels usuaris, a més, s'ha desenvolupat una simulació amb l'eina Unity per poder visualitzar el comportament del sistema.

Finalment, el projecte inclou una petit anàlisi de costos dels diferents sensors, plaques i diferents eines a utilitzar, així com l'avaluació dels protocols de comunicació que es considerin més adequats, on es tindran en compte alguns punts del tractament de dades personals com la seguretat de l'entorn a desenvolupar.

Abstract.

This project proposes the development and implementation of a system to enhance the quality of life and autonomy of people in their homes. We will focus on the economic study with different approaches to implement the complete system, including sensors, boards, and various tools, with the goal of obtaining data and analyzing it later.

The technical approach will focus on identifying patterns of user mobility at different times and under different circumstances, which could allow us to detect potential safety or health issues. This data will be done through research on different types of sensors based on the needs that arise. Additionally, a simulation has been developed using the Unity tool to visualize the system's behavior.

Furthermore, the project will include a through cost analysis of the different sensors, boards, and tools to be used, as well as an evaluation of the most suitable communication protocols, taking into account certain aspects of personal data processing, such as the security of the environment to be developed.

Índice

Índice de Figuras

Índice de Tablas

Índice de Código Fuente

1. Introducción

El envejecimiento de la población mundial es un fenómeno demográfico de gran magnitud que está transformando las sociedades actuales. Según datos de la *Organización Mundial de la Salud (OMS)*, se estima que para el año 2050, la población mundial de 60 años o más alcanzará los 2.100 millones de personas, lo que representa un aumento significativo en comparación con los 962 millones de personas que había en el año 2017.

Este envejecimiento poblacional tiene importantes implicaciones sociales, económicas y sanitarias. En el ámbito social, genera un aumento de la demanda de servicios de atención a las personas mayores, tanto en el ámbito institucional como en el domiciliario. En el ámbito económico, supone un reto para los sistemas de pensiones y seguridad social, y también para el mercado laboral, ya que se reduce la población activa y aumenta la población dependiente. En el ámbito sanitario, implica un aumento de la prevalencia de enfermedades crónicas y discapacidades, lo que requiere una mayor inversión en atención médica y preventiva.

En este contexto, la autonomía del hogar se convierte en una aspiración cada vez más relevante para las personas mayores. Vivir de forma independiente en su propio hogar les permite mantener su independencia, calidad de vida y dignidad, y les evita la necesidad de institucionalizarse o depender de terceros para realizar las actividades cotidianas.

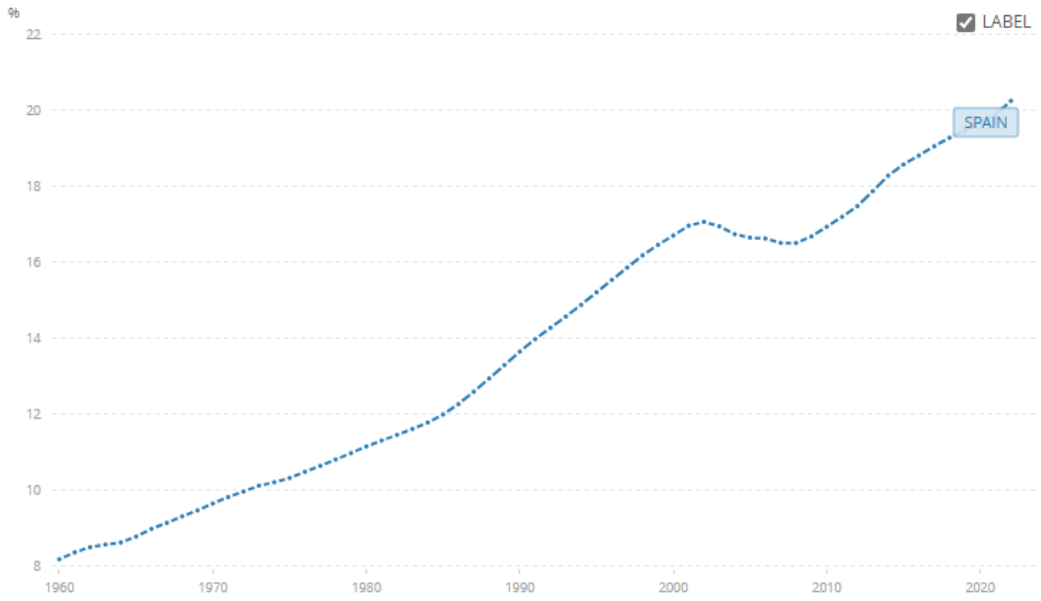

Figura 1: Porcentaje de mayores de 65 años en España

En el gráfico anterior se muestra el crecimiento porcentual de las personas mayores de 65 años en España desde el 1960 al 2020. Donde se puede apreciar que se ha incrementado de un 8.19 % a un 20.27 %, lo que demuestra un aumento en el envejecimiento de la población, una tendencia que coincide con algunos otros países industrializados.

1.1. Motivaciones

1.1.1. Requerimientos de los adultos mayores en la independencia personal

Las personas mayores que desean vivir de forma autónoma en sus hogares tienen diversas necesidades que abarcan aspectos físicos, cognitivos y emocionales:

Aspectos físicos:

- **Movilidad reducida:** Es la dificultad para caminar, subir escaleras o realizar tareas que requieran esfuerzo físico, lo que aumenta el riesgo a caídas y/o accidentes dentro del hogar.
- **Deterioro físico:** Enfermedades crónicas como la diabetes, hipertensión o enfermedades cardíacas, requerirán un control y seguimiento constante, lo que puede ser difícil de manejar para las personas que vivan en la soledad de sus domicilios.
- **Disminución de la capacidad funcional:** La pérdida de fuerza, equilibrio y coordinación puede dificultar tareas cotidianas como cocinar, limpiar o vestirse, lo que aumenta la dependencia hacia terceras personas o asistencias.

Aspectos cognitivos:

- **Deterioro cognitivo leve:** Olvidos frecuentes, dificultad para concentrarse o problemas para realizar tareas complejas pueden llegar a afectar a la capacidad de vivir de forma independiente.
- **Demencia:** Enfermedades como el Alzheimer o la demencia pueden afectar gravemente a la memoria, juicio o capacidad de realizar tareas cotidianas.
- **Depresión:** La soledad, aislamiento y falta de estimulación mental pueden aumentar el riesgo de depresión, lo que afecta negativamente a la calidad de vida.

Aspectos sociales y emocionales:

- **Aislamiento social:** La falta de interacción social puede llegar a afectar negativamente la salud mental y emocional de las personas mayores, aumentando el riesgo de depresión y soledad.
- **Falta de apoyo emocional:** La ausencia de familiares cuidadores cercanos puede dificultarla gestión de tareas cotidianas y la atención médica.
- **Dependencia emocional:** La necesidad de depender de otros para las actividades diarias puede generar sentimientos de frustración, pérdida de autonomía y baja autoestima.

1.1.2. Restricciones de las alternativas actuales

Las soluciones tecnológicas actuales para la vida autónoma de las personas mayores tienen algunas limitaciones que las hacen insuficientes para abordar todas sus necesidades de manera integral y efectiva.

- **Falta de integración:** Las soluciones existentes suelen centrarse en un aspecto específico de la vida autónoma, como la seguridad del hogar o el control médico remoto. Sin embargo, esta fragmentación en la atención impide una visión integral de las necesidades multidimensionales de las personas mayores. Esto puede generar dificultades para identificar patrones o riesgos que podrían efectuar su salud y bienestar general. Por ejemplo, un sistema de seguridad del hogar puede alertar sobre una caída, pero no proporciona información sobre el estado general de las persona mayor o las posibles causas de la caída.

- **Complejidad de uso:** Algunas soluciones tecnológicas presentan interfaces complejas, configuraciones manuales y falta de instrucciones claras, lo que dificulta su uso por parte de personas mayores con poca experiencia tecnológica. Esta barrera puede generar frustración, reducir la adopción y limitar el impacto positivo de estas soluciones podrían tener en la vida de las personas mayores. Un estudio realizado por la Universidad de Stanford, reveló que el 72 % de las personas mayores encuestadas experimentan dificultades para usar tecnologías de asistencia en el hogar debido a su complejidad.

- **Falta de personalización:** Las soluciones no siempre se adaptan a las necesidades individuales de cada persona mayor. No consideran aspectos como las capacidades físicas, las limitaciones cognitivas, las preferencias personales o el estilo de vida, lo que puede limitar su eficacia y reducir la satisfacción de los usuarios. Por ejemplo, si un sistema de monitorización del sueño que no considera las características individuales del sueño de cada persona mayor podría generar datos irrelevantes o inexactos, dificultando la detección de problemas de salud.

- **Acceso limitado:** El costo elevado de algunas soluciones tecnológicas, la falta de acceso a internet en zonas rurales y la brecha digital existente entre las personas mayores, limitan la disponibilidad de estas herramientas para todos los sectores de la población mayor. Esto genera desigualdades en el acceso a tecnologías que podrían mejorar la calidad de vida de las personas mayores. Según un informe de la Comisión Europea, el 32 % de las personas mayores de 65 años en la Unión Europea no tiene acceso a internet.

1.1.3. Propuesta de valor

Para superar las limitaciones de las soluciones existentes y ofrecer un valor diferenciador a las personas mayores, este proyecto propone un sistema tecnológico integral que integra las siguientes características:

- **Enfoque holístico:** El sistema integra soluciones para la seguridad en el hogar, el soporte social y emocional, el control médico remoto, la monitorización ambiental y otros aspectos relevantes para la vida autónoma. Aborda las necesidades multidimensionales de las personas mayores de manera integral, proporcionando una visión completa de su salud y bienestar. Por ejemplo, el sistema puede integrar datos de sensores de movimiento para detectar caídas, datos de un sistema de monitorización del sueño para evaluar la calidad del sueño y datos de un dispositivo de seguimiento de actividad física para analizar el nivel de actividad física de la persona mayor. Esta información combinada permite identificar patrones y riesgos de manera más efectiva.
- **Diseño intuitivo:** La interfaz del sistema es fácil de usar y comprender, incluso para personas mayores con poca experiencia tecnológica. Se utilizan iconos claros, menús simples y lenguaje sencillo para facilitar la navegación y la interacción con el sistema. Además, se implementan mecanismos de ayuda contextual y tutoriales interactivos para apoyar a los usuarios en el proceso de aprendizaje.
- **Personalización:** El sistema se adapta a las necesidades individuales de cada persona mayor. Considera las capacidades físicas, las limitaciones cognitivas, las preferencias personales y el estilo de vida para ofrecer una experiencia personalizada y efectiva. El sistema puede aprender de las interacciones del usuario y ajustar su comportamiento en consecuencia.
- **Accesibilidad:** El sistema es accesible y asequible para todos los sectores de la población mayor, independientemente de su situación económica o ubicación geográfica. Se implementan estrategias para reducir costos, facilitar el acceso a internet y brindar soporte técnico adecuado. Además, se consideran las necesidades de personas con discapacidades visuales, auditivas o motoras en el diseño de la interfaz y la interacción con el sistema.

El proyecto no solo ofrece tecnología, sino que también propone un enfoque integral que incluye la participación activa de las personas mayores en el diseño, implementación y evaluación del sistema. Se fomentan la retroalimentación continua y la colaboración para garantizar que el sistema responda a las necesidades y expectativas reales de los usuarios.

Se realizan pruebas de usabilidad con personas mayores en diferentes etapas del desarrollo del proyecto para identificar posibles problemas y mejorar la experiencia del usuario.

Al combinar tecnología, apoyo social y atención médica adecuada, este proyecto tiene el potencial de transformar la vida de las personas mayores, permitiéndoles vivir de forma independiente, segura y saludable en sus propios hogares.

1.2. Objetivos

El propósito principal de este proyecto es desarrollar un sistema integral, asequible y personalizado para apoyar la vida autónoma de las personas mayores, utilizando placas Arduino y una variedad de sensores para recopilar datos relevantes de la vivienda. Estos datos se utilizarán para realizar diversos análisis y representaciones gráficas, proporcionando información valiosa para mejorar la calidad de vida, la seguridad y el bienestar de las personas mayores. Además, se creará una simulación en *Unity* para obtener datos que se asemejen lo máximo posible a los datos reales.

1.2.1. Objetivos generales

Desarrollar un sistema asequible y accesible

- **Reducción de costos:** Implementar placas Arduino y sensores de bajo costo para la recolección de datos, minimizando los gastos asociados a soluciones tecnológicas tradicionales.
- **Facilidad de uso:** Diseñar un sistema intuitivo y fácil de usar, considerando las habilidades y preferencias de las personas mayores.
- **Accesibilidad universal:** Garantizar la accesibilidad del sistema para personas con diferentes capacidades físicas o cognitivas, incluyendo opciones de interacción alternativas (voz, gestos, etc.).

Recopilación y análisis exhaustivo de datos:

- **Amplia gama de sensores:** Implementar una variedad de sensores para recopilar datos relevantes del entorno hogareño, incluyendo:
  - **Factores ambientales:** Temperatura, humedad, luminosidad, calidad del aire, etc.
  - **Actividad física:** Movimiento, patrones de sueño, caídas, etc.
  - **Comportamiento:** Hábitos diarios, rutinas, interacciones con el entorno, etc.
  - **Datos biométricos:** Frecuencia cardíaca, presión arterial, niveles de glucosa, etc. (con consentimiento explícito y medidas de seguridad adecuadas).

- **Simulación en Unity:** Crear una simulación en Unity para recrear escenarios realistas del hogar y obtener datos que se asemejen a los datos reales. Esto permitirá:
  - **Validar la ubicación de los sensores:** Optimizar la colocación de los sensores para garantizar una recolección precisa y eficiente de datos.
  - **Simular actividades cotidianas:** Reproducir actividades típicas de las personas mayores para analizar patrones de movimiento y generar información contextual.
  - **Testar el sistema de análisis:** Probar el software de análisis de datos con datos simulados para identificar patrones y tendencias en un entorno controlado.
  - **Refinar la interfaz de usuario:** Evaluar y optimizar la interfaz de usuario en base a la interacción con el entorno simulado, asegurando una experiencia intuitiva y fácil de usar.

- **Análisis de datos avanzado:** Implementar algoritmos de análisis de datos complejos para:
  - **Identificar patrones y tendencias:** Detectar comportamientos habituales, cambios en la rutina y posibles riesgos para la salud o seguridad de las personas mayores.
  - **Predicción de riesgos:** Anticipar posibles situaciones de riesgo, como caídas, incendios o fugas de gas, permitiendo la intervención temprana.
  - **Personalización del sistema:** Adaptar el sistema a las necesidades y preferencias individuales de cada persona mayor, proporcionando recomendaciones y asistencia personalizadas.

Mejora integral de la calidad de vida:

- **Promoción de la independencia:** Facilitar que las personas mayores vivan en sus propios hogares por más tiempo, manteniendo su autonomía y control sobre su vida diaria.
- **Prevención de riesgos:** Detectar y prevenir posibles riesgos en el hogar, como caídas, accidentes domésticos o problemas de salud, mejorando la seguridad y bienestar de las personas mayores.
- **Monitoreo de la salud:** Brindar información valiosa sobre el estado de salud y actividad física de las personas mayores, permitiendo una detección temprana de problemas de salud y una intervención oportuna.
- **Optimización del entorno hogareño:** Crear un entorno hogareño más seguro, cómodo y adaptado a las necesidades específicas de cada persona mayor, mejorando su calidad de vida en general.
- **Soporte emocional y social:** Facilitar la comunicación y conexión con familiares, amigos y servicios de apoyo, combatiendo el aislamiento social y mejorando el bienestar emocional de las personas mayores.

1.2.2. Objetivos específicos

Implementación de sensores:

- **Selección y adquisición de sensores:** Identificar y seleccionar los sensores más adecuados en función de las necesidades específicas del proyecto y las características del entorno hogareño.
- **Instalación y configuración de sensores:** Instalar los sensores en ubicaciones estratégicas dentro del hogar, garantizando una cobertura adecuada y la recolección precisa de datos.
- **Calibración y mantenimiento de sensores:** Calibrar periódicamente los sensores para asegurar la precisión de las mediciones y realizar el mantenimiento preventivo necesario.

Integración con Arduino:

- **Programación de placas Arduino:** Desarrollar software embebido para las placas Arduino que permita la comunicación con los sensores, la adquisición de datos y la transmisión de información a la unidad central.
- **Gestión de la comunicación:** Implementar protocolos de comunicación eficientes para la transmisión de datos entre las placas Arduino y la unidad central, minimizando la latencia y maximizando la confiabilidad.
- **Monitoreo y control del sistema:** Desarrollar herramientas de monitoreo para supervisar el estado del sistema, detectar posibles fallos y realizar acciones de control cuando sea necesario.

1.3. Metodología

La metodología de este proyecto se basa en un enfoque iterativo y centrado en el usuario, que combina la investigación, el desarrollo, la implementación y la evaluación en un proceso continuo de mejora. Se ha utilizado GitHub para el control de versiones del código fuente y para documentar un diario de las actividades realizadas cada día.

1.3.1. Descripción de la investigación

- **Investigación Preliminar:** Se realizó una investigación exhaustiva sobre el envejecimiento de la población, las necesidades de las personas mayores y las soluciones tecnológicas existentes. Esta investigación proporcionó una comprensión profunda de los desafíos y oportunidades en el campo de la vida autónoma para las personas mayores.
- **Desarrollo de la Propuesta de Valor:** Basándose en los hallazgos de la investigación preliminar, se desarrolló una propuesta de valor que aborda las necesidades de las personas mayores de manera integral y efectiva.
- **Diseño e Implementación del Sistema:** Se diseñó e implementó un sistema basado en placas Arduino y una variedad de sensores. Este sistema recopila datos relevantes de la vivienda y los utiliza para realizar diversos análisis y representaciones gráficas.
- **Simulación en Unity:** Se creó una simulación en Unity para obtener datos que se asemejen lo máximo posible a los datos reales. Esta simulación recrea un domicilio a escala real y permite probar y optimizar el sistema en un entorno controlado.
- **Evaluación y Mejora:** Se realizan pruebas de usabilidad con personas mayores en diferentes etapas del desarrollo del proyecto para identificar posibles problemas y mejorar la experiencia del usuario. La retroalimentación de los usuarios se utiliza para iterar y mejorar el sistema.

1.3.2. Fases del proyecto

Entre las diferentes fases que se detallen a continuación, también se puede visualizar mediante la figura 20 un diagrama de Gantt donde se ha ido añadiendo el tiempo dedicado en cada una de las tareas junto a dependencias entre ellas.

Fase de investigación preliminar:

- **Revisión web exhaustiva:** Se realiza una revisión web exhaustiva de estudios académicos, informes gubernamentales y publicaciones relevantes sobre el envejecimiento de la población, la soledad en las personas mayores, las necesidades específicas de este grupo demográfico y las soluciones tecnológicas existentes en el ámbito de la vida autónoma. Para esto, se llevaron a cabo varias etapas para lograr una calidad y relevancia de información adecuada al proyecto, estas son las etapas principales:
    - **Identificación de fuentes:** La utilización de bases de datos académicos como *PubMed*, *Web of Science* y *Scopus*. Por otro lado, se consultaron publicaciones de diferentes organizaciones internacionales, como puede ser la *OMS*.

  - **Selección de términos de búsqueda:** Para una investigación detallada se ha tenido que filtrar por términos, como puede ser el ënvejecimiento de la población, ̈soledad en personas mayores, ̈sensores, entre otros.
  - **Evaluación de la relevancia:** Al revisar títulos, resúmenes y palabras clave de los documentos analizados se ha tenido que determinar su pertinencia respecto al tema de investigación.

- **Análisis del entorno hogareño:** Se evalúan las características del entorno hogareño típico de las personas mayores, considerando factores como el diseño, la distribución espacial, la accesibilidad, los elementos de seguridad y los posibles riesgos asociados a la soledad.

Fase de Diseño y Desarrollo:

- **Definición de requisitos del sistema:** Establecer especificaciones técnicas detalladas para el sistema, incluyendo las funcionalidades, la arquitectura, los componentes de hardware y software, y los estándares de calidad y seguridad.
- **Diseño del sistema:** Diseñar la arquitectura general del sistema, incluyendo la interacción entre los sensores, las placas Arduino, la unidad central, la interfaz de usuario y la base de datos. Desarrollo de software: Implementar el software embebido para las placas Arduino, el software de análisis de datos, la interfaz de usuario y las herramientas de administración del sistema.
- **Selección y configuración de sensores:** Elegir los sensores más adecuados en función de las necesidades específicas del proyecto, las características del entorno hogareño y las capacidades tecnológicas disponibles.
- **Integración con Arduino:** Conectar los sensores a las placas Arduino y desarrollar el software necesario para la comunicación, la adquisición de datos y la transmisión de información.
- **Creación de la simulación en Unity:** Diseñar y desarrollar una simulación en Unity que recree un entorno hogareño realista, permitiendo probar el sistema en un entorno controlado y optimizar la ubicación de los sensores, el análisis de datos y la interfaz de usuario.

Fase de Implementación y Pruebas:

- **Implementación del servidor:** Establecer un servidor para recibir y almacenar los datos enviados por los sensores en base al identificador del usuario y vivienda. Se implementarán protocolos de comunicación eficientes para minorizar el consumo y mejorar el rendimiento del sistema.
- **Monitoreo y seguimiento:** Monitorizar continuamente el rendimiento del sistema en el entorno simulado, recopilar datos de uso y analizar posibles incidencias o errores para realizar las correcciones necesarias.

2. Marco teórico

El envejecimiento es un proceso natural, complejo y polifacético que abarca cambios biológicos, psicosociales y ambientales. Si bien el texto original ofrece una visión general de las características del envejecimiento, es importante profundizar en estos aspectos para comprender mejor los desafíos y oportunidades que enfrentan las personas mayores en su búsqueda de una vida autónoma.

2.1. Envejecimiento y vida autónoma

2.1.1. Características del envejecimiento

Cambios Biológicos

- **Envejecimiento Celular:** A nivel celular, se observa una disminución en la tasa de renovación celular. Esto se asocia con un mayor riesgo de enfermedades relacionadas con la edad.
- **Cambios en el Sistema Inmunológico:** El sistema inmunológico se vuelve menos eficiente con la edad, lo que aumenta el riesgo a infecciones, enfermedades autoinmunes y algunos tipos de cáncer.
- **Deterioro del Sistema Óseo:** La pérdida de densidad ósea es un proceso natural del envejecimiento que aumenta el riesgo de osteoporosis y fracturas, especialmente en mujeres, que por estadística son las que más años viven.
- **Alteraciones en el Sistema Cardiovascular:** La presión arterial tiende a aumentar con la edad, al igual que la rigidez arterial, lo que incrementa el riesgo de enfermedades cardíacas.
- **Declive Cognitivo:** Se observa una disminución gradual en algunas funciones cognitivas, como la memoria, la atención, la velocidad de procesamiento y la flexibilidad mental.

Aspectos Psicosociales

- **Jubilación y Transiciones Vitales:** La jubilación representa una importante transición de vida que puede conllevar desafíos emocionales, sociales y económicos. La adaptación a esta nueva etapa requiere encontrar nuevos roles y actividades que proporcionen sentido y propósito a la vida.
- **Redes Sociales y Apoyo Familiar:** Las relaciones sociales sólidas y el apoyo familiar son fundamentales para el bienestar emocional y la salud mental de las personas mayores.
- **Salud Mental:** La prevalencia de depresión, ansiedad y trastornos del sueño aumenta en la vejez. Estos problemas pueden afectar significativamente la calidad de vida de las personas mayores y requerir atención profesional.
- **Bienestar Subjetivo:** La percepción de la calidad de vida y el envejecimiento exitoso dependen de factores individuales y sociales, como la salud física y mental, el estado civil, la situación económica, las relaciones sociales y la participación en actividades significativas.

Envejecimiento Diferencial

- **Variabilidad Individual:** El ritmo y la forma del envejecimiento varían considerablemente entre las personas debido a una compleja interacción de factores genéticos, estilo de vida y entorno. Algunos individuos experimentan un envejecimiento más saludable y activo que otros.
- **Desigualdades Sociales:** El acceso a atención médica de calidad, educación, oportunidades económicas y entornos seguros influye significativamente en el curso del envejecimiento y en las experiencias de las personas mayores. Las desigualdades sociales pueden exacerbar los desafíos del envejecimiento y limitar las oportunidades para una vida autónoma.
- **Envejecimiento en Diferentes Culturas:** Las percepciones y los roles sociales asociados a la vejez varían según las culturas. En algunas sociedades, las personas mayores son veneradas por su sabiduría y experiencia, mientras que en otras pueden enfrentar discriminación y marginalización.

2.1.2. Desafíos de la autonomía en la tercera edad

Más allá de los desafíos biológicos y psicosociales mencionados anteriormente, las personas mayores enfrentan obstáculos adicionales que impactan directamente a su autonomía:

Barreras Físicas y Ambientales

- **Accesibilidad:** Las dificultades para acceder a transporte público, edificios y espacios públicos limitan la movilidad y la participación social de las personas mayores.
- **Viviendas Inadecuadas:** Muchas viviendas no están adaptadas a las necesidades de las personas mayores, lo que aumenta el riesgo de caídas, accidentes y dificultades para realizar actividades cotidianas.
- **Entornos Inseguros:** Las calles, plazas y parques públicos pueden carecer de elementos que garanticen la seguridad de las personas mayores, como iluminación adecuada, superficies antideslizantes y pasamanos.

Exclusión Social y Discriminación

- **Edadismo:** Actitudes y prácticas discriminatorias basadas en la edad que limitan las oportunidades y el trato digno de las personas mayores. Estas actitudes pueden generar estereotipos negativos, prejuicios y exclusiones en diversos ámbitos de la vida, como el empleo, la educación, la atención médica y la participación social.
- **Aislamiento Social:** La soledad no deseada es un problema frecuente en la vejez que puede afectar negativamente la salud física y mental. El aislamiento social puede aumentar el riesgo de depresión, ansiedad, deterioro cognitivo y otras condiciones de salud.
- **Falta de Participación Ciudadana:** Las personas mayores pueden enfrentar dificultades para participar en la toma de decisiones que afectan sus vidas, tanto a nivel individual como comunitario. La falta de participación puede generar una sensación de impotencia y exclusión, además de limitar su capacidad para influir en su entorno y defender sus derechos.

Aspectos Económicos y Legales

- **Inseguridad Económica:** El riesgo de pobreza aumenta en la vejez, especialmente entre las mujeres mayores y las personas sin pensión adecuada. La inseguridad económica puede limitar el acceso a bienes y servicios esenciales, afectar la calidad de vida y generar estrés y ansiedad.
- **Falta de Protección Legal:** Las personas mayores pueden enfrentar dificultades para acceder a servicios legales y defender sus derechos. La falta de conocimiento legal y el acceso limitado a la justicia pueden aumentar su vulnerabilidad ante abusos y situaciones injustas.
- **Dependencia Económica:** Algunas personas mayores necesitan apoyo financiero o asistencia para realizar actividades básicas, lo que puede generar una pérdida de autonomía y control sobre su propia vida. La dependencia económica puede afectar negativamente su autoestima y bienestar emocional.

2.1.3. Tecnologías de asistencia actuales

Las tecnologías de asistencia tienen el potencial de mejorar la calidad de vida de las personas mayores, promover su autonomía y fomentar un envejecimiento activo y saludable. A continuación se presentan algunos ejemplos de tecnologías de asistencia disponibles actualmente y proyecciones para el futuro:

Tecnologías de Monitoreo y Alerta

- **Sensores Inteligentes:** Estos dispositivos pueden detectar caídas, movimientos inusuales o cambios en la temperatura corporal para alertar a cuidadores o servicios de emergencia.
- **Sistemas de Monitoreo Remoto:** Permiten el seguimiento a distancia de signos vitales, niveles de glucosa o actividad física para brindar atención preventiva y oportuna.
- **Aplicaciones Móviles de Salud:** Facilitan el automonitoreo de síntomas, el registro de medicamentos y el acceso a información médica personalizada.

Tecnologías de Asistencia para la Movilidad

- **Exoesqueletos:** Armazones robóticos que brindan soporte y asistencia para caminar, subir escaleras o realizar otras actividades físicas.
- **Scooters Eléctricos y Sillas de Ruedas Inteligentes:** Ofrecen transporte personal con mayor autonomía, navegación asistida y funciones de seguridad.
- **Prótesis y Ortesis Robóticas:** Prótesis mioeléctricas controladas por señales musculares y ortesis que brindan soporte y asistencia para mejorar la movilidad.

Tecnologías de Asistencia para la Comunicación

- **Teléfonos Inteligentes con Funciones de Accesibilidad:** Amplificación de sonido, reconocimiento de voz, escritura en pantalla y otras herramientas para facilitar la comunicación.
- **Software de Comunicación Aumentativa y Alternativa (CAA):** Símbolos, imágenes y herramientas de texto para personas con dificultades para hablar o escribir.
- **Videollamadas y Redes Sociales:** Permiten la conexión con familiares, amigos y profesionales de la salud a través de plataformas digitales.

Tecnologías de Asistencia para las Tareas Diarias

- **Robots Domésticos:** Asistencia en tareas del hogar como limpieza, cocina, compras y organización.
- **Asistentes Virtuales:** Interacción por voz para controlar dispositivos inteligentes, realizar llamadas, programar recordatorios y acceder a información.
- **Domótica y Hogares Inteligentes:** Control remoto de iluminación, electrodomésticos, sistemas de seguridad y otros dispositivos para mejorar la comodidad y la seguridad.

Proyecciones Futuras en Tecnologías de Asistencia

- **Inteligencia Artificial y Aprendizaje Automático:** Personalización de la asistencia, predicción de necesidades y adaptación a las preferencias individuales.
- **Realidad Virtual y Aumentada:** Rehabilitación física, terapia cognitiva y entrenamiento para la vida diaria en entornos simulados.
- **Impresión 3D:** Fabricación personalizada de prótesis, órtesis, dispositivos de asistencia y otros productos médicos.
- **Neurotecnología:** Interfaz cerebro-computadora para controlar dispositivos, acceder a información y mejorar la comunicación.

2.2. Internet de las cosas (IoT)

2.2.1. Concepto de IoT y su aplicación en el hogar

Cuando nos imaginamos un hogar donde las luces se encienden y apagan automáticamente según la presencia de un usuario concreto, la hora del día, la iluminación actual de la estancia, donde la temperatura se ajuste automáticamente para crear un ambiente confortable y los electrodomésticos se controlan de forma remota desde su smartphone. Esta es la realidad que el IoT está haciendo posible.

Beneficios del IoT en el hogar:

- **Mayor comodidad y conveniencia:** Los dispositivos IoT pueden automatizar tareas repetitivas como encender luces, ajustar la temperatura, controlar electrodomésticos y abrir persianas, liberando tiempo para actividades más importantes y creando un hogar más funcional y adaptado a sus necesidades.
- **Eficiencia energética y sostenibilidad:** Los dispositivos IoT pueden monitorear el consumo de energía de electrodomésticos, iluminación y otros dispositivos, identificando áreas de consumo excesivo y permitiendo tomar medidas para reducirlo. Además, pueden integrarse con sistemas de energía renovable para gestionar el consumo y almacenamiento de energía de manera eficiente, promoviendo la sostenibilidad del hogar.
- **Seguridad mejorada y protección del hogar:** Los dispositivos IoT pueden detectar intrusiones, monitorear peligros potenciales como incendios o fugas de gas y controlar el acceso al hogar, mejorando la seguridad y previniendo accidentes.
- **Personalización y entretenimiento:** Los dispositivos IoT pueden crear ambientes personalizados en el hogar ajustando la iluminación, la música, la temperatura y otros elementos en función de sus preferencias o la actividad que esté realizando. También pueden integrarse con sistemas de entretenimiento doméstico para ofrecer experiencias interactivas, como control de voz para la reproducción de música o películas, juegos controlados por movimiento, etc.
- **Mejor atención médica:** Los dispositivos IoT pueden usarse para monitorear la salud, detectar enfermedades en una etapa temprana y proporcionar atención médica personalizada, mejorando la calidad de vida y la esperanza de vida.

2.2.2. Protocolos de comunicación

Para comunicarse entre sí y con la nube, los dispositivos IoT utilizan diversos protocolos de comunicación. Algunos de los más comunes son:

- **Wi-Fi:** Es el protocolo más utilizado debido a su alta velocidad y alcance, ideal para dispositivos que necesitan transmitir grandes cantidades de datos o que se encuentran lejos del router.
- **Bluetooth:** Es ideal para la comunicación de corto alcance entre dispositivos, como un teléfono móvil y un altavoz inteligente.
- **Zigbee y Z-Wave:** Son protocolos de baja potencia diseñados para dispositivos de hogar inteligente que necesitan comunicarse a distancias más largas dentro del hogar, consumiendo menos energía.
- **MQTT (Message Queuing Telemetry Transport):** Es un protocolo de mensajería ligero diseñado para dispositivos IoT con recursos limitados, como sensores o pequeños electrodomésticos.

2.2.3. Seguridad

El Internet de las Cosas (IoT) está revolucionando el sector del cuidado de personas mayores, ofreciendo un sinfín de posibilidades para mejorar su calidad de vida, seguridad e independencia. Sin embargo, la seguridad en este ámbito es un aspecto crucial que no debe pasarse por alto. Los dispositivos IoT recopilan y transmiten datos sensibles, como información médica o hábitos de vida, convirtiéndolos en objetivos potenciales para ciberdelincuentes. Por ello, resulta fundamental implementar medidas de seguridad robustas para proteger la privacidad e integridad de estos datos.

Algunos de los aspectos claves para la protección de los datos son:

- **Cifrado de datos:** Es esencial emplear métodos de cifrado de datos de última generación, tanto para la información en tránsito como en reposo. Esto cobra especial importancia en el caso de datos de salud, que requieren la máxima protección.
- **Algoritmos robustos:** Se deben utilizar algoritmos de cifrado sólidos y contrastados, como AES (Advanced Encryption Standard) o RSA (Rivest-Shamir-Adleman), para garantizar la seguridad de los datos.
- **Autenticación robusta:** Es necesario establecer mecanismos de autenticación robustos para garantizar que solo usuarios y dispositivos autorizados puedan acceder a los datos. Esto puede incluir el uso de contraseñas seguras, autenticación de dos factores (2FA) o biometría.

Por otro lado, también hay que proporcionar una formación básica y capacitar a las personas mayores. Ya que es fundamental educar sobre los riesgos de seguridad en IoT y cómo protegerse de ellos. Esto incluirá enseñanza a crear contraseñas seguras, evitar compartir información en línea o, por ejemplo, reconocer correos electrónicos o sitios web fraudulentos.

3. Sensores: Estudio y análisis

3.1. Tipos de sensores y sus funcionalidades

Los sensores son dispositivos electrónicos que detectan y miden cambios en el entorno físico que los rodea. Estos cambios pueden ser de naturaleza diversa, como movimiento, temperatura, presión, luz, sonido, humedad, gases o presencia de objetos. La información recopilada por los sensores se convierte en señales eléctricas que pueden ser procesadas por microcontroladores, computadoras u otros sistemas electrónicos.

Los sensores se utilizan en una amplia gama de aplicaciones, desde sistemas de seguridad y control ambiental hasta dispositivos médicos y wearables. Su versatilidad y capacidad para proporcionar datos en tiempo real los convierten en componentes esenciales en la era de la Internet de las Cosas (IoT).

3.1.1. Sensores de movimiento

Los sensores de movimiento detectan cambios en la posición o el movimiento de objetos en su entorno. Estos cambios pueden ser provocados por personas, animales, vehículos o incluso vibraciones sutiles. Existen diversos tipos de sensores de movimiento, cada uno con sus propias características y aplicaciones:

- **Sensores PIR (infrarrojos pasivos):** Detectan cambios en la temperatura infrarroja emitida por objetos en movimiento. Son ampliamente utilizados en sistemas de seguridad, alarmas y automatización del hogar.
- **Sensores ultrasónicos:** Emiten ondas ultrasónicas y miden el tiempo que tarda el eco en regresar. Se utilizan para detectar la presencia de objetos a distancias cortas.
- **Sensores Doppler:** Detectan cambios en la frecuencia de una onda Doppler reflejada por un objeto en movimiento. Se utilizan en aplicaciones como radares de velocidad y sensores de proximidad.
- **Acelerómetros:** Miden la aceleración de un objeto, lo que permite detectar movimientos como caídas, vibraciones o cambios de velocidad.
- **Giroscopios:** Miden la velocidad de rotación de un objeto, lo que permite detectar cambios de orientación o giros.

3.1.2. Sensores de puertas (apertura y cierre)

Los sensores de puertas detectan si una puerta está abierta o cerrada. Estos sensores son esenciales en sistemas de seguridad, control de acceso e iluminación automática. Existen diferentes tipos de sensores de puertas, como:

- **Sensores magnéticos:** Detectan la separación o unión de dos imanes instalados en la puerta y el marco.
- **Sensores de contacto:** Detectan el contacto físico entre la puerta y el marco.
- **Sensores ópticos:** Detectan la interrupción de un haz de luz infrarroja o láser al abrirse la puerta.

3.1.3. Sensores de humedad

Los sensores de humedad miden la cantidad de vapor de agua presente en el aire, el suelo u otros materiales. Estos sensores son esenciales en aplicaciones como:

- **Control del clima en interiores:** Para mantener niveles de humedad óptimos para el confort y la salud.
- **Agricultura:** Para monitorizar la humedad del suelo y optimizar el riego de los cultivos.
- **Industria alimentaria:** Para controlar la humedad en procesos de producción y almacenamiento de alimentos.
- **Construcción:** Para detectar problemas de humedad en edificios y prevenir la formación de moho.

3.1.4. Sensores de temperatura

Los sensores de temperatura miden la cantidad de calor presente en un ambiente o en un objeto. Estos sensores son esenciales en aplicaciones como:

- **Control del clima en interiores:** Para mantener una temperatura confortable en hogares y edificios.
- **Refrigeración y congelación:** Para controlar la temperatura en refrigeradores, congeladores y cámaras frigoríficas.
- **Procesos industriales:** Para monitorizar y controlar la temperatura en procesos de fabricación.
- **Medicina:** Para medir la temperatura corporal de pacientes en hospitales y clínicas.

3.1.5. Sensores de luminosidad

Los sensores de luminosidad miden la intensidad de la luz en un ambiente. Estos sensores son esenciales en aplicaciones como:

- **Control de iluminación:** Para ajustar automáticamente la iluminación en función de la luz ambiental.
- **Fotografía y cinematografía:** Para medir la exposición a la luz y ajustar la configuración de la cámara.
- **Seguridad:** Para detectar intrusos en entornos con poca luz.
- **Agricultura:** Para monitorizar las condiciones de crecimiento de las plantas y optimizar el uso de la luz artificial.

3.1.6. Sensores de sonido

Los sensores de sonido detectan las ondas sonoras y las convierten en señales eléctricas. Estos sensores se utilizan en una amplia gama de aplicaciones, como:

- **Sistemas de seguridad:** Para detectar alarmas, roturas de cristales o sonidos inusuales.
- **Reconocimiento por voz:** Para convertir la voz humana en texto digital.
- **Música y audio:** Para grabar y reproducir sonido.
- **Asistencia para personas con discapacidades auditivas:** Para alertar a las personas sordas o con dificultades auditivas de sonidos importantes, como timbres de puerta o alarmas de incendio.

3.1.7. Otros sensores relevantes

Existen muchos otros tipos de sensores que pueden ser relevantes dependiendo del contexto específico de la aplicación. Algunos ejemplos incluyen:

- **Sensores de presión:** Detectan diferencias de presión atmosférica, utilizados en aplicaciones como la meteorología, la navegación aérea y la monitorización industrial.
- **Sensores de posición:** Responden a la ubicación de un objeto en el espacio, utilizados en robots, sistemas de navegación y control de movimiento.
- **Sensores de gas:** Detectan la presencia de gases específicos en el aire, utilizados en la detección de fugas de gas, control de calidad del aire y análisis ambiental.
- **Sensores químicos:** Detectan la presencia de sustancias químicas específicas, utilizados en análisis médico, control de procesos industriales y detección de contaminantes ambientales.
- **Sensores biológicos:** Detectan la presencia de organismos vivos o moléculas biológicas, utilizados en aplicaciones como la medicina, la biotecnología y la seguridad alimentaria.

3.2. Criterios de selección

En el mundo de la electrónica y la ingeniería, los sensores desempeñan un papel fundamental al permitirnos interactuar con el entorno físico y obtener información valiosa sobre diversos parámetros. La selección del sensor adecuado para una aplicación específica es crucial para el éxito del proyecto, ya que determina la precisión, el rendimiento, la confiabilidad y el coste general del sistema.

3.2.1. Criterios generales de selección de sensores

- **Precisión:** La precisión del sensor define la exactitud con la que mide la magnitud física de interés. Es crucial seleccionar un sensor con una precisión adecuada para la aplicación específica.
- **Rango de medición:** El rango de medición del sensor define el intervalo de valores que puede detectar. Se debe elegir un sensor cuyo rango de medición se ajuste al rango esperado de la magnitud física a medir.
- **Tiempo de respuesta:** El tiempo de respuesta del sensor define la rapidez con la que responde a los cambios en la magnitud física. Para aplicaciones que requieren mediciones en tiempo real, es importante seleccionar un sensor con un tiempo de respuesta rápido.
- **Consumo de energía:** El consumo de energía del sensor es un factor importante, especialmente en aplicaciones con limitaciones de energía o funcionamiento a batería. Se debe elegir un sensor con un bajo consumo de energía cuando sea necesario.
- **Coste:** El coste del sensor es un factor importante a considerar, especialmente en proyectos con restricciones presupuestarias. Se debe buscar un equilibrio entre las características del sensor y el precio.
- **Compatibilidad con la plataforma de desarrollo:** Es fundamental asegurarse de que el sensor seleccionado sea compatible con la plataforma de desarrollo que se utilizará, como Arduino o Raspberry Pi.
- **Facilidad de uso:** La facilidad de uso del sensor es importante para simplificar el proceso de integración y programación. Se debe elegir un sensor con documentación clara y ejemplos de código disponibles.

3.3. Investigación y análisis de sensores

3.3.1. Características técnicas, ventajas y desventajas de cada sensor

En cuanto a los diferentes tipos de sensores de movimiento, el sensor PIR destaca entre las diferentes opciones debido a su facilidad para detectar la presencia de los usuarios, eficiencia energética y sencilla instalación.

Característica	**Sensor PIR**	**Sensor ultrasónico**	**Sensor Doppler**
Marca	Panasonic, Sharp, Parallax	MaxSonar, SRF04, JSN	HBCC, HRF, Waveshare
Modelo	EK-MC100, GP2Y002S	MB1020, HC-SR04, JSN SR04T	HC-SR04, HRF04LP, DL-SR04
Tecnología	Infrarrojos pasivos	Ultrasonidos	Efecto Doppler
Detección	Movimiento	Distancia	Movimiento y velocidad
Alcance	Hasta 10 m	Hasta 8 m	Hasta 15 m
Ángulo	180°	Variable	Variable
Ventajas	Bajo consumo, fácil instalación	Medición precisa, detecta objetos pequeños	Detección movimiento/velocidad, resistente a interferencias
Desventajas	No detecta objetos estáticos, sensible a temperatura	Requiere línea de visión directa, problemas con superficies reflectantes	Mayor consumo, más complejo
Aplicaciones	Detección de presencia, alarmas, control de iluminación	Medición de distancia, robótica, aparcamiento	Puertas automáticas, control de tráfico, seguridad

Cuadro 1: Características de cada sensor de movimiento

Precisión	Los sensores PIR y Doppler son precisos en la detección de movimiento, mientras que los acelerómetros y giroscopios son precisos en la medición de aceleración y rotación.
Rango de medición	Los sensores ultrasónicos y Doppler pueden detectar movimiento a distancias más largas que los sensores PIR.
Tiempo de respuesta	Los sensores PIR y Doppler tienen tiempos de respuesta rápidos.
Coste	Los sensores PIR y ultrasónicos suelen ser más económicos que los acelerómetros y giroscopios.

Cuadro 2: Resumen sobre los sensores de movimiento

Entre los diferentes **sensores de apertura y cierre para las puertas o ventanas**, los sensores magnéticos destacan como una de las mejores opciones debido a la sencillez en instalar, eficiencia energética y alta fiabilidad. Aunque su alcance es limitado, lo que podría ocasionar fallos en el sistema a la hora de leer el estado de los objetos.

Una posible solución para evitar este posible inconveniente es combinar este sensor con uno de contacto, ya que se puede complementar de cara a mejorar la funcionalidad al detectar con mejor precisión el estado, independientemente de la distancia.

Característica	**Sensores magnéticos**	**Sensores de contacto**	**Sensores ópticos**
Tecnología	Campo magnético	Contacto físico	Luz
Detección	Apertura y cierre	Apertura y cierre	Apertura y cierre
Ventajas	Fácil instalación, bajo consumo, alta fiabilidad	Detección precisa, resistente a intemperie, varios actuadores	Alta precisión, sin contacto, resistente a suciedad
Desventajas	Alcance limitado, susceptible a interferencias	Requiere contacto físico, desgaste por uso	Mayor consumo, instalación compleja
Aplicaciones	Puertas, ventanas, cajones, armarios	Puertas, ventanas, máquinas, equipos	Puertas, ventanas, barreras de seguridad
Marcas	Reed Switches, Honeywell, Eaton	Omron, Pepperl+Fuchs, Turck	SICK, Banner Engineering, Leuze
Modelos	5500, 5816, 9000 Series	BZ, K, D-Series	OLM, DFOS2, OADM Series

Cuadro 3: Características de cada sensor de apertura y cierre

Precisión	Los sensores magnéticos y de contacto son precisos en la detección de la apertura y cierre de puertas.
Rango de medición	No aplica, ya que estos sensores detectan la apertura y cierre de puertas a corta distancia.
Tiempo de respuesta	Todos estos sensores tienen tiempos de respuesta rápidos.
Coste	Los sensores magnéticos suelen ser más económicos que los sensores de contacto y ópticos.

Cuadro 4: Resumen sobre los sensores de apertura y cierre

Los **sensores de humedad** del tipo resistivo se presentan como una de las opciones idóneas a instalar en el sistema, debido a su precio económico y facilidad de uso. Aunque estos dos puntos positivos afectan directamente a la precisión y linealidad de estos.

Una forma de instalación más compleja y precisa sería la de combinarlos con los sensores capacitivos, de forma que se aprovechan las ventajas de uno y de otro, evitando posibles fallas en el sistema o falta de datos válidos.

Característica	**Sensores resistivos**	**Sensores capacitivos**
Tecnología	Cambio de resistencia con la humedad	Cambio de capacitancia con la humedad
Ventajas	Bajo coste, fácil de usar, amplia gama de medición	Alta precisión, linealidad, resistente a la condensación
Desventajas	Menor precisión y linealidad	Mayor coste, más complejo de usar
Aplicaciones	Control de climatización, humidificadores, deshumidificadores	Equipos médicos, control de procesos industriales, estaciones meteorológicas
Marcas	HiTek, Sensirion, TE Connectivity	Rotronic, E2V, Honeywell
Modelos	HM15, SHT21, HTU21D	HC2-S, HY-2200, HIH-5000

Cuadro 5: Características de cada sensor de humedad

Precisión	Los sensores de humedad capacitivos son más precisos que los resistivos.
Rango de medición	La mayoría de los sensores de humedad pueden medir del 0 % al 100 % de humedad relativa.
Tiempo de respuesta	Los sensores de humedad suelen tener tiempos de respuesta rápidos.
Coste	Los sensores de humedad resistivos suelen ser más económicos que los capacitivos.

Cuadro 6: Resumen sobre los sensores de humedad

Debido a tener varias opciones para los **sensores de temperatura** y, como se ha especificado en varios momento, nuestro objetivo es de poder llevar esta tecnología a cualquier vivienda. Por lo que el sensor de temperatura de tipo termistor es la mejor opción, no solo por su bajo coste, sino también por su respuesta rápida y pequeño tamaño. Aunque, como es de esperar, su linealidad en la curva de temperatura es menos precisa pero es válida, ya que es un componente que al usarse en diferentes ámbitos es válido.

Característica	**Sensores semiconductores**	**Termistores**
Tecnología	Cambio de voltaje	Cambio de resistencia
Ventajas	Alta precisión, linealidad, interfaz digital	Bajo coste, respuesta rápida, tamaño pequeño
Desventajas	Mayor coste, más complejo	Menor precisión y linealidad
Aplicaciones	Equipos médicos, control industrial, electrónica	Termostatos, control de motores, sensores ambiente
Marcas	Analog Devices, Maxim, Texas Instruments	Murata, NTE, TE Connectivity
Modelos	LM35, LM34, AD590	NTC 10K, NTC 100K, NTC 1M

Cuadro 7: Características de cada sensor de temperatura

Precisión	Los sensores de temperatura de semiconductores son más precisos que los termistores.
Rango de medición	Los sensores de temperatura de semiconductores suelen tener un rango de medición más amplio que los termistores.
Tiempo de respuesta	Los sensores de temperatura suelen tener tiempos de respuesta rápidos.
Coste	Los termistores suelen ser más económicos que los sensores de temperatura de semiconductores.

Cuadro 8: Resumen sobre los sensores de temperatura

En primera instancia, alguno de los **sensores de luminosidad** típicos como los fotodiodos ofrecen un amplio rango de sensibilidad aunque en condiciones de muy baja luminosidad los fototransistores pueden ofrecer una datos de mejor calidad, aunque también hay que tener en cuenta que estos últimos pueden llegar a fallar en condiciones de luminosidad alta, limitando su rango.

Por otro lado, los fotodiodos tienen un tiempo de respuesta mejor y más rápida a los diferentes cambios de luz que pueda haber, como puede ser la de encender una luz en una de las estancias. Por lo que esta característica es esencial en nuestro entorno de desarrollo.

Característica	**Fotodiodos**	**Fototransistores**
Tecnología	Generación de corriente proporcional a la intensidad de luz	Amplificación de corriente de un fotodiodo
Ventajas	Amplio rango de sensibilidad, respuesta rápida, bajo consumo	Alta ganancia, interfaz digital, mayor sensibilidad
Desventajas	Menor sensibilidad	Mayor consumo, más complejo
Aplicaciones	Cámaras, medidores de luz, sensores de presencia	Control de iluminación, alarmas, auto-encendido
Marcas	Vishay, Avago, Hamamatsu	Fairchild, ON Semi., Rohm
Modelos	SFH6151, SFH6170, BPW34	BC557B, 2N3904, Q1N4001

Cuadro 9: Características de cada sensor de luminosidad

Precisión	Los fotodiodos y fototransistores son precisos en la medición de la intensidad de la luz.
Rango de medición	Los fotodiodos y fototransistores pueden medir una amplia gama de intensidades de luz.
Tiempo de respuesta	Los fotodiodos y fototransistores tienen tiempos de respuesta rápidos.
Coste	Los fotodiodos suelen ser más económicos que los fototransistores.

Cuadro 10: Resumen sobre los sensores de luminosidad

Como el objetivo de nuestro sistema es que sea lo más económico, eficiente y, además, que tenga una fácil instalación, entre los diferentes tipos de **sensores de sonido** y, entre ellos, los micrófonos de tipo condensador aunque ofrezcan una calidad de sonido superior y una mejor respuesta en cuanto a frecuencia, necesitan una alimentación externa lo que obliga a tener mínimo 2 instalaciones energéticas en la placa. Por lo que los micrófonos electret destacan en este aspecto, ya que tienen una fácil integración en sistemas pequeños y portátiles.

Característica	**Micrófonos de condensador**	**Micrófonos electret**
Tecnología	Conversión de ondas sonoras en señales eléctricas mediante condensador	Conversión de ondas sonoras en señales eléctricas mediante efecto electret
Ventajas	Alta calidad de sonido, amplia respuesta de frecuencia, baja distorsión	Bajo coste, tamaño pequeño, bajo consumo
Desventajas	Requiere alimentación externa	Menor calidad de sonido, mayor sensibilidad al ruido
Aplicaciones	Grabación de audio, instrumentos, sonido profesional	Teléfonos, juguetes, electrodomésticos
Marcas	AKG, Sennheiser, Shure	Knowles, Panasonic, Sonoran
Modelos	C-1000S, MK-4, SM57	SPM4043, WPM420, WM-61A

Cuadro 11: Características de cada sensor de sonido

Precisión	Los micrófonos de condensador son precisos en la detección de sonido.
Rango de medición	Los micrófonos de condensador pueden detectar una amplia gama de frecuencias de sonido.
Tiempo de respuesta	Los micrófonos de condensador tienen tiempos de respuesta rápidos.
Coste	Los micrófonos de condensador suelen ser más caros que otros tipos de sensores de sonido.

Cuadro 12: Resumen sobre los sensores de sonido

3.4. Conclusión sobre los sensores

En la monitorización de personas mayores en su domicilio, es crucial seleccionar los sensores más adecuados para garantizar una vigilancia eficaz y precisa. La elección de los sensores depende de las necesidades específicas del proyecto y de las características técnicas de cada sensor.

3.4.1. Evaluación del rendimiento

Los sensores seleccionados para este proyecto han demostrado un rendimiento excepcional en sus respectivas áreas de aplicación. El sensor PIR, por ejemplo, es ideal para la detección de presencia debido a su bajo consumo de energía y fácil instalación. Sin embargo, su incapacidad para detectar objetos estáticos puede ser un problema si la persona mayor no se mueve durante un período de tiempo prolongado. Por otro lado, el sensor ultrasónico puede complementar al sensor PIR al detectar caídas, ya que es capaz de detectar objetos pequeños y medir la distancia con precisión.

Los sensores magnéticos, por su parte, son excelentes para la detección de apertura y cierre de puertas debido a su fácil instalación, bajo consumo de energía y alta fiabilidad. Sin embargo, su alcance de detección limitado puede ser un problema si la puerta está lejos del sensor. En este caso, un sensor de contacto puede ser útil para detectar si la puerta se ha abierto o cerrado.

En cuanto a la monitorización de la humedad, los sensores resistivos son una opción económica y fácil de usar. Sin embargo, su precisión y linealidad son inferiores a las de los sensores capacitivos, que ofrecen una alta precisión y resistencia a la condensación.

Para la detección de la temperatura, los termistores son una opción económica y rápida. Sin embargo, su precisión y linealidad son inferiores a las de los sensores de temperatura semiconductores.

En cuanto a la luminosidad, los fotodiodos son una opción rápida y de bajo consumo de energía. Sin embargo, su sensibilidad es inferior a la de los fototransistores.

Finalmente, para la detección de sonido, los micrófonos de condensador ofrecen una alta calidad de sonido. Sin embargo, requieren alimentación externa, lo que puede ser un problema en algunas aplicaciones.

3.4.2. Sensores escogidos

Los sensores que han sido seleccionados para montar un posible sistema de monitoreo para las personas de la tercera edad combinan su funcionalidad, facilidad de uso y precio accesible. Estos son los sensores elegidos para su integración en el sistema:

- **PIR (motion) sensor:** Sensor de movimiento infrarojo pasivo fundamental para activar alarmas, enviar la orden de activación de luces o registrar patrones de actividad.
- **SparkFun Sound Detector:** Micrófono de tipo electret que ofrece un buen equilibrio entre calidad y precio. Donde sus dimensiones y bajo consumo son idóneos para sistemas integrados.
- **SparkFun BME280 - Atmospheric Sensor Breakout:** Sensor opcional que interesa tener para obtener datos sobre la presión atmosférica, que es un dato crucial para personas que puedan tener problemas respiratorios.
- **Water Level Sensor Module:** Permitirá la detección de fugas o desbordamientos en diferentes estancias, además de que es una solución económica para un problema evitable que puede ser desastroso.
- **Carbon Monoxide and Combustible Gas Sensor - MQ-9:** Del tipo semiconductor que permite detectar niveles peligrosos de monóxido de carbono o gases inflamables, previniendo intoxicaciones e incendios.
- **Adafruit TSL2591 High Dynamic Range Digital Light Sensor:** Sensor de luminosidad que utiliza tecnologías de fotodiodos, ideal para obtener datos sobre la luminosidad del ambiente.
- **Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2:** Encargado principalmente para obtener datos sobre la calidad del aire del ambiente, permite detectar valores como el de compuestos orgánicos volátiles (VOC) y dióxido de carbono equivalente (eCO2).
- **DHT2/11 Humity and Temperature Sensor:** Sensor que mide tanto la humedad como la temperatura del ambiente, donde la parte del sensor de humedad es de tipo capacitivo y, para el de temperatura, termistor.

Estos sensores fueron seleccionados por su rendimiento, coste, facilidad de uso y compatibilidad con las necesidades del proyecto. Juntos, proporcionan una solución de monitorización completa y eficaz para la atención de personas mayores en su domicilio.

4. Simulación

4.1. Diseño del entorno virtual

El proyecto consiste en crear un entorno virtual realista que imita el hogar típico de una persona mayor. Este espacio virtual sirve como un terreno de prueba controlado para el sistema de monitoreo de hogar inteligente, permitiendo una evaluación exhaustiva de la ubicación de sensores, la recolección de datos y la eficiencia del sistema. Al replicar las complejidades y detalles de un hogar real, podemos simular escenarios del mundo real y evaluar el rendimiento del sistema bajo diversas condiciones.

4.1.1. Recreación del domicilio

El primer paso en la creación del entorno virtual implicó la recreación de una residencia del mundo real utilizando un plano de planta (mostrado en la imagen n). Este plano muestra las dimensiones de cada habitación, la ubicación de puertas y ventanas, y el diseño general de la casa.

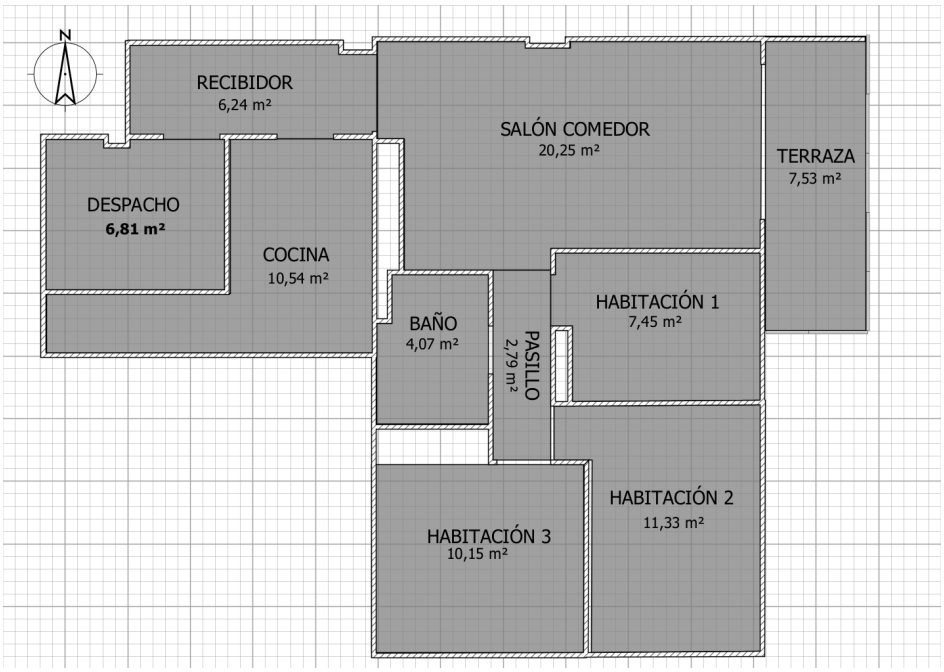

Figura 2: Plano del domicilio

Para convertir el plano a una representación digital, utilizamos un software llamado Sweet Home 3D. Esta herramienta me permitió crear un modelo 3D preciso de la casa al incorporar medidas precisas del plano original. La capacidad de exportar el modelo directamente a Unity fue una ventaja significativa de Sweet Home 3D.

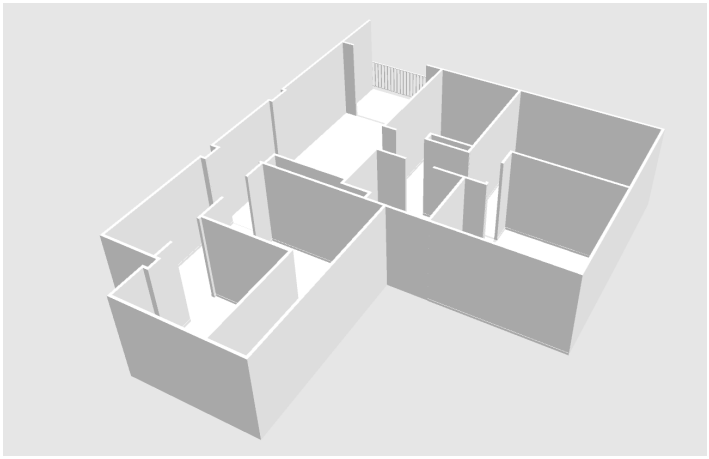

Figura 3: Vista lateral del domicilio

Sin embargo, a medida que el proyecto avanzaba, se hizo evidente que Sweet Home 3D tenía una limitación importante. El software tendía a crear objetos individuales para cada elemento dentro de la casa, lo que resultaba en un modelo fragmentado y difícil de gestionar al importarlo a Unity o usarlo para crear diferentes condicionantes. Esta fragmentación planteaba desafíos en términos de manipulación y optimización del entorno virtual, como puede ser la recreación de puertas, instalación de sensores o modificaciones del entorno.
Para abordar este problema, decidí reconstruir la casa dentro de Unity mismo. Al aprovechar el extenso conjunto de herramientas y objetos nativos de Unity, construimos una réplica lo más fiel a la realiad de la residencia. Esta minuciosa reconstrucción aseguró que el entorno virtual reflejara con precisión su contraparte del mundo real, proporcionando una plataforma fiable para simular la monitorización inteligente del hogar.

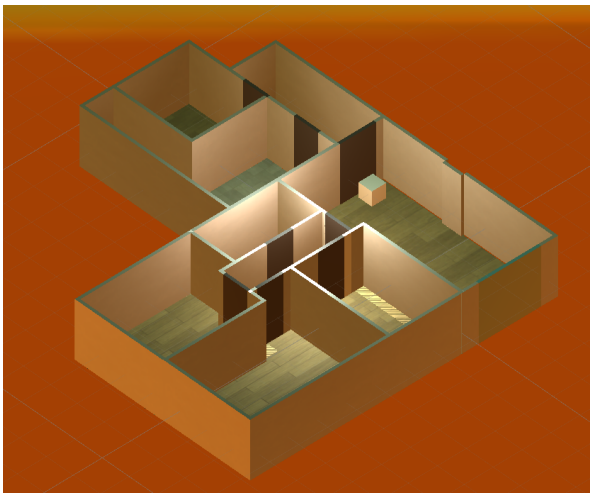

Figura 4: Vista en Unity del domicilio

4.1.2. Implementación de sensores y dispositivos

El **sensor de movimiento**, un elemento clave en el sistema de monitoreo, fue diseñado para detectar la presencia y actividad del individuo dentro del hogar simulado. Este sensor, representado en el *script DetectionSensor*, aprovecha un *BoxCollider* como área de detección, verificando si el jugador, rastreado por el componente *PlayerMovementTracker*, se encuentra dentro de sus límites.

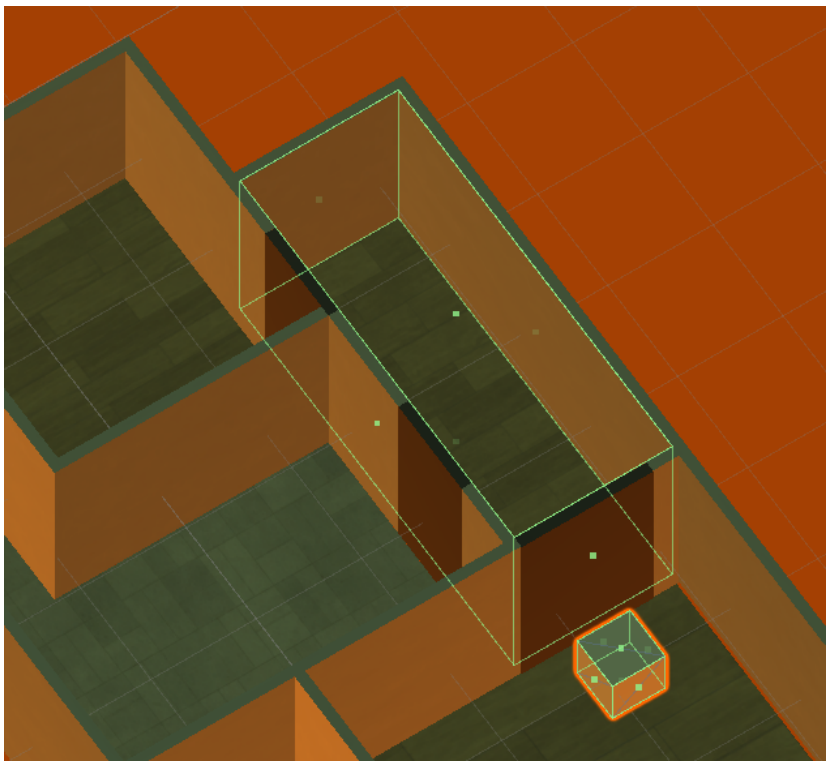

Figura 5: *Collider* del sensor de movimiento y usuario

La variable *_playerInside* registra si el jugador está dentro del área del sensor, mientras que *_hayMovimientoCache* combina esta información con el estado de movimiento del jugador (*_playerMovementTracker.IsMoving*) para determinar si hay actividad en la habitación. Estos datos se transmiten periódicamente al receptor principal (*_dataReciever*) a través del método *RecieveMovimientoData*, incluyendo el nombre de la placa padre para una identificación clara.

```
private PlayerMovementTracker _playerMovementTracker;
private BoxCollider _sensorCollider;
private DetectionSensor sensorMovimiento;
public bool _hayMovimientoCache;
public bool _playerInside;

void Start()
{
    _playerMovementTracker = FindObjectOfType<PlayerMovementTracker>();
    _sensorCollider = GetComponent<BoxCollider>(); // Obtener el
       BoxCollider
}

```

```
13 private void Update()
14 {
15     if (_sensorCollider != null && _playerMovementTracker != null)
16     {
17         _playerInside = _sensorCollider.bounds.Contains(
               _playerMovementTracker.transform.position);
18         _hayMovimientoCache = _playerInside && _playerMovementTracker.
               IsMoving;
19     }
20     _dataReciever.RecieveMovimientoData(_hayMovimientoCache,
           nombrePlacaPadre);
21 }
```

Listing 1: Código del sensor de movimiento

Explicación Detallada:

- **Variables Miembro:**
  - ***_playerMovementTracker*:** Referencia al componente *PlayerMovementTracker* responsable de rastrear el movimiento del jugador.
  - ***_sensorCollider*:** Referencia al *BoxCollider* del sensor, que define el área de detección.
  - ***_hayMovimientoCache*:** Variable que almacena el estado actual del movimiento (si el jugador está dentro del área y se está moviendo).
  - ***_playerInside*:** Indica si el jugador está actualmente dentro del área del sensor.
- **Método *Start()*:**
  - Obtiene referencias a los componentes *ISensorDataReciever* (placa de la habitación) y *PlayerMovementTracker*.
- **Método *Update()*:**
  - Verifica si el jugador está dentro del área del sensor (*_sensorCollider.bounds.Contains(...)*).
  - Actualiza la variable *_hayMovimientoCache* combinando la información de si el jugador está dentro del área y si se está moviendo.

Funcionalidad:

- El sensor de movimiento utiliza su *BoxCollider* para detectar cuando el jugador entra o sale de la habitación.
- En cada fotograma, comprueba si el jugador está dentro del área y si se está moviendo.
- Si ambas condiciones son ciertas, establece *_hayMovimientoCache* en *true*.
- El estado de movimiento (*_hayMovimientoCache*) y el nombre de la placa principal se envían a la placa de la habitación (*_dataReciever*) mediante el método *RecieveMovimientoData*.

El *script* **sensor de humedad** modela los patrones promedio de humedad de Tarragona, España, teniendo en cuenta las variaciones típicas a lo largo del día, donde podemos apreciar que, en la mayoría de casos, la humedad incrementa en primeras horas de la mañana y decrementa a medio día. Todo esto se tiene en cuenta a la hora de recrear los valores.

```
public float humedadActual;
private float _humedadAnterior;

void Start()
{
    _humedadAnterior = humedadActual;
    _dataReciever = GetComponentInParent<ISensorDataReciever>();
}

void Update()
{
    humedadActual = CicloDN.HumedadActual;
    float hActualRedondeada = Mathf.Round(humedadActual * 10.0f) * 0.1f;
    float hAnteriorRedondeada = Mathf.Round(_humedadAnterior * 10.0f) *
        0.1f;

    bool enviarData = false;
    if (hAnteriorRedondeada != hActualRedondeada)
        enviarData = true;
    if (_horaActual >= 12 && _horaActual < 16)
        enviarData = true;
    if (enviarData)
    {
        _humedadAnterior = humedadActual;
        _dataReciever.RecieveHumedadData(hActualRedondeada, enviarData,
            nombrePlacaPadre);
    }
}
```

Listing 2: Código del sensor de humedad

Explicación Detallada:

- **Variables miembro:**
  - *humedadActual*: Almacena el valor actual de humedad obtenido desde el *script* *CicloDN*.
  - *_humedadAnterior*: Guarda el último valor de humedad enviado a la placa de la estancia para detectar y comparar cambios antes de volver a enviar.
- **Método *Start()*:** Encargado de iniciar las variables de humedad para su uso en un futuro.
- **Método *Update()*:** Obtiene la humedad desde el *CicloDN.HumedadActual*, la normaliza para trabajar únicamente con un decimal de margen y, además, si estos valores han cambiado respecto al último valor enviado, se envía el nuevo dato al sistema.

Funcionalidad:
Esta implementación del sensor de humedad se encarga de obtener los valores directamente desde la rutina de *CicloDN* que es la principal gestora de los valores climatológicos en el sistema. Comprueba que el valor de la humedad haya variado respecto al último envío de información y, si es el caso, envía los datos de nuevo para tratarlos y almacenarlos en la base de datos.

El **sensor de luminosidad** es el encargado de gestionar la iluminación de manera eficiente y adaptativa. Este sensor, representado en el *script SensorLuminosidad*, interactúa con el ciclo día/noche (*CicloDN*) y el sensor de movimiento (*DetectionSensor*) para determinar cuándo encender o apagar las luces y ajustar su intensidad.

```
public Light[] luces;
public float luminosidadMinima = 10f;
public float tiempoApagado;
private float currentLuminosity;
public DetectionSensor detectionSensor; // Ref sensor de movimiento
private float tiempoSinMovimiento;
private float luminosidad;
private TimeSpan horaAnterior;

void Start()
{
    luminosidad = 0;
    detectionSensor = transform.parent.GetComponentInChildren<
        DetectionSensor>();
    tiempoApagado = (5f * ((CicloDN.DuracionDiaMin * 60f / 24f) / 3600f))
        ;
}

void Update()
{
    bool hayJugadorDentro = detectionSensor != null && detectionSensor.
        _playerInside;
    foreach (Light luz in luces)
    {
        if (cicloDN != null && !luz.enabled)
        {
            luminosidad = cicloDN.IntensidadLuminica;
        }
        if (hayJugadorDentro && !(CicloDN.Hora >= 8 && CicloDN.Hora <=
            20) && !luz.enabled)
        {
            luz.intensity = luminosidadMinima;
            luz.enabled = true;
            luminosidad = luz.intensity;
            tiempoSinMovimiento = 0f;
            horaAnterior = CicloDN.horaFormateada;
        }
        else if (luz.intensity > 0f && !hayJugadorDentro)
        {
            tiempoSinMovimiento += Time.deltaTime;
            if (tiempoSinMovimiento >= tiempoApagado)
            {
```

```
                luz.enabled = false;
            }
        }

        if (hayJugadorDentro && (CicloDN.horaFormateada.Hours >=
            horaAnterior.Hours && CicloDN.horaFormateada.Minutes !=
            horaAnterior.Minutes && (CicloDN.horaFormateada.Seconds -
            horaAnterior.Seconds) > 30))
        {
            luz.enabled = false;
        }
    }
    _dataReciever.RecieveLuminosidadData(luminosidad, true,
        nombrePlacaPadre);
}
```

Listing 3: Código del sensor de luminosidad

Explicación Detallada:

- **Variables miembro:**
  - *luces*: Arreglo de objetos *Light* que representan las luces a controlar.
  - *luminosidadMinima*: Valor umbral de luminosidad por debajo del cual se considera que la habitación está oscura.
  - *tiempoApagado*: Tiempo en segundos que la luz permanecerá encendida después de que el jugador salga de la habitación.
  - *currentLuminosity*: Almacena la luminosidad actual medida por el sensor.
  - *detectionSensor*: Referencia al componente *DetectionSensor* para detectar la presencia del jugador.
  - *tiempoSinMovimiento*: Contador de tiempo para controlar el apagado de las luces después de que el jugador salga de la habitación.
  - *luminosidad*: Almacena la luminosidad que se enviará a la placa receptora.
- **Método *Start()*:** Inicia la intensidad de cada luz a 0, para que en un principio, aunque estén habilitadas, no se muestren como encendidas Por otro lado, obtenemos la referencia al sensor de movimiento para poder usarlo a nuestro favor durante la simulación y, finalmente, calcula el tiempo de apagado de cada luz en función a la duración establecida inicialmente en la simulación.
- **Método *Update()*:** Consta de varios pasos clave que se ejecutan por cada fotograma de la simulación:
  - **Detección del jugador:** Nos encargamos de comprobar si hay un jugador dentro de la estancia mediante el sensor de movimiento y, además, iniciamos la variable *hayJugadorDentro*.
  - **Control de luces:** Se va iterando por cada una de las luces añadidas en el *array* establecido inicialmente en la simulación donde, por cada luz, se va comprobando si hay que apagarlas o no mediante una serie de condicionales, como puede ser el *hayJugadorDentro && !(CicloDN.Hora >= 8 && CicloDN.Hora <= 20) && !luz.enabled* que permite comprobar si el jugador esta dentro de la sala, la

luz esta apagada y que estemos en horas donde la luminosidad es mínima, entonces se endencederán. Por otro lado, si el jugador ya no esta en la propia sala, esperamos un tiempo calculado (*tiempoSinMovimiento*) en base al condicional de *tiempoApagado* y, si se cumple, se apagarán las luces. Finalmente, se envían los datos.

Funcionamiento:
El sensor de luminosidad utiliza la información del ciclo día/noche para ajustar la luminosidad en función de la hora y la intensidad de la propia vivienda en base al sol. Además, se coordina con el sensor de movimiento para encender las luces solo cuando es necesario, es decir, cuando el jugador está presente en la habitación y la luminosidad ambiental es baja.

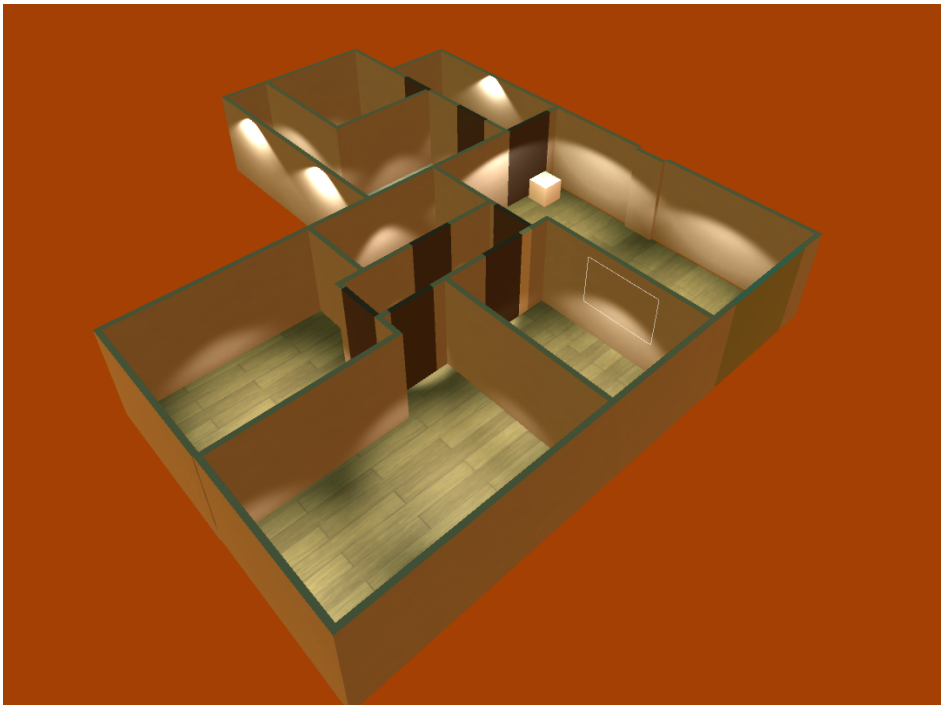

Figura 6: Puntos de luz en la vivienda

El **sensor de temperatura** es el encargado de registrar las variaciones de temperatura en el entorno simulado. El *script SensorTemperatura* se encarga de calcular la temperatura actual en base a la hora del día y la estación del año, utilizando los datos proporcionados por el *script CicloDN*.

```
public class SensorTemperatura : MonoBehaviour
{
    public float temperaturaActual;
    private float _temperaturaAnterior;
    private float _horaActual;

    void Start(){
        _temperaturaAnterior = temperaturaActual;
    }
    void Update(){
        temperaturaActual = CicloDN.TempActual;
        _horaActual = CicloDN.Hora;
        float tActualRedondeada = Mathf.Round(temperaturaActual * 10.0f)
            * 0.1f;
        float tAnteriorRedondeada = Mathf.Round(_temperaturaAnterior *
            10.0f) * 0.1f;
        bool enviarData = false;
        if (tAnteriorRedondeada != tActualRedondeada) enviarData = true;
        if (_horaActual >= 12 && _horaActual < 16) enviarData = true;
        if (enviarData)
        {
            _temperaturaAnterior = temperaturaActual;
            _dataReciever.RecieveTempData(tActualRedondeada, enviarData,
                nombrePlacaPadre);
        }
}
```

Listing 4: Código del sensor de temperatura

Explicación Detallada:

- **Variables miembro:**
  - *temperaturaActual*: Almacena la temperatura actual calculada por el sensor.
  - *_temperaturaAnterior*: Almacena la temperatura anterior para detectar cambios.
  - *_horaActual*: Almacena la hora actual obtenida del *script CicloDN*.
- **Método *Start()***: Principalmente hace comprobaciones varias e inicia la variable *_temperaturaAnterior* en base a *temperaturaActual*.
- **Método *Update()***: Obtiene la temperatura a partir de la función encargada de la climatología (*CicloDN.TempActual*) y la hora actual del sistema (*CicloDN.Hora*), donde redondeará la temperatura a un decimal y comprobará si hay cambios en estos valores. En cambio, si la hora actual es entre las 12:00-16:00, los datos se envían siempre al sistema.

Funcionamiento:
El sensor de temperatura calcula la temperatura actual en base a la hora del día y la estación del año, simulando las variaciones típicas que ocurren en un entorno real. Envía actualizaciones a la placa de la habitación solo cuando la temperatura cambia significativamente o durante un período específico del día (en este caso, entre las 12:00 y las 16:00).

4.1.3. Visualización de datos en tiempo real

La visualización de datos en tiempo real es un aspecto crítico del sistema de monitoreo del hogar inteligente, ya que proporciona información valiosa sobre las actividades y el bienestar de los residentes ancianos. Se emplearon varios métodos para lograr este objetivo.

Un enfoque involucró mostrar los datos del sensor directamente en la consola de Unity. Esto permitió obtener retroalimentación inmediata durante las fases de desarrollo y prueba, facilitando la identificación y resolución rápida de cualquier problema.

Otro método implicó transmitir los datos a un servidor remoto. Este servidor podría entonces procesar los datos, realizar análisis avanzados y generar visualizaciones que podrían ser accesibles por cuidadores o miembros de la familia a través de una interfaz web o aplicación móvil. Adicionalmente, el sistema fue diseñado para almacenar los datos del sensor en un archivo. Este archivo podría ser cargado periódicamente al servidor o accedido localmente para análisis offline.

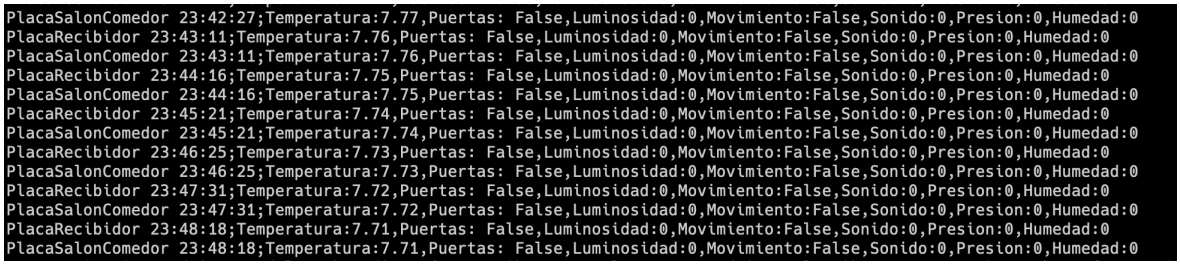

Figura 7: Primera vista de los datos recibidos

La elección del formato de archivo fue cuidadosamente considerada, con JSON emergiendo como la opción preferida debido a su estructura legible para humanos y compatibilidad generalizada con varios lenguajes de programación y herramientas de análisis de datos.

La comunicación entre las placas de circuito y los sensores inicialmente se concibió como una conexión por cable. Sin embargo, los protocolos y tecnologías específicos para esta comunicación se dejaron como tema de exploración adicional en secciones posteriores del proyecto.

4.2. Programación de la simulación

La programación de la simulación del hogar inteligente es un componente esencial en este proyecto. Mediante un diseño meticuloso y una implementación sólida, se ha logrado crear una interacción fluida entre los sensores, dispositivos y la placa maestra, lo que facilita la gestión eficiente de los datos y una simulación realista del comportamiento del usuario.

4.2.1. Interacción entre sensores y placas

La comunicación entre los sensores y la placa maestra se logra mediante una interfaz clave: *ISensorDataReciever*. Esta interfaz actúa como un contrato que define los métodos que cualquier dispositivo receptor de datos debe implementar. Cada sensor, al detectar un cambio relevante en su entorno, invoca el método correspondiente de esta interfaz para transmitir la información a la placa de la habitación.

```
1 public interface ISensorDataReciever
2 {
3     void RecieveTempData(float temp, bool enviarData, string nombrePlaca)
          ;
4     void RecieveDoorState(bool isOpen, string doorName, string
          nombrePlaca);
5     void RecieveHumedadData(float humedad, bool enviarData, string
          nombrePlaca);
6     void RecieveLuminosidadData(float luminosidad, bool enviarData,
          string nombrePlaca);
7     void RecieveMovimientoData(bool movement, string nombrePlaca);
8 }
```

Listing 5: Interficie para enviar datos a las placas desde los sensores

En este caso, la interfaz *ISensorDataReciever* incluye métodos específicos para cada tipo de sensor:

- ***RecieveTempData*:** Recibe la temperatura y un indicador de si se deben enviar los datos.
- ***RecieveDoorState*:** Recibe el estado de una puerta (abierta o cerrada) y su nombre.
- ***RecieveHumedadData*:** Recibe la humedad y un indicador de envío.
- ***RecieveLuminosidadData*:** Recibe la luminosidad y un indicador de envío.
- ***RecieveMovimientoData*:** Recibe el estado de movimiento (verdadero o falso).

Cada uno de estos métodos incluye un parámetro *nombrePlaca* que identifica la placa de la habitación a la que pertenece el sensor. Esto permite que la placa maestra sepa qué datos provienen de qué habitación.

4.2.2. Gestión de datos en la placa

Cada placa hija, que es mostrada por el *script PadreReceiver*, funciona como un punto central para recolectar información de los sensores presentes en su espacio designado (una habitación en particular). Recibe información de los sensores, la guarda temporalmente y transmite a la placa general cuando se cumplen ciertas condiciones.

```
private void ActualizarDatoPlaca(string nombrePlaca, string tipoDato,
    object valor, bool enviarData)
{
    datosActuales.hora = CicloDN.horaFormateada.ToString(@"hh\:mm\:ss");
    switch (tipoDato)
    {
        case "Temperatura":
            datosActuales.Temperatura = (float)valor;
            break;
        case "Puertas":
            datosActuales.Puertas = (string)valor;
            break;
        case "Luminosidad":
            datosActuales.Luminosidad = (float)valor;
            break;
        case "Movimiento":
            datosActuales.Movimiento = (bool)valor;
            break;
        case "Humedad":
            datosActuales.Humedad = (float)valor;
            break;
    }
    if (ultimosDatos == null)
    {
        ultimosDatos = (SensorData)datosActuales.Clone();
        GuardarDatosActualesEnArchivo(nombrePlaca);
    }
    else
    {
        TimeSpan diferenciaTiempo = TimeSpan.Parse(datosActuales.hora) -
            TimeSpan.Parse(ultimosDatos.hora);
        bool tiempoSuficiente = diferenciaTiempo.TotalSeconds >= 300;

        if (tiempoSuficiente || !SonDatosIguales(ultimosDatos,
            datosActuales))
        {
            ultimosDatos = (SensorData)datosActuales.Clone();
            GuardarDatosActualesEnArchivo(nombrePlaca);
        }
    }
}
```

Listing 6: Código para introducir datos de los sensores en la placa

La función *ActualizarDatoPlaca* actúa como un gestor de datos para el sistema de sensores simulados. Donde su principal trabajo es recibir información nueva de un sensor (identificado por *nombrePlaca*) y decidir si se almacena o no para analizarlo.

El funcionamiento es el siguiente:

- La función anota la hora exacta en que recibió el dato. Luego, comprueba de qué tipo de dato se trata (temperatura, puerta, etc.) y lo guarda en un registro temporal (*datosActuales*).
- Si es la primera vez que recibe datos de ese sensor, los guarda automáticamente en un archivo (como si fuera una primera referencia). Pero si ya tiene datos previos, se pone un poco más exigente.
- Compara los datos nuevos con los últimos que guardó. Si han pasado al menos 5 minutos o si los valores son muy diferentes, considera que los datos nuevos son relevantes y los guarda en el archivo. Si no, los descarta.
- Este proceso ayuda a evitar almacenar datos redundantes o poco interesantes. Por ejemplo, si la temperatura solo cambia un poquito cada minuto, no tiene sentido guardar todos esos cambios. Pero si la temperatura sube repentinamente 10 grados, eso sí es importante y se registra.
- Además, la función está preparada para enviar los datos a otro sistema (aunque esa parte no se muestra en el código). Esto podría ser útil para visualizar los datos en tiempo real, generar alertas o realizar análisis más complejos.

Envío de Datos a la Placa General

```
private void SendDataToMaster()
{
    try
    {
        PlacaData placaData = new PlacaData
        {
            Nombre = this.gameObject.name,
            Datos = datosPorPlaca[gameObject.name].Values.ToList()
        };
        placaGeneral.RecibirDatosPlaca(placaData);
        datosPorPlaca[gameObject.name].Clear();
    }
    catch (Exception ex)
    {
        Debug.LogError("Error al enviar datos a la placa general: " + ex.
            Message);
    }
}
```

Listing 7: Código para enviar los datos a la placa general

Este método realiza las siguientes acciones:

- Creación de *PlacaData*: Se crea un objeto *PlacaData* que contiene el nombre de la placa y una lista de objetos *SensorData* con los datos de los sensores de esa placa.
- Envío de datos: Se llama al método *RecibirDatosPlaca* de la placa general para enviar el objeto *PlacaData*.

- Limpieza de datos: Se limpian los datos almacenados en el diccionario *datosPorPlaca* para la placa actual, preparándola para el siguiente ciclo de recopilación de datos.

Placa General

La placa general es el cerebro del sistema de monitorización. Recibe los datos de todas las placas hijas, los acumula en una estructura de datos común y los guarda en un archivo JSON de forma periódica.

```
private IEnumerator GuardarDatosPeriodicamente()
{
    while (true)
    {
        yield return new WaitForSeconds(intervaloGuardado);
        registroActual.FechaHora = DateTime.Now.ToString("yyyy-MM-dd HH:
            mm:ss");
        while (datosRecibidos.TryDequeue(out var datosPlaca))
        {
            registroActual.Placas.Add(datosPlaca);
        }
        SaveDataToJson(registroActual);
        registroActual.Placas.Clear();
    }
}
```

Listing 8: Código para almacenar los datos perióricamente

La corrutina *GuardarDatosPeriodicamente* se ejecuta continuamente en segundo plano, esperando el intervalo de tiempo especificado (*intervaloGuardado*). Cuando llega el momento, toma todos los datos acumulados en la cola *datosRecibidos*, los agrega a un objeto *RegistroDatos* y llama a la función *SaveDataToJson* para guardarlos en un archivo JSON.

4.2.3. Implementación del ciclo Día/Noche

Para gestionar correctamente la temperatura, humedad, ciclos del usuario y, finalmente, el controlador de las luces inteligentes de la vivienda, el *script CicloDN* es la referencia de estos sensores ya que permite simular un ciclo real de datos y variable día a día en la simulación.

Para poder comprender esta lógica, se explica a continuación algunos de los puntos más relevantes de este *script*:

- **Enumeración para las estaciones:** Esta enumeración establece las cuatro posibles estaciones del año y pueden ser seleccionables en base al usuario a la hora de iniciar la simulación (Figura 8). Esto actuará como base para simular las temperaturas, los niveles de humedad y diferentes valores internos a la lógica de la simulación.

```
public enum Estaciones
{
    Invierno,
    Primavera,
    Verano,
    Otono
};
```

Listing 9: Enumeración de las estaciones disponibles

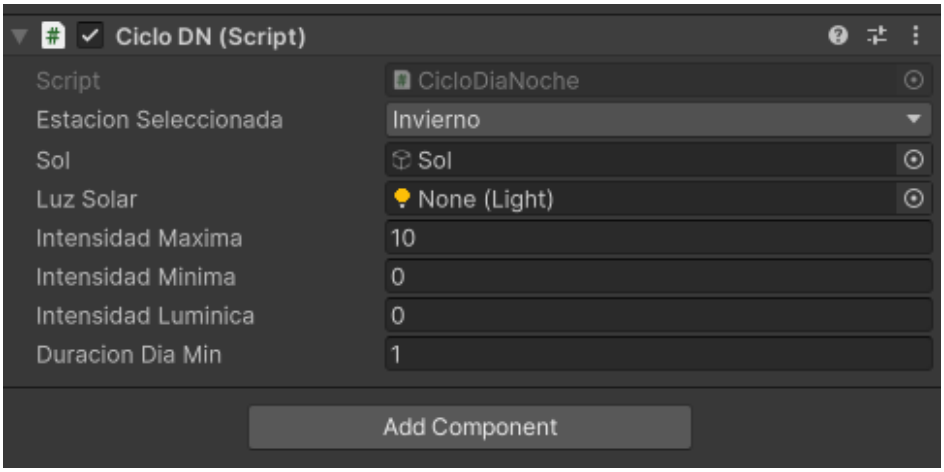

Figura 8: Selección de la estación

- **Variables clave:**
  - ***EstacionSeleccionada*:** Permite a la simulación saber cual es la estación a simular.
  - ***TempActual* y *Hora*:** Representan tanto la temperatura actual como la hora.
  - ***temperaturas*:** Matriz que contiene los valores máximos y mínimos de las temperaturas en base a la estación.
  - ***IntensidadLuminica*:** Variable accesible por el sensor de luminosidad para saber el nivel lumínico en todo momento.

- ***DuracionDiaMin*:** Controla la duración de una día en la simulación en minutos reales.

- **Método *start()*:** Se encarga de la inicialización de las variables relacionadas con el máximo/mínimo de las temperaturas en base a la selección. Por otro lado, obtiene la referencia del componente *light* del objeto Sol y, por último, calcula y establece la posición inicial del sol.

- **Método *update()*:** Núcleo de la simulación, ya que se encarga de modificar la hora de la simulación y realizar los cálculos respecto a la información que leerán los sensores, como el de humedad o temperatura. Un condicional que tiene es que cuando la *Hora* se igual a 24, volveremos a reiniciar el día y, además, la posición del sol y los valores que podrán leer los sensores en este nuevo día simulado.

4.2.4. Implementación de la rutina diaria del jugador

En el momento que tenemos los elementos básicos implementados en nuestro entono virtual (sensores, ciclo diario, etc.), ha llegado el momento de proporcionar a nuestro sujeto virtual una rutina que puede considerarse válida para cualquier usuario potencial.

Para lograr esto, se ha empleado una técnica denominada *waypoints*, que permite situar una serie de *gameObjects* en diferentes puntos del domicilio permitiendo que el jugador se desplace a estos mediante un cambio de coordenadas del usuario al *waypoint* deseado.

Esto permitirá simular una serie de patrones de movimientos que permiten obtener lecturas de los diferentes datos, por otro lado, así podemos comprobar como interactúan estos elementos entre ellos.

***Waypoints* en el domicilio**
Un *waypoint* es como un punto invisible en el entorno que, una vez distribuidos estratégicamente por cada habitación, actuarán como *trigers* de eventos. Cuando el jugador llega a cada punto de estos ocurre lo siguiente:

- **Interacción con el entorno:** Empiezan a modificarse o envira datos los diferentes sensores o actuadores.
- **Registro de datos:** Se recopila la información generada por el usuario en la estancia en concreto, como puede ser la fecha exacta y los sensores activados.

Rutina propuesta para analizar
Para simular una posible rutina de un usuario en su vivienda, se ha diseñado la siguiente rutina que, entre otras, contiene tareas cotidianas:

- **Mañana:** Despertarse, ir al baño donde se aseará, se preparará en su habitación e irá a desayunar.
- **Mediodía:** Cocinar y almorzar, trabajar o estudiar.
- **Tarde:** Tiempo libre, preparar y comer la cena.
- **Noche:** Relajarse e irse a dormir.

Factores aleatorios que pueden generarse
Para que esta simulación de la rutina sea lo más realista posible, se ha introducido una serie de elementos aleatorios:

- **Duración variable:** Cada actividad que realiza el usuario tiene un rango determinado de tiempo en la que debe de desarrollarse.
- **Localización flexible:** Algunas de las actividades propuestas, como la de comer o ver la televisión, puede que ocurran en diferentes estancias en base a la aleatoriedad de ese momento.
- **Actividades opcionales:** Ir al baño antes de comer es un ejemplo de acción que puede ejecutarse o no.

4.3. Tratado de datos de la simulación

Es este apartado, el tratado de datos se ha enfocado en varios puntos, como la implementación de un sistema genérico apto para cualquier tipo de dato y valor, garantizando tanto la privacidad como la seguridad de la información recopilada por los sensores y transmitida por la placa maestra. Aunque en este punto nos encontramos en un apartado de simulación se puede aplicar la misma metodología a la realidad, ya que la principal diferencia seria la del entorno simulado de los sensores y placa Arduino, pero el tratado de datos enviados por la placa maestra debería de tener el mismo formato, por lo que este tratado de datos se podría aplicar directamente a un proyecto real.

4.3.1. Almacenamiento de datos

En el momento que tenemos los datos enviados por la placa maestra, el almacenamiento de estos datos recopilador es fundamental para analizarlos posteriormente.

Se han implementado dos formatos para almacenar los datos, uno en formato JSON y otro en una base de datos con diferentes tablas. Por el lado de JSON, se ha escogido por su flexibilidad, legibilidad y compatibilidad con diversas herramientas de visualización y análisis. Este formato permite representar los datos de manera estructurada, utilizando pares de **clave-valor** para organizar la información de cada placa por estancia y valores de los diferentes sensores en una hora concreta.

Por otro lado, tambien se ha implementado una versión de base de datos con diferentes tablas que nos permitirá controlar en un mismo fichero y de una manera más organizada los datos, por lo que nos permitirá tener las siguientes ventajas:

- **Estructura:** Al ser un modelo relacional, permite organizar los datos en diferentes tablas relacionadas entre si mediante diferentes claves. Lo que implica una facilidad a nivel de consulta y análisis.
- **Eficiencia:** Como este servidor se alojará de manera local en el domicilio nos permite evitar la necesidad de un servidor externo.
- **Portabilidad:** Por el simple hecho de que la base de datos se almacena en un único fichero, este puede copiarse y moverse para realizar diferentes gestiones de una manera más sencilla y rápida, por lo que puede usarse en diferentes sistemas.

4.3.2. Diseño de los formatos escogidos

Base de datos

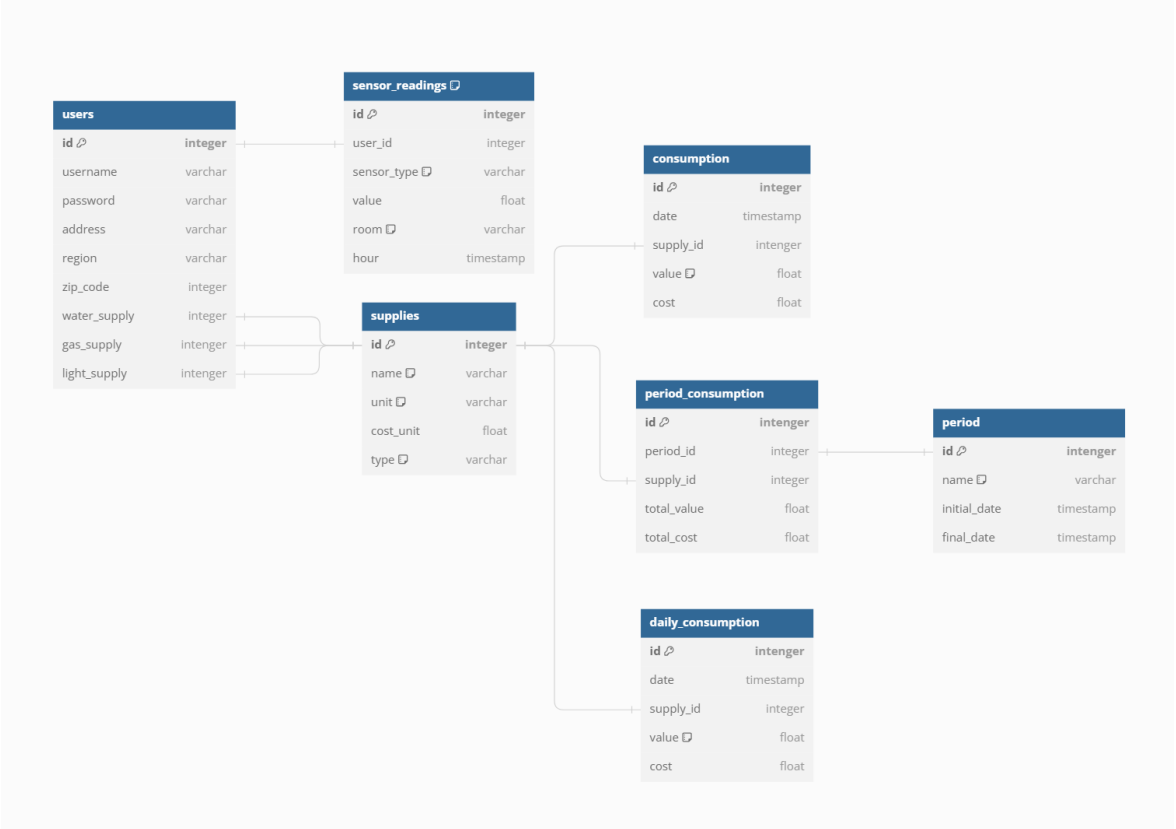

Figura 9: Diseño de la base de datos

Su diseño ha sido pensado para incluir posibles funciones, como puede ser la de incluir los datos de cada uno de los suministros contratados para así poder comparar diferentes datos en base a los recursos consumidos por la vivienda en función de los datos leídos por los diferentes sensores.

A continuación, tendremos la información respecto a las tablas incorporadas y su función en concreto:

- **Tablas de la Base de Datos:**
  - ***users*:** Almacena información del usuario que tiene instalado el sistema en su domicilio, como las credenciales de acceso, dirección y los códigos de usuario del suministro contratado.
  - ***sensor_readings*:** Registra las lecturas de los sensores en función de los datos procesados en el servidor. Aquí se registra la información de los sensores en base a la hora del sistema de la simulación y la estancia con el usuario como clave.
  - ***supplies*:** Contiene información sobre los diferentes tipos de suministros que puede tener contratado el usuario en su domicilio, con sus respectivas unidades de medida y coste por unidad.
  - ***consumption*:** Principalmente registra el consumo en cada momento de cada suministro, incluyendo la fecha, el identificador del suministro, el total consumido y el coste.

- ***daily_consumption*:** Almacena el consumo diario de cada suministro.
- ***period*:** Define los diferentes periodos de facturación en base a la fecha de inicio y fin.
- ***period_consumption*:** Registra el consumo como tal y el coste total en base al periodo.

- **Relaciones entre tablas:**
  - ***users.id - sensor_readings.user_id*:** Asocia la lectura de cada sensor con el usuario que tiene contratado el sistema.
  - ***consumption.supply_id - supplies.id*:** Vincula cada registro de consumo de cada suministro con el de la empresa gestora.
  - ***daily_consumption.supply_id - supplies.id*:** Establece la relación entre el consumo diario con el suministro correspondiente.
  - ***period_consumption.supply_id - supplies.id*:** Encargado de gestionar la relación entre el periodo y el suministro.
  - ***period_consumption.period_id - period.id*:** Relaciona el periodo del suministro con el periodo temporal a analizar.
  - ***users.gas_supply - supplies.id, users.light_supply - supplies.id, users.water_supply - supplies.id*:** Principalmente establece la relación entre el suministro contratado por el usuario con el registro de suministro genérico.

Formato JSON

Al tener una estructura jerárquica que refleja la organización física de los sensores y placa en el domicilio nos permite representar los datos en este formato, lo que permite una fácil identificación y acceso a los datos.

```
{
    "00:00:00": {
        "Temperatura": 7.8,
        "Puertas": {},
        "Luminosidad": "0",
        "Movimiento": "false",
        "Humedad": 65.0
    }
}
```

Listing 10: Contenido de un fichero JSON de una de las placas

Este código tiene el formato que se utiliza para representar datos de sensores de cada una de las placas, donde tenemos los siguientes puntos a tratar:

- **Estructura jerárquica:** Tenemos un objeto que contiene la hora como clave.
- **Datos de los sensores:** Para cada hora, existe un objeto que contiene el valor de cada uno de los sensores en ese momento determinado.

4.4. Análisis de datos

Una vez se ha completado la simulación, donde obtenemos los datos y almacenamos en una base de datos, es necesario realizar un análisis de estos valores en tiempo real. Esto se debe a que, de cara a los sanitarios o usuarios, el factor de poder visualizar los datos procesador en todo momento permite un mejor monitoreo del usuario y comprender el estado actual de este, anticipando sus necesidades de forma inmediata.

4.4.1. Técnicas para obtener información

- **Ventanas temporales deslizantes:** En lugar de aplicar filtros por horas o por fechas, se puede utilizar este tipo de técnica que permite analizar los datos de forma continua en todo momento. Lo que permitirá detectar cambios en el comportamiento del usuario de una forma más eficaz y precisa.
- **Actualización continua de estadísticas y correlaciones:** El factor de calcular y actualizar continuamente las estadísticas descriptivas y matrices de correlación en función de los datos que llegan en todo momento, permite identificar tendencias emergentes y cambios entre las variables relacionadas.
- **Minería de flujos de datos:** Al tener una base de datos orientada a analizar los datos de una manera más eficaz permite aplicar este tipo de algoritmos para descubrir patrones y anomalías en tiempo real. Esto se debe a que están diseñados para procesar grandes volúmenes de datos y pueden adaptarse a cambios en los patrones de los datos recibidos.
- **Detección de anomalías en tiempo real:** Algoritmos destinados a identificar eventos inusuales a medida que vayan ocurriendo, como puede ser una caída, una fuga de gas, etc. Esto permitirá enviar alertas inmediatas o tomar medidas en tiempo real.
- **Clustering:** Permite agrupar lecturas de los sensores similares para identificar grupos de comportamiento. Esto puede revelar patrones de movimiento, uso o consumo a lo largo del día (por ejemplo, rutinas matutinas).

4.4.2. Visualización de datos

- ***Dashborards* interactivas:** Al implementar una placa maestra en el sistema que se encargará de recibir los datos y procesarlos, esta misma herramienta puede actuar como tal si se programa adecuadamente. Ya que esto nos permitiría mostrar las lecturas en tiempo real de los sensores, consumo de suministros y los patrones de comportamiento del usuario. Además, estos se actualizarán de manera automática por cada dato procesado.
- **Gráficos dinámicos:** Al tener libre albedrío para la visualización de estos datos, es posible utilizar bibliotecas como *Ploty* o *Bokeh* para crear gráficas interactivas que se actualicen en tiempo real.
- **Diagramas de dispersión:** Poder estudiar la relación entre diferentes variables, como puede ser la de un sensor con otro o, incluso, la del consumo con los sensores, permitiría descubrir si tienen alguna relación.

- **Mapas de calor:** Al disponer de sensores de movimiento, este mapa sería muy útil para saber en que lugar se encuentra el usuario en diferentes horas del día, así como calcular el tiempo que pasa en una estancia en concreto y relacionarla con sus problemas médicos si los tuviera.

4.4.3. Detección de patrones y tendencias

- **Alertas en tiempo real:** Al tener un sistema que obtiene los datos en todo momento permite integrar sistemas de alertas que se activen en función de eventos inusuales o cambios significativos en el comportamiento del usuario. Estas alertas pueden ser enviadas a cualquier dispositivo o de cualquier manera, como por correo electrónico, SMS o a través de una aplicación móvil.

- **Predicción en tiempo real:** Al usar Python podremos utilizar modelos de aprendizaje automático en línea para predecir el comportamiento del usuario en un futuro en función de los datos recopilados. Esto permitirá al sistema anticipar necesidades y ajustar automáticamente el entorno para mejorar la eficiencia y seguridad.

- **Umbrales:** El poder establecer umbrales para las diferentes lecturas de los sensores, como puede ser la del ajuste de la luminosidad, permite controlar las desviaciones significativas de estos datos en función del umbral establecido, lo que detectará posibles eventos anómalos o cambios en el comportamiento del usuario.

- **Detección de anomalías:** Utilizar herramientas como *Isolation Forest* o *One-Class SVM* permitiría identificar lecturas de sensores o consumos inusuales.

4.5. Validación y pruebas de la simulación

Para esta sección, se han grabado una serie de pruebas que encontrarán en código QR en los diferentes apartados que continúan.

4.5.1. Apertura de puertas y envío de datos

Figura 10: Enlace QR para demostración y explicación de la apertura/cierre

4.5.2. Iluminación automática de la vivienda en diferentes condiciones

Figura 11: Enlace QR para demostración y explicación del control de luminosidad y su uso

4.5.3. Visualización del cambio de climatología y ciclo diario

Figura 12: Enlace QR para demostración y explicación del cambio climatológico

4.5.4. Rutina del usuario

Figura 13: Enlace QR para demostración y explicación de la rutina incorporada al usuario

5. Comunicación entre placas

La comunicación entre las placas es la columna vertebral del sistema de monitorización del hogar inteligente, permitiendo el flujo de información vital entre los diferentes nodos que lo componen. Una arquitectura bien diseñada y la elección adecuada de protocolos de comunicación son esenciales para garantizar la fiabilidad, eficiencia y seguridad de la transmisión de datos.

5.1. Arquitectura del sistema

5.1.1. Descripción de la topología de la red

La arquitectura del sistema se basa en una topología en estrella, con una placa maestra Arduino en el centro y varias placas hijas conectadas a ella. Esta estructura jerárquica proporciona un control centralizado y facilita la gestión de la comunicación.

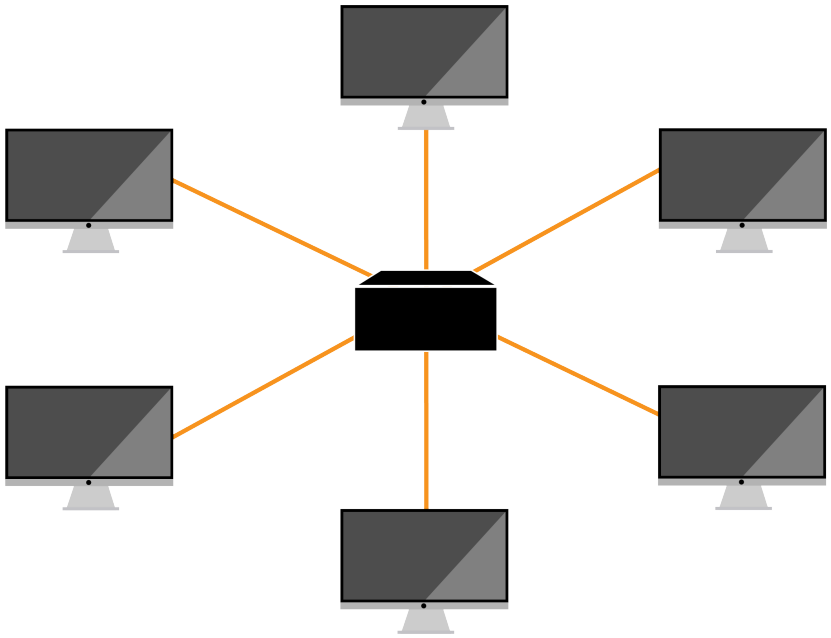

Figura 14: Topología de estrella

Placa Maestra o General
La placa maestra desempañará un papel fundamental como centro de control y coordinación del sistema, donde sus principales funciones serán:

- **Recepción de datos:** De los puntos más importantes a la hora de tratar los datos, ya que recopilará los datos enviados por las placas hijas (las que albergan los sensores).
- **Procesamiento y Almacenamiento:** Una vez recibidos los datos, se empezarán a procesar. Donde se organizarán en una estructura coherente y formato adecuado.
- **Comunicación externa:** En el momento que estos datos estén procesados, se enviarán a un servidor local que se encargará de almacenar los datos para que puedan ser accesibles desde el exterior por el cuerpo sanitario o por algún familiar.

- **Control de Dispositivos:** En función de los datos recibidos y la lógica implementada, puede comprobar el estado de cada uno de los sensores para ver si lo datos son correctos o el funcionamiento es el deseado, puede activar o desactivar actuadores como termostatos o las luces del domicilio o, inclusive, cerrar algún tipo de llave general de suministro si detecta algún tipo de parámetro extraño.

Placas Hijas

Cada placa hija, ubicada en cada una de las estancias y que aloja todos los sensores necesarios, se encarga de recopilar los datos, filtrarlos y enviarlos a la placa maestra para almacenar estos datos. Entre sus funciones, nos encontramos:

- **Recopilación de Datos:** Recibe los datos de los sensores que se encuentran conectados a la placa.
- **Preprocesamiento:** Como se ha mencionado anteriormente, se encarga de recibir los datos y filtrarlos para enviar a la placa maestra únicamente datos nuevos o cuando hay una variación de tiempo.
- **Envío de Datos:** Una vez tenemos los datos procesados para el envío, se transmiten a la placa maestra utilizando un protocolo de comunicación establecido para el hogar.

5.2. Protocolos de comunicación

5.2.1. Selección de protocolos para cada enlace

El objetivo en esta investigación ha sido la de lograr una transmisión de datos eficiente y segura. Nos encontramos con dos tipos de comunicaciones a tratar, entre la placa hija y los sensores y, por otro lado, entre las placas hijas y la placa maestra. Para esto, se han escodigo estas dos opciones:

- **I2C:** Protocolo de bus que, debido a su simplicidad y bajo consumo de energía, nos permitirá conectar múltiples sensores a una misma placa utilizando solo dos cables (SDA y SCL), lo que reducirá la complejidad del cableado y su fácil escalabilidad. Además, la velocidad de transmisión es suficiente para manejar todos los datos generados por los sensores. Una posible implementación de este protocolo se encuentra en el código fuente del anexo 10.8.

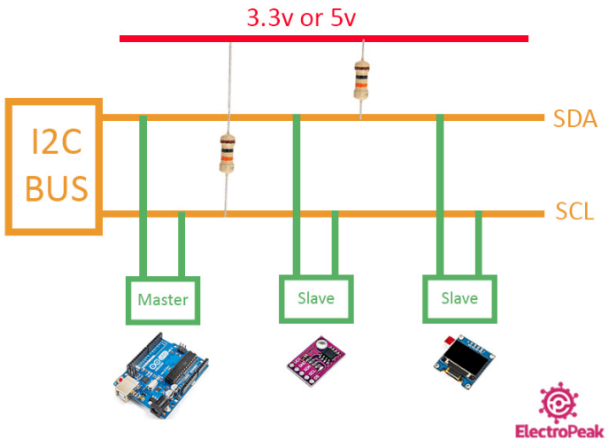

Figura 15: Protocolo I2C

- **MQTT sobre Wi-Fi:** Es un protocolo ligero y eficiente donde su modelo de publicación/suscripción permite una comunicación flexible y escalable, donde las placas hijas publican los datos de los sensores y, la placa maestra, se suscribe a los datos. EL uso de Wi-Fi proporciona la ventaja de movilidad, permitiendo colocar las placas hijas en diferentes puntos y evitando problemas de comunicación. Una posible implementación de este protocolo se encuentra en el código fuente del anexo 10.8.

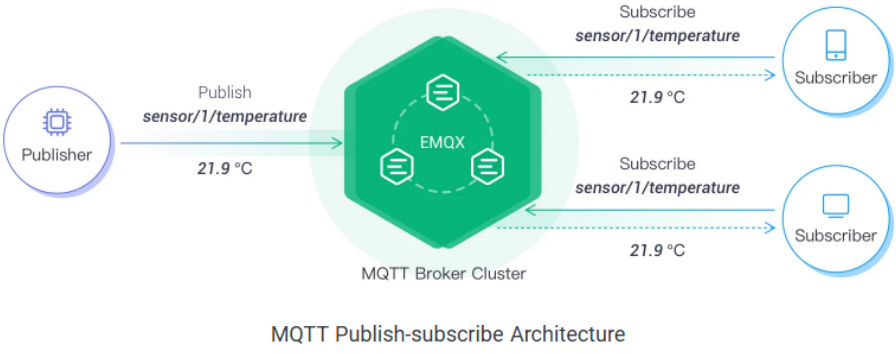

Figura 16: Protocolo MQTT

5.2.2. Seguridad en la comunicación

Para garantizar la privacidad y proteger los datos de posibles ataques, se implementan medidas de seguridad específicas para cada protocolo de comunicación utilizado:

- **I2C:** Al ser un protocolo de bus compartido, puede ser vulnerable a la intercepción y manipulación de datos. Para evitarlo en lo máximo posible, se pueden implementar alguna de estas medidas:
  - **Códigos de detección de errores:** Al no contar con un cifrado nativo, se pueden utilizar mecanismos de verificación de integridad para asegurar los datos recibidos no hayan sido alterados durante la transmisión.
  - **Cableado blindado:** Si hubiese puntos críticos en el entorno de instalación, se puede optar por la utilización de cableado blindado para protegerse ante posibles interferencias electromagnéticas externas.
- **MQTT:** Este protocolo fue diseñado integramente para entornos IoT, por lo que ofrece ciertas características de seguridad ya integradas en su entorno, pero también es posible implementar alguna de estas medidas:
  - **Cifrado TLS/SSL:** Mediante estos protocolos es posible cifrar la comunicación entre las placas y el broker. Esto nos permite garantizar una transmisión seguro y que, estos datos, no puedan leerle por terceros no autorizados.

6. Integración en un entorno real

Tras el estudio y simulación exhaustivos de sensores, funcionalidades y metodologías, se ha procedido a la investigación de componentes para su integración en un entorno real. Se han empleado diversas herramientas para seleccionar los sensores más idóneos para su implementación con la placa Arduino, buscando un equilibrio entre precio, rendimiento y compatibilidad.

En esta etapa, el enfoque se centra en la investigación y diseño de una instalación a medida para la vivienda propuesta. Se han considerado minuciosamente los diferentes sensores a implementar, su disposición estratégica dentro del domicilio y los requisitos específicos de cada espacio en términos de funcionalidad y estética.

El objetivo principal es recrear un sistema que se asemeje lo máximo posible a la simulación desarrollada, garantizando no solo eficiencia y funcionalidad, sino también una integración armoniosa en el entorno de la vivienda. Y, además, proporcionar al usuario final una seguridad y control mayor.

6.1. Esquema de la instalación

El sistema propuesto se basa en una red de placas Arduino, que destacan por su precio económico y rendimiento para entornos controlados, distribuidas por toda la vivienda. Donde cada placa actuará como un nodo independiente en el que se conectarán cada uno de los componentes necesarios (sensores, módulo de red, alimentación, etc) para monitorizar los diferentes parámetros deseados.

Estas placas o nodos se comunicarán con la placa central que actuará como servidor y donde se podrán visualizar los datos de una manera más rápida sin acceder directamente al entorno, lo que permitirá centralizar los datos y controlar todo el sistema de manera remota.

6.1.1. Descripción de los componentes necesarios

Además de los sensores seleccionados en el apartado 3.4.2, es necesario tener en cuenta los siguientes componentes para asegurar un buen rendimiento y uso:

- **Placas Arduino:** Actúan como nodos para interconectar los sensores y comunicarse con la placa maestra, encargada del procesamiento de datos. Se utilizarán modelos como Arduino Pro Mini, Nano o Uno, dependiendo de los requisitos de cada nodo.
- **Actuadores:** Dispositivos que permiten al sistema interactuar con el entorno físico, como pueden ser relés o electroválvulas.
- **Módulos de comunicación:** Permitirán la comunicación inalámbrica entre los diferentes nodos y la placa maestra.
- **Cableado y componentes electrónicos:** Principalmente para realizar la conexión entre la placa Arduino y los diferentes componentes electrónicos para tener un buen funcionamiento.

6.1.2. Disposición de los sensores

La ubicación de los sensores es crucial para el correcto funcionamiento del sistema. Algunos factores a tener en cuenta son:

- **Funcionalidad del sensor:** Es crucial colocar los sensores en puntos estratégicos para que se pueda sacar el máximo rendimiento a estos.
- **Cobertura:** Se debe procurar que el sensor abarque el máximo de área posible en la estancia en concreto.
- **Accesibilidad:** En caso de fallo en el sistema o algún tipo de error, que estos estén a mano para poder manipularlos de una manera más sencilla.
- **Estética:** A parte de la funcionalidad, hay que intentar adaptar estos elementos al entorno para implementarlos de manera discreta en el hogar.

6.1.3. Planificación del cableado y red Wi-Fi

Este punto es un aspecto crucial en el diseño del sistema propuesto, ya que nos permite visualizar una versión básica con los sensores deseados para realizar el estudio y análisis de los datos.

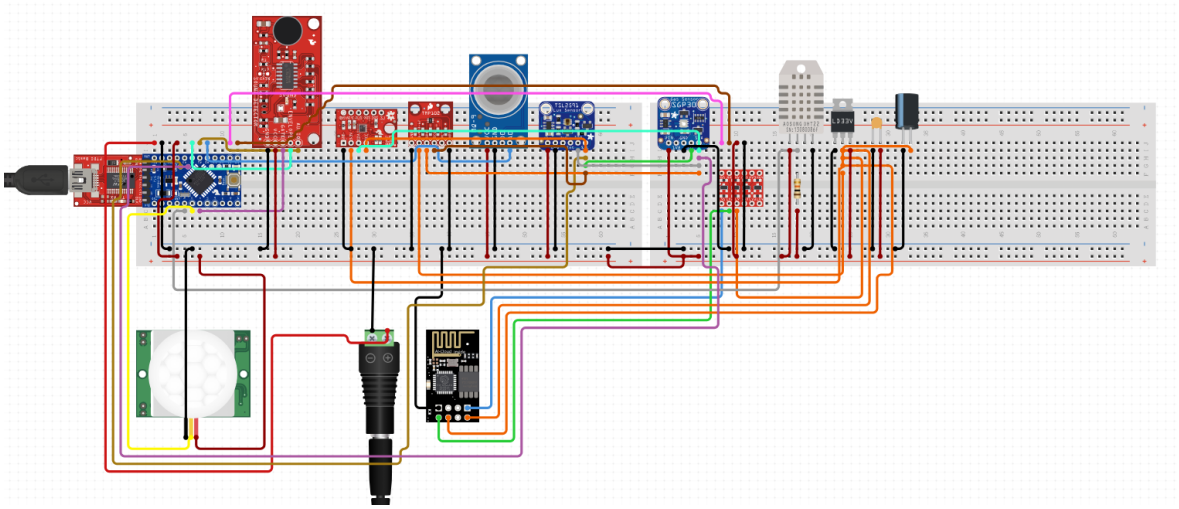

Figura 17: Prototipo de placa en cada estancia

Como se puede apreciar, en este prototipo hay más sensores de los que se han implementado en la simulación, esto es debido a su complejidad en un entorno simulado, como puede ser el sensor de sonido, de presión atmosférica o el de calidad de aire. Pero, principalmente, podemos encontrar una fuente de alimentación con una placa Arduino *Pro Mini 328* donde se interconectan mediante cables cada uno de los sensores. Por otro lado, también se añade un módulo Wi-Fi que se encargará de enviar los datos a la placa maestra. El presupuesto de esta placa al completo se encuentra en la tabla 14.

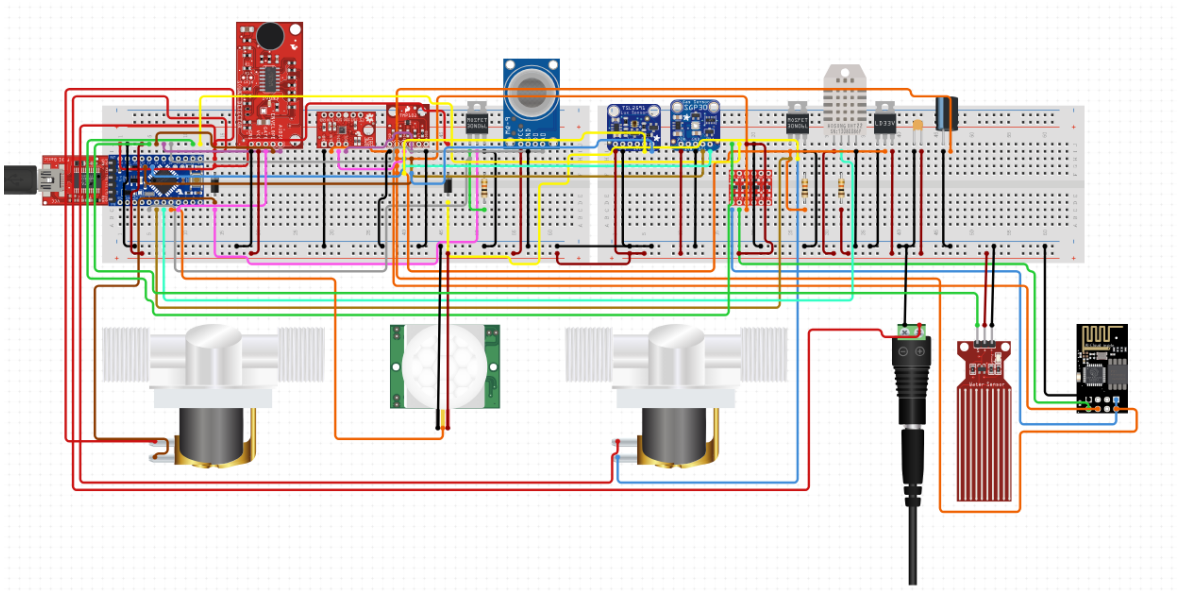

Figura 18: Prototipo de placa en la cocina

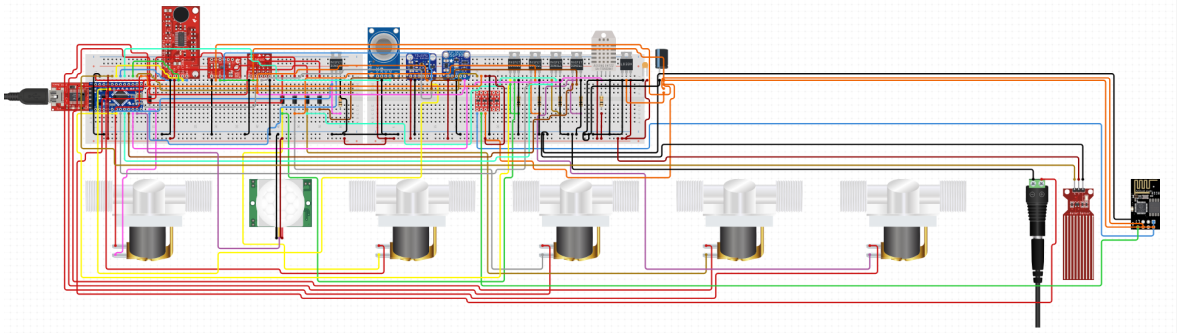

Figura 19: Prototipo de placa en el aseo

En cambio, en estas otras dos versiones, el cambio viene dado por un sensor de agua, que permitirá saber si se esta inundando esa parte de la casa. Por otro lado, también se añaden válvulas que permitirán controlar el cabal de agua que pasa en todo momento para gestionar el consumo de ese suministro y, en caso de riego de inundación, también tienen la opción de cerrarse mediante la orden de la placa que administra a estos elementos. El presupuesto de la placa en referencia a al baño se encuentra en la tabla 15, por otro lado, los datos de una placa para la cocina se encuentran en la tabla 16.

6.2. Presupuesto del proyecto

6.2.1. Coste de cada placa

Al tener la idea y objetivo de proporcionar una solución que no solo sea útil o que quede en un proyecto más, nos hemos centrado en que sea lo más accesible posible para todos los usuarios. En este estudio económico, nos hemos centrado en comprobar el coste de cada placa con todos sus componentes, aunque cabe destacar que estos valores son aproximados y no exactos. Debido a la variación entre diferentes proveedores y tamaño de la vivienda en cuestión.

En lo que trata de la vivienda, se ha realizado el estudio para un hogar con las siguientes características:

- Cinco habitaciones
- Un Aseo
- Una Cocina
- Dos pasillos

Además, como se ha mencionado anteriormente, se han desarrollado diversos diseños que se adaptan a las necesidades de cada estancia en el hogar. Esto nos proporciona como resultado tres nodos con costes diferentes en base a la localización de esta placa.

Por lo que el coste de cada placa sería de:

- **Placa genérica (por cada estancia):** En este caso, la placa incluirá los sensores básicos y tendría un coste de 150,91€. Esta placa genérica es versátil y puede ser utilizada en cualquier estancia de la casa.
- **Placa específica para la cocina:** En este caso, la placa incluirá los sensores básicos y tendría un coste de 150,91€. Esta placa genérica es versátil y puede ser utilizada en cualquier estancia de la casa.
- **Placa específica para un lavabo:** En este caso, como tenemos que controlar las tuberías del sanitario, ducha/bañera y pica, necesitaremos más electroválvulas. Esto incrementa el precio a 199,99€. Esta placa está especialmente diseñada para manejar las necesidades de un baño.

6.2.2. Cálculo total del coste de la instalación

Como se mencionó anteriormente, este análisis de costes se basa en la propuesta de la vivienda recreada en la simulación de Unity, por lo que el cálculo final es en base a este domicilio.

Es importante aclarar que estos valores solo hacen referencia a las placas, únicamente los nodos, no al sistema completa con todo funcional. Esto es debido a que el sistema requerirá un mantenimiento, instalación, software propio y diferentes herramientas, como puede ser un servidor, herramientas, etc. Por lo tanto, el precio es solo una estimación y no es definitivo.

A continuación, se muestra una tabla con el coste de cada placa en el domicilio recreado en la simulación:

Tipo de Habitación	**Cantidad**	**Coste Unitario (€)**	**Coste Total (€)**
Habitaciones	5	150,91	754,55
Baños	1	199,99	199,99
Cocinas	1	169,18	169,18
Pasillos	2	150,91	301,81

	Total	**1.425,54**

Cuadro 13: Coste total de las placas en el domicilio simulado

7. Complicaciones y soluciones

En la implementación del sistema propuesto es crucial anticipar y abordar las posibles complicaciones que puedan llegar a surgir de manera imprevista en los diferentes niveles del sistema, de esta manera se puede intentar asegurar un rendimiento ininterrumpido y proteger los datos recopilados. Ya que si no se gestionan estas complicaciones pueden comprometer la eficacia, eficiencia y fiabilidad del sistema, lo que puede provocar brechas de seguridad, datos erróneos y perjudicar el bienestar de los usuarios.

7.1. Posibles fallos en el sistema

7.1.1. Fallas en los sensores, dispositivos o placas

Estos componentes físicos son susceptibles a una gran variedad de fallos debidos a la naturaleza de la electrónica y mecánica. Por otro lado, por el lado de los elementos digitales, pueden tener problemas de seguridad o un mal tratado de datos en alguno de sus puntos.

Algunos de estos factores de riesgo en función del dispositivo y del tipo pueden ser:

- **Sensores:** Pueden llegar a provocar lecturas erróneas debido a posibles interferencias electromagnéticas, cambios bruscos en la climatología (humedad o temperatura) o por el desgaste debido al factor del tiempo. En este último caso, el tiempo, tanto la sensibilidad como la precisión pueden verse reducida.

- **Dispositivos:** Algunos de estos pueden ser las luces, electroválvulas o termostatos, que pueden llegar a presentar un mal funcionamiento debido a malas conexiones, fallos en la alimentación, conectores defectuosos o fallos en la comunicación inalámbrica, pueden ocasionar que los dispositivos reciban los datos incorrectamente o con variaciones que afecten a sistema de notificaciones.

- **Placas:** En este punto es más complicado que ocurran problemas, pero los más comunes son errores en los microcontroladores, problemas de memoria , cortocircuitos o daños en alguno de sus componentes eléctricos debido a factores energéticos, como pueden ser subidas de tensión o sobrecalentamientos. Todo esto puede provocar un fallo de la placa general, lo que ocasionaría el reemplazo de todos los componentes.

7.1.2. Errores de comunicación o de software

La comunicación entre las diferentes placas y el programa que los controla es fundamental para el funcionamiento correcto de todo el sistema pero, aun así, este conexión puede verse interrumpida o alterada por alguno de los siguientes factores:

- **Comunicación:** Las perdidas de conexión entre los dispositivos, interferencias en la señal inalámbrica o posibles fallos en los protocolos de comunicación pueden ocasionar este tipo de errores. Por otro lado, las posibles interferencias electromagnéticas que pueden venir provocadas por otros dispositivos en el domicilio, pueden alterar o bloquear la señal lo que impide la transmisión de datos. Finalmente, uno de los fallos más probables puede ser la transmisión de datos, ya que al intervenir un total de n placas con sus diferentes sensores enviando datos a la vez, puede llegar a saturar el sistema, lo que ocasionaría una pérdida de paquetes o la corrupción de la información.

- ***Software*:** El programa que se encargará del control del sistema es susceptible a errores, vulnerabilidades y errores que pueden llegar a comprometer la funcionalidad y seguridad. Uno de ellos pueden ser los posibles *bugs* en el código fuente que pueden ir desarrollándose durante el día a día, ya que es impredecible cuando uno de los datos que nos llegan pueden estar corrompidos o dañados, lo que provocará un fallo en el *software* que controla el tratado de datos. Algunas de las posibles vulnerabilidades de seguridad, como errores de programación que puedan permitir a los atacantes acceder al sistema o manipular los datos.

7.1.3. Problemas de seguridad o privacidad

Esta es una de las secciones críticas, ya que la seguridad de los datos recopilados por el sistema deben de ser abordado con la máxima exigencia posible. Estos datos contienen información personal, como puede ser la dirección de la vivienda, información de la red local, datos de contacto o las propias rutinas de los usuarios, pueden ser objeto de interés.

Algunos de los problemas que podemos apreciar son:

- **Acceso no autorizado:** Algunas personas pueden intentar acceder al sistema con diferentes fines, como robar datos, manipular el sistema para su propio beneficio o causar daños. Estos ataques pueden ser realizados a través de la explotación de vulnerabilidades, el uso de credenciales robadas o el acceso físico a los dispositivos.

- **Robo o pérdida de datos:** Estos datos recopilados por los sensores, transmitidos por la placa o procesados por la placa maestra pueden ser robados mediante la interceptación durante la transmisión, ya que al trabajar con una red *Wi-Fi* local puede ser insegura. El acceso no autorizado a la base de datos puede provocar que se divulgue esta información o se realice algún tipo de extorsión con estos datos. Por otro lado, el fallo en el almacenamiento de los datos, como errores en el disco duro, puede provocar una pérdida de datos irrecuperable.

- **Vulnerabilidad de *software*:** Como se ha ido comentando, los errores de código del sistema que controla la monitorización pueden ser explotados por atacantes para obtener el acceso al sistema completo. Algunos puntos por los que este programa puede verse comprometido puede ser debido a falta de pruebas exhaustivas o el uso de componentes de terceros que contienen vulnerabilidades que afecten directamente al sistema.

7.2. Soluciones a las posibles complicaciones

Para intentar abordar las diversas complicaciones que se han propuesto en el punto anterior es fundamental la implementación de un enfoque multicapa que combine las diferentes estrategias y tecnologías. Este modelo debe estar comprendido desde la detección temprana de fallos hasta la implementación de medidas de seguridad y planes de contingencia para diferentes situaciones que pueden provocarse.

7.2.1. Mecanismos de detección y corrección de errores

La detección con antelación de fallos puede prevenir el envío de datos erróneos o reemplazar componentes que empiecen a deteriorarse, lo que garantiza la continuidad correcta y eficaz del servicio. Para conseguir esto, se pueden implementar una serie de mecanismos que monitorizan constantemente el estado de cada uno de los componentes de sistema completo y alertar de cualquier anomalía:

- **Sensores:** La posibilidad del uso de sensores con funciones de autodiagnóstico permitirá que estos mismos dispositivos evalúen su propio estado y reporten cualquier problema que pueda estar alterando su funcionamiento normal. Un ejemplo de esto puede ser con un sensor de temperatura o humedad, ya que puede verificar si las lecturas obtenida se encuentran dentro de un rango normal, ya que este rango puede estar inicialmente con unos valores y, con el paso del tiempo, puede aproximar este baremo de datos para que sea más preciso si hubiese un posible fallo. Por otro lado, también se pueden implementar algoritmos de detección de anomalías que analizan las lecturas de cada uno de los sensores en tiempo real y buscan patrones que puedan indicar un mal comportamiento del sensor.

- **Dispositivos:** El control continuo del estado de cada uno de los dispositivos de la vivienda es crucial para detectar algún tipo de funcionamiento anormal. Esto podría incluir la verificación de parámetros como el del voltaje de cada elemento, la temperatura de funcionamiento para evitar sobrecalentamientos o comprobar el estado de la comunicación. En caso de detectar algún problema, el sistema puede enviar un alerta en forma de notificación a alguno de los dispositivos o vía SMS al usuario en el domicilio o a la asistencia técnica, lo que permite una intervención rápida para solucionar el posible fallo. Algunas otras posibilidades para implementar son el uso de dispositivos externos, como fusibles, que se desconectan automáticamente en caso de sobrecarga o cortocircuito, evitando daños mayores.

- **Comunicación:** Como se ha ido mencionando, la comunicación entre placas es un punto clave para el sistema, lo que provoca un punto crítico en esta sección y, para intentar evitar posibles errores, se pueden implementar mecanismos de detección y corrección de errores en los diferentes protocolos de comunicación. Estos mecanismos pueden incluir *checksums*, que verifican la integridad de los datos transmitidos y su retransmisión de paquetes perdidos.

- ***Software*:** A parte de implementar pruebas exhaustivas al programa de gestión del sistema para corregir posibles errores o vulnerabilidades. Además, se pueden aplicar diferentes métodos de validación de entradas y el control de excepciones.

7.2.2. Medidas de seguridad

La seguridad de los datos recopilados por los sensores por el sistema es primordial para proteger la privacidad de los usuarios y evitar, en lo máximo posible, el acceso no autorizado a información confidencial del usuario. Para garantizar esta seguridad se deben de implementar las siguiente capas de protección:

- **Cifrado:** Al utilizar el método de cifrado en los datos transmitidos entre las diferentes placas y el servidor es esencial para proteger la información de accesos no autorizados. Alguno de los algoritmos que se pueden implementar es el AES, que garantiza que los datos cifrados solo puedan descifrarse por personas autorizadas.
- **Autenticación:** La autenticación puede implementarse con diferentes métodos, como contraseñas seguras, autenticación de dos factores (2FA) o certificados digitales, lo que permite al usuario tener un acceso seguro y único en función a sus credenciales.
- **Control de acceso:** Una implementación en función al control de acceso basado en role (RBAC) permite restringir el acceso a los datos y diferentes funciones del sistema. Esto implica asignar a cada usuario un rol específico para prevenir la manipulación de datos, algunos de estos roles pueden ser el de administrador, usuario estándar o invitado.
- **Actualizaciones de seguridad:** El *software* y el *firmware* del sistema deben de intentar mantenerse actualizados en función de vulnerabilidades o errores que puedan ir ocasionándose en base al uso de cada usuario, lo que permite corregir vulnerabilidades y proteger contra futuras amenazas.

7.2.3. Planes de contingencia para situaciones de fallo

A pesar de implementar las diferentes medidas de prevención mencionadas anteriormente, es importante estar preparado para cualquier situación de fallo. Alguno de estos planes pueden ser:

- **Redundancia:** Al usar componentes como sensores duplicados o sistemas de respaldo, permite que el entorno siga funcionando incluso si alguno de los componentes falla. Por ejemplo, si un sensor falla, otro sensor puede sustituirlo temporalmente hasta que se arregle o cambie, lo que permite continuar informando al sistema de la lectura del propio sensor.
- **Batería de respaldo:** Si se diese el caso de un fallo en el suministro eléctrico del domicilio provocado por cualquier circunstancia, una batería puede mantener el sistema en funcionamiento hasta que se vuelva a restablecer el suministro. Esto permite enviar alertas y mantener las funciones principales en funcionamiento.
- **Notificaciones de fallo:** Al implementar un sistema de mensajería encargado de enviar alertas al usuario o servicio técnico permite una rápida intervención para solucionar el problema que ha activado la alarma.
- **Protocolos de recuperación:** La definición de procedimientos para recuperar el entorno del sistema en un caso de fallo del sistema completo implica una minimización del tiempo de inactividad y restauración del funcionamiento normal lo antes posible.

7.3. Adaptación a necesidades específicas

La implementación de este sistema encargado de la monitorización sobre los usuarios de la tercera edad en su hogar va más allá de una instalación de dispositivos y sensores, ya que requiere un enfoque holístico que debe de integrar la tecnología de manera fluida en la vida cotidiana de los usuarios, adaptándose a las diferentes necesidades y preferencias individuales. Este enfoque se base en los pilares de la personalización, educación y soporte técnico continuo.

7.3.1. Adaptación de las soluciones a las necesidades del usuario

Como cada hogar y usuario son únicos, cada una de las instalaciones e implementaciones del sistema debe comenzar con una evaluación del entorno y preferencias del usuario. Esto implica:

- **Evaluación personalizada:** Un especialista en el sistema debe de visitar cada uno de los hogar y realizar un estudio detallada de las características del hogar, tanto de los puntos de acceso como las diferentes áreas del hogar. Por otro lado, también se deberá de evaluar las necesidades particulares del usuario, ya que esto permite determinar que tipos de sensores y dispositivos son necesarios, dónde deben de ubicarse y cómo configurarlos para satisfacer todas las necesidades posibles.
- **Configuración flexible:** Este sistema deberá ofrecer una gama de opciones de configuración que permitan al usuario diseñar el funcionamiento del sistema según las necesidades que considere oportunas. Esto incluye, entre otras, ajustar el umbral de detección de los sensores y definir diferentes acciones de cada uno de los actuadores. Esta flexibilidad permite al sistema un análisis más personalizado y relevante.
- **Interfaz intuitiva:** Este es un punto primordial, ya que es el acceso entre el usuario y el sistema. Por lo tanto, debe de ser diseñada de manera intuitiva y fácil de usar, lo que implica utilizar un lenguaje claro y conciso, iconos reconocibles y una navegación sencilla por el menú.

7.3.2. Capacitación del usuario para el uso del sistema

para garantizar que los usuarios que utilizarán el sistema exploten al máximo el sistema propuesto, es esencial brindar una enseñanza práctica y adaptada a sus necesidades.

- **Explicar el funcionamiento del sistema:** El inicio de este aprendizaje empieza por una explicación clara y concisa del funcionamiento general del sistema. Esta explicación incluye la descripción de cómo los sensores recopilan los datos, cómo los diferentes dispositivos actúen en función a los datos obtenido y cómo la interfaz de usuario permite interactuar con el sistema.
- **Demostrar el uso de la interfaz:** Una vez que el usuario final haya comprendido el funcionamiento del sistema, se realizará una demostración práctica de las diferentes funciones que implementa el sistema, como puede ser la visualización de datos, configuración de alertas y gestión de dispositivos.
- **Proporcionar un manual de usuario:** Además de la explicación prácticas y teórica, se proporcionará al usuario un manual de usuario, donde podrá ser consultado para futuras dudas.

7.3.3. Soporte técnico

Uno de los componentes esenciales de la instalación y mantenimiento del sistema es el soporte técnico, ya que es el principal enlace entre los usuarios finales y el propio grupo que ha desarrollado el proyecto. Este debe de estar disponible para responder preguntas, resolver problemas y proporcionar asistencia en cualquier momento. Si dividimos este soporte en diferente puntos nos encontraremos con:

- **Soporte telefónico o en línea:** Debe de haber una fuente de comunicación estable y disponible para que el usuario pueda solicitar ayuda en caso de tener algún tipo de cuestión. Este deberá de estar disponible en un amplio horario y contar con personal capacitado para resolver los problemas.
- **Visitas técnicas:** En algunos casos, ya sea por error en el sistema o por mantenimiento de este, un técnico especializado que pueda visitar el hogar para realizar las tareas necesarias puede solucionar problemas más complejos con información más detallada por parte del usuario.
- **Actualizaciones remotas:** El sistema de monitoreo debe de contar con la opción de recibir e instalar las posibles actualizaciones para corregir diversos errores, para mejorar el funcionamiento del sistema o incorporar funcionalidades.

8. Conclusiones

8.1. Resumen del proyecto

8.1.1. Recapitulación de los objetivos y logros

Este proyecto ha sido diseñado y centrado en el desarrollo de un sistema de monitorización para personas de la tercera edad en su vivienda personal, utilizando Arduino y un conjunto de sensores para recopilar lecturas de los datos obtenidos sobre el entorno y actividad de los residentes del domicilio.

Por otro lado, se investigaron y analizaron diferentes tipos de sensores para determinar los más adecuados para la simulación y monitorización. Además, para permitir una comunicación correcta entre los diferentes dispositivos a integrar se exploraron dos protocolos de comunicación (*I2C* y *MQTT* sobre Wi-Fi) para garantizar una transmisión de datos eficiente y segura entre las placas de Arduino y entre la placa con los sensores.

En base a la simulación, se logró implementar con éxito la simulación con la herramienta de Unity, donde se pudo recrear una vivienda real en un entorno simulado y, además, se implementaron diferentes tipos de sensores para comprobar el comportamiento de estos y la interacción del usuario con el sistema. A parte, la programación de rutinas de comportamiento para simular en lo máximo posible las actividades diarias del jugador, como puede ser la de moverse por el entorno o interactuar con los diferentes componentes.

Finalmente, estos datos simulados que se han ido recopilando en la simulación se han procesado en diferentes formatos, un fichero de tipo JSON o en una base de datos, lo que puede permitir analizar patrones de comportamiento y evaluar la efectividad de nuestra propuesta de sistema de monitoreo.

8.1.2. Aporte del proyecto a la vida autónoma

Este proyecto tiene como objetivo la mejora de la vida autónoma de las personas de tercera edad en su vivienda, proporcionando un entorno seguro y controlado. El sistema de monitorización puede detectar diferentes situaciones de riesgo, como caídas o cambios bruscos en las diferentes lecturas de cada uno de los sensores. Permitiendo alertar a familiares cercanos o cuidadores para una rápida intervención.

Además, el constante monitoreo de la actividad y del comportamiento del usuario puede permitir identificar patrones y tendencias que permitan preveer futuras necesidades.

8.2. Limitaciones y futuras investigaciones

8.2.1. Reconocimiento de las limitaciones del proyecto

A pesar de los objetivos y logros conseguidos, este proyecto presenta limitaciones, entre estas nos encontramos:

- **Simulación en Unity:** Esta es una buena forma de probar el sistema, pero no se pueden replicar en su totalidad la cantidad de variables que se encuentran en un entorno real. Ya que algunas de estas variables, como puede ser el factor humano o las condiciones ambientales no estables pueden afectar el rendimiento.
- **Selección de sensores y protocolos de comunicación:** Aunque este enfoque es más llevado a la práctica, puede que estas diferentes tecnologías y dispositivos requieran de ajustes y adaptaciones para cada hogar.
- **Diversidad de dispositivos y tecnologías:** La gran cantidad de información y diferentes dispositivos puede dificultar la elección de cada uno de los componentes que se adecuen más a la vivienda y necesidades del usuario.
- **Implementación del sistema:** El mero factor de desarrollar e instalar este sistema en un entorno real presenta desafíos técnicos y logísticos, como la instalación de los propios sensores, compatibilidad con la infraestructura o la configuración de la red.

8.2.2. Propuesta de nuevas investigaciones

- **Pruebas en entornos reales:** Para la correcta implementación y estudio de los datos, se podrían realizar diferentes pruebas en entornos físicos y reales para evaluar el rendimiento del sistema y, además, recopilar datos más precisos sobre el impacto de este sistema en un usuario directo. Esto permitiría identificador posibles problemas técnicos, ajustar la configuración de cada sensor y, sobretodo, evaluar la aceptación y satisfacción de los usuarios con el sistema.
- **Investigación de tecnologías emergentes:** La integración de estas nuevas herramientas y dispositivos con nuestro sistema, como puede ser la inteligencia artificial y aprendizaje automático, mejoraría en gran cantidad la capacidad del sistema para detectar los diferentes patrones del usuario. El uso de diferentes algoritmos de aprendizaje automático permitiría, al sistema principal, identificar patrones más complejos y anticipar necesidades.

8.3. Impacto social del proyecto

8.3.1. Potencial del proyecto para mejorarla calidad de vida

Este proyecto tiene el potencial de mejorar la calidad de vida de las personas que se encuentran el grupo de la tercera edad, ya que al proporcionar un sistema que promueve la independencia, seguridad y bienestar de los residentes provoca que estos puedan disfrutar de una vida más plena. Esto es debido a que al mejorar su autonomía les permite mantener el control de sus vidas y, además, les permitirá seguir disfrutando de su entorno familiar y rutinas diarias.

Por otro lado, la posibilidad de detectar con antelación los posibles problemas de salud o avisar cuando hay algún tipo de emergencia reduce el riesgo de agravar las posibles enfermedades de cada usuario. Además, al poder personalizar la atención al usuario de manera única, podemos mejorar la calidad de vida adaptando el entorno a diferentes necesidades que pueden surgir.

8.3.2. Contribución a la sociedad en términos de bienestar y seguridad

Este sistema de monitoreo no solo busca el beneficio de los usuarios que lo utilizan, sino que también tiene un impacto positivo en la sociedad ya que al permitir que las persona mayores puedan vivir de forma más independiente, el sistema puede llegar a reducir la necesidad del uso de diferentes instituciones y aligerar la carga en los sistemas de salud. Lo que provoca directamente una reducción en el ahorro de recursos públicos y, sobretodo, una mejor eficiencia en la asignación de recursos.

Además, a la hora de promover tanto la seguridad como el bienestar, el proyecto contribuye a la creación de una sociedad más inclusiva y equitativa debido al abordaje de problemas como el aislamiento social o la soledad, que son factores de riesgo para la salud mental y física.

8.4. Viabilidad del proyecto en un entorno real

La viabilidad económica de este proyecto orientado a monitorear a las personas de la tercera edad en su domicilio para mejorar su vida autónoma depende de un análisis mayor de costes y beneficios que implicarán la instalación de este sistema. Aunque el coste inicial de los componentes puede parecer elevado, cabe considerar los beneficios a largo plazo a nivel de calidad de vida.

8.4.1. Costes a considerar

- **Adquisición de componentes:** El coste de las placas Arduino, sensores, actuadores y resto de componentes necesarios puede variar en función de la calidad, marca y cantidad.
- **Instalación y configuración:** La instalación del sistema requerirá de la contratación de profesionales formados en el sector de la electrónica y domótica. Además, la programación de placas e integración con el servidor puede llevar a un reto.
- **Infraestructura de red:** En caso de realizar esta instalación, la vivienda en cuestión debe de contar con una instalación previa de conexión Wi-Fi estable y con suficiente cobertura en todas las áreas para asegurar una buena comunicación entre las placas. Por otro lado, si la vivienda es demasiado grande, se deberán de instalar repetidores Wi-Fi para asegurar la buena recepción de datos por parte de la placa padre.
- **Servidor y software:** Para almacenar los datos de los sensores y poder analizarlos es necesario un servidor dedicado en el domicilio, lo que incrementará los costes de la instalación. Así como el desarrollo final del software de gestión de datos.
- **Mantenimiento:** Este sistema requerirá un mantenimiento para garantizar su correcto funcionamiento.

8.4.2. Beneficios potenciales

- **Detección temprana de problemas:** Gracias al sistema, este puede alertar a familiares o cuidadores sobre situaciones de emergencia, como caídas, fugas de gas, inundaciones, etc. Lo que permite una intervención rápida y, sobretodo, salvar vidas.
- **Monitoreo de la salud:** El seguimiento diario de la actividad, sueño otros parámetros fisiológicos puede ayudar a detectar cambios en la salud con anterioridad y poder tratar estos casos de una manera más eficaz.
- **Aumento de la autonomía y calidad de vida:** Al proporcionar un entorno más seguro y adaptado a las necesidades de cada usuario, el sistema fomentará la independencia y autonomía permitiendo permanecer en sus hogares por más tiempo.
- **Tranquilidad para familiares y cuidadores:** El monitoreo en remoto ofrece la tranquilidad necesario a los familiares y un mayor control a los cuidadores sanitarios.
- **Reducción de costes a largo plazo:** Aunque la inversión inicial puede ser elevada, el sistema puede generar ahorros a largo plazo al prevenir accidentes, evitando picos de consumo energético o reducir las necesidades médicas de cada usuario.

8.4.3. Evaluación final de la viabilidad

La viabilidad económica dependerá principalmente del equilibrio entre los costes y los beneficios. Es fundamental realizar un mayor análisis en función de la vivienda y del usuario a monitorear, ya que cada persona tiene necesidades específicas y se deberá de tener en cuenta en función de estas.

Entre los diferentes valores en base a los costos medios, se estiman los siguientes gastos:

- **Mano de obra:** El coste de la instalación puede variar en función de la vivienda y de la localización geográfica de esta, pero se estima que puede rondar entre los 500-1500€.
- **Cableado y materiales:** El coste de cables, cajas de conexiones o herramientas puede oscilar entre los 200-500€.
- **Servidor y software:** Al contar con un servidor local, este abarata más los costes ya que hay de diferentes tipos, aunque un precio promedio ronda los 300-1000€. Por otro lado, al proporcionar un software de gestión desarrollado de manera propia, no se sabría con exactitud el valor de este ya que dependerá principalmente de la oferta y demanda de los clientes.
- **Mantenimiento:** Este coste puede variar, pero es razonable considerar un abanico de entre 100 - 300€ anuales para cubrir posibles reparaciones o soporte técnico.

Teniendo en cuenta estos datos en conjunto con el precio promedio de cada placa individualmente, para la vivienda propuesta en la simulación, el coste general rondaría entre los 2525,54€ y 5725,54€.

A pesar de la inversión inicial, que es considerable, la viabilidad del proyecto debe de evaluarse a largo plazo en función de los beneficios propuestos anteriormente y del coste en función de la vivienda y del rendimiento que se le pudiera sacar.

8.5. Aportación del grado al proyecto

El Grado en Ingeniería Informática (*GEI*) ha sido un pilar fundamental para el desarrollo, ya que durante estos años me ha proporcionado una base sólida de conocimientos y habilidades en diversas áreas que han implicado adquirir un razonamiento ante problemas y una lógica para aplicar diferentes soluciones que ha sido especialmente relevante.

Entre las diferentes asignaturas del plan de estudios, las que han sido determinantes han sido las siguientes:

- **Arquitectura de Computadores (*AC*):** Los conocimientos adquiridos han sido aplicados para entender y comprender el funcionamiento de las placas Arduino y su posible interacción con diferentes dispositivos, como sensores o actuadores. Así como intentar optimizar el rendimiento del sistema, tanto en velocidad como en eficiencia.
- **Base de Datos (*BD*) y Sistemas de Información en las Organizaciones (*SIO*):** Al proporcionar una compresión profunda para diseñar e implementar tanto tablas como la base de datos utilizada, ha permitido un acceso eficiente y organizado. Por otro lado, gracias a esta última asignatura, se han adquirido diferentes habilidades para analizar datos en cantidades masivas, como las de nuestro proyecto, lo que permite estudiar estos datos y poder obtener información relevante de estos en función de parámetros específicos.
- **Programación (*PRO*):** Durante el transcurso de mis estudios han habido diversas asignaturas dedicadas íntegramente a la programación, tanto en Fundamento de Programación, como en Metodologías de Programación se ha aprendido diferentes estructuras de datos a utilizar y como trabajar con ellas, algoritmos que han sido esenciales para desarrollar el programa para las placas Arduino o, por ejemplo, el mismo *software* de análisis de datos.

Finalmente, a parte de los conocimientos teóricos, se ha fomentado a lo largo de los años el desarrollo de habilidades prácticas, como puede ser la resolución de problemas, tener un pensamiento críticos y, sobretodo, la capacidad de trabajar en equipo (aunque en este proyecto no se ha podido). Por lo que estas habilidades han sido fundamentales para superar los diferentes desafíos que se han ido encontrado durante el desarrollo del proyecto, así como la manera más eficaz de abordar los problemas de integración y compatibilidad.

8.6. Aportación del proyecto a la experiencia personal

Durante el desarrollo de las diferentes etapas del proyecto he tenido un aprendizaje invaluable, donde he tenido que aplicar los diferentes conocimientos adquiridos a un problema real y relevante. Tanto en el proceso de investigación, diseño, implementación y evaluación del sistema ha sido un reto constante que me ha obligado y exigido a salir de mi zona de confort y buscar, entre otras, soluciones creativas.

En el transcurso del proyecto he conseguido adquirir y mejorar ciertos campos, como puede ser el de la programación, diseño de la base de datos o análisis de estos mismos. Además, he mejorado en áreas como el trabajo autónomo, a gestionar el tiempo y recursos con más detenimiento y a presentar las ideas de mi cabeza en el papel de forma clara y concisa. Por otro lado, el desarrollo de este proyecto que es de una envergadura considerable, me ha proporcionado una visión más detallada y extensa del ciclo de vida de un proyecto, desde la primera idea hasta la implementación y evaluación final.

Por un lado más personal y enfocado a un futuro, este proyecto me ha hecho interesarme por un nuevo área como es el de la ciencia de datos o *Big Data* y el aprendizaje automatizado, áreas que planeo explorar en profundidad en un futuro. Esto es debido a que la simulación desarrollada en Unity es capaz de generar un gran volumen de datos en poco tiempo, lo que implica que existe la posibilidad de que, con las herramientas y técnicas adecuadas, estos datos puedan ser analizados para obtener una información aún más valiosa sobre los diferentes patrones de comportamiento del usuario y sus necesidades específicas.

Finalmente, durante el transcurso del proyecto, he reflexionado sobre el impacto social de la tecnología y su potencial para mejorar la calidad de vida de todas las personas, sobretodo las de la tercera edad. El desarrollo de este sistema puede ayudar a este colectivo a vivir en mejores condiciones y sin tantas preocupaciones por parte de sus más allegados, lo que me ha motivado a seguir investigando y explorando el campo de la tecnología asistencial y a buscar posibles formas en las que, con mis conocimientos, crear solucionar que repercutan en la sociedad.

9. Referencias

9.1. Repositorio y herramientas utilizadas

Referencias

[1] *Repositorio del proyecto*.
https://github.com/Dikiforov/TFG

[2] *Diseño de bases de datos*.
https://dbdiagram.io/

[3] *Diseño de interiores*.
http://www.sweethome3d.com/

[4] *Motor de videojuegos*.
https://unity.com/es

[5] *Diseño de circuitos electrónicos*.
https://www.circuito.io/

9.2. Marco teórico e información relevante

Referencias

[1] *Informe sobre el envejecimiento y salud*.
https://www.who.int/es/publications/i/item/9789241565042

[2] *Envejecimiento de la población*.
https://www.unfpa.org/es/envejecimiento-de-la-poblaci%c3%b3n

[3] *Últimas tendencias del envejecimiento*.
https://www.un.org/es/global-issues/ageing

[4] *Envejecimiento y salud*.
https://www.who.int/es/news-room/fact-sheets/detail/ageing-and-health

[5] *Guía práctica para personas mayores*.
https://oei.int/.../AF-PRINT-GuiaPractica-PersonasMayores.pdf

[6] *Necesidades básicas de un adulto*.
https://teleasistenciavital.com/blog/necesidades-basicas-adulto/

[7] *Necesidades de las personas mayores*.
https://www.consumer.es/solidaridad/necesidades-de-las-personas-mayores.html

[8] *Necesidades en la vida autónoma*.
https://cchs.csic.es/es/article/estudio-iegd-csic-revela-necesidades-personas-mayores-vida-autonoma-entornos-amigables

[9] *Detalles del envejecimiento*.
https://www.who.int/es/news-room/fact-sheets/detail/envejecimiento

[10] *Porcentaje de población superior a 65 años*.
https://data.worldbank.org/indicator/SP.POP.65UP.TO.ZS

[11] *Información sobre IoT*.
https://www.thethingsnetwork.org/

[12] *Información sobre IoT*.
https://blynk.io/

[13] *Tratado de datos utilizando MATLAB*.
https://thingspeak.com/

[14] *Topología de estrella*.
https://guiaexperta.website/descubre-que-es-la-topologia-de-estrella-y-como-optimiza-tu-red-de-manera-eficiente/

[15] *Topologías de red*.
https://siaguanta.com/c-tecnologia/topologia-de-red/

[16] *Protocolo de comunición MQTT*.
https://www.emqx.com/en/blog/mqtt-5-introduction-to-publish-subscribe-model

[17] *Información sobre IoT*.
https://aws.amazon.com/es/what-is/iot/

[18] *Fatiga en la toma de decisiones*.
https://pubmed.ncbi.nlm.nih.gov/28805132/

[19] *Impacto económico del envejecimiento*.
https://eleconomista.ovh/el-impacto-economico-del-envejecimiento-poblacional-consecuencias-y-soluciones-2/

[20] *La tercera edad en España*.
https://es.statista.com/temas/6532/la-tercera-edad-en-espana/#topicOverview

[21] *Independencia y sus cuidados*.
https://www.elrincondelcuidador.es/otros-cuidados/de-la-independencia-a-la-dependencia

[22] *Causas, tendencias y desafíos del envejecimiento*.
https://revistasanitariadeinvestigacion.com/el-envejecimiento-de-la-poblacion-tendencias-causas-y-desafios-futuros/

[23] *Cómo afecta a la economía*.
https://www.larepublica.net/noticia/como-afecta-a-la-economia-el-envejecimiento-de-la-poblacion

9.3. Búsqueda de sensores y dispositivos

Referencias

[1] *Posibilidades de trabajar con Arduino*.
https://www.xataka.com/basics/que-arduino-como-funciona-que-puedes-hacer-uno

[2] *Diferentes sensores de Arduino*.
https://paraarduino.com/sensores/

[3] *Protocolo MQTT*.
https://www.hivemq.com/mqtt-essentials/

[4] *Tutorial sensores Arduino*.
https://www.electrogeekshop.com/tutoriales-de-sensores-con-arduino

[5] *Componentes compatibles con Arduino*.
https://www.arduino.cc/en/hardware

[6] *Sensores y actuadores*.
https://openaccess.uoc.edu/bitstream/10609/141046/12/PLA3_Sensores%20y%20actuadores.pdf

[7] *Sensores de movimiento*.
https://eligenio.com/es/blog/sensores-de-movimiento-como-funcionan/

[8] *Tipos de sensores de movimiento*.
https://domoticaintegrada.com/sensor-de-movimiento/

[9] *Sensores de humedad*.
https://sdindustrial.com.mx/blog/sensor-de-humedad/

[10] *Sensor DHT11 y DHT2*.
https://proyectosconarduino.com/sensores/sensores-dht11-dht22-temperatura-humedad/

[11] *Tipos de sensores de temperatura*.
https://srcsl.com/tipos-sensores-temperatura/

[12] *Características de los sensores de temperatura*.
https://pirometriatecnica.com/sensores-de-temperatura-tipos-caracteristicas-y-aplicaciones/

[13] *Sensores de luminosidad*.
https://www.simonelectric.com/blog/que-es-un-sensor-de-luminosidad-y-para-que-se-utiliza

[14] *Cómo funciona un sensor de luminosidad*.
https://domoticaintegrada.com/sensor-de-luminosidad/

[15] *Automatizar sensor de luminosidad*.
https://www.simonelectric.com/blog/como-automatizar-luces-con-un-sensor-de-luminosidad

[16] *Diferentes sensores para Arduino*.
https://mentesanaliticas.com/sensores-que-son-y-cuantos-tipos-hay/?expand_article=1

[17] *Criterios para seleccionar sensores*.
https://es.slideshare.net/slideshow/criterios-de-seleccion-de-sensores/157458780

[18] *Criterios para seleccionar sensores*.
https://prezi.com/p/_tej5v3x2omh/15-criterios-de-seleccion-de-sensores-y-transductores/

[19] *Sensores ultrasónicos*.
https://lovtechnology.com/sensores-ultrasonicos-como-funcionan-y-donde-usarlos/

9.4. Simulación en Unity

Referencias

[1] *Documentación de Unity*.
https://docs.unity3d.com/ScriptReference/index.html

[2] *Ciclo día y noche*.
https://www.youtube.com/watch?v=FIjCwUj3As4

[3] *Método de interpolación*.
https://docs.unity3d.com/ScriptReference/Mathf.Lerp.html

[4] *Información sobre el uso de la interpolación*.
https://www.youtube.com/watch?app=desktop&v=RNccTrsgO9g

[5] *Interpolación linear*.
https://www.youtube.com/watch?v=q-1bFcMdUcg

[6] *Implementación de TextMeshPro*.
https://www.youtube.com/watch?v=bR0clpZvjXo

[7] *Envío de mensajes entre diferentes objetos*.
https://docs.unity3d.com/ScriptReference/GameObject.SendMessage.html

[8] *Manejo de eventos*.
https://docs.unity3d.com/ScriptReference/Events.UnityEvent.html

[9] *Tutorial de eventos*.
https://learn.unity.com/tutorial/eventos-w

[10] *Manejo de apertura de puertas*.
https://www.youtube.com/watch?v=XIwW11OI_4E

10. Anexos

10.1. Diagrama de Gantt

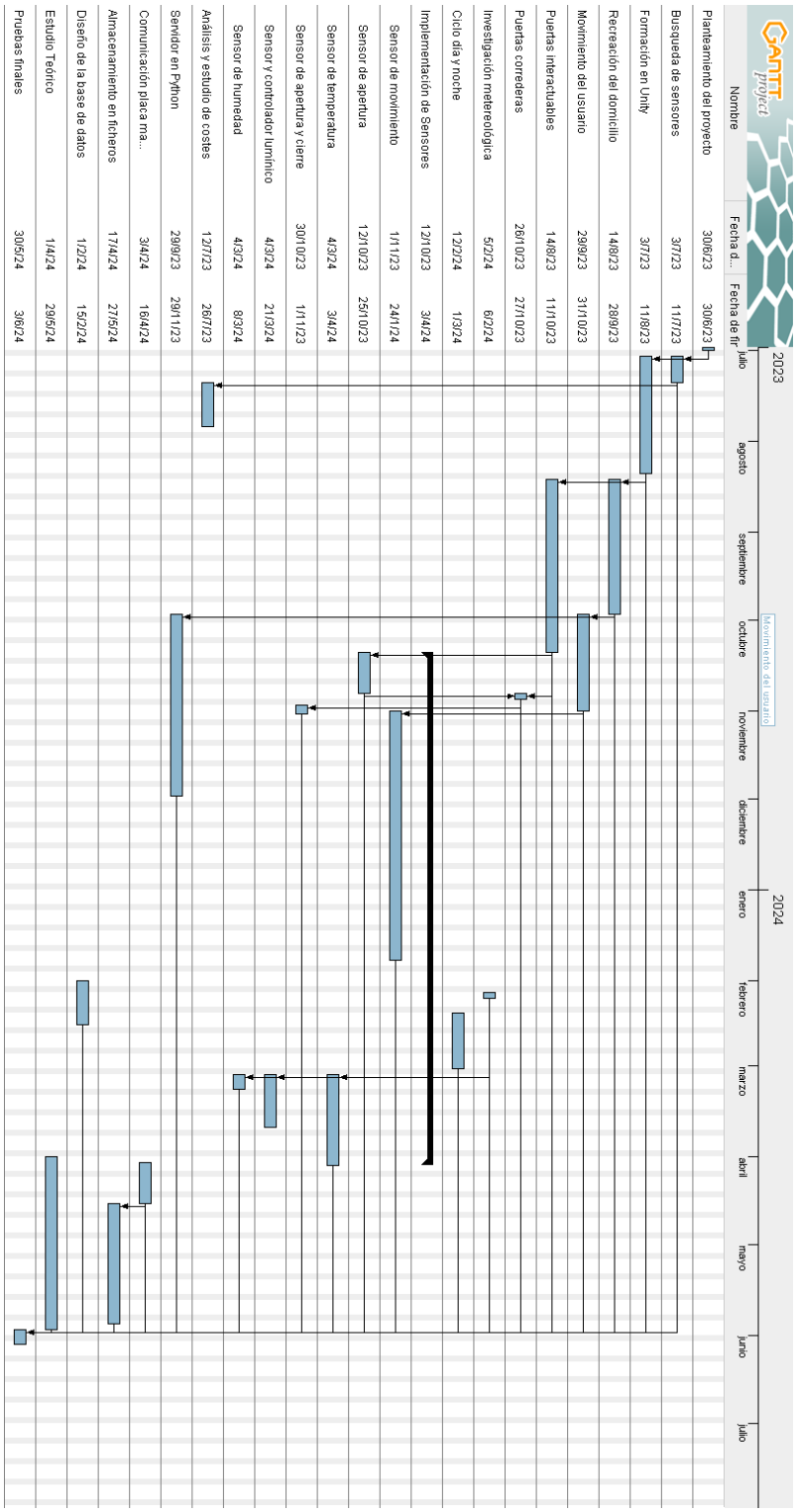

Figura 20: Diagrama de Gantt

10.2. Código fuente de los sensores

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

public class DoorController : MonoBehaviour
{
    public string nombreComponente;
    public Transform doorPivot;  // La visagra o punto de rotacion de la puerta
    private bool _isDoorOpen;
    private bool _isPlayerNearby;
    public float openAngle = -90f;
    private float _closedAngle = 0f;
    private float _animationTime = 2f;  // Duracion de la animacion en segundos
    private ISensorDataReciever _dataReciever;
    private string nombrePlacaPadre; // Variable para almacenar el nombre del padre

    void Start()
    {
        nombrePlacaPadre = transform.parent.name;
        if (doorPivot == null)
        {
            Debug.LogError("No se ha asignado una visagra a la puerta {"+nombreComponente+"}-{"+nombrePlacaPadre+"}");
        }
        _dataReciever = GetComponentInParent<ISensorDataReciever>();
    }

    private void OnTriggerEnter(Collider other)
    {
        if (!other.CompareTag("Player")) return;
        _isPlayerNearby = true;
        _isDoorOpen = _isPlayerNearby;
        _dataReciever.RecieveDoorState(_isDoorOpen, nombreComponente, nombrePlacaPadre, CicloDN.fechaActual);
    }

    private void OnTriggerExit(Collider other)
    {
        if (!other.CompareTag("Player")) return;
        _isPlayerNearby = false;
        _isDoorOpen = _isPlayerNearby;
        _dataReciever.RecieveDoorState(_isDoorOpen, nombreComponente, nombrePlacaPadre, CicloDN.fechaActual);
    }

    private IEnumerator ToggleDoorState()
    {
        var doorDistance = 113;
        _isDoorOpen = !_isDoorOpen;
        var targetAngle = _isDoorOpen ? openAngle : _closedAngle;
        var timeElapsed = 0f;

        switch (nombreComponente)
```

```
51          {
52              // Como aqui tendremos el movimiento de las puertas, en la
                    terraza ira diferente, ya que se desplazaran en el eje x
53              case "Puerta1Terraza":
54              {
55                  Vector3 initialPos = doorPivot.localPosition;
56                  Vector3 targetPos;
57                  if (!_isDoorOpen)
58                  {
59                      targetPos = new Vector3(initialPos.x - doorDistance,
                            initialPos.y, initialPos.z);
60                      _isDoorOpen = false;
61                  }
62                  else
63                  {
64                      targetPos = new Vector3(initialPos.x + doorDistance,
                            initialPos.y, initialPos.z);
65                      _isDoorOpen = true;
66                  }
67                  while (timeElapsed < _animationTime)
68                  {
69                      float t = timeElapsed / _animationTime;  // Normaliza
                            el tiempo transcurrido
70                      doorPivot.localPosition = Vector3.Lerp(initialPos,
                            targetPos, t);  // Interpola entre la posicion
                            inicial y la posicion objetivo
71                      timeElapsed += Time.deltaTime;  // Actualiza el
                            tiempo transcurrido
72                      yield return null;  // Espera hasta el proximo frame
73                  }
74                  doorPivot.localPosition = targetPos;  // Asegura que la
                        posicion final sea exacta
75                  break;
76              }
77              case "Puerta2Terraza":
78              {
79                  Vector3 initialPos = doorPivot.localPosition;
80                  Vector3 targetPos;
81                  if (!_isDoorOpen)
82                  {
83                      targetPos = new Vector3(initialPos.x + doorDistance,
                            initialPos.y, initialPos.z);
84                      _isDoorOpen = false;
85                  }
86                  else
87                  {
88                      targetPos = new Vector3(initialPos.x - doorDistance,
                            initialPos.y, initialPos.z);
89                      _isDoorOpen = true;
90                  }
91                  while (timeElapsed < _animationTime)
92                  {
93                      float t = timeElapsed / _animationTime;  // Normaliza
                            el tiempo transcurrido
94                      doorPivot.localPosition = Vector3.Lerp(initialPos,
                            targetPos, t);  // Interpola entre la posicion
                            inicial y la posicion objetivo
```

```
95                      timeElapsed += Time.deltaTime;  // Actualiza el
                            tiempo transcurrido
96                      yield return null;  // Espera hasta el proximo frame
97                  }
98                  doorPivot.localPosition = targetPos;  // Asegura que la
                        posicion final sea exacta
99                  break;
100             }
101             default:
102             {
103                 // En cambio, si es una puerta normal y corriente, se
                        desplazara en angulo de 90 grados para mostrar la
                        apertura
104                 var localRotation = doorPivot.localRotation;
105                 Quaternion initialRotation = localRotation;
106                 Quaternion targetRotation = Quaternion.Euler(
                        localRotation.eulerAngles.x, targetAngle,
                        localRotation.eulerAngles.z);
107
108                 while (timeElapsed < _animationTime)
109                 {
110                     var t = timeElapsed / _animationTime;  // Normaliza
                            el tiempo transcurrido
111                     doorPivot.localRotation = Quaternion.Lerp(
                            initialRotation, targetRotation, t);  // Interpola
                             entre la rotacion inicial y la rotacion objetivo
112                     timeElapsed += Time.deltaTime;  // Actualiza el
                            tiempo transcurrido
113                     yield return null;  // Espera hasta el proximo frame
114                 }
115                 doorPivot.localRotation = targetRotation;  // Asegura que
                         la rotacion final sea exacta
116                 break;
117             }
118         }
119         _dataReciever.RecieveDoorState(_isDoorOpen, nombreComponente,
                nombrePlacaPadre, CicloDN.fechaActual);
120     }
121 }
```

Listing 11: Código fuente completo del sensor de apertura

```
using UnityEngine;

public class SensorHumedad : MonoBehaviour
{
    public float humedadActual;
    private float _humedadAnterior;
    private float _horaActual;
    public string nombreComponente;
    public CicloDN cicloDn;
    private ISensorDataReciever _dataReciever;
    private string nombrePlacaPadre; // Variable para almacenar el nombre
        del padre

    void Start()
    {
        nombrePlacaPadre = transform.parent.name;
        cicloDn = FindObjectOfType<CicloDN>();
        _humedadAnterior = humedadActual;
        _dataReciever = GetComponentInParent<ISensorDataReciever>();
    }

    void Update()
    {
        humedadActual = CicloDN.HumedadActual;
        _horaActual = CicloDN.Hora;
        float hActualRedondeada = Mathf.Round(humedadActual * 10.0f) *
            0.1f;
        float hAnteriorRedondeada = Mathf.Round(_humedadAnterior * 10.0f)
            * 0.1f;

        bool enviarData = false;
        if (hAnteriorRedondeada != hActualRedondeada)
            enviarData = true;
        if (_horaActual >= 12 && _horaActual < 16)
            enviarData = true;
        if (enviarData)
        {
            _humedadAnterior = humedadActual;
            _dataReciever.RecieveHumedadData(hActualRedondeada,
                enviarData, nombrePlacaPadre, CicloDN.fechaActual);
        }
    }
}
```

Listing 12: Código fuente completo del sensor de humedad

```
using System;
using Unity.VisualScripting;
using UnityEngine;
using UnityEngine.Rendering;

public class SensorLuminosidad : MonoBehaviour
{
    public Light[] luces;
    public float luminosidadMinima = 10f;
    public float tiempoApagado;
    private ISensorDataReciever _dataReciever;
    private float currentLuminosity;
    public DetectionSensor detectionSensor; // Referencia al sensor de
        movimiento
    private float tiempoSinMovimiento;
    private CicloDN cicloDN;
    private float luminosidad;
    private string nombrePlacaPadre; // Variable para almacenar el nombre
         del padre
    private TimeSpan horaAnterior;

    void Start()
    {
        nombrePlacaPadre = transform.parent.name;
        luminosidad = 0;
        cicloDN = FindObjectOfType<CicloDN>(); // Obtener el script
            CicloDN
        if (cicloDN == null)
        {
            Debug.LogError("No se encontro un componente CicloDN en la
                escena.");
        }
        _dataReciever = GetComponentInParent<ISensorDataReciever>();
        if (GetComponent<BoxCollider>() == null)
        {
            Debug.LogError("El sensor de luminosidad no tiene un
                BoxCollider.");
            return;
        }

        detectionSensor = transform.parent.GetComponentInChildren<
            DetectionSensor>();
        if (detectionSensor == null)
        {
            Debug.LogError("El sensor de luminosidad no tiene un
                componente DetectionSensor.");
        }

        tiempoApagado = (5f * ((CicloDN.DuracionDiaMin * 60f / 24f) /
            3600f));
    }

    void Update()
    {
        bool hayJugadorDentro = detectionSensor != null &&
            detectionSensor._playerInside;
        foreach (Light luz in luces)
```

```
        {
            if (Input.GetKeyDown(KeyCode.Space) && hayJugadorDentro)
            {
                    luz.enabled = false;
            }
            if (cicloDN != null && !luz.enabled)
            {
                luminosidad = cicloDN.IntensidadLuminica;
            }
            if (hayJugadorDentro && !(CicloDN.Hora >= 8 && CicloDN.Hora
                <= 20) && !luz.enabled)
            {
                // Encender la luz si hay poca luz y el jugador esta
                    dentro
                luz.intensity = luminosidadMinima;
                luz.enabled = true;
                luminosidad = luz.intensity;
                tiempoSinMovimiento = 0f; // Reiniciar el contador si el
                    jugador esta dentro
                horaAnterior = CicloDN.horaFormateada;
            }
            else if (luz.intensity > 0f && !hayJugadorDentro)
            {
                // Reducir la intensidad gradualmente si la luz esta
                    encendida y el jugador NO esta dentro
                tiempoSinMovimiento += Time.deltaTime;
                if (tiempoSinMovimiento >= tiempoApagado)
                {
                    luz.enabled = false;
                }
            }
            if (hayJugadorDentro && (CicloDN.horaFormateada.Hours >=
                horaAnterior.Hours && CicloDN.horaFormateada.Minutes !=
                horaAnterior.Minutes && (CicloDN.horaFormateada.Seconds -
                horaAnterior.Seconds) > 30))
            {
                luz.enabled = false;
            }
        }
        _dataReciever.RecieveLuminosidadData(luminosidad, true,
            nombrePlacaPadre, CicloDN.fechaActual);
    }
}
```

Listing 13: Código fuente completo del sensor de luminosidad

```
using UnityEngine;

public class DetectionSensor : MonoBehaviour
{
    private PlayerMovementTracker _playerMovementTracker;
    private BoxCollider _sensorCollider; // Referencia al BoxCollider del sensor
    private ISensorDataReciever _dataReciever;
    private DetectionSensor sensorMovimiento;
    public bool _hayMovimientoCache;
    public bool _playerInside;
    private string nombrePlacaPadre; // Variable para almacenar el nombre del padre

    void Start()
    {
        nombrePlacaPadre = transform.parent.name;
        _dataReciever = GetComponentInParent<ISensorDataReciever>();
        _playerMovementTracker = FindObjectOfType<PlayerMovementTracker>();
        _sensorCollider = GetComponent<BoxCollider>(); // Obtener el BoxCollider

        if (_playerMovementTracker == null)
        {
            Debug.LogError("No se encontro el componente PlayerMovementTracker");
        }

        if (_sensorCollider == null)
        {
            Debug.LogError("El sensor de movimiento no tiene un BoxCollider.");
        }
    }

    private void Update()
    {
        if (_sensorCollider != null && _playerMovementTracker != null)
        {
            // Verificar si el jugador esta dentro del area del sensor
            _playerInside = _sensorCollider.bounds.Contains(_playerMovementTracker.transform.position);

            // Actualizar la variable _hayMovimientoCache
            _hayMovimientoCache = _playerInside && _playerMovementTracker.IsMoving;
        }
        _dataReciever.RecieveMovimientoData(_hayMovimientoCache, nombrePlacaPadre, CicloDN.fechaActual);
    }
}
```

Listing 14: Código fuente completo del sensor de movimiento

```
using UnityEngine;
public class SensorTemperatura : MonoBehaviour
{
    public float temperaturaActual;
    private float _temperaturaAnterior;
    private float _horaActual;
    public string nombreComponente;
    public CicloDN cicloDn;
    private ISensorDataReciever _dataReciever;
    private string nombrePlacaPadre; // Variable para almacenar el nombre del padre

    void Start()
    {
        nombrePlacaPadre = transform.parent.name;
        cicloDn = FindObjectOfType<CicloDN>();
        _temperaturaAnterior = temperaturaActual;
        _dataReciever = GetComponentInParent<ISensorDataReciever>();
    }

    void Update()
    {
        temperaturaActual = CicloDN.TempActual;
        _horaActual = CicloDN.Hora;
        float tActualRedondeada = Mathf.Round(temperaturaActual * 10.0f) * 0.1f;
        float tAnteriorRedondeada = Mathf.Round(_temperaturaAnterior * 10.0f) * 0.1f;

        bool enviarData = false;
        if (tAnteriorRedondeada != tActualRedondeada)
            enviarData = true;
        if (_horaActual >= 12 && _horaActual < 16)
            enviarData = true;
        //Debug.Log("Hora:" + _horaActual + "enviar_datos: " + enviarData);
        if (enviarData)
        {
            _temperaturaAnterior = temperaturaActual;
            _dataReciever.RecieveTempData(tActualRedondeada, enviarData, nombrePlacaPadre, CicloDN.fechaActual);
        }
    }
}
```

Listing 15: Código fuente completo del sensor de temperatura

```
using UnityEngine;

public class PlayerMovementTracker : MonoBehaviour
{
    public bool IsMoving { get; private set; } = false;
    private Vector3 lastPosition;

    private void Update()
    {
        if (transform.position != lastPosition)
        {
            IsMoving = true;
        }
        else
        {
            IsMoving = false;
        }

        lastPosition = transform.position;
    }
}
```

Listing 16: Código fuente completo del seguimiento del usuario

10.3. Código fuente del receptor de datos (placa por estancia)

```
using UnityEngine;
using System;
using System.Collections.Generic;
using System.Net.Sockets;
using System.Net.NetworkInformation; // Agregar para usar Ping
using System.Text;
using UnityEngine.Serialization;
using System.Linq;
using System.IO;
using Unity.VisualScripting;
using Object = UnityEngine.Object;
using Ping = System.Net.NetworkInformation.Ping; // Agregar para escribir
    en archivos

public class PadreReceiver : MonoBehaviour, ISensorDataReciever
{
    [Serializable]
    public class SensorData: ICloneable
    {
        public string hora;
        public float Temperatura;
        public string Puertas;
        public float Luminosidad;
        public bool Movimiento;
        public float Humedad;
        public object Clone()
        {
            return new SensorData
            {
                hora = this.hora,
                Temperatura = this.Temperatura,
                Puertas = this.Puertas,
                Luminosidad = this.Luminosidad,
                Movimiento = this.Movimiento,
                Humedad = this.Humedad
            };
        }
    }
    public int _serverPort = 5555;
    public string _serverIp = "192.168.1.41";
    public string _identificadorUsuario = "Nikiforov-00";

    [FormerlySerializedAs("CicloDn")] public CicloDN cicloDn;

    private bool _datosParaEnviar;

    private SensorData datosActuales = new SensorData();
    private SensorData ultimosDatos;
    private Dictionary<string, Dictionary<string, SensorData>>
        datosPorPlaca = new Dictionary<string, Dictionary<string,
        SensorData>>();


    void Start()
    {
        cicloDn = FindObjectOfType<CicloDN>();
```

```
54      }
55
56      public void RecieveTempData(float temperature, bool enviarData,
            string nombrePlaca, DateTime fecha)
57      {
58          ActualizarDatoPlaca(nombrePlaca, "Temperatura", temperature,
                enviarData, fecha);
59      }
60
61      public void RecieveDoorState(bool isOpen, string doorName, string
            nombrePlaca, DateTime fecha)
62      {
63          ActualizarDatoPlaca(nombrePlaca, "Puertas", doorName + "{" +
                isOpen + "}",
64              true, fecha);
65      }
66
67      public void RecieveHumedadData(float humedad, bool enviarData, string
             nombrePlaca, DateTime fecha)
68      {
69          ActualizarDatoPlaca(nombrePlaca, "Humedad", humedad, enviarData,
                fecha);
70      }
71
72      public void RecieveLuminosidadData(float luminosidad, bool enviarData
            , string nombrePlaca, DateTime fecha)
73      {
74          ActualizarDatoPlaca(nombrePlaca, "Luminosidad", luminosidad,
                enviarData, fecha);
75      }
76
77      public void RecieveMovimientoData(bool movement, string nombrePlaca,
            DateTime fecha)
78      {
79          ActualizarDatoPlaca(nombrePlaca, "Movimiento", movement, true,
                fecha);
80      }
81
82      private void ActualizarDatoPlaca(string nombrePlaca, string tipoDato,
             object valor, bool enviarData, DateTime fechaActual)
83      {
84          datosActuales.hora = CicloDN.horaFormateada.ToString(@"hh\:mm\:ss
                ");
85          switch (tipoDato)
86          {
87              case "Temperatura":
88                  datosActuales.Temperatura = (float)valor;
89                  break;
90              case "Puertas":
91                  datosActuales.Puertas = (string)valor;
92                  break;
93              case "Luminosidad":
94                  datosActuales.Luminosidad = (float)valor;
95                  break;
96              case "Movimiento":
97                  datosActuales.Movimiento = (bool)valor;
98                  break;
99              case "Humedad":
```

```
                datosActuales.Humedad = (float)valor;
                break;
        }
        if (ultimosDatos == null)
        {
            ultimosDatos = (SensorData)datosActuales.Clone();
            GuardarDatosActualesEnArchivo(nombrePlaca, fechaActual);
        }
        else
        {
            TimeSpan diferenciaTiempo = TimeSpan.Parse(datosActuales.hora
                ) - TimeSpan.Parse(ultimosDatos.hora);
            bool tiempoSuficiente = diferenciaTiempo.TotalSeconds >= 300;

            if (tiempoSuficiente || !SonDatosIguales(ultimosDatos,
                datosActuales))
            {
                ultimosDatos = (SensorData)datosActuales.Clone();
                GuardarDatosActualesEnArchivo(nombrePlaca, fechaActual);
            }
        }
    }

    private bool SonDatosIguales(SensorData datos1, SensorData datos2)
    {
        return datos1.Temperatura == datos2.Temperatura &&
               datos1.Luminosidad == datos2.Luminosidad &&
               datos1.Movimiento == datos2.Movimiento &&
               datos1.Humedad == datos2.Humedad;
    }

    private void EnviarDatosAlServidor(Dictionary<string, Dictionary<
        string, SensorData>> datosPorPlaca)
    {
        // Construir el mensaje en formato "habitacion hora;sensor1:
            valor1,sensor2:valor2,..."
        StringBuilder message = new StringBuilder();
        string nombrePlaca = "";
        foreach (var placa in datosPorPlaca)
        {
            nombrePlaca = placa.Key;
            var datosPlaca = placa.Value;
            foreach (var dato in datosPlaca)
            {
                string hora = dato.Key;
                SensorData sensorData = dato.Value;

                message.Append($"{nombrePlaca} {hora};");
                message.Append($"Temperatura:{sensorData.Temperatura},");
                message.Append($"Puertas:{sensorData.Puertas},");
                message.Append($"Luminosidad:{sensorData.Luminosidad},");
                message.Append($"Movimiento:{sensorData.Movimiento.
                    ToString().ToLower()},");
                message.Append($"Humedad:{sensorData.Humedad},");
                message.Remove(message.Length - 1, 1); // Eliminar la
                    ultima coma
                message.Append("\n");
            }
```

```
152         }
153
154         try
155         {
156             using (TcpClient client = new TcpClient(_serverIp,
                    _serverPort))
157             using (NetworkStream stream = client.GetStream())
158             {
159                 byte[] data = Encoding.UTF8.GetBytes(message.ToString());
160                 stream.Write(data, 0, data.Length);
161                 Debug.Log("Datos enviados al servidor.");
162             }
163         }
164         catch (SocketException socketEx)
165         {
166             if (socketEx.SocketErrorCode == SocketError.ConnectionRefused
                    )
167             {
168                 Debug.LogError("Error: El servidor rechazo la conexion.
                        Verifica la configuracion del servidor y del firewall.
                        ");
169             }
170             else if (socketEx.SocketErrorCode == SocketError.TimedOut)
171             {
172                 Debug.LogError("Error: Tiempo de espera agotado al
                        intentar conectarse al servidor. Verifica la
                        conectividad de red.");
173             }
174             else
175             {
176                 Debug.LogError("Error de conexion: " + socketEx.Message);
177             }
178             //GuardarDatosActualesEnArchivo(nombrePlaca);
179         }
180         catch (Exception e)
181         {
182             Debug.LogError("Error al enviar datos al servidor: " + e.
                    Message);
183             //GuardarDatosActualesEnArchivo(nombrePlaca);
184         }
185
186     }
187
188     private bool PingHost(string ipAddress)
189     {
190         Ping ping = new Ping();
191         PingReply reply = ping.Send(ipAddress);
192         return reply.Status == IPStatus.Success;
193     }
194
195     private void GuardarDatosActualesEnArchivo(string nombrePlaca,
            DateTime fechaActual)
196     {
197         string filePath = Path.Combine("/users/Daniil/Documents/GitHub/
                TFG/Datos/", $"datos_{_identificadorUsuario}.txt");
198
199         using (StreamWriter writer = new StreamWriter(filePath, true))
200         {
```

```
201                 writer.Write($"{nombrePlaca} {fechaActual}-");
202                 writer.Write($"Temperatura={ultimosDatos.Temperatura};");
203                 writer.Write($"Puertas={ultimosDatos.Puertas};");
204                 writer.Write($"Luminosidad={ultimosDatos.Luminosidad};");
205                 writer.Write($"Movimiento={ultimosDatos.Movimiento.ToString()
                        .ToLower()};");
206                 writer.Write($"Humedad={ultimosDatos.Humedad}");
207                 writer.WriteLine();
208             }
209         }
210 }
```

Listing 17: Código fuente completo del receptor de datos

10.4. Código fuente del ciclo diario

```
using System;
using UnityEngine;
using Random = UnityEngine.Random;

public class CicloDN : MonoBehaviour
{
    public enum Estaciones
    {
        Invierno,
        Primavera,
        Verano,
        Otono
    };

    public Estaciones EstacionSeleccionada = Estaciones.Invierno;
    public static float Hora = 0;
    public static TimeSpan horaFormateada;
    public static DateTime fechaActual = new DateTime(2024, 1, 1); //
        Comenzar el 1 de enero de 2024
    private float horaAmanecer = 6f;
    private float SolX;
    private float IniSolX;
    private float TempMinima;
    private float TempMaxima;
    public static float TempActual;
    private float[,] temperaturas = new float[,]
    {
        {12.5f, 7.8f},
        {20.1f, 14.8f},
        {27.3f, 22.4f},
        {16.4f, 12.1f},
    };
    private float HumedadMinima;
    private float HumedadMaxima;
    public static float HumedadActual;
    private float[,] humedades = new float[,]
    {
        {75f, 65f}, // Invierno (media: 70%)
        {65f, 55f}, // Primavera (media: 60%)
        {60f, 50f}, // Verano (media: 55%)
        {70f, 60f}, // Otono (media: 65%)
    };
    public GameObject Sol;
    public Light luzSolar; // Referencia al componente Light del sol
    public float intensidadMaxima; // Intensidad maxima de la luz solar
        al mediodia
    public float intensidadMinima; // Intensidad minima de la luz solar
        durante la noche
    public float IntensidadLuminica; // Propiedad publica para acceder a
        la intensidad
    public static float DuracionDiaMin;
    public float duracionDiaPorMinutos;

    void Start()
    {
        if (duracionDiaPorMinutos <= 0) duracionDiaPorMinutos = 1;
```

```
53          DuracionDiaMin = duracionDiaPorMinutos;
54          TempMaxima = temperaturas[(int)EstacionSeleccionada, 0];
55          TempMinima = temperaturas[(int)EstacionSeleccionada, 1];
56          TempActual = TempMinima;
57
58          HumedadMaxima = humedades[(int)EstacionSeleccionada, 0];
59          HumedadMinima = humedades[(int)EstacionSeleccionada, 1];
60          HumedadActual = HumedadMinima;
61
62          luzSolar = Sol.GetComponent<Light>();
63          if (luzSolar == null)
64          {
65              Debug.LogError("El GameObject del sol no tiene un componente
                    Light.");
66          }
67          IniSolX = (Hora * (-90)) * 12;
68          Sol.transform.localEulerAngles = new Vector3(IniSolX, 0, 0);
69      }
70
71      void Update()
72      {
73          Hora += Time.deltaTime * (24 / (60 * DuracionDiaMin)); //
                Obtencion de la hora en base a la duracion del dia
74          horaFormateada = TimeSpan.FromHours(Hora);
75          fechaActual = fechaActual.Date + horaFormateada;
76          if (Hora >= 24)
77          {
78              NuevosDatosHumedadTemperatura();
79          }
80
81          AjustarIntensidadLuzSolar();
82          TempActual = CalculoTemperatura();
83          HumedadActual = CalculoHumedad();
84          RotacionSol();
85      }
86
87      void AjustarIntensidadLuzSolar()
88      {
89          if (luzSolar != null)
90          {
91              float t = Mathf.InverseLerp(0f, 24f, Hora);
92              float intensidad;
93
94              if (Hora >= horaAmanecer && Hora < 12f) // Amanecer y manana
95              {
96                  intensidad = Mathf.Lerp(intensidadMinima,
                        intensidadMaxima/2,
97                      (Hora - horaAmanecer) / (12f - horaAmanecer));
98              }
99              else if (Hora >= 12f && Hora < 18f) // Mediodia y tarde
100             {
101                 intensidad = intensidadMaxima/2;
102             }
103             else // Noche
104             {
105                 intensidad = Mathf.Lerp(intensidadMaxima/2,
                        intensidadMinima, (Hora - 18f) / (24f - 18f));
106             }
```

```
107
108             intensidad = (Mathf.Round(intensidad * 10f) / 10f)/2f;
109             luzSolar.intensity = intensidad;
110             IntensidadLuminica = intensidad;
111
112             // Ajustar color para un amanecer/atardecer mas suave
113             if (Hora >= horaAmanecer && Hora < 8f) // Amanecer suave
114             {
115                 luzSolar.color = new Color(intensidad, intensidad * 0.8f,
                        intensidad * 0.6f);
116             }
117             else if (Hora >= 18f && Hora < 20f) // Atardecer suave
118             {
119                 luzSolar.color = new Color(intensidad, intensidad * 0.7f,
                        intensidad * 0.5f);
120             }
121             else if (Hora >= 8f && Hora < 18f) // Dia
122             {
123                 luzSolar.color = new Color(intensidad, intensidad * 0.9f,
                        intensidad * 0.8f); // Tono mas calido
124             }
125             else // Noche profunda
126             {
127                 luzSolar.color = new Color(intensidad * 0.5f, intensidad
                        * 0.5f, intensidad * 0.6f); // Tono azulado
128             }
129         }
130     }
131
132     void RotacionSol()
133     {
134         SolX = 15 * Hora;
135         Sol.transform.localEulerAngles = new Vector3(SolX, 0, 0);
136     }
137
138     float CalculoTemperatura()
139     {
140         float aux = 0;
141         if (Hora >= 0 && Hora < 4 && TempActual < TempMinima + 1)
142         {
143             aux = Mathf.Lerp(TempMinima, TempMinima + 1, (Hora - 0) / 4);
144         }
145         else if (Hora >= 4 && Hora < 8 && TempActual < TempMinima + 2)
146         {
147             aux = Mathf.Lerp(TempMinima, TempMinima + 2, (Hora - 4) / 4);
148         }
149         else if (Hora >= 8 && Hora < 12)
150         {
151             aux = Mathf.Lerp(TempMinima + 3, TempMaxima, (Hora - 8) / 4);
152         }
153         else if (Hora >= 12 && Hora < 16)
154         {
155             aux = TempMaxima;
156         }
157         else if (Hora >= 16 && Hora < 24)
158         {
159             aux = Mathf.Lerp(TempMaxima, TempMinima, (Hora - 16) / 8);
160         }
```

```
161
162         return aux;
163     }
164
165     float CalculoHumedad()
166     {
167         // Logica similar a CalculoTemperatura, pero ajustada para
                humedad
168         float aux = 0;
169         if (Hora >= 0 && Hora < 6 && HumedadActual < HumedadMinima + 2)
170         {
171             aux = Mathf.Lerp(HumedadMinima, HumedadMinima + 2, (Hora - 0)
                    / 6);
172         }
173         else if (Hora >= 6 && Hora < 12)
174         {
175             aux = Mathf.Lerp(HumedadMinima + 2, HumedadMaxima, (Hora - 6)
                    / 6);
176         }
177         else if (Hora >= 12 && Hora < 18)
178         {
179             aux = HumedadMaxima;
180         }
181         else if (Hora >= 18 && Hora < 24)
182         {
183             aux = Mathf.Lerp(HumedadMaxima, HumedadMinima, (Hora - 18) /
                    6);
184         }
185
186         return aux;
187     }
188
189     void NuevosDatosHumedadTemperatura()
190     {
191         Hora = 0;
192         fechaActual = fechaActual.AddDays(1);
193         TempMaxima += 0.2f * (Random.value > 0.5f ? 1 : -1);
194         TempMinima += 0.2f * (Random.value> 0.5f ? 1 : -1);
195         TempActual = TempMinima;
196         HumedadMaxima += 0.5f * (Random.value > 0.5f ? 1 : -1);
197         HumedadMinima += 0.5f * (Random.value > 0.5f ? 1 : -1);
198         HumedadActual = HumedadMinima;
199     }
200 }
```

Listing 18: Código fuente completo del ciclo diario

10.5. Código fuente de la rutina de usuario

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

public class PlayerMovementRoutine : MonoBehaviour
{
    public Transform[] waypoints;
    private float speed;
    private int currentWaypointIndex = 8; // Inicio en Habitacion 2.1
    private Coroutine routineCoroutine;
    private System.Random random = new System.Random();

    void Start()
    {
        transform.position = waypoints[8].position;
        speed = 10000000;
        routineCoroutine = StartCoroutine(SeguirRutina()); // Iniciar la
            rutina en Start
    }
        IEnumerator SeguirRutina()
    {
        while (true) // Bucle infinito para repetir la rutina cada dia
        {
            // Despertarse e ir al bano (6:00 - 8:00)
            yield return MoverAPunto(11, new int[] { 6, 5, 11 });
            yield return EsperarMinutos(5 + random.Next(10)); // 5-15
                minutos en el bano

            // Recoger la habitacion (10-20 minutos)
            yield return MoverAPunto(8, new int[] { 5, 6, 8 });
            yield return EsperarMinutos(10 + random.Next(10));

            // Preparar el desayuno (10-15 minutos)
            yield return MoverAPunto(2, new int[] { 6, 5, 4, 3, 1, 2 });
            yield return EsperarMinutos(10 + random.Next(5));

            // Desayunar (20-30 minutos)
            int desayunarEn = random.Next(2); // 0: Cocina, 1: Salon
            yield return MoverAPunto(desayunarEn == 0 ? 2 : 3,
                desayunarEn == 0 ? null : new int[] { 1, 3 });
            yield return EsperarMinutos(20 + random.Next(10));

            // Limpiar los platos (5-10 minutos, solo si desayuno en el
                salon)
            if (desayunarEn == 1)
            {
                yield return MoverAPunto(2, new int[] { 1, 2 });
                yield return EsperarMinutos(5 + random.Next(5));
            }

            // Descansar y ver la tele en el salon hasta la hora de comer
                (13:00 - 14:00)
            yield return MoverAPunto(3, new int[] { 1, 3 });
            yield return EsperarHasta(13f + random.Next(60) / 60f);

            // Ir al bano (opcional)
```

```
52              if (random.Next(2) == 0) // 50% de probabilidad de ir al bano
53              {
54                  yield return MoverAPunto(11, new int[] { 4, 5, 11 });
55                  yield return EsperarMinutos(5 + random.Next(10));
56                  yield return MoverAPunto(3, new int[] { 5, 4, 3 });
57              }
58
59              // Preparar la comida (30-60 minutos)
60              yield return MoverAPunto(2, new int[] { 1, 2 });
61              yield return EsperarMinutos(30 + random.Next(30));
62
63              // Comer en el salon (30-60 minutos)
64              yield return MoverAPunto(3, new int[] { 1, 3 });
65              yield return EsperarMinutos(30 + random.Next(30));
66
67              // Trabajar o estudiar en el salon (4 horas)
68              yield return MoverAPunto(3, new int[] { 1, 3 });
69              yield return EsperarHasta(17f);
70
71              // Ir al bano (opcional)
72              if (random.Next(2) == 0) // 50% de probabilidad de ir al bano
73              {
74                  yield return MoverAPunto(11, new int[] { 4, 5, 11 });
75                  yield return EsperarMinutos(5 + random.Next(10));
76                  yield return MoverAPunto(3, new int[] { 5, 4, 3 });
77              }
78
79              // Tiempo libre en el salon (1 hora)
80              yield return EsperarHasta(18f);
81
82              // Preparar la cena (30-60 minutos)
83              yield return MoverAPunto(2, new int[] { 1, 2 });
84              yield return EsperarMinutos(30 + random.Next(30));
85
86              // Cenar en el salon (30-60 minutos)
87              yield return MoverAPunto(3, new int[] { 1, 3 });
88              yield return EsperarMinutos(30 + random.Next(30));
89
90              // Tiempo libre en el salon (2 horas)
91              yield return EsperarHasta(22f);
92
93              // Ir a dormir (en la habitacion 2.1)
94              yield return MoverAPunto(8, new int[] { 4, 5, 6, 8 });
95
96              // Esperar hasta las 00:00 del dia siguiente
97              while (CicloDN.horaFormateada < TimeSpan.FromHours(24))
98              {
99                  yield return null;
100             }
101         }
102     }
103
104     IEnumerator MoverAPunto(int waypointIndex, int[] ruta = null)
105     {
106         // Si no se proporciona una ruta, generar una ruta por defecto al
                destino
107         if (ruta == null)
108         {
```

```
109             ruta = new int[] { waypointIndex };
110         }
111 
112         // Moverse a lo largo de la ruta en un solo frame
113         for (int i = 0; i < ruta.Length; i++)
114         {
115             int siguienteWaypoint = ruta[i];
116             while (transform.position != waypoints[siguienteWaypoint].
                    position)
117             {
118                 transform.position = Vector3.MoveTowards(
119                     transform.position,
120                     waypoints[siguienteWaypoint].position,
121                     speed * Time.deltaTime
122                 );
123                 yield return null; // Esperar al siguiente frame
124             }
125             currentWaypointIndex = siguienteWaypoint; // Actualizar el
                    indice actual
126         }
127     }
128 
129     IEnumerator EsperarHasta(float horaObjetivo)
130     {
131         TimeSpan horaObjetivoTS = TimeSpan.FromHours(horaObjetivo); //
                Convertir horaObjetivo a TimeSpan
132         while (CicloDN.horaFormateada < horaObjetivoTS)
133         {
134             yield return null;
135         }
136     }
137 
138     IEnumerator EsperarMinutos(float minutos)
139     {
140         TimeSpan tiempoInicial = CicloDN.horaFormateada; // Obtener la
                hora actual al inicio de la espera
141         TimeSpan tiempoObjetivo = tiempoInicial.Add(TimeSpan.FromMinutes(
                minutos)); // Calcular la hora objetivo
142 
143         while (CicloDN.horaFormateada < tiempoObjetivo)
144         {
145             yield return null;
146         }
147     }
148 
149     int[] ObtenerRutaPredefinida(int inicio, int destino)
150     {
151         switch (inicio)
152         {
153             case 0: // Recibidor1
154                 switch (destino)
155                 {
156                     case 1: return new int[] { 1 };       // Recibidor2
157                     case 10: return new int[] { 10 };     // Habitacion4.1
158                 }
159                 break;
160             case 1: // Recibidor2
161                 switch (destino)
```

```
                {
                    case 0: return new int[] { 0 };       // Recibidor1
                    case 2: return new int[] { 2 };       // Cocina1
                    case 3: return new int[] { 3 };       // Salon
                }
                break;
            case 2: // Cocina1
                return new int[] { 1 };                   // Recibidor2
            case 3: // Salon
                switch (destino)
                {
                    case 1: return new int[] { 1 };       // Recibidor2
                    case 4: return new int[] { 4 };       // Pasillo1
                }
                break;
            case 4: // Pasillo1
                switch (destino)
                {
                    case 3: return new int[] { 3 };       // Salon
                    case 5: return new int[] { 5 };       // Pasillo2
                    case 7: return new int[] { 7 };       // Habitacion1.1
                }
                break;
            case 5: // Pasillo2
                switch (destino)
                {
                    case 4: return new int[] { 4 };       // Pasillo1
                    case 6: return new int[] { 6 };       // Pasillo3
                    case 11: return new int[] { 11 };     // Aseo
                }
                break;
            case 6: // Pasillo3
                switch (destino)
                {
                    case 5: return new int[] { 5 };       // Pasillo2
                    case 8: return new int[] { 8 };       // Habitacion2.1
                    case 9: return new int[] { 9 };       // Habitacion3.1
                }
                break;
            case 7: // Habitacion1.1
                return new int[] { 4 };                   // Pasillo1
            case 8: // Habitacion2.1
                return new int[] { 6 };                   // Pasillo3
            case 9: // Habitacion3.1
                return new int[] { 6 };                   // Pasillo3
            case 10: // Habitacion4.1
                return new int[] { 0 };                   // Recibidor1
            case 11: // Aseo
                return new int[] { 5 };                   // Pasillo2
        }
        return new int[] { destino }; // No deberia ocurrir (mover
            directamente)
    }
}
```

Listing 19: Código fuente completo de la rutina del usuario

10.6. Código fuente en Python

```python
def insert_sensor_reading(self, user_id, sensor_type, value, room, hour):
    try:
        self.connect()
        value = float(str(value).replace(',', '.'))

        self.cursor.execute(
            """
            INSERT INTO sensor_readings (user_id, sensor_type, value,
               room, hour)
            VALUES (?, ?, ?, ?, ?)
            """,
            (user_id, sensor_type, value, room, hour)
        )
        self.connection.commit()
    except sqlite3.IntegrityError:
        pass
    except sqlite3.Error as e:
        print(f"Error al insertar lectura de sensor: {e}")
    finally:
        self.close()
```

Listing 20: Operación para insertar en la base de datos

Esta función esta diseñada para almacenar los datos de los sensores en la base de datos comentada anteriormente. Estos datos incluyen:

- ***user_id*:** Identificador del usuario que ha generado los datos.
- ***sensor_type:*** Tipo de sensor del que debemos almacenar los datos, los tipos pueden ser *temperature*, *humidity*, *movement*, entre otros.
- ***value*:** Valos que tiene el sensor.
- ***room*:** Estancia donde se ha obtenido ese valor.
- ***hour*:** Hora a la cual se ha generado el valor del sensor.

Por otro lado, cuando se conecta correctamente a la base de datos, convertimos los valores de texto a decimal, si existe, con la conversión de ',' a '.' y ejecutamos la *query* que almacena los datos. Los enviamos y, si hay algún error, gracias al tratado de datos, podremos identificarlo de manera precisa y rápida.

```python
from datetime import datetime
import os
import re
from Database_Manager import DatabaseManager

def process_data_file(file_path, db_manager, user_id):
    if not os.path.exists(db_manager.db_path):
        db_manager.create_tables()  # Crear tablas si no existen

    with open(file_path, 'r') as file:
        for line in file:
            # Expresion regular para extraer datos
            pattern = r"Placa(\w+) (\d{2}/\d{2}/\d{4} \d{2}:\d{2}:\d{2})-Temperatura=([\d,.]+);Puertas=(.*?);Luminosidad=(\d+);Movimiento=(\w+);Humedad=(\d+)"
            match = re.match(pattern, line)

            if match:
                room, date_str, temperature, doors, luminosity, movement, humidity = match.groups()
                date_time = datetime.strptime(date_str, '%d/%m/%Y %H:%M:%S')

                # Convertir valores a los tipos adecuados
                luminosity = int(luminosity)
                humidity = int(humidity)
                temperature = float(temperature.replace(",", "."))
                movement = 1 if movement.lower() == "true" else 0  # Convertir a 1 o 0

                # Insertar lecturas en la base de datos
                db_manager.insert_sensor_reading(user_id, "temperature", temperature, room, date_time)
                db_manager.insert_sensor_reading(user_id, "luminosity", luminosity, room, date_time)
                db_manager.insert_sensor_reading(user_id, "humidity", humidity, room, date_time)
                db_manager.insert_sensor_reading(user_id, "movement", movement, room, date_time)
                if doors:  # Insertar datos de puertas solo si existen
                    # Extraer el nombre de la puerta y el estado
                    door_name, state_str = doors.strip('{}').split('{')  # Elimina las llaves y divide la cadena
                    state = state_str.lower() == "true"  # Convertir a booleano

                    # Insertar en la tabla door_states
                    db_manager.execute_query(
                        """
                        INSERT INTO door_states (user_id, room, door_name, state, date)
                        VALUES (?, ?, ?, ?, ?)
                        """,
                        (user_id, room, door_name, state, date_time)
                    )

def process_data_files_in_folder(folder_path, db_manager):
```

```
    for filename in os.listdir(folder_path):
        if filename.startswith("datos_") and filename.endswith(".txt"):
            file_path = os.path.join(folder_path, filename)
            process_data_file(file_path, db_manager, filename)


ruta_datos = "C:\\Users\\Daniil\\Documents\\GitHub\\TFG\\Datos"
db_path = f"{ruta_datos}\\InformacionSistema.db"
db_manager = DatabaseManager(db_path)
process_data_files_in_folder(ruta_datos, db_manager)
db_manager.close()
```

Listing 21: Código fuente para almacenar los datos en la base de datos

Código en Python encargado de procesar los datos del archivo generado por Unity con la información de los sensores, donde:

- **Verifica la base de datos:** Comprueba que el fichero de la base de datos existe y, si no existe, lo crea desde cero.
- **Lectura del archivo y extracción de datos:** En el momento que abrimos el archivos en modo lectura, comienza a procesarse en primera instancia mediante una expresión regular que separa los datos en función del patrón especificado. Donde, si hay alguna coincidencia, extrae los valores y los asigna a variables individuales.
- **Conversión de tipos:** Como los datos nos llegan en formato de texto, hay que convertirlos para poder almacenarlos correctamente.
- **Insertar los datos en la base de datos:** Mediante la función de *insert_sensor_reading* insertamos los datos de cada sensor en la base de datos mediante la información del usuario (*user_id*).

```python
import sqlite3

class DatabaseManager:
    def __init__(self, db_path):
        self.db_path = db_path
        self.connection = None
        self.cursor = None

    def connect(self):
        self.connection = sqlite3.connect(self.db_path)
        self.cursor = self.connection.cursor()

    def close(self):
        if self.connection:
            self.connection.close()

    def create_tables(self):
        try:
            self.connect()
            with open("esquema_db.sql", "r") as f:
                self.cursor.executescript(f.read())
            self.connection.commit()
        except sqlite3.Error as e:
            print(f"Error al crear las tablas: {e}")
        finally:
            self.close()

    def execute_query(self, query, params=None):
        try:
            self.connect()
            if params:
                self.cursor.execute(query, params)
            else:
                self.cursor.execute(query)
            self.connection.commit()
            return self.cursor.fetchall()
        except sqlite3.Error as e:
            print(f"Error al ejecutar la consulta: {e}")
        finally:
            self.close()

    # --- Operaciones CRUD ---

    def create(self, table, data):
        columns = ", ".join(data.keys())
        placeholders = ", ".join("?" * len(data))
        query = f"INSERT INTO {table} ({columns}) VALUES ({placeholders})
            "
        self.execute_query(query, tuple(data.values()))

    def read(self, table, columns="*", condition=None):
        query = f"SELECT {columns} FROM {table}"
        if condition:
            query += f" WHERE {condition}"
        return self.execute_query(query)

    def update(self, table, data, condition):
```

```
        updates = ", ".join([f"{col} = ?" for col in data])
        query = f"UPDATE {table} SET {updates} WHERE {condition}"
        self.execute_query(query, tuple(data.values()))

    def delete(self, table, condition):
        query = f"DELETE FROM {table} WHERE {condition}"
        self.execute_query(query)

    # --- Metodo especifico para insertar lecturas de sensor ---

    def insert_sensor_reading(self, user_id, sensor_type, value, room,
        date):
        try:
            self.connect()  # Asegurarse de conectar antes de ejecutar la
                consulta
            value = float(str(value).replace(',', '.'))

            self.cursor.execute(
                """
                INSERT INTO sensor_readings (user_id, sensor_type, value,
                    room, date)
                VALUES (?, ?, ?, ?, ?)
                """,
                (user_id, sensor_type, value, room, date)
            )
            self.connection.commit()
        except sqlite3.IntegrityError:
            pass
        except sqlite3.Error as e:
            print(f"Error al insertar lectura de sensor: {e}")
        finally:
            self.close()
```

Listing 22: Código fuente del manejo de la base de datos

10.7. Esquema SQL para crear la base de datos

```sql
CREATE TABLE IF NOT EXISTS users (
    id INTEGER PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    password VARCHAR(255) NOT NULL,
    address VARCHAR(255),
    region VARCHAR(255),
    zip_code INTEGER,
    water_supply INTEGER,
    gas_supply INTEGER,
    light_supply INTEGER
);

CREATE TABLE IF NOT EXISTS sensor_readings (
    id INTEGER PRIMARY KEY,
    user_id INTEGER,
    sensor_type VARCHAR(255) CHECK(sensor_type IN ('movement', 'doors', '
        humidity', 'temperature', 'luminosity', 'sound')),
    value REAL,
    room VARCHAR(255),
    date DATETIME,
    UNIQUE(user_id, sensor_type, room, date),
    FOREIGN KEY (user_id) REFERENCES users(id)
);

CREATE TABLE IF NOT EXISTS supplies (
    id INTEGER PRIMARY KEY,
    name VARCHAR(255) CHECK(name IN ('water', 'light', 'gas')),
    unit VARCHAR(10) CHECK(unit IN ('m3', 'kWh')),
    cost_unit REAL,
    type VARCHAR(10) CHECK(type IN ('fix', 'variable'))
);

CREATE TABLE IF NOT EXISTS consumption (
    id INTEGER PRIMARY KEY,
    date DATETIME,
    supply_id INTEGER,
    value REAL,
    cost REAL,
    FOREIGN KEY (supply_id) REFERENCES supplies(id)
);

CREATE TABLE IF NOT EXISTS daily_consumption (
    id INTEGER PRIMARY KEY,
    date DATETIME,
    supply_id INTEGER,
    value REAL,
    cost REAL,
    FOREIGN KEY (supply_id) REFERENCES supplies(id)
);

CREATE TABLE IF NOT EXISTS period (
    id INTEGER PRIMARY KEY,
    name VARCHAR(255) CHECK(name IN ('month', 'bimonthly', 'quarter', '
        semester', 'annual')),
    initial_date DATETIME,
    final_date DATETIME
```

```
);

CREATE TABLE IF NOT EXISTS period_consumption (
    id INTEGER PRIMARY KEY,
    period_id INTEGER,
    supply_id INTEGER,
    total_value REAL,
    total_cost REAL,
    FOREIGN KEY (period_id) REFERENCES period(id),
    FOREIGN KEY (supply_id) REFERENCES supplies(id)
);

CREATE TABLE IF NOT EXISTS door_states (
    id INTEGER PRIMARY KEY,
    user_id INTEGER,
    room VARCHAR(255),
    door_name VARCHAR(255),
    state BOOLEAN,
    date DATETIME,
    FOREIGN KEY (user_id) REFERENCES users(id)
);
```

Listing 23: Esquema SQL de la base de datos

10.8. Código fuente para las placas Arduino

```
#include <Wire.h>
const int sensorAddress = 0x48; // Direccion del sensor (ejemplo)
void setup()
{
    Wire.begin(); // Iniciar el bus I2C
}

void loop()
{
    Wire.beginTransmission(sensorAddress); // Iniciar transmision al
        sensor
    Wire.write(0x00); // Enviar comando para leer datos (ejemplo)
    Wire.endTransmission();
    Wire.requestFrom(sensorAddress, 2); // Solicitar 2 bytes de datos
    if (Wire.available() == 2) {
    int valor = Wire.read() << 8 | Wire.read(); // Leer los datos
    // ... (procesar el valor del sensor)
}
```

Listing 24: Código fuente del protocolo I2C

Explicación Detallada:

- **Incluir la librería *Wire.h*:** Libreria que proporciona las funciones necesarias para utilizar el protocolo.
- **Dirección del sensor:** La línea 3 define una constante que almacena la dirección del sensor al que nos conectaremos.
- **Función *setup()*:** Se utiliza para inicializar el bus mediante la función Wire.begin(). Esto preparara a la placa hija para actuar en el bus y comunicarse con otros dispositivos.
- **Función loop():** Es el código principal de la lógica de la placa para la comunicación entre sensores y la placa. Las funciones que realiza son:
  - **Iniciar la transmisión:** La línea de *Wire.beginTransmission* inicial la comunicación con el sensor específico.
  - **Enviar comando:** *Wire.write* envía un byte de datos al sensor.
  - **Finalizar transmisión:** *Wire.endTransmission* finaliza la transmisión al sensor.
  - **Solicitar datos:** *Wire.requestFrom* solicita al sensor la cantidad de bytes a enviar por el sensor
  - **Leer datos:** Si el sensor ha enviado los datos, se leen los bytes mediante *Wire.read*. Estos bytes leídos se combinan en un único valor mediante un desplazamiento lógico.
  - **Procesar datos:** El valor, una vez leído del sensor, se procesa y utiliza según las necesidades que se requieran.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "TSSID";
const char* password = "Password";
const char* mqttServer = "broker.emqx.io"; // Direccion del broker MQTT

WiFiClient espClient;
PubSubClient client(espClient);

void setup() {
  WiFi.begin(ssid, password);
  client.setServer(mqttServer, 1883);
}

void loop() {
  if (!client.connected()) {
    reconnect();
  }
  client.loop();

  // ... (leer datos de los sensores)

  // Publicar datos en un topico MQTT
  client.publish("casa/habitacion1/temperatura", String(temperatura).
      c_str());
}
```

Listing 25: Código del protocolo MQTT

Explicación Detallada:

- **Incluir Librerías:**
  - *ESP8266WiFi.h:* Proporciona las funciones para contectarnos a una red Wi-Fi.
  - *PubSubClient.h:* Implementa el protocolo MQTT.
- **Credenciales de Red y *Broker*:** Almacenamos mediante las variables de *ssid* y *password* la información respecto a la red local. Por otro lado, el *mqttServer* almacena la dirección del *broker*.
- **Función *setup()*:** Iniciamos la conexión con la red y configura la dirección y puerto del broker MQTT.
- **Función *loop()*:**
  - **Verificar la Conexión:** *if (!client.connected()) reconnect()* comprueba si el cliente esta conectado al *broker* y, sino, volvemos a intentar conectarnos.
  - ***client.loop*:** Función que se llama periódicamente para mantener la conexión y procesar los mensajes.
  - **Publicar Datos:** publicamos el valor obtenido del sensor en MQTT.

10.9. Presupuesto detallado

Sensor	**Precio (€)**	**Cantidad**	**Total (€)**
Arduino Pro Mini 328 - 5V/16MHz	9,95	1	9,95
PIR (motion) sensor	9,95	1	9,95
SparkFun Sound Detector	10,95	1	10,95
SparkFun BME280 - Atmospheric Sensor Breakout	19,95	1	19,95
Wall Adapter Power Supply - 12VDC 2A	15,77	1	15,77
N-Channel MOFSET 60V 30A	1,46	2	2,92
10K Ohms Resistor	0,10	1	0,10
Water Level Sensor Module	0,78	1	0,78
Carbon Monoxide and Combustible Gas Sensor - MQ-9	1,60	1	1,60
Adafruit TSL2591 High Dynamic Range Digital Light Sensor	6,95	1	6,95
Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2	19,95	1	19,95
ESP8266-01 - Wifi Module	6,95	1	6,95
Logic Level Converter - Bi-Directional	2,95	1	2,95
DHT2/11 Humity and Temperature Sensor	9,95	1	9,95
Female DC Power adapater - 2.1mm jack to screw terminal block	2,00	1	2,00
Voltage Regulator 3.3V	0,55	1	0,55
Capacitor Ceramic 100nF	0,64	1	0,64
Electrolytic Decoupling Capacitors - 100uF/25V	0,27	1	0,27
USB Mini-B Cable with FTDI	3,11	1	3,11
BreadBoard	8,25	2	16,50
Jumper Wires Pack - M/M	1,95	3	5,85
Jumper Wires Pack - M/F	1,95	1	1,95
Male Headers Pack - Break-Away	0,66	2	1,32
Total			**150,91**

Cuadro 14: Componentes de una placa habitual por cada estancia

Sensor	**Precio (€)**	**Cantidad**	**Total (€)**
Arduino Pro Mini 328 - 5V/16MHz	9,95	1	9,95
12 V Solenoid Valve - 3/4"	7,95	5	39,75
Diode Rectifier - 1A 50V	0,11	5	0,55
PIR (motion) sensor	9,95	1	9,95
SparkFun Sound Detector	10,95	1	10,95
SparkFun BME280 - Atmospheric Sensor Breakout	19,95	1	19,95
Wall Adapter Power Supply - 12VDC 2A	15,77	1	15,77
N-Channel MOFSET 60V 30A	1,46	5	7,30
10K Ohms Resistor	0,10	6	0,60
Water Level Sensor Module	0,78	1	0,78
Carbon Monoxide and Combustible Gas Sensor - MQ-9	1,60	1	1,60
Adafruit TSL2591 High Dynamic Range Digital Light Sensor	6,95	1	6,95
Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2	19,95	1	19,95
ESP8266-01 - Wifi Module	6,95	1	6,95
Logic Level Converter - Bi-Directional	2,95	1	2,95
DHT2/11 Humity and Temperature Sensor	9,95	1	9,95
Female DC Power adapater - 2.1mm jack to screw terminal block	2,00	1	2,00
Voltage Regulator 3.3V	0,55	1	0,55
Capacitor Ceramic 100nF	0,64	1	0,64
Electrolytic Decoupling Capacitors - 100uF/25V	0,27	1	0,27
USB Mini-B Cable with FTDI	3,11	1	3,11
BreadBoard	8,25	2	16,50
Jumper Wires Pack - M/M	1,95	5	9,75
Jumper Wires Pack - M/F	1,95	1	1,95
Male Headers Pack - Break-Away	0,66	2	1,32
Total			**199,99**

Cuadro 15: Componentes de la placa en un aseo

Sensor	**Precio (€)**	**Cantidad**	**Total (€)**
Arduino Pro Mini 328 - 5V/16MHz	9,95	1	9,95
12 V Solenoid Valve - 3/4"	7,95	2	15,90
Diode Rectifier - 1A 50V	0,11	2	0,22
PIR (motion) sensor	9,95	1	9,95
SparkFun Sound Detector	10,95	1	10,95
SparkFun BME280 - Atmospheric Sensor Breakout	19,95	1	19,95
Wall Adapter Power Supply - 12VDC 2A	15,77	1	15,77
N-Channel MOFSET 60V 30A	1,46	2	2,92
10K Ohms Resistor	0,10	3	0,30
Water Level Sensor Module	0,78	1	0,78
Carbon Monoxide and Combustible Gas Sensor - MQ-9	1,60	1	1,60
Adafruit TSL2591 High Dynamic Range Digital Light Sensor	6,95	1	6,95
Adafruit SGP30 Air Quality Sensor Breakout - VOC and eCO2	19,95	1	19,95
ESP8266-01 - Wifi Module	6,95	1	6,95
Logic Level Converter - Bi-Directional	2,95	1	2,95
DHT2/11 Humity and Temperature Sensor	9,95	1	9,95
Female DC Power adapater - 2.1mm jack to screw terminal block	2,00	1	2,00
Voltage Regulator 3.3V	0,55	10	0,55
Capacitor Ceramic 100nF	0,64	10	0,64
Electrolytic Decoupling Capacitors - 100uF/25V	0,27	10	0,27
USB Mini-B Cable with FTDI	3,11	1	3,11
BreadBoard	8,25	2	16,50
Jumper Wires Pack - M/M	1,95	4	7,80
Jumper Wires Pack - M/F	1,95	1	1,95
Male Headers Pack - Break-Away	0,66	2	1,32
Total			**169,18**

Cuadro 16: Componentes de la placa en una cocina