PROYECTO FINAL DE INGENIERIA

SEGUIMIENTO DE PERSONAS EN SECUENCIAS DE VIDEO

Iacomuzzi, Sebastián – LU 133303

Ingeniería Informática

Tutor:

Negri, Pablo Augusto, U.A.D.E.

Octubre 30, 2017

UNIVERSIDAD ARGENTINA DE LA EMPRESA

FACULTAD DE INGENIERIA Y CIENCIAS EXACTAS

AGRADECIMIENTOS

Quisiera expresar mi más profundo y sincero agradecimiento a todas aquellas personas que con su ayuda han hecho posible la realización del presente trabajo.

En primer lugar, a mi tutor Pablo Negri, por su valiosísima orientación, seguimiento y paciencia. Su experiencia y formación han sido la principal fuente de inspiración y estimulación durante este periodo.

Al personal de la universidad, ya sea docentes, no docentes y alumnos, que a lo largo de estos años me han ayudado a llegar hasta este lugar.

Por último, a los familiares y amigos quienes durante toda esta etapa han apoyado e incentivado mi formación académica.

A todos ellos, muchas gracias.

RESUMEN

Así como usamos nuestros ojos y cerebro para lograr comprender el mundo en que vivimos, la visión artificial tiene como objetivo lograr que una computadora pueda interpretar el contenido de una imagen y a partir de ello logre tomar determinadas decisiones en una situación dada. Podemos decir entonces, que desde la perspectiva de la ingeniería lo que se busca es automatizar ciertas tareas que nuestro sistema visual puede hacer.

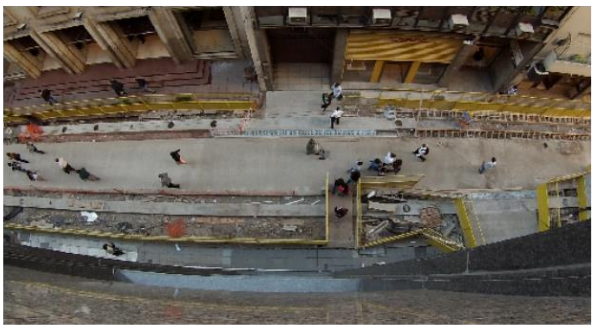
La visión artificial (o visión por computadora) y en concreto el seguimiento de personas es un tema de enorme interés debido al amplio rango de aplicaciones en las que se encuentra inmerso. Es por eso que podemos encontrar un gran número de investigaciones relacionadas con esta materia.

Entre las aplicaciones más populares podemos nombrar las siguientes:

- Control de acceso a áreas especiales (sistemas de vigilancia, control de acceso).
- Identificación de personas a distancia (videojuegos, anteojos inteligentes).
- Estadísticas de movimientos de grupos de personas (diseño de salidas de emergencia, ubicación de la publicidad).
- Análisis de tráfico (ubicación de la señalización y optimización de los caminos)
- Detección de comportamientos anómalos (detección de accidentes o casos de violencia).

La detección y el seguimiento de personas son actividades fundamentales en los trabajos que realizan los urbanistas e ingenieros de tráfico, ya sea, para el diseño de cruces viales seguros con adición de señales para peatones, como para la optimización del tráfico urbano. En la mayoría de los casos, esta detección se realiza de forma manual, pero podría hacerse automáticamente a través de la combinación de diferentes técnicas de visión por computadora.

El trabajo de investigación propuesto incluirá el análisis de distintas técnicas, algoritmos y métodos que permitan lograr tanto la clasificación como el seguimiento de los objetos a partir de un video, el cual permitirá calcular la trayectoria recorrida por cada uno de estos objetos que atraviesan la zona de interés previamente definida.



Para alcanzar nuestros objetivos, el trabajo de investigación y desarrollo se dividirá de la siguiente manera:

- Análisis e implementación de distintas técnicas y métodos de seguimiento. Entre los más representativos, podemos destacar los siguientes:
    - Flujo Óptico (u *"Optical Flow"*) con el método de Lukas y Kanade
    - *"Good Features to Track"*, de Shi y Tomasi
    - Sustracción de Fondo (o *"Background Subtraction"*)
- Desarrollo de un prototipo en lenguaje Python utilizando las librerías de visión artificial de OpenCV, que es una biblioteca libre de visión artificial utilizada en infinidad de aplicaciones, desde sistemas de seguridad con detección de movimiento, hasta aplicaciones de control de procesos donde se requiere reconocimiento de objetos. Esto se debe a que su publicación se da bajo licencia BSD, que permite que sea usada libremente para propósitos comerciales y de investigación. Dicho prototipo tendrá como finalidad primaria la de llevar a la práctica y comprobar la eficacia de cada una de las técnicas y métodos investigados, para luego podes experimentar con las distintas combinaciones y variaciones de los mismos.
- Comparación de los resultados obtenidos con el prototipo con otras investigaciones realizadas por autores reconocidos en el área.

ABSTRACT

Just as we use our eyes and brain to understand the world we live in, artificial vision aims to archive something similar for a computer, teaching it to understand the visual content of an image and as a consequence, to make decisions in a given situation. Hence, we can say that from the perspective of engineering, we want to automate certain tasks the human visual system can do.

Artificial vision (or computer vision) and specifically people tracking, is a topic of enormous interest due to the wide range of uses it presents. That is why we can find a large number of researches or works related to this subject.

Some examples of computer vision application include the following:

- Controlling access in specific resources (surveillance and supervision systems).
- Identification of people at a distance (video games, smart goggles).
- Creating movement statistics for crowds (used in emergency exit and advertising location).
- Traffic analysis (location of signals and road improvement).
- Identification of anomaly behaviors (such as accidents or violent activities).

People detection and people tracking is relevant for traffic engineers and urbanisms, working to design safe crossroads, to locate signals for pedestrians and to optimize urban traffic. On most cases, the identification is carried out manually, but it could be performed systemically through a combination of different computer techniques.

The proposed research project will include the analysis of different algorithms, methods and techniques aiming to classify and follow on objects trough a video sequence, allowing to estimate the circulated path for those in a previously selected region of interest.

In order to achieve those goals, the research and development project will be divided in the following way:

- Analysis and implementation of different techniques and follow-up methods. Among the most representatives, we can mention the following:
  - Optical Flow (Lukas and Kanade method)
  - Good Features to Track (from Shi and Tomasi)
  - Background Subtraction
- Development of a Python prototype using artificial vision tools from OpenCV, which is a free library of artificial vision used in a number of applications (such as security systems with movement detection or control applications requiring object recognition). The main reason for the use of this library is that its publication requires BSD license, which allows it to be freely used for commercial and research purposes. The prototype will be designed for practical purposes, aiming to confirm

effectiveness and efficiency of every technique and method under investigation, and to experiment combinations and variations among them.

- Comparison of the results obtained by using the prototype with research from recognized authors in the same subject.

CONTENIDOS

1. INTRODUCCION

La visión artificial (o visión por computadora) es una disciplina que abarca diversos métodos para la adquisición, procesamiento, comprensión y análisis de las imágenes que podemos capturar del mundo real con el fin de producir información simbólica que pueda ser interpretada por una computadora. Para alcanzar dicha comprensión, podrán entrar en juego numerosas disciplinas como la geometría, la estadística, la física (Figura 1).

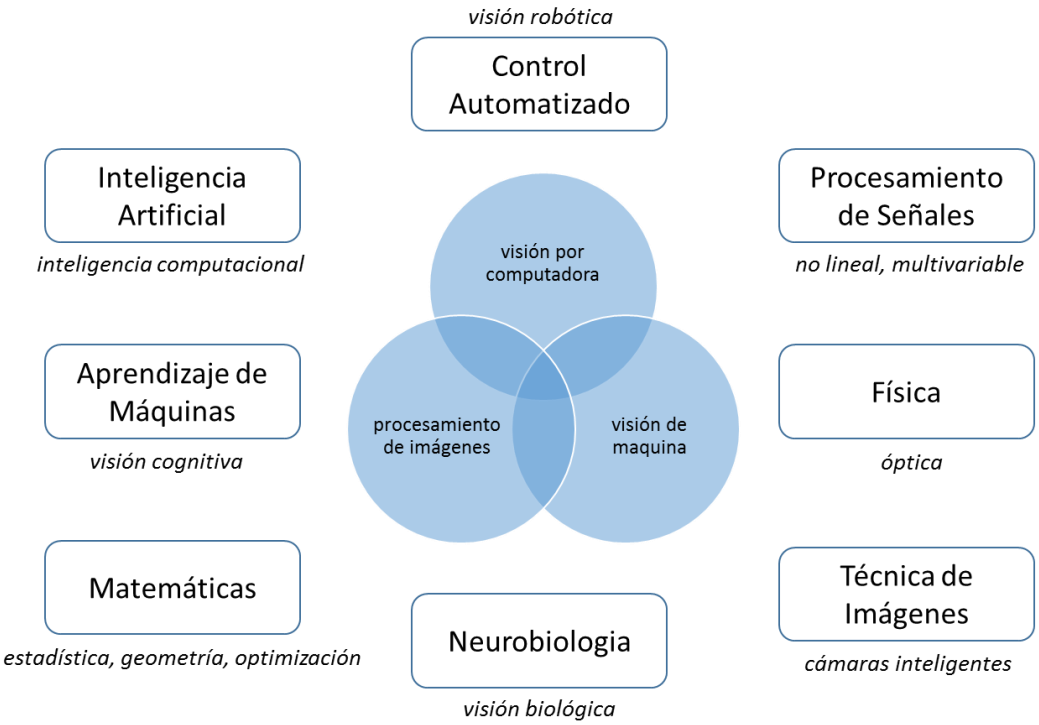


Figura 1: esquema de las relaciones entre la visión por computadora y otras áreas afines.

La adquisición de los datos se consigue por diferentes medios como secuencias de imágenes o vistas desde una o varias cámaras de video y es la primera etapa de siete (Figura 2) para lograr el procesamiento de las mismas.

Figura 2: etapas involucradas en el procesamiento de las imágenes.

1. Digitalización: es la transformación de una imagen analógica, es decir, una imagen natural capturada por algún medio (por ejemplo, una cámara) a otra digital, que será la representación de la imagen para que pueda ser interpretada por una computadora (Figura 3). En esta etapa podrían presentarse algunos problemas como la degradación de la imagen digitalizada (inclusión de ruido o pérdida de definición) generalmente causado por una mala calibración o limitación propia de la tecnología utilizada para la captura.

Figura 3: imagen interpretada por una persona (izquierda) y por una computadora (derecha)

2. Pre-procesamiento: El objetivo aquí es tratar de disminuir la degradación de la imagen para facilitarle el trabajo en las siguientes etapas. Las operaciones típicas que podemos encontrar son la de supresión de ruido y realce de contraste.
3. Segmentación: La idea en esta etapa es extraer la información contenida en la imagen. Para ello, se realiza una descomposición de la misma en unidades homogéneas con respecto a una o más características y tienen una fuerte relación con objetos del mundo real. Estas unidades dependerán de la aplicación (podrían ser por ejemplo, puntos, contornos, siluetas o regiones). Cada objeto de la imagen segmentada deberá ser etiquetado para que pueda ser integrado dentro de una descripción de la imagen original.
4. Representación: En esta etapa se parametrizan los objetos generados por la segmentación (por ejemplo, utilizando coordenadas cartesianas para localizarlos dentro de la imagen).
5. Descripción: Aquí se extrae la información (descriptores o características) de la representación elegida para permitir la posterior clasificación de los objetos (como ejemplo de descriptores podríamos nombrar el área de una región, el perímetro del contorno o los puntos más relevantes dentro del mismo).
6. Reconocimiento: Como dijimos anteriormente, aquí se clasifican los diferentes objetos de la imagen en base a sus descriptores. Los objetos detectados que presentan descriptores semejantes se agrupan en una misma clase.
7. Interpretación: Finalmente, el objetivo aquí es más amplio y dependerá de la aplicación, pero podríamos resumirlo en poder lograr darle un significado a los grupos de objetos formados en la etapa anterior.

Ahora bien, el típico problema en la visión por computadora es el de determinar si efectivamente una imagen contiene un objeto especifico. Los humanos reconocemos un sinfín de objetos en imágenes con muy poco esfuerzo, a pesar del hecho que la imagen del objeto puede variar en diferentes puntos de vista, en diferentes tamaños o escalas e incluso cuando están trasladados, rotados o parcialmente obstruidos. No obstante esta tarea es un gran desafío para los sistemas de visión por computadoras y es por eso que se han implementado numerosos métodos a lo largo de los años para poder mitigar este problema.

Podemos decir entonces que la detección de objetos es una parte fundamental en la visión por computadora que estudia cómo detectar la presencia de objetos en una imagen sobre la base de su apariencia visual, bien sea atendiendo al tipo de objeto (una persona, un automóvil) o a la instancia del objeto (mi automóvil, el automóvil del vecino). Como dijimos anteriormente, se pueden distinguir dos etapas en el proceso de detección:

- La extracción de características de una imagen.
- La búsqueda de objetos basada en dichas características (Figura 4).

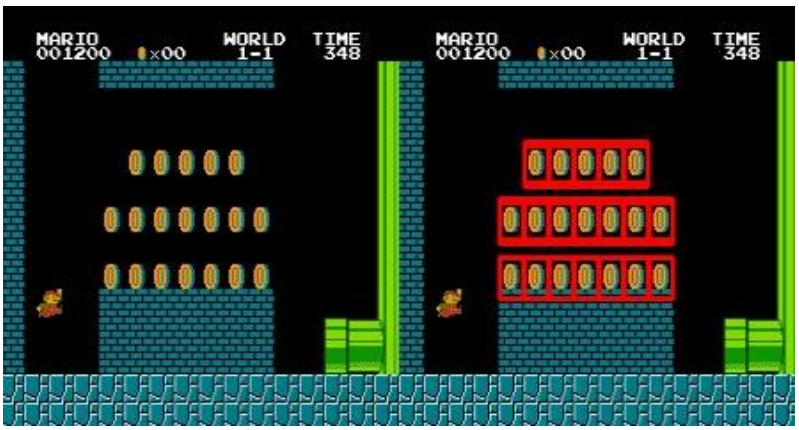


Figura 4: ejemplo sencillo de búsqueda y detección de objetos. Fuente: http://opencv-python-tutroals.readthedocs.io/

La extracción de características consiste en la obtención de modelos matemáticos que "expliquen" el contenido de la imagen con el fin de simplificar el proceso de aprendizaje de los objetos a reconocer. Dichas características son comúnmente llamadas “descriptores”. Podemos encontrar diversos tipos de descriptores, que tendrán mejor o peor rendimiento dependiendo el tipo de objeto a reconocer y a las condiciones del proceso de reconocimiento. Para el proceso de clasificación se pueden usar diferentes técnicas de aprendizaje (como por ejemplo, la regresión logística).

El mayor reto tanto en la extracción de características como la clasificación será entonces el de encontrar descriptores que sean invariantes a los cambios que pueda tener un objeto, como el de su posición o la iluminación.

Por ejemplo, si nos centramos en la detección de peatones (Figuras 5 y 6) que es el tipo de objeto a tratar en el presente trabajo, nos encontraremos con algunos de los siguientes problemas:

- Cada persona tiene una vestimenta totalmente distinta a la de la mayoría de la gente que la rodea.
- Partes del cuerpo de la persona podrían estar siendo obstruidos por algún otro objeto como un automóvil, árbol o pared.
- Cambios de escala, por ejemplo, un niño posee dimensiones totalmente diferentes a las de un adulto.

Figura 5: detección de peatones. Fuente: https://en.wikipedia.org/wiki/Pedestrian_detection

En los capítulos siguientes, analizaremos diferentes técnicas y métodos existentes para lidiar con estos desafíos, tratando no solo de detectar a los peatones a partir de una secuencia de video, sino que también lograr el etiquetado y seguimiento de cada uno de ellos.

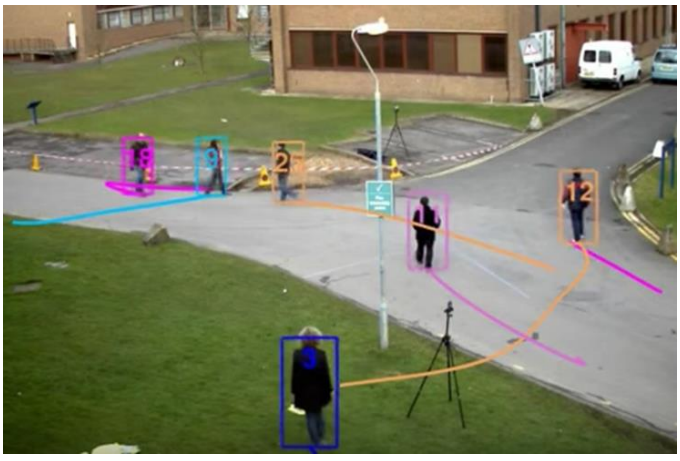


Figura 6: detección y seguimiento de peatones. Fuente: http://www.milanton.de/contracking/index.html

1.1. Objetivos

El objetivo principal de este trabajo es desarrollar una plataforma para la detección y seguimiento de personas en secuencias de video, donde, dada la posición de la cámara que se encuentra justo por encima de la región a analizar (Figura 7), se dificulta la utilización directa de alguno de los métodos conocidos en la actualidad.

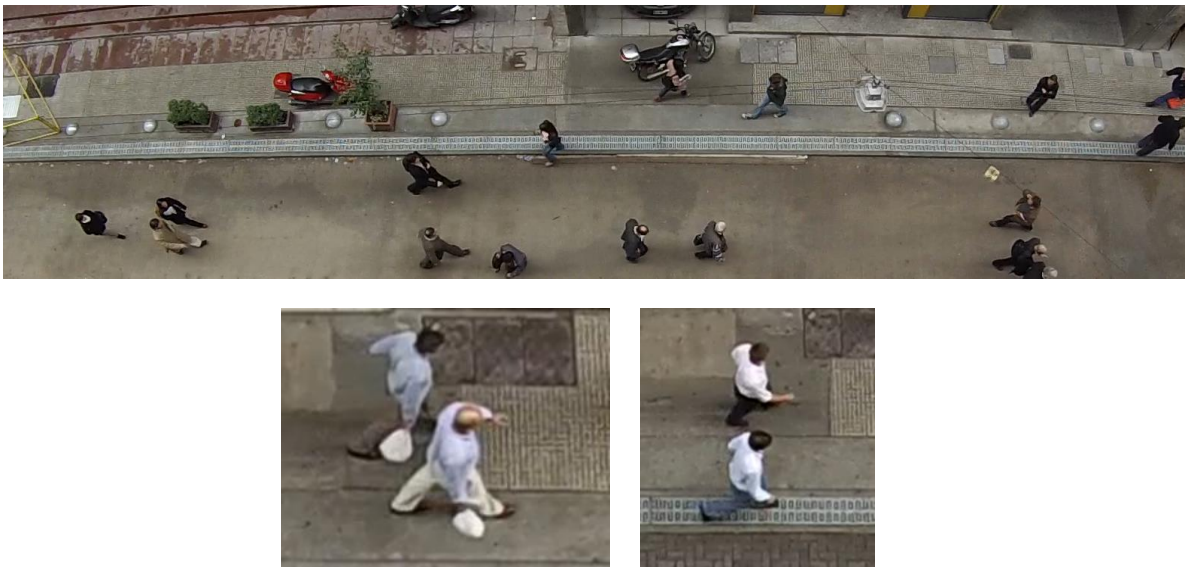


Figura 7: posición de la cámara por encima de los peatones.

Para alcanzar este propósito, se llevarán a cabo las siguientes tareas:

- Análisis de distintos métodos y algoritmos reconocidos en la actualidad sobre detección y seguimientos de objetos y personas en secuencias de video.
- Implementación de un prototipo que ponga a prueba y combine estas técnicas y algoritmos estudiados, que reciba una secuencia de video de peatones como parámetro de entrada y escribe en un archivo las posiciones de cada uno de ellos a lo largo de la secuencia a medida que va detectándolos. Logrando obtener, al finalizar, la trayectoria de cada peatón.
- Comparación de los resultados obtenidos por el prototipo con otros métodos propuestos por autores reconocidos en el área.

1.2. Relevancia del Proyecto

Como comentamos en la sección anterior, actualmente no se dispone de un método eficaz y directo para el seguimiento de personas cuando la posición de la cámara se encuentra por encima de ellas. Además de todas las dificultades que ya describimos que podrían aparecer en cualquier tipo de detección y seguimiento, nos encontraremos con dos obstáculos relevantes:

- No vemos el cuerpo entero de la persona sino que solo visualizamos porciones del mismo, como la parte superior de la cabeza, hombros y piernas.
- La apariencia de cada persona podría sufrir cambios drásticos a lo largo de la secuencia (Figura 8), al cambiar de dirección, agacharse, mover sus brazos, etc.

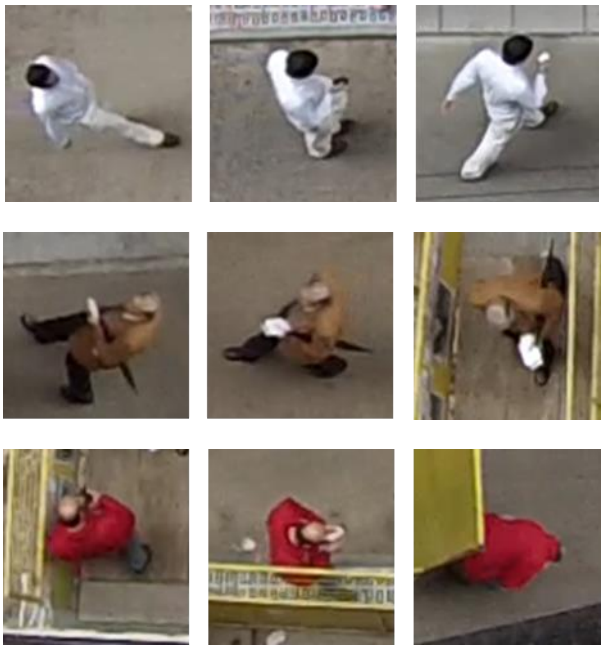


Figura 8: ejemplos de visualización de la misma persona a lo largo de su trayectoria.

El aspecto que más sustenta nuestra investigación entonces es la búsqueda de un sistema que sea capaz de detectar y lograr el seguimiento de las personas bajo estas condiciones, siendo muy útil en secuencias de video obtenidas a partir de cámaras de seguridad, cámaras peatonales o de tránsito.

Vale decir además, que el presente trabajo surgió a raíz de que investigadores de la Universidad Argentina de la Empresa (UADE) y del Instituto Tecnológico de Buenos Aires (ITBA) tenían la necesidad de lograr la detección y seguimiento de peatones para un artículo de investigación publicado en el año 2016 llamado *"Experimental characterization of collision avoidance in pedestrian dynamics"* de Parisi, Negri y Bruno, donde se analiza el comportamiento de los peatones bajo diferentes condiciones para evitar colisionar entre ellos.

En dicha investigación, muchos de los problemas previamente planteados para la detección y el seguimiento fueron simplificados utilizando videos "controlados", es decir, no se utilizaron videos aleatorios, sino que se grabó a un grupo de voluntarios pidiéndoles que usasen todos los mismos sombreros blancos (para simplificar la detección de los mismos) y además que obedezcan a un plan de trayectoria (Figura 9).

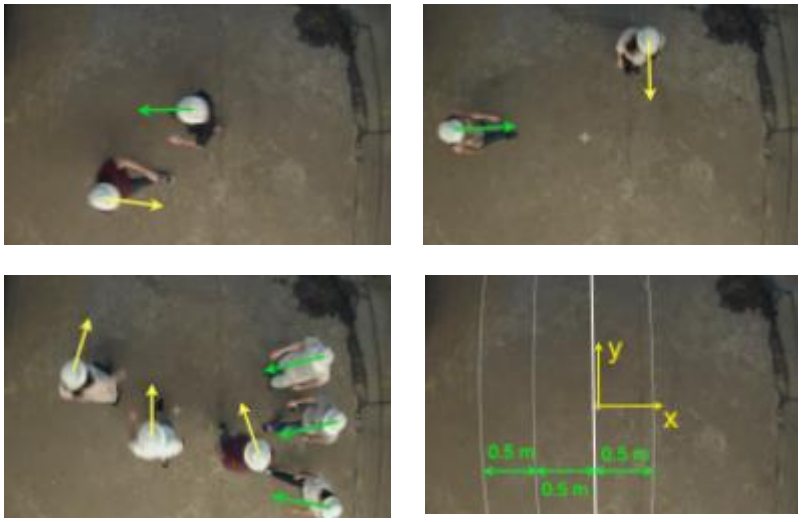


Figura 9: imágenes del experimento, las flechas indican la dirección del movimiento de cada peatón

En el presente trabajo, utilizaremos videos no controlados, obtenidos a partir de cámaras peatonales reales donde las personas tienen un comportamiento aleatorio tanto en su trayectoria como en la vestimenta, por lo que podría usarse para comprobar si los modelos matemáticos desarrollados en el artículo previamente mencionado se ajustan a la realidad.

1.3. Estructura del Informe

Al inicio del presente informe se incluye un breve resumen sobre el contenido del mismo, seguido por una introducción en la cual se establecen los objetivos de la investigación. Bajo el título de *Antecedentes* describiremos la relevancia del tema en la actualidad y como fueron evolucionando a lo largo de los últimos años los sistemas basados en secuencias de video.

Luego de definir la hipótesis de nuestro trabajo, describiremos en detalle todos los métodos y algoritmos sobre el procesamiento de imágenes que estuvimos analizando y cómo, cuáles y por qué fueron aplicados en nuestro sistema para lograr cumplir con los objetivos planteados.

Pondremos a prueba nuestro sistema y analizaremos los resultados obtenidos comparándolos con otros métodos confiables que existen hoy en día.

Para terminar, además de las conclusiones finales, en la sección de **"*Anexos*"** se podrá encontrar un resumen de cómo la presente investigación sirvió de soporte para otro proyecto final donde se buscaba conocer, a través de una cámara web, el comportamiento de un grupo de moscas sometidas a ciertas condiciones.

2. ANTECEDENTES

La detección y seguimiento de objetos a partir de secuencias de video es materia de gran interés debido a numerosas aplicaciones en que podemos utilizarla. Comenzaremos entonces enumerando algunos de los cientos de ejemplos que podemos encontrar en la actualidad:

- Automatización de procesos: robots industriales que logran un manipulado más preciso de las piezas.
- Control de acceso: detección de huella digital o rostro para permitir el acceso a un determinado recurso.
- Control de calidad: robots industriales que logran detectar los defectos más minúsculos.
- Detección de eventos: detección de movimiento y de personas en sistemas de video vigilancia.
- Modelado de objetos o entornos: análisis de imágenes médicas, modelización topográfica.
- Navegación de vehículos: vehículos autónomos, robots móviles.
- Organización de la información: indexación de base de datos de imágenes o secuencias de imágenes.
- Sistemas interactivos: video juegos que capturan los movimientos del jugador.
- Tareas de identificación: sistemas de identificación de especies.

Seguimiento

Podemos describir la tarea de seguimiento de una forma simple como el problema de estimar la trayectoria de un objeto en una imagen plana mientras éste se mueve a lo largo de una escena. En otras palabras, un sistema que realiza un seguimiento dado, se encargará de etiquetar los objetos rastreados en diferentes *"frames"* de un video. Además, dependiendo del dominio y las necesidades del sistema, éste podría ofrecer también información extra de cada objeto, como la orientación, el área o forma del mismo.

Antes de implementar cualquier tipo de sistema de seguimiento deberemos tener entonces bien en claro las respuestas para las siguientes tres preguntas básicas:

1. ¿Qué representación del objeto es la adecuada para el seguimiento?
2. ¿Qué características de la imagen se deben utilizar para lograr la detección?
3. ¿Cómo se debe modelar el movimiento, la apariencia y la forma del objeto?

Problemas frecuentes, simplificaciones y restricciones

Los sistemas de detección y seguimiento de objetos presentan dificultades al momento de implementarlos debido a múltiples factores. Podemos enumerar los siguientes problemas como los más significativos:

- Pérdida de información causada al proyectar el mundo en tres dimensiones (3D) a una representación en dos dimensiones (2D).
- Ruido en las imágenes, generalmente causado por una mala calibración o limitación propia de la tecnología utilizada para la captura (Figura 10).

Figura 10: imagen con ruido. Fuente: http://www.rupj.net/portfolio/docs/cnr-manual/

- Objetos con formas muy complejas. Por ejemplo, en un grupo de personas podemos encontrar apariencias muy diversas: tamaño, postura del cuerpo, ropa, etc. (Figura 11). Pueden darse situaciones en que la ropa se confunde con el fondo de la imagen o la indumentaria de la persona excede lo considerado "normal", como cuando lleva sombrero o una pollera que le cubre las piernas. Las variaciones en el tamaño podrían deberse también a que las personas aparecen a distintas distancias y perfiles con respecto a la cámara de video.

Figura 11: ejemplo de la variedad de colores y tamaños que podemos encontrar en un grupo de personas. Fuente: https://blog.safe.com/2013/11/fmeevangelist119/

- Objetos con movimientos muy complejos. Siguiendo con el ejemplo de las personas, los movimientos de las mismas podrían resultar impredecibles, sobre todo si se mueven en un espacio no controlado, en cualquier momento podrían cambiar de velocidad y/o trayectoria.
- Oclusiones de los objetos, ya sea parcial (Figura 12) o completa.

Figura 12: oclusión parcial.

- Naturaleza no rígida o articulada de los objetos.
- Cambios de iluminación en la escena, sobre todo cuando las imágenes provienen del exterior (Figura 13).

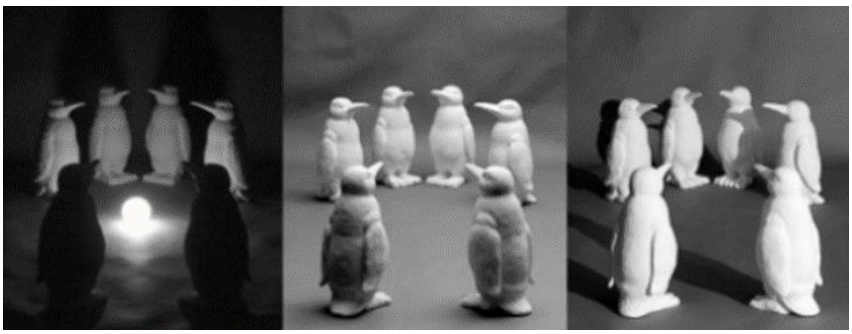


Figura 13: cambios en la iluminación. Fuente: http://slideplayer.com/slide/6370231/

- Limitaciones tecnológicas, principalmente en el procesamiento en tiempo real.

Todos estos aspectos en la mayoría de los casos no lograremos controlarlos. Debido a la gran dificultad que supone desarrollar sistemas que funcionen eficazmente en tales condiciones, lo que busca es tratar de simplificar el problema pre-estableciendo una serie de supuestos y restricciones:

- La mayoría de los algoritmos de seguimiento suponen que el movimiento del objeto es "suave", sin cambios abruptos.
- Se asume que el objeto se desplaza a una velocidad constante.
- Si es posible, resultara útil saber de antemano el número aproximado de los objetos involucrados y conocer cómo será la apariencia y forma de los mismos.

Habitualmente, los sistemas de detección y seguimiento están formados por tres etapas: representación, detección del objetivo en un *"frame"* dado y seguimiento del mismo en los *"frames"* consecutivos.

2.1. Representación de personas

Para lograr la representación de las personas dentro de un sistema debemos primero definir cómo llevaremos a cabo la representación de la forma y la apariencia de las mismas.

Representaciones de formas

- Puntos: La persona está representada por un punto (por ejemplo, su centroide) o por un conjunto de puntos. Es útil cuando cada persona ocupa regiones muy pequeñas de la imagen (Figura 14).

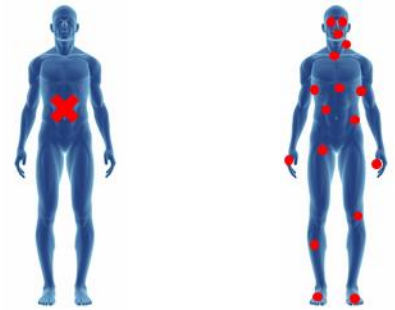


Figura 14: representación de figuras humanas a partir de puntos.

- Formas geométricas primitivas: La representación se da comúnmente a partir de rectángulos o elipses. Aunque las formas geométricas primitivas son más adecuadas para representar objetos rígidos simples, también se utilizan para el seguimiento de objetos no rígidos (Figura 15).

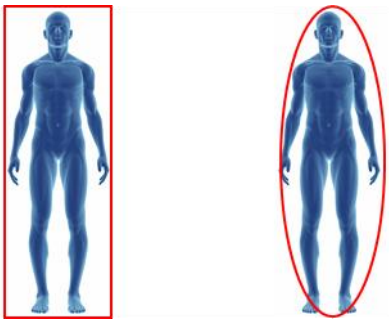


Figura 15: representación de figuras humanas a partir de formas geométricas primitivas.

- Contorno y Silueta: Contorno es el límite de la persona, por lo que la región dentro del contorno es lo que llamamos silueta. Las representaciones de siluetas y contornos son adecuadas para el seguimiento de formas no rígidas complejas (Figura 16).

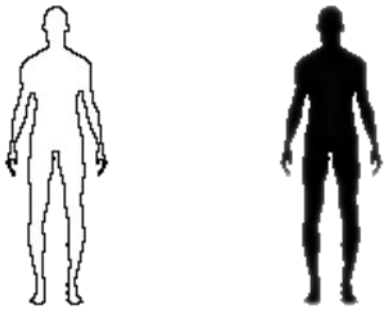


Figura 16: representación de figuras humanas a partir de su contorno (izquierda) y silueta (derecha).

- Modelos de forma articulada: Los objetos articulados están compuestos de partes del cuerpo (modeladas por cilindros o elipses) que se mantienen unidas con las articulaciones. En las personas tendremos el torso, piernas, manos, cabeza y pies conectados por articulaciones (Figura 17).

Figura 17: representación de figuras humanas a partir de un modelo de forma articulada.

- Modelos esqueléticos: El esqueleto de la persona se puede extraer aplicando la transformación del eje medial a la silueta de la misma. La representación esquelética puede usarse para modelar objetos articulados y rígidos (Figura 18).

Figura 18: representación de figuras humanas a partir de un modelo esquelético.

Representaciones de apariencias

- Plantillas o *"Templates"*: Está formado por formas geométricas simples o siluetas. Esta representación es adecuado para el seguimiento de personas cuyas posturas no varían considerablemente durante el seguimiento. Aun así, es posible lograr la auto-adaptación de la plantilla a lo largo del proceso.
- Densidades de probabilidad de la apariencia del objeto: pueden ser paramétricas (Gaussianas o Mixturas Gaussianas) o no paramétricas, como los histogramas (Figura 19) o ventana Parzen.

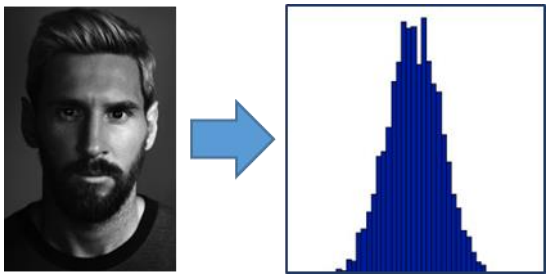


Figura 19: imagen en niveles de gris de Lionel Messi y su correspondiente histograma.

Selección de características o *"features"* para el seguimiento

En general, la propiedad más deseable de una característica visual es su singularidad, de manera que los objetos puedan distinguirse fácilmente en el espacio a partir de la misma. Las características elegidas más comúnmente son las siguientes:

- Color: Cualquier representación de colores es válida, no hay una mejor que otra. Las más populares son: RGB, HSV, L*u*v y L*a*b.
- Bordes: Menos sensible a los cambios de iluminación en comparación con la característica de color. Los algoritmos que buscan detectar los límites de un objeto suelen utilizar esta característica. Unos de los métodos más populares para la obtención de bordes, por su simplicidad y precisión, es el detector de bordes Canny (Figura 20), desarrollado por John Canny en 1986.

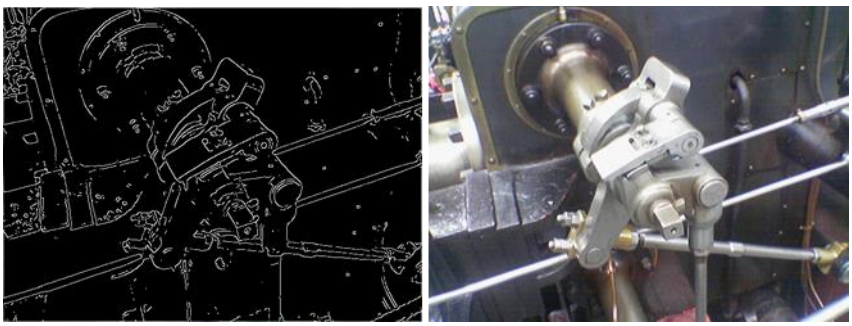


Figura 20: el detector de bordes Canny aplicado a una fotografía a color. Fuente: https://en.wikipedia.org/wiki/Canny_edge_detector

- Textura: Medida de la variación de intensidad de una superficie que cuantifica propiedades tales como suavidad y regularidad.

2.2. Detección de personas

Una vez decidido cómo vamos a representar a las personas en nuestro sistema, el siguiente paso es comenzar con la detección de las mismas. A continuación enumeraremos las técnicas más utilizadas junto con implementaciones de cada una de ellas.

Detector de puntos

La premisa aquí es encontrar puntos de interés que correspondan a zonas donde la textura local maximiza algún criterio. Estos puntos de interés no son más que mínimos o máximos locales de algún tipo de filtro que se le aplico a la imagen original. Las implementaciones más populares son las siguientes:

- Detector de esquinas Harris y Stephens (1988): consiste básicamente en la búsqueda de esquinas. Una esquina se caracteriza por ser una región de la imagen con cambios de intensidad en diferentes direcciones.
- Detector KLT de Kanade, Lucas y Tomasi (1991~1994): similar al detector de Harris pero con un mejorado criterio de selección.
- Detector de puntos SIFT de Lowe (1999): del inglés *"Scale Invariant Feature Transform"*, permite detectar y describir puntos de interés en regiones locales de una imagen que sean invariante a la orientación o escala. Puede presentar problemas en detecciones en tiempo real debido a la gran cantidad de recursos que necesita para su funcionamiento.

En la figura 21 podemos observar la detección de puntos en una misma imagen utilizando los tres métodos mencionados.

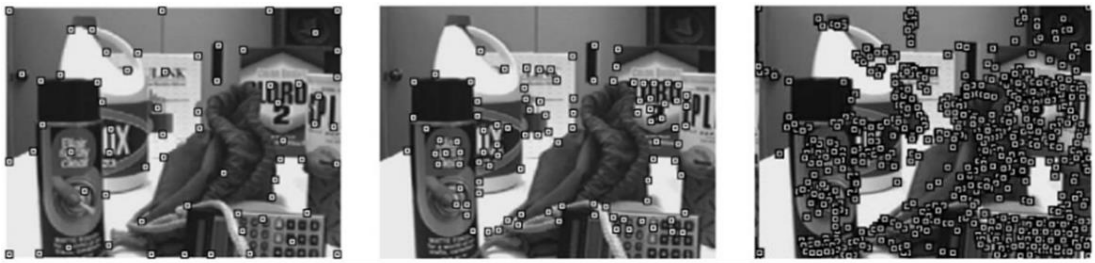


Figura 21: detección de puntos de interés aplicando (a) Harris, (b) KLT y (c) SIFT. Fuente: https://books.google.com.ar/books?id=cMioBQAAQBAJ

Sustracción de fondo

Con esta técnica, la detección de objetos se puede lograr mediante la construcción de una representación de la escena llamada el modelo de fondo (o *"background model"*) y luego lo que se trata es encontrar las desviaciones del modelo para cada *"frame"* siguiente. Cualquier cambio significativo en una región de la imagen desde el modelo de fondo va a

significar un objeto en movimiento. Por lo que la condición más importante para usar esta técnica será que la cámara de video se encuentra en una posición fija.

En la actualidad podemos encontrar un gran número de algoritmos para el modelado de fondo, a continuación enumeraremos los modelos más conocidos.

- Modelos Básicos: histograma, clasificación por intensidad de pixel.
- Modelos Estadísticos: gaussianos, basados en aprendizaje subespacial.
- Modelos de Agrupamiento: agrupamiento secuencial básico
- Modelos de Redes Neuronales: multi-evaluadas, competitivas, mapas de auto-organizado de crecimiento jerárquico.
- Modelos de Estimación: filtro de Wiener, filtro de Kalman.
- Modelos de Fondos Estadísticos Avanzados: de mezclas, híbridos, no paramétricos.
- Modelos Difusos: detección de fondo difusa, post-procesado difuso.
- Modelos de Aprendizaje Subespacial: discriminatorios, mixtos.
- Modelos Subespaciales Robustos (RPCA)
- Modelos de Minimización *"low-rank"* (LRM)
- Modelos *"Sparse"*: detección de compresión, de diccionario.
- Modelos de Dominio Transformado: transformación rápida de Fourier, transformada de Walsh, transformada discreta del coseno.

También existen varias investigaciones realizadas a partir de estas técnicas y algoritmos, entre las más conocidas podemos nombrar las siguientes:

- *"Background Modeling using Mixture of Gaussians for Foreground Detection"* de Bouwmans, El Baf y Vachon (2008): Propone la utilización de mixturas Gaussianas (MoG) para el modelado del fondo.
- *"Subspace learning for background modeling"* de Bouwmans (2009): Propone una interesante clasificación y comparación de los métodos que se basan en la reducción del número de variables aleatorias, comúnmente llamado "reducción de dimensionalidad".
- *"Statistical Background Modeling for Foreground Detection"* de Bouwmans (2009): Es otra clasificación y comparación de métodos existentes, esta vez de aquellos que utilizan modelos estadísticos.

Modelo de espacio-estado

Un enfoque alternativo a la sustracción de fondo es representar las variaciones de intensidad de un pixel en una secuencia de imágenes como estados discretos correspondientes a los eventos del entorno. Por ejemplo, para el seguimiento de automóviles en una ruta, los píxeles de la imagen pueden estar en el estado de fondo, el estado de primer plano (el automóvil en movimiento) o en el estado de la sombra.

Aprendizaje supervisado

Es una técnica de *"machine learning"* que utiliza un conjunto de datos pre-establecido (llamados *"dataset"* o datos de entrenamiento) para realizar predicciones. Estos datos de entrenamiento consisten en pares de objetos, un componente son los datos de entrada y el otro los resultados deseados. A partir de los mismos, el algoritmo de aprendizaje supervisado busca construir un modelo que pueda hacer predicciones de los valores de respuesta para un nuevo conjunto de datos. El uso de conjuntos de datos de formación más grandes a menudo producen modelos con mayor poder predictivo que pueden generalizarse bien para nuevos conjuntos de datos.

Entre las técnicas más utilizadas podemos nombrar las siguientes:

- Máquinas de vectores de soporte o SVM (del inglés *"Support Vector Machines"*): A partir de un conjunto de puntos (que a la vez es un subconjunto de un conjunto mayor) y donde cada uno de los puntos pertenece a una de dos categorías posibles, este algoritmo construirá un modelo capaz de predecir a cuál de estas dos categorías pertenece un nuevo punto.
- Mejora adaptativa o AdaBoost (del inglés *"Adaptative Boost"*): La idea principal es lograr la combinación de los resultados de varios clasificadores débiles hasta obtener un clasificador robusto.
- Redes Neuronales: Se trata de la simulación de las propiedades observadas en los sistemas neuronales biológicos a través de ciertos modelos matemáticos. Cada neurona (así se le llama a las unidades de la red) recibe "n" entradas y emite una salida. Dichas entradas se ponderan con pesos que se "aprenden" durante el proceso de entrenamiento, buscando así minimizar el error de clasificación. La salida, por su parte, viene dada por dos funciones:
    - Función de propagación: es la sumatoria de cada entrada multiplicada por el peso de su interconexión.
    - Función de activación o transferencia: transforma el resultado de la función de propagación en la salida real de la neurona mediante un proceso algorítmico. Es utilizada para acotar dicha salida. Las más elegidas son las funciones sigmoideas y la tangente hiperbólica.

2.3. Seguimiento de personas

El objetivo en el seguimiento de personas (o de cualquier otro objeto) es generar la trayectoria de la misma a lo largo del tiempo mediante la localización de su posición en cada *"frame"* del vídeo, obteniéndose información del tipo espacial representada por algún sistema conocido, por ejemplo, coordenadas cartesianas (Figura 22). Aunque también podría proporcionar la región completa en la imagen que está ocupada por la persona en cada instante de tiempo. Las tareas de detectar a la persona y establecer correspondencia entre las instancias de ella a través de distintos *"frames"* pueden realizarse por separado o conjuntamente.

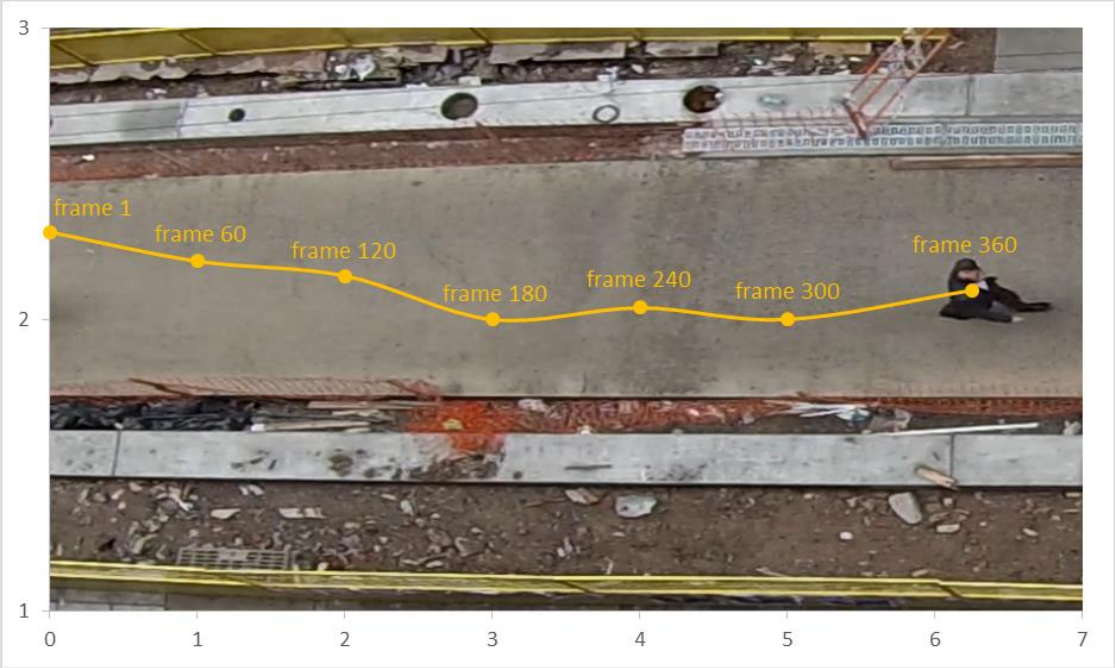


Figura 22: ejemplo de la representación de la trayectoria de una persona utilizando coordenadas cartesianas.

Actualmente existe una gran variedad de técnicas para alcanzar el seguimiento de cualquier clase de objetos. Las mismas se resumen en el cuadro de la figura 23.

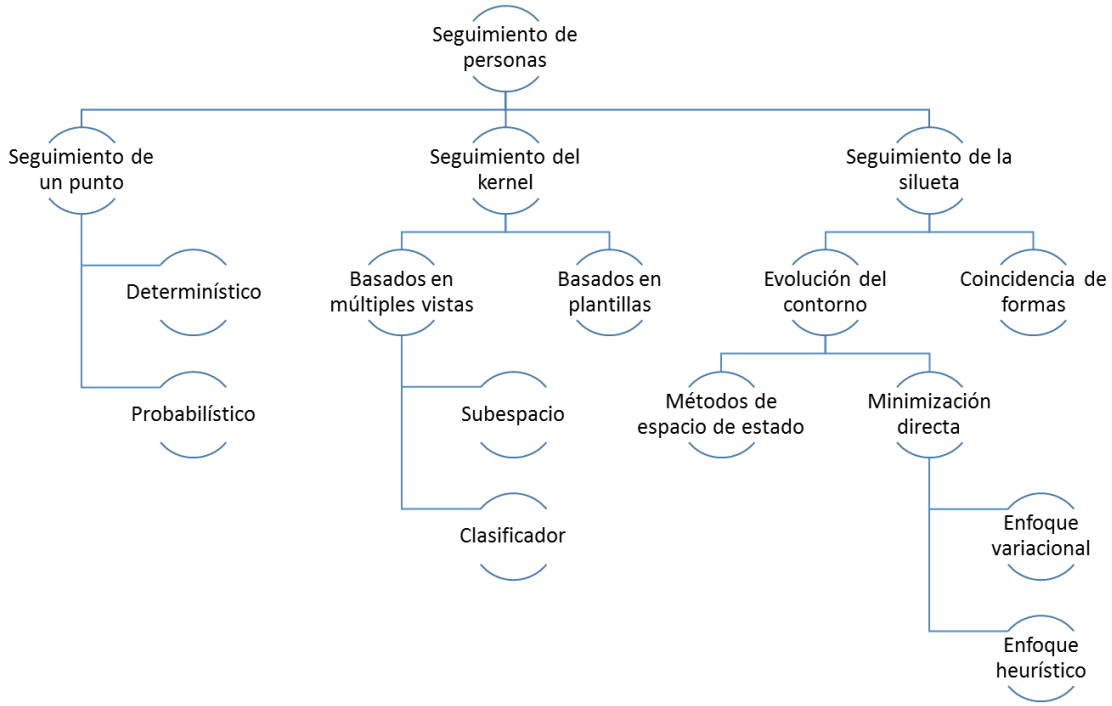


Figura 23: clasificación de las distintas técnicas para el seguimiento de objetos.

Seguimiento de un punto

El seguimiento es formulado como la correspondencia de objetos detectados representados por puntos a través de los distintos *"frames"* (Figura 24).

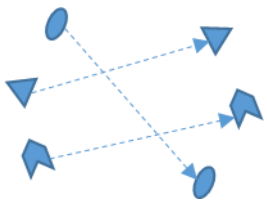


Figura 24: seguimiento de un punto.

- Métodos de correspondencia determinístico: Definen el costo de asociar cada objeto en el *"frame"* t - 1 a un solo objeto en el *"frame"* t usando un conjunto de restricciones de movimiento (Figura 25):
    - Proximidad: se asume que la ubicación del objeto no debería cambiar notablemente de un *"frame"* a otro.
    - Máxima velocidad: se define un límite superior en la velocidad del objeto y limita las posibles correspondencias a un círculo alrededor del objeto.
    - Cambio sutil de velocidad: se asume que la dirección y velocidad del objeto no cambia drásticamente.

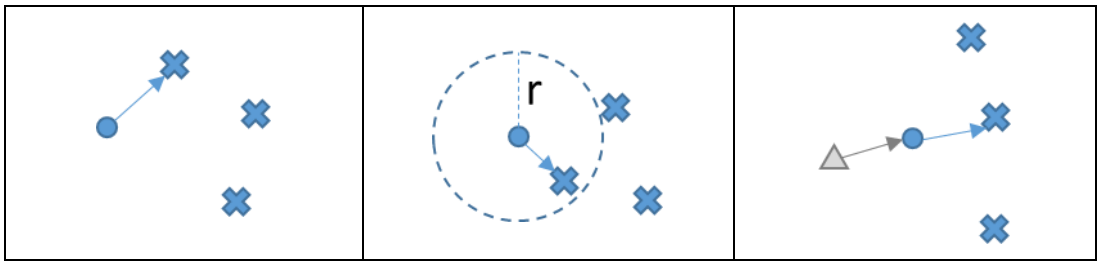


Figura 25: restricciones utilizadas por el método de correspondencia determinístico. (a) Proximidad, (b) Máxima velocidad y (c) Cambio sutil de velocidad.

- Método de correspondencia estadístico: Resuelven los problemas de seguimiento teniendo en cuenta la medición y las incertidumbres del modelo durante la estimación del estado del objeto. Estos métodos además, utilizan el enfoque del espacio de estados para modelar las propiedades del objeto tales como posición, velocidad y aceleración. Entre las implementaciones más utilizadas, podemos nombras las siguientes:
    - Filtro de Kalman (1960): Se utiliza para identificar el estado oculto (es decir, no medible) de un sistema dinámico lineal. El filtrado se compone de dos pasos: la predicción (estimación a priori) y la corrección (actualización de la medición).
    - Filtro de partículas, de Gordon, Salmond y Smith (1993): Para realizar el seguimiento, el filtro lanza de forma aleatoria un conjunto de puntos sobre la imagen en un instante dado donde luego se le asignara un valor o conjunto de valores a cada uno de ellos. A partir de estos valores, se creara un nuevo conjunto de puntos que sustituirá al anterior. Esta elección también será aleatoria, pero los valores que se han adjudicado a cada uno de los puntos permitirán que sea más probable de elegir a aquellos puntos que hayan capturado al objeto sobre el que se quiere realizar el seguimiento. Una vez que se crea el nuevo conjunto de puntos, se realizara una pequeña modificación al estado (es decir, la posición) de cada uno de ellos con el objetivo de estimar el estado del objeto (posición) en el instante siguiente.

Seguimiento del kernel

El kernel representara a la forma y apariencia del objeto. Por ejemplo, un kernel podría ser una plantilla rectangular o una forma elíptica con un histograma asociado. El seguimiento se logra por el cálculo del movimiento (transformación paramétrica, como la traducción, la rotación y *"affine"*) del kernel en *"frames"* consecutivos (Figura 26).

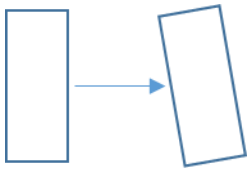


Figura 26: seguimiento del kernel.

El seguimiento del kernel se realiza típicamente calculando el movimiento del objeto, el cual es representado por una región de objeto primitiva, de un *"frame"* al siguiente. Estos algoritmos difieren en términos de la representación de la apariencia utilizada, el número de objetos rastreados y el método usado para estimar el movimiento del objeto. Entre los más utilizados, podemos nombrar los siguientes:

- Comparación de plantillas: Se trata de un método de fuerza bruta que busca en la imagen una región similar a la plantilla del objeto. Se suele utilizar la correlación cruzada para medir la similitud y características de color o intensidad para formar la plantilla.
- Mean-shift, de Fukunaga y Hostetler (1975): Es un método iterativo que también es utilizado en otras técnicas como *"clustering"*. Considera que el espacio de datos es una función de densidad de probabilidad muestreada, entonces para cada punto del conjunto de datos encuentra la moda más cercana. Para lograrlo, define una región alrededor de ese punto y calcula su media. Repitiendo el proceso hasta que converge.

Seguimiento de la silueta

El objetivo de esta técnica es encontrar en cada *"frame"* la región del objeto a través de un modelo de objeto generado usando los *"frames"* anteriores (Figura 27). El seguimiento aquí se realiza estimando la región del objeto en cada *"frame"*. Los métodos basados en esta técnica utilizan información codificada dentro de la región del objeto. Dicha información puede encontrarse en forma de modelos de densidad de aparición o en algún tipo de forma que se utilice para generar mapas de bordes.

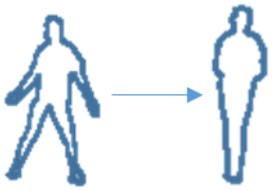


Figura 27: seguimiento de la silueta.

- Coincidencia de formas: Trata de buscar la silueta del objeto en el *"frame"* actual. Dicha búsqueda se realiza calculando la similitud del objeto con el modelo generado a partir de la hipotética silueta del objeto obtenida en el *"frame"* anterior.
- Evolución del contorno: Lo que se hace aquí es evolucionar iterativamente un contorno inicial del *"frame"* anterior hasta su nueva posición en el *"frame"* actual. Esta evolución asume que parte del objeto en el *"frame"* actual solape con la región del objeto en el *"frame"* anterior. Para cumplir con el objetivo, este método de seguimiento se puede llevar a cabo a partir de dos aproximaciones:

- La utilización de modelos de espacio de estado, para modelar la forma del contorno y el movimiento.
- La minimización directa de alguna función de energía, mediante técnicas como el descenso del gradiente.

3. HIPOTESIS

Con el presente trabajo se pretende demostrar que, a partir de la combinación de diferentes metodologías y algoritmos sobre la detección de objetos y personas en secuencias de video, es posible lograr el seguimiento de peatones en situaciones donde la posición de la cámara se encuentra por encima de los mismos, impidiendo la utilización directa de técnicas conocidas. Es decir, si utilizáramos los métodos conocidos en la actualidad para la detección de peatones en estas condiciones, el resultado no sería del todo satisfactorio.

Para ello, se experimentará con la combinación de varias metodologías y algoritmos hasta llegar a un prototipo que sea capaz no solo de detectar las personas en una secuencia de video donde la cámara se encuentra por encima de ellas, sino que además pueda lograr el seguimiento de las mismas.

4. METODOLOGIA

4.1. Procesamiento de imágenes

En esta sección describiremos los distintos métodos y algoritmos que fueron tenidos en cuenta para la realización del presente trabajo.

4.1.1. Primitivas de una imagen

Antes de comenzar, sería conveniente a modo introductorio desarrollar el concepto de algunas primitivas importantes dentro de una imagen e indicar el rol que cumplen en el procesamiento de las mismas.

Pixeles

Un pixel es la menor unidad homogénea en color que forma parte de una imagen digital. Las imágenes se forman como una sucesión de píxeles y dicha sucesión marca la coherencia de la información presentada, siendo su conjunto una matriz coherente de información para el uso digital (Figura 28).

En las imágenes de mapa de bits, o en los dispositivos gráficos, cada píxel se codifica mediante un conjunto de bits de longitud determinada, por ejemplo, puede codificarse un píxel con un byte (8 bits), de manera que cada píxel admite hasta 256 variaciones de color (2^8 posibilidades binarias), de 0 a 255.

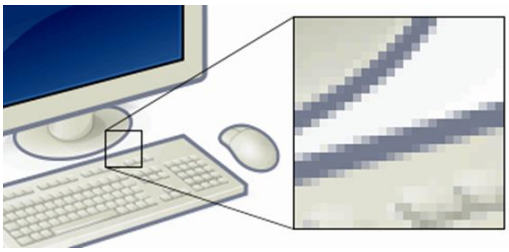


Figura 28. Ampliación de una imagen donde podemos ver los pixeles de la misma. Fuente: https://en.wikipedia.org/wiki/Computer_graphics

Modo de color

Los sistemas de procesamiento de imágenes utilizan varios modos para definir todos los posibles colores. Entre los más populares podemos nombrar: HSB (tono, saturación y brillo), RGB (rojo, verde y azul), CMYK (cian, magenta, amarillo y negro) y escala de grises.

Escala de grises

En este modo, la imagen se representa en distintos niveles de gris definidos a partir de su brillo, esto es, 0 (negro) al 255 (blanco), ver figura 29. Este modo es útil cuando solo nos interesa la intensidad de los valores, ya que al convertir una imagen a color hacia tonos de grises, las demás variables, como saturación o tono, serán descartadas.

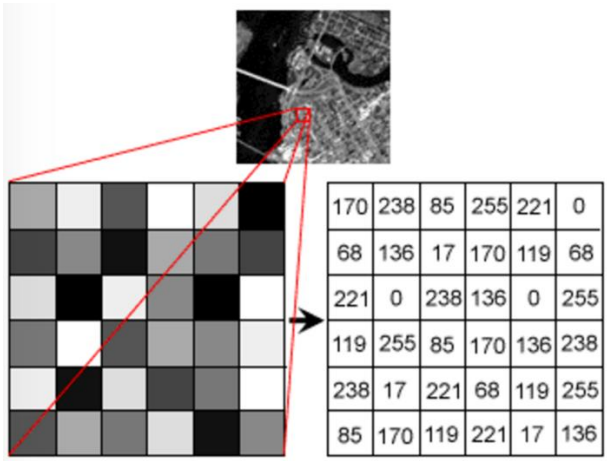


Figura 29: Ejemplo de los valores que puede tomar cada pixel en la escala de grises. Fuente: http://hosting.soonet.ca/eliris/remotesensing/bl130lec10.html

Segmentos

La segmentación de imágenes es el proceso de división de una imagen digital en múltiples segmentos (conjuntos de pixeles). El objetivo de esta segmentación es la de simplificar y/o cambiar la representación de una imagen en algo más significativo y fácil de analizar.

Se utiliza normalmente para localizar objetos y límites (líneas, curvas, etc.) dentro de una imagen. El resultado de la segmentación será entonces un conjunto de segmentos que colectivamente cubren toda la imagen, o un conjunto de contornos extraídos de la misma (Figura 30).

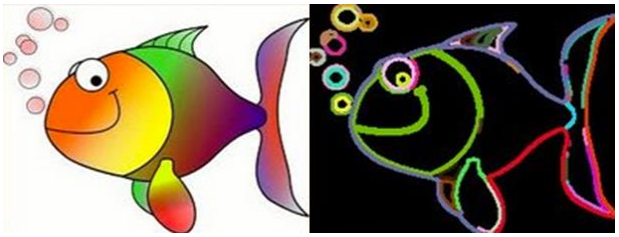


Figura 30: Ejemplo de los distintos segmentos calculados en una imagen. Fuente: http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find_contours/find_contours.html

Regiones

Continuando con la segmentación de imágenes, podemos definir una región como un conjunto de pixeles en donde todos comparten ciertas características o propiedades calculadas, como color, intensidad, textura o movimiento, y donde las regiones adyacentes son significativamente diferentes con respecto a las mismas características.

Esquinas, puntos únicos o *"features"*

Una buena analogía para explicar este concepto son los rompecabezas, donde tenemos un gran número de piezas con imágenes poco claras y necesitamos ensamblarlas correctamente para formar una gran imagen real.

La pregunta que podemos plantearnos es: ¿Qué razonamiento llevamos a cabo para resolverlo? En la mayoría de los casos buscamos en cada pieza, patrones o características específicas que sean únicas, fácilmente rastreables y comparables. Si intentamos buscar una definición formal para cada característica (llamémoslas "puntos únicos" o *"features"*), es posible que nos resulte difícil describirla en palabras, pero si nos piden que señalemos un punto único que pueda ser utilizado a través de varias imágenes (piezas), podríamos hacerlo. Es por eso que aun niños muy pequeños pueden armar rompecabezas.

Buscamos estos puntos únicos en una imagen, luego las buscamos en otras imágenes y unimos las piezas. Eso es todo. Estas habilidades están presentes en nosotros en forma inherente.

Preguntémonos ahora: ¿Qué son estas características, puntos únicos o *"features"*? E intentemos que la respuesta sea también comprensible para una computadora.

Como dijimos antes, resulta difícil explicar cómo los humanos detectan estos puntos únicos, es algo que está programado en nuestro cerebro. Pero si miramos en detalle en algunas piezas y buscamos diferentes patrones, podríamos encontrar algo interesante.

Observemos la siguiente imagen:

Figura 31: ejercicio para la detección de puntos únicos. Fuente: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html

El ejercicio es encontrar la posición exacta de las 6 piezas provistas.

- A y B son superficies planas, es difícil encontrar la ubicación exacta de estas piezas.
- C y D son mucho más simples, son bordes del edificio. Podríamos encontrar una ubicación aproximada pero la exacta seguiría siendo dificultosa, ya que, a lo largo del borde, es igual en todas partes. El borde es un punto único mucho mejor que la superficie plana, pero no lo suficiente.
- E y F son esquinas del edificio y pueden ser fácilmente ubicados, ya que generalmente las esquinas, sin importar la cantidad de estas que la imagen contenga, siempre se van a ver diferentes.

Para una mejor comprensión analicemos la siguiente imagen:

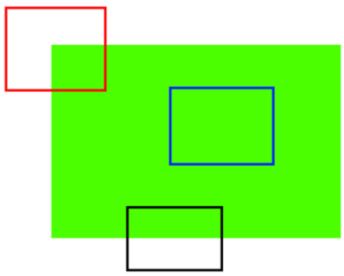


Figura 32: simplificación del ejercicio para la detección de puntos únicos.

- Así como vimos en el ejemplo del edificio, el cuadro azul es una superficie plana difícil de rastrear. No importa donde lo movamos, siempre se va a ver igual.
- El cuadro negro, es un borde. Si lo movemos en una dirección vertical, veremos un cambio notorio, pero si lo movemos horizontalmente, siempre se va a ver igual.
- El cuadro rojo es una esquina. No importa donde lo movamos, siempre se verá diferente. Por lo que podemos concluir que es único y considerarlo como un buen punto único en una imagen.

Ahora que logramos entender que son los puntos únicos, la pregunta que surge es ¿cómo encontrarlos? o ¿cómo detectar estas esquinas?

Podríamos responder de una forma intuitiva, buscando las regiones que tengan una variación máxima cuando nos movemos (levemente) en todas las regiones alrededor de ella. Esta técnica es también aplicada en distintos algoritmos de computación.

Entonces, una vez que encontramos los puntos únicos, querríamos encontrar los mismos en otras imágenes. Para esto tomaríamos una región alrededor de cada una y trataríamos de explicar con nuestras palabras que vemos: “la parte de arriba es el cielo”, “la

parte de abajo corresponde a los espejos del edificio", etc., y luego buscaríamos esas mismas áreas en otras imágenes.

Básicamente, lo que estamos haciendo es describir un punto único. De una forma similar, una computadora también podría describir la región alrededor de éste para que luego pueda ser localizado en otras imágenes. Una vez que se obtienen los puntos únicos y su descripción, podemos buscar los mismos en otras imágenes y unirlas para resolver nuestro rompecabezas, o para hacer lo que queramos con ellas.

Describiremos a continuación diferentes algoritmos que nos permitirán llevar a cabo estas tareas y que serán el primer paso para alcanzar nuestro objetivo final, el seguimiento de objetos en secuencias de video.

4.1.2. Detección de esquinas

La detección esquinas, *"corners"* o "puntos de interés" es un acercamiento usado en los sistemas de visión por computadora para extraer ciertos tipos de rasgos e inferir el contenido de una imagen (Figura 33). Es útil además para el seguimiento de objetos ya que, con la configuración que das las esquinas podemos identificar un mismo objeto en las distintas secuencias de un video. Se utiliza frecuentemente en la detección de movimiento, análisis de imagen, rastreo en video, modelado 3D y reconocimiento de objetos, entre otros.

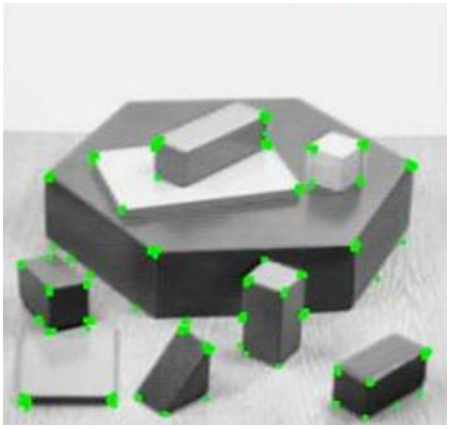


Figura 33: resultado de un típico algoritmo de detección de esquinas. Fuente: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html

Detector de esquinas de Harris

Como vimos anteriormente, una esquina o *"corner"* se caracteriza por ser una región de la imagen con cambios de intensidad en diferentes direcciones. Uno de los primeros intentos para detectar estas esquinas fue realizado por Chris Harris y Mike Stephens en el artículo *"A Combined Corner and Edge Detector"* en 1988, más conocido como *"Harris Corner Detector"*, donde se toma esta simple idea y se representa en una forma matemática, básicamente buscando la diferencia en intensidad para un desplazamiento de **(u, v)** en todas las direcciones:

$$E(u,v) = \sum_{x,y} w(x,y)[I(x+u, y+v) - I(x,y)]^2 \quad (1)$$

Donde:

- **w(x, y)** es la ventana de desplazamiento en la posición **(x, y)**
- **I(x, y)** es la intensidad en la posición **(x, y)**
- **I(x + u, y + v)** es la intensidad en la ventana desplazada**(x + u, y + v)**

Dado que estamos buscando ventanas con esquinas, buscaremos ventanas con una gran variación de intensidad. Por lo tanto, tenemos que maximizar la función **E(u, v)**. Es decir, maximizar el segundo término:

$$\sum_{x,y}[I(x+u, y\ +\ v) - I(x,y)]^2 \tag{2}$$

Usando el teorema de expansión de Taylor (donde $\mathbf{I_x}$ e $\mathbf{I_y}$ son las imágenes derivadas en las direcciones **x** e **y** respectivamente) obtendremos lo siguiente:

$$E(u,v) \approx \sum_{x,y}[I(x,y)\ +\ uI_x + vI_y - I(x,y)]^2 \tag{3}$$

Expandiendo la ecuación y cancelando adecuadamente:

$$E(u,v) \approx \sum_{x,y} u^2 I_x^2 + 2uvI_xI_y +\ v^2 I_y^2 \tag{4}$$

La cual puede ser expresada en una matriz de la siguiente manera:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{5}$$

Si llamamos M a la expresión encerrada entre paréntesis:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_xI_y \\ I_xI_y & I_y^2 \end{bmatrix} \tag{6}$$

Nuestra ecuación nos quedara de la siguiente forma:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} M \begin{bmatrix} u \\ v \end{bmatrix} \tag{7}$$

Luego de esto, se definió un sistema de puntación, básicamente una ecuación, la cual va a determinar si una ventana puedo contener una esquina o no.

$$R = \det(M) - k(\text{traza}(M))^2 \tag{8}$$

Donde:

- **det(M) = $\lambda_1\lambda_2$** *(con λ_1 y λ_2 como los autovalores de* ***M)***
- **traza(M) = $\lambda_1 + \lambda_2$**

Son los autovalores entonces, los que deciden si una región es una esquina, un borde o una superficie plana.

- Si **|R|** es un valor bajo, es decir λ_1 y λ_2 son valores bajos, la región es plana.
- Si **R < 0**, es decir $\lambda_1 >> \lambda_2$ o viceversa, la región es un borde.

- Si **R** es un valor alto, es decir λ_1 y λ_2 son valores altos y $\lambda_1 \sim \lambda_2$, la región es una esquina.

Para una mejor comprensión, podemos representarlo en el siguiente diagrama:

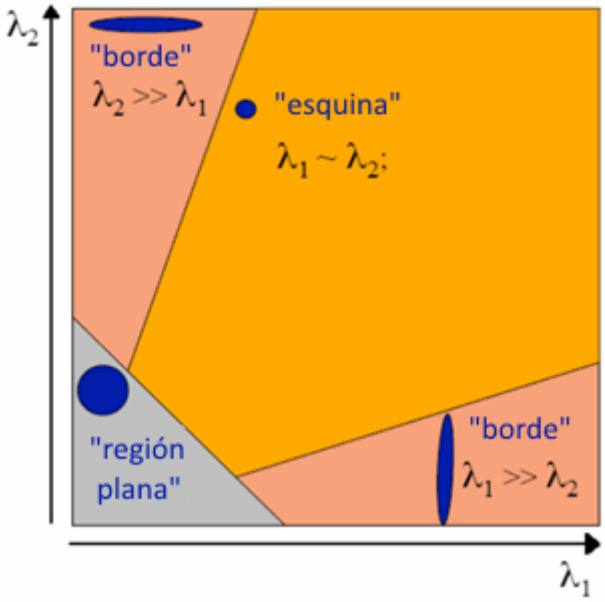


Figura 34: clasificación de las regiones según Harris.

Shi-Tomasi Corner Detector – Good Features to Track

En 1994, Jianbo Shi y Carlo Tomasi realizaron una pequeña modificación en el método de Harris, y lo publicaron en el artículo *"Good Features to Track"* obteniendo mejores resultados.

Como vimos anteriormente, la función de puntuación en el método de Harris estaba dada por esta ecuación:

$$R = \det(M) - k(\text{traza}(M))^2 \tag{9}$$

Shi y Tomasi propusieron lo siguiente:

$$R = \min(\lambda_1, \lambda_2) \tag{10}$$

Si este valor es mayor a un umbral predefinido, entonces podemos afirmar que hemos detectado una esquina.

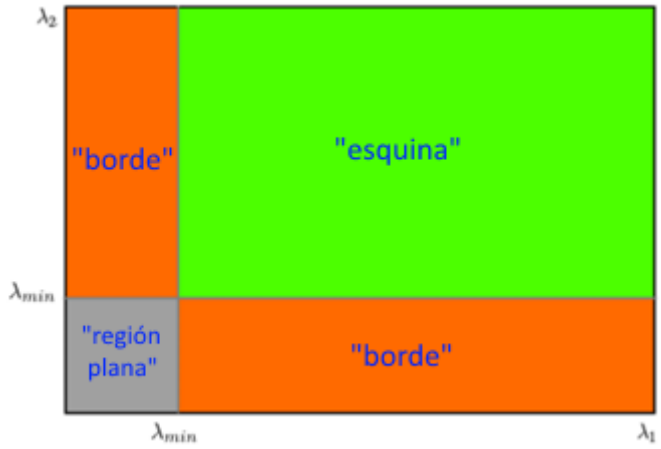


Figura 35: clasificación de las regiones según Shi-Tomasi.

Como podemos ver en la figura 35, cuando λ_1 y λ_2 se encuentran por encima de un valor establecido (λ_{min}) se considera que se ha detectado una esquina (región verde).

4.2. Dinámica en la escena

4.2.1. Flujo Óptico

El flujo óptico (u *"optical flow"*) es el patrón del movimiento aparente de objetos entre dos *"frames"* de video consecutivos causado por el movimiento del objeto o del observador (la cámara).

Se representa en un campo vectorial de dos dimensiones donde cada vector es un vector de desplazamiento que muestra el movimiento de puntos de un *"frame"* al siguiente (Figura 36). En la siguiente imagen, observamos una esfera moviéndose en 5 *"frames"* consecutivos y la flecha que nos indica el vector de desplazamiento:

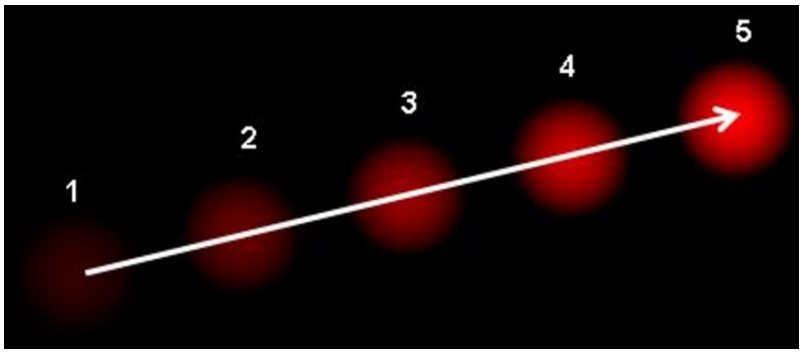


Figura 36: el vector de flujo óptico de un objeto en movimiento en una secuencia de vídeo. Fuente: http://docs.opencv.org/trunk/d7/d8b/tutorial_py_lucas_kanade.html

Flujo óptico brinda muchas aplicaciones en áreas como:

- Estructura del movimiento
- Compresión de video
- Estabilización de video

Y se basa en dos supuestos principales:

1) La intensidad del pixel de un objeto no cambia entre dos *"frames"* consecutivos.
2) Los pixeles "vecinos" poseen un movimiento similar.

Consideremos un pixel con una intensidad **I(x, y, t)** en un primer *"frame"* que se desplazó una cierta distancia **(dx, dy)** en el siguiente *"frame"* luego de un cierto intervalo de tiempo **dt**. Como vimos en el primer supuesto, la intensidad se mantiene entre ambos *"frames"*, entonces podemos decir:

$$I(x, y, t) = I(x + dx, y + dy, t + dt) \quad (11)$$

Aproximando por series de Taylor en el segundo miembro, removiendo los términos en común y dividiendo por **dt**, obtenemos lo siguiente:

$$f_x u + f_y v + f_t = 0 \tag{12}$$

Donde:

$$f_x = \frac{df}{dx} \qquad f_y = \frac{df}{dy}$$
$$u = \frac{dx}{dt} \qquad v = \frac{dy}{dt} \tag{13}$$

Esta ecuación es llamada ecuación del flujo óptico. En ella, podemos encontrar $\mathbf{f_x}$ y $\mathbf{f_y}$ que son los gradientes de la imagen y $\mathbf{f_t}$ que es el gradiente a lo largo del tiempo. Pero **(u,v)** es desconocido.

Es imposible resolver esta ecuación con 2 variables desconocidas, por lo que existen diferentes métodos para mitigar este problema. Uno de los más conocidos y eficaces es el método de Lukas-Kanade.

Método de Lukas-Kanade

Como vimos en el segundo supuesto del flujo óptico, todos los pixeles "vecinos" tienen un movimiento similar. El método de Lukas-Kanade toma un área de 3x3 alrededor del punto. Entonces podemos decir que tenemos 9 puntos con el mismo movimiento.

Para estos 9 puntos podemos encontrar sus respectivos $\mathbf{(f_x, f_y, f_t)}$. Entonces nuestro problema ahora se trata de resolver 9 ecuaciones con 2 variables desconocidas.

Una buena solución es utilizar el método de ajuste de mínimos cuadrados. A continuación se muestra la solución final:

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \sum_i f_{x_i}^2 & \sum_i f_{x_i} f_{y_i} \\ \sum_i f_{x_i} f_{y_i} & \sum_i f_{y_i}^2 \end{bmatrix}^{-1} \begin{bmatrix} -\sum_i f_{x_i} f_{t_i} \\ -\sum_i f_{y_i} f_{t_i} \end{bmatrix} \tag{14}$$

Se puede observar una similitud de la matriz inversa con el método de detección de esquinas de Harris, lo cual denota que las esquinas son buenos puntos para ser tenidos en cuenta en estos casos.

Desde el punto de vista del usuario la idea es simple: damos los puntos para rastrear y recibimos los vectores del flujo óptico para dichos puntos.

Pero nuevamente existen algunos problemas. Hasta ahora, estábamos tratando con pequeños movimientos por lo que este método fallara en movimientos grandes. Es aquí

donde aparece la idea de las pirámides: a medida que subimos en la pirámide, los pequeños movimientos son removidos y los movimientos largos se convierten en pequeños.

4.2.2. Sustracción de Fondo

La sustracción de fondo (más conocida como *"Background Subtraction"*) es una técnica ampliamente utilizada para detectar objetos en movimiento a partir de cámaras estáticas.

Por ejemplo, consideremos el caso de un contador de visitas, donde una cámara estática lleva la cuenta del número de personas entrando y saliendo de una habitación o una cámara de tráfico extrayendo información de los vehículos. En todos estos casos, primero es necesario extraer a la persona o vehículo. Técnicamente, necesitamos extraer el plano en movimiento del fondo estático.

Si tenemos una imagen del fondo sin objetos en movimiento, como una imagen de la habitación sin personas o una imagen de la ruta sin autos, es una tarea sencilla: hacemos la diferencia entre la imagen del fondo con las nuevas imágenes y obtendremos los objetos en movimiento. Pero en la mayoría de los casos, no es posible obtener tal imagen.

La detección de objetos se puede conseguir mediante la construcción de una representación de la escena llamada modelo de fondo y después encontrando las desviaciones del modelo para cada *"frame"* entrante. Cualquier cambio significativo en una región de la imagen del modelo de fondo representa un objeto en movimiento (Figura 37). Los píxeles que constituyen las regiones en proceso de cambio se marcan para su posterior procesamiento.

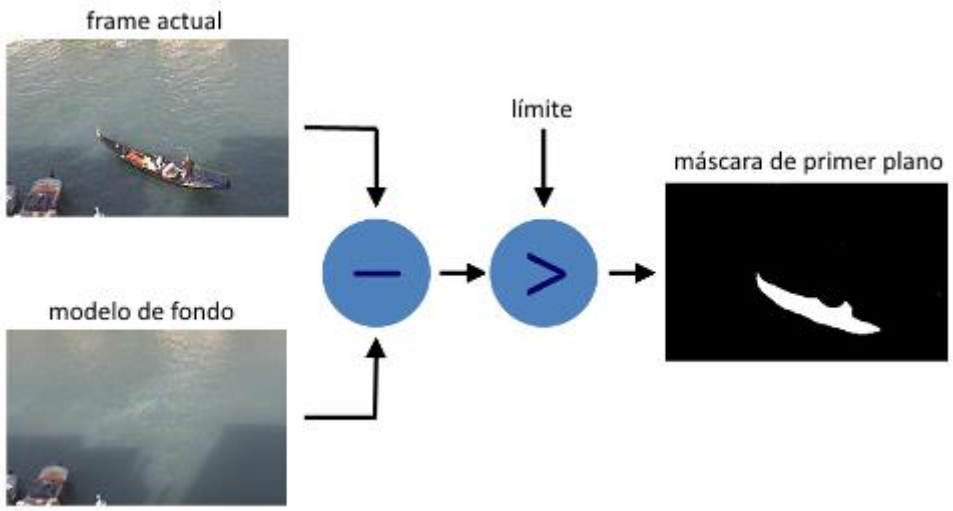


Figura 37: ejemplo de sustracción entre el *"frame"* actual y el modelo de fondo. Fuente: http://docs.opencv.org/trunk/d1/dc5/tutorial_background_subtraction.html

Muchos algoritmos y mejoras fueron propuestas para este método, a continuación describiremos los más conocidos.

Mixturas Gaussianas (MoG)

Introducido en la publicación titulada *"An improved adaptive background mixture model for real-time tracking with shadow detection"* de KaewTraKulPong y Bowden en el año 2001.

A groso modo, la idea básica de este algoritmo es modelar cada pixel de la imagen, mediante una serie de K distribuciones gaussianas (K = 3 a 5) donde la media representa el valor con más probabilidades de ser observado y la desviación estándar qué tanto puede variar ese valor. Cada componente representa el posible valor del pixel en un momento dado, es decir, el fondo ya no sólo puede tener un sólo valor, sino "n" posibles diferentes valores dependiendo de nuestro número de componentes.

Mixturas Gaussianas (MoG) 2

Es otro algoritmo que utiliza conjunto de distribuciones gaussianas basado en dos artículos publicados por Zivkovic, *"Improved adaptive Gausian mixture model for background subtraction"* en 2004 y *"Efficient Adaptive Density Estimation per Image Pixel for the Task of Background Subtraction"* en 2006.

La principal característica de este algoritmo es que selecciona el numero apropiado de distribuciones gaussianas para cada pixel (recordemos que en el caso anterior, tomábamos K distribuciones en todo el algoritmo). Esto proporciona una mejor adaptabilidad a variaciones en las escenas a causa de cambios de iluminación.

GMG

Introducido por Andrew B. Godbehere, Akihiro Matsukawa y Ken Goldberg en el artículo *"Visual Tracking of Human Visitors under Variable-Lighting Conditions for a Responsive Audio Art Installation"* en 2012, su nombre se debe a las iniciales de los autores.

El algoritmo utiliza los primeros N *"frames"* para el modelado del fondo y luego utiliza reglas de inferencia bayesiana para calcular la probabilidad de cada píxel de ser clasificado como fondo o primer plano.

Las estimaciones son adaptativas, es decir que nuevas observaciones están más fuertemente ponderadas que las observaciones antiguas para adaptarse a las variaciones en la iluminación.

4.3. Herramientas adicionales

4.3.1. Transformaciones Morfológicas

Las Transformaciones Morfológicas son operaciones que podemos hacer sobre una máscara para transformarla. Necesitaremos de dos cosas, la máscara original y un kernel.

La máscara es una imagen binaria, es decir, una matriz de 0 y 1. El kernel (o matriz de convolución) se trata de una matriz más pequeña con valores predefinidos. Según cuáles sean estos valores, vamos a realizar una transformación u otra.

Para realizar una transformación morfológica, sobreponemos el kernel encima de cada uno de los píxeles de la máscara, y entonces hacemos una operación entre los píxeles de la imagen y los valores del kernel (multiplicarlos, sumarlos, etc.).

Este proceso se llama convolución de matrices (Figura 38).

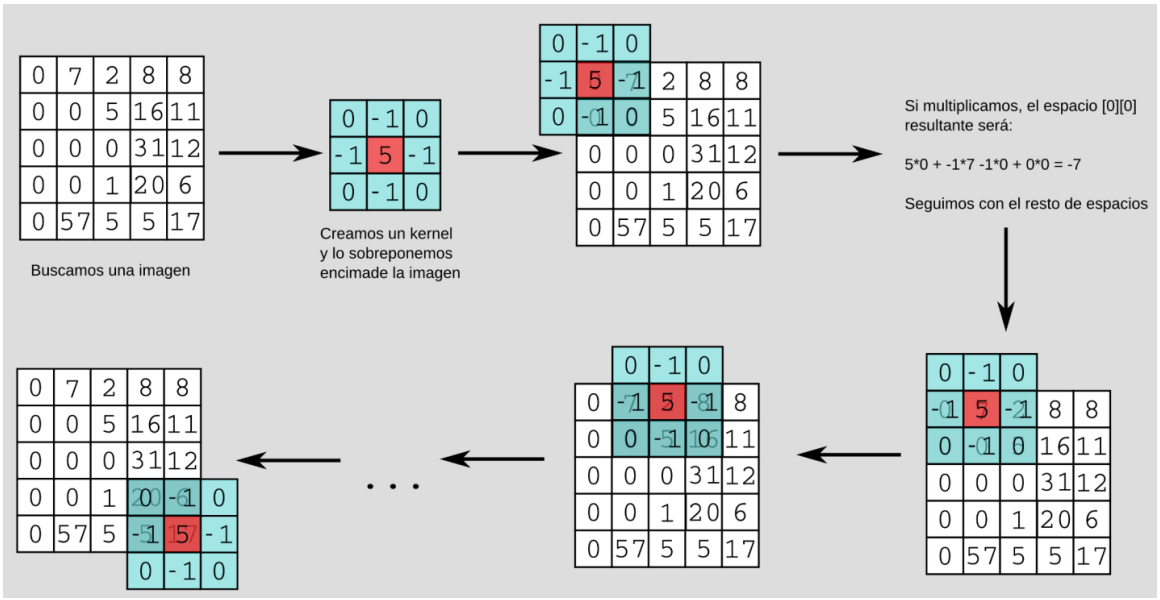


Figura 38: ejemplo de convolución de matrices. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Las trasformaciones más básicas son Erosión y Dilatación y se usan principalmente para la eliminación de ruido, el aislamiento de elementos individuales, conectar elementos dispares y la detección de protuberancias o agujeros de intensidad.

Erosión

Para entender este operador, podemos imaginarnos el efecto del agua sobre una roca que va desgastándola por el exterior. Este operador reducirá el tamaño de los objetos blancos de la máscara reduciendo su límite. Al momento de hacer la convolución, el pixel será “1” solo si todos los pixeles que quedan debajo del kernel también son “1” (Figura 39).

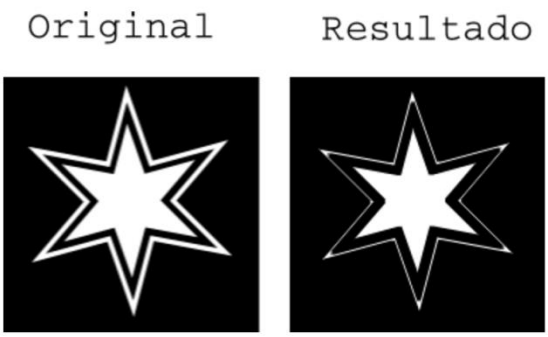


Figura 39: ejemplo de erosión. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Dilatación

Esta operación aumenta la zona blanca de la máscara. Al hacer la convolución, el píxel resultante será "1" si al menos un píxel que quede debajo del kernel también es "1" (Figura 40).

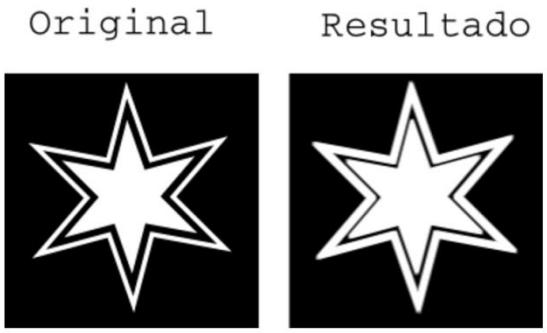


Figura 40: ejemplo de dilatación. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Gradiente Morfológico

No es más que la diferencia entre la dilatación y la erosión. Es muy útil para encontrar la silueta de los objetos (Figura 41).

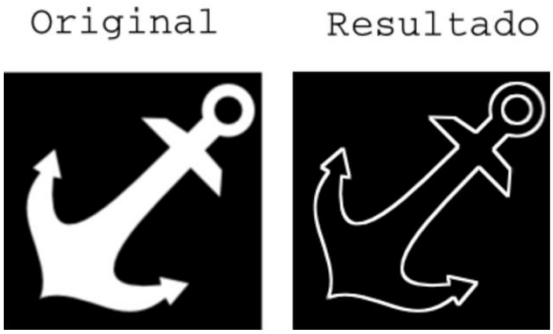


Figura 41: ejemplo de gradiente morfológico. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Apertura

También conocida como *"opening"*, se trata de una erosión seguida de una dilatación. Nos sirve para eliminar el ruido blanco sobre las zonas negras (Figura 42).

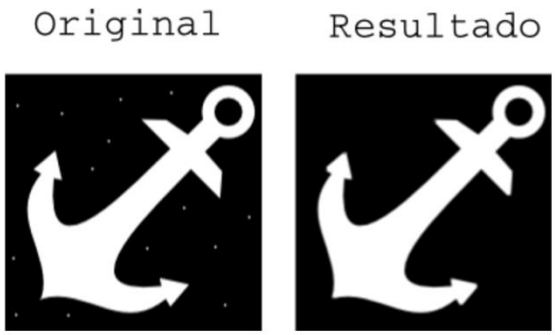


Figura 42: ejemplo de apertura. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Cierre

También conocida como *"closing"*, es lo contrario a la apertura, se trata de una dilatación seguida de una erosión. Si hay ruido negro en las áreas blancas, esta transformación se encargara de eliminarlo (Figura 43).

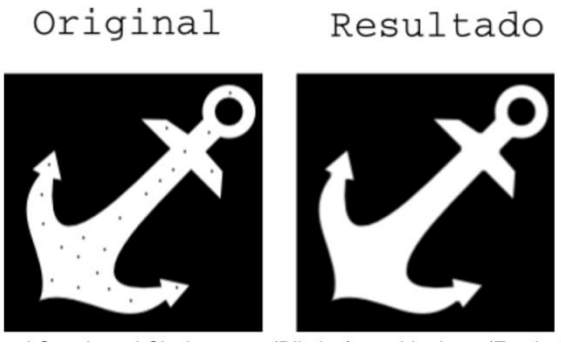


Figura 43: ejemplo de cierre. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Top Hat

Es la diferencia entre la imagen original y su apertura. Se utiliza para descubrir aquellas estructuras de la imagen que han sido eliminadas en el filtrado de apertura.

Una operación entre la imagen original y el filtrado (apertura en este caso) aumenta considerablemente el contraste de las zonas eliminadas (Figura 44).

Figura 44: ejemplo de top hat. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

Black Hat

Similar al *"Top Hat"*, es la diferencia entre el cierre de la imagen y la imagen original. Se utiliza para descubrir aquellas estructuras de la imagen que han sido eliminadas en el filtrado de cierre.

Una operación entre la imagen original y el filtrado (cierre en este caso) aumenta considerablemente el contraste de las zonas eliminadas (Figura 45).

Figura 45: ejemplo de black hat. Fuente: https://robologs.net/2015/07/26/como-filtrar-el-ruido-de-una-mascara-con-opencv/

4.3.2. Detección de bordes o contornos

La detección de bordes o contornos (*"contours"*) es una herramienta fundamental en el procesamiento de imágenes y en visión por computadora. Se basa en detección de los cambios bruscos de brillo de la imagen y tiene como objetivo capturar eventos importantes y cambios en las propiedades del mundo.

Se puede demostrar que, bajo supuestos bastantes generales para un modelo de formación de la imagen, las discontinuidades en el brillo de la imagen se corresponden con mayor probabilidad a:

- discontinuidades en la profundidad,
- discontinuidades en la orientación de las superficies,
- cambios en las propiedades del material
- variaciones en la iluminación de la escena

En el caso ideal, los resultados de aplicar un detector de bordes a una imagen pueden conducir a un conjunto de curvas conectadas que indican las fronteras de los objetos, las fronteras de las marcas en las superficies de estos objetos y curvas que corresponden a discontinuidades en la orientación de las superficies.

Al aplicar un algoritmo de detección de bordes a una imagen es posible reducir la cantidad de datos a ser procesados, por lo que permite filtrar la información que puede ser considerada como menos relevante, logrando preservar las propiedades estructurales de la imagen. Si el paso de detección de bordes fue satisfactorio, el paso de interpretar el contenido en la imagen original se puede reducir sustancialmente.

Sin embargo, no siempre se obtienen esos bordes ideales a partir de imágenes reales, con una complejidad moderada.

Los bordes extraídos de imágenes no triviales, se ven afectados por la fragmentación en las mismas, lo que significa que las curvas que representan los bordes no están conectadas o se crean bordes falsos que no corresponden a características importantes de la imagen - lo que complica la tarea de interpretar los datos en la imagen.

Los bordes extraídos de imágenes 2D a partir de escenas 3D se pueden clasificar según el punto de vista en dependientes o independientes:

- Un borde independiente del punto de vista refleja propiedades inherentes de objetos tridimensionales, tales como marcas y formas en las superficies de los mismos.
- Un borde dependiente del punto de vista puede cambiar si se cambia el punto de vista, y refleja la geometría de la escena, tales como objetos que se tapan unos a otros.

Un borde típico puede ser la arista entre un bloque rojo y uno amarillo que se encuentran uno a continuación del otro. En contraste, una recta puede ser un pequeño

número de pixeles de un color diferente a un fondo que nunca cambia. Para la recta, usualmente se detectan dos bordes, uno a cada lado de la misma. Para un ejemplo más claro, observemos la figura 46:

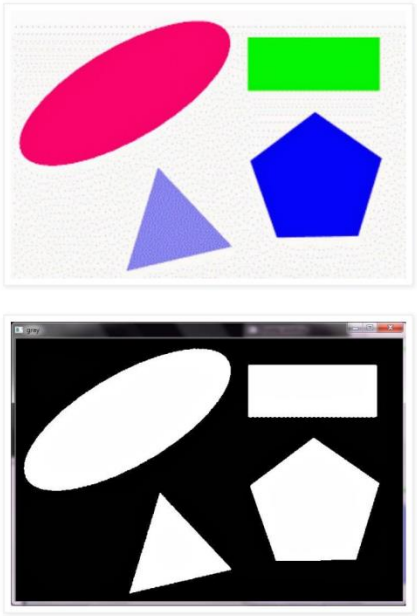


Figura 46: ejemplo de detección de bordes o contornos. Fuente: http://opencvexamples.blogspot.com/2013/09/find-contour.html

De los múltiples algoritmos de extracción de bordes nos centraremos en el que Satoshi Suzuki y Keiichi Abe publicaron en la publicación *"Topological structural analysis of digitized binary images by border following"* en el año 1985, ya que este es el utilizado por la librería OpenCV a través del método *"cv2.findContours"*.

Para comenzar, es necesario partir de una imagen binaria. Es decir, de una imagen que tan solo consta de pixeles con valor cero (0) o uno (1). Sobre ella se aplica el algoritmo que la barre, empezando por su esquina superior izquierda, a la busca de un primer pixel a uno (1), y que cuando lo encuentra es capaz de seguir la cadena de pixeles con valor uno que se encuentran unidos a él hasta volver al pixel de partida. Esa cadena de pixeles a uno (1) encontrados se denomina borde o contorno. Además, el algoritmo es capaz de encontrar todos los contornos presentes en la imagen, ya que cuando termina con uno empieza de nuevo el proceso hasta asegurarse de haber barrido la imagen por completo.

El algoritmo es bastante notable, ya que no sólo es capaz de encontrar los contornos exteriores, sino también los interiores a otros, y retornarlos clasificados jerárquicamente.

En la práctica existe toda una familia de algoritmos de extracción de contornos que recorren pixeles vecinos rotando a izquierda o derecha dependiendo del valor del pixel en curso. Lo que diferencia a éste es la forma en que etiqueta los pixeles visitados para

catalogarlos y clasificarlos. El proceso completo podría ser tedioso de explicar, pero sus pautas generales no son demasiado complejas.

Básicamente hay un contador de contornos encontrados y un buffer de pixeles recorridos. El contador se inicializa a cero (0) y el buffer con una copia de la imagen original. Se barre el buffer de arriba abajo y de izquierda a derecha. Una transición de un pixel 0 a otro 1 indica que se ha detectado un borde exterior, momento en que se suma uno al contador de contornos encontrados y se buscan todos los pixeles 1 vecinos del encontrado. Si el pixel no tiene vecinos a 1 se cambia el valor del pixel por el del contador con signo contrario y se empieza otra vez con el siguiente pixel.

En caso contrario se cambia el valor del pixel por el del contador, excepto que su valor sea mayor que 1, lo que significa que ya ha sido visitado, y se continúa buscando vecinos a 1 hasta retornar al pixel inicial. Una transición de un pixel con valor igual o mayor que 1 a otro 0 indica que se ha detectado un borde interior, momento en que se repite el mismo proceso usado para los contornos exteriores.

El algoritmo presenta una pequeña dificultad en los bordes de la imagen, y para solventarlo presupone que la imagen está rodeada por los cuatro costados de pixeles con valor 0. Es decir, que el buffer que utiliza tiene una fila más por encima y por debajo, y una columna más a derecha e izquierda, que la imagen original.

4.3.3. Transformada de la distancia

La transformada de la distancia es un operador aplicado también a imágenes binarias que proporciona una métrica o medida de la separación de los puntos en la imagen. Se calcula la distancia entre cada pixel con un "0" y el pixel con "1" más cercano.

Dado un conjunto "I", un subconjunto "G" y una función de distancia "d", la transformada de la distancia "DT" de "I" respecto a "G", asocia a cada punto "p" de "I" el valor:

$$\mathrm{DT}(p) = \text{minimo}\{d(p,q)\}\ (para\ cada\ "q"\ de\ "G") \tag{15}$$

Si el conjunto "I" es una imagen binaria y el subconjunto "G" es el conjunto de píxeles blancos de "I", la transformada de la distancia de "I" asocia, a cada píxel "p" de la imagen, la mínima distancia entre "p" y cualquier píxel blanco.

La transformada de la distancia depende enteramente de la distancia usada para calcularla.

La transformada de la distancia de una imagen "I" es una matriz del mismo tamaño que la imagen original, que almacena los valores de la transformada de la distancia de cada punto "p" en "I".

A continuación (Figura 47) un ejemplo de transformada de la distancia al fondo de la imagen usando la distancia dada por la 8-adyacencia:

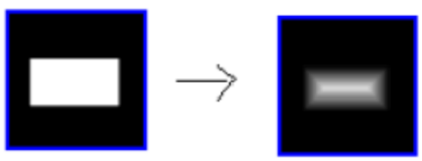
0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

→

0	0	0	0	0	0	0	0
0	1	1	1	1	1	1	0
0	1	2	2	2	2	1	0
0	1	2	3	3	2	1	0
0	1	2	2	2	2	1	0
0	1	1	1	1	1	1	0
0	0	0	0	0	0	0	0

Figura 47: ejemplo de transformada de la distancia.

La transformada de la distancia se puede representar como una imagen en escala de grises, donde el nivel de gris representa el valor de la transformada de la distancia de la imagen en el píxel correspondiente (Figura 48).



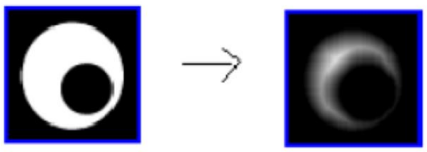


Figura 48: ejemplos de representaciones de transformada de la distancia como una imagen en escala de grises.

4.4. Resumen

A modo de resumen, finalizaremos este capítulo enumerando en la siguiente tabla (TABLA I) las ventajas y desventajas de los métodos y algoritmos detallados anteriormente.

TABLA I: ventajas e inconvenientes de los métodos estudiados.

Método/Algoritmo	Ventajas	Desventajas
Detección de Esquinas	• Útil para detectar puntos de la imagen donde se dan cambios significativos en todas las direcciones.	• Generalmente estos algoritmos son muy sensibles a rotaciones, cambios de escala o cambios de iluminación en la imagen, arrojando un gran número de falsos positivos.
Flujo Óptico	• Nos permite detectar si una esquina o punto de interés se desplazó entre un *"frame"* y otro muy cercano.	
Sustracción de Fondo	• Nos permite detectar objetos en movimiento a partir de cámaras estáticas. • Podemos distinguir las zonas de fondo o estáticas de las zonas en movimiento que corresponden al primer plano.	• Dado que el fondo no es invariante en el tiempo, es necesario realizar un modelado dinámico del fondo (técnica conocida como "entrenamiento del algoritmo de sustracción de fondo") para adaptarlo a las variaciones que se puedan producir en el mismo. • La aplicación de estos algoritmos se vuelve mucho más compleja si queremos usarlos a partir de imágenes capturadas por una cámara en movimiento. • Estos algoritmos también son sensibles a los cambios en la iluminación, la existencia de varios objetos en movimiento, la superposición de estos y los cambios en las

		estructuras estáticas
Transformaciones Morfológicas	• Combinando distintas transformaciones, podemos eliminar el ruido o falsos positivos arrojados por los algoritmos de detección de esquinas o de sustracción de fondo.	• El uso incorrecto de alguna transformación podría eliminarnos zonas relevantes para nuestro análisis, o bien podría convertir dos o más zonas en una sola.
Detección de bordes o contornos	• Nos permite detectar un objeto (más precisamente, los bordes del mismo) dentro de una imagen.	• Estos algoritmos también son sensibles a los cambios en la iluminación en la imagen, el tipo de cámaras y lentes y el movimiento de los objetos.
Transformada de la distancia	• Podemos utilizar esta técnica para obtener todos los puntos vecinos (con una distancia máxima predefinida) a un punto dado.	• La transformada de la distancia es muy sensible a pequeños cambios en el objeto. • También es muy sensible al ruido.

Como podemos observar, ningún método es completamente fiable, pero si logramos una buena combinación de los mismos podremos mitigar las falencias que presentará cada uno si lo aplicáramos por separado.

5. SISTEMA DE DETECCION Y SEGUIMIENTO

En este capítulo detallaremos paso a paso como se utilizaron los algoritmos y métodos investigados, para la implementación de un programa que lograra la detección y el seguimiento de los peatones a partir de un video dado.

Para el desarrollo se utilizó el lenguaje de programación Python v2.7 con la librería de código abierto OpenCV v2.4.9 de Willow Garage. Estas herramientas son estándares en la industria y la comunidad científica.

A modo de resumen, presentaremos primero un diagrama que ilustre los pasos que realiza nuestro programa, desde la captura de un *"frame"* del video, hasta la detección y clasificación de cada uno de los *"tracklets"* (Figura 49).

Que es un *"tracklet"*? Este concepto fue incluido en la publicación *"Pedestrian Tracking by Associating Tracklets using Detection Residuals"* de Singh, Wu y Nevatia en 2008. Con el fin de formalizar el concepto de blob, se llama de esta forma a la entidad que representa y agrupa las características y la trayectoria propias de cada peatón.

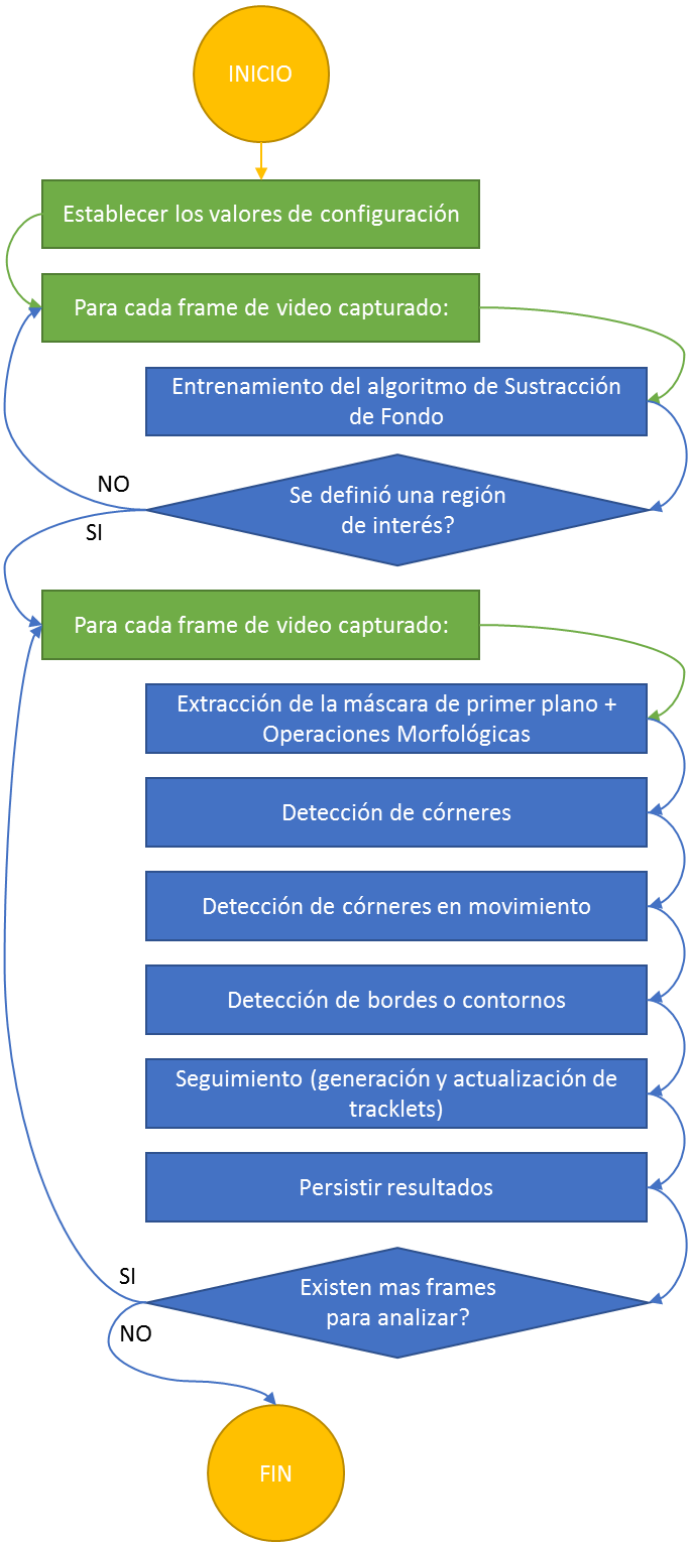


Figura 49: secuencia de etapas para la detección y seguimiento de peatones.

5.1. Inicialización

Antes de comenzar el análisis propiamente dicho del video en estudio, es necesario realizar algunas tareas de inicialización.

Establecer los valores de configuración

```
self.window_name = 'pedestrians'     # Video output window identifier
self.detect_interval = 3              # Check points every X frames
self.frame_idx = 0                    # Frames counter
self.clustering_threshold = 10        # Maximum distance between features
self.circle_radius = 15               # Radius of the circle to draw
self.track_color = (0, 255, 0)        # Color to draw (RGB)
self.min_features_per_contour = 1     # Minimum number of features per contour
self.min_contour_area = 500           # Minimum area of contour
```

A lo largo de esta sección, veremos en que se usa cada uno de estos parámetros.

Captura de video

El único parámetro de entrada de nuestro programa es la ruta donde se encuentra el video en estudio. Con este valor se inicializa el capturador de video utilizado para obtener secuencialmente todos los *"frames"* del mismo.

```
self.cap = cv2.VideoCapture(video_src)

while not self.start_analysis:
# Take frame
    ret, frame = self.cap.read()
```

Entrenamiento del algoritmo de sustracción de fondo

Dado que los videos en estudio no comienzan con una imagen totalmente estática, sino que en la mayoría de los casos existe la posibilidad de que los primeros *"frames"* ya contengan personas u objetos en movimiento, vamos a necesitar una etapa de entrenamiento del modelo de sustracción de fondo para que logre estabilizarse.

Luego de un cierto tiempo de aplicar el método *"frame"* a *"frame"*, lograremos definir entonces un modelo de fondo (*"background"*) estable que se adapta a los cambios del medio ambiente y que se actualiza de acuerdo a estos cambios, por medio del cual se obtiene la máscara de primer plano (*"foreground"*) destacando los objetos en movimiento (Figura 50).

```
# Background Subtractor (settings): history, varThreshold, bShadowDetection
self.bs = cv2.BackgroundSubtractorMOG2(100, 16.0, False)
```

```
foreground = self.bs.apply(frame_gray, learningRate=0.01)
```

Figura 50: resultado de aplicar el método de sustracción de fondo a un *"frame"*.

Definir Región de Interés

El programa permite que el usuario seleccione con el mouse la región de interés deseada (ROI), esta opción es útil cuando se quiere limitar el análisis a solo una superficie del video en estudio (Figura 51).

La librería de OpenCV ofrece métodos para realizar esta tarea fácilmente. Se implementó una clase llamada RectSelector que detecta los eventos del mouse, más precisamente, cuando el usuario presiona el botón izquierdo y arrastra el puntero dibujando un rectángulo.

```
class RectSelector:
   def onmouse(self, event, x, y, flags, param):
      x, y = np.int16([x, y])
      if event == cv2.EVENT_LBUTTONDOWN:
         self.drag_start = (x, y)
      if self.drag_start:
         if flags & cv2.EVENT_FLAG_LBUTTON:
xo, yo = self.drag_start
            x0, y0 = np.minimum([xo, yo], [x, y])
            x1, y1 = np.maximum([xo, yo], [x, y])
self.drag_rect = None
            if x1-x0 > 0 and y1-y0 > 0:
self.drag_rect = (x0, y0, x1, y1)
else:
            rect = self.drag_rect
            self.drag_start = None
            self.drag_rect = None
            if rect:
               self.callback(rect)

   def draw(self, vis):
      if not self.drag_rect:
         return False
      x0, y0, x1, y1 = self.drag_rect
cv2.rectangle(vis, (x0, y0), (x1, y1), (0, 255, 0), 2)
      return True
```

Dicha clase requiere de dos parámetros: el nombre de la ventana y el método a ejecutar una vez que el rectángulo (el ROI) se complete (en nuestro caso, es donde comienza el análisis).

```
self.rect_selector = common.RectSelector(self.window_name, self.run_analysis)

# Draw mouse selection
self.rect_selector.draw(frame)
```

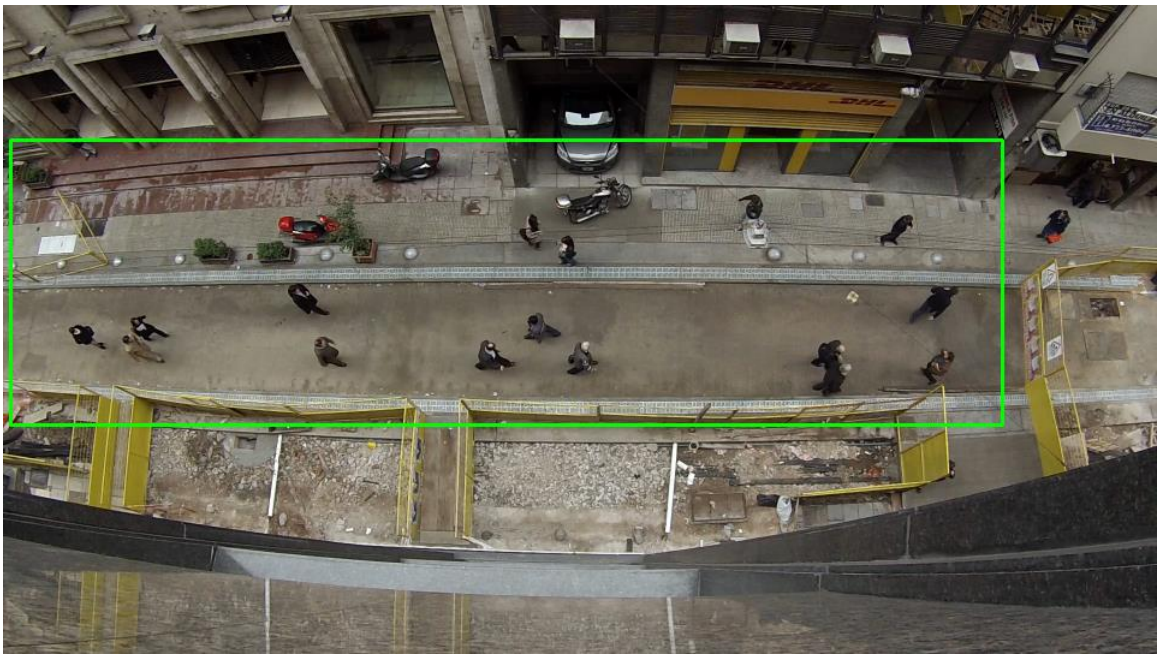


Figura 51: selección de la región de interés.

5.2. Detección

A continuación se detalla paso a paso todos los algoritmos, metodologías y transformaciones realizadas para la detección y seguimiento de las personas en el video provisto por el usuario.

Todos estos pasos se encuentran dentro de un bucle cuya salida es el final del video, es decir, cuando no haya más *"frames"* que analizar.

```
while True:
# Take frame
   ret, frame = self.cap.read()
   if frame is None:  # end
print 'DONE!'
break
```

Extracción de la máscara de primer plano (*"foreground"*)

Un algoritmo robusto de sustracción de fondo debe ser capaz de manejar los cambios de iluminación, movimientos repetitivos de desorden y cambios de escena a largo plazo. Es por eso que, de los algoritmos estudiados nos inclinamos por "Mixturas Gaussianas (MoG) 2".

Este algoritmo, expuesto como ya vimos por Zivkovic en agosto de 2004, es una mejora adaptable del modelo de mezclas gaussianas propuesto por Stauffer y Grimso para la sustracción de fondo. La diferencia radica en que el algoritmo de Zivkovic selecciona de manera apropiada el número de distribución gaussiana para el modelamiento de cada pixel de fondo, lo que permite una mejor adaptación ante las variaciones por cambios de iluminación en la escena. Adicionalmente, este algoritmo realiza también la detección de sombras.

```
# Convert image to specified color space
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# Apply background subtractor
foreground = self.bs.apply(frame_gray, learningRate=0.01)
```

Notar que además del *"frame"*, estamos especificando el parámetro *"learningRate"*, esto es la velocidad en cual el modelo se va a adaptar a los cambios en el video. Los valores bajos corresponden a un modelo de adaptación lento. Los valores altos hacen que el modelo deba adaptarse rápidamente a los cambios en la escena.

Figura 52: resultado de aplicar el método de sustracción de fondo a un *"frame"*.

Como podemos ver en la figura 52, el resultado contiene un numero considerado de "falsos positivos" o lo que comúnmente se le llama "ruido". Esto es, pixeles blancos aislados que se detectaron como si estuviesen en movimiento pero que en realidad aparecen a causa de pequeñas vibraciones de la webcam o sombras proyectadas sobre la imagen.

Utilizaremos operaciones morfológicas, donde el objetivo aquí será realizar una erosión seguida de una dilatación para eliminar ese ruido blanco sobre las zonas negras (Figura 53).

Erosión va a hacer desaparecer los puntos blancos aislados y una posterior dilatación se ocupara de reestablecer el cambio provocado por la erosión en las zonas blancas que sobrevivieron.

```
kernel = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (4, 4))

foreground = cv2.erode(foreground, kernel)
foreground = cv2.dilate(foreground, kernel)
```

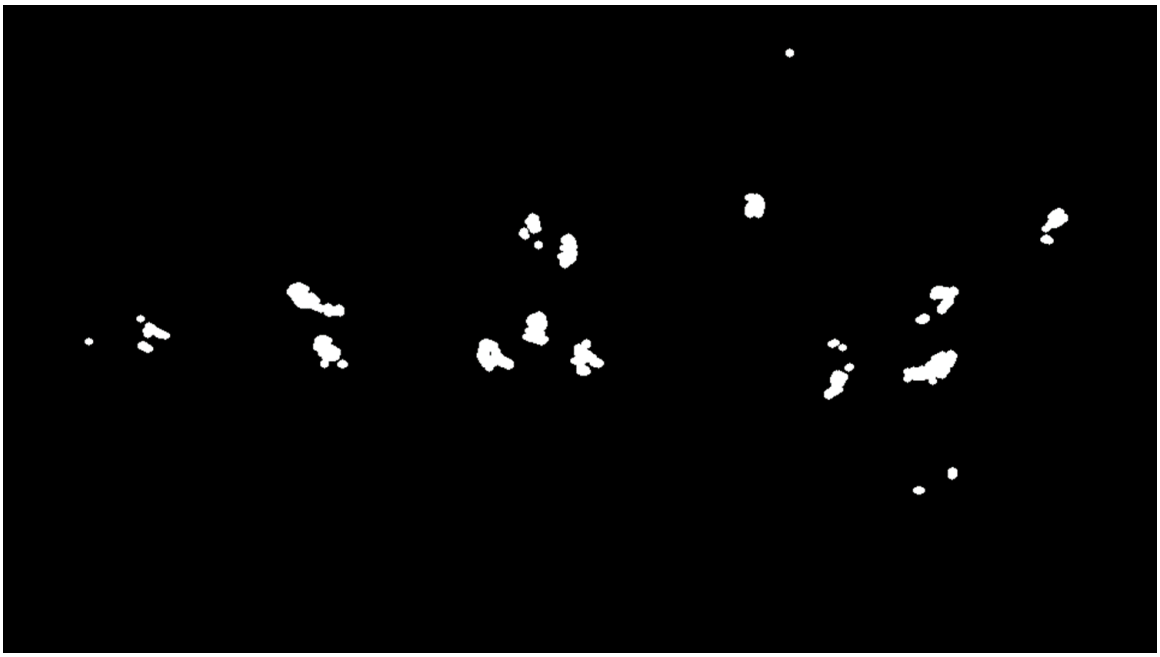


Figura 53: eliminación del "ruido" a través de técnicas de erosión y dilatación.

Detección de esquinas

Cada un numero pre establecido de *"frames"*, se llevara a cabo la detección de esquinas o *"corners"* que como ya vimos anteriormente, se caracteriza por ser una región de la imagen con cambios de intensidad en diferentes direcciones.

Para esto, utilizaremos el método de *"Good Features to Track"* (el cual es una mejora al método de detección de esquinas de Harris) utilizando como mascara el *"foreground"* previamente calculado. Esto es, solo se buscaran esquinas en las zonas del *"frame"* que estamos considerando que están en movimiento (Figura 54).

```
# Convert image to specified color space
frame_gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

# ShiTomasi Corner Detection (settings)
feature_params = dict(maxCorners=500,
   qualityLevel=0.3,
   minDistance=10,
   blockSize=7)

# Look for corner points every X (detect_interval) frames
if self.frame_idx % self.detect_interval == 0:
# Find corners with 'Good Features to Track' (using foreground as mask)
   corner_points = cv2.goodFeaturesToTrack(frame_gray, mask=foreground,
**feature_params)
```

Para mayor precisión utilizaremos además una funciona de OpenCV llamada *"cornerSubPix"* que detecta la posición exacta de las esquinas con una precisión de menos de un pixel.

```
# Refine corner locations (settings)
subpix_params = dict(zeroZone=(-1, -1),
    winSize=(10, 10),
    criteria=(cv2.TERM_CRITERIA_COUNT | cv2.TERM_CRITERIA_EPS, 20, 0.03))

if corner_points is not None:
    # Refine the corner locations
    cv2.cornerSubPix(frame_gray, corner_points, **subpix_params)
```

Figura 54: detección de esquinas en las regiones de la imagen que se consideran en movimiento.

Ahora sí, ya tenemos nuestra lista definitiva de esquinas para el *"frame"* actual, el último paso es guardarlos en el *"array"* de trayectorias.

```
# Include corner points into tracks array
for x, y in np.float32(corner_points).reshape(-1, 2):
    if roi_mask[y - 1, x - 1] != 0:  # Point is inside ROI
        self.tracks.append([(x, y)])
```

Detección de esquinas en movimiento

El siguiente paso será detectar si las esquinas calculadas en el bucle anterior realizaron un desplazamiento en el *"frame"* que se está analizando con respecto al *"frame"* anterior. Para eso utilizaremos el concepto de flujo óptico (*"optical flow"*) que no es ni más ni menos que el movimiento aparente de los píxeles de una imagen de un cuadro a otro dentro de una secuencia de video.

Hay varias alternativas para calcular el flujo óptico, una de las más populares y la que utilizaremos es el método de Lukas-Kanade basado en pirámides (aproximaciones sucesivas reduciendo la resolución de la imagen hasta llegar a la imagen original), implementado en la función *"calcOpticalFlowPyrLK"* de OpenCV. Para esto necesitaremos el *"frame"* anterior (*"prev_frame_gray"*), el *"frame"* actual (*"frame_gray"*) y las esquinas detectadas anteriormente por el método de *"Good Features to Track"* que guardamos en el *"array"* de trayectorias (*"tracks"*).

```
# Optical Flow with Lukas-Kanade (settings)
lk_params = dict(winSize=(4, 4),
maxLevel=5,
   criteria=(cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 0.03))

# Work with previous and current frames
img0, img1 = self.prev_frame_gray, frame_gray

# Get previous corner points
p0 = np.float32([tr[-1] for tr in self.tracks]).reshape(-1, 1, 2)

# Calculate Optical Flow with Lukas-Kanade
p1, st, err = cv2.calcOpticalFlowPyrLK(img0, img1, p0, None, **lk_params)
```

Como argumentos, este método admite varios parámetros pero los más importantes son:

- Tamaño de la ventana de búsqueda (*"winSize"*): determina el ancho y alto de la ventana alrededor del punto original en la cual se va a buscar el punto desplazado, un tamaño grande mejora la precisión pero puede comprometer la velocidad de cálculo.
- Número de pirámides (*"maxLevels"*): cantidad de veces que se reduce la resolución de la imagen para los sucesivos cálculos del flujo óptico. Un valor de 5 mejora el rendimiento del algoritmo respecto a no aplicar reducción, valores superiores no mejoran la eficiencia e incluso llegan a reducirla.
- Criterios de terminación (*"criteria"*): determinan cuando la función debe parar de buscar el flujo óptico para un punto determinado. En nuestro caso hemos

establecido que el algoritmo pare si el error de la solución (diferencia entre una iteración y la anterior) es menor a 0.3 píxel o si se han hecho más de 10iteraciones.

De estos tres argumentos el más notable a nivel de rendimiento es el tamaño de la ventana.

La salida del método será la nueva posición de las esquinas en el *"frame"* actual. Por lo que obtendremos una matriz con un 1 si el punto fue encontrado, de lo contrario tendremos un 0.

Como último paso, y a modo de robustecer nuestro cálculo, ejecutaremos nuevamente el método *"calcOpticalFlowPyrLK"* pero esta vez invirtiendo los parámetros de entrada, es decir, simulando que el *"frame"* actual es el *"frame"* anterior y el *"frame"* anterior es el actual. Y además, la posición inicial de las esquinas ahora es el output que obtuvimos al llamar dicho método la primera vez (*"p1"*).

```
# Run a backward-check of the optical flow points to select only good ones
p0r, st, err = cv2.calcOpticalFlowPyrLK(img1, img0, p1, None, **lk_params)
```

Comparamos el resultado calculado (*"p0r"*) con el resultado real (*"p0"*) para así ignorar de *"p1"* los puntos calculados erróneamente por el algoritmo.

```
# Select good points
d = abs(p0-p0r).reshape(-1, 2).max(-1)
good = d < 1
```

Finalmente, actualizamos nuestro *"array"* de trayectorias (*"self.tracks"*), que como podemos ver, se está transformando en un *"array"* donde guardamos la posición de cada esquina en movimiento a lo largo de la secuencia de video (Figura 55).

También pondremos en otro *"array"* (*"features"*) las esquinas en movimiento detectados en el *"frame"* actual.

```
new_tracks = []
features = []

# Track only good points
for tr, (x, y), good_flag in zip(self.tracks, p1.reshape(-1, 2), good):
if not good_flag:
continue
tr.append((x, y))
new_tracks.append(tr)
features.append((int(x), int(y)))
```

```
self.tracks = new_tracks
```

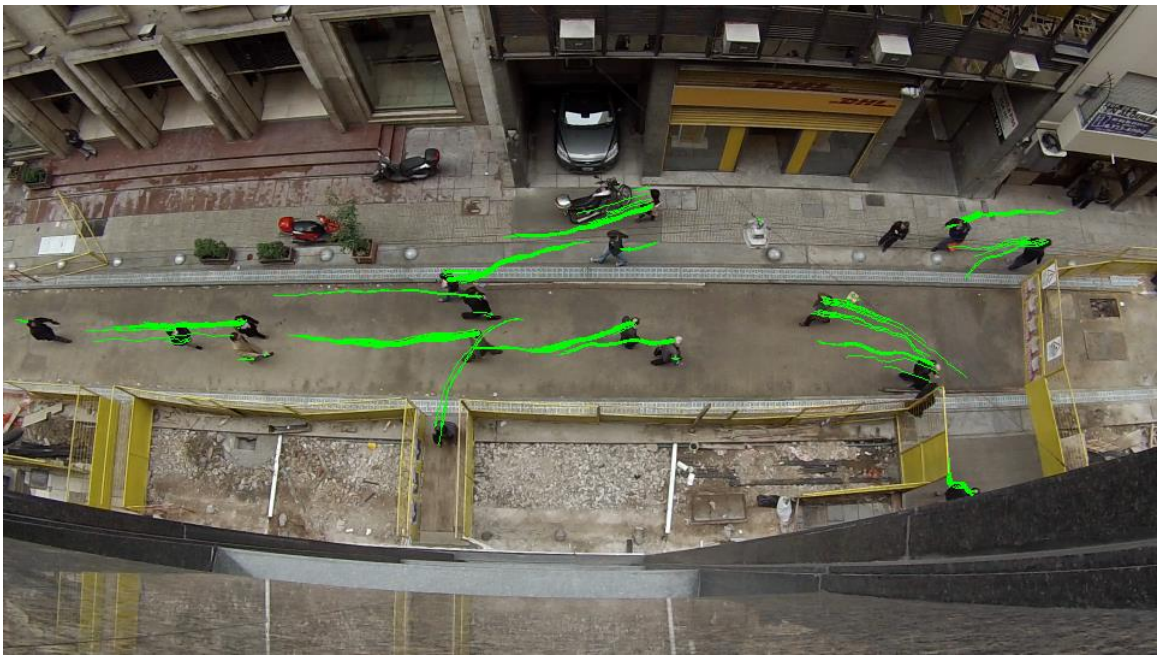


Figura 55: cálculo de la trayectoria de cada esquina en movimiento.

Detección de bordes o contornos

El siguiente paso del proceso es la detección de bordes, es decir, los contornos que proporcionan la silueta de los objetos de primer plano, permitiendo identificar sus formas (Figura 56).

Para esto utilizaremos el método *"cv2.findContours"* que implementa el algoritmo de Suzuki y Abe (detallado anteriormente). El resultado que se obtiene no es una nueva imagen, sino una colección de conjuntos de puntos sobre la imagen.

```
contours, hierarchy = cv2.findContours(foreground, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
```

Como argumentos, este método admite varios parámetros pero los más importantes son:

- Imagen inicial (*"source"*). Debe ser una imagen binaria, es decir una imagen que tan solo consta de pixeles con valor cero (0) o uno (1). En nuestro caso utilizaremos la máscara de primer plano donde recordemos que está compuesta por pixeles blancos de valor uno (1) que indican los objetos en movimiento y por pixeles negros de valor cero (0) que representan el fondo.

- Modo de retorno de los bordes (*"mode"*). Utilizaremos el valor CV_RETR_TREE para obtener no solo los contornos exteriores, sino también los interiores a otros, y retornarlos clasificados jerárquicamente.
- Método de aproximación (*"method"*). Utilizaremos el valor CV_CHAIN_APPROX_SIMPLE que comprime los segmentos horizontales, verticales y diagonales y sólo deja sus puntos finales. Por ejemplo, un borde rectangular será representado por solo 4 puntos.

Figura 56: detección de contornos en las regiones de la imagen que se consideran en movimiento.

5.3. Seguimiento

Hasta ahora hemos obtenido dos resultados importantes en el *"frame"* que estamos analizado: las esquinas en movimiento y los bordes o contornos en la máscara de primer plano.

Con esta información nos quedaran dos desafíos finales:

1. Agrupar las esquinas en movimiento de acuerdo a que contornos pertenecen, es decir, lograr una intersección de estos dos resultados y con ellos definir los blobs o *"tracklets"* presentes en el *"frame"*. Como dijimos anteriormente, un *"tracklet"* es un peatón en movimiento en nuestra secuencia de video.
2. Tener la capacidad de detectar para cada blob o *"tracklet"* que analizamos, si se trata de uno totalmente nuevo (es decir, un peatón que acaba de aparecer en la secuencia) o si se trata de un *"tracklet"* que ya detectamos en un *"frame"* anterior (es decir, un peatón que ya tenemos información de su trayectoria).

Para llevar a cabo esta tarea, utilizaremos un bucle para recorrer y analizar uno por uno los bordes o contornos calculados en el paso anterior.

```
current_blobs = []
for contour in contours:
```

Lo primero que haremos es calcular el área del contorno utilizando el método *"cv2.contourArea"* que implementa el teorema de Green. Este teorema relaciona una integral de línea a lo largo de una curva cerrada simple en un plano con una integral doble en la región encerrada.

El único objetivo de esta operación es excluir contornos cuya área no supere el mínimo pre establecido:

```
area = cv2.contourArea(contour)
if area > self.min_contour_area:
```

El siguiente paso será obtener las esquinas en movimiento que están dentro de este contorno y almacenarlos en un *"array"* (*"contour_features"*). Para eso utilizaremos el método *"cv2.pointPolygonTest"* que devolverá 1 si la esquina está dentro del contorno, 0 si está en el límite y -1 si se encuentra afuera del mismo.

```
# Take contour's features
contour_features = []
contour_features_tracks = []

for feature in features:
dist = cv2.pointPolygonTest(contour, feature, False)
```

```
if dist >= 0:  # 1: inside the contour, 0: on the contour
contour_features.append(feature)
```

Por otro lado, en otro *"array"* (*"contour_features_tracks"*) guardaremos las trayectorias, es decir, el historial de movimiento de cada esquina que pertenece al contorno.

```
for track in self.tracks:
    last_track_point = (int(track[-1][0]), int(track[-1][1]))
    if last_track_point == feature:
        contour_features_tracks.append(track)
break
```

Una última validación que realizaremos antes de la generación del *"tracklet"* será la de ignorar los contornos cuyo número de esquinas asociadas no supere el mínimo pre establecido.

```
if len(contour_features) >= self.min_features_per_contour:
```

En este momento, ya tenemos todo listo para generar un *"tracklet"* (es decir, una representación de una persona en movimiento) a partir de un contorno y sus esquinas asociadas. Lo que haremos a continuación será entonces la inicialización del mismo y el cálculo de una serie de propiedades que nos permitirán detectar si dicho *"tracklet"* es nuevo en nuestro análisis (una persona que acaba de ingresar a la secuencia de video) o si es un *"tracklet"* existente que ya fue detectado en un *"frame"* anterior.

```
blob = blobs.blob(contour, contour_features, contour_features_tracks, self.blobs,
self.frame_idx, self.invalid_area_for_new_blobs)
```

Donde:

- *contour*: es la representación del contorno o borde que estamos analizando.
- *contour_features*: es un *"array"* de las esquinas en movimiento que detectamos y que están dentro del contorno.
- *contour_features_tracks*: es un *"array"* con las trayectorias hasta el momento de las esquinas almacenadas en *contour_features*.
- *self.blobs*: es un *"array"* con todos los *"tracklets"* detectados hasta el momento (útil para saber si el contorno que estamos analizando corresponde a uno de ellos o debiéramos considerarlo como uno completamente nuevo).
- *self.frame_idx*: es el número de *"frame"* actual.
- *self.invalid_area_for_new_blobs*: es un área del video pre definida donde indicamos que no van a aparecer nuevos *"tracklets"*. Es decir, áreas del video donde sería

imposible que aparezca un nuevo peatón, por ejemplo, el centro de la imagen, ya que podemos asumir que un nuevo peatón aparecerá siempre en los bordes de la misma.

El primer paso para la inicialización del *"tracklet"* será establecer una serie de propiedades, algunas directas, es decir, usando directamente los parámetros de entrada:

```
self.contour = contour
self.last_frame_idx = frame_idx
self.last_features = features
self.last_features_tracks = features_tracks
```

Y otras propiedades que tendremos que calcular:

- Promedio de las esquinas: Tomando todas las esquinas asociadas al contorno, buscar un punto promedio entre estas.

```
last_features_avg = np.mean(self.last_features, axis=0)
self.last_features_avg = int(last_features_avg[0]), int(last_features_avg[1])
```

- Centroide: Es un punto que define el centro geométrico del contorno. La función *"cv2.moments(contour)"* nos da como output un diccionario. Nos interesa las claves "m10" (el centroide "X" del contorno), "m01" (el centroide "Y") y "m00" (el área del contorno). Si dividimos "m10" por "m00" obtenemos la media de las distancias X, que es la coordenada X del centro del contorno. Lo mismo con "m01" para obtener la coordenada Y.

```
Moment = cv2.moments(contour)
self.cent = int(moment['m10']/moment['m00']), int(moment['m01']/moment['m00'])
```

- Diámetro equivalente: Hace referencia al diámetro del círculo el cual tiene un área igual a la del área del contorno.

```
cnt_area = cv2.contourArea(contour)
self.equi_diameter = np.sqrt(4 * cnt_area / np.pi)
```

- Posibilidad de que sea un nuevo *"tracklet"*: Para eso verificaremos si el centroide previamente calculado se encuentra fuera del área pre-definida donde indicamos que no van a aparecer nuevos *"tracklets"* (*"invalid_area_for_new_blobs"*). Es decir, áreas del video donde sería imposible que aparezca un nuevo peatón (por ejemplo, en el centro del mismo). El método *"cv2.pointPolygonTest"* retornara 1 si el centroide está dentro de esta área, 0 si está en el borde y -1 si se encuentra afuera de

la misma. En nuestro caso, la posibilidad de que sea un nuevo *"tracklet"* existirá entonces cuando este método devuelva "-1".

```
can_be_new_blob = cv2.pointPolygonTest(invalid_area_for_new_blobs, self.cent,
False) < 0
```

Con todas estas propiedades establecidas, pasaremos ahora a comparar nuestro contorno actual con el *"array"* de *"tracklets"* detectados en *"frames"* anteriores que tenemos hasta el momento para así saber si se trata de un nuevo *"tracklet"* o de uno ya existente que fue identificado en un *"frame"* previo.

El primer filtro que aplicaremos a nuestro *"array"* de *"tracklets"* es para ignorar los que no fueron analizados recientemente. Esto es, utilizando la propiedad *"last_frame_idx"* nos quedaremos solo con los que fueron detectados en los últimos N *"frames"*.

```
for blob in blobs:
   if not self.isRecentFrame(blob.last_frame_idx):
      continue

# …

def isRecentFrame(self, blob_last_frame_idx, max_distance=100):
   return self.last_frame_idx - blob_last_frame_idx < max_distance
```

Con la lista de *"tracklets"* ya filtrada, haremos una comparación entre las propiedades del contorno actual con las propiedades de cada uno de ellos. Si encontramos un *"tracklet"* de esta lista cuyas características se asemejen a las del contorno actual, podemos considerar entonces que se trata del mismo *"tracklet"*.

En el siguiente diagrama (Figura 57) podemos observar las distintas comparaciones que realizaremos y luego explicaremos en detalle de que se trata cada una de ellas.

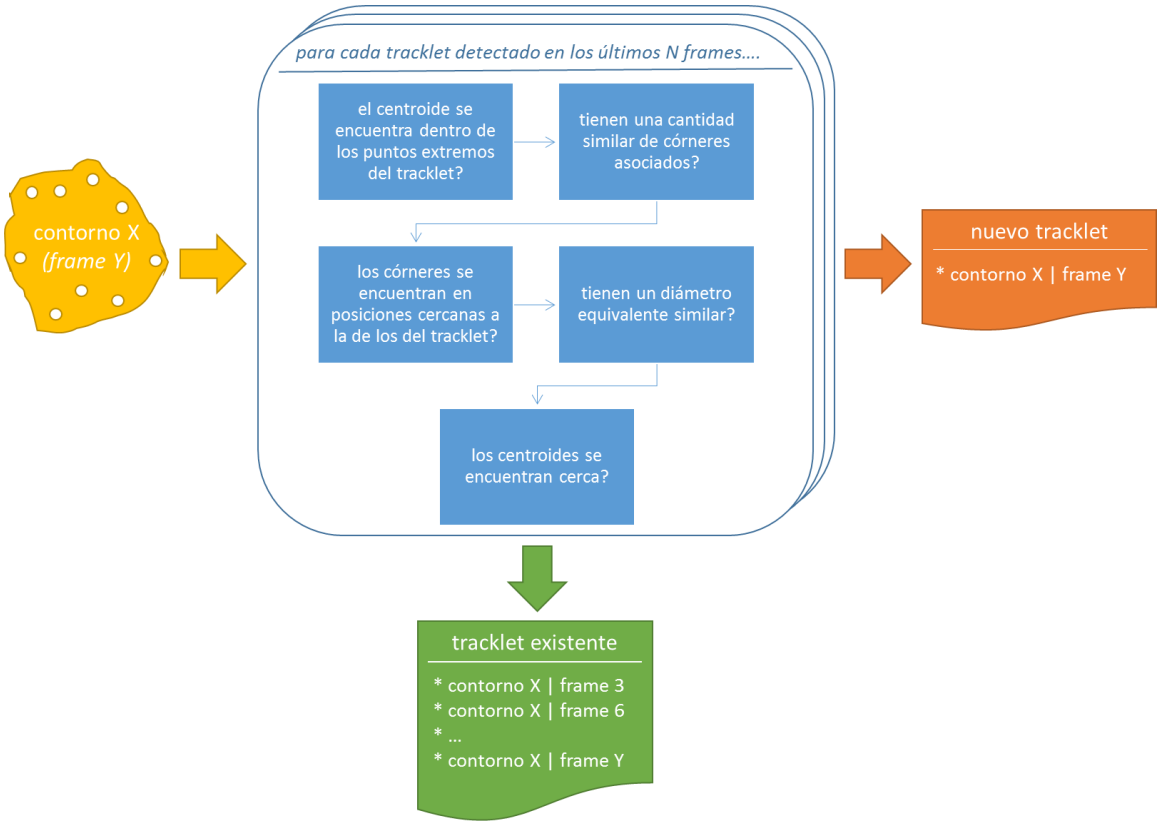


Figura 57: validaciones realizadas para detectar si un contorno debe considerarse un nuevo *"tracklet"* o uno existente.

Describiremos ahora las comparaciones realizadas:

- Centroide y puntos extremos. Lo que haremos es primero calcular los cuatro puntos extremos del contorno actual, para luego chequear que el centroide del *"tracklet"* que estamos comparando se encuentre dentro o en una zona cercana al rectángulo imaginario que formaríamos con:
  - El valor "y" de los puntos extremos norte y sur.
  - El valor "x" de los puntos extremos este y oeste.

```
def isCentroidInsideExtremePoints(self, blob_cent, gap_x=50, gap_y=50):
    left_most = tuple(self.contour[self.contour[:, :, 0].argmin()][0])
    right_most = tuple(self.contour[self.contour[:, :, 0].argmax()][0])
    bottom_most = tuple(self.contour[self.contour[:, :, 1].argmin()][0])
    top_most = tuple(self.contour[self.contour[:, :, 1].argmax()][0])

return
blob_cent[1] < (top_most[1] + gap_y) and# y1 + gap_y
blob_cent[1]> (bottom_most[1] - gap_y) and# y2 + gap_y
blob_cent[0] < (right_most[0] + gap_x) and       # x1 + gap_x
```

```
blob_cent[0] > (left_most[0] - gap_x)# x2 + gap_x
```

- Número de esquinas asociados. Lo que haremos es una resta entre la cantidad de esquinas asociadas de ambos, quedándonos con el valor absoluto del resultado para luego compararlo con el valor pre establecido.

```
self.containsSimilarNumberOfFeatures(len(blob.last_features))

# …

def containsSimilarNumberOfFeatures(self, blob_last_features_len,
max_difference=30):
    return abs(len(self.last_features) - blob_last_features_len) < max_difference
```

- Distancia promedio entre esquinas asociadas. Se calcula el punto promedio entre las esquinas de cada uno, utilizando la función *"np.mean"* que realiza la suma de los elementos a lo largo del eje y la divide por el número de elementos. Con ambos puntos promedios, comparamos que la distancia euclidiana entre los mismos no supere el valor pre establecido.

```
self.featuresDistance(blob.last_features)

# …

def featuresDistance(self, blob_last_features):
    # Average point between current features points
    avg1 = np.mean(self.last_features, axis=0)

    # Average point between blob last features points
    avg2 = np.mean(blob_last_features, axis=0)

# Euclidean distance
return np.sqrt(np.sum((avg1 - avg2)**2))
```

- Diámetro equivalente. Haremos una resta entre el diámetro equivalente de ambos entes, quedándonos con el valor absoluto del resultado para luego compararlo con el valor pre establecido.

```
self.equivalentDiameterDifference(blob.equi_diameter)

# …

def equivalentDiameterDifference(self, blob_equi_diameter):
    return abs(self.equi_diameter - blob_equi_diameter)
```

- Distancia entre centroides. Aquí solo comparamos que la distancia euclidiana entre los centroides de ambos no supere el valor pre establecido.

```
self.centroidDistance(blob.cent)

# …

def centroidDistance(self, blob_cent):
cent1 = np.array((self.cent[0], self.cent[1]))
    cent2 = np.array((blob_cent[0], blob_cent[1]))

    # Euclidean distance
    return np.sqrt(np.sum((cent1 - cent2)**2))
```

Si logramos encontrar un *"tracklet"* que mejor se adapte a todos estos requisitos, entonces podemos considerar que el contorno que estamos inicializando ya fue detectado en un *"frame"* anterior. Es decir, una persona ya detectada previamente que continua su trayectoria. Entonces lo que haremos es actualizar sus propiedades:

```
# Existing blob
existing_blob.cent = self.cent
existing_blob.contour = self.contour
existing_blob.equi_diameter = self.equi_diameter
existing_blob.last_frame_idx = self.last_frame_idx
existing_blob.last_features = self.last_features
existing_blob.last_features_tracks = self.last_features_tracks
existing_blob.last_features_avg = self.last_features_avg

# Append current centroid to the history array
existing_blob.history.append(self.cent)

# Append current features to the features array
existing_blob.features.append(self.last_features)
```

Si por lo contrario, no encontramos ninguno que se asemeje, entonces se trata de un nuevo *"tracklet"* en nuestro sistema. Es decir, una persona que acaba de aparecer en la secuencia de video.

```
# New blob
self.uid = random.getrandbits(32)
self.track_color = (np.random.randint(0, 255), np.random.randint(0, 255),
np.random.randint(0, 255))
```

```
# Initialize history array
self.history = [self.cent]

# Initialize features array
self.features = [self.last_features]

# Update blobs list
blobs.append(self)
```

Con el *"tracklet"* ya inicializado, realizaremos las operaciones de salida. Esto es, dibujar un circulo en el *"frame"* actual que identifique la posición del centroide junto con el ID del *"tracklet"*.

```
# Show blob centroid
cv2.circle(vis, blob.cent, 10, blob.track_color, -1)

# Show blob id
common.draw_str(vis, blob.cent, blob.uid)
```

Y por último, actualizamos un archivo en formato CSV donde guardamos: el ID del *"frame"*, el ID del *"tracklet"* y las posiciones "x" e "y" del centroide (Figura 58).

```
# Persist tracking
with open(self.csv_file_name, 'a') as f:
   f.write('{0},{1},{2},{3}\n'.format(
      self.frame_idx, blob.uid, blob.cent[0], blob.cent[1]))
f.closed
```

frame	id	x	y
1	940926097	1239	540
2	940926097	1235	539
3	940926097	1231	539
4	940926097	1226	541
5	940926097	1221	542
5	1188128832	1257	251
6	940926097	1216	543
6	1188128832	1253	252
7	940926097	1212	544
7	1188128832	1252	252
8	940926097	1209	545
8	1188128832	1250	251

Figura 58: ejemplo de archivo CSV con las posiciones de cada *"tracklet"* en un *"frame"* determinado.

En este punto, ya estamos listos para comenzar el análisis del próximo contorno. Y una vez que esta lista se recorra íntegramente, comenzaremos con el próximo *"frame"* de la secuencia de video hasta la finalización del mismo.

6. RESULTADOS

En este capítulo describiremos algunos aspectos básicos de la evaluación de desempeño y finalmente presentaremos los resultados obtenidos al aplicar la metodología descripta en una serie de videos de peatones tomados en la ciudad de Buenos Aires.

6.1. Evaluación de Desempeño

En los últimos años ha habido un creciente interés en realizar evaluaciones sistemáticas sobre los sistemas de seguimiento de objetos con el fin de poder medir el grado de efectividad de los mismos. Entre los proyectos más conocidos podemos nombrar:

- CHIL, *"Computers In the Human Interaction Loop"*, http:// chil.server.de/
- AMI, *"Augmented Multiparty Interaction"*, http://www. amiproject.org/
- VACE, *"Video Analysis and Content Extraction"*, http://www .informedia.cs.cmu.edu/arda/vaceII.html
- ETISEO, *"Video Understanding Evaluation"*, http://www. silogic.fr/etiseo/
- CLEAR, *"Classification of Events, Activities and Relationships"*, http://www.clear-evaluation.org/

Sin embargo, todavía no se ha llegado a un consenso general para lograr un único procedimiento de evaluación basado en principios que utilicen un conjunto de métricas objetivas comunes e intuitivas para medir el desempeño de este tipo de sistemas.

Para permitir una mejor comprensión de las métricas propuestas (TABLA II), primero explicaremos que cualidades esperamos idealmente de un sistema de seguimiento de objetos:

- Debe detectar en todo momento el número exacto de objetos presentes en la escena y estimar la posición de cada uno tan preciso como sea posible.
- Debe lograr un seguimiento constante de cada objeto en el tiempo, a cada objeto se le debe asignar un identificador único que se mantenga constante a lo largo de la secuencia (incluso después de una oclusión temporal).

Esto nos lleva entonces a los siguientes criterios de diseño para las métricas de rendimiento:

- Deberán permitir juzgar la precisión de un sistema para determinar la ubicación exacta de los objetos.
- Deberán reflejar su capacidad para realizar un seguimiento constante de las configuraciones de los objetos a través del tiempo, es decir, trazar correctamente las trayectorias de los objetos, produciendo exactamente una trayectoria por objeto.

Para determinar la efectividad de nuestra implementación, definiremos entonces nuestras propias métricas que nos permitirán:

- Cuantificar el grado de confiabilidad de la metodología propuesta.
- Poder compararla con métodos planteados por otros autores.
- Detectar bajo qué circunstancias o características de la escena nuestro método es menos eficiente.

Utilizaremos tres videos de 30 segundos de duración cada uno obtenidos a partir de una cámara estática ubicada en una calle peatonal de la ciudad de Buenos Aires (ver TABLA III).

TABLA II: métricas propuestas con su respectiva descripción.

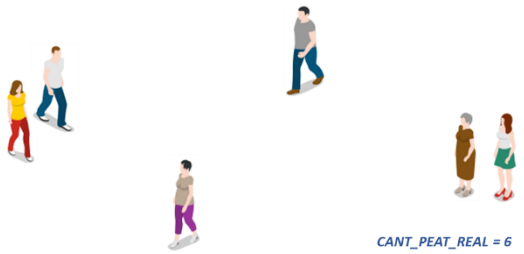
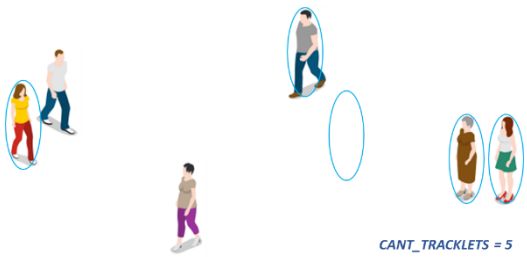
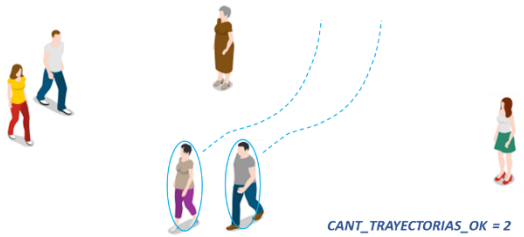
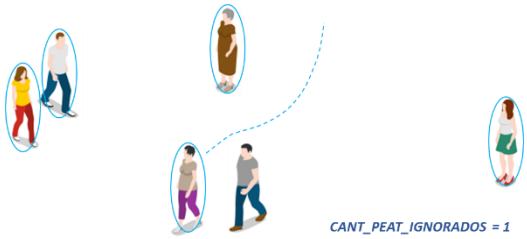
ID	Descripción
CANT_PEAT_REAL	Cantidad de peatones que entraron a escena.
	
CANT_TRACKLETS	Cantidad de *"tracklets"*. Esto es, cantidad de veces que el sistema detectó (correcta o incorrectamente) un nuevo peatón en la escena.
	
CANT_TRAYECTORIAS_OK	Cantidad de peatones que fueron detectados durante todo su recorrido sin cambiar el ID del *"tracklet"*. Esto es, el número de peatones que, una vez detectados, su trayectoria fue continuamente monitoreada hasta salir de escena o hasta que el video llegó a su fin.
	
CANT_PEAT_IGNORADOS	Cantidad de peatones que no fueron detectados nunca a lo largo de toda su trayectoria.
	

TABLA III: videos utilizados para el cálculo de las métricas.

ID	Captura
GOPRO_01.MP4	
GOPRO_02.MP4	
GOPRO_03.MP4	

Podríamos utilizar videos de más larga duración y no influiría en los resultados finales, la única razón por la cual elegimos videos más cortos es que el cálculo para cada métricas se realizara a partir de la evidencia empírica, es decir, utilizando la observación directa sobre lo que el sistema nos va informando como salida (Figura 59).

Figura 59: Ejemplo de la salida del sistema en un *"frame"* dado.

6.2. Obtención de los resultados

Ejecutaremos entonces nuestra aplicación para cada uno de los videos y tomaremos nota de los valores de las métricas propuestas.

```
~$ ./seguimiento_peatones.py [<ruta_video>]
```

Video 1 (GOPRO_01.MP4)

Figura 60: video 1 (GOPRO01.MP4).

- **CANT_PEAT_REAL:** 29
- **CANT_TRACKLETS**: 44
- **CANT_TRAYECTORIAS_OK**: 16
- **CANT_PEAT_IGNORADOS:** 2

En este video tenemos un espacio altamente reducido (Figura 60), casi la mitad de la calle se encuentra inaccesible debido a obras que se están realizando. Esto provoca que los peatones caminen con menos espacio entre ellos, aumentando la posibilidad de cruces de trayectorias. El **55.2%** de las trayectorias registradas fueron exitosas. El resto se debió principalmente a la generación de *"tracklets"* innecesarios (esto es, se asumió que se trataba de un nuevo peatón entrando en escena cuando en verdad ya existía en el sistema).

Video 2 (GOPRO_02.MP4)

Figura 61: video 2 (GOPRO2.MP4).

- **CANT_PEAT_REAL:** 38
- **CANT_TRACKLETS**: 51
- **CANT_TRAYECTORIAS_OK**: 23
- **CANT_PEAT_IGNORADOS:** 3

En este video tenemos un espacio más amplio para que los peatones se movilicen (Figura 61). La particularidad de este video con respecto a los demás es que posee un mayor número de peatones transitando. El **60.5%** de las trayectorias registradas fueron exitosas. El resto se debió nuevamente a la generación de *"tracklets"* innecesarios ya que solo 3 peatones fueron ignorados (principalmente porque se mantuvieron estáticos durante toda la secuencia).

Video 3 (GOPRO_03.MP4)

Figura 62: video 3 (GOPRO03.MP4).

- **CANT_PEAT_REAL:** 25
- **CANT_TRACKLETS**: 57
- **CANT_TRAYECTORIAS_OK**: 13
- **CANT_PEAT_IGNORADOS:** 2

En este video tenemos nuevamente una calle obstaculizada por obras en construcción, menos de la mitad de la calle se encuentra disponible para transitar (Figura 62). Esto provoca que los peatones caminen con menos espacio entre ellos, aumentando la posibilidad de cruces de trayectorias. El **52%** de las trayectorias registradas fueron exitosas. El resto se debió nuevamente (y en mayor proporción que en el primer caso estudiado) a la generación de *"tracklets"* innecesarios.

A modo de resumen, presentamos la siguiente tabla (TABLA IV) con los valores de las métricas obtenidos en los tres videos estudiados:

TABLA IV: valores de las métricas para cada video.

Métrica	**Video**		
	GOPRO_01	**GOPRO_02**	**GOPRO_03**
CANT_PEAT_REAL	29	38	25
CANT_TRACKLETS	44 *(+51.7%)*	51 *(+34.2%)*	57 *(+128%)*
CANT_TRAYECTORIAS_OK	16 *(55.2%)*	23 *(60.5%)*	13 *(52%)*
CANT_PEAT_IGNORADOS	2 *(10.3%)*	3 *(7.9%)*	2 *(8%)*

De los valores obtenidos podemos observar lo siguiente:

- El número de peatones ignorados es bajo en todos los escenarios (entre 8% y 10%), esto significa que la detección de peatones llevada a cabo por el sistema es satisfactoria. La pérdida es mínima.
- Las trayectorias exitosas se encuentran entre un 55% y 60% del total de las mismas. Ahora, si bien tenemos identificadas el resto de las mismas, podemos explicar la falta de éxito debido a que el sistema genero una excesiva cantidad de nuevos *"tracklets"* cuando en realidad se trataba de *"tracklets"* ya existentes en el sistema. Las áreas donde más se generaron este tipo de problemas fueron en lugares con espacios reducidos donde se aglutinaron más de tres o cuatro peatones con trayectorias similares (Figura 63).

Figura 63: ejemplos de situaciones en donde nuestro método tuvo inconvenientes para la detección de *"tracklets"* existentes.

6.3. Comparación con otros métodos

Utilicemos ahora las métricas definidos en la sección anterior y apliquémoslas sobre algún método de otro autor que en la actualidad sea reconocido por la comunidad. Esto nos permitirá saber cuán lejos está nuestra implementación de lograr resultados aceptables.

El método elegido fue elaborado por Anton Milan, Stefan Roth y Konrad Schindler en el año 2014 y publicado bajo el nombre de *"Continuous Energy Minimization for Multi-Target Tracking"*.

Seleccionaremos entonces un video de 30 segundos de duración (Figura 64) y ejecutaremos ambos métodos (el implementado en el presente proyecto vs el método propuesto por Milan, Roth y Schindler) tomando nota nuevamente de las métricas que ya venimos utilizando (ver TABLA II).

Figura 64: video utilizado para comparar los resultados con proyectos de terceros.

Luego de configurar y ejecutar ambos métodos para el mismo video, los resultados arrojados son los siguientes (TABLA V):

Figura 65: salida en pantalla para un *"frame"* dado de nuestro método "Seguimiento de Personas en Secuencias de Video".

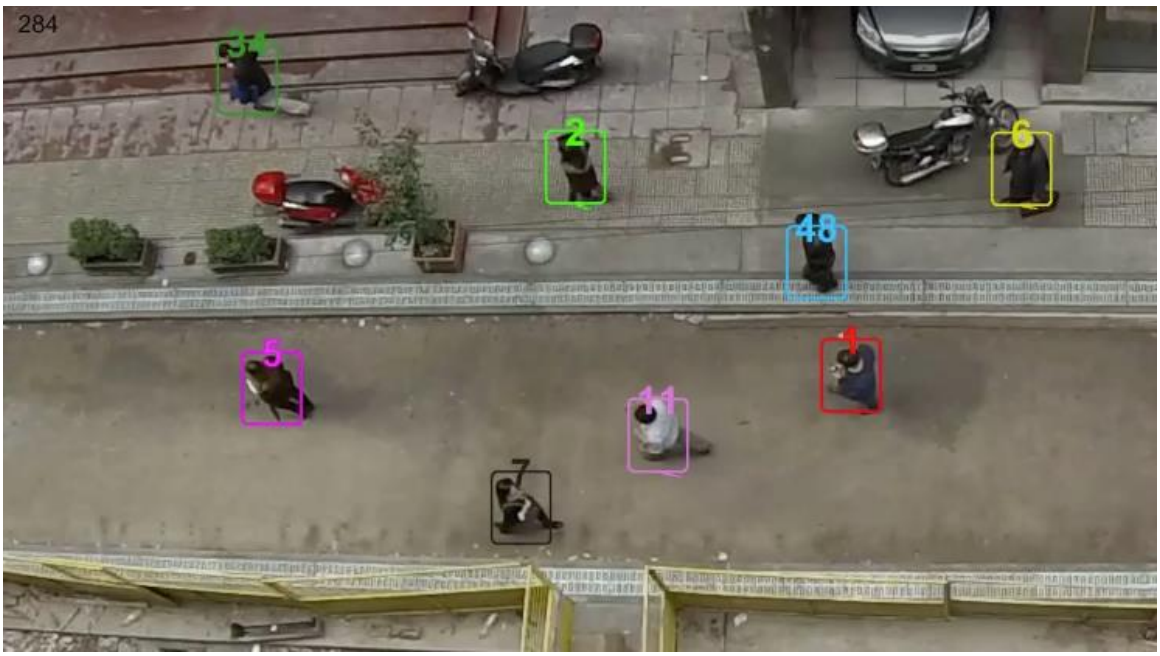


Figura 66: salida en pantalla para un *"frame"* dado del método *"Continuous Energy Minimization for Multi-Target Tracking"*.

TABLA V: valores de las métricas para cada método.

Métrica	Método	
	Seguimiento de Personas en Secuencias de Video	**Continuous Energy Minimization for Multi-Target Tracking**
CANT_ PEAT_REAL	12	
CANT_TRACKLETS	32 *(+166.7%)*	21 *(+75%)*
CANT_TRAYECTORIAS_OK	9 *(75%)*	8 *(66.7%)*
CANT_PEAT_IGNORADOS	0	0

De los valores obtenidos podemos observar lo siguiente:

- El número de trayectorias correctas es similar en los dos métodos con un porcentaje alto de eficiencia.
- El número de peatones ignorados es nulo con ambos métodos, lo que refuerza la tendencia que veníamos observando en otros videos donde nuestro método detectaba casi la totalidad de los peatones que aparecen en escena.
- Ambos métodos generan una excesiva cantidad de *"tracklets"* sobre todo nuestro método. Ahora, (y partiendo de la base de que no hubo peatones ignorados) si observamos el porcentaje de las trayectorias correctas (75% y 66.7%) podemos deducir que esa generación sin sentido de nuevos *"tracklets"* se da solo en peatones puntuales, es decir, en el 25% y 33.3% (según el método empleado) de los peatones involucrados.

7. CONCLUSIONES

El objetivo del presente trabajo era lograr la detección y seguimiento de personas en secuencias de video a partir de la captura de una cámara estática ubicada por encima de las mismas.

Luego de estudiar y analizar numerosos métodos y algoritmos logramos, a partir de la combinación de muchos de ellos, obtener un sistema que sea capaz de cumplir con nuestros objetivos.

Los resultados obtenidos en las distintas evaluaciones de desempeño arrojaron un punto fuerte en nuestro método: la detección de personas en movimiento fue lo suficientemente robusta, arrojando un número muy bajo de falsos positivos. Pero también observamos que el punto débil fue que en situaciones donde una cantidad considerable de personas se cruzaba o caminaba muy cerca entre sí, el sistema no lograba actualizar correctamente el historial de trayectorias de cada una de las personas involucradas, asumiendo que se trataba de una persona nueva en la escena o bien, asignándole un historial de trayectorias que no era la suya.

En resumen, este trabajo es un primer paso prometedor para el desarrollo de sistemas de detección y seguimiento de personas útiles para diversos fines como:

- Análisis del comportamiento de los peatones en la vía pública.
- Planificación de la señalización o salidas de emergencia en lugares altamente transitados por personas como shoppings o aeropuertos (a partir de conocer las trayectorias más comunes de las mismas, se puede decidir cuáles son los mejores lugares para ubicar señales y carteles informativos).
- Publicidad vial (si conozco cuales son los lugares más transitados por las personas, en una calle peatonal o en un lugar cerrado como un shopping o estación de subte, podría segmentar el precio de la publicidad o decidir cuál es la mejor ubicación para ubicar un stand comercial).

Por último, cabe destacar que todo lo investigado e implementado en el presente trabajo, podrá ser considerado, utilizado y/o adaptado tanto por investigadores como por docentes y alumnos para futuros proyectos en donde exista la necesidad de determinar la trayectoria de cualquier objeto en movimiento, tal como se hizo con el seguimiento de moscas en otro PFI de alumnos de la carrera de Licenciatura en Biotecnología (ver anexo).

8. BIBLIOGRAFIA

- Abe K., Suzuki S. 1985. Topological structural analysis of digitized binary images by border following. pp. 32-46, DOI: 10.1016/0734-189X(85)90016-7
- Benabbas Y., Djeraba C., Ihaddadene N. 2010. Motion Pattern Extraction and Event Detection for Automatic Visual. DOI: 10.1155/2011/163682
- Bernardin K., Stiefelhagen R. 2007. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics. DOI: 10.1155/2008/246309
- Bouwmans T. 2009. Subspace Learning for Background Modeling: A Survey. Recent Patent On Computer Science. 2 (3), pp. 223-234
- Bouwmans T., El Baf F., Vachon B. 2008. Background Modeling using Mixture of Gaussians for Foreground Detection - A Survey. Recent Patents on Computer Science, Bentham Science Publishers. 1 (3), pp. 219-237
- Bouwmans T., El Baf F., Vachon B. 2010. Statistical Background Modeling for Foreground Detection: A Survey. DOI: 10.1142/9789814273398_0008
- Bruno L., Negri P., Parisi D. 2016. Experimental characterization of collision avoidance in pedestrian dynamics. DOI: 10.1103/PhysRevE.00.002300
- Fukunaga K., Hostetler L. 1975. The estimation of the gradient of a density function, with applications in pattern recognition. DOI: 10.1109/TIT.1975.1055330
- Godbehere A., Matsukawa A., Goldberg K. 2012. Visual tracking of human visitors under variable-lighting conditions for a responsive audio art installation. DOI: 10.1109/ACC.2012.6315174
- Gonzalez R., Woods R. 1992. Digital Image Processing, Addison-Wesley Pub. ISBN: 9780201508031
- Harris C., Stephens M. 1988. A Combined Corner and Edge Detector. pp. 147-151
- Javed O., Shah M., Yilmaz A. 2006. Object tracking: A survey, ACM Computing Surveys. pp. 1-45
- Johansson A. [et al.]. 2008. From Crowd Dynamics to Crowd Safety: A Video-Based Analysis. DOI: 10.1142/S0219525908001854
- KaewTraKulPong P., Bowden R. 2002. An improved adaptive background mixture model for real-time tracking with shadow detection. DOI: 10.1007/978-1-4615-0913-4_11
- Milan A., Roth S., Schindler K. 2014. Continuous Energy Minimization for Multi-Target Tracking. pp. 58-72, DOI: 10.1109/TPAMI.2013.103

- Moya A. 2009. Estudio del filtro de partículas aplicado al seguimiento de objetos en secuencias de imágenes. Disponible en: https://core.ac.uk/download/pdf/29402235.pdf
- Nevatia R., Singh V., Wu B. 2008. Pedestrian Tracking by Associating Tracklets using Detection Residuals. DOI: 10.1109/WMVC.2008.4544058
- OpenCV. Feature Detection and Description, Harris Corner Detection. Disponible en: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_harris/py_features_harris.html#harris-corners
- OpenCV. Feature Detection and Description, Shi-Tomasi Corner Detector & Good Features to Track. Disponible en: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_shi_tomasi/py_shi_tomasi.html#shi-tomasi
- OpenCV. Feature Detection and Description, Understanding Features. Disponible en: http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html
- OpenCV. Introduction to OpenCV. Disponible en: http://docs.opencv.org/2.4/doc/tutorials/introduction/table_of_content_introduction/table_of_content_introduction.html#table-of-content-introduction
- Shi J., Tomasi C. 1994. Good Features to Track. DOI: 10.1109/CVPR.1994.323794
- Zivkovic Z. 2004. Improved Adaptive Gaussian Mixture Model for Background Subtraction. DOI: 10.1109/ICPR.2004.1333992
- Zivkovic Z. 2006. Efficient adaptive density estimation per image pixel for the task of background subtraction. pp. 773-780, DOI: 10.1016/j.patrec.2005.11.005

9. ANEXOS

9.1. Anexo A - Estudio del consumo preferencial de glutamato monosódico en Drosophilamelanogaster

La presente investigación sirvió además de soporte para el proyecto final de la carrera Licenciatura en Biotecnología de las alumnas Éttori, Sofía (LU 1027843) y Rodríguez, Carolina (LU 1025961) titulado "Estudio del consumo preferencial de glutamato monosódico en Drosophilamelanogaster".

La necesidad aquí consistió en la detección y seguimiento en tiempo real del movimiento de una mosca expuesta a distintos tratamientos: alimento estándar, que ofició de control negativo, alimento con glutamato monosódico (sal sódica del ácido glutámico, uno de los aminoácidos no esenciales más abundantes en la naturaleza) en tres concentraciones diferentes (0,5, 1 y 3%) y alimento con etanol al 5% como control positivo. Este se llevó a cabo en un tubo de vidrio (Figura 67) que permitió el desplazamiento del ejemplar mientras que fue filmado cenitalmente por una cámara conectada a una computadora que analizaba la información en vivo.

Figura 67: representación del tubo de vidrio con la mosca y los diferentes tipos de alimentos.

La figura 36 incluye los dos alimentos empleados posicionados en los extremos de un tubo de vidrio, el cual presenta un espécimen en su interior. Se detallan las distintas zonas identificadas (Estándar, Intermedio y Tratamiento).

El sistema se armó con un soporte de forma cubica; cinco tubos de vidrio con una mosca en cada uno para independizar los tratamientos; una fuente de luz ubicada en la parte inferior (lámpara circular LED); una base de acrílico para difuminar la luz y evitar que ésta interfiera en la filmación; y una cámara web (Logitech Webcam pro 9000) localizada en la parte superior (Figura 68).

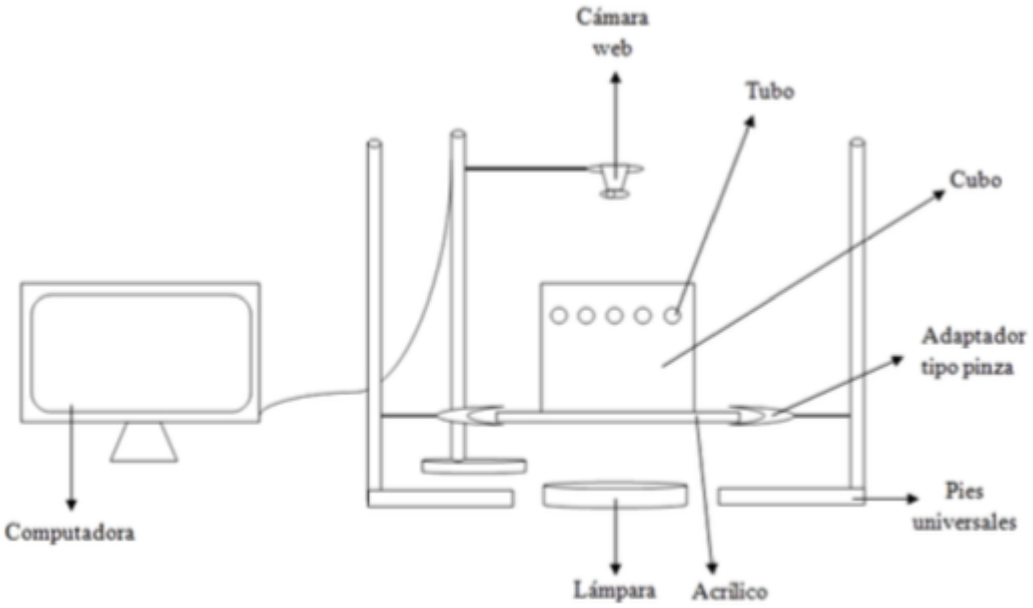


Figura 68: sistema montado para la detección del movimiento de cada mosca.

Los tubos fueron confeccionados a medida, respetando los 8 mm de diámetro y 150 mm de largo requeridos para usarse específicamente en el soporte mencionado. Tal como se indicó previamente, cada tubo poseía alimento estándar en uno de sus extremos y el tratamiento en el otro (Figura 69).

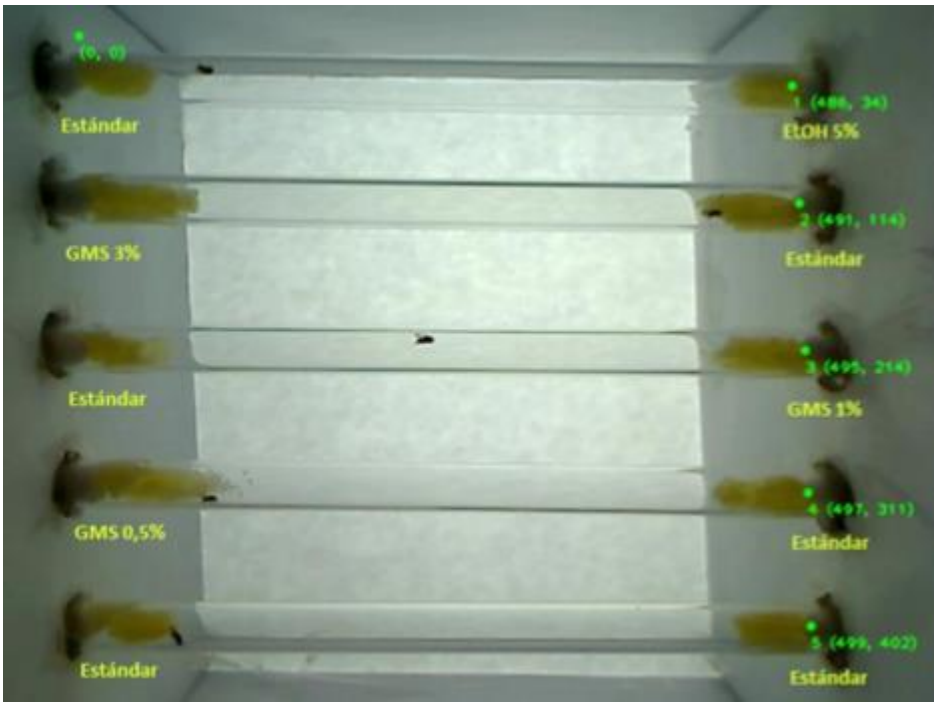


Figura 69: vista de los alimentos alternados a partir de la imagen tomada por el software.

Siendo (1) y (5) el control positivo y negativo respectivamente. A su vez, se alternó en cada experimento la posición de los tubos para asegurarse que dicha variable no afectara el tiempo de permanencia en cada alimento.

9.1.1. Detección y seguimiento

Para el monitoreo de moscas se implementó un programa escrito en Python, utilizando nuevamente la librería *"open source"* de visión computacional llamada OpenCV v2.4.9.

Al iniciarlo, se muestra el video capturado por la webcam y se solicita que el usuario señale la ubicación del origen de los ejes de coordenadas y de los alimentos. A partir de este momento, comienza el rastreo y seguimiento del organismo en estudio.

Sustracción de fondo

Esta es la primera operación que se realiza con cada *"frame"* obtenido de la webcam. Consiste en detectar objetos en movimiento a partir de una cámara fija. Esto es, lograr la distinción entre las zonas estáticas (llamadas fondo de la imagen o *"background"*) y las zonas dinámicas que se corresponden con el primer plano (*"foreground"*).

Para lograrlo esta diferenciación, se mantiene una representación de la escena mediante el modelado digital del fondo de la imagen. Esta representación contiene la información necesaria para determinar cualquier cambio significativo en una región de la imagen respecto del modelo y marcarlo entonces como primer plano.

```
#Background Subtractor initialization
self.bs = cv2.BackgroundSubtractorMOG(history=500, nmixtures=6,
backgroundRatio=0.1, backgroundRatio=1)

# …
fgmask = self.bs.apply(frame, learningRate=0.01)
```

Umbral del color negro de las moscas

Paralelamente, se realiza una sencilla operación para obtener los pixeles más oscuros del *"frame"*. Esto es, se establece un umbral y se filtran todos los pixeles por debajo de dicho valor.

```
#Initialization
UMBRAL_NEGRO = 25

# …
frame_gray = cv2.cvtColor(frame, code=cv2.COLOR_BGR2GRAY)
```

```
blackmask = cv2.compare(frame_gray, value=UMBRAL_NEGRO, cmpop=cv2.CMP_LE)
```

Intersección

A través de la intersección de los resultados de las dos operaciones descriptas anteriormente, podemos obtener entonces los pixeles oscuros que se encuentran en movimiento.

```
jointmask = cv2.bitwise_or(fgmask, blackmask)
```

Transformaciones Morfológicas

El resultado obtenido en el paso anterior podría llegar a contener un numero considerado de "falsos positivos" o lo que comúnmente se le llama "ruido". Esto es, pixeles blancos aislados que se detectaron como si estuviesen en movimiento pero que en realidad aparecen a causa de pequeñas vibraciones de la webcam o sombras proyectadas sobre la imagen.

Erosión

Para excluirlas se realiza la operación de "erosión", que como su nombre lo indica, reducirá las zonas detectadas, logrando que aquellas más pequeñas desaparezcan.

```
#Initialization
self.kerneldot = cv2.getStructuringElement(shape=cv2.MORPH_CROSS, anchor=(3, 3))

// …
jointmask = cv2.erode(jointmask, kernel=self.kerneldot, iterations=1)
```

Dilatación

Para restablecer las zonas que sobrevivieron a la "erosión" se aplica ahora la operación de "dilatación" con el objetivo de que recuperen su superficie original.

```
#Initialization
self.kernel = cv2.getStructuringElement(shape=cv2.MORPH_ELLIPSE, anchor=(5, 5))

# …
fgmask_with_morph = cv2.dilate(jointmask, kernel=self.kernel, iterations=1)
```

Detección de bordes

El objetivo en este paso es identificar las zonas que sobrevivieron a las operaciones de "erosión" y "dilatación" a través de la detección de contornos (Figura 70).

Los bordes o contornos pueden explicarse simplemente como una curva que une todos los puntos continuos, a lo largo de la frontera, que tienen el mismo color o intensidad.

Una vez que obtenemos los contornos, podremos obtener fácilmente información útil de los mismos, como por ejemplo, el centroide. Y es este valor el que consideraremos como la posición (x, y) de cada mosca detectada.

```
#Find contours
contours, hierarchy = cv2.findContours(fgmask_with_morph, mode=cv2.RETR_TREE,
method=cv2.CHAIN_APPROX_SIMPLE)

for contour in contours:
# Calculatecontour’scentroid
  moment = cv2.moments(contour)
  x, y = int(moment['m10']/moment['m00']), int(moment['m01']/moment['m00'])
```

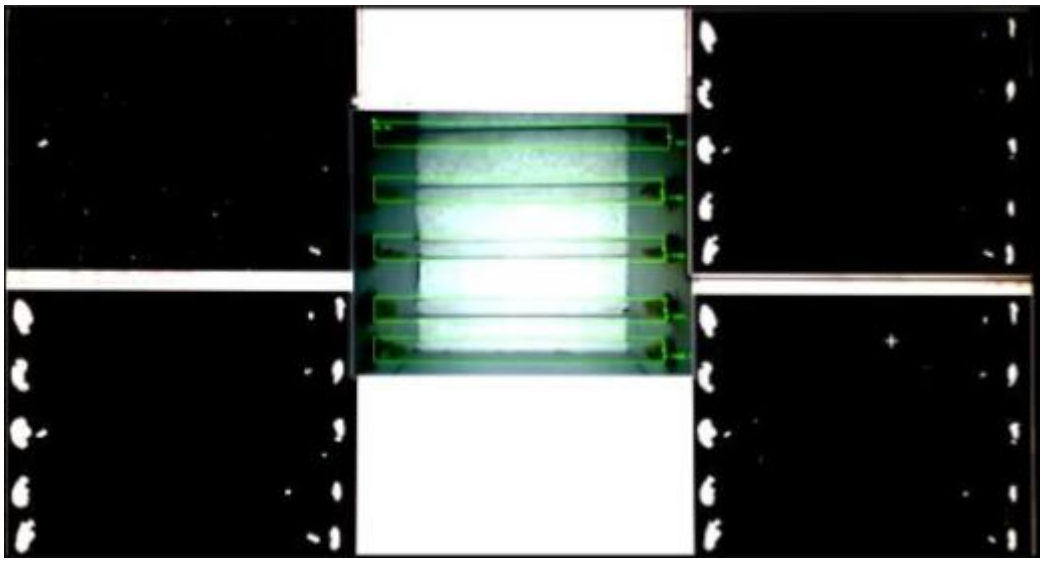


Figura 70: representación del funcionamiento de la sustracción de fondo.

Tracking

Una vez que ya detectamos todas las posiciones (x, y) de las moscas para un *“frame”* determinado el programa realiza las siguientes operaciones antes de pasar al análisis del siguiente *“frame”*:

- Se dibuja en la imagen un circulo indicando las posiciones (x, y) detectadas en ese instante.
- Se calcula la distancia euclidiana entre cada posición (x, y) y su correspondiente punto estático que fue señalado como la posición de la comida.

- Se escriben los resultados en un archivo CSV con los siguientes datos: número de *"frame"*, posición (x, y) de la mosca, posición (x, y) de la comida y la distancia euclidiana entre ambas posiciones.

9.1.2. Análisis de los resultados

El tiempo total de filmación fue de una hora, el cual se definió por evaluación estadística del comportamiento del software. Con los datos de tiempo y posición de la mosca en el eje X, se realizó un gráfico de dispersión (Figura 71), el cual permitió visualizar el trayecto recorrido por el ejemplar en el tiempo.

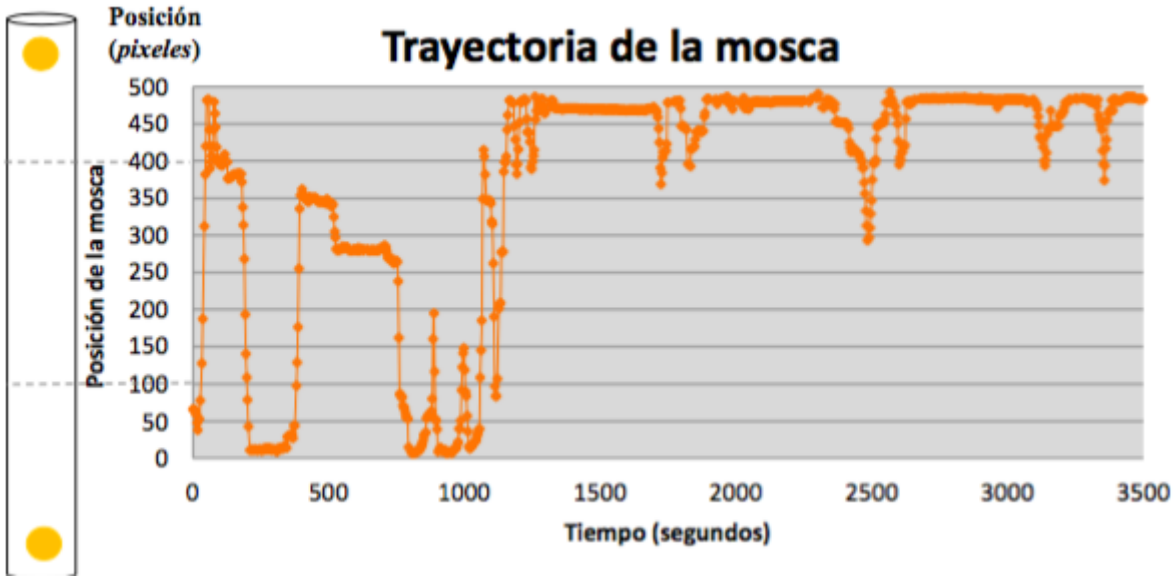


Figura 71: visualización grafica de la posición de la mosca en función del tiempo.

Se observa cómo, en un comienzo, la mosca se dirigió hacia el tratamiento, mientras que luego probó el alimento estándar para finalmente decidirse por el primero, exhibiendo un marcado consumo preferencial hacia el etanol.