Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Sistema de Seguridad Basado en Depth Camera

Security System Based on Depth Camera

Javier Alonso Delgado

La Laguna, *7* de *junio* de 2021

Universidad de La Laguna	**Trabajo de Fin de Grado**
	Sistema de Seguridad Basado en Depth Camera
Fecha: *7 de junio de 2021*	**Autor**: Javier Alonso Delgado

D. **Eduardo Magdaleno Castelló**, con N.I.F. 43.824.397-J profesor Titular de Universidad adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como tutor.

D. **Manuel Jesús Rodríguez Valido**, con N.I.F. 52.840.833-N profesor Titular de Universidad adscrito al Departamento de Ingeniería Industrial de la Universidad de La Laguna, como cotutor.

CERTIFICAN

Que la presente memoria titulada:

"Sistema de Seguridad Basado en Depth Camera"

ha sido realizada bajo su dirección por D. **Javier Alonso Delgado**,
con N.I.F. 54.117.255-A.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de junio de 2021.

Universidad de La Laguna	**Trabajo de Fin de Grado**
	Sistema de Seguridad Basado en Depth Camera
Fecha: *7 de junio de 2021*	**Autor**: Javier Alonso Delgado

Agradecimientos

En primer lugar, quisiera agradecer tanto a mi tutor, Eduardo Magdaleno, como a mi cotutor, Manuel Jesús Rodríguez, por haberme guiado en la elaboración de este trabajo, con una amplia cercanía, flexibilidad y responsabilidad.

De igual manera, agradecer al profesorado y personal de la Escuela Superior de Ingeniería y Tecnología, por todas sus enseñanzas y facilidades que han puesto a lo largo de la carrera.

Por último, a mis compañeros de clase, con los cuales he recorrido este largo camino, con especial mención a Alejandro Pérez Moreno, Álvaro Fernández Rodríguez, David Ramallo Gracia, Samuel Fumero Hernández y Adrián Ruiz Olivero, con quienes he trabajado con más cercanía y entendimiento.

Licencia

Universidad de La Laguna	**Trabajo de Fin de Grado**
	Sistema de Seguridad Basado en Depth Camera
Fecha: *7 de junio de 2021*	**Autor**: Javier Alonso Delgado

Resumen

El objetivo del presente trabajo ha sido desarrollar un sistema de seguridad que permita monitorizar la posición de una persona en tiempo real.

Utilizando una cámara de profundidad y haciendo uso de librerías de procesamiento de imágenes tales como OpenCV, se propone definir un recinto seguro sobre el cual una persona podrá moverse libremente, estableciendo un sistema de seguimiento y alarma en caso de que saliera de la zona libre de posibles peligros.

Palabras clave: Sistema de seguridad, cámara de profundidad, monitorización, OpenCV, recinto seguro.

Abstract

The main objective of this work is the development of a security system which allows real time monitoring of people's location.

Using a depth camera as well as image processing libraries such as OpenCV, it is proposed to define a safe area where a person can move freely, establishing a tracking system with alarm alerts integrated to trigger in cases where he could go out of bounds of the risk free enclosure.

Keywords: security system, depth camera, monitoring, OpenCV, safe area.

Universidad de La Laguna	**Trabajo de Fin de Grado**
	Sistema de Seguridad Basado en Depth Camera
Fecha: *7 de junio de 2021*	**Autor**: Javier Alonso Delgado

Índice de Contenido

Índice de Figuras

Índice de Tablas

Lista de Abreviaturas

ABREVIATURA	**SIGNIFICADO**
VR	Realidad Virtual
CV	Computer Vision
SDK	Software Development Kit
KNN	K-Nearest Neighbors
MOG	Mixture of Gaussians
MIL	Multiple Instance Learning
KCF	Kernelized Correlation Filters
CSRT	Channel and Spatial Reliability Tracker
TLD	Tracking Learning Detection

1. Introducción

La realización de este trabajo de fin de grado se enmarca en el **Proyecto de Reconstrucción Histórica Virtual de San Cristóbal de La Laguna** [1], el cual está siendo desarrollado por un amplio colectivo vinculado a la Universidad de La Laguna.

En dicho proyecto, se pretende recrear de forma **virtual y fidedigna** los aspectos y rasgos históricos de dicha cuidad, de tal manera que podamos adentrarnos y conocer la arquitectura, vestimenta y demás ámbitos emblemáticos.

Debido a la gran magnitud del trabajo, existen numerosas tareas que se están desarrollando de forma paralela. Por un lado, a través de modelado 3D, se está realizando una **reconstrucción del diseño arquitectónico** de la ciudad reflejado en el Plano Torriani [2].

Por otra parte, mediante tecnologías de animación y modelado, se está **recreando la vestimenta y personajes de la época** siguiendo ejemplos de cuadros, fotografías u obras de teatro actuales ambientadas históricamente.

Con todo ello, se planea lograr un **sistema de realidad virtual** mediante el cual los usuarios puedan adentrarse y conocer la histórica ciudad de San Cristóbal de La Laguna de una forma **inmersiva y emocionante**.

Y es en esta fase del desarrollo donde tiene su aplicabilidad este trabajo de fin de grado. En el sistema final, el usuario portará unas **gafas VR dentro de un recinto acotado** por el cual se moverá visualizando la ciudad virtual.

Debido a la delimitación de espacio en la estancia, es clave contar con un **sistema de control** del área de realidad virtual, mediante el cual se logre la **monitorización y seguimiento** del usuario, de forma que se pueda mover por las zonas habilitadas de forma autónoma.

2. Estado del Arte

2.1. Proyecto de Reconstrucción Histórica

Una de las tareas dentro de este gran proyecto consiste en desarrollar un **subsistema para el control y seguridad del usuario**. Este debe informar a la persona cuando salga del área de confinamiento con el fin de que no corra riesgos a la hora de moverse físicamente por el recinto.

Como propuesta para localizar y avisar al mismo, se pretende diseñar un **sistema de alarmas** para informar al usuario de posibles riesgos tales como tropiezos o choques. Es en este ámbito donde se enmarca el presente trabajo de fin de grado.

2.2. Antecedentes

En lo que a proyectos de reconstrucción histórica se refiere, existen precedentes recientes de mucho éxito como el de la ciudad de Bolonia [3] o Barcelona [4].

Dichos proyectos han servido de referencia y guía para enmarcar la línea de trabajo que se está siguiendo.

2.3. Detección de Movimiento y Tracking

En el apartado que nos ocupa, existen numerosos mecanismos y algoritmos que permiten realizar la **detección y monitorización de objetos**, entre los que destacan el tracking basado en kernel [5] y el basado en contornos [6].

Desde editores de vídeo como *Adobe After Effects* hasta dispositivos de consolas tales como *Kinect*, son algunos ejemplos de aplicaciones de detección de movimiento y tracking.

Si bien el sistema propuesto en este trabajo no requiere una complejidad tan grande, se deberán valorar los algoritmos y métodos disponibles en base a las tecnologías que se disponen para el desarrollo, e implementar aquel más adecuado y que permita realizar el proceso de control de un recinto acotado de forma óptima.

2.4. Librería de Procesamiento de Imágenes

La librería de procesamiento de imágenes utilizada es OpenCV [7], una biblioteca libre de código abierto desarrollada originalmente por Intel.

Su licencia *open source* la ha dotado de una **gran popularidad** y es una de las más utilizadas en la actualidad, con una **comunidad muy extendida** que contribuye a su crecimiento.

Gracias a ello, cuenta con implementaciones de diferentes mecanismos de detección y tracking, los cuales se estudiarán en detalle en posteriores apartados.

2.5. Lenguaje de Programación

Se ha hecho uso del lenguaje de programación **Python**. Actualmente es uno de los lenguajes más utilizados y lidera muchos de los rankings de popularidad [8], gracias a su **versatilidad**, legibilidad y curva de aprendizaje baja.

En lo que a OpenCV se refiere, la librería *opencv-python* [9] ofrece un wrapper que nos permite instalar el framework sin tener que construir a partir del código fuente.

3. Objetivos

3.1. Objetivo General del Proyecto

Como objetivo de este TFG, se pretende **diseñar e implementar un sistema de seguridad** basado en una *Depth Camera* de Intel que permite **monitorizar la posición del usuario en tiempo real.**

3.2. Objetivos Específicos

3.2.1. Detección Simultánea de Objetos

Se pretende lograr **detectar múltiples objetos** en la escena de forma simultánea, tanto al usuario como a objetos extraños u otras personas que irrumpan en la zona.

3.2.2. Acotación de Zona Segura

Se propone **definir una zona acotada** que definirá el recinto seguro sobre el que el usuario portador de las gafas de realidad virtual se podrá mover sin peligro.

3.2.3. Sistema de Alarmas

Se establece un **sistema de avisos** cuando exista una posible situación de peligro. Se deberá **alertar al usuario** que se encuentre en el recinto, principalmente mediante sonido.

4. Desarrollo

4.1. Configuración del Entorno

Al formar parte de un proyecto en marcha, la tecnología a utilizar así como entorno de trabajo estaban **previamente estudiadas y definidas**. De esta manera, se requería el desarrollo del sistema en Windows con un IDE y modelo de cámara ya establecidos.

4.1.1. Entorno de Desarrollo

Se ha instalado el IDE PyCharm de la compañía *JetBrains*. Haciendo uso de la licencia de estudiante, se ha trabajado con la versión completa de la aplicación, aunque la versión gratuita *community* es igualmente compatible con el trabajo realizado.

4.1.2. Controladores de Cámara

Para poder trabajar con la cámara, se ha instalado el paquete de desarrollo *Intel RealSense SDK 2.0*, disponible en la página web principal de dichos modelos de cámara.

4.1.3. Instalación de Librerías

La instalación de librerías se ha realizado a través del gestor de dependencias *Pip.* Configurando en *PyCharm* un intérprete de Python con *Pipenv,* se ha generado un fichero *Pipfile* que contiene las dependencias necesarias para el funcionamiento de la aplicación. De esta manera, se agiliza y minimizan posibles errores en el proceso de instalación de librerías.

Se ha hecho uso de los siguientes paquetes:

- ***opencv-python***: librería de procesamiento de imágenes. En concreto, se ha utilizado el paquete completo *opencv-contrib-python* puesto que incluye más funcionalidades, tales como los trackers que veremos más adelante.
- ***pyrealsense2****:* wrapper del SDK de RealSense para trabajar con la cámara.
- ***numpy***: funciones matemáticas de alto nivel y manejo de arrays.

4.2. Estudio y Análisis de Algoritmos de Detección

Al trabajar en el ámbito de procesamiento de imágenes, es clave realizar un estudio amplio sobre cómo abordar el problema de la forma más eficiente posible.

Como bien se adelantó en capítulos anteriores, la comunidad de OpenCV es muy extensa y, realizando búsquedas en la red, no resulta complicado encontrar aplicaciones y tutoriales [10]-[17] sobre detección y tracking con esta herramienta.

Sin embargo, la mayor de las dificultades es lograr una **solución genérica y universal** que funcione bien siempre. Las **condiciones de luminosidad** o los movimientos debidos a la **inestabilidad de la cámara** son dos factores clave que influyen totalmente en el enfoque.

Con el propósito de diseñar una solución **flexible y adaptable a cambios**, se ha optado por implementar tres tipos de soluciones diferentes, cada una con sus ventajas y desventajas.

4.2.1. Diferencia de Imágenes

Una de las técnicas más extendidas y populares por su sencillez y rapidez es la **diferencia de imágenes**. A través de este método, se define un frame de fondo de la imagen y se compara con los que se obtienen posteriormente. Si surgen cambios, tales como la aparición de un nuevo objeto que antes no estaba, será detectado al realizar la resta entre imágenes.

Si bien es **muy rápido y simple** de implementar, tiene numerosos inconvenientes en lo que a precisión se refiere. Primordialmente, los cambios ligeros de luminosidad o pequeños movimientos de la cámara pueden provocar que se detecten **falsos cambios**.

4.2.1. Sustracción de Fondo

Debido a los inconvenientes que muestra la solución anterior, se buscó una alternativa que lidiase con la mayor parte de los problemas descritos.

En escenarios en los que la cámara se encuentra fija monitorizando una única zona, es inherente pensar en métodos de **sustracción de fondo**. Aunque la técnica vista en el apartado anterior sigue dicha filosofía, existen algoritmos mucho más avanzados y robustos. En lo que a OpenCV se refiere, cuenta con varias implementaciones de sustractores de fondo, si bien hay dos que destacan en uso por encima del resto.

El primero de ellos es el denominado **KNN**, un algoritmo de sustracción de fondo basado en búsqueda de vecinos próximos [18]. Este sustractor es de los más eficientes si se cuenta con un número bajo de píxeles del primer plano.

Por otro lado, se encuentra **MOG2**, el cual utiliza la técnica de sustracción de fondo por mezcla de gaussianas [19]. Esta implementación provee una mejor adaptabilidad a ligeros cambios de escena por **variaciones de luminosidad** y factores similares. Debido a esta ventaja, ha sido el sustractor de fondo escogido.

4.2.2. Trackers

Las dos técnicas escogidas hasta el momento (diferencia y sustracción) siguen un método basado en detección. En ellas se establece un modelo de fondo para posteriormente realizar una sustracción entre frames, obteniendo finalmente una máscara del primer plano (*Fig. 1*).

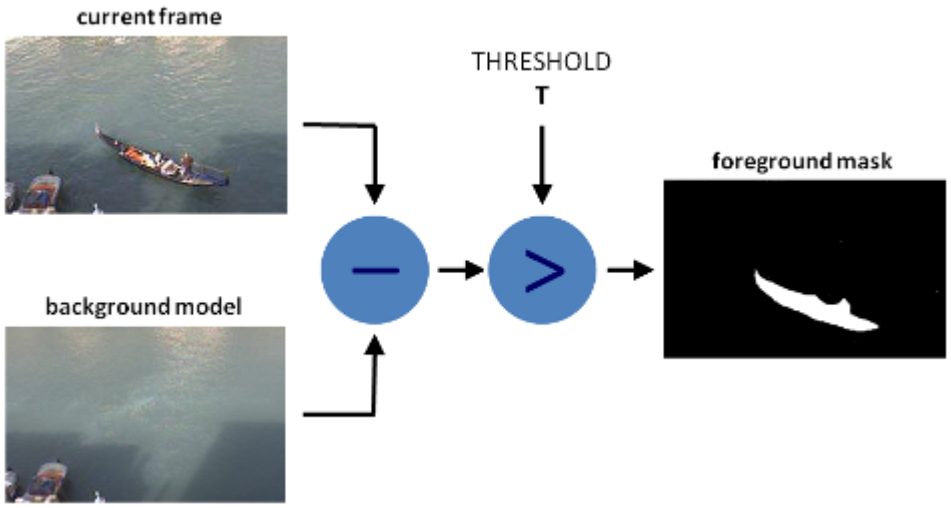

Figura 1. Diagrama de sustracción de fondo.
Fuente: OpenCV Tutorials Documentation

No obstante, existe otro tipo de metodología distinta: el **tracking o seguimiento**. Si bien la detección de objetos trata de escanear y buscar un objeto en una secuencia de imágenes, el tracking consiste en **vigilar y seguir un objeto ya seleccionado**.

Por lo general, esta técnica es más costosa computacionalmente, aunque mucho más **precisa y fiable**.

OpenCV provee hasta **siete tipos de trackers**. Cada uno de ellos posee sus propias características que le otorgan ventaja frente a otros según el ámbito de aplicación. En la Tabla 1 se puede ver un resumen de cada uno de ellos.

Nombre	Ventajas	Inconvenientes
Boosting	Algoritmo sencillo	Antiguo, casi en desuso.
MIL	Aprendizaje múltiple	Apenas reporta errores
KCF	Bastante robusto	No maneja bien oclusiones
CSRT	Alta precisión	Moderadamente lento
MedianFlow	Buen reporte de errores	Falla si hay saltos bruscos
TLD	Equilibrio rapidez/precisión	Muchos falsos positivos
MOSSE	Increíblemente rápido	Precisión

Tabla 1. Comparativa de Trackers disponibles en OpenCV

Teniendo en cuenta esta información, se ha elegido **CSRT**, un tracker basado en el algoritmo propuesto en [20] y uno de los más precisos disponible. Si bien puede afectar al rendimiento, las dos propuestas de solución anteriores son bastante rápidas, por lo que se quiere proveer de una alternativa **basada en la precisión** en lugar de la rapidez.

4.3. Codificación e Implementación del Sistema

4.3.1. Detección de Movimiento Mediante Diferencia de Imágenes

Para la implementación de este algoritmo, es clave trabajar con **imágenes en escala de grises.**

En un primer lugar, se selecciona el frame de fondo deseado (tecla *ESPACIO*) y se define como **imagen de referencia** en escala de grises. Posteriormente, se obtienen nuevos frames en los que el **usuario entrará al recinto**.

Realizando una diferencia de imágenes entre el frame de referencia y el actual, se obtiene una imagen que contendrá cualquier variación respecto a la escena *(Fig. 2)*.

Con dicho resultado, es clave realizar operaciones de **difuminado** para limpiar brusquedades y facilitar la comparación de frames. A continuación, se aplica un umbral (*threshold*) con el que se obtendrá la **máscara de detección** *(Fig. 3)*.

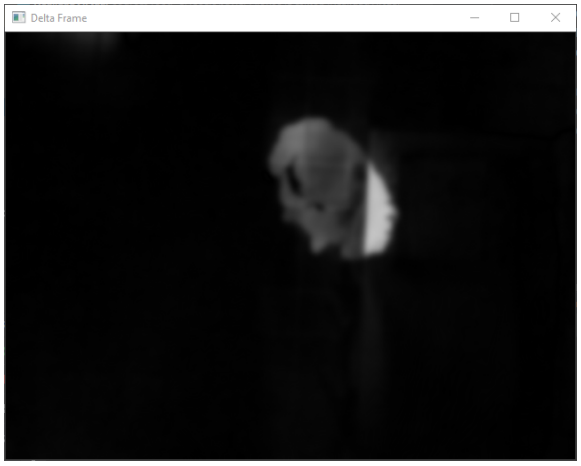

Figura 2. Frame obtenido de la diferencia

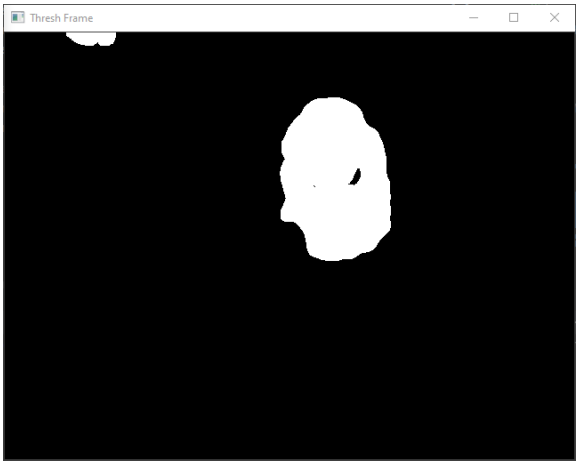

Figura 3. Máscara tras aplicar umbral

Haciendo uso de la máscara umbral, se obtienen los **bordes** de las figuras detectadas, filtrando aquellos que cuya área sea bastante reducida. Así, se traza un **rectángulo** que contiene dichos bordes y que será utilizado para monitorizar los objetos detectados (*Fig. 4)*.

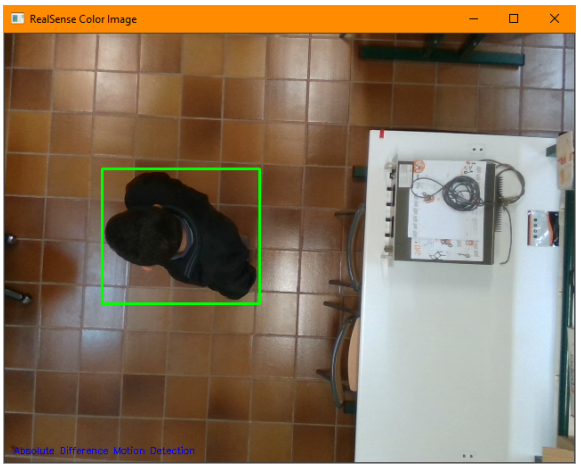

Figura 4. Detección mediante diferencia de imágenes

4.3.2. Detección de Movimiento Mediante Sustracción de Fondo

En primer lugar se configura un objeto de sustracción de fondo de tipo MOG2.

Al inicializar el programa, se deberán dejar unos segundos de captura del frame de fondo para que el algoritmo **aprenda el modelo**. Una vez hecho esto, se habrá generado una **máscara** que será utilizada para sustraer el fondo de los nuevos frames capturados *(Fig. 5)*.

Para mejorar la precisión y evitar **falsos positivos**, se aplican **transformaciones morfológicas** de apertura y cierre, utilizando una elipse como elemento estructural del kernel *(Fig. 6)*.

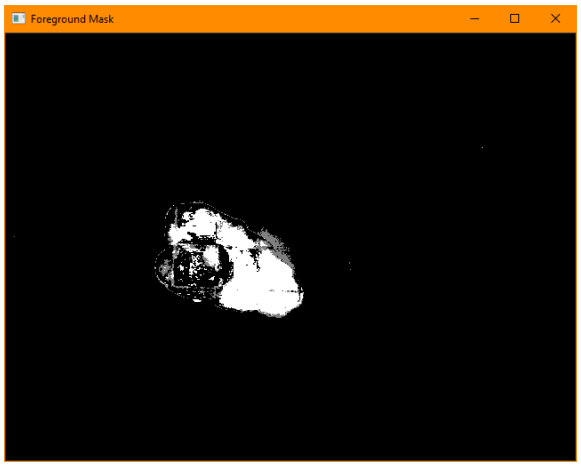

Figura 5. Máscara de sustracción de fondo

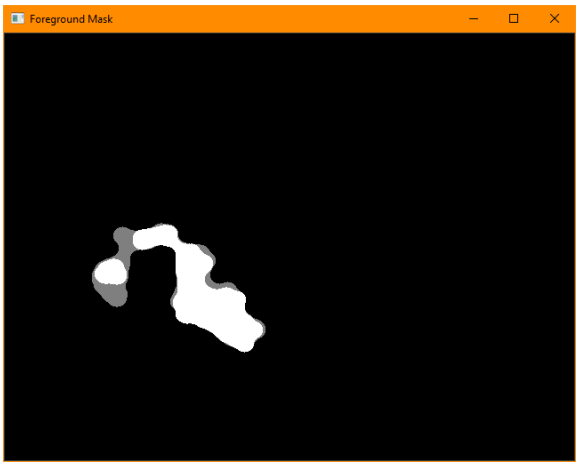

Figura 6. Máscara tras op. morfológicas

Una vez obtenida la máscara, se sigue el mismo procedimiento del algoritmo anterior para obtener los bordes y dibujar el rectángulo de contorno *(Fig. 7)*.

Figura 7. Detección mediante sustracción de fondo

4.3.3. Tracking de Movimiento Mediante CSRT

Mientras que los algoritmos de detección se basan en máscaras y aprendizaje del fondo, en este caso es indiferente iniciar el sistema con el usuario dentro o fuera del recinto.

Al ejecutar el programa, se puede cambiar el objeto a monitorizar en cualquier momento. Presionando la tecla *S*, se abrirá una ventana en la que, a través del ratón, **seleccionaremos el objeto al que hacer seguimiento** *(Fig. 8)***.**

Al confirmar la selección pulsando *ENTER*, el tracker inicializa las coordenadas del *bounding box* que envuelve al objetivo y las **actualiza automáticamente** según su movimiento.

Estas coordenadas se utilizan para dibujar dicho rectángulo en el frame *(Fig. 9)*.

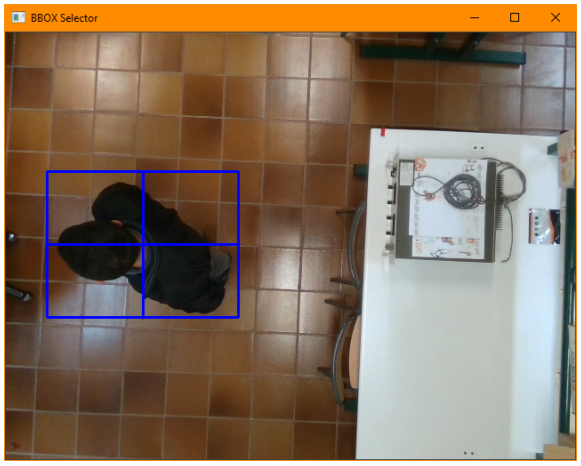

Figura 8. Selección de objeto de interés

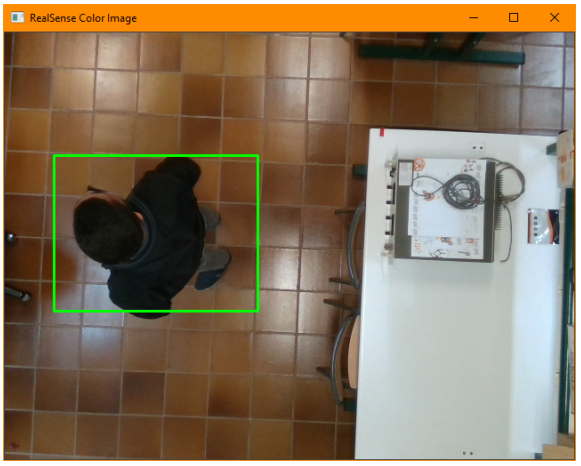

Figura 9. Bounding box de seguimiento

4.3.4. Acotación de Zona Segura

La acotación de la zona segura se ha implementado de la forma más simple pero flexible posible. Al tratarse de una cámara fija, se han definido las coordenadas de la zona segura como **parámetros**, en base a la posición de la esquina superior izquierda, ancho y alto, siguiendo la línea de coherencia con los *bounding boxes* de apartados anteriores.

De esta forma, si se sitúa la cámara en otra ubicación, o cambian las dimensiones de la zona acotada, basta con modificar los valores antes mencionados.

A la hora de detectar **situaciones de peligro**, como objetos extraños que se acerquen al recinto, o posibles tropiezos del usuario si se saliera de dicha zona, se ha optado por la solución más rápida y precisa computacionalmente.

Contando con las coordenadas previamente definidas de la zona segura, y las de los objetos detectados, se puede realizar una comprobación entre los valores de alto, ancho y punto superior izquierdo de la zona segura y los *bounding boxes*.

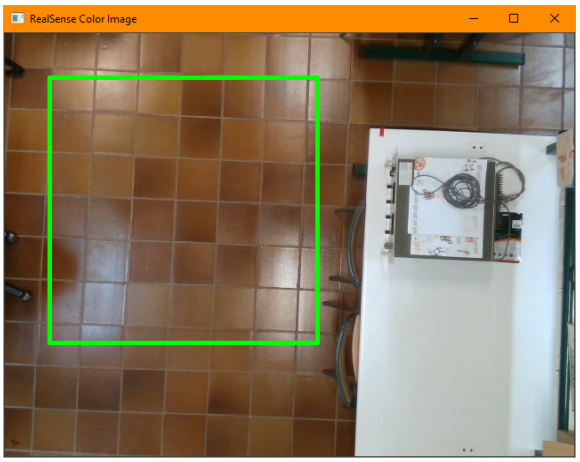

Figura 10. Zona segura

Cualquier rectángulo detectado cuyas coordenadas sean superiores a las del recinto acotado (es decir, fuera de la zona segura), será motivo de una **situación de alarma**, bien porque es un objeto extraño en la escena y no se encuentra en la zona segura, o bien porque el usuario está en el **borde del recinto de control**.

4.3.4. Sistema de Alarmas

Para el sistema de alarmas, se ha realizado un enfoque flexible de cara tanto al usuario dentro del recinto como al supervisor de la monitorización.

En primer lugar, se muestra un **texto** en el frame de la imagen que indica si se ha detectado peligro o no. Dicho mensaje apenas consume recursos y es muy útil para labores de pruebas de cámara o depuración.

Por otra parte, se ha implementado un aviso de alarma mediante **sonido**. A través de la librería *Winsound*, se produce un pitido breve cuando se detecta una situación de alarma.

Este proceso ralentiza bastante la aplicación, por lo únicamente se ha añadido al método de diferencia de imágenes, al ser el más rápido de las tres propuestas. De igual forma, se han utilizado **hilos** para reducir tales problemas de rendimiento.

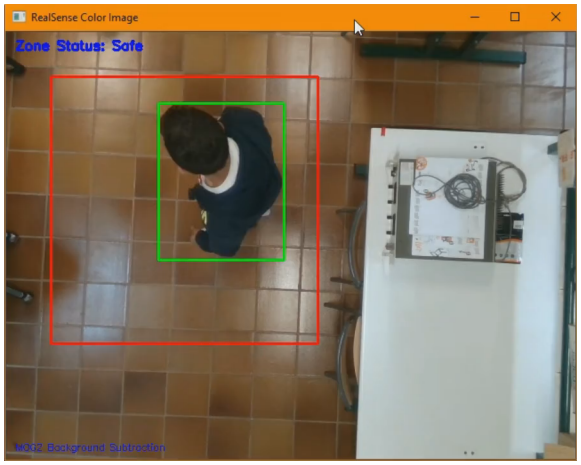

Figura 11. Situación segura

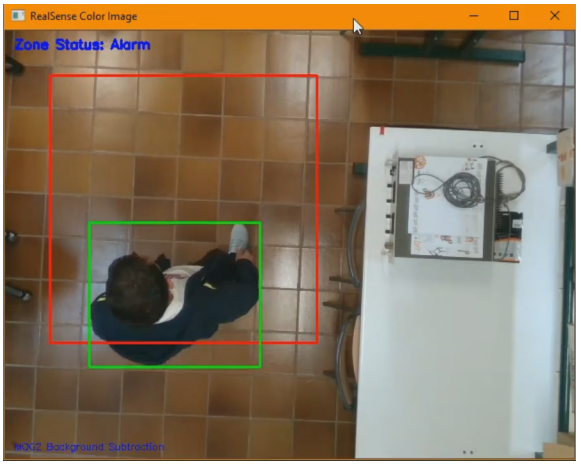

Figura 12. Situación de alarma

5. Conclusiones y Líneas Futuras

5.1. Conclusiones

En este trabajo de final de grado, se ha abordado el problema de detección, control y monitorización de personas y objetos en un recinto acotado. Formando parte del Proyecto de Reconstrucción Histórica Virtual de San Cristóbal de La Laguna, es clara la importancia y la calidad que debe tener el subsistema a desarrollar.

Es por ello que se ha realizado un estudio detallado de tecnologías y herramientas existentes en el ámbito, con el principal objetivo de lograr la solución más adecuada y contando con argumentos que la justifiquen.

Siguiendo este enfoque, se han implementado tres tipos de soluciones complementarias, documentando las ventajas y desventajas que posee cada una de ellas según el escenario en que se aplique.

En conclusión, se ha contribuido con un sistema flexible y completo, facilitando futuros desarrollos sobre una base sólida de trabajo.

5.2. Líneas Futuras

Dentro del proyecto de reconstrucción virtual, se han establecido los futuros pasos a seguir con este subsistema.

El trabajo más importante consta de conectar el subsistema desarrollado a las gafas VR mediante *bluetooth*, de forma que las alertas sonoras las reciba el usuario desde las propias gafas.

De igual forma, se pueden incluir mejoras que amplíen el abanico de funcionalidades de los programas implementados, tales como el uso de los frames de profundidad que obtiene la cámara para mostrar la distancia a la que están los objetos detectados, por ejemplo.

6. Conclusions and Future Work

6.1. Conclusions

In this thesis, we addressed the problem of detection, control and monitoring of people and objects inside an enclosed area. Being part of the *Virtual Historical Reconstruction of San Cristobal de La Laguna Project*, it is straightforward the importance and quality that the subsystem should have.

Due to that fact, a discussion on different existing technologies and tools has been provided, with the main focus of accomplishing the most suitable solution and having arguments to justify it.

Following this approach, three different and complementary solutions have been developed, documenting the advantages and disadvantages of each one depending on the scenario.

In conclusion, we have contributed with a flexible and complete solution, establishing a solid foundation on which to settle further developments.

6.2. Future Work

Inside the virtual reconstruction project, there is much work left for the future.

The most important consists in connecting the developed subsystem to the VR headset via bluetooth, allowing the sound alerts to be received through the own headset.

By the same token, there are so many potential upgrades which could increase the variety of features of the implemented programs, such as using the depth frames provided by the camera in order to show the distance to the detected objects, for example.

7. Presupuesto

En la Tabla 2 se presenta una estimación sobre el coste de la realización de este proyecto. La mano de obra incluye tanto las horas de investigación como de desarrollo.

Concepto	Tipo	Coste/Unidad	Unidades	Subtotal
Intel RealSense D435	Hardware	155€	1	155€
Computadora	Hardware	900€	1	900€
IDE PyCharm	Software	Gratuito	1	0€
Mano de Obra	RRHH	15€/h	200 horas	3000€
			TOTAL	4055€

Tabla 2. Desglose del presupuesto

A. Apéndices

Todo el código desarrollado se encuentra disponible en el siguiente repositorio:

> Javier Alonso Delgado, *Depth Camera Security System*, (2021), Repositorio GitHub, https://github.com/alu0101109251/DepthCameraSecuritySystem

A.1. Detección de Movimiento por Diferencia

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""Motion Detection using Image Difference."""

import cv2
import numpy as np
import pyrealsense2 as rs
import winsound
import threading

__author__ = "Javier Alonso Delgado"
__license__ = "CC-BY-SA-4.0"
__version__ = "1.0"
__email__ = "alu0101109251@ull.edu.es"

# Constants
AREA_THRESHOLD = 10000
RED = (0, 0, 255)
BLUE = (255, 0, 0)
GREEN = (0, 255, 0)

# Safe Zone Coordinates
(szx, szy, szw, szh) = (50, 50, 300, 300)


# Checks is rectangle is out of safe zone
def is_out_of_bounds(px, py, weight, height):
    if px <= szx or py <= szy or (px + weight) >= (szx + szw) or (py + height) >= (szy + szh):
        return True
    return False


# Makes beep sound using Windows API
def beep():
    winsound.Beep(frequency=2500, duration=750)


# Intel RealSense Camera Pipeline Configuration
pipeline = rs.pipeline(ctx=rs.context())
config = rs.config()
```

```
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
pipeline.start(config)

# Initialize first frame
firstFrame = None
alarmList = [None, None]

# Press SPACE to select the background frame
while cv2.waitKey(40) != ord(' '):
    frames = pipeline.wait_for_frames()
    firstFrame = np.asanyarray(frames.get_color_frame().get_data())
    cv2.imshow("Background Selection", firstFrame)

firstFrame = cv2.cvtColor(firstFrame, cv2.COLOR_BGR2GRAY)
firstFrame = cv2.GaussianBlur(firstFrame, (21, 21), 0)
cv2.destroyWindow("Background Selection")

try:
    while True:

        # Wait for coherent frames and grab color frame
        frames = pipeline.wait_for_frames()
        colorFrame = frames.get_color_frame()

        # Check if a frame was successfully received
        if not colorFrame:
            continue

        # Convert image to numpy arrays
        colorFrame = np.asanyarray(colorFrame.get_data())

        # Convert it to grayscale and blur it
        grayFrame = cv2.cvtColor(colorFrame, cv2.COLOR_BGR2GRAY)
        grayFrame = cv2.GaussianBlur(grayFrame, (21, 21), 0)
        # gray = cv2.normalize(gray, gray, 0, 255, cv2.NORM_MINMAX)

        # Draw safe zone and set initial status text
        alarm = 0
        text = "Safe"
        safeZone = cv2.rectangle(colorFrame, (szx, szy), (szx + szw, szy + szh), RED, 2, 1)

        # Compute the absolute difference between current frame and first frame
        deltaFrame = cv2.absdiff(firstFrame, grayFrame)

        # Threshold the image and clean up
        threshFrame = cv2.threshold(deltaFrame, 25, 255, cv2.THRESH_BINARY)[1]
        # thresh =
cv2.adaptiveThreshold(deltaFrame,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,cv2.THRESH_BINARY,11,2)
        threshFrame = cv2.dilate(threshFrame, None, iterations=2)

        # Grab and filter contours.
        contours, _ = cv2.findContours(threshFrame, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        for c in contours:
```

```
            # Calculate area and remove small elements
            if cv2.contourArea(c) < AREA_THRESHOLD:
                continue

            # Compute the bounding box for the contour and draw it on the frame
            (x, y, w, h) = cv2.boundingRect(c)
            cv2.rectangle(colorFrame, (x, y), (x + w, y + h), GREEN, 2, 1)

            # Check if the bounding box is out of the safe zone
            if is_out_of_bounds(x, y, w, h):
                text = "Alarm"
                alarm = 1

        # If there is a 'no alarm' to 'alarm' situation, trigger sound
        alarmList.append(alarm)
        if alarmList[-1] == 1 and alarmList[-2] == 0:
            threading.Thread(target=beep()).start()

        # Draw status text and detection technique in the frame
        cv2.putText(colorFrame, "Zone Status: {}".format(text), (10, 20), cv2.FONT_HERSHEY_SIMPLEX,
0.5, BLUE, 2)
        cv2.putText(colorFrame, "Absolute Difference Motion Detection", (10, colorFrame.shape[0] -
10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.35, BLUE, 1)

        # Show images
        cv2.imshow('RealSense Color Image', colorFrame)
        # cv2.imshow("First Frame", firstFrame)
        # cv2.imshow("Gray Frame", grayFrame)
        # cv2.imshow("Delta Frame", deltaFrame)
        # cv2.imshow("Thresh Frame", threshFrame)

        # Record if the user presses a key
        key = cv2.waitKey(1) & 0xFF

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

finally:

    # Cleanup the camera and close any open windows
    pipeline.stop()
    cv2.destroyAllWindows()
```

A.2. Detección de Movimiento por Sustracción

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""Motion Detection using MOG2 Background Subtraction."""

import cv2
import numpy as np
import pyrealsense2 as rs

__author__ = "Javier Alonso Delgado"
__license__ = "CC-BY-SA-4.0"
__version__ = "1.0"
__email__ = "alu0101109251@ull.edu.es"

# Constants
AREA_THRESHOLD = 5000
RED = (0, 0, 255)
BLUE = (255, 0, 0)
GREEN = (0, 255, 0)

# Safe Zone Coordinates
(szx, szy, szw, szh) = (50, 50, 300, 300)


# Check is rectangle is out of safe zone
def is_out_of_bounds(px, py, weight, height):
    if px <= szx or py <= szy or (px + weight) >= (szx + szw) or (py + height) >= (szy + szh):
        return True
    return False


# Intel RealSense Camera Pipeline Configuration
pipeline = rs.pipeline(ctx=rs.context())
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
pipeline.start(config)

# Initialize Background Subtractor
backgroundSubtractor = cv2.createBackgroundSubtractorMOG2(history=500, varThreshold=25,
detectShadows=True)

# Let subtractor learn the background model and press SPACE
while cv2.waitKey(40) != ord(' '):
    frames = pipeline.wait_for_frames()
    tempFrame = np.asanyarray(frames.get_color_frame().get_data())
    cv2.imshow("Background Detection", tempFrame)
    backgroundSubtractor.apply(tempFrame, 0.5)

cv2.destroyWindow("Background Detection")
```

```
try:
    while True:

        # Wait for coherent frames and grab color frame
        frames = pipeline.wait_for_frames()
        colorFrame = frames.get_color_frame()

        # Check if a frame was successfully received
        if not colorFrame:
            continue

        # Convert image to numpy arrays
        colorFrame = np.asanyarray(colorFrame.get_data())

        # Draw safe zone and set initial status text
        text = "Safe"
        safeZone = cv2.rectangle(colorFrame, (szx, szy), (szx + szw, szy + szh), RED, 2, 1)

        # Create Foreground Mask
        mask = None
        mask = backgroundSubtractor.apply(colorFrame, mask, 0.0)

        # Apply morphological operations to clean up the mask
        mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(21, 21)))
        mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, cv2.getStructuringElement(cv2.MORPH_ELLIPSE,
(21, 21)))
        # _, mask = cv2.threshold(mask, 25, 255, cv2.THRESH_BINARY)

        # Grab and filter contours.
        contours, _ = cv2.findContours(mask, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        for c in contours:
            # Calculate area and remove small elements
            if cv2.contourArea(c) < AREA_THRESHOLD:
                continue

            # Compute the bounding box for the contour and draw it on the frame
            (x, y, w, h) = cv2.boundingRect(c)
            cv2.rectangle(colorFrame, (x, y), (x + w, y + h), GREEN, 2, 1)

            # Check if the bounding box is out of the safe zone
            if is_out_of_bounds(x, y, w, h):
                text = "Alarm"

        # Draw status text and detection technique in the frame
        cv2.putText(colorFrame, "Zone Status: {}".format(text), (10, 20), cv2.FONT_HERSHEY_SIMPLEX,
0.5, BLUE, 2)
        cv2.putText(colorFrame, "MOG2 Background Subtraction", (10, colorFrame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX,
                    0.35, BLUE, 1)

        # Show images
        cv2.imshow('RealSense Color Image', colorFrame)
        # cv2.imshow('Foreground Mask', mask)
```

```
        # Record if the user presses a key
        key = cv2.waitKey(1) & 0xFF

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

finally:

    # Cleanup the camera and close any open windows
    pipeline.stop()
    cv2.destroyAllWindows()
```

A.3. Tracking de Movimiento

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
"""Motion Tracking using CSRT Tracker."""

import pyrealsense2 as rs
import numpy as np
import cv2

__author__ = "Javier Alonso Delgado"
__license__ = "CC-BY-SA-4.0"
__version__ = "1.0"
__email__ = "alu0101109251@ull.edu.es"

# CONSTANTS
RED = (0, 0, 255)
BLUE = (255, 0, 0)
GREEN = (0, 255, 0)

# Safe Zone Coordinates
(szx, szy, szw, szh) = (50, 50, 300, 300)

# Check is rectangle is out of safe zone
def is_out_of_bounds(px, py, weight, height):
    if px <= szx or py <= szy or (px + weight) >= (szx + szw) or (py + height) >= (szy + szh):
        return True
    return False

# Intel RealSense Camera Pipeline Configuration
pipeline = rs.pipeline(ctx=rs.context())
config = rs.config()
config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
pipeline.start(config)

# Tracker
tracker = cv2.TrackerCSRT_create()
bbox = None

try:
    while True:

        # Wait for coherent frames and grab color frame
        frames = pipeline.wait_for_frames()
        colorFrame = frames.get_color_frame()

        # Check if a frame was successfully received
        if not colorFrame:
            continue

        # Convert images to numpy arrays
        colorFrame = np.asanyarray(colorFrame.get_data())
```

```
        # Draw safe zone and set initial status text
        text = "Safe"
        safeZone = cv2.rectangle(colorFrame, (szx, szy), (szx + szw, szy + szh), RED, 2, 1)

        # Check if tracker is initialized
        if bbox is not None:
            # Update tracker
            success, bbox = tracker.update(colorFrame)

            # Tracking Success
            if success:
                # Grab the bounding box and draw it on the frame
                (x, y, w, h) = [int(p) for p in bbox]
                cv2.rectangle(colorFrame, (x, y), (x + w, y + h), GREEN, 2, 1)

                # Check if the bounding box is out of the safe zone
                if is_out_of_bounds(x, y, w, h):
                    text = "Alarm"

            # Tracking failure
            else:
                cv2.putText(colorFrame, "Tracking failure!", (100, 80), cv2.FONT_HERSHEY_SIMPLEX,
0.5, RED, 2)

        # Draw status text and detection technique in the frame
        cv2.putText(colorFrame, "Zone Status: {}".format(text), (10, 20), cv2.FONT_HERSHEY_SIMPLEX,
0.5, BLUE, 2)
        cv2.putText(colorFrame, "CSRT Tracker", (10, colorFrame.shape[0] - 10),
cv2.FONT_HERSHEY_SIMPLEX, 0.35, BLUE, 1)

        # Show images
        cv2.imshow('RealSense Color Image', colorFrame)

        # Record if the user presses a key
        key = cv2.waitKey(1) & 0xFF

        # Press 's' to select ROI to track.
        # Make selection using the mouse and press ENTER or SPACE
        if key == ord("s"):
            bbox = cv2.selectROI("BBOX Selector", colorFrame, fromCenter=False)
            tracker.init(colorFrame, bbox)
            cv2.destroyWindow("BBOX Selector")

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

finally:

    # cleanup the camera and close any open windows
    pipeline.stop()
    cv2.destroyAllWindows()
```

Referencias

[1] Universidad de La Laguna, *Reconstrucción Histórica Virtual de San Cristóbal de La Laguna,* Portal de Investigación. Visitado el: Mayo 18, 2021. [Online].
Disponible: https://portalciencia.ull.es/proyectos/41561/detalle

[2] Ayuntamiento de La Laguna, *El Plano Torriani*, Turismo de La Laguna. Visitado el: Mayo 18, 2021. [Online]. Disponible: https://turismo.aytolalaguna.es/el-plano-torriani/

[3] Gabrielle Bitelli, Giorgia Gatta, *Georeferencing of an XVIII century technical map of Bologna (Italy),* e-Perimetron, 2012. Visitado el: Marzo 10, 2021. [Online].
Disponible: http://www.e-perimetron.org/vol_7_4/bitelli_gatta.pdf

[4] Mar Santamaria-Varas, Pablo Martinez-Diez, *The Historic Charter of Barcelona (CHB)*, e-Perimetron, 2019. Visitado el: Marzo 10, 2021. [Online].
Disponible: http://www.e-perimetron.org/Vol_14_1/Santamaria-Varas_Martinez-Diez.pdf

[5] D. Comaniciu, V. Ramesh and P. Meer, *Kernel-based object tracking*, in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 25, no. 5, pp. 564-577, May 2003, doi: 10.1109/TPAMI.2003.1195991. [Online].
Disponible: https://ieeexplore.ieee.org/document/1195991

[6] A. Techmer, *Contour-based motion estimation and object tracking for real-time applications*, Proceedings 2001 International Conference on Image Processing (Cat. No.01CH37205), 2001, pp. 648-651 vol.3, doi: 10.1109/ICIP.2001.958202. [Online].
Disponible: https://ieeexplore.ieee.org/document/958202

[7] OpenCV, *Página Web Principal,* OpenCV Team, 2021. Visitado el: Marzo 10, 2021. [Online]. Disponible: https://opencv.org/

[8] Pierre Carbonnelle, *PYPL Popularity of Programming Language,* PYPL, 2021. Visitado el: Marzo 23, 2021. [Online]. Disponible: https://pypl.github.io/PYPL.html

[9] Python Software Foundation, *opencv-python package*, PyPi, 2021. Visitado el: Marzo 10, 2021. [Online]. Disponible: https://pypi.org/project/opencv-python/

[10] Adrian Rosebrock, *Basic motion detection and tracking with Python and OpenCV*, PyImageSearch, Mayo 25, 2015. Visitado el: Marzo 9, 2021. [Online]. Disponible: https://www.asme.org/engineering-topics/articles/renewable-energy/catching-the-sun

[11] Adrian Rosebrock, *OpenCV Object Tracking*, PyImageSearch, Julio 30, 2018. Visitado el: Marzo 9, 2021. [Online].
Disponible: https://www.pyimagesearch.com/2018/07/30/opencv-object-tracking/

[12] Adrian Rosebrock, *Object detection with deep learning and OpenCV*, PyImageSearch, Septiembre 11, 2017. Visitado el: Marzo 9, 2021. [Online]. Disponible: https://www.pyimagesearch.com/2017/09/11/object-detection-with-deep-learning-and-opencv/

[13] Prateek Khandelwal, *How to do intrusion detection?,* Medium, Abril 23, 2020. Visitado el: Marzo 12, 2021. [Online]. Disponible:
https://medium.com/@khandelwalprateek01/how-to-do-intrusion-detection-5e67df60802

[14] Arindom Bhattacharjee, *Build A Motion Detected Alarm System With Python*, Towards Data Science, Mayo 25, 2020. Visitado el: Marzo 12, 2021. [Online]. Disponible: https://towardsdatascience.com/build-a-motion-triggered-alarm-in-5-minutes-342fbe3d5396

[15] Sergio Canu, *Object Tracking with OpenCV and Python*, PySource, Enero 28, 2021. Visitado el: Marzo 15, 2021. [Online].
Disponible: https://pysource.com/2021/01/28/object-tracking-with-opencv-and-python/

[16] Automatic Addison, *Motion Detection Using OpenCV on Raspberry Pi 4*, Septiembre 5, 2020. Visitado el: Marzo 15, 2021. [Online]. Disponible:
https://automaticaddison.com/motion-detection-using-opencv-on-raspberry-pi-4/

[17] Satya Mallick, *Object Tracking Using OpenCV*, Learn OpenCV, Febrero 13, 2017. Visitado el: Marzo 15, 2021. [Online].
Disponible: https://learnopencv.com/object-tracking-using-opencv-cpp-python/

[18] Z. Zivkovic and Ferdinand van der Heijden, *Efficient adaptive density estimation per image pixel for the task of background subtraction*, Pattern Recognition Letters, 2006. ISSN 2006, pp. 773-780 Vol.27, ISSN 0167-8655. [Online].
Disponible: https://www.sciencedirect.com/science/article/abs/pii/S0167865505003521

[19] Z. Zivkovic, *Improved adaptive Gaussian mixture model for background subtraction*, Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004, pp. 28-31 Vol.2, doi: 10.1109/ICPR.2004.1333992. [Online].
Disponible: https://ieeexplore.ieee.org/document/1333992

[20] Alan Lukezic, Tom'as Voj'ir, Luka Cehovin Zajc, Jir'i Matas, and Matej Kristan. *Discriminative correlation filter tracker with channel and spatial reliability,* International Journal of Computer Vision, 2018. [Online].
Disponible: https://arxiv.org/abs/1611.08461