

CECAL

INSTITUTO DE COMPUTACIÓN, FACULTAD DE INGENIERÍA
UNIVERSIDAD DE LA REPÚBLICA
MONTEVIDEO, URUGUAY

PROYECTO DE GRADO INGENIERÍA EN COMPUTACIÓN

Algoritmos de inteligencia computacional para la detección de patrones de movimiento de personas

Juan P. Chavat
juan.pablo.chavat@fing.edu.uy (jpchavat@gmail.com)

Juan A. Gómez
juan.gomez.simonelli@fing.edu.uy (jagomsim@gmail.com)

Ismael Silveira
ismael.silveira@fing.edu.uy (ismaelsilca@gmail.com)

Marzo de 2015

Tutor de Proyecto:
Sergio Nesmachnow, Universidad de la República.

Algoritmos de inteligencia computacional para la detección de patrones de movimiento
de personas

J.P. Chavat, J.A. Gómez, I. Silveira

Proyecto de Grado

CECAL

Instituto de Computación - Facultad de Ingeniería

Universidad de la República

Montevideo, Uruguay, Marzo de 2015

ALGORITMOS DE INTELIGENCIA COMPUTACIONAL PARA LA DETECCIÓN DE PATRONES DE MOVIMIENTO DE PERSONAS

RESUMEN

El número de cámaras de vigilancia instaladas en los últimos años ha ido en aumento y, con ello, también aumentó el tamaño y costo de los centros de gestión y control de los sistemas de vigilancia. Aunque se ha avanzado considerablemente en el procesamiento de imágenes para la observación de hechos anómalos, aún se está lejos de prescindir del agente humano como último eslabón de la cadena de toma de decisiones.

Este trabajo realiza un recorrido por el estado del arte del procesamiento de imágenes y de la detección de patrones, a la vez que brinda una prueba de concepto de un sistema capaz de generar alertas ante hechos pre-definidos que ocurren en una secuencia de imágenes. En otras palabras, el sistema actúa de herramienta para priorizar el foco de atención de los agentes humanos involucrados.

En la implementación del sistema se utilizan tecnologías tales como Python3 y RabbitMQ, así como también métodos ya implementados, proporcionados por las bibliotecas OpenCV, Scipy, FilterPy, y Numpy, entre otras. El sistema se estructura bajo una arquitectura basada en tubos y filtros donde cada filtro representa una -o parte de una- etapa del proceso, facilitando la prueba de diferentes métodos para una misma etapa. En forma general, el sistema implementado cuenta con dos etapas principales, donde la primera etapa está dedicada a la extracción de las características de las imágenes (detección de los objetos de interés y seguimiento de su posición en el tiempo), mientras la segunda etapa se dedica a la detección del cumplimiento de patrones por parte de los objetos seguidos.

Se realiza un análisis experimental que abarca las dos etapas principales del sistema, y donde se planifican y llevan a cabo más de 1450 instancias de procesamiento. Para la etapa de extracción de características de las imágenes, se hace una agrupación de parámetros de configuración que se varían bajo las mismas condiciones, llegando de esa forma a una configuración sub-óptima de valores de los mismos. Los análisis son efectuados con el vídeo de prueba *PETS09-S2L1* y los resultados son comparados con la lista de algoritmos del portal *MOT Challenge*.

Los principales aportes de este trabajo se concentran en el estudio del marco teórico necesario para la implementación de sistemas que, en tiempo real y tomando como entrada secuencias de imágenes, sean capaces de extraer características de estas y utilizar las características para detectar el cumplimiento de patrones pre-definidos. El marco teórico proporcionado por este trabajo es un recurso de valor para cualquier equipo de trabajo cuya intención sea el estudio e implementación de sistemas de la misma índole. Además, se aporta la implementación de un sistema que aplica las técnicas relevadas, con el cual se logra proporcionar una prueba fehaciente de la viabilidad en la construcción de este tipo de sistemas.

Palabras clave: Procesamiento de imágenes, Patrones de movimiento de personas, Cámaras de seguridad, Inteligencia computacional

COMPUTATIONAL INTELLIGENCE ALGORITHMS FOR DETECTING PERSON'S MOVEMENT PATTERNS

ABSTRACT

The number of installed surveillance cameras has grown in the last years and, with it, the size and cost of the management and control centers of the surveillance systems also increased. Although there have been considerable advances in the image processing techniques for the observation of the unnormal facts, we are far away of leaving aside the human agents as the last step in the decisions chain.

This work takes a journey through the state of the art of image processing and pattern detection, plus it gives a proof of concept of a system able to warn of preset facts detected in a sequence of images. In other words, the system is a tool to prioritize the attention focus of the human agents.

In the development of the system the used technologies are Python3 and RabbitMQ, as well as some already developed methods provided by third party libraries as OpenCV, Scipy, FilterPy, and Numpy, among others. The system follows a pipes and filters based architecture where each filter represents one -or a part of a- stage in the process, making easy the trial of different methods for one stage. From a macro point of view, the system counts with two main stages, where the first stage is dedicated to the extraction of features from the images (detection of objects of interest and their tracking through the time) whereas the second stage is dedicated to the detection of patterns satisfied by the tracked objects.

An experimental analysis is made which embraces the two main stages of the system, and where there are planned and done more than 1450 experiments. For the image's features extraction stage, a grouping of configuration parameters values are varied under the same conditions, thereby achieving a sub-optimum configuration of the parameters values. The tests are run using the PETS09-S2L1 testing video and the results are compared to the MOT Challenge portal listed algorithms.

It was made a study of the theoretical frame needed for the development of a system that, in real time, is capable of extracting features from a sequence of images and detect unnormal events in those features, based on preloaded patterns. This work contributes with a knowledge base to be used by any working team interested in developing this kind of systems. Besides, it was achieved an implementation of a system that applies the techniques studied in the work, with which it was possible to provide reliable evidence on the viability of the development of this kind of systems.

Keywords: Image processing, Person's movement patterns, Surveillance cameras, Computational intelligence

Índice general

1. Introducción	1
2. Procesamiento de imágenes y detección de patrones	3
2.1. Procesamiento de imágenes	3
2.1.1. Captura	4
2.1.2. Pre-procesamiento	4
2.1.3. Segmentación	4
2.1.4. Extracción de características	5
2.1.5. Identificación de objetos	9
2.2. Detección de patrones	10
2.2.1. Segmentación	11
2.2.2. Extracción y selección de características	11
2.2.3. Clasificación	14
2.3. Aplicaciones del procesamiento de imágenes y la detección de patrones . .	15
3. Detección de patrones de movimiento anómalos en videovigilancia	19
3.1. Especificación del problema	19
3.2. Planificación del proyecto	21
3.2.1. Cronograma	21
3.3. Trabajos relacionados	21
3.3.1. Sistemas de videovigilancia	21
3.3.2. Procesamiento de imágenes	22
3.3.3. Detección y seguimiento de personas	24
3.3.4. Detección de anomalías	25
4. Arquitectura y diseño del sistema	29
4.1. Arquitectura general del sistema	29
4.2. Arquitectura del módulo Reconocimiento y seguimiento	31
4.3. Arquitectura del módulo Buscador de patrones	32
4.3.1. Método para encontrar patrones	32
4.3.2. Medición del error en el cumplimiento de patrones	34
4.4. Arquitectura de los módulos auxiliares	35
5. Implementación	37
5.1. Elección de las tecnologías utilizadas para el desarrollo	37
5.2. Comunicación entre módulos	39
5.3. Detalles generales de la implementación	40
5.4. Implementación del módulo Reconocimiento y seguimiento	43

5.4.1. Sustracción de fondo	43
5.4.2. Detección de <i>blobs</i>	45
5.4.3. Clasificación de <i>blobs</i>	46
5.4.4. Seguimiento	53
5.4.5. Parámetros de configuración del módulo Reconocimiento y seguimiento	66
5.5. Implementación del módulo Buscador de patrones	73
5.5.1. Definición de patrones	74
5.5.2. Parámetros de configuración del módulo Buscador de patrones	74
6. Análisis experimental	77
6.1. Análisis del módulo Reconocimiento y seguimiento	78
6.1.1. Métricas	78
6.1.2. Ground truth	81
6.1.3. Plan de ejecución	81
6.1.4. Ejecución de los planes de ejecución	86
6.1.5. Resultados obtenidos	88
6.2. Análisis del módulo Buscador de patrones	100
6.2.1. Métricas	101
6.2.2. Resultados obtenidos	102
7. Conclusiones y trabajo futuro	105
7.1. Conclusiones	105
7.2. Trabajo futuro	106
7.2.1. Reemplazo del algoritmo alternativo de detección de personas	106
7.2.2. Manejo de variaciones en la iluminación de las imágenes	107
7.2.3. Adaptación automática de parámetros de configuración	107
7.2.4. Generador de patrones a reconocer desde una interfaz amigable	108
7.2.5. Reconocimiento de múltiples objetos de interés	108
A. Conceptos teóricos	109
A.1. Advanced Message Queuing Protocol	109
A.2. Sustracción de fondo	110
A.2.1. Mixture Of Gaussians 2	110
A.2.2. K-Nearest Neighbours	112
A.3. Detección de blobs	112
A.3.1. Simple Blob Detector	112
A.4. Reconocimiento de objetos de interés	113
A.4.1. Histogram of Oriented Gradients	114
A.4.2. Support Vector Machines	119
A.4.3. Non-Maximum Suppression	125
A.5. Histogramas	126
A.6. Seguimiento	126
A.6.1. Filtros de Kalman	127
A.6.2. Problema de asignación	128
A.6.3. Algorítmico Húngaro	129
Bibliografía	130

Agradecimientos

Queremos expresar nuestro agradecimiento a nuestro tutor, Sergio Nesmachnow, por la constante orientación y disposición a lo largo de los dos años académicos de este proyecto.

También nos gustaría agradecer a Santiago Iturriaga, quien nos proporcionó la estructura principal de este documento y quien estuvo presente de forma constante para orientar y asistir cuando fue necesario.

Agradecemos también, enormemente, a nuestras familias y amigos por el apoyo incondicional y continuo a lo largo de estos dos años académicos. Sin el apoyo de ellos y ellas, nada de esto hubiese sido posible.

A todos ellos, gracias.

Capítulo 1

Introducción

Bajo el pretexto de la necesidad de mayor seguridad, el número de cámaras de vigilancia instaladas en los últimos años se ha visto incrementado en gran número [1][2]. En su forma más básica, la función de estas radica en capturar imágenes de áreas específicas y, en el mejor de los casos, transmitir esas imágenes a bases de operaciones, en donde existen agentes humanos que evalúan las mismas.

Los agentes humanos (también llamados visualizadores) son los encargados de detectar y reportar hechos que, según su parecer, resultan anormales o sospechosos. En muchos casos, cada visualizador se encuentra encargado de monitorear más de veinte cámaras en simultáneo [3], lo que implica que deben ser capaces de dividir su atención para controlar todas las cámaras o, en su defecto, focalizar su atención en un grupo reducido de cámaras. La sobrecarga de trabajo a la que es sometido el visualizador deriva en el problema de que no se detecten ciertos eventos que pueden resultar de interés. Si a eso se le suma el hecho de que las personas suelen adolecer de fatiga y/o aburrimiento cuando realizan trabajos monótonos (como es visualizar cámaras) [4], el problema del déficit atencional por parte de los visualizadores puede verse incrementado. En ocasiones, se intenta mitigar el problema mediante la contratación de más personal para la visualización de las cámaras, hecho que redundaría en gastos operacionales mayores, por lo cual, en muchos casos, no resulta ser una solución viable.

Este proyecto plantea como solución para los problemas mencionados la implementación de algoritmos capaces de procesar las imágenes recibidas, con el fin de detectar aquellos eventos que los visualizadores buscan. A través de este trabajo se da un marco teórico y una prueba práctica fehaciente de la viabilidad de la creación de un sistema capaz de alertar a los visualizadores sobre potenciales eventos anómalos, donde los eventos son identificados de forma automática y en tiempo real. De esta forma, es posible reducir el número de visualizadores, así como también el riesgo de que los eventos no sean visualizados debido al alto flujo de información que debe ser procesado por los visualizadores de forma concurrente.

El sistema se basa en una arquitectura de filtros y tubos, separada en dos módulos principales. El primero de estos módulos se encarga de extraer características de las imágenes, con el fin de detectar, reconocer y realizar el seguimiento de las personas presentes en las mismas. El segundo módulo recibe los datos de posición de las personas a través del tiempo, y busca inferir si hay patrones de movimiento pre-cargados en el sistema que son cumplidos por las personas detectadas. La arquitectura de filtros y tubos fue utilizada por dos razones principales. La primera razón es que permite separar las

instancias de ejecución de los módulos de seguimiento de personas y de detección de patrones en instancias diferentes. La segunda razón radica en que la utilización de filtros permite realizar cambios en los algoritmos de los mismos, o directamente sustituirlos por otros, sin afectar la implementación de los filtros anteriores ni posteriores.

Si bien el foco se ha puesto en cámaras de vigilancia, el sistema puede ser fácilmente adaptable al reconocimiento de eventos de otra índole y sobre otro tipo de actores. Por ejemplo, puede utilizarse en el control del tránsito de vehículos terrestres o marítimos, en el estudio del comportamiento de especies animales, o en el análisis de rendimiento de equipos deportivos, entre otros.

En síntesis, esta investigación brinda como principales resultados el estudio del marco teórico necesario para la implementación de un sistema que brinde una solución al problema de la sobrecarga de los visualizadores, y una prueba práctica de que es posible la implementación de un sistema capaz de detectar patrones pre-establecidos que describen el movimiento de personas, a partir de valores de configuración ajustados al escenario en el que se utiliza.

Cabe destacar que este trabajo no tiene como objetivo el desarrollo de métodos de bajo nivel para el procesamiento de imágenes. En cambio, lo que se busca es realizar un relevamiento profundo de técnicas y métodos existentes que, combinados de forma adecuada, hacen posible la creación del sistema.

El resto de este documento está estructurado de la siguiente manera: el capítulo 2 presenta una introducción a los temas de estudio, el capítulo 3 define el problema a resolver, el capítulo 4 presenta la arquitectura y el diseño del sistema, el capítulo 5 brinda todos los detalles de la implementación, el capítulo 6 expone los análisis experimentales realizados junto a sus respectivos resultados, y finalmente el capítulo 7 presenta conclusiones y propone trabajo futuro. Además se agregó un apéndice, el cual brinda una base teórica sobre ciertos aspectos tratados en este trabajo; y un anexo que brinda información detallada adicional sobre el análisis experimental.

Capítulo 2

Procesamiento de imágenes y detección de patrones

En los últimos años se ha hecho evidente el incremento en la capacidad y rapidez de procesamiento a la vez que han disminuido los costos de los dispositivos de cálculo. Esto ha redundado en un interés creciente en el desarrollo de sistemas informáticos que sean capaces de emular varias de las habilidades humanas, entre ellas la visión, en tiempo real. Como consecuencia se han generado nuevas áreas de investigación y desarrollo, como el *procesamiento de imágenes y la detección de patrones*.

El procesamiento de imágenes es, en síntesis, la aplicación de un conjunto de técnicas con el objetivo de obtener información de las imágenes o transformarlas con el fin de mejorar o cambiar su apariencia (e.g., filtros). El procesamiento pueden hacerse tanto por medios ópticos como digitales.

Por otro lado, según Webb [5], la detección de patrones es el área de investigación que estudia la operación y el diseño de sistemas que reconocen patrones en los datos.

Este capítulo presenta una introducción a la temática de procesamiento de imágenes seguido de una introducción a la temática de detección de patrones y finaliza con ejemplos de aplicación de cada una de ellas juntas y por separado.

2.1. Procesamiento de imágenes

El área de procesamiento de imágenes se encuentra ligada a la de visión computacional, y aunque ambas tienen muchos puntos en común, el objetivo final de ambas áreas es diferente. de la Escalera Hueso [6] menciona que, por un lado, la visión computacional tiene por objetivo extraer características de una imagen para su posterior descripción e interpretación por parte de la computadora, mientras que el objetivo del procesamiento de imágenes es mejorar la calidad de las imágenes para su posterior utilización o interpretación.

Digitalmente, una imagen está compuesta por píxeles. Un píxel es la unidad mínima de la imagen, ocupa un área de la misma y alberga información sobre el color correspondiente a su posición (x, y) en la imagen. Esta información se representa generalmente mediante tres valores que corresponden a los tres canales de color: rojo, verde y azul (*RGB* por *red, green, blue* en inglés). En el caso de trabajar con imágenes monocromáticas (en escala de grises), los tres valores RGB coinciden, por lo que la información denota la intensidad en la posición (x, y) de la imagen. Por lo tanto, una imagen queda repre-

sentada por una matriz $A_{M \times N}$ en la que a_{ij} representa el píxel en la posición (i, j) de la imagen. La representación matricial permite aplicar métodos matemáticos a las imágenes de forma sencilla y práctica.

La Figura 2.1 muestra las etapas a las que generalmente una imagen es sometida al ser procesada. Las subsecciones siguientes brindan una breve descripción de estas etapas.

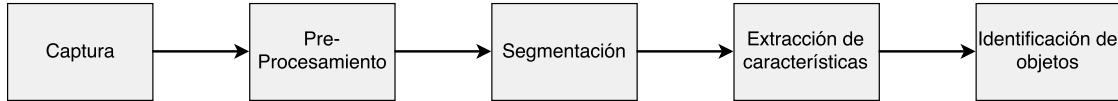


Figura 2.1: Etapas del procesamiento de imágenes

2.1.1. Captura

La captura de la imagen, también llamada adquisición, es la etapa encargada de obtener las imágenes en crudo. Las imágenes pueden ser adquiridas a través de una cámara, digitalizando fotografías, etc. Ramírez [7] recalca en su trabajo que durante la etapa de captura los elementos encargados de capturar la luz para crear la representación, suelen agregar, en mayor o menor medida, una cuota de ruido.

2.1.2. Pre-procesamiento

Las imágenes captadas por diferentes dispositivos (cámaras, escáneres, etc.), pueden presentar distintos tipos de degradaciones. Las degradaciones pueden ocurrir por diversos motivos, por ejemplo, por ruido en el sensor, borrosidad, etc. Es común que la imagen aparezca borrosa cuando la cámara no se encuentra correctamente enfocada, o cuando se produce un movimiento relativo entre el objeto o escena y la cámara, durante el tiempo que dura la exposición. La etapa de pre-procesamiento se encarga de aplicar métodos que permiten eliminar o reducir elementos que no son de interés (como el ruido, o la borrosidad) e intenta mejorar ciertas características de la imagen original (definición de contornos, brillo, etc.) haciendo uso de herramientas matemáticas [7].

2.1.3. Segmentación

La etapa de segmentación es el primer paso para la extracción de información de las imágenes [8]. Las imágenes suelen estar constituidas por regiones que presentan características homogéneas (e.g., niveles de gris, textura, etc.). Por lo general, las regiones homogéneas se corresponden con objetos de la imagen y esta etapa se encarga de descomponer la imagen en sus partes constituyentes. En otras palabras, la segmentación se encarga de separar la imagen en objetos de interés y fondo, basándose en su contorno, conectividad, o de cierto conjunto de características basadas en los píxeles de la imagen (tonos de gris, textura, los momentos, la magnitud del gradiente, etc.). La imagen queda entonces dividida en varias regiones disjuntas. Se puede decir que esta etapa intenta distinguir si un píxel pertenece a la región de un objeto, produciendo como resultado una imagen binaria.

Es posible distinguir entre segmentación completa y segmentación parcial. La segmentación completa se da cuando las regiones disjuntas de la imagen binaria corresponden directamente a objetos, lo cual es alcanzado únicamente si se dispone de un conocimiento específico del dominio de la escena (e.g., cuando los objetos de la imagen son caracteres

alfanuméricicos). La segmentación parcial se da cuando las regiones no se corresponden directamente con objetos presentes en la imagen.

Martín [9] menciona que los algoritmos de segmentación se centran en dos propiedades básicas de la imagen: discontinuidad y similitud. La discontinuidad busca cambios bruscos en los niveles de gris de los píxeles dentro de un entorno e intenta dividir la imagen a partir de ellos, detectando puntos aislados, líneas y aristas (bordes). La similitud (o similaridad) busca semejanzas en los niveles de gris de los píxeles dentro de un entorno, lo cual permite construir regiones aplicando técnicas como la umbralización, crecimiento y división, y fusión. Muñoz Pérez [8] agrega una tercera propiedad que llama *conectividad*. La *conectividad* desempeña un papel importante en la segmentación ya que los píxeles conectados son los que determinan las regiones conexas.

Los algoritmos de segmentación de imágenes monocromáticas pueden ser agrupados en cuatro categorías diferentes:

- Métodos basados en píxeles: se clasifican en métodos locales, basados en las propiedades de los píxeles, y métodos globales, basados en la información global obtenida.
- Métodos basados en bordes.
- Métodos basados en regiones, que utilizan nociones de homogeneidad y proximidad geométrica.
- Métodos basados en modelos.

2.1.4. Extracción de características

La extracción de características busca hallar, seleccionar y extraer aquellas características relevantes para lograr la identificación de los objetos de interés. Gonzalez y Woods [10] mencionan que es necesario que las características halladas cumplan ciertas propiedades:

- Robustez: la extracción de características debe ser insensible al ruido.
- Discriminabilidad: las características deben servir para distinguir objetos de clases distintas.
- Invarianza: las características deben ser invariantes ante la aplicación de distintas transformaciones (e.g., la traslación, la rotación, el escalado, etc.).

En esta etapa se busca un esquema de representación [11]. Un esquema de representación es una estructura de datos compacta que representa la información obtenida al analizar la forma geométrica de los elementos hallados en la etapa anterior. Los esquemas de representación se pueden clasificar como esquemas de representación externa y esquemas de representación interna, los cuales se describen a continuación.

Esquemas de representación externa

Los esquemas de representación externa utilizan el contorno de los objetos y sus rasgos característicos (e.g., códigos de cadena, descriptores de Fourier, aproximaciones poligonales).

Códigos de cadena La frontera del objeto es representada mediante un código incremental. Los códigos incrementales representan segmentos de línea en una retícula mediante un número de orientaciones limitadas (cuatro u ocho). Se recorre la frontera en el sentido de las agujas del reloj a partir de un punto inicial, indicando la dirección que sigue la frontera. Esta técnica presenta ciertas ventajas frente a la representación matricial de un objeto binario, como pueden ser:

- El código de cadena que se obtiene es invariante frente a la traslación, lo cual facilita la comparación de objetos.
- Se pueden obtener ciertas características del objeto a partir del contorno (perímetro, área del objeto, etc.) de forma más eficiente.
- Permite obtener una representación más compacta del objeto, ya que cada cadena puede ser codificada utilizando únicamente dos bits (para entornos de 4 vecinos), en lugar de las coordenadas (x, y) de cada píxel del contorno.

Sin embargo, la técnica de códigos de cadena, comparada con la representación matricial de un objeto, presenta las siguientes desventajas:

- Las cadenas que se obtienen suelen ser demasiado largas.
- Cualquier perturbación o ruido en la imagen produce segmentos erróneos, por lo que la cadena no describirá correctamente el contorno del objeto.

La Figura 2.2 presenta un ejemplo de código de cadena hallado para un objeto en una retícula en un entorno de cuatro vecinos. En ese caso, el código de cadena obtenido es 011100300332232221.

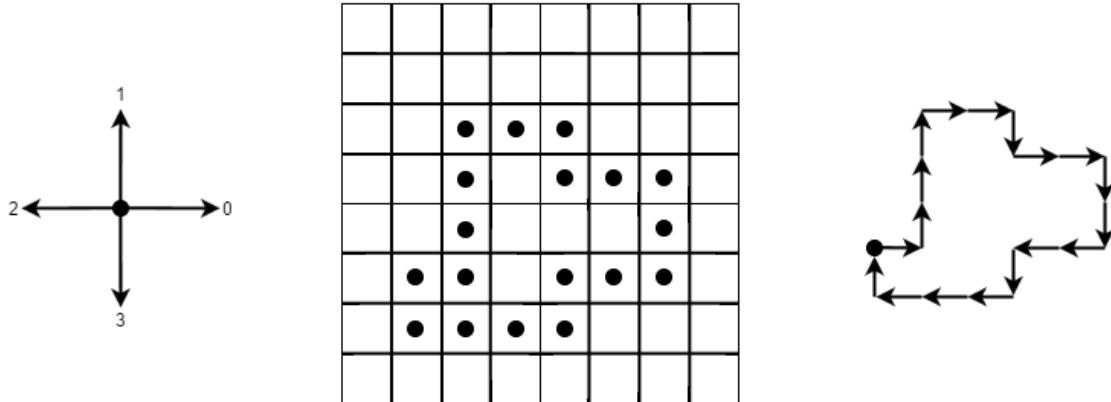


Figura 2.2: Ejemplo de código de cadena para el contorno de una figura

Descriptores de Fourier Cuando el contorno de un objeto está dado por una curva cerrada, puede ser representado mediante descriptores de Fourier, utilizando series de Fourier junto a una parametrización adecuada. Proakis y Manolakis [12] indican que los descriptores de Fourier tienen como característica la invarianza frente a transformaciones geométricas y la tolerancia ante ruido. Además, los descriptores de Fourier presentan una buena parametrización con pocos términos.

Formalmente, los descriptores de Fourier se definen dados un par de funciones periódicas $(x(t), y(t))$ un par de funciones periódicas, donde el parámetro t describe la longitud del camino de la línea de contorno calculada desde el punto de partida, se comienza seleccionando N puntos, los cuales deben ser equidistantes del contorno, y a partir de esos valores se obtiene un vector complejo $z = x + iy$ donde

$$x = (x(0), x(1), \dots, x(N - 1))$$

$$y = (y(0), y(1), \dots, y(N - 1))$$

Dado el vector complejo, se encuentra la transformada de Fourier discreta, que queda representada por la expresión

$$Z(u) = \frac{1}{N} \sum_{n=0}^{N-1} z(n) \exp\left(-\frac{2\pi n u i}{N}\right), \text{ siendo } u = 0, 1, 2, \dots, N - 1$$

Los valores definidos por $Z(u)$ son conocidos como descriptores de Fourier cartesianos del contorno, o simplemente *descriptores de Fourier*. A partir de estos valores, es posible realizar la reconstrucción del contorno del objeto aplicando la transformación inversa

$$z(n) = \frac{1}{N} \sum_{k=0}^{N-1} Z(k) \exp\left(\frac{2\pi n k i}{N}\right), \text{ siendo } n = 0, 1, 2, \dots, N - 1$$

El coeficiente $Z(0)$ representa al punto medio del objeto (centro de gravedad), denominado *centroide* y cuyo valor queda representado por

$$Z(0) = \frac{1}{N} \sum_{k=0}^{N-1} x(n) + i \frac{1}{N} \sum_{n=0}^{N-1} y(n)$$

El hecho de trabajar con la transformada de Fourier discreta implica que los coeficientes de Fourier $Z(k)$ representan las variaciones pequeñas, de las tendencias del contorno, cuando los valores de k son pequeños, mientras que cuando los valores de k son grandes, los coeficientes de Fourier representan las variaciones grandes. Formalmente, los componentes de alta frecuencia tienen en cuenta los detalles más finos del contorno, a la vez que los de baja frecuencia determinan la forma global del contorno del objeto.

Aproximaciones poligonales Se describe la frontera del objeto a partir de un polígono que la aproxima, bajo un cierto error tolerado. A partir de una curva compuesta por tramos lineales, se construye un polígono cuyos vértices representan el contorno del objeto. Ho y Chen [13] afirman que la aproximación poligonal es un método popular para la representación de formas, que provee buenas representaciones en dos dimensiones a diferentes resoluciones. La Figura 2.3 muestra los pasos necesarios para encontrar un polígono cuya forma aproxime el contorno del objeto. El algoritmo comienza detectando el eje de mayor elongación del objeto, para luego continuar añadiendo vértices, hasta obtener un polígono con cierta precisión.

Asumiendo que se tiene una curva que va del punto x_1 al punto x_N , sean x_1, x_2, \dots, x_N las coordenadas de los $N - 2$ puntos intermedios de la misma y además d_i , el punto del segmento $[x_1, x_N]$ más próximo al punto x_i , con $i = 2, 3, \dots, N - 1$, entonces la expresión

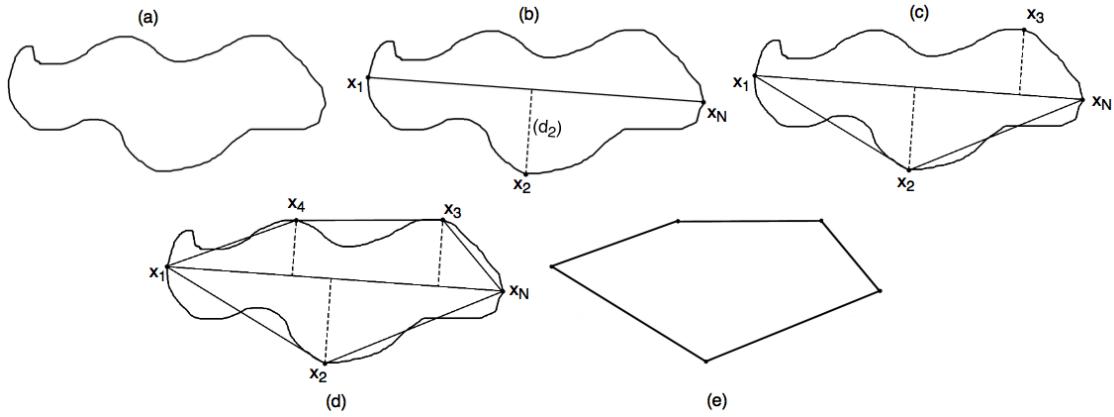


Figura 2.3: Pasos para calcular aproximaciones poligonales por partición sucesiva

$|x_i - d_i|$ representa el error de aproximación de la curva por el segmento lineal $[x_1, x_N]$, correspondiente al píxel x_i , donde $i = 2, 3, \dots, N - 1$ (Figura 2.3.b).

Para medir la bondad del ajuste se utiliza un cierto criterio. Por ejemplo, se puede utilizar el criterio de mínimo error cuadrático medio donde se minimiza el valor dado por la expresión

$$\sum_{i=2}^{N-1} |x_i - d_i|^2$$

La aproximación poligonal consiste entonces en determinar los vértices del polígono de forma tal que el error total resulte mínimo. Encontrar la solución óptima a este problema suele ser costoso en términos computacionales, por lo que, en general, se suelen utilizar otras técnicas más rápidas y eficientes que aportan soluciones cercanas a la óptima. Un ejemplo de estas técnicas son las de partición que consisten en realizar divisiones de forma recurrente de los trozos de curva representados por un segmento, de manera tal que la medida local del error sea mínima. El proceso acaba cuando el error máximo por píxel es menor que cierto valor de tolerancia prefijado. Los puntos x_1 y x_N pueden ser los puntos más alejados del contorno y dividen el mismo en dos partes, aplicando el algoritmo en cada una de ellas.

Se puede apreciar en las Figuras 2.3(c) y 2.3(d) que, en cada paso, el error por píxel es igual o menor que el error en el paso anterior. A medida que se van agregando nuevos píxeles, se mejora la aproximación.

Esquemas de representación interna

Los esquemas de representación interna describen la región que ocupa el objeto encontrado en la imagen binaria. Se utilizan características geométricas y topológicas de las regiones que ayudan a identificar y reconocer los objetos de la imagen.

El área de un objeto es un parámetro geométrico que está dado por el número de píxeles que lo componen y es fácil de calcular a partir de su código de cadena. El centro de gravedad (o *centroide*) de una región es otro parámetro geométrico y viene determinado por el conjunto de píxeles $\{(x_i, y_i), i = 1, 2, \dots, N\}$, siendo (\bar{x}, \bar{y}) el punto definido por las

expresiones

$$\bar{x} = \frac{\sum_{i=1}^N x_i}{N}, \quad \bar{y} = \frac{\sum_{i=1}^N y_i}{N}$$

Cabe destacar que no es necesario que el centro de gravedad de una región coincida con el de su contorno.

La circularidad (o compacidad) es un parámetro topológico del objeto, no depende de su tamaño y se obtiene a partir de la expresión

$$c = \frac{A}{p^2}$$

donde A es el área y p el perímetro del objeto. Los valores pequeños de *circularidad* se corresponden con objetos alargados. La *circularidad* suele normalizarse al intervalo al dividirla entre $\frac{1}{4\pi}$. Con la normalización se busca garantizar que el valor que se asigne al círculo sea uno y a su vez sea menor a cualquier otra figura geométrica. Se utiliza el valor $\frac{1}{4\pi}$ ya que es el máximo de circularidad que se puede obtener, correspondiente a la circularidad de un círculo, como se puede apreciar en la ecuación 2.1.

$$c = \frac{A}{p^2} = \frac{\pi r^2}{(2\pi r)^2} = \frac{1}{4\pi} \quad (2.1)$$

Otro de los parámetros topológicos es la rectangularidad. La rectangularidad es adimensional y se obtiene del cociente entre el área del objeto y el área de su rectángulo base (el rectángulo de menor área que envuelve completamente al objeto). La expresión de la rectangularidad es

$$r = \frac{A}{ab}$$

donde A es al área del objeto, y a y b son las medidas de los lados del rectángulo base, de mayor y menor longitud respectivamente.

Además de la rectangularidad, existen dos características topológicas muy importantes llamadas conectividad y agujeros. Una imagen segmentada se compone de regiones que tienen componentes conexas (regiones tales que dos puntos cualesquiera de ellas pueden ser unidos por una curva, la cual se encuentra contenida en esas regiones) que configuran los objetos. Un agujero es, en cambio, una región de la imagen que se encuentra completamente encerrada por una componente conexa de la misma imagen. Estas propiedades son de utilidad para definir el *número de Euler* de una imagen, obtenido a partir de la expresión

$$E = C - H$$

donde C es la cantidad de componentes conexas de la imagen y H la cantidad de agujeros de la misma. El número de Euler es invariante frente a traslaciones, rotaciones y cambios de escala de la imagen.

2.1.5. Identificación de objetos

Con el fin de categorizar el conjunto de características extraídas en la etapa de extracción de características, se utilizan modelos de toma de decisiones. Una posibilidad es utilizar algoritmos de clasificación supervisados, que asignan a un conjunto de características una clase previamente definida. Uno de estos algoritmos es el *k-Nearest Neighbors* (*k-NN*), que se presenta a continuación.

k-Nearest Neighbors

El método de clasificación de los k vecinos más cercanos es, según Santillán y Cruz [14], uno de los más utilizados en el reconocimiento de formas, debido a que se basa en una idea simple e intuitiva, que es sencilla de implementar. La idea básica del algoritmo es que una nueva entrada se clasifica como la clase más frecuente a la que pertenecen sus k vecinos más cercanos, Moujahid et al. [15]. Esto es, dado un vector \vec{x} de dimensión n a clasificar, y sea M una base de datos de referencia, construida a partir de N vectores de dimensión n y C_i la clase a la cual pertenece cada uno de los vectores de M ; el clasificador KNN se basa en la aproximación local de la densidad de probabilidad del vector \vec{x} , a partir de los k vecinos, pertenecientes a M , que se encuentran más cerca. El Algoritmo 1 presenta un pseudocódigo del algoritmo k -NN.

Como se puede apreciar en el Algoritmo 1, el paso de seleccionar aquella clase más frecuente de entre todas las clases de los vecinos más cercanos a \vec{x} , puede generar un empate. En el caso de un empate, es necesario utilizar criterios para resolverlos. Uno de los criterios de desempate utilizados indica que se debe escoger la clase con mayor probabilidad *a priori* y en el caso de que las probabilidades *a priori* coincidan, se selecciona una clase al azar del conjunto de clases en disputa.

Algoritmo 1 Algoritmo k vecinos más cercanos (k -NN)

```

1:  $k \leftarrow elegirCantidadVecinosMasCercanos()$ 
2:  $neighbors\_distances \leftarrow emptyList()$ 
3: for each  $v_M^i$  in  $M$  do
4:    $dist \leftarrow getDistance(\vec{x}, v_M^i)$ 
5:    $neighbors\_distances.insertOrdered(v_M^i, dist)$ 
6: end for
7:  $nearest\_neighbors \leftarrow getKNearestNeighbors(neighbors\_distances, k)$ 
8:  $class \leftarrow selectMostFrequentClass(nearest\_neighbors)$ 
9:  $\vec{x}.setClass(class)$ 
```

Para el cálculo de las distancias, se puede utilizar cualquier tipo de distancia que sea conveniente para el problema en cuestión (e.g., la distancia euclídea, distancia Manhattan, distancia del máximo, etc). La elección del tipo de distancia a utilizar, generalmente dependerá de factores como el tiempo de ejecución, el costo computacional o el desempeño, entre otros. Algunas variantes del algoritmo k-NN calculan las distancias asignando ciertos pesos a los elementos v_M^i , lo cual permite que se tomen en cuenta aquellos elementos que pueden ser más relevantes.

Uno de los puntos más críticos en este algoritmo es la selección del parámetro k . Por lo general, el valor de k es seleccionado de forma heurística [14]. En la práctica, se sugiere $k = \sqrt{N}$. Otra alternativa suele ser buscar el valor de k variándolo desde 1 hasta un cierto valor, luego interpretar los datos y seleccionar el valor de k que presenta mejores resultados.

2.2. Detección de patrones

Un sistema de detección de patrones consiste en particionar un universo de estudio en clases, de manera que el sistema sea capaz de asignar un elemento x a alguna de las

clases, a partir de un conjunto de características que presenta el modelo del elemento. A cada clase se le asignan elementos que cumplen características semejantes entre ellos; estas características cumplen con lo que se denomina *patrón* de características.

Cuando los patrones encontrados son, a priori, desconocidos, se dice que se trata de reconocimiento o descubrimiento de patrones (*pattern recognition*), mientras que si los patrones son preestablecidos se le llama emparejamiento o *matching* de patrones. El sistema presentado cuenta con *matching* de patrones en su segunda etapa de procesamiento.

La Figura 2.4 muestra las etapas que, generalmente, componen el proceso de detección de patrones junto a la entrada y salida de cada una. Las etapas son segmentación, extracción de características y clasificación. En ocasiones existe una etapa previa a las tres presentadas, llamada preprocessamiento, donde se opera con los datos de forma tal de reducir sus dimensiones, normalizarlos o eliminar el ruido presente a la hora de la obtención.

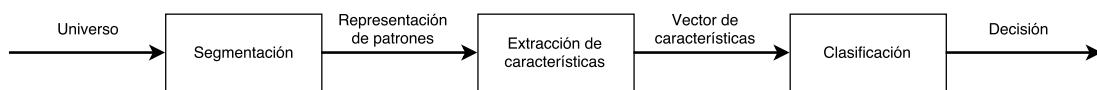


Figura 2.4: Esquema general de un sistema de detección de patrones.

Fuente: Basada en *Statistical pattern recognition*, Webb [5].

2.2.1. Segmentación

Para el caso del sistema que se presenta en este informe, en la etapa de segmentación se reciben las imágenes y se extraen los objetos de interés que presenta, el resto de la imagen se desecha. La etapa de segmentación es similar a la etapa de sistemas de procesamiento de imagen, presentada en la sección 2.1.3.

2.2.2. Extracción y selección de características

La etapa de extracción y selección de características toma como datos de entrada los resultados de la etapa de segmentación. Esta etapa analiza los objetos dados como entrada, con el fin de extraer características relevantes de éstos. Las características extraídas pueden ser cuantitativas (e.g., velocidad, distancia, etc.) o cualitativas (e.g., profesión, sexo, etc.) y son utilizadas para construir un vector llamado vector de características.

La extracción y selección de características tiene como fin:

- Extraer información relevante de los objetos que permita una mejor discriminación entre elementos de una clase y las demás
- Eliminar información redundante o irrelevante
- Reducir la dimensión del problema

La extracción y selección de características y la clasificación se encuentran fuertemente ligadas ya que el desempeño del clasificador dependerá, en gran medida, de las características extraídas.

Extracción de características

El objetivo de la etapa de extracción de características es aplicar transformaciones y combinaciones sobre, y entre, los elementos del conjunto de características original. De esta forma, se obtiene un nuevo conjunto en el que los elementos son características que resultan de mayor utilidad para etapas posteriores. No necesariamente el conjunto resultante es un subconjunto del original.

Existen varios métodos de extracción de características y cada uno presenta diferentes propiedades (e.g., no linealidad, linealidad). A continuación se presentan brevemente algunos métodos de extracción de características, expuestos en la revisión sobre la temática elaborada por Jain et al. [16]:

- *Principal Component Analysis* (PCA) es una método lineal no supervisado utilizando, típicamente, para reducir la dimensión de un conjunto de datos. Como resultado de PCA, se obtiene un conjunto características formadas a partir de los valores y vectores propios de la matriz de covarianza asosiada a los datos. El transformado de los patrones resulta en un nuevo conjunto no correlacionado. La transformación lineal está dada por $Y = XH$, donde X corresponde a la matriz de patrones, Y es la matriz derivada de patrones y H la matriz de transformaciones lineales cuyas columnas se corresponden con los vectores propios. PCA reduce el conjunto original a un conjunto menor, formado por los valores más expresivos (vectores propios cuyos valores propios asociados son los mayores).
- *Linear Discriminant Analysis* (LDA) es un método lineal, no supervisado (i.e., utiliza la información de la clase asociada a cada patrón), cuyo objetivo es extraer las características con mayor discriminabilidad. Con el fin de encontrar las características con mayor discriminabilidad, utiliza un criterio que maximiza la separación entre patrones de diferentes clases y minimiza la separación de patrones de una misma clase.
- *Kernel Principal Component Analysis* (Kernel PCA) es un método que se relaciona directamente con el método PCA. La idea de Kernel PCA es, en primero lugar, hacer un mapeo de los datos de entrada a un nuevo espacio de características F , generalmente utilizando una función no lineal. Luego, Kernel PCA toma el espacio generado F y le aplica el método PCA, que como resultado nos da un espacio de características que ahora es linealmente separable. Para evitar costo computacional, en la etapa de mapeo de los datos de entrada, Kernel PCA utiliza únicamente Mercer kernels, que pueden ser descompuestos en productos escalares. Un ejemplo de este tipo de técnicas es *Kernel trick*, explicada en la sección A.4.2 del apéndice A.

Selección de características

El principal objetivo de la etapa de selección de características es determinar un conjunto de descriptores que sea óptimo para el problema tratado. Esto suele llevarse a cabo mediante técnicas estadísticas, y en ocasiones puede ser necesario contar con un conocimiento más profundo acerca de la naturaleza del problema. A través de esta etapa se busca obtener un subconjunto de características a partir del conjunto original, de forma de optimizar cierta función objetivo preestablecida.

Existe una variedad de métodos de selección de características y en todos ellos se pueden identificar los siguientes componentes, según Ramírez Osuna [17]:

- Al menos un criterio de evaluación. El criterio de evaluación resulta útil para comparar el poder de clasificación de los distintos subconjuntos de características.
- Un procedimiento o algoritmo de búsqueda.
- Un criterio de parada.

Los métodos de selección pueden ser clasificados en estadísticos o neuronales. Los métodos estadísticos se subdividen en óptimos o sub-óptimos, siendo esta última categoría dividida según la cantidad de soluciones posibles (una o más). A continuación se presenta una breve descripción de los tipos de métodos de clasificación.

Métodos de selección óptimos Los métodos más utilizados dentro de esta categoría son los de búsqueda exhaustiva y los de ramificación y poda (*branch and bound*), caracterizados ambos por su elevado costo computacional [17].

Búsqueda exhaustiva Se estudian todas las combinaciones posibles de las características, evaluando un total de $\frac{n!}{d!(n-d)!}$ subconjuntos, siendo d el tamaño deseado para los subconjuntos y n el número de características que presenta el conjunto original. La naturaleza del método hace que la cantidad de posibilidades crezca de manera exponencial, lo que redundaría en una búsqueda poco práctica, incluso para valores moderados de n y d .

Branch and bound A diferencia de la búsqueda exhaustiva, branch and bound es un algoritmo que no evalúa todos los posibles subconjuntos de características y por tanto, es computacionalmente más eficiente. Sin embargo, es necesario que se cumpla una condición: la función de selección de características F debe ser monótona, o lo que es lo mismo, se debe cumplir que al añadir nuevas características al subconjunto, el valor de F no debe disminuir. El algoritmo presenta un enfoque *top-down* ya que se comienza con n características y se va construyendo el árbol, eliminando de forma sucesiva diferentes características.

Métodos de selección sub-óptimos con una solución Son métodos que parten de un conjunto inicial de características que se va modificando de forma iterativa a lo largo de la ejecución, hasta alcanzar el criterio de parada. El subconjunto que presente al momento de alcanzar el criterio de parada es el seleccionado.

Estos métodos son separados en dos categorías:

- Métodos *Sequential Forward Selection* (SFS). Este tipo de métodos parten de un conjunto compuesto por la mejor característica individual (aquella que presenta mayor grado discriminativo) y, en cada iteración, continúan agregando nuevas características. Presentan el inconveniente de que las características seleccionadas ya no podrán ser descartadas.
- Métodos *Sequential Backward Selection* (SBS). Los métodos SBS se inician con un conjunto que incluye todas las características y, en cada iteración, eliminan una del

conjunto. Presentan el inconveniente inverso a SFS, las características no pueden volver a ser seleccionadas luego de descartadas en una iteración.

Los métodos sub-óptimos presentan eficiencia en términos computacionales pero adolecen el problema de terminar brindando el subconjunto de las mejores m características individuales, aunque éste puede no ser el mejor subconjunto (el óptimo), como demuestran Cover y Van Campenhout [18].

Existen variantes para los métodos de selección secuencial (SFS y SBS), que permiten agregar o quitar características (*Selección secuencial generalizada, +l-r*), o regresar para considerar conjuntos previos (*Best First*).

Métodos de selección con redes neuronales Los métodos de poda de nodos son parte de la categoría de métodos de selección con redes neuronales. Estos métodos tienen como objetivo seleccionar y clasificar las características al mismo tiempo y se caracterizan por presentar un elevado costo computacional. Estos métodos eliminan aquellos nodos (características) menos importantes, con el fin de reducir la complejidad. El algoritmo entrena la red neuronal y luego elimina el nodo, finalizando cuando se alcanza el nivel de compromiso deseado entre el error de clasificación y la complejidad de la red.

2.2.3. Clasificación

En la etapa de clasificación se procesa el vector de características con el fin de tomar la decisión más acertada, i.e., se busca asignar al vector de características la clase más adecuada. Por lo general, la elección de un clasificador depende de la naturaleza del problema.

Como se mencionó en la sección 2.2.2, el desempeño de un clasificador depende de las características extraídas y, en particular, de la relación entre el número de muestras, el número de características extraídas y la complejidad del propio clasificador.

Dado un vector de características $x \in \mathbb{R}^d$, tal que $x = (x_1, x_2, \dots, x_d)$, el objetivo de un clasificador es determinar a cuál de las clases del conjunto $C = \{w_1, w_2, \dots, w_c\}$ pertenece el vector de características x . El problema se traduce a encontrar una función D que cumpla

$$\begin{aligned} D : \mathbb{R}^d &\rightarrow C \\ D(x) &= w_i, \text{ donde } i = 1, 2, \dots, c \end{aligned}$$

Para esta etapa se presentan dos grupos principales de clasificadores, *supervisados* y *no supervisados* [5]. Corso menciona que la principal diferencia entre ellos radica en el enfoque utilizado para resolver el problema de la clasificación [19].

Clasificación supervisada

En la clasificación supervisada se parte de un conjunto de elementos, denominado *conjunto de datos de entrenamiento* o *conjunto de aprendizaje*. Los elementos de este conjunto presentan la particularidad de que la clase a la cual pertenecen, es conocida de antemano [19]. Carrasco y Martínez enumeran algunos métodos de clasificación supervisada, como son *superficies de separación*, *funciones discriminantes*, *k-vecinos más cercanos* y *redes neuronales* [20].

Clasificación no supervisada

La clasificación no supervisada trata la clasificación como el descubrimiento de las clases de un problema dado. Dicho de otra forma, se tiene un conjunto de elementos descritos por un número de características, de los cuales no se conoce la clase a la que pertenecen [19]. Este tipo de clasificadores pueden ser restringidos o libres. Los clasificadores restringidos implican que el número de clases a la que se estructura la muestra de datos sea previamente definido, por el contrario, los clasificadores libres, implican que la cantidad de clases a la que se estructura la muestra de datos dependa exclusivamente de los datos.

Existen tres estrategias para abordar los clasificación no supervisados [20]:

- Jerárquica. Puede ser aglomerativa, en donde se parte de grupos formados solamente por un elemento de la muestra, para luego agruparlos poco a poco, hasta obtener un solo agrupamiento; o divisiva, en donde se parte del conjunto completo de datos de la muestra, y se va dividiendo hasta obtener únicamente grupos formados por un solo objeto de la muestra.
- Reagrupamiento. Suponiendo que se conoce la cantidad de clases que se quiere obtener, se realiza una distribución inicial de los elementos de la muestra para obtener la cantidad de agrupamientos deseada, para luego, de forma iterativa, realiza reorganizaciones hasta obtener un agrupamiento que cumpla con los criterios buscados.
- Basado en grafos. Se busca generar un grafo que contenga toda la información de semejanzas para los objetos de la muestra, para luego generar un cubrimiento del grafo.

Algunos métodos de clasificación no supervisada son *Simple Link*, *ISODATA* y *k-means*, entre otros.

Reconocimiento estadístico de patrones

El reconocimiento estadístico de patrones es un enfoque basado en la teoría de la probabilidad y la estadística. Supone que se tiene un conjunto de mediciones numéricas, con sus distribuciones de probabilidad conocidas, a partir de las cuales se calcula la probabilidad de que un vector de características pertenezca a una clase. El reconocimiento estadístico evita la propagación de errores de forma parcial o completa.

2.3. Aplicaciones del procesamiento de imágenes y la detección de patrones

En esta sección se presentan algunas aplicaciones prácticas de las técnicas de procesamiento digital de imágenes y la detección de patrones. En algunos casos se puede encontrar estas técnicas trabajando en conjunto.

Aplicaciones en medicina

Calot [21] menciona la existencia de numerosos trabajos en el área de procesamiento de imágenes aplicados a la medicina, desde la detección de anomalías cardíacas, pasando

por la localización automática de tumores o calcificaciones (a partir de mamografías), hasta la detección de tipos de tejidos pulmonares.

Strauss et al. [22] presentan algoritmos semiautomáticos capaces de obtener un contorno del ventrículo izquierdo (cavidad inferior del corazón).

La automatización en la localización de tumores a partir de imágenes comenzó a ser tratada ya en el año 1973 por Ballard y Sklansky [23]. Los autores tratan el análisis de imágenes médicas como un problema de procesamiento de información en una única imagen, aplicando técnicas basadas en el reconocimiento de patrones y procesamiento de imágenes, entre otras.

Chan et al. [24] presentan en el año 1987 uno de los primeros sistemas CAD completamente automático con aplicación en mamografías de rayos X. El sistema permitió realizar una variedad de operaciones con umbrales de tolerancia sobre mamografías digitalizadas, aplicando luego funciones discriminantes lineales, con el fin de intentar una clasificación automática de tejido normal y clasificaciones [25].

Hacia finales de la década de 1990, se presentó un nuevo sistema CAD, desarrollado por Uppaluri et al. [26], que permitió detectar seis tipos distintos de patrones de tejidos pulmonares, a partir de las características de sus texturas. Se evaluaron regiones de interés cuadradas de 961 píxeles. Para cada una de esas regiones un subconjunto óptimo de texturas fue evaluado. Se utilizó un clasificador Bayesiano que fue entrenado para utilizar ese subconjunto óptimo, logrando así reconocer los tipos de tejidos.

Aplicaciones en inteligencia artificial y robótica

Prieto et al. [27] desarrollaron un sistema de seguimiento en tiempo real, con el fin de ser utilizado en un brazo robótico, capaz de funcionar como asistente de aplicaciones médicas. La característica de seguimiento inteligente permitió, por ejemplo, una mejor maniobrabilidad de un laparoscopio. El desarrollo del sistema dio lugar a la posibilidad de sustituir la persona que asiste al cirujano manejando el laparoscopio, ya que en casos de operaciones largas, el asistente puede sufrir desgaste y perder capacidad de enfocar el punto de interés. La utilización de un brazo robótico capaz de seguir la herramienta del cirujano, elimina el factor de desgaste del asistente, ya que permite mostrar en el monitor el área de trabajo en todo momento.

El campo de la automatización industrial utiliza, muchas veces, el procesamiento de imágenes en combinación con la detección de patrones. Por ejemplo, en la supervisión automática de procesos, logrando detectar fallas o averías, así como también en el control de calidad en las líneas de producción.

Aplicaciones en seguridad

El campo de la seguridad es, probablemente, uno de los que más utiliza las técnicas de procesamiento de imágenes y detección de patrones. Liu et al. [28] mencionan que se han desarrollado analíticas de vídeo inteligentes combinando las técnicas antes mencionadas, con el fin de automatizar (y en algunos casos ejecutar en tiempo real) tareas como detección y reconocimiento de rostros, reconocimiento de huellas dactilares, detección de movimiento, detección y seguimiento de objetos, entre otras tantas.

A modo de ejemplo, al día de hoy existen sistemas capaces de procesar vídeo en tiempo real, con el fin de detectar y reconocer características biométricas (corporales o de comportamiento) de las personas. Como sugiere Abate et al. [29], las características

biométricas más comunes son las huellas dactilares y el iris, aunque menciona también que en los últimos años han sido estudiadas muchas otras características, como pueden ser la geometría de la palma de la mano, la voz o el rostro. Esta última presenta una ventaja frente a las huellas dactilares y el iris. El reconocimiento a través del iris es muy preciso, pero a la vez muy costoso de implementar y algo invasivo para las personas, mientras que el reconocimiento por huellas dactilares resulta menos invasivo, pero depende mucho de la colaboración de las personas. Es por eso que la detección y reconocimiento de rostros parece una forma adecuada para brindar seguridad, detectando personas que puedan ser no deseadas.

El reconocimiento de rostros suele ser un problema 1:N, es decir que se compara un rostro contra todos los rostros almacenados en una base de datos, buscando determinar la identidad de ese rostro. Para realizar esa comparación se aplican técnicas de procesamiento de imágenes, con el fin de extraer un conjunto de características del rostro, las cuales luego son comparadas con los conjuntos de características almacenados en la base de datos (de estos conjuntos se conoce la identidad).

Los sistemas de reconocimiento facial, suelen computar un puntaje para cada uno de los conjuntos de características de la base de datos. Ese puntaje se obtiene a partir de la comparación de dichos conjuntos con el conjunto de características del rostro que se busca identificar. Así, los puntajes son ordenados de manera descendente, de forma tal que se dispara una alerta con los rostros correspondientes a los conjuntos de características cuyos puntajes superan ese umbral.

Capítulo 3

Detección de patrones de movimiento anómalos en videovigilancia

Este capítulo presenta en detalle el problema que aborda el proyecto, junto a una planificación pautada y el relevamiento de trabajos relacionados.

3.1. Especificación del problema

La creciente necesidad de mecanismos innovadores que brinden seguridad ha motivado a las personas, la industria y los gobiernos a instalar un gran número de cámaras de vigilancia. Estas cámaras han sido instaladas tanto en ambientes internos como externos, públicos y privados. Un gran número de estas instalaciones cuentan únicamente con sistemas de almacenamiento de las imágenes por un tiempo limitado, mientras que algunas instituciones han decidido instalar centros de gestión y monitoreo de las imágenes generadas.

Debido a la gran cantidad de datos generados y a la complejidad de su procesamiento, la tarea en los centros de gestión y monitoreo se convierte en una actividad tediosa y que, en algún punto, debe ser realizada por un operador humano. El valor de estos sistemas no consta únicamente en tener un registro de lo sucedido, sino en poder dar respuesta ante hechos delictivos o accidentes eventuales en el momento que suceden. Para poder tomar acción en el momento que ocurren los hechos, es necesario poder monitorear de forma correcta todas las cámaras, lo que se traduce a disponer de una cantidad de operadores humanos que sea proporcional al número de cámaras. Un número significativo de operadores redonda en un costo operacional alto de los centros de gestión y monitoreo. Debido a su alto costo, muchos de estos centros no cuentan con el personal suficiente. La Figura 3.1(b) muestra el trabajo de un operador humano, que tiene que monitorear una gran cantidad de cámaras al mismo tiempo. En el trabajo de Calavia [30] se indica que la monotonía está asociada a la somnolencia, siendo este un factor preponderante en la disminución y fluctuación del rendimiento de los operadores, lo que puede derivar en la perdida de la confiabilidad y calidad del sistema de vigilancia.



(a) Video wall con imágenes de múltiples cámaras de seguridad

(b) Estación de trabajo con un agente humano, monitorizando múltiples cámaras de seguridad a la vez

Figura 3.1: Ejemplos de visualización de sistemas de videovigilancia.

Fuente: Tomada de Exacq, Flickr.com (a) <https://www.flickr.com/photos/exacq/4515030622>, (b) <https://www.flickr.com/photos/exacq/1224729974>.

Por otro lado, la presencia del factor humano como agente decisario deriva en un problema ético, debido a que las personas tienden a verse influenciadas por prejuicios y estereotipos sobre potenciales sospechosos. Norris y Armstrong [31] desarrollaron un análisis empírico de los sistemas de CCTV (*Closed-circuit television*) instalados en la ciudad de Londres hasta el año 1999. El estudio se basó en el comportamiento de un conjunto de operadores durante 592 horas de trabajo, durante las cuales los operadores realizaron el análisis y seguimiento en pantalla del comportamiento de personas en tres áreas separadas, tanto demográfica como socialmente, reportando alrededor de 900 observaciones. Los resultados del estudio permitieron concluir que:

- Solamente una de cada veinte observaciones requirió de intervención policial, de las cuales una de cada cuatro finalizó en arresto (usualmente, por casos menores de violencia)
- 40 % de las personas observadas fueron catalogadas como sospechosas debido a su pertenencia a determinado grupo cultural y/o étnico, del cual el 65 % eran adolescentes, 68 % negros y 47 % hombres.

El trabajo indica que, en general, los operadores son personas con pocas habilidades, mal remuneradas, y con poca educación, aunque con un gran manejo práctico de los CCTV. A pesar de que los operadores deben regirse por un código de buenas prácticas, generalmente esto no sucede ya que suelen basarse en la imagen social de una persona (vestimenta, color de piel, sexo, etc.) para calificarla como sospechosa.

Con el fin de mitigar los problemas expuestos en párrafos anteriores, se han desarrollado sistemas de vigilancia computacionales que, mediante la aplicación de técnicas de procesamiento digital de imágenes y detección de patrones, son capaces de alertar a los operadores humanos sobre la aparición de anomalías en una escena. De esta manera se refuerza la efectividad del sistema, ya que al ser una máquina quien clasifica la situación como sospechosa se elimina el prejuicio y el estereotipo aplicado por el operador y éste, a su vez, estará dedicando su esfuerzo al monitoreo de las situaciones sospechosas.

indicadas por el sistema. Por lo tanto, el sistema no sustituye al operador humano y se inserta en el proceso de monitoreo como un agente que, de forma objetiva, clasifica las diferentes escenas y enfoca la atención de los operadores humanos en las escenas con indicios sospechosos.

El objetivo de este proyecto es el estudio de algoritmos de inteligencia computacional de utilidad para la implementación de un sistema capaz de detectar patrones de movimiento de personas a partir de imágenes de vídeo. El proyecto propone trabajar principalmente con imágenes obtenidas de cámaras de seguridad ubicadas en la vía pública. El estudio se lleva a cabo bajo la premisa de que el sistema debe ser capaz de procesar información de múltiples fuentes de datos en tiempo real.

3.2. Planificación del proyecto

Previamente al inicio del proyecto se acordaron los alcances del mismo y el resultado esperado. A partir de esto se proyectó el trabajo en diferentes etapas detalladas a continuación:

3.2.1. Cronograma

- **Etapa 1** Se define el problema a resolver y se lleva a cabo un relevamiento de las áreas relacionadas al proyecto. Se elabora un informe de estado del arte de las mismas.
- **Etapa 2** Se define la arquitectura y el diseño del sistema. Se lleva a cabo un relevamiento de las tecnologías y bibliotecas disponibles. Se implementan pruebas de concepto para poder evaluar y decidir las tecnologías (bibliotecas, lenguajes, etc) a ser utilizadas.
- **Etapa 3** Se implementa el sistema de acuerdo a lo definido en las etapas anteriores.
- **Etapa 4** Se diseña y ejecuta el análisis experimental del sistema creado.
- **Etapa 5** Se documenta y se culmina con un análisis de los resultados experimentales obtenidos. Se escriben las conclusiones y las propuestas de trabajo futuro.

3.3. Trabajos relacionados

A continuación se presentan revisiones y comentarios sobre una selección de artículos, los que revisten importancia porque tratan sobre el desarrollo de distintos sistemas y la aplicación de algoritmos que son de utilidad e interés para la realización de este proyecto.

3.3.1. Sistemas de videovigilancia

Valera y Velastin [32] presentan un estudio acerca de la creciente importancia de los sistemas de vigilancia inteligentes con arquitectura distribuida y un relevamiento de los avances en el área hasta el año 2005. Destacan que las principales áreas de estudio son la detección y reconocimiento de objetos en movimiento, rastreo, análisis de comportamiento y la recuperación de imágenes. Estas áreas involucran a la visión artificial, el análisis de patrones, la inteligencia artificial, el manejo de datos, el procesamiento de señales, las

telecomunicaciones y hasta los estudios socio-éticos. Los autores clasifican los sistemas en tres categorías, sistemas de primera, segunda y tercera generación.

En el grupo de sistemas de primera generación se encuentran los sistemas analógicos llamados CCTV, los cuales presentan buen desempeño en situaciones diversas, aunque no es fácil su distribución y almacenaje. Los sistemas de segunda generación son aquellos que combinan tecnologías de visión computacional con sistemas de CCTV para la automatización de ciertos procesos. Estos sistemas incrementan la eficiencia del sistema de vigilancia debido a la creación y utilización de algoritmos que detectan ciertos eventos, permitiendo que el usuario no deba ser el encargado de detectarlos. Igualmente, para obtener algoritmos robustos de seguimiento y detección de eventos es necesario contar con herramientas que permitan realizar el análisis de comportamiento. En la tercera generación se encuentran los sistemas automáticos que cubren grandes áreas y se despliegan de forma distribuida. Estos sistemas obtienen información más precisa gracias a la combinación de diferentes sensores. A su vez, el hecho de instalarse de forma distribuida hace que el manejo de la información se vuelva una tarea compleja. En esta última categoría se encuentra el sistema de detección de patrones de movimiento que se propone desarrollar en este proyecto. La Figura 3.2 muestra un flujo tradicional de los sistemas de procesamiento de imágenes.



Figura 3.2: Flujo tradicional de un sistema de procesamiento de imágenes.

3.3.2. Procesamiento de imágenes

Piccardi [33] presenta un resumen de los principales métodos utilizados en la etapa de detección y sustracción de fondo. La sustracción de fondo (**BACKGROUND SUBTRACTION**) es una técnica utilizada para detectar objetos en movimiento en imágenes provenientes de cámaras fijas a partir de la diferencia calculada entre un *frame*, denominado frame de referencia, y otro, denominado frame actual. Un *frame* se define como un fotograma, es decir, una imagen instantánea en la que se divide el vídeo. A la diferencia hallada entre los dos *frames* se le denomina *modelo del fondo*.

Piccardi hace hincapié en siete métodos, mencionando características principales de cada uno, para luego realizar un breve análisis de rendimiento centrado en la velocidad de procesamiento, los requerimientos de memoria y la precisión. Una síntesis del análisis de los métodos es presentado a continuación:

- *Running Gaussian Average* (RGA). Se basa en encontrar una función de densidad de probabilidad gaussiana que se ajuste, de manera ideal, a los valores de los últimos n píxeles. La sección A.2.1 del apéndice A brinda información detallada sobre la distribución gaussiana. RGA es el método más eficiente (ejecución en $O(1)$) debido a que para cada píxel solamente es necesario calcular la diferencia umbralizada.
- *Temporal median filter* (MFA). Utiliza el valor de la mediana de los últimos n *frames* para construir el modelado de fondo. Los algoritmos que utilizan esta técnica tienen buen desempeño en términos de la estabilidad del modelado de fondo. La estabilidad

suele depender de la aparición de cambios bruscos, aunque poco duraderos, en la escena que se procesa. Este método tiene una eficiencia similar a RGA, aunque la actualización del modelo puede ser aproximada como una función lineal, según el número n_s de muestras.

- *Mixture of Gaussians* (MOG). Un píxel es clasificado como fondo o frente a partir del promedio obtenido al aplicar una mezcla de gaussianas sobre el valor de ese píxel. Los cambios que no son permanentes en el fondo de la escena no influyen de gran manera en el modelo final. La cantidad de distribuciones gaussianas utilizadas se representa con m , cuyo valor típicamente oscila entre 3 y 5.
- *Kernel Density Estimation* (KDE). Se calculan histogramas para los valores de los píxeles que hayan sido clasificados como fondo más recientemente. De igual manera que MOG, KDE maneja una suma de gaussianas, con la salvedad de que cada gaussiana describe una única muestra de datos. Se utilizan los últimos n frames, es por eso que la ejecución del algoritmo alcanza $O(n)$.
- *Sequential KD approximation*. Es una técnica alternativa que requiere el estudio de convergencia sobre todo el espacio de datos, lo que la convierte en una técnica con alto costo computacional. La complejidad de este método es $O(m + 1)$, en donde m representa la cantidad de modos de aproximar la función de densidad de probabilidad.
- *Concurrence of image variations*. Se basa en el principio de localidad espacial, lo que implica que si un bloque de píxeles pertenecientes al fondo sufre variaciones, existe una elevada probabilidad de que bloques de píxeles vecinos sufran variaciones similares.
- *Eigenbackgrounds*. Esta técnica aplica un principio similar al de *Concurrence of image variations*, pero en lugar de utilizar bloques de píxeles utiliza toda la imagen. El hecho de utilizar toda la imagen ayuda a evitar el efecto "particionado" que se genera al trabajar con bloques. M es la cantidad de los mejores autovectores.

Piccardi demuestra que de todos los métodos mencionados, el método RGA es el que presenta mejores rendimientos tanto en velocidad de procesamiento como en requerimientos de memoria. En cuanto a la precisión, el autor afirma que los métodos MOG y KDE presentan el mejor desempeño. La tabla 3.1 muestra una comparación de performance

Método	Velocidad	Memoria	Precisión
Running Gaussian Average	$O(1)$	$O(1)$	Baja/Media
Temporal median filter	$O(n_s)$	$O(n_s)$	Baja/Media
Mixture of Gaussians	$O(m)$	$O(m)$	Alta
Kernel Density Estimation	$O(n)$	$O(n)$	Alta
Sequential KD approximation	$O(m + 1)$	$O(m)$	Media/Alta
Coconcurrence of image variations	$O(8n/N^2)$	$O(nK/N^2)$	Media
Eigenbackgrounds	$O(M)$	$O(n)$	Media

Tabla 3.1: Métodos de sustracción de fondo y análisis de performance. *Adaptada de [33]*

para cada uno de los métodos mencionados, según su velocidad de ejecución, el consumo de memoria y la precisión que alcanzan.

López [34] presenta un sistema capaz de detectar el movimiento aparente (i.e., distorsión en la imagen capturada, provocada por el movimiento de la cámara). La investigación releva dos tipos de métodos, los que detectan el movimiento a partir de la diferencia de imágenes consecutivas y los que detectan el movimiento a partir de una única imagen. En el segundo grupo se encuentran los métodos de flujos ópticos (*Optical flows*), utilizados ocasionalmente para la sustracción de fondo, para los cuales el autor concluye que tienen un costo computacional alto e introducen ruido al proceso. El autor opta por utilizar métodos de alineamiento global y alineamiento por puntos con correspondencia. Una vez que el sistema obtiene la imagen alineada, sin movimiento aparente, ésta y la original son enviadas a un módulo segmentador. El módulo segmentador cuenta con tres capas, sustracción de fondo, proceso de etiquetado (*labeling*) y proceso de agrupamiento (*grouping*). La salida del módulo segmentador resulta en un conjunto de regiones de interés, también llamadas *blobs*. Una vez obtenido el conjunto de *blobs*, éstos son enviados al módulo de seguimiento (*tracking*) que, aplicando el algoritmo presentado por Stauffer y Grimson [35] y filtros de Kalman [36], los utiliza como factor final de decisión para la detección del movimiento aparente. Los resultados logrados son cercanos al 90 % sin utilizar el componente de *tracking* y rondan el 100 % al sumarle dicho componente.

3.3.3. Detección y seguimiento de personas

Lefloch [37] presenta un sistema de conteo de personas basado en la detección del momento en el que una persona cruza una línea virtual previamente establecida. Las imágenes son capturadas con una cámara colocada en el techo del lugar, lo cual resulta muy beneficioso ya que elimina los problemas de oclusión.

En primer lugar se aplica un método de sustracción de fondo que da como resultado una imagen binaria. Una imagen binaria es aquella que tiene únicamente dos valores posibles para sus píxeles. En este sistema, una imagen binaria determina cuál píxel pertenece al fondo y cuál al frente (área que presenta movimiento). Luego, se aplican operaciones morfológicas a la imagen binaria, con el fin de eliminar el ruido presente y las áreas pequeñas y aisladas que presentan un mínimo movimiento. Las operaciones utilizadas son la erosión, dilatación, apertura (resultado de aplicar una erosión y luego una dilatación) y cierre (resultado de aplicar una dilatación y luego una erosión).

La imagen resultado de aplicar las operaciones morfológicas mencionadas es sometida a una etapa de Detección y clasificación de *blobs*. En esta etapa se detectan las regiones de píxeles contiguas, para las cuales se calcula su bounding box (rectángulo de área mínima que contiene al *blob*). Una vez que se obtiene el conjunto de bounding boxes se descartan aquellos que no cumplen ciertos criterios en la relación ancho-alto y área. De esta forma se logra filtrar aquellos bounding boxes que potencialmente contienen personas.

El restante paso del sistema consiste en detectar cuando un bounding box cruza la línea virtual establecida, y dependiendo de que lado del rectángulo la cruce primero, se logra deducir en qué sentido transita la persona. Resulta interesante estudiar la forma en la que el autor construye su sistema, el cual se encuentra modularizado en tres subsistemas, Procesamiento del fondo (*Background process*), Segmentación (*Segmentation*) y Seguimiento y conteo (*Tracking and Counting*), sumado al hecho de que presenta un método sencillo y computacionalmente eficiente para detectar personas.

Rodriguez et al. [38] abordan el problema de detectar y seguir personas en multitudes

muy densas. Este tipo de problemas presentan una serie de dificultades como la oclusión y el cambio de forma y localización de las personas. El enfoque que toman los autores es el de detectar diferentes partes del cuerpo, y de esta forma mitigar el problema de las oclusiones que dificultan la detección del cuerpo completo, centrándose particularmente en detectar la cabeza de las personas. La aplicación de técnicas como Sustracción de fondo y Segmentación no son útiles en este tipo de escenarios, debido a que no permiten aislar a las personas presentes en la imagen (porque las mismas se presentan en multitudes muy densas). Por esta razón los autores deciden combinar el funcionamiento de detectores de objetos y de algoritmos de estimación de densidad.

Los algoritmos de estimación de densidad permiten obtener información sobre las cantidades de personas en ciertas regiones de la escena, pero no brindan información sobre la localización de éstas dentro de la región. Los autores entrenaron un detector de objetos para que detecte elementos similares a cabezas humanas. Una vez entrenado, se aplicó el detector a todas las regiones de la imagen, generando así un mapa que contiene puntuaciones. Las puntuaciones indican posible presencia de personas. El mapa de puntuaciones es combinado con los datos obtenidos por los algoritmos de estimación de densidad para obtener finalmente la detección de personas.

En el año 2012 Dollar et al. [39] elaboraron un trabajo en el cual analizan los 16 detectores de personas más relevantes al momento. En ese trabajo, los autores intentaron responder a las preguntas: *¿Los detectores actuales funcionan bien?*, *¿Cuál es el mejor enfoque?*, *¿Cuáles son las principales fallas?*. Para responder esas preguntas, realizaron evaluaciones de rendimiento de los algoritmos, bajo iguales condiciones. Las pruebas incluyeron variaciones tanto del escenario como de los conjuntos de datos.

Los autores indican que decidieron evaluar detectores pre-entrenados, obtenidos directamente de sus propios autores. Los estudios permitieron concluir que, en general, el rendimiento de los detectores de personas se encuentra lejos de la perfección, incluso bajo las condiciones más favorables. Estos autores observan una degradación del rendimiento, en gran medida cuando se trabaja con imágenes de personas cuya área se encuentra por debajo de los 30 píxeles. Además afirman que una oclusión a partir del 35 % degradada en gran medida el rendimiento. Los autores indican también la necesidad de profundizar la investigación en siete áreas: detección de personas comprendidas en imágenes de entre 30 y 80 píxeles, rendimiento ante oclusiones, utilización de características de movimiento, utilización de información temporal junto a la información de seguimiento, relación con el contexto, incorporación de nuevas características que mejoren la detección y rendimiento de los algoritmos.

3.3.4. Detección de anomalías

Leach et al. [40] describen el problema de detectar anomalías sutiles en el comportamiento de las personas. El trabajo se basa en el procesamiento de señales sociales. Para la toma de decisiones se implementa un proceso que tiene en cuenta el contexto del escenario y el contexto social. El contexto de escena reconoce 4 potenciales regiones: de tráfico (registran un alto número de trayectorias), ociosas (registran trayectorias estacionarias), de convergencia y divergencia (separan o unen trayectorias) y generales (las que no clasifican en las anteriores). El contexto social tiene sus bases bajo la premisa de que dos individuos que comparten un alto grado de información en sus trayectorias (dirección, velocidad, proximidad y solapamiento de trayectorias), tienen una dependencia social.

En la etapa de detección y extracción de trayectorias de las personas se utilizaron

investigaciones desarrolladas con anterioridad, ya que los autores no se pretendían que este fuese el objetivo del trabajo. Para la detección de peatones se utilizó un detector basado en el trabajo de Felzenszwalb et al. [41], mientras que para el seguimiento se aplicó TLD (*Tracking-Learning-Detection*), fruto del trabajo de Kalal et al. [42].

El objetivo se fijó en la cabeza de los peatones y no en el cuerpo entero con el fin de reducir los problemas que genera la oclusión. Para las pruebas se utilizaron los conjuntos de datos de PETS-2007 y Oxford Dataset.

La ejecución del sistema en el conjunto de datos PETS2007 registró un valor de TPR de 0.78 y 0.19 de FPR, mejorando 0.13 en comparación con algoritmos que no tienen en cuenta el contexto social. Los resultados dejan en evidencia que inferir conexiones sociales entre las personas ayuda a mejorar la toma de decisiones.

Cho y Kang [43] presentan en su trabajo un sistema de detección de comportamiento anormal basado en agentes híbridos y aplicado a escenarios complejos. El sistema presenta la novedad de estudiar la interacción grupal en conjunto con el comportamiento individual de las personas. Los agentes se categorizan en agentes estáticos y agentes dinámicos. Los agentes estáticos están fijos en ciertos puntos y calculan la información de movimiento temporal (velocidad y dirección) de los objetos pertenecientes al fondo de la imagen utilizando la variación del flujo óptico. Los agentes dinámicos son asignados a los objetos en movimiento y se desplazan de acuerdo al campo de flujo óptico. Estos se encargan de calcular la información de interacción social entre ellos y sus agentes vecinos usando el modelo SFM (*Social interaction Force Magnitude*) y la energía potencial de interacción.

El sistema procede de la siguiente forma: divide cada vídeo en bloques no superpuestos, en el *frame* inicial coloca un agente estático y un agente dinámico en cada centro de bloque, luego computa el campo de flujo óptico entre pares de *frames* consecutivos, extrayendo la información de los comportamientos individuales y grupales utilizando los agentes estáticos y dinámicos, representando la información como *feature words*. Como resultado el vídeo de entrada es representado por *bag-of-words* y finalmente se utiliza un clasificador SVM (*Support Vector Machine*) para clasificar la anormalidad en las actividades del vídeo.

El sistema fue implementado en Visual C++ y utiliza las bibliotecas OpenCV y LIBSVM. Se concluye que el sistema presenta un mejor rendimiento que el método SFM en pruebas sobre el conjunto de datos PETS2009 y UCSD. Además, los autores demuestran que los agentes estáticos ayudan a detectar movimiento rápido o en áreas restringidas y comportamientos anormales de sujetos individuales, mientras que los agentes dinámicos ayudan en la detección de movimiento desordenado (pánico) y movimientos de separación (mal detectado por sistemas de agentes simples).

Zhu et al. [44] presentan un método de detección de anomalías en escenarios complejos, basado en la representación del movimiento. El método presenta la novedad de utilizar información estructural, como información de patrones de contexto, lo que sus autores afirman que no ha sido explotado adecuadamente en trabajos previos ya que, en general, se utiliza como información de contexto la información visual de nivel bajo o medio presente en las regiones circundantes.

El método se centra en modelar la actividad de la multitud utilizando la información del movimiento y del contexto, considerando que todos los objetos o regiones con movimiento detectados se proporcionan información de contexto entre ellos.

Con el fin de mitigar problemas presentes en trabajos previos utilizan MB-LBP-TOP (*Multiscale Block - Local Binary Pattern coding on Three Orthogonal Planes*) y MHFC

(*Multiscale Histogram of Frequency Coefficient*). MB-LBP-TOP es una codificación, utilizada para codificar la información de contexto local, basada en la unión de los patrones MB-LBP (*Multiscale Block - Local Binary Pattern*) y LBP-TOP (*Local Binary Pattern on Three Orthogonal Planes*). MHFC es un descriptor de característica utilizado para describir la información de movimiento. El descriptor MHFC se basa en la extracción de *dense trajectories*, preferentemente con filtrado de ruido, y presenta dos beneficios: puede lograr la independencia en los datos y puede capturar de mejor forma las características del movimiento de la trayectoria.

Primero, el método extrae *dense trajectories* y aplica filtros de ruido, filtrando las trayectorias atípicas. Luego, obtiene el descriptor MHFC para describir la información del movimiento y captura la información contextual modelando la información de los patrones del contexto local, utilizando la codificación MB-LBP-TOP. Finalmente, basado en la información de los patrones de contexto y en la información del movimiento, entrena un modelo utilizando *sparse coding*. Sobre el modelo entrenado se aplica *sparse reconstruction cost* para clasificar los eventos en normales y anormales.

Para la implementación los autores utilizaron el lenguaje MATLAB. Para las pruebas experimentales utilizaron una computadora con 3 GB de RAM y con una CPU a 3.1 GHz, obteniendo un tiempo de procesamiento de 4 segundos por *frame*. Las pruebas experimentales fueron hechas sobre tres conjuntos de datos: UCSD en Ped1 y Ped2 y Subway dataset.

Los autores concluyen que el método presenta mejores resultados que los desarrollados al momento, argumentando que la mejora se debe a que el método no toma en cuenta únicamente el movimiento de la multitud, sino que también utiliza información de contexto que aportan los objetos circundantes. Además, se detecta que se dan falsas alarmas cuando hay significativa distorsión de perspectiva o el tamaño de los objetos es muy variado, lo que podría solucionarse utilizando diferente escala de objetos en el entrenamiento.

Capítulo 4

Arquitectura y diseño del sistema

Este capítulo tiene como objetivo brindar detalles de la arquitectura del sistema implementado. Inicialmente se describe la arquitectura planteada para la resolución del problema, junto a su debida justificación. A continuación, se presentan y comentan los diagramas de componentes que ilustran la complejidad e interacción de los módulos que componen la solución.

4.1. Arquitectura general del sistema

Es necesario disponer de una arquitectura que permita, en tiempo real, tomar las imágenes generadas por la fuente de datos y procesarlas secuencialmente. A su vez, la arquitectura debe ser suficientemente flexible como para permitir sustituir o agregar al procesamiento nuevos algoritmos sin conllevar grandes esfuerzos. De igual forma, debe ser capaz de escalar para soportar el procesamiento de múltiples fuentes de datos de forma concurrente.

De la información relevada de los trabajos relacionados (especialmente de [32] y [37]) se infiere que los procesos aplicados sobre las imágenes son independientes unos de otros y que se adaptan correctamente a un comportamiento en cadena. Los algoritmos relevantes bien pueden tomar como entrada la salida de su, o sus, inmediatos anteriores. La arquitectura de tubos y filtros se adapta perfectamente a este escenario. Como definieron Garlan y Shaw en [45], en una arquitectura de tubos y filtros (*pipes and filters* en inglés) los componentes del sistema tienen definido un conjunto de entradas y salidas. Las salidas sirven de entradas para otros componentes mientras que el componente en sí cumple la función de filtro. A las conexiones entre filtros se les llama tubos.

La arquitectura planteada, en términos generales, está basada en la de tubos y filtros. Cada algoritmo es visto como un filtro específico que actúa sobre los datos resultantes de uno o varios anteriores. El sistema se compone entonces de módulos independientemente funcionales. Para el problema resuelto esta arquitectura resulta de gran conveniencia ya que permite reutilizar los filtros y convierte el agregado de nuevos filtros a la secuencia en una tarea relativamente sencilla. Además, es intuitivo pensar el sistema como una secuencia de transformaciones de los datos.

A gran escala, el sistema se compone de dos módulos principales. El primero se encarga del reconocimiento y seguimiento de las personas (u objetos de interés) mientras que el segundo recibe los resultados del primero y en base a estos y a un histórico reciente busca el cumplimiento de patrones. Ambos módulos soportan múltiples ejecuciones con-

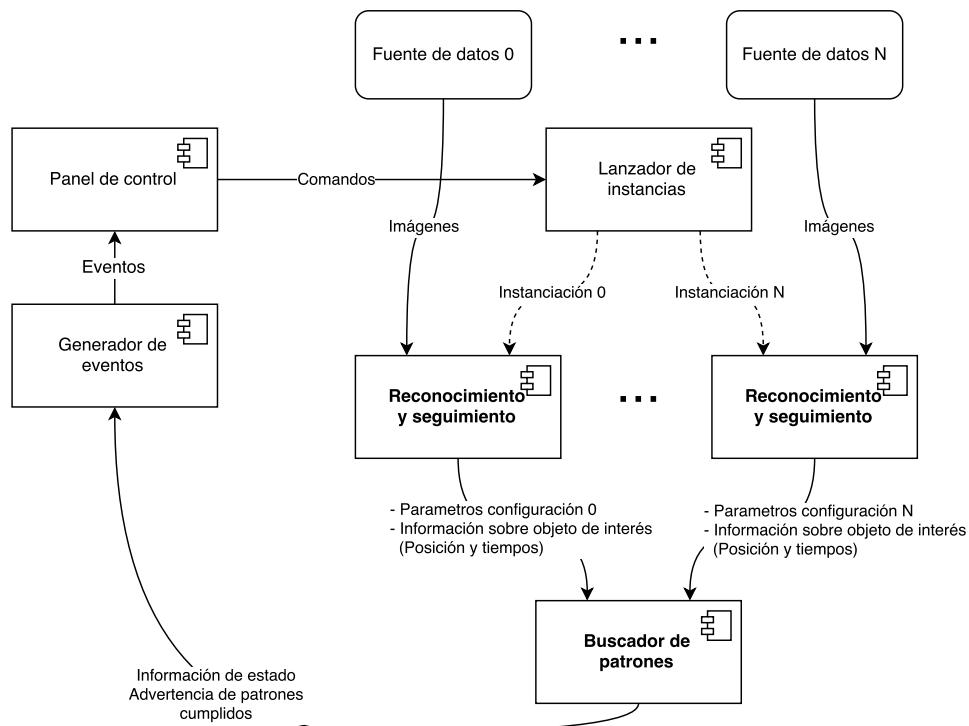


Figura 4.1: Diagrama de arquitectura del sistema.

currentes. En el caso del módulo Reconocimiento y seguimiento, estas permiten atender múltiples fuentes de datos a la vez. Estos módulos pueden ser visualizados en 4.1 con sus títulos en negrita.

Es necesario disponer de un método de transferencia de datos desde las instancias del primer módulo hacia las del segundo. Este método debe asegurar la integridad de los datos, su envío dentro de un rango de tiempo y luego de este tiempo debe tener la capacidad de expirarlos, respetar el orden en el que fueron enviados y no enviar repetidamente el mismo dato. Es también de conveniencia que la transferencia ofrezca un medio de transporte seguro y que sea un estándar abierto.

Complementariamente a los dos módulos principales, la solución presenta tres módulos auxiliares. El primer módulo auxiliar se encarga de lanzar las instancias de módulos requeridas para el correcto funcionamiento. En su inicio crea una instancia del reconocedor de patrones y una instancia de cada uno de los modulo auxiliares restantes. El segundo módulo auxiliar es un pequeño servidor web que expone un panel de control. El módulo Panel de control es capaz de comunicarse con el Lanzador de instancias para solicitarle la creación de instancias del módulo Reconocimiento y seguimiento. El Panel de control recibe los datos resultantes del sistema a través del tercer módulo auxiliar, el Generador de eventos. Una vez recibidos los resultados, el Panel de control los comunica a los clientes web suscritos al momento. El módulo Generador de eventos tiene la tarea de estar pendiente de nuevos resultados del Buscador de patrones y una vez arribados los transforma en eventos del Panel de control.

En la Figura 4.1 se visualizan todos los módulos que componen el sistema y la forma en la que interactúan entre ellos.

4.2. Arquitectura del módulo Reconocimiento y seguimiento

El módulo Reconocimiento y seguimiento presenta una arquitectura de tubo basada en filtros. La entrada al módulo corresponde a las imágenes crudas proporcionadas por la fuente de datos. Primero, la entrada es procesada por el componente Sustracción de fondo resultando como salida una imagen binaria, esto es una imagen con valores cero para los puntos de la imagen original identificados como parte del fondo (estático) y valores uno para los identificados como parte del primer plano (dinámico). Luego, la imagen binaria es enviada al componente Detección de blobs y como resultado obtenemos un conjunto de *blobs* debidamente identificados. Los *blobs* son un conjunto de puntos adyacentes entre sí que forman parte del primer plano (dinámico) de la imagen. Una vez obtenido el conjunto de *blobs* este es transferido al siguiente componente, el Filtro de blobs. El filtro descarta aquellos *blobs* que no contienen objetos de interés y clasifica los que si tienen dando como resultado un conjunto de objetos de interés identificados en el espacio de la imagen. A continuación, el conjunto de objetos de interés es enviado al componente Seguimiento el cual asocia la posición de los objetos identificadas con posiciones de objetos previamente registrados. Finalmente, el componente Seguimiento calcula el movimiento en el tiempo para cada uno de los objetos identificados y entrega la información resultante como resultado final del módulo Reconocimiento y seguimiento.

La Figura 4.2 muestra los componentes que conforman el módulo y la interacción entre ellos. El diagrama describe el flujo de los datos desde la entrada de la imagen en crudo hasta la salida resultante y se detalla el tipo de dato compartido entre un componente y el siguiente.

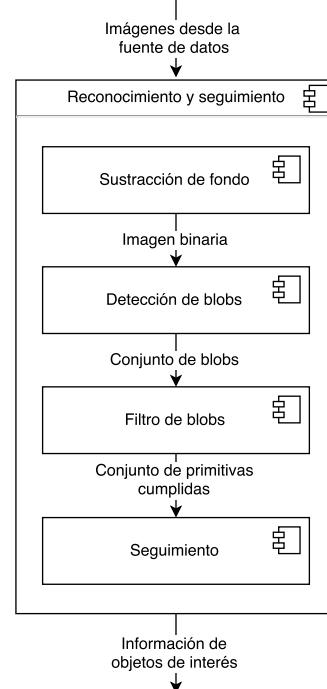


Figura 4.2: Diagrama de componentes del módulo Reconocimiento y seguimiento.

La salida del módulo depende únicamente de los datos que recibe como entrada, por lo que puede ser ejecutado en varias instancias concurrentes. Cada ejecución es capaz de procesar de forma autónoma el reconocimiento y seguimiento de una fuente de datos.

4.3. Arquitectura del módulo Buscador de patrones

El módulo Buscador de patrones recibe información de los objetos de interés proporcionada por las instancias del módulo Reconocimiento y seguimiento. Esta información contiene, entre otras cosas, la posición en el tiempo de cada objeto detectado junto a un identificador único tanto del objeto de interés como de la instancia. El módulo procesa los datos de entrada e identifica características de los objetos. A estas características identificadas se les llama *primitivas*. La secuencia formada por las primitivas identificadas se compara con una secuencia de primitivas preestablecidas. A la secuencia preestablecida se le llama *patrón*. Los patrones están formados por, además de la secuencia de primitivas, una serie de valores de propiedades asociados a cada una de estas. Si las primitivas que conforman un patrón se cumplen en determinado momento, con determinado orden y con los valores de las propiedades especificadas, entonces el patrón se cumple. La Figura 4.5 ejemplifica un caso de primitiva y una secuencia de primitivas registradas para un objeto de interés. El módulo Buscador de patrones presenta flexibilidad en la definición de patrones pero no en la definición de primitivas (excepto que se reprograme).

El módulo presenta una arquitectura de tubo basada en dos filtros. El primer filtro recibe como entrada los datos resultantes del módulo Reconocimiento y seguimiento. Los datos son integrados a un historial reciente de posiciones del objeto en cuestión. Una vez juntos, se procesan las posiciones a lo largo del tiempo y se identifican las primitivas que cumple (e.g., una persona camina).

La secuencia de primitivas pasa a ser la entrada del segundo filtro. El segundo filtro es un comparador de secuencias. Su tarea consta de comparar la secuencia de las últimas primitivas registradas para un objeto de interés frente a un conjunto de patrones previamente establecidos. El filtro genera una salida que contiene los patrones que se cumplen en la secuencia de primitivas registradas para el objeto, con determinado grado de confianza. La Figura 4.3 expone los filtros que componen el módulo Buscador de patrones junto a la interacción y datos intercambiados entre ellos.

4.3.1. Método para encontrar patrones

El método propuesto para encontrar patrones se basa fuertemente en el expuesto por Van Huis et al. [46] con las adaptaciones correspondientes al escenario planteado. Se plantea la búsqueda de patrones a través de la cercanía detectada entre una secuencia de primitivas y un conjunto de patrones previamente establecidos. La Figura 4.4 muestra un modelo de dominio parcial (solo se muestran componentes relacionados al método) de la solución planteada en la que se pueden ver las relaciones entre primitivas, patrones y objetos de interés.

Las primitivas son definidas como características básicas de los objetos de interés, pudiendo ser dependientes de la velocidad, la dirección u otro atributo de los mismos. Las primitivas son integradas a la lógica del sistema de forma estática y pueden ser clasificadas en dos tipos: primitivas mono-objetivo y primitivas multi-objetivo. Las primitivas mono-objetivo hacen referencia a un único objeto de interés e ignoran los restantes presentes en el escenario. Las primitivas multi-objetivo tienen en cuenta múltiples objetos de interés

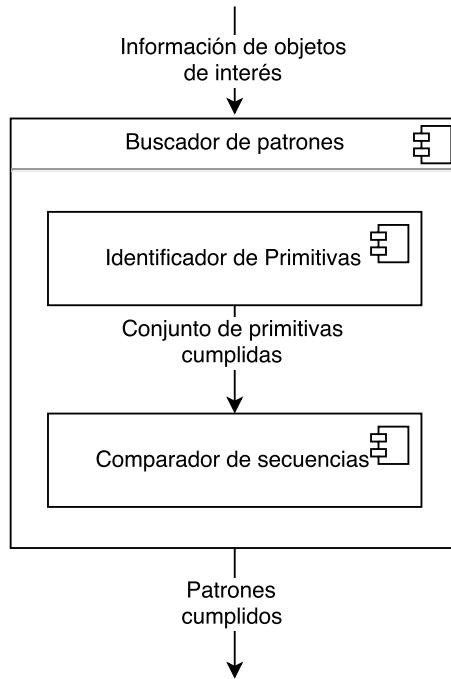


Figura 4.3: Diagrama de componentes del módulo Buscador de patrones.

presentes en el escenario. Por ejemplo, la velocidad con la que se mueve una persona puede determinar si la persona camina, corre o permanece en el mismo lugar, lo que da lugar a tres primitivas mono-objetivo relacionadas a la velocidad: Camina, Corre, Parado. Una primitiva multi-objetivo podría ser la ocurrencia de una aglomeración.

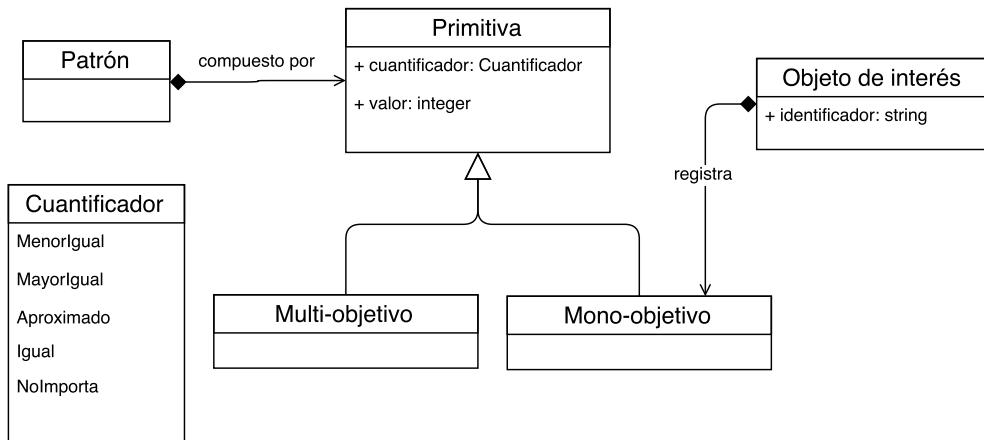


Figura 4.4: Modelo de dominio parcial del módulo Buscador de patrones.

Por otro lado están los *patrones*, los cuales se definen como secuencias de primitivas. Los patrones son integrados al sistema de forma dinámica, como entradas del mismo. Para cada primitiva que integra un patrón se define un cuantificador y un valor. Por ejemplo, puede haber en un patrón una primitiva del tipo velocidad, que se cumple si una persona se encuentra caminando por un tiempo mayor o igual a 10 segundos. En

este caso, el cuantificador es *mayor o igual* y su valor es 10.

El método planteado hace posible la extensión del conjunto de patrones a través de datos de entrada al sistema. Además, el método provee una construcción intuitiva de los patrones a partir de la secuencia de primitivas.

4.3.2. Medición del error en el cumplimiento de patrones

Un patrón puede ser cumplido de forma aproximada por una secuencia de primitivas. Esto sucede cuando todas las primitivas de un patrón aparecen en una secuencia en el mismo orden, aunque no contiguas (separadas por primitivas adicionales). Para medir el error de una secuencia de primitivas frente a un patrón se diseñó un método utilizando el concepto de distancia temporal. En una comparación, la distancia temporal se define como la sumatoria del tiempo que consumen las primitivas adicionales. Cuanto menor es la distancia temporal, más exacto es el cumplimiento del patrón y menor es el error medido. Por lo tanto una secuencia que cumple perfectamente con el patrón tiene un error valuado en cero mientras que una secuencia que no cumpla exactamente tiene un valor de error mayor a cero. Esta cuantificación es de utilidad para priorizar los patrones cumplidos por orden de exactitud y para descartar los que no se encuentren dentro de un umbral de error establecido.

A modo de ejemplo, la Figura 4.5 expone un caso en el que el objeto de interés es una persona. El patrón buscado se cumple cuando la persona camina por al menos cinco segundos, permanece quieta por aproximadamente cinco segundos y luego corre por al menos cinco segundos. La secuencia inferior indica las últimas primitivas registradas para esa persona. En este caso el sistema indica que el patrón se cumple, aunque no con la mejor medida de error, o sea mayor a cero, ya que entre la primitiva Camina y Parado se registra la primitiva Corre por pocos segundos y entre la primitiva Parado y Corre se detecta una Camina por pocos segundos. La medida de error está dada por la sumatoria de tiempo entre Camina-Parado (Corre por dos segundos) y Parado-Corre (Camina por tres segundos). Sumando los espacios temporales se obtiene una medida de error valor 5 cuando la secuencia que cumple el patrón exactamente tiene un valor de error de cero.

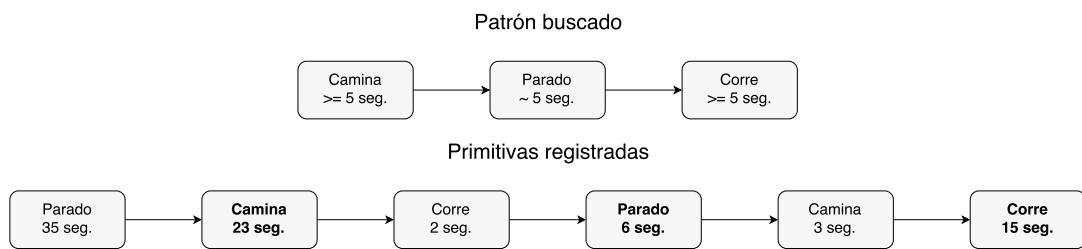


Figura 4.5: En la primer línea se ilustra la definición de un patrón mientras que en la segunda el registro de las últimas primitivas registradas para un objeto de interés. En este caso, la medida de error tiene un valor de cinco cuando el valor ideal de error es cero.

4.4. Arquitectura de los módulos auxiliares

Esta sección presenta los tres módulos auxiliares del sistema, los cuales sirven de herramienta para el control del procesamiento de fuentes de datos. La Figura 4.6 muestra los componentes de los módulos auxiliares y la interacción entre ellos y el resto de los módulos.

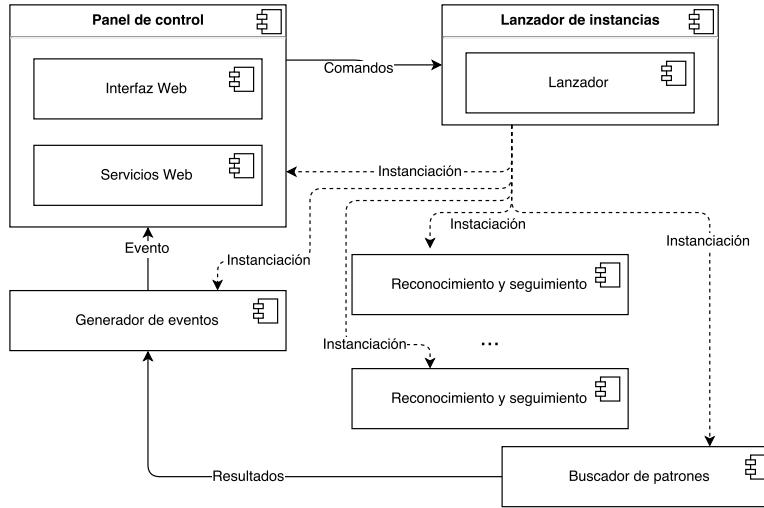


Figura 4.6: Arquitectura de los módulos auxiliares y sus interacciones.

En su inicio, el módulo Lanzador de instancias crea una instancia del Buscador de patrones, del servidor web para el Panel de control y del Generador de eventos. Luego, queda a la espera de solicitudes de nuevas instancias del módulo Reconocimiento y seguimiento, o de mensajes que contienen un comando de administración (e.g. cerrar una instancia).

El módulo Panel de control está conformado por un servicio web y una lógica de visualización sencilla. Permite al usuario final solicitar la creación de nuevas instancias, cerrar las ya creadas y visualizar en tiempo real los patrones que se van detectando en las fuentes de datos. Este módulo es reactivo, ya que realiza acciones únicamente frente a solicitudes de agentes externos.

El módulo Generador de eventos permanece bloqueado a la espera de resultados generados por el Buscador de patrones. Una vez arribado un mensaje este se convierte en un evento y se genera una solicitud al Panel de control para la distribución del evento a los clientes web.

En la Figura 4.7 queda reflejado el papel que juegan los módulos auxiliares en el sistema. El diagrama de actividad comienza cuando se recibe la solicitud de procesamiento de una fuente de datos identificada como 'cámara 0'. La solicitud es transmitida al Lanzador de instancias el cual genera una instancia de Reconocimiento y seguimiento (instancia '0') para atender la solicitud, a su vez, el módulo envía un mensaje al Generador de eventos para dar aviso de la nueva instancia. Por un lado, el Generador de eventos recibe el mensaje y genera el evento correspondiente para luego transmitirlo al Panel de Control, el Panel de Control se encarga de hacerlo llegar a los clientes web. Por otro lado, la instancia de Reconocimiento y seguimiento comienza el procesamiento de imágenes. Si la instancia de Reconocimiento y seguimiento encuentra objetos de interés, transmite la información de los mismos a el Buscador de patrones; si no quedan más imágenes que

procesar, la instancia se termina. Una vez recibida la información de los objetos de interés por parte de la instancia del Buscador de patrones, este procesa los datos en busca de cumplimiento de patrones. Si el Buscador de patrones encuentra que alguna secuencia de primitivas cumple algún patrón, genera un mensaje para el Generador de eventos, quien genera el evento correspondiente y lo transmite al Panel de Control para su difusión a clientes web.

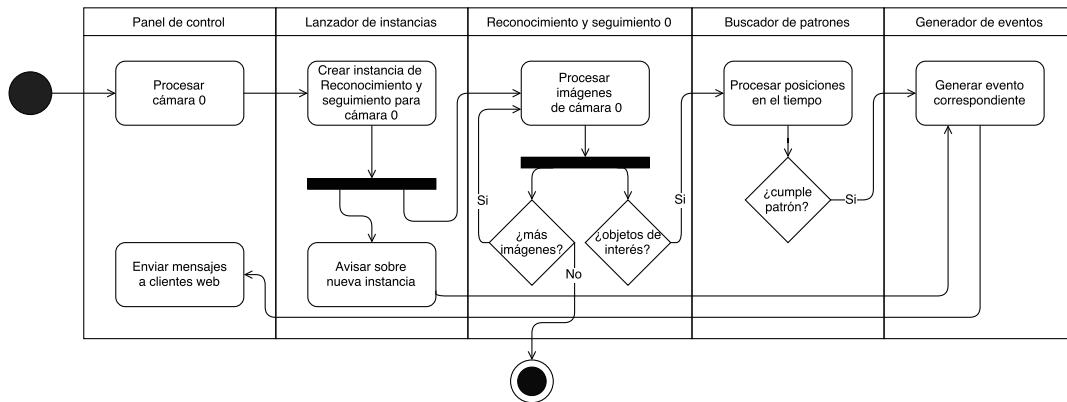


Figura 4.7: Diagrama de actividad de módulos del sistema.

Capítulo 5

Implementación

En este capítulo se describen las decisiones tomadas en el transcurso de la implementación del sistema. Se enumeran las tecnologías utilizadas junto a sus versiones así como los algoritmos, de terceros o de implementación propia, utilizados en cada etapa.

5.1. Elección de las tecnologías utilizadas para el desarrollo

La búsqueda de las tecnologías a utilizar se basó en el cumplimiento de características previamente fijadas, respondiendo a las necesidades del sistema. Estas características incluyeron:

- Utilizar un lenguaje multiplataforma, lo que permite implementar el sistema sin saber, a priori, sobre qué tipo de hardware y sistema operativo es finalmente ejecutado.
- Que el lenguaje no agregue complejidades técnicas además de las complejidades propias del problema a resolver (e.g., que gestione la memoria utilizada para evitando lidiar con *memory leaks*, que no sea fuertemente tipado).
- Que el lenguaje y las bibliotecas sean de uso gratuito y preferentemente de código abierto. Esta característica permite aprender de lo que ya está implementado, evaluar su aplicabilidad y eventualmente poder efectuar cambios en los algoritmos de terceros.
- Que el lenguaje y las bibliotecas dispongan de una comunidad amplia y activa para que los problemas encontrados con las mismas puedan ser rápidamente solucionados. Además, el hecho de tener una comunidad activa es un fuerte indicio de que el lenguaje o biblioteca se encuentra en uso en la actualidad.
- Que el lenguaje presente los mejores niveles de rendimiento posibles en las áreas que aborda el sistema. La elección de la tecnología no debe incidir de forma abruptamente negativa en el rendimiento del sistema ya que es un aspecto fundamental al procesar imágenes en tiempo real.

Teniendo en cuenta las características planteadas, se realizó un relevamiento de las tecnologías disponibles en comunidades de desarrolladores en Internet. A partir del relevamiento se seleccionaron para un estudio más detallado las tecnologías Matlab, OpenCV

sobre C/C++ y OpenCV sobre Python. Partiendo de los trabajos realizados por Mallick [47] y Coelho [48], se estudiaron las ventajas y desventajas de las tecnologías seleccionadas.

Comenzando por las ventajas que ofrece Matlab, este dispone de un mecanismo que gestiona la memoria, es multiplataforma, cuenta con buena documentación y es muy potente en los cálculos matriciales. Es importante tener en cuenta que las imágenes son matrices de vectores multidimensionales. Por otro lado, como desventajas Matlab presenta una licencia de uso comercial que no es gratuito ni económicamente accesible. El sistema de licencias de Matlab obliga al usuario a adquirir, por un lado el paquete básico y por otro un conjunto de paquetes particulares dependiendo del fin del sistema, todos ellos con sus respectivos precios. Además, al ser Matlab un lenguaje pensado para cálculos matriciales provoca que los programas se deban escribir de una forma diferente a la acostumbrada en los lenguajes de propósito general [49] [50] [51] [52]. Es bueno señalar también que Matlab está basado en Java, y este se ejecuta sobre una máquina virtual (JVM) implementada en C/C++, esto provoca, ocasionalmente, que en funcionalidades implementadas en C/C++ tengan mejor rendimiento que las implementadas en Matlab.

En el caso de OpenCV y C/C++, como ventajas se tiene que tanto OpenCV como C/C++ son de uso gratuito y de código abierto. Además, la biblioteca OpenCV está optimizada y brinda la posibilidad de paralelizar procesamiento mediante OpenCL en unidades de procesamiento gráfico (GPUs). Sumado a lo anterior, tanto OpenCV como C/C++ cuentan con grandes comunidad de desarrolladores. La documentación de C/C++, al ser un lenguaje maduro y de uso masivo, es muy buena pero no tan así la de OpenCV, aunque se encuentra en mejora continua. El manejo de memoria en C/C++ es responsabilidad del desarrollador lo que agregando una complejidad técnica extra al desarrollo del sistema.

Con respecto a OpenCV y Python, esta opción comparte varias de las ventajas que tiene OpenCV y C/C++ ya que la biblioteca, aunque utilizada desde Python, es desarrollada y compilada en C/C++. OpenCV es utilizada a través de una interfaz Python (o *wrapper* como se le llama en inglés). Además, Python es de uso gratuito y su código es abierto, cualidades que comparte con OpenCV (como se explica en el párrafo anterior). Sumado a esto, Python es un lenguaje dinámicamente tipado y cuenta con un gestor de memoria automático (llamado *garbage collector* o abreviado como *gc*), estas dos características facilitan el desarrollo permitiendo poner mayor foco en la solución del problema planteado. Python, al igual que C/C++, cuenta con una gran comunidad de desarrolladores activa, a esto agrega una alta gama de bibliotecas científicas gratuitas y abiertas (e.g. *scipy*, *numpy*, *scikit-learn*).

La Tabla 5.1 resume el cumplimiento de las características deseadas que presentan las tecnologías tenidas en cuenta para implementar el sistema. Thorne et al. presentan en su trabajo [53] evidencia del rendimiento de OpenCV en C y OpenCV en Python. El trabajo compara ambas combinaciones y concluye en que el rendimiento computacional entre las dos opciones presenta una leve diferencia a favor de OpenCV en C. Teniendo en cuenta que la diferencia de rendimiento es leve y estando presentes las facilidades de Python, expuestas anteriormente, a la hora de implementar, entre las dos opciones es conveniente OpenCV en Python. La otra opción estudiada, Matlab, presenta la desventaja de su licencia paga, su código cerrado, y además, contrario a Python, no es un lenguaje de propósito general.

A partir de este estudio se concluyó que la mejor opción tecnológica para la imple-

Características	Tecnologías		
	Matlab	OpenCV(C/C++)	OpenCV(Python)
Multiplataforma	SI	SI	SI
Tipado dinámico	SI	NO	SI
Mecanismo de gestión de memoria	SI	NO	SI
Gratis	NO	SI	SI
Código abierto	NO	SI	SI
Fuerte comunidad	SI	SI	SI
Buen rendimiento para propósito general	NO	SI	SI

Tabla 5.1: Comparativa entre las tecnologías y las características deseadas.

mentación del sistema es la biblioteca OpenCV utilizada con el lenguaje Python. En particular, para la implementación se utiliza OpenCV 3.0.0 y Python 3.4.3.

5.2. Comunicación entre módulos

Como especifica la arquitectura del sistema en la sección 4.1, el sistema hace uso de un método de comunicación entre módulos. Este método asegura la entrega, el orden, la integridad y la unicidad de los mensajes además de expirar aquellos que no han sido procesados luego de transcurrido cierto tiempo.

En el sistema que se expone en este informe, se utilizó una implementación del estándar abierto *Advanced Message Queuing Protocol* (AMQP), que se describe con mayor detalle en el apéndice A.1. Existen numerosas implementaciones de AMQP. Para el sistema se optó por la implementación *RabbitMQ* desarrollada en *Erlang/OTP* en su versión 3.2.4. Esta implementación fue pensada para soportar paralelismo y gran transferencia de mensajes en red. Aparte de *RabbitMQ*, se utilizó la biblioteca *pika* que implementa un controlador de *RabbitMQ* para el lenguaje Python.

En la Figura 5.1 se puede observar la configuración de *exchanges* y colas utilizada en el sistema. Por convención se realizan todos los intercambios de datos en formato JSON. JSON es un formato estándar, independiente del lenguaje y sencillo que está dirigido al intercambio de datos. Su conjunto de reglas se encuentran definidas en el RFC 7159 [54].

Entre el módulo Reconocimiento y seguimiento y el módulo Buscador de patrones se utiliza un *exchange* del tipo *direct*, nombrado '*track_info*'. Los generadores de mensajes se corresponden con las diferentes instancias del módulo Reconocimiento y seguimiento. Todos los mensajes del *exchange* '*track_info*' son dirigidos a una única cola llamada '*patternmaster_rcv*' y atendida por la instancia de Buscador de patrones. La instancia de Buscador de patrones es quien permanece a la espera y procesa cada mensaje a su arribo, por lo tanto decimos que esta instancia es un proceso reactivo basado en eventos. Entre estos dos módulos se intercambian dos tipos de mensajes. El primer tipo de mensaje contiene la configuración con la que el módulo Buscador de patrones debe atender a la instancia de Reconocimiento y seguimiento que figura como remitente del mensaje. El segundo tipo de mensaje contiene información precisa de un objeto de interés (e.g., un identificador único, una marca de tiempo del momento que se comenzó a reconocer, una marca de tiempo de la última vez que se actualizó, la última posición registrada en la

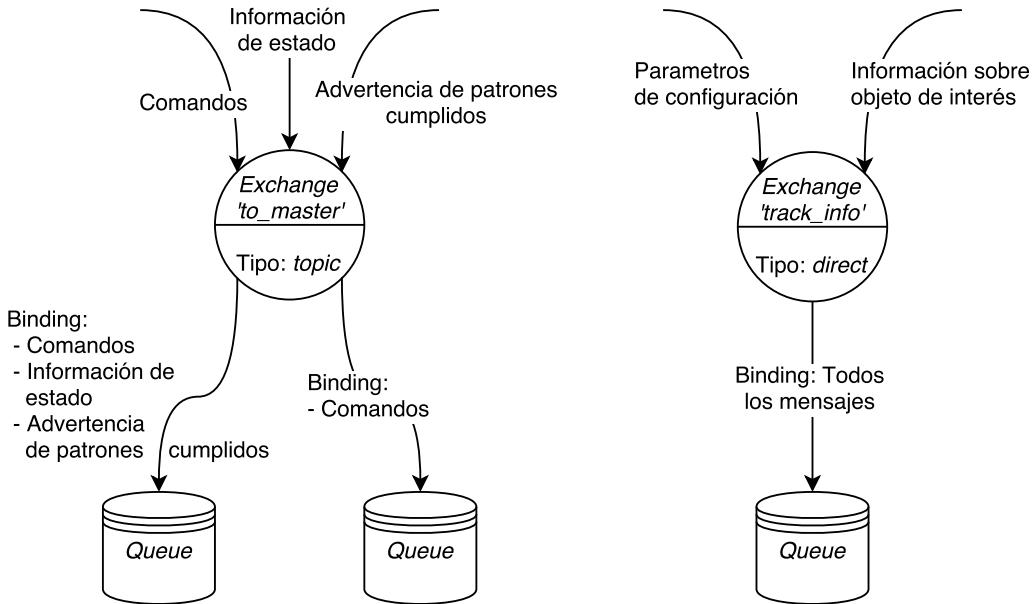


Figura 5.1: *Exchanges* definidos en el sistema.

imágen, la medida del rectángulo mínimo que contiene al objeto).

El módulo Buscador de patrones envía sus datos de salida a un *exchange* del tipo *topic* nombrado '*to_master*'. Cada mensaje es enviado con una clave de enrutamiento específica para su tipo. Los tipos posibles son: comandos, información de estado y advertencias de patrones cumplidos. A este *exchange* se suscriben el módulo Generador de eventos, a través de la cola '*web_rcv*', y el módulo Lanzador de instancias, a través de la cola '*launcher_rcv*'. En el caso del Generador de eventos, la información de *binding* corresponde a los mensajes del tipo comandos, información de estado y advertencias de patrones cumplidos, mientras que la información de *binding* para el módulo Lanzador de instancias corresponde únicamente al tipo de mensaje comandos.

En todos los mensajes se establece un tiempo de expiración parametrizado y configurable. Por defecto este tiempo es de 60 segundos y sus valores se encuentran definidos en los parámetros `STATUS_INFO_EXPIRATION_TIME`, `TRACK_INFO_EXPIRATION_TIME` y `WARNINGS_EXPIRATION_TIME`.

5.3. Detalles generales de la implementación

En la implementación de los módulos se siguió la guía de estilo de escritura estándar PEP8 definida por van Rossum et al. [55]. En su mayoría, los algoritmos de procesamiento de imágenes utilizados son implementaciones disponibles en la biblioteca OpenCV 3. La biblioteca OpenCV 3 fue escrita y compilada en lenguaje C/C++ y se utiliza desde Python a través de una interfaz que provee OpenCV 3. Para la instalación de OpenCV 3 y su correspondiente interfaz para Python 3 en plataformas Linux y MacOS se siguió la guía '*Install opencv 3.0 and python 3.4+ on ubuntu*' elaborada por Rosebrock [56]. Como herramienta de desarrollo se utilizó el IDE PyCharm 2016.1 y como sistema de control de versiones se utilizó Git, proporcionado por el servicio Bitbucket.

En el modulo Panel de Control se utilizó el micro-framework web Flask. Flask propor-

ciona una pequeña implementación de un servidor web capaz de ejecutar y publicar una aplicación desarrollada en Flask. Para comunicarle a los clientes web los eventos registrados por el sistema y para transmitir comandos desde el cliente web hacia el Lanzador de instancias se utilizó la tecnología WebSocket a través de la extensión Flask-SocketIO. Para la lógica de visualización web se utilizó el lenguaje Javascript y las bibliotecas jQuery y Underscore.

El sistema se estructura de la siguiente forma:

- Directorio *controlpanel*: contiene la aplicación web en *webmanager.py*, la lógica de visualización en *static/js/app.js* y *templates/index.html* y el Generador de eventos en *events.py*.
- Directorio *experimental_analysis*: contiene todas las herramientas referentes al análisis experimental de la solución cuyos resultados son presentados en el capítulo 6.
- Directorio *patternmaster*: contiene lo referente al módulo Buscador de patrones. El archivo *event.py* concentra las clases necesarias para el manejo de primitivas, el archivo *rule.py* contiene la clase para el manejo de patrones y la lógica para la carga de los mismos. *tracklet.py* alberga la clase que registra las primitivas identificadas para cada objeto de interés. *pattern_recognition.py* implementa la clase *Pattern-Recognition*, que contiene los métodos para calcular las primitivas que cumple un objeto de interés, comparar la secuencia de primitivas registradas con los patrones del sistema y enviar mensajes al *exchange* cuando corresponda. El archivo *__main__.py* inicia el módulo Buscador de patrones e implementa la lógica bloqueante que permanece a la espera del arribo de mensajes a ser procesados. La implementación de este módulo es presentada en detalle en la sección 5.5.
- Directorio *trackermaster*: contiene todo lo referente al módulo Reconocimiento y seguimiento. El archivo *__main__.py* contiene la lógica necesaria para el pasajes de información entre filtros y el envío de mensajes al Buscador de patrones. La implementación de los filtros se encuentran en el directorio *blackboxes* (*cajas negras* en español). El nombre del directorio hace referencia a que los filtros son una suerte de caja negra (i.e., se conoce su entrada y salida pero no como los detalles de cómo lo hace) para el bloque principal del módulo. Dentro del directorio *blackboxes* se encuentran los archivos *background_subtraction.py* que implementa los métodos de la sustracción de fondo, *blob_assignment.py* que resuelve la asignación de *blobs* a objetos de interés registrados en un histórico reciente, *blob_detection.py* que implementa la detección de *blobs* en la imagen, *histogram2d.py* que implementa un método auxiliar para el filtro de *blobs*, *person_detection.py* que implementa un filtro de *blobs* para la detección de personas, *person_detection_task.py* que implementa métodos auxiliares para el filtro de *blobs* de personas y *tracking.py* que implementa toda la lógica referente al seguimiento de objetos de interés. La implementación de este módulo es presentada en detalle en la sección 5.4.
- Directorio *utils*: contiene funciones generales que fueron implementadas para facilitar diferentes procesamientos en los módulos (e.g., un controlador para la comunicación con Redis, un controlador de instancias del módulo Reconocimiento y seguimiento, conversores de medidas).

Los módulos principales, Reconocimiento y seguimiento y Buscador de patrones, constan de sus respectivos archivos de configuración. Estos archivos definen los valores por defecto de los parámetros del sistema. En caso de que un usuario decida no ingresar sus propios valores como entrada del sistema, estos son tomados de los archivos de configuración. Los archivos se encuentran en el directorio de cada módulo y tienen la extensión `.conf`. Su sintaxis es igual a la declaración y asignación de variables en Python.

El diagrama de la Figura 5.2 muestra las instancias de los componentes del sistema en un momento dado de funcionamiento. En el diagrama se visualizan los *exchanges* y las colas con las cuales interactúan, la interacción de los archivos de configuración y sus módulos respectivos, la ejecución en paralelo de las instancias de Reconocimiento y seguimiento y el uso del archivo *patterns_definition.dat* para la carga de patrones (la sección 5.5 provee detalles sobre la implementación de la carga de patrones) por parte del módulo Buscador de patrones.

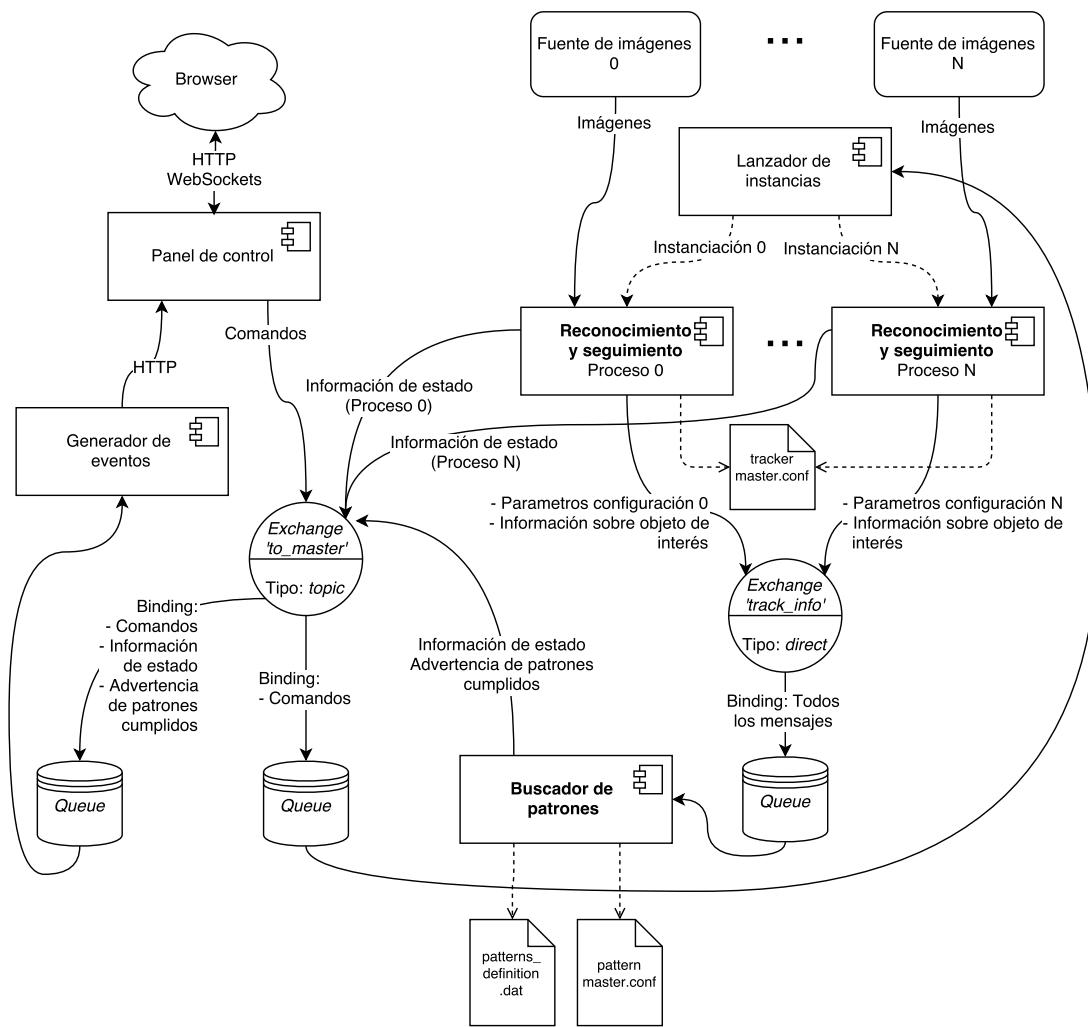


Figura 5.2: Diagrama de componentes del sistema.

5.4. Implementación del módulo Reconocimiento y seguimiento

Esta sección describe los detalles en la implementación del módulo Reconocimiento y seguimiento. En términos generales, el bloque principal del módulo expresado en el Algoritmo 2 sigue una estructura de filtros y tubos.

Algoritmo 2 Bloque principal del módulo Reconocimiento y seguimiento.

```

1: while  $framesToProcess$  do
2:    $binaryImage \leftarrow BackgroundSubtraction(frame)$ 
3:    $blobs \leftarrow BlobsDetector(binaryImage)$ 
4:    $filteredBlobs \leftarrow BlobFilter(blobs)$ 
5:    $trackedObjects \leftarrow Tracker(filteredBlobs)$ 
6:   if  $trackedObjects$  then
7:      $sendTrackedObjects(trackedObjects)$ 
8:   end if
9: end while

```

En las siguientes subsecciones se brindan detalles de la implementación de los componentes, los problemas enfrentados y la solución presentada para solventarlos.

5.4.1. Sustracción de fondo

La sustracción de fondo es un método utilizado para la detección de objetos en movimiento a partir de cámaras estáticas (i.e., cámaras sin movimiento aparente). Esta etapa de procesamiento resulta crucial en aplicaciones en las que es necesario detectar movimiento, identificar y/o realizar el seguimiento de objetos en imágenes dinámicas. Este método puede encontrarse también bajo el nombre *Extracción de primer plano* (*Foreground extraction* en inglés). El método permite seleccionar la imagen en zonas que aparecen estáticas (fondo) y zonas que presentan movimiento (frente).

En una primera instancia se implementó la sustracción de fondo a partir de la diferencia píxel a píxel entre una imagen capturada por la cámara en un tiempo $t - 1$ y otra capturada en el tiempo t . Esta implementación fue utilizada en la etapa de elección de tecnologías con el fin de evaluar el rendimiento de las diferentes opciones relevadas. Luego, al momento de implementar formalmente el componente de sustracción de fondo, se desestimó la implementación ya que presentaba escasa presión y se llevó a cabo el relevamiento de los algoritmos disponibles, tanto los incluidos en OpenCV como otros implementados por terceros particulares.

Los algoritmos seleccionados para trabajar más a fondo fueron MOG2, presentado por Zivkovic [57], y k -NN, presentado por Zivkovic y van der Heijden [58]. Estos algoritmos fueron implementados para ser utilizados en la biblioteca OpenCV. La sección A.2 del apéndice A brinda el marco teórico en el cual se basan ambos métodos.

El pseudocódigo representado por el Algoritmo 3 muestra los pasos que sigue el componente encargado de la sustracción de fondo. Primero es necesario pasar la imagen a escala de grises, ya que esto permite procesar menos información. Cuando se trabaja con imágenes a color se tienen tres canales de 8 bits cada uno, los cuales representan la información de los píxeles, mientras que para las imágenes en escala de grises se utiliza

Algoritmo 3 Pasos del algoritmo de sustracción de fondo.

- 1: $grayScaleImage \leftarrow BGRToGray(rawImage)$
 - 2: $blurImage \leftarrow GaussianBlur(grayScaleImage)$
 - 3: $rawBinaryImage \leftarrow Subtractor(blurImage)$
 - 4: $binaryImageNotNoise \leftarrow ErodeAndDilate(rawBinaryImage)$
-

un solo canal de información de 8 bits. Una vez obtenida esta imagen, se procede a suavizarla.

El proceso de suavizado (*smoothing* o *blurring* en inglés) es una técnica utilizada para reducir el ruido presente en la imagen. Las operaciones de suavizado se realizan mediante la aplicación de filtros, generalmente filtros lineales. La biblioteca OpenCV incluye implementaciones de varios de los filtros existentes. En este componente se utiliza el filtro *Gaussiano* (*Gaussian filter* en inglés) [59]. El filtro Gaussiano filtra a partir de la convolución de cada punto de la imagen con un *kernel gaussiano* para luego obtener la sumatoria como resultado final. El filtro Gaussiano es aplicado en el componente de sustracción de fondo a través de la clase *GaussianBlur* provista por OpenCV.

El siguiente paso, luego de suavizar la imagen, es aplicar la sustracción de fondo. Para esto se utiliza alguno de los dos métodos seleccionados, implementados por la biblioteca OpenCV [60]. El método MOG2, el cual utiliza *Mixture of gaussians* (explicado en la sección Sustracción de fondo del apéndice A), se aplica a través de la clase *BackgroundSubtractorMOG2* mientras que *k*-NN, que utiliza el algoritmo *k-Nearest Neighbours* (explicado en la sección K-Nearest Neighbours del apéndice A), se aplica a través de la clase *BackgroundSubtractorKNN*. La utilización de un método u otro se indica a través del parámetro de configuración *USE_BSUBLTRACTOR_KNN*, donde el valor ‘True’ indica la utilización de 1 método *k*-NN y el valor ‘False’ la utilización del método MOG2. Ambos métodos retornan una nueva imagen binaria en la cual los píxeles blancos indican movimiento y los negros indican el fondo de la imagen.

La figura 5.3 muestra las distintas etapas que transcurren desde que el componente de sustracción de fondo recibe la imagen en crudo hasta llegar a una imagen binaria. La imagen binaria obtenida como resultado de la sustracción de fondo determina las áreas que forman parte del frente de la imagen y las que forman parte del fondo. En la Figura 5.3(b) se observa como algunos elementos se detectan incompletos o demasiado cerca de otros resultando en una unión no deseada. Para mitigar este tipo de problemas se aplican operaciones morfológicas.

Las operaciones morfológicas implementadas en OpenCV son *erosión*, *dilatación*, *apertura* y *cierre* [61]. La *erosión*, aplicada a los bordes de los elementos, permite separar elementos que aparecen juntos por áreas de conexión pequeñas. La *dilatación*, en cambio, permite unir elementos que se encuentran cercanos, aplicando un engrosamiento de los bordes. Las operaciones de *apertura* y *cierre* se obtienen a partir de la combinación de las anteriores. La operación de *erosión* seguida de la de *dilatación* corresponde a la operación de *apertura* mientras que la de *dilatación* seguida de la de *erosión* corresponde al *cierre*. La aplicación de operaciones morfológicas resulta en imágenes con menos ruido y elementos mejor identificados, como se aprecia en la Figura 5.3(c).

A través del parámetro de configuración del sistema con el nombre de *GAUSSIANBLUR_SIZE* es posible definir el *kernel gaussiano* a ser utilizado para la operación de suavizado. Por otro lado, para la configuración de las operaciones morfológicas

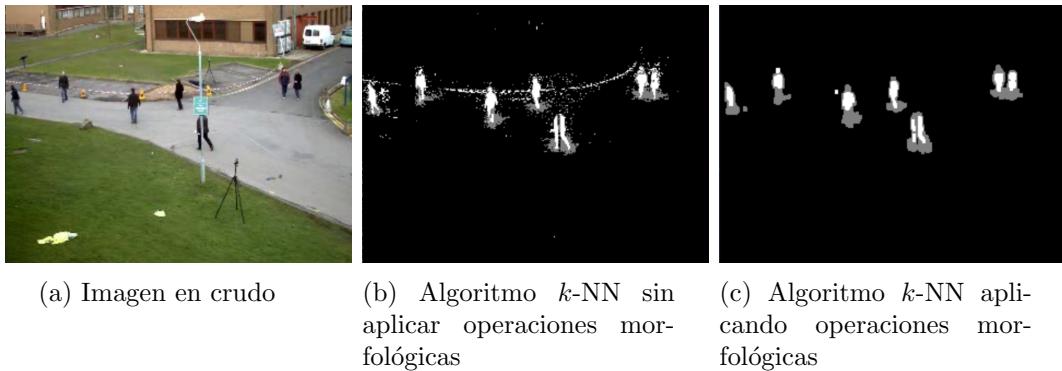


Figura 5.3: Etapas de la sustracción de fondo

se dispone de los parámetros `ERODE_SIZE` y `DILATE_SIZE` que definen el *kernel* para la operación de erosión y dilatación. `ERODE_TIMES` y `DILATE_TIMES` establecen la cantidad de veces que la operación correspondiente es aplicada.

5.4.2. Detección de *blobs*

Para la etapa de detección de *blobs* se dispone de dos algoritmos, uno proporcionado por OpenCV y otro implementado como parte del desarrollo de este proyecto a partir de la combinación de diferentes métodos. El primer algoritmo es *SimpleBlobDetector*, cuyo funcionamiento se explica de forma detallada en la sección A.3.1 del apéndice A. El segundo método se denomina *Detección de blobs por bounding boxes*. El parámetro de configuración booleano `DETECT_BLOBS_BY_BOUNDING_BOXES` permite seleccionar qué algoritmo se utiliza para la detección de *blobs*. Si el valor del parámetro es '*False*' se utiliza la detección de *blobs* *SimpleBlobDetector* de OpenCV, en caso contrario se utiliza la Detección de *blobs* por *bounding boxes*.

El método de Detección de *blobs* por *bounding boxes*, cuyo pseudocódigo es presentado en el Algoritmo 4, se compone fundamentalmente de dos pasos:

- Detección de contornos de los elementos presentes en la imagen binaria.
- Búsqueda de rectángulos de menor área que contengan a los contornos detectados. Estos rectángulos se denominan *bounding boxes*.

Algoritmo 4 Algoritmo Detección de *blobs* por *bounding boxes*

```

1: contours  $\leftarrow$  findContours(binaryImageWithoutNoise)
2: bounding_boxes  $\leftarrow$  emptyList()
3: for each contour in contours do
4:   bounding_box  $\leftarrow$  findBoundingBox(contour)
5:   bounding_boxes.append(bounding_box)
6: end for

```

Sin importar cuál de los dos algoritmos se decida utilizar, esta etapa retornará un conjunto de rectángulos, los cuales contienen la información suficiente para ejecutar la siguiente etapa, la clasificación de *blobs*.

5.4.3. Clasificación de *blobs*

A continuación se presenta las técnicas y enfoques estudiados para clasificar el conjunto de *blobs* detectados. En la etapa de clasificación de *blobs* se busca identificar qué *blobs* contienen objetos de interés y descartar el resto.

Con el fin de clasificar los *blobs* se implementaron tres técnicas diferentes que se combinan en pos de obtener mejores resultados. Las técnicas son:

- Relación de aspecto. Se clasifican los *blobs* verificando que los mismos cumplan con ciertos criterios en base a su relación de aspecto. La relación de aspecto de una imagen se define como el cociente del ancho y el alto de la misma.
- Algoritmos de inteligencia computacional. Se utilizan algoritmos pre entrenados para detectar objetos de interés, por ejemplo para detectar personas.
- Frecuencia de relación de aspecto. Se utilizan histogramas para trabajar con información estadística acerca de las dimensiones de los *blobs*.

Los *blobs* pueden ser clasificados como útiles o como no útiles. Un *blob* útil es aquel para el cual se puede asegurar, con cierto grado de certeza, que contiene un objeto de interés. Los *blobs* que no son clasificados como útiles son descartados, ya que se asume que no contienen objetos de interés y por lo tanto no aportan información relevante para las siguientes etapas del sistema.

Relación de aspecto

Esta técnica se basa en clasificar los *blobs* a partir de su relación de aspecto, definida como la relación entre el ancho y el alto de un *blob*. Si la relación tiene un valor aproximado a la que mantienen los objetos de interés en la imagen, entonces se da cierto grado de certeza de que el *blob* contiene al menos un objeto de interés. Por ejemplo, una persona tiene una proporción de dos cabezas de ancho y ocho de alto, esto significa que su relación de aspecto es de $8/2 = 4$. Por lo tanto, los *blobs* cuyo alto dividido su ancho resulten en un valor aproximado a cuatro son considerados *blobs* que contienen una persona.

La principal ventaja que presenta este enfoque es la facilidad con la que puede ser aplicado, debido a que la relación de aspecto de los objetos de interés puede ser calculada de manera sencilla mediante observación, y su comparación con el valor de referencia de relación de aspecto se logra con un esfuerzo computacional pequeño.

Sin embargo, esta técnica presenta algunos inconvenientes. El primero y más evidente es que la relación de aspecto puede variar entre escenarios (principalmente por la perspectiva), debido a que la posición desde la cual se obtienen las imágenes puede ser diferente de un escenario a otro. Otro problema es que la relación de aspecto no es útil para descartar elementos, ya que el hecho de que un elemento cumpla con la relación de aspecto no asegura que realmente sea un objeto de interés. Además, objetos de interés sobrepujados parcialmente o múltiples objetos de interés contiguos son descartados por formar parte de un *blob* de proporciones diferentes, las cuales no cumplen con la relación de aspecto establecida.

La Figura 5.4(b) muestra un ejemplo de estas situaciones problemáticas. En la figura se puede apreciar la existencia de un *blob* que difiere con la relación de aspecto de una

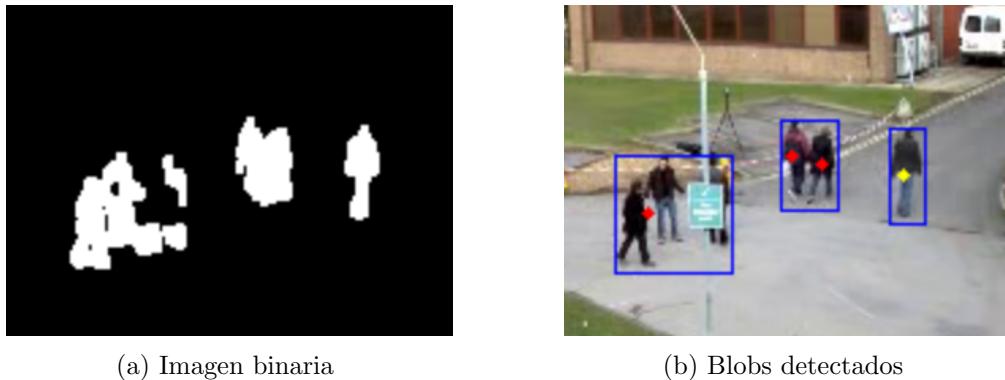


Figura 5.4: Imágenes binaria y original, en donde se aprecian los *blobs* detectados y la relación de aspecto que estos mantienen

persona pero aún así contiene dos personas (*blob* del centro). Una relación de aspecto que cumple con la estipulada como referencia puede ser observada en el *blob* de la derecha.

Como consecuencia de la poca precisión del método, es necesario combinarlo con otros para mejorar su efectividad. Además de mejorar la precisión, al utilizar el método de forma alternada y éste ser computacionalmente sencillo, se reduce la capacidad de cómputo necesaria en la etapa de clasificación de *blobs*.

La relación de aspecto de referencia se define como el valor del parámetro de configuración **ASPECT_RATIO** en forma de número decimal. Este valor depende directamente del escenario al que se aplique.

Algoritmos de inteligencia computacional

Otro enfoque para la detección de objetos de interés consiste en la utilización de algoritmos de inteligencia computacional pre-entrenados. En el caso del sistema implementado, se aplica a la detección de personas.

La biblioteca OpenCV provee métodos para la detección de distintos tipos de objetos [62]. En particular, para este trabajo se utilizó el algoritmo *Default people detector* que se encuentra entrenado para la detección de personas. *Default people detector* implementa el algoritmo HOG sugerido por Dalal y Triggs [63]. En el estudio los autores demuestran que la combinación de HOG con SVM (*Support vector machines*) aplicado a la detección de personas logra mejores resultados que el mejor método Haar presentado por Mohan et al. [64].

El algoritmo desarrollado por Dalal y Triggs en 2005 utiliza HOG para la extracción de características y SVM para la clasificación de las mismas, de esta forma se combinan para crear un detector de personas previamente entrenado con un conjunto de imágenes de personas recortadas de una variada selección de fotografías. El método se basa en recorrer la imagen utilizando una ventana de detección, que es dividida en forma de grilla y sobre la cual se extraen, utilizando HOG, los vectores de características de la imagen. Esos vectores, combinados, alimentan la entrada del algoritmo SVM, el cual se encarga de clasificar las características extraídas y decidir si se detectó una persona o no. La sección A.4 del apéndice Conceptos teóricos brinda información más detallada sobre la detección de objetos de interés en imágenes y los métodos utilizados, en particular se mencionan los métodos HOG y SVM.

El algoritmo HOG puede aplicarse en toda la imagen, en cuyo caso devuelve el conjunto de todas las personas detectadas en la misma. En este trabajo interesa encontrar los patrones de *movimiento* de las personas, por lo que se decidió buscar solamente en aquellas zonas de la imagen que presentan movimiento. Así, el detector de personas es aplicado únicamente a los *blobs* que son detectados, reduciendo de esta forma el tiempo de procesamiento ya que no se están procesando zonas de la imagen en las que no se detecta movimiento.

Algoritmo 5 Algoritmo de detección de personas en blobs

```

1: persons  $\leftarrow$  emptyList()
2: for each blob in blobs do
3:   position  $\leftarrow$  getPosition(blob)
4:   cropped_image  $\leftarrow$  crop_image(original_image, position)
5:   person  $\leftarrow$  detect_person(cropped_image)
6:   persons.append(person)
7: end for
8: persons  $\leftarrow$  apply_NonMaximumSuppression(persons)
  
```

Para este cometido es necesario obtener la región de la imagen correspondiente al *blob*. Utilizando las coordenadas del *blob* se recorta en la imagen original de entrada al sistema el área en cuestión, para luego aplicar el detector de personas solamente a esta región. El recorte va en conjunto con una pequeña transformación que se aplica de manera tal que el tamaño resultante se aproxime a 64 píxeles de ancho y 128 píxeles de alto, dimensiones para las cuales Dalal y Triggs entrenaron el algoritmo. En aquellos casos en los cuales un *blob* sobrepasa ese tamaño, ya sea porque existe más de una persona en el mismo o porque el objeto en movimiento es más grande, se busca llevar una de las dimensiones a la indicada.

A modo de ejemplo, si el ancho del recorte se encuentra más próximo a 64 píxeles que el alto a 128 píxeles, entonces se redimensiona el recorte para que tenga el ancho buscado, aunque siempre manteniendo la relación de aspecto original del *blob*. La importancia de mantener la relación de aspecto radica en que es necesario que las personas que se busca detectar mantengan su relación de aspecto original.

El detector de personas devuelve un conjunto de rectángulos que contienen a las personas halladas en cada uno de los *blobs*. Para mejorar la detección, se utiliza el algoritmo *Non-Maximum Suppression* sugerido por Rosebrock [66][65]. El mismo, luego de aplicado, permite obtener un conjunto de rectángulos menor o igual al original, logrando reducir la cantidad de elementos que se pasan a la siguiente etapa y de esta manera se reduce el tiempo de procesamiento. La sección A.4.3 del apéndice A brinda información detallada sobre el algoritmo *Non-maximum suppression*.

Frecuencia de relación de aspecto

El método Frecuencia de relación de aspecto filtra *blobs* a partir de la frecuencia con que éstos fueron identificados como contenedores de objetos de interés. El método hace uso de histogramas bi-variados para registrar la frecuencia. Toma valores rango de ancho y alto de los *blobs* y los asocia a una región del histograma. El histograma queda seccionado en sus dos dimensiones formando áreas rectangulares, como se puede ver en la Figura 5.5. La información contenida en el histograma permite filtrar los *blobs* cuyas

medidas de alto y ancho no son similares a *blobs*, que frecuentemente se detectan como contenedores de objetos de interés.

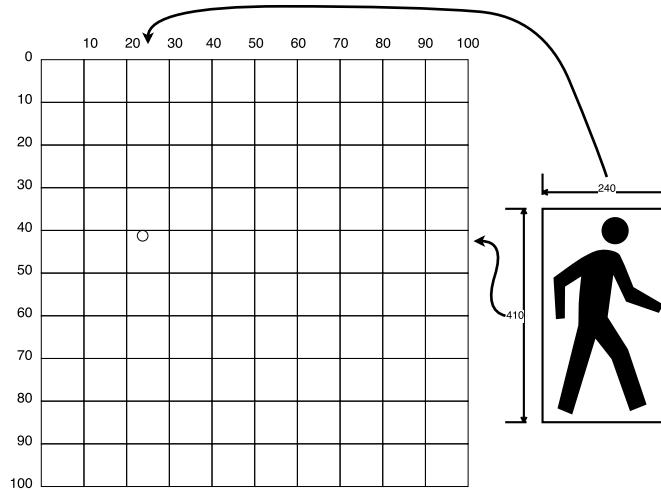


Figura 5.5: Histograma de frecuencia de tamaños de blob

El modelo es provisto de valores provenientes únicamente de aquellos *blobs* que un segundo método, considerado de buena precisión, aseguró que contiene objetos de interés. En el caso de la implementación del sistema, solo aquellos *blobs* procesados por los algoritmos de inteligencia computacional detallados en la subsección precedente aportan datos al modelo. Este método brinda la posibilidad de que parte de los *frames* no sean procesados con algoritmos computacionalmente costos y en su lugar se procesen con éste método. En la etapa de análisis experimental se ejecutan técnicas para encontrar la proporción adecuada de *frames* a ser procesados por este método y por los métodos de inteligencia computacional.

El método puede ser utilizado de dos formas. La primera es la forma trivial, esto es, se toma directamente el valor de frecuencia registrado en las coordenadas correspondientes para el *blob* en cuestión. Si este valor es mayor a cero, entonces al menos un *blob* de dimensiones similares, o sea, en el mismo rango, fue identificado como contenedor de objetos de interés. De esta forma, sin más procesamiento, el sistema toma el *blob* como contenedor de objetos de interés. La segunda forma agrega pasos intermedios antes de identificar el *blob* como contenedor de objetos de interés. En este caso se definen niveles de confianza determinados a partir de los valores de frecuencia de las regiones adyacentes a la del *blob* en cuestión. Entonces, aunque la región correspondiente a un *blob* en cuestión registre niveles bajos de frecuencia, si sus regiones adyacentes registran niveles altos, ésto incide en la decisión de descartarlo o declararlo contenedor de objetos de interés. El parámetro de configuración booleano `USE_CONFIDENCE_LEVELS` define cuál de las dos formas es aplicada en el sistema.

Cuando se utilizan niveles de confianza se definen tres niveles, determinados por la división del rango de valores de frecuencia posibles. Los niveles definidos son:

- **Nivel 0** - Es el nivel de menor confianza, por lo tanto, está determinado por medidas que registran una baja frecuencia. Si un *blob* pertenece a un área de nivel 0 y la distancia más corta de esta área a una de nivel 2 (nivel de mayor presición) supera cierto umbral establecido, entonces se asume que el *blob* no contiene objetos de

interés y es descartado. Si en cambio se encuentra a una distancia menor al umbral establecido, el *blob* no es descartado pero tampoco se asume que contiene objetos de interés, por lo que pasa a ser procesado por un método de mayor presición.

- **Nivel 1** - Es el nivel intermedio. Los *blobs* que caen en este nivel no pueden ser descartados directamente, ya que la estadística marca que tienen buena posibilidad de contener al menos un objeto de interés. Al igual que los *blobs* en nivel 0, se calcula la distancia menor a un área de nivel 2. Si la distancia es menor que cierto umbral, se asume que el *blob* contiene al menos un objeto de interés, en caso contrario el *blob* no se descarta pero pasa a ser procesado por un método de mayor presición.
- **Nivel 2** - El nivel 2 es el nivel de mayor confianza. Si un *blob* clasifica en este nivel, se asume que el mismo contiene al menos un objeto de interés, por lo tanto, no es descartado.

A través de los parámetros de configuración `CONFIDENCE_LEVEL_0` y `CONFIDENCE_LEVEL_1` se definen las cotas superiores de los valores de frecuencia para el nivel 0 y el nivel 1, mientras que el valor para el nivel 2 es constante fijado en uno. Los valores de frecuencia se encuentran normalizados.

El parámetro de configuración `SQUARE_REGION_RADIUS` define la distancia alrededor de la región del *blob* que está siendo estudiado. Esta distancia es utilizada para buscar las regiones adyacentes con mayor valor de frecuencia y a partir de éstas tomar una decisión. En el ejemplo de la Figura 5.4 la distancia utilizada es de valor uno. Para los casos en los que no se desee utilizar las regiones adyacentes, lo que sería definir la distancia en cero, se utiliza el valor booleano `USE_SQUARE_REGION_FOR_VERIFY` en '*False*'. La Figura 5.7 resume en un diagrama de flujo el algoritmo de clasificación explicado en los puntos anteriores.

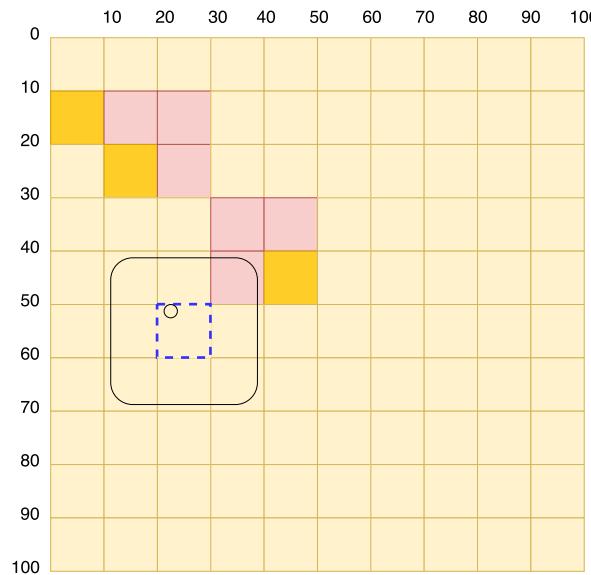


Figura 5.6: Histograma de frecuencia de tamaños de blob para un tiempo dado

A modo de ejemplo, en la Figura 5.6 se puede observar la representación de un histograma para un tiempo dado del sistema. Las áreas del histograma con nivel 2 se

representan con color rojo, las de nivel 1 en color anaranjado y las de nivel 0 en amarillo. El círculo en el punto (22, 51) representa la posición que ocupa un *blob* de 22 pixeles de ancho por 51 de alto en el histograma. El área con borde segmentado de color azul es el área que contiene la frecuencia de los *blobs* con ancho mayor igual a 20 y menor a 30 y altura mayor o igual a 50 y menor a 60. El contorno rectangular que rodea al área indica la distancia de este hacia los demás. En este caso, el *blob* es de nivel 0 pero se encuentra cercano a un área de nivel 2, por lo tanto el *blob* no es descartado y se procesa con algún algoritmo de precisión posteriormente.

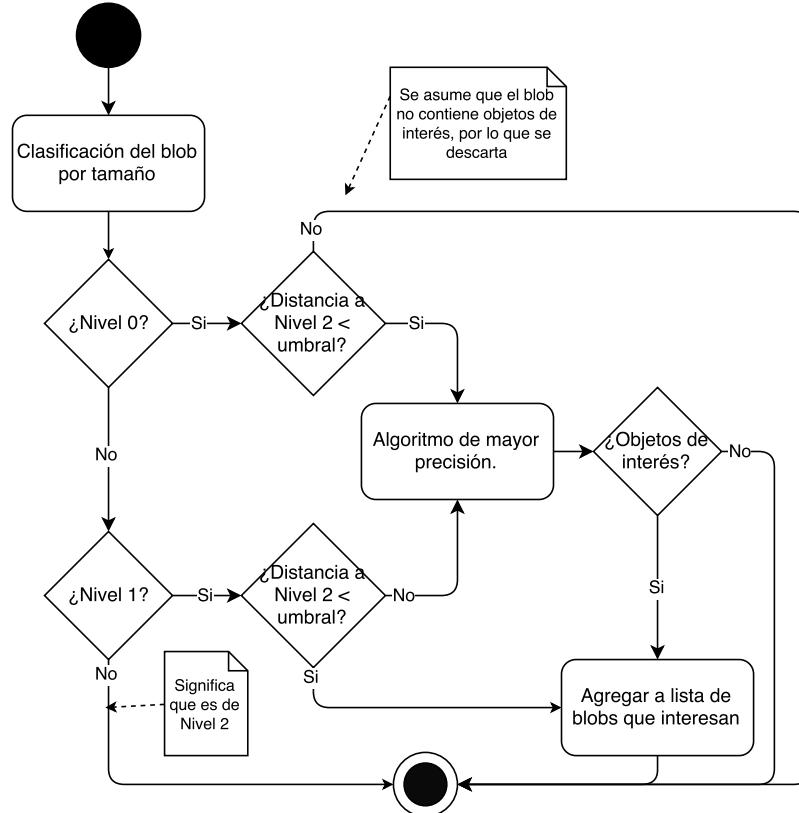


Figura 5.7: Diagrama de flujo del algoritmo de clasificación de *blobs* utilizando niveles de confianza.

Internamente se trabaja con dos histogramas bi-variados. El primero contiene la información de los *blobs* detectados e identificados con al menos un objetos de interés. Es decir, cuando se encuentra un *blob* que contiene al menos un objeto de interés, se incrementa el valor en el lugar del histograma correspondiente a los valores de ancho y alto del blob. El segundo histograma contiene información sobre los *blobs* detectados e identificados con únicamente un objeto de interés.

Para este método, el sistema permite utilizar modelos de histogramas pre-generados. También es posible crear un modelo y guardarlo para que pueda ser cargado en posteriores ejecuciones. Durante la ejecución, el modelo recibe actualizaciones de manera automática, lo cual es importante ya que en procesamientos de larga duración o en tiempo real las condiciones cambian con el tiempo. Por ejemplo, que los movimientos detectados estén más cerca o más lejos de la cámara provoca que el tamaño de los *blobs* varíe y el modelo debe ser capaz de adaptarse de forma autónoma. Cuando se permite la actua-

lización del modelo, la misma se realiza cada cierto tiempo definido en el parámetro de configuración CONFIDENCE_MATRIX_UPDATE_TIME. Durante el momento de actualización del modelo, todos los *blobs* son procesados por un algoritmo de detección considerado preciso y ningún *blob* es descartado por el método de frecuencias. Si se detecta que un lugar del histograma no se ha actualizado luego de varias iteraciones de actualización del modelo, se decrementa el valor de ese lugar. Tanto la creación como el uso del modelo pueden ser activados o desactivados según se requiera, asignando valores booleanos a los parámetros de configuración CREATE_MODEL y USE_MODEL.

Algoritmo 6 Algoritmo de clasificación de *blobs* por frecuencia de relación de aspecto en conjunto a un algoritmo de inteligencia computacional para la detección de personas

```

1: blobs_to_verify  $\leftarrow$  blobs
2: verified_blobs  $\leftarrow$  emptyList()
3: for each blob in blobs_to_verify do
4:   need_verify, discard  $\leftarrow$  histograms.checkBlob(blob)
5:   if not need_verify then
6:     blobs_to_verify.remove(blob)
7:   end if
8:   if not discard then
9:     verified_blobs.append(blob)
10:   end if
11: end for
12: if not isEmpty(blobs_to_verify) then
13:   persons, blobs_with_persons  $\leftarrow$  person_deteccor.detect(blobs_to_verify)
14:   histograms.update(persons)
15:   verified_blobs.extend(blobs_with_persons)
16: end if
```

Paralelización de la clasificación de blobs

La clasificación de *blobs* representa una carga computacional importante para el módulo Reconocimiento y seguimiento. Con el fin de agilizar esta estapa se implementan técnicas de paralelización aplicadas a la detección de personas. En concreto, se paraleliza el procesamiento de las diferentes instancias del método HOG.

Una vez que los *blobs* son filtrados, lo cual sucede en caso de que el parámetro de configuración USE_HISTOGRAMS_FOR_PERSON_DETECTION tenga el valor '*True*', el conjunto de *blobs* es convertido en un conjunto de imágenes correspondientes a cada *blob* y extraídas de la imagen principal. Estas imágenes son enviadas para ser procesadas de forma separada y paralela, siempre y cuando el valor del parámetro de configuración PERSON_DETECTION_PARALLEL_MODE tenga el valor '*True*'.

En el inicio del sistema se crea un '*pool*' de procesos. Un '*pool*' de procesos es un conjunto de procesos del sistema previamente inicializados y disponibles para procesar lo que se le solicite. Cada proceso en el '*pool*' ejecuta el método '*apply_single*' definido en el archivo '*person_detection_task.py*', contenido en el directorio '*black_boxes*' del módulo Reconocimiento y seguimiento.

El modelo de paralelismo está basado en procesos y no en hilos (*threads*). Esto se debe a una limitación tecnológica encontrada en el manejo de hilos por parte del lenguaje

Python. Python maneja los hilos a través de un GIL (*Global Interpreter Locker*) [67]. El GIL se encarga de que, para todo un sistema, se ejecute un hilo a la vez. Esto se debe a que el manejo de memoria en Python no es *thread-safe* (i.e., no funciona correctamente con hilos ejecutándose al mismo tiempo). Como consecuencia, no es posible un paralelismo real utilizando hilos y estos quedan condicionados a aplicaciones con un gran número de operaciones bloqueantes (e.g., aplicaciones con operaciones de entradas/salidas) ya que liberan la unidad de procesamiento mientras están bloqueadas y ésta es adjudicada a otros hilos que se encuentran a la espera del recurso.

Para el manejo de procesos se utilizó la biblioteca *multiprocessing* provista como biblioteca estándar en Python 3.4.x [68]. La biblioteca *multiprocessing* brinda facilidades en el uso de ‘*subprocess*’, estructura de Python que permite la creación y el lanzamiento de nuevos procesos desde un proceso padre, y la administración de los procesos creados mediante la clase *Pool* y sus métodos.

5.4.4. Seguimiento

La etapa de seguimiento de objetos de interés (*tracking*, en inglés) es la encargada de realizar el seguimiento de la posición de los objetos de interés a través del tiempo. Para realizar la tarea de seguimiento, el algoritmo implementado determina las correspondencias uno a uno, entre los *blobs* detectados en un *frame* t y los objetos de interés presentes en el *frame* anterior, $t - 1$.

Para determinar dichas correspondencias se utiliza el algoritmo Húngaro, desarrollado por Kuhn [69], sobre el cual se brinda información detallada en la sección A.6.3 del Apéndice A. Específicamente, se utiliza la implementación de la biblioteca *Munkres* [70] con adaptaciones agregadas que cubren las necesidades particulares del sistema. El algoritmo Húngaro recibe como entrada un conjunto de *blobs*, un conjunto de objetos y una función de costo; y devuelve una correspondencia entre ambos conjuntos. La correspondencia resultante es óptima respecto a la función de costo, además es una función en ambos sentidos (i.e. de los *blobs* a los objetos y viceversa), inyectiva (i.e. correspondencia uno a uno entre *blobs* y objetos), aunque no sobreyectiva (i.e. hay veces en las que no todos los *blobs* u objetos tienen correspondencia). Las primeras dos características (óptima e inyectiva) son inherentes al algoritmo Húngaro, mientras que la sobreyectividad, a pesar de que en la versión original siempre se cumple, en la versión con adaptaciones agregadas no lo hace, por dos motivos. Por un lado, mientras que en la versión original ambos conjuntos deben tener la misma dimensión, en la versión implementada se permiten diferentes dimensiones, de forma de abarcar casos en que hay más *blobs* que objetos (e.g. aparece algo nuevo en el *frame*) o más objetos que *blobs* (e.g. si un objeto desaparece); y, por el otro, en la versión implementada las funciones de costo tienen asociado un valor máximo a partir del cual una relación que lo supera no es válida, lo que puede resultar en *blobs* que no sean asignados a algún objeto. El sistema aplica el algoritmo Húngaro utilizando un conjunto de tres funciones de costo combinadas, con pesos configurables (e.g. un peso de 0.25 en la primera, 0.5 en la segunda, y 0.25 en la tercera).

Funciones de costo

Las funciones de costo utilizadas se basan en el cálculo de la distancia euclídea entre objetos y *blobs*, a partir de determinadas características:

- Posición previa
- Posición predicha
- Color

Posición previa Distancia entre la posición previa (i.e. en el *frame* anterior) de un objeto y la posición de un *blob*, con un máximo valor de validez dado por el parámetro de configuración `THRESHOLD_DISTANCE`.

Posición predicha Distancia entre la posición predicha de un objeto y la posición de un *blob*, siendo la posición predicha el valor de posición que se predijo que el objeto debe tener en el *frame* actual, con un máximo valor de validez dado por el parámetro de configuración `THRESHOLD_DISTANCE`.

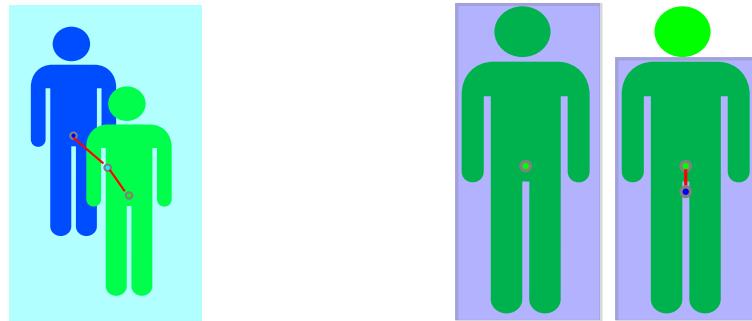
Color Distancia entre el color de un objeto y el color de un *blob*. Cuando el valor del parámetro de configuración `USE_HISTOGRAMS_FOR_TRACKING` es *False*, el color se almacena como el promedio del valor de color de los píxeles que componen al blob (i.e. píxeles resultantes de aplicar la imagen binaria procedente de la sustracción de fondo como máscara sobre la imagen del *blob*). Si, por el contrario, el valor del parámetro de configuración `USE_HISTOGRAMS_FOR_TRACKING` es *True*, el color se guarda como un histograma de color de los píxeles que componen al blob. En el primer caso la distancia se calcula de forma trivial, mientras que en el segundo caso se puede calcular mediante una de las funciones ofrecidas por las bibliotecas *OpenCV* y *ScyPy*[71], eligiendo esa función a través del parámetro de configuración `HISTOGRAM_COMPARISON_METHOD`. Para ambos casos existe un máximo valor de validez dado por el parámetro de configuración `THRESHOLD_COLOR`, el cual se debe configurar dependiendo del método a utilizar para el cálculo de la distancia. Además de ser almacenado utilizando el promedio o un histograma, un color puede ser almacenado utilizando el color que tiene mayor frecuencia en el *blob* (método para el cual el ruido en las imágenes sumado a la calidad baja de las mismas puede generar resultados imprecisos) o utilizando los *K* (e.g. tres) colores más representativos, obtenidos mediante el algoritmo de agrupación en particiones de *K-means* (*K-means clustering*) El método K-means no resulta apropiado para ejecución en tiempo real, ya que ocupa demasiado tiempo de procesamiento.

Los pesos de estas funciones se ajustan mediante parámetros, dependiendo de cuánta confianza se tiene en que la posición de los objetos y la de los *blobs* correspondientes son cercanas (e.g. si un objeto está ocluido por otros objetos durante demasiado tiempo, cuando aparezca su *blob* correspondiente, puede que este se haya alejado, por lo que no se tendría confianza en el seguimiento de ese objeto). Si no se tiene confianza, entonces se le da mayor peso a la tercera función de costo, ya que la apariencia (e.g. color) de los objetos, generalmente, es independiente del paso del tiempo. Particularmente, en el sistema se tienen dos configuraciones de pesos:

- `PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS`, utilizada en situaciones en las que se requiere un mayor peso en las funciones dependientes del tiempo (i.e. cuando la confianza en el seguimiento es alta)

- `SECONDARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS`, utilizada en situaciones en las que se requiere un mayor peso en las funciones independientes del tiempo (i.e. cuando la confianza en el seguimiento es baja)

Cuando se menciona la posición de un *blob* o de un objeto de interés, se hace referencia a las coordenadas de su centro geométrico en relación al *frame*. Debido a que el proceso de hallar un *blob* da un resultado inexacto (i.e. con ruido aleatorio producto de sombras, occlusiones y efectos propios del proceso), la posición del *blob* no siempre se ajusta de forma exacta a la del objeto (ver Figura 5.8). De esta forma, si los objetos tomaran directamente la posición de los *blobs*, la posición de los objetos haría *zig-zag* durante su seguimiento y dificultaría luego el hallazgo de patrones. Para mitigar el efecto del ruido, se utiliza un algoritmo llamado *filtro de Kalman* (*Kalman filter*, en inglés), creado por Kalman [36], sobre el cual se brinda información detallada en la sección A.6.1 del Apéndice A. Los filtros de Kalman son utilizados para el seguimiento de los objetos debido a que mantiene el estado de cada objeto, actualizándolo en cada *frame*, siguiendo un modelo de predicción y posterior corrección con una nueva medida $\mathbf{z} = (x, y)$ del objeto (i.e. posición del objeto).



(a) Caso de occlusión, con una línea roja marcando la diferencia entre el centro del *blob* y los centros de los objetos. Para representar ese ruido en las medidas, se define la desviación estándar para dichas diferencias, mediante el parámetro de configuración `NON_TRUTHFUL_MEASURES_NOISE_IN_PIXELS`.

(b) Dos posibles *blobs* que pueden surgir para un objeto. En el primero, el centro del *blob* coincide con el del objeto. En el segundo, hay una diferencia, marcada con una línea roja. Para representar ese ruido en las medidas, se define la desviación estándar para dichas diferencias, mediante el parámetro de configuración `MEASURES_NOISE_IN_PIXELS`.

Figura 5.8: Ejemplo del ruido que puede haber en las medidas, ya sea por occlusiones o por inexactitud del hallazgo de los *blobs*.

Por un lado, el estado \mathbf{x} es un vector compuesto por las variables del sistema. Debido a que en el sistema implementado los objetos de interés son personas y su velocidad puede variar, no alcanza con tener sólo variables de posición y velocidad, sino que las variables deben ser la posición $\vec{x} = (x, y)$, la velocidad $\vec{v} = (\dot{x}, \dot{y})$ y la aceleración $\vec{a} = (\ddot{x}, \ddot{y})$ del objeto (i.e. $\mathbf{x} = [x \ \dot{x} \ \ddot{x} \ y \ \dot{y} \ \ddot{y}]^t$). Cuando un filtro es creado para el seguimiento de un objeto, solamente se conoce la posición de este, por lo que el valor de posición del filtro es inicializado con la posición del *blob*, mientras que la velocidad y la aceleración son inicializadas con el vector nulo. Por otro lado, el modelo de predicción y corrección

hace uso de seis matrices, de las cuales cuatro tienen un valor fijo, una varía sus valores en cada iteración, y la sexta no se utiliza en el sistema implementado. Estas matrices son:

- Matriz de transición
- Matriz de ruido del proceso
- Matriz de transformación
- Matriz de ruido de medidas
- Matriz de covarianza del error
- Matriz de control

Matriz de transición Esta matriz \mathbf{F} representa la función de transición de estado que, aplicada al estado \mathbf{x} de un *frame* t , devuelve la predicción del estado en el *frame* siguiente, $t + 1$ (i.e. $\bar{\mathbf{x}} = \mathbf{F}\mathbf{x}$). A medida que avanza el tiempo, la velocidad de las personas puede variar, motivo por el cual \mathbf{F} es modelada con un sistema de *Newton* de aceleración constante definido por las ecuaciones $a_t = a_{t-1}$, $v_t = v_{t-1} + a_{t-1} \times \Delta t$, $x_t = x_{t-1} + v_{t-1} \times \Delta t + a_{t-1} \times \frac{\Delta t^2}{2}$, siendo el tiempo medido en segundos. Por lo tanto, la matriz \mathbf{F} queda de la forma:

$$\mathbf{F} = \begin{bmatrix} 1 & \Delta t & \frac{\Delta t^2}{2} & 0 & 0 & 0 \\ 0 & 1 & \Delta t & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \Delta t & \frac{\Delta t^2}{2} \\ 0 & 0 & 0 & 0 & 1 & \Delta t \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Matriz de ruido del proceso La matriz de ruido del proceso \mathbf{Q} indica cuál es la covarianza del error en la predicción entre un *frame* y el siguiente. Por simplicidad, se asume que el ruido es un proceso *Wiener* de tiempo discreto, y que es constante para cada período de tiempo. Además, se asume también que el ruido en el eje horizontal x es independiente del ruido en el eje vertical y , por lo que la covarianza entre cualquier variable del eje horizontal x (i.e. x , \dot{x} y \ddot{x}) y cualquiera del eje vertical y es cero. Entonces, la matriz \mathbf{Q} se define con ayuda de la función *Q_discrete_white_noise* de la biblioteca *FilterPy* [72], utilizando una varianza dada por el parámetro de configuración **VARIANCE_OF_MODEL_CHANGE_BETWEEN_STEPS**, que representa cuánto se cree que el modelo varía entre un paso y el siguiente (i.e. entre t y $t + 1$). Por ejemplo, si se utiliza una varianza de 0,15, la matriz \mathbf{Q} queda definida de la siguiente manera:

$$\mathbf{Q} = \begin{bmatrix} 0,0375 & 0,075 & 0,075 & 0 & 0 & 0 \\ 0,075 & 0,15 & 0,15 & 0 & 0 & 0 \\ 0,075 & 0,15 & 0,15 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0,0375 & 0,075 & 0,075 \\ 0 & 0 & 0 & 0,075 & 0,15 & 0,15 \\ 0 & 0 & 0 & 0,075 & 0,15 & 0,15 \end{bmatrix}$$

Matriz de transformación La matriz \mathbf{H} representa la función que transforma un estado en una medida (i.e. transforma $\mathbf{x} = [x \ \dot{x} \ \ddot{x} \ y \ \dot{y} \ \ddot{y}]^t$ en $\mathbf{z} = [x \ y]^t$). Como la unidad de medida (i.e. píxeles) es la misma tanto en el estado \mathbf{x} como en la medida \mathbf{z} , la matriz \mathbf{H} es:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \end{bmatrix}, \text{ tal que } \mathbf{x}\mathbf{H} = \mathbf{z}$$

Si las variables del estado \mathbf{x} estuviesen, por ejemplo, en metros habría que cambiar los valores en uno de la matriz por el valor que corresponda para la conversión entre metros y píxeles.

Matriz de ruido de medidas La matriz \mathbf{R} representa el ruido de las medidas $\mathbf{z} = (x, y)$, e indica cuál es la covarianza del error en las mismas. Como la posición de un objeto en el eje horizontal es independiente de la posición en el eje vertical, la covarianza entre ambas es cero. Por lo tanto, la matriz \mathbf{R} presenta valores únicamente en la diagonal principal (varianza de x y varianza de y). Estos dos valores son iguales y controlados por el parámetro de configuración `MEASURES_NOISE_IN_PIXELS` en casos en que la medida \mathbf{z} es confiable, y por el parámetro `NON_TRUTHFUL_MEASURES_NOISE_IN_PIXELS` en casos en que no lo es. En ambos casos, el parámetro de configuración representa tanto la desviación estándar de la posición en el eje horizontal σ_x como la desviación estándar de la posición en el eje vertical σ_y . Por lo tanto, la varianza se calcula como el valor del parámetro elevado al cuadrado. Entonces, la matriz \mathbf{R} queda definida como:

$$\mathbf{R} = \begin{bmatrix} \sigma_x^2 & 0 \\ 0 & \sigma_y^2 \end{bmatrix}$$

Matriz de covarianza del error La matriz de covarianza del error \mathbf{P} , representa la precisión estimada de la predicción en el estado actual. Cuando un filtro es creado para el seguimiento de un objeto, la matriz \mathbf{P} se inicializa con una diagonal, conteniendo la varianza del error de las variables del estado. Como al comienzo la posición del objeto se conoce con relativa certeza, la varianza del error de las variables de posición es bastante chica, mientras que la varianza del error de las variables de velocidad y de aceleración es grande. Estas varianzas iniciales se configuran mediante los parámetros `INITIAL_ERROR_VARIANCE_OF_POSITION` para la posición (σ_x^2 y σ_y^2), `INITIAL_ERROR_VARIANCE_OF_VELOCITY` para la velocidad ($\sigma_{\dot{x}}^2$ y $\sigma_{\dot{y}}^2$) e `INITIAL_ERROR_VARIANCE_OF_ACCELERATION` para la aceleración ($\sigma_{\ddot{x}}^2$ y $\sigma_{\ddot{y}}^2$). Es recomendable iniciar el valor del parámetro correspondiente a la posición con un valor bajo, y los valores de la velocidad y la aceleración teniendo en cuenta los máximos que estos pueden tomar (e.g. si la velocidad máxima de un objeto en la escena es de 30 píxeles por unidad de tiempo, entonces la desviación estándar σ es $\frac{30}{3} = 10$ y la varianza σ^2 es $10^2 = 100$). De esta forma, la matriz \mathbf{P} se inicializa con los siguientes valores:

$$\mathbf{P} = \begin{bmatrix} \sigma_x^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_{\dot{x}}^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_{\ddot{x}}^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_y^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_{\dot{y}}^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_y^2 \end{bmatrix}$$

Luego, en cada iteración de los filtros de *Kalman* los valores de la matriz \mathbf{P} se van actualizando e, idealmente, los valores tienden a cero.

Matriz de control La matriz \mathbf{U} representa la función de control. Debido a que en el sistema implementado los objetos de interés no tienen entradas de control (e.g. no existen eventos que hagan que los objetos giren hacia un lado o hacia el otro), esta matriz se inicializa como la matriz nula.

Cada iteración de los filtros de *Kalman* contiene dos etapas denominadas etapa de predicción y etapa de corrección. La etapa de predicción utiliza la matriz de transición para predecir el siguiente estado (i.e. $\bar{\mathbf{x}} = \mathbf{F}\mathbf{x}$), mientras que la etapa de corrección utiliza la información de una medida \mathbf{z} y la de las matrices de ruido para obtener la corrección de la posición predicha del objeto, la cual se ubica entre la posición predicha y la medida, a una distancia que depende de las matrices de ruido (i.e. se ubica más cerca de la medida si el ruido de estas es menor que el del proceso, o más lejos si es mayor). A su vez, en la segunda etapa, el algoritmo tiene un parámetro denominado *Kalman gain* que va actualizando, y gracias al cual logra adaptar el seguimiento al objeto, de forma suave (i.e. sin *zig-zag*).

A pesar de la mejora que ofrecen los filtros en el efecto de *zig-zag* de las trayectorias, existen irregularidades que no pueden solucionar. Esta limitación se debe a que el modelo de movimiento no es perfecto y las occlusiones dejan a los filtros, en ocasiones, sin medidas para poder corregir las predicciones. Con el fin de mitigar estas irregularidades, se aplica un suavizado de los filtros de *Kalman* (*Kalman filter smoothing*) mediante un retardo fijo (*fixed-lag smoother*), que es el tipo más apropiado de los tres posibles. En este tipo de suavizado, en cada iteración de los filtros se mejora el estado \mathbf{x}_{t-N} tomando en cuenta los datos aportados por los últimos N estados, a cambio de una demora de $\frac{N}{FPS}$ segundos en el reconocimiento de patrones. La utilización del suavizado del filtro y el tamaño N son configurables mediante los parámetros de configuración `KALMAN_FILTER_TYPE` y `KALMAN_FILTER_SMOOTH_LAG` respectivamente. Además del tipo de suavizado mencionado anteriormente, existen el suavizado de intervalo fijo (*fixed-interval smoothing*) y el suavizado de punto fijo (*fixed-point smoothing*). El suavizado de intervalo fijo realiza el suavizado más óptimo, aunque necesita tener todos los datos del filtro antes de calcular los estimados, por lo que se podría ejecutar sobre los datos obtenidos en un vídeo, pero no en tiempo real. Por otro lado, para el suavizado de punto fijo, se toma un punto fijo en el tiempo, para utilizar todos los estados posteriores con el fin de mejorar la predicción de dicho punto. Específicamente para el filtro de *Kalman* se utiliza la implementación que ofrece la biblioteca *OpenCV*, mientras que para el filtro suavizado se utiliza la implementación de la biblioteca *FilterPy*.

Para mantener la información sobre los distintos objetos y su seguimiento, y para manejar las occlusiones, se utilizan dos estructuras: *tracklets* y grupos.

Tracklets El *tracklet* es la estructura básica utilizada para realizar el seguimiento. Cada *tracklet* es asociado al seguimiento de un único objeto y, durante toda su vida, almacena y actualiza información relevante para el seguimiento del mismo. Esta información incluye la posición, el color, el número de *frame* en que fue creado, el último número de *frame* en el que su objeto no estaba oculto y el número de grupo al que está asociado, entre otros.

Grupos Cuando ocurren occlusiones múltiples (entre varios objetos), aparecen varios *tracklets* con un único *blob* cercano. Si este *blob* fuera asociado con una sola de esas estructuras, el resto de los *tracklets* quedarían en un estado similar al que tomarían si su correspondiente objeto hubiera desaparecido. Como no es correcto tratar a los *tracklets* de forma diferente cuando en realidad pertenecen a un mismo conjunto producto de la occlusion, se creó el concepto de *grupos*. Estos son estructuras en donde se almacenan *tracklets* y, en cada *frame*, se asocian *blobs*. Lo ideal es que cada grupo contenga únicamente un *tracklet* y un *blob* asociado, caso en el cual el *blob* representa al objeto que es seguido por el *tracklet*. Sin embargo, debido a las occlusiones se da la existencia de grupos con varios *tracklets* y uno o más *blobs* asociados. Por un lado, el caso del grupo de un único *blob* asociado a varios *tracklets* representa una occlusion múltiple, donde hay varios objetos en la imagen que se acercaron lo suficiente como para formar un único *blob*. Por otro lado, el caso del grupo de varios *blobs* asociados a varios *tracklets* se trata de objetos que estaban en occlusion múltiple y se están separando. En este último caso, el sistema divide el grupo, de manera de obtener un único *blob* por grupo.

En cada iteración del algoritmo de seguimiento, los *tracklets* pueden actualizarse o ser eliminados, dependiendo de cómo está conformado el grupo al que pertenecen y de cuánto tiempo estuvieron en él, así como del tiempo que llevan en el sistema.

Especificamente, un *tracklet* puede ser eliminado por tres motivos:

- El objeto seguido es producto de *blobs* fantasma. Si fue creado hace poco tiempo (i.e. menos segundos que los que dicta el parámetro `MIN_SECONDS_TO_BE_ACCEPTED_IN_GROUP`) y queda en un grupo que no es uno a uno, se considera que el objeto al que sigue es producto de *blobs* fantasma (e.g. ruido proveniente de la cámara o de etapas anteriores).
- Poca confianza en el seguimiento del objeto. Si hace mucho tiempo que no queda en un grupo uno a uno (i.e. más segundos que los que dicta el parámetro `MAX_SECONDS_WITHOUT_UPDATE`), no se tiene confianza suficiente en la validez de su seguimiento.
- El objeto desaparece definitivamente de la escena. Si hace mucho tiempo que no queda en un grupo con un *blob* (i.e. más segundos que los que dicta el parámetro `MAX_SECONDS_WITHOUT_ANY_BLOB`), se considera que el objeto al que sigue desapareció definitivamente de la escena (e.g. al entrar por una puerta o al salir del alcance de la cámara). También, si es asociado a un grupo sin *blob* y la predicción de su posición cae fuera del alcance de la cámara (i.e. fuera de la imagen), se considera que el objeto al que seguía se fue de la escena.

Cuando un *tracklet* no es eliminado, su información debe ser actualizada. Con el fin de actualizar la información de los *tracklets*, existen tres niveles de actualización: corrección con confianza máxima, corrección con confianza mínima y sólo predicción.

Corrección con confianza máxima El caso de corrección con confianza máxima es considerado el ideal de las correcciones. Si el *tracklet* es asociado a un grupo uno a uno, entonces el *blob* del grupo representa al objeto al que sigue el *tracklet*, por lo que el este *tracklet* se actualiza con la información de posición y apariencia del *blob*. Esta actualización implica corregir la predicción de la posición con la posición del *blob* (i.e. medida) y actualizar los datos de apariencia como el color, tamaño y posición del rectángulo que representa al *blob*.

Corrección con confianza mínima Este tipo de corrección ocurre en el caso de que un objeto se encuentre en oclusión múltiple durante mucho tiempo. Si el *tracklet* en cuestión está asociado a un grupo con varios *tracklets* (i.e. uno a varios, producto de una oclusión múltiple) y desde hace mucho tiempo que no está uno a uno (i.e. más segundos que los que dicta el parámetro `MAX_SECONDS_TO_PREDICT_POSITION`), la predicción de posición ya no es confiable para el objeto al que sigue. Por lo tanto, se corrige su predicción de posición con la posición del *blob* producto de la oclusión (i.e. el *blob* del grupo), con la intención de que el *tracklet* no se aleje de la zona en donde se encuentra el objeto al que sigue.

Sólo predicción La actualización basada solo en predicción es la situación de caso ideal cercano en el tiempo. Si no está en un grupo uno a uno, pero lo estuvo hace poco tiempo (i.e. menos segundos que los que dicta el parámetro `MAX_SECONDS_TO_PREDICT_POSITION`), se asume que la predicción de la posición aún es confiable y no se hace ninguna corrección a la misma. Esto es útil en el caso de un objeto que pasa por detrás de una columna, o que se cruza con otro objeto (i.e. oclusión múltiple) por un período corto de tiempo, ya que se puede seguir su movimiento a pesar de que no genera un *blob*, gracias a la predicción realizada antes.

El Algoritmo 7 muestra un pseudocódigo que describe cómo el sistema realiza el seguimiento de los objetos, *frame* a *frame*. Este algoritmo comienza aplicando la combinación de funciones de costo dada por el parámetro de configuración `PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS` sobre el algoritmo Húngaro, para determinar las correspondencias de costo óptimo entre los *blobs* pertenecientes al *frame* actual y los *tracklets* existentes. En esas correspondencias, cada *blob* es asignado a uno o a ningún *tracklet*. Si un *blob* es asociado a un determinado *tracklet*, entonces se lo agrega al grupo al cual pertenece ese *tracklet*. En caso contrario, significa que el *blob* representa a un nuevo objeto en el sistema, por lo que se crea un nuevo *tracklet* contenido la información del nuevo objeto y un nuevo grupo, asignando a este último el *blob* y el *tracklet*.

El paso siguiente es recorrer la lista de grupos para los cuales aún no se asignó un *blob*. Para cada uno de ellos, se busca un *blob* que se solape con los *tracklets* del grupo y sea el *blob* más cercano a los *tracklets* o, de no haberlo, se busca el *blob* más cercano a la posición de los *tracklets* del grupo, teniendo en cuenta una distancia máxima definida por el parámetro de configuración `THRESHOLD_DISTANCE`. Si fue posible hallarlo, entonces

se mueven los *tracklets* al grupo al que está asociado ese *blob*. Este caso se da cuando dos o más *blobs* del *frame* $t - 1$ se mezclan, generando oclusión y formando, en el *frame* t , un único *blob*.

Luego, cada grupo es procesado según la cantidad de *blobs* que fueron asignados al mismo. De esta forma quedan definidos tres posibles casos de estudio: que el grupo no tenga *blobs* asignados, que contenga solamente un *blob*, o que contenga dos o más *blobs* asignados.

Algoritmo 7 Algoritmo de seguimiento de objetos de interés

```

1: blobs  $\leftarrow$  blobs_en_el_frame
2: if not esta_vacio(blobs) then
3:   grupos.borrar_blobs_asignados()
4:   asignaciones, costos  $\leftarrow$  algoritmo_hungaro_primario(blobs, tracklets)
5:   for each blob in blobs do
6:     if asignaciones.esta_asignado(blob) then
7:       tracklet  $\leftarrow$  asignaciones.asignacion_de(blob)
8:       agregar_blob_a_grupo(blob, tracklet.grupo())
9:     else
10:      tracklet  $\leftarrow$  nuevo_tracklet(blob)
11:      tracklets.agregar(tracklet)
12:      grupo  $\leftarrow$  nuevo_grupo(tracklet, blob)
13:      grupos.agregar(grupo)
14:    end if
15:   end for
16:   for each grupo in grupos.sin_blob_asignado(grupos) do
17:     blob  $\leftarrow$  obtener_blob_contenedor_o_mas_cercano_al_grupo(grupo, blobs)
18:     if blob.existe() then
19:       grupo.mover_tracklets_a(blob.grupo())
20:     end if
21:   end for
22:   for each grupo in grupos do
23:     if grupo.blobs.largo() == 0 then
24:       for each tracklet in grupo.tracklets do
25:         if tracklet.objeto_desaparecio_momentaneamente() then
26:           tracklet.actualizar_con_prediccion_de_posicion()
27:         else
28:           tracklets.eliminar(tracklet)
29:         end if
30:       end for
31:     else if grupo.blobs.largo() == 1 then
32:       if grupo.tracklets.largo() == 1 then
33:         grupo.tracklets[0].actualizar_con_datos_del_blob(grupo.blobs[0])
34:       else if grupo.tracklets.largo() > 1 then
35:         {llamada a Algoritmo 8}
36:         sub_algoritmo_grupo_uno_a_muchos(grupos, tracklets, grupo)
37:       end if
38:     else
39:       if group.tracklets.length() ≥ group.blobs.length() then
40:         {llamada a Algoritmo 9}
41:         sub_algoritmo_grupo_muchos_a_muchos(grupos, tracklets, grupo)
42:       end if
43:     end if
44:     if grupo.tracklets.largo() == 0 then
45:       grupos.eliminar(grupo)
46:     end if
47:   end for
48: end if

```

Caso 1 - El grupo no tiene *blobs* asignados Esto puede corresponder a una de tres situaciones respecto al objeto que sigue cada *tracklet*:

- El objeto desapareció definitivamente de la escena o la predicción de la posición del *tracklet* cae fuera del *frame*
- el objeto desapareció momentáneamente de la escena
- El *tracklet* era resultado de *blobs* fantasma

Por un lado, la posición de los *tracklets* cuyos objetos desaparecieron momentáneamente (i.e. están sin confianza máxima desde hace poco tiempo) es actualizada con la predicción. Por otro lado, el resto de los *tracklets* son eliminados del sistema, debido a que se asume que los objetos a los que seguían no van a volver a aparecer.

Caso 2 - El grupo tiene un único *blob* asignado Si el grupo tiene un único *blob* asignado y además contiene un único *tracklet*, se está ante el caso ideal en el que se realiza la asignación uno a uno entre un *tracklet* y un *blob* (i.e. el *blob* representa al objeto que el *tracklet* sigue), por lo cual se está en el caso de confianza máxima en el *blob*, y la información del *tracklet* es actualizada utilizando todos los datos del *blob*. En cambio, si el grupo contiene más de un *tracklet*, significa que existen varios *tracklets* para un único *blob*, encontrándose en el caso en que varios objetos están ocluidos. En este último caso, primero se eliminan los *tracklets* cuyo objeto seguido es producto de *blobs* fantasma. Segundo, si aún quedan *tracklets*, se eliminan los *tracklets* para los cuales se tiene poca confianza en el seguimiento, dejando vivo al *tracklet* más antiguo. Luego de ese proceso de eliminación, si no quedan *tracklets* en el grupo, éste es eliminado. Por el contrario, si solamente quedó un *tracklet*, se está en el caso ideal, por lo que se actualiza la información del *tracklet* con los datos del *blob*. Finalmente, si quedó más de un *tracklet* en el grupo, se actualiza la posición de cada uno con la predicción sin corregir si hace poco tiempo que tenía confianza máxima, o con la predicción corregida con la posición del *blob* en el caso que tenga confianza mínima. El Algoritmo 8 muestra el pseudocódigo correspondiente a este método, mientras que la Figura 5.9 presenta un ejemplo de seguimiento en donde se generan ambos tipos de grupo.

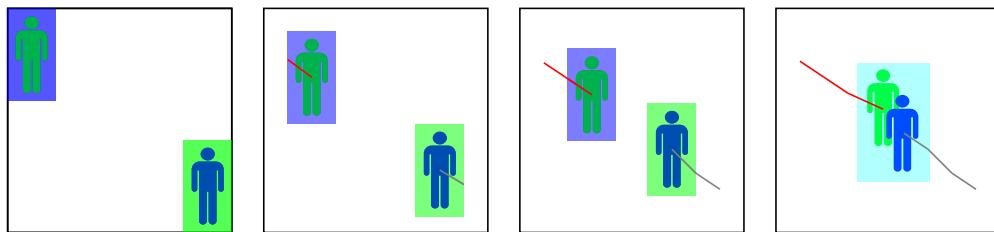


Figura 5.9: Secuencia de *frames* en la que se realiza el seguimiento de dos objetos (i.e. imágenes de personas) para los que se detectan *blobs* (i.e. rectángulos). En el último frame, ocurre una múltiple oclusión, donde aparece un grupo con dos *tracklets* (i.e. líneas) y un *blob*.

Algoritmo 8 Sub-algoritmo para grupos de un blob a muchos tracklets

```

1: grupos  $\leftarrow$  grupos_del_sistema
2: tracklets  $\leftarrow$  tracklets_del_sistema
3: grupo  $\leftarrow$  grupo_uno_a_muchos
4: for each tracklet in grupo.tracklets do
5:   if tracklet.objecto_producto_de_blobs_fantasma() then
6:     tracklets.eliminar(tracklet)
7:   end if
8: end for
9: for each tracklet in grupo.tracklets do
10:  if not es_el_tracklet_mas_antiguo(tracklet, grupo.tracklets) then
11:    if tracklet.tiene_poca_confianza_en_seguimiento() then
12:      tracklets.eliminar(tracklet)
13:    end if
14:  end if
15: end for
16: if grupo.tracklets.largo() == 0 then
17:   grupos.eliminar(grupo)
18: else if grupo.tracklets.largo() == 1 then
19:   grupo.tracklets[0].actualizar_con_datos_del_blob(grupo.blobs[0])
20: else
21:   for each tracklet in grupo.tracklets do
22:     if tracklet.objecto_desaparecio_momentaneamente() then
23:       tracklet.actualizar_con_prediccion_de_posicion()
24:     else
25:       tracklet.actualizar_con_confianza_minima_en_blob(grupo.blobs[0])
26:     end if
27:   end for
28: end if

```

Caso 3 - El grupo contiene más de un blob asignado Debido a que al comienzo de la iteración cada *blob* se asigna a un *tracklet* de forma inyectiva, en el grupo hay, como mínimo, la misma cantidad de *tracklets* que de *blobs*. Este caso representa un *blob* que contenía objetos ocluidos (i.e. los *tracklets*) que se están separando. Como cada *blob* representa un conjunto de uno o más objetos, para no perder información de una iteración a otra, se debe separar cada *blob* en un nuevo grupo y asociar los *tracklets* correspondientes. Se comienza aplicando el algoritmo Húngaro, utilizando la función de costo dada por el parámetro de configuración `PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS`, para encontrar las correspondencias de costo óptimo entre los *blobs* y aquellos *tracklets* que están sin confianza máxima desde hace poco tiempo. A partir de la información obtenida, y cuidando de nunca dejar un grupo solamente con *tracklets*, se crean nuevos grupos con aquellas parejas de *blob* y *tracklet* para las cuales fue posible encontrar una asignación de costo óptimo, a la vez que se eliminan del grupo actual. En esos casos, la información del *tracklet* es actualizada utilizando toda la información del *blob*. Si hay más *tracklets* que *blobs*, el *blob* que tiene la correspondencia de peor costo es el que se queda con los *tracklets* restantes. Si luego de aplicado el procedimiento anterior aún queda más de un *blob* en el grupo, significa que la primera función de costo utilizada en el algoritmo Húngaro

no fue del todo efectiva para determinar todas las correspondencias. Esto ocurre cuando hay *tracklets* que están con confianza mínima en sus seguimientos, por lo que es necesario utilizar una función de costo que presente mayor independencia del tiempo (e.g. funciones de costo basadas en apariencia, como lo es la diferencia de color). Por lo tanto, se aplica nuevamente el algoritmo Húngaro, esta vez utilizando la función de costo dada por el parámetro de configuración `SECONDARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS`. Nuevamente, a partir de los resultados se realiza el mismo procedimiento que con los de la anterior aplicación del algoritmo. En la Figura 5.10 se presenta un ejemplo en el que alcanza con utilizar la primer función de costo mientras que, en la Figura 5.11 es necesario utilizar la segunda función de costo. Una vez finalizada la segunda aplicación del algoritmo Húngaro, puede suceder que aún quede un único *blob* y más de un *tracklet* en el grupo. En este caso, se realiza lo mismo que en el punto anterior (i.e. grupo con un único *blob* y varios *tracklets*). El Algoritmo 9 muestra un pseudocódigo para este método.

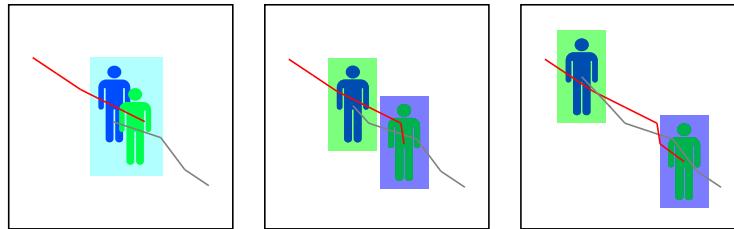


Figura 5.10: Alternativa de desenlace para la secuencia presente en la figura 5.9, donde los objetos continúan con la misma dirección. En el segundo *frame* aparece un grupo con dos *tracklets* y dos *blobs*. Dado que pasó poco tiempo desde que los *tracklets* estaban uno a uno, alcanza con aplicar el algoritmo húngaro primario (i.e. con funciones de costo dependientes del tiempo) para asociar cada *tracklet* con el *blob* correcto.

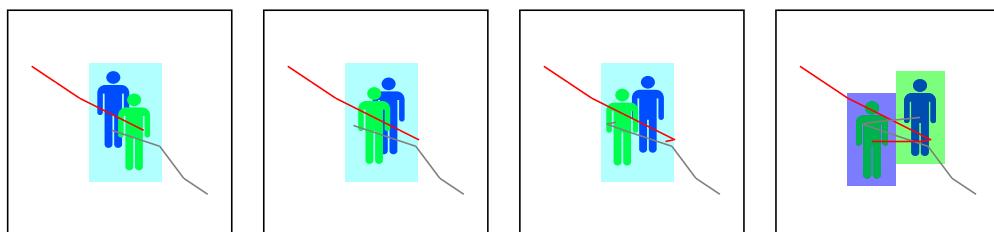


Figura 5.11: Alternativa de desenlace para la secuencia presente en la figura 5.9, donde los objetos cambian de dirección. En el cuarto *frame* aparece un grupo con dos *tracklets* y dos *blobs*. Dado que pasó mucho tiempo desde que los *tracklets* estaban uno a uno, se aplica el algoritmo húngaro secundario (i.e. con funciones de costo independientes del tiempo, como comparación por color) para asociar cada *tracklet* con el *blob* correcto.

Algoritmo 9 Sub-algoritmo para grupos de muchos blobs a muchos tracklets

```

1: grupos  $\leftarrow$  grupos_del_sistema
2: tracklets  $\leftarrow$  tracklets_del_sistema
3: grupo  $\leftarrow$  grupo_uno_a_muchos
4: blobs  $\leftarrow$  grupo.blobs
5: tracklets_a_comparar  $\leftarrow$  tracklets_con_caso_ideal_cercano(grupo.tracklets)
6: mantener_peor_blob  $\leftarrow$  blobs.largo()  $\leqslant$  tracklets_a_comparar.largo()
7: asignaciones, costos  $\leftarrow$  algoritmo_hungaro_primario(blobs, tracklets_a_comparar)
8: for each blob, tracklet in asignaciones do
9:   if not mantener_peor_blob or not tiene_peor_costo(blob, costos) then
10:    tracklet.actualizar_con_datos_del_blob(blob)
11:    grupos.agregar(nuevo_grupo(tracklet, blob))
12:    grupo.blobs.eliminar(blob)
13:    grupo.tracklets.eliminar(tracklet)
14:   end if
15: end for
16: if blobs.largo() > 1 then
17:   mantener_peor_blob  $\leftarrow$  blobs.largo()  $\leqslant$  grupo.tracklets.largo()
18:   asignaciones, costos  $\leftarrow$  algoritmo_hungaro_secundario(blobs, grupo.tracklets)
19:   for each blob, tracklet in asignaciones do
20:     if not mantener_peor_blob or not tiene_peor_costo(blob, costos) then
21:       tracklet.actualizar_con_datos_del_blob(blob)
22:       grupos.agregar(nuevo_grupo(tracklet, blob))
23:       grupo.blobs.eliminar(blob)
24:       grupo.tracklets.eliminar(tracklet)
25:     end if
26:   end for
27: end if
28: if grupo.tracklets.largo() > 1 then
29:   {llamada a Algoritmo 8}
30:   sub_algoritmo_grupo_uno_a_muchos(grupos, tracklets, grupo)
31: end if

```

5.4.5. Parámetros de configuración del módulo Reconocimiento y seguimiento

Esta sección lista los parámetros de configuración que afectan el funcionamiento del modulo Reconocimiento y seguimiento. Estos parámetros se encuentran en el archivo *trackermaster.conf* dentro del directorio *trackermaster* junto a sus valores por defecto. A continuación se presentan diferentes listas que agrupan los parámetros según el propósito que los mismos tengan en el sistema.

Sustracción de fondo

- **GAUSSIANBLUR_SIZE_(X,Y)**: vector de números enteros que indican el tamaño del núcleo Gaussiano. Los valores deben ser positivos e impares.
- **ERODE_SIZE_(X,Y)** y **ERODE_TIMES**: **ERODE_SIZE(X,Y)** es un vector de números

enteros positivos que indican el tamaño del núcleo utilizado para la operación de erosión. ERODE_TIMES indica la cantidad de veces que se aplica la operación.

- DILATE_SIZE_(X,Y) y DILATE_TIMES: DILATE_SIZE es un vector de números enteros positivos que indican el tamaño del núcleo utilizado para la operación de dilatación. DILATE_TIMES indica la cantidad de veces que se aplica la operación.
- HISTORY: número entero que indica la cantidad de *frames* que afectan el modelo de fondo.
- DIST_2_THRESHOLD: número entero que representa la distancia en píxeles utilizada como umbral en el método *k*-NN para decidir si un píxel se encuentra suficientemente cerca de una muestra de datos como para ser considerado parte de esta.
- N_SAMPLES: número entero que indica la cantidad de muestras de datos presentes en el modelo del fondo. El parámetro es utilizado en el por el método k-NN.
- KNN_SAMPLES: número entero que indica la cantidad de vecinos (la *k* en el método k-NN). *k* es la cantidad de muestras que deben estar dentro del umbral de distancia definido en DIST_2_THRESHOLD para poder afirmar que el píxel pertenece al modelo del fondo. El parámetro es utilizado en el por el método k-NN.
- DETECT_SHADOWS: valor booleano que activa o desactiva la detección de sombras. El parámetro es utilizado tanto por el método MOG2 como por el método k-NN.
- SHADOW_THRESHOLD: valor decimal que define el valor del umbral para las sombras. Un píxel es tomado como sombra cuando es más oscuro que el fondo. El umbral define cuánto más oscura puede ser la sombra. Por ejemplo, un valor de 0.5 indica que si un píxel es más de dos veces más oscuro, es sombra.
- USE_BSUBTRACTOR_KNN: valor booleano que indica cual es el método que se utiliza para la sustracción de fondo. Si el valor es ‘True’ el método *k*-NN es utilizado, de lo contrario se utiliza el método MOG2.

Detección de *blobs*

- MIN_THRESHOLD, MAX_THRESHOLD y THRESHOLD_STEP: números enteros que definen los valores del umbral utilizado para crear imágenes binarias a partir de la imagen recibida por el componente de detección de *blobs*. Estos umbrales comienzan en el valor de MIN_THRESHOLD y son incrementados de acuerdo al valor de THRESHOLD_STEP, hasta alcanzar el valor MAX_THRESHOLD. Las múltiples imágenes binarias creadas por el método *SimpleBlobDetector*, de acuerdo a los valores de estos parámetros, son utilizadas para la detección de *blobs*. Teniendo en cuenta que la etapa anterior a la detección de *blobs* (la etapa de sustracción de fondo) fue implementada de tal manera que su salida ya es una imagen binaria, estos valores no tienen consecuencia alguna sobre la imagen que reciben. En caso de que el algoritmo de sustracción de fondo sea suplantado por otro que retorne una imagen no binaria, entonces los valores de estos parámetros serán tenidos en cuenta.
- MIN_DIST_BETWEEN_BLOBS: número entero que indica la distancia en píxeles máxima entre el centro de dos *blobs* para que estos se fusionen en uno. El parámetro es utilizado por el método *SimpleBlobDetector*.

- **FILTER_BY_COLOR** y **BLOB_COLOR**: si el valor de **FILTER_BY_COLOR** es '*True*' entonces se activa el filtrado de *blobs* por color y el valor entero de **BLOB_COLOR** es tomado como umbral. En el caso del sistema implementado, la entrada ya es una imagen binaria, por lo tanto si **FILTER_BY_COLOR** está activado, **BLOB_COLOR** tiene el valor 255. De otro modo, no se detectan blobs.
- **FILTER_BY_AREA, (MIN, MAX)_AREA**: si el valor de **FILTER_BY_AREA** es '*True*' entonces se activa el filtrado de *blobs* por área y los valores enteros de **MIN_AREA** y **MAX_AREA** son tomados como límite inferior y superior para el filtrado.
- **FILTER_BY_CIRCULARITY** y **(MIN, MAX)_CIRCULARITY**: si el valor de **FILTER_BY_CIRCULARITY** es '*True*', entonces se activa el filtrado de *blobs* por circularidad (evaluando qué tan similar es la forma del *blob* a la de un círculo) y los valores decimales de **MIN_CIRCULARITY** y **MAX_CIRCULARITY** son tomados como límite inferior y superior para la circularidad aceptada.
- **FILTER_BY_CONVEXITY** y **(MIN, MAX)_CONVEXITY**: si el valor de **FILTER_BY_CONVEXITY** es '*True*' entonces se activa el filtrado de *blobs* por convexidad (que tan convexa es la forma del *blob*) y los valores decimales de **MIN_CONVEXITY** y **MAX_CONVEXITY** son tomados como límite inferior y superior para la convexidad aceptada.
- **FILTER_BY_INERTIA, (MIN, MAX)_INERTIA**: si el valor de **FILTER_BY_INERTIA** es '*True*' entonces se activa el filtrado de *blobs* por su relación de inercia (evaluando qué tan ovalada es la forma del *blob*) y los valores decimales de **MIN_INERTIA** y **MAX_INERTIA** son tomados como límite inferior y superior para el valor de la relación aceptada.
- **DETECT_BLOBS_BY_BOUNDING_BOXES**: valor booleano que indica si el método a utilizar en la detección de *blobs* es *BoundingBox* o *SimpleBlobDetector*. En caso de que *BoundingBox* esté activado, únicamente los valores de los parámetros **EXPAND_BLOBS** y **EXPAND_BLOBS_RATIO** afectan su funcionamiento y los enumerados anteriormente son ignorados.
- **EXPAND_BLOBS** y **EXPAND_BLOBS_RATIO**: si el valor booleano **EXPAND_BLOBS** es '*True*' entonces se ejecuta una transformación sobre cada *blob* para expandir sus límites, en un porcentaje de sus lados dado por el valor decimal de **EXPAND_BLOBS_RATIO**. A modo de ejemplo, si un *blob* tiene sus límites en 20 píxeles por 40 píxeles y **EXPAND_BLOBS_RATIO** un valor de 0.1, luego de la transformación los límites del *blob* pasarán a ser de 22 píxeles por 44 píxeles.

Detección de objetos de interés

- **ASPECT_RATIO**: número decimal que representa la relación de aspecto de los blobs de la imagen que contienen personas. Es el número buscado al dividir la altura del *blob* entre su ancho.
- **PADDING_(0, 1)**: vector de números enteros utilizados por el método HOG para expandir el rectángulo de búsqueda tanto en la dirección *x* (0) como en la *y* (1).

- **SCALE**: número decimal utilizado por el método HOG para definir las capas de la pirámide de imágenes escaladas. Cuanto más cercano a uno es el valor de **SCALE**, más capas tiene la pirámide y más tiempo se requiere para procesar el total de las capas.
- **WINSTRIDE_(0, 1)**: vector de números enteros que indican, para el método HOG, el avance de la ventana de búsqueda sobre el *blob*. El parámetro **WINSTRIDE_0** define el avance en la dirección *x* y el parámetro **WINSTRIDE_1** lo hace en la dirección *y*.
- **PERSON_DETECTION_PARALLEL_MODE**: valor booleano que activa/desactiva el procesamiento en paralelo del detector de personas con el método HOG.
- **BORDER_AROUND_BLOB_(0, 1)**: vector de números decimales que indican la expansión del *blob*, por dirección, en la etapa de detección de objetos de interés. Este parámetro es similar al parámetro **EXPAND_BLOBS_RATIO**.
- **USE_HISTOGRAMS_FOR_PERSON_DETECTION**: valor booleano que indica si se activa/desactiva el uso del método de frecuencia de relación de aspecto para descartar blobs.
- **FRAMES_COUNT_FOR_TRAINING_HISTOGRAMS**: número entero que indica la cantidad de frames, desde el inicio del sistema, que son tomados en cuenta para crear el modelo para el método de frecuencia de relación de aspecto.
- **CONFIDENCE_MATRIX_UPDATE_TIME**: número entero que representa la cantidad de milisegundos que deben transcurrir para que se actualice el modelo del método de frecuencia de relación de aspecto.
- **USE_CONFIDENCE_LEVELS**: valor booleano que indica si son utilizados los niveles de confianza en el método de frecuencia de relación de aspecto, o si el valor del histograma para las coordenadas del *blob* en cuestión es considerado información suficiente para descartarlo.
- **CONFIDENCE_LEVEL_(0, 1)**: números decimales que indican la cota superior del nivel 0 y del nivel 1 para el método de frecuencia de relación de aspecto.
- **USE_SQUARE_REGION_FOR_VERIFY**: valor booleano que indica si se deben tomar en cuenta las áreas adyacentes dentro de un radio alrededor del área correspondiente al *blob* en el histograma, con el fin de buscar el valor más alto en un vecindario. Este valor es tenido en cuenta para definir a qué nivel pertenece el *blob*.
- **SQUARE_REGION_RADIUS**: número entero que determina el radio del área a tomar en cuenta si **USE_SQUARE_REGION_FOR_VERIFY** tiene el valor ‘*True*’
- **CREATE_MODEL**: valor booleano que indica si el sistema debe generar un modelo de histograma para el método de frecuencia de relación de aspecto.
- **USE_MODEL**: valor booleano que indica si el sistema debe utilizar un modelo de histograma previamente creado para el método de frecuencia de relación de aspecto.

Seguimiento de objetos de interés

- **USE_HISTOGRAMS_FOR_TRACKING:** valor booleano que activa/desactiva el uso de histogramas para la comparación por color entre *blobs* detectados en el tiempo t y $t + 1$.
- **HISTOGRAM_COMPARISON_METHOD:** texto que define el método a ser utilizado en el uso de histogramas en la comparación por color. Los posibles valores provistos por OpenCV son CORRELATION, CHI_SQUARED, CHI_SQUARED_ALT, INTERSECTION, HELLINGER y KL_DIV. Los valores provistos por ScyPy son EUCLIDEAN, MANHATTAN y CHEBYSEV.
- **THRESHOLD_COLOR:** número decimal que define el umbral de aceptación, tal que dos colores puedan ser declarados como similares o no.
- **THRESHOLD_DISTANCE:** número entero que define el umbral de distancia, expresado en píxeles, entre el centro de un *blob* y otro, para ser tomados como *blobs* relacionados.
- **PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS:** vector de números decimales que indica la ponderación que se le da en el algoritmo Húngaro a la posición previa, la posición predicha y a la diferencia de color en la primera instancia de aplicación del mismo.
- **SECONDARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS:** similar al parámetro PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS, pero utilizado en la segunda instancia de aplicación del método.
- **MAX_SECONDS_WITHOUT_UPDATE:** número decimal que indica la cantidad máxima de segundos necesarios para que un *blob* que está siendo seguido, no reciba actualizaciones de posición y sea descartado.
- **MAX_SECONDS_TO_PREDICT_POSITION:** número decimal que indica la cantidad máxima de segundos transcurridos luego de la última posición recibida, en la que se mantiene prediciendo la posición del *blob*. Este tiempo debe ser menor al valor del parámetro MAX_SECONDS_WITHOUT_UPDATE.
- **MAX_SECONDS_WITHOUT_ANY_BLOB:** número decimal que indica la cantidad máxima de segundos que un *Tracklet* puede permanecer sin que se le adjudique al menos un *blob*.
- **MIN_SECONDS_TO_BE_ACCEPTED_IN_GROUP:** número decimal que indica, en segundos, el tiempo de vida mínimo necesario para que un *Tracklet* sea incluido en un grupo de *Tracks*.
- **KALMAN_FILTER_TYPE:** indica el tipo de implementación a utilizar al aplicar el método de filtros de Kalman en la predicción de posiciones. Los tipos permitidos son NORMAL, provisto por OpenCV, y SMOOTHED provisto por FilterPy. La implementación SMOOTHED permite el suavizado del trayecto predicho. La cantidad de pasos hacia atrás que son suavizados es definida por el valor del parámetro KALMAN_FILTER_SMOOTH_LAG. Este parámetro queda sin efecto cuando se utiliza el método NORMAL.

- **MEASURES_NOISE_IN_PIXELS**: número entero que representa los píxeles de desviación estándar en las medidas de los centros de los *blobs* utilizadas en el método de filtros de Kalman.
- **NON_TRUTHFUL_MEASURES_NOISE_IN_PIXELS**: en el caso de oclusiones, es el número entero que representa los píxeles de desviación estándar en la distancia entre los centros de los *blobs* ocluidos y el *blob* producto de la oclusión.
- **VARIANCE_OF_MODEL_CHANGE_BETWEEN_STEPS**: en los filtros de *Kalman*, es el número decimal que representa la varianza del error en la predicción entre un *frame* y el siguiente. Es utilizado en conjunto con la función *Q_discrete_white_noise* de la biblioteca *FilterPY* para generar la matriz **Q** de ruido del proceso.
- **INITIAL_ERROR_VARIANCE_OF_POSITION**: en los filtros de *Kalman*, es el número decimal que representa la varianza del error inicial en la posición (i.e. la varianza de la distancia entre la posición real y la medida inicial). Por ejemplo, si el máximo error entre la posición real y la medida inicial es de X_{px} , entonces la desviación estándar es $\sigma = \frac{X}{3} px$ (i.e. 99 % de los errores están en el rango de 0 a $3 \times \sigma$), por lo que la varianza es $\sigma^2 = \frac{x^2}{9}$.
- **INITIAL_ERROR_VARIANCE_OF_VELOCITY**: similar al parámetro anterior (**INITIAL_ERROR_VARIANCE_OF_POSITION**), pero representa la varianza del error inicial en la velocidad. Como la velocidad es iniciada en cero, entonces la varianza del error es la varianza de la velocidad máxima (i.e. si la velocidad máxima es de $X \frac{px}{s}$ entonces $\sigma = \frac{X}{3} \frac{px}{s}$ y $\sigma^2 = \frac{X^2}{9} \frac{px^2}{s^2}$).
- **INITIAL_ERROR_VARIANCE_OF_ACCELERATION**: similar al parámetro anterior (**INITIAL_ERROR_VARIANCE_OF_POSITION**), pero representa la varianza del error inicial en la aceleración. Como la aceleración es iniciada en cero, entonces la varianza del error es la varianza de la aceleración máxima (i.e. si la aceleración máxima es de $X \frac{px}{s^2}$ entonces $\sigma = \frac{X}{3} \frac{px}{s^2}$ y $\sigma^2 = \frac{X^2}{9} \frac{px^2}{s^4}$).

Depuración del sistema

En esta subsección se presentan los parámetros utilizados para activar o desactivar gráficos en la salida del sistema, con el fin de facilitar la depuración del éste:

- **SHOW_COMPARISONS_BY_COLOR**: valor booleano que indica si se debe mostrar una ventana con el resultado de las comparaciones de objetos de interés del tiempo t y $t + 1$, teniendo en cuenta el color.
- **SHOW_COMPARISONS_BY_COLOR_ONLY_NON_ZERO**: valor booleano que indica si se deben mostrar aquellos objetos de interés para los cuales, luego de la comparación por color, no se encontró un objeto de interés correspondiente.
- **SHOW_COMPARISONS_BY_COLOR_GLOBAL_BETTER_DECISION**: valor booleano que indica si se debe mostrar la lista ordenada por coincidencia de color de objetos de interés para el tiempo t , que son similares al tratado en el tiempo $t + 1$.
- **SHOW_COMPARISONS_BY_COLOR_GREEN**: valor booleano que indica si se debe marcar en color verde aquellas asociaciones de objetos de interés (del tiempo t con los del

tiempo $t + 1$) según el color, y que coinciden con aquellas asociaciones establecidas teniendo en cuenta la posición.

- **SHOW_COMPARISONS_BY_COLOR_GREY:** valor booleano que indica si se debe marcar en color gris aquellas asociaciones de objetos de interés (del tiempo t con los del tiempo $t + 1$) que según el color coinciden, pero que pudieron ser asociadas según la posición.
- **SHOW_COMPARISONS_BY_COLOR_RED:** valor booleano que indica si se debe marcar en color rojo aquellas asociaciones de objetos de interés (del tiempo t con los del tiempo $t + 1$) que según el color coinciden, pero que no coinciden según la posición.
- **SHOW_PREDICTION_DOTS:** valor booleano que indica si la salida de imágenes contiene puntos indicando la posición predicha para cada *blob*.
- **SHOW_VIDEO_OUTPUT:** valor booleano que indica si se presenta la salida del sistema en una ventana en forma de vídeo.
- **VERBOSE:** valor booleano que indica si, en cada frame, se presenta en la consola la información del número de frame y los fotogramas por segundo que se están procesando.

Comunicaciones del módulo

En esta subsección se presentan los parámetros de configuración referentes a las comunicaciones del módulo:

- **STATUS_INFO_EXCHANGE_HOSTADDRESS, STATUS_INFO_EXCHANGE_NAME** y **STATUS_INFO_EXPIRATION_TIME:** indican la configuración del *exchange* utilizado para el envío de mensajes de información de estado.
- **TRACK_INFO_EXCHANGE_HOSTADDRESS, TRACK_INFO_EXCHANGE_NAME** y **TRACK_INFO_EXPIRATION_TIME:** indican la configuración del *exchange* utilizado para el envío de mensajes a ser procesados por el módulo Buscador de patrones.

Funcionamiento general

En esta subsección se presentan los parámetros que inciden en el funcionamiento general del módulo:

- **INFINITE_DISTANCE:** número entero que representa una distancia infinita en píxeles.
- **LIMIT_FPS** y **DEFAULT_FPS_LIMIT:** valor booleano que indica si la entrada debe ser procesada a un ritmo fijo, definido por el valor de *frames* por segundo. El valor de FPS es extraído de la fuente de datos. En los casos en que la extracción del valor de FPS no esté habilitada, se utiliza el valor entero definido en **DEFAULT_FPS_LIMIT**.
- **SAVE_POSITIONS_TO_FILE:** valor booleano que indica si los valores de posición de los objetos de interés detectados deben ser guardados en un archivo. El fin de esta función es el de disponer de datos para poder realizar análisis de rendimiento del sistema, posteriores al procesamiento.

- IMAGE_MULTIPLIER_ON_POSITIONS_SAVE: número decimal que sirve de corrector para las posiciones guardadas cuando el parámetro SAVE_POSITIONS_TO_FILE toma el valor 'True'.

5.5. Implementación del módulo Buscador de patrones

El módulo Buscador de patrones tiene la capacidad de manejar de forma concurrente múltiples procesamientos de distintas fuentes de datos. El bloque principal del módulo es el encargado de discernir de qué fuente de datos proviene la información arribada. Esta información es en su mayoría referente a la posición, para cierto momento, de los objetos de interés detectados por instancias del módulo Reconocimiento y seguimiento. La información arriba con un identificador único, correspondiente al identificador de la instancia del módulo del que proviene. En el menor de los casos, la información corresponde a la solicitud de procesamiento de datos de una nueva instancia del módulo Reconocimiento y seguimiento. En estos casos el Buscador de patrones crea una nueva instancia de la clase *PatternRecognition* y le asigna el identificador del módulo solicitante. La clase *PatternRecognition* implementa la lógica necesaria para detectar patrones a partir de un histórico de posiciones de objetos de interés. Las instancias de esta clase son mantenidas en memoria del Buscador de patrones, e invocadas cada vez que arriba un mensaje con información de objetos de interés. Las solicitudes de nuevas instancias pueden contener una configuración personalizada a ser aplicada en la nueva instancia de *PatternRecognition*. En el caso de que una solicitud no tenga una configuración personalizada, se utiliza la configuración por defecto definida en el archivo *patternmaster.conf*. El diagrama de la Figura 5.12 ilustra la reacción del módulo Buscador de patrones frente al arribo de mensajes.

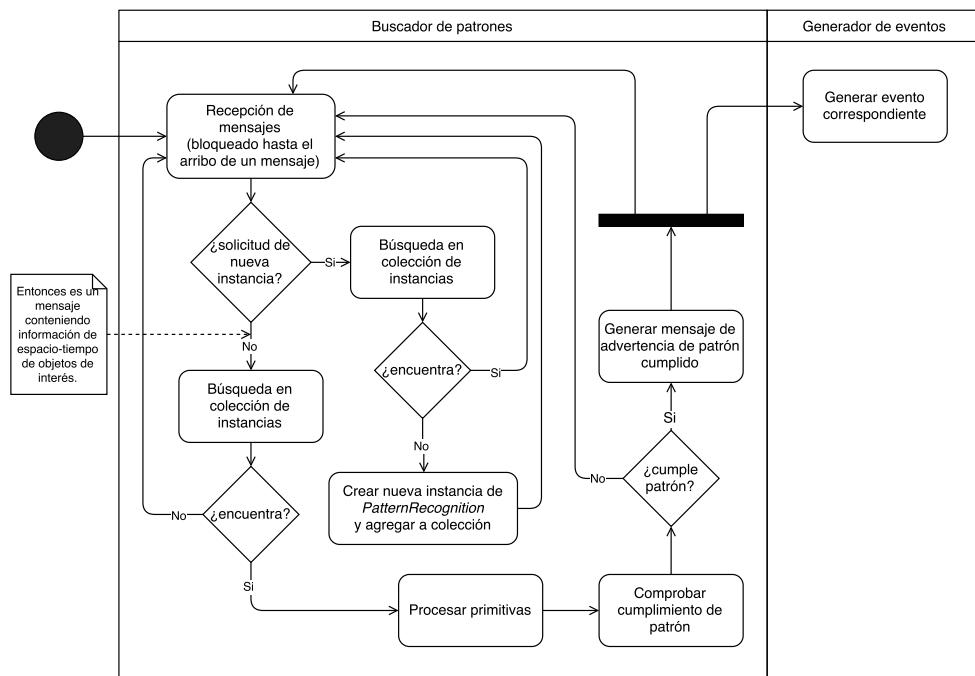


Figura 5.12: Diagrama de actividad del módulo Buscador de patrones.

La clase *PatternRecognition* contiene los estados y la lógica necesaria para la búsqueda de patrones. Tiene a su disposición la definición de los patrones cargados al momento de crear la instancia, el historial reciente de primitivas cumplidas por un objeto de interés identificado y toda la lógica necesaria para que esta información sea relacionada, con el fin de comprobar si se cumple alguno de los patrones cargados en el sistema.

5.5.1. Definición de patrones

La definición de los patrones se encuentra en un archivo con valores separados por coma, del estilo CSV [73], aunque sus líneas no tienen por qué tener la misma cantidad de valores. El archivo es de texto plano y su ubicación es detallada en la configuración del módulo como valor del parámetro **PATTERNS_DEFINITION_FILE_PATH**. Cada línea del archivo representa una secuencia de primitivas. Cada definición de primitiva (o sea, cada línea del archivo de texto) tiene la siguiente sintaxis:

```
<nombre del patrón>,[<tipo de primitiva>,<tipo de evento>,<cuantificador>,<valor>]+
```

donde ‘nombre del patrón’ es el identificador del patrón, ‘tipo de primitiva’ depende de las definidas programáticamente en el sistema (**SPEED**, **DIRECTION**, **AGGLOMERATION**), ‘tipo de evento’ depende de los definidos programáticamente en el sistema para el tipo de primitiva (en el caso del tipo de primitiva **SPEED** los posibles tipos de evento son **WALKING**, **RUNNING** y **STOPPED**), ‘cuantificador’ define de qué forma se comparan los valores de las primitivas registradas frente a las exigidas por el patrón. Los posibles valores del cuantificador son **LE** (menor o igual), **GE** (mayor o igual), **AX** (aproximado), **EQ** (igual), **NM** (valor sin relevancia). A modo de ejemplo, la primitiva que se presenta en la Figura 4.5 del Capítulo 4 tiene la siguiente representación, de acuerdo a lo definido anteriormente:

```
patron_ejemplo,SPEED,WALKING,GE,5000,SPEED,STOPPED,AX,5000,SPEED,RUNNING,GE,5000
```

5.5.2. Parámetros de configuración del módulo Buscador de patrones

Al igual que el módulo Reconocimiento y seguimiento, el módulo Buscador de patrones tiene sus propios parámetros de configuración, aunque con la diferencia de que estos son aplicados sobre la instancia de la clase *PatternRecognition* y no sobre la instancia del módulo en sí. De esta forma, es posible que cada fuente de datos sea tratada con sus propios valores de configuración guardados en la instancia de Buscador de patrones. Por ejemplo, si cada fuente de datos define rutas diferentes como valor del parámetro **PATTERNS_DEFINITION_FILE_PATH**, entonces es posible que cada fuente de datos trabaje en la búsqueda de un conjunto de primitivas diferente. Los valores por defecto se encuentran en el archivo *patternmaster.conf* bajo el directorio del módulo.

A continuación se listan y explican los parámetros disponibles en el módulo Buscador de patrones:

- **MIN_WALKING_SPEED**: número entero que especifica, en píxeles por segundo, la cota inferior de velocidad de movimiento de un objeto de interés para que se considere que éste realiza la acción de caminar.

- **MIN_RUNNING_SPEED:** número entero que especifica, en píxeles por segundo, la cota inferior de la velocidad de movimiento de un objeto de interés para que se considere que éste realiza la acción de correr.
- **MIN_ANGLE_ROTATION:** número entero que especifica, en grados, la cota inferior de rotación de un objeto de interés para que se considere que éste realiza una acción de rotación.
- **WEIGHT_NEW_DIRECTION_ANGLE:** número decimal que define el nivel de ponderación, en el rango de 0 a 1, de la nueva dirección registrada para el trayecto de un objeto de interés. Tiene el propósito de disminuir el ruido en la medición de direcciones.
- **APROX_TOLERANCE:** número entero que define la tolerancia del cuantificador AX.
- **WARNINGS_COMM_HOSTADDRESS:** dirección del servidor *RabbitMQ* utilizado para comunicar datos desde el módulo Buscador de Patrones.
- **WARNINGS_EXCHANGE_NAME:** nombre del *exchange* definido por el módulo Buscador de Patrones para comunicar datos.
- **WARNINGS_EXPIRATION_TIME:** número entero que define el tiempo, en segundos, que debe transcurrir para que un mensaje de advertencia generado por el módulo Buscador de Patrones sea expirado, sin haber sido atendido.
- **TRACK_INFO_QUEUE_HOSTADDRESS:** dirección del servidor *RabbitMQ* que contiene el *exchange* del tipo *direct* a través del cual el módulo Buscador de patrones recibe la información de las diferentes instancias del módulo Reconocimiento y seguimiento.
- **TRACK_INFO_EXCHANGE_NAME:** nombre del *exchange* del tipo *direct* por la cual el módulo Buscador de patrones recibe la información de las diferentes instancias del módulo Reconocimiento y seguimiento.
- **MIN_EVENTS_SPEED_AMOUNT:** número entero que establece un mínimo de primitivas del tipo SPEED, registradas para un objeto de interés, a ser tenidas en cuenta al momento de comprobar el cumplimiento de patrones. Este parámetro toma el valor 6 como su valor por defecto.
- **MIN_EVENTS_SPEED_TIME:** número entero que define el tiempo hacia atrás, en milisegundos, en el que son consideradas las primitivas del tipo SPEED, registradas para un objeto de interés, al momento de comprobar el cumplimiento de patrones. Este parámetro toma el valor 30000 como su valor por defecto.
- **MIN_EVENTS_DIR_AMOUNT:** número entero que establece el mínimo de primitivas del tipo DIRECTION, registradas para un objeto de interés, a ser tenidas en cuenta al momento de comprobar el cumplimiento de patrones. Este parámetro toma el valor 2 como su valor por defecto.
- **MIN_EVENTS_DIR_TIME:** número entero que define el tiempo hacia atrás, en milisegundos, en el que son consideradas las primitivas del tipo DIRECTION, registradas para un objeto de interés, al momento de comprobar el cumplimiento de patrones. Este parámetro toma el valor 30000 como su valor por defecto.

- **AGGLOMERATION_MIN_DISTANCE**: número entero que define la mínima distancia, medida en píxeles, entre objetos de interés para considerarlos parte de una aglomeración. Este parámetro representa el radio de la circunferencia que contiene los objetos de interés en cuestión.
- **PATTERNS_DEFINITION_FILE_PATH**: ruta del archivo que define las primitivas a ser buscadas.
- **GLOBAL_EVENTS_LIVES_TIME**: tiempo, en milisegundos, para que un evento global expire.
- **TRACKLETS_LIVES_TIME**: tiempo de espera, en milisegundos, desde el último arribo de información de un objeto de interés para que éste expire.

Capítulo 6

Análisis experimental

En este capítulo se presenta el análisis experimental que se realiza sobre ambos módulos del sistema (i.e. el módulo Reconocimiento y seguimiento, y el módulo Buscador de patrones). Para cada módulo se especifican: el vídeo sobre el cual se trabaja, el plan de ejecución, los resultados del plan de ejecución, y las métricas que se utilizan para comparar los resultados.

Un plan de ejecución está comprendido por uno o más planes de ejecución, y cada plan se compone de una configuración base y de uno o más bloques, los cuales contienen una lista de experimentos a ejecutar. A su vez, esta lista de experimentos surge de la combinación de valores de ciertos parámetros de configuración que se quieren estudiar. Por ejemplo, si en un bloque se incluyen los parámetros de configuración A y B , y se decide que para A se van a analizar los valores A_1 , A_2 y A_3 , mientras que para B se van a analizar los valores B_1 y B_2 , entonces el bloque incluirá los experimentos para las combinaciones: $\{(A_1, B_1), (A_1, B_2), (A_2, B_1), (A_2, B_2), (A_3, B_1), (A_3, B_2)\}$. Una vez definidos los experimentos para cada bloque, los bloques son ejecutados en orden. El primer bloque recibe la configuración base del plan, a la cual combina con la configuración de sus distintos experimentos. Una vez finalizado un bloque, se eligen las tres mejores combinaciones del mismo y se las combinan con las combinaciones del siguiente bloque (e.g. si el siguiente bloque tuviera 10 combinaciones, pasa a tener 30). Al finalizar el último bloque, se seleccionan las dos mejores combinaciones de dicho bloque y son combinadas con el primer bloque del siguiente plan de ejecución. Además, ese plan recibe como configuración base la misma que la del plan anterior, combinada con los valores de los parámetros que sean idénticos en las dos mejores combinaciones elegidas (e.g. si las mejores combinaciones son (A_1, B_1) y (A_3, B_1) , entonces el parámetro B de la configuración base se establece en B_1).

Para ambos módulos, el objetivo del análisis experimental es hallar el valor de los parámetros con los que tienen un mejor resultado y el por qué de ese buen resultado, además de observar qué tan buenos son comparado con otros algoritmos del estado del arte.

El análisis experimental de los módulos es ejecutado con el valor del parámetro de configuración `LIMIT_FPS` en *False*, con el fin de procesar el total de los *frames* del vídeo de prueba (siete por segundo) sin que los servicios del SO afecten el rendimiento computacional, y lograr así evaluar puramente el desempeño del módulo en sí.

6.1. Análisis del módulo Reconocimiento y seguimiento

En el análisis de este módulo se trabajó sobre el vídeo PETS09-S2L1 (i.e. el mismo vídeo que se utilizó durante el desarrollo), ya que contiene eventos desafiantes, como personas que se mueven por todo el escenario y producen varias occlusiones.

Como entorno de ejecución de los experimentos, se utilizó una MacBook Pro de fines del 2013 con procesador Intel Core i7, cuatro núcleos a 2 GHz, 8 GB de RAM, y OS X El Capitan; con los recursos dedicados a la ejecución del algoritmo. Además, la versión de Python utilizada fue la 3.5.0.

6.1.1. Métricas

Los aspectos importantes a calificar en la ejecución de este módulo, son que el procesamiento sea en tiempo real y que el seguimiento realizado por los tracklets refleje la realidad de la forma más exacta y precisa posible, de forma que los datos que llegan al módulo buscador de patrones sean válidos (i.e. dentro de una ventana de tiempo razonable) y de buena calidad.

Por un lado, para calificar la velocidad del algoritmo, se recaban los tiempos de procesamiento promedio y máximo por *frame*, los cuales se dividen en los tiempos de procesamiento promedio y máximo de cada filtro. En el sistema implementado, la obtención de estos tiempos se logra mediante temporizadores para el procesamiento de cada filtro, y lo esperable es que el tiempo promedio cada 20 *frames* sea menor a un segundo, de forma que un vídeo de 20 *FPS* se procese en tiempo real. El número de 20 *FPS* se justifica en base al estudio de Honovich [74], quien concluye que, a noviembre del 2011, el 94 % de las cámaras de vídeo vigilancia grababan a una tasa de 20 o menos imágenes por segundo.

Por otro lado, para calificar la exactitud y precisión del seguimiento, se recaban métricas como la precisión del conteo de objetos, la cantidad de falsos positivos y negativos, la cantidad de cambios de identidad (*identity switch*), y la cantidad de fragmentaciones. Teniendo un *ground truth* del vídeo, la precisión del conteo de objetos es fácil de hallar mediante un *script*. Se compara en cada frame la cantidad de personas que hay en el *ground truth* contra la cantidad de tracklets o blobs que hay en el sistema, y de las diferencias se obtienen un promedio, un mínimo, un máximo, y un histograma. Luego, cuanto menor es el promedio y el máximo, y cuanto más contra el cero está el histograma, mejor es la precisión del conteo de objetos. En cambio, de forma contraria a la precisión del conteo de objetos, el resto de las métricas requieren realizar un *script* complicado, o bien hacer la comparación de forma manual (i.e. viendo los vídeos del *ground truth* al mismo tiempo que los de los experimentos).

En el caso de una comparación manual, habría que limitar mucho la cantidad de parámetros de configuración a estudiar para que el análisis lleve un tiempo razonable, mientras que la posibilidad de hacer un *script* queda fuera del alcance del proyecto, debido a su complejidad. Para poder estudiar todos los parámetros deseados, sin que lleve demasiado tiempo ni tener que hacer un *script* complicado, se realizó una búsqueda de "automatic comparison scripts for multiple objects tracking algorithms", la cual arrojó como resultado el artículo *MOT Challenge* [75] de Leal-Taixé et al..

MOT Challenge es un *framework* cuyo principal objetivo es el crear un estándar para la comparación de algoritmos de seguimiento de múltiples objetivos (*Multiple Objective Tracking*). Para ello, desde su página web [76] ofrece la descarga del *ground truth* de

varias secuencias (incluyendo al vídeo sobre el que se trabajó durante el desarrollo) y una herramienta de evaluación (*DevKit*) para sistemas de seguimiento de múltiples objetivos implementada en MatLab, así como también ofrece: comparar resultados con otros algoritmos del estado del arte, y descargar distintas secuencias de dos y tres dimensiones, filmadas con cámaras fijas o en movimiento. En cuanto a la herramienta de evaluación, esta provee las siguientes métricas:

Recall Porcentaje de objetos detectados. Cuanto mayor es *Recall*, mejor es el resultado. Número real, de 0 a 100.

Precision Porcentaje de objetos correctamente detectados. Cuanto mayor es *Precision*, mejor es el resultado. Número real, de 0 a 100.

FAF Promedio de Falsas Alarmas por Frame, o *False Alarms Rate* (FAR). Cuanto menor es FAF, mejor es el resultado. Número real, mayor o igual que 0.

MT *Mostly tracked targets*. La cantidad de trayectorias del *ground truth* que son cubiertas por una trayectoria de la solución durante al menos un 80 % de su vida. Cuanto mayor es MT, mejor es el resultado. Número entero, entre 0 y la cantidad de objetos en la secuencia.

PT *Partially tracked targets*. La cantidad de trayectorias del *ground truth* que son cubiertas por una trayectoria de la solución durante un 20 % a un 80 % de su vida. Número entero, entre 0 y la cantidad de objetos en la secuencia.

ML *Mostly lost targets*. La cantidad de trayectorias del *ground truth* que son cubiertas por una trayectoria de la solución durante a lo sumo un 20 % de su vida. Cuanto menor es ML, mejor es el resultado. Número entero, entre 0 y la cantidad de objetos en la secuencia.

FP Número total de Falsos Positivos (i.e. detecciones en un frame que no corresponden a un objetivo). Cuanto menor es FP, mejor es el resultado. Número entero, mayor o igual que 0.

FN Número total de Falsos Negativos (i.e. objetivos que no fueron detectados en un frame). Cuanto menor es FN, mejor es el resultado. Número entero, mayor o igual que 0.

IDs Número total de cambios de identidad (*IDentity Switches*). Cuanto menor es IDs, mejor es el resultado. Número entero, mayor o igual que 0.

FRA Número total de FRAGmentaciones (i.e. seguimientos que fueron interrumpidos y reanudados). Cuanto menor es FN, mejor es el resultado. Número entero, mayor o igual que 0.

MOTA *Multiple Object Tracking Accuracy*. Porcentaje que combina tres fuentes de error: falsos positivos, falsos negativos y cambios de identidad. Cuanto mayor es MOTA, mejor es el resultado. Número real, de 0 a 100.

MOTP *Multiple Object Tracking Precision*. Porcentaje que representa la diferencia entre las posiciones del *ground truth* y las posiciones predichas correctamente. Como las posiciones predichas correctamente deben sobreponerse al menos un 50 % sobre

la posición del *ground truth*, este número va a partir del 50 %. Cuanto mayor es MOTP, mejor es el resultado. Número real, de 50 a 100.

MOTAL MOTA con $\log_{10}(IDs)$. Cuanto mayor es MOTAL, mejor es el resultado. Número real, de 0 a 100.

En síntesis, además de las métricas del *MOT Challenge*, las métricas utilizadas, junto a sus valores óptimos y rangos válidos son las siguientes:

Tiempos Promedio y Máximo de procesamiento de cada filtro por frame.

Tiempos para: Sustracción de fondo, Detección de *blobs*, Clasificación de *blobs*, y Seguimiento. Cuanto menor son, mejor es el resultado. Ambos son números reales, mayores que 0, expresados en segundos.

Tiempos Promedio y Máximo de procesamiento total por frame. Cuanto menor son, mejor es el resultado. Ambos son números reales, mayores que 0, expresados en segundos. El tiempo promedio tiene un valor óptimo si es menor a 0,05 segundos, ya que de esa forma se podrían procesar 20 *frames* en un segundo.

Diferencia en el conteo de personas, según los blobs. Diferencia promedio, mínima y máxima entre la cantidad de *blobs* y la cantidad de personas del *ground truth* en cada frame. Números mayores o iguales que 0. Cuanto menores son, mejor es el resultado.

Histograma de diferencia en el conteo de personas, según los blobs.

Histograma de la diferencia entre la cantidad de *blobs* y la cantidad de personas del *ground truth* en cada frame. Cuanto más acumulados contra el cero están los valores, mejor es el resultado.

Diferencia en el conteo de personas, según los seguimientos. Diferencia promedio, mínima y máxima entre la cantidad de seguimientos (i.e. *tracklets* u objetos) y la cantidad de personas del *ground truth* en cada frame. Números mayores o iguales que 0. Cuanto menores son, mejor es el resultado.

Histograma de diferencia en el conteo de personas, según los seguimientos.

Histograma de la diferencia entre la cantidad de seguimientos (i.e. *tracklets* u objetos) y la cantidad de personas del *ground truth* en cada frame. Cuanto más acumulados contra el cero están los valores, mejor es el resultado.

Diferencia en el conteo de personas, según valor interpolado. Diferencia promedio, mínima y máxima entre la cantidad interpolada de *blobs* y seguimientos, y la cantidad de personas del *ground truth* en cada frame. Números mayores o iguales que 0. Cuanto menores son, mejor es el resultado.

Histograma de diferencia en el conteo de personas, según valor interpolado.

Histograma de la diferencia entre la cantidad interpolada de *blobs* y seguimientos, y la cantidad de personas del *ground truth* en cada frame. Cuanto más acumulados contra el cero están los valores, mejor es el resultado.

6.1.2. Ground truth

En cuanto al *ground truth* del vídeo, previo al hallazgo del *MOT Challenge*, este se realizó de forma manual, y contenía la posición de cada persona (i.e. la coordenada de un punto en el centro de la persona) a cada segundo del vídeo (i.e. cada 7 *frames*). Luego del hallazgo, se decidió utilizar el *ground truth* provisto por *MOT Challenge* ya que, a pesar de no estar hecho de forma manual, sí se encuentra completo (i.e. posee todos los frames), y su precisión y exactitud es buena.

El utilizar el *ground truth* de *MOT Challenge* implicó la realización de algunos cambios en el código ya que, por un lado, las posiciones que posee están en formato de *bounding boxes* (i.e. la coordenada de la esquina superior izquierda, el ancho y el alto del rectángulo que contiene al objeto) mientras que en el sistema implementado los tracklets contenían solamente el punto central; y por otro lado, las dimensiones del vídeo utilizado para el *ground truth* son un 240 % mayores que las del vídeo utilizado durante el desarrollo del sistema. Para solucionar el problema de las dimensiones, se agregó el parámetro de configuración `IMAGE_MULTIPLIER_ON_POSITIONS_SAVE` que permite variar la escala en las posiciones cuando estas son guardadas en el archivo que se utiliza para comparar con el *ground truth*. Por otra parte, en cuanto al diferente formato de las posiciones, si se utiliza como *bounding box* el rectángulo dado por el blob en cada frame, en las oclusiones este rectángulo se reduce y luego desaparece. Por lo tanto, comparando con el *ground truth* da malos resultados. Para mejorar eso, se mantiene un registro de los *bounding boxes* únicamente mientras los blobs están uno a uno con un *tracklet* (i.e. no hay oclusión). Entonces, cuando ocurre una oclusión, el *bounding box* guardado se traslada hacia la posición que se predice para el objeto.

6.1.3. Plan de ejecución

El módulo Reconocimiento y seguimiento posee un gran número de parámetros de configuración, de los cuales en los experimentos se varían solo aquellos que trascienden el escenario, mientras se dejan constantes los relacionados al escenario a procesar (e.g., distancias y tamaños de objetos) y los relacionados a la depuración y formateo de datos resultantes. Entonces, en total se deben variar más de 40 parámetros.

Si el plan de ejecución del módulo consistiera de un solo bloque que abarcara todas las combinaciones, habría que ejecutar más de 1×10^{12} experimentos (en un supuesto de dos valores por cada parámetro), lo que incuraría en cientos de miles de años de procesamiento (a un paso de 30 segundos por experimento). Debido a que dicho plan de ejecución es impracticable, se buscaron formas de dividirlo.

Un primer acercamiento para dividir el plan de ejecución del módulo se fundamenta en que el módulo presenta una arquitectura de tubo basada en filtros, donde se asume que la exactitud y precisión de cada filtro depende únicamente de la configuración de sus variables. Por lo tanto, si se hallara la mejor configuración para las variables de cada filtro, se obtendría el mejor resultado global del módulo. Entonces, el plan de ejecución del módulo se puede dividir en un plan de ejecución por cada filtro (con un bloque cada uno), donde la unión de las mejores configuraciones de cada plan da como resultado la mejor configuración global. De esta forma, hay que hacer combinación de como mínimo 7 variables en el filtro *Detección de blobs* y como máximo 14 variables en el *filtro de blobs*. Si se toman los posibles valores de las variables (de dos a cuatro valores por cada parámetro), el *filtro de blobs* resultaría en cerca de 20000 experimentos, lo que significaría

un procesamiento de 7 días (a un paso de 30 segundos por experimento). A pesar de que se reduce muchísimo el tiempo, con esa cantidad de experimentos no resulta sencillo estudiar qué valores son mejores o peores en cada parámetro de configuración.

Finalmente, para reducir aún más la cantidad de experimentos, se logró identificar dentro de cada filtro, ciertos bloques de parámetros de configuración que actúan en partes independientes del filtro (i.e. partes secuenciales, siendo la entrada de cada una la salida de la anterior) y que, por lo tanto, se pueden analizar por separado. Entonces, la mejor combinación de parámetros para el filtro se compone de las mejores combinaciones de valores para los parámetros de cada bloque. De esta forma, la mayor cantidad de experimentos en el análisis se reduce de 20000 a 432, y el total de experimentos (i.e. sumando todos los bloques dentro de cada uno de los filtros) pasa a ser de 1458, siendo estos números mucho más manejables, tanto en tiempo como en facilidad de estudio.

Como configuración base para el plan de ejecución del primer filtro se utilizó la configuración que surgió durante el desarrollo del sistema, ya que empíricamente resultaba buena. La configuración base se puede ver en la sección I.1 del anexo I.

Durante la ejecución, para cada experimento primero se ejecuta el sistema de forma de crear un modelo (i.e. el valor del parámetro de configuración `CREATE_MODEL` es *True*) y luego se ejecuta nuevamente utilizando el modelo creado (i.e. el valor del parámetro de configuración `USE_MODEL` es *True*). De esta forma se puede obtener un mejor resultado. Además, se omiten las salidas de imagen (i.e. se asigna *False* al valor del parámetro `SHOW_VIDEO_OUTPUT`) para mejorar los tiempos de ejecución y adaptarlo más a la realidad, en la cual no es necesario estar reproduciendo la salida del sistema mientras se procesa.

En cuanto a los bloques y parámetros de configuración para cada filtro, estos se presentan en las siguientes cuatro secciones y, excepto donde se indica lo contrario, para cada parámetro los valores escogidos fueron los que se encuentran en la configuración base más dos o tres valores hacia cada lado (e.g. si el valor de la configuración es 100, entonces se escogen además el 50 y el 150).

Parámetros para el filtro Sustracción de fondo

En la Tabla 6.1 se presentan los parámetros y sus valores correspondientes para el análisis del filtro Sustracción de fondo, mientras que en la sección I.2 del anexo I se encuentra la lista de experimentos que surge de esos parámetros.

En el primer bloque se estudian los parámetros que definen la forma en que se remueven los píxeles pertenecientes al fondo (*background*) de un frame. Cuando el parámetro `USE_BSUBLTRACTOR_KNN` es *False* (i.e. se utiliza el método MOG2), los parámetros `DIST_2_THRESHOLD`, `KNN_SAMPLES` y `SHADOW_THRESHOLD` no son utilizados, mientras que el parámetro `MOG2_LEARNING_RATE` sí es utilizado. Por otro lado, cuando el parámetro `DETECT_SHADOWS` es *True*, el parámetro `FILTER_BY_COLOR` también es *True* y, en el caso en que `USE_BSUBLTRACTOR_KNN` es *True*, `SHADOW_THRESHOLD` es utilizado.

En el segundo bloque se estudian los parámetros que definen cuánto se difuminan, erosionan y dilatan los píxeles pertenecientes al frente (*foreground*) de un frame.

Bloque	Parámetro	Valores
1	HISTORY	35, 105, 175
	DIST_2_THRESHOLD	35, 140, 350
	KNN_SAMPLES	3, 5, 7
	DETECT_SHADOWS	False, True
	SHADOW_THRESHOLD	0.7, 0.8, 0.9
	FILTER_BY_COLOR	False, True
2	USE_BSUBTRACTOR_KNN	False, True
	MOG2_LEARNING_RATE	0.005, 0.0175, 0.05
	(GAUSSIANBLUR_SIZE_X; GAUSSIANBLUR_SIZE_Y)	(7;7), (9;9), (11;11)
	(ERODE_SIZE_X; ERODE_SIZE_Y)	(2;2), (3;3), (4;4)
	ERODE_TIMES	1, 3
	(DILATE_SIZE_X; DILATE_SIZE_Y)	(3;2), (4;3), (4;1)
	DILATE_TIMES	1, 3

Tabla 6.1: Plan de ejecución para el filtro Sustracción de fondo.

Parámetros para el filtro Detección de blobs

En la Tabla 6.2 se presentan los parámetros y sus valores correspondientes para el análisis del único bloque del filtro Detección de blobs, mientras que en la sección I.3 del anexo I se encuentra la lista de experimentos que surge de esos parámetros. En este bloque, se estudian los parámetros que, a partir de los píxeles del frente (*foreground*) de un frame, definen los conjuntos que forman blobs. Los parámetros MIN_AREA y MAX_AREA sólo son válidos cuando el parámetro FILTER_BY_AREA es *True*. Por otro lado, el parámetro EXPAND_BLOBS_RATIO es válido únicamente cuando el parámetro EXPAND_BLOBS es *True*. Finalmente, cuando el parámetro DETECT_BLOBS_BY_BOUNDING_BOXES es *True*, los parámetros MIN_DIST_BETWEEN_BLOBS y FILTER_BY_AREA no son utilizados.

Bloque	Parámetro	Valores
1	MIN_DIST_BETWEEN_BLOBS	3, 15, 30
	FILTER_BY_AREA	False, True
	MIN_AREA	75, 100, 125
	MAX_AREA	1500, 2000, 2500
	DETECT_BLOBS_BY_BOUNDING_BOXES	False, True
	EXPAND_BLOBS	False, True
	EXPAND_BLOBS_RATIO	0.1, 0.2, 0.3

Tabla 6.2: Plan de ejecución para el filtro Detección de blobs.

Parámetros para el filtro de blobs

En la Tabla 6.3 se presentan los parámetros y sus valores correspondientes para el análisis del filtro de blobs, mientras que en la sección I.4 del anexo I se encuentra la lista de experimentos que surge de esos parámetros.

En el primer bloque se estudian los parámetros que definen la utilización de histogramas sobre el tamaño de los blobs con el fin de filtrar blobs que no cumplen ciertas medidas. Los parámetros `CONFIDENCE_LEVEL_0` y `CONFIDENCE_LEVEL_1` sólo son válidos cuando el parámetro `USE_CONFIDENCE_LEVELS` es `True`. Por otro lado, los parámetros `CONFIDENCE_MATRIX_UPDATE_TIME` y `USE_CONFIDENCE_LEVELS` son válidos únicamente cuando el parámetro `USE_HISTOGRAMS_FOR_PERSON_DETECTION` es `True`.

En el segundo bloque se estudian los parámetros que definen la utilización del detector de personas HOG.

En el tercer bloque se estudian los parámetros que definen la utilización de un filtro para los blobs según la relación entre sus medidas de alto y su ancho. El parámetro `SQUARE_REGION_RADIUS` es válido únicamente cuando el parámetro `USE_SQUARE_REGION_FOR_VERIFY` es `True`.

Bloque	Parámetro	Valores
1	USE_HISTOGRAMS_FOR_PERSON_DETECTION	False, True
	CONFIDENCE_MATRIX_UPDATE_TIME	2500, 5000, 7500
	USE_CONFIDENCE_LEVELS	False, True
	CONFIDENCE_LEVEL_0	0.5, 0.7, 0.9
2	CONFIDENCE_LEVEL_1	0.1, 0.2, 0.3
	ASPECT_RATIO	2, 2.5, 3
	SCALE	1.01, 1.1, 1.5
	(WINSTRIDE_0;WINSTRIDE_1)	(2;2), (4;4), (8;8)
3	USE_SQUARE_REGION_FOR_VERIFY	False, True
	SQUARE_REGION_RADIUS	1, 2, 4
4	(BORDER_AROUND_BLOB_0;BORDER_AROUND_BLOB_1)	(0.25;0.25), (0.1;0.1), (0.5;0.5)
5	PERSON_DETECTION_PARALLEL_MODE	False, True

Tabla 6.3: Plan de ejecución para el filtro de blobs.

En el cuarto bloque se estudian los parámetros que definen el borde que se le agrega a los blobs antes de pasarlo al filtro de seguimiento.

Por último, en el quinto bloque se estudia el parámetro PERSON_DETECTION_PARALLEL_MODE, que define si se utiliza o no paralelismo para procesar los distintos blobs.

Parámetros para el filtro Seguimiento

En la Tabla 6.4 se presentan los parámetros y sus valores correspondientes para el análisis del filtro Seguimiento, mientras que en la sección I.5 del anexo I se encuentra la lista de experimentos que surge de esos parámetros.

Bloque	Parámetro	Valores
	USE_HISTOGRAMS_FOR_TRACKING	False, True
1	(HISTOGRAM_COMPARE_METHOD;THRESHOLD_COLOR;LOR)	(CORRELATION;1.9), (CHI_SQUARED;3.2), (CHI_SQUARED_ALT;3.8), (INTERSECTION;1.0), (HELLINGER;0.5), (KL_DIV;12.6), (EUCLIDEAN;0.8), (MANHATTAN;3.1), (CHEBYSEV;0.4)
2	PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS	(0,0,1)
2	THRESHOLD_COLOR	37.6
2	PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS	(1,0,0), (0,1,0), (0,0,1), (0.5,0.25,0.25), (0.25,0.5,0.25), (0.25,0.25,0.5), (0.33,0.34,0.33)
2	SECONDARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS	(1,0,0), (0,1,0), (0,0,1), (0.5,0.25,0.25), (0.25,0.5,0.25), (0.25,0.25,0.5), (0.33,0.34,0.33)
3	MAX_SECONDS_WAITOUT_UPDATE	1, 2, 4, 8
3	MAX_SECONDS_TO_PREDICT_POSITION	0.5, 1, 2, 4
3	MAX_SECONDS_WAITOUT_ANY_BLOB	0, 0.5, 1.5, 3.5
3	MIN_SECONDS_TO_BE_ACCEPTED_IN_GROUP	0, 0.5, 1.5, 3.5

Tabla 6.4: Plan de ejecución para el filtro Seguimiento.

En el primer bloque del filtro de seguimiento, se busca identificar cuáles son los mejores métodos para comparar los histogramas de color del seguimiento. Para ello, se indica mediante el parámetro PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS que el

filtro de seguimiento utilice solamente el color para asociar un tracklet con un nuevo blob. Respecto al parámetro `THRESHOLD_COLOR` que aparece al final, este se utiliza para cuando no se usan los histogramas (i.e. el parámetro `USE_HISTOGRAMS_FOR_TRACKING` es *False*). Por el contrario, cuando se utilizan histogramas, el `THRESHOLD_COLOR` que se utiliza es el que está en unión con `HISTOGRAM_COMPARISON_METHOD`, y que es elegido para cada método. Específicamente, para cada método se hallan los valores promedio para cuando un color es suficientemente diferente de otro, y para cuando es lo suficientemente parecido. Esto se hace con los datos proporcionados por el sistema al utilizar el parámetro de configuración `SHOW_COMPARISONS_BY_COLOR` en *True*.

En el segundo bloque se estudian los parámetros que definen qué pesos se le da a las distintas comparaciones entre blobs y tracklets (i.e. por color, por posición actual o por posición predicha) en las ejecuciones primarias y secundarias del algoritmo húngaro.

Por último, en el tercer bloque se estudian los parámetros que definen los máximos y mínimos tiempos para la existencia de los tracklets. La única restricción que tienen los parámetros de este bloque es que los parámetros `MAX_SECONDS_TO_PREDICT_POSITION` y `MAX_SECONDS_WITHOUT_ANY_BLOB` deben ser menores que `MAX_SECONDS_WITHOUT_UPDATE`.

6.1.4. Ejecución de los planes de ejecución

Para llevar a cabo un plan de ejecución, primero hay que crear los archivos de configuración para cada experimento del primer bloque. Luego, hay que ejecutar el sistema dos veces con cada configuración creada, tal que en la primera ejecución se crea el modelo de histogramas para el método de frecuencia de relación de aspecto, y en la segunda ejecución se utiliza el modelo y se obtienen las distintas métricas. Una vez finalizada la ejecución de los experimentos del bloque, hay que comparar los resultados de los experimentos y elegir los mejores tres. Después, hay que mezclar esos mejores experimentos con los experimentos del siguiente bloque, y repetir el primer paso. Por último, cuando se terminan los bloques de un plan de ejecución, se deben tomar los dos mejores resultados del último bloque y mezclarlos con la configuración base del siguiente plan de ejecución, o con el primer bloque de dicho plan. Al finalizar la ejecución de todos los planes de ejecución, los mejores dos resultados del último plan son los mejores resultados del sistema.

Creación de archivos de configuración

Debido a que hay más de 1400 experimentos en total, la tarea de crear los archivos de configuración correspondientes llevaría mucho tiempo, además de ser propensa a errores. Para disminuir dicho tiempo, se implementó un script en Python que, dada la definición de un plan de ejecución y un archivo de configuración base, crea todos los archivos de configuración correspondientes a los experimentos del plan y genera un archivo *csv* con una lista numerada de los diferentes experimentos en dicho plan.

Por ejemplo, si se tiene un plan de ejecución con dos bloques, donde el primer bloque tiene tres parámetros, siendo el primer parámetro un booleano que habilita a los otros dos parámetros; y donde el segundo bloque tiene tres parámetros, tal que el primer parámetro tiene ciertos valores dependiendo del valor del segundo parámetro; entonces la definición del plan de ejecución es la siguiente:

```

PlanDeEjemplo
##
base_de_ejemplo.conf
##
PARAM_BOOL;PARAM_1;PARAM_2
:[False] [#] [#]
:[True] [1;2;3] [2.5;3;5]
![Esta;linea;no] [se] [tiene;en;cuenta]
##
(PARAM_3;ASOC_PARAM_3);PARAM_4
:[(METODO_1;5);(METODO_1;8);(METODO_2;11);(METODO_2;13)] [A;B;C]
##

```

En este caso de ejemplo, utilizando como base el archivo *base_de_ejemplo.conf*, el script generaría 10 archivos de configuración para el primer bloque y 12 archivos de configuración para el segundo bloque. También, generaría un archivo *csv* con la siguiente lista:

Bloque 1		Configuración	PARAM_BOOL	PARAM_1	PARAM_2
1	False		N/A	N/A	
2	True		1	2.5	
3	True		1	3	
4	True		1	5	
5	True		2	2.5	
6	True		2	3	
7	True		2	5	
8	True		3	2.5	
9	True		3	3	
10	True		3	5	
Bloque 2		Configuración	(PARAM_3;ASOC_PARAM_3)	PARAM_4	
1		(METODO_1;5)		A	
2		(METODO_1;5)		B	
3		(METODO_1;5)		C	
4		(METODO_1;8)		A	
5		(METODO_1;8)		B	
6		(METODO_1;8)		C	
7		(METODO_2;11)		A	
8		(METODO_2;11)		B	
9		(METODO_2;11)		C	
10		(METODO_2;13)		A	
11		(METODO_2;13)		B	
12		(METODO_2;13)		C	

Ejecución de los bloques

Por el mismo motivo que en la creación de los archivos de configuración, la doble ejecución de los experimentos de cada bloque (i.e. una para crear el modelo y otra para utilizarlo), llevaría demasiado tiempo si se hiciera de forma manual. Entonces, para realizar las ejecuciones de una forma que sea, al menos semiautomática, se implementó un script el cual, dados los archivos de configuración, realiza la ejecución de los experimentos de forma semiautomática. Además, luego de la ejecución, calcula las distintas métricas sobre cada experimento y organiza los resultados en tablas, tanto en archivos *csv* como en

archivos *latex*. El script se divide en un script bash principal que organiza los diferentes pasos, y varios scripts Python secundarios que parsean los resultados de las diferentes métricas y crean los archivos *csv* y *latex*.

Primero, para la métrica de diferencia en el conteo de personas, el script crea un archivo *csv* por cada bloque, que contiene las distintas diferencias (mínimos, máximos y promedios; según blobs, seguimientos y valor interpolado) en cada experimento. Además, crea un archivo *latex* que contiene una tabla con la información de todos los bloques. Por otro lado, para los histogramas también genera un archivo *latex* que contiene un histograma de tres series (una de la diferencia con la cantidad de blobs, otra con la cantidad de seguimientos y otra con el valor interpolado) por cada experimento.

Segundo, para las métricas del MOT Challenge, el script ejecuta el *DevKit* con los experimentos y genera un archivo *csv* con los resultados para cada bloque y además, un archivo *latex* que contiene una tabla con los resultados de todos los bloques.

Por último, para las métricas de tiempo, el script genera un archivo *csv* con los resultados para cada bloque, junto con un archivo *latex* que contiene una tabla con los resultados de todos los bloques.

6.1.5. Resultados obtenidos

Ya con la ejecución semiautomática, se realizó la ejecución bloque a bloque de los planes de ejecución para los distintos filtros.

Durante el primer filtro (Sustracción de fondo), se eligieron los primeros tres experimentos de cada bloque, ordenados según el valor de MOTA. Luego, en los tres filtros restantes (Detección de blobs, Filtro de blobs y Seguimiento) se decidió elegir los tres mejores experimentos de cada bloque según tres métricas distintas, para generar mayor diversidad entre los experimentos seleccionados. Estas métricas son:

Métrica 1 Se elige el experimento que tiene el mayor valor de MOTA ya que, según el MOT Challenge, es el resultado que mejor realizó el seguimiento.

Métrica 2 Se elige el experimento que tiene la menor diferencia promedio en el conteo de personas (según el seguimiento).

Métrica 3 Se elige el experimento que, dentro de los mejores en MOTAL, tiene un mejor tiempo promedio de ejecución por frame; o, si el tiempo es parejo en todos, se elige el que tiene mejor diferencia promedio en el conteo de personas (según el seguimiento).

Además, la Métrica 1 y la Métrica 2 son también las métricas utilizadas para elegir los mejores dos experimentos del último bloque de los planes de ejecución intermedios, con el añadido de que los experimentos no deben tener un tiempo promedio de ejecución por frame que supere los 0.05 segundos (para que un vídeo de 20 FPS se pueda procesar en tiempo real).

Resultados del filtro Sustracción de fondo

Las mejores tres configuraciones para el primer bloque del filtro de sustracción de fondo se muestran en la Tabla 6.5.

	Primer	Segundo	Tercero
Valor de MOTA	45.9	43.6	43.2
Promedio de dif. en conteo	0.5	0.52	0.51
Tiempo promedio por frame	0.03205	0.02966	0.02814
Nro. de configuración	14	13	22
HISTORY	105	105	175
DIST_2_THRESHOLD	140	140	140
KNN_SAMPLES	5	3	3
DETECT_SHADOWS	False	False	False
SHADOW_THRESHOLD	N/A	N/A	N/A
FILTER_BY_COLOR	False	False	False
USE_BSUSTRACTOR_KNN	True	True	True
MOG2_LEARNING_RATE	N/A	N/A	N/A

Tabla 6.5: Mejores tres experimentos, para el primer bloque del filtro Sustracción de fondo.

Observando el comportamiento de los parámetros de configuración en las tablas de resultados del primer bloque que se encuentran en la sección I.6 del anexo I, se observa que:

- Los parámetros `DETECT_SHADOWS`, `SHADOW_THRESHOLD` y `FILTER_BY_COLOR` no influyen en el resultado. Respecto a los primeros dos parámetros, se debe a que en el vídeo utilizado no hay sombras para detectar, ya que el cielo está nublado.
- Cuando el parámetro `USE_BSUSTRACTOR_KNN` es `True` y, por tanto, se utiliza el algoritmo KNN, los resultados son mejores que cuando se utiliza el MOG2. Excepto, por los casos en que `DIST_2_THRESHOLD` es muy chico (35) y `KNN_SAMPLES` es muy grande (7), o viceversa.
- Los mejores resultados se dan cuando `DIST_2_THRESHOLD` tiene un valor intermedio (140), `KNN_SAMPLES` vale 3 o 5 y `USE_BSUSTRACTOR_KNN` es `True`.

Para el segundo y último bloque, las mejores tres configuraciones se muestran en la Tabla 6.6.

	Métrica 1	Métrica 2
Valor de MOTA	47.1	45.6
Promedio de dif. en conteo	0.54	0.4
Tiempo promedio por frame	0.02664	0.02703
Nro. de configuración	253	278
Nro. de configuración del bloque anterior	22	22
(GAUSSIANBLUR_SIZE_X;GAUSSIANBLUR_SIZE_Y)	(9;9)	(9;9)
(ERODE_SIZE_X;ERODE_SIZE_Y)	(2;2)	(4;4)
ERODE_TIMES	1	1
(DILATE_SIZE_X;DILATE_SIZE_Y)	(3;2)	(3;2)
DILATE_TIMES	1	3

Tabla 6.6: Mejores dos experimentos, para el segundo y último bloque del filtro Sustracción de fondo.

Por un lado, los experimentos del segundo bloque presentan una mejora de un 2.6 % en el MOTA y una mejora de un 20 % en la diferencia en el conteo de personas. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del segundo bloque que se encuentran en la sección I.6 del anexo I, se observa que los peores resultados se dan cuando (ERODE_SIZE_X,ERODE_SIZE_Y) toma un valor alto (4,4) y ERODE_TIMES tiene también un valor alto (3). Esta observación en general se cumple, aunque hay al menos un caso extremo en que dos configuraciones tienen esos valores y son idénticas en los demás parámetros, excepto por el valor del parámetro KNN_SAMPLES, pero son totalmente opuestas en el resultado (configuraciones 68 y 176).

Resultados del filtro Detección de blobs

Las mejores tres configuraciones para el primer y único bloque del filtro de detección de blobs se muestran en la Tabla 6.7.

	Métrica 1	Métrica 2
Valor de MOTA	47.1	45.6
Promedio de dif. en conteo	0.54	0.4
Tiempo promedio por frame	0.02678	0.02682
Nro. de configuración	3	127
Nro. de configuración del plan anterior	253	278
MIN_DIST_BETWEEN_BLOBS	N/A	N/A
FILTER_BY_AREA	N/A	N/A
MIN_AREA	N/A	N/A
MAX_AREA	N/A	N/A
DETECT_BLOBS_BY_BOUNDING_BOXES	True	True
EXPAND_BLOBS	True	True
EXPAND_BLOBS_RATIO	0.2	0.2

Tabla 6.7: Mejores dos experimentos, para el primer y último bloque del filtro Detección de blobs.

Por un lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del primer bloque que se encuentran en la sección I.7 del anexo I, se observa que:

- Los mejores resultados se dan cuando `DETECT_BLOBS_BY_BOUNDING_BOXES` es *True*, `EXPAND_BLOBS` es también *True* y `EXPAND_BLOBS_RATIO` tiene un valor intermedio (0.2).
- Cuando `DETECT_BLOBS_BY_BOUNDING_BOXES` es *False*, los mejores resultados se dan cuando `EXPAND_BLOBS` es *False*. Luego, cuando `EXPAND_BLOBS` es *True*, los mejores resultados se dan cuando `EXPAND_BLOBS_RATIO` tiene un valor intermedio (0.2), mientras que los peores resultados se dan cuando `EXPAND_BLOBS_RATIO` tiene un valor chico (0.1).

Por otro lado, los experimentos del primer bloque no presentan ninguna mejora en la detección comparado con los experimentos del último bloque del filtro anterior. Esto se debe a que los valores de los parámetros de los mejores experimentos de este bloque son los mismos valores que los que tienen los parámetros en el archivo de configuración base y, por tanto, los mismos valores que se utilizaron para el filtro anterior.

Resultados del filtro de blobs

Las mejores tres configuraciones para el primer bloque del filtro de blobs se muestran en la Tabla 6.8.

	Métrica 1	Métrica 2	Métrica 3
Valor de MOTA	47.9	45.6	47.5
Promedio de dif. en conteo	0.55	0.4	0.52
Tiempo promedio por frame	0.02542	0.02674	0.02718
Nro. de configuración	29	35	4
Nro. de configuración del plan anterior	3	127	3
USE_HISTOGRAMS_FOR_PERSON_DETECTION	True	True	True
CONFIDENCE_MATRIX_UPDATE_TIME	5000	2500	2500
USE_CONFIDENCE_LEVELS	False	True	True
CONFIDENCE_LEVEL_0	N/A	0.7	0.7
CONFIDENCE_LEVEL_1	N/A	0.1	0.1

Tabla 6.8: Mejores tres experimentos, para el primer bloque del filtro de blobs.

Por un lado, los experimentos del primer bloque presentan una mejora de un 1.7 % en el MOTA. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del primer bloque que se encuentran en la sección I.8 del anexo I, se observa que:

- El parámetro CONFIDENCE_MATRIX_UPDATE_TIME no influye en el resultado, así como, con algunas pocas excepciones, tampoco influye en el resultado el parámetro CONFIDENCE_LEVEL_1.
- Los experimentos que fueron mezclados con el mejor experimento del filtro anterior según la Métrica 1, son los que resultan mejores también en este primer bloque.
- Los 62 experimentos del bloque se pueden dividir en diez conjuntos de experimentos que tienen los mismos resultados.
- Dentro de los experimentos que fueron mezclados con el mejor experimento del filtro anterior según la métrica 1, los resultados, ordenados por la métrica 1, se organizan según los siguientes valores: primero, aquellos para los cuales los valores de los parámetros USE_CONFIDENCE_LEVELS o USE_HISTOGRAMS_FOR_PERSON_DETECTION son *False*; segundo, aquellos cuyos valores para el parámetro CONFIDENCE_LEVEL_0 se encuentran mayormente en 0.9 y alguno en 0.7; tercero los que presentan valores para el parámetro CONFIDENCE_LEVEL_0 en 0.5; y cuarto los que toman valores para el parámetro CONFIDENCE_LEVEL_0 en 0.7.
- Dentro de los experimentos que fueron mezclados con el mejor experimento del filtro anterior según la métrica 2, los resultados, ordenados por la métrica 1, se organizan según los siguientes valores: primero, los que presentan valores para el parámetro CONFIDENCE_LEVEL_0 en 0.9 o en 0.7; segundo, aquellos cuyo valor para el parámetro USE_CONFIDENCE_LEVELS es *False*; tercero los que muestran valores del parámetro CONFIDENCE_LEVEL_0 en 0.5; y cuarto el que tiene USE_HISTOGRAMS_FOR_PERSON_DETECTION en *False*.

Para el segundo bloque, las mejores tres configuraciones se muestran en la Tabla 6.9.

	Métricas 1 y 3	Métrica 2
Valor de MOTA	48.3	46.6
Promedio de dif. en conteo	0.44	0.39
Tiempo promedio por frame	0.05717	0.0267
Nro. de configuración	77	78
Nro. de configuración del bloque anterior	35	35
ASPECT_RATIO	3	3
SCALE	1.1	1.1
(WINSTRIDE_0;WINSTRIDE_1)	(4;4)	(8;8)

Tabla 6.9: Mejores tres experimentos, para el segundo bloque del filtro de blobs.

Por un lado, los experimentos del segundo bloque presentan una mejora de un 0.8 % en el MOTA y una mejora de un 2.5 % en la diferencia en el conteo de personas. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del segundo bloque que se encuentran en la sección I.8 del anexo I, se observa que:

- Cuanto menor es el valor del parámetro **SCALE**, mayor es el tiempo promedio de ejecución por frame (llegando en algunos casos a ser de 1 segundo por frame) y, en general, también es mejor el resultado.
- Cuanto mayor es el valor del parámetro **SCALE**, peor es el resultado, especialmente en el conteo de personas.
- El resto de los parámetros no presentan un comportamiento claro.

Para el tercer bloque, las mejores tres configuraciones se muestran en la Tabla 6.10.

	Métrica 1	Métricas 2 y 3
Valor de MOTA	48.3	46.6
Promedio de dif. en conteo	0.44	0.39
Tiempo promedio por frame	0.05721	0.02675
Nro. de configuración	2	7
Nro. de configuración del bloque anterior	77	78
USE_SQUARE_REGION_FOR_VERIFY	True	True
SQUARE_REGION_RADIUS	2	4

Tabla 6.10: Mejores tres experimentos, para el tercer bloque del filtro de blobs.

Observando el comportamiento de los parámetros de configuración en las tablas de resultados del tercer bloque que se encuentran en la sección I.8 del anexo I, se observa que cuando el valor del parámetro **USE_SQUARE_REGION_FOR_VERIFY** es *True*, se obtiene un mejor resultado que cuando es *False*. Además se observa que los distintos valores del parámetro **SQUARE_REGION_RADIUS** no afectan el resultado.

Para el cuarto bloque, las mejores tres configuraciones se muestran en la Tabla 6.11.

	Métrica 1	Métricas 2 y 3
Valor de MOTA	48.3	46.6
Promedio de dif. en conteo	0.44	0.39
Tiempo promedio por frame	0.05962	0.02664
Nro. de configuración	1	4
Nro. de configuración del bloque anterior	2	7
(BORDER_AROUND_BLOB_0; BORDER_AROUND_BLOB_1)	(0.25;0.25)	(0.25;0.25)

Tabla 6.11: Mejores tres experimentos, para el cuarto bloque del filtro de blobs.

Por un lado, los experimentos del cuarto bloque no presentan ninguna mejora en la detección comparado con los experimentos del bloque anterior. Esto se debe a que los valores de los parámetros de los mejores experimentos de este bloque son los mismos valores que los que tienen los parámetros en el archivo de configuración base y, por tanto, los mismos valores que se utilizaron para el bloque anterior. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del cuarto bloque que se encuentran en la sección I.8 del anexo I, se observa que cuanto mayor son los parámetros BORDER_AROUND_BLOB_0 y BORDER_AROUND_BLOB_1, peor es el resultado.

Para el quinto y último bloque, las mejores tres configuraciones se muestran en la Tabla 6.12.

	Métrica 1	Métrica 2
Valor de MOTA	48.3	46.6
Promedio de dif. en conteo	0.44	0.39
Tiempo promedio por frame	0.04631	0.02373
Nro. de configuración	1	3
Nro. de configuración del bloque anterior	1	4
PERSON_DETECTION_PARALLEL_MODE	True	True

Tabla 6.12: Mejores tres experimentos, para el quinto y último bloque del filtro de blobs.

Por un lado, los experimentos del quinto bloque no presentan ninguna mejora en la detección comparado con los experimentos del bloque anterior. Esto se debe a que PERSON_DETECTION_PARALLEL_MODE es el único parámetro del quinto bloque y lo que hace es ejecutar la detección de personas en paralelo, por tanto, no mejora la detección sino que lo que mejora es el tiempo de procesamiento. Por otro lado, observando el comportamiento del parámetro de configuración en las tablas de resultados del cuarto bloque que se encuentran en la sección I.8 del anexo I, se observa que cuando el parámetro PERSON_DETECTION_PARALLEL_MODE es *True*, el tiempo de procesamiento promedio por frame mejora entre un 10 % y un 20 % en comparación a cuando el valor del parámetro es *False*. Por esta mejora en el tiempo es que se eligieron los experimentos donde el parámetro vale *True* sobre aquellos en que vale *False*. A pesar de la mejora en el tiempo durante el análisis experimental, cuando no se utiliza un modelo para la detección de personas (i.e. en una ejecución normal), el tiempo de procesamiento resulta mejor cuando

se utiliza el parámetro PERSON_DETECTION_PARALLEL_MODE en *False*.

Resultados del filtro de seguimiento

Las mejores tres configuraciones para el primer bloque del filtro de seguimiento se muestran en la Tabla 6.13.

	Métrica 1	Métrica 2	Métrica 3
Valor de MOTA	49.8	43.6	45.5
Promedio de dif. en conteo	0.41	0.37	0.42
Tiempo promedio por frame	0.05656	0.02639	0.02647
Nro. de configuración	6	15	16
Nro. de configuración del plan anterior	1	3	3
USE_HISTOGRAMS_-FOR_TRACKING	True	True	True
(HISTOGRAM_COMPARISON_METHOD; THRESHOLD_COLOR)	(HELLINGER; 0.5)	(INTERSECTION; 1.0)	(HELLINGER; 0.5)
PRIMARY_HUNG_ALG_-COMPARISON_METHOD; HOD_WEIGHTS	(0,0,1)	(0,0,1)	(0,0,1)
THRESHOLD_COLOR	N/A	N/A	N/A

Tabla 6.13: Mejores tres experimentos, para el primer bloque del filtro de seguimiento.

Por un lado, los experimentos del primer bloque presentan una mejora de un 3.1 % en el MOTA y una mejora de un 5.1 % en la diferencia en el conteo de personas. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del primer bloque que se encuentran en la sección I.9 del anexo I, se observa que:

- Los mejores resultados se dan si se usan histogramas para la comparación de color (i.e. USE_HISTOGRAMS_FOR_TRACKING es *True*).
- Como método de comparación de los histogramas (parámetro HISTOGRAM_COMPARISON_METHOD) el mejor método es HELLINGER, seguido por: MANHATTAN, CHI_SQUARED_ALT, INTERSECTION y EUCLIDEAN. Contrariamente, los peores son: CHEBYSEV, KL_DIV y CHI_SQUARED.

Para el segundo bloque, las mejores tres configuraciones se muestran en la Tabla 6.14.

	Métrica 1	Métrica 2	Métrica 3
Valor de MOTA	51.8	51.1	48.3
Promedio de dif. en conteo	0.37	0.34	0.42
Tiempo promedio por frame	0.04622	0.04605	0.0237
Nro. de configuración	44	41	120
Nro. de configuración del bloque anterior	6	6	15
PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS	(0.33,0.34,0.33)	(0.25,0.25,0.5)	(0.5,0.25,0.25)
SECONDARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS	(0,1,0)	(0.25,0.25,0.5)	(1,0,0)

Tabla 6.14: Mejores tres experimentos, para el segundo bloque del filtro de seguimiento.

Por un lado, los experimentos del segundo bloque presentan una mejora de un 4.0 % en el MOTA y una mejora de un 8.1 % en la diferencia en el conteo de personas. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del segundo bloque que se encuentran en la sección I.9 del anexo I, se observa que:

- Los mejores resultados se dan cuando el parámetro PRIMARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS tiene los pesos distribuidos, sin anular ninguno de los tres componentes.
- El parámetro SECONDARY_HUNG_ALG_COMPARISON_METHOD_WEIGHTS no cambia significativamente los resultados. Esto se puede deber a que hay una buena sustracción de fondo y detección de blobs, por lo que no ocurren muchos casos de oclusión donde aplique el algoritmo húngaro secundario.

Para el tercer y último bloque, las mejores tres configuraciones se muestran en la Tabla 6.15.

	Métrica 1	Métrica 2	Métrica 3
Valor de MOTA	52.7	50.7	50.8
Promedio de dif. en conteo	0.43	0.33	0.52
Tiempo promedio por frame	0.04595	0.04606	0.02381
Nro. de configuración	7	279	294
Nro. de configuración del bloque anterior	44	41	120
MAX_SECONDS_WITHOUT_UPDATE	1	8	1
MAX_SECONDS_TO_PREDICT_POSITION	0.5	4	0.5
MAX_SECONDS_WITHOUT_ANY_BLOB	0.5	0.5	0.5
MIN_SECONDS_TO_BE_ACCEPTED_IN_GROUP	1.5	1.5	0.5

Tabla 6.15: Mejores tres experimentos, para el tercer y último bloque del filtro de seguimiento.

Por un lado, los experimentos del tercer bloque presentan una mejora de un 1.7 % en el MOTA y una mejora de un 2.9 % en la diferencia en el conteo de personas. Por otro lado, observando el comportamiento de los parámetros de configuración en las tablas de resultados del tercer bloque que se encuentran en la sección I.9 del anexo I, se observa que:

- El parámetro **MIN_SECONDS_TO_BE_ACCEPTED_IN_GROUP** ofrece los mejores resultados cuando tiene un valor menor a dos segundos y mayor a cero. Si por el contrario, el parámetro tiene un valor de cero, surgen muchos falsos positivos (tracklets fantasmas) que no desaparecen rápidamente. En cuanto al caso en que tiene un valor mayor a dos segundos, sucede que muchos seguimientos válidos se eliminan al entrar a un grupo (e.g. por oclusión), al no tener el suficiente tiempo de vida.
- El parámetro **MAX_SECONDS_TO_PREDICT_POSITION** ofrece los mejores resultados cuando tiene un valor igual o menor a un segundo y mayor a cero. Con esos valores, cuando un objeto entra en oclusión se predice su posición por un tiempo razonable, tal que si el objeto sale rápido de la oclusión, la predicción funciona; en cambio, si permanece mucho tiempo en oclusión, se deja de predecir antes de que la predicción se vuelva imprecisa.
- El parámetro **MAX_SECONDS_WITHOUT_ANY_BLOB** ofrece los mejores resultados cuando tiene un valor menor a dos segundos. Con esos valores, se eliminan rápidamente los tracklets que pierden al objeto, y los tracklets que siguen a objetos que salen de la escena. Si el tiempo fuera mayor, los tracklets cuyos objetos desaparecen quedarían vivos y se asociarían con otros objetos.
- El parámetro **MAX_SECONDS_WITHOUT_UPDATE** ofrece los mejores resultados cuando tiene un valor menor o igual a cuatro segundos. Con esos valores, permite que los objetos entren en oclusión por hasta cuatro segundos sin eliminar sus seguimientos. Si el valor fuera mucho más grande, podrían quedar tracklets fantasmas mucho tiempo dentro de un grupo y generar cambios de tracklet entre objetos.

En síntesis, durante el análisis experimental se mejoró el MOTA un 14.8 % y se mejoró el conteo de personas en un 34 %, comparando con los mejores tres experimentos resultantes del primer bloque del filtro de sustracción de fondo. Si se compara el MOTA de 52.7 obtenido en el sistema implementado, contra los mejores algoritmos de seguimiento participantes del MOT Challenge, que tienen un MOTA de 55 en promedio, se ve un muy buen resultado de el sistema implementado. Sin embargo, el MOTA de los algoritmos del MOT Challenge surge de un promedio sobre distintas escenas, varias de las cuales son mucho más complejas que la escena sobre la que se trabajó en el análisis experimental, por lo que la comparación con dichos algoritmos no es justa.

En cuanto al resto de las métricas a estudio, el resultado de las métricas del MOT Challenge para los tres experimentos elegidos se muestra en la Tabla 6.16, mientras que el resultado de la diferencia en el conteo de personas se muestra en la Tabla 6.17 y en los histogramas de las figuras 6.1, 6.2 y 6.3. Además, para los tres mejores experimentos se muestran los tiempos promedio de procesamiento por cada filtro en la Tabla 6.18 y los tiempos máximos de procesamiento por cada filtro en la Tabla 6.19.

Conf	Rcll	Prcn	FAR	MT	PT	ML	FP	FN	IDs	FM	MOTA	MOTP
7	78.6	76.3	1.31	10	9	0	1040	910	63	253	52.7	64.2

279	78.9	74.5	1.44	10	9	0	1148	898	55	235	50.7	64.1
294	75.3	76.5	1.24	8	11	0	986	1053	58	234	50.8	64.3

Tabla 6.16: Resultados del MOT Challenge para los tres mejores experimentos del último bloque del filtro de seguimiento.

Conf	Nro. de Personas vs GT			Nro. de Tracklets vs GT			Nro. interpolado vs GT		
	Media	Mín.	Máx.	Media	Mín.	Máx.	Media	Mín.	Máx.
7	0.67	0	4	0.43	0	3	0.43	0	3
279	0.67	0	4	0.33	0	3	0.33	0	3
294	1.14	0	5	0.52	0	4	0.52	0	4

Tabla 6.17: Resultados de la diferencia en el conteo de personas para los tres mejores experimentos del último bloque del filtro de seguimiento.

Conf	Detección y Sustracción de fondo		Detección clasificación de blobs		Detección de personas	Seguimiento	Total
	0.00357	0.00051	0.03894	0.00294	0.04595		
7	0.00355	0.00053	0.03902	0.00296	0.04606		
294	0.00356	0.00049	0.01712	0.00265	0.02381		

Tabla 6.18: Tiempos promedio de ejecución, en segundos, para los tres mejores experimentos del último bloque del filtro de seguimiento.

Conf	Detección y Sustracción de fondo		Detección clasificación de blobs		Detección de personas	Seguimiento	Total
	0.00552	0.00094	0.09065	0.00657	0.10369		
7	0.00559	0.00080	0.08779	0.00562	0.09980		
294	0.00578	0.00091	0.04628	0.00556	0.05853		

Tabla 6.19: Tiempos máximos de ejecución, en segundos, para los tres mejores experimentos del último bloque del filtro de seguimiento.

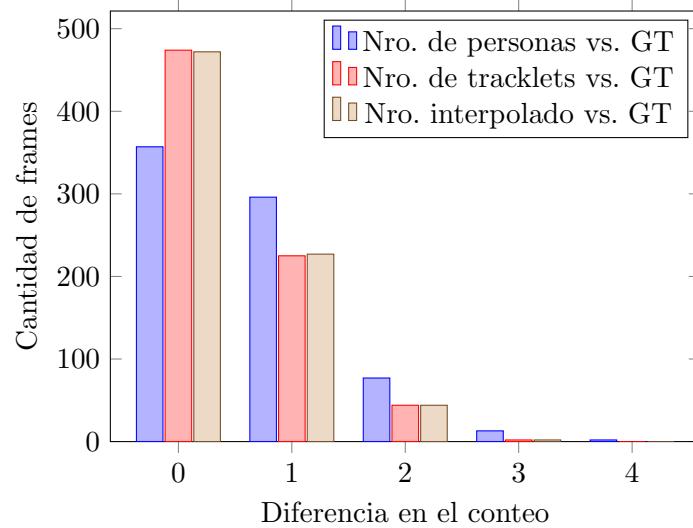


Figura 6.1: Histograma de la diferencia en el conteo de personas para el experimento número 7 del último bloque del filtro de seguimiento.

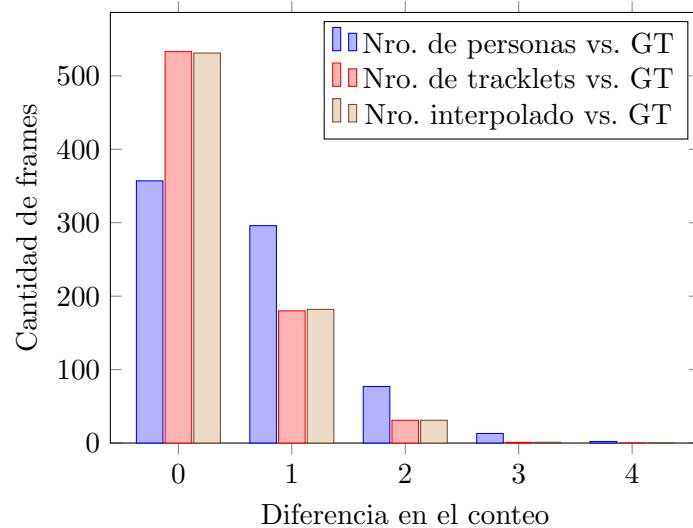


Figura 6.2: Histograma de la diferencia en el conteo de personas para el experimento número 279 del último bloque del filtro de seguimiento.

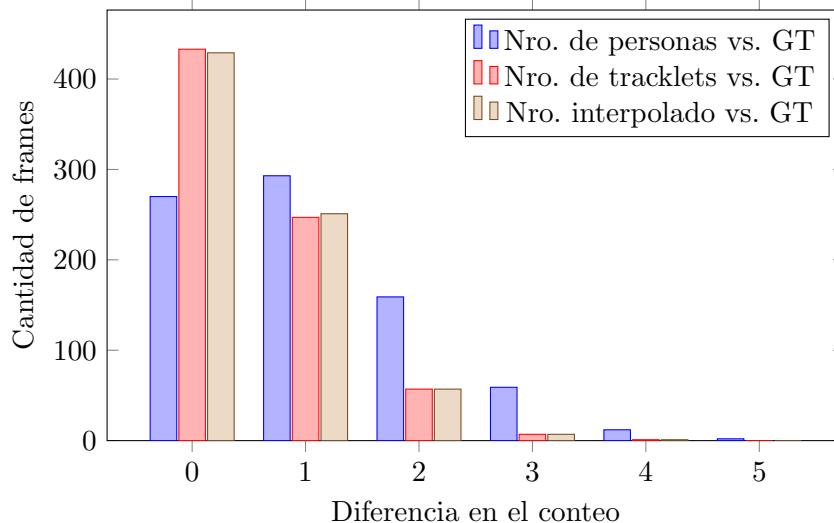


Figura 6.3: Histograma de la diferencia en el conteo de personas para el experimento número 294 del último bloque del filtro de seguimiento.

6.2. Análisis del módulo Buscador de patrones

Para el análisis del módulo Buscador de patrones se grabó un vídeo en un ambiente controlado. El vídeo tiene una duración de 2 minutos con 51 segundos, una resolución de 808 x 600 píxeles y fue grabado con luz natural. Los actores del vídeo son nueve y caminan de forma aleatoria por el escenario, saliendo y entrando de la escena en reiteradas ocasiones. Durante el transcurso del vídeo ocurren un conjunto de hechos de interés, los cuales se pretende detectar con el sistema que presenta este informe. Los tipos de hechos a detectar hacen uso de las primitivas que el sistema provee, combinándolas en diferentes ordenes, cantidades y valores, según el hecho que describen. Durante el análisis experimental del módulo Buscador de patrones, no se hace uso del mismo vídeo que se utilizó para el análisis experimental del módulo Análisis del módulo Reconocimiento y seguimiento, ya que ocurren combinaciones de primitivas que derivan en patrones muy sencillos.

Los hechos de interés que ocurren en el vídeo son: aglomeración de personas, simulación de arrebato, merodeo y giros bruscos en el trayecto de una persona. A continuación se detallan, en orden cronológico, los hechos de interés ocurridos en el vídeo.

- Aglomeración 1: Nueve personas caminan dispersas por la escena. En el minuto 00:23 todas caminan hacia el centro del escenario, permaneciendo en ese lugar hasta el minuto 00:28.
- Arrebato 1: Una persona (víctima) camina lentamente por el centro del escenario, cuando en el minuto 00:57 otra persona (victimario) emprende camino desde la esquina superior derecha hacia la víctima. Al llegar el victimario a la posición de la víctima en el minuto 01:00, comienza un forcejeo de unos dos segundos. El victimario le quita la pertenencia a la víctima y emprende viaje, corriendo, hacia la esquina inferior izquierda del escenario, arribando en el minuto 01:07.
- Arrebato 2: Una persona (víctima) camina lentamente por el escenario, otra persona

(victimario) se acerca caminando y lo intercepta en el minuto 01:36, forcejean durante 6 segundos y el victimario se da a la fuga con las pertenencias de la víctima, corriendo, hasta desaparecer de la escena en el minuto 01:45.

- **Giro 90:** Una persona ingresa caminando a la escena en el minuto 02:09. Continúa caminando hasta llegar al centro en el minuto 02:15, y realiza un giro de 90 grados, continuando con su caminata hasta llegar al borde inferior de la escena en el minuto 02:29, momento en el cual se detiene.
- **Aglomeración:** Nueve personas caminan dispersas por el escenario. En el minuto 02:30 todas caminan hacia el centro del escenario, permaneciendo en ese lugar hasta el minuto 02:51.

Se definen cuatro patrones a ser detectados. Estos modelan el comportamiento de los hechos de interés ocurridos en el vídeo. A continuación se presentan los parámetros a ser buscados.

- **arrebato:** Caminar durante un tiempo mayor a 3,5 segundos y luego correr por un tiempo mayor a 2,8 segundos.
- **five_is_too_much:** Detectar una aglomeración de al menos 5 personas, por un período de tiempo mayor a 1,5 segundos.
- **four_is_too_much:** Detectar una aglomeración de al menos 4 personas, por un período de tiempo mayor a 1,9 segundos.
- **walk_rotate_walk:** Caminar durante al menos dos segundos y realizar un giro de aproximadamente 90 grados, para luego continuar caminando por al menos dos segundos.

Como entorno de ejecución de los experimentos, se utilizó una computadora portátil Sony Vaio modelo 2015 con procesador Intel Core i7-4500U con cuatro núcleos a 1.8 GHz, 8 GB de RAM, y sistema operativo Ubuntu 16.04 y versión de Python 3.5.2. Todos los recursos, a excepción de aquellos utilizados por el sistema operativo, quedan disponibles para la ejecución de este sistema.

6.2.1. Métricas

Para el módulo Buscador de patrones se busca evaluar que los patrones detectados reflejen la realidad de forma exacta. En el caso de este módulo, la exactitud significa detectar los patrones preestablecidos en el momento preciso en el que ocurren. Se entiende como bueno el hecho de que, cuanto más próximo al momento inicial del hecho éste sea reportado, mejor calificado debe estar el rendimiento del módulo Buscador de patrones. Además, es necesario tener en cuenta los falsos positivos y falsos negativos que puedan suceder.

Teniendo en cuenta lo antes mencionado, se toma la distancia temporal (en segundos) entre el inicio real del hecho y el momento en el que se reporta, siempre que se hable de un caso de detección verdadera positiva.

6.2.2. Resultados obtenidos

Esta subsección muestra una comparación entre los hechos reportados por el módulo Buscador de patrones (en orden cronológico) y los hechos reales que ocurren en el vídeo. Al finalizar se detallan aquellos eventos ocurridos pero no reportados.

El primer reporte corresponde a el hecho denominado Aglomeración 1. En la Figura 6.4 se puede observar que dos reportes fueron generados. El primero a los 00:25 minutos, con la información de una aglomeración conformada por 5 personas, y el segundo en el minuto 00:26, con la información de una aglomeración conformada por 6 personas. La distancia temporal desde el inicio real del hecho hasta el primer reporte es de dos segundos. La cantidad real de personas en la aglomeración es nueve, lo que resulta en cuatro y tres personas más que las reportadas en cada caso.

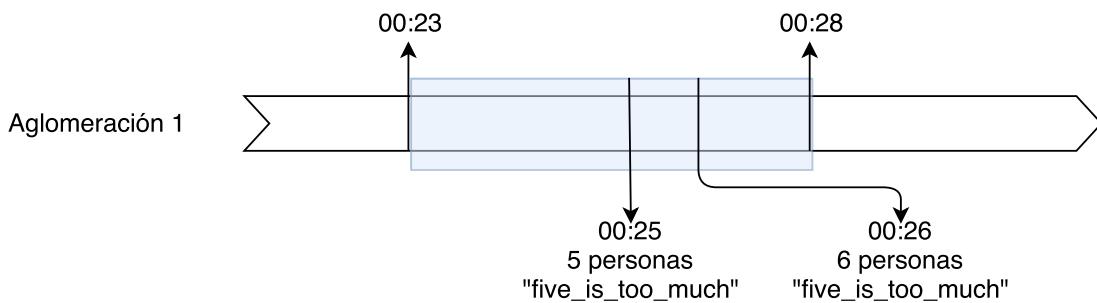


Figura 6.4: Detección del hecho de interés Aglomeración 1.

El segundo reporte corresponde al hecho denominado Arrebato 1. En la Figura 6.5 se puede observar que el reporte es generado en el minuto 1:04, mientras que la etapa más significativa del arrebato, esto es, el forcejeo entre la víctima y el victimario, ocurre en el minuto 1:00. Por lo tanto, la distancia temporal entre el Arrebato 1 y su correspondiente reporte es tomada desde el minuto 1:00 al minuto 1:04, dando como resultado cuatro segundos.

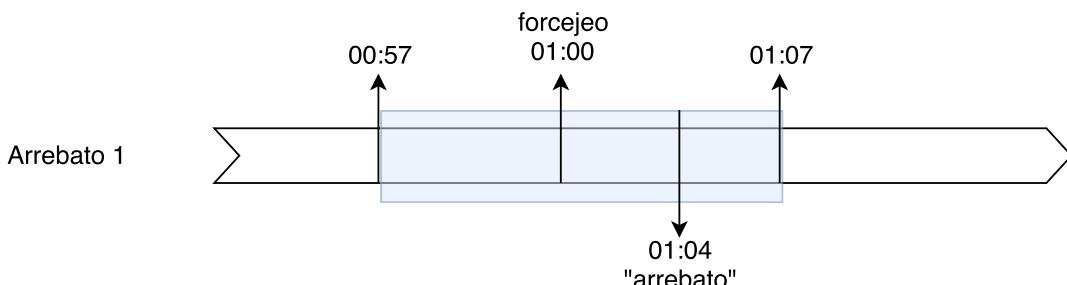


Figura 6.5: Detección del hecho de interés Arrebato 1.

El tercer reporte registrado corresponde al hecho denominado Aglomeración 2. En la Figura 6.6 se puede observar que se generó un único reporte en el minuto 02:49. El reporte generado indica que la aglomeración fue conformada por cuatro personas. La distancia temporal desde el inicio real del hecho hasta el reporte es 19 segundos, mientras que cantidad real de personas en la aglomeración es nueve, lo que frente a las cuatro reportadas indica un error de cinco.



Figura 6.6: Detección del hecho de interés Aglomeración 2.

Los reportes detallados anteriormente fueron el total de los generados por el sistema durante el procesamiento del vídeo. No se registraron reportes de hechos inexistentes (i.e., no existen falsos positivos), pero la cantidad de hechos reportados es menor a la cantidad total de hechos ocurridos (i.e., existen falsos negativos).

El primer caso no reportado corresponde al hecho denominado Arrebato 2. En la Figura 6.7 se puede observar que el seguimiento de la persona que actúa de victimario es perdido a causa de la oclusión con la víctima y un tercer actor durante el forcejeo. Este es un caso claro en el que el algoritmo de resolución de grupos de *blobs* genera un resultado erróneo. Al perderse el seguimiento, se genera una nueva instancia de la estructura que sigue a la persona, perdiendo el historial reciente de primitivas cumplidas, de manera que la información que arriba al módulo Buscador de patrones es incompleta e insuficiente para cumplir con el patrón pre cargado denominado *arrebato*.



Figura 6.7: Escena en la que se desarrolla el hecho de interés Arrebato 2.

El segundo caso no reportado corresponde a el hecho denominado Giro 90. En este caso se observa que el módulo Buscador de patrones experimenta una gran sensibilidad a las pequeñas variaciones de posición entre dos posiciones consecutivas arribadas al módulo. De esta forma, se detecta una secuencia de giros durante trayectos que deberían ser tenidos en cuenta en su totalidad como un único giro. El hecho de detectar una secuencia de giros en lugar de uno único da el indicio de que para procesar este tipo de primitivas es necesario utilizar un historial más extenso de las últimas posiciones registradas.

Capítulo 7

Conclusiones y trabajo futuro

Esta sección del informe presenta un conjunto de conclusiones objetivas que se desprenden del proceso de investigación e implementación del sistema. Además, se presentan propuestas de trabajo futuro, enfocadas en corregir metodologías implementadas e impulsar la continuidad del trabajo de mejora e innovación sobre los métodos utilizados en el sistema.

7.1. Conclusiones

En este proyecto de grado se realiza un estudio de las técnicas de procesamiento de imágenes e inteligencia computacional aplicadas en sistemas de videovigilancia. Además, se brinda la arquitectura e implementación de un sistema que aplica las técnicas relevadas y que tiene la capacidad de procesar imágenes en tiempo real desde múltiples fuentes de información en paralelo.

En la implementación se utilizan técnicas de procesamiento de imágenes e inteligencia computacional basadas en distribuciones de probabilidad Gaussiana, clasificadores supervisados no paramétricos como k-NN y SVM, y algoritmos de asignación óptima como el Algoritmo Húngaro o de predicción de movimientos como Filtros de Kalman, entre otras técnicas. La mayor parte de las implementaciones utilizadas es provista por bibliotecas de código abierto como OpenCV, Scipy y Numpy, entre otras. Además, en la implementación se utilizan tecnologías tales como Python, RabbitMQ y Javascript.

El trabajo explora un conjunto de ramas de la temática, aportando una base de estudio de valor para cualquier equipo de trabajo que tenga la intención de implementar sistemas de este tipo. Además, mediante la implementación del sistema, se aporta una prueba fehaciente de la viabilidad en la construcción de dichos sistemas.

El análisis experimental del módulo encargado del reconocimiento y seguimiento registra resultados que se encuentran dentro de la media de los sistemas que tienen un fin similar. Con respecto al módulo encargado de la búsqueda de patrones, los resultados obtenidos indican que la propuesta de detectar primitivas para luego buscar secuencias pre-cargadas de estas (la definición de patrones), es en parte viable. Sin embargo, se debe solucionar la detección de patrones de corta duración, los cuales distorsionan la detección de la secuencia de los mismos. Este último problema podría ser tratado como ruido en una señal.

Para ambos módulos principales se hace evidente la falta de algoritmos que, mediante la aplicación de diferentes heurísticas, puedan calcular valores óptimos (o, en su defecto,

sub-óptimos) de los parámetros de configuración. El hecho de que se definen más de 50 parámetros de este tipo redonda en un alto costo de instalación en masa para múltiples escenarios. Por otro lado cabe destacar que, de forma consciente, se dejaron de lado problemas relacionados a los cambios de luz o movimientos aparentes en la cámara y que, sin el estudio y la integración al sistema de técnicas que resuelvan estos problemas, resulta prácticamente imposible aplicar en espacios abiertos un sistema como el implementado.

7.2. Trabajo futuro

En esta sección se desarrollan puntos que describen ciertas funcionalidades que escapan al alcance del proyecto académico, pero que resultan relevantes en una hipotética puesta en producción o en la continuidad de la investigación del problema que se resuelve. También, se presentan puntos que, finalizada la etapa de implementación, se detectó que pudieron ser resueltos de una mejor forma.

7.2.1. Reemplazo del algoritmo alternativo de detección de personas

En la Sección 5.4.3 se presenta un algoritmo de clasificación de personas que tiene en cuenta la relación de aspecto de los *blobs* que las contienen. Si bien este algoritmo cumple con los requerimientos planteados (disminuir la carga de procesamiento que se tendría en caso que se aplicara el reconocedor de personas de forma continua), presenta problemas en escenarios con cierto grado de dinamismo. Se entiende que un escenario es altamente dinámico cuando el rango de las dimensiones de los *blobs* que contienen al menos a una persona varía en el tiempo. Por ejemplo, un escenario en el que las personas caminan por un sendero cercano a la cámara en la mañana y por otro sendero muy alejado de la cámara en la noche, es considerado como un escenario dinámico. En este caso, los *blobs* siguen distintos rangos de dimensiones dependiendo de la hora del día. Es por esto que se propone la sustitución del algoritmo de detección de personas por algoritmos de clasificación que permitan una actualización de patrones de entrenamiento de forma *online* y sin mayores esfuerzos.

A modo de prueba de concepto, se recabaron datos del ancho y el alto de los *blobs* reconocidos en una instancia del sistema y la clase correspondiente a estos. Los valores posibles son 0 cuando el reconocedor de personas no detecta ninguna persona y 1 cuando detecta al menos una persona. Al conjunto de datos se le aplicó el clasificador k-NN. Utilizando la herramienta *Weka* [77], se buscó la mejor combinación de valores de *k* y pesos aplicados a la distancia, obteniendo como resultado un valor de *k* igual a 27 y un peso de distancia inverso a la misma. Las pruebas del algoritmo con el clasificador k-NN, utilizando el valor de *k* obtenido, arrojó un 82 % de aciertos y un 18 % de error. La Figura 7.1 muestra un gráfico de las áreas correspondientes a cada clase (clase persona en rojo, clase no persona en azul) y los puntos del conjunto de datos de entrenamiento.

Al algoritmo se le podrían integrar otro tipo de características, como histogramas de color por zona del *blob* (por ejemplo, particionando el *blob* en tres filas horizontales correspondientes a la cabeza, torso y piernas). Además se podrían tener en cuenta niveles de confianza del resultado, utilizando el porcentaje de vecinos correspondiente a la clase a la que clasifica (i.e. de los 27 vecinos tomados en cuenta, cuantos son de la clase resultado).

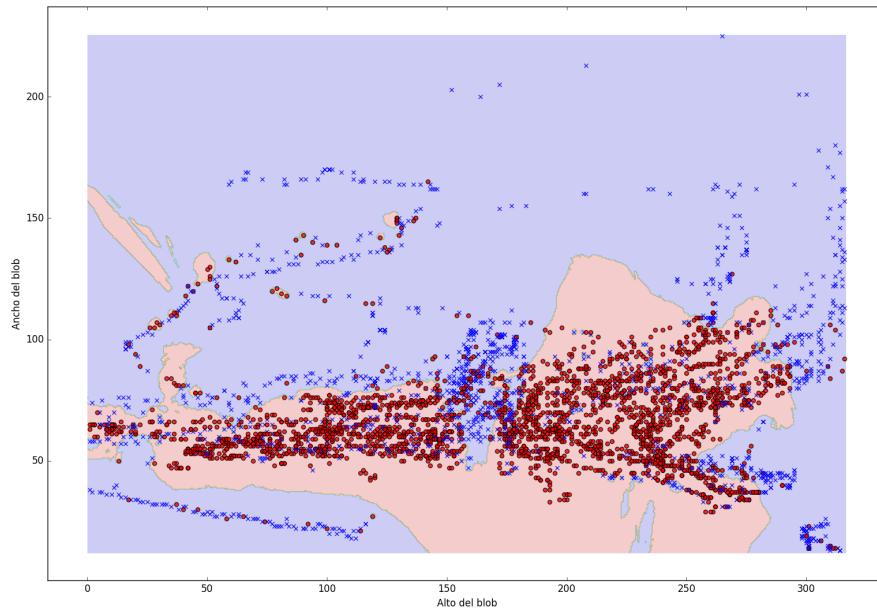


Figura 7.1: Clasificador k-NN que simula el algoritmo de reconocimiento de personas con una exactitud cercana al 82 %. Gráfico construido con biblioteca *Matplotlib*.

7.2.2. Manejo de variaciones en la iluminación de las imágenes

Las imágenes tomadas por cámaras de vigilancia instaladas en la vía pública se exponen constantemente a cambios en la iluminación, ya sea por las luces de los vehículos que circulan, por el alumbrado público, cambios en el clima o cualquier otro imprevisto. Esta constante exposición afecta considerablemente el resultado del tratamiento de imágenes, ya que provoca que los valores de los píxeles varíen demasiado. Se propone la investigación e implementación de técnicas para mitigar el problema de la iluminación. Una opción posible sería utilizar filtros de Kalman para predecir los valores de los píxeles (o grupos de píxeles) y poder conocer así cuándo una variación en la iluminación no debería afectar en las detecciones. Otra opción puede ser estudiar la similitud entre *frames*, lo cual ayudaría a mitigar, sobre todo, los problemas que generan los cambios bruscos de iluminación.

7.2.3. Adaptación automática de parámetros de configuración

El sistema implementado presenta un gran número de parámetros de configuración, siendo algunos generales y otros ligados directamente al escenario que se procesa. Se plantea la implementación de subsistemas que, mediante las metaheurísticas adecuadas, se encarguen de adaptar de forma autónoma los parámetros de funcionamiento del sistema: desde las relaciones de aspecto de las personas, pasando por velocidades consideradas para caminata y corrida, hasta la eliminación de blobs por su forma anómala. Esto brindaría al sistema una gran facilidad para lidiar con la heterogeneidad de escenarios en los que puede trabajar y ahorraría gran parte del proceso de instalación en nuevos escenarios,

al disminuir notoriamente la etapa de ajustes de parámetros.

7.2.4. Generador de patrones a reconocer desde una interfaz amigable

El sistema tiene predefinidos patrones a reconocer, los cuales son una serie ordenada de primitivas junto a un cuantificador y a un valor. El sistema carga la definición de los patrones interpretando un archivo en formato CSV (Comma-Separated Values) definido por el usuario. Se plantea como trabajo futuro integrar al Panel de control una interfaz amigable, la cual facilite la creación de nuevos patrones. De esta manera, se mitigarían errores de sintaxis en el archivo de definición y se permitiría a cualquier usuario, independientemente de su capacidad técnica, integrar patrones al sistema.

7.2.5. Reconocimiento de múltiples objetos de interés

Existen aplicaciones del sistema en las que resulta de gran valor el reconocimiento de múltiples objetos de interés. Por ejemplo, la detección de la matrícula de los vehículos que transitan en la escena, el reconocimiento de armas de fuego, el conteo de diferentes tipos de vehículo para estudios viales, etc. Es así que se plantea la modificación en la arquitectura del filtro *Filtro de blobs*, tal que permita integrar fácilmente múltiples filtros al sistema. Por ejemplo, brindando herramientas al usuario para que implemente su propio filtro siguiendo una interfaz proporcionada y luego integre el código en forma de *plug-in* al sistema.

Apéndice A

Conceptos teóricos

Este apéndice es de vital importancia para la total comprensión de los conceptos utilizados en la implementación del sistema. Se presenta en forma de apéndice con el fin de no interrumpir el flujo normal de lectura de quienes cuenten con el marco teórico necesario, a la vez que sirve de consulta para aquellos lectores que cuenten con el mismo de forma parcial.

A.1. Advanced Message Queuing Protocol

Advanced Message Queuing Protocol (AMQP) es la especificación del estándar abierto que nace en 2006 en el ámbito financiero, como informa su autor J. O'Hara en [78]. El protocolo pertenece a la capa de aplicación y es por definición un protocolo binario, consecuentemente más veloz que protocolos de texto plano (prioriza ser altamente eficiente en lugar de ser más amigable al humano).

AMQP define un conjunto de comportamientos tanto para quien sirve los mensajes como para quien los consume. Utiliza una combinación de técnicas de encolamiento y enrutamiento junto a la exactitud y seguridad de la transmisión.

El estándar se puede dividir en dos áreas principales: el modelo de enrutamiento y el modelo de transporte. El protocolo especifica cuidadosamente la semántica del modelo de enrutamiento con el fin de alcanzar mayores niveles de interoperabilidad entre diferentes sistemas. El modelo de transporte puede utilizar diferentes protocolos de transporte, siendo el principal TCP/IP.

Los mensajes en AMQP son autocontenidos, de larga vida, inmutables y opacos. El tamaño de los mismos es ilimitado, siendo un mensaje de 4GB tan manejable como uno de 4KB. Los mensajes contienen cabezales que AMQP utiliza para el enrutamiento.

Las colas (*Queues*) son el concepto central del protocolo y son el destino final de todo mensaje. Pueden ser persistidas en memoria o en disco y pueden buscar y re-ordenar mensajes.

Los intercambiadores (*Exchanges*) son el servicio de entrega de mensajes. Contienen instrucciones de cómo y hacia dónde dirigir los mensajes. El cliente es quien elige el tipo de *exchanges* a utilizar. En [79] se listan cuatro tipos posibles: *direct*, *topic*, *fanout* y *headers*. Los *exchanges* del tipo *direct* encolan los mensajes únicamente en aquellas colas en las que coincide exactamente la clave de enrutado con la del cabezal del mensaje. Son utilizadas generalmente para mensajes de unidifusión. Los del tipo *topic* entregan una copia del mensaje a cada cola cuyo receptor expresa su interés mediante una expresión

regular, si la clave de enrutamiento del mensaje cumple el patrón. Los *exchanges fanout* entregan a todas las colas subscriptas al *exchange*. Son generalmente utilizados para *broadcasting*. Por último, los *exchanges* del tipo *headers* examinan todos los cabezales de un mensaje comparándolos frente a predicados de consultas provistos por los clientes interesados. Sin importar el tipo de *exchange*, estos no guardan mensajes aunque sí los valores de *binding*. Los *bindings* son los argumentos suministrados a los *exchanges* para habilitar el ruteo de mensajes. La asociación entre claves de ruteo y colas de destino la provee el emisor de los mensajes en el caso de los *exchanges* del tipo *direct*, y el receptor en el caso de los *exchanges* del tipo *topic*.

Existen varias implementaciones del protocolo entre las que destacan iMatix OpenAMQ, la cual es desarrollada en C y se le adjudica ser la primer implementación; Qpid bajo licenciamiento Apache o Rabbit desarrollada en Erlang/OTP y pensada para sopor tar paralelismo y gran cantidad de transferencias en red. Para esta última se dispone de *pika*, biblioteca que implementa los controladores de Rabbit para utilizarlo desde Python.

A.2. Sustracción de fondo

Sea $\vec{x}^{(t)}$ el valor RGB de un píxel en el tiempo t , la sustracción de fondo se define como el problema de decidir si ese píxel pertenece al fondo de la imagen o a algún elemento del frente. Todos los píxeles, tanto los del frente como los del fondo, quedan representados por probabilidades. Los píxeles pertenecientes al frente son representados por $p(\vec{x}^{(t)}|FG)$, mientras que los del fondo quedan representados por $p(\vec{x}^{(t)}|BG)$. La decisión de si un píxel pertenece al fondo de la imagen viene dada por el resultado de

$$p(\vec{x}^{(t)}|BG) > c_{thr} (= p(\vec{x}^{(t)}|FG)p(FG)/p(BG)), \quad (\text{A.1})$$

en donde $p(BG)$ y $p(FG)$ denotan la probabilidad de que un píxel cualquiera pertenezca al fondo o al frente de la imagen respectivamente. Existen variadas técnicas y algoritmos para realizar la sustracción de fondo. En este trabajo se aplicaron dos que ya cuentan con su desarrollo en la biblioteca OpenCV: *Mixture Of Gaussians 2* y *K-Nearest Neighbours*.

A.2.1. Mixture Of Gaussians 2

El algoritmo *Mixture Of Gaussians 2 (MOG2)* de OpenCV trabaja representando cada píxel de la imagen como una mezcla de K gaussianas. *MOG2* clasifica como un *Métodos paramétrico* ya que necesita la especificación de una distribución (distribución normal, en este caso) para efectuar los cálculos. Los *Métodos paramétricos* presentan estimaciones con respecto a los parámetros de la población de interés y seleccionan la cantidad de gaussianas adecuada para cada píxel (Mordvintsev y Abid Rahman [80]).

OpenCV dispone de una implementación de este método que es utilizada a través de una clase que implementa un *Gaussian Mixture Model background subtractor* (Zivkovic [57], [58]).

Distribución Gaussiana

Una distribución gaussiana (o distribución normal) es la distribución de probabilidad más importante debido a la cantidad de fenómenos que pueden ser explicados gracias a ella. También es denominada campana de Gauss debido a que su función de probabilidad, al ser representada gráficamente, denota una forma de campana (ver Figura A.1a).

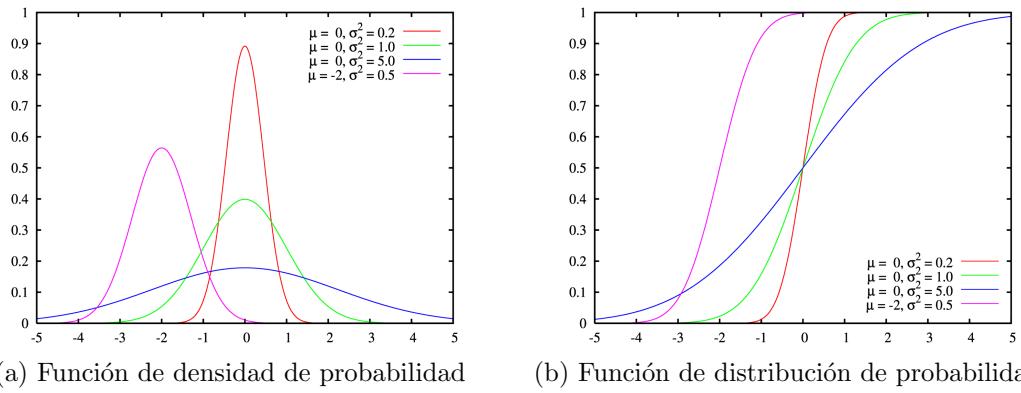


Figura A.1: Funciones de densidad y distribución de probabilidad para distintos ejemplos de campanas de Gauss, con sus respectivos valores de media (μ) y desviación estándar (σ).
Fuente: Inductiveload (Trabajo propio) [Dominio Público], Wikimedia Commons (a) - https://en.wikipedia.org/wiki/File:Normal_Distribution_PDF.svg, (b) - https://en.wikipedia.org/wiki/File:Normal_Distribution_CDF.svg.

Una de las características más importantes de esta distribución es que casi cualquier distribución, tanto discreta como continua puede ser aproximada a una normal, bajo ciertas condiciones.

La distribución normal, así como la curva que la representa, presentan las siguientes características[81]:

- La curva normal tiene forma de campana con un solo pico en el centro de la distribución, como se muestra en la Figura A.1a. La media aritmética, la mediana y la moda de la distribución son iguales y se encuentran en el pico.
- Es simétrica alrededor de su media.
- La curva normal es asintótica, lo que significa que la curva se acerca cada vez más al eje X , pero jamás llega a tocarlo.

Un gran número de variables aleatorias continuas pueden ser representadas mediante una distribución gaussiana. Una variable aleatoria continua es aquella que puede asumir un número infinito de valores dentro de determinado rango. Para indicar que una variable aleatoria X sigue una distribución normal de media μ y desviación estándar σ se utiliza la siguiente expresión:

$$X \sim N(\mu, \sigma)$$

Se denota P como la probabilidad de que una variable aleatoria X alcance un valor determinado entre dos números reales a y b . El valor de P coincide con el área encerrada bajo la curva de la *función de densidad de probabilidad* $f(x)$ entre los puntos a y b , siendo esa función:

$$f(x) = \frac{e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}}{\sigma\sqrt{2\pi}}$$

es decir:

$$P(a \leq X \leq b) = \int_a^b f(x)dx \quad (\text{A.2})$$

entonces:

- La distribución normal es simétrica respecto a su media μ
- El área total encerrada bajo la curva $f(x)$ vale 1:

$$\int_{-\infty}^{+\infty} f(x)dx = P(-\infty < X < +\infty) = 1$$

- Debido a que X es una variable aleatoria continua se cumple:

$$P(X = a) = \int_a^a f(x)dx = 0, \forall a \in \mathbb{R} \Rightarrow P(X \leq a) = P(X < a)$$

A.2.2. K-Nearest Neighbours

El algoritmo *K-Nearest Neighbours* (KNN) pertenece al tipo de algoritmos denominado *Métodos no paramétricos*. Los *Métodos no paramétricos* son aquellos que no se encuentran sujetos a la forma de la distribución de la población y no requieren que las observaciones estén dadas en escala de intervalo.

KNN utiliza *KDE: Kernel Density Estimation* (KDE) para estimar la función de densidad de probabilidad (también denominada *Núcleo*) de la muestra. KDE es un *Método no paramétrico* utilizado para determinar el núcleo de una muestra aleatoria.

OpenCV cuenta con una implementación del algoritmo KNN para la sustracción de fondo. El algoritmo es utilizado mediante la clase *BackgroundSubtractorKNN*. La misma fue desarrollada a partir del trabajo de Zivkovic y van der Heijden [58].

A.3. Detección de blobs

Existen variadas técnicas que permiten extraer blobs a partir de una imagen. La extracción de blobs de una imagen es un proceso por el cual un algoritmo se encarga de reconocer partes conexas en la imagen, para luego devolver valores que representen las mismas (su centro, área, etc.)

Para este trabajo, una de las utilizadas fue *SimpleBlobDetector*, la cual cuenta con su implementación en la biblioteca OpenCV.

A.3.1. Simple Blob Detector

El algoritmo *Simple Blob Detector*, implementado por la clase *SimpleBlobDetector* de OpenCV, sigue una serie de pasos sencillos [82] que se enumeran a continuación:

1. Convierte la imagen original en varias imágenes binarias mediante la aplicación de *umbralización* con varios umbrales, definiendo valores máximos y mínimos y la distancia entre los umbrales intermedios. La umbralización es un método de segmentación cuyo objetivo es convertir una imagen en escala de grises a una nueva con solo dos niveles.
2. Extrae componentes conexas de cada imagen binaria mediante la aplicación de la operación *findContours* de OpenCV y calcula sus centros.
3. Agrupa los centros de varias imágenes binarias de acuerdo a sus coordenadas. Los centros que se encuentren a una distancia menor o igual a la definida según los parámetros del algoritmo forman un único blob.

4. Para los grupos obtenidos, calcula el centro y el radio de cada blob y devuelve los datos de localización y tamaño de los mismos.

La clase, además, permite aplicar filtros extra a los *blobs* obtenidos:

- Color: se filtran aquellos *blobs* para los cuales la intensidad de la imagen binaria medida desde su centro difiere con respecto al valor establecido.
- Área: se filtran aquellos *blobs* cuyo valor de área se encuentra fuera de un intervalo definido.
- Circularidad: se filtran aquellos *blobs* cuyo valor de circularidad se encuentra fuera de un intervalo definido. La circularidad se define como el resultado de $\frac{4\pi \cdot \text{Área}}{\text{Perímetro}^2}$. El valor 1 significa que es un círculo.
- Relación de inercia: se filtran aquellos *blobs* cuyo valor de relación de inercia (*Inertia ratio* en inglés) se encuentra fuera de un intervalo definido. La relación de inercia puede ser vista como aquella que indica cuan ovalada es la forma del *blob*. Un valor de relación de inercia igual a 0 indica una línea, mientras que un valor igual a 1 indica un círculo.
- Convexidad: se filtran aquellos *blobs* cuyo valor de convexidad se encuentra fuera de un intervalo definido. La convexidad se define como $\frac{\text{Área del blob}}{\text{Área de la envolvente del blob}}$. La envolvente del *blob* es aquella forma convexa que envuelve al *blob* y que mejor se ajusta a la forma del mismo.

A.4. Reconocimiento de objetos de interés

En el campo del procesamiento de imágenes y la visión por computadora, se denombra *reconocimiento de objetos* a la tarea de encontrar y/o identificar objetos en una imagen o secuencia de imágenes. El reconocimiento de objetos toma como base algoritmos de emparejamiento, aprendizaje y reconocimiento de patrones, utilizando técnicas que pueden ser basadas en apariencia o basadas en características.

Se procesa la imagen, eliminando la mayor parte de la información que no es relevante para el reconocimiento de los objetos, extrayendo solamente aquellos parámetros útiles para la identificación de los mismos. Como menciona Rossius, “*La selección de estos parámetros es una de las tareas más críticas en el reconocimiento.*” [83].

El esquema general para la detección de objetos de interés consta de tres pasos:

- Generación de candidatos: dentro de la imagen de entrada se generan zonas que son candidatas a contener objetos del tipo de los buscados. Una forma de realizar esto es desplazar una ventana de tamaño fijo a través de toda la imagen y a diferentes escalas, logrando generar una gran cantidad de candidatos.
- Clasificación de candidatos: se estudia cada una de las zonas generadas en la etapa anterior de manera individual utilizando un clasificador. El clasificador indica cuales de las zonas procesadas son potencialmente instancias de los objetos buscados.
- Refinamiento de la decisión: se corrigen ciertas situaciones que aparecen por la aplicación del clasificador. Por ejemplo, el solapamiento de detecciones en la misma zona.

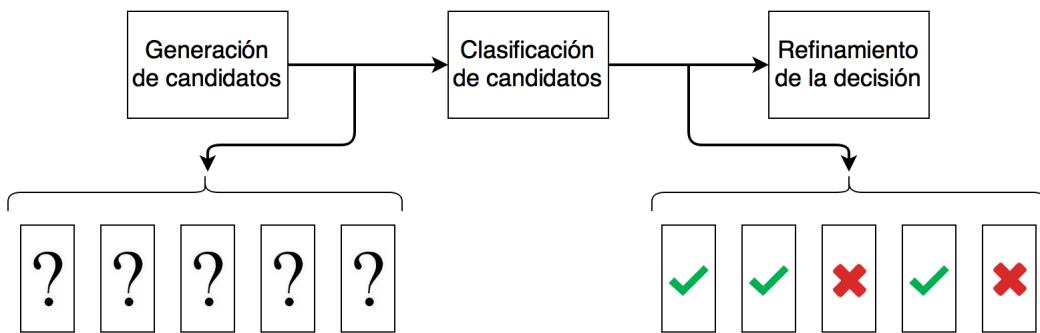


Figura A.2: Esquema general para la detección de objetos de interés.
Fuente: Basada en Valveny, Universitat Autònoma de Barcelona [84].

Centrándose en la clasificación de candidatos, es posible reconocer que intervienen dos componentes. Un *descriptor*, el cual permite representar el contenido de una ventana mediante vectores, y un *claseficador* que permite determinar como se fija la frontera entre las zonas que pertenecen al objeto y aquellas que pertenecen al fondo.

Existen varios métodos que se pueden utilizar para los descriptores y los clasificadores. En este trabajo se utilizó *HOG* como método para el descriptor y *SVM* como método de clasificación. La biblioteca OpenCV cuenta con una implementación de un detector de personas que utiliza *HOG* y *SVM*.

A.4.1. Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) es un algoritmo de extracción de características utilizado en sistemas de reconocimiento de objetos. Este método se basa en la evaluación de histogramas locales (normalizados) de los gradientes orientados de la imagen en una grilla densa ([63], [85]). A diferencia de otros algoritmos utilizados para la descripción de objetos, como puede ser Local Binary Patterns (LBP), HOG explota la información del gradiente de la imagen.

El gradiente de la imagen aporta información que puede ser relevante ya que, en general, valores altos del gradiente se corresponden con el contorno o silueta de los objetos. Esto se debe a que los gradientes describen la apariencia y forma de los grupos de píxeles [86].

Este descriptor utiliza la información del gradiente a través de histogramas locales. Los histogramas locales son calculados en celdas de pequeño tamaño distribuidas por toda la imagen [84].

Los histogramas proporcionan información sobre las distintas orientaciones de los contornos que dominan en cada una de las posiciones de la imagen. Esta información permite distinguir la forma de los objetos que aparecen en la imagen, lo cual permite utilizar esa información como base para la detección y el reconocimiento de distintos objetos.

Finalmente, los histogramas que fueron calculados localmente en cada una de las posiciones de la imagen, son agrupados en bloques de un tamaño mayor y utilizados para normalizar la representación final y hacerla más invariante ante cambios de iluminación y distorsiones en la imagen. La representación final resulta en la concatenación de cada uno de los bloques obteniendo la representación global de la imagen en forma de vector de características.

Gradiente

El gradiente se puede definir como el cambio en la intensidad de una imagen en cierta dirección, aquella dirección para la cual ese cambio de intensidad es máximo [87]. Se calcula para cada píxel de la imagen y queda definido por dos valores:

- La dirección en donde el cambio de intensidad es máximo.
- La magnitud del cambio en esa dirección.

Esos dos valores permiten distinguir diferentes situaciones acerca de la configuración local (alrededor del píxel), en relación al cambio de contraste y a la forma.

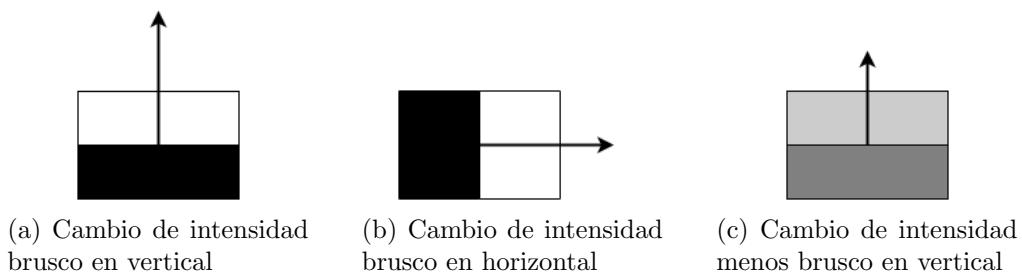


Figura A.3: Cambios en la intensidad entre píxeles vecinos.
Fuente: Basada en Valveny, Universitat Autònoma de Barcelona [87].

Para el descriptor HOG, el gradiente se calcula a partir de la diferencia de intensidad de los píxeles vecinos, tanto en dirección horizontal como vertical. Dado un punto (x, y) para el cual se desea calcular el gradiente, se calcula la diferencia en la dirección horizontal (dx) y en la dirección vertical (dy)

$$\begin{aligned} dx &= I(x+1, y) - I(x-1, y) \\ dy &= I(1, y+1) - I(x, y-1) \end{aligned} \tag{A.3}$$

A partir de las ecuaciones (A.3) se calcula la orientación global σ y la magnitud g del gradiente como:

$$\begin{aligned} \sigma(x, y) &= \arctan \frac{dy}{dx} \\ g(x, y) &= \sqrt{dx^2 + dy^2} \end{aligned} \tag{A.4}$$

La Figura A.3 muestra tres casos de cambio de intensidad entre píxeles adyacentes. La Figura A.3(a) refleja un cambio de intensidad en el cual se obtiene un vector gradiente de dirección vertical y una magnitud máxima. La Figura A.3(c) muestra el mismo caso pero con una magnitud menor debido a que el cambio de intensidad es menor. La Figura A.3(b) muestra el caso en que el gradiente tiene dirección horizontal y magnitud máxima.

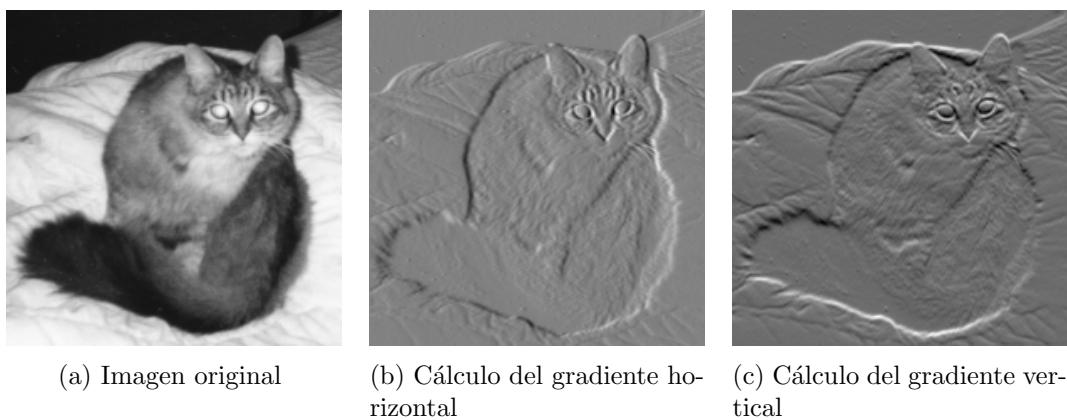


Figura A.4: Cálculo del gradiente en las direcciones horizontal y vertical.
Fuente: Basada en Njw000 (Trabajo Propio) [Dominio Público], Wikimedia Commons https://commons.wikimedia.org/wiki/File:Intensity_image_with_gradient_images.png.

Las ecuaciones anteriores son válidas tanto para imágenes en escala de grises como a color. Para las imágenes en escala de grises el cálculo del gradiente se realiza directamente utilizando como información el valor de intensidad de la imagen en ese píxel mientras que para el cálculo del gradiente en imágenes a color es necesario calcular el gradiente para cada uno de los tres canales (R, G, B) y luego seleccionar aquel gradiente del canal que presenta mayor magnitud. Con esto se prioriza el color dominante de forma local en cada uno de los píxeles.

Histogramas de orientaciones en una celda

Valveny [88] muestra que el cálculo de los histogramas de orientaciones necesita que ciertos parámetros sean definidos:

- Tamaño de la celda: en general se suelen utilizar valores de entre 6 y 8 píxeles, tanto en ancho como en alto.
- División del rango de orientaciones en un número de intervalos fijo: en caso que se tome la orientación del gradiente con signo, el rango de orientaciones varía entre 0° y 360° , mientras que si se trabaja con la orientación del gradiente sin signo, el rango de valores varía entre 0° y 180° . Esta opción hace que dos gradientes con la misma dirección pero sentidos inversos sean considerados equivalentes, por lo que quedan asignados al mismo intervalo.
- Cantidad de intervalos en los que se divide el rango de orientaciones: utilizando la orientación del gradiente sin signo, 9 intervalos suele ser un valor adecuado. Así, cada intervalo agrupa un rango de orientaciones de 20° cada uno.

Cada uno de los gradientes calculados quedará finalmente asociado a algunos de los intervalos definidos, según su orientación y magnitud. Así, el valor de cada intervalo del histograma final, se obtiene acumulando la magnitud de cada uno de los gradientes

asignados al intervalo. La expresión matemática que formaliza lo dicho anteriormente es:

$$h(k) = \sum_{(x,y) \in C} \omega_k(x, y) g(x, y) \quad (\text{A.5})$$

donde C representa una celda, (x, y) un píxel perteneciente a esa celda (con su respectivo gradiente), k es una posición determinada en el histograma, $g(x, y)$ es la acumulación de la magnitud de los gradientes, $\delta\theta$ es el rango del intervalo del histograma y $\omega_k(x, y)$ es un factor que determina la asociación del gradiente a dicho intervalo y se define como

$$\omega_k(x, y) = \begin{cases} 1 & \text{Si } (k - 1)\delta\theta \leq \theta(x, y) < k\delta\theta \\ 0 & \text{en caso contrario} \end{cases}$$

Este modelo de cálculo de los histogramas de orientaciones, si bien es bastante simple puede presentar ciertos problemas. Por ejemplo, gradientes con orientaciones muy similares pueden quedar asignados a intervalos diferentes, lo cual puede ocasionar que pequeñas variaciones en la imagen de entrada provoquen variaciones significativas en la solución final. Una solución posible a este problema es asignar cada gradiente a los dos intervalos más cercanos, con un peso proporcional a la distancia de la orientación al centro del intervalo, lo que hace que el factor de asociación del gradiente al intervalo se calcule como:

$$\omega_k(x, y) = \max \left(0, 1 - \frac{\theta(x, y) - \theta_k}{\delta\theta} \right) \quad (\text{A.6})$$

en donde $\theta(x, y)$ es la orientación del gradiente en el punto (x, y) , θ_k es la orientación del centro del intervalo k y $\delta\theta$ es el rango del intervalo del histograma. Esta formulación permite que cada gradiente contribuya a los dos intervalos más cercanos, en función de su orientación.

Generalizando el resultado de (A.5) para todas las celdas de la imagen se obtiene:

$$h_{ij}(k) = \sum_{(x,y) \in C_{ij}} \omega_k(x, y) g(x, y) \quad (\text{A.7})$$

donde para cada una de las celdas C_{ij} se tiene su correspondiente histograma h_{ij} , en el cual se acumulan los gradientes $g(x, y)$ ponderados por el factor de asignación $\omega_k(x, y)$, para todos los puntos (x, y) de la celda.

Este esquema en el cual cada gradiente pertenece únicamente al histograma de la celda a la que pertenece puede presentar el mismo problema que se mencionó anteriormente para la asignación de las orientaciones. Píxeles muy cercanos pueden llegar a ser asignados a celdas diferentes, por lo que pequeños cambios en la forma o la localización del objeto, puede dar lugar a cambios significativos en la representación final. Para solucionarlo se aplica la misma estrategia, en donde cada píxel es asignado a las cuatro celdas más cercanas, asignando un peso proporcional a la distancia del píxel al centro de cada celda. Calculando las respectivas distancias d_{ij}^x y d_{ij}^y del píxel al centro de la celda, se obtienen los factores de asignación $\omega_{ij}^x(x, y)$ y $\omega_{ij}^y(x, y)$ de cada uno de los píxeles a la celda, en

las direcciones x e y .

$$\begin{aligned}\omega_{ij}^x(x, y) &= \max \left(0, 1 - \frac{d_{ij}^x}{\delta x} \right) \\ \omega_{ij}^y(x, y) &= \max \left(0, 1 - \frac{d_{ij}^y}{\delta y} \right)\end{aligned}\quad (\text{A.8})$$

Combinando esos valores con el factor de asignación de cada gradiente a uno de los intervalos del histograma ($w_k(x, y)$) y aplicándolo a (A.7) se obtiene:

$$h_{ij}(k) = \sum_{(x,y)} \omega_{ij}^x(x, y) \omega_{ij}^y(x, y) w_k(x, y) g(x, y) \quad (\text{A.9})$$

Necesidad de normalizar

Uno de los principales objetivos de cualquier descriptor es el de conseguir la máxima invarianza posible ante cambios que se produzcan en la imagen (iluminación, aspecto, etc.) [89]. Cuando se producen cambios de iluminación en la imagen, la intensidad de los gradientes en la misma también cambia, lo que provoca que los valores de los histogramas también se vean alterados.

Para minimizar las diferencias en la descripción de las imágenes, es conveniente normalizar los valores de los histogramas. Se busca que la magnitud global del gradiente sea similar en ambas imágenes. Es posible que los cambios de iluminación no sean constantes a lo largo de toda la imagen por lo que no es suficiente aplicar una única normalización uniforme a toda la imagen, sino que es preferible una normalización local que se adapte a cada una de las diferentes zonas. Con este fin se introduce el concepto de **bloque**, el cual no es más que la agrupación de cierta cantidad de celdas vecinas. HOG se suele configurar para que utilice bloques de dos celdas en horizontal por dos celdas en vertical. Se obtienen los histogramas correspondientes a cada una de las celdas del bloque y se concatenan, lo cual permite obtener una representación vectorial del bloque.

$$v = (x_1, \dots, x_n)$$

La normalización de los histogramas se efectúa a nivel de bloque. Esto significa que se calcula la normalización del vector v , resultado de concatenar todas las celdas. El vector es normalizado utilizando la norma $L2$, por lo que se obtiene un nuevo vector v' tal que

$$v' = \frac{v}{\sqrt{\|v\|_2^2 + \epsilon}}$$

De esta forma se garantiza que la magnitud global del vector es 1, a la vez que se minimizan las variaciones locales en el vector debido al contraste.

Descriptor final

Como se menciona en [88], el descriptor final se obtiene solapando los bloques donde, para cada bloque, se calcula el histograma normalizado de sus celdas. Luego se concatenan todos los resultados de los histogramas de bloque.

Cada celda contribuye a la descripción de varios bloques debido al solapamiento y a que para cada bloque se calcula una normalización distinta.

El descriptor presenta los siguientes parámetros:

- Tamaño de la celdas
- Signo del gradiente
- Cantidad de intervalos del histograma de orientaciones
- Cantidad de celdas por bloque

Estos parámetros terminan por determinar la cantidad de dimensiones que tiene el descriptor HOG, o lo que es lo mismo, la cantidad de componentes que tiene el vector

$$\text{hog} = (x_1, \dots, x_n)$$

siendo n la cantidad de componentes del vector. Esta cantidad se obtiene a partir de

$$n = b * c_b * i \quad (\text{A.10})$$

donde b es la cantidad de bloques de la imagen, c_b es la cantidad de celdas por bloque e i es el número de intervalos de los histogramas de orientaciones. La cantidad de bloques b se calcula como el resultado del producto de la cantidad de bloques en la componente horizontal b_x por la cantidad de bloques en la componente vertical b_y , donde

$$\begin{aligned} b_x &= c_x - c_{b_x} + 1 \\ b_y &= c_y - c_{b_y} + 1 \end{aligned}$$

siendo c_x y c_y la cantidad de celdas de la imagen en las componentes horizontal y vertical respectivamente y c_{b_x} y c_{b_y} la cantidad de celdas por bloque en las componentes horizontal y vertical.

A.4.2. Support Vector Machines

El algoritmo *Support Vector Machines* (SVM) fue desarrollado por Boser et al. y presentado en el año 1992 [90]. Es un algoritmo de clasificación, el cual en su formulación básica es un algoritmo clasificador binario lineal. Se le llama binario porque permite distinguir únicamente entre dos clases, y lineal porque la frontera de clasificación es una línea recta en un espacio bidimensional y un hiperplano en un espacio multidimensional. Esta formulación puede ser extendida para soportar la clasificación en múltiples clases lo cual suele ser un problema común en el procesamiento de imágenes.

En general, no existe una única línea recta (o hiperplano, dependiendo del espacio en el cual se trabaje) que permita separar los datos en clases, como se muestra en la Figura A.5. El principal objetivo de este algoritmo es encontrar una línea recta (o hiperplano) de separación que maximice el margen de los datos, lo que se traduce a encintran aquella línea recta (o hiperplano) que se encuentre a la mayor distancia posible de los puntos de datos de cada categoría [91].

SVM sigue un **modelo discriminativo** ya que clasifica las muestras sin necesidad de calcular las funciones de densidad de probabilidad de las clases, al contrario de los **modelos generativos** que sí lo hacen [92]. SVM clasifica las muestras a partir de un conjunto de entrenamiento lo suficientemente representativo. El plano intermedio de la región más amplia del espacio de características, que queda definido por el margen, es

la solución buscada. A este plano también se le llama hiperplano solución. La Figura A.5 muestra dos posibles hiperplanos que cumplen el objetivo de separar el espacio de datos en clases. El hiperplano 'B' lo separa con un margen pequeño mientras que el hiperplano 'A' lo hace con el margen máximo y por lo tanto representa al hiperplano solución buscado.

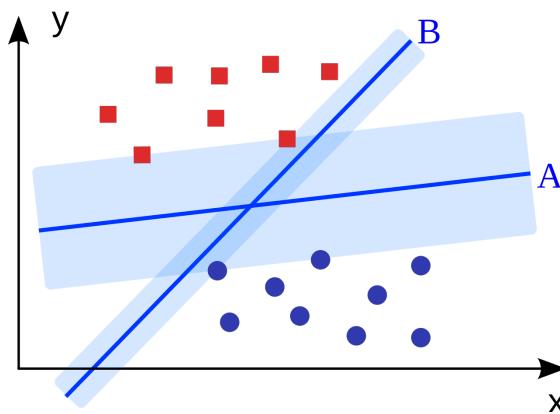


Figura A.5: Dos posibles hiperplanos que partitionan el espacio de muestras, con sus respectivos márgenes.
Fuente: Fabian Bürger (Trabajo propio) [Creative Commons], Wikimedia Commons https://de.wikipedia.org/wiki/Datei:Svm_intro.svg.

Para hallar el hiperplano solución, SVM solo tiene en cuenta un número limitado de muestras del conjunto de entrenamiento. Estas muestras presentan ciertas propiedades concretas y son denominadas **vectores de soporte** (*Support Vectors*). Los vectores de soporte son aquellos puntos para los cuales la distancia entre los planos que los contienen, la cual es denominada margen, es la máxima posible [93].

Esta manera de resolver el SVM puede plantearse matemáticamente como un problema de optimización en donde la función que se desea optimizar es el margen [94]. El desarrollo matemático de SVM se puede encontrar en la sección Desarrollo matemático de este mismo apéndice.

SVM también puede ser aplicado a conjuntos de datos que no sean linealmente separables, o sea, aplicado a conjuntos de datos en los cuales existe solapamiento de clases. Dos conjuntos de puntos son linealmente separables si existe una función lineal que los separa. Para lograr que SVM sea eficiente en este tipo de escenario, existen dos enfoques posibles.

El primer enfoque es utilizar un *márgen suave*, en el cual se permite que existan ciertos vectores que violan la condición del margen. Esto puede hacer que ciertas muestras queden erróneamente clasificadas.

El segundo enfoque es utilizar la técnica denominada *kernel trick*. Esta técnica se basa en el hecho de que conjuntos no linealmente separables pueden transformarse en linealmente separables llevándolos a un espacio de dimensión mayor. En el nuevo espacio de características ya se puede aplicar SVM sin problemas. Esta transformación presenta una propiedad interesante y es que no es necesario encontrar la transformación entre los espacios de características, solo es necesario aplicar el producto escalar en el nuevo espacio para encontrar la solución del SVM.

Desarrollo matemático

Como mencionan Valveny y Vilariño [94], el hecho de que SVM sea un clasificador lineal implica que es posible encontrar un hiperplano solución, tal que el mismo puede ser expresado a partir de la ecuación

$$w^T x_i + b = 0 \quad (\text{A.11})$$

donde w es un vector ortogonal al hiperplano solución y b es el coeficiente de intersección. El hiperplano solución se obtiene como el hiperplano medio a otros dos hiperplanos h^+ y h^- , los cuales contienen los vectores de soporte de ambas clases y cumplen que

$$\begin{aligned} h^+ &\rightarrow w^T x_i + b = +1 \\ h^- &\rightarrow w^T x_i + b = -1 \end{aligned} \quad (\text{A.12})$$

siendo “+1” la etiqueta para los ejemplos positivos y “-1” la etiqueta para los ejemplos negativos. Esto puede ser apreciado de forma gráfica en la Figura A.6.

Así, de las ecuaciones (A.11) y (A.12) se puede inferir la condición de clasificación, que queda definida como

$$y_i(x^T x_i + b) \geq 1 \quad (\text{A.13})$$

donde la variable y_i corresponde a las etiquetas antes mencionadas. De esta forma, la condición de clasificación implica que todas las muestras clasificadas correctamente tiene que pertenecer a la región situada más allá del margen.

El margen se puede calcular a partir de la distancia de los dos hiperplanos h^+ y h^- , según la ecuación

$$m = d^+ + d^- = 2 \frac{1}{\|w\|} = \frac{2}{\|w\|} \quad (\text{A.14})$$

siendo d^+ y d^- las distancias a los hiperplanos h^+ y h^- respectivamente

$$d^+ = d^- = \frac{|wx + b|}{\|w\|} = \frac{1}{\|w\|}$$

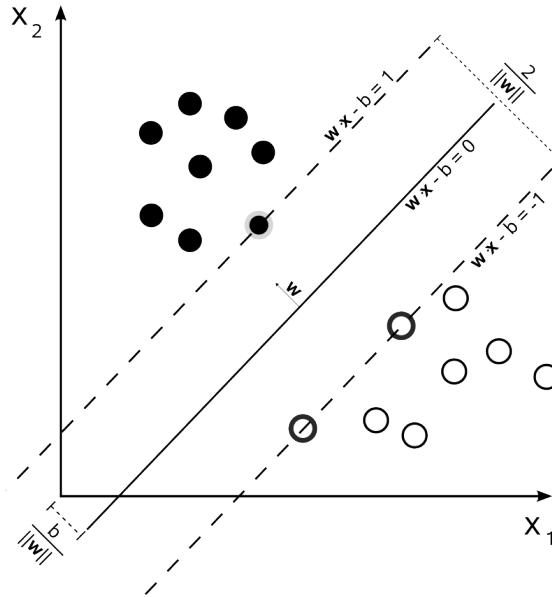


Figura A.6: Hiperplanos h^+ , h^- y solución con sus respectivas ecuaciones, mencionadas en A.11 y A.12.

Fuente: Cyc (Trabajo propio) [Dominio Público], Wikimedia Commons https://commons.wikimedia.org/wiki/File:Svm_max_sep_hyperplane_with_margin.svg.

Una forma de resolver el problema de maximización del margen es minimizar el problema inverso, es decir

$$\begin{aligned} \min_{w,b} \Phi(w) &= \frac{1}{2} \|w\|^2, \\ y_i(w^T x_i + b) &\geq 1 \end{aligned} \tag{A.15}$$

en donde $\|w\|^2$ es el resultado del producto escalar entre w^T y w .

Así, el problema de clasificación se ha transformado ahora en un problema de optimización cuadrática estándar. La resolución de este tipo de problemas se plantea a partir de una función auxiliar L , denominada *Lagrangiano*,

$$L(x, \alpha_i) = f(x) + \sum_i \alpha_i g_i(x) \tag{A.16}$$

en donde $f(x)$ representa la función que se quiere optimizar, $g_i(x)$ son las restricciones a las que se encuentra sujeto el problema y α son factores de multiplicaciones conocidos como *multiplicadores de Lagrange*. De esta manera, la solución al problema de optimización (denominada *SVM primal*) se plantea como

$$\max_{\alpha} (\min_{w,b} (L(w, b, \alpha))) \tag{A.17}$$

Para este caso se cumple

$$\begin{aligned} f(x) &\rightarrow \Phi(w) \\ g_i(x) &\rightarrow y_i(w^T x_i + b) - 1 \geq 0 \end{aligned}$$

por lo que, sustituyendo en (A.16) se obtiene

$$L(w, b, \alpha_i) = \frac{1}{2} w^T w - \sum_i \alpha_i [y_i(w^T x_i + b) - 1] \quad (\text{A.18})$$

Minimizar L implicará entonces igualar a cero las derivadas parciales de L respecto a w y a b , obteniendo de esta manera

$$\begin{aligned} \frac{dL}{dw} &= w - \sum_i \alpha_i y_i x_i = 0 \Rightarrow w = \sum_i \alpha_i y_i x_i \\ \frac{dL}{db} &= -\sum_i \alpha_i y_i = 0 \Rightarrow \sum_i \alpha_i y_i = 0, \text{ donde } \alpha_i \geq 0 \end{aligned} \quad (\text{A.19})$$

Sustituyendo en (A.18) y desarrollo matemático de por medio, se obtiene el lagrangiano L expresado como

$$L(w, b, \alpha_i) = -\frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i = \Theta(\alpha) \quad (\text{A.20})$$

obteniendo una nueva función Θ , para la cual se buscará hallar el máximo.

Esto implica que se obtiene un nuevo problema de optimización, el cual se denomina *SVM dual*

$$\begin{aligned} \max_{\alpha} \Theta(\alpha) &= -\frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i, \\ \text{sujeto a } &\sum_i \alpha_i y_i = 0, \text{ donde } \alpha_i \geq 0 \end{aligned}$$

De (A.19) se desprende que algunos multiplicadores de Lagrange tomarán valores iguales a cero. Así, los α_i que sean mayores a cero definirán cuales son las muestras que constituyen los vectores de soporte.

La formulación dual del problema ayuda a resolver el caso en donde los datos no son linealmente separables.

Kernel trick

Como se mencionó anteriormente, una manera de resolver el problema cuando el espacio de datos no es linealmente separable consiste en transformar el conjunto de datos de forma de obtener uno linealmente separable. Esto se logra al llevar los datos a un espacio de dimensión mayor, como se muestra en la Figura A.7. A esta técnica se la denomina *Kernel trick*.

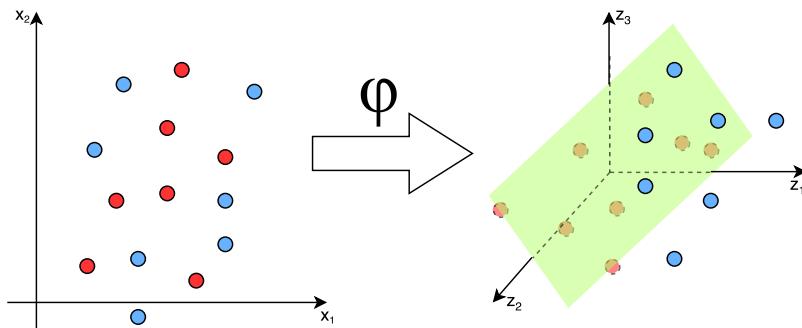


Figura A.7: Técnica kernel trick.

En primer lugar se define una función de mapeo φ y la función del kernel K , tales que

$$\begin{aligned} x &\mapsto \varphi(x) \\ K(x, z) &= \varphi(x)^T \varphi(z) \end{aligned}$$

Este kernel se integra fácilmente a la formulación dual del problema, lo que resulta en una nueva forma de plantearlo

$$\begin{aligned} \max_{\alpha} \Theta(\alpha) &= -\frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i, x_j) - \sum_i \alpha_i \\ b &= y_i - w^T \varphi(x_i) = y_i - \sum_j y_j \alpha_j K(x_j, x_i) \\ f(x) &= \operatorname{sgn}\left(\sum_i y_i \alpha_i K(x, x_i) + b\right) \end{aligned} \quad (\text{A.21})$$

Una ventaja de esta aproximación es que no existe la necesidad de definir un nuevo kernel para cada problema, ya que existen diferentes tipos de kernel estándar (polinomial, sigmoide, kernel de intersección, etc.) que resultan bastante útiles en la práctica.

Suavización del margen

La otra aproximación mencionada para la resolución del problema cuando el conjunto de datos no es linealmente separable es la de *Suavización del margen*. En este caso, se agrega una tolerancia a errores de clasificación. Para ello se define un nuevo conjunto de variables denominado *variables de holgura* (ξ), tal que

$$\xi_i \geq 0$$

Las variables de holgura se incorporan a la formulación del problema, relajando la condición de clasificación, por lo que la formulación del SVM primal toma ahora la forma

$$\begin{aligned} \min_{w, b, \Xi} \Theta(w) &= \frac{1}{2} w^T w + C \sum_i \xi_i \\ \text{sujeto a } y_i(w^T \Phi(x_i) + b) &\geq 1 - \xi_i \end{aligned} \quad (\text{A.22})$$

en donde el parámetro C se toma como parámetro de regularización y permite controlar el equilibrio entre la maximización del margen y la minimización del error.

En cambio, para la formulación dual del SVM, el agregado de las variables de holgura hace necesario incluir una inecuación, la cual va a modular la contribución de los vectores con valor α distinto de cero. Así, la formulación del problema dual queda representada

por

$$\begin{aligned}
 \max_{\alpha} \Theta(\alpha) &= -\frac{1}{2} \sum_i \sum_j \alpha_i \alpha_j y_i y_j K(x_i x_j) + \sum_i \alpha_i \\
 \text{sujeto a } &\begin{cases} 0 \leq \alpha_i \leq C \\ \sum_i \alpha_i y_i = 0 \end{cases} \\
 b &= y_i(1 - \xi_i) - \sum_j y_j \alpha_i K(x_j, x_i) \\
 f(x) &= \operatorname{sgn}\left(\sum_i y_i \alpha_i K(x, x_i) + b\right)
 \end{aligned} \tag{A.23}$$

A.4.3. Non-Maximum Suppression

En el procesamiento de imágenes, suele ser un problema común la aparición de múltiples detecciones que resultan redundantes por tratarse de lo mismo.



Figura A.8: Ejemplo de aplicación del algoritmo Non-Maximum Suppression sobre una imagen.

Fuente: Basada en Robbie Sproule, Flickr.com https://commons.wikimedia.org/wiki/File:Private_factory_guard.jpg.

Non-maximum Suppression (NMS) es un algoritmo de post-procesamiento clave en muchas aplicaciones en el campo de la visión por computadora. En el contexto de la detección de objetos suele ser utilizado para pasar de una cierta cantidad de detecciones superpuestas o muy cercanas a una única que represente la unión de las anteriores [95].

Como mencionan Neubeck y Van Gool [96], *Non-Maximum Suppression* se basa en la búsqueda de máximos locales en vecindarios. Un máximo local es aquel que es más grande que todos sus vecinos. Un vecindario es una región alrededor del píxel que se está considerando. Puede suceder que el mismo valor sea encontrado más de una vez en la imagen, por lo que es necesario decidir cual píxel debe ser eliminado. En la práctica, en caso de empates se eliminan todos (o todos excepto uno) de acuerdo a cierto orden predefinido.

A.5. Histogramas

Un histograma es una representación gráfica de una variable en forma de barras. Cada barra representa una categoría o clase, siendo la superficie de cada barra proporcional a la frecuencia de los valores representados. Los histogramas son una herramienta que ayuda a la toma de decisiones en la resolución de problemas, debido a que mediante el histograma se pueden identificar las pautas de comportamiento del conjunto de datos y extraer conclusiones sobre el mismo.

En este trabajo se utilizaron histogramas bi-variados (o histogramas en tres dimensiones) para representar la distribución que presentan las dimensiones de los *blobs* (alto y ancho) que fueron identificados como contenedores de objetos de interés. Los histogramas bi-variados pueden ser considerados como la conjunción de dos histogramas simples de manera que se puede estudiar al mismo tiempo la frecuencia de la co-ocurrencia de valores en dos variables. En este caso se habla de la conjunción de histogramas para el alto y el ancho de los *blobs* en los cuales se detectaron objetos de interés.

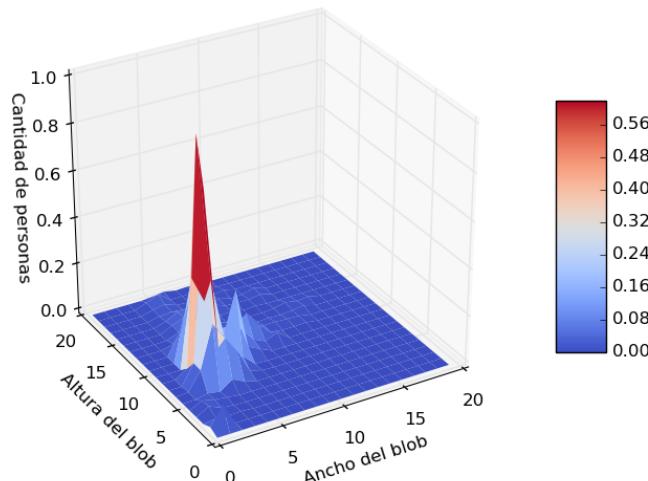


Figura A.9: Ejemplo de representación gráfca de histograma bi-variado. Modelo generado por el sistema procesando vídeo PETSc2007.

A.6. Seguimiento

Una vez detectados los *blobs*, filtrados e identificados los objetos de interés que contienen se procede a calcular su posición en el tiempo, a esta última etapa se le llama seguimiento, o también conocido como *tracking* en inglés.

Para poder efectuar el seguimiento de los objetos de interés se utilizan técnicas de predicción de posición combinadas con tecnicas de optimización de asignación. En el caso de la predicción se utilizaron Filtros de Kalman (*Kalman filters*) [36] y para la optimización de asignaciones el Algoritmo húngaro, también conocido como algoritmo de Kuhn-Munkres [97].

En síntesis, la predicción de posición consiste en, dado un historial reciente de posiciones en tiempo menor igual al tiempo t de un objeto de interés, calcular la posición más probable para el tiempo $t+1$. El problema de asignación consiste en, dadas las posiciones de los objetos de interés en un tiempo t y un conjunto de *blobs* que contienen objetos de interés, en el tiempo $t+1$, se calcula una asociación uno a uno entre los objetos de interés del tiempo t y los blobs del tiempo $t+1$.

A.6.1. Filtros de Kalman

El modelo de Filtros de Kalman, desarrollado por Kalman en el año 1960, apunta a resolver el problema general de estimar el estado x , de un proceso de tiempo discreto, asumiendo que el estado de un sistema en el tiempo t evoluciona desde el estado anterior en el tiempo $t-1$ de acuerdo a la ecuación

$$x_t = F_t x_{t-1} + B_t u_t + w_t, \quad (\text{A.24})$$

en donde:

- x_t es el vector de estado que contiene la información de interés para el sistema (posición, velocidad, etc.) en el tiempo t
- u_t es el vector que contiene las entradas de control, si estas existen (por ejemplo, fuerza)
- F_t es la matriz de transición de estado (o modelo), la cual es utilizada para calcular el efecto que producen los valores de los parámetros en el tiempo $t-1$ sobre el sistema en el tiempo t
- B_t es la matriz de entrada, la cual es utilizada para calcular el efecto que produce cada uno de los parámetros de control del vector u_t sobre el vector de estado
- w_t es el vector que contiene los valores de ruido del proceso, para cada uno de los parámetros del vector de estado. Se asume que el mismo sigue una distribución normal, cuya covarianza viene dada por la matriz de covarianza del proceso Q_t

A su vez, las medidas del sistema también pueden ser representadas según la ecuación

$$z_t = H_t x_t + v_t \quad (\text{A.25})$$

en donde:

- z_t representa al vector de medidas
- H_t es la matriz de transformación, la cual permite traducir los valores del vector de estado al dominio de medidas
- v_t representa al vector que contiene el ruido en las medidas observadas en el vector de medidas. Se asume (al igual que para w_t) que es ruido gaussiano¹ y que su matriz de covarianza es R_t

¹El ruido gaussiano es aquel tipo de ruido en el cual todos los píxeles varían sus valores siguiendo una distribución gaussiana

Así, definiendo una matriz de transición que modele el movimiento de una persona en vídeo, mediante la posición, velocidad y aceleración, éste algoritmo permite predecir la posición de la persona en el siguiente *frame*. Dado que la predicción no es perfecta, es necesario aplicar correcciones cada vez que se toma una nueva medición. En el caso de un vídeo, cada vez que se obtiene un nuevo frame. De esta manera se consigue medir la magnitud del error [36].

A.6.2. Problema de asignación

El *Problema de asignación* consiste en encontrar una asignación óptima de un conjunto de recursos a un conjunto de tareas [98]. El problema debe cumplir las siguientes propiedades para poder ser formulado como un *Problema de asignación*:

1. La cantidad de recursos es la misma que la cantidad de tareas
2. Cada recurso es asignado a una única tarea
3. Cada tarea es realizada utilizando un único recurso
4. Existe una matriz C_{ij} que representa los costos de asignar el recurso i a la tarea j
5. El objetivo es encontrar una combinación de asignaciones que minimice los costos totales

La resolución de este tipo de problemas se basa en el *Teorema fundamental de la asignación*, el cual dice que si a todos los elementos de una fila o columna de una determinada matriz de rendimientos, se le resta o se le suma una cantidad constante, la asignación óptima no varía.

El modelo matemático utiliza variables binarias x_{ij} que representan la asignación de un recurso a una tarea, según

$$x_{ij} = \begin{cases} 1, & \text{si el recurso } i \text{ es asignado a la tarea } j \\ 0, & \text{en caso contrario} \end{cases} \quad \text{para } \begin{cases} i = 1, 2, \dots, n \\ j = 1, 2, \dots, n \end{cases}$$

El costo total de la asignación se obtiene a partir de la suma de los productos de cada variable x_{ij} por el costo asignado C_{ij} , según se indica en (A.26)

$$Z_{\min} = \sum_{i=1}^n \sum_{j=1}^n C_{ij} x_{ij} \tag{A.26}$$

Las propiedades 2 y 3 quedan representadas mediante

$$\begin{aligned} \sum_{j=1}^n x_{ij} &= 1, \text{ para } i = 1, 2, \dots, n \\ \sum_{i=1}^n x_{ij} &= 1, \text{ para } j = 1, 2, \dots, n \end{aligned} \tag{A.27}$$

Así, el modelo completo de asignación se obtiene de añadir las restricciones de no negatividad y la de variables binarias, obteniendo

$$Z_{\min} = \sum_{i=1}^n \sum_{j=1}^n C_{ij} x_{ij}, \text{ sujeto a } \begin{cases} x_{ij} \text{ binarias, } \forall i, j \\ x_{ij} \geq 0 \\ \sum_{j=1}^n x_{ij} = 1, \text{ para } i = 1, 2, \dots, n \\ \sum_{i=1}^n x_{ij} = 1, \text{ para } j = 1, 2, \dots, n \end{cases} \quad (\text{A.28})$$

Un algoritmo que resuelve este problema es el Algoritmo Húngaro. Este algoritmo fue utilizado en la etapa de seguimiento del sistema y se presenta en detalle en la siguiente sección.

A.6.3. Algoritmo Húngaro

El *Algoritmo húngaro*, también conocido como *algoritmo de asignación de Munkres* o *algoritmo de Kuhn-Munkres*, es un algoritmo de optimización que resuelve problemas de asignación en tiempo $O(n^3)$. El problema que resuelve es análogo a resolver la asignación de un conjunto de trabajadores a un conjunto de tareas, en donde todas las posibles combinaciones (*trabajador, tarea*) son conocidas y tienen asignadas un valor. Este valor determina el costo de asignar esa tarea a ese trabajador. La resolución consiste en encontrar las parejas (*trabajador, tarea*) tal que se minimice el costo total.

El algoritmo fue desarrollado por Kuhn en 1955 basándose fundamentalmente en los primeros trabajos de otros dos matemáticos húngaros: Dénes König y Jenő Egerváry [69]. Luego, en el año 1957 fue revisado por Munkres [97].

El algoritmo recibe como entrada una matriz $C_{n \times n}$, denominada matriz de costos, la cual indica el costo $c_{i,j}$ de asignar la tarea i al trabajador j . Una *asignación* es un conjunto de n elementos que representan posiciones en la matriz de costos. La suma de las n entradas de una asignación es denominada *costo*. Una asignación con el menor costo posible es denominada *asignación óptima*. Se puede decir que el Algoritmo Húngaro permite encontrar una asignación óptima para una matriz de costos dada.

Método general

El Algoritmo Húngaro consste en aplicar una los siguientes cinco pasos [99]:

1. Construir una tabla de tamaño $n + 1 \times n + 1$. La primer columna se utiliza para etiquetar los recursos candidatos a asignar mientras que la primer fila se utiliza para etiquetar las tareas. En las intersecciones quedan representados los costos de asignación asociados.
2. Para cada fila se identifica el menor costo y se le resta a los costos de la misma fila.
3. Para la matriz obtenida del paso anterior, sobre cada columna se identifica el menor costo y se resta a los costos de la misma columna. La matriz resultante de ejecutar este paso se denomina *Matriz de costos reducida*.
4. Se trazan líneas horizontales o verticales (o ambas) en busca de cubrir todos los ceros de la matriz de costos reducidos, utilizando el menor número de líneas posible.

Si el número de líneas es igual a la cantidad de filas o columnas de la matriz, entonces se logra obtener la solución óptima buscada. En caso que el número de líneas sea menor que la cantidad de filas o columnas de la matriz, se debe proceder con el siguiente paso.

5. Se busca el menor elemento entre aquellos valores que no fueron cubiertos por las líneas del paso anterior. Se resta ese valor a los elementos que no se encuentran cubiertos y se suma ese mismo valor a los elementos que se encuentran en las intersecciones de las líneas horizontales y verticales. Una vez finalizado este paso se retorna al paso anterior.

Bibliografía

- [1] D. L. República. Bonomi anunció que se instalarán 2.000 cámaras más en montevideo, 2016. URL <http://www.republica.com.uy/bonomi-anuncio-se-instalaran-2-000-camaras-mas-montevideo/580698/>. Último acceso 13-11-2016.
- [2] D. E. País. El corrimiento de delincuentes obliga a más cámaras de vigilancia, 2014. URL <http://www.elpais.com.uy/informacion/corrimiento-delincuentes-obliga-mas-camaras.html>. Último acceso 13-11-2016.
- [3] L. R. 21. Ministerio del interior instalará más de 1.000 cámaras de video-vigilancia en principales puntos de canelones, 2016. URL <http://www.monografias.com/trabajos57/fatiga-aburrimiento/fatiga-aburrimiento.shtml>. Último acceso 14-11-2016.
- [4] J. Choque. Fatiga y aburrimiento, 2008. URL <http://www.monografias.com/trabajos57/fatiga-aburrimiento/fatiga-aburrimiento.shtml>. Último acceso 14-11-2016.
- [5] A. Webb. *Statistical pattern recognition*. John Wiley & Sons, 2003.
- [6] A. de la Escalera Hueso. Visión por computador. *Fundamentos y métodos.*: Prentice Hall, 2001.
- [7] B. Ramírez. Procesamiento digital de imágenes. Agosto 2006.
- [8] J. Muñoz Pérez. Segmentación de imágenes. 2009. Último acceso 09-09-2015.
- [9] M. Martín. Técnicas clásicas de segmentación de imagen. *Inspección Visual Automática*, 2002.
- [10] R. Gonzalez y R. Woods. Digital image processing. Nueva Jersey, 2008.
- [11] J. Muñoz Pérez. Representación de formas y descripción. http://www.lcc.uma.es/~munozp/documentos/procesamiento_de_imagenes/temas/pi_cap8.pdf, 2009. Último acceso 09-09-2015.
- [12] J. Proakis y D. Manolakis. *Tratamiento digital de señales*. 1998.
- [13] S.-Y. Ho y Y.-C. Chen. An efficient evolutionary algorithm for accurate polygonal approximation. *Pattern Recognition*, 34(12):2305–2317, 2001.

- [14] E. Santillán y C. Cruz. Detección y clasificación de objetos dentro de un salón de clases empleando técnicas de procesamiento digital de imágenes. *Universidad Autónoma Metropolitana, México, Tesis de Maestría*, 2008.
- [15] A. Moujahid, I. Inza, y P. Larrañaga. Tema 5. clasificadores K-NN.
- [16] A. K. Jain, R. P. W. Duin, y J. Mao. Statistical pattern recognition: A review. *IEEE Transactions on pattern analysis and machine intelligence*, 22(1):4–37, 2000.
- [17] D. Ramírez Osuna. *Desarrollo de un método de procesamiento de imágenes para la discriminación de maleza en cultivos de maíz*. PhD thesis, 2013.
- [18] T. Cover y J. Van Campenhout. On the possible orderings in the measurement selection problem. *IEEE Transactions on Systems, Man, and Cybernetics*, 7(9):657–661, 1977.
- [19] C. Corso. Aplicación de algoritmos de clasificación supervisada usando Weka. *Córdoba: Universidad Tecnológica Nacional, Facultad Regional Córdoba*, 2009.
- [20] J. Carrasco y J. Martínez. Reconocimiento de patrones. *Komputer Sapiens*, 2(3), 2011.
- [21] E. Calot. *Reconocimiento de patrones en imágenes médicas basado en sistemas inteligentes*. Enrique Calot, 2008.
- [22] W. Strauss, B. Zaret, P. Hurley, T. Natarajan, y B. Pitt. A scintiphographic method for measuring left ventricular ejection fraction in man without cardiac catheterization. *The American journal of cardiology*, 28(5):575–580, 1971.
- [23] D. Ballard y J. Sklansky. Tumor detection in radiographs. *Computers and Biomedical Research*, 6(4):299–321, 1973.
- [24] H.-P. Chan, K. Doi, S. Galhotra, C. Vyborny, H. MacMahon, y P. Jokich. Image feature analysis and computer-aided diagnosis in digital radiography. i. automated detection of microcalcifications in mammography. *Medical physics*, 14(4):538–548, 1987.
- [25] J. Duncan y N. Ayache. Medical image analysis: Progress over two decades and the challenges ahead. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):85–106, 2000.
- [26] R. Uppaluri, E. Hoffman, M. Sonka, P. Hartley, G. Hunnighake, y G. McLennan. Computer recognition of regional lung disease patterns. *American Journal of Respiratory and Critical Care Medicine*, 160(2):648–654, 1999.
- [27] C. Prieto, J. Febres, M. Cerrolaza, y R. Miquelarena. Sistema de Visión Artificial para el Control de Movimiento de un Asistente Robótico Médico. *Mecánica Computacional*, XXIX(66):6619–6629.
- [28] H. Liu, S. Chen, y N. Kubota. Intelligent video systems and analytics: A survey. *IEEE Transactions on Industrial Informatics*, 9(3):1222–1233, 2013.

- [29] A. F. Abate, M. Nappi, D. Riccio, y G. Sabatino. 2d and 3d face recognition: A survey. *Pattern Recognition Letters*, 28(14):1885–1906, 2007.
- [30] L. Calavia. Caracterización semántica de espacios: Sistema de Videovigilancia Inteligente en Smart Cities. 2013.
- [31] C. Norris y G. Armstrong. *The maximum surveillance society: The rise of CCTV*. Berg Publishers, 1999.
- [32] M. Valera y S. Velastin. Intelligent distributed surveillance systems: a review. In *Vision, Image and Signal Processing, IEE Proceedings-*, volume 152, pages 192–204. IET, 2005.
- [33] M. Piccardi. Background subtraction techniques: a review. In *Systems, man and cybernetics, 2004 IEEE international conference on*, volume 4, pages 3099–3104. IEEE, 2004.
- [34] H. López. Detección y seguimiento de objetos con cámaras en movimiento. *Proyecto fin de carrera, Universidad Autónoma de Madrid*, 2011.
- [35] C. Stauffer y E. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on*, volume 2. IEEE, 1999.
- [36] R. Kalman. A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, 82(1):35–45, 1960.
- [37] D. Lefloch. Real-time people counting system using video camera. *Department of Computer Science and Media Technology, Gjøvik University College, Norway*, 2007.
- [38] M. Rodriguez, I. Laptev, J. Sivic, y J.-Y. Audibert. Density-aware person detection and tracking in crowds. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2423–2430. IEEE, 2011.
- [39] P. Dollar, C. Wojek, B. Schiele, y P. Perona. Pedestrian detection: An evaluation of the state of the art. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 34(4):743–761, 2012.
- [40] M. J. Leach, E. Sparks, y N. Robertson. Contextual anomaly detection in crowded surveillance scenes. *Pattern Recognition Letters*, 44:71–79, 2014.
- [41] P. Felzenszwalb, R. Girshick, D. McAllester, y D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE transactions on pattern analysis and machine intelligence*, 32(9):1627–1645, 2010.
- [42] Z. Kalal, J. Matas, y K. Mikolajczyk. Online learning of robust object detectors during unstable tracking. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1417–1424. IEEE, 2009.
- [43] S.-H. Cho y H.-B. Kang. Abnormal behavior detection using hybrid agents in crowded scenes. *Pattern Recognition Letters*, 44:64–70, 2014.

- [44] X. Zhu, X. Jin, X. Zhang, C. Li, F. He, y L. Wang. Context-aware local abnormality detection in crowded scene. *Science China Information Sciences*, 58(5):1–11, 2015.
- [45] D. Garlan y M. Shaw. An introduction to software architecture. *Advances in software engineering and knowledge engineering*, 1(3.4), 1993.
- [46] J. Van Huis, H. Bouma, J. Baan, G. Burghouts, P. Eendebak, R. den Hollander, J. Dijk, y J. van Rest. Track-based event recognition in a realistic crowded environment. In *SPIE Security+ Defence*, pages 92530E–92530E. International Society for Optics and Photonics, 2014.
- [47] S. Mallick. OpenCV (C++ vs Python) vs MATLAB for Computer Vision. <http://www.learnopencv.com/openCV-c-vs-python-vs-matlab-for-computer-vision/>, 2015. Último acceso 25-05-2016.
- [48] L. Coelho. Why Python is Better than Matlab for Scientific Software. <https://metarabbit.wordpress.com/2013/10/18/why-python-is-better-than-2Dmatlab-for-scientific-software/>, 2013. Último acceso 31-05-2016.
- [49] D. Hull. Top 10 MATLAB code practices that make me cry. <http://blogs.mathworks.com/videos/2010/03/08/top-10-matlab-code-practices-that-make-me-cry/>, 2010. Último acceso 26-07-2016.
- [50] Mathworks. Strategies for Efficient Use of Memory. http://www.mathworks.com/help/matlab/matlab_prog/strategies-for-efficient-use-of-memory.html, . Último acceso 26-07-2016.
- [51] Mathworks. Techniques to Improve Performance. http://www.mathworks.com/help/matlab/matlab_prog/techniques-for-improving-performance.html, . Último acceso 26-07-2016.
- [52] S. McGarrity. Programming patterns: maximizing code performance by optimizing memory access, 2007.
- [53] B. Thorne, R. Grasset, y R. Green. Using Python in Computer Vision: Performance and Usability. 2009.
- [54] T. Bray. The javascript object notation (json) data interchange format. Internet Requests for Comments, March 2014. ISSN 2070-1721. URL <http://www.rfc-editor.org/rfc/rfc7159.txt>.
- [55] G. van Rossum, B. Warsaw, y N. Coghlan. PEP 8-style guide for python code. *Python.org*, 2001.
- [56] A. Rosebrock. Install OpenCV 3.0 and Python 3.4+ on Ubuntu. <http://www.pyimagesearch.com/2015/07/20/install-opencv-3-0-and-python-3-4-on-2Dubuntu/>, 2015. Último acceso 13-06-2016.
- [57] Z. Zivkovic. Improved adaptive Gaussian mixture model for background subtraction. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 2, pages 28–31. IEEE, 2004.

- [58] Z. Zivkovic y F. van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, 27(7):773–780, 2006.
- [59] OpenCV dev team. Smoothing Images - Gaussian Blur. http://docs.opencv.org/2.4/doc/tutorials/imgproc/gaussian_median.blur_bilateral_filter/gaussian_median.blur_bilateral.filter.html#gaussian-filter, 2014. Último acceso 01-07-2016.
- [60] OpenCV dev team. How to Use Background Subtraction Methods. http://docs.opencv.org/3.1.0/d1/dc5/tutorial_background_subtraction.html#gsc.tab=0, 2015. Último acceso 01-07-2016.
- [61] OpenCV dev team. Morphological Transformations. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html, . Último acceso 01-07-2016.
- [62] OpenCV dev team. Object Detection. http://docs.opencv.org/2.4/modules/gpu/doc/object_detection.html, . Último acceso 11-05-2016.
- [63] N. Dalal y B. Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.
- [64] A. Mohan, C. Papageorgiou, y T. Poggio. Example-based object detection in images by components. *IEEE transactions on pattern analysis and machine intelligence*, 23(4):349–361, 2001.
- [65] A. Rosebrock. Pedestrian Detection OpenCV. <http://www.pyimagesearch.com/2015/11/09/pedestrian-detection-opencv/>, November 2015. Último acceso 09-05-2016.
- [66] A. Rosebrock. Non-Maximum Suppression for Object Detection in Python. <http://www.pyimagesearch.com/2014/11/17/non-maximum-suppression-object-2ddetection-python/>, November 2014. Último acceso 24-05-2016.
- [67] Python Software Foundation. GIL. <https://docs.python.org/3.4/library/multiprocessing.html>, 2016. Último acceso 11-07-2016.
- [68] Python Software Foundation. multiprocessing — Process-based parallelism. <https://docs.python.org/3.4/glossary.html#term-global-interpreter-lock>, 2016. Último acceso 11-07-2016.
- [69] H. Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.
- [70] B. Clapper. Munkres implementation for Python. <http://software.clapper.org/munkres/index.html>, 2010. Último acceso 23-08-2016.
- [71] The SciPy community. SciPy. <http://docs.scipy.org/doc/scipy/reference/index.html>, 2016. Último acceso 23-08-2016.

- [72] R. Labbe Jr. FilterPy - Kalman filters and other optimal and non-optimal estimation filters in Python. <https://github.com/rlabbe/filterpy>, 2015. Último acceso 23-08-2016.
- [73] Y. Shafranovich. Common format and MIME type for comma-separated values (CSV) files. 2005.
- [74] J. Honovich. Average Frame Rate Video Surveillance 2011, 2011. Último acceso 12-11-2016.
- [75] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, y K. Schindler. MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking. *arXiv:1504.01942 [cs]*, April 2015. URL <http://arxiv.org/abs/1504.01942>. arXiv: 1504.01942.
- [76] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, y K. Schindler. MOT Challenge: Multiple Object Tracking Benchmark, 2015. URL <https://motchallenge.net/>. Último acceso 24-10-2016.
- [77] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, y I. H. Witten. The WEKA data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18, 2009.
- [78] J. O’Hara. Toward a commodity enterprise middleware. *Queue*, 5(4):48–55, 2007.
- [79] Pivotal. High-level Overview of AMQP 0-9-1 and the AMQP Model. <https://www.rabbitmq.com/tutorials/amqp-concepts.html>. Último acceso 06-06-2016.
- [80] A. Mordvintsev y K. Abid Rahman. Background Subtraction. http://opencv-python-tutorials.readthedocs.org/en/latest/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html, 2013. Último acceso 24-11-2015.
- [81] Á. Juan, M. Sedano, y A. Vila. La distribución Normal. http://www.uoc.edu/in3/emath/docs/Distrib_Normal.pdf, 2015. Último acceso 27-06-2016.
- [82] OpenCV dev team. Blob Detection Using OpenCV (Python, C++). http://docs.opencv.org/2.4/modules/features2d/doc/common_interfaces_of_feature_detectors.html, 2015. Último acceso 24-11-2015.
- [83] S. Rossius. *Reconocimiento de objetos mediante WebCam en tiempo real*. PhD thesis, 2013.
- [84] E. Valveny. Detector de peatones basado en HOG + SVM. <https://www.coursera.org/learn/deteccion-objetos/lecture/AMcD1/14-1-detector-de-2Dpeatones-basado-en-hog-svm>, 2016. Último acceso 15-06-2016.
- [85] A. García López. Reconocimiento de objetos inmersos en imágenes estáticas mediante el algoritmo HOG y RNA-MLP. 2015.
- [86] R. Alzate. Reconocimiento de especies arbóreas usando procesamiento digital de imágenes. <https://www.overleaf.com/articles/reconocimiento-de-2Despecies-arboreas-usando-pdi/xzsyhssxdnqs/viewer.pdf>, May 2015. Último acceso 17-05-2016.

- [87] E. Valveny. HOG - Cálculo del gradiente. <https://www.coursera.org/learn/deteccion-objetos/lecture/WSJ9t/14-2-hog-calculo-del-gradiente>, 2016. Último acceso 15-06-2016.
- [88] E. Valveny. HOG - Cálculo del descriptor. <https://www.coursera.org/learn/deteccion-objetos/lecture/uAEKB/14-4-hog-calculo-del-descriptor>, 2016. Último acceso 15-06-2016.
- [89] E. Valveny. HOG - Cálculo de los histogramas. <https://www.coursera.org/learn/deteccion-objetos/lecture/GBJYS/14-3-hog-calculo-de-los-histogramas>, 2016. Último acceso 15-06-2016.
- [90] B. Boser, I. Guyon, y V. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, pages 144–152. ACM, 1992.
- [91] A. Kowalczyk. SVM Tutorial. <http://www.svm-tutorial.com/>, 2014. Último acceso 20-05-2016.
- [92] E. Valveny y F. Vilariño. Support Vector Machines - Conceptos básicos. <https://www.coursera.org/learn/deteccion-objetos/lecture/iGdzT/14-5-support-vector-machines-svm-conceptos-basicos>, 2016. Último acceso 17-06-2016.
- [93] R. Berwick. An idiot's guide to Support Vector Machines (SVMs). *Retrieved on October, 21:2011*, 2003.
- [94] E. Valveny y F. Vilariño. Support Vector Machines - Desarrollo matemático. <https://www.coursera.org/learn/deteccion-objetos/lecture/JXyER/14-6-support-vector-machines-svm-desarrollo-matematico>, 2016. Último acceso 17-06-2016.
- [95] R. Rothe, M. Guillaumin, y L. Van Gool. *Non-maximum suppression for object detection by passing messages between windows*. Springer, 2014.
- [96] A. Neubeck y L. Van Gool. Efficient non-maximum suppression. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, volume 3, pages 850–855. IEEE, 2006.
- [97] J. Munkres. Algorithms for the assignment and transportation problems. *Journal of the Society for Industrial and Applied Mathematics*, 5(1):32–38, 1957.
- [98] F. Correa Florez. Unidad 8 - Modelo de asignación - Investigación de operaciones. <http://docplayer.es/2818018-Unidad-8-modelo-de-asignacion%2Dcaracteristicas-de-asignacion-metodo-hungaro-o-de-matriz-reducida.html>, 2015. Último acceso 24-06-2016.
- [99] B. Salazar López. Problemas de asignación. <http://www.ingenieriaindustrialonline.com/herramientas-para-el-ingenero%2Dindustrial/investigaci%C3%B3n-de-operaciones/problemas-de%2Dasignaci%C3%B3n/>, 2010. Último acceso 24-06-2016.