MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

CLOUD-BASED PEOPLE COUNTING SYSTEM

Autor: Mariano Colmenar Cascón

Director: Álvaro Pérez Bello

Madrid

Agosto de 2021

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

.....Cloud – Based People Counting System.....

.....

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2020/2021. es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Mariano Colmenar Carcón Fecha: 22../ 08../ 2021

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Fecha: 23../ 08../ 2021

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

TRABAJO FIN DE MÁSTER

CLOUD-BASED PEOPLE COUNTING SYSTEM

Autor: Mariano Colmenar Cascón

Director: Álvaro Pérez Bello

Madrid

Agosto de 2021

SISTEMA CONTADOR DE PERSONAS BASADO EN LA NUBE

Autor: Colmenar Cascón, Mariano.

Director: Pérez Bello, Álvaro.

Entidad colaboradora: Altair Engineering Inc.

RESUMEN DEL PROYECTO

1. Introducción

1.1. Definición del problema

El proyecto consiste en el desarrollo de un sistema contador de personas basado en la nube. El dispositivo se diseñará desde cero y el resultado del proyecto será un prototipo del dispositivo, que tendrá plena funcionalidad.

La motivación para llevar a cabo el proyecto nace de un estudio de mercado de las tecnologías existentes para monitorizar la ocupación, llevado a cabo para el mercado español. En dicho estudio, se vio claramente que en España hay muy poca presencia de sistemas contadores de personas de bajo coste, lo que representa una oportunidad para lanzar un dispositivo contador de personas de bajo coste.

El dispositivo contador de personas de este proyecto estaba dirigido inicialmente a gimnasios, sin embargo, tras un estudio de mercado realizado en profundidad, se vio claramente que la industria con mayor potencial era la de venta minorista. Aun así, los gimnasios y oficinas se mantienen como industrias objetivo con el fin de diversificar los potenciales clientes.

La tecnología para conteo de personas de bajo coste tiene ventajas claras para los negocios y sus clientes:

Por un lado, los negocios podrán tener una visión más precisa de la demanda y puntos de alta ocupación en sus centros, permitiéndoles llevar a cabo proyecciones de demanda más precisas y ejecutar con éxito estrategias de negocio basadas en datos de ocupación.

Por otro lado, los clientes de dichos negocios también se beneficiarán de este sistema, ya que este les permitirá ahorrar tiempo mediante la toma de mejores decisiones basadas en datos de ocupación. Las ventajas de este sistema para los negocios y sus clientes han sido un factor clave en la motivación para llevar a cabo este proyecto.

1.2. Estado del arte

Existen varias formas de estimar la ocupación en un local. Las tecnologías más habituales están basadas normalmente en sensores luminosos o térmicos, WIFI, ancho de banda ultra, o cámaras:

- **Sensores de haz:** En las tecnologías de conteo de personas basadas en sensores de haz, el receptor y el transmisor están ubicados cada uno a un lado del punto de acceso. Cuando el haz detecta una persona caminando a través de él, la cuenta aumenta.

- **Contadores térmicos:** Esta tecnología mide la temperatura de la zona e identifica a las personas por medio de su calor corporal, y después genera imágenes usando radiación infrarroja.

- **Contadores monoculares 2D:** Estos sensores utilizan una sola lente para el conteo. Se instalan en el techo de la habitación, y detectan los objetos en movimiento. Normalmente combinan un algoritmo de detección de personas de aprendizaje profundo y un detector de objetos en movimiento.

- **Contadores WIFI:** Esta tecnología funciona por geolocalización siempre que el dispositivo esté conectado a la red WIFI.

- **Sensores de tiempo de vuelo:** En este tipo de sensores, se envía una señal a los objetos que se encuentran debajo del sensor, y en el instante en el que el reflejo de la señal rebota de vuelta en el sensor, se incrementa la cuenta.

- **Contadores estéreo 3D:** La principal diferencia que tienen estos contadores con respecto a los contadores monoculares es que en este caso se utilizan dos lentes. De forma que cada lente captura imágenes diferentes, y finalmente se construye una imagen con profundidad 3D combinando la información captada por ambas lentes.

1.3. Objetivos del proyecto

Los principales objetivos del proyecto son los siguientes:

- **Desarrollar un producto IoT conectado que sea escalable y tenga impacto:**

Este proyecto consiste en desarrollar un producto independiente basado en la nube. El producto se enfoca en resolver un problema del mundo real, que es el de la monitorización de la ocupación en negocios físicos. Además, se trata de un proyecto escalable, ya que es aplicable a prácticamente todos los negocios físicos, y tiene el potencial de tener un gran impacto en estos. El proyecto no solo abarca el diseño del sistema contador de personas, sino también el desarrollo de un prototipo tangible.

- **Desarrollar un producto a partir del cual se pueda crear un negocio en el futuro**

El propósito de este proyecto no solo es desarrollar un prototipo, sino también sentar las bases para la creación de un negocio real. El dispositivo contador de personas desarrollado en este proyecto representa una buena oportunidad de negocio, dada la ausencia de tecnología contadora de personas de bajo coste en el mercado y las ventajas que dicha tecnología supone para los negocios.

- **Potenciar la transformación de negocios físicos hacia un modelo basado en datos**

Tanto los negocios físicos como los digitales deben llevar a cabo una transición hacia un modelo basado en datos para mantenerse competitivos. Este proyecto servirá a los negocios físicos y les permitirá tomar decisiones basadas en datos. Les permitirá también calcular tasas de conversión para cada área específica de su negocio, identificar oportunidades únicas reconociendo las horas con picos de demanda para cada área, y optimizar su gestión de inventarios y de personal, dando lugar finalmente a una mayor rentabilidad.

- **Ayudar a los clientes de los negocios a ahorrar tiempo:**

El sistema permitirá a los clientes de los negocios disponer de la información de ocupación que necesiten para tomar decisiones más inteligentes sobre si ir a un centro u a otro, y sobre el mejor momento para ir, permitiéndoles utilizar su tiempo de forma más inteligente y ahorrar tiempo.

2. Metodología

El desarrollo del prototipo del sistema contador de personas se lleva a cabo en 4 fases:

- Diseño preliminar del prototipo
- Desarrollo del contador de personas de visión artificial con Raspberry Pi
- Conectar la Raspberry Pi a la nube con el software de Altair SmartWorks
- Análisis del presupuesto

2.1. Diseño preliminar del prototipo

Tras un profundo análisis de las opciones de hardware disponibles para el desarrollo del sistema, se toma la decisión de que la Raspberry Pi 4 es el mejor dispositivo para la implementación del sistema contador de personas, ya que proporciona una buena relación de un procesador potente a un bajo precio.

El siguiente paso en el diseño del hardware consiste en la elección de la cámara. Se decide que la opción más conveniente y rentable es la cámara para Raspberry Pi de 8MP.

Se compran también otros componentes para el correcto funcionamiento de la Raspberry Pi, como un ventilador, una carcasa, una tarjeta microSD, y un cable micro-HDMI. Estos componentes se explican con mayor nivel de detalle en el análisis del presupuesto. El hardware resultante se muestra en la siguiente figura:

Figura 1: Hardware del prototipo del dispositivo contador de personas

Con respecto al software, tras analizar diferentes posibilidades de software, se considera que la mejor opción es utilizar una combinación de un algoritmo de detección de objetos y un algoritmo de rastreo de objetos. Esto se consigue a través de paquetes de visión artificial de

OpenCV y programando en Python. El código utilizado para este prototipo está basado en una guía publicada en *pyimagesearch* de Adrian Rosenbrock, llamado "OpenCV People Counter" *[5]*. En la siguiente figura se muestra un esquema del funcionamiento del sistema contador de personas:

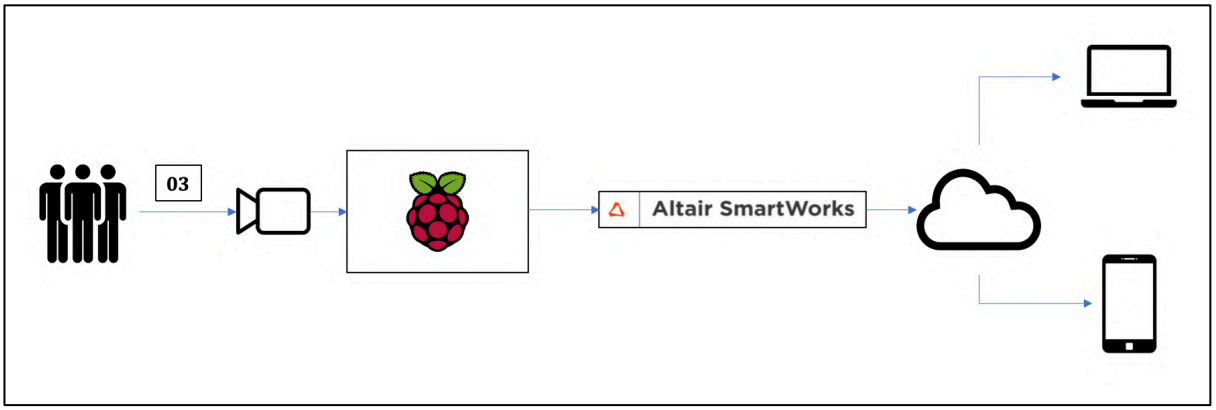

Figura 2: Esquema del funcionamiento del sistema

2.2. Desarrollo del contador de personas de visión artificial con Raspberry Pi

El contador de personas estará programado en Python, y utilizará paquetes de OpenCV *[5]*. Además, utilizará una combinación de un algoritmo de detección de objetos y un algoritmo de rastreo de objetos. Existen varias diferencias entre un detector de objetos y rastreador de objetos:

Detector de objetos:

El detector de objetos se utiliza para identificar los nuevos objetos entrando o saliendo del marco. Es muy pesado a nivel computacional. Para el prototipo, se prueban dos métodos de detección de objetos diferentes, con el fin de determinar cuál de los dos funciona mejor:

1. **Detector de personas de disparo único de aprendizaje profundo (MobileNetSSD):** Se trata de un algoritmo de detección de personas basado en un modelo entrenado, el cual es extremadamente preciso para la detección de personas, pero es muy pesado a nivel computacional. De forma que este algoritmo de detección de personas funciona perfectamente en un MacBook Pro (procesador: 2,7 GHz Dual-Core Intel Core i5) pero no funciona tan bien en la Raspberry Pi 4 (procesador: 1,5 GHz ARM Cortex-A72).
2. **Detector de objetos en movimiento:** Se trata de un algoritmo capaz de detectar cualquier objeto que se mueva dentro del marco. La precisión de este algoritmo es inferior, ya que no es específico para la detección de personas, sino que detecta cualquier tipo de objeto en movimiento. También requiere una mayor calibración antes de poder usarse. En cualquier caso, es significativamente menos pesado a nivel

computacional, de manera que supone la mejor opción para la Raspberry Pi. Los argumentos de entrada que se deben introducir en la calibración son las áreas de contorno máximas y mínimas, y el límite de binarización. Las áreas de contorno máximas y mínimas son los límites del tamaño que se consideran atribuibles a una persona, teniendo en cuenta la altura a la que está colocada la cámara. Por su parte, el límite de binarización es el gradiente de color mínimo aceptable entre la figura y el fondo requerido para la detección de un objeto. Cuanto menor sea el límite, más sensible será la detección.

Además, el detector de objetos en movimiento puede utilizarse con o sin sustracción de fondo:

- **Sin sustracción de fondo:** En este caso el fotograma inicial se toma como el marco de referencia. De forma que la detección de objetos se basa en las diferencias entre el fotograma actual y el marco de referencia. Sin embargo, este método puede dar lugar a problemas en la detección cuando en el fotograma inicial ya hay un objeto, dado que tomará dicho objeto como parte del fondo. También puede dar lugar a problemas cuando hay cambios en la iluminación del fondo.

- **Con sustracción de fondo:** En este caso el marco de referencia es dinámico, y es una media ponderada de todos los fotogramas hasta el fotograma actual. La detección de objetos se basa en las diferencias entre el fotograma actual y el marco de referencia dinámico (media ponderada de los fotogramas anteriores). Este método es óptimo porque se ajusta a los cambios en el fondo, como pueden ser cambios en la iluminación u objetos en el fondo.

Rastreador de objetos:

El rastreador de objetos se encarga de asignar un ID específico a cada persona en el marco. También se encarga de rastrear el objeto (persona) según se mueve por el marco, mediante un algoritmo de rastreo de centroides. El rastreador funciona calculando las distancias euclídeas entre objetos en el fotograma actual y objetos en el fotograma anterior. A continuación, asume que las parejas de objetos que tienen la menor distancia euclídea entre el objeto en el fotograma anterior y el objeto en el fotograma actual corresponden al mismo ID.

El algoritmo combinado de detección y rastreo de objetos se divide en dos fases:

1. **Detectando:** Esta fase solo se ejecuta cada N fotogramas, dado que el detector de objetos es más pesado a nivel computacional. En esta fase el algoritmo de detección de personas se ejecuta para identificar si nuevos objetos han entrado o abandonado el marco.
2. **Rastreando:** Para cada objeto detectado durante la fase de detección, se crea un rastreador para rastrear dicho objeto que se mueve por el marco. Dado que es más ligero a nivel computacional, este algoritmo se ejecuta en cada fotograma hasta llegar al fotograma N, en el cuál se ejecuta el algoritmo de detección de objetos (fase 1). Tras esto, se repite la secuencia.

Con respecto a las aplicaciones de este sistema combinado, se implementan dos tipos de lógica distintos, dependiendo de la ubicación física en la que se vaya a colocar la cámara:

- **Control de acceso:** Si la cámara se coloca en un punto de acceso, el propósito del contador de personas es mantener la cuenta de cuántas personas entran o salen a través de dicho acceso (por ejemplo: 6 personas han entrado y 4 han salido del departamento de moda de mujer).
- **Control de zona:** Si la cámara se coloca en el centro de una sección, el propósito del sistema será monitorizar cuántas personas se encuentran en dicha sección en cada instante (por ejemplo: hay 7 personas en el puesto de zapatería de hombre).

En la siguiente figura se muestra un diagrama de bloques del funcionamiento del sistema:

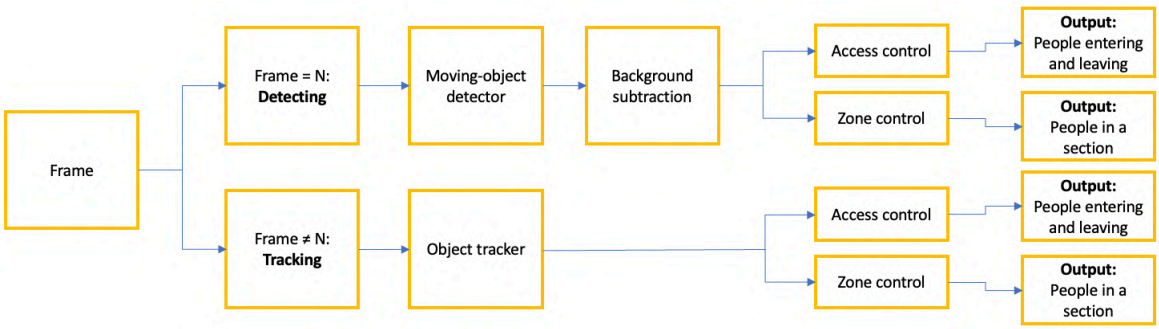

Figura 3: Diagrama de bloques del funcionamiento del sistema

2.3. Conectar la Raspberry Pi a la nube con el software de Altair SmartWorks

Para conectar el sistema a la nube, se utiliza el software de Altair SmartWorks, que ofrece tecnología IoT avanzada en un entorno de arquitectura abierta, el cuál es extremadamente útil para la aplicación específica de este proyecto.

En la memoria del proyecto se presentará una explicación detallada de cómo el dispositivo se conecta a la nube con Altair SmartWorks, así como una descripción de los comandos utilizados en el código.

A modo de introducción, SmartWorks es una plataforma de desarrollo de aplicaciones IoT de bajo código. Permite desarrollar aplicaciones IoT de manera sencilla, dado que ofrece las herramientas para montar la aplicación rápidamente *back-end*, así como el entorno y los elementos para el *front-end* y la visualización de datos por medio de *dashboards* en tiempo real. SmartWorks se utiliza en diferentes industrias, como por ejemplo equipamiento de construcción, bienes industriales, robótica, productos electrónicos de consumo, energía, y maquinaria pesada.

En este proyecto hay dos partes diferentes dentro de la plataforma de SmartWorks: *back-end* y *front-end*.

Back-end hace referencia a la parte que el usuario final no ve. Incluye conectar la Raspberry Pi a SmartWorks y crear los *spaces, things,* y *properties* necesarios para subir la información recogida por la cámara de la Raspberry Pi a la nube.

Front-end incluye la visualización y presentación de datos dentro de la plataforma SmartWorks. La visualización se consigue por medio de un *dashboard* creado con *Panopticon.* El *dashboard* resultante para la visualización de datos se muestra en la siguiente figura:

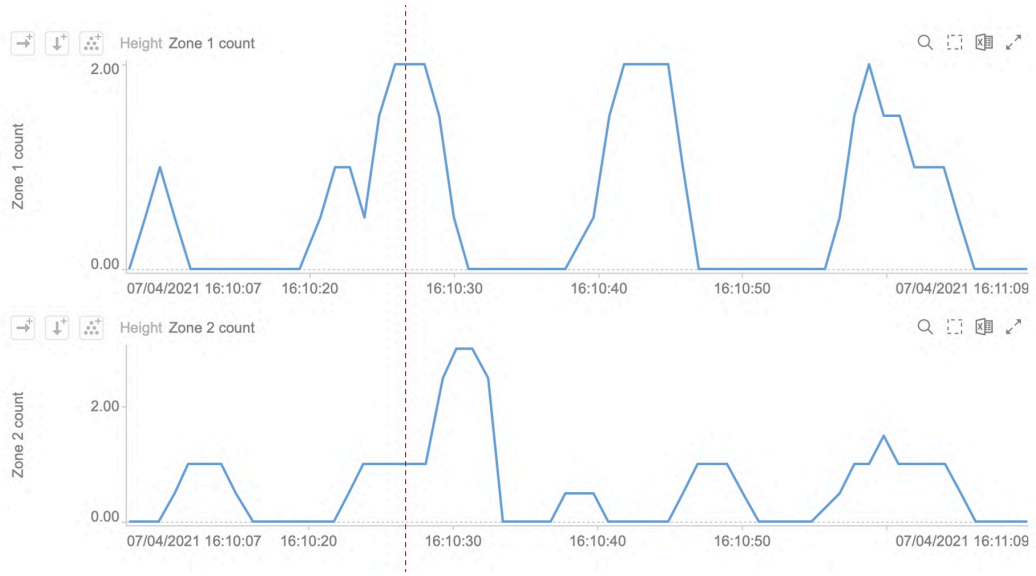

Figura 4: Dashboard utilizado para la visualización de datos

2.4. Análisis del presupuesto

Con respecto al presupuesto requerido para este prototipo, la lista de materiales se muestra en la siguiente tabla:

Nombre del componente	**Precio**
Kit LABISTS Raspberry Pi 4: - Raspberry Pi 4 RAM 4GB - MicroSD 32GB - Disipadores de calor - Ventilador - Carcasa - Cable micro HDMI - Cable 5V 3A	110€
Cámara LABISTS Raspberry Pi 8MP	30€
Total	**140€**

Tabla 1: Lista de materiales del prototipo

3. Resultados

3.1. Experimentos del sistema contador de personas

Se realizan algunos experimentos para probar el prototipo. El objetivo principal a alto nivel de los experimentos es determinar si el prototipo es lo suficientemente preciso, dado que esto significaría que es posible crear un producto de bajo coste capaz de monitorizar la ocupación

de un local con una buena precisión. Además, cada uno de estos experimentos está diseñado para lograr un objetivo específico. La siguiente tabla muestra una descripción de los diferentes experimentos y sus objetivos:

Experimento	Descripción	Objetivo
Hogar	El vídeo de este experimento está grabado en un hogar, que es una ubicación de muy baja ocupación. El sistema medirá el número total de personas en esa parte del hogar en un instante concreto.	Determinar la precisión del sistema contador de personas en una zona específica de una ubicación de baja ocupación (un hogar).
Aeropuerto-Zona	El vídeo de este experimento está grabado en un aeropuerto. El sistema medirá el número total de personas en una zona específica para un instante concreto.	Determinar la precisión del sistema contador de personas en una zona específica de una ubicación de alta ocupación (un aeropuerto).
Aeropuerto-Acceso	El vídeo de este experimento está grabado en un aeropuerto. En este caso se considera que el sistema está localizado en un punto de acceso, de forma que contará el número de personas entrando y saliendo de la zona.	Determinar la precisión del sistema contador de personas a la hora de contar el número de personas entrando y saliendo de una ubicación de alta ocupación.

Tabla 2: Descripción de los experimentos

En la siguiente tabla se muestra un resumen de los resultados de los experimentos:

Experimento	Situación testada	Precisión
1. Hogar	Ubicación de baja ocupación. Total de personas en una zona.	100%
2. Aeropuerto-Zona	Ubicación de alta ocupación. Total de personas en una zona.	80%
3. Aeropuerto-Acceso	Ubicación de alta ocupación. Personas entrando y saliendo de una zona.	80%

Tabla 3: Resultados de los experimentos

3.2. Modelo de centro comercial en SmartWorks

Para un mayor entendimiento de las funcionalidades de la plataforma de SmartWorks, y de las formas en las que puede ser útil para el alcance de este proyecto, se construye un modelo en SmartWorks con 21 centros comerciales.

Los centros comerciales están ubicados en diferentes partes de España. Cada centro tiene las siguientes variables:

- **Nombre:** nombre específico del centro comercial.
- **Provincia:** provincia española en la que se encuentra el centro comercial.
- **Coordenadas de latitud y longitud:** se deben introducir dichas coordenadas para que SmartWorks pueda localizar los centros en un mapa.
- **Ocupación:** número total de personas en todo el centro en un instante de tiempo concreto.
- **Capacidad:** máximo número de personas permitido en el centro.
- **Tasa de ocupación:** la ocupación en un momento concreto dividida entre la capacidad del centro.

Las variables "Nombre", "Provincia", "Latitud y longitud" no cambian, mientras que "Ocupación" y "Tasa de ocupación" cambian en función de la ocupación del centro.

Por otro lado, se considera que cada centro tiene 10 departamentos. Para cada departamento se modelan 2 dispositivos diferentes: un dispositivo contador de personas de *acceso*, y un dispositivo contador de personas de *zona*.

Finalmente, todos los elementos se juntan en el *dashboard* mostrado en la siguiente figura:

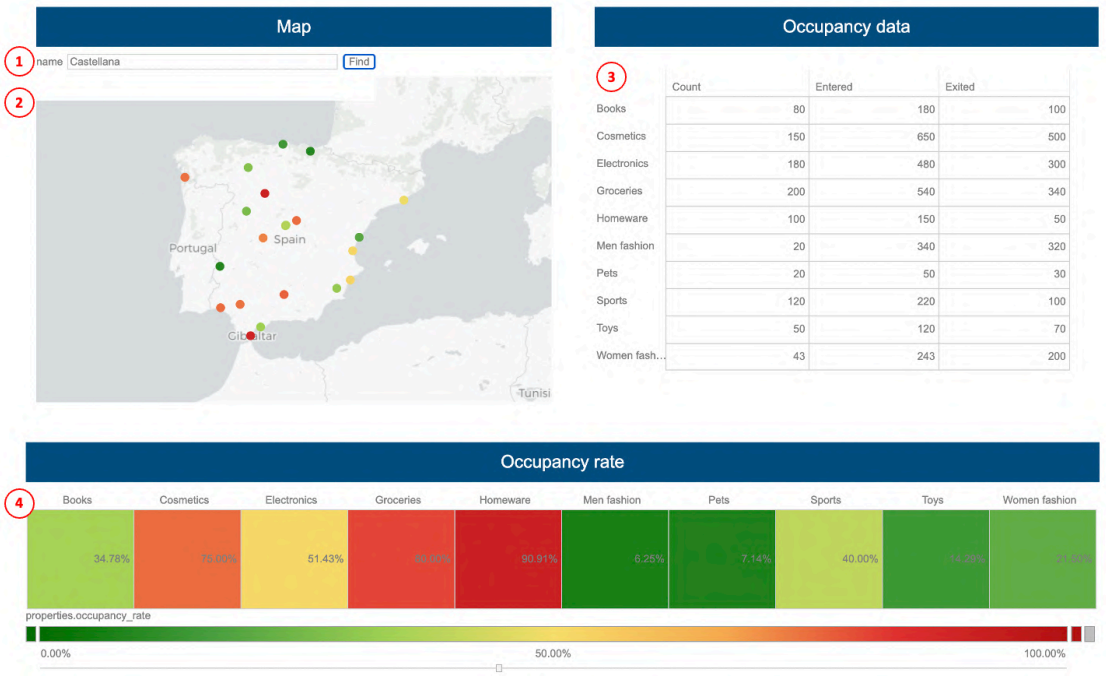

Figura 5: Dashboard de visualización del modelo de centro comercial en SmartWorks

El *dashboard* está formado por los siguientes elementos:

1. **Barra de búsqueda del centro comercial:** para que el usuario introduzca el nombre del centro comercial, de forma que la tabla (3) y el mapa de calor (4) muestren la información para ese centro comercial específico.
2. **Mapa:** para mostrar la ubicación de todos los centros comerciales, así como su tasa de ocupación según la escala de colores.
3. **Datos de ocupación recogidos:** muestra los datos de ocupación recopilados por los dispositivos de *acceso* y de *zona* en cada departamento del centro comercial introducido en la barra de búsqueda (1).
4. **Mapa de calor:** muestra la tasa de ocupación de los diferentes departamentos del centro comercial introducido en la barra de búsqueda (1) de forma visual según la escala de colores.

3.3. Dimensionamiento de mercado y motivación económica

Se lleva a cabo un dimensionamiento de mercado para obtener una visión del potencial económico y de la motivación económica del proyecto.

El mercado objetivo en el lanzamiento será el mercado español. Dado que el sistema es muy versátil y puede utilizarse en diferentes industrias, el dimensionamiento de mercado se lleva a cabo asumiendo un cliente grande en cada industria. Las industrias objetivo son:

- Grandes negocios de venta minorista:
    - Supermercados
    - Centros comerciales
    - Moda
    - Menaje del hogar
    - Deportes
- Gimnasios
- Oficinas y espacios de coworking

La siguiente tabla muestra un resumen del dimensionamiento de mercado:

Industria	Sub-sector	Cliente objetivo	Tamaño medio de tienda (m^2)	Nº de centros en España	Facturación anual potencial para el negocio (millones €)
Venta minorista	Supermercados	Mercadona	1.800	1.621	6,8
	Centros comerciales	El Corte Inglés	30.000	94	6,5
	Moda	Zara	2.200	449	2,3
	Menaje del hogar	IKEA	30.000	18	1,3
	Deportes	Decathlon	2.000	170	0,8
Gimnasios		Altafit	1.400	83	0,3
Oficinas		Utopicus	2.720	15	0,1
Total			**70.120**	**2.450**	**18,1**

Tabla 4: Dimensionamiento de mercado

4. Conclusiones

La principal conclusión extraída del proyecto es que es factible construir un dispositivo de bajo coste que sirva para monitorizar la ocupación con buena precisión.

El proyecto muestra las potenciales ventajas que un sistema contador de personas de bajo coste puede tener para negocios y espacios públicos. La falta de este tipo de tecnología a precios bajos representa una gran oportunidad para lanzar dicho producto.

Durante el desarrollo del prototipo, se vio claramente que una combinación de un detector de objetos y un rastreador de objetos era la mejor forma para solucionar los problemas de eficiencia. Después, en el proceso de adaptar el algoritmo para hacer que funcione en un dispositivo con un procesador menos potente (Raspberry Pi 4), se concluyó que el algoritmo de aprendizaje profundo debía ser reemplazado por un detector de objetos en movimiento, más ligero a nivel computacional. También se pudieron ver en los experimentos las ventajas de utilizar sustracción de fondo, permitiendo al sistema adaptarse a cambios en el entorno.

Con respecto a la conectividad a la nube, la plataforma de Altair SmartWorks utilizada en el proyecto ha hecho posible conectar el dispositivo a la nube y diseñar visualizaciones intuitivas. Algunas funcionalidades adicionales de SmartWorks también resultaron útiles a la hora de construir el modelo de centro comercial. El modelo permite al usuario monitorizar fácilmente los niveles de ocupación en los distintos departamentos de centros comerciales, avisando cuando la ocupación de un departamento se acerca a su máxima capacidad.

5. Referencias

Las siguientes referencias han sido utilizadas para la realización del proyecto, de forma que también están contenidas implícitamente en este resumen:

[1] Room occupancy estimation through WIFI, UWB, and Light Sensors mounted on doorways- Hessam Mohammadmoradi. University of Houston

[2] https://v-count.com/people-counting-technologies-a-comprehensive-guide/

[3] https://es.v-count.com/soluciones/contador-de-personas/

[4] https://www.footfallcam.com/Product/FootfallCam-3D-Max

[5] https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/

[6] https://www.hackster.io/phfbertoleti/counting-objects-in-movement-using-raspberry-pi-opencv-015ba5

[7] https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/

[8] https://www.elconfidencial.com/empresas/2021-01-06/supermercados-mercadona-carrefour-dia_2896439/

[9]https://www.expansion.com/empresas/distribucion/2019/05/12/5cd7e79822601d15048b45df.html

[10] https://www.elcorteingles.es/centroscomerciales/es/eci/centros?page=4

[11] https://elpais.com/economia/2019/03/13/actualidad/1552475635_649725.html

[12] https://es.wikipedia.org/wiki/Zara

[13] https://www.revistagq.com/noticias/articulo/ikea-mas-grande-del-mundo-superficie-cuanto-mide-filipinas-campos-de-futbol

[14] https://www.enterat.com/estilo/ikea-tiendas-espana.php

[15] https://www.business-standard.com/article/pti-stories/decathlon-to-meet-30-per-cent-local-sourcing-norms-in-5-years-114060100108_1.html

[16] https://www.palco23.com/equipamiento/decathlon-un-negocio-de-1660-millones-y-170-tiendas-que-pivota-al-centro-de-espana.html

[17] https://www.palco23.com/fitness/altafit-sigue-creciendo-pese-a-la-crisis-con-una-nueva-apertura-en-madrid.html

[18] https://www.elconfidencial.com/empresas/2019-10-22/utopicus-coworking-oficinas-flexibles-bra_2147603/

CLOUD-BASED PEOPLE COUNTING SYSTEM

Author: Colmenar Cascón, Mariano.

Director: Pérez Bello, Álvaro.

Collaborating entity: Altair Engineering Inc.

PROJECT SUMMARY

1. Introduction

1.1. Problem definition

The project consists of the development of a cloud-based people counting device. The device will be designed from scratch and the result of the project will be a prototype of the device, which will have full functionality.

The motivation for carrying out the project was found in a market research of existent occupancy tracking technologies in the Spanish market. In the research, it was clearly visible that there is little presence of low-cost people counting systems in Spain, which is an indicator of an opportunity to launch a low-cost people counting device.

The people counting device was initially targeted to gyms, but after a thorough market research, it was clear that the industry with the highest potential was retail. Although gyms and offices would also be target industries for diversification purposes.

Additionally, low-cost people counting technology has some clear advantages for businesses and their customers.

Businesses would have a more accurate vision of their demand and high-occupancy spots in their stores, allowing them to carry out more accurate demand forecasts and to deploy successful business strategies based on occupancy data.

Businesses' customers will also benefit from this system, allowing them to save time by making decisions on whether to go to one business on another based on occupancy data. The advantages for both stakeholders was a key factor in the motivation for carrying out this project.

1.2. State of the art

There are several ways of estimating the occupancy in a room. The most common technologies are usually based on either light and thermal sensors, WIFI, ultra band-width, or cameras:

- **Beam sensors:** In people counting technologies based on beam sensors, receiver and transmitter are located side by side in the access point. When the beam detects a person walking through it, the count increases.

- **Thermal counters:** This technology measures the room temperature and identifies people through their body heat, and then generates images using infrared radiation.

- **2D monocular counters:** These sensors use a single camera lens for counting. They are installed on the room ceiling, and they detect moving objects. They usually combine a deep learning people detector algorithm and a moving object detector.

- **WIFI counters:** This technology works through geolocation while the WIFI is enabled and there is access to it.

- **Time of flight sensors:** In this type of sensors, a signal is sent to objects under the sensor and then increase the counter when the reflection of the signal bounces back to the sensor.

- **3D stereo counters:** The main difference from monocular counters is that in 3D stereo counters 2 camera lenses are used. This way, each lens captures different images, and when the information captured by both lenses is combined, an image with 3D spatial depth is constructed.

In conclusion, the existent technology for counting people is expensive and the costs outweigh the benefits.

1.3. Project objectives

The main objectives of this project are the following:

- **Develop an IoT connected product that is scalable and will have an impact:**

This project consists of developing an independent, cloud-based product. The product addresses a real-life issue, which is keeping track of the occupancy in physical businesses. Additionally, this project is scalable, since it is applicable to almost every physical business, and it is likely to have an impact. Furthermore, this project not only involves the design of the people counting system, but also the development of a real, tangible product.

- **Develop a product from which a real business could be built in the future:**

The purpose of this project is not only to develop a prototype, but to set the bases for creating a real business in the future. Therefore, the people counting device developed in this project is a good business opportunity due to the lack of low-cost people counting technology in the market and the advantages of this technology for businesses.

- **Boost the transformation of physical businesses towards a data-driven model**

Both physical and digital businesses need to shift towards a data-driven model in order to stay competitive. This project will serve physical businesses and allow them to make data-driven decisions. It will enable businesses to calculate conversion rates of each specific physical area of their business, identify unique opportunities by recognizing peak demand hours for each area, and optimize their stock and staff management, eventually leading to higher profitability and business health.

- **Help the businesses' customers save time**

This system will give the businesses' clients the occupancy information they need to make smarter decisions on whether to go to one business or another one, and when to go, in order to make a smart use of their time. Once again, this system will allow them to make data-driven decisions.

2. Methodology

The development of the occupancy tracking system prototype consists of 4 main phases:

- Preliminary design of the prototype
- Development of the Computer Vision people counter with Raspberry Pi
- Connecting the Raspberry Pi to the cloud using Altair SmartWorks IoT software
- Analysis of the budget

2.1. Preliminary design of the prototype

After thoroughly analyzing the available hardware options for developing such a system, a decision was made that the Raspberry Pi 4 was the best device to implement an occupancy tracking system, given it provides a good tradeoff of a fairly powerful processor at a very low price.

The next step in the hardware design was to choose a camera, the most affordable and convenient option was the Raspberry Pi 8MP camera.

Some other components were also purchased for the correct functioning of the Raspberry Pi, such as a fan, a case, a microSD card, and a micro-HDMI cable. These components will be explained with more detail in the budget analysis. The resulting hardware is shown in the following figure:

Figure 1: View of the people counting device prototype

Regarding software, after analyzing all the software possibilities, it was considered that the best option is a combination of an object detection algorithm and an object tracking algorithm. Furthermore, this is achieved through Computer Vision OpenCV packages and coded in

Python. The code used for this prototype is based on a guide posted on *pyimagesearch* by Adrian Rosenbrock, which is called "OpenCV People Counter". *[5]*

A schematic view of the functioning of the system is shown in the following figure:

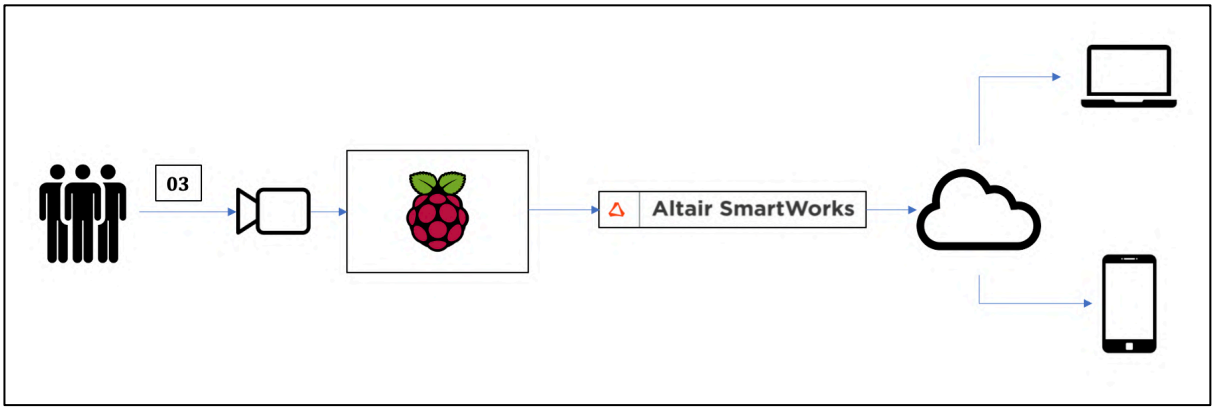

Figure 2: Schematic view of the functioning of the system

2.2. Development of the Computer Vision people counter with Raspberry Pi

As mentioned earlier, the people counter will be coded in Python, and will use OpenCV packages. *[5]* Furthermore, it will use a combination of an object detection and an object tracking algorithm. There are several differences between an object detector and an object tracker:

Object detector:

An object detector is used to identify any new objects entering or leaving the frame. It is very heavy at a computational level. For this prototype two different object detection methods were tried in order to determine which one works best:

1. **Deep learning single shot people detector (MobileNetSSD)**: This is a people detection algorithm based on a trained model, which is extremely accurate for detecting people, but is very heavy at a computational level. Therefore, this object detector worked perfectly on a MacBook Pro (processor: 2,7 GHz Dual-Core Intel Core i5) but did not work as well on the Raspberry Pi 4 (processor: 1,5 GHz ARM Cortex-A72).
2. **Moving-object detector:** This is an algorithm used for detecting any object which is moving in the frame. The accuracy of this algorithm is lower, given it is not specific for detecting people, but rather used for detecting any kind of moving object. It also requires a more thorough calibration before using it. However, it is significantly less heavy at a computational level, therefore it is the best option for the Raspberry Pi. The

two input arguments that should be entered upon calibration are the minimum and maximum contour area, and the binarization threshold. The minimum and maximum contour areas are the size limits that we consider that can be attributed to a person, considering the height at which the camera is located. The binarization threshold, on the other hand, is the minimum acceptable color gradient between the figure and the background required for detecting an object. The lower the threshold, the more sensible the detection.

Additionally, the moving object detector can be used with or without background subtraction:

- **Without background subtraction:** In this case the initial frame is taken as the reference frame. Therefore, the object detection is based on the differences between the current frame and the reference frame. However, this method may lead to problems in the detection whenever there already is an object in the initial frame, since it takes the given object as part of the background. It may also cause problems whenever there are changes in the lighting of the background.

- **With background subtraction:** In this case the reference frame is dynamic, and it is a weighted average of all the frames up to the current frame. The object detection is based on the differences between the current frame and the reference frame (weighted average of all the previous frames). This method is optimal because it adjusts itself to changes in the background, such as changes in the lighting or objects in the background.

Object tracker:

The object tracker oversees assigning a specific ID to each person in the frame. It is also in charge of tracking the object (person) as it moves around the frame, through a centroid tracking algorithm. The tracker works by calculating the Euclidean distances between objects in the current frame and objects in the previous frame. Then, it assumes that couples of objects that have the minimum Euclidean distance between the object in the previous frame and the object in the current frame belong to the same object ID.

The combined object detection and tracking algorithm is divided in two phases:

1. **Detecting:** This phase is only executed every N frames, given the object detector is very heavy at a computational level. In this phase the object detection algorithm is executed to recognize if any new objects have entered or left the frame.
2. **Tracking:** For each object detected during the detection phase, an object tracker is created to track that object that moves around the frame. Given it is less heavy at a computational level, this object tracking algorithm is executed every frame until the N-th frame is reached, in which the object detection algorithm is then executed (phase 1). After that, the whole sequence is repeated.

Regarding the application of this combined system, there will be two different varieties of logic to be implemented, depending on the physical location in which the camera is set:

- **Access control:** If the camera is placed at an access point, the purpose of the people counter would be to keep the count of how many people enter or leave through that access (e.g., 6 people entered and 4 left the men's fashion department).
- **Zone control:** If the camera is set in the middle of a section, the purpose would be to keep track of how many people are in that section in each moment (e.g., there are 4 people at the men's shoes stand).

A block diagram of the functioning of the system is shown in the following figure:

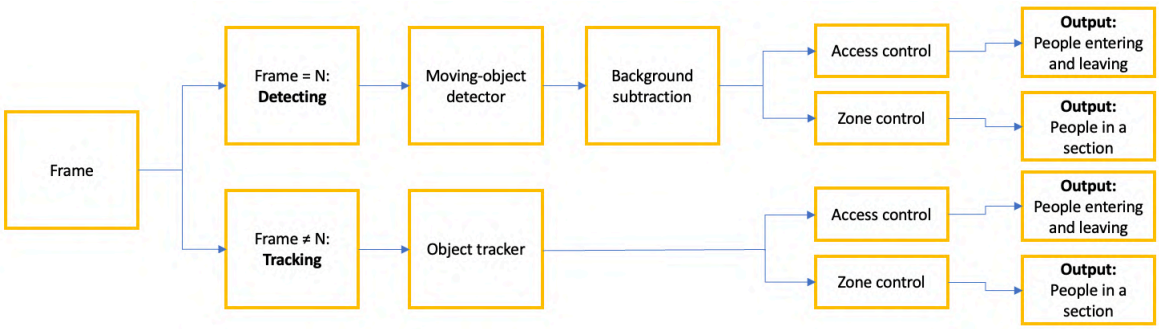

Figure 3: Block diagram of the system functioning

2.3. Connecting the Raspberry Pi to the cloud using Altair SmartWorks IoT software

To connect the system to the cloud, Altair SmartWorks products will be used. Altair SmartWorks delivers advanced IoT technology within an open-architecture environment, which will be extremely useful for our specific application.

A detailed explanation will be presented of how the device will be connected to the cloud through Altair SmartWorks, and a description of the commands used in the script.

As an introduction, SmartWorks is a low-code IoT application development platform. It allows to develop IoT applications in a simple manner, given it provides the tools to rapidly build the application backend, as well as the environment and elements for the frontend and data visualization through real-time dashboarding. SmartWorks is used in various industries, such as building equipment, industrial goods, robotics, consumer electronics, energy, and heavy machinery.

For this project there will be 2 different parts within the SmartWorks platform: back-end and front-end.

Back-end refers to the part that the final user will not see. It includes connecting the Raspberry Pi to SmartWorks and creating the necessary *Spaces*, *Things,* and *Properties* to upload the data collected through the Raspberry Pi camera to the cloud.

Front-end includes the data visualization and presentation within the SmartWorks platform. The visualization is achieved by means of a dashboard created with *Panopticon*. The resulting dashboard is shown in the following figure:

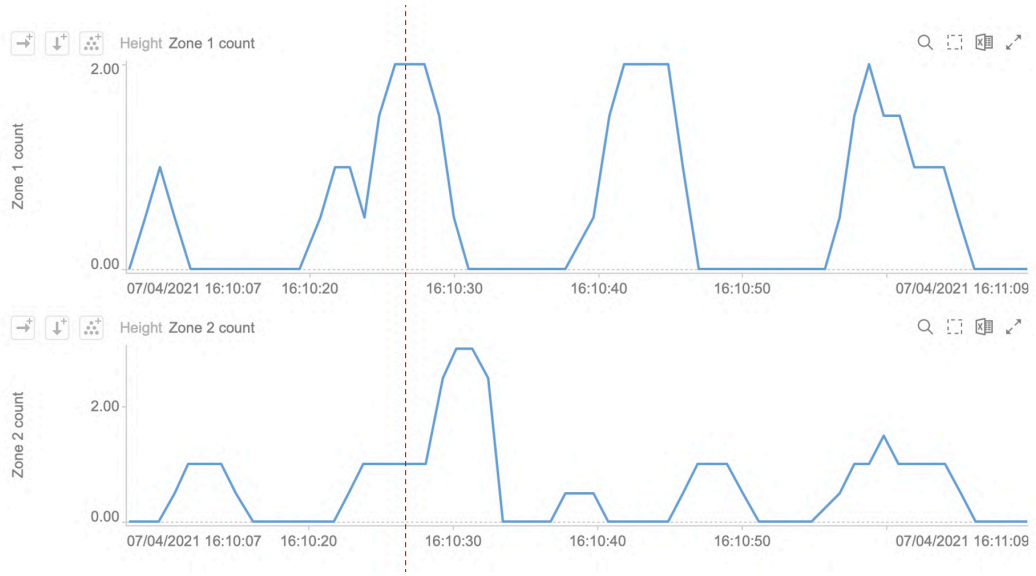

Figure 4: Dashboard used for data visualization

2.4. Analysis of the budget

Regarding the budget used for this prototype, the bill of materials is shown in the following table:

Component name	Price
LABISTS Raspberry Pi 4 Kit: - Raspberry Pi 4 RAM 4GB - MicroSD card 32GB - Heat sinks - Fan - Case - Micro HDMI cable - 5V 3A cable	110€
LABISTS Raspberry Pi Camera 8MP	30€
Total	**140€**

Table 1: Bill of materials of the prototype

3. Results

3.1. People counting system experiments

Some experiments were run to test the prototype. The main high-level objective of the experiments is to determine whether the prototype is accurate enough, since it would mean that it is possible to create a low-cost product that can track the occupancy level with a good

accuracy. Furthermore, each of the experiments is designed to meet a low-level objective. A description of the different experiments and their objectives is shown in the following table:

Experiment	Description	Objective
Home	The video in this experiment is recorded at home, which is a very low-occupancy location. The system will measure the total number of people in that part of the house at a particular moment.	Determine the accuracy of the occupancy tracking system in a specific area of a low-occupancy location (a home).
Airport-Zone	The video in this experiment is recorded at an airport. The system will measure the total number of people in a specific zone at a given moment.	Determine the accuracy of the occupancy tracking system in a specific area of a high-occupancy location (an airport).
Airport-Access	The video in this experiment is recorded at an airport. In this case the system will be considered to be located at an access point, therefore it will count the number of people entering and exiting the area.	Determine the accuracy of the occupancy tracking system when counting the number of people entering and leaving a high-occupancy location.

Table 2: Description of the experiments

A summary of the results is shown in the following table:

Experiment	Situation Tested	Accuracy
1. Home	Low-occupancy location. Total people in an area.	100%
2. Airport-Zone	High-occupancy location. Total people in an area.	80%
3. Airport-Access	High-occupancy location. People entering and exiting an area.	80%

Table 3: Results of the experiments

3.2. Shopping centre SmartWorks model

To gain a deeper understanding of the functionalities of the SmartWorks platform, and the ways in which it can be useful for the scope of this project, a model has been built in SmartWorks with 21 shopping centres.

The shopping centres are located in different parts of Spain. Each shopping centre has the following variables:

- **Name:** specific name of the shopping centre.
- **Province:** Spanish province in which the shopping centre is located.
- **Latitude and longitude coordinates:** these coordinates must be input in the system so that SmartWorks is able to locate them in a map.

- **Occupancy:** total number of people in the entire shopping centre at a particular moment in time.
- **Capacity:** maximum number of people allowed in the centre.
- **Occupancy rate:** the occupancy for a particular moment in time divided by the capacity of the centre.

The variables "Name", "Province", "Latitude and Longitude" do not change, whereas "Occupancy" and "Occupancy rate" do change depending on the occupancy of the centre.

Additionally, it has been considered that each shopping centre has 10 different departments. For each department, two different devices have been modelled: an *access* people counting device, and a *zone* people counting device. The reason for this is to take advantage of the full potential of the prototype, since it is capable of counting the total number of people in an area (*zone* type) and counting the number of people that have entered and exited an area (*access* type). The information collected by both devices is contrasted, since the total number of people counted by the *zone* device should be equal to the number of people that have entered minus the number of people that have exited the area, which are counted by the *access* device.

All elements are finally put together in a dashboard. A screenshot of the resulting dashboard is shown in the following figure:

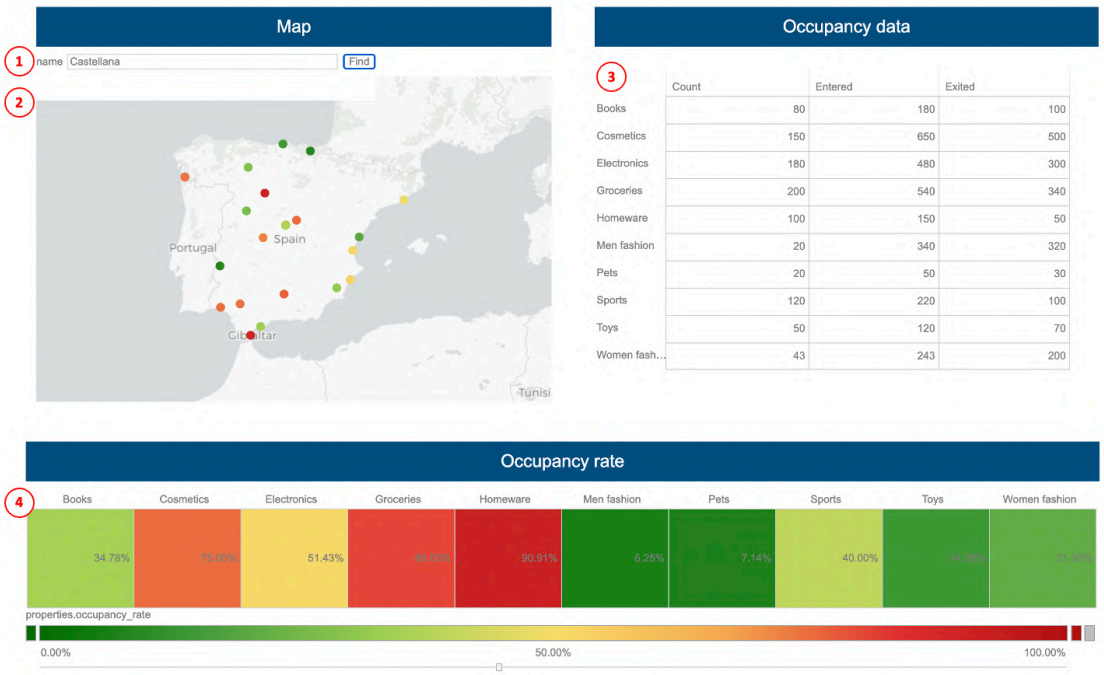

Figure 5: Visualization dashboard for the shopping centre SmartWorks model

The dashboard consists of the following elements:

1. **Shopping centre searchbar:** for the user to type in the name of the shopping center, then the table (3) and heatmap (4) will show the information for that particular shopping centre.
2. **Map:** to display the location of all the shopping centers, as well as their occupancy rate according to the color scale.
3. **Collected occupancy data:** shows the occupancy data collected by the *zone* and *access* devices in each department of the shopping centre typed in the searchbar (1).
4. **Heatmap:** shows the occupancy rate of the different departments of the shopping centre typed in the searchbar (1) in a visual way according to the color scale.

3.3. Market sizing and economic motivation

A market sizing is carried out to have a view of the full economic potential and economic motivation of this project.

The target market at launch would be the Spanish market. Given the system is very versatile and can be used in different industries, the market sizing will be carried out assuming one large client in each industry. The target industries are:

- Large retail businesses:
    - Groceries
    - Malls
    - Fashion
    - Homeware
    - Sports
- Gyms
- Office and coworking companies

A summary of the market sizing is shown in the following table:

Industry	Sub-sector	Target client	Average size (m^2)	Number of stores in Spain	Potential yearly revenue for the business (million €)
Retail	Groceries	Mercadona	1.800	1.621	6,8
	Malls	El Corte Inglés	30.000	94	6,5
	Fashion	Zara	2.200	449	2,3
	Homeware	IKEA	30.000	18	1,3
	Sports	Decathlon	2.000	170	0,8
Gyms		Altafit	1.400	83	0,3
Offices		Utopicus	2.720	15	0,1
Total			**70.120**	**2.450**	**18,1**

Table 4: Market sizing

4. Conclusions

The main conclusion drawn from the project is that it is feasible to build a low-cost device which can track occupancy with a good accuracy.

Furthermore, the project shows the potential advantages for businesses and public spaces of implementing a low-cost people counting system. The lack of this type of technology at affordable prices is a clear indicator of an opportunity to launch such a product.

While developing the prototype, it was clear that a combination of an object detector and an object tracker was the best possible approach to tackle efficiency issues. Then, in the process of adapting the algorithm to make it work in a device with a less powerful processor (Raspberry Pi 4), it was concluded that the Deep Learning Single Shot Detector had to be replaced with a Moving Object Detector, which, although less accurate, it is much lighter at a computational level, and more convenient for the Raspberry Pi. Additionally, the advantages of using dynamic background subtraction were visible in the experiments, allowing the system to adapt to changes in the environment.

Regarding cloud connectivity, the Altair SmartWorks platform used in the project has made it possible to connect the device to the cloud and design user-friendly visualizations. Further functionalities of SmartWorks were also useful when building the shopping centre model. The model allows the user to easily monitor occupancy levels in the different departments of shopping centres, warning whenever the occupancy of a department is reaching its maximum capacity.

5. References

The following references were used in the project; therefore they are also implicit in this summary.

[1] Room occupancy estimation through WIFI, UWB, and Light Sensors mounted on doorways- Hessam Mohammadmoradi. University of Houston

[2] https://v-count.com/people-counting-technologies-a-comprehensive-guide/

[3] https://es.v-count.com/soluciones/contador-de-personas/

[4] https://www.footfallcam.com/Product/FootfallCam-3D-Max

[5] https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/

[6] https://www.hackster.io/phfbertoleti/counting-objects-in-movement-using-raspberry-pi-opencv-015ba5

[7] https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/

[8] https://www.elconfidencial.com/empresas/2021-01-06/supermercados-mercadona-carrefour-dia_2896439/

[9]https://www.expansion.com/empresas/distribucion/2019/05/12/5cd7e79822601d15048b45df.html

[10] https://www.elcorteingles.es/centroscomerciales/es/eci/centros?page=4

[11] https://elpais.com/economia/2019/03/13/actualidad/1552475635_649725.html

[12] https://es.wikipedia.org/wiki/Zara

[13] https://www.revistagq.com/noticias/articulo/ikea-mas-grande-del-mundo-superficie-cuanto-mide-filipinas-campos-de-futbol

[14] https://www.enterat.com/estilo/ikea-tiendas-espana.php

[15] https://www.business-standard.com/article/pti-stories/decathlon-to-meet-30-per-cent-local-sourcing-norms-in-5-years-114060100108_1.html

[16] https://www.palco23.com/equipamiento/decathlon-un-negocio-de-1660-millones-y-170-tiendas-que-pivota-al-centro-de-espana.html

[17] https://www.palco23.com/fitness/altafit-sigue-creciendo-pese-a-la-crisis-con-una-nueva-apertura-en-madrid.html

[18] https://www.elconfidencial.com/empresas/2019-10-22/utopicus-coworking-oficinas-flexibles-bra_2147603/

Table of Contents

1. Introduction 4

2. State of the art 5

3. Development of the project 9

3.1. Preliminary design of the prototype 9

3.2. Development of the Computer Vision people counter with Raspberry Pi 11

3.2.1. Overview 11

3.2.2. Explanation of the code 14

3.3. Connecting the Raspberry Pi to the cloud using Altair SmartWorks 21

3.4. Analysis of the results and budget 26

4. Experiments 27

4.1. People counting system experiments 27

4.2. Shopping centre SmartWorks model 33

5. Economic motivation for the project: Justification through a market sizing 42

6. Conclusions 47

7. Sources of information 48

Table of Figures

Figure 1: Beam sensors used for people counting systems 5
Figure 2: Functioning of a thermal counter 6
Figure 3: 2D monocular counter 6
Figure 4: Time of flight sensor 7
Figure 5: 3D stereo counter 7
Figure 6: Raspberry Pi 4 used for the prototype 9
Figure 7: Raspberry Pi camera used for the prototype 10
Figure 8: View of the occupancy tracking system prototype 10
Figure 9: Schematic view of the functioning of the system 11
Figure 10: Block diagram of the system functioning 13
Figure 11: Sample frame before applying background subtraction and a Gaussian filter 17
Figure 12: Sample frame after applying background subtraction and a Gaussian filter 17
Figure 13: Resulting image after binarization 19
Figure 14: SmartWorks Studio working environment 22
Figure 15: MQTT credentials for Zone 1 24
Figure 16: Dashboard used for data visualization 25
Figure 17: Home experiment, first instant 28
Figure 18: Home experiment, second instant 28
Figure 19: Home experiment, third instant 28
Figure 20: Airport-zone experiment, first instant 29
Figure 21: Airport-zone experiment, second instant 30
Figure 22: Airport-zone experiment, third instant 30
Figure 23: Airport-zone experiment, fourth instant 31
Figure 24: Airport-access experiment, first instant 31
Figure 25: Airport-access experiment, second instant 32
Figure 26: SmartWorks collection with all the shopping centres 35
Figure 27: Color scale used for the occupancy rates 35
Figure 28: Resulting map with the shopping centres 36
Figure 29: Information shown on the map for the Zorrilla shopping centre 36
Figure 30: Information shown on the map for the Conquistadores shopping centre 36
Figure 31: Occupancy rate heatmap for the Castellana shopping centre 38
Figure 32: Occupancy rate heatmap for the Alicante shopping centre 38
Figure 33: Visualization dashboard for the shopping centre SmartWorks model 40
Figure 34: Potential yearly revenue for the business 46

Table of Tables

Table 1: Bill of Materials of the prototype 26
Table 2: Description of the experiments 27
Table 3: Results of the experiments 32
Table 4: Information of the modelled shopping centres 34
Table 5: Occupancy information for the Castellana shopping centre 39
Table 6: Occupancy information for the Alicante shopping centre 39
Table 7: Market sizing 45

1. Introduction

The project consists of the development of a cloud-based people counting device. The device will be designed from scratch and the result of the project will be a prototype of the device, which will have full functionality.

The motivation for carrying out the project was found in a market research of existent occupancy tracking technologies in the Spanish market. In the research, it was clearly visible that there is little presence of low-cost people counting systems in Spain, which is an indicator of an opportunity to launch a low-cost people counting device.

The people counting device was initially targeted to gyms, but after a thorough market research, it was clear that the industry with the highest potential was retail.

For the development of the prototype, a Raspberry Pi 4 will be used, as well as a Raspberry Pi camera, and the Altair SmartWorks software for connecting the device to the cloud.

The main objective of the project is to prove that a cloud-based people counting device prototype can be built with a low budget and high accuracy.

Further objectives involve developing an IoT connected product which is scalable and will have an impact by addressing a relevant issue. Additionally, the device will have the potential to boost the transformation of physical businesses towards a data driven model, which is a top priority for businesses in order to stay competitive. Finally, the people counting system will help the businesses' clients save time, allowing them to access occupancy information to make smarter decisions based on data.

2. State of the art

There are several ways of estimating the occupancy in a room. The most common technologies are usually based on either light and thermal sensors, WIFI, ultra band-width, or cameras *[1]*. Each of these technologies has some advantages and disadvantages:

- **Beam sensors:** In people counting technologies based on beam sensors, receiver and transmitter are located side by side in the access point. When the beam detects a person walking through it, then the count increases. The main advantage of this technology is the low cost. However, it can lead to many errors if the place is very crowded, given that people walking side by side would be counted as a single person. Another disadvantage is that beam sensors do not have sense of direction, which makes it hard to determine whether the people are entering or exiting the area.

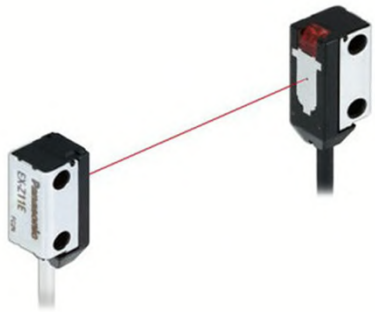

Figure 1: Beam sensors used for people counting systems

- **Thermal counters:** This technology measures the room temperature and identifies people through their body heat, and then generates images using infrared radiation. These thermal sensors are usually installed on the room ceiling or on top of the access points. This technology has several limitations. For instance, if the room is crowded, the system might have difficulties differentiating between people who are close to each other, given that the entire area will have a similar temperature due the body heat. Furthermore, thermal counters are a bad option for outdoor people counting, because sunlight is a large liability for thermal counting.

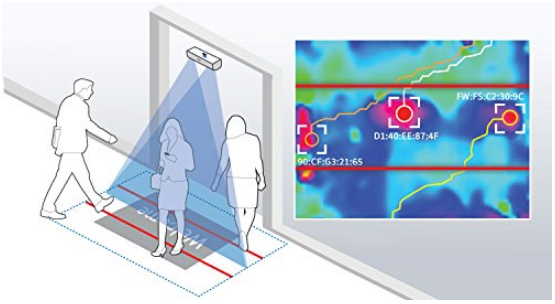

Figure 2: Functioning of a thermal counter

- **2D monocular counters:** These sensors use a single camera lens for counting. They are installed on the room ceiling, and they detect moving objects. They usually combine a deep learning people detector algorithm and a moving object detector. The deep learning people detector requires a lot of computational capacity but is very accurate when it comes to detecting people. On the other hand, the moving object detector is capable of separating the moving object from the background reference frame, which can be static (no background subtraction), or dynamic (background subtraction). These counters are inexpensive and easy to install, and they are able to track the direction in which the people are walking, which is useful for detecting how many people are entering or exiting the room. This project uses the 2D monocular counter technology because it uses a single-lens Raspberry Pi camera.

Figure 3: 2D monocular counter

- **WIFI counters:** This technology works while the WIFI is enabled and there is access to it. The setup of these counters is complex, and the results might have a large error because this sensor depends on having a WIFI access point.

- **Time of flight sensors:** In this type of sensors, a signal is sent to objects under the sensor and then increase the counter when the reflection of the signal bounces back to the sensor. There are different kinds of signals that can be sent: laser, infrared, or ultrasound. The main limitations of time of flight sensors occur in places with a lot of sunlight, and also in places in which the distance between transmitter and receiver are far away from one another.

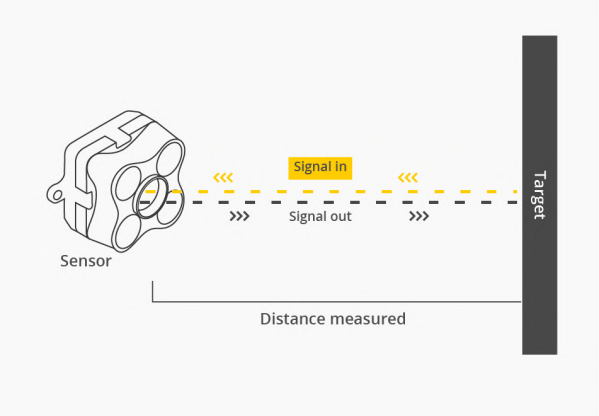

Figure 4: Time of flight sensor

- **3D stereo counters:** The main difference from monocular counters is that in 3D stereo counters 2 camera lenses are used. This way, each lens captures different images, and when the information captured by both lenses is combined, an image with 3D spatial depth is constructed. This system provides a high accuracy, being better able to recognize humans because it has an additional height (z axis) variable. However, this type of technology is expensive *[2]*.

Figure 5: 3D stereo counter

To better illustrate the state of the art of people counting devices, two people counting devices which are currently manufactured by other companies will be analyzed thoroughly:

V-Count

V-Count's flagship product is the V-Count 3D Alpha+. This device uses 3D stereo vision technology and has a 98% accuracy. The device has a bidirectional functionality which allows the count of people entering and exiting the area simultaneously. The people counting device might have some advantages such as an accurate calculation of conversion rates, staff optimization and peak hour identification, among others *[3]*.

V-Count 3D Alpha+ uses WIFI technology, which enables to contrast the information measured by the camera with the information collected through WIFI geo-location.

V-Count is focused on retail businesses, museums and libraries, airports, supermarkets, shopping centers, and other events and exhibitions.

However, V-Count 3D Alpha+ has a high price of 1200$ per camera, which is much higher than the cost of our prototype for this project.

FootfallCam

FootfallCam 3D Max is a 3D stereo people counter with a 99,5% accuracy. Their devices have a very wide coverage area, which reduces the number of required devices by 40% *[4]*.

The device has a 25-year expected lifespan. The price of each camera is 560$, still much higher than our prototype.

By analyzing the state of the art, it can be concluded that existent people counters have a high accuracy but also a high price. Therefore, the product developed in this project can be defined as a low-cost people counter, designed for applications that do not require the best accuracy.

3. Development of the project

The development of the occupancy tracking system prototype consists of 4 main phases:

1. Preliminary design of the prototype
2. Development of the Computer Vision people counter with Raspberry Pi
3. Connecting the Raspberry Pi to the cloud using Altair SmartWorks IoT software
4. Analysis of the budget

3.1. Preliminary design of the prototype

Hardware:

After thoroughly analyzing the available hardware options for developing such a system, a decision was made that the Raspberry Pi 4 was the best device to implement an occupancy tracking system, given it provides a good tradeoff of a fairly powerful processor at a very low price. An image of a Raspberry Pi 4 is shown in the following figure:

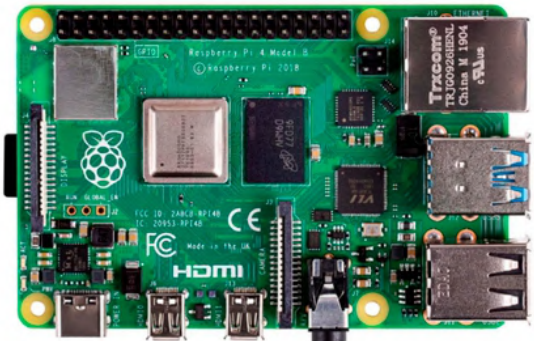

Figure 6: Raspberry Pi 4 used for the prototype

The next step in the hardware design was to choose a camera, the most affordable and convenient option was the Raspberry Pi 8MP camera, which is shown in the following figure:

Figure 7: Raspberry Pi camera used for the prototype

Some other components were also purchased for the correct functioning of the Raspberry Pi, such as a fan, a case, a microSD card, and a micro-HDMI cable. These components will be explained with more detail in the budget analysis. The resulting hardware is shown in the following figure:

Figure 8: View of the occupancy tracking system prototype

Software:

After analyzing all the software possibilities, it was considered that the best option is a combination of an object detection algorithm and an object tracking algorithm. Furthermore, this is achieved through Computer Vision OpenCV packages and coded in Python. The code used for this prototype is based on a guide posted on *pyimagesearch* by Adrian Rosenbrock, which is called “OpenCV People Counter”. *[5]*

A schematic view of the functioning of the system is shown in the following figure:

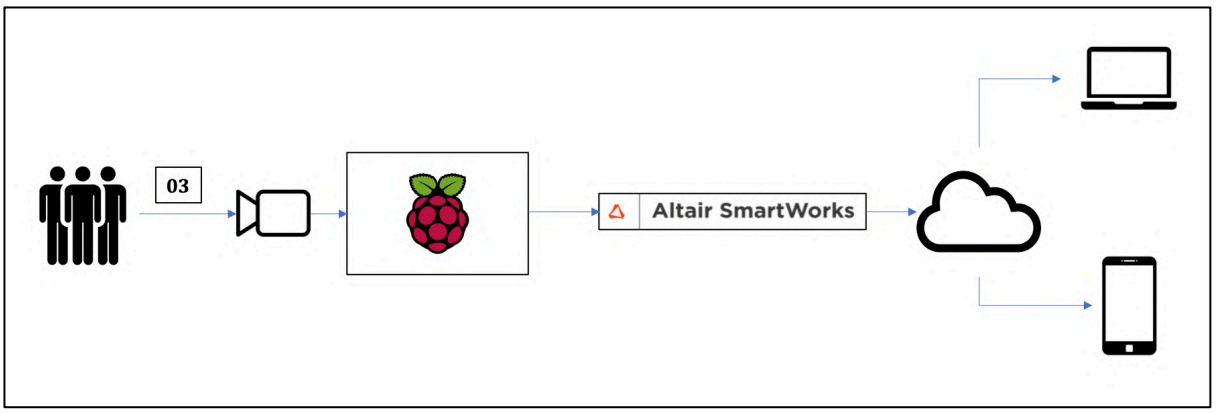

Figure 9: Schematic view of the functioning of the system

3.2. Development of the Computer Vision people counter with Raspberry Pi

3.2.1. Overview

As mentioned earlier, the people counter will be coded in Python, and will use OpenCV packages. *[5]* Furthermore, it will use a combination of an object detection and an object tracking algorithm. There are several differences between an object detector and an object tracker:

Object detector:

An object detector is used to identify any new objects entering or leaving the frame. It is very heavy at a computational level. For this prototype two different object detection methods were tried in order to determine which one works best:

1. **Deep learning single shot people detector (MobileNetSSD**): This is a people detection algorithm based on a trained model, which is extremely accurate for detecting people, but is very heavy at a computational level. Therefore, this object detector worked perfectly on a MacBook Pro (processor: 2,7 GHz Dual-Core Intel Core i5) but did not work as well on the Raspberry Pi 4 (processor: 1,5 GHz ARM Cortex-A72).
2. **Moving-object detector:** This is an algorithm used for detecting any object which is moving in the frame. The accuracy of this algorithm is lower, given it is not specific for detecting people, but rather used for detecting any kind of moving object. It also requires a more thorough calibration before using it. However, it is significantly less heavy at a computational level, therefore it is the best option for the Raspberry Pi. The two input arguments that should be entered upon calibration are the minimum and maximum contour area, and the binarization threshold. The minimum and

maximum contour areas are the size limits that we consider that can be attributed to a person, considering the height at which the camera is located. The binarization threshold, on the other hand, is the minimum acceptable color gradient between the figure and the background required for detecting an object. The lower the threshold, the more sensible the detection.

Additionally, the moving object detector can be used with or without background subtraction:

- **Without background subtraction:** In this case the initial frame is taken as the reference frame. Therefore, the object detection is based on the differences between the current frame and the reference frame. However, this method may lead to problems in the detection whenever there already is an object in the initial frame, since it takes the given object as part of the background. It may also cause problems whenever there are changes in the lighting of the background.

- **With background subtraction:** In this case the reference frame is dynamic, and it is a weighted average of all the frames up to the current frame. The object detection is based on the differences between the current frame and the reference frame (weighted average of all the previous frames). This method is optimal because it adjusts itself to changes in the background, such as changes in the lighting or objects in the background.

Object tracker:

The object tracker oversees assigning a specific ID to each person in the frame. It is also in charge of tracking the object (person) as it moves around the frame, through a centroid tracking algorithm. The tracker works by calculating the Euclidean distances between objects in the current frame and objects in the previous frame. Then, it assumes that couples of objects that have the minimum Euclidean distance between the object in the previous frame and the object in the current frame belong to the same object ID.

The combined object detection and tracking algorithm is divided in two phases:

1. **Detecting:** This phase is only executed every N frames, given the object detector is very heavy at a computational level. In this phase the object detection algorithm is executed to recognize if any new objects have entered or left the frame.
2. **Tracking:** For each object detected during the detection phase, an object tracker is created to track that object that moves around the frame. Given it is less heavy at a computational level, this object tracking algorithm is executed every frame until the N-th frame is reached, in which the object detection algorithm is then executed (phase 1). After that, the whole sequence is repeated.

Regarding the application of this combined system, there will be two different varieties of logic to be implemented, depending on the physical location in which the camera is set:

- **Access control:** If the camera is placed at an access point, the purpose of the people counter would be to keep the count of how many people enter or leave through that access (e.g., 6 people entered and 4 left the men's fashion department). The complete script for the access control can be found in *Annex III*.
- **Zone control:** If the camera is set in the middle of a section, the purpose would be to keep track of how many people are in that section in each moment (e.g., there are 4 people at the men's shoes stand). The complete script for the zone control can be found in *Annex II*.

A block diagram of the functioning of the system is shown in the following figure:

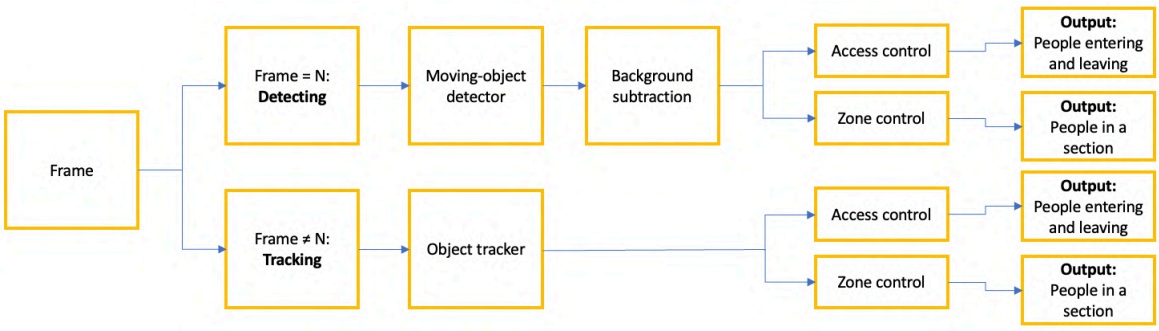

Figure 10: Block diagram of the system functioning

3.2.2. Explanation of the code

The algorithm used by the device is coded in Python. The complete final code is based on a post from Adrian Rosenbrock's blog pyimagesearch *[5]*. A thorough explanation of the evolution and functions of each part of the code is presented in the following lines.

The initial algorithm from *pyimagesearch [5]* combines an object detection algorithm and an object tracker algorithm. For the object tracking algorithm they use a MobileNet Single Shot Detector (SSD), which requires a lot of computational capacity. The complete initial code from *pyimagesearch [5]* can be found in *Annex I.*

The code uses the following libraries:

```
from pyimagesearch.centroidtracker import CentroidTracker
from pyimagesearch.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
```

The initial code is composed of the following main parts:

1. **Import of all libraries and packages:** some of the main libraries used are numpy, imutils, opencv, and dlib
2. **Definition of the arguments of the command line:** these are the arguments that will be entered as inputs through the terminal of the device. Depending on the inputs, the algorithm can be used to count the number of people of a previously recorded video or access the camera live. Another relevant input is the confidence interval that the user wants for the detection process. To be specific, the algorithm will assign a level of confidence to each detected person, and if this is lower than the limit set by the user, then the person detected will not be counted. Another important argument on the command line is the frequency of running the detector (MobileNet Single Shot Detector). Because the detector is heavier than the tracker, the detector is only run every N frames, and the N number is an input in the terminal.

3. **Initialize all functions and variables:** The object detector is an algorithm which is trained to detect different types of objects. Therefore, when initializing the detector, the object type “person” should be the only object type selected, to make sure that no other objects are counted as people by the device. As mentioned before, the device can count the people live from the camera or from a previously recorded video, which should also be initialized. Other relevant variables and functions that should be initialized are the height and width of the frames which will be processed, the trackers, the objects, and variables such as the amount of people entering or exiting the place, or the amount of people who are in a specific area in a particular moment.

4. **Main script loop:** The main script is a loop which will process each frame. There are 3 main states for the system. The first state is the waiting state, in which the system is on hold until a person is detected. The second state is the detecting state, which happens every N frames when the object detector is running. The last state is the tracking state, which takes place in every other frame, and in which the people that have been detected in that frame are tracked with a unique ID. For every frame, the algorithm checks whether the current frame is the N^{th} frame or any other frame, to determine whether the object detector or the object tracker should be executed.

    a. **N^{th} frame:** The object detector is run. First the trackers are initialized. Then, the detections are filtered to ensure that they belong to the object type “person” for a certain confidence interval input by the user. Finally, a tracker is assigned to every detected person.
    b. **Every other frame:** The object tracker is run. First, the centroid tracker assigns the ID of each object to their location. Then, the direction of the moving person is detected by calculating the difference in the location of the person in two separate moments in time. The direction is only used in the devices which are located at access points to determine whether each person is entering or exiting the area. After that, the total count of people who have entered and exited the area is updated.

The initial script works well on a MacBook Pro (processor: 2,7 GHz Dual-Core Intel Core i5). However, the purpose of this project is to embed the people counting algorithm and cloud connectivity in a low-cost device of a small size, so that it can be used to track the occupancy in physical businesses such as retail businesses. As mentioned earlier, the low-cost device is chosen to be a Raspberry Pi 4 (processor: 1,5 GHz ARM Cortex-A72). However, the initial script is extremely heavy and cannot run on a Raspberry Pi. The following lines will explain how the initial code was modified to work on a Raspberry Pi 4 with the highest possible accuracy.

The main issue with the initial code was that the Deep Learning Single Shot People Detector (MobileNet SSD) was too heavy for the Raspberry Pi. In this context, the best possible alternative was to replace the Deep Learning Single Shot People Detector for a Moving Object Detector, which is lighter at a computational level. The Moving Object Detector algorithm consists of 4 main phases *[6]*:

Phase 1:

The first step is to compare the actual frame with a reference frame. This means that an initial reference frame is needed to identify if there is an object which is moving. Then, to detect the moving object, a background subtraction mechanism is applied. Applying this background subtraction, the system will identify the pixels and shapes which are in the current frame but were not in the reference frame, resulting in the detection of only the shapes which are changing or moving. The name of “background subtraction” refers to the method by which the detection is achieved. Through this methodology each pixel of the current frame is subtracted from the reference frame in terms of color, so the result is a new frame with a black background, and the object which is new or has moved appears in a gray color. Furthermore, for greater efficiency regarding the time that the system will need to process the image, it is important to transform the entire frame into a grayscale. The reason for this is that in grayscale every pixel has a single-color information, whereas in the standard color coding (red, blue, and green), each pixel has 3 color datapoints (one for each of the three main colors). Transforming the entire frame into grayscale allows the image processing to be 3 times faster, which is extremely convenient. After that, a Gaussian filter is applied to the frame to blur the image and achieve smoother contours of the shapes, which will ease posterior image processing.

Considering the following initial image:

Figure 11: Sample frame before applying background subtraction and a Gaussian filter

An example of a frame after background subtraction and using a Gaussian filter is shown in the following image *[6]*:

Figure 12: Sample frame after applying background subtraction and a Gaussian filter

The commands used to transform the image into grayscale and apply a Gaussian Blur filter are the following:

```
GrayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
GrayFrame = cv2.GaussianBlur(GrayFrame, (21, 21), 0)
```

Another important thing to consider is that background subtraction can be static or dynamic. This means that the reference frame used for subtraction can be either the same initial static

frame always, or it can be a dynamic weighted average of all the frames up to the current frame. The static background subtraction has some disadvantages, because if there is an object in the initial reference frame, this object is considered as part of the background, leading to mistakes. Furthermore, if the lightning of the area changes, then the algorithm might detect the entire area as a changing object, because the color of the entire current frame will be different to the color of the initial reference frame. These mistakes can be avoided using a dynamic background subtraction *[7]*.

To run the code with **static background subtraction**:

- The number of fames (N) between every execution of the moving object detector algorithm should be set to 2:

```
ap.add_argument("-s", "--skip-frames", type=int, default=2,
    help="# of skip frames between detections")
```

- The code should run according to the following script:

```
if avg is None:
    #print("[INFO] starting background model...")
    #avg = GrayFrame.copy().astype("float")
    avg = GrayFrame
    #rawCapture.truncate(0)
    continue

#Background subtraction and image binarization
#cv2.accumulateWeighted(GrayFrame, avg, 0.5)
FrameDelta = cv2.absdiff(GrayFrame, avg)
#FrameDelta = cv2.absdiff(GrayFrame, cv2.convertScaleAbs(avg))
```

To run the code with **dynamic background subtraction**:

- The number of fames (N) between every execution of the moving object detector algorithm should be set to 30:

```
ap.add_argument("-s", "--skip-frames", type=int, default=30,
    help="# of skip frames between detections")
```

- The code should run according to the following script:

```
if avg is None:
    #print("[INFO] starting background model...")
    avg = GrayFrame.copy().astype("float")
    #avg = GrayFrame
    #rawCapture.truncate(0)
    continue

#Background subtraction and image binarization
cv2.accumulateWeighted(GrayFrame, avg, 0.5)
#FrameDelta = cv2.absdiff(GrayFrame, avg)
FrameDelta = cv2.absdiff(GrayFrame, cv2.convertScaleAbs(avg))
```

Phase 2:

The second step is to binarize the image. This way, the resulting image will only have two colors: black and white. To be specific, moving objects will have a white color, and the background will be in black. In this context, it is extremely important to choose an appropriate binarization limit. The algorithm will analyze each pixel of the frame after applying the background subtraction and grayscale transformation, and if the value of the color of the pixel is higher than the binarization limit, that pixel will be transformed into white, otherwise, the pixel will be transformed into black. The binarization limit should be input by the user, and calibrated for each specific environment:

```
BinarizationThreshold = 25
```

The lower the threshold, the higher the sensitivity of the detections. Binarization is done with the following command:

```
FrameThresh = cv2.threshold(FrameDelta, BinarizationThreshold, 255,
    cv2.THRESH_BINARY)[1]
```

The resulting image after binarization is shown in the following figure:

Figure 13: Resulting image after binarization

As seen in the figure above, the two moving people are displayed in white, and the background is in black.

Phase 3:

The next step involves carrying out image dilatation. The reason for this is that many times holes can be found inside the contour of objects due to lightning or colors. The holes within objects can lead to errors in the system since it might detect the holes as objects. To apply image dilatation the following command is used:

```
FrameThresh = cv2.dilate(FrameThresh, None, iterations=2)
```

Phase 4:

The next step involves searching for the contours and it is extremely important *[6]*. The algorithm analyses every detected contour and measures its size. If the size of the contour is higher than the limit, then the object is counted as a person. The limit of the minimum contour area size from which an object can be considered a person is input by the user. Therefore this part requires some calibration, since the minimum contour area depends on the height at which the camera is located (the further the camera is from the people, the smaller the size of the people in the frame). The minimum contour area size is a variable defined in the script as follows:

```
MinContourArea = 3000
```

The following function is used to find all the contours:

```
cnts = cv2.findContours(FrameThresh.copy(), cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
```

The following loop is used to check which of the found contours is larger than the minimum contour area and therefore can be considered a person:

```
for c in cnts:

    if cv2.contourArea(c) < MinContourArea:
        continue

    QttyOfContours = QttyOfContours+1
```

3.3. Connecting the Raspberry Pi to the cloud using Altair SmartWorks

To connect the system to the cloud, Altair SmartWorks products will be used. Altair SmartWorks delivers advanced IoT technology within an open-architecture environment, which will be extremely useful for our specific application.

In the following lines, a detailed explanation will be presented of how the device will be connected to the cloud through Altair SmartWorks, and a description of the commands used in the script.

As an introduction, SmartWorks is a low-code IoT application development platform. It allows to develop IoT applications in a simple manner, given it provides the tools to rapidly build the application backend, as well as the environment and elements for the frontend and data visualization through real-time dashboarding. SmartWorks is used in various industries, such as building equipment, industrial goods, robotics, consumer electronics, energy, and heavy machinery.

For this project there will be 2 different parts within the SmartWorks platform: back-end and front-end.

Back-end

This part refers to the part that the final user will not see. It includes connecting the Raspberry Pi to SmartWorks and creating the necessary *Spaces*, *Things,* and *Properties* to upload the data collected through the Raspberry Pi camera to the cloud.

The first step in the SmartWorks environment is to create a *Space*, in this case the *Space* is called “twozones”. Then the *Collection* “counters” is created (a *Collection* is a data table). Because this *Space* will contain information of the occupancy in 2 different areas, then 2 different *Things* must be created: “Zone1” and “Zone2”. Both *Things* share the same property, which is called “people_count”. This property is the variable that will contain the number of people in both zones. Furthermore, each of the *Things* has a unique id, and unique

login username and password credentials, which are needed to connect to those *Things* from the device (in this case the Raspberry Pi) and modify the information stored in them.

A screenshot of the SmartWorks working environment for this example is shown in the following figure:

Figure 14: SmartWorks Studio working environment

A practical example will be used to better illustrate this topic. The *Space* ("twozones") could represent the ground floor of a mall. Then the *Things* "Zone1" and "Zone2" could be used to represent the women's shoes and men's shirts sections, respectively. Additionally, the *Property* "people_count" will contain the information of the number of customers in each of the sections at any given moment.

Once the SmartWorks environment is set up, the next step is to add the appropriate commands to the Python script, which will be used to send data from the Raspberry Pi to the cloud.

In this context, the following lines will present several parts of the script used to send the information recorded from the Raspberry Pi in "Zone 1" to the SmartWorks *Space*, along with explanations of each command.

The first step is to create a *Space*, which is an isolated work environment within SmartWorks. Resources of one *Space* cannot interact with resources of a different *Space*. As mentioned earlier, in this case the *Space* "twozones" has been created in SmartWorks, so the variable is stored to use it in the script:

```
96  #SmartWorks-create a space for the area being recorded
97  space = "twozones"
```

The next step is to define the Model Database. This is the place where all the information from the application can be stored. The Model Database is composed of tables called *Collections.* Each *Collection* contains the information in different *Things*. These *Things* are modeled according to the *Web of Things* standards. In this case, a collection called “counters” is created, therefore it is stored in a variable in the script:

```
99  #SmartWorks-create a collection for all the area spaces
100 collection = "counters"
```

After that, we store the ID from the thing that was created earlier in the SmartWorks Studio:

```
102 #SmartWorks- we store the thing id
103 thing_id_1 = "01F83CSPWX9MHXQY8Z9RTAAMN5"
```

The next step is to configure the MQTT connection. MQTT is the standard for IoT messaging, which enables messaging from device to cloud and from cloud to device. This is accomplished through a Publish/Subscribe pattern which decouples the client that sends a message (publisher) from the clients that receive the messages (subscribers).

All client connections are managed by a broker. To configure the MQTT broker connection, the information of the host (URL of MQTT broker) is needed, along with the user and password credentials. The aforementioned information for “Zone 1” is shown in the following figure:

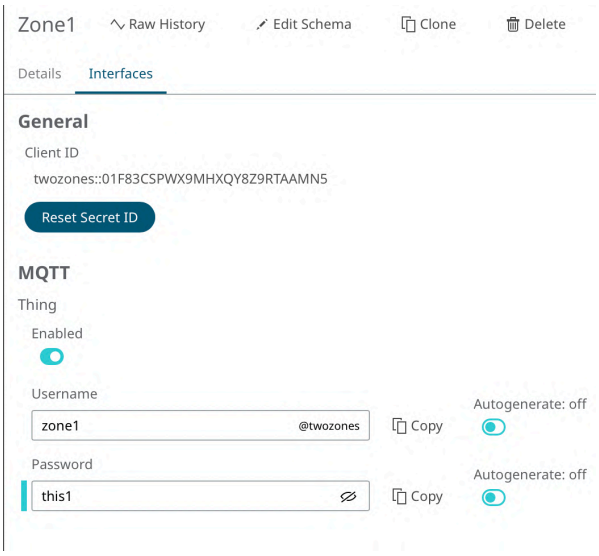

Figure 15: MQTT credentials for Zone 1

Which are then stored in variables on the script:

```
#SmartWorks- MQTT Variables
host="mqtt.swx.altairone.com"
user_1 = 'zone1@twozones'
password_1 = 'this1'
```

Then, the following commands must be run in order to connect to the broker:

```
#Smartworks- We connect to the broker
client = mqtt.Client()
client.username_pw_set(username=user_1,password=password_1)
client.connect(host, port=1883, keepalive=10, bind_address="")
```

The next step is to define the MQTT topics, which are paths inside the broker. They allow the user to publish information in different topics and subscribe to topics as well. The topics are used by the broker to filter the messages to each client. Therefore, it is required to build the topics to send data to the platform. In this specific case, two types of topics will be used; data topics, which allow to send and receive information of a specific thing, and property topics, which allow to update the *Thing's* properties.

The following commands are run in order to see the outputs of the data and property topic paths:

```
#SmartWorks-sending data
data_topic_1 = "set/{}/collections/{}/things/{}/data".format(space, collection, thing_id_1)
print ("Data topic: "+data_topic_1)

property_topic_1 = "set/{}/collections/{}/things/{}/properties/{}".format(space, collection, thing_id_1, "people_count")
print ("Property topic: "+property_topic_1)
```

Note that the property towards which this topic is led is "people_count", which is the only property that the *Thing* "Zone 1" has.

The final steps involve updating the *Thing* properties. To do this, the payload (relevant information that we want to send) of the property is built as a *JSON* object with the name of the property as the primary key on that object. In this case, the information in the variable "counter1" is the number of people in Zone 1, which was counted by the people counting algorithm. Then a *JSON* object is built and the information from the counter is stored in it.

```
300     #SmartWorks- publish all data and properties
301     jsonarray1 = {
302             "people_count": str(counter1),
303             }
```

Finally, the count is published in MQTT through the data and property topics defined earlier:

```
304     client.publish(topic = data_topic_1 ,payload = str(json.dumps(jsonarray1)))
305     property_people_count = {
306             "people_count": str(counter1)
307             }
308     client.publish(topic = property_topic_1 ,payload = str(json.dumps(property_people_count)))
```

Front-end

This part includes the data visualization and presentation within the SmartWorks platform. The visualization is achieved by means of a dashboard created with *Panopticon*. The resulting dashboard is shown in the following figure:

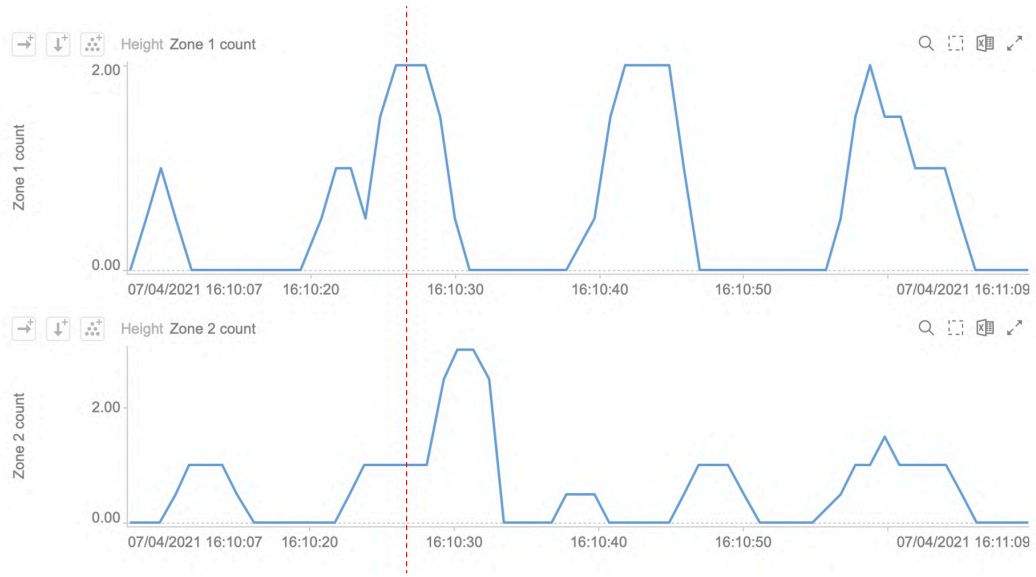

Figure 16: Dashboard used for data visualization

Note that on July 4th 2021 at 16:10:27 there were 2 people in Zone 1 and 1 person in Zone 2.

3.4. Analysis of the results and budget

Regarding the budget used for this prototype, the bill of materials is shown in the following table:

Component name	Price
LABISTS Raspberry Pi 4 Kit: - Raspberry Pi 4 RAM 4GB - MicroSD card 32GB - Heat sinks - Fan - Case - Micro HDMI cable - 5V 3A cable	110€
LABISTS Raspberry Pi Camera 8MP	30€
Total	**140€**

Table 1: Bill of Materials of the prototype

4. Experiments

4.1. People counting system experiments

The next step is to run some experiments to test the prototype. The main high-level objective of the experiments is to determine whether the prototype is accurate enough, since it would mean that it is possible to create a low-cost product that can track the occupancy level with a good accuracy. Furthermore, each of the experiments is designed to meet a low-level objective. A description of the different experiments and their objectives is shown in the following table:

Experiment	Description	Objective
Home	The video in this experiment is recorded at home, which is a very low-occupancy location. The system will measure the total number of people in that part of the house at a particular moment.	Determine the accuracy of the occupancy tracking system in a specific area of a low-occupancy location (a home).
Airport-Zone	The video in this experiment is recorded at an airport. The system will measure the total number of people in a specific zone at a given moment.	Determine the accuracy of the occupancy tracking system in a specific area of a high-occupancy location (an airport).
Airport-Access	The video in this experiment is recorded at an airport. In this case the system will be considered to be located at an access point, therefore it will count the number of people entering and exiting the area.	Determine the accuracy of the occupancy tracking system when counting the number of people entering and leaving a high-occupancy location.

Table 2: Description of the experiments

Experiment 1: Home

The first experiment consists in a recording of 36 seconds that takes place in the stairway of a house. In this experiment, two men enter the monitored location at different times. The people counting system should be able to keep track of the total number of people in that area at any given moment. The sequence is the following:

1. The man in the black sweater enters the area (the system count is 1, so accuracy is 100%):

Figure 17: Home experiment, first instant

2. The man in the black sweater leaves the area and the man in the gray sweater enters the area (the system count is 1, so accuracy is 100%):

Figure 18: Home experiment, second instant

3. The man in the black sweater and the man in the gray sweater are both in the area walking in opposite directions (the system count is 2, so accuracy is 100%):

Figure 19: Home experiment, third instant

The result of this experiment is very positive. The system recognized both men perfectly and assigned an ID to each of them. The system count matches the real number of people in the area at every single moment. Therefore, it can be concluded that the accuracy of the system at a low-occupancy location is approximately of 100%.

Experiment 2: Airport-Zone

The second experiment consists in a recording of 17 seconds that takes place at an airport. In the recording, many people enter and leave the monitored area. Therefore, this experiment is used to test the system at a high-occupancy location. The system will count the number of people in the area at any given moment within the recording. The sequence is the following:

1. At first, there is only one woman in the area (the system count is 1, so accuracy is 100%):

Figure 20: Airport-zone experiment, first instant

2. Two new men enter the area, plus the woman they make 3 people in total (system count is 2, so accuracy is 67%):

Figure 21: Airport-zone experiment, second instant

3. Then the woman leaves, and 4 new people enter the area almost simultaneously. Plus the two men from earlier, they make 6 people in total (system count is 6, so accuracy is 100%):

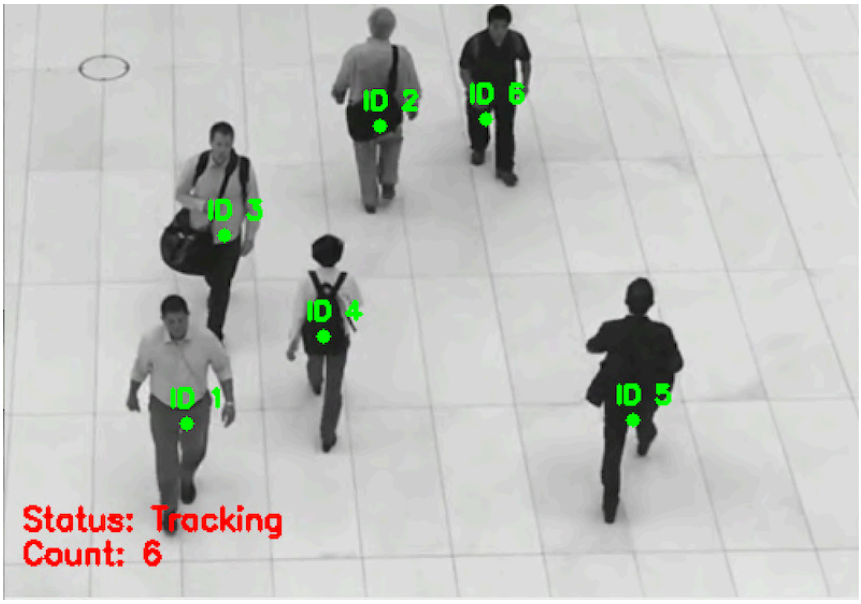

Figure 22: Airport-zone experiment, third instant

4. After that, a man is left alone (system count is 2, so accuracy is 50%):

Figure 23: Airport-zone experiment, fourth instant

Finally, the average accuracy of the 4 different analyzed moments would be approximately 80%. Therefore, it can be concluded that the accuracy of the occupancy tracking system is 80% when counting the total number of people in an area of a high-occupancy location.

Experiment 3: Airport-Access

The third experiment consists in the same 17-second recording than in experiment 2. The main difference is that in this case it will be considered that the camera is located at an access point. Therefore, the system will count the number of people entering (up) and exiting (down) the area.

The functioning of the system is shown in the following figure:

Figure 24: Airport-access experiment, first instant

The final count of the system after the 17 seconds of recording is the following:

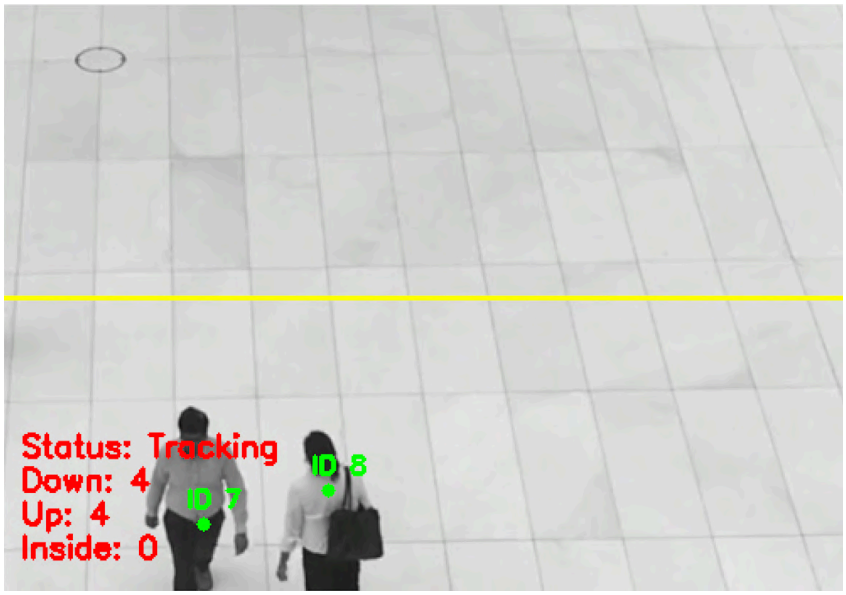

Figure 25: Airport-access experiment, second instant

As seen in the figure shown above, the final system count after 17 seconds is 4 up (entered) and 4 down (exited), when the real situation was that 6 people entered and 4 people left. This means that in this experiment the system was 80% accurate in counting the people entering and leaving the area.

A summary of the results is shown in the following table:

Experiment	Situation Tested	Accuracy
1. Home	Low-occupancy location. Total people in an area.	100%
2. Airport-Zone	High-occupancy location. Total people in an area.	80%
3. Airport-Access	High-occupancy location. People entering and exiting an area.	80%

Table 3: Results of the experiments

The results of the 3 experiments were positive, given the system proved to have a 100% accuracy in low-occupancy areas and 80% accuracy at high-occupancy areas. This is a good accuracy given the low cost of the prototype (140€).

4.2. Shopping centre SmartWorks model

To gain a deeper understanding of the functionalities of the SmartWorks platform, and the ways in which it can be useful for the scope of this project, a model has been built in SmartWorks with 21 shopping centres.

The shopping centres are located in different parts of Spain. Each shopping centre has the following variables:

- **Name:** specific name of the shopping centre.
- **Province:** Spanish province in which the shopping centre is located.
- **Latitude and longitude coordinates:** these coordinates must be input in the system so that SmartWorks is able to locate them in a map.
- **Occupancy:** total number of people in the entire shopping centre at a particular moment in time.
- **Capacity:** maximum number of people allowed in the centre.
- **Occupancy rate:** the occupancy for a particular moment in time divided by the capacity of the centre.

The variables “Name”, “Province”, “Latitude and Longitude” do not change, whereas “Occupancy” and “Occupancy rate” do change depending on the occupancy of the centre.

The information for all the 21 shopping centres is shown in the following table:

Name	Province	Latitude (º)	Longitude (º)	Occupancy rate (%)
Princesa	Madrid	40.430339315136884	-3.7152529157115537	36.0%
Portal del Ángel	Barcelona	41.3864011973775	2.1726193501370195	48.7%
Nervión	Sevilla	37.38694239725724	-5.97164937345722	74.1%
Pintor Sorolla	Valencia	39.471198037539544	-0.37145170224150587	52.2%
Algeciras	Algeciras	36.13905354089084	-5.448298542804771	89.3%
Santander	Santander	43.43781179435032	-3.8397462073359794	15.0%
Fray Luis de León	León	42.5909161408182	-5.5691439886658385	28.6%
Costa Luz	Huelva	37.255798628561166	-6.9419438446251265	75.0%
María Auxiliadora	Salamanca	40.97390998258383	-5.657077450641563	26.3%
Marbella	Málaga	36.490968614766366	-4.952827188819054	33.3%
Vigo	Pontevedra	42.23219586540358	-8.717017231005068	74.1%
Ferial Plaza	Guadalajara	40.6229071189541	-3.1671342580362185	75.0%
Paseo de Morella	Castellón	39.98787526147964	-0.047679223678125174	20.0%
Ego Gain	Guipuzkoa	43.18450174894915	-2.4785458399924787	9.1%
Zorrilla	Valladolid	41.63761467631146	-4.7399670580097135	90.5%
Myrtea	Murcia	38.0194939165548	-1.1641763128254032	31.6%
Talavera de la Reina	Toledo	39.96255087415447	-4.826409192205402	71.4%
Conquistadores	Badajoz	38.873279457761164	-6.974807490990397	8.3%
Alicante	Alicante	38.34421025610232	-0.4892461319762198	53.2%
Jaén	Jaén	37.775770050633774	-3.7859950443844657	76.5%
Castellana	Madrid	40.44857932545574	-3.691932201509702	37.2%

Table 4: Information of the modelled shopping centres

As seen in the table above, all 21 shopping centres are located in Spain. Furthermore, their occupancy rates are very diverse, with Conquistadores (Badajoz) being the shopping centre with the lowest occupancy rate (8.3%) and Zorrilla (Valladolid) the one with the highest occupancy rate (90.5%).

The next step is to represent the shopping centres in a map using the *Panopticon* tool for visualizations in SmartWorks. To do this, the information in the table above must be input into the SmartWorks workspace and collection (in this case the collection is called *shopping_centres*), where each of the shopping centres is created as a *thing* within that collection. The resulting collection is shown in the following figure:

Things Models

Title	ID	Labels	Province	latitude	longitude	name	occupancy	capacity	occ
shopping_c...	01FBSNCRT...	label1	Madrid	40.4303393...	-3.7152529...	Princesa	900	2500	36
shopping_c...	01FBSNCRS...	label1	Barcelona	41.3864011...	2.17261935...	Portal del A...	1460	3000	48.67
shopping_c...	01FBSNCR...	label1	Sevilla	37.3869423...	-5.9716493...	Nervión	2000	2700	74.07
shopping_c...	01FBSNCRK...	label1	Valencia	39.4711980...	-0.3714517...	Pintor Soro...	1200	2300	52.17
shopping_c...	01FBSNCR...	label1	Algeciras	36.1390535...	-5.4482985...	Algeciras	2500	2800	89.29
shopping_c...	01FBSNCRE...	label1	Santander	43.4378117...	-3.8397462...	Santander	300	2000	15
shopping_c...	01FBSNCR...	label1	León	42.5909161...	-5.5691439...	Fray Luis d...	600	2100	28.57
shopping_c...	01FBSNCR...	label1	Huelva	37.2557986...	-6.9419438...	Costa Luz	1800	2400	75
shopping_c...	01FBSNCR7...	label1	Salamanca	40.9739099...	-5.6570774...	Maria Auxil...	500	1900	26.32
shopping_c...	01FBSNCR4...	label1	Malaga	36.4909686...	-4.9528271...	Marbella	1000	3000	33.33
shopping_c...	01FBSNCR3...	label1	Pontevedra	42.2321958...	-8.7170172...	Vigo	2000	2700	74.07
shopping_c...	01FBSNCQ...	label1	Guadalajara	40.6229071...	-3.1671342...	Ferial Plaza	2100	2800	75
shopping_c...	01FBSNCQ...	label1	Castellon	39.9878752...	-0.0476792...	Paseo de M...	400	2000	20
shopping_c...	01FBSNCQ...	label1	Guipuzkoa	43.1845017...	-2.4785458...	Ego Gain	200	2200	9.09
shopping_c...	01FBSNCQ...	label1	Valladolid	41.6376146...	-4.7399670...	Zorrilla	1900	2100	90.48
shopping_c...	01FBSNCQ...	label1	Murcia	38.0194939...	-1.1641763...	Myrtea	600	1900	31.58
shopping_c...	01FBSNCQ...	label1	Toledo	39.9625508...	-4.8264091...	Talavera de...	2000	2800	71.43
shopping_c...	01FBSNCQ...	label1	Badajoz	38.8732794...	-6.9748074...	Conquistad...	200	2400	8.33
shopping_c...	01FBSNCQ...	label1	Alicante	38.3442102...	-0.4892461...	Alicante	1010	1900	53.16
shopping_c...	01FBSNCQ...	label1	Jaen	37.7757700...	-3.7859950...	Jaen	1500	1960	76.53
shopping_c...	01FBPQDN...	label1	Madrid	40.4485793...	-3.6919322...	Castellana	963	2590	37.18

Figure 26: SmartWorks collection with all the shopping centres

Then, with the *Panopticon* tool, a map is created in which all of the shopping centers are represented. Each shopping centre is shown in a particular color, depending on their overall occupancy rate. The colors follow a scale from red to green, in which red is the highest occupancy rate and green is the lowest occupancy rate, according to the following figure:

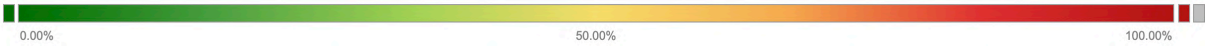

Figure 27: Color scale used for the occupancy rates

The resulting map is the following:

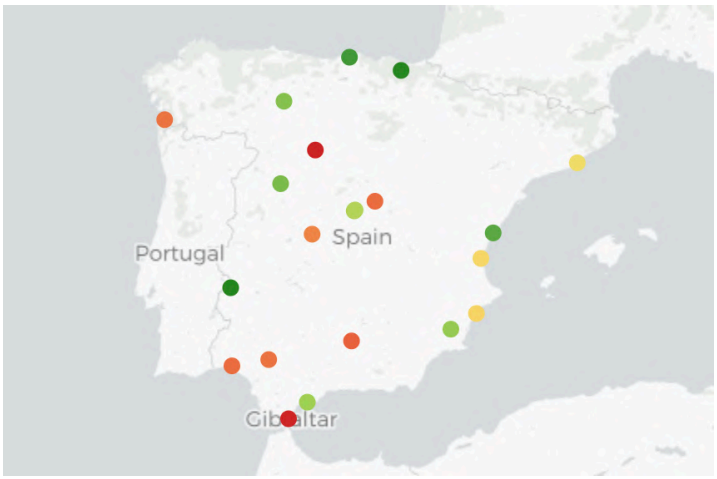

Figure 28: Resulting map with the shopping centres

As seen in the map, the shopping centre with the highest occupancy is Zorrilla (Valladolid):

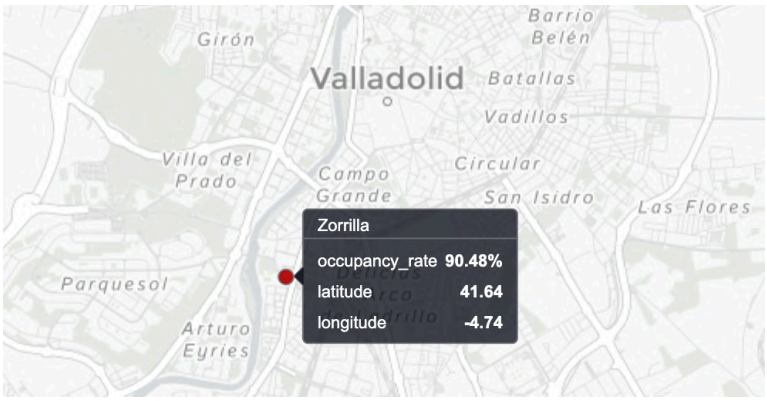

Figure 29: Information shown on the map for the Zorrilla shopping centre

And the shopping centre with the lowest occupancy rate is Conquistadores (Badajoz):

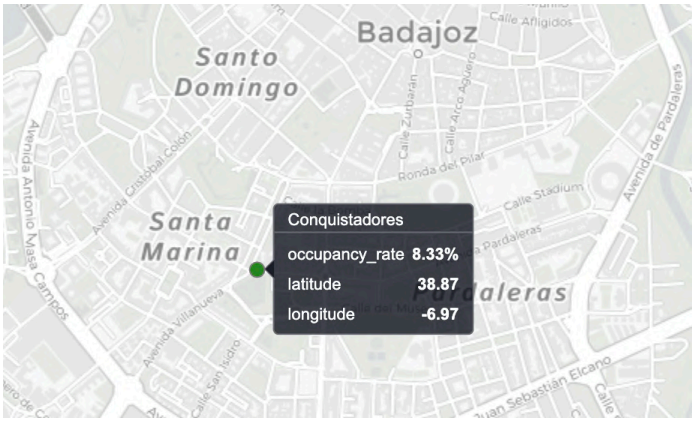

Figure 30: Information shown on the map for the Conquistadores shopping centre

The next step is to model the departments of the shopping centres. It has been considered that each shopping centre has the following 10 departments:

- Women's fashion
- Men's fashion
- Electronics
- Pets
- Toys
- Groceries
- Cosmetics
- Books
- Sports
- Homeware

For each department, two different devices have been modelled: an *access* people counting device, and a *zone* people counting device. The reason for this is to take advantage of the full potential of the prototype, since it is capable of counting the total number of people in an area (*zone* type) and counting the number of people that have entered and exited an area (*access* type). The information collected by both devices is contrasted, since the total number of people counted by the *zone* device should be equal to the number of people that have entered minus the number of people that have exited the area, which are counted by the *access* device.

Each device has been created as a *thing* in the *devices* collection in SmarWorks. Furthermore, the *zone* device *things* and the *access* device *things* have different properties but also some in common:

- **Zone device properties:** department, shopping centre, total count, capacity, occupancy rate.
- **Access device properties:** department, shopping centre, total entered, total exited.

The values of the aforementioned properties can be seen in the visualization dashboard for different shopping centres.

For the occupancy rate property a heatmap has been created, as seen in the figure below for the Castellana shopping centre:

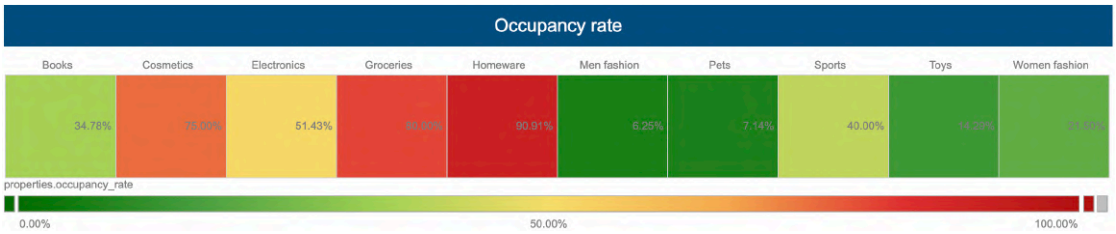

Figure 31: Occupancy rate heatmap for the Castellana shopping centre

As seen in the figure above, the department with the highest occupancy rate in the Castellana shopping centre is the Homeware department (90.91%), whereas the department with the lowest occupancy is the Men's fashion department (6.25%).

The heatmap created to show the occupancy rate for the Alicante shopping center is shown in the following figure:

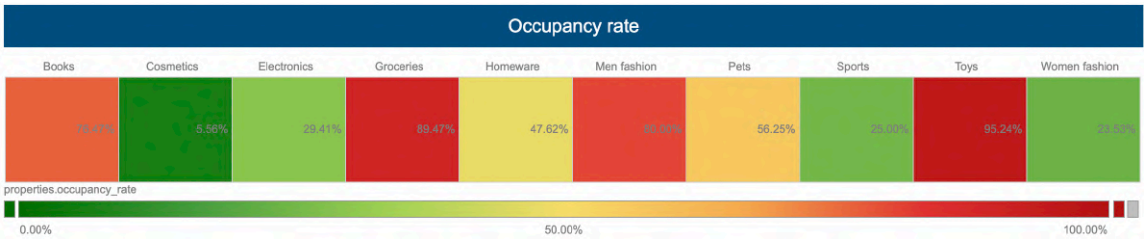

Figure 32: Occupancy rate heatmap for the Alicante shopping centre

As seen in the figure above, the department with the highest occupancy rate in the Alicante shopping centre is the Toys department (95.24%), whereas the department with the lowest occupancy is the Cosmetics department (5.56%).

Regarding the other properties, a table has been created in the visualization dashboard. The property values for the Castellana shopping centre are shown in the following table:

Occupancy data

	Count	Entered	Exited
Books	80	180	100
Cosmetics	150	650	500
Electronics	180	480	300
Groceries	200	540	340
Homeware	100	150	50
Men fashion	20	340	320
Pets	20	50	30
Sports	120	220	100
Toys	50	120	70
Women fash…	43	243	200

Table 5: Occupancy information for the Castellana shopping centre

Note that the data in the *Count* column would be collected by a *zone* device, whereas the data on the *Entered* and *Exited* columns would be collected by an *access* device.

The property values for the Alicante shopping centre are shown in the following table:

Occupancy data

	Count	Entered	Exited
Books	130	430	300
Cosmetics	10	30	20
Electronics	50	350	300
Groceries	170	670	500
Homeware	100	600	500
Men fashion	160	560	400
Pets	90	290	200
Sports	60	160	100
Toys	200	300	100
Women fash…	40	240	200

Table 6: Occupancy information for the Alicante shopping centre

The last step was to put all the aforementioned elements together in a dashboard. A screenshot of the resulting dashboard is shown in the following figure:

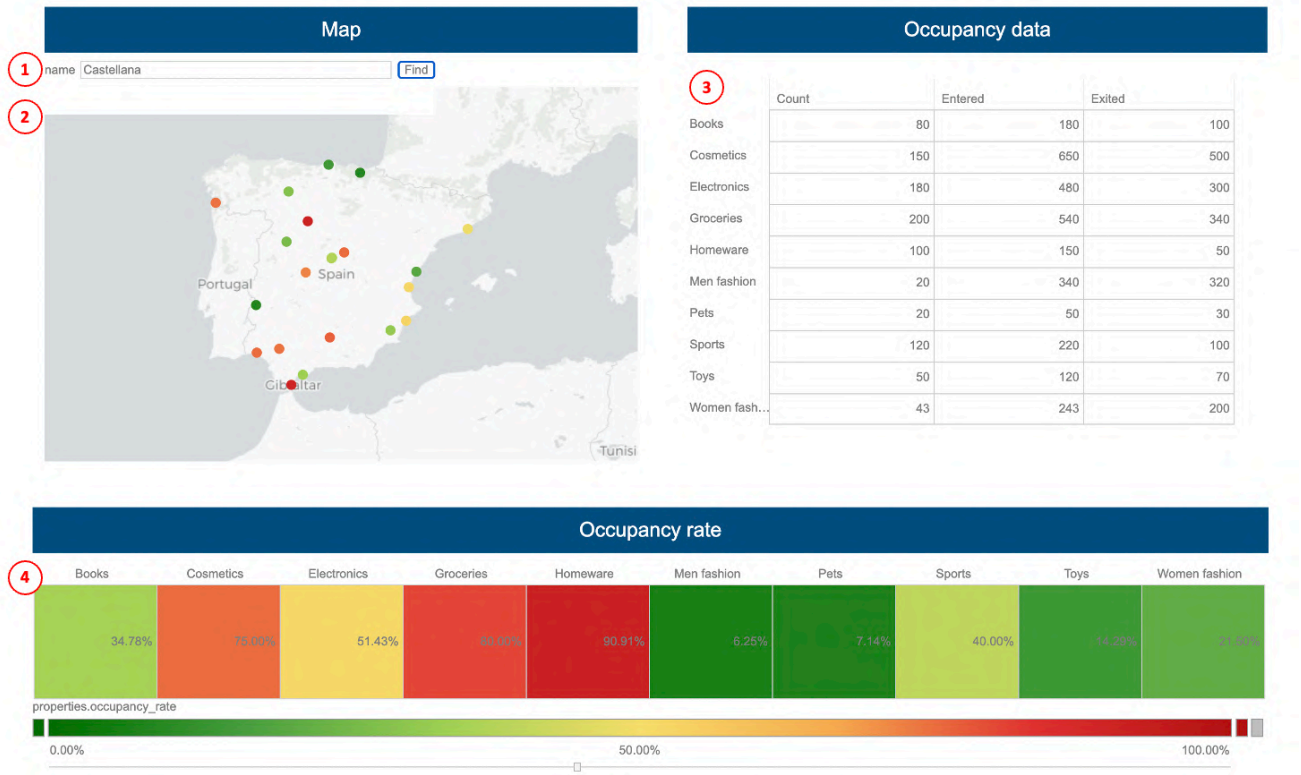

Figure 33: Visualization dashboard for the shopping centre SmartWorks model

The dashboard consists of the following elements:

1. **Shopping centre searchbar:** for the user to type in the name of the shopping center, then the table (3) and heatmap (4) will show the information for that particular shopping centre.
2. **Map:** to display the location of all the shopping centers, as well as their occupancy rate according to the color scale.
3. **Collected occupancy data:** shows the occupancy data collected by the *zone* and *access* devices in each department of the shopping centre typed in the searchbar (1).
4. **Heatmap:** shows the occupancy rate of the different departments of the shopping centre typed in the searchbar (1) in a visual way according to the color scale.

Finally, a further functionality has been implemented in the SmartWorks platform, which involves sending a message to the store manager whenever a department is at a 99% occupancy rate of its maximum capacity.

In this case, the system will send an email with the message “Hello, occupancy alert” to the store manager, which is achieved through a function which includes the following commands:

```
if occupancy_rate <= 99:
    return {"status_code":204}

data={
  "occupancy_alert":{
    "data":{
        "occupancy_rate":occupancy_rate
    }
  }

}


message="Hello, occupacy alert "
server=smtplib.SMTP('smtp.gmail.com',587)
server.starttls()
server.login('peoplecountingsystem@gmail.com','uuu222UUU')
server.sendmail('peoplecountingsystem@gmail.com','thestoremanager@gmail
  .com',message)
server.quit()
```

As seen in the script, whenever the occupancy rate in a department is above 99%, the system will send an email to the store manager.

5. Economic motivation for the project: Justification through a market sizing

A market sizing is carried out to have a view of the full economic potential and economic motivation of this project.

The target market at launch would be the Spanish market. Given the system is very versatile and can be used in different industries, the market sizing will be carried out assuming one large client in each industry. The target industries are:

- Large retail businesses:
    - Groceries
    - Malls
    - Fashion
    - Homeware
    - Sports
- Gyms
- Office and coworking companies

Additionally, the market sizing will be made for the business' basic service, which is the people counting system, charged as a yearly fee. Initial calculations used in the market sizing are the following:

- 1 device for every 50m^2 of surface
- COGS of 1 system: 100€
- Price of each system: 350€
- Unit gross margin of 71%

$$350\frac{€}{system} \times \frac{1\ system}{50m^2} = 7\frac{€}{m^2}\ revenue$$

If a 3-year fidelity is assumed (a relatively conservative assumption given SaaS assumes a 5 year fidelity), the desired total price of each system is divided by 3:

$$\frac{\left(7\,\frac{€}{m^2}\ revenue\right)}{3\ years} = 2{,}33\,\frac{€}{m^2 \times\ year}\ revenue$$

As mentioned above, the next step for the market sizing would be to calculate yearly revenues assuming that we capture one large client in each of the target industries.

Groceries retail- Mercadona:

- *Mercadona* average size: 1.800 m^2 *[8]*
- Number of *Mercadona* supermarkets in Spain: 1.621 in 2020 *[8]*
- Potential yearly revenue for the business:

$$\frac{(1.800\ m^2)}{1\ Mercadona} \times 2{,}33\,\frac{€\ revenue}{m^2 \times\ year} = 4.194\ \frac{€\ revenue}{1\ Mercadona \times\ year}$$

Considering all the *Mercadona* supermarkets:

$$4.194\ \frac{€\ revenue}{1\ Mercadona \times\ year}\ \times\ 1.621 = \frac{6{,}8\ million\ €\ revenue}{year}$$

Malls- El Corte Inglés:

- *El Corte Inglés* average size: 30.000 m^2 *[9]*
- Number of *El Corte Inglés* stores in Spain: 94 stores *[10]*
- Potential yearly revenue for the business:

$$\frac{(30.000\ m^2)}{1\ Corte\ Inglés} \times 2{,}33\,\frac{€\ revenue}{m^2 \times\ year} = 69.900\ \frac{€\ revenue}{1\ Corte\ Inglés \times\ year}$$

Considering all the *El Corte Inglés* stores:

$$69.900\ \frac{€\ revenue}{1\ Corte\ Inglés \times\ year}\ \times\ 94 = \frac{6{,}57\ million\ €\ revenue}{year}$$

Fashion retail- Zara:

- *Zara* store average size: 2.200 m^2 *[11]*
- Number of *Zara* stores in Spain: 449 stores *[12]*
- Potential yearly revenue for the business:

$$\frac{(2.200\ m^2)}{1\ Zara} \times 2{,}33\,\frac{€\ revenue}{m^2 \times\ year} = 5.126\ \frac{€\ revenue}{1\ Zara \times\ year}$$

Considering all the *Zara* stores:

$$5.126\ \frac{€\ revenue}{1\ Zara \times\ year}\ \times\ 449 = \frac{2{,}3\ million\ €\ revenue}{year}$$

Homeware- IKEA:

- *IKEA* store average size: 30.000 m^2 *[13]*
- Number of *IKEA* stores in Spain: 18 stores *[14]*

$$\frac{(30.000\ m^2)}{1\ IKEA} \times 2{,}33\ \frac{€\ revenue}{m^2 \times\ year} = 69.900\ \frac{€\ revenue}{1\ IKEA \times\ year}$$

Considering all the *IKEA* stores:

$$69.900\ \frac{€\ revenue}{1\ IKEA \times\ year}\ \times\ 18 = \frac{1{,}26\ million\ €\ revenue}{year}$$

Sports retail- Decathlon:

- *Decathlon* store average size: 2.000 m^2 *[15]*
- Number of *Decathlon* stores in Spain: 170 stores *[16]*

$$\frac{(2.000\ m^2)}{1\ Decathlon} \times 2{,}33\ \frac{€\ revenue}{m^2 \times\ year} = 4.660\ \frac{€\ revenue}{1\ Decathlon \times\ year}$$

Considering all the *Decathlon* stores:

$$4.660\ \frac{€\ \text{revenue}}{1\ \text{Decathlon} \times\ year}\ \times\ 170 = \frac{0{,}79\ million\ €\ \text{revenue}}{\text{year}}$$

Gyms- Altafit:

- *Altafit* gym average size: 1.400 m^2 *[17]*
- Number of *Altafit* gyms in Spain: 83 gyms *[17]*

$$\frac{(1.400\ m^2)}{1\ Altafit} \times 2{,}33\ \frac{€\ revenue}{m^2 \times\ year} = 3.262\ \frac{€\ revenue}{1\ Altafit \times\ year}$$

Considering all the *Altafit* gyms:

$$3.262\ \frac{€\ revenue}{1\ Altafit \times\ year} \times\ 83 = \frac{0{,}27\ million\ €\ revenue}{year}$$

Coworking- Utopicus:

- *Utopicus* offices average size: 2.720 m^2 *[18]*
- Number of *Utopicus* offices in Spain: 15 offices in 2020 *[18]*

$$\frac{(2.720\ m^2)}{1\ Utopicus} \times 2{,}33 \frac{€\ revenue}{m^2 \times\ year} = 6.338\ \frac{€\ revenue}{1\ Utopicus \times\ year}$$

Considering all the *Utopicus* offices:

$$6.338\ \frac{€\ revenue}{1\ Utopicus \times\ year} \times\ 15 = \frac{0{,}1\ million\ €\ revenue}{year}$$

A summary of the market sizing is shown in the following table:

Industry	Sub-sector	Target client	Average size (m^2)	Number of stores in Spain	Potential yearly revenue for the business (million €)
Retail	Groceries	Mercadona	1.800	1.621	6,8
	Malls	El Corte Inglés	30.000	94	6,5
	Fashion	Zara	2.200	449	2,3
	Homeware	IKEA	30.000	18	1,3
	Sports	Decathlon	2.000	170	0,8
Gyms		Altafit	1.400	83	0,3
Offices		Utopicus	2.720	15	0,1
Total			**70.120**	**2.450**	**18,1**

Table 7: Market sizing

As seen in the table shown above, an addressable market formed by one player from each industry would have a yearly market size of 18,1 million € in Spain.

The target clients selected for each industry are market leaders and do not have any people counting system implemented in their stores. The reason why only one player in each industry has been selected for sizing the market is because it is already ambitious to capture these 7

players entirely. In fact, the strategy is to initially sell the business' people counting system to a small number of stores for each of the 7 companies, with the goal of finally expanding to every store of these companies. The rationale behind this is that these companies will want to implement the system in all their stores once they try it, because it will give them great advantage over their competitors, since they will be the only ones in the market with a people counting system. Once the 7 selected companies are captured entirely, the business will continue capturing other large players in Spain within those industries, and finally, expand internationally.

For a further market and target client analysis, a breakdown of the potential yearly revenue for the business is shown in the following chart:

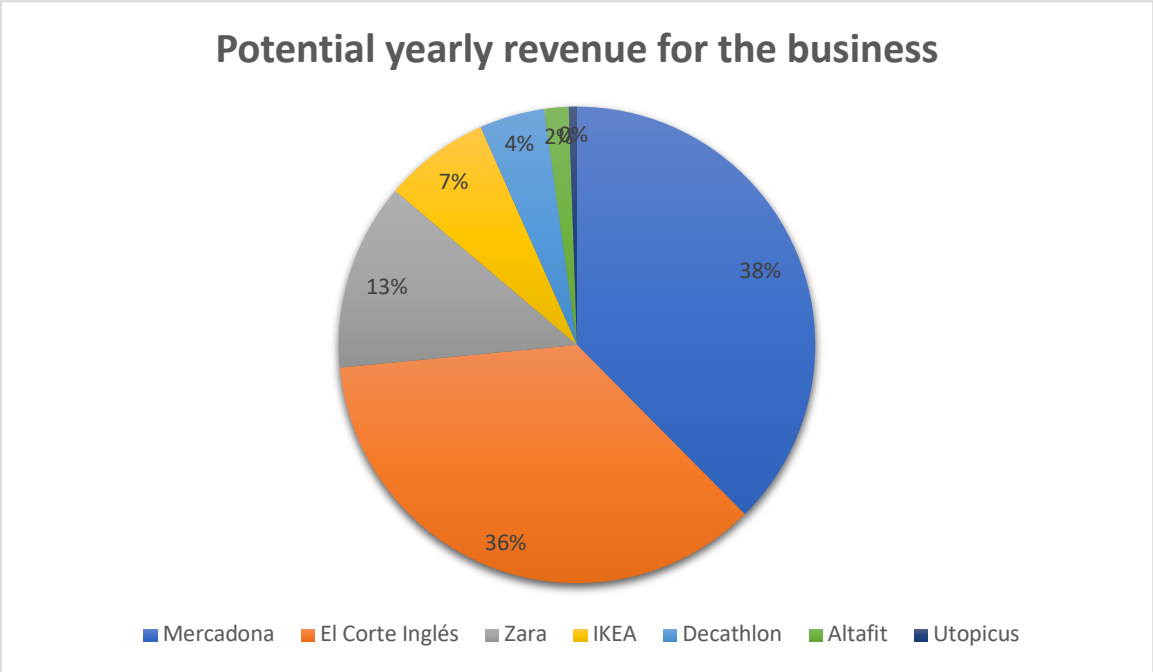

Figure 34: Potential yearly revenue for the business

As seen in the graph above, the 2 largest clients in terms of potential revenues for the business are *Mercadona* and *El Corte Inglés*, which are included in the grocery retail and mall industries, respectively. Some industries like gyms and office and coworking companies might seem less attractive because they are currently not as large. However, it is also interesting to target them to have a diversified client portfolio, since these industries might grow in the future or might unlock potential internationalization opportunities.

6. Conclusions

The main conclusion drawn from the project is that it is feasible to build a low-cost device which can track occupancy with a good accuracy.

Furthermore, the project shows the potential advantages for businesses and public spaces of implementing a low-cost people counting system. The lack of this type of technology at affordable prices is a clear indicator of an opportunity to launch such a product.

While developing the prototype, it was clear that a combination of an object detector and an object tracker was the best possible approach to tackle efficiency issues. Then, in the process of adapting the algorithm to make it work in a device with a less powerful processor (Raspberry Pi 4), it was concluded that the Deep Learning Single Shot Detector had to be replaced with a Moving Object Detector, which, although less accurate, it is much lighter at a computational level, and more convenient for the Raspberry Pi. Additionally, the advantages of using dynamic background subtraction were visible in the experiments, allowing the system to adapt to changes in the environment.

Regarding cloud connectivity, the Altair SmartWorks platform used in the project has made it possible to connect the device to the cloud and design user-friendly visualizations. Further functionalities of SmartWorks were also useful when building the shopping centre model. The model allows the user to easily monitor occupancy levels in the different departments of shopping centres, warning whenever the occupancy of a department is reaching its maximum capacity.

7. Sources of information

[1] Room occupancy estimation through WIFI, UWB, and Light Sensors mounted on doorways- Hessam Mohammadmoradi. University of Houston

[2] https://v-count.com/people-counting-technologies-a-comprehensive-guide/

[3] https://es.v-count.com/soluciones/contador-de-personas/

[4] https://www.footfallcam.com/Product/FootfallCam-3D-Max

[5] https://www.pyimagesearch.com/2018/08/13/opencv-people-counter/

[6] https://www.hackster.io/phfbertoleti/counting-objects-in-movement-using-raspberry-pi-opencv-015ba5

[7] https://www.pyimagesearch.com/2015/06/01/home-surveillance-and-motion-detection-with-the-raspberry-pi-python-and-opencv/

[8] https://www.elconfidencial.com/empresas/2021-01-06/supermercados-mercadona-carrefour-dia_2896439/

[9]https://www.expansion.com/empresas/distribucion/2019/05/12/5cd7e79822601d15048b45df.html

[10] https://www.elcorteingles.es/centroscomerciales/es/eci/centros?page=4

[11] https://elpais.com/economia/2019/03/13/actualidad/1552475635_649725.html

[12] https://es.wikipedia.org/wiki/Zara

[13] https://www.revistagq.com/noticias/articulo/ikea-mas-grande-del-mundo-superficie-cuanto-mide-filipinas-campos-de-futbol

[14] https://www.enterat.com/estilo/ikea-tiendas-espana.php

[15] https://www.business-standard.com/article/pti-stories/decathlon-to-meet-30-per-cent-local-sourcing-norms-in-5-years-114060100108_1.html

[16] https://www.palco23.com/equipamiento/decathlon-un-negocio-de-1660-millones-y-170-tiendas-que-pivota-al-centro-de-espana.html

[17] https://www.palco23.com/fitness/altafit-sigue-creciendo-pese-a-la-crisis-con-una-nueva-apertura-en-madrid.html

[18] https://www.elconfidencial.com/empresas/2019-10-22/utopicus-coworking-oficinas-flexibles-bra_2147603/

Annexes

Annex I: Initial code from *pyimagesearch* with a Single Shot Detector

```
# USAGE
# To read and write back out to video:
# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \
#   --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel --input videos/example_01.mp4 \
#   --output output/output_01.avi
#
# To read from webcam and write back out to disk:
# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \
#   --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel \
#   --output output/webcam_output.avi

# import the necessary packages
from pyimagesearch.centroidtracker import CentroidTracker
from pyimagesearch.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("-i", "--input", type=str,
    help="path to optional input video file")
ap.add_argument("-o", "--output", type=str,
    help="path to optional output video file")
ap.add_argument("-c", "--confidence", type=float, default=0.4,
    help="minimum probability to filter weak detections")
ap.add_argument("-s", "--skip-frames", type=int, default=30,
    help="# of skip frames between detections")
args = vars(ap.parse_args())

```

```
# initialize the list of class labels MobileNet SSD was trained to
# detect
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# if a video path was not supplied, grab a reference to the webcam
if not args.get("input", False):
    print("[INFO] starting video stream...")
    vs = VideoStream(src=0).start()
    time.sleep(2.0)

# otherwise, grab a reference to the video file
else:
    print("[INFO] opening video file...")
    vs = cv2.VideoCapture(args["input"])

# initialize the video writer (we'll instantiate later if need be)
writer = None

# initialize the frame dimensions (we'll set them as soon as we read
# the first frame from the video)
W = None
H = None

# instantiate our centroid tracker, then initialize a list to store
# each of our dlib correlation trackers, followed by a dictionary to
# map each unique object ID to a TrackableObject
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = {}

# initialize the total number of frames processed thus far, along
# with the total number of objects that have moved either up or down
totalFrames = 0
totalDown = 0
totalUp = 0

```

```
# start the frames per second throughput estimator
fps = FPS().start()

# loop over frames from the video stream
while True:
    # grab the next frame and handle if we are reading from either
    # VideoCapture or VideoStream
    frame = vs.read()
    frame = frame[1] if args.get("input", False) else frame

    # if we are viewing a video and we did not grab a frame then we
    # have reached the end of the video
    if args["input"] is not None and frame is None:
        break

    # resize the frame to have a maximum width of 500 pixels (the
    # less data we have, the faster we can process it), then convert
    # the frame from BGR to RGB for dlib
    frame = imutils.resize(frame, width=500)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # if the frame dimensions are empty, set them
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    # if we are supposed to be writing a video to disk, initialize
    # the writer
    if args["output"] is not None and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 30,
            (W, H), True)

    # initialize the current status along with our list of bounding
    # box rectangles returned by either (1) our object detector or
    # (2) the correlation trackers
    status = "Waiting"
    rects = []

```

```
    # check to see if we should run a more computationally expensive
    # object detection method to aid our tracker
    if totalFrames % args["skip_frames"] == 0:
        # set the status and initialize our new set of object trackers
        status = "Detecting"
        trackers = []

        # convert the frame to a blob and pass the blob through the
        # network and obtain the detections
        blob = cv2.dnn.blobFromImage(frame, 0.007843, (W, H), 127.5)
        net.setInput(blob)
        detections = net.forward()

        # loop over the detections
        for i in np.arange(0, detections.shape[2]):
            # extract the confidence (i.e., probability) associated
            # with the prediction
            confidence = detections[0, 0, i, 2]

            # filter out weak detections by requiring a minimum
            # confidence
            if confidence > args["confidence"]:
                # extract the index of the class label from the
                # detections list
                idx = int(detections[0, 0, i, 1])

                # if the class label is not a person, ignore it
                if CLASSES[idx] != "person":
                    continue

                # compute the (x, y)-coordinates of the bounding box
                # for the object
                box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
                (startX, startY, endX, endY) = box.astype("int")

                # construct a dlib rectangle object from the bounding
                # box coordinates and then start the dlib correlation
                # tracker
                tracker = dlib.correlation_tracker()
                rect = dlib.rectangle(startX, startY, endX, endY)
                tracker.start_track(rgb, rect)

                # add the tracker to our list of trackers so we can
                # utilize it during skip frames
                trackers.append(tracker)

```

```
    # otherwise, we should utilize our object *trackers* rather than
    # object *detectors* to obtain a higher frame processing throughput
    else:
        # loop over the trackers
        for tracker in trackers:
            # set the status of our system to be 'tracking' rather
            # than 'waiting' or 'detecting'
            status = "Tracking"

            # update the tracker and grab the updated position
            tracker.update(rgb)
            pos = tracker.get_position()

            # unpack the position object
            startX = int(pos.left())
            startY = int(pos.top())
            endX = int(pos.right())
            endY = int(pos.bottom())

            # add the bounding box coordinates to the rectangles list
            rects.append((startX, startY, endX, endY))

    # draw a horizontal line in the center of the frame -- once an
    # object crosses this line we will determine whether they were
    # moving 'up' or 'down'
    cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)

    # use the centroid tracker to associate the (1) old object
    # centroids with (2) the newly computed object centroids
    objects = ct.update(rects)

    # loop over the tracked objects
    for (objectID, centroid) in objects.items():
        # check to see if a trackable object exists for the current
        # object ID
        to = trackableObjects.get(objectID, None)

        # if there is no existing trackable object, create one
        if to is None:
            to = TrackableObject(objectID, centroid)

```

```
        # otherwise, there is a trackable object so we can utilize it
        # to determine direction
        else:
            # the difference between the y-coordinate of the *current*
            # centroid and the mean of *previous* centroids will tell
            # us in which direction the object is moving (negative for
            # 'up' and positive for 'down')
            y = [c[1] for c in to.centroids]
            direction = centroid[1] - np.mean(y)
            to.centroids.append(centroid)

            # check to see if the object has been counted or not
            if not to.counted:
                # if the direction is negative (indicating the object
                # is moving up) AND the centroid is above the center
                # line, count the object
                if direction < 0 and centroid[1] < H // 2:
                    totalUp += 1
                    to.counted = True

                # if the direction is positive (indicating the object
                # is moving down) AND the centroid is below the
                # center line, count the object
                elif direction > 0 and centroid[1] > H // 2:
                    totalDown += 1
                    to.counted = True

        # store the trackable object in our dictionary
        trackableObjects[objectID] = to

        # draw both the ID of the object and the centroid of the
        # object on the output frame
        text = "ID {}".format(objectID)
        cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)

```

```
245     # construct a tuple of information we will be displaying on the
246     # frame
247     info = [
248         ("Up", totalUp),
249         ("Down", totalDown),
250         ("Status", status),
251     ]
252 
253     # loop over the info tuples and draw them on our frame
254     for (i, (k, v)) in enumerate(info):
255         text = "{}: {}".format(k, v)
256         cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
257             cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
258 
259     # check to see if we should write the frame to disk
260     if writer is not None:
261         writer.write(frame)
262 
263     # show the output frame
264     cv2.imshow("Frame", frame)
265     key = cv2.waitKey(1) & 0xFF
266 
267     # if the `q` key was pressed, break from the loop
268     if key == ord("q"):
269         break
270 
271     # increment the total number of frames processed thus far and
272     # then update the FPS counter
273     totalFrames += 1
274     fps.update()
275 
276 # stop the timer and display FPS information
277 fps.stop()
278 print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
279 print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))
280 
281 # check to see if we need to release the video writer pointer
282 if writer is not None:
283     writer.release()
284 
```

```
# if we are not using a video file, stop the camera video stream
if not args.get("input", False):
    vs.stop()

# otherwise, release the video file pointer
else:
    vs.release()

# close any open windows
cv2.destroyAllWindows()

```

Annex II. Complete final code with a moving object detector, dynamic background subtraction, and cloud connectivity with SmartWorks: Zone 1

```
# USAGE
# To read and write back out to video:
# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \
#   --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel --input videos/example_01.mp4 \
#   --output output/output_01.avi
#
# To read from webcam and write back out to disk:
# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \
#   --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel \
#   --output output/webcam_output.avi

# import the necessary packages
from pyimagesearch.centroidtracker import CentroidTracker
from pyimagesearch.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
from IPython.display import clear_output
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
import modules.smartworks_sdk as swx
import modules.credentials as credentials
import paho.mqtt.client as mqtt
import time
import json

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("-i", "--input", type=str,
    help="path to optional input video file")
ap.add_argument("-o", "--output", type=str,
    help="path to optional output video file")
ap.add_argument("-c", "--confidence", type=float, default=0.4,
    help="minimum probability to filter weak detections")
ap.add_argument("-s", "--skip-frames", type=int, default=30,
    help="# of skip frames between detections")
args = vars(ap.parse_args())

```

```
# initialize the list of class labels MobileNet SSD was trained to
# detect
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# if a video path was not supplied, grab a reference to the webcam
if not args.get("input", False):
    print("[INFO] starting video stream...")
    vs = VideoStream(src=0).start()
    time.sleep(2.0)

# otherwise, grab a reference to the video file
else:
    print("[INFO] opening video file...")
    vs = cv2.VideoCapture(args["input"])

# initialize the video writer (we'll instantiate later if need be)
writer = None

# initialize the frame dimensions (we'll set them as soon as we read
# the first frame from the video)
W = None
H = None

# instantiate our centroid tracker, then initialize a list to store
# each of our dlib correlation trackers, followed by a dictionary to
# map each unique object ID to a TrackableObject
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = {}

# initialize the total number of frames processed thus far, along
# with the total number of objects that have moved either up or down
totalFrames = 0
counter1 = 0
counter2 = 0

```

```
MinContourArea = 3000
BinarizationThreshold = 25  #Adjust ths value according to your usage
# start the frames per second throughput estimator
fps = FPS().start()

avg = None

#SmartWorks-create a space for the area being recorded
space = "twozones"

#SmartWorks-create a collection for all the area spaces
collection = "counters"

#SmartWorks- we store the thing id
thing_id_1 = "01F83CSPWX9MHXQY8Z9RTAAMN5"
# thing_id_2= "01F83CXR2K740CAP333CDDTQYG"

#SmartWorks- MQTT Variables
host="mqtt.swx.altairone.com"
user_1 = 'zone1@twozones'
password_1 = 'this1'
# user_2 = 'zone2@twozones'
# password_2 = 'this2'

#Smartworks- We connect to the broker
client = mqtt.Client()
client.username_pw_set(username=user_1,password=password_1)
client.connect(host, port=1883, keepalive=10, bind_address="")

# client = mqtt.Client()
# client.username_pw_set(username=user_2,password=password_2)
# client.connect(host, port=1883, keepalive=10, bind_address="")

#SmartWorks-sending data
data_topic_1 = "set/{}/collections/{}/things/{}/data".format(space, collection, thing_id_1)
print ("Data topic: "+data_topic_1)

```

```
property_topic_1 = "set/{}/collections/{}/things/{}/properties/{}".format(space, collection, thing_id_1, "people_count")
print ("Property topic: "+property_topic_1)

# data_topic_2 = "set/{}/collections/{}/things/{}/data".format(space, collection, thing_id_2)
# print ("Data topic: "+data_topic_2)
#
# property_topic_2 = "set/{}/collections/{}/things/{}/properties/{}".format(space, collection, thing_id_2, "people_count")
# print ("Property topic: "+property_topic_2)

# loop over frames from the video stream
while True:
    # grab the next frame and handle if we are reading from either
    # VideoCapture or VideoStream
    frame = vs.read()
    frame = frame[1] if args.get("input", False) else frame

    # if we are viewing a video and we did not grab a frame then we
    # have reached the end of the video
    if args["input"] is not None and frame is None:
        break

    # resize the frame to have a maximum width of 500 pixels (the
    # less data we have, the faster we can process it), then convert
    # the frame from BGR to RGB for dlib
    frame = imutils.resize(frame, width=500)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # if the frame dimensions are empty, set them
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    # if we are supposed to be writing a video to disk, initialize
    # the writer
    if args["output"] is not None and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 30,
            (W, H), True)

    # initialize the current status along with our list of bounding
    # box rectangles returned by either (1) our object detector or
    # (2) the correlation trackers
    status = "Waiting"
    rects = []

```

```
    # check to see if we should run a more computationally expensive
    # object detection method to aid our tracker
    if totalFrames % args["skip_frames"] == 0:
        # set the status and initialize our new set of object trackers
        status = "Detecting"
        trackers = []

        #gray-scale convertion and Gaussian blur filter applying
        GrayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        GrayFrame = cv2.GaussianBlur(GrayFrame, (21, 21), 0)

        if avg is None:
            #print("[INFO] starting background model...")
            avg = GrayFrame.copy().astype("float")
            #avg = GrayFrame
            #rawCapture.truncate(0)
            continue

        #Background subtraction and image binarization
        cv2.accumulateWeighted(GrayFrame, avg, 0.5)
        #FrameDelta = cv2.absdiff(GrayFrame, avg)
        FrameDelta = cv2.absdiff(GrayFrame, cv2.convertScaleAbs(avg))
        FrameThresh = cv2.threshold(FrameDelta, BinarizationThreshold, 255, cv2.THRESH_BINARY)[1]

        #Dilate image and find all the contours
        FrameThresh = cv2.dilate(FrameThresh, None, iterations=2)
        cnts = cv2.findContours(FrameThresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        #_, cnts, _ = cv2.findContours(FrameThresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        QttyOfContours = 0

        #check all found countours
        for c in cnts:

            if cv2.contourArea(c) < MinContourArea:
                continue

            QttyOfContours = QttyOfContours+1

            #draw an rectangle "around" the object
            (x, y, w, h) = cv2.boundingRect(c)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
            # compute the (x, y)-coordinates of the bounding box
            # for the object
            #box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
            #(startX, startY, endX, endY) = box.astype("int")

            # construct a dlib rectangle object from the bounding
            # box coordinates and then start the dlib correlation
            # tracker
            tracker = dlib.correlation_tracker()
            rect = dlib.rectangle(x, y, x + w, y + h)
            tracker.start_track(rgb, rect)

            # add the tracker to our list of trackers so we can
            # utilize it during skip frames
            trackers.append(tracker)

    # otherwise, we should utilize our object *trackers* rather than
    # object *detectors* to obtain a higher frame processing throughput
    else:
        # loop over the trackers
        for tracker in trackers:
            # set the status of our system to be 'tracking' rather
            # than 'waiting' or 'detecting'
            status = "Tracking"

            # update the tracker and grab the updated position
            tracker.update(rgb)
            pos = tracker.get_position()

            # unpack the position object
            startX = int(pos.left())
            startY = int(pos.top())
            endX = int(pos.right())
            endY = int(pos.bottom())

            # add the bounding box coordinates to the rectangles list
            rects.append((startX, startY, endX, endY))

    # draw a horizontal line in the center of the frame -- once an
    # object crosses this line we will determine whether they were
    # moving 'up' or 'down'
    cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)

    # use the centroid tracker to associate the (1) old object
    # centroids with (2) the newly computed object centroids
    objects = ct.update(rects)

```

```
    counter1 = 0
    counter2 = 0

    # loop over the tracked objects
    for (objectID, centroid) in objects.items():
        # check to see if a trackable object exists for the current
        # object ID
        to = trackableObjects.get(objectID, None)

        # if there is no existing trackable object, create one
        if to is None:
            to = TrackableObject(objectID, centroid)

        # otherwise, there is a trackable object so we can utilize it
        # to determine direction
        else:
            if centroid[1] < H // 2:
                counter1 += 1
            elif centroid[1] > H // 2:
                counter2 += 1

        # store the trackable object in our dictionary
        trackableObjects[objectID] = to

        # draw both the ID of the object and the centroid of the
        # object on the output frame
        text = "ID {}".format(objectID)
        cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)

    # construct a tuple of information we will be displaying on the
    # frame
    info = [
                ("Zone2 (Bottom)", counter2),
        ("Zone1 (Top)", counter1),
        ("Status", status),
    ]
```

```
    #SmartWorks- publish all data and properties
    jsonarray1 = {
            "people_count": str(counter1),
            }
    client.publish(topic = data_topic_1 ,payload = str(json.dumps(jsonarray1)))
    property_people_count = {
            "people_count": str(counter1)
            }
    client.publish(topic = property_topic_1 ,payload = str(json.dumps(property_people_count)))

#   jsonarray2 = {
#             "people_count": str(counter2),
#             }
#   client.publish(topic = data_topic_2 ,payload = str(json.dumps(jsonarray2)))
#   property_people_count = {
#             "people_count": str(counter2)
#             }
#   client.publish(topic = property_topic_2 ,payload = str(json.dumps(property_people_count)))


    # loop over the info tuples and draw them on our frame
    for (i, (k, v)) in enumerate(info):
        text = "{}: {}".format(k, v)
        cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
            cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)

    # check to see if we should write the frame to disk
    if writer is not None:
        writer.write(frame)

    # show the output frame
    cv2.imshow("Frame", frame)
    key = cv2.waitKey(1) & 0xFF

    # if the `q` key was pressed, break from the loop
    if key == ord("q"):
        break

    # increment the total number of frames processed thus far and
    # then update the FPS counter
    totalFrames += 1
    fps.update()

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

```

```
# check to see if we need to release the video writer pointer
if writer is not None:
    writer.release()

# if we are not using a video file, stop the camera video stream
if not args.get("input", False):
    vs.stop()

# otherwise, release the video file pointer
else:
    vs.release()

# close any open windows
cv2.destroyAllWindows()

```

Annex III. Complete final code with a moving object detector, dynamic background subtraction, and cloud connectivity with SmartWorks: Accesses

```
# USAGE
# To read and write back out to video:
# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \
#   --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel --input videos/example_01.mp4 \
#   --output output/output_01.avi
#
# To read from webcam and write back out to disk:
# python people_counter.py --prototxt mobilenet_ssd/MobileNetSSD_deploy.prototxt \
#   --model mobilenet_ssd/MobileNetSSD_deploy.caffemodel \
#   --output output/webcam_output.avi

# import the necessary packages
from pyimagesearch.centroidtracker import CentroidTracker
from pyimagesearch.trackableobject import TrackableObject
from imutils.video import VideoStream
from imutils.video import FPS
from IPython.display import clear_output
import numpy as np
import argparse
import imutils
import time
import dlib
import cv2
import modules.smartworks_sdk as swx
import modules.credentials as credentials
import paho.mqtt.client as mqtt
import time
import json


# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-p", "--prototxt", required=True,
    help="path to Caffe 'deploy' prototxt file")
ap.add_argument("-m", "--model", required=True,
    help="path to Caffe pre-trained model")
ap.add_argument("-i", "--input", type=str,
    help="path to optional input video file")
ap.add_argument("-o", "--output", type=str,
    help="path to optional output video file")
ap.add_argument("-c", "--confidence", type=float, default=0.4,
    help="minimum probability to filter weak detections")
ap.add_argument("-s", "--skip-frames", type=int, default=30,
    help="# of skip frames between detections")
args = vars(ap.parse_args())

```

```
# initialize the list of class labels MobileNet SSD was trained to
# detect
CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
    "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
    "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
    "sofa", "train", "tvmonitor"]

# load our serialized model from disk
print("[INFO] loading model...")
net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])

# if a video path was not supplied, grab a reference to the webcam
if not args.get("input", False):
    print("[INFO] starting video stream...")
    vs = VideoStream(src=0).start()
    time.sleep(2.0)

# otherwise, grab a reference to the video file
else:
    print("[INFO] opening video file...")
    vs = cv2.VideoCapture(args["input"])

# initialize the video writer (we'll instantiate later if need be)
writer = None

# initialize the frame dimensions (we'll set them as soon as we read
# the first frame from the video)
W = None
H = None

# instantiate our centroid tracker, then initialize a list to store
# each of our dlib correlation trackers, followed by a dictionary to
# map each unique object ID to a TrackableObject
ct = CentroidTracker(maxDisappeared=40, maxDistance=50)
trackers = []
trackableObjects = {}

# initialize the total number of frames processed thus far, along
# with the total number of objects that have moved either up or down
totalFrames = 0
totalDown = 0
totalUp = 0
insideBalance = 0

```

```
MinContourArea = 3000
BinarizationThreshold = 25  #Adjust ths value according to your usage
# start the frames per second throughput estimator
fps = FPS().start()

avg = None

#SmartWorks-create a space for the area being recorded
space = "accesses"

#SmartWorks-create a collection for all the area spaces
collection = "people"

#SmartWorks- we store the thing id
thing_id = "01F813AS2JSXG0XNTKRNW29EGS"

#SmartWorks- MQTT Variables
host="mqtt.swx.altairone.com"
user = 'people@accesses'
password = 'cualquiera'

#Smartworks- We connect to the broker
client = mqtt.Client()
client.username_pw_set(username=user,password=password)
client.connect(host, port=1883, keepalive=10, bind_address="")


#SmartWorks-sending data
data_topic = "set/{}/collections/{}/things/{}/data".format(space, collection, thing_id)
print ("Data topic: "+data_topic)

property_topic = "set/{}/collections/{}/things/{}/properties/{}".format(space, collection, thing_id, "going_in")
print ("Property topic: "+property_topic)


# loop over frames from the video stream
while True:
    # grab the next frame and handle if we are reading from either
    # VideoCapture or VideoStream
    frame = vs.read()
    frame = frame[1] if args.get("input", False) else frame

    # if we are viewing a video and we did not grab a frame then we
    # have reached the end of the video
    if args["input"] is not None and frame is None:
        break

```

```
    # resize the frame to have a maximum width of 500 pixels (the
    # less data we have, the faster we can process it), then convert
    # the frame from BGR to RGB for dlib
    frame = imutils.resize(frame, width=500)
    rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    # if the frame dimensions are empty, set them
    if W is None or H is None:
        (H, W) = frame.shape[:2]

    # if we are supposed to be writing a video to disk, initialize
    # the writer
    if args["output"] is not None and writer is None:
        fourcc = cv2.VideoWriter_fourcc(*"MJPG")
        writer = cv2.VideoWriter(args["output"], fourcc, 30,
            (W, H), True)

    # initialize the current status along with our list of bounding
    # box rectangles returned by either (1) our object detector or
    # (2) the correlation trackers
    status = "Waiting"
    rects = []

    # check to see if we should run a more computationally expensive
    # object detection method to aid our tracker
    if totalFrames % args["skip_frames"] == 0:
        # set the status and initialize our new set of object trackers
        status = "Detecting"
        trackers = []

        #gray-scale convertion and Gaussian blur filter applying
        GrayFrame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
        GrayFrame = cv2.GaussianBlur(GrayFrame, (21, 21), 0)

        if avg is None:
            #print("[INFO] starting background model...")
            avg = GrayFrame.copy().astype("float")
            #avg = GrayFrame
            #rawCapture.truncate(0)
            continue
```

```
        #Background subtraction and image binarization
        cv2.accumulateWeighted(GrayFrame, avg, 0.5)
        #FrameDelta = cv2.absdiff(GrayFrame, avg)
        FrameDelta = cv2.absdiff(GrayFrame, cv2.convertScaleAbs(avg))
        FrameThresh = cv2.threshold(FrameDelta, BinarizationThreshold, 255, cv2.THRESH_BINARY)[1]

        #Dilate image and find all the contours
        FrameThresh = cv2.dilate(FrameThresh, None, iterations=2)
        cnts = cv2.findContours(FrameThresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        cnts = imutils.grab_contours(cnts)
        #_, cnts, _ = cv2.findContours(FrameThresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

        QttyOfContours = 0

        #check all found countours
        for c in cnts:

            if cv2.contourArea(c) < MinContourArea:
                continue

            QttyOfContours = QttyOfContours+1

            #draw an rectangle "around" the object
            (x, y, w, h) = cv2.boundingRect(c)
            cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)


            # compute the (x, y)-coordinates of the bounding box
            # for the object
            #box = detections[0, 0, i, 3:7] * np.array([W, H, W, H])
            #(startX, startY, endX, endY) = box.astype("int")

            # construct a dlib rectangle object from the bounding
            # box coordinates and then start the dlib correlation
            # tracker
            tracker = dlib.correlation_tracker()
            rect = dlib.rectangle(x, y, x + w, y + h)
            tracker.start_track(rgb, rect)

            # add the tracker to our list of trackers so we can
            # utilize it during skip frames
            trackers.append(tracker)

```

```
    # otherwise, we should utilize our object *trackers* rather than
    # object *detectors* to obtain a higher frame processing throughput
    else:
        # loop over the trackers
        for tracker in trackers:
            # set the status of our system to be 'tracking' rather
            # than 'waiting' or 'detecting'
            status = "Tracking"

            # update the tracker and grab the updated position
            tracker.update(rgb)
            pos = tracker.get_position()

            # unpack the position object
            startX = int(pos.left())
            startY = int(pos.top())
            endX = int(pos.right())
            endY = int(pos.bottom())

            # add the bounding box coordinates to the rectangles list
            rects.append((startX, startY, endX, endY))

    # draw a horizontal line in the center of the frame -- once an
    # object crosses this line we will determine whether they were
    # moving 'up' or 'down'
    cv2.line(frame, (0, H // 2), (W, H // 2), (0, 255, 255), 2)

    # use the centroid tracker to associate the (1) old object
    # centroids with (2) the newly computed object centroids
    objects = ct.update(rects)

    # loop over the tracked objects
    for (objectID, centroid) in objects.items():
        # check to see if a trackable object exists for the current
        # object ID
        to = trackableObjects.get(objectID, None)

        # if there is no existing trackable object, create one
        if to is None:
            to = TrackableObject(objectID, centroid)

        # otherwise, there is a trackable object so we can utilize it
        # to determine direction
```

```
        else:
            # the difference between the y-coordinate of the *current*
            # centroid and the mean of *previous* centroids will tell
            # us in which direction the object is moving (negative for
            # 'up' and positive for 'down')
            y = [c[1] for c in to.centroids]
            direction = centroid[1] - np.mean(y)
            to.centroids.append(centroid)

            # check to see if the object has been counted or not
            if not to.counted:
                # if the direction is negative (indicating the object
                # is moving up) AND the centroid is above the center
                # line, count the object
                if direction < 0 and centroid[1] < H // 2:
                    totalUp += 1
                    to.counted = True

                # if the direction is positive (indicating the object
                # is moving down) AND the centroid is below the
                # center line, count the object
                elif direction > 0 and centroid[1] > H // 2:
                    totalDown += 1
                    to.counted = True

        # store the trackable object in our dictionary
        trackableObjects[objectID] = to

        # draw both the ID of the object and the centroid of the
        # object on the output frame
        text = "ID {}".format(objectID)
        cv2.putText(frame, text, (centroid[0] - 10, centroid[1] - 10),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2)
        cv2.circle(frame, (centroid[0], centroid[1]), 4, (0, 255, 0), -1)

    insideBalance = totalDown - totalUp

    # construct a tuple of information we will be displaying on the
    # frame
    info = [
        ("Inside", insideBalance),
                ("Going in", totalDown),
        ("Going out", totalUp),
        ("Status", status),
    ]

```

```
311         #SmartWorks- publish all data and properties
312         jsonarray = {
313                 "tot_in": str(insideBalance),
314                 "going_in": str(totalDown),
315                 "going_out": str(totalUp),
316                 }
317         client.publish(topic = data_topic ,payload = str(json.dumps(jsonarray)))
318         property_tot_in = {
319                 "tot_in": str(insideBalance)
320                 }
321         property_going_in = {
322                 "going_in": str(totalDown)
323                 }
324         property_going_out = {
325                 "going_out": str(totalUp)
326                 }
327
328         client.publish(topic = property_topic ,payload = str(json.dumps(property_tot_in)))
329         client.publish(topic = property_topic ,payload = str(json.dumps(property_going_in)))
330         client.publish(topic = property_topic ,payload = str(json.dumps(property_going_out)))
331
332         # loop over the info tuples and draw them on our frame
333         for (i, (k, v)) in enumerate(info):
334             text = "{}: {}".format(k, v)
335             cv2.putText(frame, text, (10, H - ((i * 20) + 20)),
336                 cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255), 2)
337
338         # check to see if we should write the frame to disk
339         if writer is not None:
340             writer.write(frame)
341
342         # show the output frame
343         cv2.imshow("Frame", frame)
344         key = cv2.waitKey(1) & 0xFF
345
346         # if the `q` key was pressed, break from the loop
347         if key == ord("q"):
348             break
349
```

```
    # increment the total number of frames processed thus far and
    # then update the FPS counter
    totalFrames += 1
    fps.update()

# stop the timer and display FPS information
fps.stop()
print("[INFO] elapsed time: {:.2f}".format(fps.elapsed()))
print("[INFO] approx. FPS: {:.2f}".format(fps.fps()))

# check to see if we need to release the video writer pointer
if writer is not None:
    writer.release()

# if we are not using a video file, stop the camera video stream
if not args.get("input", False):
    vs.stop()

# otherwise, release the video file pointer
else:
    vs.release()

# close any open windows
cv2.destroyAllWindows()

```

Annex IV. Project alignment with the United Nations Sustainable Development Goals

There is certainly an alignment between this project and the United Nations Sustainable Development Goals (SDGs). The project is directly and indirectly aligned with several SDGs, however, it has been determined that the project is most directly aligned with the following SDGs:

- **Goal 8: "Promote sustained, inclusive and sustainable economic growth, full and productive employment and decent work for all".** With this project, physical businesses will be able to improve their business strategies based on data, therefore driving sustainable growth. Furthermore, this can be turned into a business which will potentially provide a decent job to many people directly, and even to more people indirectly.

- **Goal 9: "Build resilient infrastructure, promote inclusive and sustainable industrialization, and foster innovation".** This is an innovative project with the mission of providing affordable occupancy tracking technology to brick-and-mortar businesses. This will enable brick-and-mortar businesses to enhance their business decisions based on occupancy data and innovate in a world which is becoming increasingly data driven.

- **Goal 11: "Make cities and human settlements inclusive, safe, resilient, and sustainable".** As many cities become increasingly populated, this project will potentially play a key role on providing affordable technology to track the occupancy in different indoor spaces and potentially outdoors. This will provide not only businesses but also citizens with convenient occupancy information allowing them to use their time more efficiently, contributing to a sustainable growth and development of urban areas.