FACULTAD DE MATEMÁTICA, ASTRONOMÍA Y FÍSICA

DEPARTAMENTO DE COMPUTACIÓN

Detección de Patrones Publicitarios

Tesis realizada por Illbele Maximiliano para la licenciatura en Ciencias de la Computación en la Universidad Nacional de Córdoba

---

Dirigida por:
Doctor Oscar Humberto Bustos

Agradecimientos

- A mi familia, mis padres Eduardo y Marcela quienes me forman como persona con su apoyo constante, sus consejos y la convivencia. A mis hermanos Alejandro, Patricia, Fernando, Marina y Paula gracias por lo vivido!
- A mi director Dr. Bustos Humberto, principalmente por su calidez como persona, quién siempre saluda con la misma sonrisa a conocidos o extraños y por guiarme desde el comienzo en mi tesis, inclusive cuando yo nunca antes había trabajado con imágenes ni videos.
- A la Dra. Brandán Laura con quien tuve interesantes charlas de la vida y recibí consejos técnicos para desarrollar los documentos formales relacionados al testing.
- A mis compañeros: Agustín, Alan, Andrés, Daniel, Emanuel, Emiliano, Federico, Fernando, Franco, Gonzalo, Kevin, Mauro, Milagro y Rodrigo, por acompañarme a lo largo de estos grandiosos 5 años.
- Al Lic. Lis Diego, docente de la facultad y CTO de Infoxel, quién colaboró con parte de la validación y puesta en práctica de esta tesis.

Resumen

El objetivo principal de este trabajo fue estudiar distintos algoritmos de reconocimiento de patrones en imágenes para la detección de logos publicitarios. A lo largo de este trabajo describo dos implementaciones en este ámbito; la primera permite detectar automáticamente los delimitadores de los canales de televisión espacios publicitarios mediante template matching y la segunda permite detectar logos en eventos deportivos mediante Speeded Up Robust Features (S.U.R.F.). Ambas implementaciones fueron en python y utilizan especialmente las librerías: OpenCV, numpy y mock.

Abstract

The aim of this work was to study different pattern recognition algorithms in images to detect advertising brands. Along this work I describe two implementations in this area; the first can automatically detect the delimetes from the TV channels indicating ad space through template matching and the second can detect advertising brands at sporting events through Speeded Up Robust Features (S.U.R.F.). Both implementations are in Python and use especially the libraries: OpenCV, numpy and mock.

Key words: **Patter Recognition, Image detection and registration, OpenCv**.

Introducción

El marketing en las empresas ha llevado durante años a intentar establecerse en el mercado mediante diversas estrategias, la más común es la identificación de sus valores mediante un isotipo[1]. Estas imágenes intentan reflejar los valores de la empresa y sus objetivos.

Es por esto que el esponsoreo y la publicidad en un medio tradicional como la televisión, es usualmente su estrategia para captar potenciales clientes o intentar establecerse en el mercado.

Motivado por esto intenté desarrollar una aplicación que permitiera detectar estos logos para reconocer a dichas empresas.

La idea original es que se puede tomar imágenes representativas de estas marcas, para distinguirlas, reduciendo el problema a implementar un algoritmo para detectar dichas imágenes dentro de un video.

El reconocimiento de patrones permite diversas aplicaciones, no sólo permite distinguir las marcas publicitarias sino también detectar delimitadores de espacios publicitarios, empleados en los medios audiovisuales de Argentina a partir de la sanción de la ley de medios, en 2010. Este trabajo muestra precisamente dos aplicaciones que tienen en su núcleo algoritmos de detección de patrones y

[1]Isotipo: se refiere a la parte, generalmente gráfica de la disposición espacial en diseño de una marca, ya sea corporativa, institucional o personal, comúnmente llamada logo.

Motivación

La publicidad en las camisetas evolucionó, desde un acuerdo comercial que realizó la por entonces estatal Austral con Argentinos Juniors para retener a Diego Maradona en 1979, hacia una estrategia de alianza entre clubes y marcas [Ieco].

Los clubes y las empresas coinciden en que los patrocinios en las camisetas son más abarcativos de lo que parecen. Un claro ejemplo de esto es la marca Budweiser que fue main sponsor del River campeón del Apertura 2004 y esto determinó que los indicadores mejoraron muchísimo ese año. Del mismo modo, "La imagen de Petrobas quedó vinculada a la peor época en la historia de River".

Celia Mosto, directoria de CIO, declara que: "el patrocinio establece una relación afectiva con los aficionados y la forma en que se construye ese vínculo es central para esa marca". Es importante para fidelizar clientes la calidad del club con el cual vinculan su imagen.

Todo para que la inversión no quede como un hecho publicitario que expira en el instante en que desaparece la camiseta.

En el cuadro (1) veremos que en el último año las empresas invirtieron más de 15 millones de dólares para ser sponsors de los clubes con más convocatoria de la Argentina.

Equipos	**Sponsors**	**Monto anual en dólares**
River	BBVA Francés y Netshoes	6.900.000
Boca	BBVA Francés y Citröen	6.460.000
Racing	Banco Hipotecario	1.600.000
Independiente	Oca y Banco Ciudad	1.600.000
San Lorenzo	Banco Ciudad	1.000.000

Cuadro 1: Inversión en el año 2014

Por estos motivos, ligados a mi pasión por el deporte, me pareció desafiante desarrollar una aplicación que detecte estas marcas en los distintos formatos que pueden aparecer en un evento deportivo y cuantifique la contraparte ligada a una inversión como ser la cantidad de segundos, tamaños en los que aparece la marca, lugares que ocupa en la pantalla y el segundo de cada aparición.

Estos datos pueden ser de utilidad a un analista de marketing para decidir estrategias de mercado, indicar lugares donde conviene agregar sus marcas, medir si las apariciónes de su marca en el evento fueron en momentos convenientes o no, o simplemente medir cuántos segundos por temporada se verá su marca si invierte en determinado club.

Reconocimiento de Patrones

El reconocimiento de patrones es la ciencia que se ocupa de los procesos sobre ingeniería, computación y matemáticas relacionados con objetos físicos o abstractos, con el propósito de extraer información que permita establecer propiedades de entre conjuntos de dichos objetos.

Los patrones se obtienen a partir de los procesos de segmentación, extracción de características y descripción, donde cada objeto queda representado por una colección de descriptores[Pattern Classification]. El sistema de reconocimiento debe asignar a cada objeto su categoría o clase (conjunto de entidades que comparten alguna característica que las diferencia del resto). Para poder reconocer los patrones se siguen los siguientes procesos, mostrados en la figura (1):

1. Adquisición de datos
2. Extracción de características
3. Toma de decisiones

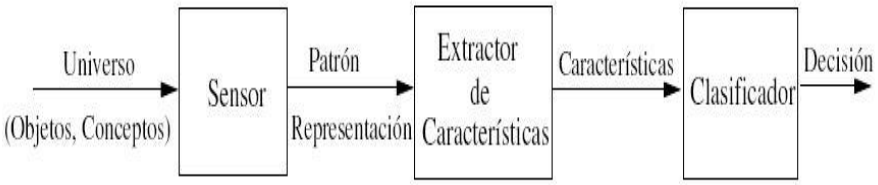


Figura 1: Procesos para el reconocimiento de patrones

Para el problema en cuestión, la adquisición de datos va a realizarse mediante una grabación de una cámara de televisión, es decir que buscaremos patrones en uno o más videos que cubren un evento deportivo.

Se estudiarán en el presente trabajo distintas maneras de extraer características de las imágenes que forman un video según la finalidad, precisión y tiempo de procesamiento que demande cada objetivo.

Por último se realizará una medición mediante el estadístico Precision-Recall para validar el resultado presentado[F-Factor].

Índice general

Teoría de las imágenes

La mayoría de las imágenes digitales representan imágenes continuas de la realidad, a excepción de las imágenes artificiales.

Imágenes continuas y representación matemática

Existen dos maneras de representar imágenes:

- Representación determinística de las imágenes: se define una función matemática y se consideran las propiedades de los puntos.
- Representación estática de imágenes, la imágen es especificada como un promedio de propiedades.

1.1. Template matching

Una técnica utilizada para reconocer patrones dentro de una imagen es el **template matching**.

Sea **A** una imagen (de tamaño W*H): W es el ancho (Width) y H es la altura (Height) y sea **P** un patrón (de tamaño wxh), el resultado del template matching es una imagen **M** (de tamaño (W-w+1) * (H-h+1)), donde cada pixel de M(x,y) indica la "verosimilitud" (probabilidad) de que el rectángulo [x,y] - [x+w-1, y+h-1] de A contenga el patrón **P**.

Para generar esta matriz M tenemos que definir una distancia entre dos imágenes, estas posibilidades serán analizadas con profunidad en la sección 1.1.2.

A manera de adelanto de la metodología **template matching** puede observarse un ejemplo en la imagen 1.2 y su valor de verosimilitud para cada pixel.

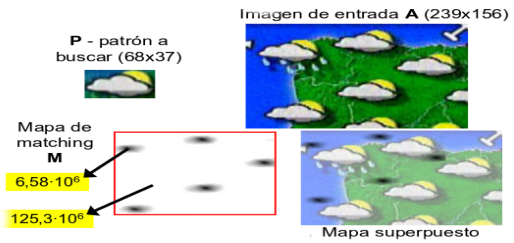


Figura 1.2: Template Matching a partir del patrón **P** y la imagen **A**.

El template matching, raramente es exacto por el ruido generado por el contraste o las diferencias de luz entre el patrón y la imagen.

1.1.1. Opencv

¿Qué es OpenCV?

OpenCV[OpenCV] es una librería open source de computer vision disponible en `http://SourceForge.net/projects/opencvlibrary`. La librería está escrita en C y C++ y corre bajo Linux, Windows and Mac OS X. Hay un desarrollo activo en las interfaces para Python, Ruby, Matlab, y otros lenguajes.

OpenCV fue diseñada para el cómputo eficiente con un foco fuerte en las aplicaciones de ejecución de tiempo real ya que tiene la ventaja de corre en múltiples procesadores.

Uno de los objetivos de OpenCV es el de proveer una infraestructura simple en vision-computer que ayude a los desarrolladores a construir limpiamente aplicaciones sofisticadas rápidamente.

Implementación de Opencv.

La libería Opencv, implementa el template matching mediante la función cvMatchTemplate(), no está basado en histogramas; más bien, la función matchea el ajuste entre una imagen y una entrada mediante el corrimiento de la imagen origen al template empleando alguno de los métodos descriptos a continuación.

En la figura 1.3 puede verse como se va desplaza la ventana y se va computando para cada vértice, superponiendo el patrón sobre la imagen, la distancia entre ambas imágenes.

Figura 1.3: Se Busca el patrón en la imagen comparando una región del mismo tamaño del patrón recorriendo la totalidad de la imagen.

1.1.2. Métodos De Matching

Notación[LearnOpenCV]:

- I: Imagen.
- T: Template o Patrón.
- R: Resultado.
- w: Ancho en píxeles de la imagen del patrón.
- h: el alto en píxeles de la imagen del patrón.

Método de diferencia de Cuadrados

Este método verifica la diferencia de los cuadrados, es decir que match será perfecto cuando tienda a 0 y malo cuando sea mucho mayor.

$$R_{sq\_diff}(x,y) = \sum_{x',y'}[T(x',y') - I(x+x',y+y')]^2$$

Matching de correlación

Este método multiplicativo para matchear imágenes, entonces un match será mejor a medida que el número sea mayor y será minimo cuando tienda a 0.

$$R_{ccorr}(x,y) = \sum_{x',y'}[T(x',y') * I(x+x',y+y')]^2$$

Coeficiente de correlación de Match

Este método para matching, relaciona una imagen para si misma, entonces devuelve un $x \in [-1,1]$: -1 significa que son totalmente distintas las imágenes, 0 que ambas imágenes no están relacionadas, es decir que los puntos están alineados azarosamente y 1 para indicar un matching perfecto.

$$R_{sq\_diff}(x,y) = \sum_{x',y'}[T'(x',y') * I'(x+x',y+y')]^2$$

$$T'(x',y') = T(x',y') - \frac{1}{(w*h)\sum_{x'',y''} T(x'',y'')}$$

$$I'(x+x',y+y') = I(x+x',y+y') - \frac{1}{(w*h)\sum_{x'',y''} T(x'',y'')}$$

1.1.3. Métodos normalizados

A la hora de trabajar con normas, y distancias, hay métodos que normalizan estas distancias de manera de reducir, ciertos errores que pueden de venir de efectos de luz.

En los 3 casos el método de normalización es el mismo:

$$Z(x,y) = \sqrt{\sum_{x',y'} T(x',y')^2 * \sum_{x',y'} T(x+x',y+y')^2}$$

1.1.4. Análisis preliminar

Este es uno de los métodos más simples para la detección de patrones en imágenes, la principal **desventaja** es que sólo busca match del mismo tamaño que el patrón que estamos buscando, por lo que no es útil para detectar cambios de tamaños.

1.2. SIFT: Scale invariant Features Transform.

SIFT significa detección invariante de puntos característicos y descriptores de la imagen, es uno de los métodos de detección de puntos invariantes ampliamente utilizados en la literatura[PAPER SIFT].

Este detector transforma los datos de la imagen en descriptores invariantes a la traslación, escala, rotación y en cierta medida al cambio de iluminación. Cada uno de los puntos extraídos se considera una característica de la imagen y se describe mediante su posición, escala, orientación y su vector descriptivo, el cual contiene 128 posibles atributos.

Como el nombre lo sugiere, **Scale invariant**, es debido a que sift detecta la orientación del gradiente dominante y su localización, recordando su histograma local y registrándolo.

A partir de las características [LearnOpenCV,321] locales, se busca conseguir invariancia a la escala, orientación, parcialmente a cambios de iluminación, etc. También se puede utilizar para buscar correspondencias entre diferentes puntos de vista de una misma escena. Estas características locales se almacenan en los denominados descriptores.

Debido a SIFT almacena la orientación dominante lo hace a su vez invariante **con respecto a rotación**.

1.2.1. Algoritmo

Consiste en cuatro pasos desarrollados a continuación [Aracil]:

1. **Detección de máximos y mínimos espacio-escala**: este paso consiste en la búsqueda de puntos en la imagen que puedan ser **keypoints**[2]. Se realiza usando diferencias de funciones gaussianas para hallar puntos interesantes que sean invariantes a la escala y a la orientación.

[2]Puntos Claves

2. **Localización de los keypoints**: de los puntos obtenidos en el paso anterior se determinan la ubicación y la escala de los mismos, de los cuales se seleccionan los keypoints basándose en la medida de la estabilidad de los mismos.

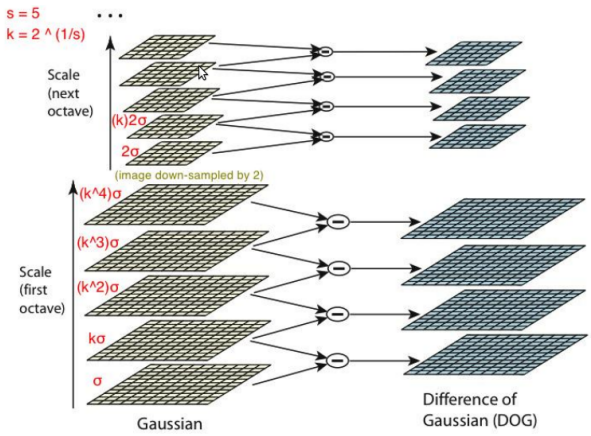
3. **Asignación de la orientación**: a cada ubicación del keypoint se le asigna una o más orientaciones, basado en las orientaciones de los gradientes locales de la imagen.

4. **Descriptores de los keypoints**: Los gradientes locales se miden y se transforman en una representación que permite importantes niveles de la distorsión de la forma local y el cambio en la iluminación.

Detección de máximos y mínimos espacio-escala

Esta es la etapa donde los puntos de interés, que se llaman puntos clave (keypoint) en el marco de SIFT, se detectan. Para ello, la imagen se procesa con filtros gaussianos a diferentes escalas, y luego se calcula la diferencia de los sucesivos puntos Gaussianos que hemos encontrado en la imagen. Los máximos y mínimos de esta función proporcionan las características más estables, con lo que esto serán nuestros keypoints.

En concreto, un DOG de la imagen $D(x, y, \sigma)$ viene dada por $D(x, y, \sigma) = L(x, y, k_i\sigma) - L(x, y, k_j\sigma)$, donde $L(x, y, k\sigma)$ a escala $k\sigma$, es decir: $L(x, y, k\sigma) = G(x, y, k\sigma) * I(x, y)$



Localización de los keypoints

La detección de espacio-escala da como resultado demasiados candidatos a Keypoints, algunos de los cuales son inestables. El siguiente paso en el algoritmo es realizar un ajuste detallado de los datos más cercanos para la localización exacta, la escala y proporción de curvaturas principales. Esta información permite poder rechazar puntos que tienen bajo contraste (y por tanto son sensibles al ruido) o están mal localizados.

Lo primero que haremos será, para cada keypoint que obtenemos, haremos una interpolación de los datos para determinar con precisión la posición de cada uno, con lo que mejoramos la estabilidad de nuestro método de obtención. Todo esto se realiza mediante la expresión de Taylor de la ecuación de diferencias gaussianas $D(x, y, \sigma)$. Esta nueva

expresión vendría dada por:

$$D(p) = D + \frac{\partial D^T}{\partial p}p + \frac{1}{2}p^T\frac{\partial^2 D}{\partial p^2}p$$

Sabiendo que p es el keypoint obtenido del apartado anterior, derivamos la expresión y nos queda de la siguiente manera:

$$\hat{p} = -\Big(\frac{\partial^2 D^{-1}}{\partial p^2} * \frac{\partial D}{\partial p}\Big) = -\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial x \partial y} \\ \\ \frac{\partial^2 D}{\partial y \partial x} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}^{-1} \begin{bmatrix} \frac{\partial D}{\partial x} \\ \\ \frac{\partial D}{\partial y} \end{bmatrix}$$

A partir de la expresión anterior, podemos sacar una expresión que nos sirve para eliminar los puntos que tienen muy bajo contraste. Sustituyendo la ecuación anterior podemos sacar otra que nos sirve para calcular el contraste del punto para poder así eliminar los que no nos sirven.

La ecuación sería la siguiente:

$$D(\hat{p}) = D + \frac{1}{2}\frac{\partial D^T}{\partial p}\hat{p}$$

A partir de esta ecuación, tenemos que establecer un umbral mínimo al que deben de llegar los keypoints para no ser rechazados. Entonces todos aquellos que no cumplan la condición $|D(\hat{p})| > 0{,}03$ serán eliminados.

Ahora también se deben descartar aquellos puntos que posean una pobre localización a lo largo de un borde, ya que la función diferencia de gaussianas posee una alta respuesta a lo largo de los bordes.

Un pico pobremente definido en la función diferencia de gaussianas indica que habrá una larga curvatura principal en la dirección del borde, pero pequeña en la dirección perpendicular del mismo. La curvatura principal se puede procesar a partir de una matriz hessiana de 2 x 2, evaluada en el keypoint.

$$H = \begin{bmatrix} D_{xx} & D_{yx} \\ D_{xy} & D_{yy} \end{bmatrix}$$

Las derivadas necesarias para obtener la matriz H se han calculado tomando diferencias con los vecinos del punto de muestreo. Los autovalores de la matriz H son proporcionales a las curvaturas principales de D. Se toma α como el mayor de los auto valores y β como el menor.

Entonces, se puede obtener la suma de los autovalores a partir de la de H y su producto del determinante:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta$$

$$Det(H) = D_{xx}D_{yy} - (D_{xy})^2 = \alpha\beta$$

Si las curvaturas tienen signos diferentes, entonces dicho punto se debe descartar, ya que no está representando a un máximo o a un mínimo.

Entonces, si se toma r como la relación que existe entre los dos autovalores ($\alpha = r\beta$), se obtiene:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{\alpha\beta} = \frac{(r\beta+\beta)^2}{r\beta^2} = \frac{(r+1)^2}{r}$$

Sólo depende de la relación que existe entre los autovalores. El $\frac{(r+1)^2}{r}$ es mínimo cuando los dos auto-valores son idénticos y aumenta conforme va creciendo r.

Entonces para verificar la relación entre las curvaturas sólo es necesario calcular la relación existente entre el cuadrado de la traza de H y su determinante.

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r}$$

Un valor umbral bastante razonable, sería tomar $r = 10$. Así, de este modo, descartamos los puntos inestables por una pobre localización en un borde.

Asignación de la orientación

La asignación de una orientación a los keypoints es muy importante, ya que si se consigue una orientación coherente basada en las propiedades locales de la imagen, el descriptor puede ser representado en relación de dicha orientación y por lo tanto ser invariante a la rotación. Este enfoque contrasta con la de otros descriptores invariantes a la orientación, que buscan propiedades de las imágenes basadas en medidas invariantes a la rotación.

La desventaja de este enfoque es que limita el número de descriptores que se pueden usar y rechaza mucha información de la imagen.

Para establecer una orientación adecuada, este método se basa en el gradiente local de la imagen alrededor de los keypoints. Para ello se utilizará la imagen suavizada por la gaussiana a la mayor escala determinada por el keypoint, de modo que todos los cálculos se realizan de un modo invariable a la escala.

Por cada imagen de muestreo, $L(x, y)$, la magnitud del gradiente, $m(x, y)$, y la orientación, $\theta(x, y)$, se precalculan usando diferencias de píxeles:

$$m(x,y) = \sqrt{\Big(L(x+1,y) - L(x-1,y)\Big)^2 - \Big(L(x,y+1) - L(x,y-1)\Big)^2}$$

$$\theta(x,y) = \tan^{-1}\Big(\frac{L(x,y+1) - L(x,y-1)}{L(x+1,y) - L(x-1,y)}\Big)$$

Un histograma de orientación se forma a partir de las orientaciones de los gradientes de los puntos de muestreo que se encuentran dentro de la región que rodea al keypoint. El histograma de orientación posee 36 divisiones que abarcan los 360 o del rango de orientaciones.

Cada muestra añadida al histograma es pesada por su magnitud del gradiente y por una máscara gaussiana circular con un valor de $\sigma = 1{,}5$ veces el valor que posea el keypoint.

Los picos en el histograma de orientaciones corresponden a las direcciones dominantes de los gradientes locales. Se detecta el mayor pico del histograma, y entonces, si existen

otros máximos superiores al 80 % del pico principal, éstos se utilizarán para crear otros keypoints con nuevas orientaciones.

Por lo tanto, existirán varios keypoints con la misma localización, pero con diferentes orientaciones. Solamente el 15 % de los keypoints disponen de orientaciones múltiples, pero estos puntos contribuyen significativamente en la estabilidad del método. Por último, para determinar con mayor exactitud la localización del pico en el histograma, este se interpola con una parábola que es fijada por los tres puntos más cercanos a cada pico.

Descriptores de los keypoints

Una vez que ya se tienen todos los keypoints de la imagen, se tiene que segmentar la vecindad del keypoint en 4 x 4 regiones de 4 x 4 píxeles. Una vez que ya se ha dividido la vecindad del keypoint, se genera un histograma de orientación de gradiente para cada región. Para ello, se utiliza una ponderación gaussiana con un ancho $\sigma = 4$ píxeles.

Pero dicha construcción presenta un gran problema, ya que en el caso de un pequeño desplazamiento espacial, la contribución de un pixel puede pasar de una casilla a otra, lo que provoca cambios repentinos del descriptor. Este desplazamiento también puede deberse al hecho de una pequeña rotación.

Para evitar estos problemas, todos los píxeles contribuyen a todos los vecinos, tanto en magnitud como en orientación. Esta contribución se multiplica por un peso $1 - d$, donde d es la distancia al centro de la casilla.

Esta distancia está normalizada al tamaño de una región, es decir, la longitud de una región de 4 píxeles tiene una distancia $d = 1$.

Además, el valor mínimo del peso $1 - d$ es de 0, ya que los pesos no pueden ser negativos.

Como los histogramas de orientación de cada región están divididos en 8 barras, por cada vecindad del keypoint se puede construir un histograma tridimensional de 4 x 4 x 8 valores, formando así un vector con todos estos valores.

Algoritmo Comparación del vecino más próximo (Nearest Neightbour)

Una vez que ya se tienen calculados los descriptores, ya sabemos que son un conjunto de 128 elementos que nos indican la orientación alrededor de un punto de interés, así que cuando encontremos características similares en diferentes puntos, esto quiere decir que están refiriéndose a la misma zona.

Donde dif_i es la diferencia euclídea entre el elemento a_i de un descriptor de la imagen A, y b_i su correspondiente de un descriptor de la imagen B. La variable dif_{total} es la suma de las diferencias euclídeas que hay entre los 128 elementos de ambos descriptores.

$$dif_i = \sqrt{(a_i - b_i)^2}$$

$$dif_{total} = \sum_{i=1}^{128} dif_i$$

Realizando esta serie de operaciones entre cada descriptor de la imagen A con cada uno de la imagen B, podremos decidir cuales son iguales, eligiendo siempre el que haya dado una diferencia euclídea total menor.

1.3. SURF: Speeded-Up Robust Features.

1.3.1. Historia

SURF (speed up robust feature) es un detector de características locales, y fue presentado por primera vez por Herbet Bay en el 2006 y se inspira en el descriptor SIFT, pero presentando ciertas mejoras, como:

- Velocidad de cálculo superior sin ocasionar pérdida del rendimiento.
- Mayor robustez ante posibles transformaciones de la imagen.

Estas mejoras se consiguen mediante la reducción de la dimensionalidad y complejidad en el cálculo de los vectores de características de los puntos de interés obtenidos, mientras continúan siendo suficientemente característicos e igualmente repetitivos.

"La tarea de encontrar puntos correspondientes entre dos imágenes de la misma escena o partes de objetos forma parte de muchar aplicaciones de **computer vision** "[Paper Surf]
.

La búsqueda de puntos discretos entre imágenes puede dividirse en tres principales etapas:

1. **Puntos de interés**: se eligen de ubicaciones distintivas de la imagen como ser esquinas, manchas e intersecciones.

   La propiedad más valiosa de un detector de puntos de interés es su repetitibilidad.

   La repetibilidad expresa la confianza de un detector para encontrar los mismos

   puntos fisicos de interés desde distintos puntos de vista de la imagen.

2. **Generación de descriptores**: a continuación, los vecinos de cada punto de interés son representadas por un vector. Este descriptor tiene que ser característico y al mismo tiempo robusto al ruido, desplazamientos, características geométricas y deformaciones fotométricas.

3. **Matching de Descriptores**: finalmente, los descriptores son matcheados entre las diferentes imágenes. El matching está basado en la distancia entre los vectores por ejemplo la Mahalanobis o la distancia Euclídea.

La dimensión de los descriptores tiene un impacto directo de el tiempo que toman, mientras más pequeñas sean las dimensiones son deseables para un rápido matching de puntos de interés.

Sin embargo, pequeñas diemensiones son en general menos distintivas que los de grandes dimensiones en distintas zonas.

En general, uno tiene que tomar un balance entre los requerimientos: simplificar la detección manteniendo su precisión y reducir el tamaño de los descriptores manteniéndolo lo suficientemente distintivo.

1.3.2. Descripción

SURF es otro de los algoritmos más utilizado para la extracción de puntos de interés en el reconocimiento de imágenes. La extracción de los puntos la realiza detectando en primer lugar los posibles puntos de interés y su localización dentro de la imagen.

Es mucho más rápido que el método SIFT, ya que los keypoints contienen muchos menos descriptores debido a que la mayor cantidad de los descriptores son 0. Este descriptor se puede considerar una mejora debido a que la modificaciones que supondría en el código no serían excesivas, ya que el descriptor SURF utiliza la gran mayoría de las funciones que utiliza el descriptor SIFT.

1.3.3. Algoritmo

A continuación se presenta una explicación detallada del algoritmo SURF, explicando fundamentalmente cómo detecta puntos de interés (keypoints), cómo dedice la asignación de la orientación y por último obtención del descriptor SURF.

Detección de puntos de interés

La primera de las etapas del descriptor SURF es idéntica a la del descriptor SIFT en cuanto a la detección de puntos de interés se refiere.

El descriptor SURF hace uso de la matriz Hessiana, más concretamente, del valor del determinante de la matriz, para la localización y la escala de los puntos. El motivo para la utilización de la matriz Hessiana es respaldado por su rendimiento en cuanto a la velocidad de cálculo y a la precisión.

Lo realmente novedoso del detector incluido en el descriptor SURF respecto de otros detectores es que no utiliza diferentes medidas para el cálculo de la posición y la escala de los puntos de interés individualmente, sino que utiliza el valor del determinante de la matriz Hessiana en ambos casos. Por lo tanto dado un punto $p = (x, y)$ de la imagen, la matriz Hessiana $H(p, \sigma)$ del punto p perteneciente a la escala σ se define como:

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{xy}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix}$$

Ganancia de Velocidad mediante la Matriz Integral

$$X = (x, y)^T$$

$$I_\Sigma(X) = \sum_{i=0}^{i \leq x} \sum_{j=0}^{j \leq J} I(i,j)$$

Representa la suma de todos los pixeles de la imagen de entrada I, con una región rectangular formada por el origen y X.

Una vez calculada la matriz integral, calcular la intensidad de un área sólo requiere la suma de 4 términos y 4 accesos a memoria como se ve en la figura 1.4.

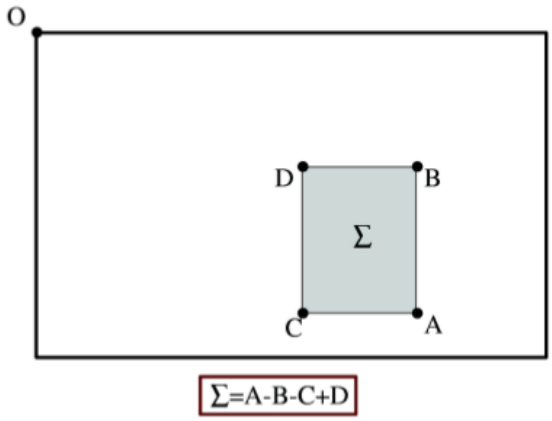


Figura 1.4: Matriz Integral

Donde $L_{xx}(X, \sigma)$ es el producto de segundo orden de la Gaussiana, $\frac{\partial^2}{\partial x^2} g(\sigma)$ con la imagen I en el punto (x, y) similarmente para $L_{xy}(X, \sigma)$ y $L_{yy}(X, \sigma)$.

Las aproximaciones de las derivadas parciales se denotan como $D_{xx}, D_{xy} y D_{yy}$, y el determinante se calcula de la siguiente manera:

$$Det(H_{approx}) = D_{xx}D_{yy} - (0,9D_{xy})^2$$

Donde el valor de $0,9$ está relacionado con la aproximación del filtro Gaussiano.

En la siguiente imagen se puede observar la representación de la derivada parcial de segundo orden de un filtro gaussiano discretizado y la aproximación de la derivada implementada en el caso del descriptor SURF.

La imagen de salida obtenida tras el producto de la imagen original con un filtro de dimensiones 9 x 9, que corresponde a la derivada parcial de segundo orden de una gaussiana con $\sigma = 1,2$, es considerada como la escala inicial o también como la máxima resolución espacial ($s = 1,2$, correspondiente a una gaussiana con $\sigma = 1,2$). Las capas sucesivas se obtienen mediante la aplicación gradual de filtros de mayores dimensiones, evitando así los efectos de aliasing en la imagen. El espacio escala para el descriptor SURF, al igual que en el caso del descriptor SIFT, está divido en octavas. Sin embargo, en el descriptor SURF, las octavas están compuestas por un número fijo de imágenes como resultado del producto de la misma imagen original con una serie de filtros cada vez más grande. El incremento o paso de los filtros dentro de una misma octava es el doble respecto del paso de la octava anterior, al mismo tiempo que el primero de los filtros de cada octava es el segundo de la octava predecesora.

Finalmente para calcular la localización de todos los puntos de interés en todas las escalas, se procede mediante la eliminación de los puntos que no cumplan la condición de máximo en un vecindario de 3 x 3 x 3. De esta manera, el máximo determinante de la matriz Hessiana es interpolado en la escala y posición de la imagen.

Asignación de la Orientación

La siguiente etapa en la creación del descriptor corresponde a la asignación de la orientación de cada uno de los puntos de interés obtenidos en la etapa anterior.

Es en esta etapa donde se otorga al descriptor de cada punto la invariancia ante la rotación mediante la orientación del mismo.

El primer paso para otorgar la mencionada orientación consiste en el cálculo de la respuesta de Haar, figura 1.5, en ambas direcciones x e y mediante las funciones siguientes:

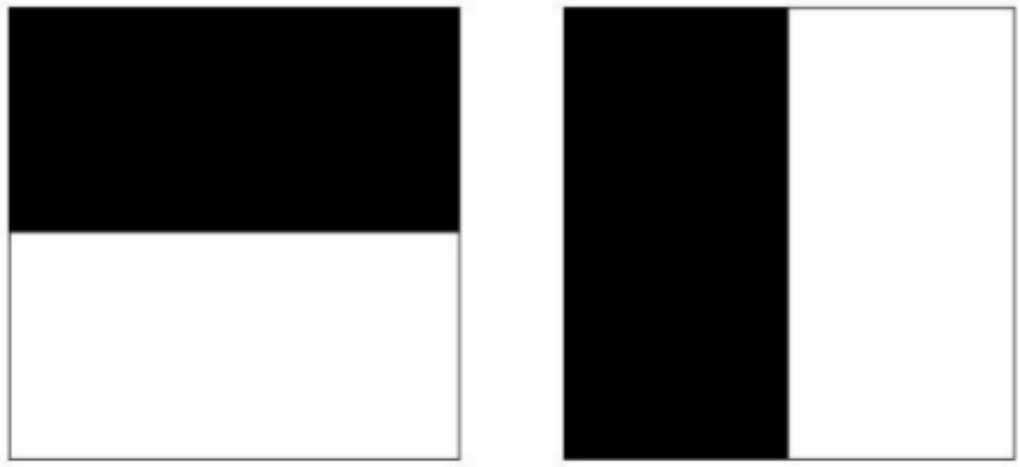


Figura 1.5: Respuesta de Haar

Donde el color negro es el valor -1 y el blanco es $+1$.

La etapa de muestreo depende de la escala y se toma como valor s. Se toma el valor 4s, siendo s la escala en la que el punto de interés ha sido detectado, por tanto dependiente también de la escala, como referencia, donde a mayor valor de escala mayor es la dimensión de las funciones onduladas.

Tras haber realizado todos estos cálculos, se utilizan imágenes integrales nuevamente para proceder al filtrado mediante las máscaras de Haar y obtener así las respuestas en ambas direcciones. Son necesarias únicamente 6 operaciones para obtener la respuesta en la dirección x e y. Una vez que las respuestas onduladas han sido calculadas, son ponderadas por una gaussiana de valor $\sigma = 2,5s$ centrada en el punto de interés. Las respuestas son representadas como vectores en el espacio colocando la respuesta horizontal y vertical en el eje de abscisas y ordenadas respectivamente. Finalmente, se obtiene una orientación dominante por cada sector mediante la suma de todas las respuestas dentro de una ventana de orientación móvil cubriendo un ángulo de $\frac{\pi}{3}$.

Descriptores SURF

Se construye como primer paso una región cuadrada de tamaño $20s$ alrededor del punto de interés y orientada en relación a la orientación calculada en la etapa anterior. Esta región es a su vez dividida en 4 x 4 sub-regiones dentro de cada una de las cuales se calculan las respuestas de Haar de puntos con una separación de muestreo de 5 x 5 en ambas direcciones. Por simplicidad, se consideran dx y dy las respuestas de Haar en las direcciones horizontal y vertical respectivamente relativas a la orientación del punto de interés.

Para dotar a las respuestas dx y dy de una mayor robustez ante deformaciones geométricas y errores de posición, éstas son ponderadas por una gaussiana de valor $\sigma = 3{,}3s$ centrada en el punto de interés. En cada una de las sub-regiones se suman las respuestas dx y dy obteniendo así un valor de dx y dy representativo por cada una de las sub-regiones. Al mismo tiempo se realiza la suma de los valores absolutos de las respuestas dx y dy en cada una de las sub-regiones, obteniendo de esta manera, información de la polaridad sobre los cambios de intensidad.

En resumen, cada una de las sub-regiones queda representada por un vector v de componentes:

$$\nu = \left(\sum dx, \sum d_y, \sum |d_x|, \sum |d_y|\right)$$

y por lo tanto, englobando las 4 x 4 sub-regiones, resulta un descriptor SURF, figura 1.6, con una longitud de 64 valores para cada uno de los puntos de interés identificados.

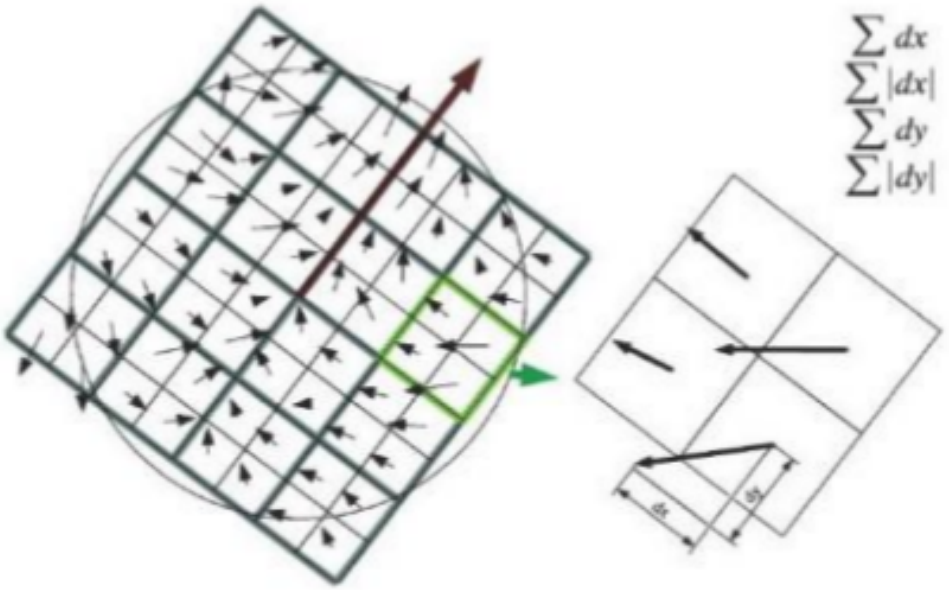


Figura 1.6: Descriptor SURF

1.3.4. Matching de Descriptores

Fast Approximate Nearest Neighbor Search

Una vez identificados los puntos de interés tan sólo debemos establecer el matching entre ambos. El método más ampliamente usado es el de agrupar los puntos de interés mediante el algoritmo de la búsqueda de los vecinos más cercanos.

1.4. Ciratefi

Ciratefi es es una técnica de match templates en escala de grises, el algoritmos se compone de tres sucesivos filtros excluyendo pixels que no tienen posibilidad de matching con el template deseado[Ciratefi].

Este algoritmo es invariante a la rotacion escala y traslación está diseñado principalmente para computar el matching en paralelo aprovechando que todos sus pasos son paralelizables.

El principal problema de color template matching es la constancia del color, es decir como extraer información de manera que permanezca constante con la iluminación, por ello este algoritmo en su versión más estable está optimizado para matching en imágenes en escalas de grises.

1.4.1. Algoritmo

- A: Imagen a buscar.
- T: el template a ser buscado.

A. First step - Circular Sampling Filter (Cifi)

Se crea un conjunto de radios : $\{r_0, r_1, \ldots, r_{l-1}\}$

$$C_A[x, y, k] = \frac{1}{P_k} \sum_{\theta=0}^{P_k-1} A\Big(x + r_k \cos(\frac{2\pi\theta}{P_k}), y + r_k \operatorname{sen}(\frac{2\pi\theta}{P_k})\Big)$$

$P_K = \mathbf{round}(2\pi r_k)$
$0 \leq k < l$
Luego dado un Template a buscar y n escalas, $\{s_0, s_1, \ldots, s_{n-1}\}$

$$C_t[i, k] = \frac{1}{P_k} \sum_{\theta=0}^{P_k-1} T\Big(x + r_k \cos(\frac{2\pi\theta}{P_k}), y + r_k \operatorname{sen}(\frac{2\pi\theta}{P_k})\Big)$$

:x_0, y_0 es el pixel central de T y $0 \leq i < n$

$$CisCorr_{A,T}(x, y) \max_{i=0}^{n-1} \Big[Corr(C_t[i], C_a[x, y])\Big]$$

Donde $Corr(a, b)$ es la normalizacion entre los vectores a y b.
Cada pixel (x, y) se filtra primero por algún umbral t_1 mendiante la condición:

$$if\ CisCorrA, T(x, y)\ \geq\ t_1$$

La probable escala para (x, y) es s_i donde i es el argumento que maximiza: $CisCorr$

B. Second step - Radial sampling filter (Rafi)

Este paso usa la proyección de la imagen A y B sobre un conjunto de lineas radiales para emplear el segundo grado de filtros.

Rafi estima el ángulo probable de rotación para cada pixel candidato.

El largo de las lineas radiales ($\lambda = r_{l-1}s_i$) se calcula de acuerto al máximo radio r_{l-1} y la probable escala s_i calculada por Cifi.

Para cada pixel de la imagen A se computa la matriz R_A, considerando un conjunto de m ángulos $(\alpha_0, \alpha_1, ..., \alpha_{m-1})$, como se describe a continuación:

$$R_A[x, y, j] = \frac{1}{\lambda} \sum_{t=0}^{\lambda} A(x + t\cos\alpha_j, y + t\sin\alpha_j)$$

En otras palabras, $R_A[x, y, j]$ es el promedio de la escala de grises de píxelles de A en la línea radial con vértice en el pixel: (x, y), con tamañ λ y ángulo α_j.

Luego T es se procesa mediante un vector R_T con m características:

$$R_t[j] = \frac{1}{r_{l-1}} \sum_{t=0}^{\eta-1} T(x_0 + t\cos(\alpha_j), y_0 + t\sin(\alpha_j))$$

Luego **Rafi** computa la correlación **RasCorr** entre los vectores $R_A[x, y]$ and R_T para cálcular el mejor mátching en distintos ángulos:

$$RasCoor_{A,T}(x, y) = \max_{j=0}^{m-1}[Corr(R_A[x, y], cshift_j(R_t))]$$

$if RasCorr_{A,T}(x, y) \geq t_2$, para alún umbral t_2.

El probable ángulo de rotación del pixel (x, y) es α_j donde j es el argumento que **maximiza RasCorr**.

C. Third step - Template matching filter (Tefi)

Este paso filtra el a un segundo grado los píxeles candidatos usando un convención template matchin con el coeficiente de corelación como métrica.

Esta tarea es rápida porque Cifi y Rafi computaron la probable escala y ángulo de cada cada pixel candidato.

Tefi computa el coeficiente de correlacion usando un umbral t_3, para evaluar que tan buenos son los píxeles obtenidos tras el segundo filtro. ■

Detección de espacios publicitarios en televisión.

Desde Infoxel, empresa dedicada a medir el tiempo de las publicidades en los medios tradicionales, me sugirieron la idea de aprovechar el estudio que venía realizando en imágenes para intentar automatizar la detección de espacios publicitarios. Porque si un sistema puede detectar automáticamente estos segmentos podrían llegar a concentrar el esfuerzo de medir la publicidad en una franja de pocas horas por día y no a las 24.

Esta automatización es posible debido a que en Argentina se sancionó la ley de medios, decreto 1225/2010, que establece en el artículo 82, que los medios televisivos sólo pueden destinar entre 12 y 8 minutos cada hora de transmisión para emitir publicidad dependiendo si son canales de aire o cable respectivamente. Y que el tiempo de emisión de publicidad debe ir contenido dentro del espacio publicitario, **segmento delimitado por separadores que indiquen su comienzo y finalización**. Indicando que sólo para espectáculos deportivos podrán agregarse banners, siempre que no superen el 30 % del tamaño total de pantalla pudiéndose ubicar en la parte inferior de la misma mientras no interrumpa momentos trascendentes en la continuidad del mismo.

El problema planteado fue el siguiente: tengo un directorio con archivos de video que corresponden a un segmento de un medio televisivo se busca almacenar en otro directorio los videos que sólo contengan espacios publicitarios, sin perder la referencia al medio y al tiempo en que fueron transmitidos.

2.1. Desarrollo

Del primer planteo del problema se detectan distintos conceptos, con persistencia y funcionalidad, que en un diseño orientado a objetos deben ser abstraídos en una clase.

Cada **medio** televisivo está identificado unívocamente por un nombre, por ejemplo Canal 12, Magazine, Tn, Canal 7. A partir de la ley de medios, es obligatorio el uso de separadores que indiquen el comienzo y fin del espacio publicitario. Estos separadores deben ser reconocidos una vez por un operario y almacenados en una base de datos como **patrones**. Como un medio cuenta con varios patrones estos deben indicar si marcan el comienzo o el fin del espacio publicitario y a su vez almacenan la referencia a la imagen utilizada como patrón para el futuro motor de matching.

Una vez que tenemos almacenado los patrones de cada medio, debemos analizar los archivos de video. Para cada video, asociado por el nombre al medio, se buscan estos delimitadores mediante una técnica de template matching con los patrones.

Un video está formado por una sucesión de imágenes, llamados frames, según el tipo de compresión puede tener típicamente entre 15 a 60 frames por segundo.

Entonces se reduce el problema de detectar un patrón en un video a detectar un patrón en un frame del mismo, que es una imagen.

Entonces debe implementarse un algoritmo que recorra el video frame por frame y realizando esta comparación, a este algoritmo se le llamará motor.

El motor cuando detecte el matching debe almacenar el segundo del video donde hubo matching y el patrón que se detectó, a esta instancia se la denominó **acierto**.

Cuando el motor termina de analizar todos los videos, tenemos una lista de aciertos, que indican para cada medio qué patrones se detectaron. A partir de ahí se agrupan los aciertos del mismo medio, denominado **certezas**, este grupo debe reducirse a una lista de pares que indiquen comienzo y fin por cada medio, es decir un espacio publicitario abstraido en una clase **Tanda**.

El script que transforma una instancia de la clase certezas en una lista de Tandas se denominó aciertos.py.

Finalmente con la lista que almacena la información de una tanda se recortan los videos y se mueven al directorio de destino.

Diagrama de Clases

Del diseño orientado a objetos se culminó en el diagrama de clases de la figura 2.7, en él podemos identificar las clases necesarias para resolver el problema y sus atributos y procedimientos principales.

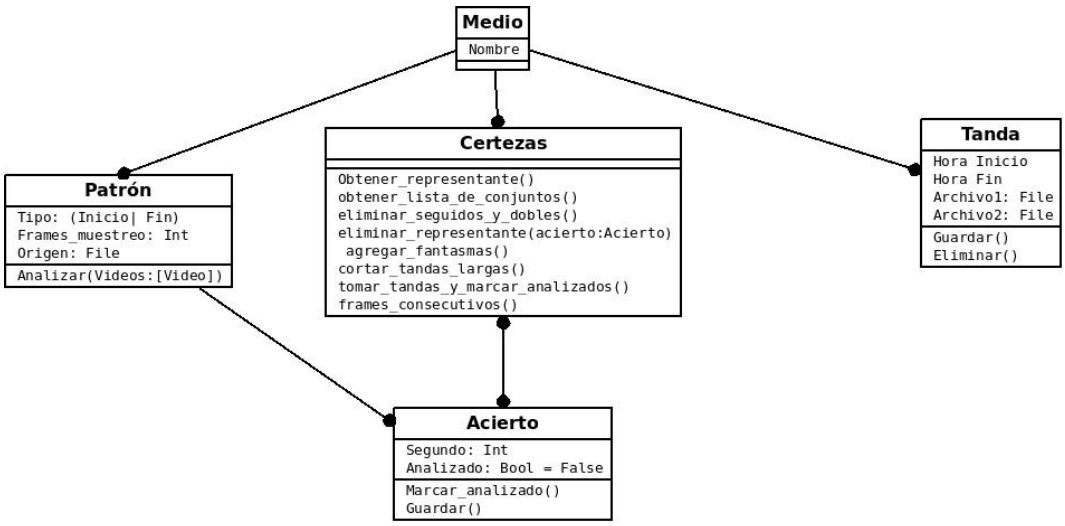


Figura 2.7: Diagrama de Clases: Detector de Tandas Publicitarias

La clase Patrón

Esta clase, es la que implementa la noción de tener una imagen de referencia que va a indicar si es de tipo Fin o Inicio, referido a un medio.

El atributo frames_muestreo, es un campo que indica cada cuántos frames del video debe buscarse dicho patrón.

La clase Medio

Esta clase, implementa la noción del medio televisivo, su nombre, identificador, no posee lógica.

La clase Acierto

Un acierto, será una detección de algún patrón, en determinado archivo, se almacena en que segundo fue la misma.
El atributo analizado, indica si este acierto fue revisado por el creador de tandas.

La clase Certezas

Modelo principal de la aplicación, almanena varios aciertos de un medio, el script aciertos.py, es el encargado de transformar esta cantidad de aciertos, en una sucesión de tandas publicitarias.

La clase Tanda

Producto final, tabla para almacenar el comienzo y fin del espacio publicitario de determinado medio.

2.1.1. Elección del método de Matching

Por los conocimientos adquiridos en la facultad en Python se decidió utilizar la librería Opencv para poder analizar rápidamente distintos algoritmos y concentrar el estudio en definir un umbral acorde para el matching.
Se optó por el método de correlación normalizado **TM_CCOEFF_NORMED** ya que este método tiene en cuenta en su cálculo el tamaño del patrón y por ser normalizado es de fácil interpretación porque un resultado cercano a 0 indica que no hay mátching y por lo contrario el resultado cercano a 1 indica que sí lo hay. De todas maneras para otro proyecto es importante hacer pruebas para decidir cuál es el mejor método en cuanto a performance requerida ya sea tanto en tiempo como en precisión.

2.1.2. Matching y Aciertos

El método MatchTemplate, devuelve un arreglo con las distancias de los puntos, al estar normalizado, ésta toma valores entre 0 (Cuando no hay match) y 1 cuando se da un Match perfecto.
Continuamente se lleva a cabo el siguiente proceso:

1. Para cada medio, tomamos los videos a analizar.
    - Simplemente filtramos los archivos codificados con el nombre de ese medio, ordenados temporalmente.

2. Para cada video, tomar un frame cada frames_muestreo.

   Frames_muestreo

   Este atributo está presente en cada patrón, e indica cada cuántos frames debe buscarse el patrón.

   ¿Cómo se establece?

   Valor Inicial: al agregarse un patrón, se analiza en los últimos n videos, cuál es la moda de frames consecutivos en los que aparece dicho patrón, este es el valor inicial, esto se hace para no comparar todos los frames del video ya que es muy costoso y no es necesario.

   Adecuación automática del valor: a medida que el motor va detectando tandas, es posible que al no tomar todos los frames que contiene un video y saliendo en muy pocos frames el patrón, no se compare. Entonces, a medida que el motor detecta esta situación va decrementando en una unidad el atributo frames_muestreo.

3. Para cada frame, utilizar el método MatchTemplate con cada patrón del medio.

4. Si existe un punto donde la precisión es mayor al 80 %, tenemos un Match , **creamos un Acierto**.

Código de Motor.py encargado de crear los aciertos:

```
Motor.py
for video in videos:
  patrones = Averiguar_patrones(video)
  Buscar(video,patrones){
    for frame in video:
      aciertos = buscar_patron(frame)
      almacenar(aciertos)
  }
```

```
def buscar_patron(frame,patron):
    puntos = []
    contador = 0

    #Lo transformo a gris
    img_gris = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    template = cv2.imread(PATRONES + "%s.jpg"  %patron[0],0)

    # Averiguo el ancho y alto de la imagen
    w, h = template.shape[::-1]

    #Metodo de comparacion
    resultados = cv2.matchTemplate(img_gris,template,cv2.TM_CCOEFF_NORMED)

    #Coeficiente de igualdad 0<x<1
    umbral = 0.8
```

```
17
18      #Filtro por el coeficiente de igualdad
19      loc = np.where( resultados >= umbral)
20      # Devuelve una tupla de arrays (y,x) con los puntos de intersección
21      # Recorro los x,y, es decir *loc para ver el contenido de los arrays
22      # zip para juntar las dos listas
23      # [::-1] Para invertir los arrays
24
25      # Obtengo los puntos de intersección
26      for pt in zip(*loc[::-1]):
27          x = pt[0]
28          y = pt[1]
29          # Me fijo si ese patron ya lo tenia
30          if not esta(puntos,x,y,w,h):
31              contador += 1
32              cv2.rectangle(frame, pt, (pt[0] + w, pt[1] + h), (0,0,255), 2)
33              puntos.append(pt)
34      return contador > 0
```

2.1.3. Crear Tandas a partir de los Aciertos

En este punto tenemos la tabla Aciertos llena para cada medio.

A partir de allí para cada Medio, creamos una instancia de la clase Certezas, utilizadas justamente para agrupar los aciertos de un medio.

Para llegar a distinguir tandas, es decir, pares de aciertos de distinto tipo, de la manera (Inicio,Fin), hay que hacer alguna limpieza de duplicados, posibles faltantes o existencias de supuestas tandas de más de 30 mimnutos.

A continuación se explayarán los principales métodos y filtros que se aplican a este conjunto de aciertos para llegar a las tandas:

1. Eliminar_dobles_y_repetidos(): lo que hace es detectar que acierto es igual a otro dentro de un margen establecido de segundos, esto se debe a que al compara a una tasa menor de un segundo se detectan multiples match del mismo patrón en un lapso muy corto de tiempo, entonces sólo tomamos uno de estos como representante.

2. Agregar_fantasmas: en ocasiones cuando el canal no pone una placa o se le pasa al motor alguna placa separadora llegan detecciones de 2 Inicios o 1 fin, por lo que se procede a agregar un acierto según el caso:

   - a: I I , se crea IFI
   - b: F, se crear IF

   El acierto se inventa a un tiempo promedio de duración de tanda del medio o 12 minutos, en su defecto para no tomar todo el tiempo donde no hay una tanda publicitaria.

3. En este punto la lista de aciertos es de la forma [I,F,I,F ...], solo se procede a verificar que al comparador no se le haya pasado una tanda completa de modo que se cree una tanda muy larga, en dicho caso se toma el promedio de duración de las

tandas de ese medio, y se crean dos tandas uan desde el inicio de la misma hasta el inicio + promedio y la segunda tomando los ultimos n minutos hasta el final: tal que n es el promedio de duración de las tandas del medio más dos o tres minutos.

4. Finalmente si el largo de la misma es par se toman de a dos y se generan las tandas, si es impar el último es Inicio, por lo que deja como no analizado en la base de datos para la proxima corrida.

Aciertos.py

Código de Aciertos.py

```
Código encargo de crear las tandas a partir de los aciertos
tandas = []
diccionario = obtener_aciertos()
for medio,aciertos in diccionario.items():
    certezas = Certeza(medio,aciertos)
    certezas.eliminar_seguidos_y_dobles(aciertos)
    certezas.agregar_fantasmas(res)
    tandas += certezas.tomar_tandas_y_marcar_analizados()
crear_tandas(tandas)
```

2.1.4. Testing

Para el diseño de test se utilizaron las librerías mock y unittest de python.

Un apartado especial a las utilidades de la librería mock ya que al requerir un sistema de archivos para el funcionamiento del sistema, mock permite definir el resultado de una función en sucesivas llamadas.

Para la entrega fue requerido el testeo total de las funcionalidad del sistema y se definieron los siguientes casos de test.

Diseño de casos de test: Inventar Fin

1. Verificar invento de fin separdo por PromedioTiempoTanda:

   - **Requisito a testear**: que se invente un fin con duración promedio tiempo tanda a partir del primer inicio, en el caso que en la base de datos haya dos inicios.
   - **Configuración del Test**: crear dos Inicios separados por un tiempo mayor a promedio tiempo tanda en un segundo.
   - **Salida Esperada**: Ambos inicios no se modifican y se agrega un Fin intermedio separado del primer inicio por PromedioTiempoTanda.
   - **Salida**: Ambos inicios no se modifiraron y se agregó un Fin intermedio separado del primer inicio por PromedioTiempoTanda.
   - **Match**: Sí.

2. Verificar invento de fin un segundo antes del segundo Inicio en el mismo origen:

- **Requisito a testear**: que se invente un fin con duración promedio tiempo tanda.
- **Configuración del test**: crear dos Inicios con distancia menor a promedio tiempo de tanda en el mismo origen.
- **Salida Esperada**: Ambos inicios no se modifican y se agrega un Fin intermedio un segundo antes del segundo Inicio.
- **Salida**: Ambos inicios no se modificarón y se agregó un Fin intermedio un segundo antes del segundo Inicio.
- **Match**: Sí.

3. Se inventa un fin Con duración PromedioTiempoTanda en Origen intermedio:

   - **Requisito a testear**: que se invente un fin con duración promedio tiempo tanda.
   - **Configuración del test**: crear dos Inicios con distancia menor a promedio tiempo de tanda en el mismo origen.
   - **Salida Esperada**: Ambos inicios no se modifican y se crea un Fin un segundo antes del segundo Inicio con su mismo Origen.
   - **Salida**: Ambos inicios no se modificaron y se creó un Fin un segundo antes del segundo Inicio con su mismo Origen.
   - **Match**: Sí.

4. Se inventa un fin Con duración menor a promedio tanda en el segundo Origen:

   - **Requisito a testear**: que se invente un el fin con final un segundo antes del comienzo del segundo inicio.
   - **Configuración del test**: crear dos Inicios con distancia menor a promedio tiempo de tanda con distintos orígenes.
   - **Salida Esperada**: Ambos inicios no se modifican y se agrega un Fin un segundo antes del segundo Inicio, con su mismo origen.
   - **Salida**: Ambos inicios no se modificaron y se agregó un Fin un segundo antes del segundo Inicio, con su mismo origen.
   - **Match**: Sí.

Diseño de casos de test: Inventar Inicios

1. Verificar que se cree un Inicio.

   - **Requisito a Testear**: Se inventa un Inicio Con duración PromedioTiempoTanda
   - **Configuración del Test**: Crear un Fin En el segundo PromedioTiempoTanda
   - **Salida Esperada**: El Fin no se modifica y se agrega Un Inicio separado del Fin por PromedioTiempoTanda.

- **Salida**: El Fin no se modificó y se agregó un Inicio separado del Fin por PromedioTiempoTanda.
- **Match**: Sí

2. Verificar que se cree un Inicio

   - **Requisito a Testear**: Se inventa un Inicio hasta el Fin.
   - **Configuración del Test**: Crear un Fin En un segundo $<$ PromedioTiempoTanda
   - **Salida Esperada**: El Fin no se modifica y se agrega un Inicio, en el segundo 0 Del Origen del Fin.
   - **Salida**: El Fin no se modificó y se agregó un Inicio en el segundo 0 del Origen del Fin.
   - **Match**: Sí

3. Verificar que se cree un Inicio en el segundo 0 del Origen aunque la tanda sea muy corta.

   - **Requisito a Testear**: Se inventa un Inicio en el segundo 0 Del origen.
   - **Configuración del Test**: Crear un Fin en el segundo 10, con 1 solo Origen en la Base de Datos.
   - **Salida Esperada**: El Fin no se modifica y se crea un Inicio en el segundo 0.
   - **Salida**: El Fin no se modificó y se creó un Inicio en el segundo 0.
   - **Match**: Sí.

4. Verificar que se cree un Inicio a una distancia de PromedioTiempoTanda del Fin.

   - **Requisito a Testear**: Se inventa un Inicio Con duración PromedioTiempoTanda Hasta el segundo del Fin
   - **Configuración del Test**: Crear un Fin en un segundo $<$ PromedioTiempoTanda y con 2 Origenes anteriores de 10minutos cada Uno.
   - **Salida Esperada**: El Fin no se modifica y se crea un Inicio separado ppr PromedioTiempoTanda del Fin en el primer Origen anterior.
   - **Salida**: El Fin no se modificó y se creó un Inicio separado ppr PromedioTiempoTanda del Fin en el primer Origen anterior.
   - **Match**: Sí.

Diseño de casos de test: Cortar tandas Largas

1. Verificar que no se modifique una tanda correcta (Duración $<$ PromedioTiempoTanda)

   - **Requisito a Testear**: No se modifica el par ni se crean nuevos Aciertos.
   - **Configuración del Test**: Crear un Inicio y Fin on una distancia Menor MaximaCantidadMinutosTanda.
   - **Salida Esperada**: El inicio y el Fin no se modifican.
   - **Salida**: El Inicio y el Fin no se modificaron.
   - **Match**: Sí.

2. Verificar que se creen dos Aciertos Intermedios de manera de Dividir una tanda larga en dos tandas, la primera que empiece en el Inicio de la tanda Larga y la segunda que termine en el segundo del Fin de la tanda larga, ambas con duración promedio tiempo tanda.

   - **Requisito a Testear**: Se inventa un Fin desde el Inicio con DuracionPromedioTanda y se crea un Inicio con duracion máxima Promedio Tiempo Tanda.
   - **Configuración del Test**: Crear un Inicio y Fin Con una distancia Mayor a MaximaCantidadMinutosTanda, crear archivos 3 archivos intermedios con duración 20minutos.
   - **Salida Esperada**: El inicio y el Fin originales no se mofican y se agrega un Inicio a una distancia PromedioTiempoTanda del Fin y un Fin a una distancia PromedioTiempoTanda del Inicio Orignal (Estos no deben superponerse).
   - **Salida**: El Fin e Inicio final no están superpuestos y formaron junto a los Aciertos Originales Dos tandas de Duración PromedioTiempoTanda.
   - **Match**: Sí.

2.2. Conclusión Detector Espacios Publicitarios

2.2.1. Tiempo de procesamiento de un Medio

Tomando como referencia que a un medio generalmente se le asignan dos patrones, uno que inidique el inicio y otro el fin, el motor demora en promedio 45 segundos por cada 10 minutos de video analizado, por lo que procesar la cantidad de video generado por un medio en un dia le lleva **108 minutos**.

Al agregar un patrón se incrementa un tiempo promedio de 30 minutos por dia en un medio, variando según el tamaño del patrón.

2.2.2. Automatización del proceso de detección de tandas

Una vez establecidos los patrones que identifican los separadores del medio, el proceso de corte de tandas publicitarias es totalmente automático. Este proceso antes de la implementación de esta aplicación era manual, y aproximadamente les llevaba, mediante una interfaz de corte, $1hs$ cada cuatro medios.

También incide en el momento de corte, ya que antes sólo se detectaban estas tandas publicitarias a determinada hora del día, por lo que se generaba una demora, en la revisión. Mediante esta aplicación al estar corriendo todo el tiempo, la demora sólo se limita 10minutos, la espera de generar los archivos.

2.2.3. Reducción de tiempo de Procesamiento

A partir de esta posibilidad, es posible utilizar el comparador de imágenes para detectar estos separadores y reducir $24hs$ de video por medio a $3hs$ tal como lo establece la ley de medio audiovisuales.

Esto les permite al motor que detecta las publicidades analizar tan sólo este periodo de tiempo.

2.2.4. Problemas encontrados

Inicio y Fin solapados

Este problema surgió al princio ya que al tener un umbral del 80 %, hay situaciones donde si agregamos toda la frase **fin espacio publicitario**, el motor detecta como ambos patrones en las dos situaciones, ver figura 2.8, por ello aconsejamos cortar la palabra fin como patrón distintivo.

Esto no soluciona el hecho de que se detecten ambos patrones en el separador que indica el fin, por ello se agregó una lógica de que si en un periodo breve de segundo, el motor detecta Inicio y fin, el script aciertos.py, los reúne y toma como acierto representante al Fin.

Falsos Positivos

Los falsos positivos generalmente suceden frente a dos razones:

Figura 2.8: Problema generado por la inclusión del patrón de inicio de espacio publicitario en la placa de fin de espacio publicitario.

1. Umbral del 80 %: al fijar el parámetro de certeza, en ciertas situaciones, ambos patrones, son detectados sin ser iguales, figura 2.9.

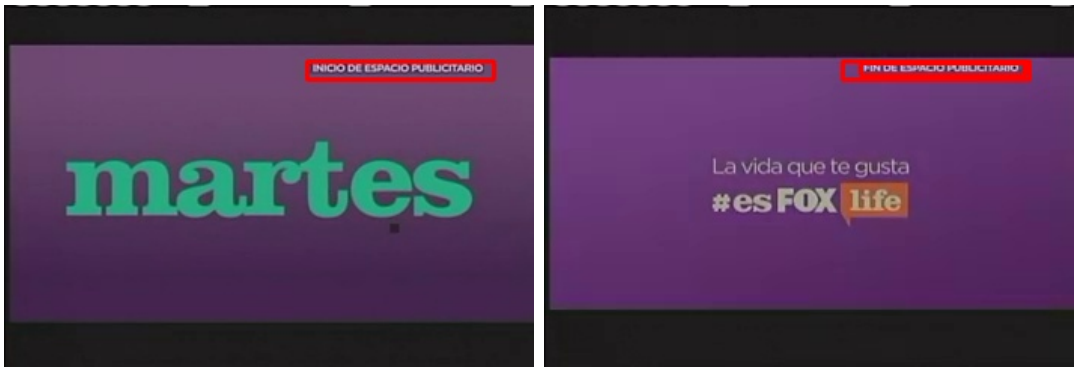


Figura 2.9: Falso positivo generado por el umbral al 80 %

2. Patrones borrosos: al tomarse el patrón de un video, es probable que existan frames donde éste no se encuentre estable, por lo que si se toma dicho patrón como separador, comienza a generar falsos positivos, figura 2.10.

Figura 2.10: Detecciones al tomar patrones borrosos, que contengan letras.

3. Calidad baja del video: al tener una calidad baja del video, necesariamente debemos cortar patrones de gran tamaño ya que sino dicho patrones no son distinguibles de imágenes comunes.

Ausencia de placas distintivas

Esto sucedió en el medio **Cosmopolitan**, que utiliza la misma placa para indicar el inicio como la finalización del video, figura 2.11.

La lógica actual del proyecto, no contempla dicha situación, pero podría solucionarse tomando como tandas, sólo aquellos segmentos menores a determinada cantidad de tiempo.

Figura 2.11: Placa utilizada en Cosmopolitan para marcar el espacio publicitario

Detector de Logos

3.1. Elección del método de Matching

Debido a que los logos en los espectáculos deportivos, van cambiando de ángulos y tamaños y no siempre están visibles de manera completa, me decidí por combinar el método de matchtemplate con patrones fijos más la utilización del método SURF.

3.1.1. Estableciendo parámetros al comparador

- **Umbral del comparador fijo**: éste parámetro no puede ser menor a un 80 % de igualdad, debido a que un umbral menor genera una enorme cantidad de falsos positivos, fundamentalmente si el patrón tomado es pequeño.
- Definir **cantidad de puntos** invariantes a tomar: surf = cv2.SURF(300)
- **Matching de descriptores**: se definió un umbral del 80 %.

  Detalle de dos métodos de matching de descriptores:

```
1  def matching_flann(descriptor_1, descriptor_2, umbral = 0.8):
2      flann = cv2.flann_Index(descriptor_2)
3      index_2, distancias = flann.knnSearch(descriptor_1, 2)
4      marcara = distancias[:,0] / distancias[:,1] < umbral
5      index_1 = np.arange(len(descriptor_1))
6      aciertos = np.int32( zip(index_1, index_2[:,0]) )
7      return aciertos[mascara]
8
9  def match_case(descriptor_1, descriptor_2, umbral = 0.8):
10     res = []
11     for i in range(len(descriptor_1)):
12         distancias = normal( descriptor_2 - descriptor_1[i] )
13         ubicacion_1, ubicacion_2 = dist.argsort()[:2]
14         r = distancias[ubicacion_1] / distancias[ubicacion_2]
15         if r < umbral:
16             res.append((i, ubicacion_1))
17     return np.array(res)
```

- **Definición de matching**: una vez que tenemos definidos la cantidad de puntos que matchearon, se define otro $umbral = \frac{len(matched_{p_1})}{len(p_1)}$ donde p_1 es la lista de puntos característicos de la imagen del patrón.

3.2. Desarrollo

Para generara una muestra ilustrativa de los features del motor se tomó el evento deportivo: Final del grand slam **Abierto de Australia** del 2014.

Características del video:

- Formato de Compresión: Avi.
- Calidad: 400px * 700px .
- Duración: 3Hs 16minutos.

Se realizó una prueba tomando los siguientes patrones, extraídos del video, figura 3.12.

Figura 3.12: Logos utilizados, de izquierda a derecha: Kia, Melbourne, MLC y Medibank

3.2.1. Invarianza a rotación de ángulos

A continuación mostraremos cómo se comporta el motor cuando el patrón en la imagen está rotado. La secuencia de imágenes de la figura 3.13 muestra la capacidad del motor de detectar el patrón de Kia en distintos ángulos. Del mismo modo la secuencia de imágenes de la figura 3.14 muestra la misma capacidad con el patrón de Medibank.

Figura 3.13: Patrón detectado de Kia rotado en distintos ángulos.

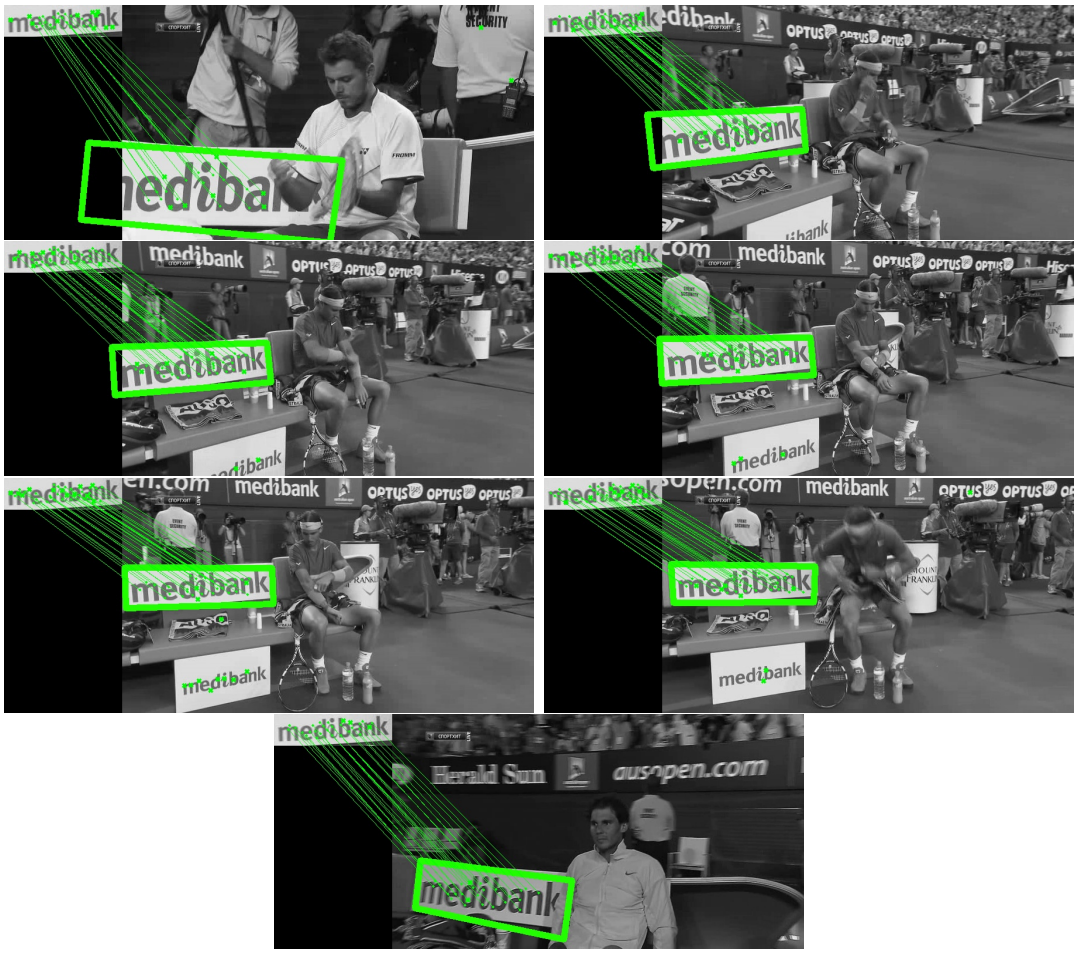


Figura 3.14: Patrón detectado de Medibank rotado en distintos ángulos.

3.2.2. Invarianza a cambios de escala

A continuación mostraremos cómo se comporta el motor cuando el patrón en la imagen está en distintos tamaños, este hecho en la transmisión de un evento deportivo se da porque el camarógrafo decide hacer zoom en distintos momentos de alguna zona para mostrar algún gesto o una visión panorámica.

La secuencia de imágenes de la figura 3.15 muestra la capacidad del motor de detectar el patrón de Kia en distintos tamaños. Del mismo modo la secuencia de imágenes de la figura 3.16 y 3.17 muestra la misma capacidad para el patrón de Mlc y Melbourne respectivamente .

Figura 3.15: Patrón detectado de Kia en distintos tamaños.

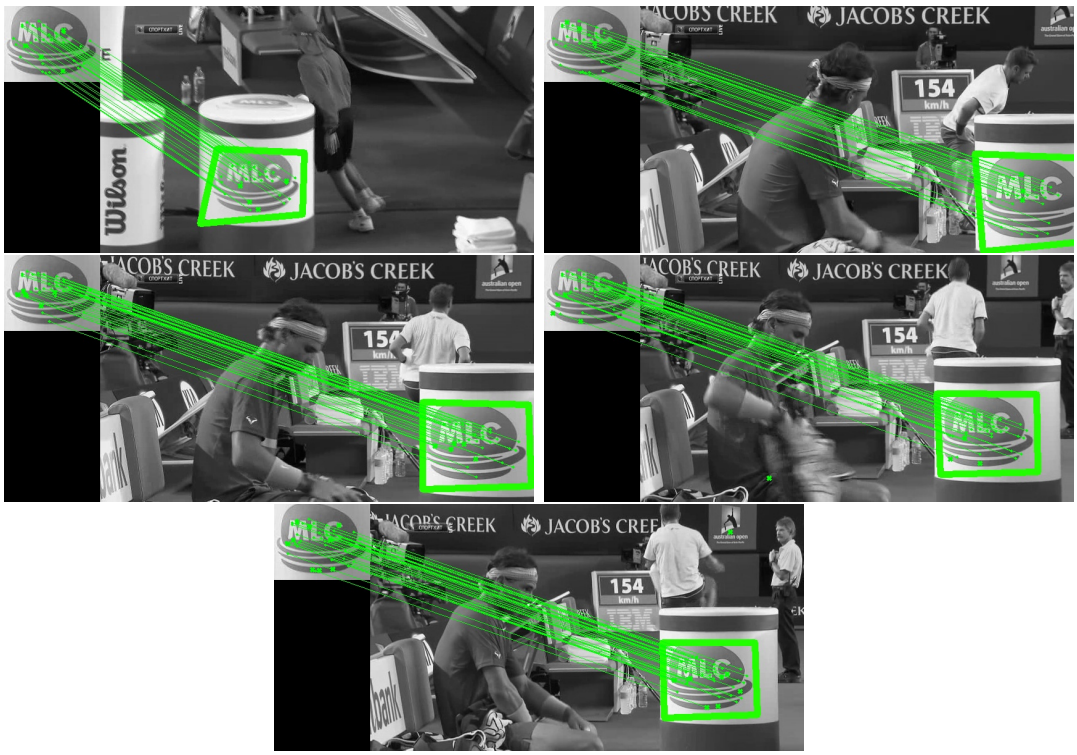


Figura 3.16: Patrón detectado de Mlc en distintos tamaños.

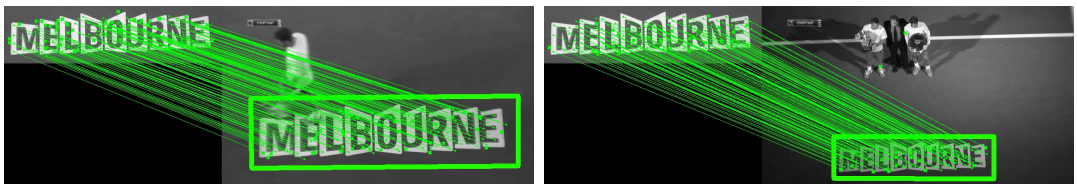


Figura 3.17: Patrón detectado de Melbourne en distintos tamaños.

3.2.3. Paralelo.py

Se creó un script para emplear la cantidad de procesadores al máximo, con un procesador i7, es decir mediante 8 procesadores. El script toma como input un directorio donde están todos las imágenes del video descomprimido y asigna módulo n procesos las mismas cantidades.

3.2.4. Area Detectada

El motor calcula el área donde el patrón fue detectado y con esto clasifica las detecciónes según el porcentaje de pantalla que estos ocupan, estos parámetros son ajustables, inicialmente se toman los siguiente valores:

Tamaño	Porcentaje
Pequeño	$< 25\,\%$
Mediano	$< 40\,\%$
Grande	Caso contrario

3.2.5. Problemas Encontrados

Falsos Positivos

Los falsos positivos se debieron fundamentalmente a dos causas:

- Patrones borrosos o pequeños, al ser un comparador de puntos caracterśiticos el método SURF al construir los descriptores, sólo junta una pequeña cantidad, generalmente se generean problemas con menos de 10, ya que al conseguir matchear una pequeña cantidad de puntos, se logra un porcentaje alto de aciertos.

  Es por esto que el motor advierte esta condición, señalando como patrones obsoletos aquellos en los que el motor detecta menos de 10 puntos característicos.

- Logos de marcas circulares, le método SURF advierte que al ser invariante a escala y rotación, hay que tener cuidado de dar de alta patrones circulares, ya que si encuentra alguna circunferencia en la imagen de destino con similar ilumicación va a tener muy altas probabilidades de matchear.

  De todas maneras, este falso positivo que se observa en la figura 3.18, sólo se produce con algunos patrones circulares que el motor sigue detectando, y generalmente puede eliminarse si se mide que la superfice de la zona detecta consiste en una región de pocos píxeles.

Figura 3.18: Falso positivo con un patrón circular.

Del mismo modo es importante destacar que este patrón cuando aparece, ver figura 3.19, es detectado con la misma performance que los demás.

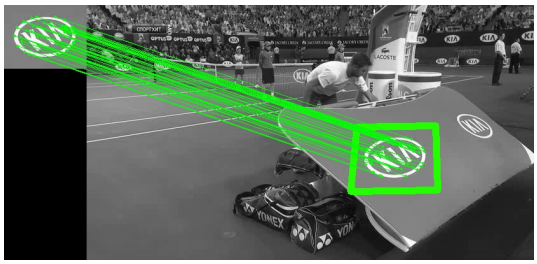


Figura 3.19: Detección de patrón circular.

Errores al estimar el tamaño de detección

Al estimar la zona de acierto, por un polinomio que encierra los puntos de matching, fue necesario tomar una media recortada de estos puntos, ya que usualmente algún punto caracteerístico es detectado en otra región como lo muestra la figura 3.20.

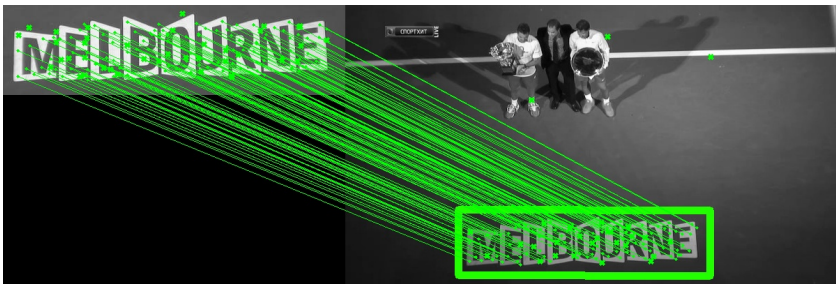


Figura 3.20: Puntos sueltos detectados de igual características.

3.2.6. Validación

Para realizar una medición del sistema se pidió los datos a Infoxel S.A. medidos manualmente entregados a Coca-Cola Argentina para el mundial.

Se decidió hacer un script "validación.py"que realice una comparación automática entre el csv generado por el programa y los datos provistos por la empresa prueba sobre el partido Argentina Bosnia disputado el 14 de Junio del 2014 y medir los siguientes items en distintas corridas del programa aumentando la cantidad de Fps

- Calidad: al tener datos revisados manualmente podemos revisar la cantidad de falsos positivos detectados sobre el total de detecciones.
- Cantidad: Cantidad de segundos totales detectados. Se considera que se detectó publicidad en un segundo si se detectó en por lo menos 1 frame.

Los datos recibidos al ser manuales no tienen precisión en el segundo exacto de la detección, sino que son precisos hasta el minuto, por lo que se agregó un margen de 30 segundos para considerar que un acierto es un point.

Para analiizar una prueba de reconocimiento de patrones con clasificación binaria existen estadísticos y test clásicos asociados [F-Factor, 2007] para determinar la validación del experimento.

Uno de los más conocidos, utilizado especialmente en química es el cálculo del estadístico F, que se calcula de la siguiente manera.

Sean:

- Tp, True Positive: el sistema lo detectó y existía. (Acierto)
- Fp, False Positive: el sistema lo detectó y no existía. (Confusión)
- Fn, False Negative: el sistema no lo detectó y existía. (Omisión)

$$Precision = \frac{Tp}{Tp + Fp}$$

$$Recall = \frac{Tp}{Tp + Fn}$$

$$F = 2 \times \frac{(Precision \times Recall)}{(Precision + Recall)}$$

Los resultados fueron los siguientes:

Fps	Calidad	Cantidad	Tiempo	Estadístico F
1	0.955936	41 %	13'	0.407982
2	0.860837	59 %	26'	0.702077
4	0.849074	87 %	51'	0.858648
8	0.946211	111 %	1Hs 40'	0.927108

3.3. Conclusión Detector de Logos

Se observó que mientras más frames por segundo (Fps) analice el sistema éste logra detectar más apariciones.

El aumento de en porcentaje de falsos positivos detectados no está ligado a la cantidad de frames que se analicen si no al umbral fijado en el matching, ya que el análisis de cada caso es independiente.

Una de las principales características de la prueba del motor es que se emplearon patrones cortados directamente del video de origen, esto no quiere decir que no pueden tomarse los logotipos de las marcas de otra fuente, pero al variar tanto, fundamentalmente en los distintos colores en los que se presentan estos logos, es preferible utilizar los patrones del video, de lo contrario para lograr una mayor detección hay que encontrar logos donde coincida la luz a la que esta sometido y los colores en los que se encuentra.

Conclusión

El ámbito de reconocimiento de patrones ha ido mutando desde la comparación de histogramas, la búsqueda de una sub-matriz estableciendo una norma a métodos invariantes a escala, rotación y traslación. Sin embargo no necesariamente el método que realiza una búsqueda con más características puede ser el más apto para nuestra finalidad. Esto es porque debemos tener en cuenta:

- El tiempo que lleva procesar las imágenes según el algoritmo elegido.
- La calidad del video requerida para lo que queremos detectar.

No obstante si previamente descomprimimos el video en frames logramos al reducir el problema a detectar un patrón en una imagen. Por esto podemos paralelizar aprovechando al máximo nuestra cantidad de procesadores ya que logramos dividir el tiempo de procesamiento.

Ambas aplicaciones logran analizar el video en menos tiempo que su duración:

- El detector de tandas publicitarias puede analizar 24 horas de video de un medio en 108 minutos.
- El detector de logos en un evento deportivo de 2 horas de duración puede ser analizado en 1 hora 40 minutos.

Esto indica que ambos sistemas soportan un análisis en tiempo real del video.

Ambas aplicaciones realizadas generan falsos positivos esto hace que no puedan ir inmersas dentro de una cadena crítica de procesamiento sin supervisión. Pero sí pueden ser de gran utilidad para la generación información estadística necesario en informes de marketing deportivo.

4.1. Trabajos Futuros

4.1.1. Detección de marcas Publicitarias

Una vez separado el espacio publicitario, se puede seguir inspeccionando hasta llegar a que marcas publicitaron en él.

La lógica emplearía fuertmente el asunto de que luego de una publicidad el medio está obligado a ir a una placa negra, que dura una pequeña cantidad de frames, indetectable por el ojo humano, pero sí por un sistema. Esto separaría una tanda en una decena publicidades.

A posteriori utilizando el sistema de detección de logos se podría hallar las marcas que invirtieron en dicho segmento publicitario. Procesando sólo segmentos de interés.

4.1.2. Limitar área de búsqueda en el comparador

Otra idea que puede aplicarse es tomar el x,y de la imagen original y directamente matchear las imágenes, utilizando sus histogramas, de todas maneras algunos medios no ponen una placa fija del espacio publicitario por lo que habría que aceptar ciertos desplazamientos y tomar mayor cantidad de frames por esta misma razón.

4.1.3. Elegir inteligentemente los frames, o construir Frames Robustos

Para evitar procesar frames parecidos, puede realizarse una comparación de frames consecutivos y filtrar sólo aquellos con un cambio mayor a un umbral .

Si nuestro video de origen no tiene alta calidad podemos tomar varios frames consecutivos y juntarlos para armar uno con menos ruido.

4.1.4. Aplicaciones en un Smart T.V.

En un futuro, no muy lejano, explotará un mercado de aplicaciones para los Smart T.V. Al poder delimitar automáticamente el espacio publicitario es posible hacer una app que silencie el televisor o reduzca el volumen al detectar el comienzo del espacio publicitario y lo regrese a su valor al finalizar

4.1.5. Reutilizar la lógica para Tandas publicitarias en Radios

Simplemente habría que utilizar un comparador de sonidos generando aciertos, y con la misma lógica generaríamos un cortador de tandas publicitarias en la radio.

4.1.6. Detección de múltiples situaciones del mismo Logo

Este problema se resuelvería de la siguiente manera, una vez detectado un patrón en deteminada región, se aplica una máscara a la misma, y vuelven a calcularse los descriptores de la imagen de interés pudiendo volver a encontrar el mismo patrón.

Para el caso del buscador de tamaño fijo esta situación ya está resuelta como puede observarse en la siguiente imagen:

Figura 4.21: Múltiple detección del mismo patrón.

Bibliografía

Aracil : Rafael Aracil López "Desarrollo de un sistema cognitivo de visión para la navegación robótica" Proyecto final de carrera Ingeniería Técnica Superior de Informática de Sistemas. Julio 2012.

LearnOpenCV : Gary Bradski, Adrian Kaehler. Title: "Learning OpenCV" O'Reilly Media. Septiembre 2008 (214-217).

OpenCV Open Source Computer Vision Library (OpenCV), `http://sourceforge.net/projects/opencvlibrary/`.

PAPER SIFT : D. Lowe. "Distintive Image Features From Scale Invariant Keypoints". International Journal of Computer Vision. Vol. 60 N^{ro} 2, pp. 91-110. Noviembre 2004. ISSN: 09205691. DOI: 10.1023/B:VISI.0000029664.99615.94.

Impl. SIFT : Vedaldi A., An implementation of SIFT detector and descriptor, UCLA CSD Tech. Report 070012 (2006).

PAPER SURF Ryuji Funayama, Hiromichi Yanagihara, Luc Van Gool, Tinne Tuytelaars, Herbert Bay, "ROBUST INTEREST POINT DETECTOR AND DESCRIPTOR", publicado en Noviembre 2009.

Impl. SURF : H. Bay, T. Tuytelaars, and L. V. Gool, "Surf: Speeded up robust features," in In ECCV, pp. 404-417, 2006.

FLANN : Marius Muja, David G. Lowe. Fast Approximate Nearest Neighbors with Automatic Algorithm Configuration, 2009.

F-Factor : Powers, David M W (2007/2011). Evaluation: From Precision, Recall and F-Factor to ROC, Informedness, Markedness & Correlation. Journal of Machine Learning Technologies 2 (1): 37-63

Pattern Classif. : Richard O. Duda, Peter E. Hart, David G. Stork (2001) Pattern classification (2da edición), Wiley, New York, ISBN 0-471-05669-3.

Ciratefi : Sidnei Alves de Araujo, Hae Yong Kim, "Ciratefi: An RST-invariant template matching" Volume 18 Issue 1, January 2011 Pages 75-90.

Ieco : Suplemento iEco, Clarín, "El campeonato de las marcas,7 de Diciembre 2014, pags 10 -11."