

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA

EVALUACIÓN DE MÉTODOS DE DETECCIÓN DE SILUETAS PARA CLASIFICACIÓN DE
ACCIONES BASADO EN MuHAVI UN CONJUNTO DE DATOS PARA RECONOCIMIENTO DE
ACCIONES HUMANAS

JORGE ANTONIO SEPÚLVEDA ORTEGA
16 de julio de 2014

UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA

EVALUACIÓN DE MÉTODOS DE DETECCIÓN DE SILUETAS PARA CLASIFICACIÓN DE
ACCIONES BASADO EN MUHAVI UN CONJUNTO DE DATOS PARA RECONOCIMIENTO DE
ACCIONES HUMANAS

Trabajo de Titulación presentado en conformidad a los requisitos para obtener el Título de
Ingeniero Civil en Informática

Profesor guía: DR. SERGIO A. VELASTÍN

JORGE ANTONIO SEPÚLVEDA ORTEGA
16 de julio de 2014

Agradecimientos

Quisiera agradecer en primer lugar a mi familia, especialmente a mi esposa por su comprensión, apoyo durante este camino. A mis hijos por la paciencia. Mencionar a mi hija del “medio” por su ayuda en la corrección ortográfica y la redacción de los primeros capítulos.

Agradecer también a mi profesor guía, Dr. Sergio A. Velastín por su disposición de aceptarme en primer lugar como su alumno tesista sin conocerme previamente. Por los consejos, opiniones, puntos de vistas que espero esten bien reflejados en este trabajo. También, por las amenas charlas que surgían en las reuniones de avances, más alla del trabajo de tesis.

Me gustaría mencionar en estos agradecimientos a los académicos del departamento de Ingeniería Informática. Por la dedicación y motivación del trabajo en clases. Sería injusto mencionar sólo algunos, y dejar a otros fuera. También agradecer a don Miguel Fuentes, coordinador del programa vespertino, lo más probable que él no recuerde, pero fue él la primera persona que me aconsejó para entrar al programa vespertino.

Muchas Gracias.

“Whenever you find yourself on the side of the majority, it is time to pause and reflect.”

Mark Twain

Resumen

Este trabajo de tesis coloca a disposición de la comunidad de investigación en visión por computador, una pareja de herramientas de software que facilita, la integración, ejecución y posterior evaluación de desempeño, de algoritmos que realizan separación de siluetas de su imagen de fondo (*Background Subtraction*). El sistema de software se divide en dos módulos principales; el primero crea una capa de abstracción (clases C++) para integrar los distintos algoritmos que desarrolla la comunidad de investigación, y proporciona un conjunto de interfaces que permiten una rápida y fácil integración con este sistema de software. Incorpora además, el algoritmo en estado de arte *SAGMM* (*Self-Adaptive Gaussian Mixture Model*) orientado a separación imágenes de fondo, basado en mixtura de componentes gaussiano *GMM* (*Gaussian Mixture Model*), y sirve de ejemplo para la integración de nuevos algoritmos en este sistema. El segundo módulo es un programa ejecutable que engloba un conjunto de métricas de evaluación de calidad, empleadas en los sistemas de clasificación y reconocimiento de patrones. Se detallan las mediciones estadísticas de rendimiento en clasificación binaria ‘*sensitividad*’ y ‘*especificidad*’, asimismo, la métrica *F-Measure* que es el promedio ponderado de *Precision* y *Recall*, o el coeficiente de correlación de Matthews *MCC*. De manera que el ser utilizadas en su conjunto proporcionan una idea general del desempeño total de un algoritmo. El programa de evaluación de desempeño, funciona en base a la comparación de resultados, necesita de entrada un conjunto de imágenes resultantes (máscaras de siluetas), generadas por el algoritmo que se intenta evaluar y su correspondiente imagen de referencia. Este trabajo hace un uso extensivo de curva de operaciones características *ROC*, para comparar rendimiento de distintos algoritmos y localizar un punto de operación óptimo de los algoritmos que se evalúan. Se realiza también la evaluación de un grupo de algoritmos basado en mixtura de componentes gaussianas, sobre el conjunto de datos MuHAVI, el resultado genera y publica el conjunto total de siluetas resultantes con el algoritmo estado de arte *SAGMM* incluido en el primer modulo de software. El trabajo concluye con el análisis de las métricas más relevantes usadas en la evaluación de los algoritmo.

Índice General

Agradecimientos	I
Resumen	III
Índice de Figuras	VII
Índice de Tablas	X
1. INTRODUCCIÓN	1
1.1. Antecedentes y motivación	1
1.2. Descripción del problema	4
1.3. Solución propuesta	4
1.4. Objetivos y alcances del proyecto	5
1.4.1. Objetivo general	5
1.4.2. Objetivos específicos	5
1.4.3. Alcances	6
1.5. Metodologías y herramientas utilizadas	6
1.6. Resultados obtenidos	7
1.7. Organización del documento	7
2. ESTADO DEL ARTE	9
2.1. Modelo de Mixtura de Gaussianas	10
2.2. Evaluación de rendimiento	13
2.3. Conjunto de datos	14
3. ALGORITMOS SUSTRACCIÓN DEL FONDO	17
3.1. Introducción	17

3.2.	Modelo General	17
3.3.	Mixtura de distribuciones Gaussianas - GMM	23
3.3.1.	Actualización parámetros del Algoritmo	24
3.4.	Modelo Mixtura de Gaussianas auto-adaptativo	25
3.4.1.	Factor de iluminación global	26
3.4.2.	Filtro espacial y temporal	27
3.5.	Modelo GMM incluido en Micro-Controladores	29
3.6.	Modelo No-Paramétrico de fondo de imagen	31
3.7.	Resumen	33
4.	MÉTRICAS DE EVALUACIÓN DE RENDIMIENTO	35
4.1.	Introducción	35
4.2.	Enfoques de evaluación	35
4.3.	Métodos de evaluación de calidad	36
4.3.1.	Error Cuadrático Medio y Relación a Señal Ruido	36
4.3.2.	Métricas de Calidad Estática	37
4.3.3.	Curva de operaciones característica	40
4.3.4.	Métricas de Percepción	41
4.3.5.	Similaridad Estructural	44
4.3.6.	D-Score	45
4.4.	Resumen de métodos de evaluación	46
5.	DESCRIPCIÓN DEL SISTEMA DE SOFTWARE	49
5.1.	Introducción	49
5.2.	Captura de requerimientos	50
5.3.	Arquitectura de sistema de software	51
5.4.	Herramientas de Soporte Base	54
5.4.1.	Herramientas de desarrollo	54
5.4.2.	Biblioteca OpenCV de Visión por Computador	54
5.4.3.	Biblioteca Boost	55
5.5.	Descripción de Implementación	56
5.5.1.	Sistema Base Integración Algoritmos de Visión por Computador	57
5.5.2.	Diagrama de clases de implementación	58
5.5.3.	Herramienta de evaluación de desempeño	61
5.5.4.	Operación de la implementación	62

5.6. Resumen	64
6. EXPERIMENTACIÓN	65
6.1. Introducción	65
6.2. Conjunto de datos de acciones humanas - MuHAVI	65
6.3. Procedimiento de Evaluación	67
6.4. Evaluación de Rendimiento	69
6.4.1. Comparación puntos de operación	73
6.4.2. Intervalo de confianza curva operaciones características	77
6.5. Resumen	80
7. CONCLUSIONES	87
A. PROCESAMIENTO DE UNA SECUENCIA	93
B. CURVAS DE OPERACIONES DESAGREGADAS POR SECUENCIA	95

Índice de Figuras

1.1.	Aplicaciones en visión por computador	2
1.2.	Diagrama de bloques etapas visión por computador	3
2.1.	KTH Dataset	15
2.2.	Weizmann Dataset	15
2.3.	IXMAS Dataset	16
2.4.	MuHAVI Dataset	16
3.1.	Imágenes 570, 610 de secuencia “ <i>Kick Camera 3 Person 4</i> ”	18
3.2.	Cambios de intensidad en el tiempo del valor de un pixel	19
3.3.	Histograma secuencia completa “ <i>Kick Camera 3 Person 4</i> ”	20
3.4.	Gráfico dispersión 3D secuencia completa “ <i>Kick Camera 3 Person 4</i> ”	21
3.5.	Gráfico dispersión 2D secuencia completa “ <i>Kick Camera 3 Person 4</i> ”	22
3.6.	Procesamiento temporal de tres imágenes	27
3.7.	Filtro temporal spacial	28
3.8.	Comparación de efectividad del filtro en una secuencia completa	29
3.9.	Aproximación de actualización de los pesos	30
3.10.	Representación contadores en 4 bytes	31
4.1.	Métricas de calidad estática	38
4.2.	Matriz de Confusión	39
4.3.	Curva de operaciones características	41
4.4.	Pesos pixeles falso positivo y negativo	43
4.5.	Pesos usando función tipo sigmoidal	44
4.6.	Valores D-Score basados en la distancia desde la referencia ground-truth	46
5.1.	Casos de uso de los requerimientos establecidos en el proyecto	51
5.2.	Arquitectura modular de software	53

5.3. Estructura de datos <i>OpenCV Mat</i>	56
5.4. Diagrama UML de patrones de diseño <i>Plantilla y Estrategia</i>	58
5.5. Diagrama UML de clases del sistema base	59
5.6. Diagrama de secuencia	61
5.7. Diagrama UML herramienta de evaluación	62
5.8. Diagrama en bloques de operación de ambos elementos, sistema base de algoritmos y herramienta de evaluación de desempeño	63
 6.1. Disposición de las 8 cámaras sobre el escenario	66
6.2. Ejemplo de acciones humanas con imágenes de referencias	68
6.3. Curvas de operaciones características obtenidas durante experimentación, usando dos valores diferentes de factor de aprendizaje	72
6.4. Mediciones de rendimiento global obtenidos para un factor de aprendizaje $\alpha = 0,001$	74
6.5. Comparación de cuadros con dos distintos valores de <i>threshold</i> algoritmo SAGMM.	75
6.6. Cuadros comparativos métricas de rendimiento de MCC, F-Measure, y D-Score ($\alpha = 0,001$ y $\alpha = 0,0002$).	81
6.7. Cuadros comparativos métricas de percepción PSNR y MSSIM ($\alpha = 0,001$ y $\alpha = 0,0002$).	82
6.8. Resultado siluetas comparación diferentes algoritmos.	83
6.9. Curva operaciones promedio junto con intervalo de confianza de 95 % ($\alpha = 0,001$)	84
6.10. Curva operaciones promedio junto con intervalo de confianza de 95 % ($\alpha = 0,0002$)	85
 B.1. Gráfica resultado global algoritmo SAGMM consolidado por acción	96
B.2. Resultados globales de la ejecución del algoritmo SAGMM separado por actor y camara.	97
B.3. Gráfica resultado global algoritmo NP consolidado por acción	98
B.4. Resultados globales de la ejecución del algoritmo NP separado por actor y camara.	99
B.5. Gráfica resultado global algoritmo UCV_STAIRCASE consolidado por acción . .	100
B.6. Resultados globales de la ejecución del algoritmo UCV_STAIRCASE separado por actor y camara.	101
B.7. Gráfica resultado global algoritmo UCV_LINEAR consolidado por acción	102
B.8. Resultados globales de la ejecución del algoritmo UCV_LINEAR separado por actor y camara.	103
B.9. Gráfica resultado global algoritmo MOG2 consolidado por acción	104
B.10. Resultados globales de la ejecución del algoritmo MOG2 separado por actor y camara.	105

Índice de Tablas

6.1.	Tabla de acciones humanas definidas en MuHAVI	67
6.2.	Tabla que relaciona el valor de α y número de frames necesarios para que un objeto se mantenga estático.	70
6.3.	Área bajo la curva de los algoritmos	71
6.4.	Métricas de desempeño con factor de aprendizaje 0.001	76
6.5.	Métricas de desempeño con factor de aprendizaje 0.001 y morfología	76
6.6.	Métricas de desempeño con factor de aprendizaje 0.0002	77
6.7.	Métricas de desempeño con factor de aprendizaje 0.0002 y morfología	77

CAPÍTULO 1

INTRODUCCIÓN

1.1. ANTECEDENTES Y MOTIVACIÓN

Visión por computador es una disciplina que intenta emular las capacidades de la visión humana para identificar y clasificar diferentes objetos desde una imagen. Es un área de investigación muy amplia, que se entrelaza con diferentes campos de investigación en el ámbito de las ciencias de la computación y el desarrollo tecnológico. Tiene como desafío facilitar máquinas computacionales con capacidad de percepción humana, en otras palabras comprender el medio ambiente, tomar acciones, aprendizaje, y mejorar desempeño de acciones tomadas. Visión por computador intenta hacer una descripción del mundo que los seres humanos pueden observar, a través de imágenes para hacer una reconstrucción de sus principales propiedades.

Hay en el último tiempo un constante desarrollo de nuevas técnicas y aparatos tecnológicos (figura 1.1), que incorporan algoritmos empleados en visión por computador, en heterogéneos campos de aplicación. Se está experimentando algoritmos inteligentes en vehículos que permiten evitar obstáculos, reconocer peatones o viajar de un punto a otro sin la intervención de un conductor. En dispositivos audiovisuales, como cámaras fotográficas digitales, algoritmos de reconocimiento de figuras y rostros es una característica normal incluida en estos artefactos. Se utilizan además, las técnicas de análisis y reconocimiento de patrones en imágenes médicas, o clasificación de acciones humanas en cámaras de vigilancia, entre muchas otras aplicaciones.

Las diferentes etapas de la cadena de procesamiento en visión por computador constituyen un conjunto extenso de técnicas, algoritmos y métodos provenientes, entre otros, del mundo de procesamiento digital de imágenes y clasificación del área de máquinas de aprendizaje computacional. En cada una de estas etapas además, se emplean diferentes aproximaciones para dar solución al problema que se intenta resolver. La figura 1.2, es un diagrama general de las distintas etapas que



Figura 1.1: Imágenes de aplicaciones en visión por computador

intervienen en un sistema de visión por computador. Una primera etapa consiste en la adquisición de secuencia de imágenes. Las diferentes imágenes del campo de aplicación que se quiere estudiar, son obtenidas mediante algún dispositivo sensor (cámara de vídeo) que transforma una escena del mundo real en una señal electrónica. Las señales son muestreadas y discretizadas para ser convertidas en una matriz de números que representan la escena en un conjunto de imágenes digitalizadas. La etapa de pre-procesamiento se refiere principalmente al mejoramiento de la entrada por medio de técnicas de procesamiento digital de imágenes. Se aplican métodos en el dominio espacial (transformaciones, filtros espaciales, extensión de contraste, procesamiento y ecualización de histogramas, entre otros) y en el dominio de la frecuencia (técnicas basadas en modificar la transformada de *Fourier* de una imagen) para acondicionar las imágenes que permitan resaltar alguna propiedad que requiera la aplicación específica. El proceso de segmentación es una etapa de subdivisión de objetos en una imagen. Se agrupan componentes de una imagen en elementos que tengan alguna característica propia, como forma, color o distribución estadística del color. Un procedimiento similar complementario a la segmentación, es la sustracción del fondo de imagen en una secuencia, esta consiste en modelar el fondo de imagen y sustraer mediante diferentes técnicas los objetos que están sobre ese fondo modelado. La etapa de extracción de características transforma las imágenes de entrada en una representación de elementos característicos agrupados en un vector de propieda-

des de la imagen o vector característico. Estos vectores sirven de base en la etapa de clasificación o reconocimiento de patrones.

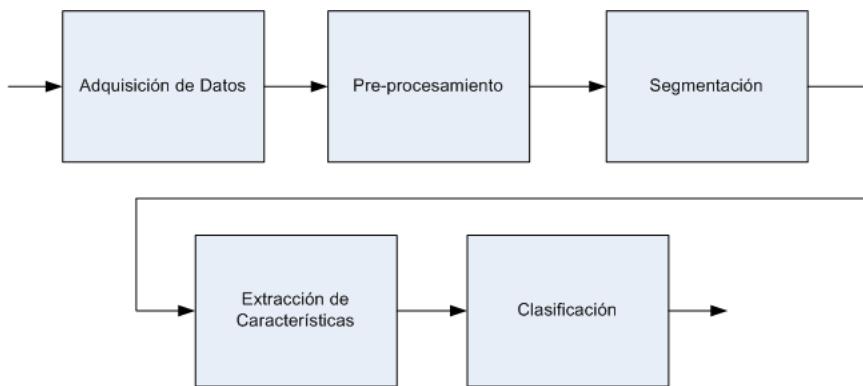


Figura 1.2: Diagrama de bloques genérico de las diferentes etapas en visión por computador

El reconocimiento y clasificación de actividades humanas es una de las tareas más complejas dentro del quehacer de visión por computador, la cual tiene un desarrollo constante dentro de los grupos dedicados a la investigación dentro de este campo. Existe un desafío permanente para desarrollar nuevas técnicas y algoritmos que mejoren los métodos de detección y clasificación de siluetas (“foreground”) desde un segundo plano (“background”) en una secuencia de imágenes, en diferentes condiciones de iluminación y sombras. Se emplean por ejemplo, algoritmos que construyen un modelo estadístico de los “píxeles” de la imagen de fondo, o algoritmos que comparan distribuciones (histogramas) para hacer el seguimiento de objetos en una secuencia de video (“*mean shift tracking*”).

Se requiere un conjunto preestablecido de imágenes o videos, que posibiliten evaluación y comparación de las técnicas y métodos utilizados en segmentación, clasificación y reconocimiento de actividades humanas. En este contexto, existe una variedad de conjuntos de datos (“datasets”) disponibles en la comunidad, para ser usados como parte del proceso de evaluación y comparación de algoritmos. Estas base de datos públicas, se componen de imágenes y acciones en secuencia, tomadas desde el mundo real en diferentes ángulos de observación, condiciones ambientales e iluminación.

Los investigadores requieren alguna referencia que les permita evaluar y comparar sus modelos. Deben asegurar que el modelo desarrollado cumple con los objetivos propuesto originalmente. En este ambiente surgen diferentes metodologías de evaluación de modelos, así como los conjuntos de datos que permiten emular un medio ambiente sobre el cual los algoritmos pueden ser ejecutados.

La finalidad de este trabajo de tesis, consiste en una primera etapa, implementar una plataforma de software que permita, incorporar y evaluar nuevos algoritmos de visión por computador.

Es un sistema prototipo de software que pretende establecer una base estándar de evaluación de desempeño de distintos tipos de algoritmos, relevantes en el desarrollo del campo de visión por computador. Este sistema de software viene para ayudar a reducir los tiempos de desarrollos de nuevos algoritmos y evaluar cambios de comportamiento de pequeñas modificaciones en algoritmos desarrollados. Proporciona una infraestructura base para desarrollar y posteriormente evaluar desempeño de nuevos software en visión por computador. Una segunda parte se propone consolidar las métricas de evaluación más utilizadas del campo de clasificación, en una herramienta de software única. Se busca de esta manera, obtener una métrica final de evaluación. El escenario de evaluación utilizado es MuHAVI [36], un conjunto de datos públicos que dispone diferentes acciones y condiciones orientado a la detección de acciones humanas.

1.2. DESCRIPCIÓN DEL PROBLEMA

Este trabajo plantea la utilización de un conjunto de datos de reconocimiento de acciones humana “MuHAVI” [36] (“*Multicamera Human Action Video dataset*”), como soporte de comparación y evaluación de rendimiento, de varios algoritmos de sustracción de imágenes de fondo (*Background Subtraction*). Para esto se propone consolidar un conjunto de métricas de evaluación, utilizadas en clasificación, en una herramienta de software que permita evaluar y comparar las siluetas resultantes de estos algoritmos, con una versión anotada de siluetas que dispone la versión mejorada del conjunto de datos MuHAVI-MAS (“*Manually Annotated Silhouette*”).

Se busca adaptar el algoritmo desarrollado por Zezhi Chen [8], propuesto inicialmente para localización, clasificación y seguimiento de vehículos, y otras versiones equivalentes de sustracción de fondo, en un único sistema de software que incorpore estas diferentes aproximaciones y generar un conjunto total de siluetas de las diferentes secuencias en MuHAVI para evaluar el rendimiento final de cada algoritmo.

1.3. SOLUCIÓN PROPUESTA

Se proyecta desarrollar un sistema de software, basado en un modelo orientado a objetos, que permita evaluar algoritmos de detección y clasificación de actividades humanas. Este software además, implementa e incorpora diferentes versiones de algoritmos de sustracción de fondos, para proporcionar un conjunto de siluetas como resultado final. Facilitando el proceso de evaluación de la herramienta de software principal.

El código implementado para esta solución estará constituido por una biblioteca de clases en

C++, que permitiría re-usar su código para diferentes tipos de algoritmos de detección de actividades humanas.

Este trabajo considera varios artefactos de software como entrega final; un sistema de evaluación de algoritmos, un “framework” de software que incorpore los diferentes algoritmos de sustracción de fondo, y un conjunto de siluetas necesarias para los algoritmos implementados y otras versiones de estos.

El resultado será un prototipo de software que puede constituir la base para el desarrollo de una plataforma de evaluación de algoritmos empleados en visión por computador.

1.4. OBJETIVOS Y ALCANCES DEL PROYECTO

1.4.1. Objetivo general

Implementar un sistema prototipo de software, que incorpore algoritmos de sustracción de fondo empleados en visión por computador, y desarrollar una herramienta de software de evaluación de desempeño para los algoritmos implementados, con el propósito de generar automáticamente siluetas desde conjunto de datos de reconocimiento de acciones humana usando el algoritmo de mejor rendimiento.

1.4.2. Objetivos específicos

- Implementar el algoritmo de sustracción de fondo propuesto por Zezhi Chen[8] en un programa ejecutable desarrollado en un lenguaje de programación C++ orientados a objeto.
- Desarrollar un sistema de software base que incorpore otros algoritmos de sustracción de fondo.
- Implementar una herramienta de software que consolide las métricas más utilizadas en sistemas de clasificación.
- Evaluar las siluetas resultantes de los distintos algoritmos ejecutados sobre MuHAVI.
- Generar curvas comparativas del desempeño de los algoritmos.
- Generar y publicar el conjunto total de siluetas resultantes del algoritmo con mejor desempeño para de todas las secuencias del conjunto de datos MuHAVI

1.4.3. Alcances

Este trabajo tesis considera varios artefactos como entrega final; desarrollo de un software prototípico que incorpora algoritmos de sustracción de fondo, identificación de métricas de evaluación de calidad, implementación de un programa que evalúa siluetas obtenidas por los distintos algoritmos, un conjunto de siluetas producidas automáticamente, evaluación de desempeño del conjunto de algoritmos de detección de fondo incluidos en el sistema de software prototípico. El resultado será un prototípico que puede constituir la base para el desarrollo de un “framework” de software para evaluación de otros algoritmos de usados en la detección de actividades humanas.

1.5. METODOLOGÍAS Y HERRAMIENTAS UTILIZADAS

La metodología a utilizar en el desarrollo de este trabajo de tesis, se apoyará en una etapa de investigación, implementación y verificación de resultados. Esta metodología estará basada en las actividades del método científico

- Identificación del problema. Existe un nuevo método de detección, seguimiento y clasificación de vehículos en autopistas urbanas que podría adaptarse para ser usado en detección de actividades humanas sobre una base de datos denominada MuHAVI.
- Planteamiento de una hipótesis. Los algoritmos que componen el método Z Chen [8] pueden ser adaptados e implementados para ser usados sobre MuHAVI en la detección de actividades humanas.
- Revisión de la literatura: Esta etapa consiste en hacer una revisión de las distintas publicaciones que dieron origen a la mejora de este método propuesto por Z Chen [8]. Además, revisar los actuales trabajos de clasificación basado en redes neuronales que permitan reconocimiento de actividades humanas.
- Implementación del modelo: Se basa en la implementación de los algoritmos de este método en clases C++ basados en el framework de visión por computador “OpenCV”.
- Etapa de validación y verificación certifica que la implementación realice detección de siluetas y posterior clasificación de estas. Es en esta etapa donde se hará un uso intensivo del software implementado para hacer una clasificación total de MuHAVI.
- Análisis de resultados y conclusiones, los resultados obtenidos en la etapa anterior son contrastados y comparados para obtener conclusiones sobre el método implementado.

1.5.0.1. Herramientas de desarrollo

- Sistema operativo Linux Ubuntu 12.04
- Biblioteca de maquinas de aprendizaje y visión por computador OpenCV versión 2.4.4 (“Open Source Computer Vision Library”)
- Compilador GNU C++
- Sistema de control de versiones "GIT"
- CMake.

1.5.0.2. Ambiente de desarrollo

Es proyecto será realizado en dependencias particulares, se dispondrá de un equipo computacional para hacer el desarrollo de esta aplicación de software.

1.6. RESULTADOS OBTENIDOS

Este trabajo de tesis presenta los siguientes resultados

- Un sistema de software base, implementado en lenguaje de programación C++, que permite agregar y utilizar algoritmos de sustracción de imágenes de fondo usados comúnmente en el área de visión por computador.
- Implementación de un algoritmo de basado en mixtura de componente gaussianos, denominado ‘*Self-Adaptive Gaussian Mixture Model*’ (SAGMM).
- Implementación de una herramienta de software que permite hacer evaluación de desempeño de los resultados obtenidos por algoritmos de sustracción de imágenes de fondo, empleando las métricas más comunes en evaluación de rendimiento.
- Generación y publicación de siluetas resultantes, utilizando SAGMM, de todas las secuencias del conjunto de datos MuHAVI.

1.7. ORGANIZACIÓN DEL DOCUMENTO

El primer capítulo de este trabajo de titulación se presenta el proyecto a desarrollar, se menciona el alcance, objetivo general, y los objetivos específicos necesarios para lograr el objetivo general.

Se detalla las herramientas necesarias para construir el proyecto, finalmente se hace una breve descripción de la solución propuesta. En segundo capítulo se hace una revisión bibliográfica de los principales temas abordados en la tesis. Se detalla el estado actual de los métodos de sustracción de fondo en la comunidad de investigación en visión por computador, especialmente en los métodos de modelado estadístico. Se mencionan las métricas más utilizadas para evaluar segmentación de imágenes y clasificación de actividades, también se hace una breve mención de los distintos conjunto de datos construidos especialmente para evaluación de algoritmos de detección de actividades. El tercer capítulo se dedica exclusivamente a describir los algoritmos implementados en este trabajo de tesis, se detalla el algoritmo de sustracción de fondo usando mixtura de componentes Gaussianas. El capítulo cuarto describe los métodos empleados para evaluar calidad, y se hace diferencia entre métodos de evaluación subjetiva y objetiva. El quinto capítulo se dedica a mencionar las herramientas, los criterios y las decisiones para llevar a cabo este trabajo de tesis. Se describe también a nivel de bloques las distintas implementaciones de software realizadas, se presentan diagramas UML de paquetes y clases de los sistemas implementados. El sexto capítulo, detalla el resultado de las experimentaciones, y sus análisis, se muestran también las curvas características de resultados. Se agrega un apéndice con gráficas de los resultados obtenidos separados por por algoritmo. Finalmente el capítulo de las conclusiones, proporciona una idea del resultado general, se hacen las comparaciones de los diferentes algoritmos y se indican los posibles trabajos futuros que se podrían abordar a partir de este trabajo.

CAPÍTULO 2

ESTADO DEL ARTE

Los algoritmos de sustracción de fondo (“*Background Subtraction Algorithms*”) modelan esencialmente el segundo plano (“*Background*”) de una imagen con la finalidad de detectar movimiento de objetos (“*Foreground*”) en una secuencia de video. Aplicaciones como vigilancia a través de video, detección de movimiento y clasificación de acciones, necesitan primero modelar la imagen de fondo y luego detectar los objetos móviles.

Modelar el fondo de imagen plantea una serie de desafíos que han sido abordados desde diferentes perspectivas por los investigadores. Cambios de luminosidad, movimientos simultáneos, ambigüedad (imprecisión) sombra y fondo, entre otros, constituyen parte de los problemas que deben superar los algoritmos de sustracción de fondo para detectar y clasificar eventos en una secuencia de imágenes. Se han elaborado en la comunidad variados conjunto de datos (“*Datasets*”) [22, 23, 35, 36, 43] que emulan escenarios y condiciones con el objetivo de evaluar robustez, rendimiento, calidad en clasificación, de los diferentes modelos de fondo desarrollados. Esta lista de escenarios, es un conjunto de situaciones generales normalmente encontradas en secuencias de video, las cuales constituyen un punto de comparación y referencia entre los diferentes algoritmos implementados en la comunidad. Una descripción más detallada de estas situaciones generales es mencionada en [38].

Numerosos métodos de modelado de fondo han sido construidos considerando estas diferentes condiciones. Un estudio publicado por *Bouwmans*[3] (2011) presenta un cuadro general de estos métodos y hace una clasificación en diferentes categorías dependiendo de la forma de construir el modelo del fondo. Un modelo básico consiste en usar promedios, medianas o análisis de histogramas en el tiempo. Una técnica simple es calcular el promedio de una escena sin objetos en movimientos, restar nuevos cuadros de esta imagen y comparar a través de un umbral. Otro tipo de modelo se construye basado en distribuciones estadísticas, particularmente en distribuciones del

tipo “*Gaussianas*”, usando variables estadísticas para clasificar los píxeles. Se construyen modelos con redes neuronales en la que una red entrenada podría distinguir entre un píxel perteneciente al fondo o al objeto en movimiento. Otros modelos emplean “*clusters*” para agrupar píxeles en diferentes elementos dentro de una secuencia. También existen modelos que emplean filtros especiales para hacer una estimación del fondo, como filtros de “*Kalman*” o “*Tchebychev*”.

Bouwmans[3] también evidencia un principio de funcionamiento de la mayoría de los métodos de sustracción de fondo, independiente del tipo de modelo e implementación usado. Establece ciertas etapas que se cumplen en los modelos de fondo: modelado de fondo, inicialización del fondo, mantenimiento del fondo, detección (*foreground*), elección de medida de la característica (píxel, block, cluster), elección del tipo de característica (color, borde, texturas).

2.1. MODELO DE MIXTURA DE GAUSSIANAS

Mixtura de Gaussianas (*Gaussian Mixture Model - GMM*) ha sido uno de los métodos más mencionados en la literatura como herramienta de modelo estadístico para obtener el fondo de una secuencia de imágenes. Mixturas finita[32] de distribuciones es una herramienta de modelado estadístico, análisis de datos e inferencias, usado como base para proporcionar modelos descriptivos de distribuciones en distintas áreas de aplicaciones. Tiene la propiedad (entre otras) de estimar parámetros en un esquema de aprendizaje no-supervisado, de observaciones producidas por un conjunto de fuentes aleatorias. En métodos de sustracción de fondo se utiliza mixtura de Gaussianas para modelar en el tiempo los valores de un píxel (o un vector de valores en espacio de colores RGB) y clasificar en diferentes categorías, generando “*clusters*” de píxeles los cuales representan las diferentes mixturas de componentes.

Uno de los primeros trabajos realizados con distribuciones estadísticas, que marca el inicio de investigaciones relacionadas con la utilización de mixtura de distribuciones para modelar fondo, es el trabajo de *Friedman* y *Russell* [17]. Ellos proponen, en un sistema de vigilancia del tráfico automovilístico, obtener el fondo mediante una composición fija de distribuciones Gaussianas. Modelan el valor de intensidad de un píxel con tres distribuciones predeterminadas, asociadas a tres diferentes tipos de objetos: vehículos, sombras, y camino (fondo de la imagen). Utilizan una versión incremental del algoritmo esperanza-maximización[10] (*EM*), para inicializar y actualizar (como mecanismo de aprendizaje no-supervisado) los parámetros de la mixtura del modelo. La clasificación de píxeles es realizada comparando varianza con un umbral definido por la distancia *Mahalanobis*. Durante el proceso de inicialización las tres distribuciones son etiquetadas por una heurística que determina como sombra el componente más oscuro, la distribución de vehículos se

relaciona con una mayor varianza, y la distribución del camino con la menor varianza.

Una aproximación similar es el trabajo de *Stauffer* y *Grimson*[37]. Ellos modelan el valor de un píxel con una mixtura de Gaussianas, a diferencia de *Friedman* y *Russell*[17] que utilizan un conjunto predeterminado de distribuciones para modelar un píxel. Ellos señalan que modelar el fondo mediante un conjunto fijo de distribuciones Gaussianas, el funcionamiento del sistema podría funcionar incorrectamente, con los píxeles que no estén dentro de las distribuciones prefijadas. En su propuesta plantean, que la persistencia y la varianza de cada Gaussiana en la mixtura, determina las distribuciones que pueden corresponder a la imagen de fondo. Los píxeles que no se ajustan con las Gaussianas del fondo, se consideran parte de elementos en movimiento (*Foreground*) y sólo son parte de éste, cuando existe consistente evidencia para convertirla en una nueva Gaussiana de las mixturas del fondo de imagen. Elementos con desplazamiento lento, por ejemplo, no se incorporan en el fondo debido a que su varianza es mayor con respecto a las varianzas que describen el fondo. Cada píxel es modelado por una mixtura de K Gaussianas (K es una valor entre 3 y 5) y los parámetros se inicializan utilizando un algoritmo *K-Means* (*Friedman* y *Russell*[17] utilizan un algoritmo EM[10]). Este método además, incorpora dos importante parámetros, que serán utilizados para futuras publicaciones. Formulan una constante de aprendizaje α , y T un factor de proporción de los datos que podrían ser considerados parte del fondo.

La estimación de los parámetros en mixtura de componentes es computacionalmente alta y requiere muchos recursos computacionales, en términos de procesamiento y hardware. Una mejora del algoritmo de mixtura de Gaussianas es presentado por *Zivkovic* y *Heijden* [47] (2006). Muestran desde una perspectiva Bayesiana, un criterio de selección en tiempo real, del número adecuado de componentes en un píxel. Esto es, adaptar de manera automática el número de componentes a una escena. Es un algoritmo que estima los parámetros de la mixtura en tiempo real y simultáneamente selecciona el número de Gaussianas, usando una distribución a priori de *Dirichlet*. El número K es adaptado en forma dinámica a la característica de la distribución (multimodalidad) de cada píxel; un píxel podría estar correctamente descrito por una Gaussiana y otro diferente podría quedar representado por un número mayor de Gaussianas (por ejemplo ondulaciones de en la superficie de un lago). Esta nueva aproximación, se basa en trabajos anteriores [4, 16, 46] que intentan mejorar los problemas que presentan el algoritmo esperanza-maximización (*EM*) para estimar parámetros de una mixtura (caer en un mínimo local si no es inicializado apropiadamente).

En el contexto de un sistema de detección, clasificación y seguimiento de vehículos [8] en ciudad, *Zezhi Chen* [7] (2011) menciona como desventajas, en una mixtura de Gaussianas, la sensibilidad a los cambios bruscos de iluminación, e identificación de sombras móviles producidas por vehículos dentro de una escena. Comenta además, el inconveniente de éste modelo en el uso

intensivo de recursos computacionales y el cuidado que requieren la sintonización de sus parámetros. Propone un modelo de mixtura Gaussianas auto-adaptativo, el cual consiste en una mejora del método de *Zivkovic y Heijden* [47]. Introduce un procedimiento para obtener un factor numérico que determina los cambios en la iluminación global. Este procedimiento se basa en un estudio comparativo de algoritmos que corrigen cambios de intensidad global [44]. Combina el factor de iluminación y un registro de actualización de las mixturas para mejorar tasa de aprendizaje, convirtiéndola en una tasa dinámica que mitiga los cambios bruscos de iluminación global. Propone también un filtro espacial y temporal [9] para abordar los problemas de ruido y vibración en las cámaras. La tasa de aprendizaje dinámica propuesta logra buena estimación de la media y varianza en caso de fondos que cambian rápidamente. En caso contrario, la tasa dinámica se aproxima a la constante de aprendizaje original[47].

Incluir el algoritmo *Mixtura de Gaussianas* en un micro-controlador es el desafío que abordan *Salvadori y Petracca* [34] en un trabajo del año 2012. Ellos usan este método en procesadores de bajo consumo, bajo costo, y sin procesamiento en punto flotante. Para reducir el uso de memoria y el costo de computación, hacen una aproximación en la precisión de números enteros. Crean nuevas reglas de actualización de los parámetros que modelan las componentes Gaussianas de este método, hacen una definición de un operador de redondeo para actualizar los valores medio y de varianza de los componentes, que emulan la actualización de parámetros del método original. Los pesos que determinan la pertenencia de un componente Gaussiano al fondo de una imagen son representados por contadores. Localizan los tres parámetros definido para cada gaussiana en un número fijo de bits en un conjunto de 4 bytes, e incrementan estos contadores en función de la actualización de estos parámetros. Esta aproximación crea restricciones con el número del factor de aprendizaje, determinadas por los valores mínimos de la varianza y valor medio.

Un método más flexible para tratar con fondos de imagen dinámicos; fluctuaciones de cámaras, movimientos de las hojas de un árbol, etc. Es el algoritmo de estimación de fondo no-paramétrico propuesto por *Elgammal et al.* [13]. Plantea modelar la función de probabilidad mediante el uso de un “Kernel” normal Gaussiano. Este modelo no requiere seleccionar el número de componentes Gaussianas, estiman la función de kernel usando información de historia reciente, con el propósito de capturar cambios rápido en una escena. Sin embargo, una de las mayores desventajas de este modelo, está relacionado con el costo computacional, éste requiere mantener en memoria varios cuadros de una secuencia.

2.2. EVALUACIÓN DE RENDIMIENTO

Las variables más mencionadas en la literatura que miden segmentación, corresponden a las métricas definidas para evaluar rendimiento en sistemas de recuperación de información (*Information Retrieval - IR*). Área que viene de la teoría de clasificación. En algoritmos de sustracción de imágenes de fondo se usan frecuentemente las variables *Precision* y *Recall* [1, 33], para evaluar la clasificación de los píxeles en una imagen. La combinación de estas dos variables definen además *F-Measure* [24], que es una medida global del rendimiento de un algoritmo.

Estas variables se usan principalmente, para un tipo de evaluación estática, en la cual el resultado estimado (de una imagen o secuencia de ellas) es comparado con su versión equivalente anotada (*Ground-Truth*). Método conocido como de discrepancia empírica. En general este tipo de evaluaciones utilizan el promedio de las mediciones de verdadero y falso positivo (*TP* y *FP* respectivamente), a la vez verdadero y falso negativos (*TN* y *FN* respectivamente). Combinaciones de estas mediciones, obtenidas desde una secuencia de imágenes, derivan en variables conocidas como tasa de falsos positivos (*False Positive Rate - FPR*) o tasa de falsos negativos (*False Negative Rate - FNR*) [28]. En [33] emplean estas variables para construir nuevas medidas de calidad, formulando los conceptos denominados “buena detección” (*Good Detection*) y “buena discriminación” (*Good Discrimination*) [33]. Buena detección, se refiere principalmente a minimizar los falsos negativos (*FN*) y buena discriminación es minimizar falsos positivos, eso es conseguir un buen compromiso entre las variables de “*Recall*” y “*Precision*”.

El algoritmo Wallflower[38] trabajo del año 1999, propone un nuevo modelo de sustracción y mantenimiento del fondo. Éste consiste de estadísticas asociadas a un modelo del fondo. Definen como metodología de evaluación, un conjunto de 10 obstáculos representativos que un modelo ideal debiera superar: objetos desplazados (*moved objects*), cambios graduales de iluminación (*time of day*), cambios repentinos de iluminación (*light switch*), variaciones (*waving tree*), camuflaje, algoritmo no entrenado (*bootstrapping*), coloración homogénea (*foreground aperture*), objetos inactivos (*sleeping person*), objetos activos (*waking person*), y sombras (*shadows*). Este trabajo, recomienda aplicar la metodología de evaluación propuesta, en cada uno de los diferentes algoritmos para comparar su rendimiento de clasificación, usando falsos negativos (píxeles no reconocidos dentro de un objeto) y falsos positivos (píxeles identificados erróneamente como fondo) como métricas de evaluación de rendimiento.

Un segundo enfoque de evaluación, intenta cuantificar la percepción visual subjetiva que podría tener un observador independiente del resultado final de segmentación. Se definen los conceptos de precisión espacial y estabilidad temporal [6] [40] como métricas objetivas. La idea es ponderar el resultado de la segmentación de acuerdo con la distancia de los píxeles clasificados (background-

d/foreground) al borde de la imagen segmentada. Mientras mayor es la distancia de un pixel mal clasificado del borde de un objeto, mayor es su ponderación. Para esto se propone, por ejemplo una ponderación del tipo logarítmica para los falsos positivos y una ponderación lineal para falsos negativos [6] [40]. Los falsos negativos son más relevantes a mayor distancia, por esto se ponderan linealmente. En [28] proponen una forma similar de evaluación, pero definen una función *sigmoidal* para ponderar ambas variables FP y FN.

En el desafío BMC (*Background Model Challenge*) [39], se plantean una serie de métricas como criterio de evaluación de rendimiento. Para esto proponen dos tipos de evaluaciones, una que mide *calidad estática* y otra *calidad dinámica*. Para medir calidad estática, piden usar *F-measure* que relaciona Precision y Recall, mesurando el total de estas variables de cada escena, es decir, para cada una de las imágenes de la secuencia se debe obtener F-Measure, promediando el total de todos los valores. También, solicitan medir la relación señal a ruido de cada una de las imágenes en la secuencia (*FSNR*). Las mediciones de calidad dinámica, es un concepto que se refiere se refiere a la percepción de segmentación en el tiempo. Utilizan una métrica denominada “medida de percepción” (*perceptual measure-SSIM*), y otra denominada “D-score”[27].

2.3. CONJUNTO DE DATOS

Se encuentran disponibles en Internet una amplia variedad de conjunto de imágenes, para ser utilizados como base de evaluación y comparación de nuevos métodos, en las diferentes áreas visión por computador. Estas base de datos, dependiendo de la realidad, facilitan secuencias en diferentes condiciones y conjuntos más elaborados proporcionan imágenes anotadas (*metadata*). Por ejemplo, existen base de datos de peatones [11, 14] para ser usadas en sistemas de asistencia a la conducción de vehículos o sistemas de protección a peatones. Otras como Mobo (*CMU Motion of Body*) [23], creada para investigar la forma de caminar de los seres humanos, lo cual permitiría evaluar algoritmos que permitan identificar a las personas por la forma de caminar.

Para clasificación de acciones humanas, existen en la literatura varios conjuntos estándar. KTH [35] es un conjunto de seis acciones humanas diferentes entre ellas, ejecutadas por 25 actores, con una cámara estática (25 cuadros/segundo), en cuatro ambientes distintos: interior, exterior, exterior con diferentes escaleras, exterior con diferentes prendas de vestir. Fue introducida para evaluación de reconocimiento de acciones usando mediciones locales de puntos de interés espacio-temporal (*spatiotemporal interest points - STIPs*). Contiene 598 secuencias de videos y 2391 acciones. Cada video tiene un promedio de cuatro segundos de duración.

El conjunto de datos Weizmann [22] (ver figura 2.2), fue construido para comprobar robustez de

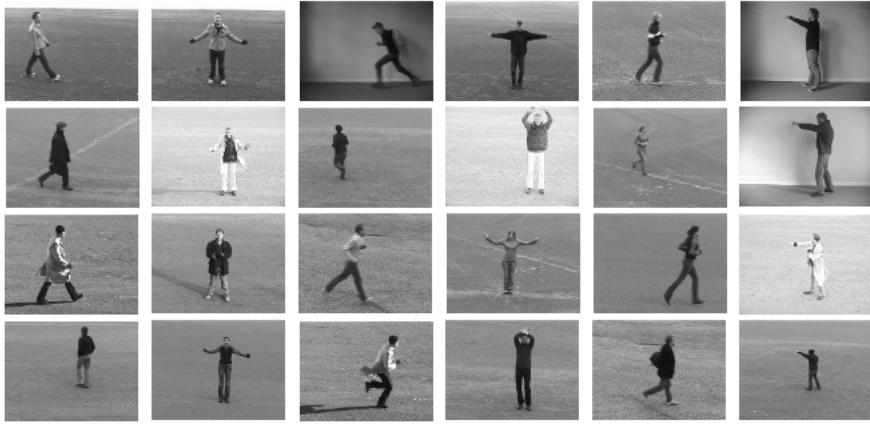


Figura 2.1: Conjunto de Datos KTH, desarrolla seis actividades humanas ejecutadas por 25 actores en cuatro diferentes ambientes, facilitando 2391 acciones en 598 secuencias

un método detección y clasificación de acciones humanas utilizando el concepto volumen espacio-tiempo. Consiste de 90 secuencias de baja resolución, diez acciones distintas ejecutadas por nueve actores en diferentes escenarios y imagen de fondo. Las imágenes son capturadas por una cámara estática localizada frente a estos actores.



Figura 2.2: Conjunto de Datos Weizmann consiste en diez acciones realizadas por nueve actores

MuHAVi [36] Es un conjunto de acciones humanas recolectados desde ocho ubicaciones diferentes (8 cámaras), en un escenario irregular emulando iluminación nocturna de la calle. Contiene 17 clases diferentes de actividades, ejecutado por 14 actores (figura 2.4). INRIA XMAS [43] (figura 2.3) muy similar a MuHAVi [36], se compone de 11 actividades cotidianas, por ejemplo, mirar la hora, darse vuelta, o caminar, entre otras. Cada acción es ejecutada en tres oportunidades por diez actores, cinco hombres y cinco mujeres. Una de las características de este conjunto, se refiere que la posición y dirección de las actividades son realizadas en forma libre por los actores, de esta

manera se logra que las actividades sean invariante a la vista (*view-invariance*).



Figura 2.3: Conjunto de Datos IXMAS, es un ejemplo de 11 actividades cotidianas elaborada con diez actores



Figura 2.4: Conjunto de Datos MuHAVI, son 7 actores desarrollando 17 acciones diferentes en un escenario simulando iluminación nocturna.

CAPÍTULO 3

ALGORITMOS SUSTRACCIÓN DEL FONDO

3.1. INTRODUCCIÓN

Esta capítulo hace una presentación de tres métodos basados en modelos estadísticos para describir el segundo plano de una secuencia y hacer separación de objetos en movimiento de las imágenes del fondo. Se describe el modelo de *Zivkovic y Heidjen* [47] de Mixtura de Gaussianas, el cual constituye la base de la mejora propuesta por *Zezhi Chen* [8], se incluye el trabajo de *Salvadori y Petracca* [34] que incorpora el modelo de mixtura de componentes gaussianos (*GMM*) en micro-controladores. Estos modelos son los algoritmos sobre los cuales se desarrolla este trabajo de tesis. También se describe un modelo no-paramétrico (Elgammal et al[13]) para describir el fondo de una imagen. Todos los modelos basado en mixtura de componentes gaussianos han sido incluidos en el desarrollo del sistema de software que permite ejecutar, para luego comparar y evaluar desempeño sobre las secuencias proporcionadas por MuHAVI, descritos más adelante en el capítulo 5. El algoritmo de *Zivkovic y Heidjen* se denomina *MOG* (“*Mixture of Gaussians*”) y el de *Zezhi Chen SAGMM* (“*Self-Adaptive Mixture of Gaussians*”) y *UCV* para los algoritmos del trabajo de *Salvadori y Petracca*.

3.2. MODELO GENERAL

La idea general del proceso de sustracción de fondo, es obtener y mantener actualizado (con nuevas imágenes) el modelo de la escena del fondo (*background*) de una secuencia de imágenes, con el propósito de detectar movimientos inusuales, objetos en movimiento, dentro de esta secuencia. Se modela el comportamiento de cada uno de los píxeles de la escena, mediante una función

de densidad de probabilidad. El concepto se fundamenta, en la suposición que las imágenes de una escena sin objetos en circulación, tienen un comportamiento regular que puede ser descrito por un modelo estadístico. Un evento podría, por ejemplo, ser detectado identificando las partes de la imagen, que no se ajustan con el modelo que describe el fondo. Un nuevo píxel sólo va a ser considerado parte del fondo, si su valor queda completamente descrito por la función de densidad de probabilidad que es mantenida en el modelo de la imagen de fondo.

Se utiliza el valor de intensidad de un píxel, en el tiempo (o en el transcurso de una secuencia), para construir una función de densidad de probabilidad, que corresponde al modelo estadístico que describe el comportamiento del píxel. Esta densidad de probabilidad puede ser estimada mediante la construcción de modelos paramétricos o no-paramétricos. El primer tipo de modelo utiliza la combinación lineal de un conjunto de distribuciones Gaussianas, en el que cada componente (Gaussiano) es caracterizado por un grupo de parámetros, que identifica los elementos que emergen durante el avance de las imágenes. El segundo tipo es construido a través de una estimación no-paramétrica de la función de densidad de probabilidad. A diferencia del tipo de modelado paramétrico, que estima un conjunto de parámetros de una función de densidad, el no-paramétrico emplea una función '*kernel*' (Normal, Uniforme, Epanechnikov, Gaussiano) para construir la forma de la función de densidad de probabilidad.

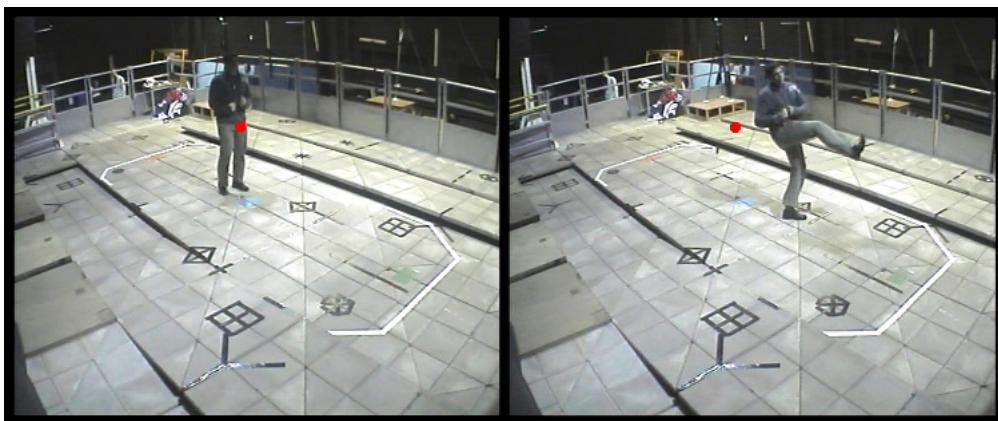


Figura 3.1: Imágenes 570 y 610 de la secuencia MuHAVI-MAS “*Kick Camera 3 Person 4*”. El punto en rojo señala la posición de un píxel en dos eventos diferentes

La separación de píxeles entre elementos del fondo y otros objetos, puede ser explicado en forma simple, mediante las dos imágenes de la figura 3.1, tomadas de la secuencia MuHAVI “*Kick Camera 3 Person 4*”[36]. Ambas imágenes señalan el estado de un píxel (punto en rojo) en dos momentos diferentes. El píxel indicado en la imagen de la figura 3.1(a), es parte en ese instante del actor (*foreground*), ubicado en medio del escenario, y su valor queda determinado por los colores de su vestimenta. Unos segundos después el actor se encuentra en una posición diferente, realizando

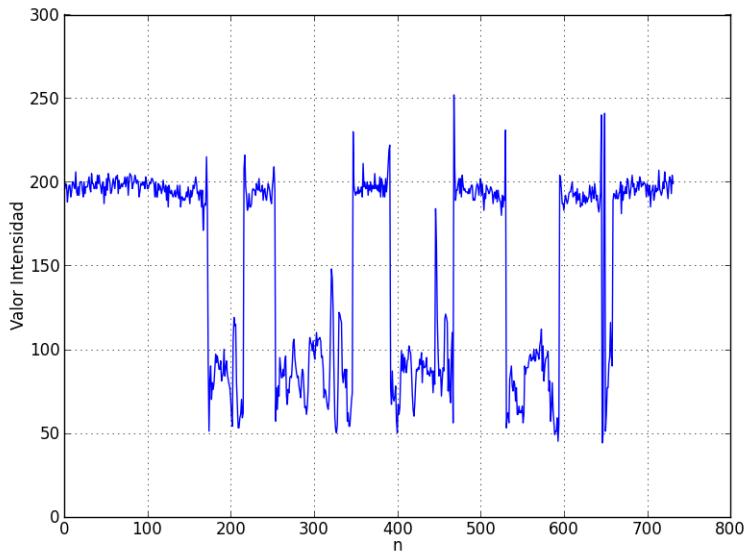


Figura 3.2: Cambios del valor de intensidad de un color en la secuencia completa.

una determinada acción. En este nuevo instante el valor del píxel queda definido por los colores del escenario (*background*). La gráfica de la secuencia completa (730 imágenes), que describe el comportamiento del píxel se muestra en la figura 3.2. Los cambios repentinos de intensidad (cambio del nivel 200 al rango de valores 50-100), reflejan los pasos del actor en medio del escenario en la posición donde se encuentra localizado este píxel. Por razones computacionales, se asume que los valores de intensidad de los tres componentes en el espacio RGB (azul, verde y rojo) tienen la misma varianza [47], de esta manera la matriz de covarianza queda definida como una matriz identidad multiplicado por ese único valor de varianza.

Al observar el rango valores asociados al escenario en la figura 3.2, se evidencian valores más constantes (o menor dispersión) que los del actor cuando pasa por este píxel. Esto también se puede apreciar en el modelo de la secuencia completa que describe el histograma de la figura 3.3(a). En esta imagen se pueden distinguir dos curvas bien definidas, el fondo de imagen y el actor en movimiento. La mixtura en este ejemplo, se construye ajustando la función de densidad a dos distribuciones Gaussianas, figura (3.3)(b). De esta manera, el modelo que describe este píxel en particular para esta secuencia, queda completamente definido por una composición de dos funciones Gaussianas. Se puede ver también en esta figura, que la imagen de fondo puede ser asociada con la componente de menor varianza, debido principalmente a la menor dispersión de valores del fondo. Sin embargo, el menor valor de frecuencia puede ser relacionado con el actor en movimiento, debido al menor tiempo que éste cruza el escenario.

Se aprovecha estas características de la Mixtura de Gaussianas, para hacer una separación entre

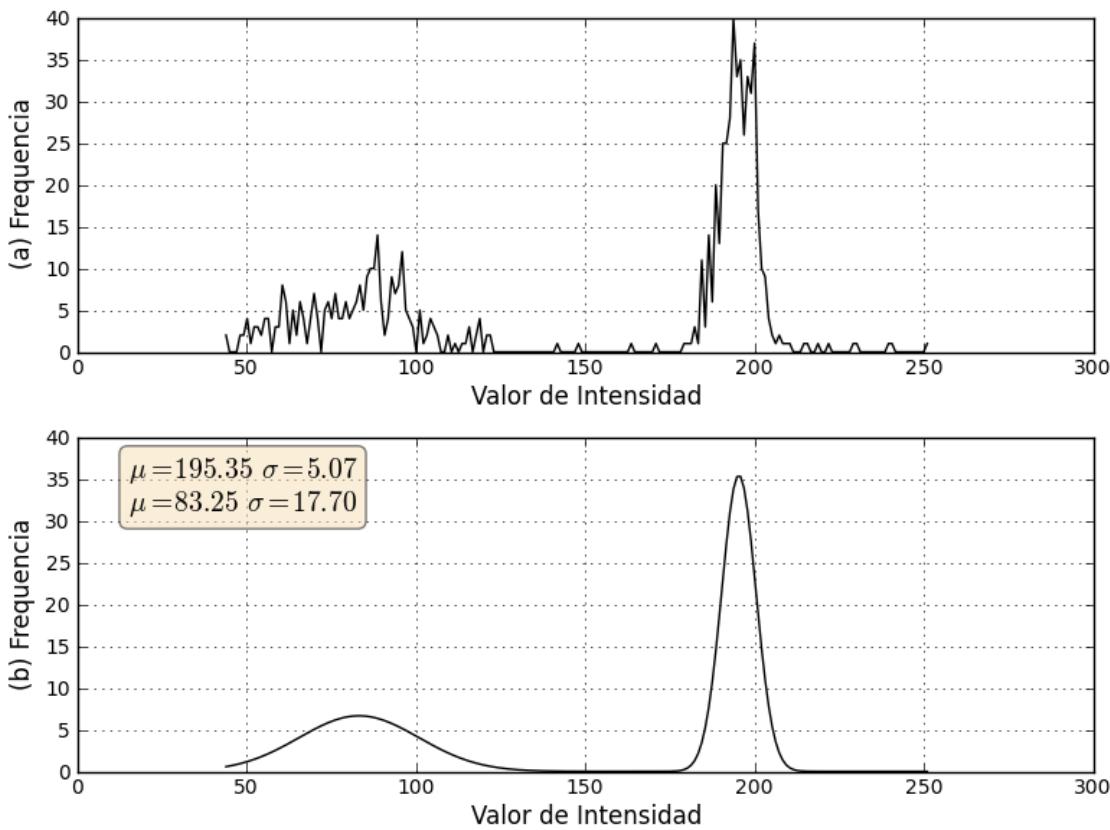


Figura 3.3: Histograma secuencia completa “*Kick Camera 3 Person 4*”

foreground y *background*. Así cuando la componente Gaussiana que describe algún píxel no se ajuste a la distribución con menor varianza, será considerando parte de un objeto en movimiento. No obstante, si este mismo objeto queda estático por un tiempo considerable, la varianza de su distribución Gaussiana comenzará a disminuir y tomar más relevancia (mayor ponderación relativa) que la varianza del componente del fondo, hasta llegar un momento que será considerado parte del fondo de imagen.

Los gráficos de las figuras 3.4 y 3.5 sirven de ejemplo para entender el procedimiento de actualización de parámetros, de los diferentes componentes Gaussianos definidos en el algoritmo. Estas figuras son gráficos de dispersión de la secuencia completa (*Kick Camera 3 Person 4*) para el mismo píxel del ejemplo señalado en la figura 3.1. La dispersión en la gráfica 3.4, conforma principalmente dos nubes o *clusters* que se asocian con los dos elementos principales de la secuencia: el escenario y las acciones del actor. El *cluster* más compacto de puntos, localizado en el rango 150-250 de los tres ejes de colores, resume los diferentes tiempos que el píxel fue parte del escenario, mientras el otro *cluster* localizado en la parte inferior refleja la cantidad de veces que el píxel fue parte del actor durante sus movimientos.

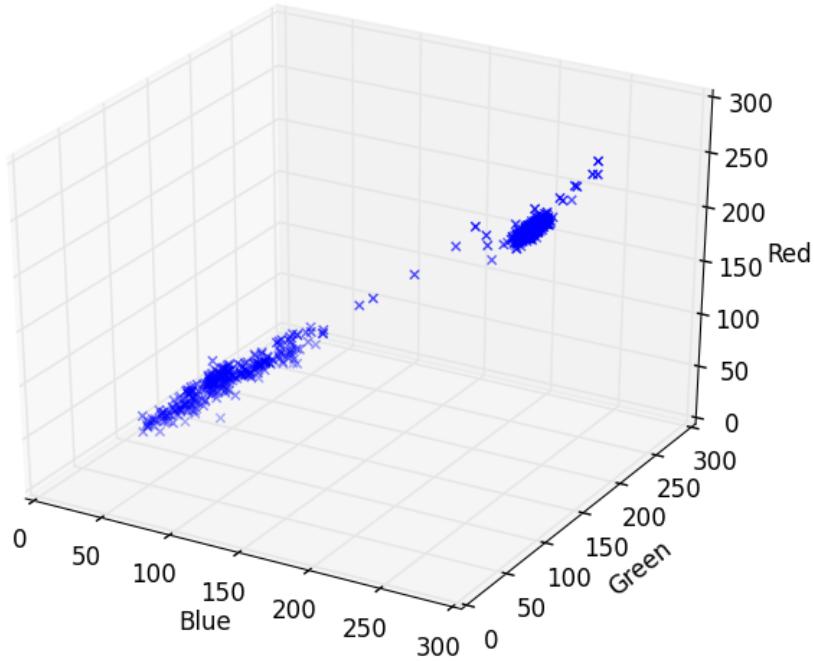


Figura 3.4: Gráfico de dispersión en secuencia “*Kick Camera 3 Person 4*” que muestra la nube de puntos relacionados con el escenario y el actor en movimiento.

Durante el tiempo que dura la ejecución del algoritmo, los componentes Gaussianos de un píxel van siendo creados o suprimidos en tiempo real, dependiendo exclusivamente de los nuevos valores de entrada y el tiempo de permanencia de estos valores, en alguno de los diferentes *clusters* de la secuencia. Los valores presentan un comportamiento dinámico en directa relación con los eventos de la secuencia; éstos se pueden mover entre *clusters* o bien mantenerse en uno de ellos por un largo periodo. Se aprovecha esta división de *clusters* para hacer la separación entre *background* y *foreground*. La gráfica de la figura 3.5 (dispersión verde-azul) destaca cuatro componentes (*clusters*) definidos por el algoritmo, en el instante que el actor se encuentra detenido de pie en medio del escenario, ejemplo de la figura 3.1(a). Los dos puntos en rojo revelan la posición del píxel en los *clusters* para las dos imágenes de ejemplo en la figura 3.1. Cada uno de estos componentes tiene un factor de ponderación π_k que indica el tiempo de permanencia del píxel en la nube de puntos. Este factor de ponderación es incrementado en la medida que el valor del píxel se mantenga dentro de un margen (de uno de los componentes), determinado por la distancia de Mahalanobis, del punto hacia cada uno de los *clusters*. Para discriminar que el píxel es parte del actor (o el es-

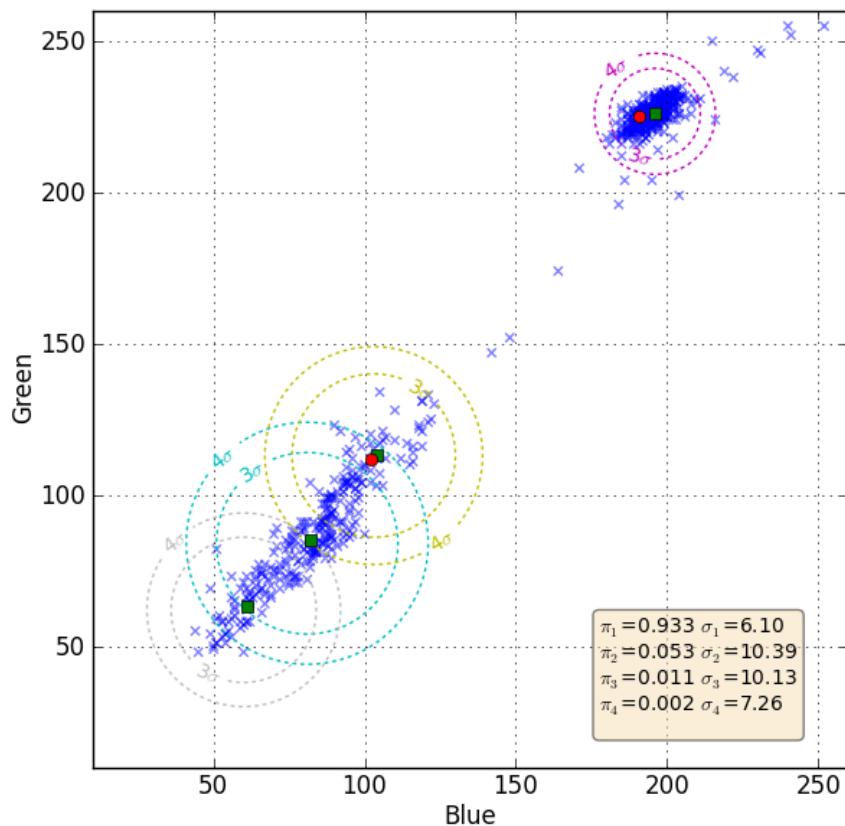


Figura 3.5: Gráfico de dispersión secuencia “Kick Camera 3 Person 4”, que muestra dos *cluster* bien definidos

cenario), el algoritmo calcula las distancias del pixel a los “centroides” (media) de los diferentes *clusters* establecidos en ese instante. El punto es asociado con algún *cluster*, si el resultado de la distancia de Mahalanobis es menor a tres veces su varianza. Si el punto no es vinculado a ninguno de los clusters definidos en ese momento, el algoritmo crea uno nuevo y deja el punto ligado con este nuevo cluster. Una vez que el algoritmo ha identificado el punto en alguno de los clusters, se realiza la clasificación, ya sea *background* o *foreground* y por lo tanto el pixel queda clasificado por la característica del cluster, al que ha sido asociado. Esto es, si el cluster en particular tiene un factor de ponderación más alto que todos los demás, ese cluster es *background* y en consecuencia el pixel también. En caso contrario, el pixel que pertenece a un cluster con factor de ponderación inferior, por lo tanto es algún objeto en movimiento.

3.3. MIXTURA DE DISTRIBUCIONES GAUSSIANAS - GMM

La superposición de distribuciones Gaussianas que describen el comportamiento de un pixel, se define en la ecuación (3.1). Esta es una sumatoria de “ K ” componentes Gaussianos $\mathcal{N}(x|\mu_k, \Sigma_k)$, cada uno ponderado por un factor multiplicativo π_k , que establece una función de densidad de probabilidad general para el píxel $p(x)$. El factor de ponderación π_k , es el grado de influencia de cada componente en el comportamiento global del píxel que esta siendo modelado. Los valores de ponderación pueden ser interpretados, como probabilidades de ocurrencias de las diferentes clases, *Background* y *Foreground*, representadas por los diferentes componentes Gaussianos; por ejemplo, π_1 podría representar la probabilidad que un píxel sea la imagen de fondo. La suma total de los factores π_k es 1, indicado en la ecuación (3.2).

$$p(x) = \sum_{k=1}^K \pi_k \mathcal{N}(x|\mu_k, \Sigma_k) \quad (3.1)$$

$$\sum_{k=1}^K \pi_k = 1 \quad (3.2)$$

Una de las principales tareas del algoritmo que implementa mixtura de Gaussianas, es mantener actualizado los K valores estimados de media (μ_1, \dots, μ_k), covarianza ($\Sigma_1, \dots, \Sigma_k$) y el factor de ponderación (π_1, \dots, π_k), por cada píxel de una nueva entrada. Se utilizan los valores de intensidad en el espacio de colores RGB, para construir la mixtura de componentes Gaussianas de un píxel.

La estimación de los parámetros $\vec{\theta} = \{\pi_1, \dots, \pi_k, \mu_1, \dots, \mu_k, \Sigma_1, \dots, \Sigma_k\}$ de las distintas mixturas se realiza mediante el método de Máxima Verosimilitud (*Maximum Likelihood Estimate - MLE*). De esta manera, los parámetros estimados por máxima verosimilitud de un conjunto t de muestras $\mathcal{X} = \{\vec{x}^{(1)}, \dots, \vec{x}^{(t)}\}$, quedan determinados por la ecuación (3.3). Una explicación bien detallada de estimación mediante el método de máxima verosimilitud se puede ver en el capítulo 3 de la referencia [12].

$$\hat{\vec{\theta}} = \arg \max_{\vec{\theta}} (\log p(\mathcal{X}; \vec{\theta})) \quad (3.3)$$

Debido a la dificultad de encontrar una solución analítica de *MLE*, se emplea una aproximación numérica iterativa para maximizar la función de verosimilitud. El algoritmo de esperanza-maximización (EM) [10], se usa para encontrar soluciones de máxima verosimilitudes. Este es un procedimiento iterativo que busca un máximo local de la función log-verosimilitud. Este algoritmo es fácil de implementar, sin embargo una de sus limitaciones está en la posibilidad de converger en un máximo local cuando no se ha inicializado.

3.3.1. Actualización parámetros del Algoritmo

Esta parte, describe el método de actualización recursivo que se utiliza en la estimación de los parámetros (μ_k , Σ_k , π_k) de cada componente Gaussiano, establecido por *Zivkovic y Heidjen* [47]. Por razones computacionales, las matrices de covarianza son consideradas Isotrópicas, es decir la matriz de identidad multiplicado por un único valor de varianza.

El modelo del fondo, es estimado desde un conjunto “ X ” de entrenamiento, el cuál mantiene una historia específica de muestras por cada píxel. Para adecuarse a los posibles cambios que puedan producirse, las muestras en este conjunto se actualizan constantemente con nuevas entradas y simultáneamente las más antiguas son eliminadas. Un periodo de adaptación “ T ” (100 cuadros) es elegido para mantener la historia más reciente del fondo, y en el instante “ t ” se tiene el conjunto de entrenamiento $\mathcal{X}_T = \{x^{(t)}, \dots, x^{(t-T)}\}$, desde el cual se estima la función de densidad del fondo.

La parte central del algoritmo, consiste en actualizar constantemente los parámetros $\vec{\theta}$ de las distribuciones Gaussianas, que describen el estado de cada píxel en un momento determinado. Las ecuaciones recursivas para estimar los parámetros de una muestra nueva $x^{(t)}$ en tiempo t se presentan a continuación:

$$\hat{\pi}_k \leftarrow \hat{\pi}_k + \alpha(o_k^{(t)} - \hat{\pi}_k) - \alpha C_T \quad (3.4)$$

$$\hat{\mu}_k \leftarrow \hat{\mu}_k + o_k^{(t)}(\alpha/\hat{\pi}_k)\vec{\delta}_k \quad (3.5)$$

$$\hat{\sigma}_k^2 \leftarrow \hat{\sigma}_k^2 + o_k^{(t)}(\alpha/\hat{\pi}_k)(\vec{\delta}_k^T \vec{\delta}_k - \hat{\sigma}_k^2) \quad (3.6)$$

$$D_k^2(\vec{x}^{(t)}) = \vec{\delta}_k^T \vec{\delta}_k / \hat{\sigma}_k^2 \quad (3.7)$$

$$\vec{\delta}_k = \vec{x}^{(t)} - \hat{\mu}_k$$

El factor constante α , define una curva de decaimiento exponencial para limitar la influencia de los datos más antiguos. Se usa esta constante, como reemplazo de T mencionado anteriormente, relacionando ambas constante por la siguiente relación: $\alpha = 1/T$.

El algoritmo mantiene un conjunto dinámico de componentes por cada píxel, esto significa que el número de componentes del modelo es diferente en periodos distintos de la secuencia. Se determina que una nueva muestra es cercana con algunos de los componentes si la distancia de *Mahalanobis* (3.7) es por ejemplo, menor que tres. Fijando el valor del *ownership* $o_k^{(t)}$ en uno si

la muestra es cercana y en cero para el caso contrario. Si la muestra no es cercana a alguno de los componentes ($o_k^{(t)} = 0$), se crea uno nuevo con los siguientes valores por defecto $\pi_{k+1} = \alpha$, $\hat{\mu}_{k+1} = \bar{x}^{(t)}$ y $\hat{\sigma}_{k+1} = \sigma_0$, con σ_0 un valor de inicialización de la varianza fijado en el algoritmo.

Este algoritmo representa también los diferentes elementos que surgen en la secuencia, como un conjunto de *clusters* en tiempo real. Un nuevo objeto que ingresa en la secuencia se caracteriza por un *cluster* adicional con un valor de ponderación π_k muy inferior a los otros componentes.

Para determinar el píxel como una imagen de fondo, se ordena los factores de ponderación en forma descendente, y los B *clusters* con mayor ponderación se aproximan al modelo del fondo como se indica en la siguiente ecuación.

$$B = \arg \max_b \left(\sum_{k=1}^b \hat{\pi}_k > (1 - c_f) \right) \quad (3.8)$$

El parámetro c_f es un criterio de medición que indica la cantidad de los datos que pueden ser considerados objetos de *foreground* sin la influencia del modelo de *background*. De esta forma si un nuevo objeto se mantiene estático durante algún tiempo, este será presentado como un *cluster* adicional, y su factor de ponderación π_{k+1} será incrementado en la medida que se mantenga estático. Sí se mantiene lo suficiente, y el factor π_{k+1} supera c_f , entonces este objeto puede ser considerado parte del fondo.

3.4. MODELO MIXTURA DE GAUSSIANAS AUTO-ADAPTATIVO

Este método se origina en un contexto de vigilancia de tráfico en ciudad. Es el resultado de un sistema de detección y clasificación automática de vehículos [8], que utiliza una variante del método de mixtura de Gaussianas[47], para hacer una separación entre vehículos en circulación y su pista. La modificación propuesta aborda el problema de cambios bruscos en la iluminación, que podrían transformar todo el segundo plano (*background*) de la imagen, en primer plano (*foreground*). Además, la tasa normal de aprendizaje se transforma en una tasa dinámica en tiempo real. Se intenta con esta modificación, obtener una tasa que se adapte en forma dinámica a los cambios de iluminación. También se utiliza como método de procesamiento previo, un filtro espacial y temporal[9] para compensar las alteraciones producidas por vibraciones de las camaras. Finalmente para afrontar el problema de cambios en iluminación local, como sombras y reflexiones de luz, modifican un algoritmo de extracción de sombras [25], para incorporar el factor de iluminación global.

3.4.1. Factor de iluminación global

En [7], se menciona el modelo de detección de cambio invariante a iluminación (*ICDM*), al proceso que determina el factor de cambio de iluminación global. Éste es un procedimiento que realiza el cociente uno a uno entre un conjunto “*s*” de pixeles de una imagen de entrada (i_c) y su imagen de referencia (i_r). El factor de iluminación global “*g*” se determina como la mediana de todas las divisiones resultantes, “*Median of Quotient (MofQ)*”.

$$g = \underset{(s \in S)}{\text{median}} \left(\frac{i_{r,s}}{i_{c,s}} \right) \quad (3.9)$$

Se define además un contador “ c_k ” por cada componente Gaussiano en la mixtura. Este contador se utiliza para mantener un registro de cuantos puntos han contribuido en la estimación de parámetros de cada Gaussiana, cada vez que los parámetros de un componente Gaussiano es actualizado, el contador también es incrementado. En caso que un nuevo componente se agrega a la mixtura este contador es inicializado a uno.

El factor de aprendizaje α es usado como base para construir un nuevo factor de aprendizaje β , el cual utiliza el contador de componentes Gaussianas mencionado anteriormente. Las nuevas ecuaciones de actualización de parámetros se detallan a continuación:

$$\beta_k = \alpha(h + c_k)/c_k \quad (3.10)$$

$$\hat{\mu}_k \leftarrow \hat{\mu}_k + o_k^{(t)}(\beta/\hat{\pi}_k)\vec{\delta}_k \quad (3.11)$$

$$\hat{\sigma}_k^2 \leftarrow \hat{\sigma}_k^2 + o_k^{(t)}(\beta/\hat{\pi}_k)(\vec{\delta}_k^T \vec{\delta}_k - \hat{\sigma}_k^2) \quad (3.12)$$

$$c_k \leftarrow c_k + 1 \quad (3.13)$$

$$\hat{\vec{\delta}}_k = g \bullet \vec{x}^t - \hat{\mu}_k \quad (3.14)$$

Si el valor de un pixel de la imagen de fondo varia muy rápidamente, el valor del contador c_k para ese píxel no será muy grande, por lo que el valor de β aumentará. Actualizando de esta manera el fondo más rápidamente. Por otra parte si el fondo es muy estable, el valor de β se aproximará a α . También se observa que la distancia de Mahalanobis es balanceada por el factor de cambio de iluminación global g .

3.4.2. Filtro espacial y temporal

Este filtro que prepara la imagen antes de ser procesada por el método modificado de mixturas. Realiza un suavizado de cada componente espectral. En una imagen se habla de dominio espacial para referirse a los píxeles localizados en la matriz de valores y dominio espectral para los componentes de frecuencia obtenidos después de aplicar la transformada de Fourier. El domino espacial se relaciona con el dominio espectral, por medio del teorema de la *convolución*. La convolución (descripción muy básica), es el proceso mover una mascara o kernel píxel a píxel sobre una imagen y obtener un resultado de cada píxel de acuerdo con los coeficientes de la mascara. Los filtros del tipo Gaussiano, tienen la particularidad que en ambos espacios (espacial y espectral) son representados por una función Gaussiana.

$$K_{h_s, h_t} = \frac{C}{h_s h_t} k \left(\left\| \frac{x^s}{h_s} \right\|^2 \right) k \left(\left\| \frac{x^t}{h_t} \right\|^2 \right) \quad (3.15)$$

C es una constante de normalización, x^s es la parte espacial y x^t la parte temporal, $k(x)$ es un perfil común de kernel (Gaussiano) usado en ambos dominios, temporal y espacial. Las variables h_s y h_t son el ancho de banda de los kernels.

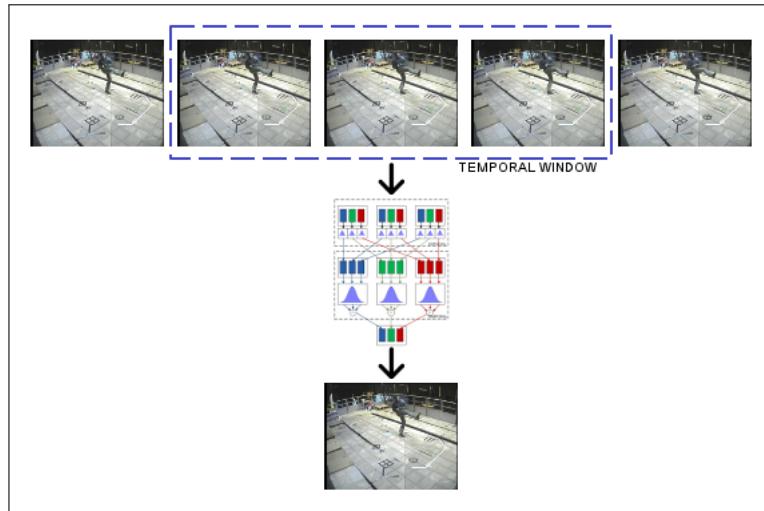


Figura 3.6: Procesamiento temporal de tres imágenes de una secuencia.

La figura 3.6 es un ejemplo del funcionamiento de este filtro. Se define una ventana temporal que define el número de imágenes que serán procesadas simultáneamente por este filtro. La entrada del filtro, es el número de imágenes que mantiene la ventana temporal, y su salida es la imagen filtrada que será la nueva entrada del algoritmo. Esta ventana puede ser visualizada como un marco que se desliza sobre la secuencia y entrega una imagen procesada como salida.

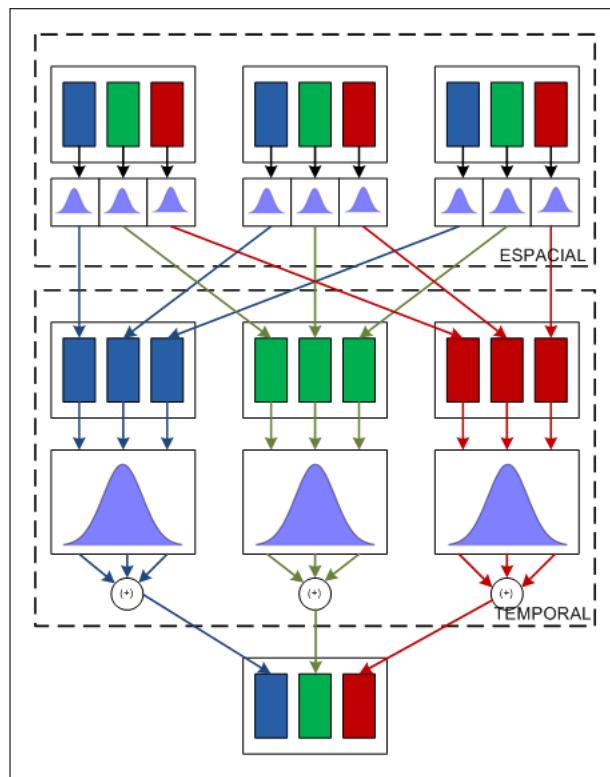


Figura 3.7: Diagrama en bloques del filtro temporal y espacial. La parte de filtro temporal toma tres imágenes y aplica un filtro Gaussiano, el filtro espacial junta los componentes y aplica otro kernel Gaussiano.

Una representación en bloques de la implementación de este filtro se puede notar en la figura 3.7. La primera etapa es el procesamiento espacial, en que aplica un proceso de difuminado (*blurring*) a cada uno de los colores de una imagen. Este procedimiento, es un suavizado Gaussiano que permite reducir el ruido en la imagen de cada uno de los componentes. La segunda etapa, consiste en agrupar por colores el resultado del filtrado espacial, para luego utilizar un segundo filtro Gaussiano, en cada uno de los tres componentes de las imágenes agrupadas por colores. El resultado es una suma de los tres componentes previamente filtrado. Finalmente, cada uno de los componentes se vuelven a juntar en una sola imagen con tres colores filtrados temporalmente.

La efectividad de este filtro puede ser comparado en los gráficos de la figura 3.8. La imagen 3.8(a) es el cluster de la imagen de fondo en funcionamiento normal del algoritmo sin filtro aplicado sobre las imágenes. En cambio la imagen de la figura 3.8(b) es el resultado final del cluster de la imagen de fondo, aplicando procesamiento previo a las imágenes antes de ingresar en el algoritmo. Se puede observar que el cluster final filtrado es mucho más compacto que el otro. Finalmente esta de filtrado es una ayuda en el proceso de separación del fondo de imagen de los elementos móviles.

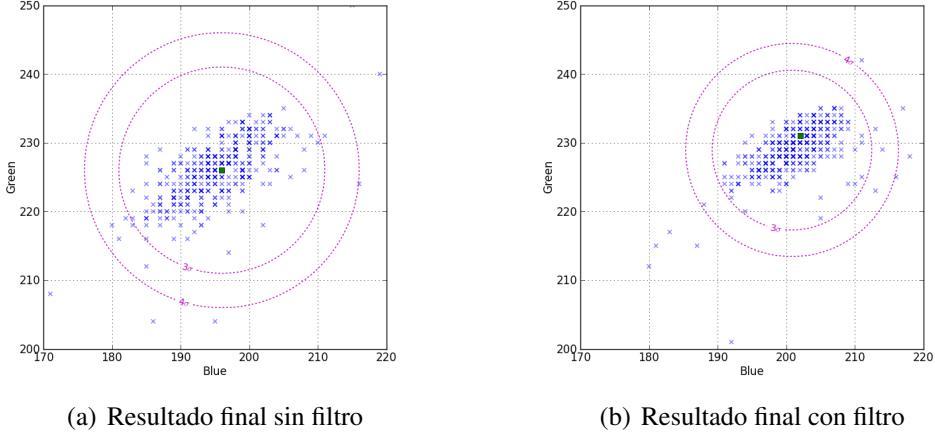


Figura 3.8: Comparación de efectividad del filtro Espacial-Temporal aplicado en la secuencia completa

3.5. MODELO GMM INCLUIDO EN MICRO-CONTROLADORES

Este método [34], denominado en este trabajo tesis como *UCV*, aborda el problema de incluir el modelo *GMM* en un microcontrolador sin capacidad de procesamiento en punto flotante, en microcontroladores de bajo consumo y bajo costo. En este método, al igual que el original, cada pixel de una imagen de fondo se continua representando por una mixtura de G componentes gaussianas, y cada componente por los tres parámetros que describen una gaussiana; media (μ), varianza (σ^2) y un peso (w). Además, se siguen realizando las mismas reglas de actualización definidas previamente en 3.10, 3.11, 3.12. Sin embargo, estas reglas de actualización son re-definidas mediante una operación de *redondeo*. Se define para esto los conceptos de *granularidad* (γ), y pasos de actualización *updating step* (ξ).

$$b = G_ROUND(a, \xi, \gamma) = \begin{cases} \lfloor \frac{a}{\gamma} \rfloor * \gamma + \gamma & \text{if } (a - \lfloor \frac{a}{\gamma} \rfloor * \gamma) \geq \xi \\ \lfloor \frac{a}{\gamma} \rfloor * \gamma & \text{if } (a - \lfloor \frac{a}{\gamma} \rfloor * \gamma) < \xi \end{cases} \quad (3.16)$$

Con *granularidad* γ definida desde un conjunto $B = \{b_i \in N | b_{i+1} - b_i = \gamma\}$, $\xi = \frac{\gamma}{2}$, y a el número a ser redondeado.

$$\mu_{i+1} = G_ROUND(\mu_i + k_i d_i, \xi_\mu, \gamma_\mu) \quad (3.17)$$

$$\sigma_{i+1}^2 = G_ROUND(\sigma_i^2 + k_i(D_i - \sigma_i^2), \xi_\sigma, \gamma_\sigma) \quad (3.18)$$

Con $k_i = \frac{\alpha}{w_i}$, p_i valor del de entrada, $d_i = p_i - \mu_i$, y $D_i = d_i^2$

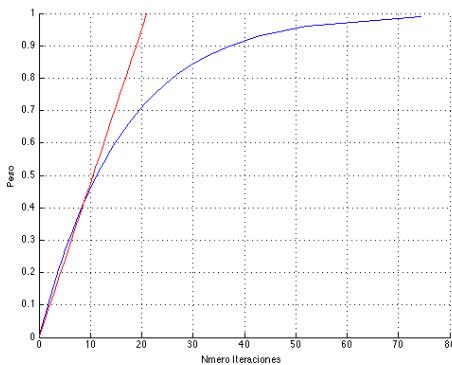
En contraste la regla de actualización de los pesos w de cada componente gaussiano, no depende directamente del valor de entrada del pixel p_i , este depende del número de veces (contador s) que el pixel k pertenezca a una componente gaussiana g . En caso que el pixel no pertenezca a esa componente gaussiana el contador s se decrementa. Se especifica el número de iteraciones necesarias para alcanzar un peso w a través de la siguiente relación:

$$S(w, \alpha) = \frac{\log_{10}(1 - w)}{\log_{10}(1 - \alpha)} \quad (3.19)$$

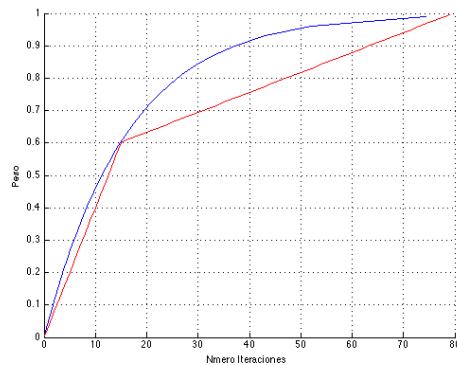
Con α el factor de aprendizaje. La relación anterior simplifica la actualización de los pesos a una operación de incremento y decremento de un contador. Pero esta relación no puede ser implementada en micro-controladores, por el problema de procesamiento en punto flotante. De esta manera se definen aproximaciones lineales como solución para obtener el valor del peso w desde un contador de enteros.

La aproximación lineal (figura 3.9(a)) aproxima la relación definida en 3.19 por una recta que pasa por el origen y un punto $P_0 = (S(w_0, \alpha), w_0)$ con $w_0 = 0,5$, y $m = 2S(w_0, \alpha)$.

$$w(s) = \frac{s}{m}$$



(a) Aproximación Lineal



(b) Aproximación con dos líneas

Figura 3.9: Aproximación de actualización de los pesos.

Las curvas de la figura 3.9 señalan una simplificación de las reglas de actualización del valor w , que determina si una componente Gaussiana pertenece al fondo de la imagen (*Background*) o la silueta (*Foreground*), este valor w es el mismo parámetro $\hat{\pi}_k$ mencionado en la ecuación 3.5. Este método aproxima la actualización del parámetro de peso w (*weight*) haciendo aproximación lineal de las curvas logarítmicas como se señala en las curvas de la figura 3.9. En el primer caso, aproximación con una recta, la linea recta pasa entre el origen y el punto $w_0 = 0,5$ y el contador hace

un decremento o incremento para lograr el valor de peso requerido. En la segunda figura 3.9(b), un ejemplo de aproximación con dos rectas (puede ser más de dos rectas), cada recta aproxima una componente Gaussiana diferente, una con mayor peso que la otra, la primera curva aproxima a una componente gaussiana de menor peso (*Foreground*), y la segunda de mayor peso (*Background*). La figura 3.10 se ilustra una representación (ejemplo tomado desde el trabajo [34]) de dos bytes por componente Gaussiano, con la aproximación de una linea (figura 3.10(a)) y dos rectas (3.10(b)).

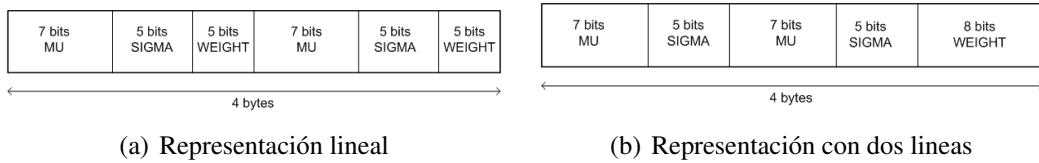


Figura 3.10: Representación de contadores en 4 bytes.

3.6. MODELO NO-PARAMÉTRICO DE FONDO DE IMAGEN

El modelo no-paramétrico de Elgammal et al[13], plantea que cambios muy bruscos del valor de intensidad de un pixel, en periodos muy cortos de tiempo, no es modelado apropiadamente por un pequeño conjunto de distribuciones Gaussianas, resultando un modelo del fondo no muy preciso. Se demuestra que la distribución de intensidad, para una imagen con cambios significativos es del tipo multimodal. Este método propone obtener solamente la información más reciente de una secuencia de imágenes, y actualizar continuamente el modelo del fondo, de esta manera capturar cambios repentinos producidos en una escena.

En aplicaciones de vigilancia exterior, una escena de fondo que se quiera modelar contiene varios elementos no estáticos, como arbustos o ramas de árboles que se mueven con el viento. Estos elementos producen cambios considerables en los valores de intensidad de un pixel. Se genera en consecuencia un modelo de distribución muy ancho, que abarca un amplio espectro de escala de grises, resultando una detección muy pobre, debido que los colores de los elementos móviles se confunden con los colores contenidos en la imagen de fondo.

Cada píxel en el modelo del fondo se representa por una función de densidad de probabilidad $P_r(x_t)$ obtenida de x_1, x_2, \dots, x_N muestras recientes de los valores de un píxel. La probabilidad que el valor de un pixel sea x_t en el tiempo t será estimado de manera no-paramétrica utilizando un estimador kernel K , como se indica en la siguiente relación.

$$P_r(x_t) = \frac{1}{n} \sum_{i=1}^N K(x_t - x_i) \quad (3.20)$$

En esta ecuación x_t es una muestra de un pixel en una imagen nueva que requiere ser clasificado y K es una función kernel del tipo Normal $\mathcal{N}(x|\mu, \Sigma)$, Σ representa el ancho de banda de la función de kernel. Se asume que existe independencia entre los distintos canales de colores, por ende Σ es una matriz escalar y todos los colores tienen la misma varianza, como se explica en el punto 3.3.1, de esta manera la estimación de densidad se reduce a la siguiente ecuación

$$P_r(x_t) = \frac{1}{n} \sum_{i=1}^N \prod_{j=1}^d \frac{1}{\sqrt{2\pi\sigma_j^2}} e^{-\frac{1}{2}\frac{(x_{t_j} - x_{i_j})^2}{\sigma_j^2}} \quad (3.21)$$

Una vez que la densidad de probabilidad ha sido estimada, el píxel x_t es clasificado como imagen de fondo (*background*) u objeto en movimiento (*foreground*) si la el valor estimado de la densidad está arriba o abajo de cierto valor de umbral. Se usa entonces, esta estimación densidad de probabilidad para considerar un pixel como un elemento móvil, sí $P_r x_t < th$. La variable th es un valor global de umbral que puede ser ajustado para lograr un porcentaje específico de falsos positivos.

Existen dos fuentes de variaciones en el valor de intensidad de un pixel, una es debido a estos cambios bruscos de intensidad, que se proyectan en el pixel en diferentes tiempos y variaciones locales de intensidad debido a un efecto borroso (blurring). El ancho de banda de la función kernel (σ) refleja estas variaciones locales de intensidad en la imagen y en el tiempo. La varianza local de cada uno de los canales de colores es calculada como la mediana de la diferencia $(x_1 - x_{i+1})$ dos pares consecutivos del total de la muestra, independiente para canal de color.

$$\sigma = \frac{m}{0,68\sqrt{2}} \quad (3.22)$$

Este método desarrolla además, una segunda de etapa que contribuye en la eliminación de falsos positivos no detectados en la primera etapa del proceso de sustracción de fondo. Estos falsos positivos (denominados falsas detecciones en el trabajo de Elgammal [13]) corresponden por una parte, a ruido aleatorio homogéneo presente en toda una imagen y por otra parte, movimientos que no están representados en el modelo de fondo, como desplazamiento de cámaras o movimientos de ramas de árboles, ambos producidos por la fuerza de viento. Se asume que estos desplazamientos son pequeños entre cuadros de imágenes consecutivos, de esta manera, esto permitiría decidir si un píxel detectado es un elemento del fondo que ha sido movido en una pequeña zona de la región detectada. Se definen dos tipos de probabilidades, la probabilidad de desplazamiento de un pixel $P_N(x_t)$, y la probabilidad de desplazamiento de un componente P_c . P_N corresponde a la máxima probabilidad del valor de un píxel x_t pertenece a la distribución del fondo en algún punto de la

región $N(x)$.

$$P_N(x_t) = \max_{y \in N(x)} P_r(x_t | B_y) \quad (3.23)$$

$$P_c = \prod_{x \in C} P_N(x) \quad (3.24)$$

La probabilidad de desplazamiento de componente P_c se emplea para evitar de detecciones reales sean eliminadas por la primera restricción, se emplea para asegurar que un elemento móvil (*foreground*) se ha desplazado en una zona cercana y no sólo algunos pixeles. Si este componente conectado se desplazado desde la escena del fondo, su probabilidad sería muy baja. Un pixel será, de esta forma, considerado parte de la escena de fondo si cumple con $(P_N(x) > th_1) \wedge (P_c(x) > th_2)$.

Existen dos mecanismos para actualizar el modelo de fondo, con cada nueva muestra de un pixel. En ambos casos cada vez que se agrega una nueva muestra, la muestra más antigua se elimina del conjunto de muestras, para asegurar que la estimación de densidad de probabilidad se base en muestras recientes

- Actualización selectiva: Se agrega una nueva muestra al modelo únicamente si la muestra es un píxel de la imagen de fondo.
- Actualización a ciegas: Una nueva muestra se agrega al modelo, independiente si pertenece o no pertenece a la imagen de fondo.

Junto con estos mecanismos de actualización, el modelo de Elgammal et al[13] mantiene además, dos modelos de fondos: largo y corto plazo

- Modelo de corto plazo, es un modelo reciente de la escena que utiliza las primeras N muestras, utilizando actualización selectiva.
- Modelo de largo plazo, es un modelo que emplea un número mayor de muestras (el total de la ventana de muestras) y por lo tanto contiene una versión más estable de la escena. Las muestras de este modelo se actualizan utilizando el mecanismo de actualización a ciegas.

3.7. RESUMEN

En este capítulo se ha hecho una revisión detallada de los principales algoritmos basados en mixtura de componentes gaussianas (*GMM*). La idea principal de estos algoritmos consiste en

modelar el valor de intensidad de un pixel (o un grupo en el espacio de colores RGB) y construir una función de densidad en el transcurso del tiempo, la densidad se puede estimar mediante una mezcla de componentes Gaussianas. Se discrimina si un pixel pertenece a un objeto en movimiento o la imagen de fondo, verificando el comportamiento de las varianzas de las componentes Gaussianas. Se ha descrito principalmente el modelo de Zivkovic[46] y Chen[8]. El próximo capítulo tratará de las principales métricas de evaluación de desempeño, se hará una descripción de los diferentes métodos de evaluación y las métricas que sirven para esos métodos.

CAPÍTULO 4

MÉTRICAS DE EVALUACIÓN DE RENDIMIENTO

4.1. INTRODUCCIÓN

Se encuentra en la literatura un conjunto heterogéneo de métodos de evaluación, que sirven para valorar la calidad de segmentación de un algoritmo. Pero, no existe acuerdo entre los investigadores para usar una metodología de evaluación general, que pueda proporcionar un criterio de comparación común, de medición del desempeño de los algoritmos de separación entre *Foreground* y *Background*. Asimismo, se han puesto a disposición de la comunidad una gran cantidad de escenarios de evaluación ("datasets"), entre estos [36], que establecen una serie de condiciones (ambientales, iluminación, etc) bajo las cuales, los algoritmos pueden experimentar diversas situaciones que les permitiría contrastar y evaluar su comportamiento en diferentes situaciones.

Estos métodos han sido clasificados en la literatura y dependiendo de la naturaleza del evaluador, en diversas metodologías que establecen dos categorías principales; *método de evaluación subjetiva* [31] y *método de evaluación objetiva*. Este capítulo presenta una descripción de los variados enfoques y métricas empleados en la actualidad para hacer evaluación de calidad, centrando el análisis en el criterio denominado evaluación objetiva.

4.2. ENFOQUES DE EVALUACIÓN

Las metodologías de evaluación *subjetivas*, se basan en la valoración que hace un grupo de observadores, al resultado final de un proceso de segmentación. No obstante, requiere un gran número de evaluaciones, para producir un resultado con relevancia estadística. En contraste con el método anterior, las metodologías denominadas *objetivas*, buscan una forma de evaluación automática que

no requiera la intervención humana en el proceso de medición. En general, el resultado final es una mixtura de métricas, que elaboran un ranking de la calidad de segmentación en una imagen o secuencia de vídeo. Igualmente, el tipo de métricas empleadas en la evaluación objetiva, se dividen en métricas *analíticas* y *empíricas*. Las métricas analíticas requieren conocimiento del algoritmo, para evaluar sus requerimientos, complejidad, y estructura. Las métricas del tipo empíricas en cambio, miden calidad del resultado final de un proceso de segmentación.

Los métodos de evaluación del tipo empíricos se dividen a su vez, en métodos de *discrepancia empírica* o *evaluación relativa*, y métodos de *evaluación autónomos*. Los métodos de discrepancia, se basan en medición de disparidad entre una imagen con anotaciones manuales o de referencia, “ground truth”, y el resultado de segmentación. Las imágenes con anotaciones manuales funcionan como base de comparación, son máscaras binarias que etiqueta un pixel como background o foreground y tienen la finalidad de proporcionar un conjunto de datos independiente y objetivo, con el cual un resultado obtenido puede ser comparado y medido. Pero, estas máscaras son construidas manualmente, por lo que tienen en forma inherente un cierto grado de error e incertidumbre, que influye en la clasificación final como el mejor resultado que puede ser obtenido. Los métodos autónomos, en cambio, son métodos del tipo *no-supervisados* [45] donde no requieren una imagen de referencia, evalúan el grado de igualdad de un conjunto de características de imágenes segmentadas previamente determinadas por evaluadores humanos.

Los métodos de evaluación más empleados son los basados en discrepancia empírica, los diferentes modelos propuestos en la literatura plantean una medición objetiva tratando de representar la percepción humana. Las distintas aproximaciones hacen una ponderación del resultado final de acuerdo con la visibilidad de los errores.

4.3. MÉTODOS DE EVALUACIÓN DE CALIDAD

Esta parte se enfoca principalmente en métodos de evaluación de calidad basados en *discrepancia empírica*, métodos que hacen una evaluación de percepción de los resultados. En los siguientes párrafos se hace una descripción y análisis de las métricas más recurrentes encontradas en la literatura para hacer evaluación de clasificación de objetos.

4.3.1. Error Cuadrático Medio y Relación a Señal Ruido

Una de las formas más simples de medir calidad es el *error cuadrático medio* (Median Squared Error - MSE) y la *relación señal a ruido* (Peak Signal-to-Noise - PSNR) [18] [39], el cual hace un promedio de el cuadrado de las diferencias de intensidad entre los pixeles de la imagen obtenida

y su par de referencia, *ground-truth*. A pesar, que es un método simple de calcular y promediar, ambas mediciones no tienen buena correlación con las medidas de percepción humana. En [41] se demuestra mediante ejemplos, que *MSE* no es una buen indicador para medir la fidelidad y calidad de una imagen. Se constatan valores similares de *MSE* en varias imágenes distorsionadas artificialmente (con respecto a una imagen original), no obstante las imágenes presentan calidades visuales muy diferentes. También se hace notar en imágenes con pequeñas modificaciones geométricas, imperceptibles desde el punto de la calidad de percepción, valores de *MSE* muy grandes.

$$MSE = \frac{1}{N} \sum_{i=1}^N (x_i - y_i)^2 \quad (4.1)$$

$$PSNR = 10 \log_{10} \frac{L^2}{MSE} \quad (4.2)$$

Donde N es el número de píxeles de la imagen, x_i e y_i es el pixel i de la imagen de referencia y resultante, respectivamente, y L es el rango dinámico del valor de los píxeles. Para 8 bits este valor es 255.

4.3.2. Métricas de Calidad Estática

Estas métricas de calidad, son elaboradas realizando la comparación individual de un pixel de la imagen resultante, con el pixel equivalente de su imagen de referencia o *ground-truth*. Es un procedimiento de comparación pixel-pixel, que entrega como resultado una serie de mediciones base, usadas posteriormente para construir estas métricas de calidad. La imagen de la figura 4.1(a) es la máscara de una silueta después de haber sido separada de su imagen de fondo. Esta máscara señala en diferentes colores los resultados de comparación con su imagen de referencia (figura 4.1(b)). El número *TP* (*verdaderos positivo*) de esta imagen, corresponde a los píxeles de la silueta (*foreground*) correctamente clasificados. *TN* (*verdadero negativo*) es el número de píxeles de la imagen de fondo (*background*) seleccionados correctamente. *FP* (*falso positivo*) es el número de píxeles pertenecientes a la imagen de fondo seleccionados incorrectamente como píxeles perteneciente a la silueta. *FN* (*falso negativo*) corresponde a un número de píxeles de la silueta incorrectamente clasificados como imagen de fondo. El resultado de estas mediciones se colocan en una matriz de confusión (conocida también como tabla de contingencia) para formar la base de todas métricas de evaluación de calidad, como se indica en figura 4.2.

Una condición positiva de la imagen de referencia en esta tabla, se refiere a todos los píxeles que circunscriben la silueta (*foreground*) que un algoritmo debiera separar del fondo de imagen, y la condición negativa son todos los píxeles que están en el fondo de la imagen. En otro aspecto,

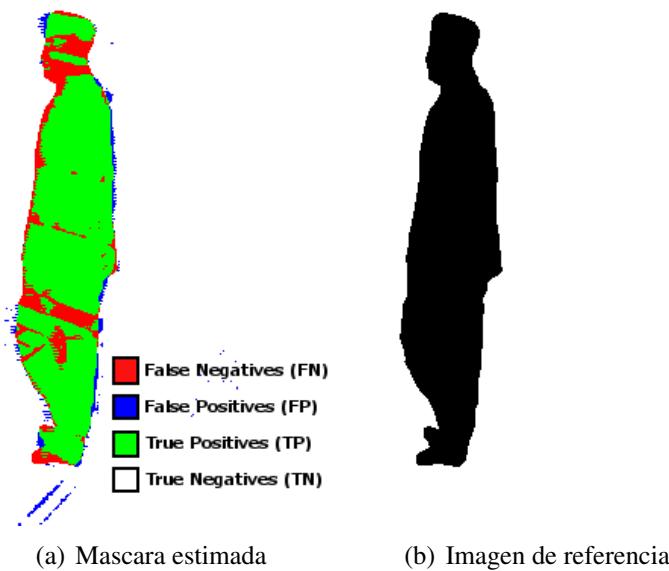


Figura 4.1: Métricas de calidad estática, imagen de referencia y mascara estimada por algoritmo

la condición positiva en el resultado de la separación (mascara) son todos los píxeles que fueron clasificados pertenecientes a la silueta, y la condición negativa son todos los píxeles clasificados en la imagen de fondo. Los números localizados en la diagonal principal de esta matriz representan decisión correcta (TP y TN) y los números de diagonal opuesta (FN y FP) son los errores (“confusión”) realizados durante la clasificación, en este caso son los errores del algoritmo al equivocar píxeles de la imagen de fondo con píxeles de la silueta y píxeles de una silueta con los de la imagen de fondo.

$$Precision = \frac{\text{Total píxeles en foreground clasificados correctamente}}{\text{Total píxeles imagen resultante}}$$

$$Recall = \frac{\text{Total píxeles foreground clasificados correctamente}}{\text{Total foreground píxeles de imagen referencia}}$$

$$TruePositiveRate = \frac{\text{Positivos (silueta) clasificados correctamente}}{\text{Total positivos}}$$

$$FalseNegativeRate = \frac{\text{Negativos (background) clasificados incorrectamente}}{\text{Total negativos}}$$

		Imagen de Referencia	
		Positivo (Foreground)	Negativo (Background)
Mascara Resultado	Positivo	TP	FP
	Negativo	FN	TN
Total		TP+FN	FP+TN

Figura 4.2: Matriz de Confusión

A partir de esta matriz de contingencia, se determinan una serie de métricas comunes de calidad, que se utilizan en los algoritmos de clasificación. La **tasa verdaderos positivos** (*true positive rate*) es una medida de proporción de los positivos (píxeles de la silueta) seleccionados correctamente, esta métrica también es conocida como *recall* o *sensitivity*. La **tasa falsos positivos** (*false positive rate*) o tasa de falsas alarmas, corresponden a la proporción de negativos (píxeles *background*) clasificados incorrectamente. Una medida complementaria a la tasa de falsos positivos es la métrica de *specificity*, el cual es una proporción de negativos (*background*) clasificados correctamente. En sistemas de recuperación de información (*Information Retrieval*) las medidas de *precision* (proporción de píxeles seleccionados correctamente), *recall* y media armónica *F-Measure* [5] [24] [39] (medida de desempeño total) son muy utilizadas como métricas de evaluación de rendimiento, esta medición esta entre el rango 0, 1. En el caso de separación de foreground y background, un valor alto de *F-measure* es un indicador para una mejor segmentación del foreground.

$$Precision = \frac{TP}{TP + FP} \quad (4.3)$$

$$True\ Positive\ Rate = \frac{TP}{TP + FN} \quad (4.4)$$

$$False\ Positive\ Rate = \frac{FP}{FP + TN} \quad (4.5)$$

$$sensitivity = recall$$

$$specificity = 1 - fpr$$

$$F - Measure = \frac{1}{\frac{1}{precision} + \frac{1}{recall}} \quad (4.6)$$

Las mediciones de la matriz de confusión se pueden relacionar también, con un indicador de correlación denominado *Matthew Correlation Coefficient - MCC*. Ésta es una medida de calidad de clasificación de dos clases diferentes. Es un coeficiente de correlación entre la referencia y el resultado obtenido (observado). El rango de valores para este coeficiente, varía entre $-1, 1$ ($MCC \in [-1, 1]$). Un resultado con valor 1 indica un resultado perfecto, el 0 representa un valor no mejor que un resultado aleatorio, y -1 es una total diferencia entre los obtenido y la referencia.

$$MCC = \frac{TPXTN - FPXFN}{\sqrt[2]{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \quad (4.7)$$

4.3.3. Curva de operaciones característica

Los algoritmos de sustracción de imágenes de fondo o separación de *foreground* y *background*, son métodos que clasifican un píxel perteneciente a la imagen de fondo o a un objeto que está en movimiento, como una silueta del actor (*foreground*). La curva de operaciones características (*receiver operating characteristics - ROC*) es un gráfico que permite visualizar en forma global el desempeño de clasificación de un algoritmo. Un punto en esta curva relaciona la respuesta de clasificación, en este caso proporción de píxeles clasificados correctamente, con la proporción de píxeles negativos incorrectos. Describe un balance relativo entre beneficios (*verdaderos positivos*) y costos (*falsos positivos*). La curva se construye con el resultado de varias operaciones distintas, utilizando el mismo conjunto de datos, pero modificando algún parámetro característico de este algoritmo. Esto permite comparar el desempeño con diferentes configuraciones de un algoritmo.

La figura 4.3 es una curva ROC de ejemplo, que muestra el resultado de dos algoritmos diferentes. El punto $(0,0)$ de la esquina inferior izquierda representa una estrategia de nunca producir una clasificación positiva, un algoritmo localizado en este punto no genera errores de falso positivo, pero tampoco obtiene verdadero positivos. El punto $(1,1)$ es opuesto al anterior, siempre tiene clasificaciones positivas. El punto $(0,1)$ es el resultado de clasificación perfecta, los píxeles en *foreground* fueron detectados completamente, sin errores en los píxeles de la imagen de fondo. Un punto en el espacio ROC es mejor que otro, si esta localizado más cerca de la esquina superior izquierda que el otro punto; el algoritmo presenta una tasa de verdaderos positivos alta y una tasa

de falsos positivos baja. Algoritmos que se ubican en la parte izquierda del gráfico, cercano al eje X, se pueden considerar “convervadores”, hacen una clasificación positiva teniendo buena evidencia, tienen muy poco errores de falso positivo, pero tienen también una baja tasa de verdaderos positivos. Al contrario, algoritmos que se ubican en la parte superior derecha de la gráfica, son algoritmos más libres, hacen clasificaciones positivas con poca evidencia, clasificando todos los positivos correctamente, pero con una alta tasa de falsos positivos.

La recta $y = x$ representa una estrategia de estimación aleatoria, en algoritmos de clasificación binaria. Cualquier punto en esta recta, implica el mismo valor de clasificación para ambas clases. Por ejemplo, un algoritmo que hace una estimación aleatoria de obtener una silueta la mitad del tiempo, se puede esperar que obtenga la mitad de silueta y el fondo de imagen en forma correcta, ubicando esta clasificación en el punto (0.5, 0.5) de la gráfica ROC. Sí se estima por ejemplo que la estimación de la silueta es positiva el 90 % del tiempo, se puede esperar conseguir un 90 % de tasa de verdaderos positivos, pero con un incremento de la tasa de falsos negativos a un 90 % también.

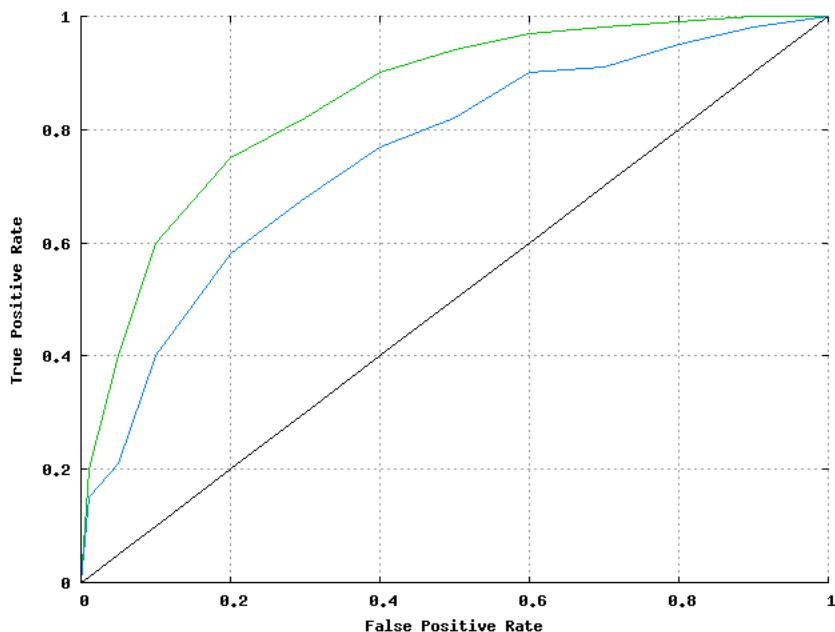


Figura 4.3: Curva de operaciones características

4.3.4. Métricas de Percepción

Estas métricas intentan ponderar el concepto de percepción de un observador, sobre el resultado final de segmentación. Se propone cuantificar los conceptos de *exactitud espacial* (spatial accuracy) y *estabilidad temporal* [28] [6] o *coherencia temporal* [40]. Ponderan, mediante algún

tipo de función definida, la discrepancia entre una mascara estimada y la de su referencia (ground-truth), tomando en cuenta la inexactitud potencial que puede contener la mascara de referencia, la localización de los pixeles con errores relativo al borde de la mascara de referencia, y el tipo de errores (FP, FN).

4.3.4.1. Contexto espacial y temporal

El contexto espacial [6] de un pixel errado, es caracterizado por la distancia más cercana de un objeto detectado. Esta propiedad relaciona el foco de atención de un observador hacia a un objeto que le llama la atención, errores del tipo falso negativo (pixel clasificado erróneamente como foreground) son más importantes en la medida que la distancia es mayor. El contexto temporal considera las diferencias de calidad en el tiempo, para esto identifican dos tipos de fenómenos; un efecto sorpresa y un efecto fatiga. El efecto sorpresa se relaciona con los cambios en la exactitud espacial y el efecto fatiga se relaciona con el hecho de un observador se acostumbra a cierto tipo de calidad en el tiempo. Estos dos contextos son relacionados en las siguientes expresiones.

$$\nu_w(n) = 1 - \epsilon_w(n) \quad (4.8)$$

$$\zeta_w(n) = \frac{\nu_w(n)}{2} [1 + \alpha_t \frac{d}{dn} \nu(n)] \quad (4.9)$$

Siendo $\nu_w(n)$ un valor de ponderación de la exactitud espacial, $\frac{d}{dn} \nu(n)$ variación espacial en el tiempo (mientras mayor esta variación más pequeño su coherencia temporal) y $\zeta_w(n)$ una medida de calidad de percepción espacial-temporal. Se define una *exactitud espacial absoluta y relativa* (normalización por el total de pixeles) como la contribución de las ponderaciones de w_p^i (falso positivo), y w_n^j (falso negativo) .

$$w_p^i = \frac{\alpha_p \log(d_p^i + 1)}{D} \quad (4.10)$$

$$w_n^j = \frac{\alpha_n d_n^j}{D} \quad (4.11)$$

$$\epsilon_w() = \sum_{i=1}^{\epsilon_p(n)} w_p^i + \sum_{j=1}^{\epsilon_n(n)} w_n^j \quad (4.12)$$

Como se indica en la figura 2, ambas ponderaciones o pesos w_p^i y w_n^j aumentan con la distancia, pero influyen en forma diferenciada en la ponderación final de exactitud espacial. Los pesos para

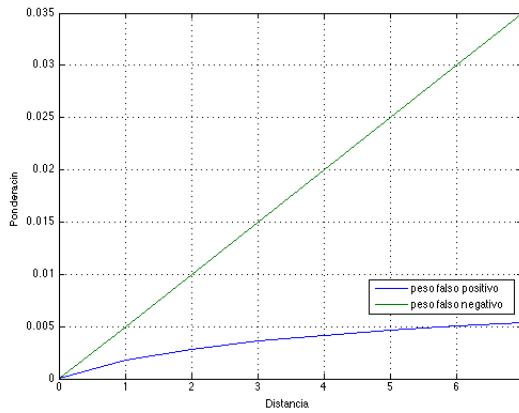


Figura 4.4: Pesos pixeles falso positivo y negativo

falsos positivos se incrementan en forma lineal, y estos valores son mayores con los pesos de los falsos negativos a la misma distancia.

4.3.4.2. *Observaciones para cuantificación espacial*

Las mediciones de exactitud espacial del trabajo propuesto en [28], se basan en observaciones que toman en consideración, el error producido al generar las máscaras de referencia en forma manual. Se menciona que los pixeles errados, muy cerca de los bordes de las máscaras de referencia, se presumen causados por la inexactitud durante la creación de estas máscaras. A su vez, una segunda observación indica que estos mismos errores muy cercanos al borde, tienen poco efecto comparado con otro que se encuentra muy lejano del borde. Pese a que estos pixeles en el borde hacen que un objeto parezca más grueso o delgado, estos no cambian la forma del objeto. Una tercera observación, señala que el impacto de los diferentes tipos de errores durante el procesamiento no son similares, por lo que hacen las distinción de medir en forma independiente falsos negativos y falsos positivos.

Se propone una función ponderación del tipo sigmoidal, de esta forma errores muy cercanos al borde de la máscara tienen un valor (peso) menor a uno y otros errores lejanos al borde se aproximan a uno.

$$S(d; \alpha) = \frac{1 - e^{-\alpha \cdot d}}{1 + e^{-\alpha \cdot d}} \quad (4.13)$$

Esta función S , tiene como variable d distancia de un pixel en error al borde de la máscara, y α es parámetro que modela la tolerancia de la inexactitud de las máscaras de referencia en sus bordes. Un parámetro α pequeño conlleva una capacidad de tolerancia mayor.

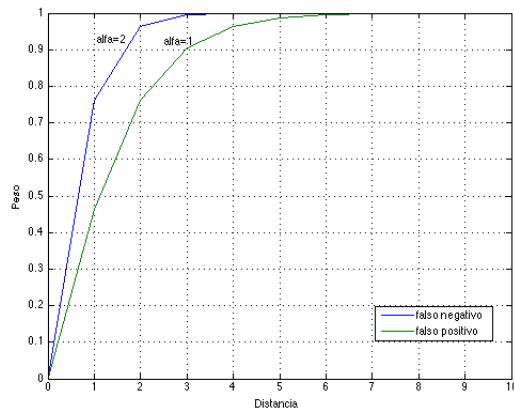


Figura 4.5: Pesos usando función tipo sigmoidal

4.3.5. Similaridad Estructural

El índice de *similaridad estructural - SSIM* (*Structural SIMilarity*) [39] [42] es un método de medición de similaridad entre dos imágenes. Es una medida de calidad de una imagen de entrada contrastada con otra de buena calidad. Este índice usa un nuevo enfoque para medir calidad, se basa en la hipótesis que la principal función del ojo humano es extraer información estructural de su campo visual, y además el Sistema Visual Humano (*Human Visual System - HVS*) está muy adaptado para realizar esta operación. Por lo tanto una medición de distorsión estructural podría ser una buena aproximación. Se plantean entonces, en este nuevo enfoque, moverse desde mediciones de errores (métodos vistos anteriormente) a realizar mediciones de distorsión estructural. Sin embargo, el nuevo problema que se debe enfrentar es cómo definir y cuantificar distorsiones estructurales.

El modo de medir y obtener el índice *SSIM* supone una secuencia de imágenes de entrada en escala de grises además de la secuencia de imágenes de referencia, ground-truth. Se considera la secuencia de referencia como la secuencia con las imágenes de buena calidad, de esta forma la medida de similaridad sirve como una medición de la calidad de la otra secuencia de entrada. La manera de medir similaridad es separar la tarea en tres componentes; *luminancia*, *contraste*, y *estructura*.

$$SSIM(x, y) = f(l(x, y), c(x, y), s(x, y)) \quad (4.14)$$

$$SSIM(S, G) = \left(\frac{2\mu_S\mu_G + c_1}{\mu_S^2 + \mu_G^2 + c_1} \right) \cdot \left(\frac{2\sigma_S\sigma_G + c_2}{\sigma_S^2\sigma_G^2 + c_2} \right) \cdot \left(\frac{\mu_{SG} + c_3}{\mu_S\mu_G + c_3} \right) \quad (4.15)$$

Este índice de calidad es una combinación de tres factores, pérdida de correlación, distorsión

media, y varianza de la distorsión. El primer componente es el coeficiente de correlación lineal entre S y G y su rango dinámico varia en $[-1, 1]$. El segundo componente mide similaridad entre valores medios, y su rango de valores está entre $[0, 1]$. El tercer componente mide similaridad entre las varianzas de ambas imágenes de una secuencia, y su rango dinámico está entre $[0, 1]$. La relación final aplicada sobre secuencia de vídeo viene dada por la siguiente formula.

$$SSIM(S, G) = \frac{1}{n} \sum_{i=1}^n \frac{(2\mu_{S_i}\mu_{G_i} + c_1) \cdot (2\text{cov}_{S_iG_i} + c_2)}{(\mu_{S_i}^2 + \mu_{G_i}^2 + c_1) \cdot (\sigma_{S_i}^2 + \sigma_{G_i}^2 + c_2)} \quad (4.16)$$

Donde, S es un conjunto de n imágenes resultantes, G conjunto de las imágenes de referencia (ground-truth), μ_{S_i} μ_{G_i} media de ambas secuencias, σ_{S_i} y σ_{G_i} desviaciones estándar, y $\text{cov}_{S_iG_i}$ covarianza de S_i y G_i . Las constantes $c_1 = (k_1 + L)^2$ $c_2 = (k_2 + L)^2$ con valores $k_1 = 0,01$ $k_2 = 0,03$ y $L = 255$.

4.3.6. D-Score

D-Score brinda un criterio de *disimilitud* entre una imagen de referencia y la imagen de entrada, esta métrica sólo considera fallas en la medición de los resultados. El costo de un error depende de la distancia con el objeto más cercano de la referencia, teniendo una ponderación mayor los errores en el rango intermedio. Esto debido que este tipo de errores impactan de mayor manera el reconocimiento de formas. Por ende, errores en zonas lejanas/cercanas tendrán un D-score cercano a cero.

$$DT(x) = \min(d(p, x), \forall x \in X) \quad (4.17)$$

El costo de error se basa en una *Transformada de Distancia* (Distant Transform - DT) de la referencia ground-truth, con X un conjunto de pixeles de referencia, $d(p, x)$ es la distancia del pixel p al pixel x , entonces $DT(x)$ es la distancia mínima entre un pixel x y el punto de referencia más cercano. Para calcular $DT(x)$ se usa la siguiente relación

$$D-score(x) = \exp(-(\log_2(2 \cdot DT(x)) - \alpha)^2) \quad (4.18)$$

De acuerdo con la figura 4, se tiene una tolerancia de 3 pixeles desde la referencia, esto debido que ese tipo de errores locales no afecta realmente el proceso de reconocimiento. De esta misma forma tomando un valor de $\alpha = 5/2$, [39], se permiten que ocurran errores a más de 10 pixeles de distancia. D-score entrega un índice que señala el nivel de reconocimiento de objetos, en la medida que este valor es menor, mejor es el reconocimiento, independiente del valor de errores que

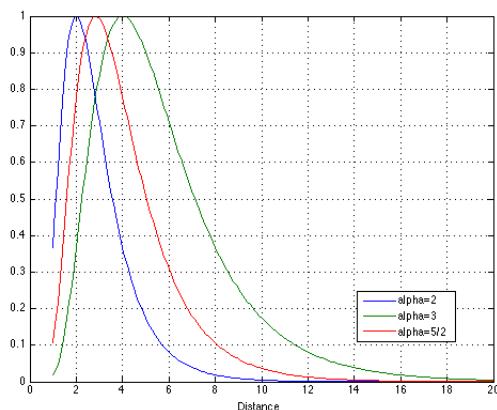


Figura 4.6: Valores D-Score basados en la distancia desde la referencia ground-truth

entreguen otros índices, como el *F-Measure*.

4.4. RESUMEN DE MÉTODOS DE EVALUACIÓN

Se ha hecho una descripción de la mayoría de los métodos de medición utilizados actualmente, basados principalmente en el paradigma de evaluación por discrepancia o error. Una menor discrepancia con una referencia, brindan un ranking de mayor calidad de clasificación o comparación de algoritmos. En general, no existe una métrica ideal, que pueda ser utilizada como herramienta de comparación universal, entre los diferentes algoritmos existentes. De igual modo, estas métricas pueden ser empleadas en conjunto, de manera complementaria, aprovechando las diferentes características de ellas. Un caso concreto es el trabajo desarrollado para el desafío BMC [39].

Las métricas más empleadas sin duda son “*Specificity*” ($1 - fp\ rate$), “*Sensitivity*” (recall) y todas sus demás variantes, estas métricas proporcionan una medida general del resultado de clasificación. Presentan el compromiso entre la habilidad del algoritmo para identificar los píxeles correspondientes a una silueta (*sensitivity*), y la habilidad del mismo algoritmo para identificar los píxeles pertenecientes a la imagen de fondo (*specificity*). Estas dos métricas se colocan en una curva de ROC que permite comparar desempeño con otro algoritmo bajo las mismas condiciones.

Las diferentes métodos basadas en el concepto de percepción se aproximan a una forma de evaluación que podría hacer un observador humano, para esto, hacen una combinación de un contexto espacial con un contexto temporal. Los errores se ponderan de acuerdo con distancia al borde de la mascara de referencia, la estabilidad de esta percepción espacial es además considerada en la evaluación de calidad. Los distintos enfoques encontrados de esta metodología se diferencian en el tipo de función empleada para hacer la ponderación espacial de los errores con respecto a la

distancia del borde de su mascara. Evalúan con mayor ponderación el error falso negativo y localizado lejano al borde de la mascara. La métrica de “D-Score”, a diferencia de la descripción previa castiga fuertemente los pixeles errados en el borde de la mascara, porque este tipo de errores no permiten realizar una buena clasificación.

Un diferente enfoque presenta el índice de similaridad, este ya no utiliza las mediciones de percepción para evaluar calidad. Postula medición de distorsión estructural, para esto se propone una combinación de mediciones de luminancia, contraste y estructura. Que vienen siendo análisis de correlación entre dos imágenes, cuantificación de sus varianzas y sus medias.

Finalmente, estas distintas métricas pueden ser empleadas en conjunto o en forma individual. Pero una combinación de ellas podría brindar una mejor claridad desde el punto de vista de la calidad de los diferentes algoritmos. Pueden entregar una mejor discriminación de las ventajas y falencias en diversas condiciones ambientales. Se podría determinar un buen algoritmo para ciertas condiciones, pero también determinar sus desventajas en otras situaciones.

CAPÍTULO 5

DESCRIPCIÓN DEL SISTEMA DE SOFTWARE

5.1. INTRODUCCIÓN

En el capítulo 3 se ha hecho una revisión de los algoritmos que hacen separación de una silueta de su imagen de fondo (*Background Subtraction*), específicamente en algoritmos basados en mixtura de componentes gaussianos (*Gaussian Mixture Model*). Posteriormente, en el capítulo 4 se hizo una descripción de las principales métricas utilizadas en la comunidad de investigación de visión por computador, para hacer evaluación de rendimiento de estos algoritmos. El siguiente paso, es colocar ambos elementos en el contexto de este trabajo, en una herramienta de software disponible para la comunidad de investigación, que permita incluir nuevos desarrollos y sea de utilidad para hacer evaluación de algoritmos. Se requiere entonces articular ambas áreas en una herramienta de software que permita utilizar uno de estos algoritmos sobre el conjunto de datos MuHAVI, y emplear las métricas de rendimiento para obtener una idea global de desempeño del algoritmo seleccionado. Este capítulo se enfoca en el proceso de desarrollo de software que finaliza con la implementación de ambos elementos (algoritmos y métricas de desempeño) en dos módulos que constituyen el producto final de software, y cumple con los requerimientos planteados en el capítulo 1. Se hace una descripción de las distintas etapas metodológicas involucradas en el desarrollo del sistema de software, se discute el diseño y la estructura de software propuesto, así como las principales herramientas (*OpenCV*) escogidas para realizar el desarrollo.

5.2. CAPTURA DE REQUERIMIENTOS

Se plantea en el comienzo del proyecto la idea de aprovechar las características de MuHAVI [36] como conjunto de datos de experimentación, para ser utilizada con el método de *Foreground Detection* propuesto por Zezhi Chen[8], un algoritmo orientado a realizar seguimiento y detección de vehículos, el cual podría ser aplicado para generar y colocar a disposición de la comunidad de investigación, todas las siluetas generadas a partir este método. En una segunda instancia se propone comparar las siluetas manuales (*ground-truth*) que tiene MuHAVI con los resultados de este método y otros métodos adicionales similares. Una última parte implementar algún método de detección de actividad y hacer un estudio comparativo.

De este plan inicial y después de varias revisiones, surgen los objetivos del proyecto registrados en el capítulo 1, y que sirven de base de los requerimientos funcionales capturados para el desarrollo de la implementación de esta herramienta, que se describen a continuación.

- Desarrollar un software como sistema base, para incorporar algoritmos que utilicen métodos de separación de siluetas de su imagen de fondo (*Foreground Detection* o *Background Subtraction*)
- Implementar el algoritmo *SAGMM*[8] (*Self-Adaptive Gaussian Mixture Model*) de separación de la siluetas de su imagen del fondo e incorporarlo en el sistema base.
- Implementar una herramienta de software que consolide las métricas escogidas para realizar una evaluación global de desempeño del resultado de estos algoritmos.

Con los requerimientos definidos se establecen dos escenarios de casos de usos, realizado por dos actores: un investigador que ejecuta y verifica los algoritmos, y la comunidad de investigación (otro actor investigador) que utiliza la herramienta con las métricas y evalúa resultados. Ambos casos de uso se ilustran en la figura 5.1. En este primer prototipo se dejan fuera varios escenarios, como los casos de fallas y el procedimiento de compilar el programa ejecutable para agregar un nuevo algoritmo en el sistema principal.

- **Caso de uso 1:** Ejecutar algoritmo para verificación.

Investigador agrega nuevo algoritmo desarrollado en la comunidad en el sistema de software para utilizarlo y verificarlos con el conjunto de datos MuHAVI.

1. Investigador modifica archivos de configuración para incluir algoritmo en el sistema
2. Investigador ejecuta programa principal con base datos MuHAVI

3. Investigador verifica siluetas obtenidas después del procesamiento
- **Caso de uso2:** Obtener desempeño general de algoritmo
 1. Investigador obtiene siluetas resultantes de un procesamiento anterior.
 2. Investigador obtiene imágenes de referencia *ground-truth* provenientes de MuHAVI.
 3. Investigador ejecuta herramientas de medición de desempeño.

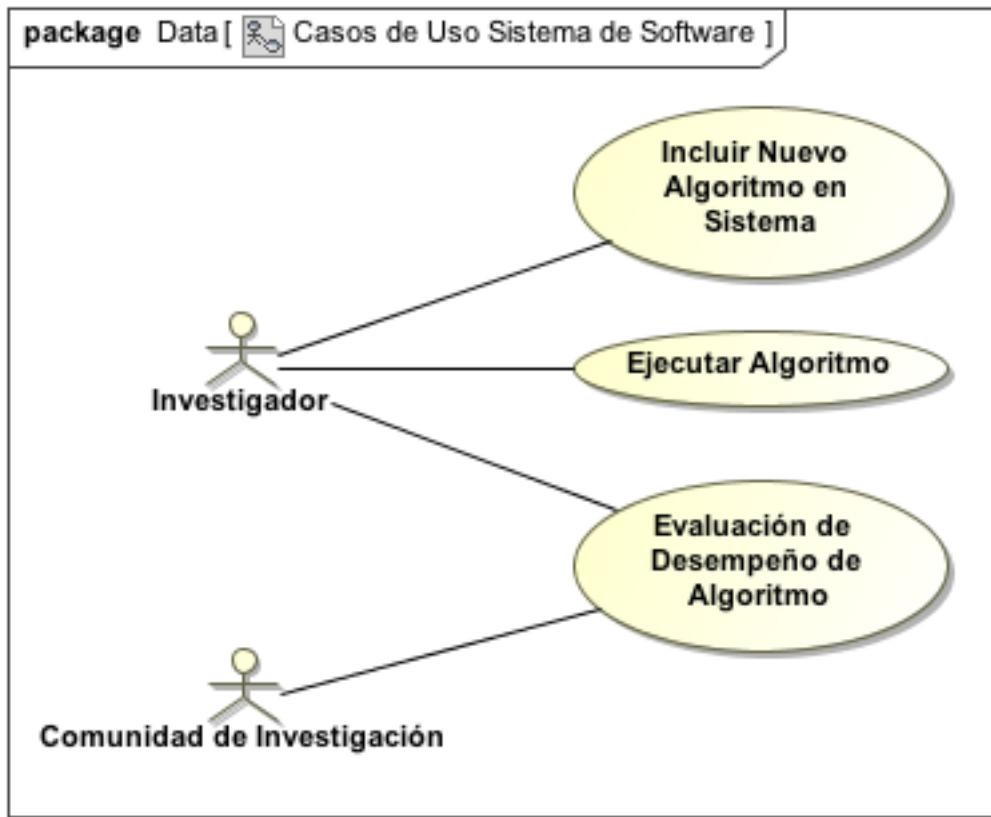


Figura 5.1: Casos de uso de los requerimientos establecidos en el proyecto. El investigador agrega y utiliza un algoritmo desde el sistema. El investigador usa la herramienta de métricas para obtener evaluación de desempeño y los coloca a disposición de la comunidad (la comunidad de investigación usa los resultados obtenidos).

5.3. ARQUITECTURA DE SISTEMA DE SOFTWARE

El diseño de la arquitectura de software está basada en módulos independientes constituida en una estructura de tres capas; herramientas base, núcleo del software, y herramientas de software.

Cada módulo dentro de una capa, es una unidad funcional independiente, que suministra bibliotecas de clases para construir los programas o clases de los módulos en las capa superiores. Asimismo, los módulos de un mismo nivel pueden utilizar las bibliotecas de sus vecinos, como el caso, de los programas utilitarios ubicados en la capa del medio. La imagen de la figura 5.2 es una vista a nivel de módulos, de la arquitectura de software diseñada para esta solución, una descripción más detallada por módulo se realiza en la siguiente sección. Los módulos de la parte inferior de esta imagen (en color naranja pálido) corresponden a todas las biblioteca de software externas, utilizadas para construir este software. Los módulos localizados en la parte media y superior, son los distintos componentes implementados en este proyecto.

El diseño del software en módulos se basa principalmente en la independencia de los requerimientos. Cada módulo es una unidad de implementación independiente que se relaciona con los requisitos establecidos. El sistema base para incluir algoritmos y la herramienta de métricas, son unidades que están definidas para funcionar independiente y no está dentro de los requerimientos que funcionen en forma simultánea. Esta estructura además permite mejorar tareas de mantenimiento, depuración y arreglo de problemas, porque permite enfocarse directamente en el módulo que requiere mejorarse y no perturba el funcionamiento del otro módulo.

Descripción de los módulos señalados en el diagrama en la figura 5.2.

- **Herramientas de desarrollo:** es el módulo que proporciona compiladores, programas de configuración para construir la solución de software.
- **OpenCV:** es un conjunto de bibliotecas C++ de software abierto (*Open Source*) usadas en visión por computador.
- **Biblioteca Boost:** Utilizada para el manejo de caracteres y *strings* en C++, expresiones regulares y control de opciones de entrada en programa principal.
- **Sistema Base:** Parte principal del software, éste realiza integración de algoritmos de separación *foreground* y *background*.
- **Medición de desempeño:** Esta parte incorpora todas las métricas que posibilitan hacer evaluación de performance.
- **Programas Utilitarios:** Conjunto de clases y programas básicos que sirven de apoyo a los otros módulos.
- **Algoritmos:** Esta módulo proporciona las clases que encapsulan los diferentes algoritmos, que se intenta evaluar.

- **Scripts de Análisis:** Programas desarrollados en lenguaje de programación *Python* que ayudan en la creación de gráficas de desempeños, empleados para hacer análisis de los datos resultantes.
- **Aplicaciones:** Esta parte corresponde a los clientes que hacen uso de las herramientas proporcionadas por este sistema de software.

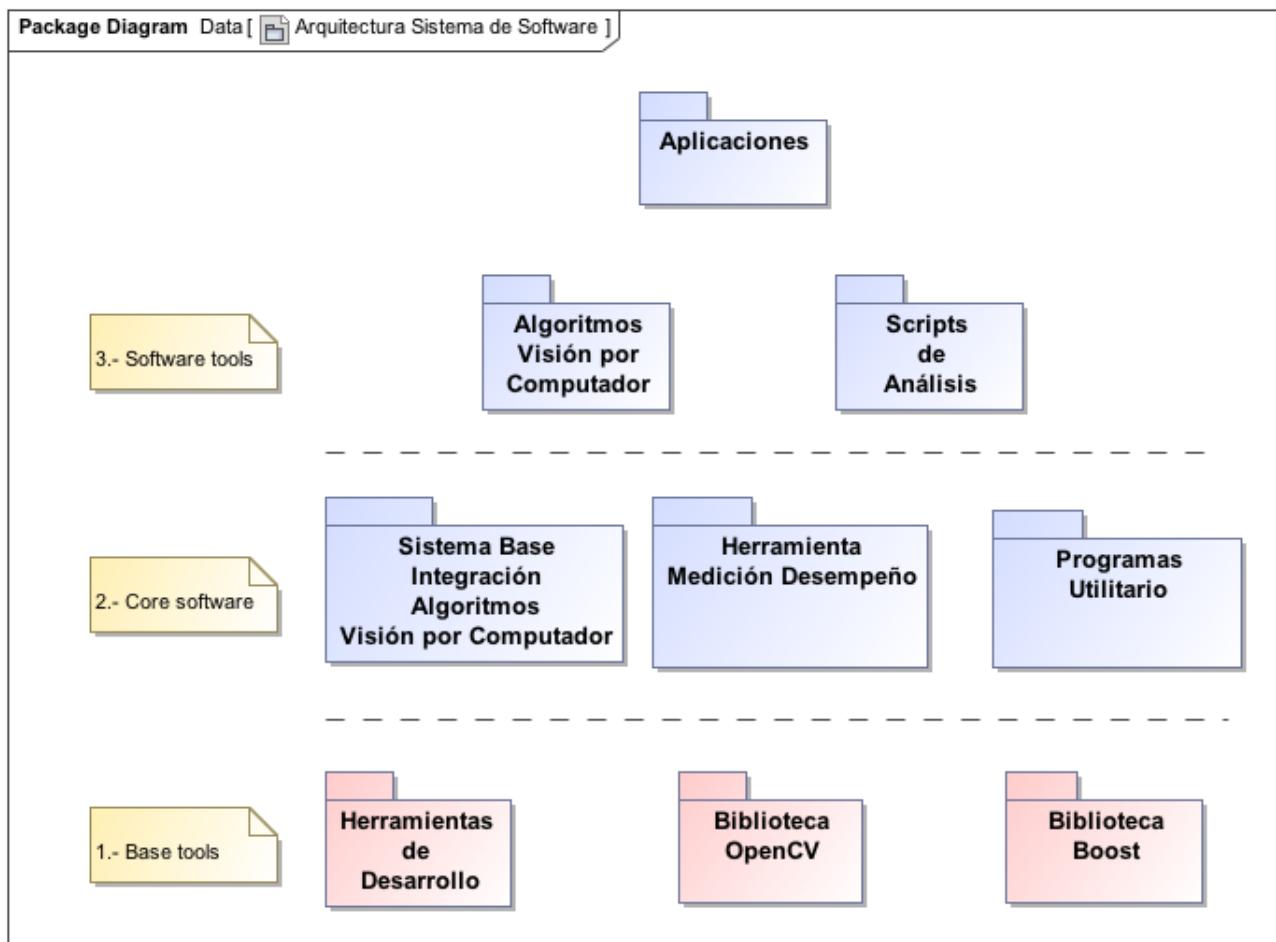


Figura 5.2: Diagrama de la arquitectura de software. Se definen tres niveles de capas, la capa inferior suministras las herramientas base (clases, bibliotecas, archivos de configuración) para construir las capas superiores. La capa del medio comprende distintos módulos que constituyen el núcleo del sistema de software, en esta capa se coloca la implementación de la herramienta de métricas, el sistema base que incorpora los algoritmos que se intenta evaluar, y programas utilitarios. La última capa contiene los módulos con las bibliotecas de los algoritmos y programas de *scripting* que ayudan en el análisis de las métricas de rendimiento.

5.4. HERRAMIENTAS DE SOPORTE BASE

5.4.1. Herramientas de desarrollo

El desarrollo de los diferentes módulos de software ha sido realizado en una distribución Linux, **Ubuntu 12.04 LTS precise**. Ubuntu es un sistema operativo Linux, basado en Debian, que incluye todas las herramientas necesarias, para hacer la implementación de los paquetes de software de esta solución. Abajo se especifican las principales herramientas involucradas en el de desarrollo.

- **Compilador:** GNU GCC versión 4.6
- **CMake:** Grupo de herramientas, *Open Source* independiente del sistema operativo, diseñado para construir, validar paquetes de software. Se ha usado esencialmente para generar los archivos *Makefile* nativos al sistema operativo Ubuntu. La utilidad de CMake, radica en la independencia de la plataforma sobre la cual se está desarrollando. De esta manera, en caso de ser necesario, los módulos de software también pueden ser compilados y ejecutados en otros ambientes (*MAC OS X*, otros “distros” de *Linux*, o *Windows*).
- **Servidor de repositorios GitHub**[20]: Mantiene control de cambios de versión de los distintos módulos. Este servicio de repositorios público, se basa en *GIT* un sistema de control de versiones distribuidos. Las principales características, entre otras, es un sistema de control de los datos, almacena el estado de un proyecto a nivel de directorios y archivos. Crea un registro completo (*snapshot*) del repositorio que esta siendo almacenado, esto facilita las tareas de recuperación, creación de ramas de desarrollo (*branch*) y unión (*merge*) de diferentes versiones de un módulo. El proyecto completo puede ser localizado en el siguiente URL: <https://github.com/jorgesep/BGS>.

5.4.2. Biblioteca OpenCV de Visión por Computador

OpenCV[26] (*Open Source Computer Vision Library*) es una biblioteca especializada en visión por computador, facilita una API (*Application Programming Interface*) con una serie de módulos necesarios para construir una aplicación en visión por computador. El módulo principal, define las estructuras básicas, sobre la cual se desarrollan muchas de las aplicaciones, incluido los módulos de software construidos en este proyecto de tesis. OpenCV define, un grupo de tipos básicos (estructuras) que constituyen la base para construir y manipular los distintos elementos que se presentan en visión por Computador. Se establece un arreglo de varias dimensiones, denominado *Mat* (figura 5.3), que posibilita contener en forma nativa la estructura de una imagen. Esta estructura, permite

realizar todas las operaciones matriciales básicas para manipular imágenes. Incluye también, un modulo de procesamiento de imágenes (filtros lineales y no-lineales, transformación geométrica de imágenes, conversión de espacio de colores, histogramas, entre otros). Un módulo de análisis de videos, con varios algoritmos, entre ellos el algoritmo (*Background Subtraction*) original de Zivkovic y Heijden [47] evaluado en este trabajo. Se agregan además, algoritmos de calibración, detectores de características, detección de objetos, una interfaz (*gui*) para la captura de imágenes de video, *codecs* de imágenes. También incluye un módulo para trabajar con GPU.

La estructura de datos *Mat*, es una clase C++ de n-dimensiones, constituida en dos partes. Un encabezado, con información de tamaño, método utilizado para almacenar, dirección de localización de memoria de los datos, y otros tipo de información relacionada con el tipo de datos. Mantiene también, un puntero a la zona de memoria donde se encuentra la matriz de píxeles, y su dimensión depende del método empleado para almacenar los datos. El encabezado de la matriz es constante, pero su tamaño varía dependiendo del tipo de imagen que esta siendo manipulada. La figura 5.3 es un ejemplo de esta matriz, la zona demarcada por el rectángulo en rojo, es una zona de interés que se desea estudiar. La zona es una matriz de números manejada por una estructura *Mat*, en el caso de una imagen definida en el espacio RGB, se almacenan los valores de cada píxel, en forma contigua, en el orden: blue, green, y red. Las líneas de códigos que se indican a continuación, es un ejemplo de la facilidad de uso de la estructura de datos *Mat*.

```

1 Mat A, C;                                // creates just the header parts
2 A = imread("File.jpg", CV_LOAD_IMAGE_COLOR); // allocate matrix
3 Mat B(A);                                 // Use the copy constructor
4 C = A;

```

Código 5.1: Ejemplo de uso estructura MAT

La zona es una matriz de números manejada por una estructura *Mat*, en el caso de una imagen definida en el espacio RGB, se almacenan los valores de cada píxel, en forma contigua, en el orden: blue, green, y red.

5.4.3. Biblioteca Boost

Boost [2] se ha usado principalmente para evitar el problema que se encuentra al manipular archivos, operar con *strings* de caracteres en C++. *Boost* tiene varias bibliotecas utilitarias, y en este proyecto se han usado las que permiten manejar expresiones regulares, búsqueda de caracteres en una linea de *strings*, reemplazo de caracteres, manipulación de archivos.(*boost filesystem*, *boost system*, *boost program_options*, *boost reg_exp*).

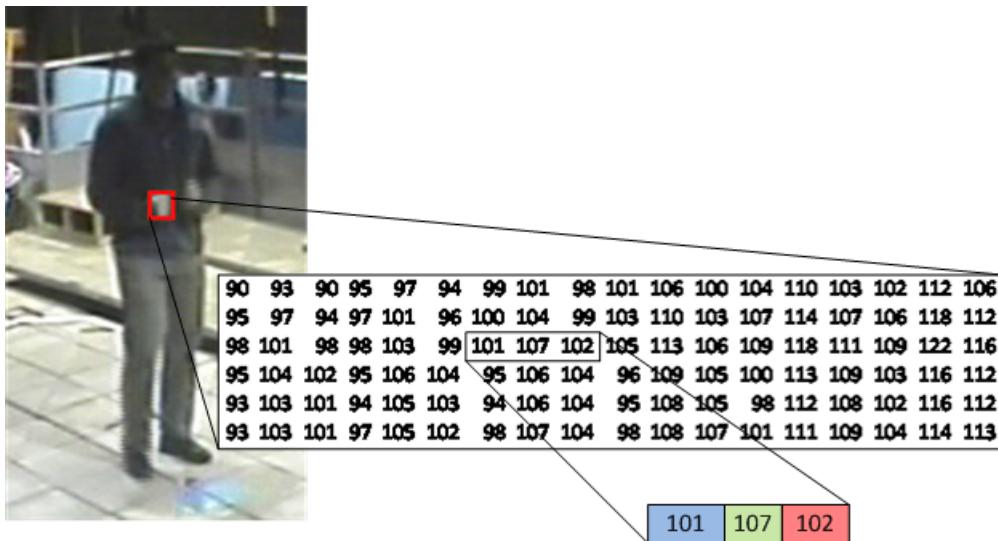


Figura 5.3: La estructura de datos *OpenCV Mat* es un arreglo dividido en dos partes. Un encabezado, con información de tamaño, método utilizado para almacenar, dirección de localización de memoria de los datos. Un puntero a la zona de memoria donde se localiza la matriz de pixeles; números almacenados en forma contigua en el orden *blue, green, red*, en el caso del espacio de colores RGB.

5.5. DESCRIPCIÓN DE IMPLEMENTACIÓN

Se elige *C++* como lenguaje de programación para la implementación del sistema de software, y lenguaje de programación en *Python* para el desarrollo de *Scripts* que manipulan los datos de resultado y gráficas de desempeño. Los criterios de selección empleados para estos lenguajes de programación se listan a continuación.

- *OpenCV* está optimizado para trabajar en *C++*
- *C++* es un lenguaje de programación con orientación objetos. Se elige programación orientada a objetos por las características de *encapsulación* y *polimorfismo* que ofrece este tipo de programación. Estas características son aprovechadas por las clases que implementan el sistema para encapsular los diferentes algoritmos que se desea verificar.
- Los gráficos de desempeño fueron realizados en *Matplotlib*[30], una biblioteca basada en *Python* para construir gráficos.
- Experiencia de programación en ambos lenguajes

5.5.1. Sistema Base Integración Algoritmos de Visión por Computador

Para abordar la implementación del sistema base, asimismo la herramienta de medición de métricas, se escogió el concepto de *Diseño de Patrones*[19] como estructura base de las clases que implementan ambas soluciones. *Diseño de Patrones* (*Design Patterns*[19]) es un conjunto de soluciones orientada a objetos de problemas repetitivos, que aprovechan la experiencia para enfrentar un problema conocido, re-usando una solución comprobada. En la literatura se los encuentra dividido en tres grupos, dependiendo de la relación entre clases y objetos que se usa.

- **Patrones Estructurales** (Structural Patterns): Son Patrones que se usan para formar grandes estructuras de objetos y clases. Facilitan el trabajo de modificar estructuras de jerarquías de clases y sus asociaciones. Este tipo de patrones usa herencia para hacer composición de interfaces o implementaciones. Por ejemplo una clase que mezcla dos clases (múltiple herencia) en una clase, ésta clase combina las propiedades de ambas clases padres en una nueva clase independiente. De esta forma, este tipo de Patrones describe formas de componer objetos para agregar nuevas funcionalidades.
- **Patrones Creacionales** (Creational Patterns): Son Patrones que tienen el propósito de facilitar el trabajo de creación (*instanciación*), inicialización y configuración de objetos y clases. Independizan a un sistema de lo forma como los objetos son creados, compuestos, y representados. Se les denomina de esta manera porque crean cosas, como clases, implementación de interfaces, atributos, etc.
- **Patrones Comportamiento** (Behavioral Patterns): Son Patrones orientados a la interacción y comunicación entre objetos, utilizan herencia para distribuir el comportamiento entre clases. Se usan para trabajar con algoritmos y la comunicación entre clases. Por ejemplo, un algoritmo usado por una clase es sólo parámetro que puede ser ajustado en tiempo de ejecución.

El desafío en diseño de patrones es localizar el *Patrón* que se ajuste al problema que se quiere solucionar. Para esta implementación se escoge la alternativa de los *Patrones de Comportamiento*, interesa seleccionar el tipo de algoritmo (*comportamiento*) antes de la ejecución del programa principal, del sistema de software. Es tipo de patrones permite definir una clase interfaz para exponer sólo los métodos (funcionalidades) necesarios para el funcionamiento del sistema de software. Dentro de esta categoría se estudiaron dos tipos de patrones que podrían haber sido adecuados para la implementación: Método de Plantilla (*Template Method*), y Estrategia (*Strategy*), escogiendo este último como el más adecuado.

El patrón del método plantilla, define una clase base de algoritmo (*template*) y a partir de esa base se pueden crear sub-clases (clases hijos) que deriven las características particulares (comportamiento) del algoritmo que se desea exponer. El patrón de estrategia define un conjunto encapsulado de algoritmos que pueden ser intercambiados de acuerdo con alguna configuración en tiempo de ejecución, es decir se definen algoritmos en clases independientes y se elige alguno de ellos (selección de comportamiento) dependiendo de la estrategia en tiempo de ejecución. Se escoge el patrón de estrategia por la característica de seleccionar en tiempo de ejecución alguno de los algoritmos incorporados dentro del sistema. En cambio el método plantilla se requiere seleccionar los algoritmos a utilizar en tiempo de compilación.

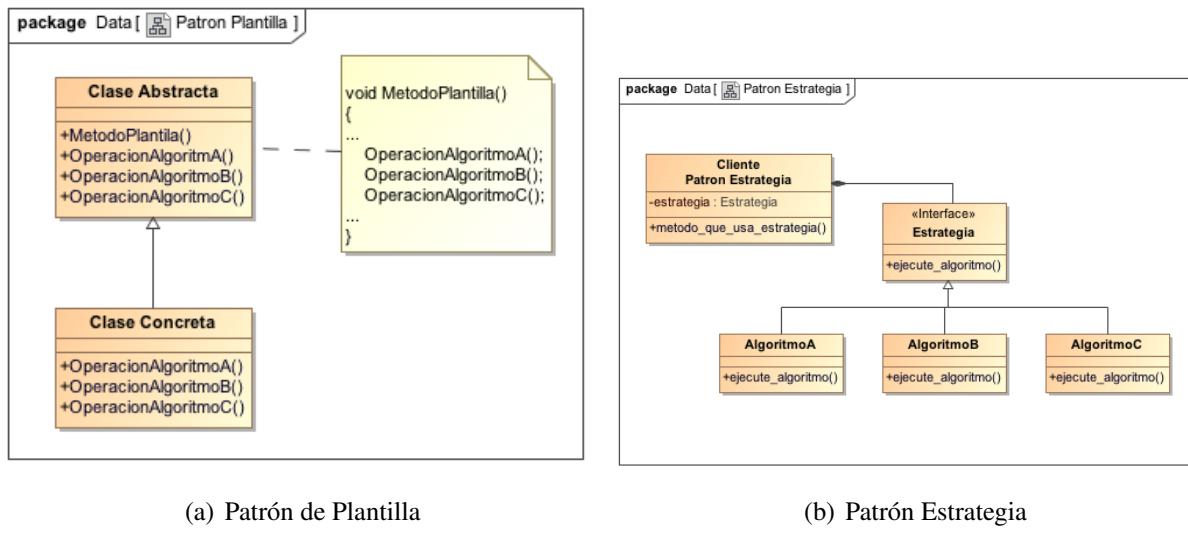


Figura 5.4: Diagrama UML de los patrones estudiados para implementación del sistema base. El método plantilla selecciona mediante sub-clases (herencia) en tiempo de compilación, el patrón de estrategia selecciona un algoritmo en tiempo de ejecución.

5.5.2. Diagrama de clases de implementación

El patrón de diseño estrategia encapsula los algoritmos para hacerlos intercambiables, al ser invocados por algún cliente en tiempo de ejecución. Se basa en la idea de encapsular el comportamiento de la parte que cambia. Aprovecha la propiedad de “*polimorfismo*” en programación orientada a objetos, para definir una interfaz común, sobre la cual los diferentes “*comportamientos*” (algoritmos) son implementados. La figura 5.5 ilustra el diagrama de clases de la implementación para el sistema base.

- **Interfaz IBGSAlgorithm:** Es una clase abstracta de C++ que representa las funcionalidades principales de los algoritmos que se desean encapsular, para esto, se definen los principales

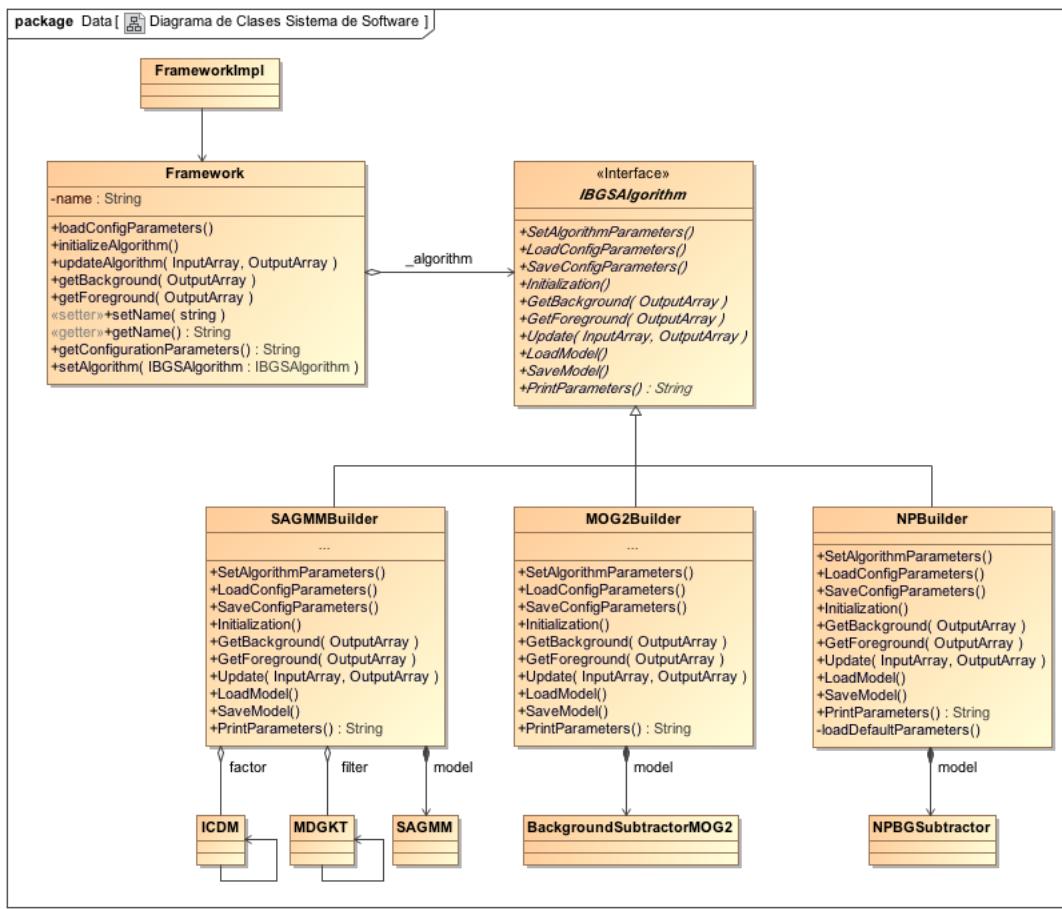


Figura 5.5: Diagrama UML de las distintas clases del sistema base. Se define una clase de tipo interfaz (**IBGSAAlgorithm**) que define y expone las principales funcionalidades de los algoritmos. Las clases **<Algoritmo>Builder** se encargan de encapsular los algoritmos haciendo una composición de ellos. La clase **Framework** implementa cada uno de los métodos de la clase **IBGSAAlgorithm** para que estos puedan ser usados por la implementación principal **FrameworkImpl**.

métodos (funcionalidades) que la clase cliente debe llamar para usar un algoritmo.

- **Clase <ALGORITMO>Builder:** Es una clase concreta que integra un algoritmo en su definición, mediante *composición* de un objeto algoritmo, esto significa que la clase *Builder* es responsable de la creación y destrucción de las partes del objeto algoritmo. Esta clase hace una abstracción de cada algoritmo exponiendo únicamente las operaciones esenciales que se necesitan en la clase principal del sistema base. En la figura 5.5 se ilustran las clases **SAGMMBuilder**, **MOG2Builder**, **NPBuilder**, las cuales corresponden a los algoritmos que se incluyen en el sistema base.
- **Clase Framework:** Es la clase principal que implementa cada uno de los métodos definidos

en la clase interfaz *IBGSFramework*.

- **Programa Principal *FrameworkImpl*:** Es la implementación del programa ejecutable, el cual crea las instancias de todos los algoritmos integrados en el sistema. Abajo se lista una parte de código ejemplo (Código 5.2) que declara e inicializa los distintos objetos algoritmos, del tipo de clase *Framework*.

```

1 { ...
2     Framework* mog2 = new Framework();
3     mog2->setAlgorithm(new MOG2Builder());
4     mog2->setName("MOG2");
5     mog2->loadConfigParameters();
6     mog2->initializeAlgorithm();
7
8     Framework* np = new Framework();
9     np->setAlgorithm(new NPBuilder(col, row, nch));
10    np->setName("NP");
11    np->loadConfigParameters();
12    np->initializeAlgorithm();
13
14    Framework *sagmm = new Framework();
15    sagmm->setAlgorithm(new SAGMMBuilder());
16    sagmm->setName("SAGMM");
17    sagmm->loadConfigParameters();
18    sagmm->initializeAlgorithm();
19    ...
20 }
```

Código 5.2: Ejemplo de instanciación distintos objetos de algoritmos en programa principal. Cada algoritmo en este ejemplo, se declara como un tipo de la clase *Framework* y luego se utilizan los métodos que configuran ese algoritmo.

El diagrama de secuencia en la figura 5.6 es un ejemplo de los mensajes que son intercambiados entre un cliente (*FrameworkImpl*), la clase *Framework* y algún algoritmo definido en el sistema (*SAGMM*, *MOG2*, etc.). El cliente inicia enviando un mensaje al objeto *Framework* para indicar el tipo de algoritmo a utilizar, método *setAlgorithm*. Luego se realiza una configuración de parámetros del algoritmo que requiere para su funcionamiento (método *LoadConfigParameters*). La configuración de un algoritmo se realiza por medio de un archivo XML, para esto se aprovecha una funcionalidad de *OpenCV* para leer archivos de configuración en formato XML. El método *initializeAlgorithm* se encarga de finalizar la configuración del algoritmo (con los parámetros leidos desde

el archivo XML). Una vez que los pasos previos han finalizado, el cliente está en condiciones de leer las imágenes (máscaras) que al algoritmo están procesando.

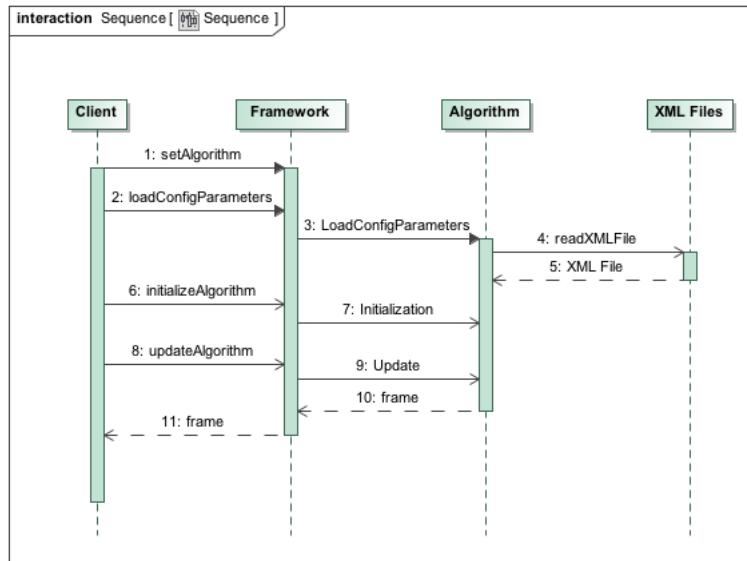


Figura 5.6: Diagrama de secuencia de ejecución de un algoritmo

5.5.3. Herramienta de evaluación de desempeño

Se basa principalmente de una clase denominada “*Performance*” (ver diagrama UML de figura 5.7), compuesta de una serie de operaciones (métodos) encargados de realizar los diferentes cálculos de desempeños. Se han dispuesto una serie de métodos públicos (*getters*), que permiten acceder a un valor de atributo privado dentro de la clase, para obtener el valor de las distintas métricas. Asimismo, se han definido un grupo de estructuras que permite agrupar y almacenar las diferentes métricas por tipo de medida, en la medida que van realizando las comparaciones entre una imagen y su referencia.

- ***Similarity Struct***: Contiene las métricas de similaridad: *DScore*, *MSSIM*, y relación señal a ruido *PSNR*. Estas métricas son obtenidas a nivel *frames*, es decir, la comparación se realiza entre dos imágenes.
- ***ContingencyMatrix Struct***: Mantiene los datos resultante de la matriz de confusión. Son métricas obtenidas por comparación a nivel de pixel, de una imagen de entrada con su imagen de referencia.

- **CommonMetrics Struct:** Es una estructura que mantiene las métricas comunes, de *sensibilidad* y *especificidad*.
- **StatMetrics Struct:** Estructura compuesta que almacena los valores de media y mediana de las métricas comunes.
- **GlobalMetrics Struct:** es una estructura compuesta que almacena un sumario de todas las métricas.

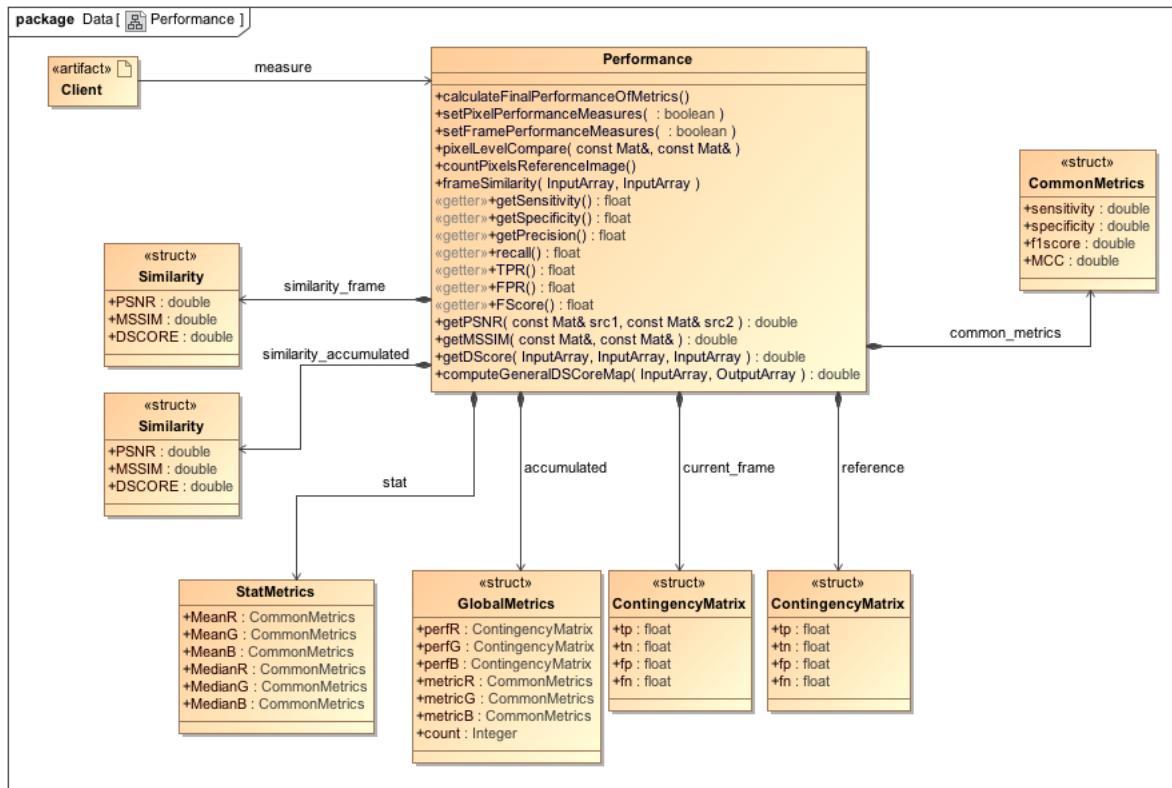


Figura 5.7: Diagrama UML herramienta de evaluación

5.5.4. Operación de la implementación

En la imagen de la figura 5.8 se muestra un esquema de funcionamiento de la implementación de las clases del sistema base en dos programas ejecutables. De manera informativa se muestra ambos programas funcionando de manera combinada, pero el funcionamiento de ambos es totalmente independiente. El programa ejecutable *bgs_framework*, bloque localizado en la parte superior de la figura 5.8, es una implementación de la clase *Framework*, el programa recibe de entrada una secuencia de imágenes desde el conjunto de datos MuHAVI, lee desde un archivo XML (localizado

en un directorio local donde se ejecuta la aplicación) y genera un conjunto de imágenes resultantes (máscaras de siluetas) que van siendo almacenadas en un directorio local a medida que se ejecuta la aplicación.

En una segunda parte, los resultados son comparados con las imágenes de referencia (*ground-truth*) que proporciona MuHAVI, mediante el programa *pmbgs*. Esta aplicación es una implementación que contiene un objeto de la clase *Performance*, es decir declara este objeto y luego lo define, y posteriormente utiliza los métodos que proporciona esta clase para ir leyendo las distintas métricas resultantes de la comparación que se está realizando.

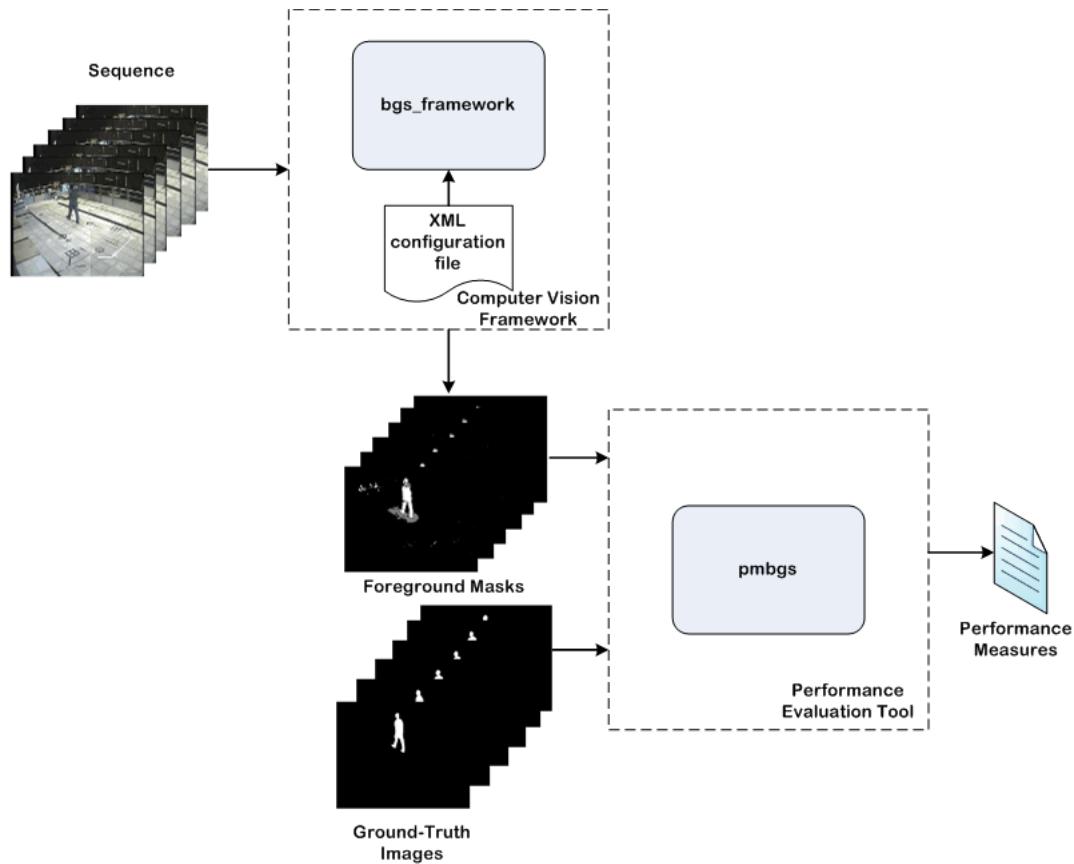


Figura 5.8: Diagrama en bloques que señala la operación del sistema base de los algoritmos y la herramienta de evaluación de desempeño, sólo de manera informativa se muestran ambos módulos funcionando en forma combinada. El sistema base de algoritmos obtiene parámetros de configuración de un archivo XML y procesa una secuencia de imágenes (MuHAVI). La herramienta de evaluación, posteriormente *off-line*) compara las imágenes resultantes del procesamiento del algoritmo con las imágenes de referencia (ground-truth) que proporciona MuHAVI, generando como resultado un archivo de texto con las métricas de desempeño por secuencia.

5.6. RESUMEN

Este capítulo hace una descripción de las distintas actividades realizadas en la implementación del sistema de software de este proyecto, así como también una descripción de las diferentes clases C++ desarrolladas, a nivel de diagrama UML, que conforman el sistema base de algoritmos y mediciones de desempeño del proyecto. Sin una idea previa, para utilizar alguna metodología conocida de desarrollo de software, todo el avance dentro del proyecto, las diferentes etapas que se fueron ejecutando y finalizando dan cuenta de una metodología muy parecida (sino igual) a un modelo del tipo *Waterfall*. Todas las actividades siguieron una linea secuencial, que se inició con la captura de los requerimientos, luego continuó con el diseño de la solución, posteriormente la implementación, para finalizar con la puesta en marcha, verificación y mantenimiento (solución de problemas encontrados). Afortunadamente, desde el punto de vista de esta metodología, los requerimientos fueron constante y no hubo variación de ellos durante el transcurso del proyecto. Uno de los problemas críticos dentro de esta metodología, consiste exactamente en el cambio de los requerimientos, cualquier modificación de estos, requiere una nueva revisión desde el comienzo, lo que en definitiva significa alargar los plazos del proyecto. La lógica de secuencia de las secciones dentro de este capítulo describen en cierta forma el proceso de desarrollo seguido en la etapa de implementación.

Con respecto a las diferentes decisiones de implementación, muchas de ellas fueron tomadas basadas en experiencias anteriores, que ayudarían a rebajar los tiempos de implementación. Ese es el caso por ejemplo, de utilizar lenguaje de programación C++ y no otro similar como *Java*, también por compatibilidad con la plataforma de visión por computador *OpenCV*, el cual en su mayor parte está desarrollado en C++, pero es compatible con otros lenguajes de programación. Con respecto a la utilización de patrones de diseño, esta decisión sigue la lógica de reutilizar código y experiencias documentadas, además de asegurar un código más robusto (problemas de punteros, asignación de memoria, *bugs*). Sin embargo, utilizar patrones requirió tiempo para escoger el más adecuado, mismo tiempo que a lo mejor se podría haber utilizado para desarrollar desde cero. En síntesis, siempre queda una incertidumbre cuál alternativa escoger.

Finalmente el sistema de software fue implementado y verificado con extensas pruebas sobre el conjunto de datos MuHAVI, el siguiente capítulo se dedica exclusivamente a revisar los resultados de las métricas de evaluación obtenidas con la herramienta de evaluación implementada al ser utilizada con las imágenes resultantes del sistema de algoritmos.

CAPÍTULO 6

EXPERIMENTACIÓN

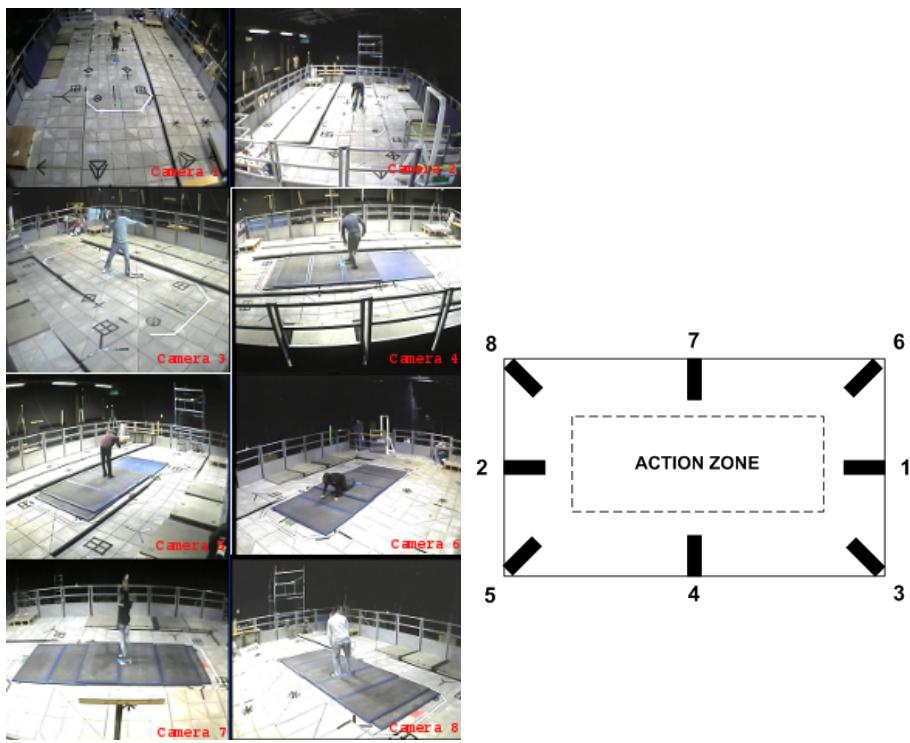
6.1. INTRODUCCIÓN

En el capítulo 4 se han mencionado un conjunto de métricas individuales objetivas que se enfocan en destacar alguna característica específica de un algoritmo (por ejemplo da mayor ponderación a los errores que se encuentran en el rango intermedio de un objeto de referencia por el impacto en el reconocimiento de formas) y que en su conjunto dan una idea global de desempeño de un algoritmo. En general la comunidad de investigación ha propuesto diversos y variados métodos para evaluar los algoritmos que utiliza este proyecto, presentando muchas veces sus resultados en una de estas métricas. El propósito de este capítulo es revisar estas métricas y verificar las más adecuadas para este proyecto.

Para evaluación de desempeño se han utilizado dos enfoques complementarios; medición a nivel de píxel, basado obtener las tasa de falsas alarmas (falsos objetos detectados) y fallas de detección (zonas que no son detectadas) para construir la curva de operaciones características (ROC), descritas en la sección 4.3.3 del capítulo 4 y mediciones a nivel de cuadro (imágenes), las cuales son métricas que miden similaridad (*PSNR, SSIM, D-SCORE*) o correlación (*MCC, F-Measure*) entre la imagen (mascara) resultante y su referencia (*ground-truth*). La combinación de ambos enfoques, proporcionan un rendimiento global de los algoritmos evaluados en cada una de las secuencias del conjunto de datos MuHAVI [36].

6.2. CONJUNTO DE DATOS DE ACCIONES HUMANAS - MUHAVI

El conjunto de datos MuHAVI[36] (*Multicamera Human Action Video*), es un repositorio de secuencias de video con 17 clases diferentes de acciones humanas, tomadas desde distintos ángulos de observación (ver descripción en tabla 6.1). Las acciones son realizadas sobre un escenario



(a) Vista de las 8 cámaras sobre el escenario con diferentes ejemplos de acciones
(b) Disposición de las 8 cámaras sobre el escenario

Figura 6.1: Esquema general con disposición de las 8 cámaras en el escenario.

rectangular, iluminado con un tipo de focos empleados en vías públicas para alumbrado nocturno. Las diferentes acciones son registradas por 8 cámaras dispuestas en las cuatro esquinas y costados de este escenario rectangular, como se indica en los ejemplos de la figura 6.1. El tipo de cámaras e iluminación, así como la distancia de las cámaras a los actores, y las irregularidades del escenario, simula un típico escenario de vigilancia nocturna de circuito cerrado por televisión (CCTV) y son parte de los desafíos que deben abordar los algoritmos de pruebas, al usar MuHAVI como datos de experimentación.

Este conjunto de datos fue desarrollado para satisfacer varios propósitos: evaluación de métodos de reconocimiento de acciones humanas basados en siluetas (*silhouette-based human action recognition* - SBHAR), que pueden hacer uso del conjunto de imágenes anotadas manualmente como medio de comparación de resultados, evaluación de segmentación contrastando resultados de algoritmos con las imágenes *ground-truth* que dispone este conjunto de datos, suministrar un conjunto de pruebas en investigaciones orientadas a la construcción de imágenes 3D, debido a sus imágenes similares desde diferentes ángulos o vistas.

Cada una de las acciones descritas en la tabla 6.1 fueron desarrolladas por 14 actores, y cada actor ejecuta 3 veces una determinada acción, resultando un total de $8 \times 17 \times 14 = 1904$ de segmentos

ACTION CLASS	ACTION NAME
C1	WalkTurnBack
C2	RunStop
C3	Punch
C4	Kick
C5	ShotGunCollapse
C6	PullHeavyObject
C7	PickupThrowObject
C8	WalkFall
C9	LookInCar
C10	CrawlOnKnees
C11	WaveArms
C12	DrawGraffiti
C13	JumpOverFence
C14	DrunkWalk
C15	ClimbLadder
C16	SmashObject
C17	JumpOverGap

Tabla 6.1: Tabla que describe el total de acciones humanas contenidas en MuHAVI

de video. Sin embargo, sólo se encuentran disponibles para la comunidad $8 \times 17 \times 7 = 952$ segmentos de video (las acciones realizadas por 7 actores), en secuencias de imágenes JPEG. Dentro del total de videos que proporciona MuHAVI, existe un subconjunto secuencias con imágenes de referencia o *ground-truth* (imágenes segmentadas correctamente), que permite evaluar el rendimiento de los algoritmos a nivel de píxel. Esto es, detectar todos los píxeles activos dentro de una imagen y luego contrastar con los píxeles de su imagen de referencia. Este subconjunto está compuesto por las cinco primeras clases indicadas en la tabla 6.1 (*WalkTurnBack*, *RunStop*, *Punch*, *Kick*, y *ShotGunCollapse*), ejecutadas por dos actores (denominados *Person1* y *Person4*), desde las cámaras 3 y 4, obteniendo un total de $5 \times 2 \times 2 = 20$ acciones. Las acciones realizadas por el actor denominado *Person1*, de cada una de las secuencias, contiene un gran número imágenes previas a la acción que va ser ejecutada, esto garantiza, en caso de ser necesario, el tiempo necesario para que los algoritmos puedan estimar el modelo del fondo de imagen.

6.3. PROCEDIMIENTO DE EVALUACIÓN

La evaluación de los algoritmos se basa principalmente en la comparación de sus curvas de operaciones características (Subsección 4.3.3), obtenidas mediante la realización de extensos experimentos. De manera visual se puede determinar el algoritmo con mejor desempeño, localizando la curva más cercana al borde superior izquierdo de la gráfica, es decir la curva ubicada sobre las demás gráficas resultantes. Un mejor desempeño se asocia con la curva que presenta una mayor área, con respecto a las otras en un rango específico de evaluación dentro del gráfico. Una mayor

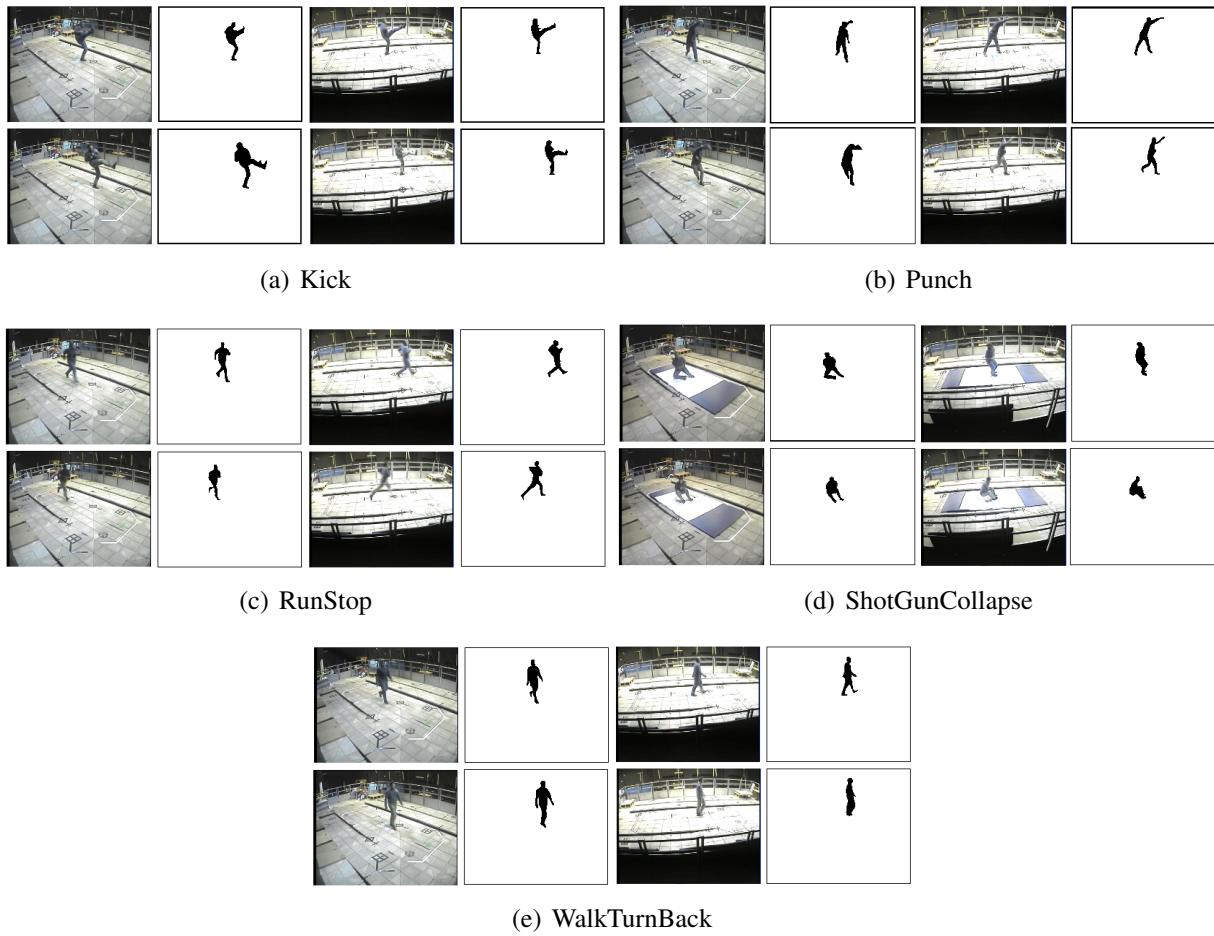


Figura 6.2: Ejemplo de acciones humanas desde diferentes vistas del escenario junto con sus respectivas imágenes de referencia (*ground-truth*)

exactitud de evaluación se obtiene fijando una tasa de falsos negativos, y luego comparando los valores de verdaderos positivos para el punto de operación determinado, el algoritmo con mejor desempeño es aquel con una tasa mayor de verdaderos positivos con respecto a todos los demás.

Los parámetros más relevantes, para el funcionamiento de los distintos algoritmos descritos en el Capítulo 3, son el factor de aprendizaje (*Learning Rate*) y el umbral (*Threshold*) que determina la pertenencia de un píxel a la imagen de fondo, o al actor en movimiento. En el caso del algoritmo no-paramétrico (descrito en la sección 3.6), α es un parámetro del color recomendado en 0,3 y *Threshold* es la probabilidad que un píxel sea *foreground*. Para el desarrollo de las pruebas se escogieron dos valores de factor de aprendizaje, $\alpha = 0,001$ y $\alpha = 0,0002$ (tabla 6.2) y se establecieron 21 valores de umbral en un rango de 1 a 100, Los primeros números de este conjunto tiene intervalos de separación 0.5 y 2, los cuales tienen como finalidad agregar más detalles en la parte superior de la curva.

La construcción de una curva ROC para cada uno de los algoritmo, se consigue manteniendo fijo uno de estos parámetros de configuración (*Learning Rate*) durante la ejecución de todas las secuencias de experimentación, y variando consecutivamente el segundo parámetro (*threshold*) dentro de un rango de valores previamente establecidos. De esta forma, una curva de operaciones correspondiente a una secuencia completa de experimentación, consiste de un conjunto de experimentos delimitados por el par ordenado $\langle \text{learning rate}, \text{threshold} \rangle$. La salida del procesamiento de un algoritmo, resulta en un punto dentro de la curva de operaciones características, determinados por los valores promedios de TPR y FPR (tasa de verdadero y falsos positivos respectivamente). Finalmente la curva de operaciones es un conjunto de puntos en la gráfica que son producto de la ejecución independiente de un algoritmo por cada par ordenado de estos parámetros.

El procedimiento de evaluación se realiza sólo con las secuencias de videos que contiene imágenes de referencia, éstas son las clases de acciones C1 a C5 señaladas en la tabla 6.1. Un punto TPR, FPR en la curva global de operaciones de un algoritmo consiste, como se indica en el párrafo anterior, de una serie de ejecuciones sobre estas secuencias. El resultado global es un promedio de las ejecuciones del mismo algoritmo sobre las 20 secuencias de la tabla 6.1. La salida del procesamiento de una secuencia (por ejemplo *WalkTurnBack Person1 Camera_3*) entrega un valor promedio de tasa de verdadero y falsos positivos, éste resultado posteriormente se promedia con las salidas procesadas de las otras secuencias, produciendo un valor global de TPR y FPR para el algoritmo en evaluación con la pareja de parámetros (*learning rate* y *threshold*) seleccionados para el procedimiento de evaluación en particular. De este modo, la construcción de una curva global de operaciones para el total secuencias de vídeo, con dos actores (*Person1*, *Person2*), adquiridas por dos cámaras (*Camera3* y *Camera4*), un valor de factor de aprendizaje (0,001 ó 0,0002), y 21 valores seleccionados para el umbral, requiere $5 \times 2 \times 2 \times 1 \times 21 = 420$ ejecuciones de un mismo algoritmo. Considerando que el algoritmo que menos demora en procesar el total de las secuencias es cerca de 9 minutos comparado con otro que demora 25 minutos, se tiene un procesamiento total de 3 horas y 9 horas respectivamente.

6.4. EVALUACIÓN DE RENDIMIENTO

La figura 6.3 ilustra las curvas de operaciones obtenidas durante el desarrollo de la experimentación, estas reflejan los resultados conseguidos con dos factores de aprendizaje propuestos para los ensayos (0,001 y 0,0002). La tasa de aprendizaje $\alpha = 0,001$, de acuerdo con la tabla 6.2, considera aproximadamente unos 100 cuadros antes que un objeto móvil detenido sea estimado como imagen de fondo, y 500 cuadros para $\alpha = 0,0002$. Las curvas incluidas en los gráfico constituy-

α	FRAMES
0.0001	1054
0.0002	527
0.0003	351
0.0004	263
0.0005	211
0.001	105
0.002	53
0.003	35
0.004	26
0.005	21

Tabla 6.2: Tabla que relaciona el valor de α y número de frames necesarios para que un objeto se mantenga estático.

yen una combinación global promedio del procesamiento de las 20 secuencias (5 secuencias de acciones humanas, dos actores, y dos cámaras) descritas en la tabla 6.1, para cada algoritmo en evaluación. Se incorpora también dos gráficos adicionales, con resultados de experimentación utilizando morfología; éste aplica un filtro de erosión que elimina cualquier conjunto de menos de 4 pixeles conectados. Una descripción más detallada con resultados desagregados por acciones, por cámara, y actores se encuentra en Apéndice B.

Las curvas logradas en la experimentación están sobre la linea recta $x = y$, de estimación aleatoria (sección 4.3.3), dentro del rango de valores $[0 - 1]$ en el eje de las ordenadas (verdaderos positivos) y $0 - 0,5$ para el eje de las abscisas (falsos positivos). La forma de las curvas, en todos los diagramas de la figura 6.3, sigue un patrón que relaciona el comportamiento de ambas métricas en forma directa; se observa que en la medida que el valor de una métrica cambia (aumenta o disminuye) en una dirección de su eje, la otra también cambia en el mismo sentido, pero no en la misma proporción. El incremento (o disminución) en los valores de ambas tasas se consigue modificando en forma discreta los niveles *threshold*. En el caso de los algoritmos basado en mixtura de Gaussianas (*SAGMM*, *MOG2*, *UCV Linear* y *Staircase*) incrementos en este valor causan disminución de la tasa de verdaderos y falsos positivos. Pero, en el caso del algoritmo No-Paramétrico incremento en el *threshold* produce aumento en ambas tasas. Se debe recordar, que la tasa de verdaderos positivos (ver sección 4.3.3), para la evaluación de reconocimiento de siluetas, es una medida que indica la proporción de pixeles correctamente seleccionados en la silueta, y la tasa de falsos positivos (falsas alarmas) indica una proporción de pixeles de *background* seleccionados incorrectamente como pixeles perteneciente a la silueta.

En general el algoritmo SAGMM [8] presenta un desempeño promedio mejor con respecto a los algoritmos basado en mixtura de componentes Gaussianas, esto debido que la curva de operaciones

se encuentra localizada más cerca del borde superior izquierdo de la gráfica, sobre las otras curvas. Sin embargo la curva del algoritmo No-Paramétrico [13] la sigue de cerca al estar levemente debajo de la curva SAGMM, el cual puede ser interpretado como un rendimiento similar entre ambos algoritmos. El siguiente algoritmo en esta clasificación general de desempeño, se encuentra MOG2 [47] y la versión *Linear* del algoritmo UCV [34]. Se puede advertir que el algoritmo UCV modelo *staircase* [34] mejora su desempeño al sobrepasar la curva UCV *Linear* y luego la de MOG2, entre el rango del 2 % y 6 % de la tasa de falsos positivos, pero sólo con la tasa de aprendizaje $\alpha = 0,001$ y su desempeño es inferior al utilizar un factor de aprendizaje de 0,0002 con o sin morfología. Esta evaluación cualitativa también se ratifica comparando el área bajo la curva, entre el rango [0 – 0,15] del eje falsos positivos y [0,4 – 1,0] en el eje verdaderos positivos (indicado en la tabla 6.3) con un valor $\alpha = 0,001$. SAGMM tiene un área mayor (0,12278) que los otros algoritmos, continua el No-Paramétrico (0,11771), MOG2 (0,09908), UCV en su versión *Staircase* (0,09083), y finalmente *Linear* (0,08310).

Algoritmo	ΔX	ΔY	Área
sagmm	0,15 – 0,001	0,89 – 0,40	0.123
np	0,15 – 0,007	0,89 – 0,63	0.113
mog2	0,15 – 0,002	0,75 – 0,40	0.099
ucv_staircase	0,15 – 0,019	0,80 – 0,40	0.090
ucv_linear	0,15 – 0,016	0,72 – 0,40	0.083

Tabla 6.3: Área bajo la curva de operaciones de cada algoritmo.

El resultado con $\alpha = 0,0002$, figura 6.3(b), modifica ligeramente la localización de la curva SAGMM con respecto a la figura 6.3(a) y continúa, junto con el algoritmo NP, presentando mejor desempeño en relación a los demás. Mejora también el rendimiento de los algoritmos MOG2 y UCV *Linear* al producirse un desplazamiento de ambas curvas por encima de su localización anterior. Pero, la versión UCV *Staircase* deteriora notablemente su desempeño, al producir un desplazamiento de su curva hacia el lado derecho; disminuye la tasa de verdaderos positivos y aumenta la tasa de falsos positivos. La mayor influencia del aumento de la tasa de aprendizaje ($\alpha = 0,0002$) se manifiesta en los algoritmos SAGMM y MOG2, al aumentar los valores de *threshold* para alcanzar la misma tasa de falsos positivos (notar linea roja en 0,03). Un aumento en el *threshold* incide en una mejora de las medidas de rendimiento como MCC, FMeasure, y DScore (4.3.4).

Un fenómeno diferente se produce al utilizar el operador morfológico de ‘erosión’ (Capítulo 9 de [21]), en las figuras 6.3(c) y 6.3(d) se aprecia un efecto de compresión hacia el lado izquierdo de las gráficas, es decir, la tasa de falsos positivos disminuyen y en una proporción menor la tasa de verdaderos positivos. Particularmente los algoritmos UCV deterioran su desempeño, la versión

Linear por ejemplo no logra sobrepasar el 60 % en la tasa de verdaderos positivos, y la versión *Staircase* no supera el el 76 %, con ambos factores de aprendizaje. El filtro morfológico elimina falsos positivos y negativos (ver 4.3.2), pero, también en menor medida reduce positivos (pixeles de una silueta), principalmente en las zonas de borde entre la silueta y la imagen de fondo, lo que explica las disminución de la tasa de falsos positivos en todas las curvas.

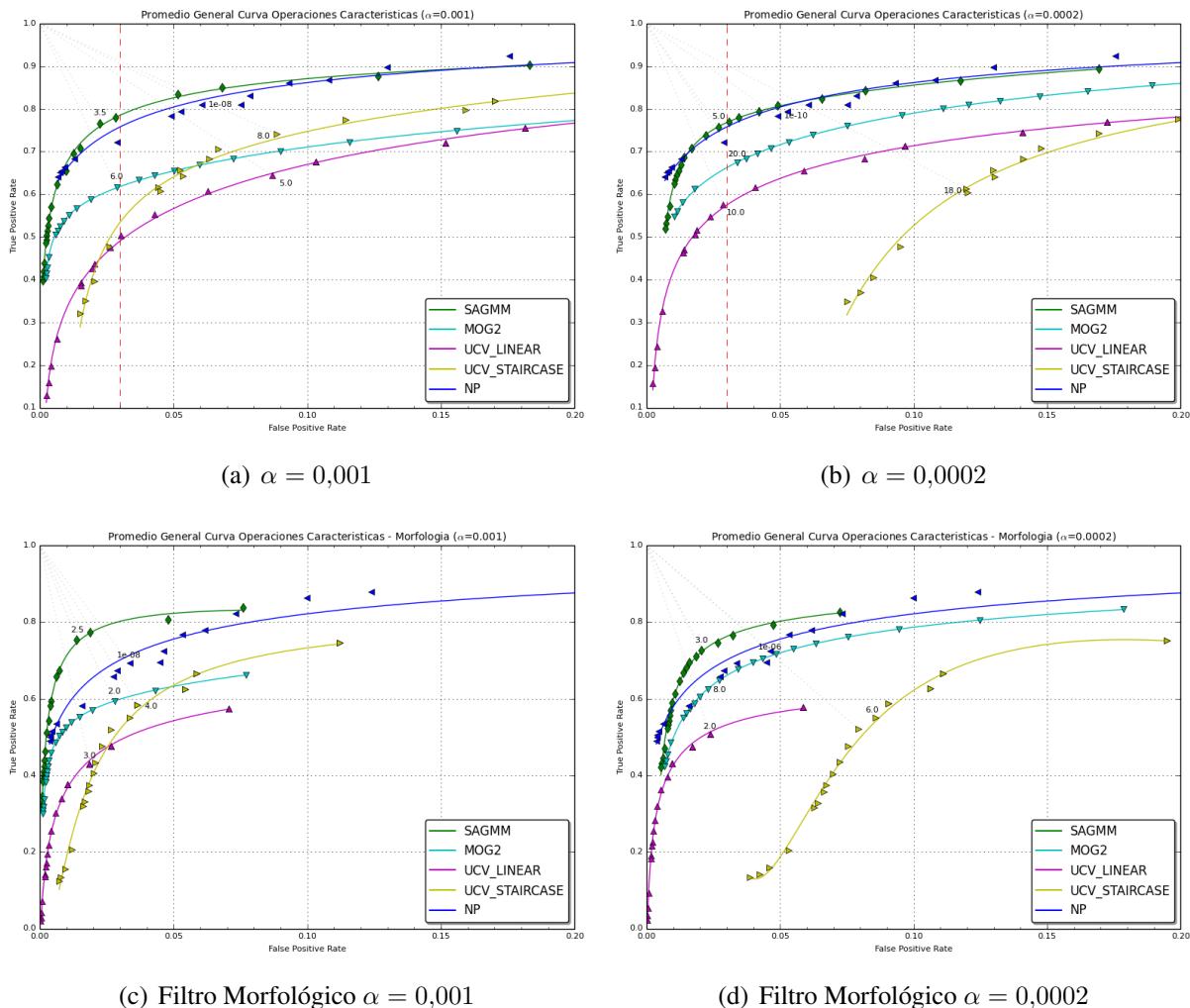


Figura 6.3: Curvas de operaciones características para dos valores diferentes de factor de aprendizaje de cada algoritmo. En ambas gráficas se ha dejado constante el valor de aprendizaje ($\alpha = 0,001$ y $\alpha = 0,0002$ respectivamente) y modificado el valor de umbral (distancia de Mahalanobis). Incrementos en el valor de umbral resulta en una disminución constante de la tasa de verdaderos y falsos positivos. Las imágenes inferiores corresponden a experimentos similares con los mismos valores de α , aplicando morfología a las imágenes resultantes.

6.4.1. Comparación puntos de operación

El criterio usado para seleccionar el punto de operación de un algoritmo en su curva de operaciones, consiste en escoger el lugar de la curva más cercano al punto de clasificación perfecta (0, 1), ver linea segmentada gris en figura 6.3, este es el punto de mejor compromiso entre la tasa de verdadero y falso positivos. Las tablas 6.4, 6.5, 6.6, y 6.7 reporta valores de métricas de rendimiento y desempeño de los puntos de operaciones escogidos de cada algoritmo, para los dos factores de aprendizaje estudiados con y sin morfología. Las métricas registradas en estas tablas se recopilan en las imágenes de la figura 6.4. La variación general de todas estas métricas de rendimiento (*MCC-Matthew's Correlation Coefficient, F-Measure, D-Score, PSNR, y MSSIM*) con respecto a diferentes valores de *threshold* (distancia de Mahalanobis) se muestran en las figuras 6.6 y 6.7

Los resultados generales de la figuras 6.6 y 6.7 señalan, desde la perspectiva de las métricas de rendimiento, el valor óptimo de la distancia de *Mahalanobis* que debiera ser elegido para operar con los algoritmo. Sin embargo, estos valores de *threshold* no resultan en mejores puntos en la curva de operaciones características. Esto se evidencia al confrontar las curvas de rendimiento (figuras 6.6 y 6.7) con las tablas (6.4, 6.5, 6.6, y 6.7) de puntos óptimos de operación seleccionados para comparar cada uno de los algoritmos. Por ejemplo, el valor óptimo de *MCC* (0,61) y *F-Measure* (0,60) en *SAGMM* se alcanza con un *threshold* de 6 y 10 (4,5 y 5 con morfología) para $\alpha = 0,001$ y $\alpha = 0,0002$ respectivamente. Utilizando estos valores óptimos para seleccionar el punto de operación en la curva ROC, resulta en tasas de verdadero y falso positivos inferiores a los escogidos en forma manual, de acuerdo con el criterio del punto más cercano a la zona de clasificación perfecta. El valor *threshold* = 6 (óptimo de acuerdo con la figura 6.6(a) de *MCC* para $\alpha = 0,001$) origina el punto $\langle 0,62 - 0,0065 \rangle$ en la curva de operaciones, pero el punto de operación seleccionado (tabla 6.4) corresponde al par ordenado $\langle 0,76626 - 0,02246 \rangle$, con *threshold*=3.5. Esta última selección sacrifica falsos positivos (errores) en 1.6 % (en el rango 0-1), pero mejora el rendimiento de clasificación aumentando la tasa de verdaderos positivos en un 14 %. Contrastando *MCC* (0,53) y *F-Measure* (0,50) del punto elegido con sus pares del punto óptimo, ambas cifras se diferencian en un 7 % y 10 % respectivamente del punto considerado óptimo. Comparando además el valor de la métrica *D-Score* para ambos puntos en discusión (óptimo y escogido), existe una diferencia de 0,00151 entre ambas, un 20 % del valor mínimo (notar que para *D-Score* interesa esta métrica sea el mínimo posible), señalando que se ha escogido un punto de operación (no el óptimo) que tolera un 20 % de incremento en errores de bajo costo (pixeles errados que no afectan el reconocimiento de una silueta). El impacto de la diferencia en esta última métrica se puede observar en la figura 6.5(b) con una menor cantidad de falsos positivos con respecto a la figura 6.5(a), pero en este último es posible aún reconocer la silueta a pesar de contener más falsos positivos, es decir

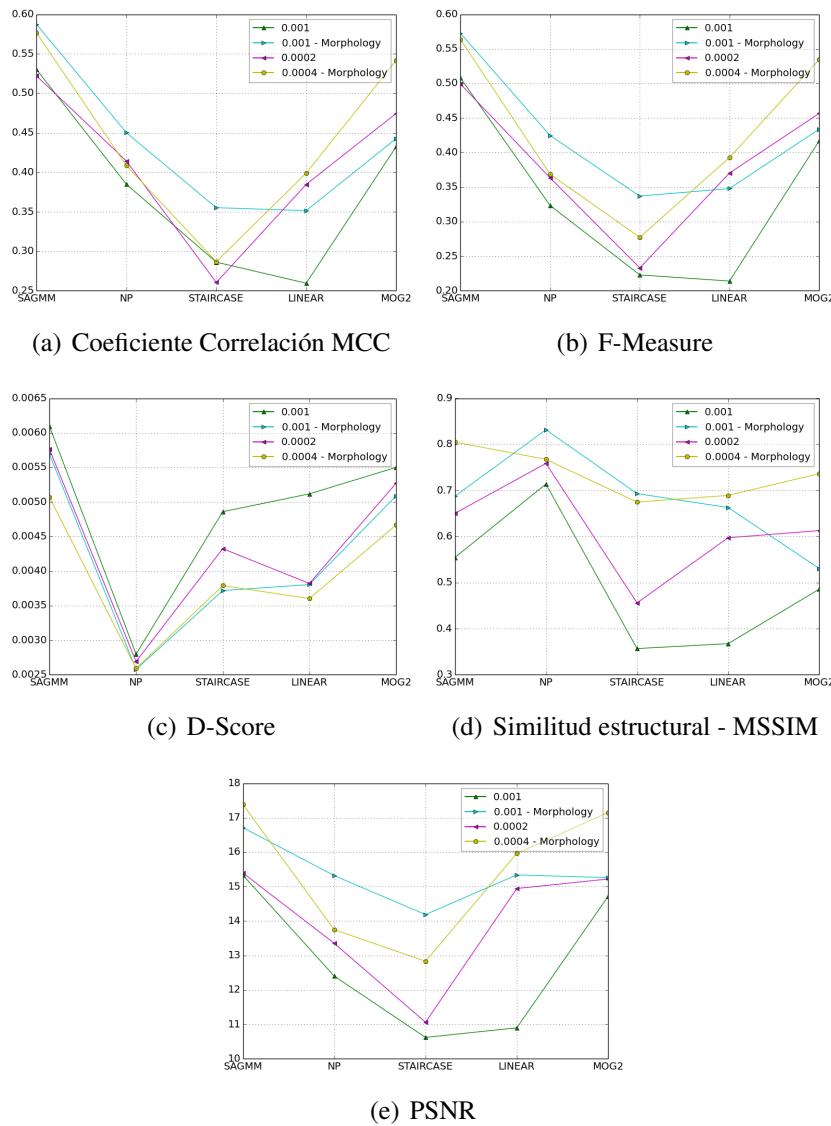


Figura 6.4: Comparación de métricas de calidad de los puntos de operación escogidos señalados en las tablas 6.4, 6.5, 6.6, y 6.7

los falsos positivos no afectarían el proceso de segmentación. En síntesis, la elección del punto de operación es un compromiso que requiere contrastar estas métricas de rendimiento con la curva de operaciones de desempeño del algoritmo para escoger el punto más adecuado con el propósito del algoritmo. Se debe mencionar que el punto de operación es un resultado general, quedan fuera particularidades, y este punto corresponde a un promedio que consolida los distintos resultados del algoritmo enfrentado a las diferentes dificultades que proporcionan las secuencias.

En promedio, con un factor de aprendizaje en 0,001 y un *threshold* de 3,5 (tabla 6.4), el algoritmo SAGMM tiene un rendimiento de un 76,6 % en la clasificación de los píxeles de siluetas y un

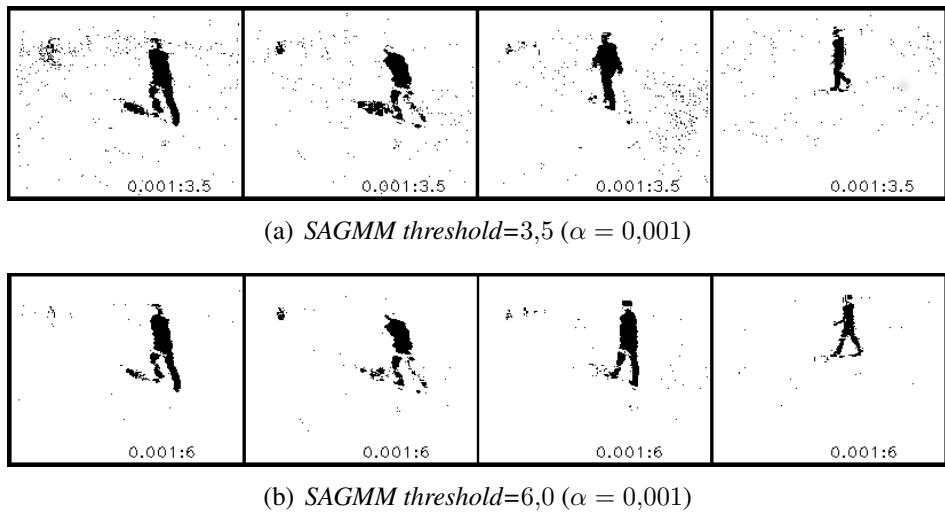


Figura 6.5: Comparación de imágenes de diferentes secuencias para dos valores de *threshold* (3,5 y 6) en algoritmo *SAGMM*. Los dos primeros cuadros en ambas filas corresponden a la secuencias con peor desempeño (*RunStop*) y los dos siguientes al mejor desempeño (*WalkTurnBack*).

error de 2,2 % en la clasificación de pixeles de la imagen de fondo. El algoritmo No-Paramétrico con *threshold* 1×10^{-8} tiene una tasa de 80 % en verdaderos positivos, pero un error de 6 % en falsos positivos. En contraste el algoritmo MOG2 tiene un 61 % (*threshold* 6) tasa de verdaderos positivos y 2.8 % en errores falso positivos, UCV *Staircase* presenta un 74 % (*threshold* 8) y un 64 % para UCV *Linear*, ambos con una tasa de falsos negativos similar, cercana al 8.8 %. Los cuadros de la figura 6.4 confirman lo evidenciado por las curvas de operaciones de la sección anterior, SAGMM tiene un mejor desempeño con respecto a los otros algoritmos. Los valores del coeficiente de correlación MCC (*Matthew's Correlation Coefficient*) y *F-Measure* aventaja los resultados de los otros algoritmos en todos los casos. En la medida que el valor de MCC se acerca a 1, significa una buena predicción en la clasificación y SAGMM se encuentra en un rango entre [0,50 – 0,57], siendo su valor máximo 0,6 (0,67 con morfología). La misma situación ocurre con *F-Measure*, está métrica se entiende como el promedio ponderado de las proporciones *precision* y *recall* y los valores obtenidos durante la experimentación son muy similares al coeficiente de correlación *MCC*. Las figuras 6.6(a) 6.6(b), 6.6(c), y 6.6(d) revelan que ambas métricas producen similares resultados, las curvas siguen la misma tendencia, por lo que la utilización de estas dos métricas en conjunto no agregan mayor información. El coeficiente de correlación *MCC* es una medida más robusta para predecir el resultado de clasificación, debido que se construye utilizando las 4 mediciones de *TP*, *FN*, *TN*, y *FP* registrados en la matriz de confusión (4.2). Algo parecido ocurre con las mediciones de similaridad *PSNR* [41] y *MSSIM* [42], estas métricas intentan evaluar similitud entre dos cuadros, pero en los gráficos 6.7(a), 6.7(b), 6.7(c), y 6.7(d) se advierte que las curvas tienen el

el mismo comportamiento, ambas curvas alcanzan un valor cercano al máximo en $threshold = 10$, y tienen un incremento lineal entre los valores 1y6 de $threshold$, siendo la pendiente de $MSSIM$, entre estos dos puntos, mayor a la de $PSNR$ permitiendo obtener información más detallada ante pequeñas variaciones de $threshold$. Esta característica de similitud entre ambas métricas (para evaluación de siluetas) no aporta mayor información de diferenciación, cualquiera de los dos se puede usar como métrica de evaluación de calidad. Esta semejanza de medición entre ambas métricas se puede explicar por la particularidad de las imágenes que se están evaluando, la medida de similaridad estructural intenta medir calidad simulando el ojo humano. Es muy apropiada para diferenciar distorsiones y ruido en imágenes en escala de grises y colores RGB. Pero en este caso de evaluación de siluetas, se compara sólo imágenes binarias (blanco y negro) y se pierden las particularidades de una imagen como el brillo, contraste que utiliza la métrica $MSSIM$ para evaluar calidad de una imagen. Finalmente, a pesar que la medida de $F\text{-Measure}$ o MCC que presenta el algoritmo NP (o las dos versiones de UCV) son menores que $SAGMM$, éste al contrario tiene un mejor valor de $D\text{-Score}$ (comparativamente menor que los otros algoritmos). Esto indica que NP tiene muchos errores ($F\text{-Measure}$ bajo por ejemplo), pero esos errores tienen muy bajo costo y no afectarían el reconocimiento de una silueta durante segmentación.

Métricas	SAGMM	NP	STAIRCASE	LINEAR	MOG2
Threshold	3.5	1e-08	8	5	6
TPR	0.77	0.81	0.74	0.64	0.62
FPR	0.02	0.06	0.09	0.09	0.03
TPR SE(%)	3.73	3.24	2.50	2.33	6.27
FPR SE(%)	0.35	1.14	1.21	1.80	0.97
MCC	0.53	0.38	0.29	0.26	0.43
FMeasure	0.51	0.32	0.22	0.21	0.42
DSCORE	6.1e-03	2.8e-03	4.9e-03	5.1e-03	5.5e-03
MSSIM	0.55	0.71	0.36	0.37	0.49
PSNR	15.33	12.40	10.62	10.90	14.72

Tabla 6.4: Métricas de desempeño con un factor de aprendizaje 0.001.

Métricas	SAGMM	NP	STAIRCASE	LINEAR	MOG2
Threshold	2.5	1e-08	4	3	2
TPR	0.75	0.69	0.58	0.43	0.59
FPR	0.01	0.03	0.04	0.02	0.03
TPR SE(%)	4.01	4.25	2.30	2.61	5.99
FPR SE(%)	0.43	0.83	0.81	0.64	1.20
MCC	0.59	0.45	0.36	0.35	0.44
FMeasure	0.57	0.42	0.34	0.35	0.43
DSCORE	5.7e-03	2.6e-03	3.7e-03	3.8e-03	5.1e-03
MSSIM	0.69	0.83	0.69	0.66	0.53
PSNR	16.71	15.32	14.19	15.34	15.26

Tabla 6.5: Métricas de desempeño con un factor de aprendizaje 0.001 y morfología.

Métricas	SAGMM	NP	STAIRCASE	LINEAR	MOG2
Threshold	5	1e-10	18	10	20
TPR	0.76	0.78	0.61	0.58	0.67
FPR	0.03	0.05	0.12	0.03	0.03
TPR SE (%)	3.68	3.58	3.62	2.70	3.59
FPR SE (%)	0.86	1.07	3.57	0.50	1.80
MCC	0.52	0.41	0.26	0.38	0.47
FMeasure	0.50	0.36	0.23	0.37	0.46
DSCORE	5.8e-03	2.7e-03	4.3e-03	3.8e-03	5.3e-03
MSSIM	0.65	0.76	0.46	0.60	0.61
PSNR	15.40	13.36	11.06	14.95	15.22

Tabla 6.6: Métricas de desempeño con un factor de aprendizaje 0.0002.

Métricas	SAGMM	NP	STAIRCASE	LINEAR	MOG2
Threshold	3	1e-06	6	2	8
TPR	0.73	0.72	0.52	0.51	0.65
FPR	0.02	0.05	0.08	0.02	0.03
TPR SE (%)	3.99	3.79	2.51	3.42	3.73
FPR SE (%)	0.88	1.21	2.51	1.01	2.19
MCC	0.58	0.41	0.29	0.40	0.54
FMeasure	0.56	0.37	0.28	0.39	0.53
DSCORE	5.1e-03	2.6e-03	3.8e-03	3.6e-03	4.7e-03
MSSIM	0.80	0.77	0.67	0.69	0.74
PSNR	17.38	13.75	12.83	15.96	17.15

Tabla 6.7: Métricas de desempeño con un factor de aprendizaje 0.0002 y morfología.

6.4.2. Intervalo de confianza curva operaciones características

Las gráficas de la figura 6.3 se construyen realizando un promedio de todas las instancias independientes de experimentación (*Kick Person1 Camera3*, *Kick Person1 Camera4*, *WalkTurnBack Person1 Camera3*, ..., etc). La comparación de la respuesta de clasificación entre varios de los algoritmos se complementa con la medición de la varianza de cada punto promediado, y en forma más específica generando intervalos de confianza [29] alrededor de los puntos TP/FP, que permiten establecer bandas de confianza (probabilidad $1 - \alpha = 95\%$) alrededor de la curva de operaciones. Existen varias técnicas [15] para combinar curvas ROC independientes y construir una curva que representa el comportamiento general de todas las instancias involucradas. Una de estas técnicas es ‘*thresholding averaging*’, la cual consiste básicamente en promediar todos los puntos generados por un valor de *threshold* y es el procedimiento mediante el cual se han generado las curvas de la figura 6.3.

La figuras 6.9 y 6.10 presenta los intervalos de confianza de 95 % de las curvas de operaciones, destacadas en líneas segmentadas azules rodeando la curva promedio. Se agregan además, en líneas segmentadas de otros colores, las curvas promedio por acción destacando el aporte individual a la curva promedio general. Las bandas que envuelven las curvas de operaciones señalan, con una

confianza del 95 %, el rango de valores en el cual se encuentra el valor promedio de la tasa de verdaderos positivos, con posibilidad de un 5 % que el valor se encuentre fuera de esta banda. La construcción de estas bandas no toma en cuenta los *FP* (falsos positivos) y usa sólo los intervalos de *TP* (verdadero positivos) para construir las banda de confianza. Los puntos superior e inferior de un intervalo se calculan mediante la ecuación 6.1, se obtiene el promedio (\bar{X}_{TP}) y desviación estándar (s) de las muestras generadas de un valor de *threshold*, esto es el valor promedio *TP* de cada secuencia de acción, y finalmente se divide por la raíz cuadrada del número de muestras.

$$\mu \in \bar{X}_{TP} \pm 1,96 \frac{s}{\sqrt{n}} \quad (6.1)$$

El algoritmo con mayor varianza es *MOG2*, y se manifiesta en la amplitud del ancho de su bandas de confianza comparado con las otras figuras; 6.2 % y 3.5 % ($\alpha = 0,001$ y $\alpha = 0,0002$ respectivamente) de acuerdo con las tablas 6.4 y 6.4. Son las acciones *WalkTurnBack* y *Punch* las que contribuyen con este efecto, debido a la distancia que separa ambas curvas del promedio general. Pero es la acción *Punch* y en menor medida *ShotGunCollapse* las que empeoran el desempeño global, y más específicamente, de acuerdo con la figura B.9(a) del apéndice ??, son las secuencias registradas por la cámara 3 las que deterioran el rendimiento general. Los algoritmos *SAGMM* y *NP* presentan similar intervalos de confianza para los dos factores de aprendizaje, comprobándose en la tabla 6.4 con $\alpha = 0,001$ *SAGMM* tiene un error estándar (*SE*) de 3.5 % y 3.2 % para el algoritmo *NP*. Con estos valores de confianza se puede afirmar que ambos algoritmos presentan un desempeño similar, porque sus curvas promedios son parecidas, *NP* ligeramente inferior, y sus intervalos de confianza se sobreponen. De esta forma, cualquier punto de las curvas queda delimitado por un 95 % de posibilidad que este dentro de esas bandas. Al igual que *MOG2* son las acciones *Punch* y *ShotGunCollapse* que deterioran el rendimiento en *NP*, pero en este caso son las acciones registradas por la cámara 4 las que disminuyen el desempeño de este algoritmo. Con *SAGMM* son las acciones *RunStop* y menor medida *ShotGunCollapse* que se localizan debajo de la curva de promedio general, con $\alpha = 0,0002$ se agrega también *WalkturnBack* especialmente las acciones registradas por la cámara 3, actor *Person4*. A pesar que en términos generales el desempeño de *UCV* en sus dos versiones (*Linear* y *Staircase*) es menor comparado con los otros algoritmos, ambos presentan la menor varianza de todos con $\alpha = 0,001$, para $\alpha = 0,0002$ se demostró que el desempeño fue notablemente inferior. El error estándar de la tabla 6.4 es de 2.5 % y 2.2 % para *Staircase* y *Linear* respectivamente, y se verifica además por el menor ancho en los intervalos de confianza en ambas figuras. Todas las curvas de acciones promedio en *Staircase* se encuentran muy ajustadas a su promedio general y en el caso de *Linear* es la acción *ShotGunCollapse* registrada por la cámara 3 y específicamente la acción ejecutada por el actor *Person4* la que se encuentra debajo

del promedio.

La mayoría de la secuencia de acciones registradas por la cámara 3 son las que contribuyen al deterioro del desempeño general, esta cámara particularmente se encuentra en un borde del escenario y la distancia al centro del escenario es mayor de las otra cámara que se encuentra en un costado del escenario.

6.5. RESUMEN

En este capítulo se ha enfocado principalmente en describir y analizar los distintos resultados obtenidos durante la experimentación, se ha discutido el impacto de estas métricas en la selección de un punto de operación más óptimo. La curva de operaciones características es la herramienta más importante para comparar desempeño entre distintos algoritmos. Mediante la observación visual es posible discriminar y determinar el mejor desempeño de un algoritmo; sólo localizando la curva cerca del borde superior izquierdo o la que está emplazada sobre las otras. Una forma más precisa de diferenciar consiste en comparar el área bajo la curva que generan las curvas de operaciones, dentro de un rango específico de evaluación. Se ha discutido también, la forma de encontrar un punto de operación óptimo dentro de la curva operaciones, contrastando esta selección con las métricas de rendimiento mencionadas en el capítulo 4. Se ha demostrado que las métricas de *MCC* y *F-Measure* son indicadores de calidad muy parecidos, en la evaluación de desempeño de los algoritmos propuestos para este proyecto, al ser operados sobre el conjunto de datos MuHAVI. Asimismo, *D-Score* resulta una buena métrica para discriminar falsas alarmas (falsos positivos) de poco impacto que no afectan el reconocimiento de imágenes. También se ha notado que las métricas *PSNR* y *MSSIM*, que miden similaridad entre imágenes, tienen una curva de respuesta análoga a las variaciones de *threshold*. La característica principal de la herramienta de similaridad (*MSSIM*) no es apropiada para medir calidad de los resultados, debido que las imágenes que se evalúan pierden las características particulares que contienen las imágenes en escala de grises o colores. En síntesis, las curvas de operaciones, en complemento con las métricas de correlación (*MCC*) y *D-Score* constituyen un buen conjunto de herramientas, que permiten conseguir el punto de operación apropiado, un compromiso entre una buena discriminación y una buena detección de siluetas.

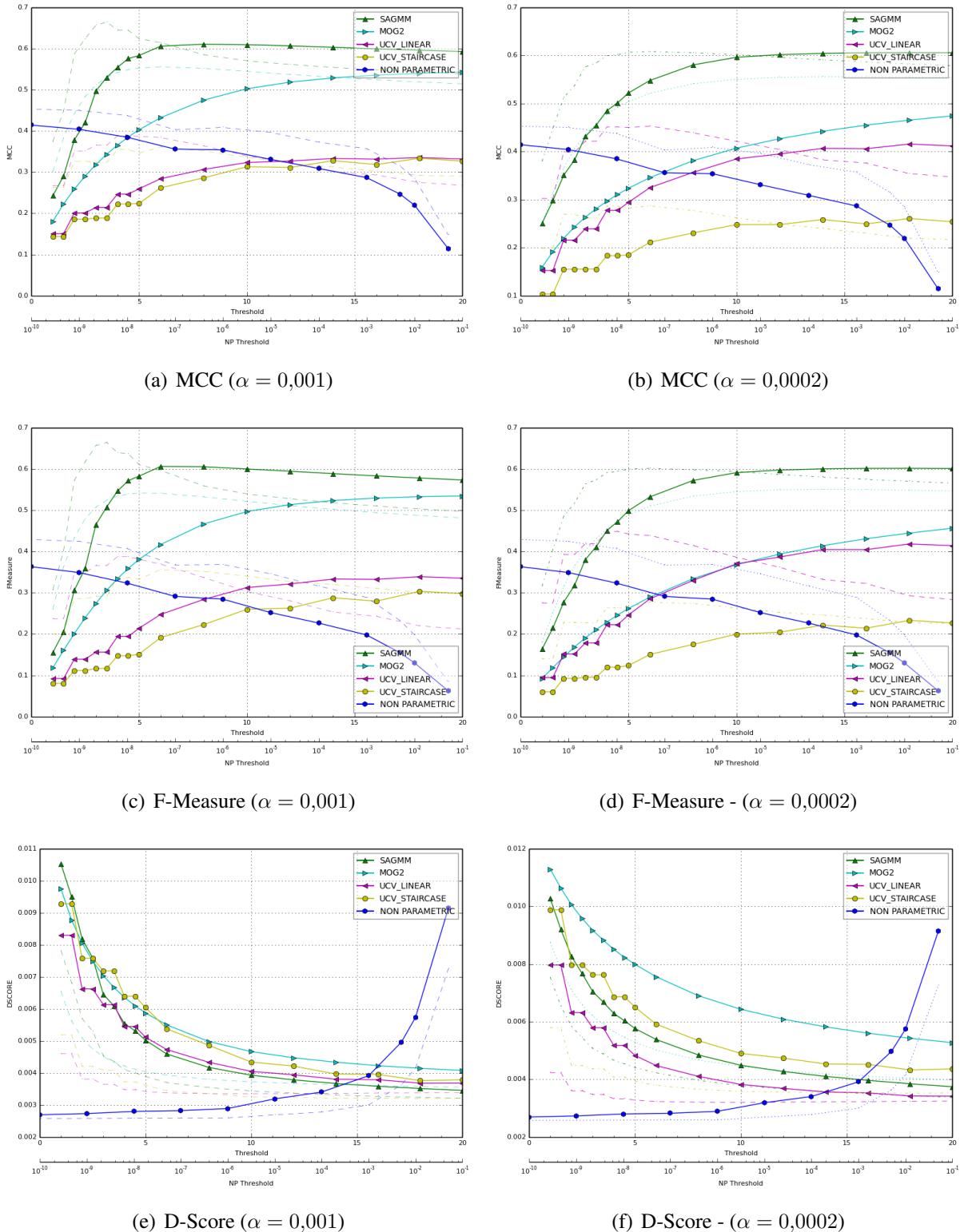


Figura 6.6: Cuadros comparativos generales de métricas de rendimiento *MCC*, *F-Measure* y *D-Score* con tasa de aprendizaje $\alpha = 0,001$ y $\alpha = 0,0002$, lineas segmentadas en los cuadros indican el resultado con morfología

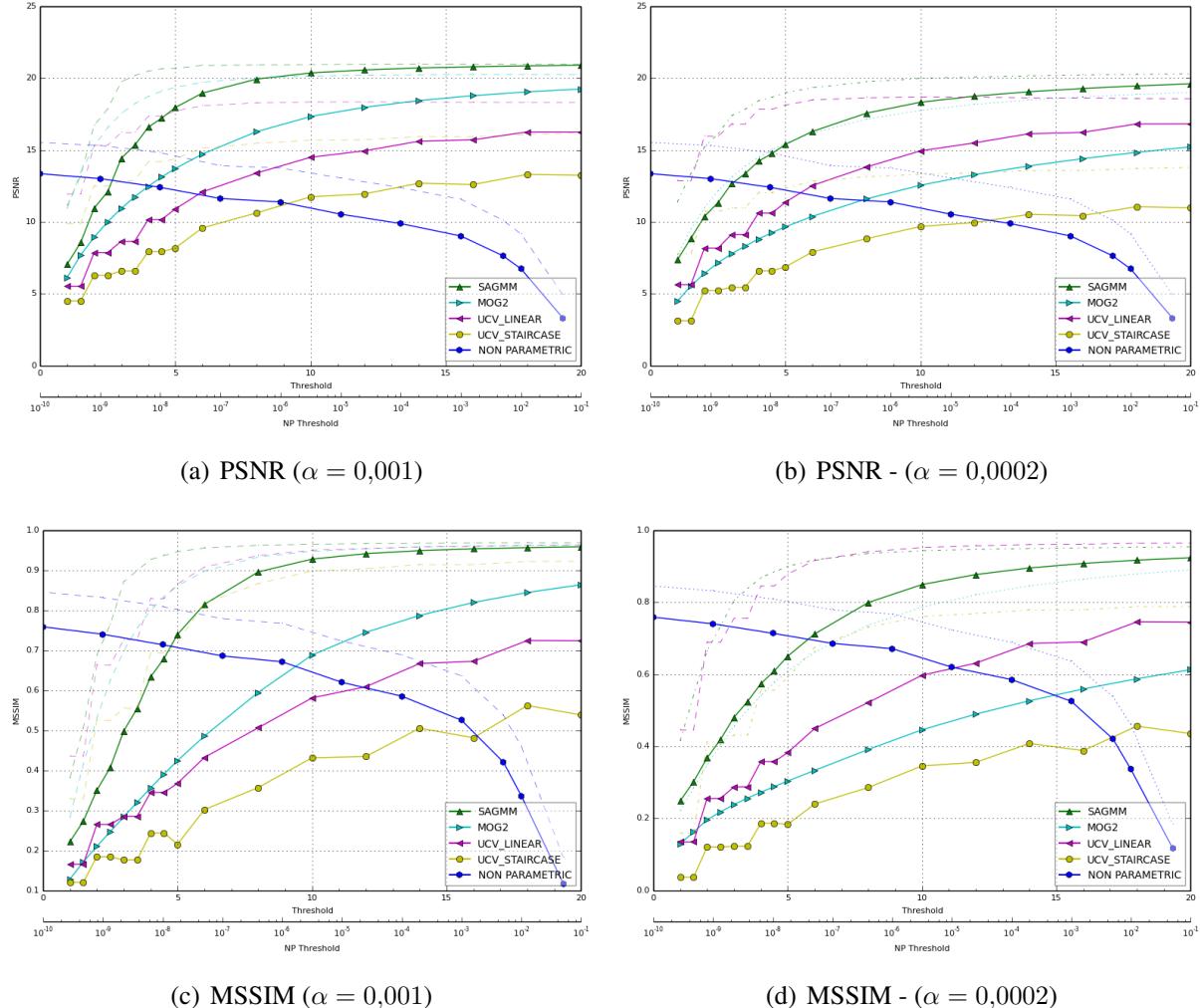


Figura 6.7: Cuadros comparativos generales de métricas de percepción PSNR y MSSIM con las tasas de aprendizaje $\alpha = 0,001$ y $\alpha = 0,0002$, la líneas segmentadas son los resultados utilizando morfología

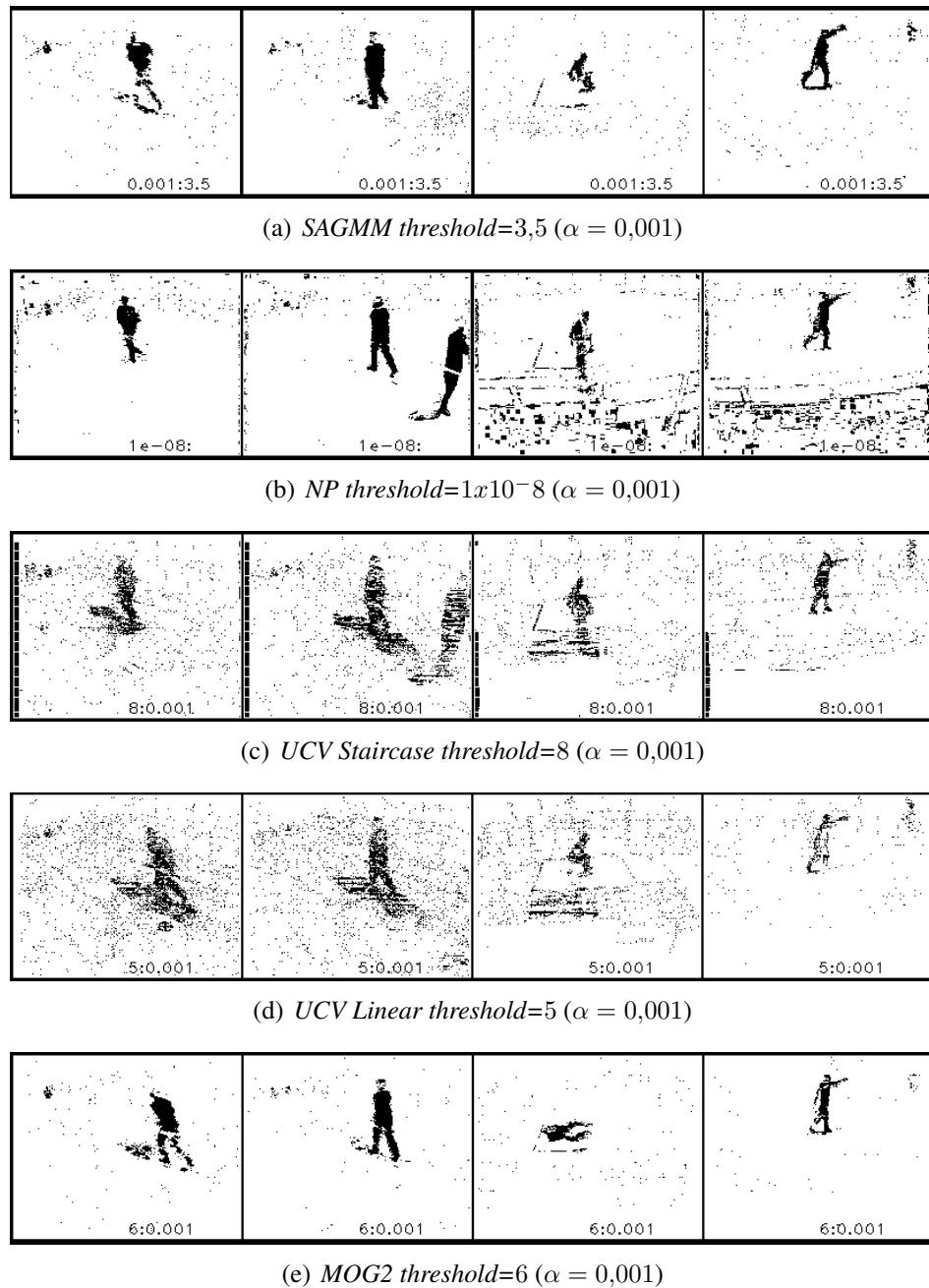


Figura 6.8: Imagenes resultantes de los diferentes algoritmos mencionados en la tabla 6.4

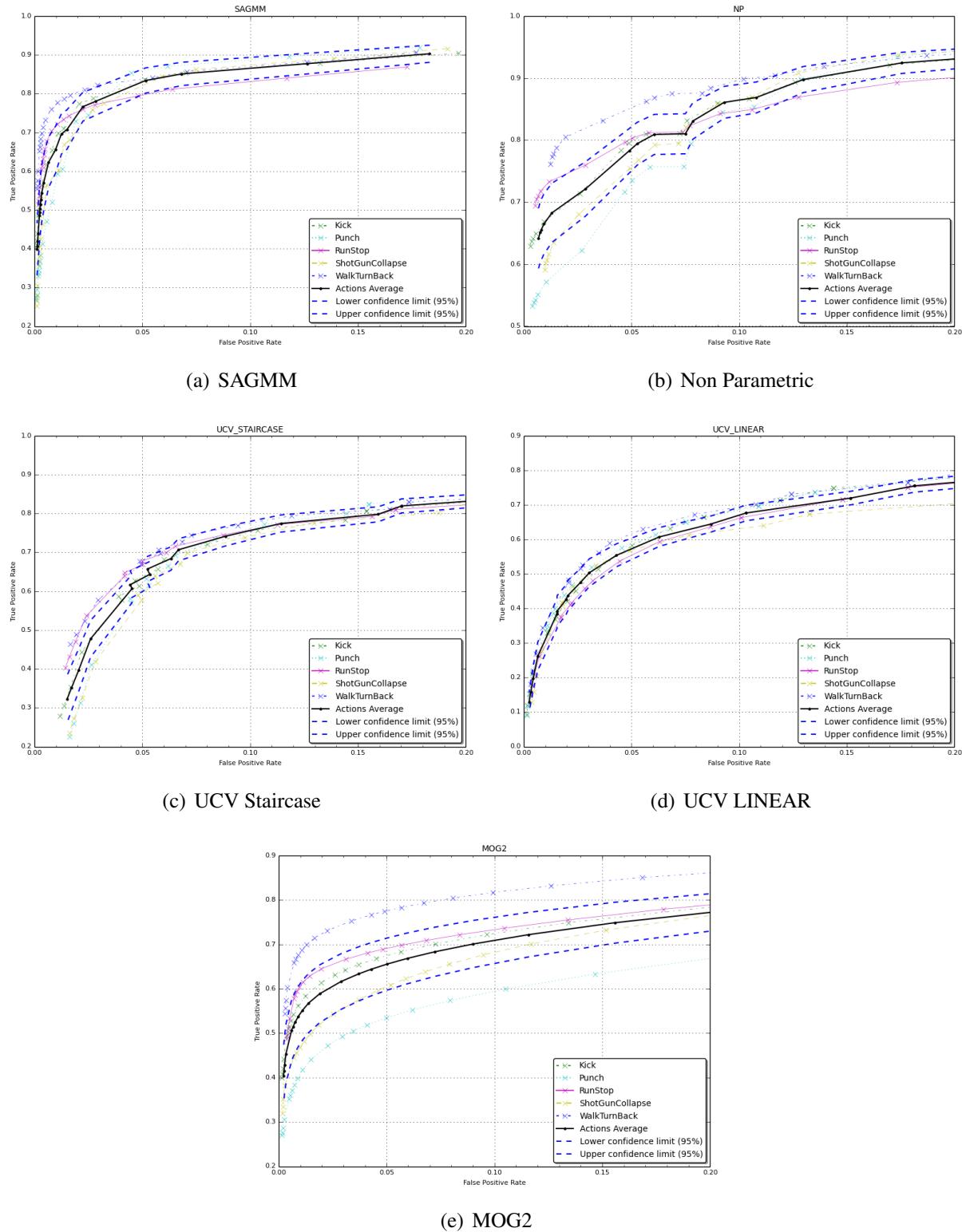


Figura 6.9: Curvas promedios de operaciones características resultantes separadas por algoritmo con intervalo de confianza de 95 % ($\alpha = 0,001$)

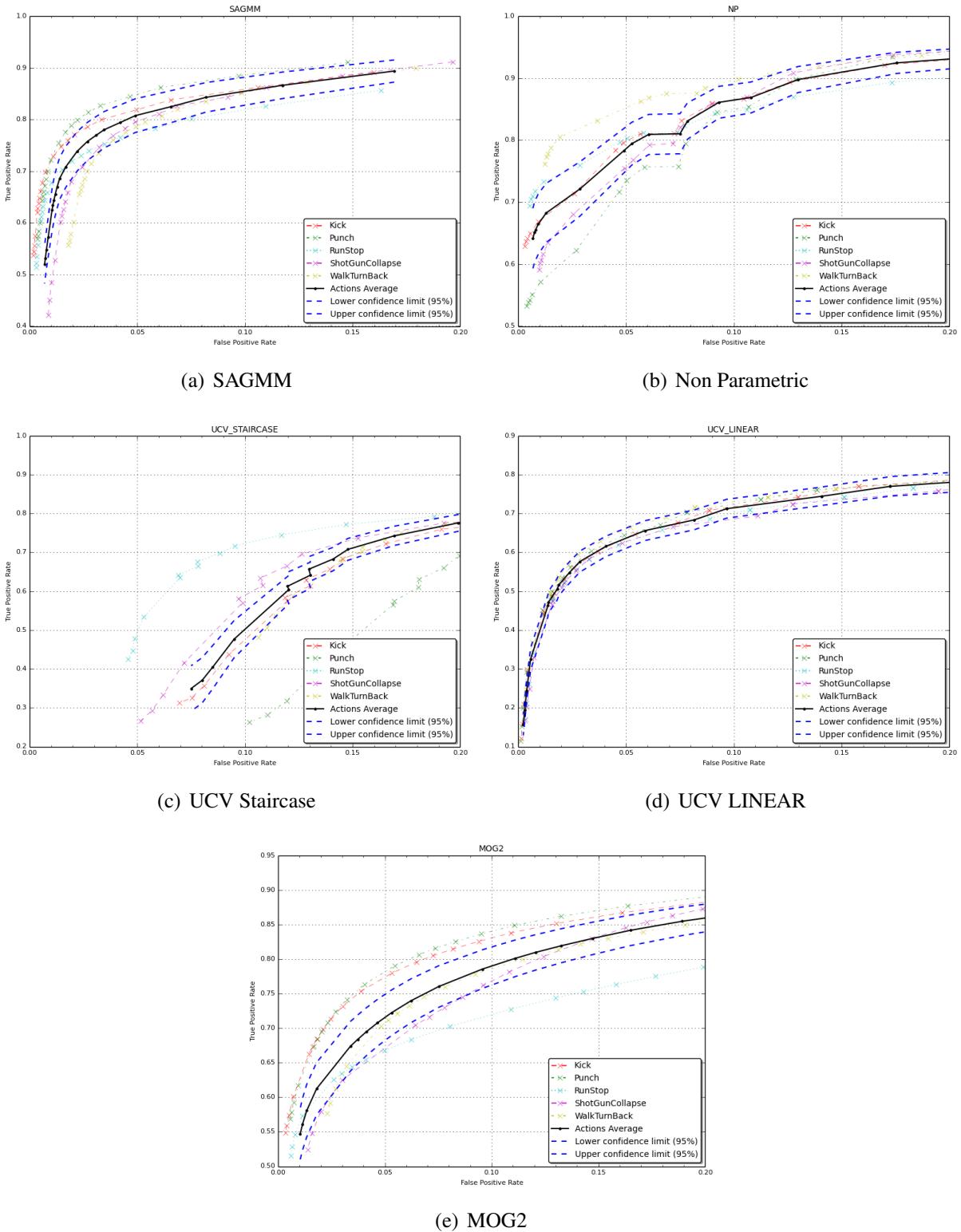


Figura 6.10: Curvas promedios de operaciones características resultantes separadas por algoritmo con intervalo de confianza de 95 % ($\alpha = 0,0002$)

CAPÍTULO 7

CONCLUSIONES

Este trabajo de tesis fue desarrollado en base a tres dimensiones. En la primera parte se propone el objetivo principal de implementar un sistema de software, que permitiese la incorporación de algoritmos de sustracción de imágenes de fondo, empleados en el campo de investigación de visión por computador. En una segunda parte se proyecta el desarrollo de una herramienta de evaluación de desempeño, independiente de los algoritmos incorporados en este sistema, que incluyera un conjunto de métricas utilizadas comúnmente en visión por computador (u otras disciplinas de maquina de aprendizaje computacional, reconocimiento de patrones, sistemas de recuperación de información, etc) y además funcionase como un sistema de evaluación autónomo, con la finalidad de evaluar sólo máscaras de imágenes resultantes sin necesidad de conocer la lógica de funcionamiento del algoritmo en evaluación. Y un último aspecto se relaciona con el conjunto de datos MuHAVI, que actúa como el componente de evaluación del desarrollo de este proyecto de tesis, en base a este conjunto de datos surgen desafíos que se fueron superando durante las distintas etapas de trabajo.

Una de las salidas de este proyecto corresponde al sistema de software, que incorpora un grupo de algoritmos basados principalmente en mixtura de componentes Gaussiano, orientados a realizar sustracción de imágenes de fondo, procedimiento también denominado por sus nombres en inglés, *Background Subtraction* o *Foreground and Background separation*. La implementación del software utiliza como sustento el conjunto de biblioteca de clases C++, disponibles en la plataforma abierta de visión por computador *OpenCV* y desarrollado en una plataforma *Linux* de 32 bits. El sistema final corresponde a un conjunto de bibliotecas C++, archivos xml de configuración y un grupo de programas de computación que permiten la ejecución y utilización de estos algoritmos de visión por computador en cualquier secuencia de video disponible. Este software tiene la flexibilidad de incorporar nuevos algoritmos, y requiere para esto las clases y métodos empaquetadas en

una biblioteca tipo *Linux*. Sin embargo, este último requerimiento crea una restricción importante, y deja fuera por incompatibilidad de sistema operativos, una buena parte de antiguos y nuevos prototipos que la comunidad de investigación en este campo está desarrollando.

El segundo producto del proyecto se manifiesta como respuesta a la inquietud de una búsqueda de evaluación de los algoritmos incorporados en el sistema anterior. Interesaba saber de manera imparcial el rendimiento de los algoritmos utilizados en un conjunto de datos como MuHAVI. Se hace una investigación y se evidencia que existen bastantes métricas de evaluación de desempeño, y la mayoría de ellas basadas en discrepancia, es decir, comparación de una imagen resultado (máscara) con su referencia. En este proyecto se implementa una herramienta que incluye varias de estas métricas, cada una de ellas destaca en forma individual alguna característica de un algoritmo, pero en su conjunto entregan una visión más completa de desempeño. El software es un programa ejecutable que admite dos entradas, las máscaras binarias (imágenes de siluetas) y las referencias (*ground-truth*), el resultado es un archivo de texto con un resumen por imagen de las mediciones realizadas por el programa. Las métricas integradas más importantes, se fundamentan en el resultado de clasificación a nivel de pixel. El programa identifica si un pixel ha sido bien seleccionado y a partir de esa comparación puede determinar el nivel de clasificación de un algoritmo. Identifica el nivel de '*especificidad*' y '*sensitividad*', es decir, realiza una medición de la proporción de positivos y negativos correctamente clasificados. Estas dos mediciones permiten generar un número distinto de métricas orientadas a responder diversas preguntas en la evaluación de desempeño. En el transcurso de la experimentación se logra determinar que el coeficiente de correlación *MCC* y *F-Measure*, en este proyecto, se pueden considerar mediciones de desempeño equivalentes. Lo mismo sucede con las métricas de *PSNR* y *MSSIM* las dos producen similares resultados, por lo que no se requiere la utilización de ambas. Una de las mediciones más relevantes encontradas durante este trabajo, es la métrica *D-Score*, la cual entrega una medición que descarta los falsos positivo lejanos y pondera mayormente los pixeles falsos positivos cercanos a la forma del objeto (siluetas en este proyecto) que se intenta reconocer, de esta manera, se corrobora que esta métrica entrega una buena indicación, de la factibilidad de reconocer la forma de la silueta en procesos posteriores. El mayor inconveniente de esta métrica es el extensivo procesamiento computacional, realiza extensos cálculos de distancia que condiciona su uso a la potencia del equipamiento usado. De todas manera esta restricción computacional en un futuro cercano no será inconveniente, por el vertiginoso avance en potencia de los equipos de computación.

Una tercera parte de este proyecto, es el análisis de los resultados de la experimentación. Todas las conclusiones de desempeño de los algoritmos, se basan en los análisis de mediciones globales promedios obtenidas durante la experimentación. Una medición es el promedio general de resul-

tados parciales de ejecuciones realizadas sobre cada secuencia del conjunto de datos MuHAVI. El promedio de mediciones por algoritmo es un indicador global, que oculta el desempeño particular de los algoritmos evaluados sobre alguna secuencia específica. MuHAVI expone desafíos adicionales, fuera del problema básico de reconocer una acción humana, que puede interesar observar en la evaluación final y se ocultan en las mediciones globales resultantes. El tipo iluminación usado en la construcción de este conjunto de datos genera problemas que pueden resultar difíciles de resolver por los algoritmos e incide en el deterioro de su desempeño final; estos son figuras fantasma (sombras de los actores proyectadas en el escenario por más tiempo de lo necesario), cambios pequeños en la iluminación global que puede confundir al algoritmo e invertir el reconocimiento de una figura del fondo (o viceversa), y las sombras de los actores proyectadas en el escenario que se confunden como una extensión de la figura humana. Sin embargo, el promedio general de las mediciones en su conjunto construye un buen indicador de desempeño de los algoritmos. La etapa de experimentación por ejemplo ha determinado, contrastando curvas de operaciones y comparando las mediciones de las métricas desempeño, que el algoritmo *SAGMM* presenta rendimiento mejor con respecto a los otros algoritmos, y esto se confirma visualmente al observar algunas imágenes de resultados tomadas al azar, que ratifican la calidad de la silueta con respecto a los algoritmo similares.

Con respecto al desempeño individual de los algoritmos escogidos. Se ha demostrado por ejemplo que el algoritmo de Zezhi Chen [8] denominado *SAGMM* es una mejora del algoritmo de Zivkovic y Heijden [47] denominado *MOG2 (Mixture Of Gaussians)*. *SAGMM* agrega esencialmente dos nuevos componentes al algoritmo original. Incorpora, en la etapa previa de procesamiento, un filtro de procesamiento temporal (cadena continua de cuadros dentro de una secuencia) y espacial (filtro gaussiano entre pixeles de un vecindario) que favorece la estabilidad a nivel de imagen y posibilita mejor discriminación de componentes gaussianos durante el procesamiento del algoritmo. Asimismo incluye un factor de iluminación global con el propósito de compensar cambios bruscos de luminosidad. Estos dos nuevos elementos se evidencian en el mejor desempeño de éste sobre *MOG2*. La implementación de este algoritmo, junto con *MOG2*, forma la base del sistema desarrollado para este proyecto. A pesar, que los algoritmos UCV *Staircase* y *Linear* presentan un desempeño desmejorado con respecto a los otros basados en mixtura de componentes Gaussianas, estos algoritmos están orientados a ser incorporados en sistemas embebidos basados en micro-controladores, en consecuencia abordan restricciones y complejidades que constituyen las fortalezas y ventajas de *UCV*, que los otros algoritmos no consideran dentro de sus diseños. Esto último revela una carencia de este trabajo, y es dejar fuera las métricas de evaluación de los tiempos de ejecución, o la medición de la cantidad de recursos computacionales (el uso de memoria o

carga de CPU) requeridos por un algoritmo. Estas mediciones agregarían una nueva dimensión a la evaluación global y van en el camino de destacar particularidades que pueden tener los algoritmos y no se reflejan en el contexto global.

MuHAVI por su parte es un conjunto de datos que proporciona con muchas vistas (disposición de cámaras en diferentes ángulos) de la misma acción (o del conjunto de acciones), eso conlleva resultados con diferentes varianzas de las mediciones globales. La última parte de la experimentación intenta identificar estas varianzas en el resultado global de la curva de operaciones, al obtener los intervalos de confianza del resultado promedio, y generar un rango (o banda) de valores con una probabilidad del 95 % donde se localiza el resultado. Pero se necesita un análisis estadístico más extenso de los resultados, el estudio de los intervalos de confianza sólo considera 20 muestras globales generales por algoritmo (promedio de 5 secuencias, con 2 actores y 2 cámaras) y considera una distribución de muestreo normal de la media de las muestras, para hacer inferencia estadística de la curva de operaciones. Este mismo estudio se podría hacer considerando por ejemplo una distribución binomial por la naturaleza binaria de las muestras (pixel de una imagen en dos estados) o bien una distribución t (*Student*) por el tamaño pequeño de las muestras. Igualmente, se asumió una distribución normal subyacente al estimar la media estadística del resultado de una secuencia como un indicador global (promedio de la tasa de verdadero y falsos positivos). El conjunto de valores que resultan del procesamiento de una secuencia (por algoritmo) constituye un cluster de valores que no necesariamente se pueden estimar como una distribución normal (o binormal con ambos valores), esto por consiguiente deja abierta la inquietud de considerar el promedio general como un buen indicador global de rendimiento. De esta forma se podría buscar otra estadística que mejor identifique el respuesta de un algoritmo, por ejemplo usar la mediana en vez del promedio. Un trabajo futuro debiera intentar estimar la distribución subyacente de los resultados de una secuencia; estimar los sesgo de la distribución, modelar como una distribución binomial, o usar un modelo no-paramétrico para estimar las distribuciones del resultado de procesamiento de las secuencias. Por otra parte, no queda claro donde surge la variabilidad de los resultados, se puede relacionar esta variabilidad con la respuesta de los algoritmos a los cambios sutiles de iluminación en MuHAVI, afecta el color de la ropa de los actores (ropa oscura en un escenario con poca iluminación) en el resultado global.

Finalmente como trabajo futuro, queda pendiente incorporar un nuevo módulo en el sistema de software que implemente algún algoritmo de reconocimiento de acciones. Con el propósito de corroborar los resultados obtenidos en este trabajo de tesis, verificar que los algoritmos evaluados con buen desempeño entregan una buena base para realizar reconocimiento de acciones. Esto abre nuevas preguntas, las métricas para evaluar los algoritmos estudiados sirven igualmente para evaluar

los algoritmos de detección de acciones, se necesitaría un tipo diferente de métricas, el concepto de discrepancia como método de evaluación sigue siendo válido en detección de acciones. En la misma línea, pero por el lado de la implementación de software, es necesario, que todos los algoritmos que se implementen a futuro usen alguna técnica de paralelismo, como cluster con tarjetas GPU, esto reduciría sin duda enormemente los tiempos de experimentación, pero introduce los desafíos técnicos de implementar estos algoritmos en kernels GPU para la ejecución.

Apéndice A

PROCESAMIENTO DE UNA SECUENCIA

En esta última parte explica en forma simple, paso a paso, el funcionamiento de este sistema para procesar una secuencia de video.

Paso 1: Descargar la última versión de software desde repositorio GitHub, y construir el software

```
1 $ git clone https://github.com/jorgesep/BGS.git
2 $ cd BGS
3 BGS $> mkdir build
4 BGS $> cd build
5 BGS/build $> cmake ..
6 BGS/build $> make
```

Paso 2: Ejecutar la aplicación de *bgs_framework* localizada en el directorio bin.

```
1 BGS/build$ ./bin/bgs_framework -i WalkTurnBack-Camera_3-Person1.avi
```

Paso 3: Ejecutar la aplicación de evaluación de desempeño *pmbgs*.

```
1 BGS/build$ ./bin/pmbgs -g ~/Ground-Truth/WalkTurnBackPerson1Camera3 -i
mog2_mask/0
2 5.210804e-01 1.944766e-03 6.413351e-01 6.586170e-01      5.262960e-01
1.828969e-03 6.508549e-01 6.636832e-01      467/467
```


Apéndice B

CURVAS DE OPERACIONES DESAGREGADAS POR SECUENCIA

Se incluyen el resultado obtenido por secuencias de cada algoritmo para $\alpha = 0,001$ y $\alpha = 0,0002$.

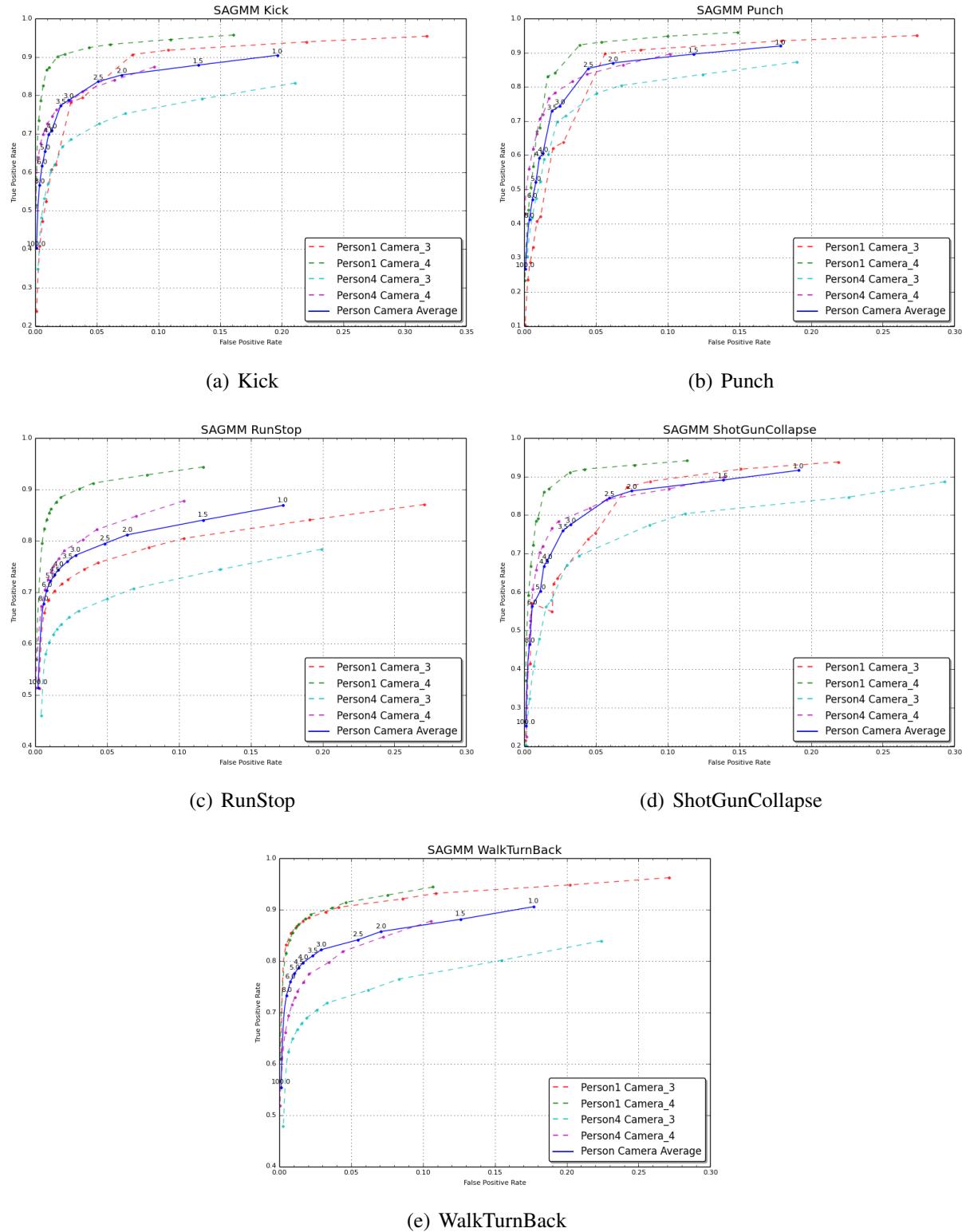


Figura B.1: Resultado global separado por acción de algoritmo SAGMM

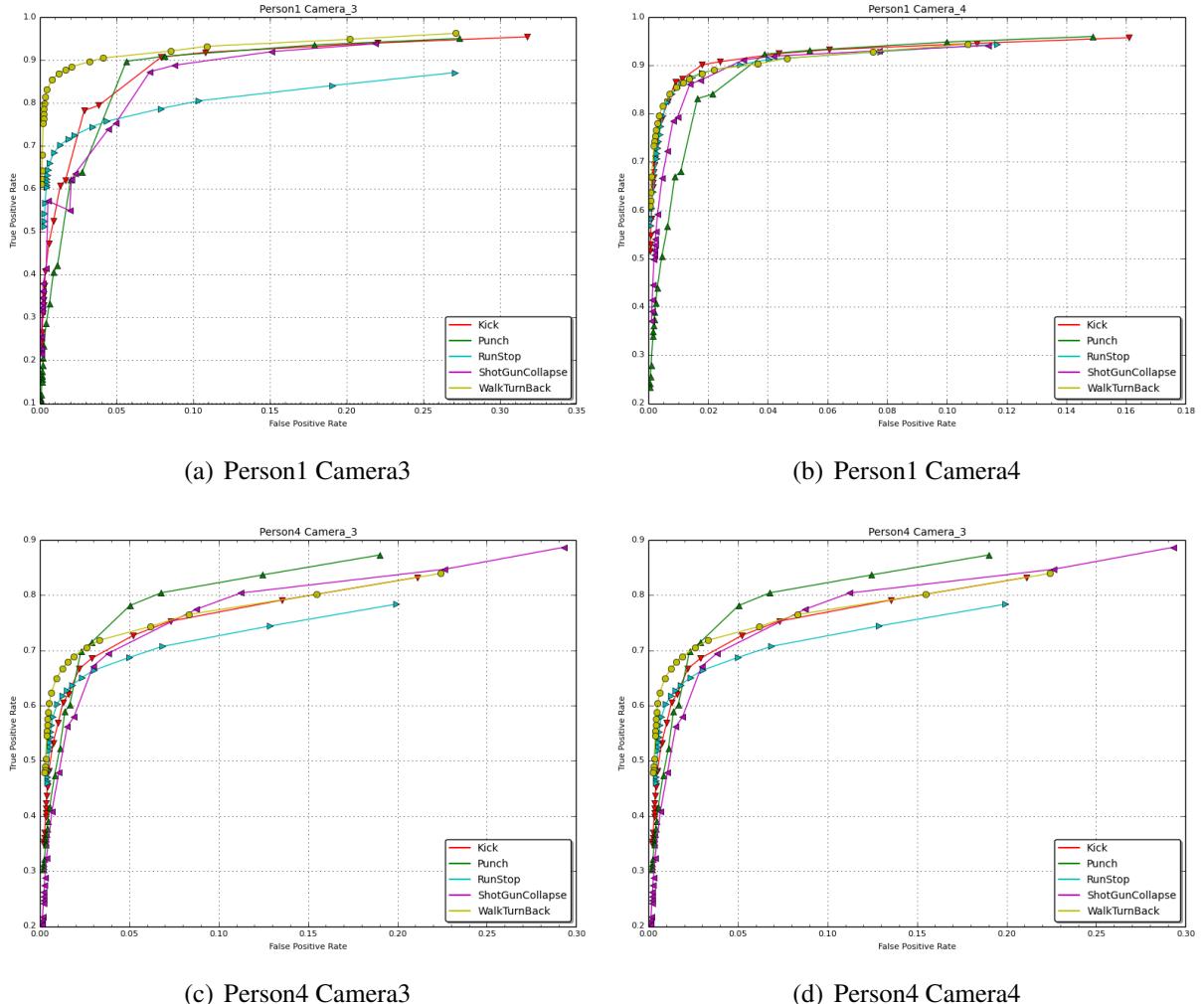


Figura B.2: Resultados globales de la ejecución del algoritmo SAGMM separado por actor y cámara. *Person1* muestra un buen desempeño

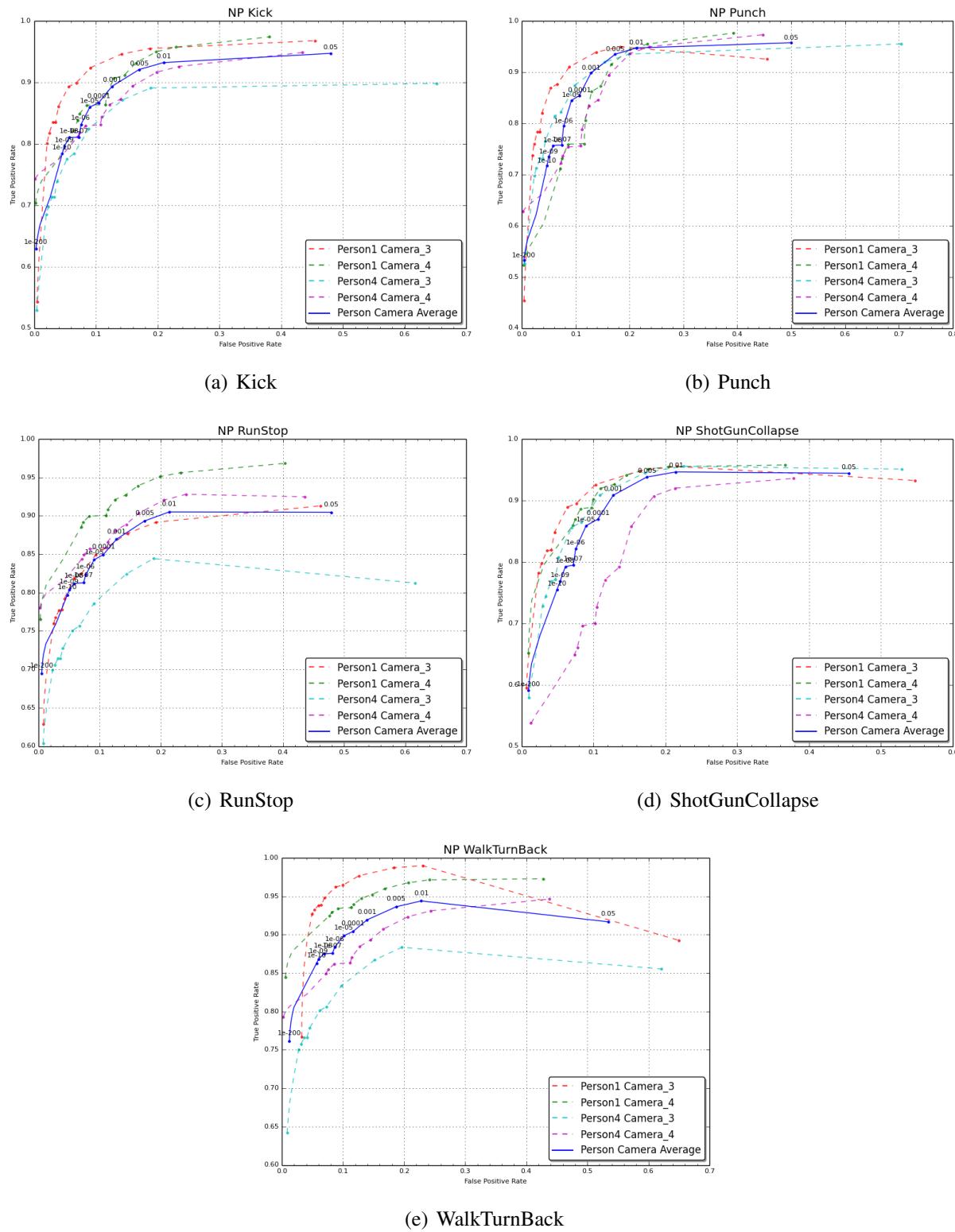


Figura B.3: Resultado global separado por acción de algoritmo NP

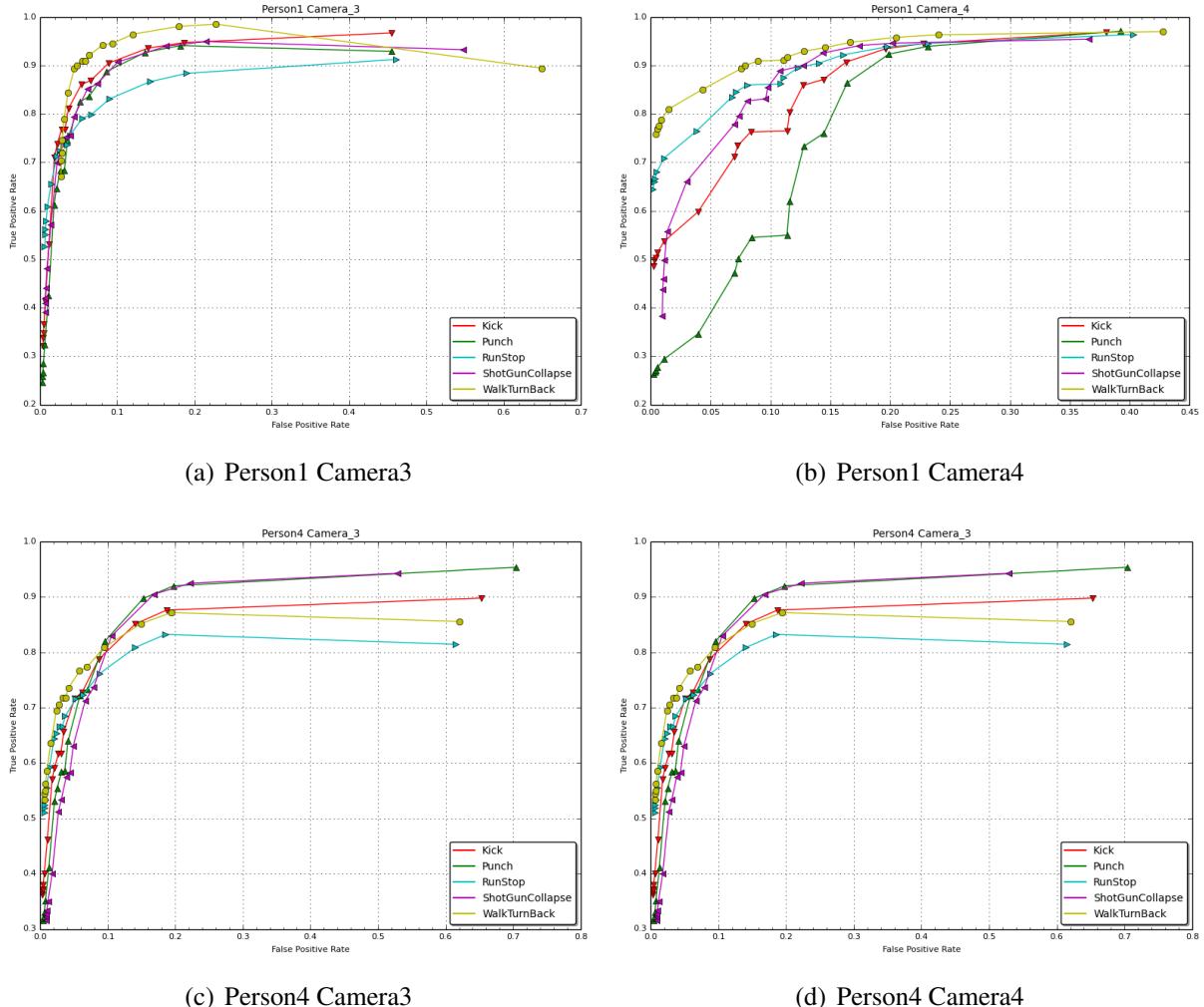


Figura B.4: Resultados globales de la ejecución del algoritmo NP separado por actor y cámara. *Person1* muestra un buen desempeño

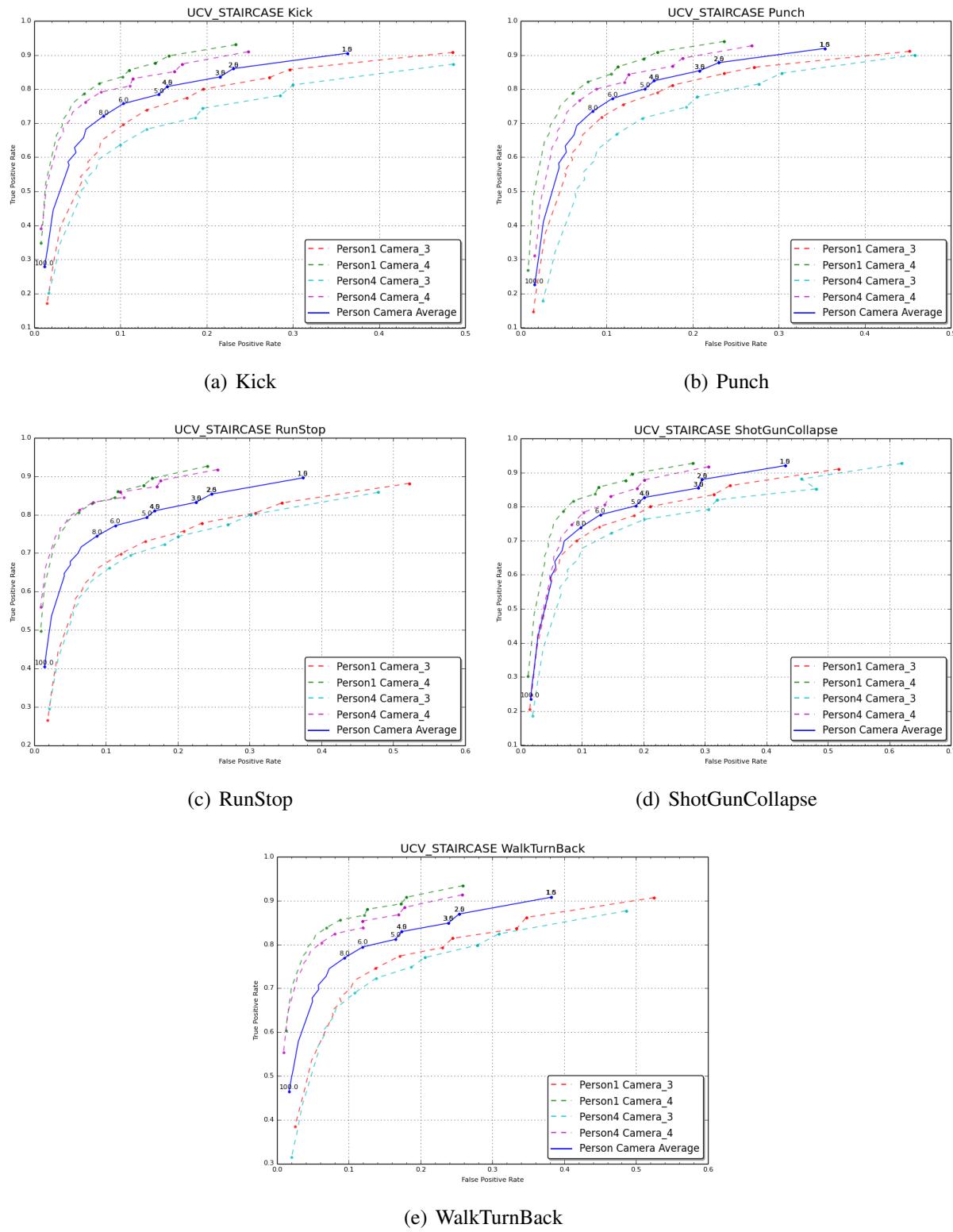


Figura B.5: Resultado global separado por acción de algoritmo UCV_STAIRCASE

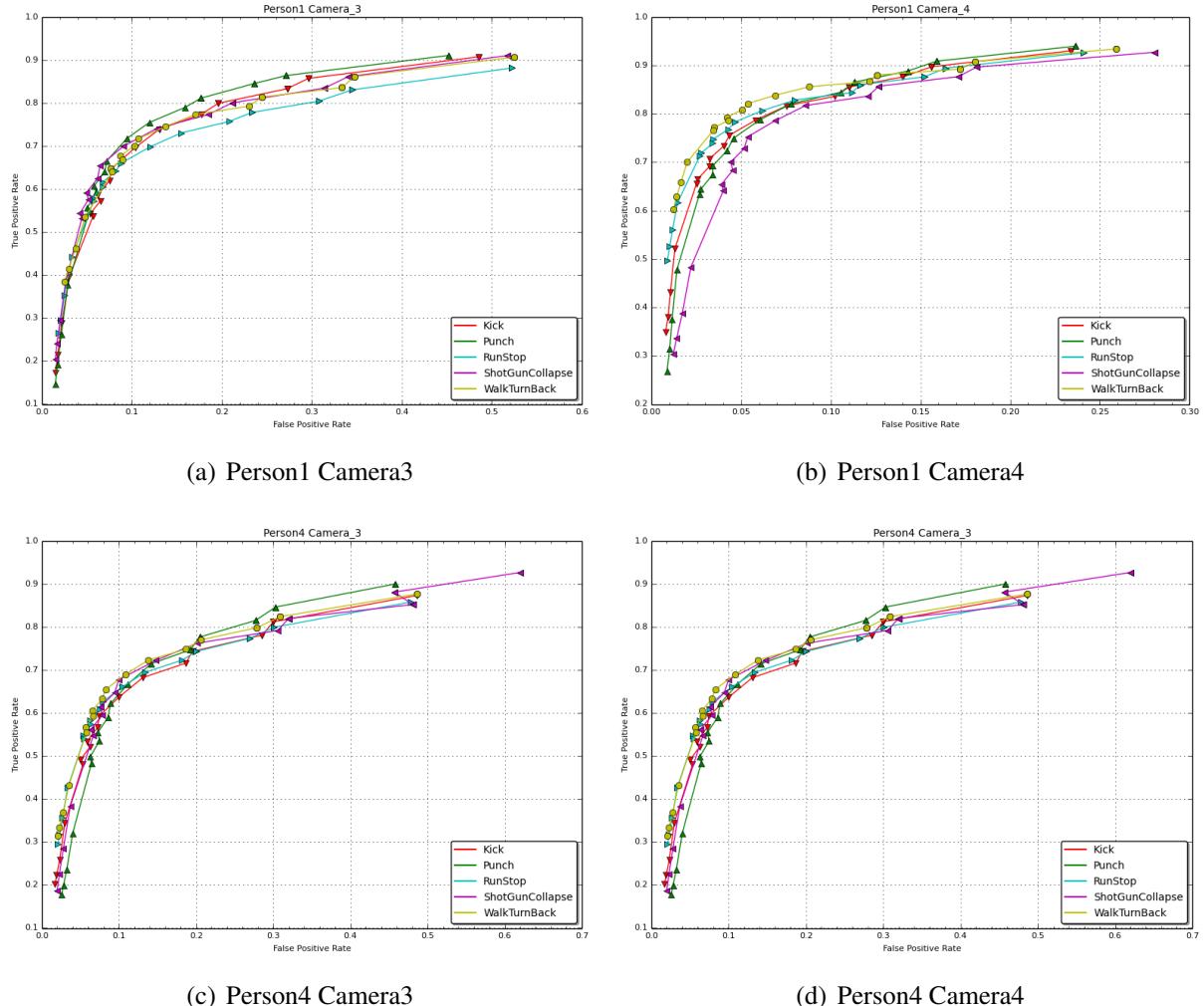


Figura B.6: Resultados globales de la ejecución del algoritmo UCV_STAIRCASE separado por actor y cámara. *Person1* muestra un buen desempeño

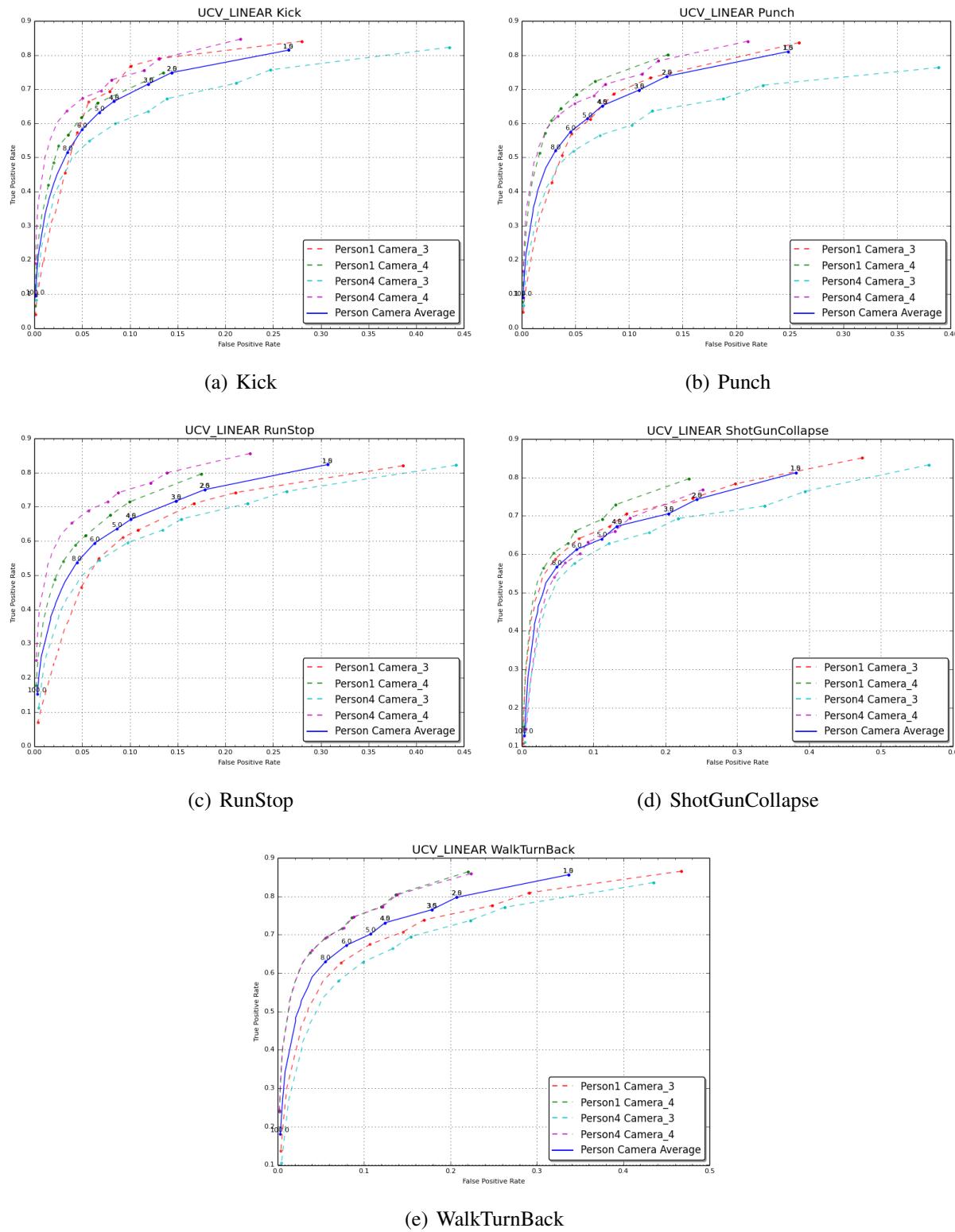


Figura B.7: Resultado global separado por acción de algoritmo UCV_LINEAR

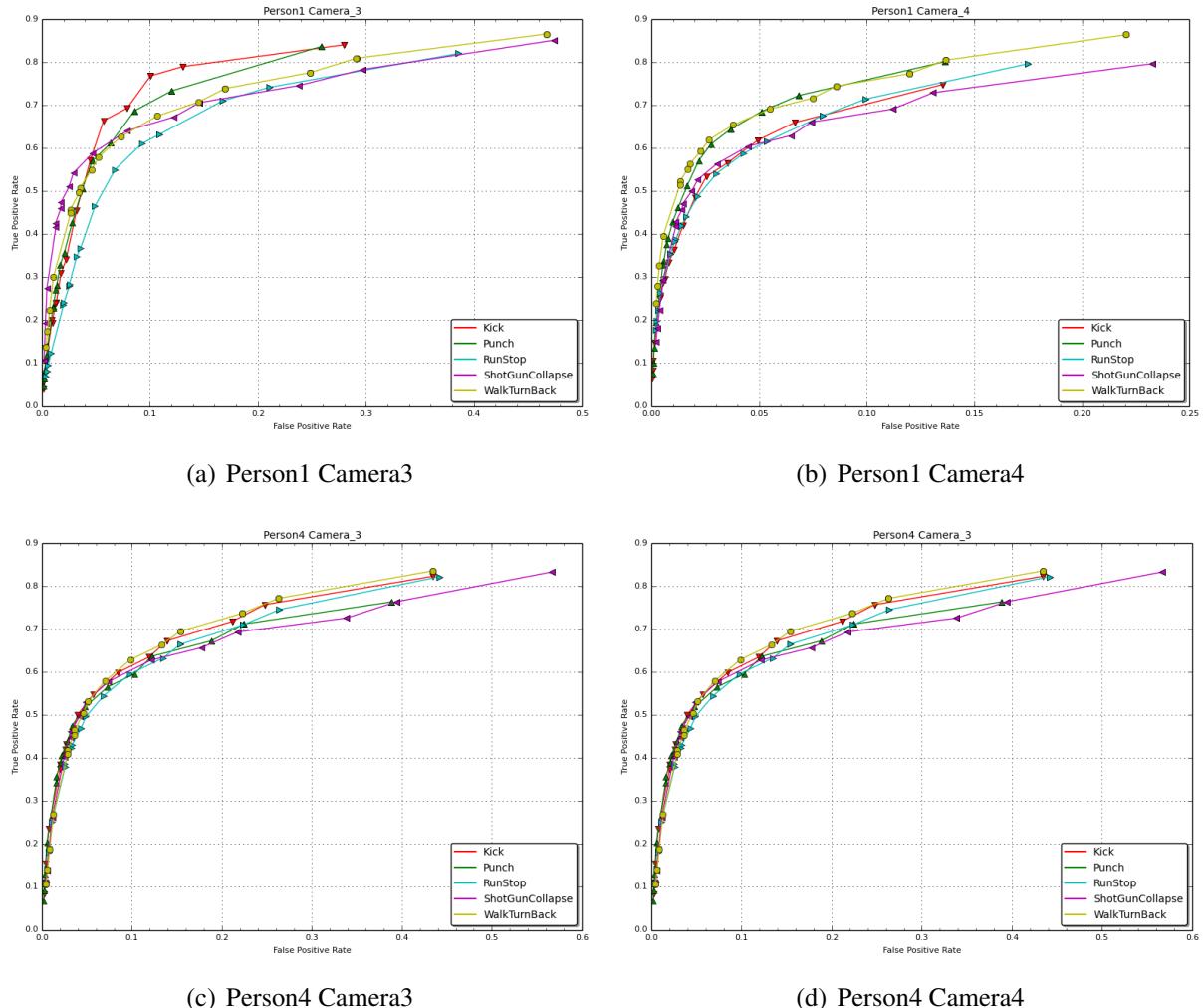


Figura B.8: Resultados globales de la ejecución del algoritmo UCV_LINEAR separado por actor y cámara. *Person1* muestra un buen desempeño

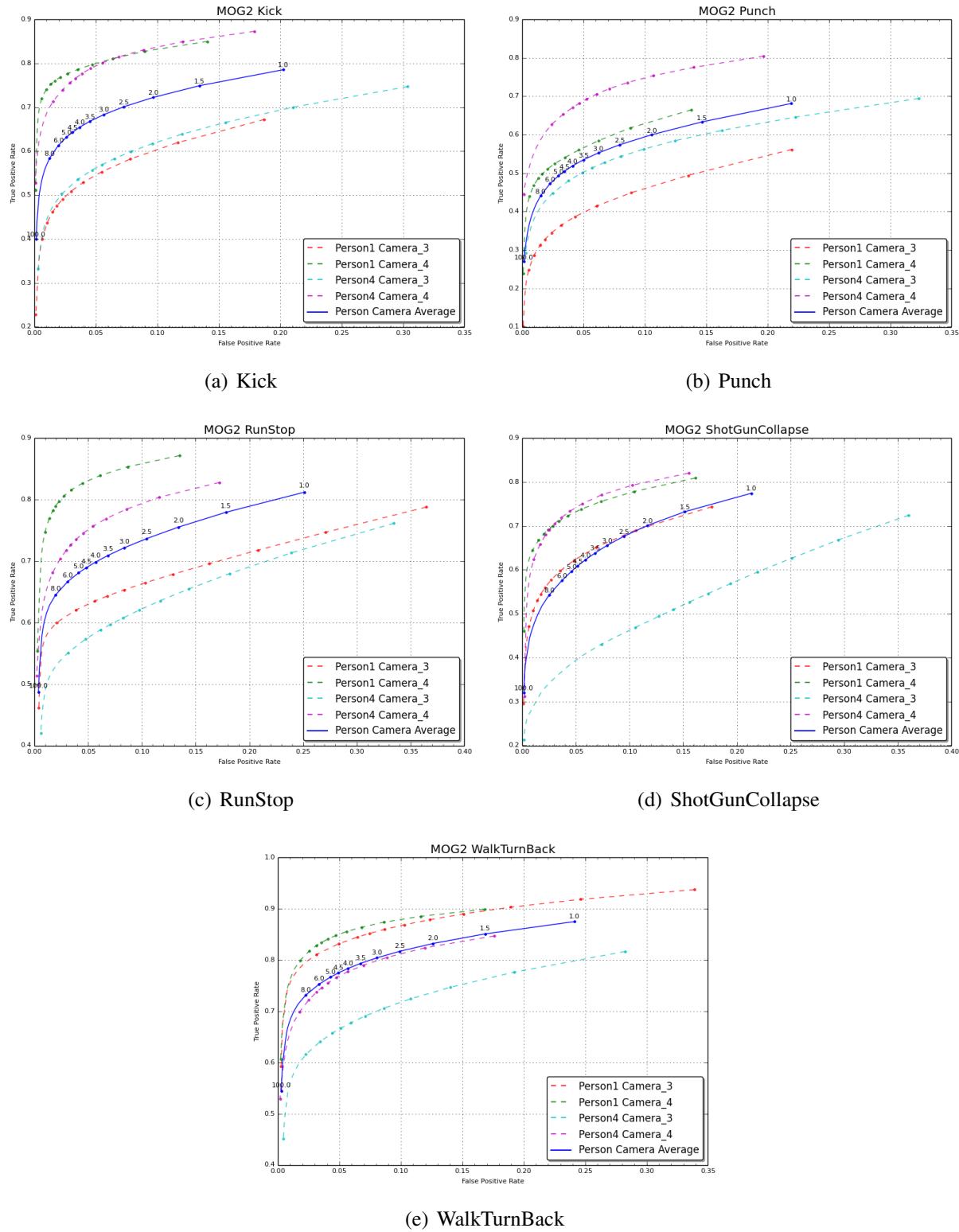


Figura B.9: Resultado global separado por acción de algoritmo MOG2

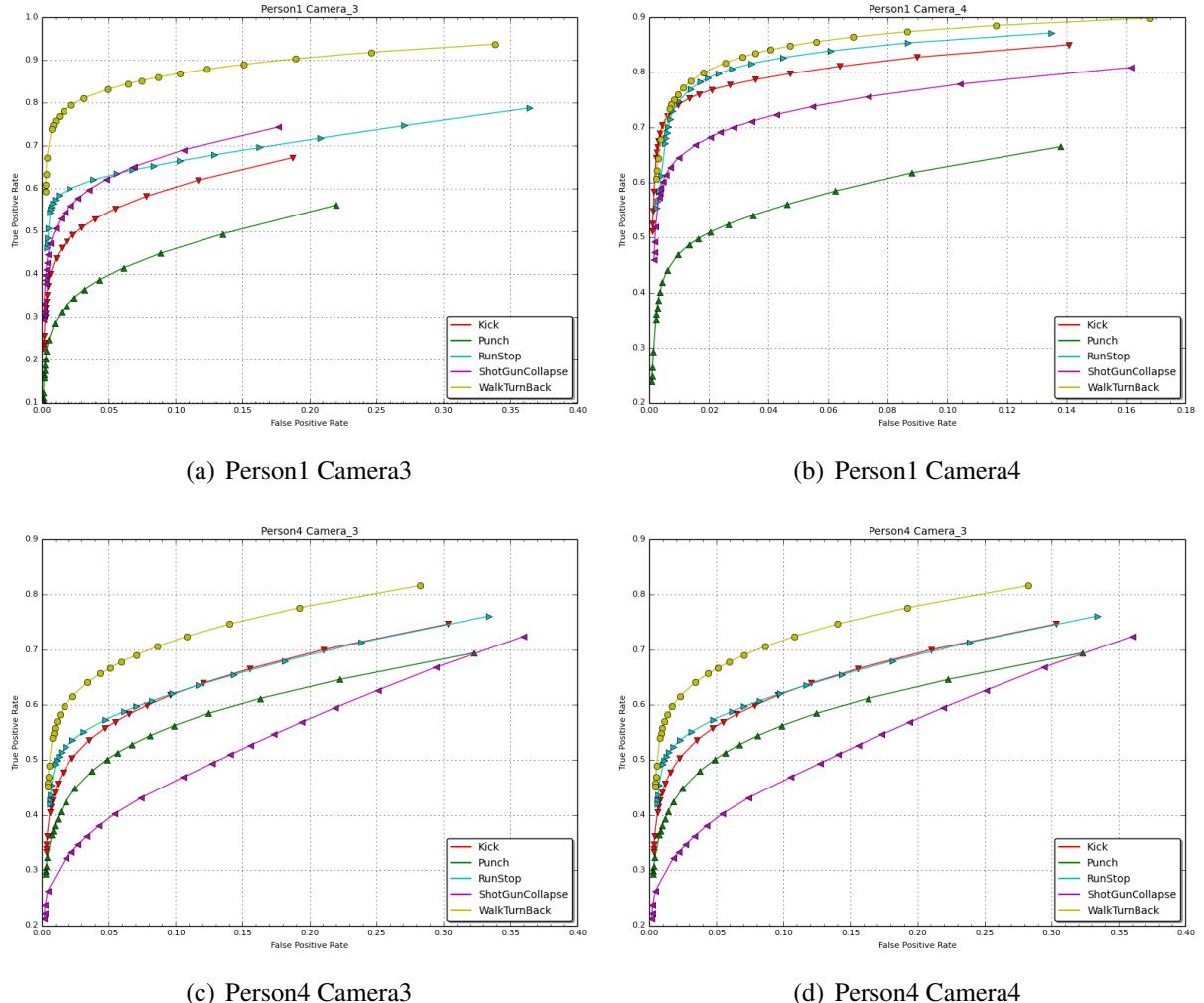


Figura B.10: Resultados globales de la ejecución del algoritmo MOG2 separado por actor y cámara. *Person1* muestra un buen desempeño

Bibliografía

- [1] Y. Benezeth, P.-M. Jodoin, B. Emile, H. Laurent, and C. Rosenberger. Review and evaluation of commonly-implemented background subtraction algorithms. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, 2008.
- [2] boost. Boost c++ libraries, 2013.
- [3] T. Bouwmans. Recent advanced statistical background modeling for foreground detection: A systematic survey. *Recent Patents on Computer Science*, 4(3), 2011.
- [4] Matthew Brand. Structure learning in conditional probability models via an entropic prior and parameter extinction. *Neural Computation*, 11(5):1155–1182, 1999.
- [5] S. Brutzer, B. Hoferlin, and G. Heidemann. Evaluation of background subtraction techniques for video surveillance. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, page 1937–1944. IEEE, June 2011.
- [6] A. Cavallaro, E.D. Gelasca, and T. Ebrahimi. Objective evaluation of segmentation quality using spatio-temporal context. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages III–301–III–304 vol.3, 2002.
- [7] Z. Chen and T. Ellis. Self-adaptive gaussian mixture model for urban traffic monitoring system. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1769–1776, 2011.
- [8] Zezhi Chen, T. Ellis, and S.A. Velastin. Vehicle detection, tracking and classification in urban traffic. In *Intelligent Transportation Systems (ITSC), 2012 15th International IEEE Conference on*, pages 951 –956, September 2012.
- [9] Zezhi Chen, Nick Pears, Michael Freeman, and Jim Austin. Background subtraction in video using recursive mixture models, spatio-temporal filtering and shadow removal. In George

- Bebis, Richard D. Boyle, Bahram Parvin, Darko Koracin, Yoshinori Kuno, Junxian Wang, Renato Pajarola, Peter Lindstrom, André Hinkenjann, Miguel L. Encarnaçāo, Cláudio T. Silva, and Daniel S. Coming, editors, *ISVC (2)*, volume 5876 of *Lecture Notes in Computer Science*, pages 1141–1150. Springer, 2009.
- [10] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.
- [11] Piotr Dollár, Christian Wojek, Bernt Schiele, and Pietro Perona. Pedestrian detection: An evaluation of the state of the art. *PAMI*, 34, 2012.
- [12] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.
- [13] Ahmed M. Elgammal, David Harwood, and Larry S. Davis. Non-parametric model for background subtraction. In *Proceedings of the 6th European Conference on Computer Vision-Part II*, ECCV ’00, pages 751–767, London, UK, UK, 2000. Springer-Verlag.
- [14] A. Ess, B. Leibe, and L. Van Gool. Depth and appearance for mobile scene analysis. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8, 2007.
- [15] Tom Fawcett. An introduction to roc analysis. *Pattern Recogn. Lett.*, 27(8):861–874, June 2006.
- [16] Mario A T Figueiredo and A.K. Jain. Unsupervised learning of finite mixture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(3):381–396, 2002.
- [17] Nir Friedman and Stuart Russell. Image segmentation in video sequences: A probabilistic approach. In *Proceedings of the Thirteenth Conference on Uncertainty in Artificial Intelligence, UAI’97*, page 175–181, San Francisco, CA, USA, 1997. Morgan Kaufmann Publishers Inc.
- [18] Borko Furht and Oge Marques. *Handbook of Video Databases: Design and Applications (pp. 1041-1078)*. CRC Press, Inc., Boca Raton, FL, USA, 1 edition, 2003.
- [19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.
- [20] GitHub. Build software better, together, 2013.

- [21] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2nd edition, 2001.
- [22] Lena Gorelick, Moshe Blank, Eli Shechtman, Michal Irani, and Ronen Basri. Actions as space-time shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 29(12):2247–2253, December 2007.
- [23] Ralph Gross and Jianbo Shi. The CMU motion of body MoBo database. Technical report, 2001.
- [24] Sonsoles Herrero and Jesús Bescós. Background subtraction techniques: Systematic evaluation and comparative analysis. In Jacques Blanc-Talon, Wilfried Philips, Dan Popescu, and Paul Scheunders, editors, *Advanced Concepts for Intelligent Vision Systems*, volume 5807 of *Lecture Notes in Computer Science*, pages 33–42. Springer Berlin Heidelberg, 2009.
- [25] Thanarat Horprasert, David Harwood, and Larry S. Davis. A statistical approach for real-time robust background subtraction and shadow detection. pages 1–19, 1999.
- [26] itseez. Open source computer vision, 2013.
- [27] C. Lallier, E. Reynaud, L. Robinault, and L. Tougne. A testing framework for background subtraction algorithms comparison in intrusion detection context. In *Advanced Video and Signal-Based Surveillance (AVSS), 2011 8th IEEE International Conference on*, pages 314–319, 2011.
- [28] Leyuan Liu and Nong Sang. Metrics for objective evaluation of background subtraction algorithms. In *Image and Graphics (ICIG), 2011 Sixth International Conference on*, pages 562–565, 2011.
- [29] Sofus A. Macskassy and Foster J. Provost. Confidence bands for roc curves: Methods and an empirical study. In José Hernández-Orallo, César Ferri, Nicolas Lachiche, and Peter A. Flach, editors, *ROCAI*, pages 61–70, 2004.
- [30] matplotlib. matplotlib is a python 2d plotting library, October 2013.
- [31] K. McKoen, R. Navarro-Prieto, B. Duc, E. Durucan, F. Ziliani, and T. Ebrahimi. Evaluation of video segmentation methods for surveillance applications. In *Proc. European Signal Processing Conference*, Tampere, Finland, 2000.

- [32] G. J. McLachlan and D. Peel. *Finite mixture models*. Wiley Series in Probability and Statistics, New York, 2000.
- [33] A. Prati, I. Mikic, M.M. Trivedi, and R. Cucchiara. Detecting moving shadows: algorithms and evaluation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(7):918–923, 2003.
- [34] Claudio Salvadori, Dimitrios Makris, Matteo Petracca, Jesus Martinez-del Rincon, and Sergio Velastin. Gaussian mixture background modelling optimisation for micro-controllers. In George Bebis, Richard Boyle, Bahram Parvin, Darko Koracin, Charless Fowlkes, Sen Wang, Min-Hyung Choi, Stephan Mantler, Jürgen Schulze, Daniel Acevedo, Klaus Mueller, and Michael Papka, editors, *Advances in Visual Computing*, volume 7431 of *Lecture Notes in Computer Science*, pages 241–251. Springer Berlin Heidelberg, 2012.
- [35] Christian Schüldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: A local SVM approach. In *In Proc. ICPR*, page 32–36, 2004.
- [36] Sanchit Singh, Sergio A. Velastin, and Hossein Ragheb. MuHAVi: a multicamera human action video dataset for the evaluation of action recognition methods. In *Proceedings of the 2010 7th IEEE International Conference on Advanced Video and Signal Based Surveillance, AVSS ’10*, page 48–55, Washington, DC, USA, 2010. IEEE Computer Society.
- [37] Chris Stauffer and W. Eric L. Grimson. Adaptive background mixture models for real-time tracking. In *Computer Vision and Pattern Recognition*, volume 2, page 2246–2252, 1999.
- [38] K. Toyama, J. Krumm, B. Brumitt, and B. Meyers. Wallflower: principles and practice of background maintenance. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 255–261 vol.1, 1999.
- [39] Antoine Vacavant, Thierry Chateau, Alexis Wilhelm, and Laurent Lequière. A benchmark dataset for outdoor Foreground/Background extraction. In Jong-Il Park and Junmo Kim, editors, *Computer Vision - ACCV 2012 Workshops*, volume 7728 of *Lecture Notes in Computer Science*, pages 291–300. Springer Berlin Heidelberg, 2013.
- [40] P. Villegas and X. Marichal. Perceptually-weighted evaluation criteria for segmentation masks in video sequences. *Image Processing, IEEE Transactions on*, 13(8):1092–1103, 2004.
- [41] Zhou Wang and A.C. Bovik. Mean squared error: Love it or leave it? a new look at signal fidelity measures. *Signal Processing Magazine, IEEE*, 26(1):98–117, 2009.

- [42] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612, 2004.
- [43] Daniel Weinland, Remi Ronfard, and Edmond Boyer. Free viewpoint action recognition using motion history volumes. In *COMPUTER VISION AND IMAGE UNDERSTANDING*, 2006.
- [44] P.J. Withagen, K. Schutte, and F.C.A. Groen. Global intensity correction in dynamic scenes. *International Journal of Computer Vision*, 86(1):33–47, 2010.
- [45] Hui Zhang, Jason E. Fritts, and Sally A. Goldman. Image segmentation evaluation: A survey of unsupervised methods. *Comput. Vis. Image Underst.*, 110(2):260–280, May 2008.
- [46] Z. Zivkovic and F. van der Heijden. Recursive unsupervised learning of finite mixture models. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(5):651–656, 2004.
- [47] Zoran Zivkovic and Ferdinand van der Heijden. Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern Recogn. Lett.*, 27(7):773–780, May 2006.