Universidad de
La Sabana

SEMINARIO DE INVESTIGACIÓN FORMATIVA

JHON STEWAR RAYO MOSQUERA

PROFESOR

JUAN PABLO GARZÓN RUIZ

Chía, Noviembre de 2019

Contenido

Requerimientos funcionales

Diseño

Casos de prueba

Resultados

Conclusiones

Referencias

Pregunta de Investigación

Cómo diseñar un sistema eficiente y escalable de vigilancia automatizada cuyo fin sea detectar y rastrear actividades sospechosas.

Objetivos

Proponer un método computacional óptimo para detectar y rastrear actividades sospechosas.

Específicos

- Analizar numerosos algoritmos para análisis de video.
- Analizar las tecnologías más recientes para procesamiento de video.
- Realizar una demostración que satisfaga los requerimientos.

Cronograma

En el siguiente diagrama se presentan las tareas a realizar para el correcto desarrollo del proyecto, así como el tiempo y las fechas estimadas de ultimación de cada actividad.

Actividades\Semanas	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Formulación problema de investigación														
Revisión bibliográfica														
Revisión del estado del arte														
Planteamientos objetivos														
Ponderación de las tecnologías a usar														
Diseño de la solución														
Programación de los algoritmos														
Pruebas														
Análisis resultados														
Redacción y presentación final														

Fig. 1. Cronograma. Fuente: Elaboración propia.

Estado del arte

El sistema que se busca diseñar supone el uso de procesos algorítmicos que logren detectar y seguir objetos en metrajes. En ambos escenarios, una imagen extraída del metraje funciona como entrada para los algoritmos, y se espera como salida la posición del objeto que se ha detectado o se está rastreando. En cuestión, el objeto al que se refiere en esta aplicación, son personas, a las cuales se les quiere vigilar con el fin de detectar alguna actividad sospechosa.

1. Algoritmos

En primer lugar, se presentan varios algoritmos para la detección de objetos en imágenes, se discute su funcionamiento y su viabilidad en el proyecto.

1.1. Red neuronal artificial

Una red neuronal artificial son sistemas computarizados que simulan en cierta medida las redes neuronales biológicas que componen los cerebros. Estos sistemas aprenden a resolver problemas de clasificación, identificación, entre otros, mediante la consideración de muestras (Nielsen, 2019). Su funcionamiento está basado en el uso de unas capas y unos nodos que representan una 'neurona'. Estas neuronas pueden transmitir información unas a otras, y así influenciar otras capas. Estos sistemas se ajustan a medida que van siendo entrenados con información, por lo que pueden volverse cada vez más precisos.

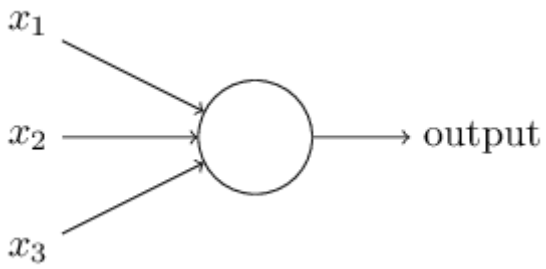

Fig. 2. Neurona artificial. Fuente: Nielsen.

Más detalladamente, una neurona artificial, véase Fig. 2, consiste de unas entradas, x_1, x_2,...,así como ciertos pesos asociados, w_1, w_2, …,que determinan la importancia de las entradas con respecto a la salida. También, se considera un valor de sesgo general o bias, b.
Así pues, la salida está determinada por la función (1).

$$y = (w_1 \cdot x_1) + (w_2 \cdot x_2) + \ldots + (w_n \cdot x_n) + b \qquad (1)$$

Sin embargo, el rango de la función (1) no es muy conveniente, por lo que con el fin de limitar los valores posibles de la salida, se pasa a través de una función cuyo rango sea [0,1]. Un ejemplo, es la función sigmoide (2).

$$\sigma(y) = \frac{1}{1 + e^{-y}} \qquad (2)$$

La ventaja de la función sigmoide (2) es que el resultado varía poco con respecto a pequeños cambios en las entradas, tal como se aprecia en la gráfica de la función, véase Fig. 3.

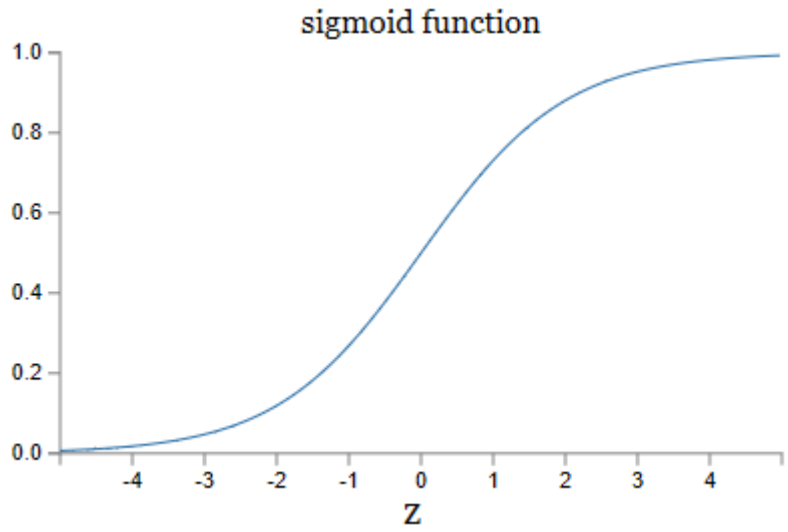

Fig. 3. Función sigmoide. Fuente: Nielsen.

Ahora bien, las neuronas pueden constituir capas dependiendo de su función dentro de la red neuronal. Hay tres tipos de capas a considerar.

La capa de entrada (Input layer) constituye las neuronas que reciben los datos iniciales, por ejemplo, las neuronas pertenecientes a esta capa podrían almacenar un valor de 0 a 1 que representa los valores de los píxeles de una imagen (Nielsen, 2019).

La capa de salida (Output layer) contiene la neurona o neuronas que almacenan el resultado. Por ejemplo, si se quisiera identificar cierto objeto en una imagen, la neurona de salida podría ser un número de 0 a 1 que indicará que tan probable es que el objeto en la imagen sea el objeto a identificar (Nielsen, 2019).

Las capas escondidas (Hidden layers) son las que determinan ciertos ‘filtros’ con el fin de obtener el resultado esperado. En otros términos, son las que definen en mayor medida el comportamiento de la red neuronal para la entrada de datos (Nielsen, 2019).

Es importante notar que una neurona de una capa supone estar conectada a todas las neuronas de la siguiente capa, cuyos arcos definen los pesos asociados.

Las ideas presentadas anteriormente se sumarizan en Fig. 4.

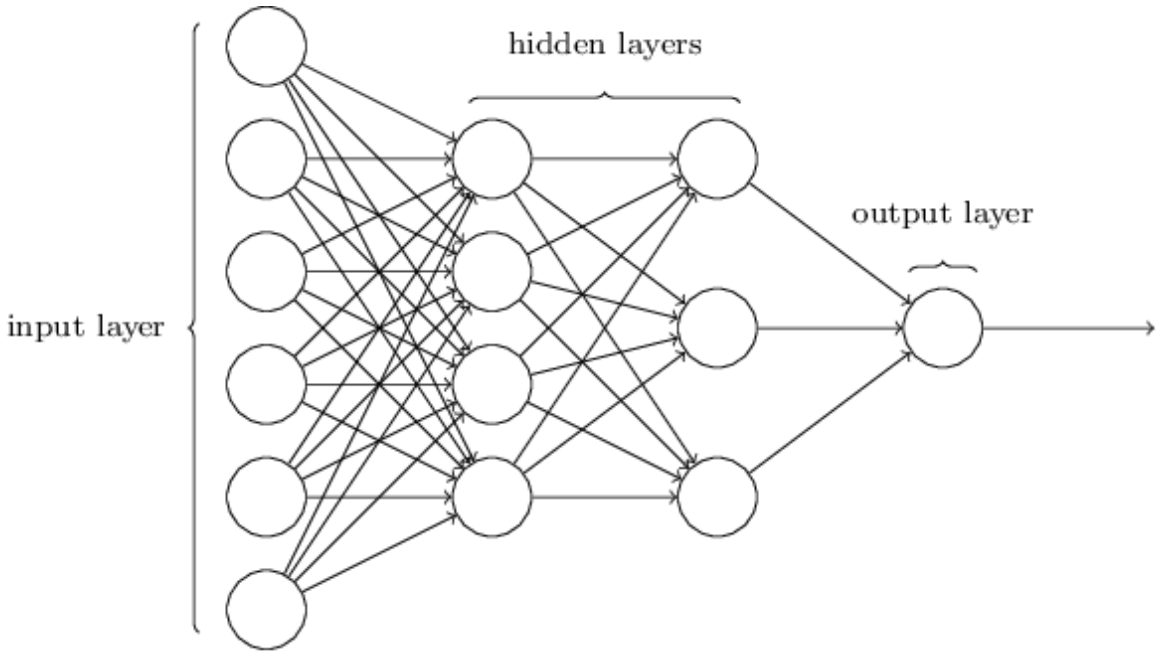

Fig. 4. Red neuronal. Fuente: Nielsen.

Nótese que si se quisiera determinar los valores adecuados para producir la salida correspondiente, se tendría que saber de antemano el valor de la los pesos y del sesgo, lo cual resulta complejo e incluso imposible dependiendo de la aplicación. De esta manera, resulta práctico y conveniente que estos valores sean ajustados por un algoritmo concorde a una muestra significativamente grande de la entrada de datos, a esto se le suele referir como aprendizaje autómata.

La gradiente descendente es un algoritmo que resulta útil en este escenario, ya que se utiliza para encontrar los mínimos locales de una función. La función que queremos aplicar, en este caso, debe relacionar de alguna forma los pesos y los valores de sesgo con la salida, considerando los datos de muestra. Por lo que se define la función costo (3) como:

$$C(w,b) = \frac{1}{2n}\sum_x (y(x) - a)^2 \quad (3)$$

Donde, w representa todos los pesos involucrados en la red neuronal y b todos los valores de sesgo de la misma, n es el número total de datos de muestra, y(x)

es un vector con las activaciones de las neuronas de salida, y a es el vector de salidas esperadas.

Cuando la función se aproxima a cero, entonces significa que y(x) es muy similar a a, por lo que se han encontrado los valores de pesos y sesgos más ideales para esa red neuronal.

Dentro del campo de la detección de objetos, existe una red neuronal específica conocida como *red neuronal convolucional (CNN).* Este tipo de redes neuronales tienen ciertas capas específicas para tratar con imágenes, entre ellas, está una capa de convolución y kernels, una capa RELU, una capa pool y la capa tradicional (MathWorks, s.f.), El propósito es detectar ciertas características progresivamente en la imagen como líneas, curvas y formas más elaboradas.
La principal ventaja es que solo se deben ajustar los valores de los kernels a través de aprendizaje supervisado, lo que resulta menos exigente en términos computacionales que una red neuronal básica.

Este algoritmo se ha optimizado incluso más, existen variantes especializadas en detección de objetos en imágenes que se listan a continuación.

Region-Based CNN (R-CNN) es un algoritmo que reduce el número de lugares en los que podría estar el objeto a identificar basados en ciertos factores como textura, color, etc. El algoritmo, primero, genera potenciales regiones y las clasifica según ciertas características, estas regiones, luego, se refinan a partir de máquinas de vectores de soporte. Suele ser un algoritmo que toma cierto tiempo tanto para la fase de entrenamiento como para la ejecución misma.

Fast R-CNN es un algoritmo que se propuso posteriormente, su funcionalidad es muy parecida a la de Region-Based CNN, con la excepción de que la imagen se procesa en su totalidad para determinar la clasificación de las regiones. Este método es más rápido puesto que algunas regiones se solapan y permiten optimizar cálculos.

Faster R-CNN es un algoritmo que sugiere un método más eficaz para determinar las zonas de interés. Utiliza una red de propuesta de región que acelera el proceso.

El tiempo de ejecución de estos tres algoritmos está determinado, en mayor medida, por la cantidad de regiones de propuesta que se encuentren. Fig. 5. comprende un cuadro comparativo de estos procedimientos.

	Características
Region-Based CNN	Detección de objetos y entrenamiento de la red lento.
Fast R-CNN	Detección de objetos medianamente veloz.
Faster R-CNN	Eficiencia de tiempo de ejecución óptima.

Fig. 5. Cuadro comparativo redes convolucionales. Fuente: MathWorks..

Los tres algoritmos son muy precisos y tienen un rango de error muy bajo, eso sí, siempre y cuando se entrene el modelo apropiadamente, lo que implica tener una gran cantidad de datos.

Además, a pesar de la eficiencia de *Faster R-CNN* comparado con las demás redes convolucionales, sigue siendo poco eficiente para detección de objetos en tiempo real.

1.2. YOLO - You only look once

Este algoritmo aplica una sola red neuronal a toda la imagen con ciertas características únicas. Se divide la imagen en una cuadrícula de S x S y se determina la probabilidad de que haya un objeto dentro de unas definidas N cajas delimitadores (Redmon & Farhadi, s.f.). Esta parte se calcula con el uso de algoritmos de regresión.

El resultado será ciertas cajas con un grado de confianza asociado, sin embargo, la mayoría tendrá un valor probabilístico bajo de que haya un objeto en ella, por lo que se puede definir un umbral para descartar y evitar falsos positivos.

El algoritmo es muy eficiente y resulta conveniente cuando se buscan resultados en tiempo real. Fig. 6. presenta un ejemplo de cómo se vería decodificada la información de la imagen a través de este procedimiento.

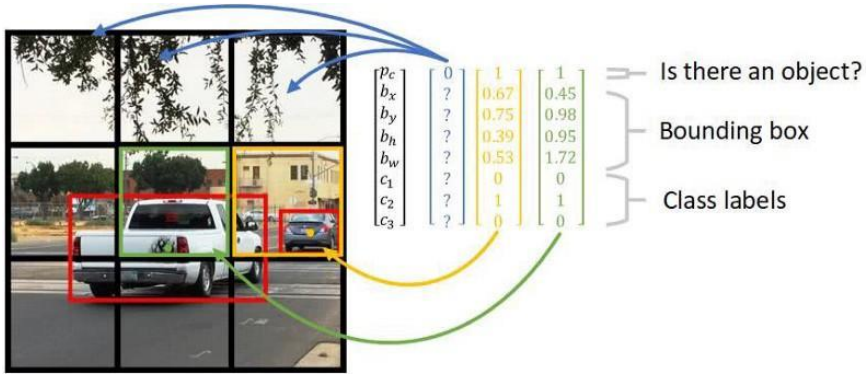

Fig. 6. Ejemplo YOLO. Fuente: Heartkilla.

Se ha comprobado, en la práctica, que es muy útil para aplicaciones en tiempo real. Algunas desventajas del algoritmo comprenden fallos en la detección de objetos ubicados en grupos y de objetos pequeños (Heartkilla, 2019).

Los siguientes algoritmos son usados para el seguimiento de objetos en video. Si bien, es importante notar que estos, en gran medida, dan por hecho que se conoce o se ha detectado previamente la posición del objeto. Por lo que son métodos ligeros que buscan predecir la posición del objeto en el siguiente frame por medio del análisis de información ya conocida.

Por otra parte un método denso es aquel donde se utilizan algoritmos de detección; como los presentados anteriormente, en cada frame, lo cual resulta computacionalmente más exigente.

1.3. Filtro de Kalman

Es un algoritmo estadístico que predice la posición del objeto basado en previa información y verifica la existencia del objeto en la posición prevista.
Es un proceso recursivo que considera ciertas incertidumbres en las mediciones y retroalimenta esta información dada algunas ecuaciones (Naidu & Kumar, 2016). El ciclo del filtro de Kalman se aprecia en Fig. 7.

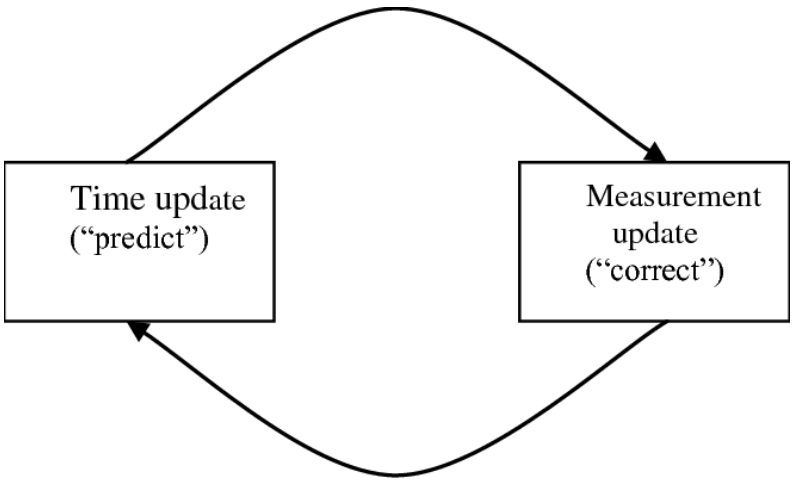

Fig. 7. Ciclo del filtro de Kalman. Fuente: Naidu & Kumar..

En general, durante la etapa de predicción se estima el siguiente estado y el error de covarianza basado en la información previa. Luego, durante la etapa de corrección se miden los valores de las variables en juego y se verifica la precisión de la predicción. Existen fórmulas definidas para estas fases que apoyan el funcionamiento del algoritmo.

Es un algoritmo eficiente en términos de ejecución. Además, siempre y cuando la medición de los datos sea buena, el resultado tendrá un buen nivel de confianza.

1.4. Mean shift

Es un algoritmo que se usa para encontrar los máximos de una función de densidad. Es una forma eficiente de abordar el problema de seguimiento de objetos siempre y cuando su apariencia pueda definirse como un histograma (Azarola, 2017).
Su funcionamiento es iterativo y sigue esta serie de instrucciones:

1. Se define una ventana.
2. Dado un punto x, se calcula la siguiente ecuación:

$$m(x) = \frac{\sum_{xi \in N(x)} K(xi - x)xi}{\sum_{xi \in N(x)} K(xi - x)}$$

   donde N(x) es una función que devuelve los pixeles vecinos de x y k es un kernel.

3. Se actualiza el valor de x, $x \leftarrow m(x)$.
4. Se repite hasta que los puntos estén quietos o salgan del alcance del video.

Esencialmente, el algoritmo se puede resumir en que se buscan los vecinos que afectan un punto de datos, y los mueve acordemente, así pues, los puntos más cercanos tienen más influencia que aquellos que están más alejados.

En cuanto a la confianza, o grado de precisión del algoritmo, se ha demostrado que es susceptible a errores en escenarios complejos, por ejemplo, cuando el objeto se traslada muy rápidamente, o en condiciones de lluvia o neblina.

Ahora bien, para la escogencia del algoritmo se tiene en cuenta la precisión del mismo y la eficiencia en términos de tiempo de ejecución. Nótese que se escoge un algoritmo para cada tarea, es decir, para la detección y el seguimiento. Fig. 8. presenta una ponderación de los algoritmos de detección, donde se da un valor de 0 a 5 por cada factor y se muestra un total.

	Rango de error 50%	**Tiempo de ejecución 50 %**	**Total**

Region based CNN	Muy bajo (5)	Muy prolongado (1)	3
Faster R-CNN	Bajo (4)	Prolongado (2)	3
You only look once	Medio (3)	Rápido (4)	3.5

Fig. 8. Ponderación algoritmos de detección. Fuente: Elaboración propia.

Fig. 9. presenta una ponderación de los algoritmos de seguimiento.

	Rango de error 50%	**Tiempo de ejecución 50 %**	**Total**
Filtro de Kalman	Bajo (4)	Rápido (4)	4
Mean shift	Alto (2)	Prolongado (2)	2

Fig. 9. Ponderación algoritmos de seguimiento. Fuente: Elaboración propia.

De esta forma, se utilizará el algoritmo *You only look once* para la función de detección. Mientras que para el seguimiento en video se hará uso del *Filtro de Kalman.*

Existen distintas APIs que proveen una colección de algoritmos de visión artificial bastante amplia, las cuales se pueden usar para implementar distintas funciones específicas o lograr una gama de resultados bastante novedosos.

2. Application Programming Interfaces (APIs)

2.1. Open CV

Una librería de Software enfocada en la visión artificial y el aprendizaje de máquina. La librería cuenta con cerca de 2500 algoritmos optimizados, que pueden ser utilizados para una variedad de tareas desde reconocimiento hasta seguimiento de objetos en movimiento.

2.2. Tensor-Flow

Una librería de código abierto enfocada a algoritmos de aprendizaje de máquina que fue desarrollada por Google para, principalmente, construir y entrenar distintos tipos de redes neuronales que abarcan una amplia variedad de aplicaciones.

2.3. Video AI

Una librería de Google enfocada exclusivamente a modelos de análisis de videos. Entre sus características están transcripción de audio, detección de texto y seguimiento de objetos. Ofrece mil minutos de video almacenado de manera gratuita.

2.4. Amazon Rekognition

Una librería para utilizar en los servidores de Amazon que abstrae bastante los conceptos relacionados de aprendizaje de máquina y ofrece al usuario una API versátil cuyas funcionalidades abarcan identificación de objetos, personas, texto, reconocimiento facial, entre otros. El servicio tiene un costo de $0.1 por minuto analizado de video.

En cuanto al Hardware, se busca un dispositivo que cumpla con algunos requisitos que se han definido, como lo son la programabilidad o los costos. Se introducen las marcas principales dentro del mercado.

3. Hardware

3.1. Cámaras Inteligentes

Son dispositivos que además de capturar imágenes y/o vídeos, son capaces de extraer información de los mismos como descripción de eventos e incluso pueden tomar decisiones orientadas a un sistema automatizado.
Algunas de las marcas más representativas del mercado disponibles son Vision Components, WYZE, EZVIZ, entre otras.
Asimismo, se pueden usar placas de computadora reducidas para incrustar funcionalidades específicas en cámaras tradicionales.

Hay, en términos generales, dos formas de afrontar el problema para hacer uso de mejores recursos computacionales frente a distintas desventajas y ventajas que esto conlleva.

4. Paradigmas de computación

4.1. Edge Computing

Es un paradigma que refiere al procesamiento y análisis de datos en tiempo real dado que ocurre en los mismos dispositivos de recolección de datos o muy cerca a estos.

En si, se reduce significativamente el consumo de banda ancha, la latencia,e incluso se reducen los problemas de seguridad puesto que no se envían datos a un servidor externo a través de la red.

Sin embargo, cuando se hace necesario correr complejos algoritmos y/o procesos de alto nivel de cómputo, los dispositivos enfrentan ciertas complicaciones. Por lo que se hace necesario el diseño e implementación de algoritmos eficientes que aprovechen al máximo la capacidad del Hardware.

4.2. Cloud Computing

Es un paradigma que refiere al acceso a Hardware y/o Software de manera remota a través de una red. Esto permite una centralización del procesamiento y almacenamiento de la información.

Ofrece altos niveles de cómputo para tareas que lo requieran, así como mantenimiento, accesibilidad y seguridad de la información.

Sin embargo, trae consigo problemas de conectividad, latencia, es decir se presentan demoras en la transmisión de los datos, que pueden llegar a ser significativas en ciertos escenarios.

Se estudian, igualmente, servicios que se ofrecen que destacan por estar a la vanguardia, y podrían de una u otra forma resultar útiles en la solución que se persigue.

5. Programabilidad

5.1. IFTTT

IFTTT significa “If this, then that”, lo que traduce “Si esto, entonces aquello”. Es un servicio basado en internet que conecta aplicaciones, dispositivos y otros servicios con el objetivo de disparar algún evento (Causa-Efecto) (Martin & Finnegan, 2019). Un ejemplo de un applet sería: Silenciar un dispositivo Android cuando se llegue al trabajo. Estos applets pueden ser elaborados libremente siempre y cuando el dispositivo o la aplicación puedan hacer o recibir peticiones web, esto se conoce como *webhooks*.

Se estima que hoy en día se han elaborado más de 54 millones de applets IFTTT.

Ponderación

Dentro de las distintas posibilidades en cuanto al Hardware que se pueden usar, se realiza una comparativa para contrarrestar cada una de las ventajas y desventajas de cada una de las distintas opciones disponibles.

La *Programabilidad* determina que tanto se puede modificar el comportamiento del sistema según las necesidades requeridas. La *Accesibilidad* determina la facilidad y posibilidad de transmitir o acceder a la información que graba cada cámara para ser analizada luego en un equipo de cómputo distinto. El *Precio* consiste en la relación económica y de calidad de cada producto. El *Tiempo de entrega* refiere a la facilidad y el tiempo que toma tener el producto disponible para trabajarlo.

	Programabilidad	**Accesibilidad**	**Precio**	**Tiempo de entrega**
Vision Components Smart Cameras	Programables en C/C++. Soportan desarrollo Open Source con herramientas tales como OpenCV.	Ofrecen una API exclusiva llamada VC Lib para procesamiento de imagen.	$5.000.000	Depende de la compañía de envíos
WYZE	No es programable, ni soporta ningún tipo de API.	Está restringida por la funcionalidad ofrecida por la App de WYZE.	$120.000	Disponible para entrega en Amazon.
EZVIZ	No es completamente programable.	Compatible con IFTTT.	$170.000	Disponible para entrega en Amazon.
Raspberry pi Camera Module	Completamente programable.	Se puede ajustar fácilmente a cualquier protocolo y/o tecnología. Ejemplo, IFTTT	$150.00	Disponibilidad inmediata.

Fig. 10. Cuadro comparativo entre las alternativas para el proyecto. Fuente: Elaboración propia.

La ponderación consiste de comparar cada uno de los productos frente a las alternativas. Cada criterio tiene un peso dado, y el puntaje calculado determina la viabilidad de cada opción.

La Raspberry pi Camera Module es la mejor elección puesto que es un microprocesador que se puede programar según la necesidad del usuario, esto implica también que los datos se pueden procesar en un dispositivo de mejor computo, y su precio con relación a las

especificaciones del producto son aceptables. Además, permite usar cualquier cámara para la realización del proyecto.

	Programabilidad 30%	**Accesibilidad 40%**	**Precio 20%**	**Tiempo de entrega 10%**	**Total**
Vision Components Smart Cameras	5	4	0	3	3.4
WYZE	0	2	5	5	2.3
EZVIZ	2	4	5	5	3.7
Raspberry pi Camera Module	4	4	5	5	4.3

Fig. 11. Ponderación. Fuente: Elaboración propia.

Metodología

Para la correcta gestación del proyecto se realizan una serie de actividades críticas en el proceso. Estas actividades deben llevarse a cabo apropiadamente, por lo que se explica la metodología que se usa a lo largo de la ejecución del presente proyecto.

Ahora bien, se retoma a un concepto de metodologías de desarrollo de Software conocidas como metodologías ágiles. Estas metodologías pretenden lograr una entrega rápida del software de manera incremental, y además son adaptables al cambio. (Pressman, 2005).

En detalle, se hace uso de la metodología *Programación Extrema* (XP), la cual comprende cuatro actividades estructurales, sean estas, planeación, diseño, desarrollo y pruebas. (Pressman, 2005).

Las actividades a desarrollar, entonces, están ligadas a la anterior metodología. A continuación, se explica con más detalle de que se compone cada fase.

La fase de planeación implica la identificación de los requerimientos funcionales del sistema, por lo que, en este caso, se retoma a la pregunta de investigación. Luego, el diseño comprende la especificación de la arquitectura que se usará, así como la estructura interna del sistema. En este escenario, puesto que se hace uso de herramientas externas, se discuten y estudian, además, la arquitectura de estos sistemas externos.

Seguidamente, el desarrollo alude a la codificación de la solución de Software y las pruebas refieren a una seria de estrategias para evaluar y verificar que se haya cumplido con los requerimientos propuestos en un inicio.

Finalmente, se analizan los resultados obtenidos y se discute la viabilidad del sistema en términos computacionales, financieros y sociales.

Requerimientos funcionales

En esta sección se describen los requerimientos funcionales que han sido identificados para la construcción del sistema inteligente de detección y rastreo de actividades sospechosas. La Fig. 12. muestra el resumen de estos.

	Requerimiento de usuario	**Requerimiento de sistema**	
#1	Detectar personas en una grabación de video en tiempo real.	**#A**	Resaltar el área donde se ha detectado la persona.
		#**B**	Actualizar la detección cada cierto tiempo.
#2	Proveer retroalimentación sobre actividades sospechosas que se han percibido.	**#A**	Informar el tiempo que una persona ha permanecido en video.
		#B	Rastrear a las personas en video.

Fig.12. Requerimientos funcionales.

Diseño

Se decidió usar la librería de código abierto OpenCV para el desarrollo de la solución (OpenCV, 2019). Dado eso, se estudiaron los métodos y clases pertinentes para la implementación de los algoritmos previamente ponderados.

Con el diseño del software, se busca lograr un producto que sea escalable, fácil de mantener, entendible y óptimo. En vista de este hecho, se hace uso del patrón de diseño de Software *Facade,* el cual pretende reducir la complejidad para el cliente.

El objetivo de este patrón de diseño es el de proporcionar una interfaz fácil de manejar que abarque un sistema complejo. El patrón se comprende de dos partes principales. La fachada o *Facade* y las subclases (Gamma, E., Vlissides, J., Helm, R., & Johnson, R., 1994) . Estas últimas son las que, en últimas, implementan la funcionalidad del procedimiento. En Fig. 13. se aprecia una estructura general del patrón *Facade.*

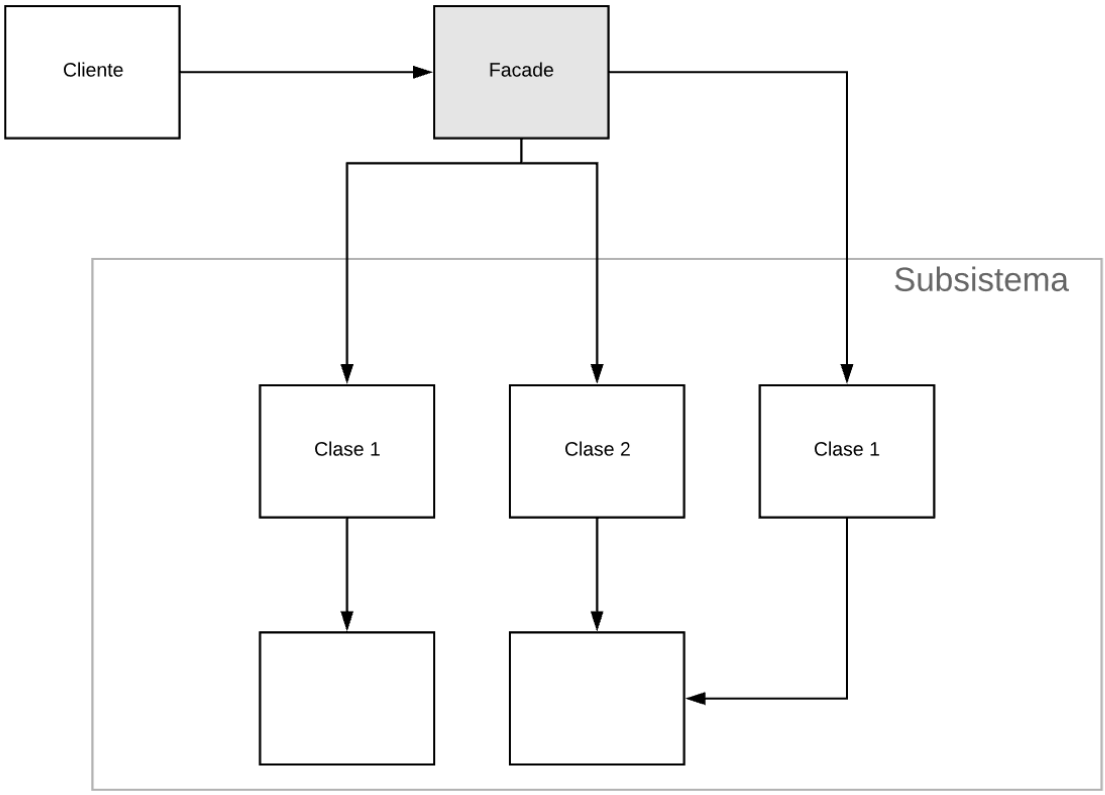

Fig. 13. Patrón de diseño Facade. Fuente: Elaboración propia.

En este caso particular, el patrón *Facade* resulta bastante útil en cuanto que la identificación y rastreo de personas es una tarea compuesta de complejos modelos matemáticos, por lo que poder ofrecer una interfaz desde la cual se pueda abstraer varios de los conceptos ligados a análisis de videos, termina siendo beneficiosos para el usuario, sea este, el programador.

Se utiliza el lenguaje unificado de modelado (UML) con el fin de ilustrar la organización de las clases, tanto provistas por OpenCV como las diseñadas propiamente. En Fig. 14. se aprecia el modelo realizado.

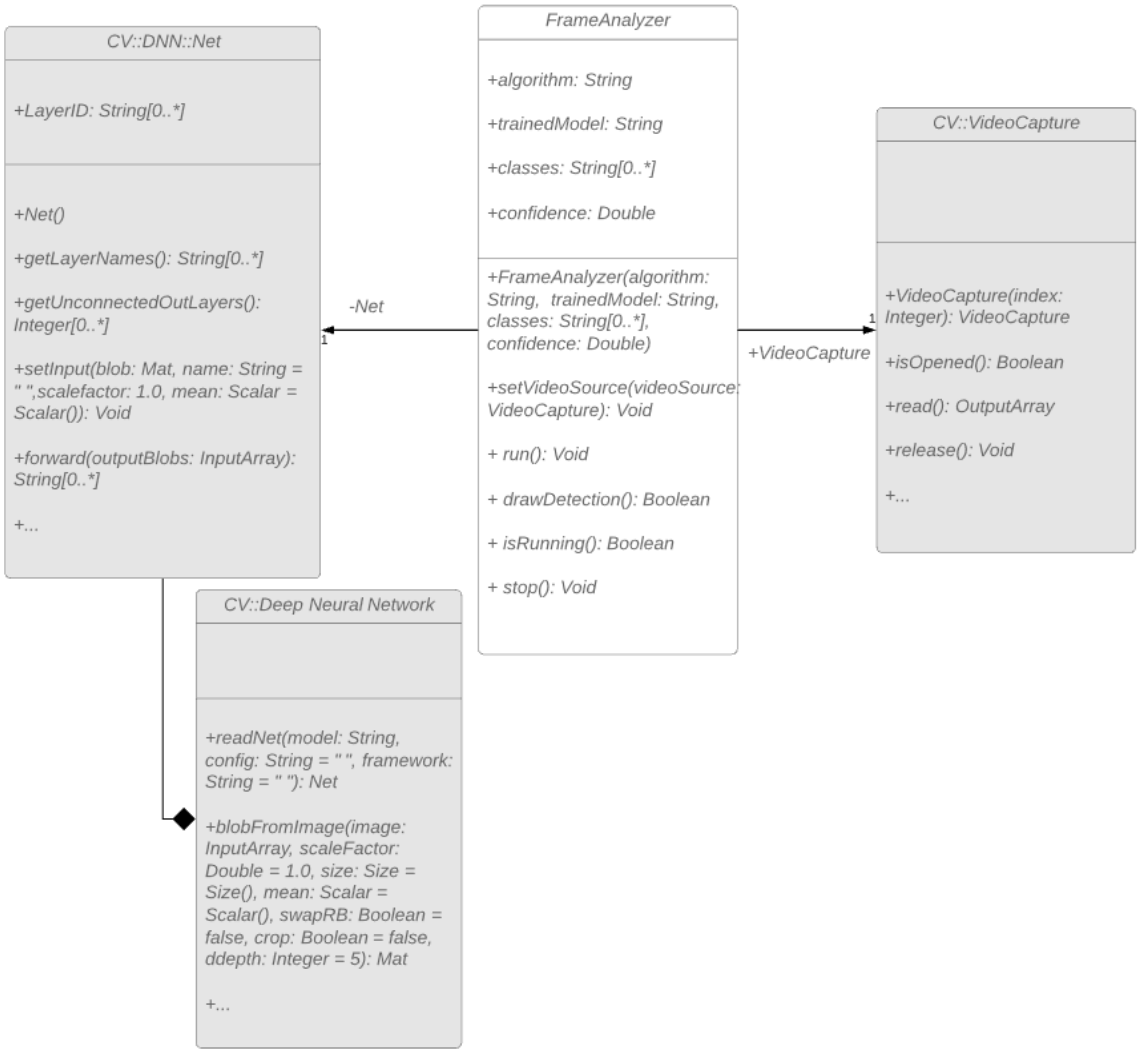

Fig. 14. Diagrama de clases. Fuente: Elaboración propia.

Las clases resaltadas en gris claro corresponden a clases de OpenCV. Nótese que por simplicidad estas solo muestran los métodos y atributos a usar. La estructura propuesta se trató de mantener tan simple como fuera posible, considerando, escalabilidad, funcionalidad y mantenimiento.

En Fig. 15. se listan y se describen los métodos considerados vitales de la clase *CV::DNN::Net.*

Método	**Funcionalidad**	**Parámetros**	**Retorna**
+Net()	Constructor por defecto.	Ninguno.	Nada.

+getLayerNames():String[0..*]	Obtener el nombre de las capas de la red neuronal.	Ninguno.	Arreglo de strings que representan los nombres de cada una de las capas de la red neuronal.
+getUnconnectedOutLayers(): Integer[0..*]	Obtener las capas de salida de la red neuronal.	Ninguno.	Arreglo de enteros que representan los índices de cada una de las capas de salida de la red neuronal.
+setInput(blob: Mat, name: String = " ", scaleFactor: Double = 1.0, mean: Scalar = Scalar()): Void	Establece el nuevo valor de entrada para las capas de entrada de la red neuronal.	*blob:* Conjunto de píxeles conectados de la imagen. *name:* El nombre de la capa de entrada. (Opcional) *scaleFactor:* Una escala de normalización. (Opcional) *mean:* valores de la resta del promedio. (Opcional)	Nada.
+forward(outputBlobs: InputArray): String[0..*]	Computa la salida de la red neuronal especificada. Nótese que si se pasan las capas de salida, se calculan los valores de salida para toda la red neuronal.	*outputBlobs:* Capas para las cuales se desea calcular el valor de salida	Arreglo con los valores de salida de las capas listadas de la red neuronal.

Fig. 15. Métodos CV::DNN::Net. Fuente: OpenCV.

Como se puede ver, esta clase es mayormente responsable por el manejo de la red neuronal, es decir, tanto de las capas de entrada, escondidas y de salida.

Ahora bien, se presenta en Fig. 16. las funciones necesarios del módulo *CV::Deep Neural Network.*

Método	**Funcionalidad**	**Parámetros**	**Retorna**
+readNet(model: String, config: String = " ", framework: String = " "): Net	Lee una red de aprendizaje profundo en alguno de los formatos aceptados. La red neuronal debe haber sido entrenada previamente.	*model:* Archivo binario que contiene los pesos de la red neuronal entrenada. Se aceptan formatos .pb, .net, .weights, entre otros. *config:* Archivo que contiene la configuración de la red neuronal. Se aceptan formatos como .xml, .cfg, entre otros. *framework:* Nombre explícito del marco de trabajo para determinar un formato válido.	Un objeto tipo CV::DNN::Net que representa la red neuronal especificada.
+blobFromImage(image: InputArray, scaleFactor: Double = 1.0, size: Size = Size(), mean: Scalar = Scalar(), swapRB: Boolean = false, crop: Boolean = false, ddepth: Integer = 5):Mat	Crea un conjunto de píxeles conectados a partir de una imagen. Opcionalmente, puede recortarse la imagen desde el centro, restar valores promedio y escalar.	*image:* Imagen a convertir. *scaleFactor:* valor para escalar la imagen. *size:* tamaño de la imagen de salida.	Un arreglo de 4 dimensiones representando el objeto creado.

		mean: valor escalar de los promedios a restar de los canales de la imagen. *swapRB:* Indica si intercambiar el primer y último canal de la imagen es necesario. *crop:* Indica si la imagen será recortada después de redimensionarla. *ddepth:* Profundidad del conjunto de píxeles de salida.	

Fig. 16. Funciones CV::Deep Neural Network. Fuente: OpenCV.

El propósito general de este módulo es proveer funciones generales para los distintos tipos de redes neuronales que pueden haber. De hecho, eso explica la relación entre este módulo y la clase *CV::DNN::Net* que se encuentra contenida en ella.

Ahora bien, se necesita un módulo que permita llevar a cabo una interacción en tiempo real con una cámara. La clase, *CV::VideoCapture* resulta conveniente en este escenario. En la Fig. 17. se muestran los métodos a usar.

Método	**Funcionalidad**	**Parámetros**	**Retorna**
+VideoCapture(index: Integer):VideoCapture	Abre una cámara para captura de video.	*index:* Identificador de la cámara. La cámara por defecto tiene índice 0.	Un objeto de tipo VideoCapture que representa la cámara que ha sido abierta.

+isOpened():Boolean	Determina si la cámara se encuentra abierta, en otras palabras, lista para funcionar.	Ninguno.	Verdadero si la cámara ha sido inicializada previamente para captura de video. De lo contrario, falso.
+read():OutputArray	Decodifica y devuelve el siguiente frame de video.	Ninguno.	Un arreglo que representa el siguiente frame de video.
+release():Void	Cierra el dispositivo de captura de video.	Ninguno.	Nada.

Fig. 17. Métodos CV::VideoCapture. Fuente: OpenCV.

Finalmente, se diseñó una clase *FrameAnalyzer* que permite mejorar el flujo de comunicación entre las distintas clases de OpenCV que ofrecen mayor parte de la funcionalidad. Los miembros de esta clase se describen en Fig. 18.

Miembro	**Descripción**
+algorithm: String	Representa el nombre del algoritmo a usar para el análisis de los frames del video. Por ejemplo, “yolov3” representa la versión 3 del algoritmo YOLO.
+trainedModel: String	Representa el nombre del archivo binario con los pesos del modelo entrenado de la red neuronal.
+classes: String[0..*]	Representa un arreglo de strings donde cada entrada es una etiqueta de un objeto que se desea analizar dentro del video.
+confidence: Double	Representa un límite inferior aceptable de confianza.
-net: CV::DNN::Net	Representa una red neuronal con las configuraciones necesarias del algoritmo que se esté usando.
+videoCapture: CV::VideoCapture	Representa un objeto de tipo CV::VideoCapture que permite manejar la cámara.

Fig. 18. Miembros FrameAnalyzer. Fuente: Elaboración propia.

Las acciones por las que toma responsabilidad la clase *FrameAnalyzer* se especifican en Fig. 19.

Método	**Funcionalidad**	**Parámetros**	**Retorna**
+FrameAnalyzer(algorithm: String, trainedModel: String, classes: String[0..*], confidence: Double)	Constructor de la clase cuyo último fin es inicializar el algoritmo, el modelo entrenado, las etiquetas de los objetos a analizar, y el nivel de confianza.	*algorithm:* Nombre del algoritmo a usar. *trainedModel:* Ubicación del archivo binario que almacena los pesos del modelo de la red neuronal entrenado. *classes:* Arreglo de strings donde cada entrada representa una etiqueta de objetos a analizar. *Confidence:* Límite inferior aceptable de confianza.	Nada.
+setVideoSource(videoSource: VideoCapture):Void	Asigna una referencia al dispositivo de vídeo para analizar los frames. Nótese que la instancia de VideoCapture debe estar inicializada antes llamar a este método.	*videoSource:* Instancia que referencia el dispositivo de grabación de video. Esta instancia debe estar inicializada.	Nada.
+run():Void	Empieza a ejecutar el algoritmo. Nótese que se debe haber llamado previamente al	Ninguno	Nada

	método que referencia el dispositivo de video. Además, no debe llamar a este método más de una vez.		
+drawDetection():Boolean	Dibuja un rectángulo en el área del frame que ha sido correctamente analizado.	Ninguno.	Verdadero si se dibujó algún rectángulo. Falso de lo contrario.
+isRunning():Boolean	Determina si el algoritmo está siendo ejecutado. Se recomienda llamar a este método antes de correr el algoritmo.	Ninguno.	Verdadero si el algoritmo está ejecutándose. Falso de lo contrario.
+stop():Void	Realiza un proceso de limpieza sobre los datos. Nótese que detiene la ejecución del algoritmo si este se esta ejecutando.	Ninguno.	Nada.

Fig. 19. Métodos FrameAnalyzer. Fuente: Elaboración propia.

Esta clase permite generar un nuevo nivel de abstracción que facilita el entendimiento del código fuente para otros clientes.

Nótese que para lograr el resultado esperado usando una instancia de la clase *FrameAnalyzer* es importante considerar la máquina de estados asociados al mismo. Véase Fig. 20.

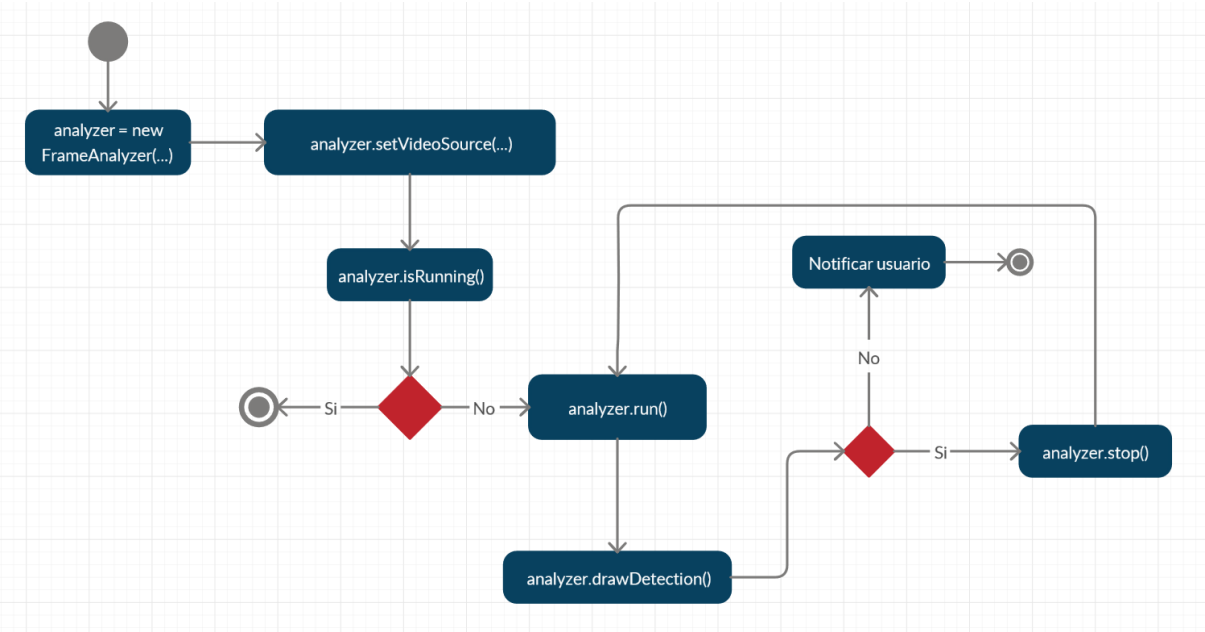

Fig. 20. Máquina de estados. Fuente: Elaboración propia.

La máquina de estados describe el algoritmo asociado a la detección y rastreo de personas desde una capa de abstracción superior.

Casos de prueba

Con el fin de identificar la completa funcionalidad del sistema se plantean una serie de pruebas.

Cada prueba consta de una descripción, requerimiento funcional que mide, pasos a seguir, resultado esperado y resultado obtenido.

Detección de objetos	**Código caso de prueba**	#1
Descripción: Verificar la correcta clasificación de objetos por parte del sistema. Se usarán los siguientes objetos para tal fin: teléfono celular, balón, camisa, zapatos y persona.		
Pasos: Tomar cada uno de los objetos en tiempos no solapantes y ponerlos frente a la cámara.		
Resultado esperado: Para el objeto persona, el sistema debe dibujar un rectángulo en el área donde ésta se ubique. Para los demás objetos, el sistema no debe dibujar nada sobre los frames.		
Resultado obtenido: El sistema resaltaba correctamente el área donde se ubica una persona dentro del metraje.		

Fig. 21. Caso de prueba #1. Fuente: Elaboración propia.

Detección simultánea de personas	**Código caso de prueba**	#2
Descripción: Verificar la correcta clasificación de múltiples personas en video por parte del sistema. Se posicionarán de 2 a 4 personas en video para tal fin.		
Pasos: Ubicar dos personas de contexturas diferentes frente a la cámara. Ubicar tres personas de contexturas diferentes frente a la cámara. Ubicar cuatro personas de contexturas diferentes frente a la cámara.		
Resultado esperado: El sistema debe correctamente dibujar un rectángulo enmarcando a cada persona que sea visible a la cámara, sin importar, la cantidad de individuos presentes.		
Resultado obtenido: El sistema reportó la ubicación de varias personas en el frame. La detección se perdía por momentos cuando las personas se cruzaban, pero se considera un aspecto que se puede tolerar.		

Fig. 22. Caso de prueba #2. Fuente: Elaboración propia.

Detección continua de personas	**Código caso de prueba**	#3

Descripción: Verificar la correcta detección y rastreo de personas en video por parte del sistema. Las personas se moverán en direcciones aleatorias durante 10 segundos.
Pasos: Posicionar una persona al alcance de la cámara. La persona no debe dejar de moverse en direcciones aleatorias siempre permaneciendo en el rango de vista del dispositivo de video.
Resultado esperado: El sistema debe actualizar el rectángulo que se dibuja sobre la persona fluidamente según la nueva posición del individuo.
Resultado obtenido: El programa actualiza correctamente la ubicación de un individuo.

Fig. 23. Caso de prueba #3. Fuente: Elaboración propia.

Informe de actividad sospechosa	**Código caso de prueba**	#4
Descripción: Verificar la veracidad del sistema en cuanto al rastreo de personas.		
Pasos: Ubicar una persona frente a la cámara. La persona debe moverse en direcciones aleatorias y salir del alcance de la cámara después de *t* segundos. Realizar el procedimiento para $t = 6$ s, $t = 10$ s y $t = 25$ s.		
Resultado esperado: El sistema debe indicar en pantalla el tiempo que cierta persona fue visible a la cámara. Se espera obtener los tiempos resultantes $t = 6$ s, $t = 10$ s y $t = 25$s.		
Resultado obtenido: El sistema reportó tiempos cercanos a los reales, y se consideró un rango de error tolerable.		

Fig. 24. Caso de prueba #4. Fuente: Elaboración propia.

Es importante señalar que no se supone proceder a los siguientes casos, en orden ascendente, si no se ha cumplido satisfactoriamente los anteriores.

Adicionalmente, se discute la trazabilidad de cada una de las pruebas planteadas previamente, para ello se evalúa qué requerimiento funcional mide cada prueba. Véase Fig. 25.

	Requerimiento #1A	**Requerimiento #1B**	**Requerimiento#2 A**	**Requerimiento #2B**
Caso de prueba #1	X			
Caso de prueba #2	X			
Caso de prueba #3		X		X

Caso de prueba #4			X	

Fig. 25. Trazabilidad de los casos de prueba. Fuente: Elaboración propia.

Como se puede visualizar, los casos de prueba, en su totalidad, permiten evaluar el correcto funcionamiento del sistema en cuanto que se satisfagan los requerimientos identificados en una primera instancia.

Resultados

Se desarrolló un producto que siguió el diseño sugerido usando, como se estableció, la librería OpenCV. El programa se enfrentó a las pruebas planteadas y se anotaron los resultados obtenidos respecto a los requerimientos funcionales del sistema. Fig. 26. ilustra el funcionamiento del programa.

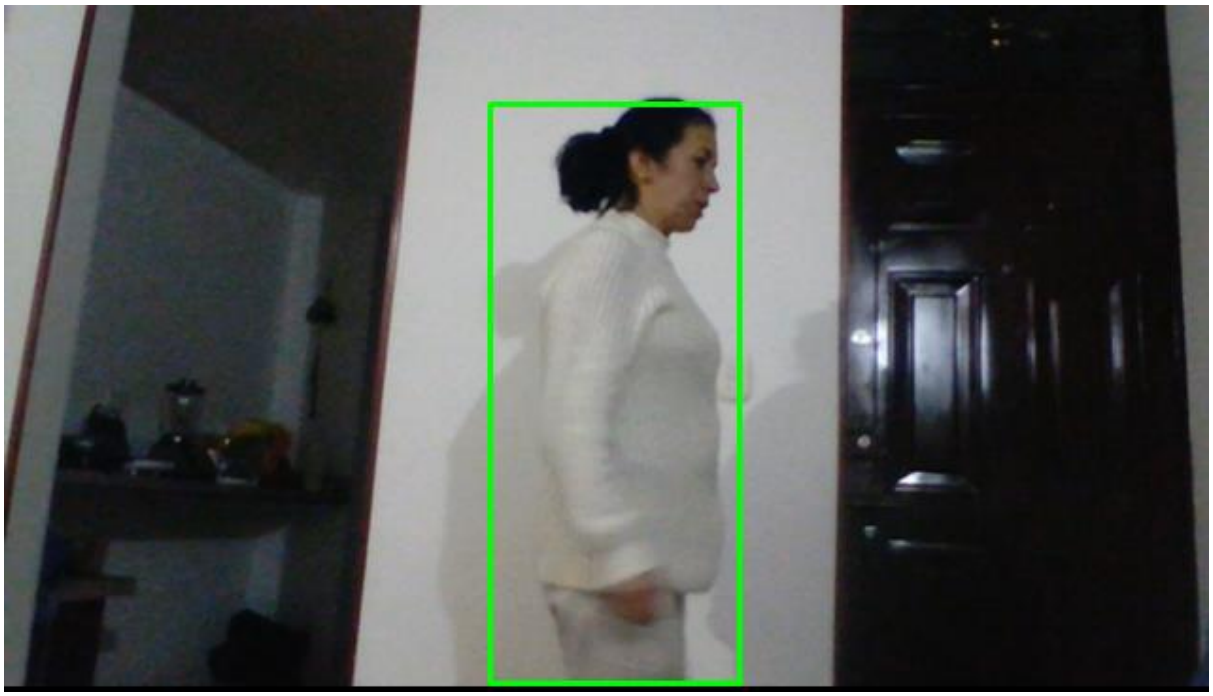

Fig. 26. Demostración de la funcionalidad del sistema.

Conclusiones

Se obtuvo un producto funcional, que no solo demuestra las aplicaciones que tienen las tecnologías emergentes, sino que de la misma manera demuestra un funcionamiento versátil.
Comparado con otros métodos de detección, el estudiado a lo largo de este documento se destaca por su portabilidad. No obstante, todavía existe un amplio rango de progreso que permita mejorar el tiempo de respuesta en ambientes que requieren toma de decisiones inmediatas.

Referencias

Arroyo, R., Yebes, J., Bergasa, L., Daza, I., & Almazán, J. (2015). Expert video-surveillance system for real-time detection of suspicious behaviors in shopping malls. Expert Systems with Applications. Recuperado de https://www.sciencedirect.com/science/article/pii/S0957417415004182

Azarola, I. (2017). Segmentación por medio del método Mean Shift: Estado del arte. Recuperado de https://ri.itba.edu.ar/bitstream/handle/123456789/1198/Informe.pdf?sequence=1&isAllowed=y

Biuk-Aghai, R., Si, Y., Fong, Si., & Yan, P. (s.f.). Security in Physical Environments: Algorithms and System for Automated Detection of Suspicious Activity. Department of Computer and Information Science, University of Macau, Macau. Recuperado de https://www.semanticscholar.org/paper/Security-in-Physical-Environments-%3A-Algorithms-and-Biuk-Aghai-Si/bcd74d86c4877e4726fb3210b0d954f15fa6511b

Elafi, I., Jedra M., & Zahid, N. (2016). Unsupervised detection and tracking of moving objects for video surveillance applications. Pattern Recognition Letters. Recuperado de https://www.semanticscholar.org/paper/Unsupervised-detection-and-tracking-of-moving-for-Elafi-Jedra/fbafa07fad60ae4006ae251e59869656ff9cde17

Gamma, E., Vlissides, J., Helm, R., & Johnson. R. (1994). Design Patterns: Elements of Reusable Object-Oriented Software.

Heartkilla. (2019). yolo-v3. Recuperado de https://github.com/heartkilla/yolo-v3

Ko, K, & Sim, K. (2017). Deep convolutional framework for abnormal behavior detection in a smart surveillance system. Engineering Applications of Artificial Intelligence. Recuperado de https://www.sciencedirect.com/science/article/abs/pii/S0952197617302579

Machine-vision Research Group. (2018). An overview of deep-learning based object-detection algorithms. Recuperado de https://medium.com/@fractaldle/brief-overview-on-object-detection-algorithms-ec516929be93

Martin, J., & Finnegan, M. (2019). What is IFTTT? How to use If This, Then That services. Recuperado de https://www.computerworld.com/article/3239304/what-is-ifttt-how-to-use-if-this-then-that-services.html

MathWorks. (s.f.). Getting started with R-CNN, Fast R-CNN, and Faster R-CNN. Recuperado de https://www.mathworks.com/help/vision/ug/getting-started-with-r-cnn-fast-r-cnn-and-faster-r-cnn.html#mw_5ad75928-8822-4277-a1f6-6a762a5bda32

Murugavel, M. (s.f). Multiple Object Tracking algorithms. Recuperado de https://medium.com/@manivannan_data/multiple-object-tracking-algorithms-a01973272e52

Naidu, V., Sivakumaran, N., & Kumar, V. (2016). Six object tracking algorithms: a comparative study. Recuperado de https://www.researchgate.net/publication/307956877_Six_Object_Tracking_Algorithms_A_Comparative_Study

Nielsen, M. (2019). Using neural nets to recognize handwritten digits. Recuperado de http://neuralnetworksanddeeplearning.com/chap1.html

OpenCV. (2019). Deep Neural Network module. Recuperado de https://docs.opencv.org/3.4/d6/d0f/group__dnn.html

OpenCV. (2019). cv::dnn::Net Class Reference. Recuperado de https://docs.opencv.org/3.4/db/d30/classcv_1_1dnn_1_1Net.html

OpenCV. (2019). cv::VideoCapture Class Reference. Recuperado de https://docs.opencv.org/3.4/d8/dfe/classcv_1_1VideoCapture.html

Pressman, R. (2005). Ingeniería del Software. Un enfoque práctico. Recuperado de http://cotana.informatica.edu.bo/downloads/ld-Ingenieria.de.software.enfoque.practico.7ed.Pressman.PDF.

Redmon, J., & Farhadi, A. (s.f.). YOLOv3: An incremental improvement. Recuperado de https://pjreddie.com/media/files/papers/YOLOv3.pdf