Universidad de Alcalá
Escuela Politécnica Superior

GRADO EN INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

TRABAJO DE FIN DE GRADO

Reconstrucción de mapas con detección de cambios a partir de Google Street View

Autor: Carlos Gómez Huélamo

Tutor: Luis Miguel Bergasa Pascual

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Reconstrucción de mapas con detección de cambios a partir de Google Street View"

Carlos Gómez Huélamo

2017

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO EN INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

"Reconstrucción de mapas con detección de cambios a partir de Google Street View"

Autor: Carlos Gómez Huélamo

Universidad: Universidad de Alcalá

País: España

Profesor tutor: Luis Miguel Bergasa Pascual

Presidente: Manuel Ocaña Miguel

Vocal 1º: Carlos Andrés Luna Vázquez

Vocal 2º: Luis Miguel Bergasa Pascual

Calificación: ……..

Fecha: 25/09/2017

"Hay una fuerza motriz más poderosa que el vapor, la electricidad y la energía atómica… la voluntad"

Albert Einstein

"Hijo, sé fuerte…"

Mi madre

Agradecimientos:

Este trabajo supone el culmen a cuatro años de carrera maravillosos, cargado de emociones, triunfos y a veces tropiezos que no hicieron nada más que alimentar un sentimiento que poco a poco fue tomando forma en mí: Querer ser un gran ingeniero, una persona lógica, que admira al mundo y se fascina de la constante evolución presente en él. En primer lugar agradezco a mi tutor Luis Miguel Bergasa Pascual el haberme dado la oportunidad de hacer este trabajo, en mi opinión muy bonito, visual e interesante. Igualmente agradezco a los compañeros del equipo RobeSafe la ayuda que me prestaron en determinados momentos, de donde me llevo un buen amigo, Roberto Arroyo, que además de lo gran profesional que es, sin duda, es mejor aún mejor como persona. Indudablemente, doy gracias (y regracias) a Dr. Pablo Fernández Alcantarilla por sus sabios consejos y ayudas.

A mi gran amigo Samuel, un espectacular estudiante de quien siempre he aprendido un poquito en nuestras largas conversaciones.

A mis amigos Diego, Iván, Jorge, Rodrigo, Luis, Rubén y Daniel, por ayudarme a desconectar cuando lo necesitaba.

A mis amigos de la universidad, especialmente a Pablo, Rubén, Sergio Pérez, Ramón, Edwin, Adrián, Alejandro, Antonio y Sergio Rodríguez, unas bellísimas personas y grandes ingenieros en el futuro.

A mi familia, en especial a mi padre y mi hermana, por aguantarme en casa.

Y a ti mami. Siento mucho que no hayas podido oírme en la siesta como iba, como me quejaba, como practicaba contigo ni que no puedas venir a mi presentación. Jamás olvidaré las últimas tres palabras que me dijiste en el aeropuerto con un hilito de voz, Hijo, sé fuerte...

Te prometí que te dedicaría un pedacito de nuestra canción favorita, aunque probablemente no me escuchases, y aquí lo tienes:

"…mis alas en el viento, necesitan de tus besos, acompáñame en el viaje que volar sólo no puedo. Y sabes que eres la princesa de mis ojos encantados, cuantas guerras he librado por tenerte a mi lado, no me canso de buscarte, no me importaría arriesgarte si al final de esta aventura yo lograra conquistarte…"

David Bisbal, "Mi princesa"

Espero que algún día estés orgullosa de mí, allá donde estés.

Índice general:

Índice de figuras:

Índice de tablas

Códigos de interés

Conceptos:

1. *Google Street View*: prestación de Google Maps y de Google Earth que proporciona panorámicas a nivel de calle (360 grados de movimiento horizontal y 290 grados de movimiento vertical), permitiendo a los usuarios ver partes de las ciudades seleccionadas y sus áreas metropolitanas circundantes.

2. Geolocalización: URL captada de la aplicación *Google Street View* que contiene todos los datos pertinentes para la elaboración del panorama y mapa de profundidad.

3. Cambio general estructural: Cambio que se etiquetará en este TFG como resultado de observa la diferencia entre dos secciones de momentos distintos pero misma posición. Puede ser cambios en las fachadas, reparaciones, construcción/demolición de edificios, señales de tráfico o una combinación de estas mismas.

4. Mapa de profundidad: Mapa que representa la profundidad de una imagen en una estructura planar, apoyada de un código de colores para la interpretación de esta tercera dimensión.

5. Síntesis de imágenes: Técnica de obtención de imágenes virtuales a partir del panorama y mapa de profundidad generados con la geolocalización.

6. *Keypoint*: Punto característico de una imagen capaz de ser fácilmente identificado y por tanto participar en la correspondencia de una imagen con otra similar.

7. Extracción de características: Técnica de obtención de *keypoints* a partir de una imagen.

8. Descripción de características: Técnica de asignación de características, tales como orientación o características de la región circundante entre otras, para caracterizar a un *keypoint*.

9. Detector de esquinas de Harris: Extractor de *keypoints* especializado en las intersecciones de bordes o contornos en una imagen, es decir, en esquinas.

10. SIFT (Scale Invariant Feature Transform): Técnica capaz de detectar y describir *keypoints* con invarianza ante transformaciones de la imagen. Cuenta con un vector de descripción de 124 elementos.

11. SURF (Speeded-Up Robust Features): Técnica capaz de detectar y describir *keypoints* mejorada respecto a SIFT, ya que al descartar la mayoría de los descriptores por no presentar, computacionalmente es mucho más rápido. Cuenta con un vector de descripción de 128 elementos.

12. *Matching*: Correspondencia entre los *keypoints* similares obtenidos de dos imágenes para su posterior alineamiento.

13. *Groundtruth*: Clasificación verdadera de una imagen. Constituye un mapa de segmentación de la imagen donde las regiones a detallar están coloreadas (especificado por su clase semántica) y el resto es un fondo homogéneo monocolor (generalmente blanco o negro).

14. RNA (Red Neuronal Artificial): Modelo computacional basado en un gran conjunto de unidades neuronales simples (neuronas artificiales), de forma aproximadamente análoga, el comportamiento observado en los axones de las neuronas en los cerebros biológicos.

15. CNN (Red Neuronal Convolucional): Tipo de red neuronal artificial especializada en la clasificación, segmentación o etiquetado (entre otras cosas) de imágenes.

16. FPR (False Positive Rate): Proporción de datos erróneos (negativos) que producen resultados positivos. En este caso corresponde a la proporción de no cambios reales que sin embargo la red etiqueta.

17. FNR (False Negative Rate): Proporción de datos buenos (positivos) que producen resultados negativos. En este caso corresponde a la proporción de cambios reales que no son etiquetados por la red.

18. TPR (True Positive Rate): Proporción de datos buenos (positivos) que producen resultados positivos. En este caso corresponde a la proporción de cambios reales que son etiquetados por la red.

19. TNR (True Negative Rate): Proporción de datos erróneos (negativos) que producen resultados negativos. En este caso corresponde a la proporción de no cambios reales que correctamente no han sido identificados por la red.

20. *Recall:* Proporción de muestras buenas obtenidas de un conjunto del conjunto de muestras totales.

21. Precision: Proporción de muestras buenas respecto del subconjunto muestreado.

Acrónimos

1. GSV: *Google Street View*

2. SIFT: Scale Invariant Feature Transform

3. SURF: Speeded-Up Robust Features

4. RANSAC: Random sample consensus

5. MSAC: M-estimator SAmple and Consensus

6. RNA: Red Neuronal Artificial

7. CNN: Convolutional Neural Network (Red Neuronal Convolucional)

8. DNN: Deconvolutional Neuronal Network (Red Neuronal Deconvolucional)

9. FPR: *False Positive Rate*

10. FNR: *False Negative Rate*

11. TPR: *True Positive Rate*

12. TNR: *True Negative Rate*

13. TFG: Trabajo de Fin de Grado

Resumen:

Este trabajo presenta una herramienta, desarrollada en Matlab, para la anotación de cambios a lo largo del tiempo sobre secciones de imágenes panorámicas de *Google Street View*. Para ello se aplican técnicas de cálculo de profundidad, síntesis de imágenes cambiando el punto de vista de la escena, extracción de características sobre las imágenes transformadas, alineamiento entre imágenes y etiquetado manual de sus diferencias, con objeto de crear una imagen patrón (*groundtruth*) de los cambios. La idea es alinear las imágenes correspondientes a un mismo lugar tomadas por el coche de *Google* en distinta fecha y desde una posición diferente, para poder apreciar mejor los cambios.

Utilizando esta herramienta, se ha creado una base de datos de cambios de 909 secciones útiles para entrenar una Red Neuronal Convolucional (CNN) que sea capaz de detectar los cambios de una forma automática. Para ello, se ha utilizado la red DNN (Deconvolutional Neural Network), desarrollada por el Dr. Pablo Fernández Alcantarilla [17] con quien se ha colaborado para la obtención de resultados.

El TFG incluye una comparativa del rendimiento de esta red respecto a otras propuestas del estado del arte, demostrando un comportamiento muy superior a los métodos existentes en la actualidad.

Conceptos clave: *Google Street View*, detección de cambios, red neuronal convolucional (CNN), *keypoint*, *matching*.

Abstract:

This work presents a tool, developed in Matlab, for the annotation of changes over time on sections of panoramic images of Google Street View. In order to do this, techniques of depth calculation, image synthesis (changing the point of view of the scene), extraction of features on the transformed images, alignment among images and manual labeling of their differences are applied, in order to create a pattern image (groundtruth) of the changes. The target is to align the images corresponding to the same place taken by the Google car on different date and from a different position, to better appreciate the changes.

Using this tool, a database of 909 useful sections has been created to train a Convolutional Neural Network (CNN) that is capable of detecting changes in an automatic way. For this purpose, a DNN network (Deconvolutional Neural Network), developed by Dr. Pablo Fernández Alcantarilla [17], has been used with whom I have collaborated to obtain results.

This works includes a comparison of the performance of this netowork with other proposals of the state-of-art, showing a really better behavior than existing methods today.

Key concepts: Google Street View, change detection, convolutional neural network (CNN), keypoints, matching.

Resumen extendido:

El ser humano se encuentra en un mundo cambiante, donde la tecnología avanzada a velocidad supersónica (especialmente desde la segunda mitad del siglo XIX hasta la actualidad), y por tanto el hombre debe adaptarse a ella para asegurar su supervivencia. Lo que antes se consideraba una tecnología de lujo para las clases altas de la sociedad, hoy en día todo el mundo tiene un *smarptphone* , desarrollados en apenas dos décadas, mejorando la vida de las personas (siempre que se usen convenientemente) y realizando tareas que antes sólo estaban al alcance de las computadoras de sobremesa.

Estos avances afectan a múltiples campos, entre ellos la tecnología. Hoy en día, el desarrollo de sistemas basados en visión artificial son una cuestión clave en el mundo de la robótica, y de forma más concreta, en el desarrollo de sistemas inteligentes de transporte. Esto es debido principalmente a que los sistemas basados en visión artificial (rama de la inteligencia artificial) están pensados para sustituir la acción humana y permitir la navegación autónoma de estos mismos, permitiendo descanso a la persona, ahorro en coste y por tanto eficiencia energética.

Sin embargo, para permitir la navegación autónoma, uno de los aspectos más importantes es el reconocimiento del entorno por parte del vehículo, básicamente para saber por donde va y que no ocurra una tragedia. La actualización y reconstrucción de mapas son dos de los aspectos con más auge a nivel mundial en el campo de la visión artificial, mayormente basado en redes neuronales para la detección de estos cambios estructurales. Por ejemplo, supóngase que un coche autónomo está conduciendo por la M30 dirección Madrid. Este, para saber si va bien deberá detectar su entorno. Sin embargo, uno de los principales problemas que tendrá es cuando se tenga que enfrentar a zonas con cambios estructurales que no estaban la última vez que pasó por allí. Es por eso que la actualización de los mapas usados en su localización es tremendamente útil.

Este trabajo está tiene como objetivo final la modificación de una red neuronal convolucional (CNN) ya existente que sea capaz de detectar y etiquetar (segmentación de la imagen) de forma eficiente los cambios existentes entre dos imágenes, aparentemente conteniendo la misma información. Dichas imágenes o secciones provendrán de imágenes panorámicas a su vez generadas a partir de geolocalizaciones de Google Street View.

La gran ventaja del uso de *Google Street View* en el ámbito de la reconstrucción de mapas es que, mientras que los trabajos previos siempre han usado bases de datos (imágenes en este caso) hechas por ellos mismos, mediante cámaras estáticas que captan la misma imagen, a la misma hora, consiguiendo por tanto un alineamiento perfecto (evidentemente debido a que el punto de vista es el mismo) y además que sólo sean notorios los cambios estructurales en vez de cambios en la iluminación, vegetación, etc…el trabajo presente dota a este estudio de reconstrucción de mapas un impulso sobresaliente: Uso de una herramienta global, gratuita y accesible por todo el mundo como Google Street View, para la captación de geolocalizaciones que permita ampliar de forma exponencial las bases de datos de entrenamiento y test para la red neuronal, mejorando por tanto su eficiencia. También debe tomarse en cuenta que *Street View*, al no pasar el coche por el mismo sitio siempre, presentará entre las secciones casi siempre desalineamiento, desajustes de luz, cambio de estación, vegetación, etc... aumentando la complejidad de la detección de cambios. Sin embargo, estos puntos desafiantes realmente suponen una ventaja porque obliga a los desarrolladores de la red (en este caso el Dr. Pablo Fernández Alcantarilla, ex miembro del grupo RobeSafe de la Universidad de Alcalá y actual investigador de visión artificial en la empresa iRobots) a mejorar la técnica y depurarla.

En **primer lugar**, con una duración estimada de tres meses, **se creó una aplicación en MatLab** (de una extensión aproximada de 4600 líneas distribuidas en 12 funciones, 5 de ellas interfaces, todas elaboradas por el autor de este TFG, obviando el código externo captado cuyas pretensiones eran parecidas) para, **a partir de enlaces de *Google Street View*, la obtención de sus imágenes panorámicas, mapas de profundidad, sintetización de vistas, extracción de características, matching entre pares de**

secciones y finalmente etiquetado con un groundtruth binario de los cambios generales estructurales presentes entre ambas. Estos conceptos serán el groso de la memoria, abarcando desde el capítulo 2 *Herramienta Google Street View* hasta el capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea*.

En **segundo lugar**, con una duración estimada de un mes, se buscó por todo el mundo un total **de 567 geolocalizaciones**, cada una presentando dos URLs para hacer un análisis temporal entre ambas (por tanto, 1134 URLs) en un proceso muy tedioso dados los parámetros de búsqueda que se debían seguir y se especificarán en este trabajo. Estas 559 geolocalizaciones se divieron en 354 para el entrenamiento de la red neuronal y 213 para el test, cada una pudiendo generar (como máximo) tres pares de secciones.

En **tercer lugar, procesamiento de las 354 geolocalizaciones** de la primera base de datos y etiquetado manual una a una, obteniendo 909 pares de secciones (con su respectivo groundtruth o etiquetado) de un máximo de 1062, por tanto presentando la herramienta un gran rendimiento. Para este proceso se tomó aproximadamente un mes.

En cuarto lugar, el Dr. Pablo Fernández Alcantarilla entrena la red neuronal mediante la modificación de la arquitectura de su DNN (red devoncolucional) presente en un trabajo suyo del mismo ámbito *Street-View change detection from Deconvolutional Networks* así como el procesamiento de los resultados obtenidos a partir de la herramienta creada.

El objetivo final de este trabajo es la mejora de una red neuronal convolucional, tanto en la posibilidad de crear una base de datos enorme gracias a la herramienta global de Street View, así como modificar su arquitectura, para mejorar la eficiencia en esta detección de cambios estructurales y futura reconstrucción de mapas.

Las principales herramienta usadas en este TFG han sido: Excel (para el almacenaje de las geolocalizaciones), *Google Street View* (para la búsqueda de las geolocalizaciones) y MatLab (para la creación de la aplicación que procesa los datos).

Capítulo 1

Introducción y estado del arte:

En este proyecto se propondrá un **sistema compacto para realizar la detección de cambios estructurales a partir de vistas sintéticas obtenidas de geolocalizaciones obtenidas por medio de la herramienta global de *Google Street View*.** Esta detección y autoetiquetado se llevarán a cabo mediante una red neuronal deconvolucional (DNN) variante de una red neuronal convolucional (CNN) utilizada en procesamiento de imágenes, principalmente en el ámbito de la segmentación y etiquetado de imágenes.

Este sistema está justificado por la necesidad de actualizaciones cada vez más frecuentes y eficientes en los mapas a gran escala usados en la navegación autónoma de los vehículos. Un ejemplo de coche autónomo desarrollado por Google es el siguiente:

Figura 1.1: Coche autónomo de Google Street View

El método de entrenamiento de la red neuronal para el autoetiquetado de los cambios estructurales encadena, a partir del uso de dos APIs proporcionadas por *Google Street View*, la reconstrucción del panorama asociado a una **geolocalización a pie de calle** (es decir, a partir de una URL que se almacenará en una base de datos de Excel), la **generación de su mapa de profundidad, la síntesis de imágenes, la extracción de características, el alineamiento y finalmente el etiquetado manual elaborando un groundtruth binario** que presenta el siguiente formato:

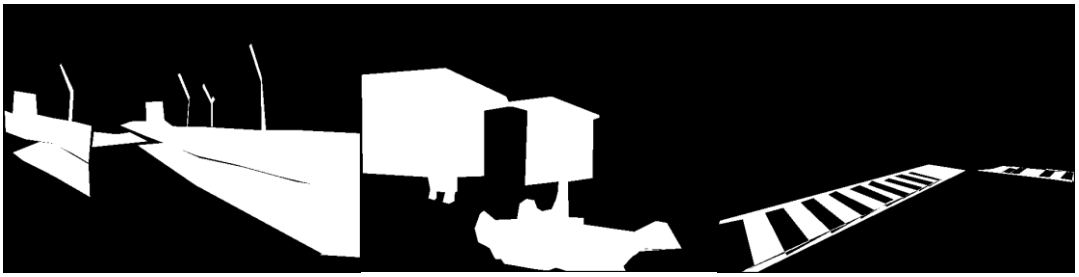


Figura 1.2: Ejemplo de Groundtruth

La red neuronal es entrenada a partir de un dataset creado para este trabajo, con un orden de magnitud más grande que los datasets existentes, basado en el concepto de **gratuito, accesible por todo el mundo y portable de *Google Street View*,** permitiendo crear grandes basos de datos que además

contienen datos desafiantes debido a las variaciones de apariencia debido a los cambios de estaciones e iluminación.

Respecto al estado del arte, en lo que a detección de cambios estructurales en zonas urbanas se refiere, otros autores pudieron comprobar como a **escala grande** (de tamaño ciudad) y sobre **periodos de tiempo que van desde estaciones a años**, **el paisaje visual urbano es un entorno altamente dinámico**, con muchos cambios generales estructurales, tales como reparaciones, demoliciones de edificios, construcciones de nuevos, señales de tráfico o cambios en las fachadas **[17] [18].**

Desde el punto de vista de un sistema de conducción autónoma, es **esencial mantener un mapa actualizado**, ya que los panoramas son susceptibles a cambiar por medio de estos cambios estructurales generales. Cuanta más alta es la frecuencia con que el mapa se actualiza, más robusta será la navegación del sistema, y más se plantea que sea mejorado en un futuro. Por ejemplo, si el coche anterior autónomo de *Street View* (figura 1) tuviera un mecanismo (en este caso la DNN) capaz de detectar los cambios entre dos situaciones temporales para una misma geolocalización, en vez de tomar siempre todos los datos de un entorno y procesarlos, esto supondría un inmenso ahorro tanto en tiempo como en recursos.

Los mapas actualizados no sólo son útiles para temas de navegación autónoma robusta, sino que también pueden ser muy útiles en otras aplicaciones tales como monitorizar la disponibilidad de espacio de un parking o la disponibilidad de cierre de carreteras (o derivados) debido a trabajos en la misma.

Un ejemplo de los resultados obtenidos en este trabajo:

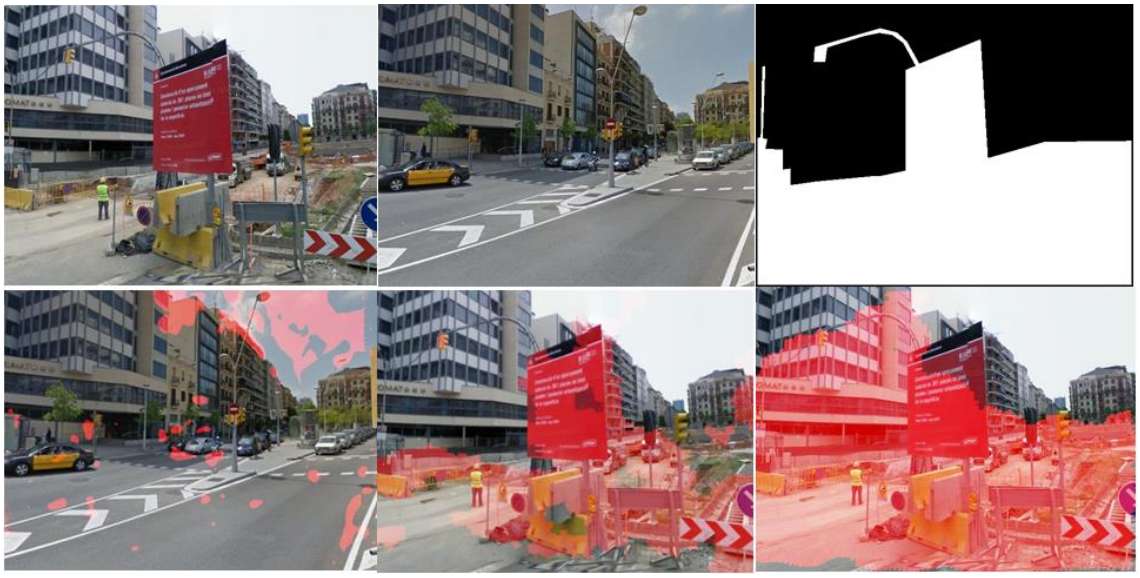


Figura 1.3: Ejemplo de resultados

En el ejemplo que se muestra en la figura [1.3]: La primera fila, de izquierda a la imagen de la derecha, es la imagen sintética del panorama 1, imagen sintética del panorama 2 con corrección de offset y *groundtruth*. En la segunda fila, un ejemplo de autoetiquetado basado en el descriptor *handcrafted* DAISY [21], el etiquetado obtenido mediante una red neuronal preentrenada basada en la superpixelación **[18]** y el etiquetado mediante una red neuronal CDNet elaborada para este proyecto.

Las imágenes del panorama 1 y 2 exhiben cierta variabilidad y pueden incluir cambios de interés, como los mencionados cambios estructurales, pero también a partir de ruidos generados por el cambio del punto de vista, las condiciones ambientales (estaciones, tiempo atmosférico) o cambios dinámicos debido a los vehículos, transeúntes y vegetación.

Inspirado en el reciente y enorme éxisto de las redes neuronales convolucionale profundas (*deep Convolutional Neural Networks*, CNNs) en el aprendizaje invariante de las imágenes para las tareas de procesado de estas imágenes a gran escala, para este trabajo se adopta una profunda arquitectura deconvolucional basada en CNNs para la tarea de detección de cambios estructurales. La deconvolución se puede definir como el conjunto de operaciones matemáticas empleadas en la restauración de señales, imagénes, etc…(es decir, un conjunto discrito y finito de datos) para recuperar datos que han sido degradados (en este caso suavizados) mediante convolución. La arquitectura diseñada para este trabajo ha sido la siguiente:

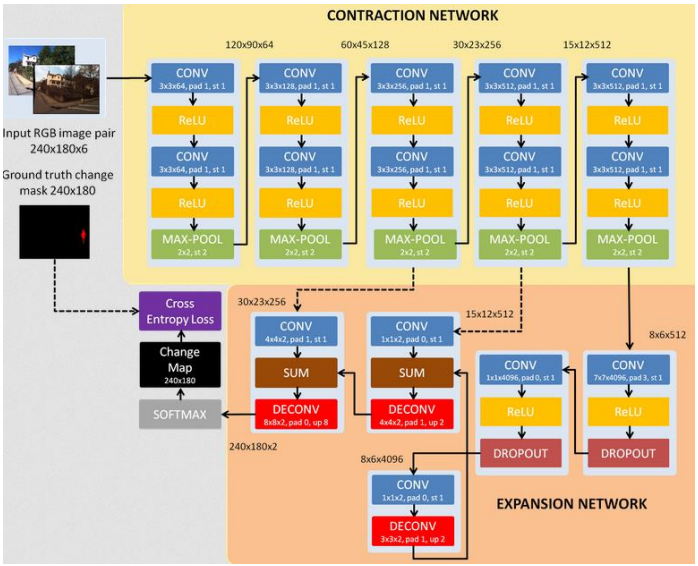


Figura 1.4: Arquitectura de la DNN

Como se puede ver en la figura [1.4], la arquitectura está basada en una fase de contracción (con bloques de convolución, regulador no lineal ReLU y proceso de reducción de muestreo o *maxpooling*) y una fase de expansión (con bloques similares añadiéndole sumatorios y *dropouts*). En el capítulo 6 (“CNN y resultados finales”) se retomará de forma más detallada el concepto de redes neuronales y la aplicación en este TFG.

Un flujograma del proceso total se muestra a continuación:

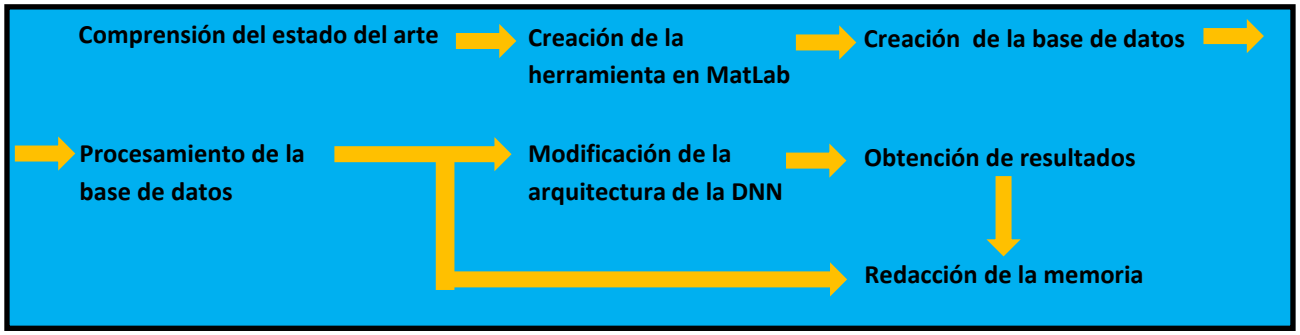


Figura 1.5: Flujograma del proceso total

Capítulo 2

Herramienta Google Street View:

La base de todo este Trabajo de Fin de Grado está cimentada sobre la **herramienta *Google Street View***. En ella se buscarán las geolocalizaciones que posteriormente servirán de análisis para obtener los paquetes requeridos (pares de imágenes de primer y segundo panorama y el etiquetado de diferencias) para el entrenamiento de la red. Para poder definir claramente este soporte, primero debe ser descrita la compañía a la que pertenece (Google) así como el ámbito en el que trabaja.

2.1. Compañía Google:

Google es una compañía principal subsidiaria de la multinacional estadounidense ***Alphabet Inc.*** La empresa *Alphabet Inc.* es una empresa multinacional estadounidense cuyo principal cometido es el desarrollo de productos y servicios relacionados con Internet, software, dispositivos electrónicos y otras tecnologías. También incluye a empresas de sectores como la biotecnología, salud, telecomunicaciones y la domótica.

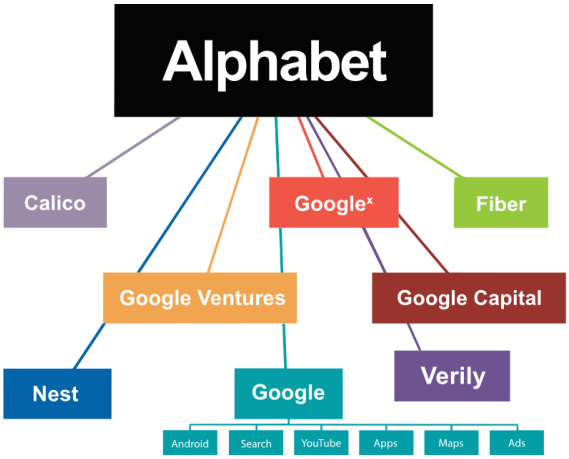


Figura 2.1: Organigrama de *Alphabet Inc.*

Las principales funciones de Google es el desarrollo de productos y servicios tales como el motor de búsqueda de Google, el sitio web YouTube, servicios como Google Maps o Gmail, entre muchos otros.

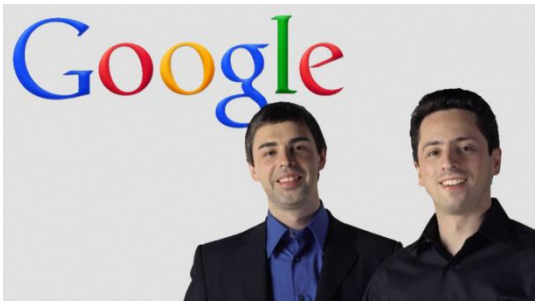


Figura 2.2: Fundadores de Google, a la izquierda **Larry Page**, a la derecha **Sergey Brin**

Fundada en 1998, pasa a denominarse Google Inc., estrenándose el 27 de Septiembre del mismo año su motor de búsqueda (fecha que se considera su aniversario). La compañía fue adquiriendo otras empresas a una velocidad jamás vistas, por cifras astronómicas, tales como los 1.000 millones de dólares

pagados por el servicio de correo electrónico Gmail en 2004 o los 1.650 millones de dólares pagados por la plataforma de vídeos YouTube. En 2007, Google se convierte la marca más valiosa del mundo, valorada en 66.000 millones de dólares, superando a empresas gigantes como Microsoft, General Electric o CocaCola. En este mismo año también se lanza Google Street View, corazón de este TFG, el cual resulta como una variante a pie de calle del servicio Google Maps. No será hasta 2015 cuando Google Inc. pase a ser principal subsidiaria de la matriz *Alphabet Inc.*

La ubicación de las principales instalaciones de la compañía Google Inc. (principal subsidiaria) es Mountain View (California), y recibe el nombre de ***GooglePlex*.**

Figura 2.3: *GooglePlex*

Como principal subsidiaria dentro del conglomerado *Alphabet Inc*., su especialización son los productos y servicios relacionados con Internet, software, dispositivos electrónicos y otras tecnologías. El término subsidiario indica que Google Inc. sustituye a la parte principal (en este caso *Alphabet Inc*.) en caso de que sea necesario, es decir, tiene la responsabilidad principal en sus acciones en defecto de la multinacional. Es decir, la principal tarea de *Alphabet Inc*. es proporcionar un mejor orden y estabilidad al conglomerado de empresas, pero cada una, y especialmente Google son responsables de sus acciones.

El principal producto de *Google Inc.* es el motor búsqueda de contenido en Internet del mismo nombre, aunque también ofrece otros productos y servicios como el correo electrónico llamado Gmail, sus servicios de mapas *Google Maps, Google Street View* y *Google Earth*, el sitio web de vídeos YouTube y otras utilidades web como *Google Libres* o *Google Noticias*, *Google Chrome* y la red social *Google+*. Por otra parte, lidera el desarrollo del sistema operativo basado en *Linux, Android*, orientado a *smartphones*, tabletas, televisores, automóviles y gafas de realidad aumentada (*Google Glass*).

Para este TFG las principales utilidades será el servicio de mapas a nivel global, es decir, el compendio que agrupa *Google Maps*, *Google Street View* y *Google Earth*. Este TFG está íntegramente ligado al servicio a pie de calle global de Google Street View, que se definirá en el siguiente apartado.

2.2. Google Street View:

Tras tratar los principales productos de Google, ahora se detallará la principal herramienta usada en este TFG (a parte de MatLab), esto es, ***Google Street View***.

Google Street View es una herramienta derivada de *Google Maps* y *Google Earth* que proporciona **panorámicas a nivel de calle**, permitiendo a los usuarios ver zonas de las ciudades seleccionadas y las áreas metropolitanas circundantes, siempre y cuando se haya registrado información de dicha zona.

- *Google Maps* representa el "padre" de *Street View* puesto que en sí es el servidor de aplicaciones de mapas en la web de *Google Inc*. Este servicio ofrece imágenes de mapas desplazables, fotografías por satélite del mundo (lo que representa a *Google Earth*) e incluso rutas entre diferencias ubicaciones o imágenes a pie de calle con *Street View*.

- *Google Earth* es un programa informático (una aplicación disponible para computadoras, *tablets* y *smartphones*) que muestra un globo virtual que permite visualizar la cartografía del planeta en base a fotografía satelital, fotografías aéreas y modelos creados por computadoras (artificiales por tanto). Sin embargo, no puede decirse que *Google Maps* sea el padre de *Google Earth*, a pesar de que hoy en día el compendio de Maps agrupe tanto a Earth como Street View, puesto que el programa originalmente fue financiado por la CIA estadounidense (Agencia Central de Inteligencia) bajo el nombre *EarthViewer 3D*, siendo adquirida por *Google Inc.* en 2004.

En resumen, dentro del compendio de servicio de mapas a nivel global, todos se encuentran relacionados pero en primer lugar se encuentra Google Maps.

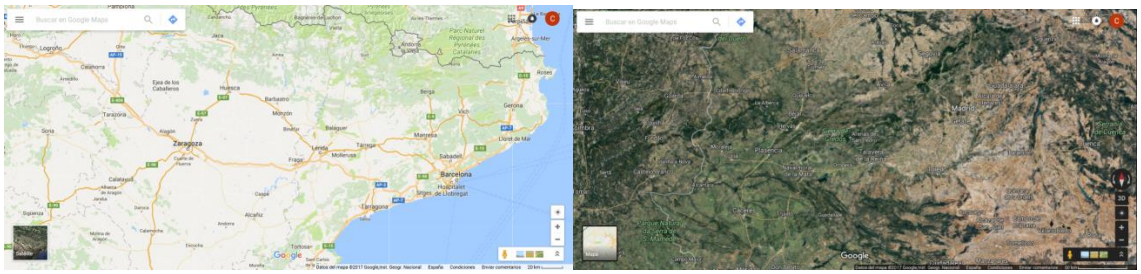


Figura 2.4: *Google Maps* en modo mapa (izquierda) y en modo satelital terrestre (derecha)

Google Street View (de ahora en adelante se denominará GSV para facilitar la lectura) se introdujo en primer lugar en Estados Unidos el 25 de Mayo de 2007. Cuando se lanzó el servicio, tan sólo cinco ciudades estadounidenses estaban incluidas (Nueva York, Los Ángeles, Washington D.C., Detroit y Chicago). Desde entonces, GSV se ha expandido a 31 países europeos, 7 latinoamericanos, 17 asiáticos, 5 africanos y la Antártida.

Algunas de sus principales características son:

- Las panorámicas a pie de calle presentan, desde la misma geolocalización, un recorrido de **360 grados en horizontal y 290 grados en vertical.**
- Se puede **navegar a través de estas imágenes** utilizando los cursores del teclado o usando el ratón. Además, en mayo de 2009 se introdujo una novedad de navegación en la aplicación, basada en los datos proporcionados por la tecnología láser, que permite una navegación más rápida a lo largo del recorrido.
- Para la toma de fotografías se tiene en cuenta el **clima, horario y temperatura**, de manera que se tomen fotografías parejas. Al final de este capítulo se describirá de forma detallada cuales han sido los principales parámetros para la recolección de URL, pero cabe resaltar que la tendencia es que los panoramas generados presenten el mismo tiempo atmosférico (nublado, soleado, etc…), bajo número de transeúntes, vehículos, etc…. Para la consecución de efectivos mapas de profundidad y posterior etiquetado.
- Toma de imágenes en **años/meses distintos** para una misma geolocalización (es decir, igual o casi igual latitud y longitud) que es de gran utilidad para observar la evolución de las ciudades. Esta característica será el corazón de análisis de este trabajo.

- Para asegurar que la experiencia de los usuarios que ven las imágenes de *Google Maps* (tanto Earth, Maps como tal y Street View) sea positiva y útil, desde Mayo de 2008 en Manhattan (Nueva York), en un esfuerzo de la compañía por respetar tanto las obligaciones morales como las legales que este este servicio implica, la compañía creó un **sistema de identificación y emborronamiento de caras.**

  Estas políticas de desenfoque no sólo se aplican a caras, sino también a **matrículas de vehículos** móviles en todas las imágenes documentadas por Street View. Sin embargo, a parte de la innovadora tecnología de reconocimiento de caras y matrículas desarrollado por Google Inc., la participación activa del ciudadano es muy importante: Si se ve una cara o matrícula que es necesario desenfocar más, o incluso si una persona quiere que se desenfoque su casa o coche por completo, basta con enviar una solicitud a través de GSV y reportar el problema.

  No obstante, debe tenerse en cuenta que, una vez que Google desenfoca una imagen, su efecto es permanente. En otras palabras, si se envía una solicitud para que se desenfoque una casa privada en las imágenes de Street View de Google, también se desenfocarán todas las imágenes del historial y las imágenes que se hagan en el futuro de dicha casa.

Figura 2.5: A la izquierda, servicio a pie de calle con máquina de tiempo encuadrada en rojo. A la derecha, máquina del tiempo para la elección de pasada

Precisamente este pequeño recuadro es el corazón de este TFG puesto que será la **máquina del tiempo** que permitirá navegar entre distintos años para una misma calle, obteniendo las URLs pertinentes y almacenándolas en una base de datos.

- **Las imágenes de GSV no se ofrecen en tiempo real.** El recorrido por las calles, características, tiempo atmosférico, etc… sólo muestran el día en el cual los vehículos y/o personas externos tomaron las imágenes de dicha ubicación.

  Por otra parte, el contenido visible, tal y como se ha dicho, debe atender a las políticas de privacidad, así como posibles quejas, por lo que tarda varios meses en procesarlas. Es por ello que es imposible ver un recorrido del mismo mes en el cual se está redactando este documento.

  Por último, tal y como se ha dicho anteriormente, en aquella ubicaciones donde se han hecho dos o más pasadas (es decir, se han recopilado dos o más imágenes, es posible activar la función “Máquina del tiempo” donde navegar a través de las distintas pasadas.

- Las imágenes que hay en GSV proceden de dos fuentes: Bien las tomadas por los distintos **vehículos de Google**, bien las **proporcionadas por la ciudadanía** (el nombre de atribución y el icono muestran si la imagen fue realizada por Google o en su defecto por un autor externo en cuyo caso las imágenes podrían estar protegidas por derecho de autor a menos que haya aceptado transferir la propiedad de las imágenes a un tercero).

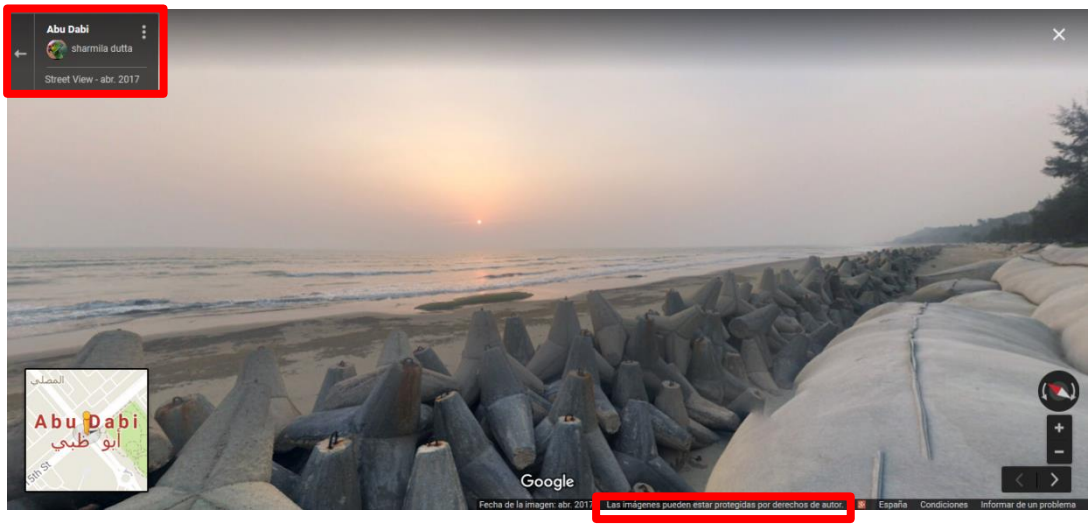


Figura 2.6: Geolocalización de autor particular

Figura 2.7: Geolocalización de © 2017 Google

Para el caso en que el usuario quiera proporcionar imágenes a la aplicación, éste debe contar con algún accesorio ofertado por GSV en su propia página (por ejemplo una cámara 360 º conectada a la aplicación de GSV, montada sobre un casco), respetando la iluminación y espaciamiento adecuados, tal y como se puede ver en la figura [2.8].

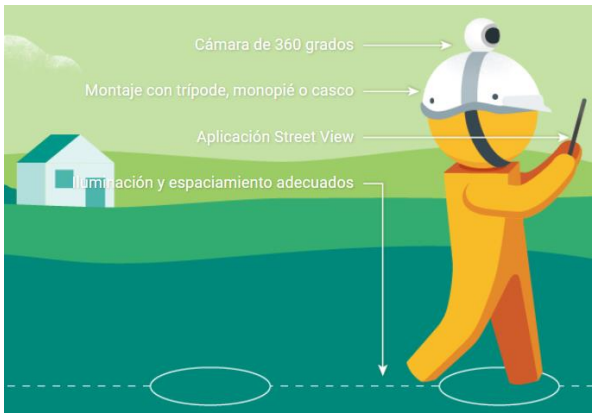


Figura 2.8: Usuario captando geolocalizaciones

Por último, antes de centrar la lectura en la API de GSV usada en este TFG, parámetros de la URL, almacenaje, etc…, en el apartado "Conocer" de la página oficial [1] se puede observar la superficie mundial (en azul más oscuro) ya explorada:

Figura 2.9: En azul, actuales territorios del planeta con servicio de GSV

Debajo de dicho mapa se puede seleccionar el país y ver los compromisos que tienen con cada región los exploradores de GSV, ya sea para hacer su primera pasada (primera toma de imágenes fotográficas) o para repetir. Por ejemplo, en el caso de España:

Region	District	Time
Galicia	La Coruña, Lugo, Pontevedra, Orense	Abril 2017 – Octubre 2017
Principado de Asturias	Asturias	Abril 2017 – Octubre 2017
Cantabria	Santander	Abril 2017 – Octubre 2017
País Vasco	Vizcaya, Guipúzcoa, Álava	Abril 2017 – Octubre 2017
Comunidad Foral de Navarra	Navarra	Abril 2017 – Octubre 2017
La Rioja	La Rioja	Abril 2017 – Octubre 2017
Aragón	Huesca, Zaragoza, Teruel	Abril 2017 – Octubre 2017

Figura 2.10: Compromisos de GSV con España en 2017

2.3. Aplicación de Google Street View para el proyecto:

Llegados a este punto, es trascendental diferenciar una cosa: Para la recolección de las URLs no se ha utilizado la API de GSV [2] sino que se han extraído directamente desde el servicio de GSV implementado dentro de *Google Maps*.

OBS: Una **API** (acrónimo en inglés de *Application Programming Interface*) es un conjunto de subrutinas, funciones y procedimientos o métodos en la programación orientada a objetos que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción. Se suele representar con una ventana, iconos en la pantalla, etc... para que el usuario interactúe.

Mientras que la *Google Street View Image API* permite al usuario insertar una imagen panorámica estática de Street View (no interactiva) o una miniatura en una página web sin usar JavaScript, en este TFG directamente se ha navegado por *Google Maps* (y por tanto dentro en su servicio a pie de calle) extrayendo las URL que Google ha ido colgando en la red.

2.3.1. Formato de la URL:

A pesar de que el formato las URLs ha ido evolucionando con el tiempo, las URLs utilizadas en este TFG presentan el siguiente formato:

https://www.google.es/maps/@52.4108303,-1.5110647,3a,75y,291.12h,97.69t/data=!3m7!1e1!3m5!1sS9jTGoLkveFGvWvydR9EUQ!2e0!5s20121001T000000!7i13312!8i6656

El análisis de estos datos es trascendental para un uso efectivo de estos mismos. En primer lugar, se va a proceder a diseccionar este ejemplo de URL, definiendo así los parámetros que representan:

1. En primer lugar se puede observar que la **URL pertenece al servicio de Maps dentro de Google**. No se hace referencia a Street View puesto que este servicio está embebido dentro de Google Maps, aunque realmente este formato de URL, debido a los parámetros de a continuación, es característico del servicio de a pie de calle.

2. Desde la barra / que se encuentra la derecha "maps" hasta la siguiente barra / se encuentra la **información que representa la geolocalización de la URL**, es decir, donde se encuentra y como está orientada.

- **Los dos primeros números representan la localización en formato decimal, siendo el primer valor la latitud y el segundo lugar la longitud** (aunque también podría verse una cadena de texto, por ejemplo Chagrin Falls, OH). En caso de que se estuviera usando la API web de GSV para insertar una imagen, al proporcionar una cadena de texto acerca de la localización, la API buscaría la panorámica anterior capturada más cerca de dicha ubicación (para construir así la máquina del tiempo). **En caso de especificar un valor de latitud/longitud, el servicio busca en un radio de 50 metros la fotografía más cercana a esa ubicación.**
- La conclusión que se debe sacar de que exista la posibilidad de tomar fotografías en posiciones ligeramente diferentes es que la localización que el usuario intente integrar en la web se integre con otra panorámica cuando se actualicen las imágenes. Este error ha aparecido repetidas veces en el TFG (para un mismo valor de latitud/longitud, por ejemplo en una intersección de varias calles, la referencia de la calle no era la misma, por tanto a la hora de elaborar el panorama con el código a disposición la correspondencia no era correcta). Se volverá a hacer referencia a este error en el [capítulo 2] *Elaboración de la imagen panorámica y mapa de profundidad* donde se trata la elaboración del panorama.

A continuación se definen brevemente los conceptos de latitud y longitud:

- **Latitud de un punto:** Medida del ángulo formado por el plano ecuatorial con la línea que une a este punto al centro de la Tierra. Evidentemente el valor de 0 º es la referencia y representa al plano ecuatorial o Ecuador (Plano perpendicular al eje de rotación de un planeta y que pasa por su centro. Divide la superficie del planeta en dos partes: el hemisferio norte y el hemisferio sur).
- **Longitud de un punto:** Medida del ángulo formado por el semiplano del eje de la Tierra que pasa por el meridiano de Greenwich y el semiplano del eje de la Tierra que pasa por el punto. A diferencia de la latitud, la longitud no cuenta con una referencia natural como es el Ecuador, sino que depende del meridiano 0 o de Greenwich que corresponde a 0 º de longitud, adoptado en 1884 y que pasa por el Real Observatorio de Greenwich a las afueras de Londres.

  Las reglas adoptadas, y que se cumplen dentro de las URLs extraídas, son:

  A. La latitud y longitud presentan formato decimal y no únicamente entero, lo que aumenta la precisión al ubicar la localización.

B. El rango de la latitud es [-90 º, 90 º] siendo valores negativos para el Hemisferio Norte, positivos para el Hemisferio Sur y 0 º para el Ecuador.

C. El rango de la longitud es [-180 º, 180 º] siendo valores negativos al Oeste del meridiano de Greenwich, positivos al Este y 0 º este mismo.

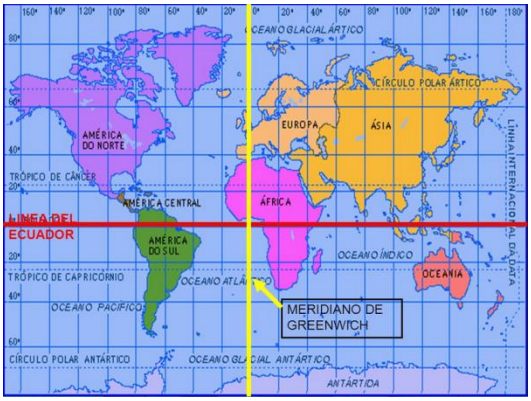


Figura 2.11: Latitud y longitud en un mapa plano

Si dentro de una misma localización se manipulan los controles mostrados en la figura 2.12:

Figura 2.12: Dentro de la elipse roja se distingue un imán que al pulsarlo redirige al usuario al norte y dos botones "+" y "-" para hacer zoom

Con el imán se modificaría el *heading angle* (se explicará a continuación), y con los botones +/- el *Field of View*. En caso de que el usuario rote la imagen con el ratón pinchado, tampoco se modificaría la ubicación, sino que lo harían los ángulos *pitch* y *heading*.

3. El siguiente **parámetro (3a) es un curioso zoom** que únicamente se modifica cuando el usuario no está en el servicio a pie de calle sino en el modo tridimensional viendo la ciudad. Supóngase que el usuario está navegando por una localización de la ciudad de Coventry usando el siguiente enlace:

https://www.google.es/maps/@52.4108303,-1.5110647,3a,75y,291.12h,97.69t/data=!3m7!1e1!3m5!1sS9jTGoLkveFGvWvydR9EUQ!2e0!5s20121001T000000!7i13312!8i6656

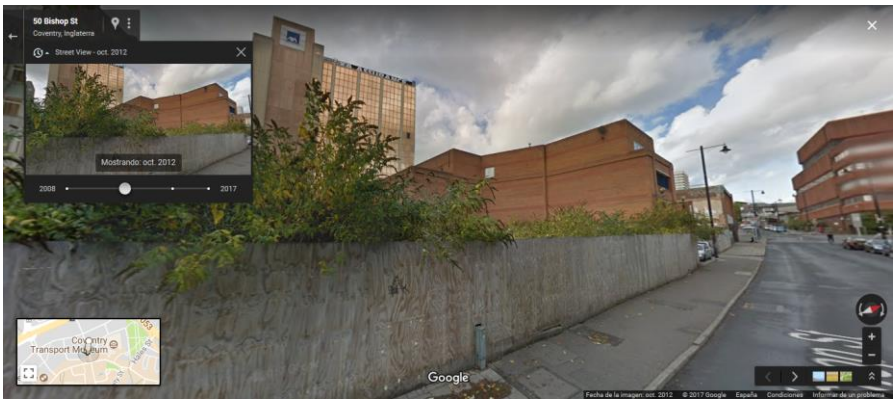


Figura 2.13: Ejemplo de geolocalización original

Si se acerca, **bien con los botones "+/-", bien con la ruleta del ratón, dicho zoom no se ve modificado** (de hecho insertando un número distinto de 3, al presionar *Enter* vuelve a 3. Entonces, **¿Qué se modifica? → Los valores del Field of View** (valor asociado a la letra y, en este caso 75y), ***Heading angle*** (valor asociado a la letra h, en este caso 291.12h) **y el *Pitch angle*** (valor asociado a la letra t, en este caso 97.69t) de los cuales se hablará a continuación.

Esta afirmación se puede comprobar dando zoom hacia delante:

https://www.google.es/maps/@52.4108303,-1.5110647,3a,50.1y,298.53h,92.75t/data=!3m7!1e1!3m5!1sS9jTGoLkveFGvWvydR9EUQ!2e0!5s20121001T000000!7i13312!8i6656

Figura 2.14: Ejemplo de geolocalización con zoom "+"

Se puede ver cómo se han modificado los valores anteriores, porque el **campo de visión es menor** (no el tamaño de la pantalla evidentemente, sino que lo que se ve ahora es un zoom con respecto a lo que había antes), lo que implica directamente que **el nuevo Norte de la imagen se ha modificado (*Heading angle*) así como la inclinación de cámara (*Pitch angle*).**

Por otra parte, es curioso comprobar que basta con dos clicks en el botón menos (o en su defecto dos pequeños movimiento de la ruleta del ratón) para salir del servicio de pie de calle a la imagen satelital. **Cuando el usuario se ubica en una localización sin realizar ningún movimiento después, siempre hay dos valores constante por defecto → 3a, 75h.**

Una vez con el botón menos resulta en 3a,90h (como se ha dicho el 3a no se modifica). Como ahora el campo de visión es mayor, resulta en 90h. Tras introducir otra vez:

https://www.google.es/maps/@52.4104076,-1.5093274,127a,35y,291.74h,45t/data=!3m1!1e3

Figura 2.15: Imagen satelital tridimensional original

La nueva URL es mucho más corta a la hora de identificar la geolocalización en el apartado de data. La latitud y la longitud se han modificado (dado que la posición ahora es la del avión, satélite, etc… que haya computarizado esta superficie 3D).

Se puede observar que el ***Field of View* ha pasado a ser 35h, valor que no se modificará mientras se esté en la visión 3D satelital.** En cambio, el zoom que antes era constante ahora ha pasado a 127a, a diferencia del 3a constante que había antes. Si se acerca o se aleja, todos los parámetros se modifican (incluidas latitud y longitud) a excepción del *Field of View*, como se observa en la siguiente localización:

https://www.google.es/maps/@52.4094643,-1.5061724,361a,35y,290.77h,44.88t/data=!3m1!1e3

Figura 2.16: Imagen satelital tridimensional alejada

En este caso el zoom es de 361, en vez de 127, por lo tanto es inversamente proporcional → A mayor número, más alejado está de la superficie de la Tierra.

Esta regla del zoom también se cumple si en vez de tener una visión tridimensional de la ciudad se tiene una visión en dos dimensiones:

https://www.google.es/maps/@52.4091002,-1.5121676,1212a,35y,290.77h/data=!3m1!1e3

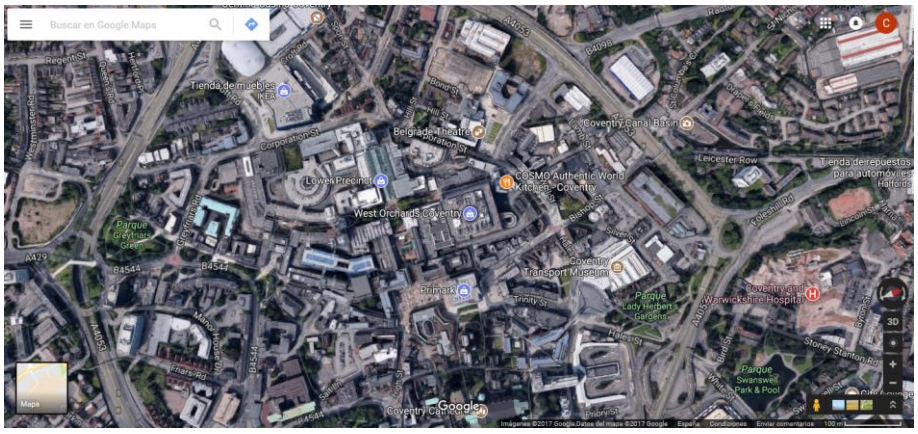


Figura 2.17: Imagen satelital bidimensional

A excepción de que el parámetro del *Pitch angle* es suprimido puesto que ahora no existe ninguna inclinación de la cámara.

4. El **cuarto parámetro** de esta lista de valores es el ***Field of View***, también llamado en español **Campo de Visión o Campo de Perspectiva.** Dentro de URL es el **valor que acompaña a la y.**

Field of View (FOV): También llamado campo de visión, es la extensión del mundo observable que es visto por una lente (ya sea de una cámara, ojo humano, etc…) para un momento dado. En el caso de instrumentos ópticos o sensores es el ángulo sólido a través del cual se puede llegar a percibir la radiación electromagnética.

Aunque los términos Ángulo de Visión (AOV) y Campo de Visión (FOV), son totalmente equivalentes, dado que realmente esa extensión del mundo observable corresponde a un ángulo, es más correcto en términos fotográficos y de procesamiento de imágenes (*Image Processing*), precisamente las dos bases de este TFG, el término AOV. A pesar de ello, **en el diseño del código de MatLab se hace como referencia al FOV**, pero no es más que nomenclatura puesto que realmente el concepto del *Field of View* es una desambiguación del AOV cuando no se está hablando en términos fotográficos o de procesamiento de imágenes.

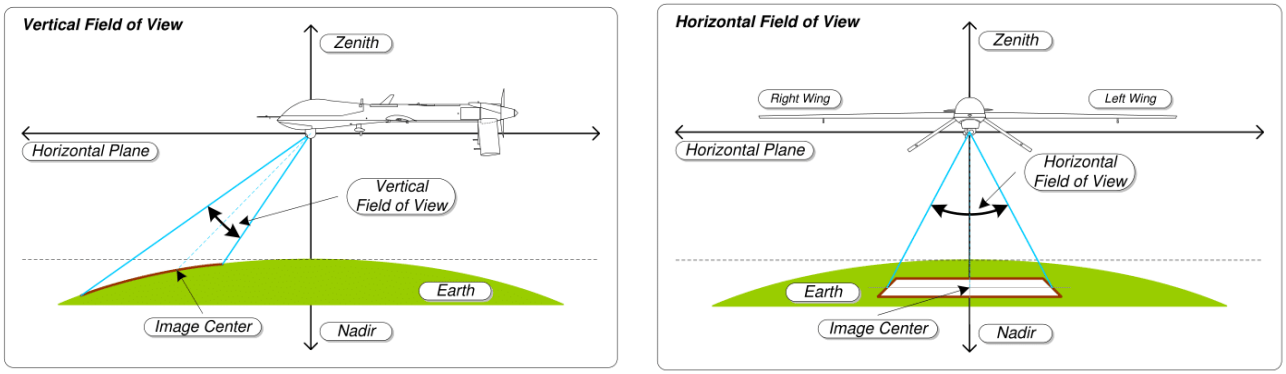


Figura 2.18: FOV vertical (izquierda) ; FOV horizontal (derecha)

Por tanto, el **Ángulo de Visión (AOV)** es la extensión angular de una escena dada representada por una cámara (como se puede ver la definición es equivalente al FOV).

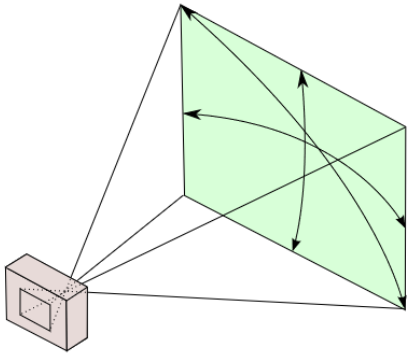


Figura 2.19: Proyección del ángulo de visión

2.3.1.1. Cálculo del ángulo de visión de una cámara:

Para modelos de proyección de cámaras sencillas (*pinhole* [22]), la distancia focal efectiva (distancia entre el centro óptico de la lente y el foco o punto focal) y las dimensiones de la imagen definen completamente el AOV. El AOV puede ser medido horizontalmente (de izquierda a derecha de la imagen), verticalmente (de arriba abajo de la imagen) o incluso diagonalmente (desde una esquina del marco hasta la opuesta), tal y como se ha visto en la figura 2.18.

La fórmula resultante es:

$$\alpha = 2 \cdot \operatorname{arctg}\left(\frac{d}{2f}\right) \tag{2.1}$$

Siendo α el campo de visión en grados sexagesimales, d la distancia en la dirección deseada (vertical, horizontal, diagonal) en mm y f la distancia focal, también en mm.

Muchos instrumentos ópticos, en particular los prismáticos, caracterizan su FOV con una de estas dos maneras: FOV angular (especificado en grados) o FOV linear (especificado en un cociente de longitudes).

Usando la aproximación de ángulos pequeños:

$$\operatorname{sen}(\alpha) \approx \alpha \quad ; \quad \cos(\alpha) \approx 1 - \frac{\alpha^2}{2} \quad ; \quad \operatorname{tg}(\alpha) \approx \alpha \tag{2.2}$$

Se pueden obtener las siguientes fórmulas:

$$A \approx \frac{360º}{2\pi} \cdot \frac{M}{1000} \quad ; \quad M \approx 1000 \cdot \frac{2\pi \cdot A}{360º} \tag{2.3}$$

Siendo M el FOV linear (expresando en mm) y A el FOV angular (expresado en grados).

Específicamente dentro de este caso, el **valor asociado a la letra "y" representa el FOV horizontal** de la imagen (expresado en grados) con un valor máximo permitido de 120 º. Cuanto más se presione el botón "+" dentro de la herramienta, mayor será el nivel de zoom y menor el valor del FOV.

5. El siguiente parámetro a definir es el **valor que acompaña a la letra h en la URL, este es el *Heading angle.***

Este ángulo indica la orientación de la cámara según la brújula, es decir, es el ángulo entre la dirección a la que apunta la "punta" o "nariz" de un objeto con una dirección de referencia (por ejemplo, el

Norte). En la figura [22], el *heading angle* correspondería al ángulo existente entre el vector 2, esto es, punta de la barca, hacia 1, norte geográfico.

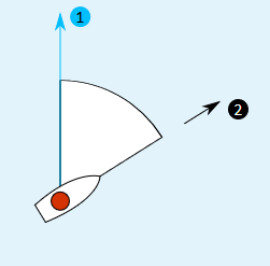


Figura 2.20: *Heading angle*

Los valores aceptados van de 0 º a 360 º(ambos valores indican el Norte, 90 indica el Este y 180 el Sur, es decir, presenta un sentido horario).

En caso de insertar una imagen con la API de Street View y no especificar dicha orientación, se calcularía un valor que orientará la cámara hacia la localización (latitud y longitud) especificada desde el punto en el que se tomó la fotografía más cercana.

Este ángulo puede notarse, perfectamente cuando se rota la vista dentro del servicio de Street View con el ratón, **observando como aumenta hacia 360 º si se rota hacia la derecha o disminuye a 0 º (solapándose con 360 º el valor) si se rota hacia la izquierda.**

6. El último ángulo a definir es el ***Pitch angle* o ángulo de inclinación de la cámara**. El *Pitch angle*, con un valor predeterminado de 90 º, **especifica el ángulo superior o inferior de la cámara con respecto al vehículo usado por Street View**. Dentro de la URL es el **valor que acompaña a la letra t.** Generalmente, aunque no siempre, corresponde con el plano horizontal, correspondiendo valores de:

- 90 hasta 179 grados (valor máximo apuntando directamente al cielo) a un ángulo ascendente.

- 90 grados hasta 1 grado (apuntando al suelo) corresponden a un ángulo descendente.

Ejemplo de ángulo máximo (cielo) [2.21]:

https://www.google.es/maps/@39.4480215,-2.6718854,3a,75y,143.54h,**179t**/data=!3m6!1e1!3m4!1sX2Cw4jKn2KPydBEV8_6kNg!2e0!7i13312!8i6656

Ejemplo de ángulo mínimo (suelo) 2.21:

https://www.google.es/maps/@39.4480215,-2.6718854,3a,75y,64.15h,**1t**/data=!3m6!1e1!3m4!1sX2Cw4jKn2KPydBEV8_6kNg!2e0!7i13312!8i6656

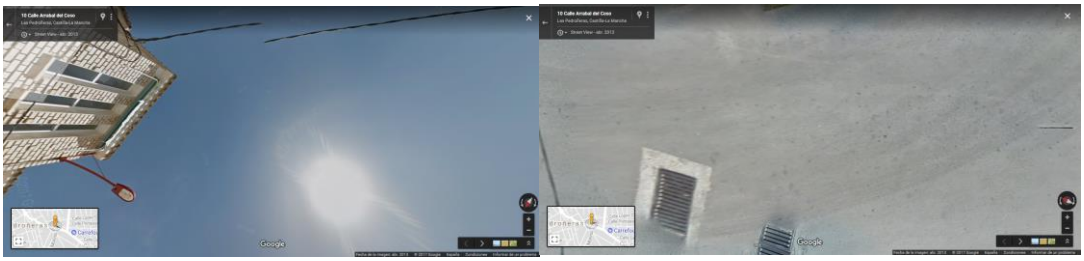


Figura 2.21: *Heading angle* máximo (izquierda igual a cielo) y mínimo (derecha igual a suelo)

Un **resumen de los ángulos** mencionados anteriormente, el ***Pitch angle* (inclinación vertical) y del *Heading angle* (también referido como *Yaw angle*, indicando la inclinación horizontal)**, puede verse en la figura 2.23:

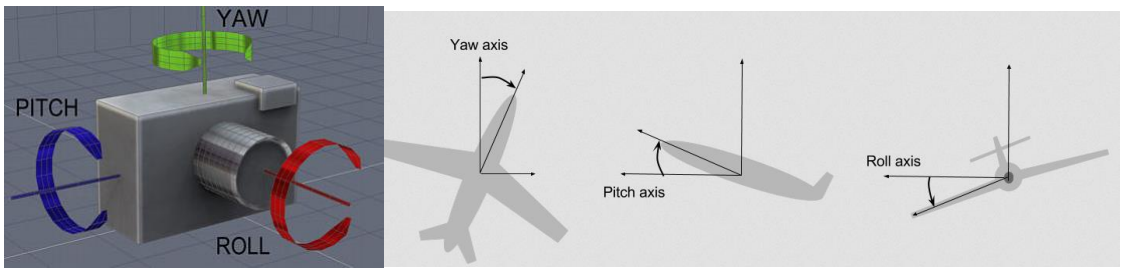


Figura 2.22: Resumen de ángulos

El servicio de *Street View* no permite un movimiento de Roll porque eso supondría voltear la imagen, cosa que no tiene ningún sentido para sus propósitos.

7. Para terminar la URL, el último parámetro importante es el **panoid.**

El panoid es la identificación del panorama en específico (es decir, se corresponden con la ubicación en la que está el usuario) y **no se modifica con el zoom o rotación de la ubicación**, a menos que se provoque una traslación hacia otro punto, con lo cual el identificador cambia.

El panoid se puede observar en la URL como el código alfanumérico existente, en el apartado de data= después de la combinación !1s y !2e. En el siguiente ejemplo se puede observar que se corresponde con X2Cw4jKn2KPydBEV8_6kNg.

https://www.google.es/maps/@39.4480215,-2.6718854,3a,75y,64.15h,1t/data=!3m6!1e1!3m4!1s**X2Cw4jKn2KPydBEV8_6kNg**!2e0!7i13312!8i6656

2.3.2. Base de datos de geolocalizaciones:

Tras haber diseccionado la URL, ¿Cuál es el siguiente paso? → **Pasar horas y horas buscando geolocalizaciones por todo el mundo con el fin de obtener una base de datos lo suficientemente grande para poder entrenar la red neuronal de una forma eficiente al final de proceso.**

Tal y como se comentará en el capítulo 4 *Sintetización de imágenes a partir de imágenes panorámicas*, cada URL es capaz de dar proporcionar tres secciones. En total, para la misión de entrenamiento de la URL se han recolectado 579 geolocalizaciones separadas en dos bases de datos, una para entrenamiento (que contiene 354) y otra para test (que contiene 225). De todas ellas se han analizado 354 localizaciones, dando lugar a una cantidad de 354·3 = 1.062 teóricas secciones, siendo el número real 909, por tanto resultando en una eficiencia del 85,6 %. El tiempo estimado para la búsqueda de las geolocalizaciones ha sido de treinta días, más otros treinta días adicionales para el procesamiento de estos datos (siendo hecho por una única persona, el autor de este documento). Se estima que para el entrenamiento efectivo de una red neuronal se debe contar miles (incluso decenas) de datos, por lo que sería un trabajo futuro ampliar esta base de datos y etiquetado con ayuda de un pequeño equipo de personas.

La siguiente imagen ilustra una pequeña parte de la base de datos, elaborada en Microsoft Excel:

Figura 2.23: Base de datos de Excel. Contiene las geolocalizaciones a analizar

1	**Instructions**
2	**Please, insert in each line the analysis you want to perform.**
3	**The month and year for panorama 1 and panorama 2 must**
4	**be different and both URL should be referenced to the same**
5	**latitude and longitude in Google Street View (although GPS**
6	**coordinetes can be slightly different due to the different**
7	**camera positions).**
8	
9	**Each time that Matlab reads the Excel, it will analize only a**
10	**line, analizing two temporary situations and their changes**
11	**for the same geolocation.**

	Change zone	City	Country
13	50_Bishop_St	Conventry	Great_Britain
14	Roman_Coliseum	Rome	Italy
15	Centre_Pompidou	Malaga	Spain
16	St_Bárbara_Square	Madrid	Spain
17	1_R._Barão_da_Gamboa	Rio de Janeiro	Brazil
18	Calle_Príncipe_Alfonso	Lorca	Spain
19	1174_S_Figueroa_St	Los Angeles	U.S.A.
20	803_W_Olympic Blvd	Los Angeles	U.S.A.

Las siguientes imágenes muestran los primeros datos de dicha base de datos: **Esta primera imagen 2.24 muestra, en la esquina superior izquierda, como proceder en la búsqueda de URL. A continuación se detallarán las reglas usadas para la búsqueda de datos.** En azul se puede ver las columnas

Figura 2.24: Zona de cambio, ciudad y país

correspondientes a la zona de cambio (es decir, **lugar emblemático, calle y número, etc…), dato que sirve para geolocalizarlo de forma textual. También se puede ver la ciudad y el país al que pertenece.**

	Month Panorama 1	Year Panorama 1	Month Panorama 2	Year Panorama 2
13	October		2012 September	2015
14	September		2010 July	2014
15	November		2009 July	2016
16	October		2013 June	2016
17	August		2014 December	2014
18	May		2012 May	2016
19	May		2011 February	2017
20	April		2009 October	2014

En esta segunda imagen 2.25 se muestra el **mes y año de ambos panoramas.**

Figura 2.25: Mes y año de ambos panoramas

Este mes y año son una información muy útil a la hora de hacer eficiente la búsqueda → Si alguien se equivoca en el proceso y ha confundido alguna URL de fila, al saber perfectamente la localización con los otros tres datos anteriores y el mes y año de ambos panoramas puede enmendarse fácilmente el error.

	URL panorama 1	URL panorama 2
13	https://www.google.es/maps/@52.4108303,-1.5110	https://www.google.es/maps/@52.4108244,-1.51106
14	https://www.google.es/maps/@41.8894489,12.490!	https://www.google.es/maps/@41.8894691,12.49053
15	https://www.google.es/maps/@36.7188588,-4.4122	https://www.google.es/maps/@36.718872,-4.4122517
16	https://www.google.es/maps/@40.4271366,-3.6959	https://www.google.es/maps/@40.4271186,-3.696013
17	https://www.google.es/maps/@-22.897906,-43.197!	https://www.google.es/maps/@-22.897909,-43.19759
18	https://www.google.es/maps/@37.6730259,-1.6989	https://www.google.es/maps/@37.6730333,-1.698993
19	https://www.google.es/maps/@34.0426787,-118.26	https://www.google.es/maps/@34.0426422,-118.2662
20	https://www.google.es/maps/@34.0451481,-118.26	https://www.google.es/maps/@34.0451411,-118.2642
21	https://www.google.es/maps/@34.0451599,-118.26	https://www.google.es/maps/@34.0451877,-118.2645
22	https://www.google.es/maps/@34.0454701,-118.26	https://www.google.es/maps/@34.0454724,-118.2657
23	https://www.google.es/maps/@34.0463843,-118.26	https://www.google.es/maps/@34.0463815,-118.2664

Esta imagen 2.26 muestra el corazón de la base de datos. Estas son las **URL almacenadas, puesto que hay dos situaciones temporales (dos tiempos distintos) para una misma geolocalización.**

Figura 2.26: URLs de ambos panoramas

1		N.B:		
2		If Low Resolution == 0 → The panorama will		
3		Have 6656 x 13312 x 3 uint8 dimensions		
4				
5		If Low Resolution == 1 → The panorama will		
6		Have 1536 x 3584 x 3 uint8 dimensions		
7				
8		The dimensions must be the same for both		
9		panoramas to ensure the aligment.		
10				
11				
12	Maximum Depth	Type of change	Low Resolution	
13		40 Structural change	1	
14		40 No changes	1	
15		40 Structural change	1	
16		40 Structural change	1	
17		40 No changes	1	
18		40 Structural change	1	
19		40 Repairements	1	
20		40 Structural change	1	

Finalmente, **las últimas tres columnas 2.27 indican la profundidad máxima** (se ha instalado un estándar de 40 m para determinar el mapa de profundidad), **el tipo de cambio y la resolución. La resolución de una imagen viene determinada por el múltiplo de los píxeles en horizontal y los píxeles en vertical.**

Figura 2.27: Máxima profundidad, tipo de cambios y resolución del panorama

Si el parámetro ***Low* resolution en el Excel es igual a 1**, el **panorama estará en baja resolución** en forma de matriz tridimensional igual a **1536 (pixeles) x 3584 (pixeles) x 3 (capas, porque es RGB),** mientras que si es igual a **0**, el panorama estará en **alta resolución en forma de matriz tridimensional igual a 6656 (pixeles) x 13312 (pixeles) x 3 (capas, porque es RGB).**

Se ha instalado el **valor 1 en toda la columna** (es decir, panorama en baja resolución) porque los resultados obtenidos son muy similares y sin embargo computacionalmente es mucho más costoso tratar un panorama con un número de píxeles mucho mayor.

2.3.3. Proceso de búsqueda de geolocalizaciones:

Para poder obtener unos buenos datos y facilitar el futuro alineamiento y etiquetado, el usuario deberá seguir las siguientes reglas para la obtención de una geolocalización:

1. Sumergirse en Google Maps y arrastrar a Pegman (el muñequito de Street View ) hasta una zona del mapa que presente un tono azul grisáceo (eso demuestra que ahí se ha hecho al menos una pasada).

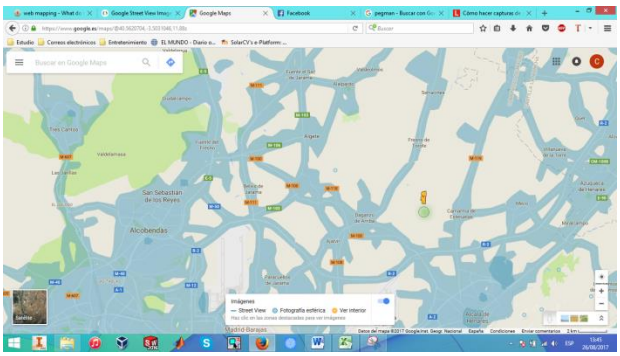


Figura 2.28: Posicionamiento semialeatorio con *Pegman*

Otra opción es, **si el usuario se quiere ubicar en alguna localización en concreto**, utilizar el buscador de la esquina superior izquierda para así posicionarse donde desee. Entonces sólo debe posicionarse en la bandera que aparece con Pegman:

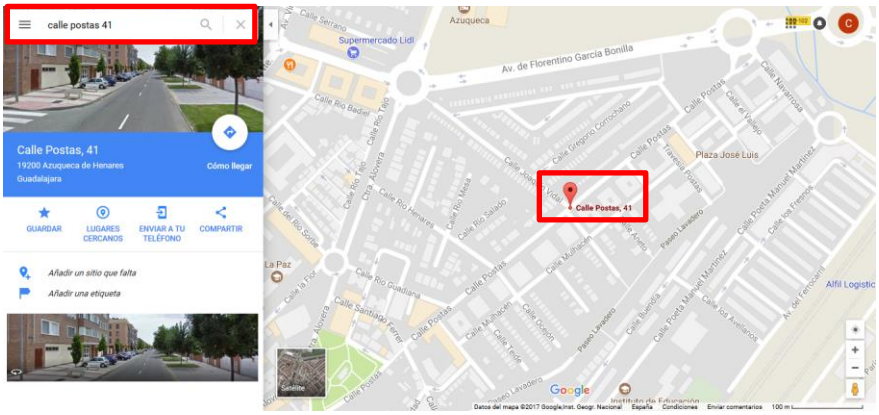


Figura 2.29: Posicionamiento con *Pegman* referenciado a la bandera

3. Ubicados ya dentro del servicio a pie de calle, **la mejor opción es buscar dentro de la máquina del tiempo** aquella localización que presente un tiempo atmosférico con el cielo despejado (sin nubes) para que no puedan ser confundidas con otro objeto, poca vegetación, pocos árboles así como transeúntes. Un ejemplo de ello es la Figura 2.30:

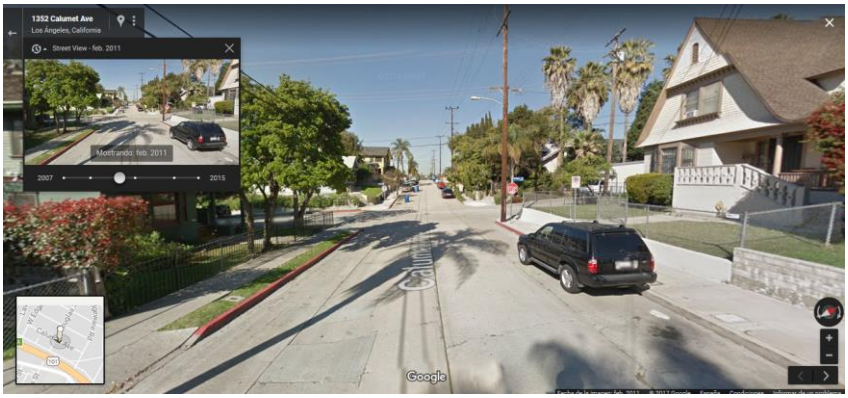
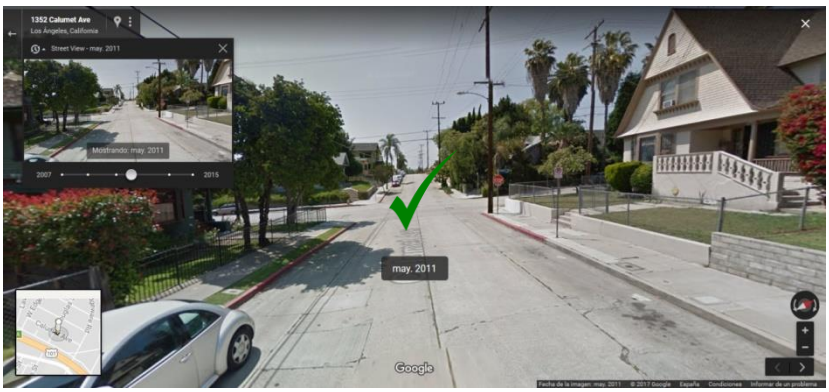


Figura 2.30: Geolocalización óptima

4. El siguiente objetivo es **buscar en la máquina del tiempo aquella pasada que tenga una correspondencia prácticamente perfecta**, es decir, **mínima variación en latitud, longitud y demás ángulos**. Un truco muy útil es, en vez de ir clickando en todas, se puede ir comprobando dentro de la pequeña ventana de la máquina del tiempo, viendo si la nueva ubicación se ajusta (por ejemplo, si el árbol de la pasada original ha desaparecido, si la farola está en el mismo sitio, de la casa sólo se ven tres ventanas etc). A continuación se muestran en primer lugar una pasada errónea con respecto a la original y en segundo lugar una pasada que se ajusta cuasiperfectamente (es imposible perfectamente puesto que el coche, posición de la cámara, ajuste del técnico, … nunca serán iguales al 100 % para pasadas distintas):

Se puede observar que en esta primera imagen aparece un nuevo tejado de la casa (a la derecha) que antes no aparecía, por tanto tiene un *heading angle* mayor (rotado hacia la derecha).

Figura 2.31: Correspondencia incorrecta respecto a la figura 2.30



En esta nueva pasada la correspondencia es prácticamente perfecta.

Figura 2.32: Correspondencia correcta respecto a la figura 2.30

Obsérvese que **aunque las correspondencias geométricas de los elementos sean prácticamente perfectas** (misma porción de la casa, del árbol, de la farola, etc) si esta nueva pasada contiene un exagerado número de transeúntes, coches, etc se descartará y se procederá a la búsqueda de una pasada con mejores características. Este fenómeno se ha dado sobre todo en ciudades de Americana latina (México D.F.), Río de Janeiro, Tokyo o Hong Kong, donde las calles son en su mayoría estrechas, abarrotadas de mercados, comercios y por tanto gente, que imposibilita un correcto análisis.

Cabe resaltar que no existe una propiedad conmutativa en las acciones cuando se opera con GSV, no es lo mismo acercar una vista y luego alejarla, que alejarla y luego acercarla, sobre todo si se opera con varios términos de zoom, puesto que la aplicación, tal y como se explicó anteriormente, siempre tiende a aproximar a la geolocalización más próxima, no en la que el usuario estaba antes.

5. Finalmente **se comparan ambas localizaciones haciendo una estimación de qué es lo que ha cambiado más entre ambas pasadas** para posteriormente incluirlo en la base de datos. Es muy importante destacar que aunque la tendencia es comparar la pasada más actual en el tiempo, los cambios se comparan recíprocamente, es decir, no sólo se debe ver que en 2007 en la Calle Postas había unas obras, sino que en 2017 ha aparecido un edificio y se han instalado pequeñas papeleras, con lo cual habría un compendio de "Cambio estructural", "Reparaciones" y "Pequeños elementos", aunque fueran en años distintos.

Los cambios se han clasificado en las siguientes clases

- **Cambios estructurales (Structural change):** Cambio de notable magnitud como puede ser la demolición/construcción de un eficio, carpas, grandes contenedores, isletas, plazas, etc…
- **Reparaciones (Repairements):** Todo lo relacionado con las reparaciones, desde conos, vallas, máquinas, material de obra, reparaciones en las fachadas (por ejemplo los típicos telones verdes y azules que impiden que caigan materiales de obra a la calle), obras que se puedan observar en la imagen, etc…
- **Fachadas (Facades):** Todos aquellos elementos de las fachadas que antes no estaban (o al revés), como puede ser un cartel enorme, un anuncio que sobresale perpendicular a la fachada (típico para anunciar un comercio, etc…)
- **Señales de tráfico (Traffic signals):** Todas las señales de tráfico y marcas en la carretera como puede ser líneas continuas/discontinuas que antes no estaban, flechas, etc…
- **Pequeños elementos (Small elements):** Todos aquellos elementos que no están asociados a ninguna de las anteriores categorías y están dispuestos por la calle, como pueden ser papeleras, pequeños contenedores, farolas, etc…
- **Ningún cambio (No changes):** Si el usuario no aprecia en el panorama ningún cambio a destacar. **En este TFG no se etiquetarán cambios asociados a personas, coches ni vegetación**.

Estas categorías no son estáticas, sino que se puede y debe combinar con otras, por ejemplo es muy común la combinación de cambio estructural con pequeños elementos en aquellos panoramas donde se presenta un gran cambio o la combinación de reparaciones con fachadas. A continuación se muestran los cambios detectados:

Category	Number
Repairements	126
Facades	37
Traffic signals	11
Structural change	75
No changes	30
Combinations	75

Tabla 2.1: Clasificación de cambios de la base de datos

De esta clasificación se puede elaborar la siguiente estadística:

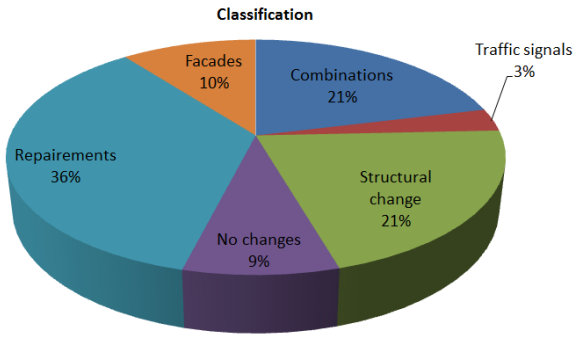


Tabla 2.2: Clasificación en formato sector circular

Una vez almacenadas las URL, la base de datos de Excel es linkada a MatLab, procediendo al análisis de la geolocalización escogida por el usuario, tal y como se verá en el Anexo II *Manual de Usuario*.

2.3.4. Código usado para la obtención de parámetros de interés a partir de la URL:

A continuación se mostrará el código desarrollado en MatLab para obtener los datos relevantes de estas URL. Nótese que aunque el código ha sido desarrollado en MatLab, dado que se ha escrito bajo sistema operativo Ubuntu 16.04 y Windows no reconoce caracteres especiales, tales como diéresis, tildes, etc escritas en Ubuntu, **el código se va a mostrar con la aplicación NotePad++** (aplicación para visualizar el código, modificarlo, etcmuy similar al Gedit de Ubuntu).

```
rango_lectura=sprintf('A%d:L%d',posicion,posicion);

[num,txt,raw] = xlsread('URL_DataCollection_CGH.xlsx',1,rango_lectura);
save('Excel_information.mat','num','txt','raw');

global location city country;

location=strjoin(txt(1));
city=strjoin(txt(2));
country=strjoin(txt(3));

pan1 = strjoin(txt(1,8));
pan2 = strjoin(txt(1,9));

maxDepth = num(1,6); % Maximum depth to calculate the depth map. It will be common for both panoramas.
lowRes = num(1,8);   % Panorama resolution. It will be common for both panoramas.

date1=sprintf('%s %d', strjoin(txt(1,4)), num(1,1));
date2=sprintf('%s %d', strjoin(txt(1,6)), num(1,3));

format long;  % From here to the end of this function, the data will be calculate with decimal notation

% First URL

angulos1 = strfind(pan1,'@');
angulos2 = strfind(pan1,'/data');

angulos = pan1(angulos1(1):angulos2(1)-1); % Angle information

anguloscomas=strfind(angulos,',');
arroba=strfind(angulos,'@');

latitud=angulos(arroba+1:anguloscomas(1)-1);
longitud=angulos(anguloscomas(1)+1:anguloscomas(2)-1);

Latitud1=str2double(latitud); % Panorama latitude
Longitud1=str2double(longitud); % Panorama longitude

unknown1=str2double(angulos(anguloscomas(2)+1:anguloscomas(3)-2)); % Zoom Xa

FoV1=str2double(angulos(anguloscomas(3)+1:anguloscomas(4)-2)); % Field of View (FOV)

heading_angle1=str2double(angulos(anguloscomas(4)+1:anguloscomas(5)-2)); % Heading angle

pitch_angle1=str2double(angulos(anguloscomas(5)+1:end-1)); % Pitch angle

inpanoidA1=strfind(pan1,'!1s');
inpanoidA2=strfind(pan1,'!2e');

panoid1=pan1(inpanoidA1(1)+3:inpanoidA2-1); % Panorama identifier
```

Código de interés 2.1: Extracción de datos de interés de la URL

2.3.5. Ventajas y desventajas del uso de Google Street View:

Para terminar el capítulo donde se ha presentado las distintas herramientas de Google, queda por hablar de las ventajas y desventajas que supone el uso de Google Street View para estos propósitos.

Ventajas:

- Probablemente la ventaja más importante es que es una **herramienta global**, es decir, puede ser usada por todo el mundo **siempre que esta persona disponga de una conexión**. Este hecho contribuye a que varias personas en todo el mundo puedan estar trabajando de forma simultánea en la aplicación y con ello aumentan la base de datos mucho más rápido (actividad en paralelo).
- **Muy fácil de utilizar**. Basta con introducirse en *Google Maps* y escribir la localización exacta que se quiere documentar (y posteriormente arrastrando el muñeco Pegman hasta la bandera), o bien como se ha hecho para este TFG buscar zonas posicionando a Pegman en cualquier parte de interés.
- **Ofrece la capacidad de rotación, traslación y acercamiento** con el simple uso del ratón o teclado, sin necesidad de tener que salirse y volver a introducir una localización (o arrastrar a Pegman hasta

otra), con lo cual basta con ir pinchando en la pantalla a lo largo de la calle para hacer una extraordinaria labor documental.

- Ofrece **información adicional sobre gran parte de los lugares del mundo** (en este caso esta ventaja no ha sido utilizada, pero para temas históricos, documentales, etc podría ser de gran ayuda).

Desventajas:

- **Requiere de una conexión obligatoria a Internet**, por lo que con malas condiciones coberturas, con mucha gente en el servidor, etc la aplicación no se puede utilizar.
- **Hoy en día no se ha terminado de instalar este servicio a pie de calle de todo el mundo,** bien por falta de presupuesto, falta de interés en cierta zona o incluso por políticas de privacidad autóctonas de una zona (como puede ocurrir en ciertos países Árabes, como Abu Dabi, China, Corea del Norte, etc…)
- **A pesar de que se sabe que el servicio no es en tiempo real, las ciudades desarrolladas cuentan con muchas más pasadas que otras ciudades con menor presupuesto**, lo que dificulta en gran medida investigar la evolución de ciudades en desarrollo. Es por ello que este TFG se ha localizado en el análisis de ciudades desarrolladas.

Capítulo 3

Elaboración del panorama y mapa de profundidad:

Tras haber detallado el uso de la herramienta GSV y todo el proceso de identificación de URLs, almacenaje y uso final, de aquí hasta el final de este TFG todos los capítulos tratarán acerca de temas relacionados, de una u otra forma, con la visión artificial.

Si bien el objetivo de este TFG no es el estudio conceptual de la visión artificial, merece la pena destacar los fundamentos básicos para ahondar de una forma más eficiente temas tales como mapas de profundidad, síntesis de imágenes o detección de características entre otras cosas.

3.1. Fundamentos de la visión artificial:

La visión artificial (VA) o visión por computador es una disciplina científica que incluye métodos para **adquirir, procesar, analizar y comprender las imágenes del mundo real con el fin de producir información numérica o simbólica para que puedan ser tratados por un computador.** Estas imágenes a analizar pueden provenir de un mundo 2D (por ejemplo la fachada de un edificio) o un mundo 3D (una calle).

Tal y como los humanos usan sus ojos y cerebros para comprender el mundo que les rodea, **la visión por computador trata de producir el mismo efecto para que las computadoras puedan percibir y comprender una imagen o secuencia de imágenes y actuar según convenga en una determinada situación.** Esta comprensión se consigue gracias a distintos campos como la geometría, la estadística, la física y otras disciplinas.

Dicho de otra forma, la visión artificial podría definirse como el proceso ordenado de las siguientes cinco acciones:

- Captación de la información visual (imágenes) por medio de un sensor (cámara).
- Extracción y caracterización de la información visual mediante un procesador (ordenador o computadora, de ahí el de nombre de *computer vision* en inglés).
- Análisis de las características, obtención de resultados y posterior interpretación.

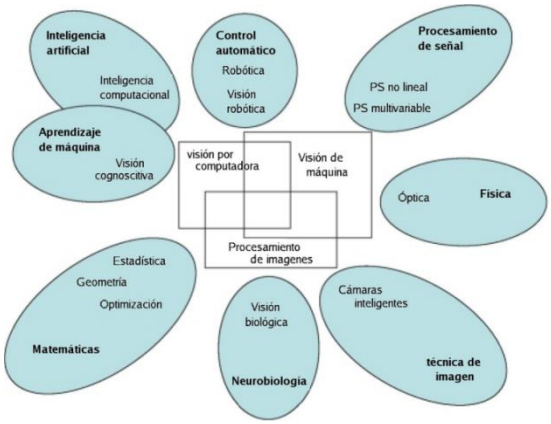


Figura 3.1: Campos y áreas afines relacionados con la visión artificial

Tal y como se puede observar en la figura 3.1, la visión artificial es una rama de la inteligencia multidisciplinar con aplicaciones en diversos campos como:

- Inspección industrial y control de calidad (verificación de etiquetado y códigos, inspección de soldaduras, clasificación de piezas, etc…)
- Identificación biométrica (huella, pisadas, firmas) y reconocimiento de caras y gestos
- Procesamiento de señales
- Control de tráfico (reconocimiento de matrículas, control de flujo, sistemas de ayuda a la conducción)
- Guiado de robos industriales y vehículos autónomos
- Psicología
- Neurología

Hay muchas tecnologías que utilizan la visión por computador, entre las cuáles se encuentran el reconocimiento de objetos, la detección de eventos, la reconstrucción de una escena (mapping) y la restauración de imágenes.

El principal objetivo de la VA es la obtención de información relevantes a partir de imágenes. Sin embargo, esta información relevante puede y debe cambiar en función de la aplicación. Por ejemplo, un sistema de tráfico tendrá como misión tareas tales como contar vehículos, detectar intrusiones, medir velocidades, medir dimensiones, clasificar modelos, etc mientras que una cámara de vigilancia estará especializada en el reconocimiento facial, seguimiento de personas, etc… Es por ello que a mayor importancia de la aplicación, mayor será la complejidad (tanto computacional como de memoria). **Una técnica muy ingeniosa es la de introducir información a priori (relacionados con la aplicación) para poder comparar datos.** Este es uno de los fines de este Trabajo de Fin de Grado → **Elaboración de una base de datos de imágenes sintetizadas a partir de GSV para poder entrenar una red neuronal que posteriormente sea capaz de autoetiquetar cambios estructurales.** Poco a poco esta frase irá tomando forma en el documento.

Descritos ya los fundamentos básicos de la visión artificial, ahora se deben tratar dos conceptos fundamentales para abordar un mapa de profundidad (y también el capítulo 5 *Extracción y descripción de puntos característicos en vistas sintéticas*), el color y el concepto de imagen panorámica. Es de especial importancia el estudio de los fundamentos del color pues la computadora captará y procesará la información emulando el comportamiento del ojo humano.

3.2. Fundamentos del color:

El color es un tema complejo y daría perfectamente para un estudio en sí mismo. Conceptualmente, es la **impresión producida por un tono de luz en los órganos visuales, es decir, es la percepción visual que se genera en el cerebro al interpretar las señales nerviosas enviadas por los fotorreceptores de la retina del ojo**, que a su vez interpretan y distinguen las distintas longitudes de onda que captan la parte visible del espectro electromagnético (aproximadamente de 400 a 700 nm).

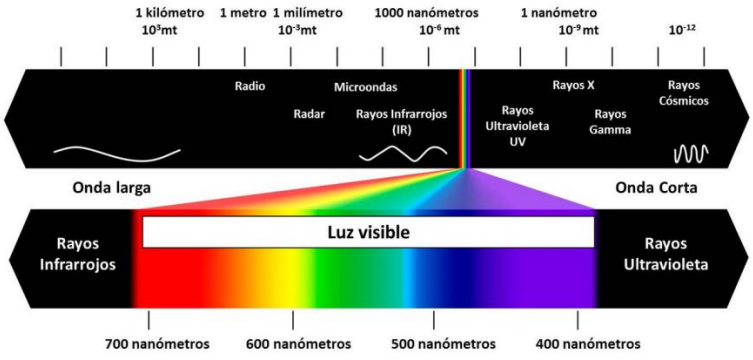


Figura 3.2: Espectro electromagnético

Todo cuerpo iluminado absorbe una parte de las ondas electromagnéticas y refleja las restantes. Precisamente un objeto se ve como es no por las ondas que absorbe, sino por las que refleja, que son interpretadas en el cerebro como distintos colores según las longitudes de ondas correspondientes.

Por ejemplo, un tomate es rojo porque de los colores primarios absorbe el amarillo y el azul y refleja el rojo, mientras que un plátano es amarillo porque absorbe el color azul y refleja los colores rojo y verde los cuales sumados dan el amarillo. Un cubo es negro porque absorbe todo el espectro y una pelota es blanca porque refleja todas las ondas electromagnéticas.

3.2.1. Aspectos que caracterizan al color:

Los aspectos básicos que caracterizan al color son tres: **Brillo, matiz y saturación.**

- **Brillo:** Incorpora el **concepto cromático de luminosidad**, es decir, la sensación que indica si un área del campo visual está más o menos iluminada.
- **Matiz:** Concepto asociado con la **longitud de onda dominante en la mezcla de longitudes de onda de luz** (sensación que indica si un área parece similar al rojo, amarillo, verde o azul o a una proporción de ellos).
- **Saturación:** Pureza relativa o **cantidad de luz blanca mezclada con un matiz** (inversamente proporcional a la cantidad de luz blanca). **Si se unen los conceptos de matiz y saturación el resultado es la cromaticidad.** Por tanto un color puede ser definido por su brillo y su cromaticidad (matiz+saturación).

3.2.2. Espacios de color:

Por otra parte, un **espacio (o modelo) de color** es una **especificación de un sistema de coordenadas 3D donde cada color está representado por un punto.**

Los espacios de color son siempre tridimensionales (de no ser así, sería un sistema bidimensional en blanco y negro). Los espacios más frecuentes son:

- **RGB (Red, Green, Blue):** Usado habitualmente en imágenes digitales, pantallas de ordenador o videocámaras.
- **CYM (Cyan, Yellow, Magenta):** Utilizado en impresoras de color.
- **YUV/YIQ (Intensity/Chromaticity):** Se usa en la transmisión de señales de televisión.
- HSV (Hue/Saturation/Value): Este espacio se usa típicamente en imágenes artísticas.

En el procesamiento de imágenes el espacio más utilizado es el RGB (hecho que concuerda con este TFG), aunque muchas aplicaciones de tratamientos de imágenes digitales requieren la transformación entre el espacio RGB y otros espacios de color.

El modelo RGB está basado en la síntesis aditiva, con el que es posible representar un color mediante la mezcla por adición de los tres colores luz primarios (Red = Rojo, Green = Verde, Blue = Azul).

La cantidad de colores RGB requeridas para formar un color particular se denominan colores triestímulos. Si se identifican estos colores triestímulos (cantidades de RGB) con los coeficientes R (*Red* = Rojo), G (*Green*=Verde) y (Blue = Azul), un color se puede definir por las proporciones que representan cada uno (o dicho de otra forma, coeficientes tricromáticos x, y, z valorados de 0 a 1):

$$x = \frac{R}{R+G+B}; y = \frac{G}{R+G+B}; z = \frac{B}{R+G+B} \rightarrow x + y + z = 1 \quad (3.1)$$

Siendo “x” la cantidad de rojo sobre la mezcla total, “y” para el verde y “z” para el azul.

Aunque el intervalo de valores podría ser cualquiera, **es frecuente que cada color primario se codifique con un byte (8 bits). Así, de manera usual, la intensidad de cada uno de las componentes se mide según una escala que va del 0 al 255** $(255 = 2^8)$. Este hecho se refleja en el TFG, dado que tras la lectura de una imagen (generalmente todas tienen formato RGB) generará una matriz tridimensional **M x N x 3**, siendo M la anchura en píxeles de la imagen, N la altura y 3 por los tres planos RGB. En cada hueco de cada plano (realmente una matriz bidimensional) se almacenará un valor de 0 a 255.

Por lo tanto, el rojo puro se obtiene con (255,0,0), el verde puro con (0,255,0) y el azul puro con (0,0,255), obteniendo, en cada caso un color resultante monocromático. En este modelo, la ausencia de color (o color negro) se obtiene cuando las tres componentes son 0, (0,0,0). Obviamente, el color blanco se forma con los tres colores primarios a su máximo nivel (255,255,255). Esto se puede reflejar en el siguiente cubo, donde **la escala de grises es la diagonal que une el color blanco con el negro.**

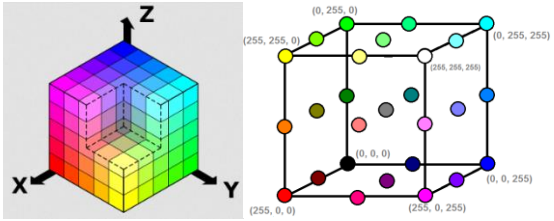


Figura 3.3: Cubo RGB

3.2.3. Profundidad de color:

La **profundidad de color o bits por pixel (bpp)** es un concepto de la computación gráfica que se refiere a la **cantidad de bits de información necesarios para representar el color de un píxel en una imagen digital.** Debido a la naturaleza del sistema binario de numeración, una profundidad de n bits implica que cada píxel de la imagen puede tener 2^n posibles valores y por lo tanto, representar 2^n colores distintos.

Al principio, la limitación en la profundidad de color de la mayoría de los monitores limitaba cada pixel a 6 bits (2 bits por cada color primario), es decir, una gama limitada de 216 colores ($2^6 = 256$). Sin embargo, con el dominio de los monitores de 24 bits (un byte = 8 bits por cada color primario) generó una impresionante gama de 16,7 millones de colores ($2^{24} = 2^8 \cdot 2^8 \cdot 2^8 = 256 \cdot 256 \cdot 256 = 16{,}7$ millones de colores)

3.3. Percepción de la profundidad y profundidad de campo [3]:

Es importante entender el concepto de profundidad en una imagen, diferenciándolo de otro tipo de profundidad que se conoce como profundidad de campo (*Depth of Field, DOF*) cuyo concepto ayuda a entender el problema de la nitidez y futura extracción de características del capítulo 5 *Extracción y descripción de puntos característicos en vistas sintéticas*.

- La percepción de la profundidad es una derivación del concepto generalista de profundidad. Mientras que definiendo la profundidad como tal indica la distancia de un elemento con respecto a un plano horizontal de referencia cuando dicho elemento se encuentra por debajo, **la percepción de profundidad está relacionado con el mundo tridimensional**, siendo la relación del objeto y el observador visual de percibir el mundo en tres dimensiones. Dicho de otra forma, **la visión estereoscópica** (captación de mundo tridimensional por los ojos) es la **percepción de profundidad de un visión binocular** (es decir, que utiliza ambos ojos).
- Por otra parte, **la profundidad de campo** (Depth of Field DOF) es una **característica de la percepción de profundidad que indica la zona en la cual la imagen captada por los ojos** es **nítida, y por tanto, está enfocada.**

Una forma de hacer la estimación de profundidad de una imagen es dividir esta misma en regiones cuadradas no superpuestas. A partir del análisis de estas regiones se puede obtener información sobre la profundidad absoluta, es decir, la profundidad de una región respecto al resto de la imagen, así como la profundidad relativa, es decir, la profundidad de una región respecto a sus regiones adyacentes.

- La **profundidad** absoluta de una región indica cuál es su **profundidad en el contexto de la imagen global**. Para estimar la profundidad absoluta de una región no basta con un análisis de las características locales de la región, sino que las características globales de la imagen son requeridas también.

  Para obtener una buena estimación **se suele trabajar con la imagen en diferentes escalas** (es decir, más grande o más pequeña. Este trabajo en varias escalas es una de las bases de las técnicas de los extractores de características del capítulo 5 *Extracción y descripción de puntos característicos en vistas sintéticas*). **Esto es útil para encontrar la profundidad absoluta dado que hay regiones con distintas profundidades que presentan comportamientos distintos cuando son analizados a diferentes escalas.** Un ejemplo es el cielo, dado que es una región muy homogénea (suponiendo ausencia de nubes, etc) es muy fácil de identificar, mientras que la hierba a escalas distintas se ve también distinto (porque está constituido por un montón de hojas, fibras, etc…) luego la estimación de su profundidad es mucho más costosa.

- Por otro lado, la **profundidad relativa de una región** indica cuál es su **profundidad respecto a las regiones adyacentes.** Mientras que para el cálculo de la profundidad absoluta se consideraban las características locales de la región y las globales de la imagen, para el cálculo de la relativa se consideran las características de las regiones adyacentes. Así, dos regiones adyacentes (colindantes) con similares características de textura y color tendrán profundidades similares, mientras que si estas características son notoriamente distintas, sus profundidades también lo serán. Esta filosofía de trabajo es la base de los extractores de características, es decir, la comparación con regiones adyacentes.

3.4. Mapa de disparidad:

Con los conceptos de color, el modelo RGB, percepción de profundidad, profundidad de color y profundidad de campo ya definidos, es hora de detallar el concepto de mapa de disparidad y cómo se ha obtenido en la herramienta.

El mapa de profundidad de una imagen (también llamado **mapa de disparidad**) es la **representación plana de la estructura tridimensional de la imagen original**, es decir, aunque dimensionalmente en anchura y altura ambas imágenes (la original y la del mapa) son iguales, **para representar la profundidad de un punto con respecto a la cámara** (y por tanto obteniendo la tercera dimensión o profundidad) **esto se lleva a cabo mediante un código de colores asociado a una cierta profundidad** (por ejemplo, si el código 0 indica blanco y lo más cerca posible y el 255 el negro y lo más lejano posible, cuanto más blanco es un punto significa que está más cerca del punto de vista o por el contrario cuanto más negro sea, más lejos está del punto de vista). Este código de colores debe ser previamente asignado por código.

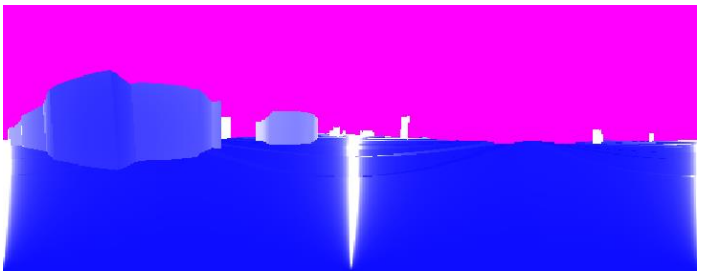


Figura 3.4: Ejemplo de mapa de profundidad obtenido en este trabajo

De la definición anterior se puede extraer por tanto una conclusión: **La reconstrucción de la profundidad de una imagen es el proceso por el que a partir de imágenes bidimensionales** (como puede ser la imagen panorámica de la que se dispone) **se puede obtener información tal que permita una recreación de la distribución espacial real** (por ejemplo, para una futura estructura 3D).

Aunque normalmente la estimación de la profundidad está basada en algoritmos de reconstrucción de la profundidad a partir de dos o más imágenes, tal y como sucede **en este TFG la profundidad se ha reconstruido a partir del análisis de características de la imagen panorámica generada** (como variaciones de la textura o el color, que por supuesto a también pueden aportar información sobre profundidad.

No es objeto de este TFG la demostración matemática de como se ha determinado la composición del mapa de profundidad (además que el archivo .json, que se detallará a continuación, proviene de una API de GSV de la cual, evidentemente, la empresa no publica su método de cálculo, simplemente ofrecen el mapa ya hecho), ya que esta reconstrucción suele estar basada no sólo en características locales, sino también en modelos probabilísticos. Este modelo probabilístico (Gaussiano o Laplaciano) establece relaciones entre la profundidad de diferentes regiones de la imagen mediante un campo aleatorio de Markov.

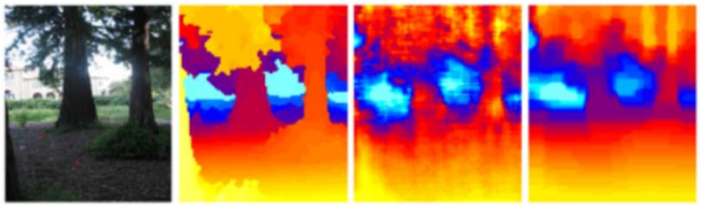


Figura 3.5: De izquierda a derecha: Imagen original, mapa de profundidad a partir de modelo Gaussiano y mapa de profundidad a partir de modelo Laplaciano

Se puede observar que el mapa de profundidad basado en el modelo probabilístico Laplaciano se ajusta más a la imagen original (es decir, menor cantidad de ruido), ya que está basado en derivadas de segundo orden, homogeneizando el resultado.

Otro ejemplo de mapas de profundidad es la Figura 3.6 cuyo mapa está basado en la comparación entre dos imágenes para la obtención de la profundidad.

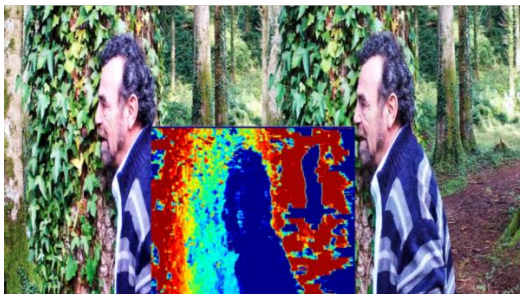


Figura 3.6: Obtención del mapa a partir de dos imágenes con pequeña diferencia de pose

Esta será una de las mejoras futuras en este TFG: Mejora del mapa de profundidad de la imagen panorámica duplicando la base de datos de URLs en Excel y por tanto elaborando la profundidad a partir de la comparación de ambos panoramas, cuyo modelo matemático es el siguiente:

$$D = f \cdot \frac{B}{d_x} \qquad ; \qquad d_x = x_r - x_l \tag{3.2}$$

Siendo f la distancia focal de ambas cámaras [mm], B la línea base (distancia entre las cámaras [m]), d_x la diferencia de proyección de un pixel en la imagen derecha (x_r) e izquierda (x_l) y finalmente D la profundidad asociada a cada pixel.

El mapa de disparidad obtenido en este TFG está basado en el *paper* de referencia *"3D City Reconstruction From Google Street View"* [4].

Para la representación del mapa de profundidad se deberán seguir los siguientes pasos: **Construcción de la imagen panorámica, composición del mapa de profundidad y finalmente representación del mapa de profundidad.**

3.4.1. Construcción de la imagen panorámica:

Este paso es la base del TFG para poder realizar el resto de acciones. Dicho de una forma no técnica, las secciones que se van a etiquetar en la interfaz final proceden del seccionamiento o "troceado" del panorama generado al principio.

Una **imagen panorámica** es aquella en la que se muestra un panorama, correspondiendo generalmente a una **rotación de la cámara de 360 º** (es decir, barrido del *heading angle* de 360 º) **a partir de una posición fija de esta misma.**

En primer lugar, se debe **acceder a la base de datos de Excel, leer la URL correspondiente y extraer los datos de interés de esta URL** y en especial el identificador de panorama o panoid. Dicho código de interés fue mostrado al final del capítulo anterior donde se enseñaba la recolección de datos de interés de ambas URLs [1].

Con el panoid ya identificado (por ejemplo 'fYWuanKZYNYobHjYu925MQ'), se debe **especificar la resolución que se le dotará al panorama**. A pesar de que GSV puede proporcionar resoluciones por encima de 13.312 x 6.656 pixeles, para este caso en particular una resolución de 3.584 x 1.536 pixeles es más que suficiente (Nótese que esa alta resolución es lo que en la base de datos de Excel correspondería con el parámetro Low Resolution = 0 y la baja resolución con dicho parámetro = 1).

En ambos casos la resolución de cada "azulejo" o cuadrito es la misma, 512 x 512 píxeles, la diferencia radica en que en alta resolución hay 26 x 13 secciones (horizontal x vertical respectivamente) y en baja

resolución hay 7 x 3 secciones, con lo cual la resolución de imagen, medida en puntos por pulgada, es menor en este caso.

Para este caso en particular, **el objetivo es componer en una única imagen todo el panorama de 360 º** desde la posición fija en un teselado de 7 x 3 "azulejos", cada uno con una resolución cuadrada de 512 x 512 píxeles, ya que **Google no permite descargar directamente la imagen panorámica**.

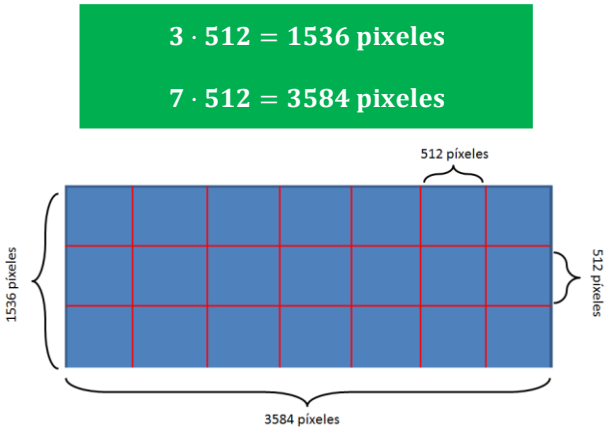


Figura 3.7: Estructura del panorama en baja resolución

Por tanto, para recomponer este panorama "teselado" se deben descargar una a una las 21 secciones que conforman el panorama. Esta descarga está ofertada gratuitamente por la **API Google Maps Rest**, habiendo a disposición dos enlaces para la descarga de estas secciones en función de si se quieren en alta resolución [5] o en baja resolución [6].

En ambos casos, los argumentos que **se deben introducir mediante código** son 3, un dato en formato *string* (% s) que es el **panoid**, y dos datos de tipo entero (% d) que son la **posición en X** de la sección (de 0 a 6 en el formato en baja resolución, de 0 a 22 en alta resolución) y la **posición en Y** de la sección (de 0 a 2 y de 0 a 12 respectivamente).

Con dichos enlaces y conexión a Internet, mediante la función *imread* (explicada en el anexo de funciones de MatLab) se obtiene la matriz tridimensional correspondiente a la sección, de dimensiones 512 x 512 x 3 (porque está en código RGB). Al solaparlas todas convenientemente el resultado es una única imagen RGB de dimensiones 3584 x 1536 que es el panorama deseado. A continuación se muestra el código MatLab 3.1 que genera los panoramas

Función getPanorama.m

```
function [Ipan] = getPanorama(panoid, lowRes) %lowRes

if (nargin == 0)
    panoid='fYWuanK2YNYobHjYu925MQ';
end;

if (lowRes == 0) % 6656 x 13312 x 3 uint8
    xtiles = 26;
    ytiles = 13;
else            % 1536 x 3584  x 3 uint8
    xtiles = 7;
    ytiles = 3;
end

width = 512*xtiles;
height = 512*ytiles;
Ipan = zeros(height, width, 3, 'uint8');
tile_height = height/ytiles;
tile_width = width/xtiles;

for ix=0:xtiles-1
    for iy=0:ytiles-1

        if (lowRes == 0)
            st_url = sprintf('http://cbk0.google.com/cbk?output=tile&panoid=%s&zoom=5&x=%d&y=%d', panoid, ix, iy);
        else
            st_url = sprintf('https://geo0.ggpht.com/cbk?cb_client=maps_sv.tactile&authuser=0&hl=en&panoid=%s&output=
```

```
35          end
36
37          im = imread(st_url);
38          ypos = ((iy)*tile_height+1):(iy+1)*tile_height;
39          xpos = ((ix)*tile_width+1):(ix+1)*tile_width;
40          Ipan(ypos, xpos, :) = im;
41      end
42  end
43
44  end
```

Código de interés 3.1: Generación del panorama

Se itera mediante dos bucles *for* anidados para obtener la información de cada azulejo, situándolo cada uno en su posición correspondiente (de [0,0] a [2,6]).

Por ejemplo, con este panoid: **XLbnbAMdygDrzmDr909c2w** , sustituyéndolo por %s y haciendo el doble bucle *for* anidado para los datos %d x e y en la URL:

https://geo0.ggpht.com/cbk?cb_client=maps_sv.tactile&authuser=0&hl=en&panoid=%s&output=tile&x=%d&y=%d&zoom=3

El azulejo [0,0] se muestra en la siguiente imagen [37].

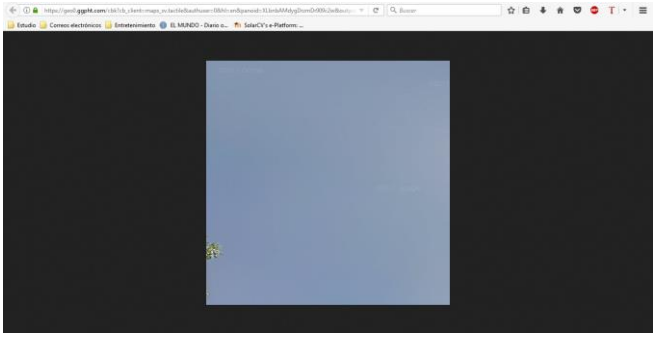


Figura 3.8: Azulejo obtenido a partir de la API Google Maps Rest

Posteriormente, mediante la lectura con *imread* de esta URL (siempre que se tenga conexión a Internet), se logra obtener la información (modelo RGB) del azulejo, o lo que es lo mismo, su matriz tridimensional numérica correspondiente. Iterando este proceso para las posicione [0,0], [1,0], [2,0] se obtiene:

A la izquierda se puede observar perfectamente como estos tres azulejos unidos convenientemente originan la primera columna del panorama (se ha ampliado y delimitado cada azulejo con una línea roja para que puedan ser estudiados en un mayor detalle).

Se aprecia por ejemplo como **la distorsión de la imagen aumenta en las zonas más alejadas del centro del panorama (ver elipse azul, esta columna corresponde con la primera),** hecho que se observa perfectamente cuando la panorámica final se ha reconstruido.

Figura 3.9: Primera columna de azulejos del panorama

En la imagen panorámica final 3.10 se aprecia aún más la distorsión comentada en el párrafo anterior: Las aceras que antes eran rectas próximas a la posición de la cámara, para la elaboración del panorama tornan a curvas, al igual que los edificios. Las elipses azules que se muestran a continuación son claros ejemplos de esa distorsión.

Figura 3.10: Panorama generado a partir del solapamiento de los azulejos

Esta distorsión realmente es un problema, puesto que si se intentaran tomar extractores de características del panorama para el cálculo de un offset global el resultado estaría ciertamente falseado.

Esto es así porque **las zonas con menor distorsión corresponden a la columna central y sus proximidades** (ver rectángulo morado 3.10), y a medida que se aleja del centro las vistas comienzan a distorsionarse. Se debe recordar que el panorama extendido no es más que una proyección en un plano cartesiano de los 360 º del barrido del *yaw angle*, por tanto el ángulo en grados y diferencia de píxeles en la horizontal pueden ser relacionados, como se verá a continuación. Por ejemplo, si en el *offset* central (el más cercano a la realidad) da lugar a 5 grados de diferencia en términos de *heading angle*, una chimenea cerca de un lateral presentaría un offset mucho mayor por el simple hecho de que está en la zona distorsionada de ambas imágenes. Este concepto podrá plasmarse de forma mucho más clara en el capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea*, donde se trata este cálculo del offset.

3.4.2. Composición del mapa de profundidad:

Tras haber compuesto la imagen panorámica, se procede a la elaboración de su mapa de profundidad. Al igual que para la elaboración del panorama, la **API Google Maps Rest** proporciona un enlace de descarga [7]:

https://maps.google.com/cbk?output=json&cb_client=maps_sv&v=4&dm=1&pm=1&ph=1&hl=en&panoid=%s

Con un único dato formato string (cadena %s) a especificar, este es, el **panoid.** Ya especificado, MatLab leerá dicha URL cuya salida es un archivo tipo JSON con la representación comprimida del mapa de profundidad. El usuario no verá en ningún momento la descarga de este archivo, pero si se va directamente a ese enlace, puede observarse que el archiv .json presenta la siguiente estructura:

panoid.json

Siendo panoid el identificador que se introduce en %s la de URL de la API. Un ejemplo:

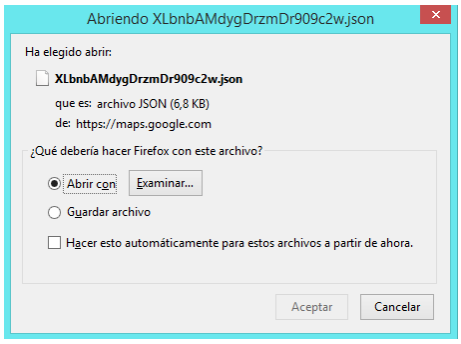


Figura 3.11: Archivo .json descargado

Se puede observar como el archivo descargado es guardado con el nombre del panoid introducido a la URL de la API (en este caso **XLbnbAMdygDrzmDr909c2w**).

Un archivo **JSON** (acrónimo en inglés de *JavaScript Object Notation*) es un **archivo de texto para el intercambio de datos.** JSON es un subconjunto de la notación literal de objetos de JavaScript (un tipo de lenguaje de programación). Hoy en día JSON se puede considerar como un lenguaje independiente en lo que a intercambio de datos se refiere.

La **sintaxis de JSON** viene definida por los tipos de datos disponibles:

Tipo	Descripción
Números	**Positivos y negativos. Opcionalmente pueden contener parte fraccional separada por puntos.**
Cadenas	**Entre doble comilla, permitiendo secuencias de cero o más caracteres así como cadenas de escape (las clásica \n, \\, \r etc…).**
Booleanos	**Representan valores booleanos, pudiendo ser true o false.**
null	**Representa el valor nulo**
Array	**Representa una lista ordenada de cero o más valores los cuales pueden ser de cualquier tipo. Los valores se separan por comas y el vector entre corchetes. Ej: [“juan”, “pedro”, “Jacinto”].**
Objetos	**Colecciones no ordenadas de pares de la forma <nombre>:<valor> separados por comas y entre llaves. El nombre es una cadena (es decir, entre doble comilla) y el valor puede ser de cualquier tipo.**

Tabla 3.1: Sintaxis del lenguaje JSON

Por ejemplo, los siguientes caracteres encerrados entre llaves corresponden al primer objeto del archivo JSON que se detallará a continuación.

```
1  {"Data":{"image_width":"13312","image_height":"6656","tile_width":"512","tile_height":"512"
```

```
1  ,"image date":"2012-08","imagery type":1,"copyright":"© 2017 Google"}
```

Si volvemos al archivo JSON que se ha descargado a partir de este enlace dispuesto por la API Google Maps Rest, y se abre con el editor de textos Notepad++ (donde se está mostrando el código del TFG), se puede observar una **única fila con 9.669 columnas** (cada columna es igual a un carácter).

Los primeros 842 caracteres (que se pueden observar en los siguientes recortes, aunque todos pertenecen a la misma fila) están distribuidos en distintos objetos de acuerdo a la temática (información general, información angular, localización, etc…). Esencialmente especifican lo siguiente:

- **Anchura de la imagen**: 13312 píxeles; **altura de la imagen:** 6656 (La API por defecto saca la imagen panorámica con estas dimensiones, 13312 x 6656, e igualmente extrae el mapa de profundidad en referencia a estas dimensiones, lo que antes se había denominado alta calidad, mientras que el panorama como tal en los propósitos de la aplicación presenta unas dimensiones de 3584 x 1536 píxeles. Este tratamiento de la API mejora la exactitud del mapa)

- **Altura y anchura de cada azulejo:** 512 píxeles (26 azulejos en la horizontal por 13 en la vertical).

- **Copyright:** © 2017 Google

- **Proyección:** Esférica (de ahí la distorsión que se había comentado anteriormente).

- **Información angular:** Inclinación en términos del *pitch angle*, inclinación en términos del *yaw angle*).

- **Localización numérica:** En términos de latitud y longitud en formato decimal. Es importante resaltar que la API indica una **latitud y longitud originales** y **aparte unas actuales** puesto que se ha aproximado a la imagen panorámica georegistrada más cercana.


```
:"1.04"},"Location":{"panoId":"XLbnbAMdygDrzmDr909c2w","zoomLevels":"5","lat":"40.571940",
"lng":"-3.259011","original_lat":"40.571953","original_lng":"-3.259032","elevation_wgs84_m":"689.93
```


- **Localización textual:** Descripción literal del número y calle donde el usuario estaba ubicado al extraer la URL, región (incluyendo ciudad y comunidad autónoma/región) y país.

Se muestran a continuación los primeros 842 caracteres correspondientes a estos datos que se acaban de comentar.


```
{"Data":{"image_width":"13312","image_height":"6656","tile_width":"512","tile_height":"512"
,"image_date":"2012-08","imagery_type":1,"copyright":"© 2017 Google"},"Projection":
{"projection_type":"spherical","pano_yaw_deg":"231.97","tilt_yaw_deg":"128.43","tilt_pitch_deg"
:"1.04"},"Location":{"panoId":"XLbnbAMdygDrzmDr909c2w","zoomLevels":"5","lat":"40.571940","lng":"
lng":"-3.259011","original_lat":"40.571953","original_lng":"-3.259032","elevation_wgs84_m":"689.93
39904","description":"33 Calle Postas","streetRange":"33","region":"Azuqueca de Henares, Castile-
La Mancha","country":"Spain","elevation_egm96_m":"637.822876"},"Links":[{"yawDeg":"50.83","panoId"
":"S90E2Fthha3fNbc95Vzu2A","road_argb":"0x80fdf872","description":"Calle Postas"},{"yawDeg":"231.
.72","panoId":"4vsVMRPmnbMFB6hyHlYnLw","road_argb":"0x80fdf872","description":"Calle Postas"}],
```


Tras estos 842 caracteres, el resto del documento corresponde al mapa de profundidad codificado alfanuméricamente (en concreto hay un total de 8797 caracteres entre las dobles comillas que representan dicho mapa).


```
"model":{"depth_map":"eJztmn1wFOUdx5NwCYTXSkIgmIARBSRgESGB3F5277B1cGzrCxVLQTr42mqlatUaMVvN4BsoDW9W
```


Este código alfanumérico comprimido se ha calculado a partir de la siguiente filosofía: **Calcular la distancia desde la cámara a la superficie más cercana en cada pixel del panorama.**

Con ayuda de las **librerías JSONLAB y encoder** en conjunto (para la codificación/decodificación de los archivos de extensión .json a un formato entendible por MatLab y viceversa) **se decodifica los datos que estaban en base 64** (un tipo de sistema de numeración posicional que usa el 64 como base) y **se transforman en un array de bytes enteros sin signo** (uint8), se puede obtener su header (es decir, su identificador), planos de referencia, etc De hecho, **cada pixel del mapa de profundidad final** (que cuenta

con unas dimensiones de 512 x 256 píxeles, mucho menor que los 13312 x 6656 de la panorámica RGB que estudia la aplicación) **se corresponde con uno de los diversos planos de referencia**, cada uno definido por su vector normal y distancia a la cámara.

No es objeto de este TFG explicar detalladamente el proceso matemático para la obtención del mapa de profundidad usando un modelo de cámara *pinhole*, pero en resumen **para calcular la profundidad de un pixel** se debe determinar el **punto de intersección de un rayo que comienza en el centro de la cámara y el plano correspondiente al pixel.** Iterando para todos los planos, el mapa de profundidad está constituido por una matriz bidimensional de 512 x 256 elementos con formato en coma flotante de 32 bits. Como se ha comentado en el párrafo anterior, la resolución es mucho menor que la panorámica RGB.

El código correspondiente a la adquisición del archivo .json, decodificación y extracción de las características principales es el siguiente 3.2:

Función getDepth_panorama1.m

```
function [depth1,header1,ids1,planes1] = getDepth_panorama1(estructura1)

% N.B. ids = labels term in other functions

panoid = estructura1.Panoid;
maxDepth = estructura1.maxDepth;

st_url = sprintf('https://maps.google.com/cbk?output=json&cb_client=maps_sv&v=4&dm=1&pm=1&ph=1&hl=en&panoid=%s', panoid);
jsonFile = urlread(st_url);
rawDepth = loadjson(jsonFile);

% Append = in order to make the length of the array a multiple of 4
while (mod(length(rawDepth.model.depth_map),4) ~= 0)
    rawDepth.model.depth_map = [rawDepth.model.depth_map '='];
end

% Replace '-' by '+' and '_' by '/'
ids = find(rawDepth.model.depth_map == '-');
rawDepth.model.depth_map(ids) = '+';
ids = find(rawDepth.model.depth_map == '_');
rawDepth.model.depth_map(ids) = '/';

% Decode and de-compress the raw depth map
depthCompressed = base64decode(rawDepth.model.depth_map);
depthRaw = zlibdecode(depthCompressed);

% Parse the header and the planes information ->  parse = analizar
header1 = parseDepthHeader(depthRaw);
[planes1, ids1] = parsePlanes(header1, depthRaw);

% Compute the final depth map
depth1 = computeDepthMap(header1, planes1, ids1, maxDepth);

end
```

Código de interés 3.2: Obtención del archivo .json y decodificación

Obviamente existe una función idéntica para el panorama 2 pero con referencia a la estructura 2. Al final de la función puede observarse una llamada a la función ***computeDepthMap*** que cuenta como argumentos el header, los planos de referencia, sus identificadores y la profundidad máxima (maxDepth). Esta profundidad máxima está especificada por el usuario en la base de datos de Excel.

URL panorama 2	Maximum Depth	Type of change	Low Resolution
https://www.google.es/maps/@52.4108244,-1.51106	40	Structural change	1
https://www.google.es/maps/@41.8894691,12.49053	40	No changes	1
https://www.google.es/maps/@36.718872,-4.412251	40	Structural change	1
https://www.google.es/maps/@40.4271186,-3.69601	40	Structural change	1
https://www.google.es/maps/@-22.897909,-43.19759	40	No changes	1
https://www.google.es/maps/@37.6730333,-1.69899	40	Structural change	1
https://www.google.es/maps/@34.0426422,-118.266	40	Repairements	1
https://www.google.es/maps/@34.0451411,-118.264	40	Structural change	1
https://www.google.es/maps/@34.0451877,-118.264	40	Structural change	1
https://www.google.es/maps/@34.0454724,-118.265	40	Structural change	1

Figura 3.12: Máxima profundidad de la base de datos

Tras hacer diversas pruebas, se estableció un valor de 40 metros como distancia para ya considerar máxima profundidad. Para valores mayores el error era importante, teniendo en cuenta que este varía de forma cuadrática con la distancia.

Ahora, para el cálculo de la profundidad de cada punto se considera su plano asociado y se computa la distancia de la formada indicada en la siguiente función 3.3:

Función computeDepthMap.m

```
function [depth] = computeDepthMap(header, planes, ids, maxDepth)

depth = zeros(header.height, header.width);

for h=0:header.height-1

    for w=0:header.width-1

        planeId = ids(h*header.width+w+1);
        phi = 2*pi*((header.width-w-1)/(header.width-1)) + .5*pi;
        theta = pi*((header.height-h-1)/(header.height-1));
        v = [sin(theta)*cos(phi), sin(theta)*sin(phi), cos(phi)];

        if (planeId > 0)

            % Get plane parameters
            plane = planes(planeId+1);

            % Compute ray-plane intersection
            t = plane.d / (v*plane.n');
            v = v*t;

            % Fill the depth information
            z = v*v';
            depth(h+1,w+1) = sqrt(z);
        else
            depth(h+1,w+1) = maxDepth;
        end
    end
end
```

Código de interés 3.3: Cálculo del mapa de profundidad

Siendo la matriz ***depth*** una matriz bidimensional de 512 x 256 elementos (cada elemento de la matriz presenta la distancia asociada a cada pixel de la imagen panorámica), y por tanto, el **mapa de profundidad**. Sin embargo, esta información no sirve de nada si no se traduce esta **distancia a un código de colores** para poder visualizar dicha imagen. Para ello se cuenta con la siguiente función 3.4:

Función displayDepthMap.m

```
function [imgDepth] = displayDepthMap(depth, maxDepth)

[nr_rows, nr_cols] = size(depth);
imgDepth = uint8(zeros(nr_rows, nr_cols, 3));

for h=1:nr_rows
    for w=1:nr_cols
        di = depth(h,w);

        if (di==maxDepth)  % Purple sky
           imgDepth(h,w,1) = 255;
           imgDepth(h,w,2) = 0;
           imgDepth(h,w,3) = 255;
        end

        if (di ~= maxDepth)
        value = di*255/50;
        imgDepth(h,w,1) = uint8(value);
        imgDepth(h,w,2) = uint8(value);
        imgDepth(h,w,3) = 255;
        end
    end
end
```

Código de interés 3.4: Representación con colores del mapa de profundidad

En primer lugar, se toman las medidas de la matriz bidimensional (512 x 256) y se elabora una matriz nula tridimensional, es decir, tres planos nulos, cada uno con 512 x 256 elementos.

Con un doble *for* anidado, para cada elemento se extrae de la matriz *depth* su distancia correspondiente. **Si dicha distancia es igual a la *maxDepth*** (en este caso 40 metros), el color es único: 255 0 255, que en modelo RGB corresponde al morado (código elegido por el usuario, pero se podría poner cualquier otro). Por otra parte, **si la distancia de dicho elemento es distinta a la máxima profundidad asignada**, se calculará un valor a partir de la siguiente fórmula:

$$\text{value} = d_i \cdot \frac{255}{50} \tag{3.3}$$

Siendo d_i la distancia del elemento, el numerador 255 porque el modelo RGB presenta un byte de información por cada plano (de 0 a 255) y el denominador es arbitrario, aunque con dicho valor los resultados del mapa son bastante buenos.

Es muy importante tener en cuenta una cosa: **El color "infinito"** (en este caso correspondiente al morado) **no sólo se asignara a aquellos objetos o estructuras que se encuentren a una distancia exactamente igual a 40 metros**, sino que tal y como se ha elaborado la función *computeDepthMap*, **todo aquello que esté por encima de 40 m** (por ejemplo el cielo, que tiene una distancia evidentemente infinitamente mayor a 40 m) se le asignará dicho color.

Por ejemplo, en el caso del cielo, no importa la máxima profundidad que se pusiera que nunca sería computable (por propia física) por lo tanto en la matriz *depth* sus elementos siempre tendrían asignados ese valor de *maxDepth* y finalmente el color infinito asociado.

Algunos ejemplo de panorámica y sus respectivos mapas de disparidad son los siguientes:

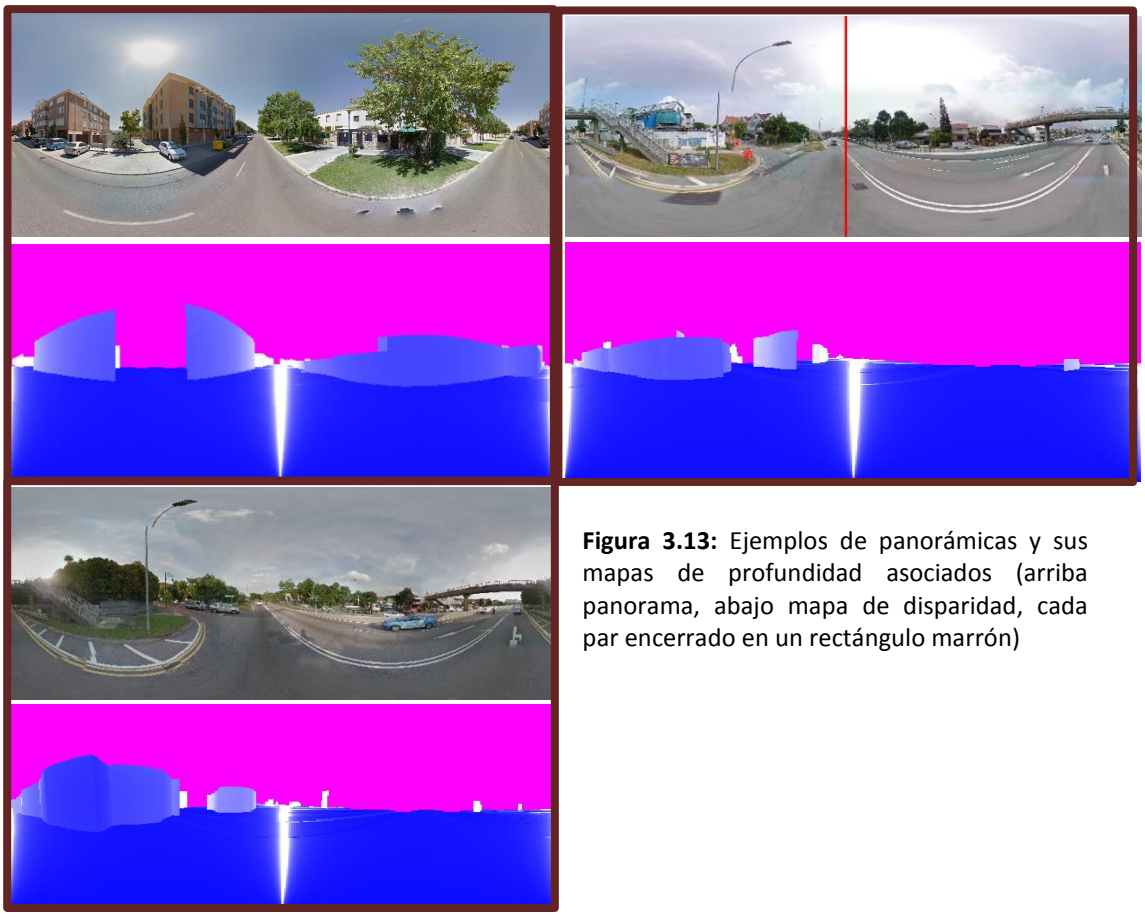


Figura 3.13: Ejemplos de panorámicas y sus mapas de profundidad asociados (arriba panorama, abajo mapa de disparidad, cada par encerrado en un rectángulo marrón)

Cuanto más definidas están las estructuras (más grandes y cerca de la cámara), **menos vegetación, transeúntes y vehículos** (tres cosas que el código tiende a evitar), **los resultados son mejores.**

Por último, cabe comentar que dentro de la aplicación, la interfaz *Initial information* creada calcula todas estas imágenes comentadas anteriormente (panorama 1, panorama 2, mapa de profundidad 1, mapa de profundidad 2), permitiendo mostrarlas, proporcionando su información inicial. Esta interfaz será detallada en el capítulo *6: Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea* donde se trata el tema del alineamiento.

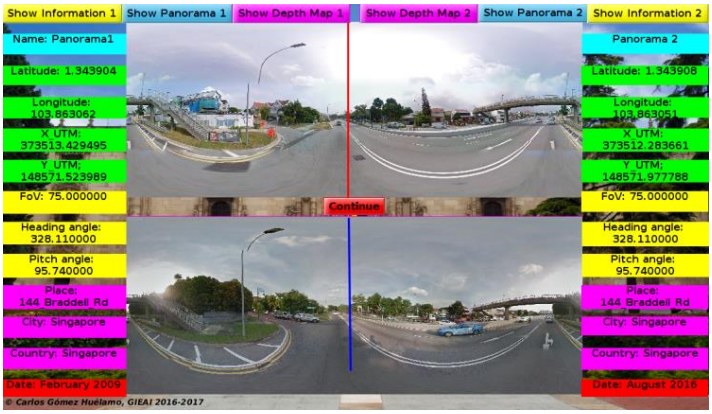


Figura 3.14: Interfaz *Initial information*

3.4.3. Reconstrucción 3D mediante nube de puntos:

Adicionalmente al mapa de profundidad, gracias al trabajo [4] y con unas pequeñas modificaciones (incluyendo la función de la nube de puntos, máxima profundidad, etc…) se puede representar a grandes rasgos la nube de puntos de dicho mapa de profundidad usando el siguiente código 3.5:

Función Pointcloud_CGH.m

```
[depth,header,ids,planes] = getDepth(panoid, maxDepth);

for ydepth=0:header.height-1

    lat=(ydepth*180/header.height)-90; % Both latitude and radious are calculated in the same row
    r=cos(lat*pi/180);


    for xdepth=0:header.width-1

        di = depth(ydepth+1,xdepth+1);

        index=(ydepth+header.height*xdepth+1);

        if(di<maxDepth) % maxDepth should be adjusted conveniently to the situation. In this case 40 m is a good number

            lng=(1-xdepth/header.width)*360-180; % The longitude value is the same for the 512 elements of a column

            xpostraspuestocondi(index)=-(r*cos(lng*pi/180))'*di;
            ypostraspuestocondi(index)=(sin(lat*pi/180))'*di;
            zpostraspuestocondi(index)=(r*sin(lng*pi/180))'*di;

        elseif (di>maxDepth || di==maxDepth)

        xpostraspuestocondi(index)=0;
        ypostraspuestocondi(index)=0;
        zpostraspuestocondi(index)=0;

        end
    end
end

TT=pointCloud([xpostraspuestocondi(:),zpostraspuestocondi(:),-ypostraspuestocondi(:)]);

figure

pcshow(TT)

titulo1='Y axis';
titulo2='X axis';
titulo3='Z axis';
ylabel(titulo1);
xlabel(titulo2);
zlabel(titulo3);
```

Código de interés 3.5: Cálculo de la nube de puntos y representación

En la nube de puntos sólo toman coordenadas x y z computables aquellas distancias menores a la máxima profundidad. Aquellas que sean mayores o iguales, tomaran el vector nulo $[0\ 0\ 0]$. Finalmente se elabora una nube de puntos con los vectores $[x_n\ y_n\ z_n]$, siendo n el elemento correspondiente dentro de la matriz de 512 x 256.

Un ejemplo de panorámica, mapa de profundidad y reconstrucción 3D:

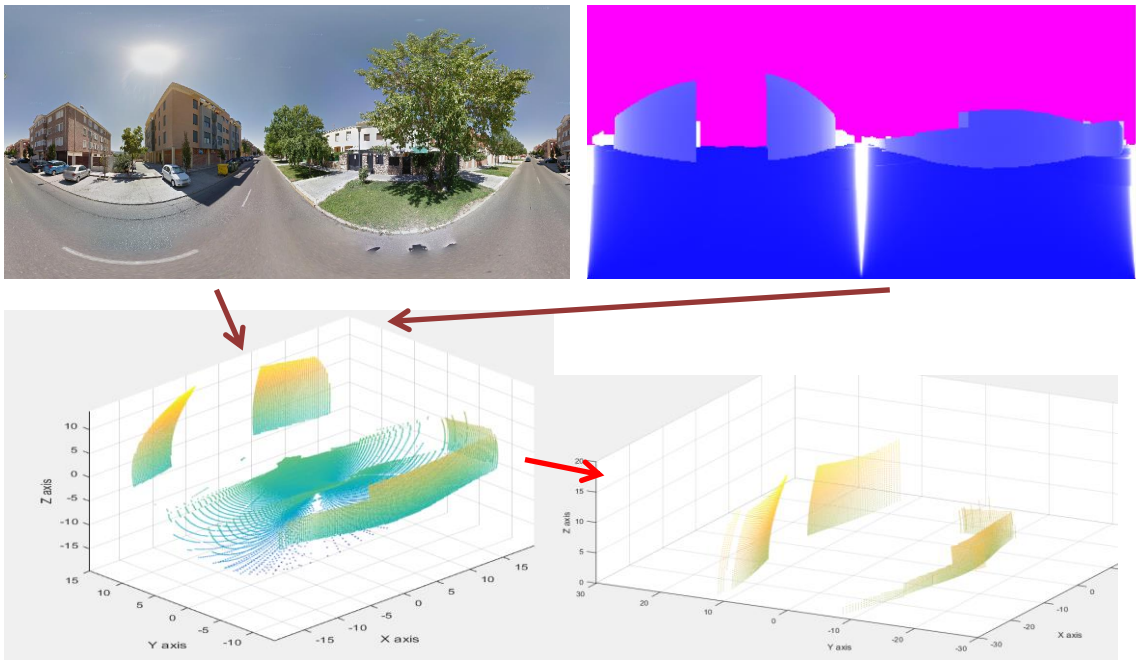


Figura 3.15: Reconstrucción 3D

Si se acerca el cursor al centro en la tercera imagen, puede observarse **círculos concéntricos que convergen** en un único punto (señalado con una flecha roja). Este punto evidentemente es la **posición de la cámara** a partir del cual se ha hecho el barrido 360 º y posteriormente se ha determinado el mapa de profundidad y la nube de puntos. Además, **si se elimina el semieje z negativo** y sólo se deja la parte positiva, que equivaldría a descartar la información del suelo, se obtiene una fiel reconstrucción 3D correspondiente a la representación tridimensional a partir de la estructura planar del mapa de profundidad. Evidentemente, cuanto mejor sea el mapa de profundidad, mejor será la reconstrucción 3D.

Capítulo 4

Sintetización de imágenes a partir de imágenes panorámicas:

Tras haber tratado en el capítulo anterior acerca del mapa de profundidad y la imagen panorámica, se tiene toda la materia prima para elaborar secciones o vistas sintéticas a partir de dicho panorama.

En este capítulo se abordará el **problema del reconocimiento visual del lugar a gran escala en situaciones donde la escena** (es decir, a lo largo del TFG, los términos sección, imagen y escena serán indistintos) **sufre un importante cambio de aspecto, por ejemplo debido a la iluminación** (el color de un edificio se resalta de una forma muy diferente si el cielo está despejado o nublado), cambios de estaciones (que son realmente las responsables del cambio en el tiempo), el paso de los años (envejecimiento), modificaciones estructurales (construcciones o demoliciones de edificios), … Tales situaciones representan el mayor reto para los actuales métodos de reconocimiento de lugares a gran escala.

Para abordar ese problema, se estudiarán los siguientes puntos:

> **1.** Demostrar que la **coincidencia entre dos secciones** (de la misma localización) **que presentan grandes cambios en el aspecto de la escena** (por ejemplo fuerte cambio en la iluminación, modificación estructural grande, abundante vegetación, aparición de transeúntes) **se hace mucho más fácil cuando la imagen de consulta** (que en este caso corresponde a la sección antigua, es decir, las tres imágenes sintetizadas del primer panorama) **y la sección nueva** (cada una de las tres imágenes sintetizadas del segundo panorama) **se representan aproximadamente desde el mismo punto de vista.**
>
> **2.** A partir del concepto de que **el punto de vista sea aproximadamente igual para ambas**, se implementará el código desarrollado en el trabajo [8] "24/7 place recognition by view synthesis" para conseguir una **eficiente síntesis de las secciones así como una compacta representación de la imagen.**

4.1. Problemas actuales en el reconocimiento de imágenes:

En estos años recientes la visión artificial ha encontrado un tremendo progreso en el problema visual de reconocimiento a gran escala. Hay que tener en cuenta que en la ciencia en general cuantos mayores son los avances en un campo, más se da cuenta el usuario del largo camino que queda aún por recorrer para poder asemejar una computadora al cerebro humano.

Si se muestra dos imágenes a una persona las cuales presentan entre sí un gran cambio, por ejemplo una más pequeña que la otra, trasladada, rotada, con vegetación, **el cerebro humano es tan complejo que fácilmente** (en mayor o menor medida) **será capaz de identificar características invariantes locales** comunes en ambas imágenes que le servirán para comparar la periferia de dicha características y determinar lo que ha cambiado o se ha mantenido invariante. Esto en cambio es una **tarea inmensamente ardua para una computadora** y es precisamente el tema del siguiente capítulo 5 *Extracción y descripción de puntos característicos en vistas sintéticas*, que trata la extracción de características de la imagen. Serán estos extractores de características los que identifiquen/reconozcan puntos característicos de la imagen a través de cambios moderados en el punto de vista, escala u oclusión parcial por otros objetos (como pueden ser transeúntes, vehículos o vegetación).

4.2. Concepto de síntesis de imágenes:

A continuación se muestran cuatro imágenes que ponen de manifiesto el concepto de síntesis de una imagen desde el punto de vista de otra.

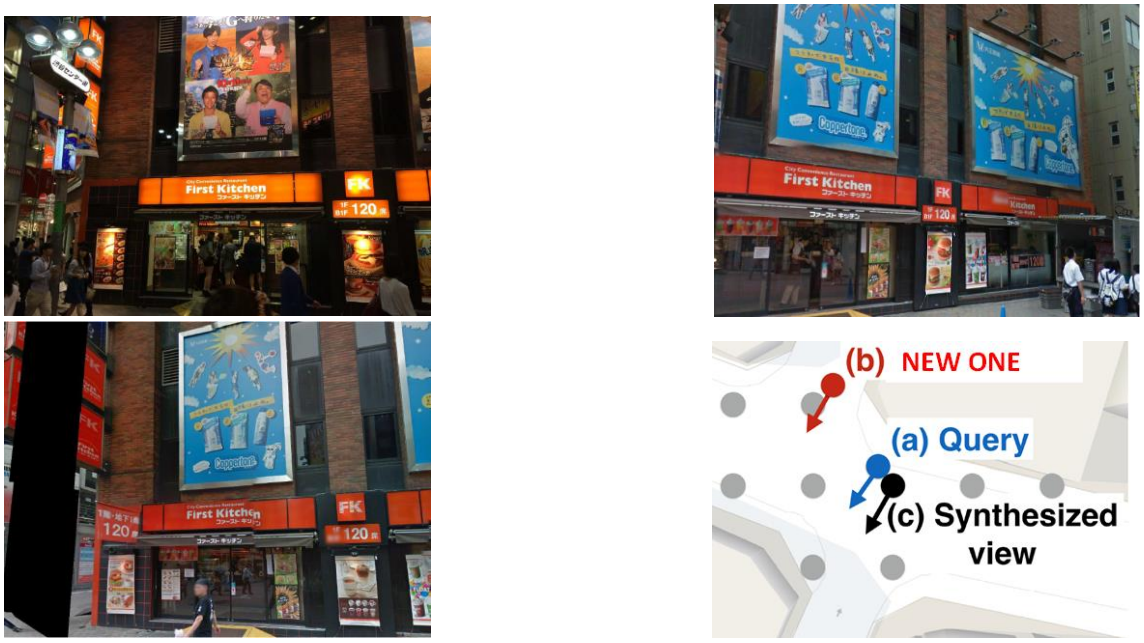


Figura 4.1: Síntesis de imágenes

La figura 4.1 presenta cuatro imágenes:

- En la esquina superior izquierda se encuentra la **imagen antigua (en este TFG será la del panorama 1)**, que es considerada como la *query* image (denominación anglosajona a la imagen de consulta o referencia).
- En la esquina superior derecha está la **sección nueva (panorama 2)**. El cerebro humano rápidamente es capaz de identificar que el cartel grande rectangular verticalmente colocado, así como el letrero donde se puede leer *"First Kitchen"* de la imagen de referencia coincide con la información de la primera mitad izquierda de la nueva imagen. Por tanto el punto de vista está claramente rotado.
- Mediante el código usado de este paper (cuya implementación matemática no es objetivo en este TFG, sino tan sólo sus características principales) la **imagen nueva** (esquina superior derecha) **es sintetizada (es decir, creada artificialmente) a partir del punto de vista de la imagen antigua** (esquina inferior izquierda), igualando en la medida de lo posible la rotación/traslación y logrando así igualar el vector director del punto de vista.
- La última imagen (esquina inferior izquierda) muestra los **vectores directores de los distintos puntos de vista: Tras la sintetización, la imagen nueva cuenta con un punto de vista** que ha rotado hasta obtener la misma dirección y prácticamente se ha trasladado al mismo punto (aunque no es perfecto ya la flecha azul y negra no se superponen).

Por tanto, el proceso de **síntesis puede definirse como el conjunto de operaciones matemáticas que permiten crear una imagen artificial**, que virtualmente no existe, derivada de una imagen original modificando alguno de sus parámetros, ya sea incluyendo objetos sintéticos, modificando la iluminación o, como en este caso, cambiando el punto de vista.

Las franjas negras suelen aparecer cuando tras la síntesis, el código intenta adaptar a las zonas de la imagen original secciones de la imagen nueva que no existen (en el ejemplo 4.1, la sección nueva a la izquierda del primer cartel azul hay poca pared, mientras que en la imagen original hay bastante a la izquierda, por tanto aparece una franja negra). Este hecho se comprobará perfectamente en el capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea* donde se trate el alineamiento entre dos secciones y las franjas negras resultantes de dicho alineamiento.

Solucionando el problema del punto de vista se tendría una implicación práctica muy significativa y es que los **cambios más importantes en la apariencia de la escena son más fáciles de localizar desde puntos de vista similares.**

Si se sintetiza (y posteriormente se refuerza con un alineamiento conveniente) una misma geolocalización en varias pasadas temporales, **la base de datos creada podría usarse para analizar cambios en el tiempo en aplicaciones de arquitectura, arqueología y planos urbanos, históricas, ingenieriles, etc** o incluso de forma altruista para visualizar el mismo lugar con diferentes iluminaciones, estaciones o el simple hecho de ver la zona atrás en el tiempo. Ese análisis de cambios se realiza mediante la comparación con estos extractores de características, que funcionarán mejor cuanto más idénticos sean los puntos de vista de la imagen del panorama 1 y del panorama 2.

Existen dos cuestiones principales acerca de la sintetización de imágenes:

1. ¿Cómo se puede eficientemente sintetizar puntos de vista virtuales para una base de datos tan grande?

2. ¿Cómo se deben representar las imágenes sintéticas de un modo que sea robusto a los cambios grandes en la apariencia de la escena?

Para abordar estas cuestiones, el trabajo [8] en primer lugar desarrolla un **método de síntesis de vistas** que pueda renderizar vistas virtuales **a partir de la información recogida en las imágenes panorámicas de GSV y sus mapas de profundidad asociados, no requiriendo de un modelo 3D preciso de la escena.**

Es importante tener en cuenta una cosa: Por definición la renderización de imágenes es el proceso de obtención de imágenes mediante el cálculo de iluminación partiendo de un modelo 3D. Sin embargo, con este código no es necesario la nube de puntos del panorama (aunque en el capítulo anterior se elaboró un código que modelaba de una forma aproximada el mapa de profundidad en tres dimensiones), sino que basta con la información del panorama y la distancia de cada pixel a la cámara, que no es más que la información contenida en el mapa de profundidad.

Aunque a menudo las imágenes resultantes están ligeramente distorsionadas (por ejemplo esas franjas negras, que una farola esté cortada,…) **la imagen sintética resultante es suficiente para la tarea de reconocimiento de cambios a gran escala.** Además, la **ventaja clave de este TFG** es que la herramienta Google Street View, al estar disponible para todo el mundo, abre la posibilidad de un **verdadero reconocimiento visual a escala planetario.**

En segundo lugar, **para representar las imágenes el código usa descriptores basados en gradientes locales (SIFT en este caso) a través de múltiples escalas**. Estos descriptores son los más robustos frente a grandes cambios en la apariencia debido a la iluminación, envejecimiento, modificaciones estructurales, etc… ya que no se basan en la detección repetible de características invariantes locales como es el Laplaciano de Gauss. En el capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea* donde se trata el alineamiento se puede comprobar también que los mejores resultados de extracción de características son obtenidos con las técncias SURF y SIFT, descriptores basados en gradientes locales.

4.3. Método para la síntesis de imágenes y representación:

El reconocimiento de lugares a gran escala es a menudo formulado como un problema derivado de la recuperación de imágenes a partir de una imagen de consulta (imagen del primer panorama en este caso) que sirve de referencia y una imagen del segundo panorama que se tiende a alinear con esta primera. Ambos panoramas provienen de la base de datos de GSV.

La estructura 3D del entorno (para un determinado punto temporal) podría ser también reconstruida de antemano y comparar entonces las imágenes de otro punto temporal para esa comparación de cambios.

Por tanto, el etiquetado que se hará al final del TFG (generando el *groundtruth* que contiene la información de los cambios) se puede hacer de dos formas:

1. Sintetizando ambos panoramas y comparándolos.

2. Sintetizando un único panorama y elaborando una nube de puntos tridimensional.

Ambas opciones han demostrado un excelente rendimiento en el alineamiento entre dos imágenes cuando hay cambios moderados en la escala y punto de vistas, cambios modelados por los detectores de características invariantes locales. Sin embargo, la coincidencia entre las variaciones de apariencia tales como cambios en la iluminación, estaciones o incluso el envejecimiento de las estructuras es aún un desafío. Para este TFG se ha elegido la primera opción por ser la menos compleja de ambas.

Para sintetizar estas vistas adicionales se usarán principalmente dos tipos de datos:

1. Imagen panorámica obtenida previamente. Cada panorama captura 360 º x 178 º horizontal y vertical ángulo de visión respectivamente (básicamente una semiesfera), y tiene el tamaño de 3.584 x 1.536 píxeles (en baja resolución).

2. Mapa de profundidad asociado a dicha imagen panorámica. El mapa de profundidad está codificado como un conjunto de parámetros del plano 3D (vector normal y distancia para cada plano) con una resolución de 512 x 256 píxeles.

Todas las vistas de una posición particular de la nueva posición de la cámara (dicho de otra forma, la cámara virtual), están sintetizadas a partir del panorama y del mapa de profundidad más próximo de la imagen de GSV más cercana.

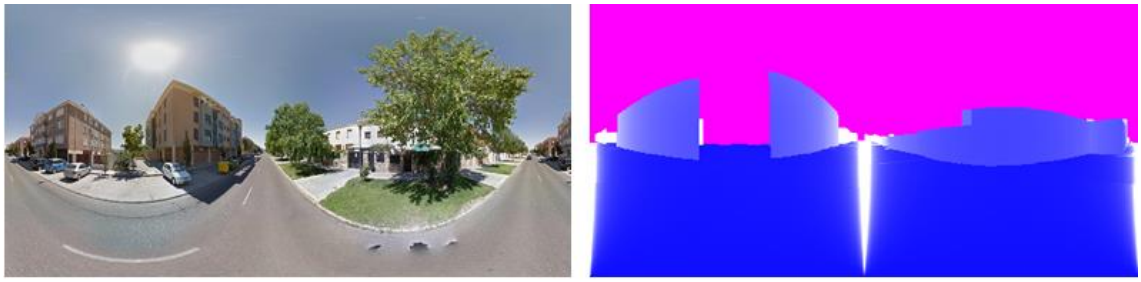


Figura 4.2: Datos de entrada para la síntesis

Como se puede observar, el mapa de profundidad debe ser mejorado puesto que aún tiene margen de mejora para asimilar la complejidad existente en la imagen panorámica, repercutiendo directamente en la síntesis. Por tanto, esta será una de las futuras mejoras, basándose en el desplazamiento de un mismo panorama, es decir, misma geolocalización y momento temporal, determinando la profundidad a partir de este desplazamiento.

A pesar de que el mapa de profundidad proporciona una estructura 3D basta, tal y como se demostrará en posteriores capítulos los descriptores resultantes y la aplicación de estas secciones en la herramienta creada para el etiquetado serán altamente satisfactorios. Además, datos están disponibles a nivel mundial, la herramienta abre la posibilidad de una síntesis de vistas, reconocimiento de imágenes y etiquetado de diferencias a nivel mundial.

El proceso de la síntesis de vistas está estructurado en dos pasos: Síntesis de la localización virtual de la cámara y a continuación síntesis de las vistas individuales.

1. Sintetizar la localización virtual de la cámara es equivalente a trasladar su posición y rotar pertinentemente el vector director de la vista (flecha negra en la imagen) para que sea más o menos paralelo al de la imagen de consulta (flecha azul en la imagen).

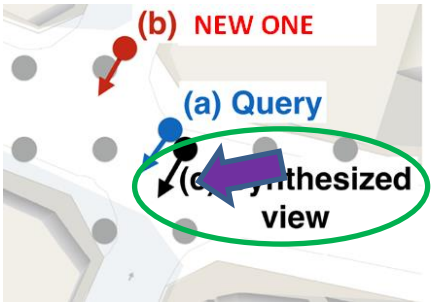


Figura 4.3: Localización de la cámara virtual

Para esta misión, **primero se generan las posiciones de cámaras candidatas en una red regular de 5 x 5 m** en el mapa que cubre la posición original de la cámara según la geolocalización de GSV. Sólo se generan las posiciones de la cámara que están en un radio de 20 m desde la trayectoria original de GSV, donde la trayectoria es obtenida conectando las posiciones de la cámara de la calle vecina. Esto es así porque más allá de un radio de 20 m a menudo se generan ruidos significativos en las imágenes sintéticas.

Por otra parte, el mapa de profundidad disponible es usado para descartar aquellas posiciones de la cámara que caerían dentro de los edificios. El paper 24/7 Tokyo muestra un gráfico de las posibles posiciones virtuales de la nueva cámara.

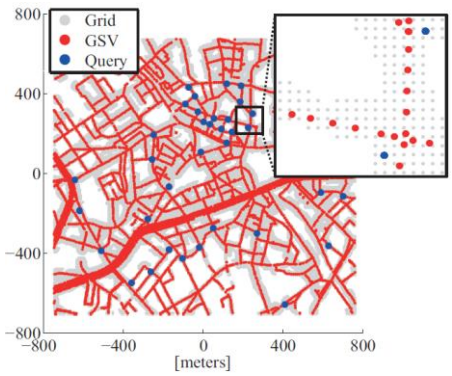


Figura 4.4: Posiciones de la cámara candidatas

En este gráfico se muestran en rojo las posiciones de las imágenes original en GSV, las posiciones de las vistas sintéticas (red de 5 x 5 m) mostradas en gris y las posiciones de las imágenes de consulta en azul.

2. El proceso para la síntesis de imágenes virtuales se hace a partir del trazado de rayos estándar con interpolación bilineal. En el campo de la fotografía y mundo de la imagen digital, el proceso de interpolación es usado para conseguir un tamaño mayor de la imagen inicial, rellenando la información que falta con datos «inventados» a partir de un algoritmo específico.

Se denomina **interpolación** al proceso de **obtención de nuevos datos partiendo de un conjunto discreto de estos mismos.** Extrapolando el concepto a una función, sabiendo dos valores en el eje X y su equivalencia en el eje Y se puede saber la equivalencia de un punto en el eje X que se encuentre intermedio entre los dos puntos previos).

La interpolación más sencilla es la lineal, donde se usa un polinomio de interpolación de grado uno, tal que el valor que toma un punto x del eje X entre dos extremos x1 y x2 es el siguiente:

$$f(x \mid x_1; x_2) = f(x_1) + \frac{f(x_2) - f(x_1)}{x_2 - x_1} \cdot (x - x_1) \qquad (4.1)$$

Esta fórmula se aprecia mejor con el siguiente gráfico:

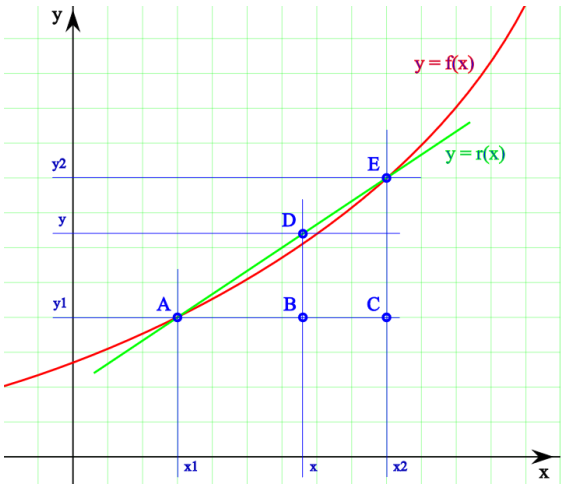


Figura 4.5: Interpolación lineal

Siendo y=f(x) la función real, y=r(x) el polinomio de primer grado (es decir, una recta) para aproximar dicha curva y f(x) la incógnita a determinar.

4.3.1. Interpolación Nearest Neighbor:

La interpolación más simple en el procesado de imágenes es el **método del vecino más cercano (Nearest neighbor),** el cual simplemente es capaz de **aumentar el tamaño de cada pixel**. Para ello este método se fija en el pixel más cercano al punto interpolado. Un ejemplo gráfico del aumento del tamaño de la imagen es el siguiente:

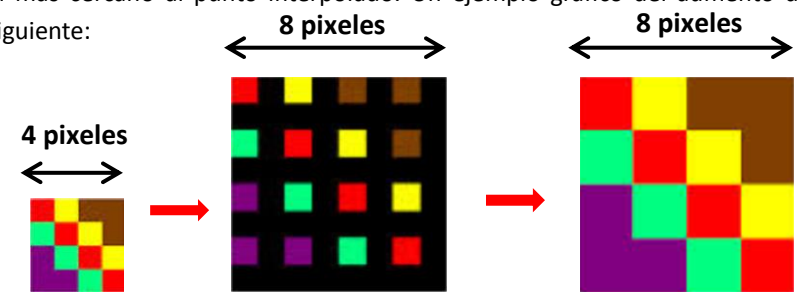


Figura 4.6: Algoritmo Nearest Neighbor

El primer cuadrado tiene 4 píxeles de lado, a continuación se dobla el número de píxeles por lado. Evidentemente los 16 primeros píxeles (4·4 = 16) se mantienen dentro de los 64 píxeles actuales (8·8=64), pero los restantes 48 es información desconocida. El código (es decir, el color) de cada uno de los píxeles negros será el color del píxel original más cercano que tenga. Se debe empezar a construir este método desde la periferia hacia el centro.

Un ejemplo más realista de este método es el siguiente:

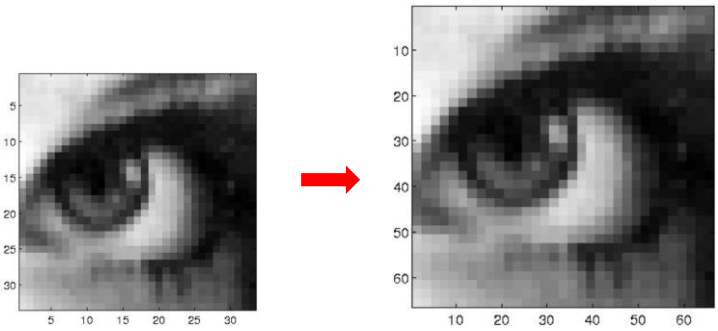


Figura 4.7: Algoritmo Nearest Neighbor aplicado a imágenes

Se puede observar que se ha duplicado el tamaño de la imagen. La equivalencia gráfica del método Nearest neighbor es la siguiente:

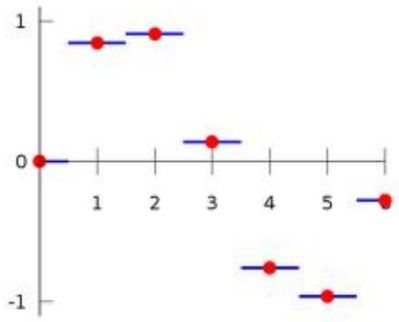


Figura 4.8: Equivalencia gráfica de Nearest Neighbor

Siendo el método más básico el comentado anteriormente. Es por ello que este método sólo es usado en aplicaciones donde se requiere poco tiempo de procesado o tan solo una burda aproximación en el resultado.

4.3.2. Interpolación bilineal:

El siguiente método en eficiencia es el **método bilineal** [9], el cual es uno de los más usados en el proceso de imágenes por ser **equilibrado en términos de buen resultado y moderado tiempo de procesado.**

La interpolación bilineal matemáticamente es una **extensión de la interpolación lineal antes citada para interpolar funciones de dos variables** (por ejemplo dependientes tanto de x como de y). Mientras que para la interpolación lineal se podía extrapolar a obtener datos intermedios en una gráfica 1D, la interpolación lineal, dado el trabajo con dos variables, se puede asemejar a la **obtención de datos intermedios en una malla regular 2D** (igualmente para la interpolación cúbica se podría asemejar a la obtención de datos en un cubo regular 3D).

La idea principal es realizar una interpolación lineal en una dirección y después en la otra. Aunque ambos procesos son lineales, la combinación de estas dos interpolaciones lineales no es lineal, sino cuadrática.

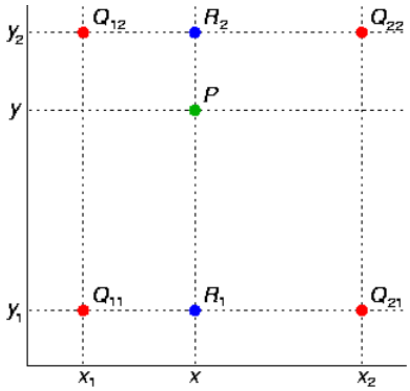


Figura 4.9: Interpolación bilineal

En el ejemplo anterior se pretende encontrar el valor del punto verde P, el cual depende de las variables x e y, es decir, $\mathrm{P}(\mathrm{x},\mathrm{y})$, siendo conocidos cuatro valores: $\mathrm{Q}_{11} = \mathrm{f}(\mathrm{x}_1,\mathrm{y}_1)$, $\mathrm{Q}_{12} = \mathrm{f}(\mathrm{x}_1,\mathrm{y}_2)$, $\mathrm{Q}_{21} = \mathrm{f}(\mathrm{x}_2,\mathrm{y}_1)$, $\mathrm{Q}_{22} = \mathrm{f}(\mathrm{x}_2,\mathrm{y}_2)$.

En primer lugar se hace una interpolación lineal en la dirección de X:

$$f(R_1) \approx \frac{x_2-x}{x_2-x_1} \cdot f(Q_{11}) + \frac{x-x_1}{x_2-x_1} \cdot f(Q_{21}) \tag{4.2}$$

$$f(R_2) \approx \frac{x_2-x}{x_2-x_1} \cdot f(Q_{12}) + \frac{x-x_1}{x_2-x_1} \cdot f(Q_{22}) \tag{4.3}$$

Siendo respectivamente $R_1 = (x, y_1)$ y $R_1 = (x, y_2)$. Es muy importante entender que los puntos R iguales a dos coordenadas implican que ambas coordenadas son desconocidas y cambiantes (por ejemplo que fueran dependientes de una variable independiente como puede ser el tiempo).

A partir de las dos interpolaciones anteriores se interpola en el eje y:

$$f(P) \approx \frac{y_2-y}{y_2-y_1} \cdot f(R_1) + \frac{y-y_1}{y_2-y_1} \cdot f(R_2) \tag{4.4}$$

Sustituyendo las expresiones obtenidas anteriormente por $f(R_1)$ y $f(R_2)$ respectivamente en la última fórmula, la fórmula general para hallar el valor de un punto intermedio $A = \mathrm{f}\big(\mathrm{P}(\mathrm{x},\mathrm{y})\big)$ queda de la siguiente forma:

$$\begin{aligned} A = \mathrm{f}\big(\mathrm{P}(\mathrm{x},\mathrm{y})\big) \approx \frac{1}{(x_2-x_1)\cdot(y_2-y_1)} \cdot \big(& f(Q_{11})\cdot(x_2-x)\cdot(y_2-y) + \\ & f(Q_{21})\cdot(x-x_1)\cdot(y_2-y) + \\ & f(Q_{12})\cdot(x_2-x)\cdot(y-y_1) + \\ & f(Q_{22})\cdot(x-x_1)\cdot(y-y_1) \quad \big) \end{aligned} \tag{4.5}$$

Evidentemente, la **fórmula final resultante es igual si primero se obtienen el valor de los puntos R mediante interpolación lineal en la dirección X y posteriormente interpolación lineal en el eje Y que conmutando el proceso.**

Si se simplifica el sistema de coordenadas escogiendo uno tal que las cuatro esquinas de la malla sean los puntos (0,0), (0,1), (1,0) y (1,1), la fórmula de interpolación se simplifica según la siguiente ecuación:

$$\mathrm{f}\big(\mathrm{P}(\mathrm{x},\mathrm{y})\big) \approx f(0,0)\cdot(1-x)\cdot(1-y) + f(1,0)\cdot x\cdot(1-y) + f(0,1)\cdot(1-x)\cdot y + f(1,1)\cdot x\cdot y \quad (4.6)$$

O en forma matricial:

$$\mathrm{f}\big(\mathrm{P}(\mathrm{x},\mathrm{y})\big) \approx \begin{bmatrix}1-x & x\end{bmatrix}\cdot\begin{bmatrix}f(0,0) & f(0,1)\\ f(1,0) & f(1,1)\end{bmatrix}\cdot\begin{bmatrix}1-y\\ y\end{bmatrix} \quad (4.7)$$

Sin embargo, aunque el apellido de la interpolación es bilineal lo que podría sugerir connotaciones lineales, **esta interpolación es no lineal porque de hecho está generada a partir del producto de dos funciones lineales** (tal y como se ha podido ver en el ejemplo anterior con el múltiplo de las interpolaciones [4.5]):

$$(a_1\cdot x + a_2)\cdot(a_3\cdot y + a_4) \quad (4.8)$$

Fórmula que se puede reescribir como:

$$b_1 + b_2\cdot x + b_3\cdot y + b_4\cdot x\cdot y \quad (4.9)$$

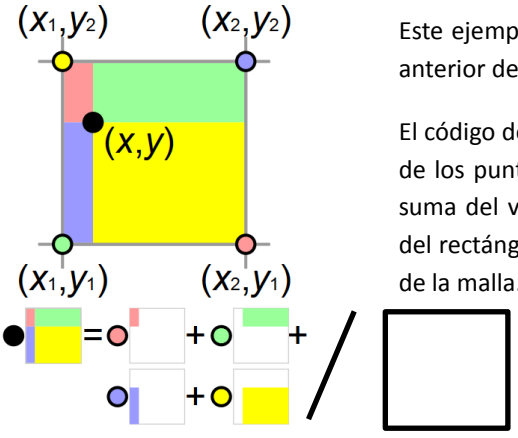
Siendo $b_1 = f(0,0)$, $b_2 = f(1,0) - f(0,0)$, $b_3 = f(0,1) - f(0,0)$ y $b_4 = f(0,0) - f(1,0) - f(0,1) + f(1,1)$.

Función claramente no lineal, puesto que no es un polinomio de primer orden.

En ambos casos ($(a_1\cdot x + a_2)\cdot(a_3\cdot y + a_4)$ o bien $b_1 + b_2\cdot x + b_3\cdot y + b_4\cdot x\cdot y$), el número de coeficiente a determinar son cuatro (a_1, a_2, a_3, a_4 *o bien* b_1, b_2, b_3, b_4) porque para determinar un pixel intermedio de una región se debe conocer el valor de cuatro puntos circundantes.

Evidentemente los procesos intermedios son interpolaciones lineales si se hacen paralelos al eje X (donde la Y es constante) o al eje Y (donde la X es constante). A lo largo de cualquier otra recta con pendiente, la interpolación resultaría cuadrática.

Finalmente, el resultado de una interpolación bilineal es independientemente del orden de las interpolaciones previas, si primero se han hecho sobre el eje X o el Y, siendo la fórmula y el resultado aproximadamente los mismos.



Este ejemplo demuestra de una forma muy visual y geométrica la anterior demostración matemática:

El código de colores del punto negro es dependiente de los valores de los puntos circundantes. El valor del punto negro es igual a la suma del valor en cada punto coloreado multiplicado por el área del rectángulo del mismo color, todo ello dividido por el área total de la malla.

Figura 4.10: Interpolación bilineal de forma gráfica

Si se traduce los anteriores términos al campo de la visión artificial, la interpolación lineal para determinar el valor de un pixel tiene en cuenta el valor de los píxeles conocidos en una malla de 2x2 píxeles, es decir, 4 píxeles adyacentes.

Por tanto, en la **síntesis de imágenes para el cálculo del valor del píxel se toma el promedio ponderado de estos cuatro píxeles** (está ponderado en función de la distancia del píxel al punto de interés, puesto que aportará aquel que esté más cercano) y se calcula el valor interpolado.

A continuación se puede ver un ejemplo del cálculo del píxel de interés, **donde las píxeles circundantes están a la misma distancia, por tanto la ponderación es la misma para todos**, y finalmente el valor interpolado no es más que la suma del valor de todos ellos dividido entre cuatro, o dicho de otra forma, su media aritmética.

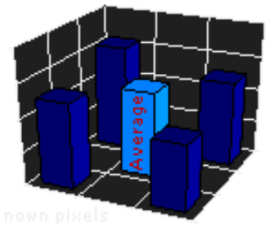


Figura 4.11: Cálculo del pixel a partir de cuatro adyacentes

Los resultados de este método para la anterior imagen ejemplo:

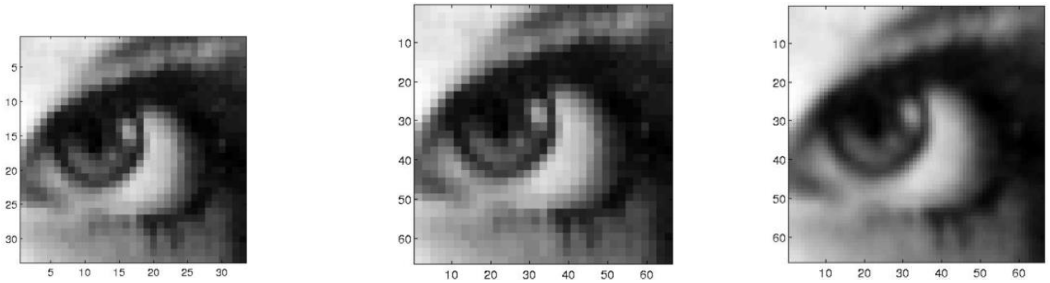


Figura 4.12: A la izquierda, imagen original. Centro, método Nearest Neighbor. A la derecha, método bilineal

Se puede observar como la nitidez es mucho mayor en el método bilineal que en el del vecino más próximo. Gráficamente se puede resumir de la siguiente forma:

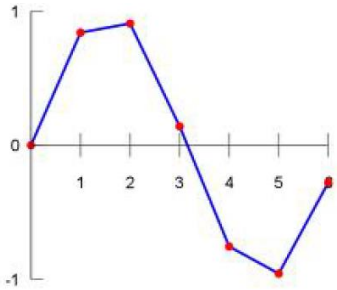


Figura 4.13: Equivalencia gráfica del método Bilineal

Mientras que en el método del vecino más cercano los valores alrededor de los datos conocidos tomaban el mismo valor, **ahora se establecen polinomios de primer grado** (básicamente rectas que unen los puntos), lo que mejora drásticamente la eficiencia del proceso. **La interpolación cúbica es**

mejor aún que la bilineal, pero en términos de equilibrio tiempo de procesado/resultados, la bilineal es suficiente en la mayoría de los procesos.

Retomando el proceso de la creación de vistas sintéticas, para cada pixel (que se está construyendo en este momento) **se emite un rayo desde el centro de la cámara virtual** (situada en la posición vertical seleccionada del paso uno), **el cual intersecta con la estructura 3D planar proporcionada por el mapa de profundidad a disposición**. A continuación, **este rayo proyecta la intersección producida** (con el mapa de profundidad) hacia el panorama de GSV y **finalmente se interpola bilinealmente el pixel que se quiere a partir de los cuatro píxeles adyacentes** (tal y como se ha visto en el proceso de interpolación bilineal).

4.3.3. Ejemplo de síntesis de imágenes en la aplicación creada:

En este TFG en concreto, **para cada localización de la cámara virtual se han generado tres perspectivas de imágenes (con un vector de yaw angle [0º 120º 240 º]** con las siguientes características:

Concepto	Valor
Anchura	640 píxeles
Altura	480 píxeles
Resolución horizontal	96 ppp (punto por pulgada)
Resolución vertical	96 ppp
Profundidad de cada pixel	24 bits (modelo RGB)

Tabla 4.1: Características de las imágenes sintetizadas

Siendo el campo de visión FoV elegido por el usuario, en este caso 35 grados y una inclinación de la cámara en la vertical (*Pitch angle*) igual a 5 grados.

Un ejemplo es el siguiente:

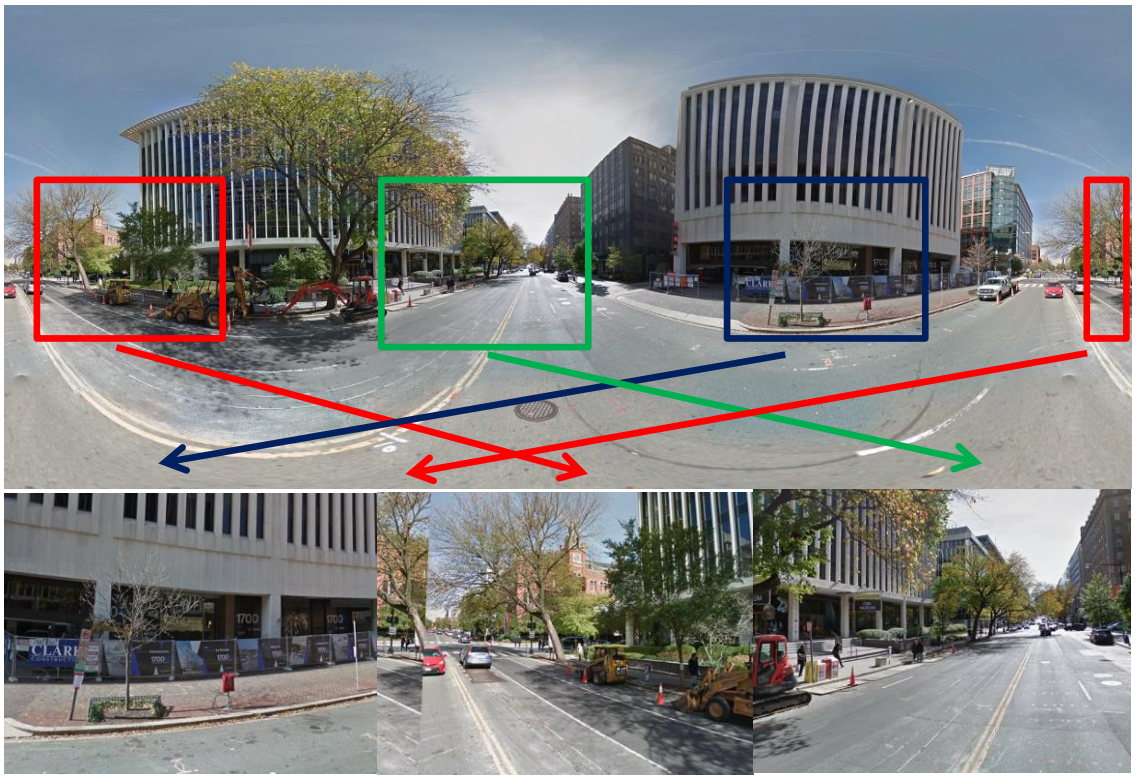


Figura 4.14: Ejemplo de síntesis de imágenes

La intención de escoger estos dos valores (FoV = 35 grados; Pitch angle = 5 grados) es captar la **información principal del panorama teniendo en cuenta que se van a sintetizar sólo tres secciones**, por lo que conceptualmente a cada sección le debería corresponder 120 º de los 360 º que equivale el panorama extendido.

Sin embargo, **un bajo campo de visión obtiene muy buenos resultados con edificios que están a cierta distancia**, por lo que prácticamente el 100 % de la información principal está recogida dentro de la imagen, además de **eliminar en su mayor parte la carretera** (por debajo) **y el cielo** (por encima), tal y como se puede apreciar en el cuadro verde (Tercera sección).

Sin embargo, en la primera sección (sección azul) por abajo la carretera está bien recortada pero aproximadamente un 60 % del edificio excede por encima, por lo que en este caso el campo de visión se debería aumentar elevando la inclinación vertical de la cámara, igual que pasa en cierta medida con la sección roja (segunda sección). Esto indica que una de las mejoras de este TFG es un **reconocimiento de la imagen para obtener la información inicial y elaborar una matriz de campos de visión y de ángulos de inclinación vertical para ajustar la síntesis de la imagen**, manteniendo por supuesto la resolución final de la imagen sintética (640 x 480 píxeles).

Otro de los problemas en la síntesis de imágenes es el ghosting, es decir, cuando una de las secciones toma parte de la información más oriental del panorama, información idéntica al extremo contrario:

Figura 4.15: Ejemplo de *ghosting*

Este problema es otra de las mejoras futuras del TFG, problema que radica en la información de los azulejos de la última columna que se descargan desde la API de Google Street View. En todos los panoramas es una constante, ese mismo recuadro se repite en ambos extremos como si estuviera solapado. Una **posible solución sería recortar la imagen panorámica con ayuda de MatLab**, justo el rectángulo rojo derecho. Sin embargo, el **problema persiste en el mapa de profundidad**, puesto que se extrae a partir del *panoid* del panorama actual (y por tanto con ese solapamiento), **por lo que habría que elaborar un mapa de profundidad propio (otra futura mejora)** y no usar la API proporcionada por GSV.

4.3.4. Código usado para la síntesis de imágenes en ambos panoramas:

Tal y como se tratará en el capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea* alineamiento, el usuario puede tomar parte en el proceso de síntesis de imágenes en tiempo real (no modificando el código), en concreto en el proceso de síntesis de las secciones del panorama 2. Para ello, indicará qué columna del segundo panorama corresponde a la

columna inicial del primero, calculada a partir de la función *at_gsv2viewsynth_init_col_panorama1.m* (© Dr. Pablo Fernández Alcantarilla), dejando constancia en ambos cual debe ser el *yaw angle = 0º* para que comience la síntesis de estas imágenes.

Tras calcular el *offset global* que se debe desplazar el segundo panorama para que el *yaw angle = 0º* en ambos casos coincida, se procede a la síntesis de estas imágenes:

1. Función createSynthViews_panorama1.m

Para el panorama 1, la función es muy sencilla, simplemente consiste en dos bucles anidados. Como en este caso el vector del pitch angle sólo tiene un elemento (5 grados) y el yawRange es el vector [0 120 240], sintetizándose obviamente tres secciones.

El offset para la estructura 1 es 0 por defecto, ya que es la referencia.

A continuación el conjunto *vl_rodr + at_gsv2viewsynth_panorama1* es el encargado de calcular las matrices de rotación y traslación pertinentes (proceso matemático que no se tratará en este TFG) y finalmente se guardan en el disco en un formato .mat (es decir, formato archivo de MatLab donde la imagen es guardada como una matriz tridimensional para que ocupe mucho menos espacio que si se guarda en formato .png o .jpg).

Esencialmente qR es una matriz de rotación tridimensional calculada a partir de la función vl_rodr (se encuentra dentro de la toolbox VL_FEAT y aplica la fórmula de Rodrigues). Sin entrar en más detalle, la fórmula de Rodrigues es un algoritmo eficiente para la rotación de un vector en el espacio, dado un eje de referencia y un ángulo de rotación.

```
% Carlos Gómez Huélamo
% GIEAI 2013-2017 Final Degree Project
% Last modification date: 28/5/2017

% This function creates a set of synthetic views and save the views to disk

function createSynthViews_panorama1(estructura1,depthMap1)

load('Excel_information.mat');
location=strjoin(txt(1));
city=strjoin(txt(2));
country=strjoin(txt(3));

for ii=1:length(estructura1.Pitch_angle)
  for jj=1:length(estructura1.yawRange)

    angle = estructura1.yawRange(jj) + estructura1.offset;

    imgFile2 = sprintf('output/Sections/%s**%s**%s**Reference_%d', country, city, location, jj);
    qR = vl_rodr(estructura1.Pitch_angle(ii)/180*pi*[-1;0;0]) * vl_rodr(angle/180*pi*[0;-1;0]);
    [oimg,label,lambda] = at_gsv2viewsynth_panorama1(qR,estructura1,depthMap1);

    save([imgFile2 '.mat'],'oimg');
  end
end

end
```

Código de interés 4.1: Síntesis del panorama 1

2. Función createSynthViews_panorama2.m

La función para el panorama 2 es idéntica, con la diferencia de que **la primera vez que se llame a la función** ($\boldsymbol{flat == 0}$), las vistas sintéticas del panorama 2 se crearán con un offset en el ángulo (angle) introducido a la función *vl_rodr* que corresponde al **offset global** calculado a partir de la diferencia de columnas iniciales de la interfaz *Initial_information.*

La segunda vez que se llame a la función $(flat == 1)$, además del offset global antes calculado se sumará el offset individual de cada sección, pudiendo ser 0 si se decide mantener o bien el valor calculado si se cree oportuno corregir un alineamiento adicional.

Todo este proceso de alineamiento (considerado un proceso de realimentación puesto que se corrige el offset para después volver a alinear mediante transformaciones homogéneas cada par de secciones) será tratado con detalle en el capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea.*

```
% Carlos Gómez Huélamo
% GIEAI 2013-2017 Final Degree Project
% Last modification date: 28/5/2017

% This function creates a set of synthetic views and save the views to disk

function createSynthViews_panorama2(estructura2, estructura1,depthMap2, flat)

load('Excel_information.mat');
location=strjoin(txt(1));
city=strjoin(txt(2));
country=strjoin(txt(3));

% If flat == 0, only the global offset is considered

if (flat==0)
for ii=1:length(estructura2.Pitch_angle)
  for jj=1:length(estructura2.yawRange)

    if(estructura2.global_offset>0)

        angle = estructura2.yawRange(jj) - estructura2.global_offset;

    elseif(estructura2.global_offset<0)

        angle = estructura2.yawRange(jj) + estructura2.global_offset;

    elseif(estructura2.global_offset==0)

        angle = estructura2.yawRange(jj);


    end

    imgFile2 = sprintf('output/Sections/%s**%s**%s**Newest_without_offset_%d', country, city, location, jj);
    qR = vl_rodr(estructura2.Pitch_angle(ii)/180*pi*[-1;0;0]) * vl_rodr(angle/180*pi*[0;-1;0]);
    [oimg,label,lambda] = at_gsv2viewsynth_panorama2(qR,estructura2, estructura1, depthMap2, flat);

    % Save the synthetic RGB images to disk

    save([imgFile2 '.mat'],'oimg');
  end
end

% If flat == 1 (after the feedback), the sum of the global offset and each individual offset is considered
else
  for ii=1:length(estructura2.Pitch_angle)
  for jj=1:length(estructura2.yawRange)

    if((estructura2.offset(jj)+estructura2.global_offset)>0)

        angle = estructura2.yawRange(jj) - (estructura2.offset(jj)+estructura2.global_offset);

    elseif((estructura2.offset(jj)+estructura2.global_offset)<0)

        angle = estructura2.yawRange(jj) + (estructura2.offset(jj)+estructura2.global_offset);

    elseif((estructura2.offset(jj)+estructura2.global_offset)==0)

        angle = estructura2.yawRange(jj);

    end

    imgFile = sprintf('%s/%s_%03d.jpg', estructura2.outputDir_After_alignment, estructura2.name2, jj);
    qR = vl_rodr(estructura2.Pitch_angle(ii)/180*pi*[-1;0;0]) * vl_rodr(angle/180*pi*[0;-1;0]);
    [oimg,label,lambda] = at_gsv2viewsynth_panorama2(qR,estructura2, estructura1, depthMap2, flat);

    % Save the synthetic RGB images to disk
    imwrite(oimg, imgFile);
  end
end

end

end
```

Código de interés 4.2: Síntesis del panorama 2

Es por ello que entre las carpetas "buffers" (carpetas de tránsito de secciones intermedias) se encuentran:

Figura 4.16: Carpetas de almacenamiento intermedio o *buffers*

- Una carpeta para las **secciones y primer panorama** (referencia) → **Panorama1_Referencia**
- Una carpeta para las **secciones y segundo panorama antes de la realimentación**. Estas secciones únicamente presentan el offset global derivado de la diferencia de columnas iniciales → **Panorama2_Before alignment**
- Una carpeta para las **secciones y segundo panorama después de la realimentación**. Estas secciones, además de contar con el offset global, presentan un offset individualizado →**Panorama2_After alignment_with panorama 1**
- Una carpeta con las **secciones del segundo panorama** (tras la realimentación), tras aplicar transformación homogénea para alinear la segunda sección con la primera en términos de rotación, traslación, … → **Alignment_each_section**

3. Función at_gsv2viewsynth_panorama2(1).m

Esta función realmente es el corazón del proceso de síntesis de imágenes. Es llamada dentro de las anteriores funciones *createSynthViews_panorama2.m* y *createSynthViews_panorama1.m*. Se ha puesto un (1) en el título de la función puesto que para el primer panorama es idéntica pero la estructura es la primera (estructura1, referente al primer panorama) y no la segunda (estructura 2, referente al segundo panorama).

La implementación no es objeto de este TFG, pero básicamente es la traducción a código del concepto para la obtención de píxeles en la imagen virtual, tal y como se vio anteriormente:

- **Emisión de un rayo desde el centro de la cámara virtual.**
- **Intersección con la estructura 3D planar proporcionada por el mapa de profundidad.**
- **Proyección de esta intersección hacia el panorama.**
- **Interpolación bilineal para obtener el pixel de salida a partir de los cuatro píxeles adyacentes.**

Nótese que el resultado final, es decir, la imagen, se obtiene en la línea de código 109:

```
109    oimg = mh_iminterp(img,pts,bilinear);
```

Siendo *oimg* la imagen sintética de salida de dimensiones 640 x 480 píxeles, *mh_iminterp* una función de MatLab que permite hacer la interpolación (bilineal en este caso, tal y como se puede observar en el

tercer argumento de la función), *img* la imagen panorámica que contiene toda la materia prima y *pts* los puntos que se han ido determinado a lo largo de la función (a partir de las proyecciones, etc…) que corresponde a la zona de interés.

```
% at_gsv2viewsynth_panorama2.m
% This functions performs the image synthesis with bilinear interpolation
% Date: 04-07-2017
%*************************************************************************
function [oimg,olabel,lambda] = at_gsv2viewsynth_panorama2(qR,estructura2, estructura1,depthMap2, flat)

% Get the params from gsv struct
oimw = estructura2.oimw;
oimh = estructura2.oimh;
FoV = estructura2.FoV;

% Compute rotation matrix

if (flat==0)
R = qR*vl_rodr((estructura2.Heading_angle/180)*pi*[0; 1; 0]); % Initial synthesis
else
R = qR*vl_rodr((estructura1.Heading_angle/180)*pi*[0; 1; 0]); % After feedback process
end

C = estructura2.C;
T = -R*[C(2); 0; -C(1)];

img = estructura2.panorama;
scale = size(img,2)/512;

% Convert the planes information from GSV to camera coordinates
[pN, pD] = convertPlanes(depthMap2);

% Convert the labels format
plabel = convertLabels(depthMap2);

% Convert image points to normalized unit vectors
K = [oimw/2*FoV 0 oimw/2; 0 oimw/2*FoV oimh/2; 0 0 1];
[gx, gy] = meshgrid(1:oimw,1:oimh);
gx = gx'; gy = gy';
X = [gx(:)'; gy(:)'; ones(1,oimw*oimh)];
Y = at_l2normalize_col(K\X);

psz1 = size(plabel,1);
psz2 = size(plabel,2);

% In case we have enough motion
if sum(abs(T)) > eps

    % Compute the depth for all the planes
    lambda = bsxfun(@rdivide, pN'*(R'*T) - pD',pN'*(R'*Y));

    % Retrieve the closest plane from the ones having a consistent label.
    x = -ones(1,oimw*oimh);
    y = x;
    d = at_synthreproj2(Y,lambda,R',T,plabel);
    d = [NaN(1,size(d,2)); d];
    [s,olabel] = min(d,[],1);

    % Set the NaN to the maxDepth value
    inl = isnan(s);
    s(inl) = estructura1.maxDepth;

    inl = ~isnan(s);
    Z = bsxfun(@times,s(1,inl),Y(:,inl));
    Z = R'*bsxfun(@minus,Z,T);
    Z = at_l2normalize_col(Z);

    th = atan2(Z(1,:),Z(3,:));
```

```
    phi = asin(Z(2,:));
    [u,v] = at_sph2rct(th,phi,psz1,psz2);
    u = scale*u;
    v = scale*v;

    u(floor(u)==0) = 1;
    v(floor(v)==0) = 1;
    x(1,inl) = u;
    y(1,inl) = v;

else

    % Compute depth information
    lambda = bsxfun(@rdivide,-pD',pN'*(R'*Y));

    % Rotate the vectors and transform them to spherical coordinates
    Z = R'*Y;
    Z = at_l2normalize_col(Z);
    th = atan2(Z(1,:),Z(3,:));
    phi = asin(Z(2,:));

    % Apply equirectangular projection
    [u,v] = at_sph2rct(th,phi,psz1,psz2);
    x = scale*u;
    y = scale*v;

    % Take care with the horizontal and vertical coordinates
    x(floor(x)==0) = 1;
    y(floor(y)==0) = 1;
    uu = round(u);
    uu(uu==0) = 1;
    uu(uu>512) = 512;
    vv = round(v);
    vv(vv==0) = 1;
    vv(vv>256) = 256;

    sidx = sub2ind(size(plabel),uu,vv);
    olabel = plabel(sidx)+1;
end

% Perfrom image synthesis with bilinear interpolation
bilinear = 1;
pts = []; pts.x = reshape(x,oimw,oimh)';
pts.y = reshape(y,oimw,oimh)';
oimg = mh_iminterp(img,pts,bilinear);

end

%*************************************************************************
% This function convers the format of the planes and labels and adjusts the
% coordinates between the new camera and GSV planes
function [pN, pD] = convertPlanes(depthMap1)

% Important we need to remove the first plane since it is not valid!!
CT = [-1  0  0 0; 0  0  1 0; 0 -1  0 0; 0 0 0 1];
nr_planes = length(depthMap1.planes);
pN = zeros(3, nr_planes-1);
pD = zeros(1, nr_planes-1);
end

end

%*************************************************************************
% This function convers the labels into the required format by 24/7
function [plabel] = convertLabels(depthMap1)

nr_cols = depthMap1.labelsdim(1);
nr_rows = depthMap1.labelsdim(2);
plabel = zeros(nr_cols,nr_rows);

for h=0:nr_rows-1
    for w=0:nr_cols-1
        plabel(w+1,h+1) = depthMap1.labels(h*nr_cols+w+1);
    end
end

end
```

Código de interés 4.3: Obtención de la imagen sintética

4.4. Ventajas y desventajas de la síntesis de imágenes:

Por último, para concluir el capítulo es importante resaltar las ventajas y desventajas de la síntesis de imágenes:

Ventajas:

- El método propuesto habilita el **reconocimiento verdadero de lugares a escala mundial.**
- A pesar de que las imágenes renderizadas presentan ruido a menudo (objetos cortado, cierta distorsión, alguna franja negra, etc…), los **resultados son los suficientemente buenos para mejorar de una forma muy notable el rendimiento de los extractores de características**, vitales para el futuro alineamiento de las imágenes y poder proceder al etiquetado de cambios.

Desventajas/limitaciones:

En general, las típicas causas de fallo para la síntesis de imágenes son:

- Imágenes de noche con muy baja iluminación (sin embargo, dado que ninguna localización de la base de datos de Excel presenta horario nocturno, esta limitación no importa para este TFG).
- **Lugares con abundante vegetación.**
- **Lugares complejos que dificulten la representación de la estructura 3D planar en el mapa de profundidad** (transeúntes, vehículos, etc…)

Capítulo 5

Extracción y descripción de puntos característicos en vistas sintéticas:

El capítulo 5 *Extracción y descripción de puntos característicos en vistas sintéticas* y capítulo 6 *Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea* son la parte de este TFG en la que más ha profundizado (tanto en creación de código, investigación del algoritmo y redacción) el autor de este TFG, por lo que se explicará con un **mayor grado de detalle tanto el código como el proceso matemático de estos mismos.**

Aunque las **características invariantes locales** han sido satisfactoriamente usadas durante casi dos décadas por hacer coincidir el punto de vista y la escala, a menudo estas características **no son fáciles de localizar** (las mismas para ambas secciones) **ante fuertes efectos de perspectiva o cambios mayores en la iluminación de la escena.** Es por ello que hoy en día (aunque depende de la aplicación), es preferible el uso de **descriptores densamente muestreados** (por ejemplo SIFT o SURF que son capaces de extraer un mayor número de características antes iguales cambios en la perspectiva).

La síntesis de imágenes estudiada en el capítulo anterior se efectúa con el mismo ángulo *pitch* para generar las secciones en ambos panoramas. Aunque la base de datos de *URLs* se ha realizado a conciencia de tal forma que la diferencia de pose en traslación sea lo más pequeña posible (contando con que el coche de GSV nunca pasa en dos meses u años distintos por el mismo sitio), la necesidad de alineamiento es imperante.

Para alinear algo, y en concreto dos imágenes con aparentemente la misma información, se deben **extraer características que correspondan al mismo** (como puede ser la esquina superior izquierda de un edificio, una señal de tráfico, etc).

Un flujograma secuencial de cómo funciona la parte de sintetización y alineamiento de la aplicación (que será tratada en el Anexo II *Manual de usuario*) se muestra a continuación:

Figura 5.1: Flujograma de síntesis, extracción y alineamiento

Dichas características serán de vital importancia para el alineamiento horizontal/vertical y la transformación homogénea en caso de que la síntesis de imágenes no diese lugar a la misma información. Este alineamiento será tratado posteriormente.

En este capítulo se tratarán los puntos 2 y 3, y en el siguiente los puntos 4, 5 y 6, puesto que el punto 1 (síntesis de las imágenes de ambos panoramas) fue el tema tratado en el capítulo anterior.

En primer lugar se debe tratar tres temas de vital importancia, definición, detección y descripción de un punto característico.

5.1. Definición de punto característico:

Uno de los aspectos más estudiados en aplicaciones relacionadas con la visión artificial es la **correspondencia entre dos imágenes.** Este tipo de tareas es utilizado en un gran número de aplicaciones, como por ejemplo el reconocimiento de objetos (señales de tráficos, matrículas o rostros, entre otros), patrones o seguimiento de objetos, entre otros.

Para establecer esa correspondencia, también llamada *matching*, se debe llevar a cabo la extracción y posterior descripción de puntos de interés.

Un **punto de interés** (puntos característico, *keypoint* o *feature point*) es aquel **punto de una imagen que tiene una posición bien definida y que puede ser detectado de forma robusta**. Esto significa que un punto de interés puede ser una esquina (en el que se especializa, por ejemplo, el detector de Harris), pero también puede ser un punto aislado de intensidad local máxima o mínima, final de una línea o aquel punto de la curva donde es localmente máximo.

5.2. Detección de puntos característicos:

La detección corresponde a la **primera etapa del procesamiento de una imagen en la tarea de correspondencia**, que implica la **individualización de puntos cuyas características ayudan en su conjunto a definir el objeto contenido en una imagen.** Se pretende que los detectores sean eficientes respecto de los recursos computacionales que utilizan. Esto influye directamente en la cantidad de cuadros por segundo (pixeles) se pueden procesar y en las potentes plataformas hardware sobre las cuales se deben implementar los procesos en caso de requerir tiempo real.

Esta fase es absolutamente necesaria en este TFG, probablemente la más importante (después de tener ya las imágenes sintéticas de iguales dimensiones y aparentemente igual información), puesto que **la evidencia en la que se basa el alineamiento entre imágenes**, y de hecho concluir que la correspondencia previa entre cada par de imágenes sintéticas es correcta, será **mayor cuantos más *keypoints* sean capaces de ser extraídos y relacionados.**

La extracción de *keypoints* (o *feature points*) **permite reducir la información con la que se trabaja**, es decir, en vez de comparar cada pixel correspondiente de ambas imágenes, sólo se comparan los puntos más característicos (esquinas pronunciadas, bordes, puntos máximos o mínimos locales de luminosidad, etc…) reduciendo el tiempo de cómputo. Esto es conocido como **procesamiento *sparce*.**

En general no existe una regla para saber qué tipos de puntos de interés hay que extraer, pero dependiendo del tipo de imagen conviene localizar unos u otros. **Los puntos de interés más comunes son las regiones de alta/baja luminosidad, bordes, esquinas** (realmente la intersección de dos bordes). Por ejemplo, en una imagen que presente mucha geometría (un edificio con un gran número de bordes y esquinas) convendrá usar el detector de esquinas de Harris, costoso computacionalmente, pero especializado en esta tarea.

5.2.1. Características de un buen detector de keypoints:

Un buen detector de *keypoints* debe tener la mayor parte o todas de las siguientes características:

1. Precisión: El detector debe localizar con exactitud el *keypoint*, tanto en escala como en posición.

2. Repetibilidad: Cuantos más puntos de interés sean capaces de ser extraídos en ambas imágenes, mejor. De hecho, la **matriz calidad** (localizada en la función *Show_checked_alignment.m*) es una extensión de este concepto. Es una matriz 3x3, siendo cada elemento una combinación entre distintas técnicas de extracción (HARRIS, SURF, SIFT) y distintos métodos de estimación de alineamiento tras el filtro MSAC (similarity, affine y projective), en cada elemento de la matriz se guardan el número de inliers:

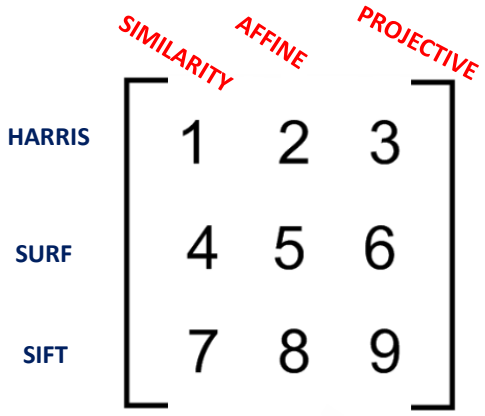


Figura 5.2: Matriz de calidad del proceso

3. Eficiencia computacional: No debe consumir muchos recursos de la computadora.

4. Robustez: Si un par de *keypoints* son detectados entre dos imágenes, el proceso debe ser robusto y ante una invarianza de cambios geométricos y/o iluminación, debería detectar también dicho par. Evidentemente esta robustez será menor/mayor en función del detector y también del tiempo computacional límite dotado.

5. Distintividad: El detector debe ser capaz de detectar entre distintos tipos de puntos característicos, por ejemplo esquinas de puntos de alta luminosidad, etc…

No existe un único método para la detección de esquinas, bordes, etc Los principales están basados en la curvatura, el gradiente, los *keypoints* con modelo predefinidio, *Scale-Space*, imágenes binarias, etc

Este campo de la visión artificial es muy amplio (muchos descriptores) y complejo (procesos físico matemáticos no triviales para la elaboración y comprensión de estos métodos).

La mayoría de estos métodos son válidos para la detección de bordes, siendo probablemente un borde (también llamado contorno) la principal característica a extraer de una imagen (puesto que realmente una esquina es la intersección de dos bordes). **Estos métodos no son sensibles a cambios de intensidad de la imagen**, tal y como se demuestra en el paper 24/7 [8], donde comparan imágenes sintéticas (y se extraen características de todas) en distintos momentos del día. En este TFG en concreto estas técnicas no procederán porque todas las localizaciones de la base de datos se encuentran tomadas en horario diurno (con luz solar).

Para entender los descriptores SURF, Harris y SIFT primero es necesario entender cómo funciona uno de los más simples algoritmos (antiguo pero eficiente) que existe en el ámbito de detección: El algoritmo de Canny, basado en máscaras de convolución y la primera derivada.

5.2.2. Algoritmo de Canny [10]:

Los puntos que pertenecen a un **borde** son **zonas de píxeles donde existe un cambio abrupto en la escala de gris**. Esto es mucho más fácil de demostrar con un ejemplo simple:

Figura 5.3: Detección de bordes mala (arriba) ; Detección de bordes mala (abajo)

Intuitivamente el cerebro es capaz de detectar fácilmente los bordes de la rosa 5.3 (su contorno) puesto que en esa zona de píxeles la escalas de grises cambia abruptamente de rojo a negro. Por otra parte, cuando menos nítida sea una imagen, que es lo que le pasa por ejemplo a una persona con miopía (no enfoca a media/larga distancia), la visión estereoscópica se difumina y la escala de color no cambia de forma tan abrupta, sino que está todo mucho más borroso. Por eso esta persona no detecta los contornos de objetos a larga distancia 5.3.

A pesar de ser un algoritmo ciertamente antiguo en relación con los inmensos avances tecnológicos actuales (el algoritmo de Canny data de 1986), se convirtió rápidamente en un algoritmo de detección de bordes estandar. Está enfocado en los siguientes puntos:

1. Buena detección: El algoritmo debe marcar el mayor número de bordes posibles en la imagen.

2. Buena localización: Los bordes marcados deben estar lo más cerca posible del borde de la imagen real.

3. Respuesta mínima: El borde de una imagen sólo debe estar marcado una vez. Siempre que sea posible, el ruido de la imagen no debe crear falsos bordes.

Un ejemplo del algoritmo de Canny se muestr a continuación:

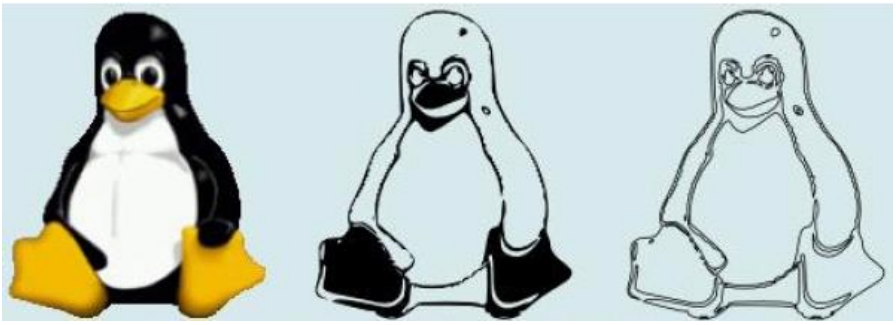


Figura 5.4: Ejemplo algoritmo de Canny

A su vez, este algoritmo se puede resumir en las siguientes fases:

Figura 5.5: Flujograma del algoritmo de Canny

1. Suavizar: Es imposible que todas las imágenes tomadas desde una cámara no contengan una pequeña muestra de ruido. Para evitar que el ruido se confunda con los bordes, este ruido se reduce (difuminándolo y por tanto **homogeneizando la información buena de la imagen con el ruído**) aislando la imagen mediante la aplicación de un **filtro de Gauss.**

Un filtro de Gauss es un objeto matemático que se aplica a la matriz que corresponde a la información de la imagen, consiguiendo **suavizarla y por tanto eliminar parte del ruido**. Realmente, el ruido no desaparece, sino que al difuminarse parece que se nota menos y por tanto se dice que se ha eliminado una parte de él.

Su fórmula es:

$$G(x,y) = \frac{1}{2\cdot\pi\cdot\sigma^2}\cdot e^{\left[-\frac{x^2+y^2}{2\cdot\sigma^2}\right]} \tag{5.1}$$

Siendo σ la desviación típica elegida para el proceso.

Sin embargo, hay que tener en cuenta que el suavizado no debe ser excesivo para no perder detalles en la imagen y dificultar así la obtención del gradiente.

Un ejemplo se imagen suavizada es el siguiente:

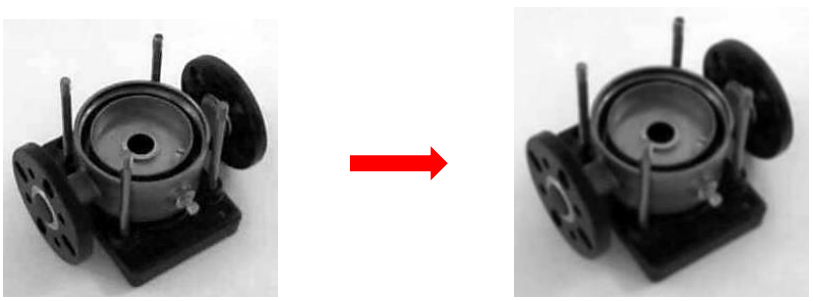


Figura 5.6: Ejemplo suavizado

2. Cálculo del gradiente (magnitud y orientación) **del vector gradiente** (primera derivada) **de cada pixel**: **El algoritmo de Canny básicamente encuentra bordes donde la escala de grises de intensidad de la imagen cambia más**. Estas áreas se determinan a partir de los gradientes de la imagen, los cuales se determinar a su vez a partir del Sobel-operador. El gradiente de una imagen f(x,y) en un punto (x,y) se define como el vector bidimensional dado por la ecuación:

$$G[f(x,y)] = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x} \cdot f(x,y) \\ \frac{\partial}{\partial y} \cdot f(x,y) \end{bmatrix} \quad (5.2)$$

Por tanto se está derivando la intesidad de la escala de gris, o dicho de otra forma, como varía el color en la dirección X para un punto y como varía el color en la dirección Y para ese mismo punto o pixel.

El primer paso es aproximar el gradiente en la dirección X-Y y después aplicar dicho operador.

Por tanto la **magnitud del gradiente** se puede establecer como la hipotenusa que forman los gradientes en ambas direcciones:

$$|G| = \sqrt{G_x^2 + G_y^2} \quad (5.3)$$

Siendo $|G|$ la magnitud del gradiente, G_x el gradiente en X y G_y el gradiente en Y.

Por otra parte, el **cálculo del ángulo:**

$$\theta = \arctan\left(\frac{|G_y|}{|G_x|}\right) \quad (5.4)$$

Siendo θ el ángulo del gradiente. Entre esta ecuación (5.4) y la anterior (5.3), módulo y ángulo quedan definidos.

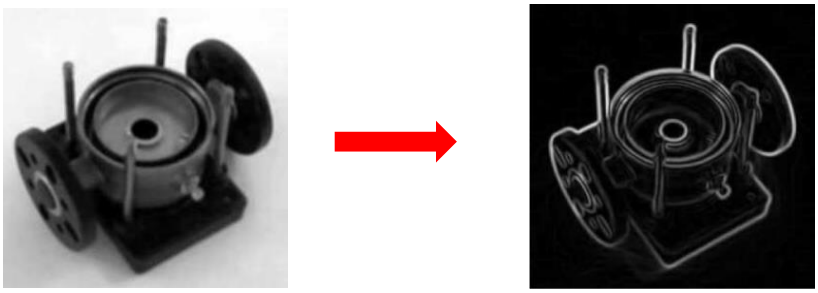


Figura 5.7: Ejemplo tras aplicar los gradientes

3. No supresión máxima: El propósito de este paso es **adelgazar los bordes de la imagen hasta que sólo tengan un pixel de anchura.** Se consideran cuatro direcciones: 0º, 45 º, 90 º y 135 º respecto al eje horizontal. Para cada píxel se encuentra la dirección que mejor se aproxime a la dirección del ángulo del gradiente.

En este punto se observa si el valor de la magnitud de gradiente es más pequeño que al menos uno de sus dos vecinos en la dirección del ángulo obtenida en el apartado anterior. Si es más pequeña, el píxel toma el valor 0 (color negro), y sino, se asigna la magnitud del gradiente. El resultado es la imagen del gradiente anterior pero con los bordes adelgazados.

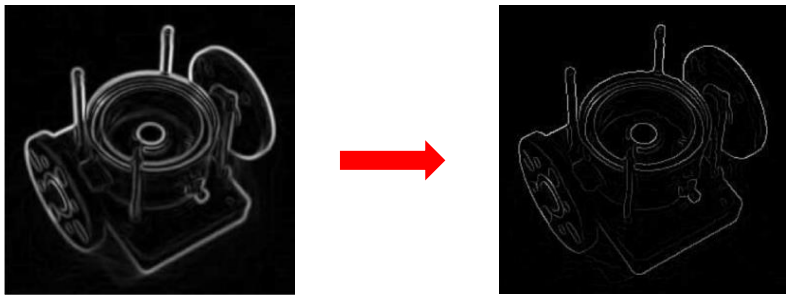


Figura 5.8: Ejemplo tras adelgazar los gradientes obtenidos

4. Umbralización doble: Aunque la mayoría de bordes resultantes de la supresión no máxima anterior son efectivamente los bordes de la imagen, **algunos pueden ser causados por variaciones de ruido o color, por ejemplo en superficies rugosas.**

La forma más sencilla será el uso de un umbral, de modo que sólo los bordes más fuertes de un cierto valor de intensidad son preservados. La umbralización doble, usada en Canny, implica que se fijan dos umbrales de intensidad:

- **Aquellos píxeles por encima del umbral superior serán fuertes.**
- **Aquellos píxeles por debajo del umbral inferior son suprimidos.**
- **Aquellos píxeles entre ambos umbrales son marcados como débiles.**

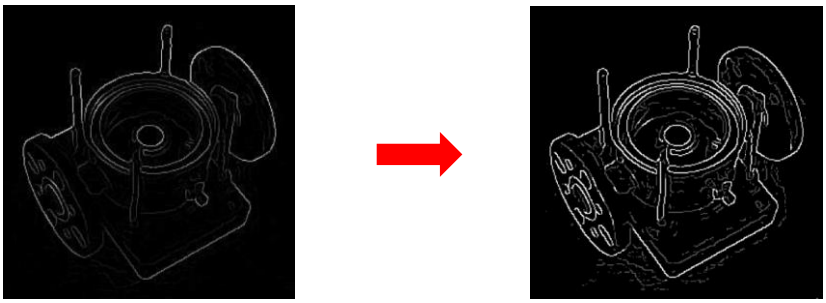


Figura 5.9: Ejemplo tras adelgazar la umbralización doble

5. Seguimiento por histéresis: Este es el paso final. **Los bordes fuertes del paso anterior se interpretan como bordes finales, los bordes débiles también y si y solo sí están conectados a los fuertes.** Si no están conectados a un borde fuerte, se pueden considerar debidos a variaciones de ruido y/o color, por lo que son suprimidos.

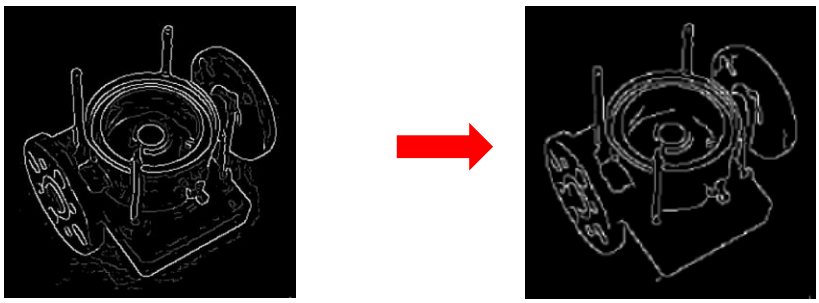


Figura 5.10: Ejemplo tras el seguimiento por histéresis

El algoritmo de Canny fue el primer método de detección usado para este TFG, principalmente al inicio para testear los distintos métodos. La gran ventaja de utilizar este método es que si las imágenes sintéticas están muy bien alineadas, para hacer el groundtruth (segmentación final de la imagen) esta

detección de bordes resultaba de una gran ayuda y de hecho se ha utilizado como referencia para el etiquetado en ciertas imágenes (porque básicamente ya está proporcionando la información de los contornos).

Sin embargo, **al encontrar bordes cercanos al umbral de detección, un pequeño cambio en la intesidad de este podría causar un gran cambio en su topología**, provocando errores de correspondencias (tal y como se ha podido comprobar en la mayor parte de las imágenes, dada la complejidad visual de muchas geolocalizaciones, en especial aquellas que tienen transeuntes, vehículos, vegetación, etc…).

Es por ello que se investigó de una forma más profunda en cuáles podían ser los mejores extractores de acuerdo a las características más comunes de las localizaciones, y se llegó a la conclusión que debían usarse detectores basados en puntos de interés en vez de detección de bordes (como el algoritmo de Canny expuesto antes).

Dentro de estas técnicas de para tratar *keypoints*, en este TFG se han usado tres en concreto: **El detector de esquinas de Harris (sólo extractor o detector), la técnica SURF y la técnica SIFT (estas dos últimas técnicas, aunque pueden actuar como detectores y descriptores, se caracterizan principalmente por su proceso de descripción)**, estando las dos primeras técnicas implementadas en la biblioteca Computer Vision de la versión R2017a de MatLab, mientras que la técnica SIFT proviene de una librería externa (adaptada a MatLab) denominada VLFEAT.

5.2.3. Detector de esquinas de Harris [11]:

La detección de esquinas es una aproximación usado en los sistemas de visión artificial para extraer la intersección de bordes o esquinas. La detección de esquinas frecuentemente se usa en la detección del movimiento, rastreo en vídeo, modelado 3D o como para este TFG, procesamiento de la imagen y reconocimiento de objetos.

Un **detector de esquinas** presenta la ventaja de **usar las esquinas en vez de bordes como principal meta de detección ya que las esquinas son poco sensibles a cambios en su rotación y escala**. El principio de búsqueda de un detector de esquinas es el **análisis de una región a partir de una posición inicial para poder determinar en qué tipo de región se encuentra este mismo**. Las tres regiones principales que se pueden encontrar en una imagen son:

- **Borde:** También denominado contorno. No hay cambio en la dirección del borde.
- **Región plana:** No hay contorno y por tanto no se detecta ningún cambio de dirección.
- **Esquina:** Intersección de dos o más bordes. Por tanto existe un cambio de intensidad notable en una o más direcciones.

Precisamente este último es el principio de búsqueda de estos detectores: Encontrar puntos caracterizados por ser regiones con cambios de intensidad notables en varias direcciones, es decir, encontrar esquinas.

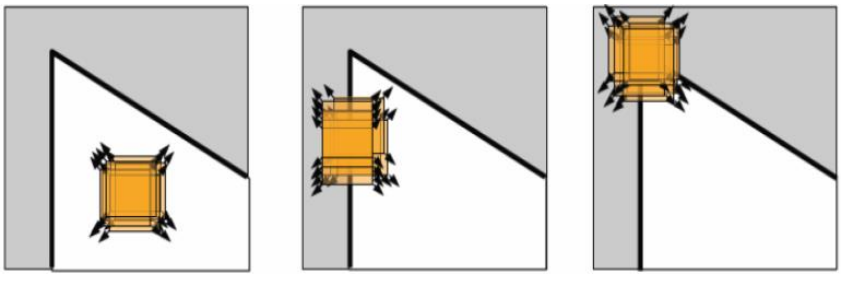


Figura 5.11: Reconocimiento de la zona

Los detectores de esquinas normalmente no son robustos y requieren de una supervisión especializada para impedir el efecto de errores individuales en la tarea de reconocimiento. Una forma de determinar la calidad de un detector de esquinas es su **destreza para obtener la misma esquina en imágenes similares pero que cuenten con diferentes poses** (es decir, distinta rotación y/o traslación), diferentes condiciones de iluminación,….

Una forma simple de detección de esquinas podría ser usando la **correlación**. La correlación indica la relación lineal y proporcionalidad entre dos variables estadísticas (en este caso entre los puntos de interés a comparar de imágenes distintas). Se considera que dos variables cuantativas están correlacionadas cuando los valores de una de ellas varían sistemáticamente con respecto a los valores homónimos de la otra variable: En este caso, **dos puntos de interés (dos esquinas) estarían correlacionados si al disminuir ciertos valores/característicos de uno de ellos también lo hace el otro y viceversa**. Sin embargo, el uso de la correlación es un proceso costoso computacionalmente. Por ello, un acercamiento alternativo es el modelo propuesto por Moravec.

5.2.3.1. Algoritmo de Moravec [11]:

Es uno de los primeros detectores de esquinas propuesto. El principio de funcionamiento es el siguiente:

- **Análisis de cada pixel en la imagen para detectar si hay una esquina**, considerando la similitud con un parche (o ventana como se citó anteriormente) centrado en el píxel cercano, solapando ambos.
- Se toma la **suma de diferencias cuadradas (SDC)** que se definirá a continuación en Harris entre ambos parches. Un bajo número indica que la intensidad es muy parecida y por tanto hay similitud.
    - Si el pixel está en una región de **intensidad uniforme**, los parches cercanos parecerán similares.
    - Si el píxel está en un **borde**, los parches cercanos en una dirección perpendicular al borde serán muy diferentes, pero los cercanos en dirección paralela (es decir, en el propio borde), sólo producirán un pequeño cambio.
    - Si el píxel está en una **región con variación en todas las direcciones (esquina)**, ningún parche cercano parecerá similar.
- Finalmente, la **fuerza de la esquina se define como la suma de cuadrados SDC más pequeña** entre un parche y sus vecinos en la horizontal, vertical y ambas diagonales. Cuanto más pequeño sea, más característico será este punto de interés en forma de esquina.

  Sin embargo, este método basado únicamente en la SDC entre parches cercanos era insuficiente para imágenes complejas con un gran número de contornos, por lo que se decidió crear el detector de Harris.

5.2.3.2. Algoritmo de Harris-Stephens:

En general denominado simplemente detector de Harris, este detector presenta una mejora respecto a Moravec puesto que el **tratamiento es diferencial**, es decir, se considera el diferencial del valor de la esquina con respecto a esa dirección, en lugar de usar estas ventanas o parches para el análisis. Dicho de otra forma, este algoritmo **mejora mucho la precisión del algoritmo anterior**. Cronológicamente en este TFG fue el segundo extractor (y el primero basado en puntos característicos en vez de bordes como era el algoritmo de Canny) en ser usado.

Para el uso del detector de Harris (y también de la técnica SURF) en MatLab, la imagen debe ser **traducida del modelo RGB a la escala de grises.**

Al igual que antes, **para determinar matemáticamente si se ha detectado una esquina, el parche centrado en el píxel de origen se desplaza en un salto (n, m) y se compara cada píxel antes y después**

del desplazamiento, considerando el sumatorio de todas las SDC (en inglés SSD, Sum of Squared Differences), denotada por $SS(n,m)$:

$$SS(n,m) = \sum_x \sum_y w(x,y) \cdot (I(x+n, y+m) - I(x,y))^2 \quad (5.5)$$

Siendo $I(x,y)$ la intensidad de la escala de grises de la imagen original, $I(x+n, y+m)$ la intensidad de la imagen original desplazada por el vector (n,m) y finalmente $w(x,y)$ la ventana sobre la que se realiza el análisis.

Una esquina está caracterizada precisamente por lo contrario a una región plana: Variación grande del sumatorio de SDC en todas las direcciones (x,y). Es decir, **cuanto más grande sea SS(n, m), mayor será la posibilidad de que ese punto de interés sea una esquina.**

Para vectores de desplazamiento (n,m) pequeños, se puede aproximar la intensidad en el punto siguiente (añadido el desplazamiento), mediante una serie de Taylor (objeto matemático cuyo objetivo es la aproximación de una función en otra más simple).

$$I(x+n, y+m) \approx I(x,y) + I_n(x,y) \cdot n + I_m \cdot (x,y) \cdot m \quad (5.6)$$

Siendo respectivamente I_n e I_m las derivadas parciales de I:

Esto conduce a la siguiente aproximación:

$$SS(n,m) \approx \sum_x \sum_y w(x,y) \cdot (I_n(x,y) \cdot n + I_m \cdot (x,y) \cdot m)^2 \quad (5.7)$$

O de forma equivalente:

$$SS(n,m) \approx \sum_x \sum_y w(x,y) \cdot (n^2 \frac{\partial^2 I}{\partial n^2} + v^2 \frac{\partial^2 I}{\partial m^2} + 2nm \frac{\partial^2 I}{\partial n \cdot \partial m}) \quad (5.8)$$

Y en forma de matriz:

$$SS(n,m) \approx (n\ m) \cdot K \cdot \binom{n}{m} \quad (5.9)$$

Siendo K el siguiente sumatorio:

$$K = \sum_x \sum_y w(x,y) \cdot \begin{pmatrix} I_n^2 & I_n \cdot I_m \\ I_n \cdot I_m & I_m^2 \end{pmatrix} \quad (5.10)$$

K es la matriz característica de Harris (también denominada de autocorrelación), pudiéndose ver el promedio de la variación en todas las direcciones (x,y). Evidentemente, si una ventana es redonda, la respuesta es isotrópica, es decir, igual en todas las direcciones.

A contiuación se definen los autovalores (escalares que acompañan a los autovectores capaces de cambiar la dirección en conjunto de una imagen) λ_1 y λ_2, obteniéndose en función de ellos, tres tipos de regiones posibles:

1. Si los valores de ambos autovalores son pequeños (≈ 0), la función de autocorrelación (K) es plana, y por tanto esa región de la imagen presenta una intensidad constante → **Región plana.**

2. Si λ_1 ≈ 0 y λ_2 tiene un valor positivo grande (o viceversa), la función de Harris presenta rizado → **Borde.**

3. Si ambos autovalores son elevados positivamente, la función presenta cambios bruscos y existe una **esquina.**

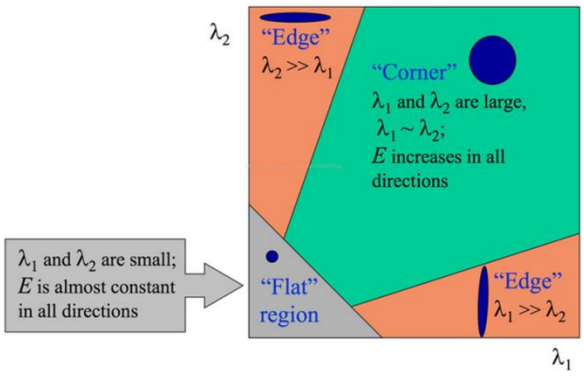


Figura 5.12: Ejemplo gráfico de tipo de región según los autovalores

En conclusión, cuando en una región ambos autovalores presentan valores positivos grandes, se deduce que localmente existen dos direcciones importantes en los gradientes y que por tanto se trata de una esquina.

Sin embargo, a pesar de ser uno de los detectores más usados, **el algoritmo de Harris presenta una gran desventaja:** Computacionalmente es muy costoso ya que para hallar una esquina de forma precisa, para todos los píxeles se debe determinar la primera (y después segunda) derivada de cada uno, además de para cada caso el cálculo del determinante y su matriz de Harris K. Además, **otro defecto es que el algoritmo de Harris únicamente puede actuar como detector, y no como descriptor**, por lo que el proceso es menos compacto, comprobándose en este TFG que dentro de las 909 secciones analizadas con la combinación Harris+descriptor correspondiente, el resultado de *inliers* (buenas correspondencias) es mucho menor que con las otras dos técnicas (SURF y SIFT).

Por ello, se planteó la posibilidad de introducir los códigos de estas dos técnicas, **SIFT y SURF** (de hecho SURF surge como una derivación de la técnica SIFT) que ofrece unos **extractores de características mucho más rápidos y eficientes**. Cronológicamente, dado que la técnica SURF sí que está implementada en la toolbox Computer Vision de MatLab R2017a, fue la primera en ser incorporada al código para poco tiempo después implementar la librería externa VLFEAT [13] adaptada a MatLab que contenía la información de SIFT. Sin embargo, puede considerarse que se implementaron de forma simultánea para la comparativa de resultados.

Una de las **grandes ventajas de estas dos técnicas,** es que, a diferencia del algoritmo de Harris que únicamente podía actuar como detector, **estas dos técnicas (SURF y SIFT) son capaces de detectar y describir según convenga.**

5.3. Descripción de puntos característicos:

A continuación de la detección viene la descripción. **Los parámetros calculados hasta ahora** (posición, escala de los puntos característicos) **forman un sistema de coordenadas bidimensional que localiza una determinada región de la imagen.** El siguiente paso es, utilizando esa información, obtener un **descriptor para cada zona de interés de forma que al aplicar variaciones en la iluminación, modificación del punto de vista (siempre hablando en términos moderados) los puntos sean nuevamente caracterizados.** Es decir, el subproceso o etapa de descripción fortalece el proceso general de reconocimiento de zonas entre imágenes.

- Un **detector ideal de puntos de interés localiza dichos puntos de forma repetida a pesar del cambio de punto de vista** (imprescindible en la tarea de este TFG dado el campo del punto de vista por la localización del coche), es decir, debe ser fiable ante transformaciones de la imagen.

- Un **descriptor ideal proporciona de cada *keypoint* información relevante y distintiva de la región circundante**, de forma que la misma estructura pueda ser reconocida si es localizada en otra imagen. En otras palabras, un descriptor es un conjunto ordenado de datos (números) que contienen información acerca de la imagen.

Por ejemplo, en el capítulo 3 *Elaboración del panorama y mapa de* profundidad se explicaba el proceso de obtención de las imágenes panorámicas y los mapas de profundidad, con su modelo de matriz tridimensional (modelo RGB) y la información del mapa de profundidad, se puede caracterizar perfectamente a la imagen, **por tanto se podría considerar a una imagen digital como un descriptor.** Sin embargo, el **número de píxeles es tan alto en una imagen** (así como la memoria que ocupan, dado que en modelo RGB cada pixel son 24 bits) **que el uso de descriptores para tan sólo obtener información de los puntos más relevantes** (los anteriores citados *keypoints*), **mejora enormemente el proceso.**

Implica un verdadero desafío describir de manera eficiente puntos de interés en forma estable y compacta, con representaciones robustas invariantes a la escala (si la imagen es más grande o más pequeña, o como podría ocurrir en este TFG con la cercanía/lejanía de la cámara en una pasada respecto a otra pasada del coche), rotación, diversas transformaciones, ruido (por ejemplo si el cielo está iluminado en una pasada y nublado en otra, el brillo de un edificio variará evidentemente, pero el punto característico deberá mantenerse invariante, independientemente de esta variación), etc.

Tras tratar el proceso de descripción, a continuación se deben detallar las dos técnicas con mayor rendimiento de este TFG, SURF y SIFT, que son capaces de actuar como detector y descriptor en su propio flujograma.

5.4. Técnica SIFT [12]:

El algoritmo SIFT, implementado por **David Lowe**, está basado en el filtro Gabor de Christoph von der Malsburg, cuyo principio de funcionamiento está basado en las características locales. David G. Lowe mezcló estás características locales basadas principalmente en la escala con su propia escala SIFT, resultando en un **potentísimo método capaz de detectar objetos incluso con oclusión parcial, cierta distorsión y cambios de iluminación.**

El término **SIFT** es un acrónimo inglés que significa ***Scale-Invariant Feature Transform***. En otras palabras, es una transformación de la información que proporciona una imagen en coordenadas invariantes a la escala en el ámbito local.

A partir de estas características locales extraídas, se busca conseguir invarianza ante una modificación en la escala, la orientación, parcialmente cambios de iluminación, … Un ejemplo sencillo se muestra a continuación (estando los outliers, es decir, emparejamiento incorrectos, ya eliminados puesto que ese resultado *red-cyan* proviene de la función *estimateMatchedFeatures* de la toolbox Computer Vision de MatLab que aplica el filtro MSAC, una variante del RANSAC) donde se puede observar que a pesar del cambio en la escala y la rotación del fotógrado, son identificados una gran cantidad de pares característicos.

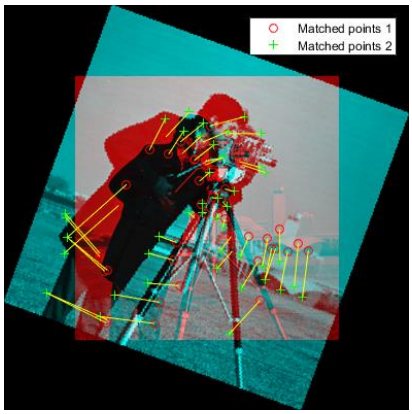


Figura 5.13: Correspondencia SIFT

También se puede utilizar la técnica SIFT para **buscar correspondencias entre diferentes puntos de vista de una misma escena**, esencial para este TFG cuando ciertas vistas sintéticas no están perfectamente alineadas. Posteriormente, estas características locales se almacenan en descriptores que están alojados igualmente dentro del algoritmo SIFT, esto es, SIFT puede actuar tanto como detector o como descriptor en función de las circunstancias.

5.4.1. Flujograma SIFT:

El algoritmo usado en esta técnica se muestra a continuación:

Figura 5.14: Flujograma SIFT

1. Detección de máximos y mínimos basándose en Scale-space: El primer paso es la búsqueda de puntos en la imagen que puedan ser *keypoints*. Para ello se aplica el método de diferencias de funciones gaussianos que sean invariantes a la escala y la orientación.

2. Localización de los *keypoints*: De los puntos anteriores, se debe determinar su posición (localización) y escala, de los cuales se seleccionan los *keypoints* finales basándose en la estabilidad de estos mismos.

3. Asignación de la orientación: A cada *keypoint* se le asigna una o más localizaciones, en función de las orientaciones de los gradientes locales de la imagen.

4. Descripción de los *keypoints*: Los gradientes locales son medidos y transformados en una representación que permite importantes niveles de la distorsión de la forma local y cambio en la iluminación.

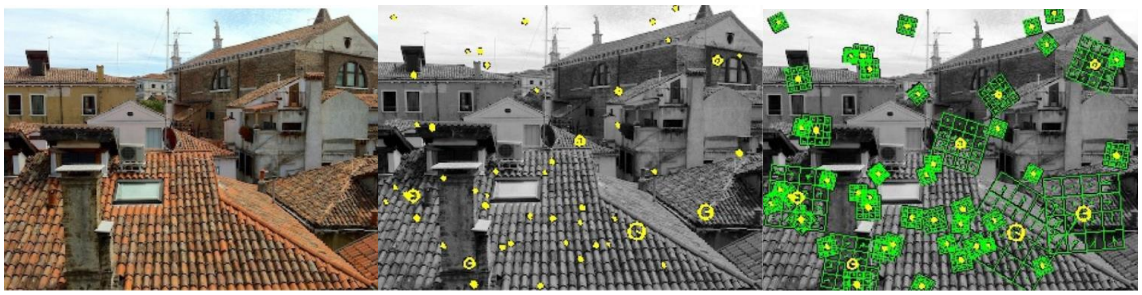


Figura 5.15: Izquierda, imagen original. Centro, imagen en escala de grises con detectores. Derecha, imagen en escala de grises con detectores y descriptores, Ejemplo extraído de la librería VLFEAT

Además de los keypoints, debe ser definido otro término conocido como ***frame*** (o marco en español) que **representa un objeto geométrico, como puede ser un punto, círculo o elipse que representa la localización y forma de uno de estos *keypoints*.**

5.4.2. Scale-Space:

Antes de detallar el proceso del algoritmo, debe estudiarse un concepto nuevo como es el ***Scale-Space*** (Escala-Espacio):

La distancia a la que están ubicados los objetos afecta a su percepción → Un objeto a mayor distancia presenta un tamaño menor en el campo visual (al igual que en una imagen) que el mismo objeto situado más cerca del punto de vista. Por tanto, un mismo objeto (una casa) puede aparecer con distintas dimensiones en función de la distancia al observador. **Para obtener una descripción precisa de los objetos que sea robusta ante tales variaciones en tamaño se utiliza la técnica *Space-Scale*.**

Koenderink (1984) y Linderberg (1994) demostraron como la forma de obtener una **familia de imágenes derivadas (es decir, su escala espacio)** $\boldsymbol{L(x, y, \sigma)}$ era mediante la **convolución de la función gaussiana de desviación típica σ y escala variable y la imagen de entrada** $\boldsymbol{I(x, y)}$, siendo esta función gaussiana:

$$G(x, y) = \frac{1}{2 \cdot \pi \cdot \sigma^2} \cdot e^{[-\frac{x^2+y^2}{2 \cdot \sigma^2}]} \tag{5.11}$$

Por tanto:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y) \tag{5.12}$$

Donde * es el operador convolución. Sin entrar en demostraciones matemáticas, la **convolución de imágenes consiste en una suma pesada de píxeles vecinos respecto del píxel fuente**. Los pesos son determinados por una matriz llamada **máscara de convolución**, que determinará los coeficientes a aplicar sobre una determinada área. La posición del valor central se corresponde con la posición del pixel de salida. Hay dos tipos de máscaras:

- Máscaras de convolución usadas para suavizar una imagen (también llamadas filtros paso bajo, cuya misión es la eliminación de ruido), sus coeficientes suman un total de 1.

- Máscaras de convolución usadas para la detección de bordes presentan coeficientes positivos y negativos que en total suman 0.

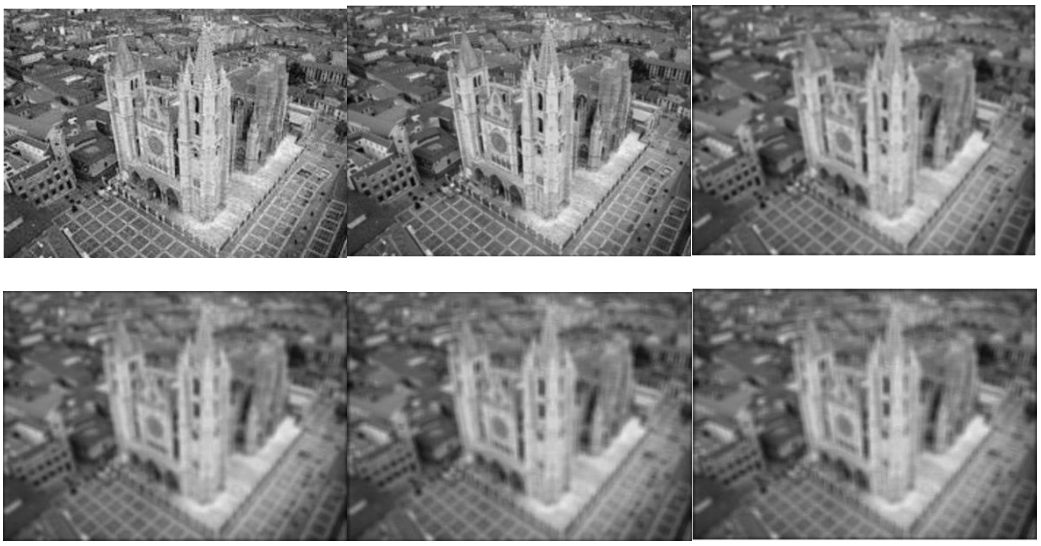


Figura 5.16: Familia de imágenes derivadas (suavizadas) $L(x, y, \sigma)$. Fila superior, de izquierda a derecha: σ = 0, 1 y 2. Fila inferior, de izquierda a derecha: σ = 4, 8 y 16.

Al calcular la familia escala espacio de una imagen original se consigue una invarianza respecto al escalado de esta misma, es decir, un mismo punto característico, si se identifica bien, será bien detectado tanto en una imagen como en una que tenga una escala 1:2.

Ya descrito el concepto de Scale-Space, se puede dar paso al algoritmo SIFT.

5.4.3. Algoritmo SIFT:

Como se comentó en el apartado anterior, el algoritmo SIFT se puede dividir en los siguientes pasos: 1. Detección de máximos y mínimos basados en espacio-escala, 2. Localización de los *keypoints*, 3. Asignación de la orientación y 4. Descripción de los *keypoints.*

5.4.3.1. Detección de máximos y mínimos basados en Scale Space:

Esta primera etapa se puede dividir a su vez en dos subetapas, la primera que corresponde al cálculo del espacio y escala y posteriormente al cálculo de la diferencia de gaussianas, y la segunda subetapa a la búsqueda de extremos locales a partir de esta DoG.

Para obtener de forma eficiente ubicaciones estables de los keypoints, **Lowe propuso calcular en primer lugar los valores extremos del espacio escala de la diferencia de Gaussianas de la imagen**, $D(x, y, \sigma)$, que se puede calcular como la **diferencia entre dos escalas consecutivas** (siendo una escala la cantidad de desviación típica aplicada a una imagen del mismo tamaño y una octava un conjunto de imágenes del mismo tamaño pero con desviaciones típicas o escalas distintas) **separadas por un factor multiplicativo constante k:**

$$D(x, y, \sigma) = \big(G(x, y, k \cdot \sigma) - G(x, y, \sigma)\big) * I(x, y) =$$

$$= L(x, y, k \cdot \sigma) - L(x, y, \sigma) \quad (5.13)$$

Nótese que la diferencia de gaussianas no es más que hallar la ejecución de dos desenfoques gaussianos en la imagen con distinta desviación típica y hallar la diferencia para obtener el resultado final. Dado que en este caso está convolucionado por la imagen de entrada, el resultado ($L(x, y, k \cdot \sigma) - L(x, y, \sigma)$) es una diferencia de familias de imágenes derivadas con distinta desviación típica (cada familia).

Las **principales razones** que llevaron a Lowe a tomar esta diferencia de gaussianas fueron:

- Dado que el cálculo de las imágenes suavizadas es obligatorio ($L(x, y, \sigma)$) para la descripción de características en el Space-Scale**, simplemente con iterar el proceso con un factor de multiplicación k y restando los resultados se obtiene $\boldsymbol{D(x, y, \sigma)}$**.
- **Presenta una buena aproximación al Laplaciano** (operador elíptico de segundo orden) de la Gaussiana normalizado a la escala ($\sigma^2 \cdot \nabla^2 \cdot G$), ya que la normalización del Laplaciano utilizando la varianza σ^2 es necesaria para que exista **una invarianza real frente a un cambio en la escala**.
- Los **máximos y mínimos** de $\sigma^2 \cdot \nabla^2 \cdot G$ producen las **características de imágenes más estables**, al compararlas por ejemplo con el algoritmo de Harris estudiado anteriormente.

Con el *Scale*-Space se crean espacios reduciendo sucesivamente el tamaño de la imagen original (además de distintas escalas debido a la desviación típica de la función gaussiana con la que se convoluciona). A cada uno de estos espacios se le denomina octava, y se obtienen eliminando una de cada dos filas y columna sobre la octava anterior (por tanto, reduciendo a la mitad). Este proceso es conocido como **"Obtención de puntos característicos a partir de los extremos del espacio-escala de diferencias de Gaussianas (DoG) dentro de una pirámide de diferencia de gaussianas".**

Una **pirámide Gaussiana** se construye por tanto mediante la **convolución con una función gaussiana G de desviación típica cada vez mayor y reduciendo la dimensión respecto de la octava anterior.** Finalmente, la **pirámide de diferencia de Gaussianas** (nótese que una pirámide es Gaussiana y otra de diferencia de gaussianas) se calcula a partir de las **diferencias entre dos niveles adyacentes de esta primera.** La relación entre dos desviaciones típicas consecutivas viene determinada por k (factor de multiplicación constante). El conjunto de estas n imágenes de suavizado progresivo es una escala, al dividir todas esas mismas por la mitad segunda octava (con el mismo suavizado progresivo), etc

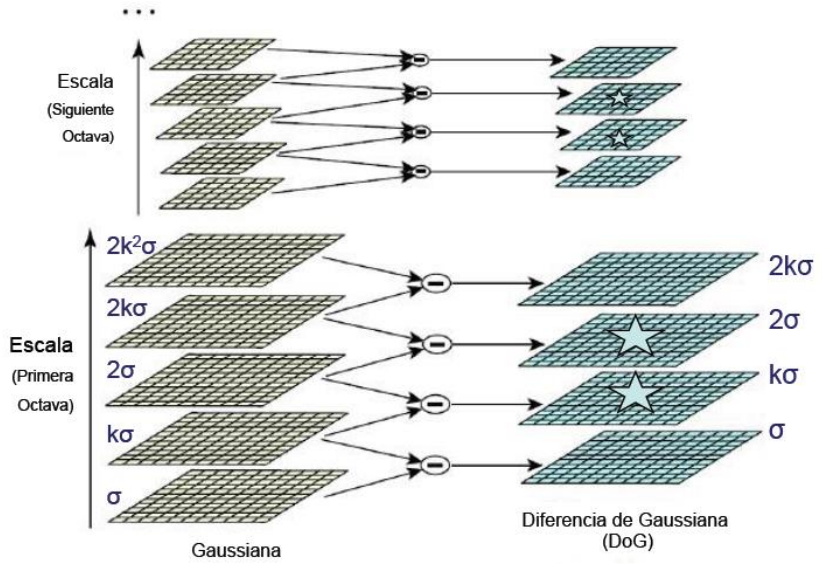


Figura 5.17: Pirámide Gaussiana y pirámide de diferencia de Gaussianas

Nótese que la búsqueda de máximos y mínimos locales se hace sobre la columna de la derecha, es decir, sobre la diferencia de gaussianas y no sobre la pirámide gaussiana como tal, siendo las imágenes con una estrella las únicas válidas porque son las únicas que presentan (en la columna de la derecha) nivel tanto superior como inferior.

Aunque el número de octavas y escalas dependen de la implementación (condicionadas por el tamaño y detalle de la imagen original), **generalmente 4 octavas y 5 escalas de desviación típica por octava son ideales para el algoritmo de detección de características invariantes.**

La segunda subetapa de esta primera etapa del algoritmo es la **detección de máximos y mínimos locales de la pirámide DoG**. Para ello, **cada punto de la muestra se compara con sus ocho píxeles adyacentes en la imagen en que se encuentra y también con los nueve píxeles** correspondientes (incluyendo su correspondiente píxel central) de la imagen que se encuentra a una **escala inferior y superior**. Un ejemplo perfecto se muestra a continuación:

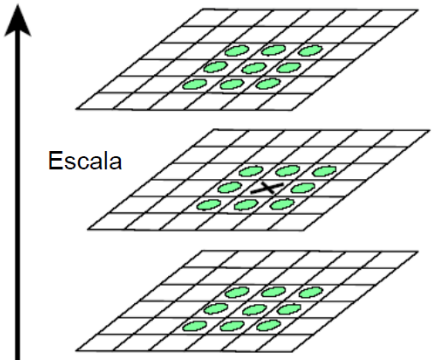


Figura 5.18: Comparación con píxeles periféricos de la misma escala, inferior y superior dentro de la misma octava

El píxel a evaluar está marcado con una cruz X y tanto sus adyacentes (en vertical, horizontal y diagonales) de la propia imagen, como los correspondientes en la escala inferior y superior (incluyendo los pixeles correspondientes al pixel marcado) están marcados con círculos verdes.

El punto se seleccionará como un posible extremo sólo si el valor de la función $\boldsymbol{D(x, y, \sigma)}$, sustituida para esa desviación típica σ y coordenadas x e y (para identificar al correspondiente pixel) **es mayor que todos los valores de los círculos verdes** (los otros 26, cada uno con sus correspondientes coordenadas y σ), **por lo que será un máximo, o bien es menor y será un mínimo.**

5.4.3.2. Localización de los keypoints:

El paso 1 (determinación de máximos y mínimos con *Scale-Space*), tiene la desventaja de proporcionar demasiados candidatos a *keypoints*, algunos inestables. **El siguiente paso es determinar la localización exacta, escala y proporción de curvaturas principales**. Esta información permitirá rechazar puntos mal localizados o sensibles al ruido.

En primer lugar, para cada *keypoint* se interpolan los datos obtenidos para determinar con precisión la posición de cada uno, mejorando la robustez del método. Esto se realiza mediante una **aproximación de Taylor de la ecuación DoG ($\boldsymbol{D(x, y, \sigma)}$):**

$$D(p) = D + \frac{\partial D^T}{\partial p} \cdot p + \frac{1}{2} \cdot p^T \cdot \frac{\partial^2 D}{\partial p^2} \cdot p \tag{5.14}$$

Donde p es el *keypoint* obtenido de la primera etapa. Si hacemos la derivada de p (el valor del *keypoint*):

$$p' = -\left(\frac{\partial^2 D^{-1}}{\partial p^2} \cdot \frac{\partial D}{\partial p}\right) = -\begin{bmatrix} \frac{\partial^2 D}{\partial x^2} & \frac{\partial^2 D}{\partial y \cdot \partial x} \\ \frac{\partial^2 D}{\partial y \cdot \partial x} & \frac{\partial^2 D}{\partial y^2} \end{bmatrix}^{-1} \cdot \begin{bmatrix} \frac{\partial D}{\partial x} \\ \frac{\partial D}{\partial y} \end{bmatrix} \tag{5.15}$$

Obteniendo así la expresión matemática para el extremo local (máximo o mínimo). Combinando las dos ecuaciones anteriores se obtiene:

$$D(p') = D + \frac{1}{2} \cdot \frac{\partial D^T}{\partial p} \cdot p' \tag{5.16}$$

Ecuación a partir de la cual se debe establecer un **umbral mínimo de contraste** (diferencia con el resto de regiones) al que deben llegar aquellos keypoints para no ser rechazados. Por ejemplo, si el umbral es 0.03 (o -0.03 en caso de mínimo), todos aquellos puntos que no cumplan $|D(p')| > 0.03$ serán eliminados.

Por otra parte, también se deben **descartar aquellos candidatos a *keypoints* que posean mala localización a lo largo de un borde**, ya que la función DoG posee una alta respuesta (a mayor DoG más probabilidad de que sea un punto de interés) a lo largo de los bordes.

Además, un pico pobremente definido en la función DOG indica que habrá una larga curvatura principal en la dirección del borde, pero pequeña en la dirección perpendicular del mismo. Esto intuitivamente es obvio:

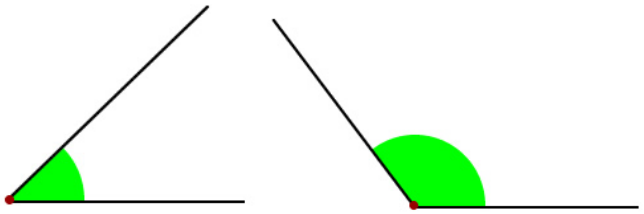


Figura 5.19: Ángulo agudo; ángulo obtuso

La curvatura principal se puede procesar a partir de una matriz hessiana bidimensional, evaluada en el keypoint:

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{5.17}$$

Cada elemento de la matriz son **derivadas calculadas a partir de tomar diferencias con los píxeles del punto de muestreo.** Los autovalores extraídos de la matriz son por tanto proporcionales a las curvaturas principales de la función DoG.

Tomando α como el mayor de los autovalores y β como el menor, se pueden establecer las siguientes ecuaciones:

$$Tr(H) = D_{xx} + D_{yy} = \alpha + \beta \tag{5.18}$$

$$Det(H) = D_{xx} \cdot D_{yy} - D_{xy}^2 = \alpha \cdot \beta \tag{5.19}$$

Siendo Tr la traza de la matriz hessiana (una traza en álgebra lineal es el sumatorio de los elementos de la diagonal principal de la matriz considerada, en este caso $D_{xx} + D_{yy}$) y evidentemente Det el determinante.

Si las curvaturas presentan signos diferentes (dirección del borde y dirección perpendicular al *keypoint*), este candidato se debe descartar ya que no será máximo ni mínimo. Si se toma r como la razón entre ambos autovalores:

$$r = \frac{\alpha}{\beta} \tag{5.20}$$

Entonces:

$$\frac{Tr(H)^2}{Det(H)} = \frac{(\alpha+\beta)^2}{r \cdot \beta^2} = \frac{(r+1)^2}{r} \tag{5.21}$$

La expresión anterior por tanto sólo depende del valor entre los autovalores. Este cociente es mínimo cuando ambos autovalores son iguales (por tanto r sería 1 y el cociente 4), y dado que r tiene como denominador el autovalor mayor, conforme aumenta la diferencia entre ambos autovalores, también crece r y también aumenta dicha expresión.

En conclusión, para verificar la relación entre las curvaturas, el cuadrado de la traza sobre el determinante debe cumplir:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(r+1)^2}{r} \tag{5.22}$$

Generalmente un valor de umbral razonable es tomar la relación de autovalores r igual a 10, por tanto la relación de curvaturas:

$$\frac{Tr(H)^2}{Det(H)} < \frac{(10+1)^2}{10} < 12.1 \tag{5.23}$$

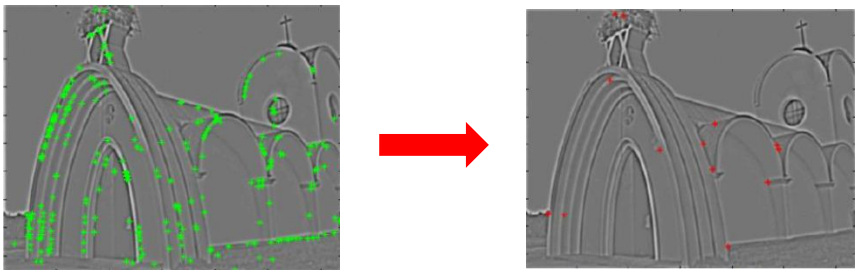


Figura 5.20: Izquierda, imagen con todos los *keypoints* ; Derecha, sólo aquellos que no han sido descartados

5.4.3.3. Asignación de la orientación:

Esta es la tercera etapa del algoritmo y es muy importante. Esto es así porque **si la orientación del *keypoint* es coherente,** basada en las propiedades locales de la imagen, el descriptor (cuarta etapa) puede ser relacionado con dicha orientación y por tanto ser **invariante a la rotación (invarianza a escala y rotación, objetivo trascendental en SIFT).**

Sin embargo, esta forma de proceder no la tienen todos los descriptores invariantes a la orientación, los cuales buscan propiedades de las imágenes en base a medidas invariantes a la rotación. La desventaja de este enfoque en SIFT es la limitación del número de descriptores que pueden ser usados y por tanto rechaza mucha información de la imagen.

Para lograr una adecuada orientación, este método está basado en el **gradiente local de la imagen alrededor de los keypoints.** Se utilizará la imagen suavizada por la gaussiana a la mayor escala determinada por el *keypoint*, de forma que los cálculos se realicen de un modo invariable a la escala

(debido al *Scale-Space*). Por cada imagen de muestreo L(x,y), se pueden precalcular la magnitud del gradiente M(x,y) y su orientación θ(x,y) (evidentemente todas las variables son función del pixel X Y):

$$M(x,y) = \sqrt{(L(x+1,y) - L(x-1,y)^2 + \big(L(x,y+1) - L(x,y-1)\big)^2} \quad (5.24)$$

$$\theta(x,y) = arctg\left(\frac{L(x,y+1)-L(x,y-1)}{L(x+1,y)-L(x-1,y)}\right) \quad (5.25)$$

A partir de las dos ecuaciones anteriores, (5.24) y (5.25) , se debe formar un **histograma de orientación** a partir de las orientaciones de los gradientes de los puntos de muestreo que se encuentran dentro de la región circundante al keypoint. Evidentemente deberá existir un histograma para cada uno de los *keypoints* resultantes de la etapa anterior.

Este histograma de orientación presenta 36 divisiones que abarcan los 360 º del rango de orientaciones. Cada muestra añadida al histograma es ponderada por su magnitud del gradiente y por una máscara gaussiana circular con un valor de 1.5 veces la desviación típica que posea el keypoint. Un ejemplo se muestra a continuación:

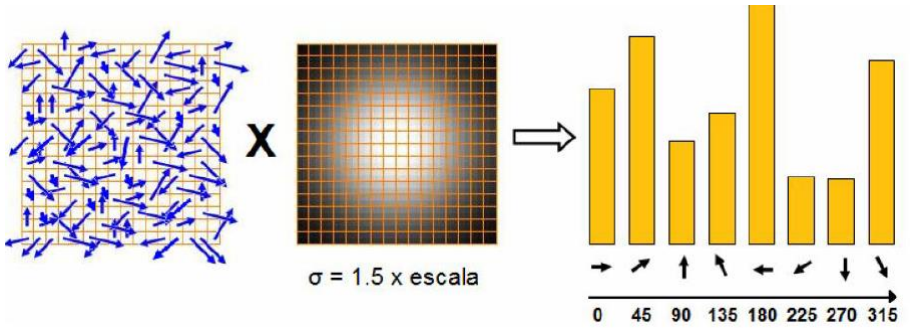


Figura 5.21: A la izquierda, región de gradientes 16 x 16; en el centro máscara gaussiana de desviación = 1.5, a la derecha histograma final de orientaciones

Los picos en el histograma de orientaciones corresponden a las direcciones dominantes de los gradientes locales (en el ejemplo mostrado la orientación dominante del local son 180 º). Se detecta el mayor pico del histograma (en este caso 180 º) y en caso de existir otros máximos superiores al 80 % del pico principal se utilizarán estos otros máximos para crear otros keypoints con nuevas orientaciones.

Por tanto, es **posible que existan keypoints con la misma localización pero con diferentes orientaciones**. Aproximadamente un 16 % de los keypoints disponen de orientaciones múltiples (o lo que es lo mismo, para un mismo punto existen dos o más keypoints de distinta orientación) que al contrario a lo que se pueda pensar aumenta enormemente la estabilidad del método, sobre todo cuando se intenta hacer el alineamiento entre imágenes mediante transformaciones homogéneas.

Finalmente, para determinar con mayor exactitud la localización del pico en el histograma, este histograma se interpola con una parábola que es fijada por los tres puntos más cercanos a cada pico (no se entrará en detalle en este proceso).

5.4.3.4. Descripción del keypoint:

Como ya se comentó anteriormente, la ventaja principal de las técnicas SIFT y SURF frente a otro tipo de técnicas es que estas dos presentan un **modo de detector y de descriptor**.

Mediante las tres etapas anteriores, a cada keypoint obtenido se les ha asignado una **posición en la imagen, una escala y una orientación**. Esta última etapa (cálculo del descriptor) consiste en caracterizar al *keypoint* y al mismo tiempo que presente gran invarianza frente a las dos grandes variaciones que quedan, como son los cambios de iluminación y el punto de vista 3D.

El proceso del cálculo del descriptor es largo y no es trivial. De forma resumida consta de los siguientes pasos:

1. Obtención de la magnitud y obtención de gradiente en una ventana de 16 x 16 muestras centrada en el keypoint.

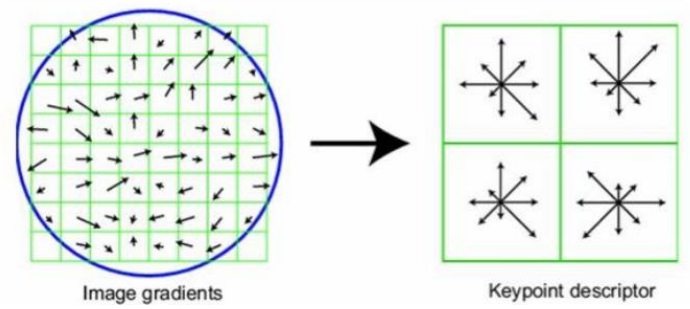


Figura 5.22: Gradientes de la imagen; Descriptor del *keypoint*

2. Cálculo de un histograma de barras que corresponden con las **direcciones angulares posibles del gradiente** para cada una de las 16 regiones que se obtienen al dividir la ventana inicial 16x16 en 16 ventanas 4x4. Cada barra del histograma de orientaciones se corresponde con 45 grados, es decir, el primero de 0 º a 44 º, el segundo de 45 º a 89 º, etc… hasta el último que va de 315 º a 359 º (puesto que 360 realmente es 0 º). El histograma presentará las orientaciones de los gradientes de esta ventana.

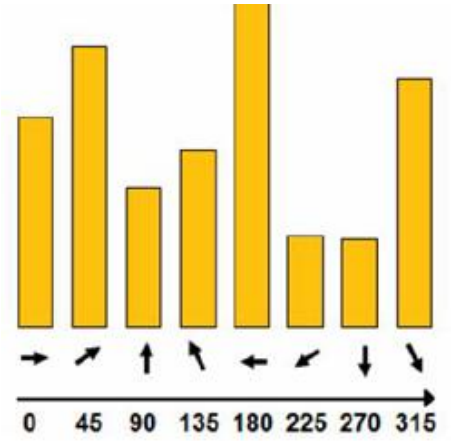


Figura 5.23: Direcciones angulares posibles del gradiente

Sin embargo, dicha construcción del histograma presenta un problema, y es que en caso de que exista un pequeño desplazamiento espacial (o pequeña rotación), la contribución de un pixel puede pasar a otra región y provocar cambios repentinos del descriptor.

Para evitar estos problemas, **se asume que todos los píxeles contribuyen a todos los vecinos tanto en magnitud como en orientación**. Esta contribución está ponderada obviamente por la distancia al centro, siendo el centro el keypoint central:

- Para el mismo píxel, obviamente su distancia (d) es 0, por tanto la contribución a si mismo es máxima.
- A medida que se aleja, d crece y la resta tiende a 0, disminuyendo la contribución.
- Obviamente la distancia está normalizada al tamaño de una región, es decir, la longitud de una región de 4x4 píxeles se toma como 1, por tanto la d máxima puede ser 1 que será para los píxeles más allá del extremo de la región, que no presentan ninguna contribución al central.
- Como la d máxima para una región es 1, la contribución de un píxel al central nunc apuede ser negativa.

3. Finalmente se normaliza un vector de 128 elementos que se obtiene al encadenar las ocho barras de cada histograma por los 16 histogramas correspondientes (puesto que hay 16 regiones, por tanto 16 · 8 barras = 128 elementos).

Este vector de 128 elementos es lo que se conoce como descriptor, que en resumidas cuentas consiste en un conjunto de elementos que contiene las orientaciones principales de un keypoint.

5.5. Técnica SURF [12]:

La técnica **SURF (*Speed-Up Robust Feature*)** es otro detector de variables locales, al igual que la técnica citada anteriormente SIFT, que puede actuar tanto en **modo detector como descriptor en función de las circunstancias**. Fue presentado por primera vez por **Herbert Bay** en 2006 y está inspirado en el descriptor SIFT presentando las siguientes mejoras:

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

Estas mejoras son obtenidas mediante la **reducción de la dimensionalidad y complejidad en el cálculos de los vectores de características** (es decir, los descriptores) de los puntos de interés obtenidos a pesar de seguir siendo lo suficientemente característicos y repetitivas.

SURF es otro de los algoritmos más utilizados en la extracción de puntos de interés en el reconocimiento de imágenes. Al igual que la técnica SIFT, la **extracción de puntos** la realiza detectando en primer lugar los candidatos a *keypoints* y después su localización dentro de la imagen.

Sin embargo, esta **técnica es mucho más rápida que SIFT** (nótese que cronológicamente hay siete años de diferencia, que en el mundo de la tecnología y en especial de la visión artifical es un mundo) ya que los ***keypoints* contienen muchos menos descriptores debido a que se descartan la mayoría de ellos por ser igual a 0**. Además, el descriptor de SURF se puede considerar una mejora en código, ya que con apenas pequeñas modificaciones respecto al código de SIFT (prácticamente la mayoría de las funciones son comunes) se consigue la reducción de estos descriptores.

5.5.1. Flujograma SURF:

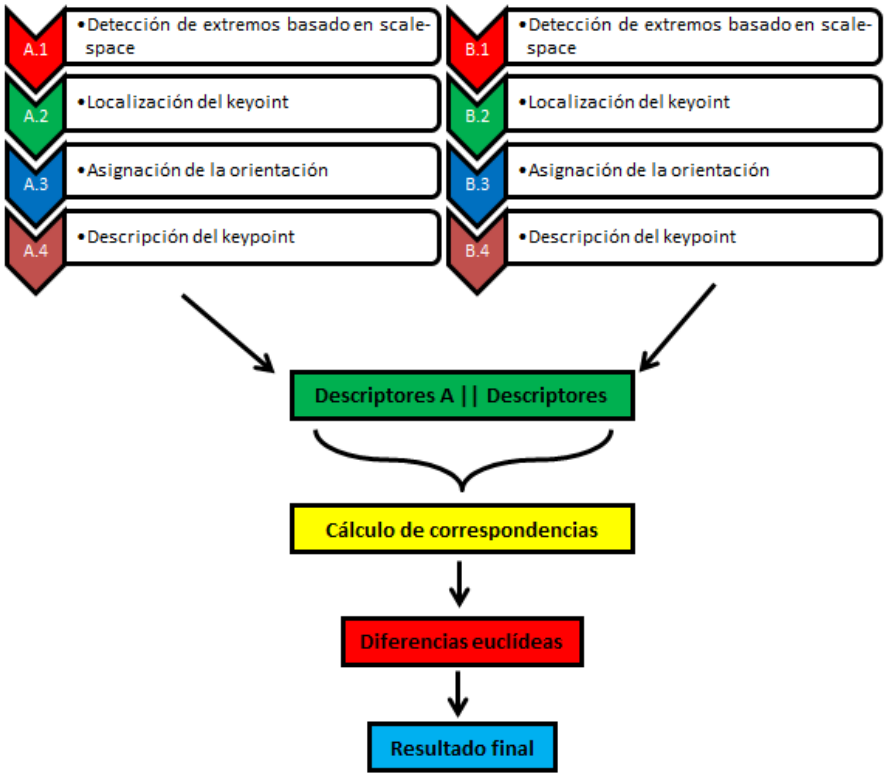


Figura 5.24: Flujograma SURF

5.5.2. Algoritmo SURF:

A continuación se explica el flujograma anterior 5.24 que corresponde al algoritmo de la técnica SURF.

5.5.2.1. Detección de keypoints y localización:

Al igual que con la técnica SIFT, la primera etapa para la determinación de estos puntos característicos es su **detección**. Los puntos de interés deben ser **encontrados en diferentes escalas**, entre otras cosas porque la búsqueda de correspondencias a menudo requiere su comparación en las imágenes donde se les ve a diferentes escalas.

La técnica SURF, en modo detector, hace uso de la **matriz Hessiana** y en concreto de su determinante para la localización y escala de los puntos. El motivo principal para la utilización de la matriz Hessiana es el gran **rendimiento en términos de velocidad de cálculo y precisión**. La novedad que presenta la técnica SURF (tanto en modo detector como en modo descriptor) frente a otras técnicas, particularmente en la detección, es que **no utiliza distintas medidas para el cálculo de la posición y la escala de los *keypoints*** individualmente (uno para escala; otro para localización), sino que utiliza el valor del determinante de la matriz Hessiana para ambos.

Siendo p un punto de la imagen I(x,y), es decir, p(x,y), la matriz Hessiana H(p, σ) del punto p se define como:

$$H(p,\sigma) = \begin{bmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{xy}(p,\sigma) & L_{yy}(p,\sigma) \end{bmatrix} \quad (5.26)$$

Siendo $L_{xx}(p,\sigma)$ la convolución de segundo orden de la función gaussiana $\frac{\delta^2}{\delta x^2} \cdot G(\sigma)$ con la imagen I en el punto x, y misma deducción para $L_{xy}(p,\sigma)$ y $L_{yy}(p,\sigma)$.

Al igual que en la técnica anterior, se puede aproximar la matriz Hessiana anterior a una matriz bidimensional con las derivadas parciales en las direcciones X e Y, denotadas como D_{xx}, D_{xy} y D_{yy}.

Por tanto, la matriz Hessiana de forma aproximada queda de la siguiente forma:

$$H(p,\sigma) = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{xy} & D_{yy} \end{bmatrix} \tag{5.27}$$

Y quedando el determinante:

$$\det(H_{aprox}) = D_{xx} \cdot D_{yy} - (0.9 \cdot D_{xy})^2 \tag{5.28}$$

Nótese que el coeficiente 0.9 del sustraendo del determinante procede de la aproximación pertinente del filtro gaussiano anterior.

En el ejemplo que se muestra a continución 5.25 se observa la representación de la derivada parcial de segundo orden de un filtro gaussiano (discretizado) y la aproximación de la matriz Hessiana de familia de imágenes suavizadas $L_{xx}(p,\sigma)$ por una matriz Hessiana de derivadas parciales:

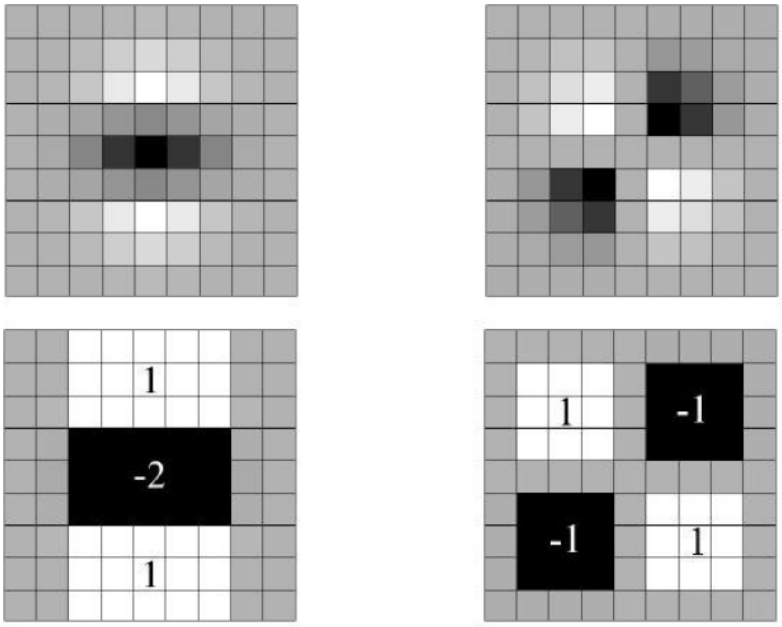


Figura 5.25: Representación de la derivada parcial de segundo orden de un filtro gaussiano

La imagen resultante, obtenida tras la convolución de la imagen original con un filtro de dimensión 9 x9, que se correspondiende con la derivada parcial de segundo orden de una gaussiana con desviación típica igual a 0.5 es considerada como la escala inicial (o dicho de otra forma, con la máxima resolución espacial). Por tanto, para un tamaño de 0.5, se corresponde con una gaussiana de 0.5.

Las **capas sucesivas son obtenidas mediante la aplicación gradual de filtros más grandes** (mayor dimensión), evitando el efecto de aliasing en la imagen. El concepto de *aliasing* procede del procesamiento de señales (Teoría de control) siendo éste el efecto que causa que señales continuas distintas se muestren indistinguibles cuando se muestran digitalmente. Si se hace una analogía en visión

artificial, el aliasing implicaría que la imagen no puede ser reconstruida convenientemente si sólo se utiliza el primer filtro gaussiano.

El *scale-space* para la técnica SURF, igual que con SIFT, está dividido en octavas. Debe recordarse que una **octava** es un conjunto de imágenes de igual dimensión que presentan a su vez una **escala** de suavizado en función de la desviación típica. Sin embargo, para la técnica SURF, las octavas están compuestas por un número fijo de imágenes como resultado de la convolución de imagen original por una serie (fija) de filtros de tamaño cada vez más grande. El incremento de los filtros (incremento del tamaño) dentro de una misma octava es el doble respecto del mismo filtro de la octava anterior, al mismo tiempo que el primero de los filtros de cada octava es el segundo de la octava predecesora (anterior). Evidentemente, la escala se sigue manteniendo y en cada escalón de una octava la desviación típica se aumenta con un factor de constancia k.

Finalmente, para calcular la **localización de los puntos de interés en todas las escalas, se eliminan aquellos puntos que no cumplan la condición de máximo en un "vecindario" de 3 x 3 x 3 puntos.** De esta forma, el máximo determinante de la matriz Hessiana se interpola en la escala y posición de la imagen.

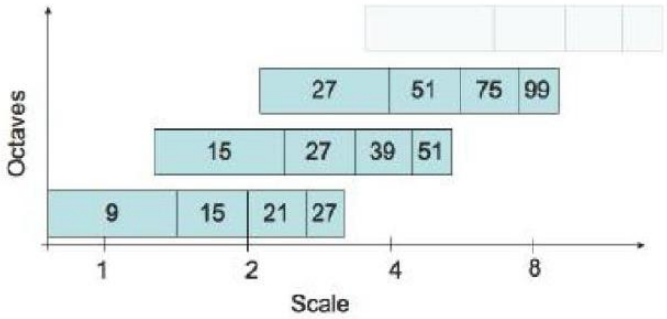


Figura 5.26: Ejemplo de la escala para el detector SURF.

Se puede observar como el segundo filtro de una escala corresponde al primero de la siguiente

5.5.2.2. Asignación de la orientación:

Como ya se comentó, el objetivo principal de un **descriptor es proporcionar una descripción única y robusta de un conjunto de datos, en este caso describiendo la distribución de intensidad u orientación del contenido circundante al keypoint evaluado**. Como está generado basándose en el área circundante a un punto de interés, se debe obtener un vector descriptor para cada keypoint.

El primer paso para obtener el descriptor con la técnica SURF es el cálculo de la orientación del keypoint. Para obtener dicho punto **invariante a la orientación** (precisamente el objetivo que se busca con el alineamiento) e iluminación se utiliza el **wavelet de Haar** (herramienta matemática que produce un análisis binario de la imagen. No se entrará en su concepto matemático) sobre las direcciones X e Y en una región circular de radio 4·s, siendo s la escala a la que pertenece el keypoint (por tanto indicando su desviación típica o dicho de otra forma su suavizamiento).

Los puntos de interés de la técnica SURF presentan una **repetibilidad característica**, lo cual significa que si un punto es considerado fiable, el detector encontrará el mismo punto bajo diferentes puntos de vista (escala, orientación, rotación, etc)

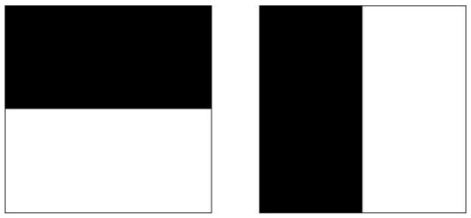


Figura 5.27: Cálculo de la respuesta de Haar (Wavelet-Haar): Negro = -1, Blanco = 1

Tras realizar los cálculos pertinentes con el wavelet de Haar, se utilizan imágenes integrales de nuevo para proceder al filtrado mediante máscaras de Haar y obtener así las respuestas en ambas direcciones. La ventaja de este método es que con tan solo seis operaciones se puede obtener la respuesta en ambas direcciones X e Y. Una vez que las respuestas onduladas (resultados de la máscara de Haar) han sido calculadas, son suavizadas por una gaussiana de desviación típica $\sigma = 2.5 \cdot s$ (siendo s la escala del keypoint) centrada en el punto de interés.

Estas respuestas son representadas como **vectores en un sistema de coordenadas tal que el eje de ordenadas (vertical) es la respuesta de Haar en el eje Y y en el eje de abcisas (horizontal) es la respuesta de Haar en el eje X.**

Finalmente, se obtiene una orientación dominante por cada sector a través de la suma de todas respuestas dentro de una ventana que cubre un ángulo de $\pi/3$.

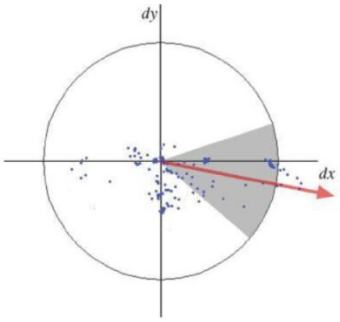


Figura 5.28: Respuestas en X e Y de Haar

Se puede observar como la orientación dominante está determinada por la ventana de ángulo $\pi/3$ (sector circular de color oscuro) donde se encuentra la mayoría de las respuestas.

5.5.2.3. Descripción del keypoint:

Tras haber detectado el keypoint con técnica scale-space, localizarlo posteriormente de forma precisa y en última instancia asignarle orientaciones, se debe proceder a la descripción del keypoint para que contenga la suficiente información para ser invariante frente a **cambios de iluminación y punto de vista 3D.**

Se construye en primer lugar una región cuadrada de $20 \cdot s$ (s es la escala del keypoint a evaluar) alrededor del punto de interés y orientada en relación con la orientación obtenida en la etapa anterior. A su vez, esta región de interés se divide en 4 x 4 subregiones, dentro de las cuales se calculan las respuestas de Haar con una separación de muestreo de 5 x 5 puntos en ambas direcciones.

Por simplicidad, se consideran Rx y Ry respectivamente las respuestas de Haar en la dirección horizontal y vertical relativas a la orientación del punto de interés.

Para dotar a las respuestas Rx y Ry de una mayor robustez ante deformaciones geométricas y errores de posición, ambas son suavizadas (ponderadas) con una función gaussiana de valor 3.3 · s, centrada en el keypoint. En cada una de las 25 subregiones se suman las respuestas Rx y Ry, obteniendo así un valor de Rx y Ry representativo para cada una de las subregiones. Al mismo tiempo, se realiza la suma de los valores absolutos de ambas respuestas obteniendo esa manera información de la polaridad sobre los cambios de inensidad.

Cada una de las 25 subregiones queda determinada por tanto por un vector de componentes v:

$$v = (\sum R_x, \sum R_y, \sum |R_x|, \sum |R_y|) \tag{5.29}$$

Como el vector tiene cuatro componentes, y existen 16 subregiones (4 x 4), resulta en un **descriptor SURF de 64 componentes** (4 x 4 x 4) para cada uno de los puntos de interés identificados, a diferencia del descriptor SIFT que presentaba 128 componentes (justo el doble).

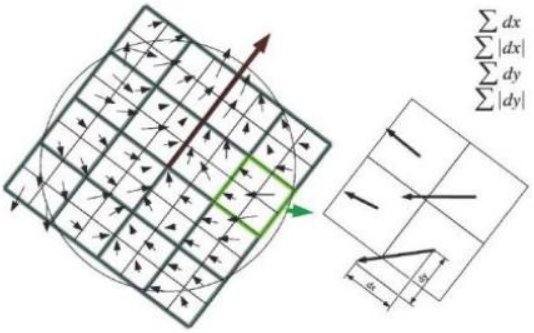


Figura 5.29: Descriptor SURF

Se puede observar en la figura como dx (también llamado Rx, e igualmente dy = Ry, simple nomenclatura) es la proyección en X del vector (con su magnitud y orientación) de dicho punto en la subregión, que es lo que se calcula con la respuesta de Haart.

Ya descritas las técnicas SIFT y SURF con un grado moderado de detalle (puesto que ambas técnicas para su total compresión precisan de explicaciones matemáticas no triviales, pero que no son objetivo de este TFG), se realizará una última comparación entre ambas técnicas.

5.6. Diferencias principales entre las técnicas SIFT y SURF:

Las diferencias principales entre ambos algoritmos son:

- **Velocidad de aplicación** (por eliminarse la mitad de los elementos en el descriptor SURF, es más rápido, a pesar de que en ambos obtenemos puntos de interés invariantes en escala, orientación, rotación, cambios de iluminación y punto de vista 3D).
- Mientras que **en SIFT se guarda la posición, escala y orientación para un keypoint** (ya que es posible que **en una misma posición X, Y de la imagen pueda haber dos keypoints de diferente escala u orientación**), en SURF **no se guarda la escala ni la orientación** (ahorrando en recursos, **porque para una posición X, Y sólo puede aparecer un *keypoint***) a cambio de registrarse la matriz de segundo orden y signo del Laplaciano para el posterior cálculos de las respuestas de Haart.
- **La respuesta de invarianza ante la rotación es más satisfactoria en SURF que en SIFT** (mayor acierto en los futuros inliers tras el filtro MSAC).

- Entre la versión del descriptor de **64 elementos (SURF) y la versión de 128 (SIFT),** en general el vector de 64 presenta cierta mejoría debido a que al tener menos descriptores el vector tendrá **menos probabilidad de fallo al ser el número de elementos menor.**

En conclusión, se puede decir que el algoritmo SURF es una mejoría respecto al algoritmo SIFT, si bien siempre dependerá de la aplicación para la que se use.

Finalizado el capítulo de los descriptores, es hora de hablar de la última etapa del proceso para el alineamiento de imágenes. En capítulos anteriores se trató la adquisición de imágenes (vistas sintéticas a partir del panorama y el mapa de profundidad), en este capítulo se ha tratado la detección y descripción de puntos característicos (así como la definición en moderado grado de detalle los principales algoritmos usados para este TFG, que han sido el detector de contornos de Canny, el detector de esquinas de Harris y los detectores/descriptores de características SURF y SIFT) en el siguiente capítulo se tratará el proceso de matching o emparejamiento de estos *keypoints*.

Capítulo 6

Alineamiento de vistas sintéticas mediante corrección de offset y transformación homogénea:

En los capítulos anteriores se trató la síntesis de imágenes, detección y descripción de puntos característicos. Si se observa el flujograma 6.1 que comprende la aplicación desde que están los panoramas y mapas de profundidad, hasta que se etiquetan las imágenes:

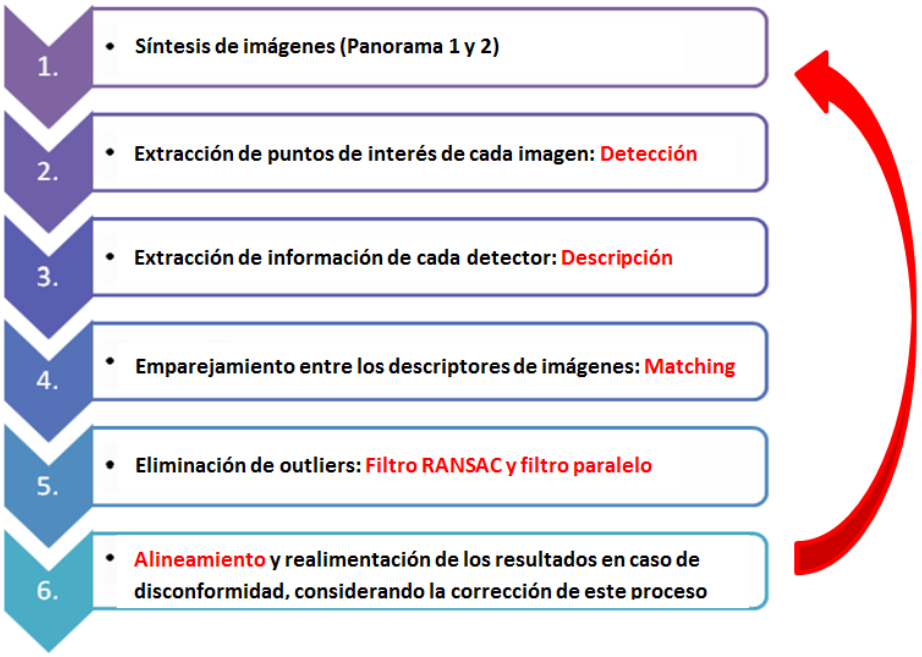


Figura 6.1: Flujograma parcial del proceso de la aplicación

Se puede observar que las etapas 1 (síntesis), 2 (detección de keypoints) y 3 (descripción de keypoints) ya han sido completadas. En este capítulo se tratarán las tres etapas siguientes: Emparejamiento de los descriptores o ***matching*, eliminación de los *keypoints* indeseados a base de filtros y alineamiento de las imágenes**. Por último, se tratará la última etapa de la aplicación creada que es el etiquetado de las imágenes para la elaboración de los ***groundtruth***, necesarios como datos para el entrenamiento de la Red Neuronal Convolucional (*Convolutional Neural Network, CNN*).

6.1. Concepto de Emparejamiento o matching:

El concepto de ***matching* o emparejamiento** realmente es muy simple: Consiste en **determinar qué píxel** (punto) **de la imagen nueva** (en este caso, la sección del segundo panorama) **se corresponde con su homólogo en la sección antigua** (en este caso, la sección del primer panorama).

En la práctica es casi imposible que dos secciones sintetizadas (por ejemplo la primera del primer panorama de las tres que se generan, y la homóloga del segundo panorama) estén perfectamente alineadas, debido a la distinta ubicación del coche de GSV en cada pasada, la iluminación de cada pasada, altura de la cámara, entre otras cosas. **Por eso mismo las técnicas que mejores resultados ofrecen (SURF y SIFT) son las que presentan las características de invarianza a la rotación, orientación, escala, cambios en la iluminación y punto de vista 3D.** Esto quiere decir que si en la sección 1 se ha

determinado un *keypoint*, por ejemplo, la esquina de la ventana de un edificio, si se define y caracteriza bien (descripción), ante cambios en la posición (por ejemplo que la ventana estuviese en otro sitio respecto de la imagen inicial), cambio de color por el cambio de luminosidad, cambio en la escala por desplazamiento hacia la estructura del coche, etc… debería emparejarlo igualmente.

Evidentemente, es imposible el alineamiento exacto de todos y cada uno de los píxeles de la segunda sección con los de la primera. **Las funciones utilizadas en MatLab para el emparejamiento tienden a corresponder únicamente los *keypoints***, por tanto, el proceso de emparejamiento de puntos va ligado íntegramente con la etapa anterior, el proceso de descripción, siendo muy importante la técnica utilizada (SIFT, SURF, etc) para la obtención de un mayor o menor número de correspondencias.

Una conclusión que se puede sacar del apartado anterior es la siguiente: **Cuantos más *keypoints* hayan sido identificados entre imágenes, más referencias habrá para el futuro *matching* y más se ajustarán ambas imágenes**. Imagínese un marco de una puerta, cuantos más clavos o puntos de referencia se pongan, mayor fiabilidad tendrá el resultado final con respecto al resultado esperado porque se han cumplido muchos requisitos. Por otro lado, cuantos menos clavos se pongan, el marco de la puerta tendrá menos probabilidad de estar alineado respecto al resultado final.

Esta conclusión queda aún más reforzada tras el filtrado de estos *keypoints* con el filtro MSAC y la corroboración de este filtro con un filtro paralelo, para determinar el número final de inliers para una combinación de técnicas (Harris, SURF y SIFT) con técnica de estimación geométrica (Similarity, Affine y Projective). Esta matriz de calidad se verá en el apartado de alineamiento dentro de este capítulo.

Esta etapa de *matching* es de suma importancia en la aplicación para el futuro alineamiento de las secciones. Para que este alineamiento se produzca de forma correcta, es necesario la eliminación de las falsas correspondencias, también denominadas ***outliers.***

Aunque la técnica del *matching* tiende a corresponder aquellos keypoints que presentan descriptores idénticos, siempre o casi siempre existe un pequeño número de puntos característicos que se han correspondido de forma errónea. Para este TFG se estudiarán tres tipos de filtros: El **algoritmo del vecino más próximo, el filtro MSAC basado en el filtro RANSAC y el filtro paralelo** (los dos primeros disponibles en MatLab y el último creado por el autor).

Sin embargo, a pesar de que la conclusión anterior se cumple casi siempre, es deber del usuario que esté ejecutando la herramienta la **supervisión** de estas la misma, ya que la característica *random* del filtro MSAC (derivado del RANSAC) puede ocasionar que una combinación con menos inliers finales (*keypoints* buenos) tenga una representación mucho más ajustada a la realidad que otra combinación con más inliers fnales. Para la supervisión automática del filtro MSAC se ha elaborado un filtro basado en paralaje que se explicará a continuación.

6.2. Algoritmo del vecino más próximo:

Este algoritmo está basado en el **cálculo de las diferencias euclídeas entre los elementos de cada descriptor** (básicamente hallando la hipotenusa que los une de un triángulo virtual formando por las coordenadas). Una vez calculados los descriptores (en el caso de SIFT un vector de 128 elementos y en el caso de SURF un vector de 64 elementos), se sabe la orientación alrededor de un punto de interés, por tanto cuando se encuentren descriptores con características similares en teoría la región debe ser la misma.

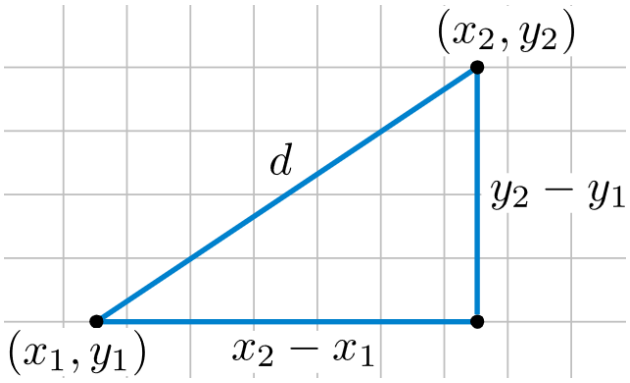
Para diferenciar los keypoints buenos de los malos se puede usar el citado algoritmo de comparación del vecino más próximo (NN, Nearest Neighbor):

$$dif_i = \sqrt{(a_i - b_i)^2} \tag{6.1}$$

$$dif_{total} = \sum_1^{nºelementos} dif_i \tag{6.2}$$

Siendo nº elementos la longitud del descriptor, 128 para SIFT y 64 para SURF, dif_i la diferencia euclídea (evidentemente se hace la diferencia tanto para la coordenadas X como para la coordenadas Y) entre el elemento a_i perteneciente a la imagen A (en este caso, sección del primer panorama) y el elemento b_i perteneciente a la imagen B (en este caso, sección del segundo panorama) por lo que se puede reescribir de la siguiente forma:

$$dif_i = \sqrt{\big(a_i(x) - b_i(x)\big)^2 + \big(a_i(y) - b_i(y)\big)^2} \tag{6.3}$$



Evidentemente, la variable dif_{total} es igual a la suma de las diferencias euclídeas que hay entre los nº elementos de ambos descriptores. Realizando esta serie de operaciones entre cada descriptor de la primer sección con los de la segunda, se podrá decidir cuales son iguales, tomando como referencia que a menor diferencia euclídea total, mayor probabilidad de que ambas regiones sean la misma.

Figura 6.2: Distancia euclídea

6.3. Filtro MSAC:

Este filtro de exclusión de los outliers está basado en el algoritmo **MSAC**, el cuál está basado a su vez en el algoritmo **RANSAC** [14]. Para ello se aplicará en este proceso la función ***estimateGeometricTransform*** de la toolbox Computer Vision de MatLab que en base a estas *matching features* (correspondencias) entre ambas imágenes (procedentes de la correspondiente técnica Harris, SURF o SIFT y un determinado método de transformación Similarity, Affine o Projective) se determina la matriz de transformación de la segunda sección con respecto a la primera y los *inliers* (buenas correspondencias) efectivos de cada par de secciones.

```
[tform_SIFT,inlierPtsDistorted_SIFT,inlierPtsOriginal_SIFT] = estimateGeometricTransform...
(matchedPtsDistorted_SIFT,matchedPtsOriginal_SIFT,'similarity');
```

RANSAC (RANdom SAmple Consensus, publicado en 1981 por Fischler y Bolles) es un método iterativo que permite **calcular los parámetros de un modelo matemático a partir de un conjunto de datos observados que contiene valores atípicos o aberrantes** (que se salen de la tendencia, por ejemplo dentro de una población con una media de 1.80 m, un valor atípico sería una persona que mida 2.25 m o en su defecto 1.35 m).

Este algoritmo se define como **no determinista** dado que los parámetros obtenidos son razonables sólo con una cierta probabilidad, dependiendo del número de iteraciones realizadas (a mayor número de iteraciones, más probable que los parámetros obtenidos sean razonables).

Los datos consisten en ***inliers***, que son los datos cuya distribución es explicada por un conjunto de parámetros del modelo (en el caso de las imágenes, los *inliers* corresponden con los **buenos *keypoints***) y

los **valores atípicos o *outliers*** (en el caso de las imágenes, corresponden con los malos ***keypoints***) que son los datos que no encajan en el modelo.

Estos *outliers* pueden provenir de valores extremos de ruido, mediciones erróneas, hipótesis incorrectas etc. En el caso del procesamiento de imágenes estos factores de error se deben principalmente a **cambios en la iluminación, perspectiva o ruido presente en la imagen**, por lo que sería equivalente a decir que dos *keypoints* se han correspondido con descriptores muy parecidos pero no idénticos, y que sin embargo estén en localizaciones totalmente diferentes de la imagen. Cuanto mayor sea la complejidad de los descriptores usados, mayor fiabilidad tendrán las correspondencias, aunque el costo computacional será mayor, como puede ocurrir con los descriptores SURF y SIFT frente a otros descriptores menores como Harris.

El algoritmo RANSAC asume que dado un conjunto de buenos *keypoints* (*inliers*), existe un procedimiento para estimar los parámetros de un modelo que explica dicha distribución.

La diferencia de este algoritmo frente al del algoritmo del vecino más próximo (NN) estudiado en el apartado anterior (6.2) es que este último dotaba a todos los elementos del vector con el mismo peso, por lo que la presencia de un *outlier* podría distorsionar el modelo obtenido, mientras que con RANSAC no ocurre eso.

Por ejemplo, un ejemplo sencillo es la inserseción de una línea en un conjunto de observaciones sobre un plano bidimensional. Si efectivamente hay un gran número de *inliers,* una línea se podrá insertar de forma que la mayoría de *inliers* queden sobre esta misma y los valores atípicos o *outliers* queden fuera. La gran diferencia entre RANSAC y el algoritmo NN es que para este último la línea se hace por mínimos cuadrados aproximando a todos los puntos (incluyendo a los atípicos, por lo que la línea, a pesar de quedar centrada sobre todo donde haya más elementos, presentará errores de desviación hacia los atípicos) mientras que RANSAC sólo toma en cuenta los inliers para determinar la línea (de ahí el término iterativo, se determinan los inliers para determinar la línea para corroborar los inliers, etc)

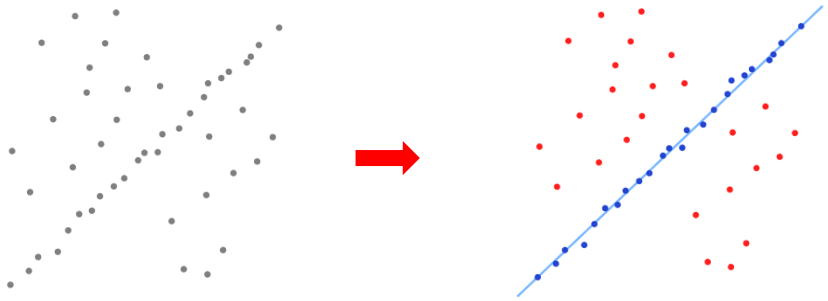


Figura 6.3: Sin aplicar RANSAC al conjunto de datos ; Aplicando RANSAC

6.3.1. Algoritmo RANSAC:

La idea principal de RANSAC es hacer uso de la **mínima cantidad de datos posible para estimar un módelo de buenos *keypoints*** y posteriormente ver cuántos puntos se ajustan al modelo estimado, por tanto, se dice que RANSAC es capaz de filtrar datos (en mayor o menor medida) que contienen gran cantidad de errores. Los pasos del proceso iterativo son los siguientes:

1. Selección de un **subconjunto aleatorio** de los datos originales ("*inliers* hipotéticos").

2. Un **modelo se construye alrededor** de esos datos originales.

3. Se **prueban los datos** (el resto que queda tras la elección del subconjunto) **contra el modelo ajustado.** Los datos que inicialmente no se encontraban dentro del subconjunto de *inliers* hipotéticos pero que sin embargo se ajustan al modelo estimado, se incorporan al modelo.

4. Si se han **clasificado suficientes puntos como buenos** dentro del subconjunto, se puede dar el modelo estimado por bueno.

5. La **iteracción** de los cuatro pasos anteriores permite aumentar el número de puntos del subconjunto mejorando indirectamente el modelo.

El número de veces que se itera suele ser fija, generando un modelo que sea rechazado porque no hay suficientes puntos o de otra forma si el subconjunto cada vez tiene más datos después de la iteracción del proceso es aceptado.

Existen **cuatro parámetros principales** para caracterizar al modelo RANSAC:

- **t y d**, los cuales son los principales parámetros del modelo y deben ser determinados a partir de requisitos específicos relacionados con la aplicación y el conjunto de datos.
- **k, número de iteracciones**, se puede determinar a partir de cálculos teóricos.
- **p, parámetro que determina la probabilidad de que RANSAC solo seleccione *inliers* a partir del conjunto de datos de entrada cuando eligen n puntos con los que se estiman los parámetros del modelo**. Sin embargo, a menudo el parámetro p se suele dividir en dos parámetros, renombrando p como la probabilidad de que el algoritmo produzca un resultado útil y w la probabilidad de hallar un inlier cada vez que se selcciona solo un punto.

$$w = \frac{Nº\ inliers\ en\ el\ conjunto}{Nº\ datos\ totales\ en\ el\ conjunto} \tag{6.4}$$

Normalmente w no se conoce previamente aunque se puede estimar. Suponiendo que los n puntos requeridos para la estimación se seleccionan de forma independiente, $\boldsymbol{w^n}$ **es la probabilidad de que todos los resultados sean inliers** (debe recordarse que las probabilidades se multiplican, no se suman. Por ejemplo, si un sábado tiene una probabilidad de que llueva del 50 %, al igual que el domingo, la probabilidad de que llueva en todo el fin de semana no es del 50 % + 50 % = 100 %, sino que será del 50 % · 50 % = 25 %) y $1 - w^n$ es la probabilidad de que al menos uno de estos n valores sea un valor atípico, si se itera k veces, la probabilidad de que el algoritmo nunca seleccione un conjunto de n puntos donde todos son inliers debe cumplir:

$$1 - p = (1 - w^n)^k \tag{6.5}$$

Donde como se comentó anteriormente p indica la probabilidad de que el conjunto de datos sólo presente inliers.

Para estimar el número de iteracciones:

$$k = \frac{\log(1-p)}{\log(1-w^n)} \tag{6.6}$$

Asumiendo por supuesto que los n datos se seleccionan de forma independiente. Esta independencia quiere decir que un punto que haya sido seleccionado una vez, se sustituye pudiendo ser seleccionado de nuevo en la siguiente iteracción, o sea, no hay descarte. El valor de k iteracciones resultante es el límite superior teórico en caso de que los puntos se seleccionen sin reemplazo (cosa que no es así normalmente, cuando se va un valor del conjunto entra otro).

6.3.2. Principales ventajas y desventajas del filtro RANSAC:

Ventajas:

1. Estimación robusta de los parámetros con alta precisión incluso cuando existe un número significativo de los valores atípicos.

2. Comprimoso mediante el cálculo de un **mayor número de iteracciones** incrementando la probabilidad de obtener un **modelo razonable.**

Desventajas:

1. No hay tiempo máximo para calcular los parámetros.

2. Cuando el **número de iteracciones calculadas se limita a la solución obtenida**, puede no ser un resultado óptimo.

3. RANSAC **no siempre es capaz de encontrar la configuración óptima** incluso para modelos notablemente repletos de valores atípicos, por lo que en general tiene un rendimiento pobre cuando el número de inliers tiene una presencia inferior al 50 %.

4. Requiere el **establecimiento de umbrales** para problemas específicos, es decir, no es un recurso general sino que deben ser ajustados ciertos parámetros para el aumento de fiabilidad.

5. RANSAC sólo puede **estimar un modelo para un conjunto de datos en particular.**

6.3.3. Derivación de RANSAC: Filtro MSAC:

Debido a que en sus inicios en 1981 presentaba más desventajas que ventajas, y dado su gran horizonte de futuro, RANSAC se convirtió en una herramienta fundamental en los campos de la visión artificial y procesamiento de imágenes.

RANSAC **puede ser sensible a la elección del umbral del ruido** adecuado que define qué puntos se ajustan al modelo. Si el umbral es demasiado grande, los puntos tienden a ser clasificados por igual, mientras que si el umbral es muy pequeño, los parámetros estimados tienden a ser inestables, es decir, simplemente añadiendo o quitando un dato al conjunto, puede fluctuar los parámetros. Para compensar este defecto, se propuso la variante **MSAC** (utilizada en este TFG, M-estimador SAmple and Consensus) cuya idea principal es, además de estimar los parámetros anteriores, **evaluar adicionalmente la calidad del conjunto de datos de forma aleatoria.**

6.4. Filtro paralelo:

A pesar de que los resultados tras el filtro MSAC son muy buenos, eliminando la gran mayoría de los outliers, **es posible que dentro de los *inliers* exista un mínimo porcentaje de malas correspondencias**, pero sin embargo fatales para la posterior transformación.

Por tanto, se ideó durante la creación del TFG un **filtro basado en el paralaje** de las matching features para la correción (en caso de que existieran valores atípicos) del resultado tras el filtro MSAC. No es un filtro exacto, sino más bien empírico y está basado en determinar el módulo y ángulo de cada par de teóricos *inliers* correspondidos, estando los *inliers* de la segunda sección desplazados 640 unidades en la horizontal (porque ambas imágenes se representan en la interfaz en modo *montage*) y alineadas en la vertical.

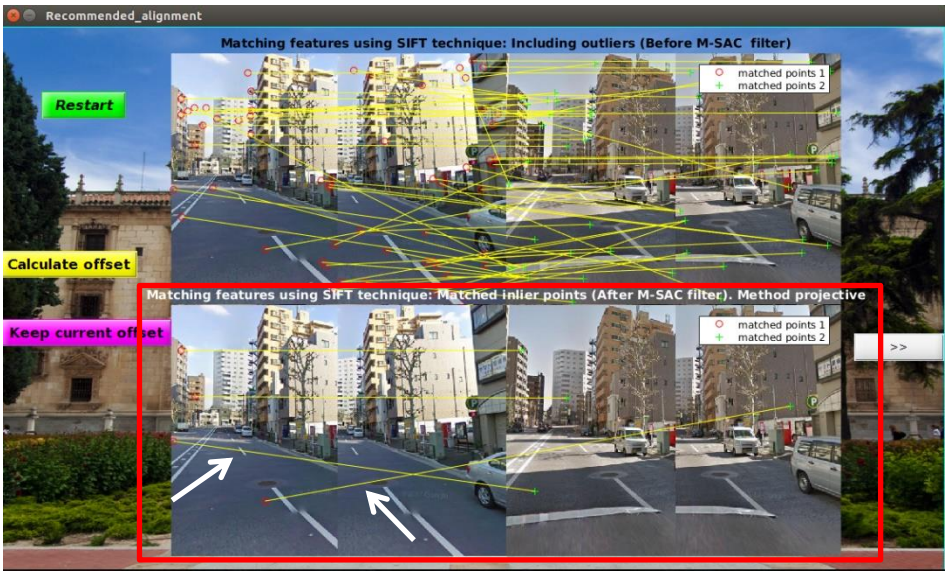


Figura 6.4: Ejemplo de malas correspondencias

En el recuadro rojo de la Figura 6.4 se puede observar dos malas correspondencias, incluso después de haber aplicado el filtro MSAC.

La explicación es la siguiente:

1. En primer lugar se determina la técnica a evaluar en este momento (Harris, SURF o SIFT) así como el método de transformación especificado en la función del filtro MSAC *estimateGeometricTransform* (Similarity, affine, projective), y se almacena en la variable k la diferencia en **coordenada X de cada par de *inliers*** (estando el valor X de los *inliers* de la segunda sección desplazado por 640 unidades) y en la variable y la **diferencia en coordenada Y.**

```
k=(effective_points_distorted(i,1)+640)-effective_points_original(i,1); % X difference
l=effective_points_distorted(i,2)-effective_points_original(i,2); % Y difference
```

2. Se toma el primer par de inliers, tomando la hipótesis de que este es bueno y se establece su **diferencia euclídea**, es decir, el módulo que los separa:

```
modulos(1)=sqrt(k^2+l^2);
```

3. Se determina el **ángulo existente entre el ángulo que habría que girar el punto 1** (original) **respecto del punto 2** (nuevo) para colocarlos en la misma fila (considerando igualmente que están cada uno en ambas imágenes, por tanto existe un desfase 640 en la horizontal). Esto se hace a partir de una serie de reglas basadas en el sistema cartesiano:

```
if(k>=640 && l>=0)
    angle=(360/(2*pi))*asin(l/modulos(1)); % For each pair of inliers, it is the angle from the original
                                           % inlier to the distorted inlier
elseif (k<640 && l>=0)
    angle=180-(360/(2*pi))*asin(l/modulos(1));
elseif (k>=640 && l<0)
    angle=-(360/(2*pi))*acos(k/modulos(1));
else (k<640 && l<0)
    angle=180+(360/(2*pi))*acos(-k/modulos(1));
end
```

Con las anteriores fórmulas **los ángulos resultantes están comprendidos en el rango de 0 a 359 º**, es decir, si la diferencia en X es mayor que 640 (lo que implicaría que el punto de la sección 2 está más a la derecha en su sección respecto de lo que está el punto de la sección 1 con la suya) y la diferencia en Y es mayor que 0 (el punto de la sección 2 está más arriba que el punto de la sección 1), significa que el ángulo existente entre ambos puede tomar un rango de valores de 1 a 89 º, es decir, todo el primer cuadrante.

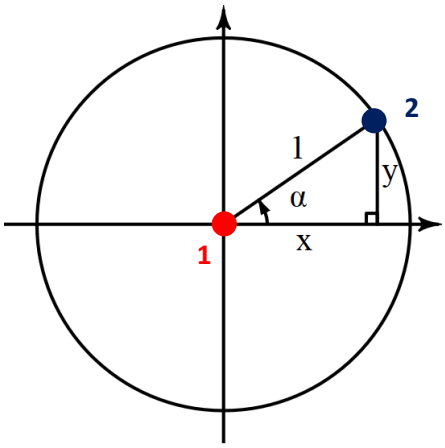


Figura 6.5: Alineamiento de ángulos

La deducción es similar para el resto de cuadrantes. Con esto se consigue determinar el ángulo del vector que une ambos puntos con el *montage* citado.

4. A continuación se ***itera n-1 veces***, siendo n el número de *inliers* teóricos de la combinación a analizar en el momento, restándole – 1 porque el primer par de *inliers* se ha usado para determinar el módulo y ángulo del primer vector. Este paso es crucial: Aunque la síntesis de imágenes puede originar una falta de alineamiento entre las secciones de un mismo par, ante un moderado cambio en la traslación (con tendencia a mínimo, de ahí el proceso tan tedioso en la búsqueda de geolocalizaciones), los vectores de correspondencia entre los inliers buenos deberían tener un módulo muy similar y respetar una regla de paralaje, básicamente que el ángulo determinado sea muy similar para cada par.

Pero estas reglas de módulo y argumento no pueden ser tajantes, sino que debido a dichos cambios en la traslación, deben dejarse unas ciertas **tolerancia**, que en el caso del módulo será de un 10 % y en el caso del ángulo 5 grados. Precisamente el objetivo que se busca con el *montage* es **suavizar los ángulos de los *inliers*** a partir de la deducción de que a mayor hipotenusa en un triángulo rectángulo, menor ángulo:

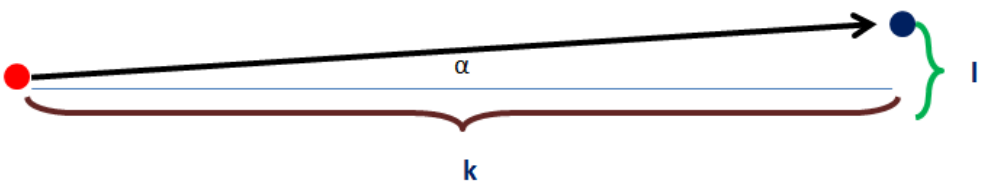


Figura 6.6: Suavizado de ángulos

```
for i=2:index(n) % From the second because the first pair of keypoints is the reference

    if(metodo==1 || metodo==2) % SURF or HARRIS

        k=(effective_points_distorted.Location(i,1)+640)-effective_points_original.Location(i,1); % X difference
        l=effective_points_distorted.Location(i,2)-effective_points_original.Location(i,2); % Y difference

    else

        k=(effective_points_distorted(i,1)+640)-effective_points_original(i,1); % X difference
        l=effective_points_distorted(i,2)-effective_points_original(i,2); % Y difference

    end
modulos(i)=sqrt(k^2+l^2);

% The conditions for validating a combination (SIFT+affine, etc...) are every pair of inliers
% must maintain both angle and module values very similar to the angle and module of reference.
% If one pair fails, the transformation could be aberrant and the combination is descarded.

if (modulos(i)>=0.9*modulos(1) && modulos(i)<= 1.1*modulos(1)) % +/- 10 % of tolerance in module

    % If the module comparison is ok, the angle comparison is carried out

    if(k>=640 && l>=0)

        angle=(360/(2*pi))*asin(l/modulos(i));

    elseif (k<640 && l>=0)

        angle=180-(360/(2*pi))*asin(l/modulos(i));

    elseif (k>=640 && l<0)

        angle=-(360/(2*pi))*acos(k/modulos(i));

    else (k<640 && l<0)

        angle=180+(360/(2*pi))*acos(-k/modulos(i));

    end

    angulos(i)=angle;

    if((abs(angulos(i))>=(abs(angulos(1))-5)) && (abs(angulos(i))<=abs(angulos(1))+5)) % +/- 5 degrees in angle

        correcto(n)=correcto(n)+1;

    else
        break; % If the angle comparison is not ok, the combination is discarded
    end
    else
        break; % If the module comparison is not ok, the combination is discarded
    end
end
```

Código de interés 6.1: Filtro paralelo

Como se puede observar en el código 6.1, si el módulo del vector que une el par de *inliers* i es menor que el 90 % del módulo de referencia, o superior al 110 % de este mismo se descarta (porque como se comentó anteriormente, ante pequeños cambios de traslación, el vector debe tener una distancia muy parecida para todos los pares de keypoints, ya que la ventana del edificio puede haberse movido respecto a la primera sección, pero esta misma no se ha estrechado, alargado etc por lo que en términos relativos esa distancia se debe respetar).

En caso de que el módulo comprenda esas reglas, basándose en la misma afirmación de que relativamente la región circundante de los *keypoints* es la misma, se debe cumplir que el valor absoluto del ángulo para el par i esté entre el valor absoluto del ángulo de referencia menos cinco grados y ese mismo ángulo más cinco grados. Empíricamente se ha comprobado que este compendio de tolerancias (módulo y ángulo) ayudado por el concepto de suavizamiento con el *montage* es más que suficiente para descartar aquellos resultados del filtro MSAC que aún presentan malas correspondencias (por ejemplo un pixel azul del cielo con un píxel azul de una ventana con regiones circundantes muy parecidas, puntos en la carretera, etc), por ello es conveniente que la elección de la geolocalización presente las siguientes características:

- **Poco o nulo cambio en la iluminación.**
- **Poco o nulo movimiento de traslación.**
- **Poca vegetación.**

- **Poco cielo (ajuste de la inclinación vertical y FoV).**
- **Poca carretera.**
- **Bajo número de transeuntes y vehículos.**

```
if (correcto(n)==index(n)-1) % correct(n) will store the number of valid inliers after the parallel filter

   switch metodo

      case 1

         if(exist('permiso_HARRIS.mat','file')~=2)

             permiso_HARRIS=zeros(1,3); % The first time of the loop (n=1)

         else

             load('permiso_HARRIS.mat'); % n=2 or n=3

         end

            permiso_HARRIS(n)=1; % n can be 1 (similarity), 2 (affine) or 3 (projective)
            save('permiso_HARRIS.mat','permiso_HARRIS');
```

Código de interés 6.2: Activación de la técnica si cumple los requisitos

Un contador denominado correcto(n) (siendo n la transformación geométrica usada en el filtro MSAC) aumentará su valor si se cumple para el par i **tanto la regla del módulo como la del ángulo**. En caso de que este contador valga el número de pares de *inliers* para la combinación analizada menos uno, se dará por buena la combinación.

Obsérvese que el número de *inliers* para la combinación está almacenada en la estructura index.Harris (o SURF o SIFT según corresponda), el cual corresponde a un vector fila de tres elementos, siendo el primero para la transformación geométrica similarity, el segundo para la transformación affine y el tercero para la transformación projective.

Al darse por buena la combinación, se activará a uno el correspondiente elemento (*1 = similarity, 2 = affine, 3 = projective*) del vector permiso.

En conclusión, con esta cuarta etapa del filtro paralelo se analiza cada vez que se la llama una técnica con sus tres tipos de transformaciones geométricas, logrando descartar (casi) siempre aquellas combinaciones que presentan correspondencias erróneas.

5. Si se observa la función *Alignment_CGH.m* :

```
case 1

    if(bandera.HARRIS~=0) % bandera.Harris won't be zero if unless there are 2 keypoints
                          % that is, at least similarity anaylis

        Check_Alignment_CGH(indice, bandera, metodo, inliers); % Call parallel filter

        % permiso is a variable that indicates if a combination (method+anaylisis) is valid
        % in terms of good inliers

        if(exist('permiso_HARRIS.mat','file')==2)

            load('permiso_HARRIS.mat')
            permiso.HARRIS=permiso_HARRIS;
            delete('permiso_HARRIS.mat');
            analisis_global=0; % With analisis_global = 0, the function Show_checked_alignment fill up
                               % the calidad matrix

            if(preciso==1) % N.B. preciso==1 -> We enter in the function Fragmentos_alineados (sections)

               Show_checked_alignment(permiso, metodo, matchedfeatures,...
               inliers, indice, transformations, imagen1gray,imagen2gray,...
               Ia,Ib, Fragmentos_alineados, jj, analisis_global)

            elseif(preciso==0) % N.B. preciso==1 -> We enter in the function Panoramas_alineados (Panoramas)

                Show_checked_alignment(permiso, metodo, matchedfeatures,...
                inliers, indice, transformations, imagen1gray,imagen2gray,...
                Ia,Ib, Panoramas_alineados, jj, analisis_global)

            end

        end

    end
```

Código de interés 6.3: Muestra de resultados intermedios de correspondencia

Como se comentó en la etapa anterior, cada vez que se llama a la función *Check_Alignment_CGH.m* , esta se hace para una única técnica (Harris, SURF o SIFT) y dentro se itera con los tres tipos de transformaciones. En caso de que al menos una de las tres **combinaciones** sea válida (todos y cada uno de los pares de *inliers* se dan por buenos) al menos una de las posiciones del vector permiso_(técnica) será 1 y se procederá a la elaboración de la matriz de calidad con la función *Show_checked_alignment.m.*

```
switch metodo

    case 1

        ok=permiso.HARRIS;

        for i=1:3
            if (ok(i)==1)
                switch i
                    case 1
                        calidad(metodo,i)=indice.HARRIS(i);
                    case 2
                        calidad(metodo,i)=indice.HARRIS(i);
                    case 3
                        calidad(metodo,i)=indice.HARRIS(i);
                end
            end
        end
```

Figura 6.7: Proceso de relleno de la matriz calidad para la técnica Harris

Dentro esta función, se cargará el vector permiso pertinente (supóngase que por ejemplo la función está con la técnica de Harris). Mediante un *for* se comprueba si dicho valor es 1 (es decir, la combinación fue válida). Si es 1, el elemento de la matriz calidad recibirá el número de inliers de dicha combinación. En caso contrario, será igual a 0. Dicho proceso se repetirá mediante una iteracción de las funciones *Alignment_CGH.m* → *Check_alignment_CGH* → *Alignment_CGH.m* → *Show_checked_alignment.m* para cada técnica de extracción.

El resultado final es una matriz de calidad de 3 x 3 elementos que presenta la siguiente forma:

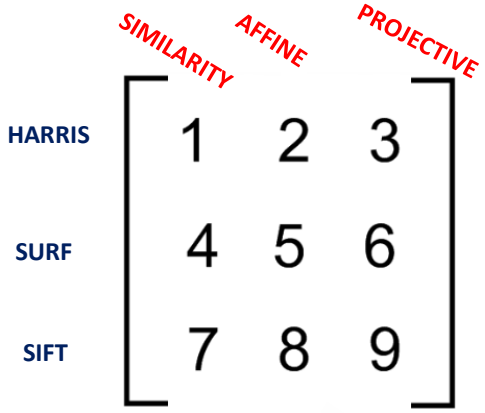


Figura 6.8: Matriz de calidad del proceso de obtención de *inliers*

Por tanto, realmente esta matriz es una **matriz que mide la calidad del proceso**, puesto que la repetibilidad (o mayor número de pares entre dos imágenes) es un parámetro de calidad.

Tras la elaboración de la matriz de calidad, se elegirán los **tres máximos de la matriz, por tanto, cada máximo representará una combinación**, siendo éstas las tres recomendaciones que se mostrarán al usuario para la elección de la futura transformación geométrica de alineamiento, que se verán a continuación.

Antes de pasar al siguiente apartado que trata acerca del alineamiento y las transformaciones geométricas, es conveniente decir que **este filtro paralelo no siempre es exacto**, ya **que a veces la tolerancia del ángulo es excesiva y se permite demasiada diferencia**, lo que genera una mala correspondencia. Por otra parte, las tolerancias de las reglas de módulo y ángulo, y el suavizado del ángulo con la técnica *montage* son totalmente **empíricos**, creados durante la creación de este TFG, por tanto, son conceptos expuestos a ser mejorados en un futuro. Sin embargo, la **gran ventaja** que presenta es que como las reglas de módulo y argumento siempre están basadas en la hipótesis de que el primer par de *keypoints* sea un par con garantías, pero si precisamente fuera uno de los muy pocos que son defectuosos, el filtro descartaría el método de inmediato **ahorrando computacionalmente** mucho tiempo en vez de analizar todos los pares.

A continuación se muestran dos ejemplos del resultado de los *inliers* tras aplicar el filtro MSAC y el filtro paralelo. El primer ejemplo 6.9 muestra un resultado excelente (como casi todos los casos) mientras que el segundo caso 6.10 presenta malas correspondencias:

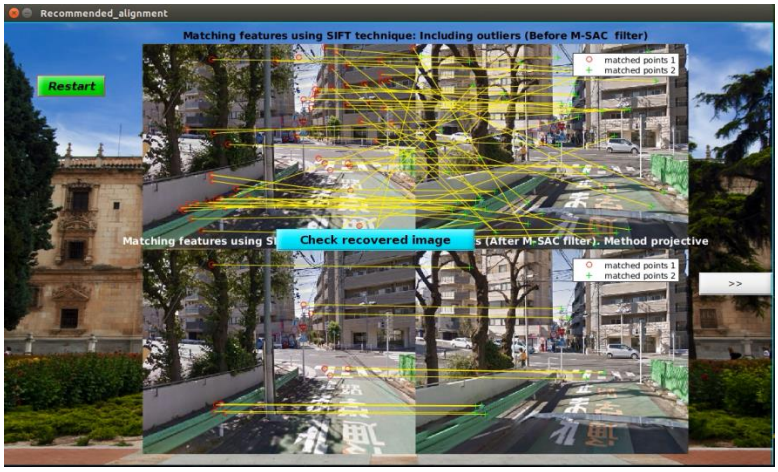


Figura 6.9: Buen resultado tras filtro RANSAC+paralelo

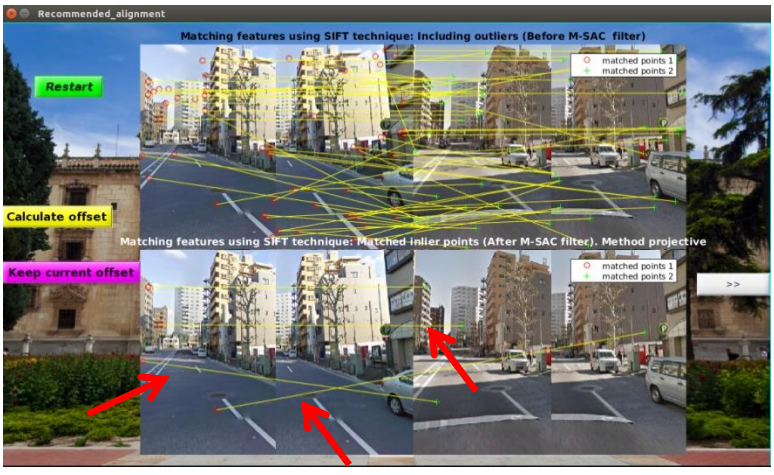


Figura 6.10: Mal resultado tras filtro RANSAC+paralelo

Como la interfaz de *Recommended_alignment.m* se muestra dos veces (en el primer paso de la realimentación para el alineamiento del offset individual y en el segundo paso para el alineamiento con transformación homogénea), se debe ceder en ambas interfaces la posibilidad de que el usuario esté de acuerdo o no con el resultado obtenido tras ambos filtros. En el siguiente apartado se mostrará estas formas de proceder y sus consecuencias, así como el alineamiento de las imágenes.

6.5. Alineamiento:

Tras la etapa de correspondencia o *matching*, computacionalmente se tienen los recursos suficientes para proceder con el alineamiento. Alinear dos imágenes representa una transformación de la posición, rotación etc de la sección a alinear (en este caso la sección del segundo panorama) con respecto a la sección de referencia (en este caso la sección del primer panorama) con la pretensión de hacer coincidir los *keypoints* validados como buenos *inliers* en las etapas anteriores.

Como se comentó en apartados anteriores, el concepto de alineamiento en este TFG está basado en una realimentación:

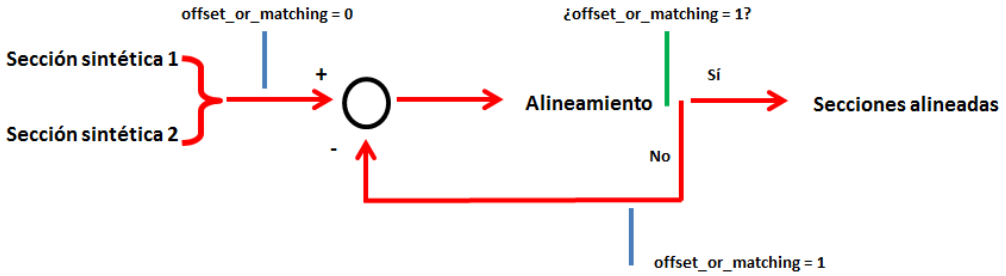


Figura 6.11: Alineamiento basado en realimentación

La explicación de este proceso de realimentación es muy simple. Inicialmente el sistema tiene como input un **par de secciones de imágenes sintéticas**, la sección del panorama 1 que actuará como referencia y la sección del panorama 2 que presenta un offset global calculado en la interfaz *Initial_information.m.*

Para el cálculo de este offset global es necesario obtener la columna inicial del panorama 1, es decir, su *yaw angle* = 0 º a partir del cual comienzan a sintetizarse las imágenes. Esto se consigue mediante el código de interés 6.4:

```
% Pablo Fernández Alcantarilla
% at_gsv2viewsynth_init_col.m
% Function that returns the initial column in the panoramas that corresponds
% to a yaw angle equals to 0
% Date: 04-07-2017
%**************************************************************************
function [init_col] = at_gsv2viewsynth_init_col(qR, estructura1,depthMap1)

% Get the params from gsv struct
oimw = estructura1.oimw;
oimh = estructura1.oimh;
FoV = estructura1.FoV;

% Compute rotation matrix
R = qR*vl_rodr((estructura1.Heading_angle/180)*pi*[0; 1; 0]);
C = estructura1.C;
T = -R*[C(2); 0; -C(1)];

img = estructura1.panorama;
scale = size(img,2)/512;

% Convert the labels format
plabel = convertLabels(depthMap1);

% Convert image points to normalized unit vectors
K = [oimw/2*FoV 0 oimw/2; 0 oimh/2*FoV oimh/2; 0 0 1];
[gx, gy] = meshgrid(1:oimw,1:oimh);
gx = gx'; gy = gy';
X = [gx(:)'; gy(:)'; ones(1,oimw*oimh)];
Y = at_l2normalize_col(K\X);

psz1 = size(plabel,1);
psz2 = size(plabel,2);

% Rotate the vectors and transform them to spherical coordinates
Z = R'*Y;
Z = at_l2normalize_col(Z);
th = atan2(Z(1,:),Z(3,:));
phi = asin(Z(2,:));

% Apply equirectangular projection
[u,v] = at_sph2rct(th,phi,psz1,psz2);
```

```
x = scale*u;
y = scale*v;

% Take care with the horizontal and vertical coordinates
x(floor(x)==0) = 1;
y(floor(y)==0) = 1;

pts = [];
pts.x = reshape(x,oimw,oimh)';
pts.y = reshape(y,oimw,oimh)';
init_col = min(min(pts.x));

end

%*************************************************************************
% This function convers the format of the planes and labels and adjusts the
% coordinates between the new camera and GSV planes
function [pN, pD] = convertPlanes(depthMap1)

% Important we need to remove the first plane since it is not valid!!
CT = [-1  0  0 0; 0  0  1 0; 0 -1  0 0; 0 0 0 1];
nr_planes = length(depthMap1.planes);
pN = zeros(3, nr_planes-1);
pD = zeros(1, nr_planes-1);
 pD = zeros(1, nr_planes-1);
for i=2:nr_planes
    n = depthMap1.planes(i).n;
    v = [n depthMap1.planes(i).d];
    nt = CT*v';
    pN(:,i-1) = nt(1:3);
    pD(1,i-1) = nt(4);
end

 end

 %*************************************************************************
 % This function convers the labels into the required format by 24/7
 function [plabel] = convertLabels(depthMap1)

 nr_cols = depthMap1.labelsdim(1);
 nr_rows = depthMap1.labelsdim(2);
 plabel = zeros(nr_cols,nr_rows);

for h=0:nr_rows-1
    for w=0:nr_cols-1
        plabel(w+1,h+1) = depthMap1.labels(h*nr_cols+w+1);
    end
 end

 end
```

Código de interés 6.4: Cálculo de la columna inicial para la síntesis de imágenes en el panorama 1

El usuario deberá dibujar con una polilínea dentro del segundo panorama qué columna contiene la misma información que la columna calculada del primer panorama. Este procedimiento se explicará en mayor detalle en el manual de usuario del Anexo II *Manual de usuario.*

Un ejemplo es el siguiente:

Figura 6.12: Ejemplo de alineamiento de columnas iniciales

Tal y como se aprecia en la Figura 6.12, la columna roja de la azul están muy distanciadas en la horizontal, pero ambas corresponden a la misma información, por lo que las secciones comienzan a sintetizarse a partir de dicho punto y a alinearse (a priori) en lo que al *heading angle* se refiere. **Cuanto mejor seleccionadas estén las URLs en la base de datos**, mayor tendencia habrá a que la información de ambos panoramas esté situada en casi idénticas columnas, como sucede con el siguiente caso:

El cálculo del offset global es ciertamente sencillo. Se encuentra dentro del botón "Continuar" de la interfaz *Initial_information.m* (que aparece cuando se ha seleccionado la polilínea azul del segundo panorama), y el código principal es el siguiente:

```
global polyline; % With this polylne we must select the corresponding initial column for the
                 % second panorama
polyline = getline(handles.axes2);
X=[polyline(1,1) polyline(2,1)];
Y=[polyline(1,2) polyline(2,2)];

ImageWidth = length(estructura1.panorama);

% The overall offset for the second panorama is stored now from the previous choice (blue initial column)
estructura2.global_offset=(mean(polyline(:,1))-mean(init_col))/ImageWidth*360;
```

Código de interés 6.5: Antioffset global

Siendo *polyline* la polilínea del panorama 2 (columna azul) e *ImageWidth* el ancho del panorama (3584, que realmente se corresponde con el número de columnas). De esta forma, el offset global se calcula a partir del valor de la columna central de la polilínea en su proyección en el eje de abcisas (teniéndose en cuenta que presenta una anchura normalizada de 2.5) menos el valor de la columna central de la columna inicial (igualmente presenta una anchura normalizada de 3, que representa realmente ocho columnas), eso sobre el ancho del panorama y multiplicándolo por 360, o lo que es lo mismo, **expresar la equivalencia de ese desfase horizontal en su relativo desfase en el *heading angle* al sintetizar,** tal y como se demuestra en la ecuación 6.7:

$$Offset\ global\ en\ grados = \frac{Columna\ central\ panorama\ 2 - Columna\ central\ panorama\ 1}{Ancho\ panorama} \cdot 360 \quad (6.7)$$

Evidentemente, tal y como se puede observar en la función *createSynthView_panorama2.m*, para la corrección de este offset se debe aplicar un antioffset, es decir, si el offset obtenido es positivo significa que la información del panorama 2 con respecto a la del panorama 1 está más a la derecha, como es el caso que se vio en la figura 6.12, y deducción análoga para el caso contrario. Ello se muestra con las siguientes condiciones:

```
if(estructura2.global_offset>0)
    angle = estructura2.yawRange(jj) - estructura2.global_offset;
elseif(estructura2.global_offset<0)
    angle = estructura2.yawRange(jj) + estructura2.global_offset;
elseif(estructura2.global_offset==0)
    angle = estructura2.yawRange(jj);
```

Si se toma como referencia el ejemplo de la figura 6.12 donde las columnas iniciales estaban sumamente desplazadas, tras aplicar la corrección del offset global y posterior síntesis, los resultados son muy satisfactorios, tal y como se muestra en la figura 6.13:

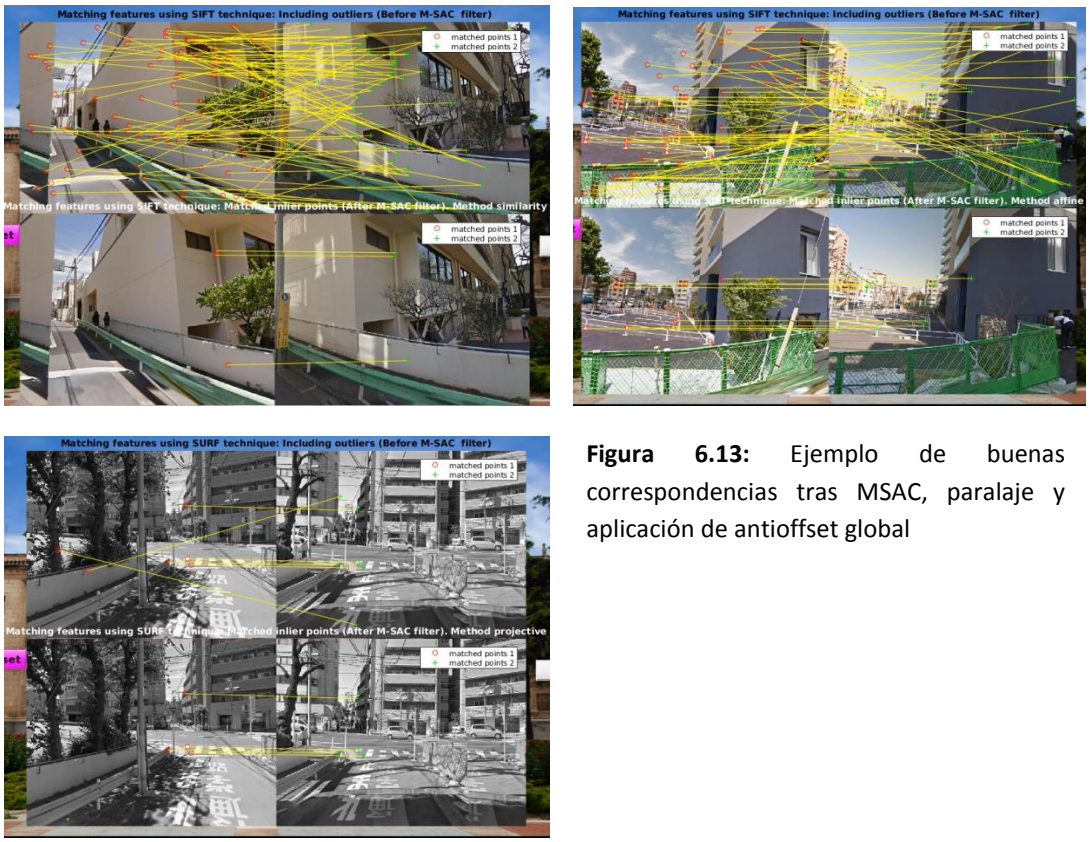


Figura 6.13: Ejemplo de buenas correspondencias tras MSAC, paralaje y aplicación de antioffset global

Se puede comprobar que la información de ambas secciones es prácticamente la misma. Sin embargo, ese "prácticamente" es una lacra para la red neuronal que se tratará en el siguiente capítulo: El objetivo final de este TFG es que una CNN sea capaz de localizar y etiquetar cambios de forma automática sin la presencia de un ser humano para el etiquetado de cambios estructurales entre imágenes. **Cuando menos alineadas estén las imágenes, más ruido se introducirá a la CNN y más sufrirá para etiquetar.**

6.5.1. Primera fase de la realimentación:

En esta primera fase se obtiene el alineamiento mediante corrección de offset individual, en caso de que la síntesis tras el offset global no hubiese sido suficiente. En primer lugar, con la variable *offset_or_matching* = 0 se llamará a la función *Recommended_alignment.m*. En esta primera fase se procederá a un alineamiento individual de cada par de imágenes, en caso de que el usuario lo desee.

La apariencia de la interfaz *Recommended_alignment.m* será la de la Figura 6.10, donde se pueden observar dos botones principales para el cálculo del offset:

- ***Keep current offset*** (mantiene el offset individual presente entre imágenes, por defecto igual a cero).
- ***Calculate offset*** (cálculo del offset individual en caso de que se considere que hay suficiente desplazamiento para invocar este botón).

Tanto en la primera fase como en la segunda de la realimentación existe a disposición del usuario un botón *Restart* en caso de error en la aplicación o de no estar de acuerdo con los resultados obtenidos (por ejemplo que haya habido uno o dos pares de imágenes sin sintetizar) y unas flechas para desplazarse a lo largo de las mejores combinaciones. Estos botones serán mejor explicados en el **Anexo II *Manual de* usuario**.

El cálculo de este offset es muy trivial pero a la vez está construido de una forma ingeniosa: Está basado en hallar la **media aritmética de la diferencia en el eje X de los puntos de la sección 2 con respecto a los puntos de la sección 1** (en este caso sin añadirle 640 unidades a los puntos de la segunda sección, puesto que se pretende ver cuantos grados en el *heading angle* para que ambas vistas sintéticas contaran con la misma información).

El código principal para el cálculo en caso de presionar *Calculate offset* se detalla a continuación:

```
ImageWidth = length(estructura1.panorama); % Panorama width = 3584 (low resolution)

puntos_utiles = length(inliers_metodo.original); % Number of pairs

numerando = zeros(puntos_utiles,1);
numerando_final = 0;

try % HARRIS and SURF

    % It is going to be calculated the X offset for each pair. That is the difference of X position, divided by
    % the image width and multiplied by 360, to translate the result into degrees

    for i=1:puntos_utiles

        numerando(i) = (inliers_metodo.distorted.Location(i,1) - inliers_metodo.original.Location(i,1))/ImageWidth*360;
        numerando_final=numerando_final+numerando(i); % Addition

    end

catch % SIFT

    for i=1:puntos_utiles

        numerando(i) = (inliers_metodo.distorted(i,1) - inliers_metodo.original(i,1))/ImageWidth*360;
        numerando_final=numerando_final+numerando(i);

    end

end

offsetangle = numerando_final/puntos_utiles; % Arithmetic average

offset_medio(jj)=offsetangle; % jj can be 1, 2 or 3, depending on the section
save('offset_medio.mat','offset_medio');

estructura2.offset = offset_medio; % The synthetic views will be created with an additional horizontal offset
```

Código de interés 6.6: Cálculo de offset individual

En primer lugar se define el ancho del panorama y el número de puntos útiles que es realmente el número de *inliers* resultantes tras aplicar el filtro MSAC y el filtro paralelo. La estructura *"try and catch"* simplemente se usa porque las estructuras de las técnicas SURF y Harris son diferentes a la de SIFT, pero el principio es idéntico: Hallar en una variable llamada numerando_final el sumatorio de las diferencias angular de todos los pares de *keypoints*. Para ello será muy importante navegar por las tres recomendaciones ofrecidas al usuario y ver cual contiene las mejores correspondencias de todas (puesto que no por ser la primera la que tiene un mayor número de pares quiere decir que en su totalidad la combinación sea más fiable). Finalmente, para hallar la media aritmética, simplemente se divide dicho valor entre el número de pares. En cualquier caso, sea la combinación que sea, la operación será la siguiente (6.8):

$$offset(jj) = \frac{\left(\sum_i^n (inlier\ segunda\ sección\ (i,1) - inlier\ primera\ sección)\right) \cdot \frac{360}{Ancho\ panorama}}{n} \qquad (6.8)$$

Siendo jj el par de secciones que está siendo analizado (primero, segundo o tercero), i el número del par y n el número de pares. Al igual que antes, el ancho del panorama se corresponde con 3584, considerando baja resolución siempre.

Un ejemplo de alineamiento individual se muestra a continuación. Intuitivamente se puede percibir como antes de la primera fase de la realimentación, la segunda sección está desfasa hacia la derecha respecto de la primera sección. Por tanto, el antioffset angular se debe hacer hacia la izquierda, quedando un resultado bastante decente en la segunda figura:

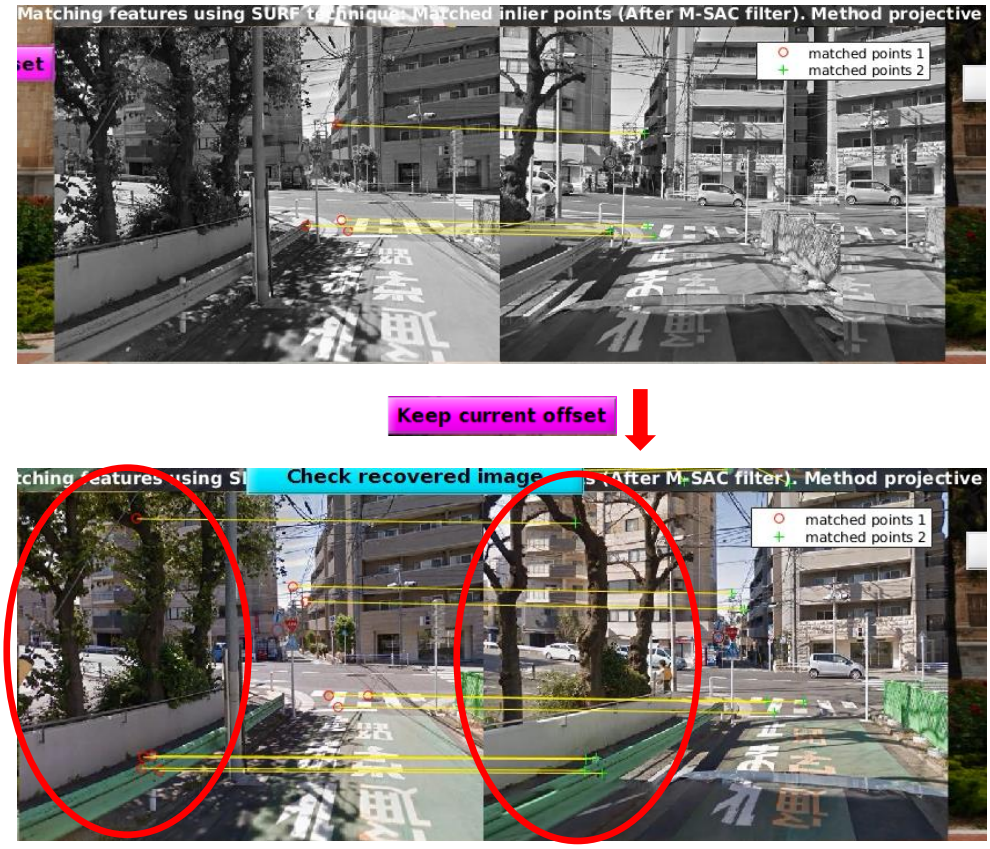


Figura 6.14: Ejemplo de buen alineamiento con correción de offset individual

La nueva síntesis de imágenes de nuevo es producida con la función *createSynthViews_panorama_2.m,* con la peculiaridad de que al invocar de nuevo la función, la variable flat = 1, por lo cual se ejecuta la segunda parte de la función, que además de sintetizar en base al offset global, se le suma el offset individual de cada sección, por lo cual el antioffset debe contrarrestar a ambos:

```
 % If flat == 1 (after the feedback), the sum of the global offset and each individual offset is considered
else
   for ii=1:length(estructura2.Pitch_angle)
   for jj=1:length(estructura2.yawRange)

     if((estructura2.offset(jj)+estructura2.global_offset)>0)

         angle = estructura2.yawRange(jj) - (estructura2.offset(jj)+estructura2.global_offset);

     elseif((estructura2.offset(jj)+estructura2.global_offset)<0)

         angle = estructura2.yawRange(jj) + (estructura2.offset(jj)+estructura2.global_offset);

     elseif((estructura2.offset(jj)+estructura2.global_offset)==0)

         angle = estructura2.yawRange(jj);

     end
```

6.5.2. Segunda fase de la realimentación:

El segundo paso de la realimentación, **tras alinear los offset individuales de cada una** (en teoría deben estar en la horizontal muy alineadas) es el **alineamiento mediante la aplicación de la transformación homogénea** sobre la segunda sección para alinear los *keypoints* de esta con los de la primera. Esta transformación homogénea tendrá más referencias cuantos más *keypoints* tenga, por eso **en general la mejor transformación suele corresponderse con la primera combinación recomendada.**

Tras finalizar el proceso de alineamiento individual (ya sea para una, dos o tres secciones), la variable offset_or_matching es igual a 1. A continuación, se itera de nuevo todo el proceso de la función

Alignment_CGH.m , incluido la extracción de características, estimación el filtro MSAC, filtro en paralelo, etc… y al final se volverá a llamar a la función *Recommended_alignment.m* pero ahora la interfaz presenta un aspecto distinto:

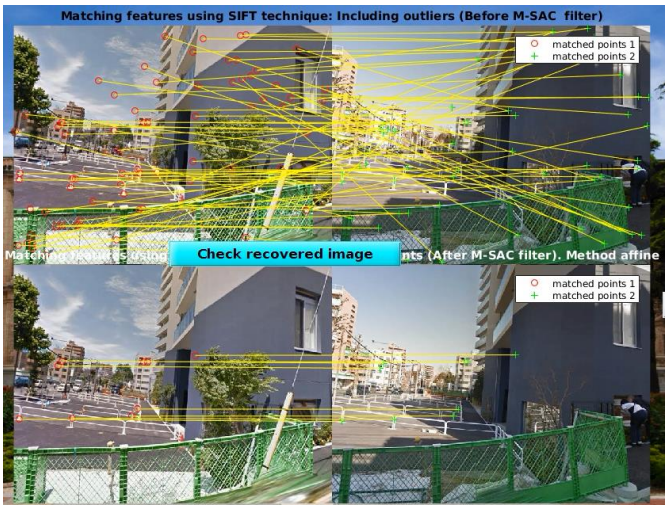


Figura 6.15: Aspecto de *Recommended_alignment.m* en la segunda fase de la realimentación

En este caso, en vez de presentar un botón de alineamiento, presenta un botón llamado ***Check recovered image.*** Este botón muestra en un tercer eje centrado en la interfaz la segunda sección tras aplicarle la transformación obtenida con la función *estimateGeometricTransform*, que se basaba en el filtro MSAC.

Antes de ver resultados, es necesario definir con un mayor grado de detalle esta función y los tipos de transformaciones que se pueden introducir:

La **función *estimateGeometricTransform*** pertenece a la toolbox *Computer vision* de MatLab.

Su sintaxis completa es:

$$[tform, inlierpoints1, inlierspoints2] = estimateGeometricTransform(matchedPoints1, matchedPoints2, transformType)$$

Siendo matchedpoints1 los keypoints (en bruto) tras la extracción de características de las primeras etapas relacionados con los matchedpoints2 de la segunda sección y transformType el tipo de transformación para la estimación.

Como se comentó anteriormente, esta función excluye las malas correspondencias (outliers) de los matchedpoints de la entrada usando el algoritmo MSAC (variante del RANSAC). Hay que tener en cuenta que el resultado de esta función podría no ser idéntico si se aplicara el algoritmo RANSAC dada la característica aleatoria del filtro MSAC frente al RANSAC.

6.5.2.1. Tipos de transformaciones [15]:

Conceptualmente, una **transformación** es una **herramienta matemática que desplaza o convierte los datos dentro de un sistema de coordenadas.**

Las funciones de transformación se basan en la **comparación de las coordenadas de puntos de origen y destino** (en este caso los *inliers* de ambas imágenes). Estas transformaciones tendrán la misión hacer coincidir la misma ubicación de los *inliers* destino con los de origen.

El tipo de transformación que admite la función *estimateGeometricTransform*, como se citó en capítulos anteriores, puede adoptar alguna de las tres técnicas que se describen a continuación:

❖ **A. Transformación de similaridad (Similarity):**

La transformación de similaridad **escala, gira y traduce los datos a alinear** mediante una ecuación cuyos parámetros se generan a partir de las coordenadas. Con esta transformación, el mínimo de pares de inliers para el cálculo de la matriz será dos. Sin embargo, no puede aplicar un escalado o un sesgo diferencial a los ejes de forma independiente, pero es muy útil cuando la sección a alinear únicamente requiere de una rotación o traslación (bien en la vertical, bien en la horizontal) conteniendo ambas secciones prácticamente la misma información.

La función de la transformación de similitud es:

$$x' = A \cdot x + B \cdot y + C \cdot y' = -B \cdot x + A \cdot y + F \tag{6.9}$$

Siendo:

$$A = s \cdot \cos(t)$$

$$A = s \cdot \sin(t)$$

$$C = Translación\ en\ la\ dirección\ de\ X$$

$$F = Translación\ en\ la\ dirección\ de\ Y$$

$$s = Escalado\ (igual\ en\ ambas\ direcciones)$$

$$t = Ángulo\ de\ rotación\ (medido\ antihorario\ desde\ el\ eje\ X)$$

Aunque la transformación de similitud **necesita un mínimo de dos puntos** para poder ejecutar la función *estimateGeometricTransform* sin que de error, se necesitan un mínimo de tres pares de puntos para que se produzca un error cuadrático medio (RMS).

Ejemplo de transformación de similaridad:

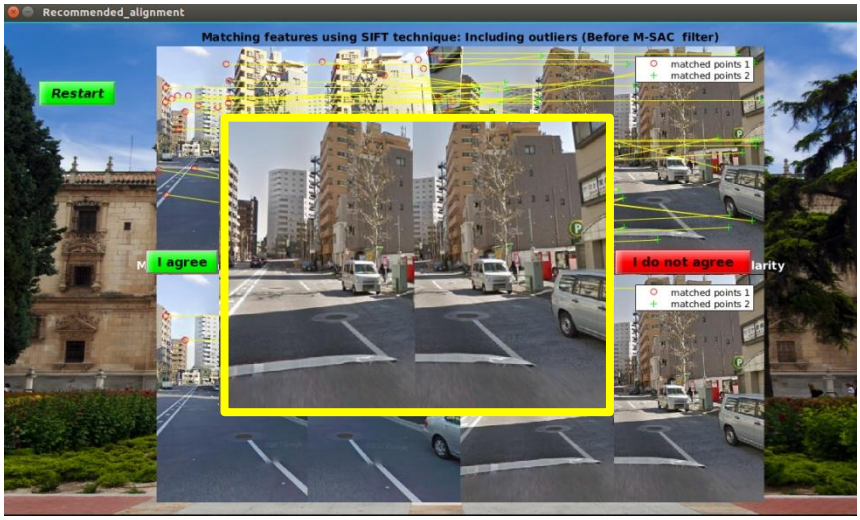


Figura 6.16: Imagen obtenida tras la transformación *similarity* (ejes centrales, dentro del cuadro amarillo)

Como se puede apreciar, la segunda sección se ha transformado, rotando muy ligeramente y desplazando hacia la derecha, dejando entre ver una pequeña franja negra (ya que no puede sintetizar de ahí información). Es obvio que **cuanto menor sea el tamaño de las franjas negras dentro de la sección transformada querrá decir que prácticamente ambas secciones contenían la misma información, es decir, el alineamiento de offset global + offset individual fue muy bueno**. Por el contrario, si tras el alineamiento individual de la primera fase de realimentación aún estaban moderadamente desalineadas, las franjas negras serán mucho mayores.

❖ **B. Transformación afín (Affine):**

La transformación afín es una **transformación lineal** que **preserva el paralelismo de las rectas tras la transformación pero no así de los ángulos**. Además de **escalar y poder girar los datos** (lo único que podía hacer la transformación de similaridad), es capaz de **sesgar** (torcer por un lado) e incluso hacer una **reflexión del plano**. Son requeridos un **mínimo de tres pares de puntos de control** para que se pueda aplicar dicha técnica.

La función de la transformada afín es la siguiente:

$$x' = A \cdot x + B \cdot y + C \cdot y' = D \cdot x + E \cdot y + F \tag{6.10}$$

Siendo x e y las coordenadas de la segunda sección, x' e y' son las coordenadas transformadas (que en teoría deben corresponderse con las coordenadas x e y de los keypoints correspondientes en la primera sección). Los términos A, B, C, D, E y F se determinan comparando la ubicación de la posición de los keypoints de cada par. **Esta opción es la más recomendada para la mayoría de transformaciones, por encima de la transformación de similaridad** (dos puntos mínimos requeridos) **y la transformación proyectiva** (cuatro puntos mínimos requeridos).

Reescrito en forma matricial, las coordenadas del punto transformado P' es igual a la matriz afín Ma por las coordenadas del punto original P.

$$P' = M_A \cdot P \tag{6.11}$$

$$\begin{pmatrix} x' \\ y' \\ 1 \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \tag{6.12}$$

El resumen de subtransformaciones capaz de hacer es el siguiente:

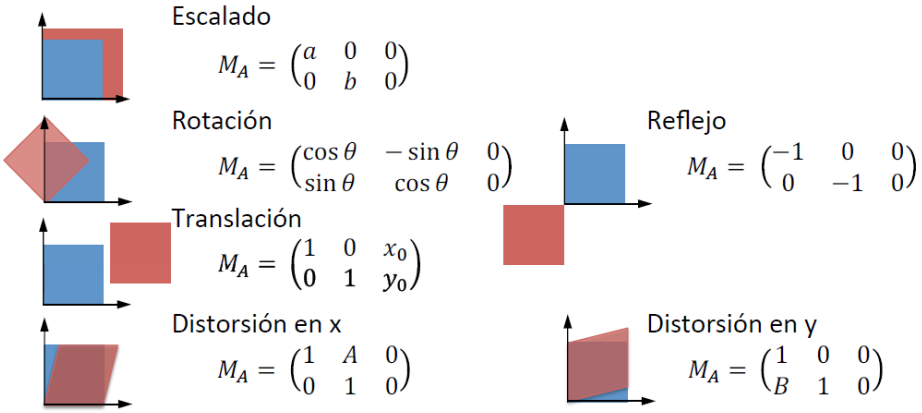


Figura 6.17: Operaciones dentro de la transformación afín

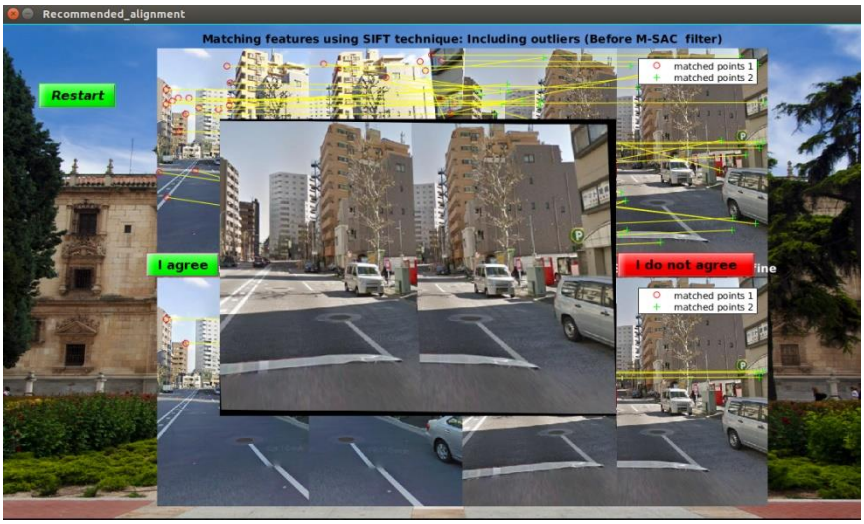


Figura 6.18: Imagen obtenida tras la transformación *affine*

❖ **C. Transformación proyectiva (Projective):**

Esta transformación es más general que la afín y **no preserva generalmente el paralelismo entre rectas**. La transformación proyectiva se usa generalmente si la imagen de entrada a alinear se tomó a partir de una fotografía aérea. La precisión de la transformación dependerá del terreno de la superficie fotografiada, del ángulo de la cámara y el suelo y también de la elevación desde la que se tomó la fotografía. Este tipo de transformación necesita de un mínimo de cuatro pares de control para llevarse a cabo.

Su ecuación es la siguiente:

$$x' = \frac{A \cdot x + B \cdot y + C}{G \cdot x + H \cdot y + 1} \cdot y' = \frac{D \cdot x + E \cdot y + F}{G \cdot x + H \cdot y + 1} \tag{6.13}$$

Los términos A, B, C, D, E, F, G y H se determinan comparando la ubicación de la posición de los keypoints de cada par. En forma matricial (siendo el elemento de la matriz $M_{33} = 1$, de ahí que se puedan saber las coordenadas a partir de cuatro pares de puntos):

$$P' = M_p * P$$

$$\begin{pmatrix} x' \\ y' \\ w \end{pmatrix} = \begin{pmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \qquad \begin{cases} x' = \dfrac{M_{11}x + M_{12}y + M_{13}}{M_{31}x + M_{32}y + M_{33}} \\ y' = \dfrac{M_{21}x + M_{22}y + M_{23}}{M_{31}x + M_{32}y + M_{33}} \end{cases}$$

Ejemplo de transformación proyectiva:

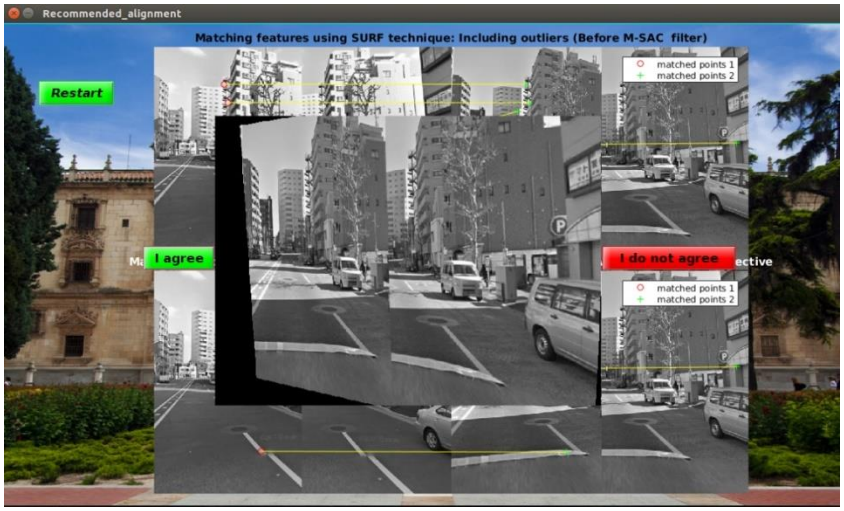


Figura 6.19: Imagen obtenida tras la transformación *projective*

Para terminar las transformaciones, se explica a continuación un caso especial de la transformación proyectiva, esto es, la **homografía**:

Una homografía es un **tipo de transformación proyectiva** que determina una **correspondencia entre dos figuras geométricamente planas**. Se aplica cuando hay una rotación pura (sin desplazamiento en ninguno de los tres ejes) y además la profundidad es nula (es decir, una homografía está pensada para figuras planas). Al igual que la clásica transformación proyectiva, la matriz se puede obtener a partir de cuatro pares de puntos de control como mínimo.

$$\begin{pmatrix} wu' \\ wv' \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X_w \\ Y_W \\ Z_w \\ 1 \end{pmatrix}$$

$$\begin{pmatrix} wu' \\ wv' \\ w \end{pmatrix} = H_{(3x3)} * \begin{pmatrix} u \\ v \\ 1 \end{pmatrix}$$

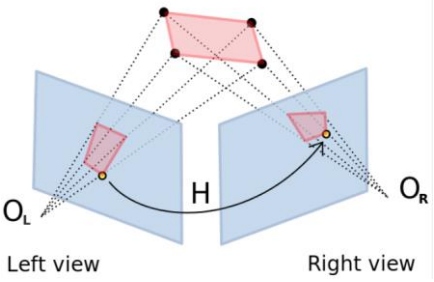


Figura 6.20: Homografía

Evidentemente, la matriz *tform* calculada con la función *estimateGeometricTransform* equivale respectivamente a cada una de las matrices características de cada transformación. El usuario deberá decidir con qué transformación se queda en función del ajuste de esta a la sección original. En caso de que el usuario no esté conforme con ninguna de las transformaciones, simplemente deberá clickar en continuar como si le convenciese una para luego dar paso al etiquetado manual.

Finalmente, para terminar con el apartado de alineamiento se debe comentar que una de las **futuras mejoras** es, además del alineamiento con la modificación del heading angle y la posterior transformación homogénea, aplicar la traslación del centro de la cámara en coordenadas UTM [23].

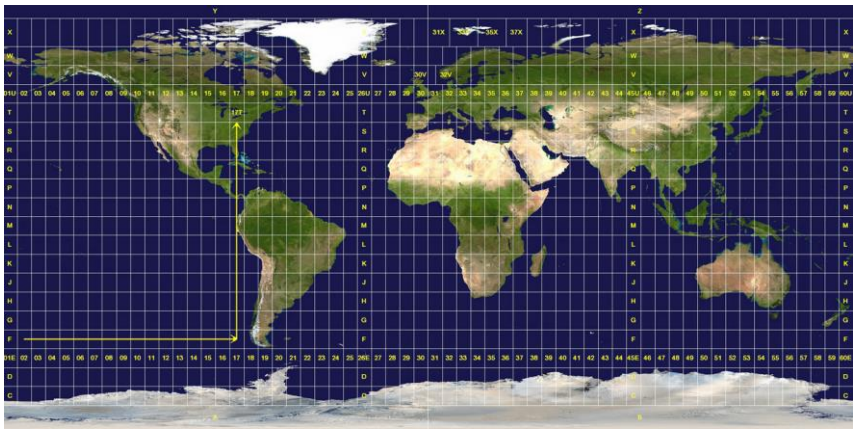


Figura 6.21: Zonas y husos UTM

El sistema de coordenadas universal transversal de Mercator (***Universal Transverse Mercator,*** **UTM**) es un sistema de coordenadas basado en la proyección cartográfica de Mercator (geógrafo y matemático flamenco), construida como la proyección de Mercator, pero en vez de tangente al Ecuador, se hace secante al meridiano.

A diferencia del sistema general de coordenadas geográficas, expresadas en longitud y latitud (por ejemplo en la URL de GSV), las magnitudes en el sistema UTM **se expresan en metros al nivel del mar** (base de la proyección del elipsoide de referencia).

- **Husos UTM:** El eje de abcisas se divide en 60 husos de 6 º de longitud cada uno (360 º en total, lógico al expandir la longitud de la circunferencia terrestre). Cada huso tiene asociado un meridiano central, que es donde se sitúa el origen de coordenadas junto con el Ecuador (referencia horizontal).
- **Bandas UTM:** El eje de ordenadas se divide en 20 bandas de 8 º, que van desde las letras C a la X, excluyendo las letras I y O por su parecidos a los números 1 y 0 respectivamente (al ser un sistema de EEUU, evidentemente tampoco tiene la letra Ñ).
- **Notación:** Cada cuadrícula del mapa UTM se define con el número del huso (eje horizontal) y la letra de la zona (eje vertical). Por ejemplo, la Península Ibérica se encuentra en su totalidad en las combinaciones 29 T, 29 S, 30 T y 30 S.
- **Aplicaciones:** Las aplicaciones web de cartografía (por ejemplo, Google Maps y en su defecto Google Street View, usan actualmente la proyección de Mercator. En concreto emplean la variante que supone que el planeta es una esfera en vez de un geoide como es realmente. Esto se justifica por dos razones:
    - Como toda proyección cilíndrica, en cualquier punto del planeta la dirección N-S siempre es vertical y la E-O siempre horizontal.
    - En segundo lugar, los edificios no se distorsionan en la proyección.

Estas dos anteriores virtudes permiten compensar el defecto de las significativas distorsiones de escala que introduce la proyección, sobre todo en las regiones cerca de los polos.

Por defecto, **en ambas estructuras de los panoramas, la posición de la cámara es [0 0].** Teniendo en cuenta que el usuario debe seleccionar ambas URLs con el máximo cuidado posible, en teoría el desplazamiento en traslación debería ser nulo, pero en muchas ocasiones no es así.

Una posible solución es dotarle al centro de la cámara en la síntesis de las secciones del segundo panorama, en vez del vector cero [0 0], el siguiente vector (6.14):

$$estructura2.C =$$

$$[estructura1.xutm - estructura2.xutm \quad estructura1.yutm - estructura2.ytm] \text{ (6.14)}$$

```
if (flat==0)
R = qR*vl_rodr((estructura2.Heading_angle/180)*pi*[0; 1; 0]); % Initial synthesis
else
R = qR*vl_rodr((estructura1.Heading_angle/180)*pi*[0; 1; 0]); % After feedback process
end

C = estructura2.C;
T = -R*[C(2); 0; -C(1)];
```

Sin embargo, se ha comprobado en varias ocasiones que los **resultados son muy malos** dado que la modificación del centro de la cámara está relacionada con la información del mapa de profundidad, el cual es inexacto. La mejora del mapa de profundidad y consecuentemente el alineamiento posicional es una de las futuras mejoras de este trabajo.

6.6. Etiquetado y elaboración del groundtruth:

Tras proceder con el alineamiento y validar los resultados, se llega a la última interfaz, denomina ***matching.m***, donde se procederá al etiquetado de las diferencias entre estas mismas.

El concepto que se usa para el etiquetado es el del ***groundtruth*** que no es más que una **clasificación verdadera de la imagen según secciones, es decir, una segmentación de esta misma**. Esta clasificación será uno de los tres datos que posteriormente servirán para entrenar la red neuronal convolucional. El código de la aplicación para la elaboración de este etiquetado (principalmente los botones, dentro de esta interfaz, denominados *Add label*, *Remove label*, *Show binary groundtruth*) está basado en el trabajo de Andreas Geiger, que se le ocurrió escribir una herramienta adaptada a lenguaje MatLab para anotar polígonos dentro de las imágenes cuando trabajaba en *3D Traffic Scene Understanding from Movable Platforms (PAMI 2014),* denominada **LibLabel [16].**

Liblabel es una herramienta muy simple (menos de cuatrocientas líneas de código originales) muy fácil de utilizar que consiste en la anotación de las imágenes con polígonos o polilíneas, fácil de configurar y capaz de cnfigurar dichos polígonos en regiones de imágenes, generando mapas de segmentación y mapas de instancias (**en este TFG se usarán mapas de segmentación**). También es posible, cuando se termina un polígono, que el usuario especifique la clase semántica a la que pertenece:

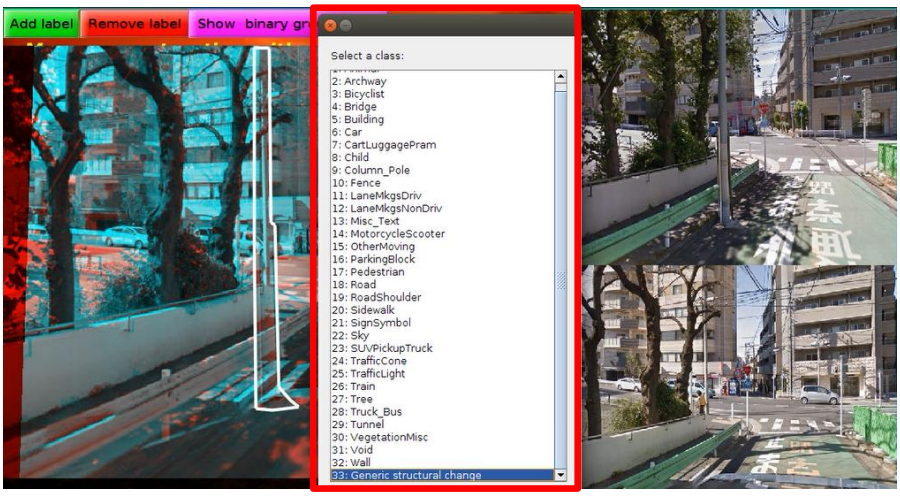


Figura 6.22: Elección de clase de cambio

A toda clase se le asigna un color y también es capaz de eliminar un etiquetado si no está conforme con él.

Un ejemplo de etiquetado complejo es el siguiente:

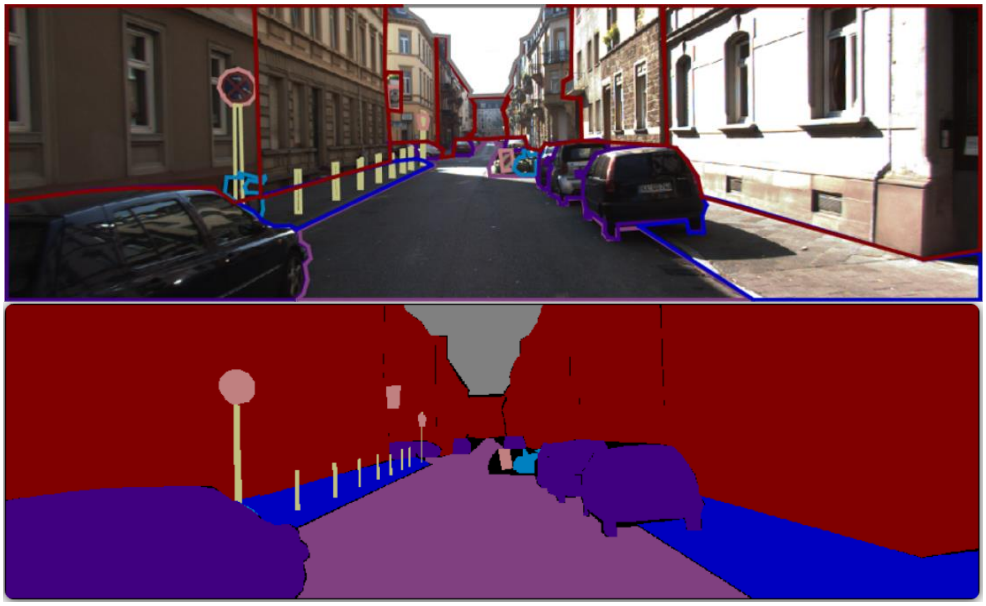


Figura 6.23: Arriba, etiquetado. Abajo, Mapa de segmentación con varias clases

Se ha denominado el ejemplo anterior como complejo no porque sea más o menos compleja la imagen (si bien el etiquetado entre secciones será más difícil cuantos más objetos pequeños hayan, confusión entre la imagen etc) sino porque se ha hecho una **segmentación total**, dotándole de una **clase semántica a cada objeto** (por ejemplo, malva para la carretera, azul para el pavimento, morado más oscuro para los coches, etc). En cambio, para este TFG el etiquetado será mucho más simple porque se etiquetará **todo con la misma clase semántica denominada Cambio Estructural General (*Generic Structural Change*),** situado en la clase 33 de la lista semántica.

Estos cambios estructurales están reflejados en las Tablas 2.1 y 2.2 del capítulo 2 *Herramienta Google Street View*.

El código de las principales acciones es el siguiente:

Añadir una etiqueta (Add label)

```
imshow(I);

polyline = getline(handles.axes1); % The polyline only is accepted in the overlay axis

class = chooseClass(); % The class will always be "General structural change"

if ~isempty(class)

    L{end+1}.class  = class;
    L{end}.polyline = polyline;

    % When we finish the polyline, showandSaveLabels shows the label in the overlay with
    % our chosen color

    showAndSaveLabels(I,L,packages_dir,file_name,file,number_of_files);
```

Código de interés 6.7: Añadir un polígono

El usuario comienza con una polilínea y va clickando hasta que presiona enter o doble click. A continuación le aparece la ventana de elección de clase (en este caso *Generic structural change*). El polígono se actualiza en tiempo real con *showandSaveLabels*.

Borrar una etiqueta (Remove label)

```
imshow(I);

[u,v] = ginput(1);
label_num = getClosestLabel(L,u,v); % Removes the closest label to the zone where
                                    % we have clicked in

if ~isempty(label_num)

        L(label_num)=[];

        if(size(L)==[1 0]) % It is empty

            L=[];

        end

        showAndSaveLabels(I,L,packages_dir,file_name,file,number_of_files);

end
```

Código de interés 6.8: Borrar un polígono

Toma la etiqueta más cercana con getClosestLabel. En caso de no haber ninguna, no puede borrar nada. En caso de detectar una (la más próxima, de ahí la función *getClosestLabel*) es borrada. Con la función *showandSaveLabels* se actualizan los polígonos.

Mostrar el groundtruth binario (Show binary groundtruth)

```
for file=1:length(files)

  file_name = files(file).name;
  Im = imread([image_dir '/' file_name]);

  if exist([packages_dir '/' file_name(1:end-4) '.mat'],'file')

    load([packages_dir '/' file_name(1:end-4) '.mat']);
    SR = zeros(size(Im,1),size(Im,2),'uint8'); % Segments R
    SG = zeros(size(Im,1),size(Im,2),'uint8'); %          G
    SB = zeros(size(Im,1),size(Im,2),'uint8'); %          B
    IR = zeros(size(Im,1),size(Im,2),'uint8'); % Instances R
    IG = zeros(size(Im,1),size(Im,2),'uint8'); %           G
    IB = zeros(size(Im,1),size(Im,2),'uint8'); %           B
    [v,u] = find(SR==0);

    for priority=1:2
      for i=1:length(L)
        if labels{L{i}.class}{1}==priority

          in = inpoly([u v],L{i}.polyline);
          SR(in) = labels{L{i}.class}{2}(1)*256;
          SG(in) = labels{L{i}.class}{2}(2)*256;
          SB(in) = labels{L{i}.class}{2}(3)*256;
          IR(in) = randi([0,255]);
          IG(in) = randi([0,255]);
          IB(in) = randi([0,255]);

        end
      end
    end
```

```
    S = zeros(size(Im,1),size(Im,2),3,'uint8');
    Im = zeros(size(Im,1),size(Im,2),3,'uint8');
    S(:,:,1) = SR; S(:,:,2) = SG; S(:,:,3) = SB;
    Im(:,:,1) = IR; Im(:,:,2) = IG; Im(:,:,3) = IB;
    imwrite(S,[segment_dir '/' file_name(1:end-4) '.png']);
    save('segment.mat','S');

    imwrite(Im,[instance_dir '/' file_name(1:end-4) '.png']);

  elseif (~exist([packages_dir '/' file_name(1:end-4) '.mat'],'file'))

    S = zeros(size(Im,1),size(Im,2),3,'uint8');
    Im = zeros(size(Im,1),size(Im,2),3,'uint8');
    S(:,:,1) = 0; S(:,:,2) = 0; S(:,:,3) = 0;
    Im(:,:,1) = 0; Im(:,:,2) = 0; Im(:,:,3) = 0;
    imwrite(S,[segment_dir '/' file_name(1:end-4) '.png']);
    save('segment.mat','S');

  end

end

imshow(S); % S is the segments information, that is, the color poligons into a black background
```

Código de interés 6.9: Generación del mapa de segmentación

En caso de que exista un archivo .mat dentro del directorio packages que contenga algún polígono (y sus respectivas coordenadas), lo representará automáticamente (igual pasa si se borrara una etiqueta). En caso de que no se haya detectado ninguna diferencia, al ejecutar este botón quedará en negro el *groundtruth.*

Para este TFG se han etiquetado un total de 909 secciones a una media de 2 minutos por etiquetado. Al final del capítulo 7: Aplicación de la *Red neuronal convolucional (CNN) en la detección de cambios y resultados finales*, en el apartado de resultados finales, se pueden observar diez ejemplos de *groundtruth*, asociado a la diferencia binaria entre ambas secciones.

Capítulo 7

Aplicación de una Red Neuronal Convolucional (CNN) en la detección de cambios y resultados finales

Este es el último capítulo del proceso secuencial aplicado en este trabajo. En el segundo capítulo se estudió la forma de obtención de localizaciones. En el tercero, la elaboración de los mapas de profundidad e imágenes panorámica. En el cuarto, la síntesis de imágenes virtuales a partir del mapa de profundidad y panorama anteriores. En el quinto, extracción de características y métodos de obtención. En el sexto se estudió la correspondencia de dichos puntos característicos así como el alineamiento entre imágenes. Finalmente, en este séptimo capítulo se estudia la **posibilidad de entrenamiento de una CNN a partir de paquetes** y posterior autoetiquetado de imágenes que presentan cambios estructurales.

Obsérvese que cada paquete contiene tres datos: La **sección del primer panorama**, la **sección del segundo panorama alineada en la horizontal mediante corrección de offset global e individual**, y el ***groundtruth***, es decir, el etiquetado de diferencias entre ambas imágenes).

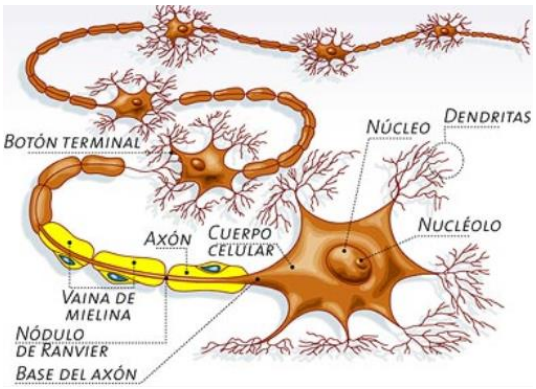
Para la obtención de los resultados, el autor de este TFG ha contado con la colaboración del doctor **Pablo Fernández Alcantarilla**, ex miembro del grupo *RobeSafe* en la Universidad de Alcalá y actual investigador en visión artificial dentro de la compañía *iRobots* (Londres, Reino Unido). Además, el estudio de este capítulo supone una continuidad a uno de sus trabajos ***Street-View Change Detection with Deconvolutional Networks*** [17].

El campo que estudia las redes neuronales es ciertamente complejo y totalmente en auge hoy en día. En este trabajo tiene por objeto el estudio detallado delas CNNs. Sin embargo, es requerido tanto la definición de una red neuronal artificial como de su derivada convolucional para afrontar de la mejor forma obtenida la interpretación de los resultados obtenidos por parte del Dr. Pablo Fernández Alcantarilla.

7.1. Concepto de Red Neuronal Artificial:

Una **Red Neuronal Artificial (RNA)** es un **modelo matemático** (computacional) **inspirado en el comportamiento biológico de las neuronas y de cómo se organizan formando la estructura del cerebro.** Aunque aún está en estudio, desde hace muchos años se sabe que el cerebro es un sistema altamente complejo donde aproximadamente hay cien mil millones (10^{11}) en la corteza cerebral, formando entre ellas más de 500 billones ($5 \cdot 10^{14}$), siendo la media de conexiones por neurona entre 50.000 y 100.000.

El cerebro se puede interpretar como una supercomputadora o sistema inteligente. Aunque las computadoras actuales pueden ser muy rápidas en el procesamiento de información, para tareas muy complejas tales como el reconocimiento y clasificación de patrones, mientras que computacionalmente demandan mucho tiempo y esfuerzo, el cerebro humano es capaz de resolverlas (por ejemplo, reconocimiento de un rostro familiar dentro de una multitud) sin apenas esfuerzo.



Aunque hay distintos tipos de neuronas biológicas, a la izquierda se encuentra un ejemplo común:

- El cuerpo central (**Soma**) contiene el núcleo celular.
- La prolongación del soma: **Axón.**
- Ramificaciones terminales del soma: **Dentritas**.
- Zona de conexión entre una neurona y otra: **Sinapsis.**

Figura 7.1: Neurona biológica

La función principal de estas neuronas es la **transmisión de la información de impulsos nerviosos**. Estos viajan por toda la neurona comenzando por las dendritas hasta las terminaciones del axón, donde la información pasa de una neurona a otra mediante conexión sináptica.

Por tanto, una red neuronal artificial (RNA) pretende emular precisamente el objetivo de las redes neuronales biológicas: Responder ante estímulos del mundo exterior y aprendizaje de este mismo. Es por tanto que se propuso en el año 1943 el primer modelo de RNA capaz de emular, aunque de forma limitada, una red biológica.

7.1.1. Modelo neuronal de McCulloch-Pitts:

Este fue el primer modelo matemático de una **neurona artificial**, creado con el fin de llevar a cabo tareas iguales. Fue un trabajo conjunto entre el psiquiatra y neuroanatomista Warren McCulloch y el matemático Walter Pitts en el año 1943.

La siguiente imagen muestra un gráfico comparativo de la arquitectura de las neuronas artificiales:

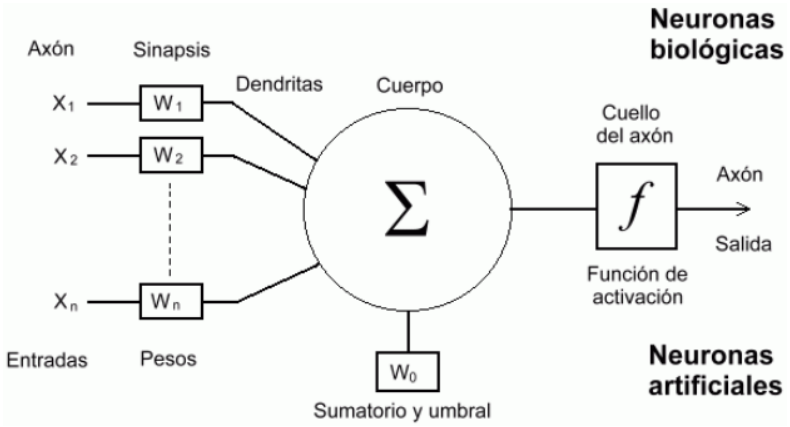


Figura 7.2: Comparación entre neurona biológica y artificial

Este modelo neural de n entradas consta de:

- Un conjunto de entrada $x_1, x_2, \ldots . x_n$, cuyo análogo biológico son los n axones.
- Pesos sinápticos $w_1, w_2, \ldots . w_n$ correspondientes a cada entrada, cuyo análogo biológico es la sinapsis o conexión sináptica.
- Una función de agregación **Σ** (un sumatorio) que correspondería con el soma o cuerpo de la neurona donde se procesa todo.
- Una función de activación **f** o cuello del axón.
- Una saluda **Y** que correspondería a otro axón biológicamente.

Básicamente, este modelo computacionalmente se puede ver como una caja negra, con a la izquierda una entrada y a la derecha una salida y su equivalencia biológica es una neurona (detrás de la cual hay miles de millones de neuronas y delante hay igualmente miles de millones de neuronas, que forman por tanto esta red neuronal).

Las **entradas** son el **estímulo** que la neurona artificial recibe del entorno, y consecuentemente la **salida** es la **respuesta a dicho estímulo**. Una de las principales características de una neurona artificial es que **puede adaptarse al medio que le rodea y aprender de él modificando el valor de sus pesos sinápticos**, también conocidos como parámetros libres del modelo, ya que pueden ser adaptados y modificados en función de la tarea que se quiere llevar a cabo. Dicho de otra forma, un peso es una **ponderación de cada entrada para que influya de mayor o menor manera**, como dejar pasar más o menos líquido de una tubería donde por cada tubería del conjunto circula un fluido distinto, generando a la salida una mezcla con una composición singular.

Con este modelo de McCulloch-Pitts, la salida neuronal será por tanto:

$$Y = f \cdot (\sum_{i=1}^{n} w_i \cdot x_i) \tag{7.1}$$

Siendo f la función de activación de la neurona, la cual se elige de acuerdo a la tarea que se desea realizar. Entre las funciones de activación más comunes dentro del campo de las redes neuronales artificiales, se pueden destacar principalmente:

- Función identidad (identificada con una rampa).
- Función escalón.
- Función sigmoidea.
- Función gaussiana.
- Función sinusoidal.

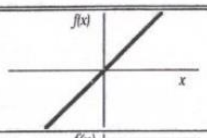
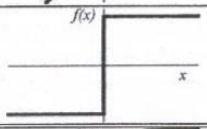
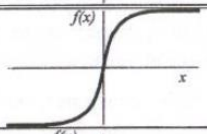
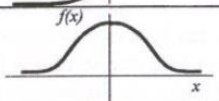
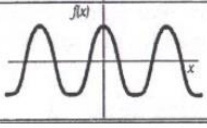
	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	
Gaussiana	$y = Ae^{-Bx^2}$	$[0,+1]$	
Sinusoidal	$y = A\,\mathrm{sen}(\omega x + \varphi)$	$[-1,+1]$	

Figura 7.3: Representación gráfica de las principales funciones de activación

A continuación se muestra un ejemplo para entender de una forma más profunda el funcionamiento de la neurona así como de un nuevo concepto, el **perceptrón**:

Supóngase que se quiere aplicar el modelo neural anterior para realizar tareas de **clasificación binaria en un plano**, por lo que sólo serán necesarias dos entradas (x1 y x2, con sus respectivos pesos w1 y w2). Además, se considerará una tercera entrada b cuyo peso w3 = 1, cuyo resultado por tanto es b.

Si la función de activación se identifica con un escalón:

$$f(s) = \begin{cases} 1 & si\ s \geq 0 \\ -1 & si\ s < 0 \end{cases} \tag{7.2}$$

Siendo s la entrada a la función de activación, o lo que es lo mismo, la salida tras el sumatorio. Por tanto, la salida neuronal estará definida por:

$$Y = \begin{cases} 1 & si\ w_1 \cdot x_1 + w_2 \cdot x_2 + b \geq 0 \\ -1 & si\ w_1 \cdot x_1 + w_2 \cdot x_2 + b < 0 \end{cases} \tag{7.3}$$

Supóngase ahora que existen dos clases a clasificar en el plano: **C1**, que son círculos rojos, y **C2**, que son círculos azules, cada elementos representado dentro de un sistema de coordenadas por tanto cada uno contando con un par de coordenadas {x,y}. Además, supóngase (para facilitar el ejemplo), que **es posible trazar una línea que separe estrictamente ambas clases:**

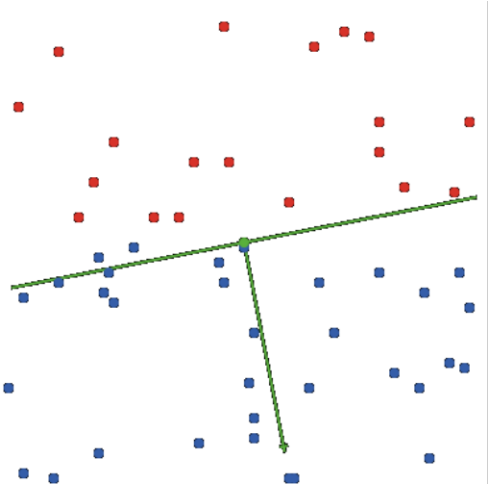


Figura 7.4: Clasificación de clases

Por tanto se dice que una neurona artificial clasifica correctamente ambas clases si dados los pesos sinápticos w1 y w2, y el término b, la recta con ecuación:

$$y = \frac{-w_1}{w_2} \cdot x - \frac{b}{w_2} \tag{7.4}$$

Ecuación que debe satisfacer la recta verde que se observa en el ejemplo y separa las clases, donde $\frac{-w_1}{w_2}$ es la pendiente de la recta y $-\frac{b}{w_2}$ es la recta de origen. Dicha ecuación ha sido despejada a partir de la ecuación implícita de la recta mostrada anteriormente $w_1 \cdot x_1 + w_2 \cdot x_2 + b = 0$, relacionándose ambas a partir de una ecuación matemática básica. Obsérvese que si el punto tomado $\{x_0, y_0\}$ pertenece a la clase C1 (círculos rojos), entonces $w_1 \cdot x_0 + w_2 \cdot y_0 + b < 0$ (porque la línea está por debajo de los puntos rojos) y viceversa con los círculos azules. Por tanto, dado el punto de origen $\{x_0, y_0\} \epsilon\ C_1\ U\ C_2$, la neurona clasifica de la siguiente manera:

$$\{x_0, y_0\}\ \epsilon\ C_1 \rightarrow Y = -1 \tag{7.5}$$
$$\{x_0, y_0\}\ \epsilon\ C_2 \rightarrow Y = 1 \tag{7.6}$$

Si ahora se toman dos clases nuevas C_3 y C_4, distintas de las anteriores pero también separables linealmente, la neurona anterior no podría clasificar de forma correcta estas dos nuevas clases, puesto que tras la definición de la ecuación a la neurona anterior se le ha dotado de unos pesos y entrada b con

valores concretos de forma que la recta pase exactamente entre ambos puntos, por tanto **la recta anterior no sería válida** (suponiendo que ambas distribuciones de puntos no son exactamente iguales).

Sin embargo, dada la ya construida ecuación punto-pendiente, que se mantiene invariante porque se hace la clasificación sobre un plano (ecuación 7.2), **se pueden modificar los parámetros anteriores w1, w2 y b, a los nuevos parámetros w1', w2' y b' para que la nueva recta satisfaga la nueva distribución.** Esta transición es lo que se conoce como método de aprendizaje de una neurona, proceso que permite modificar los parámetros libres (pesos) con el fin de que la neurona se adapte a realizar distintas tareas.

El método de aprendizaje más común es el **método de error-correción**, permitiendo así adaptar los pesos (parámetros libres) con el fin de realizar la tarea a cometer de la mejor forma posible (en este sencillo ejemplo la clasificación de las clases C1 y C2). Para aplicar dicho método es necesario:

- **Un conjunto de entrenamiento, D** (en este caso equivaldría al conjunto de puntos C1 y C2, es decir, C1 U C2).
- Un **seleccionador** o instructor que se encargará de tomar datos aleatoriamente.
- **Valores iniciales de los pesos w1', w2' y b'** arbitrarios (pueden ser cualesquiera).

El procedimiento será el será el siguiente:

- El **seleccionador toma un elemento** $\{\boldsymbol{x_0}, \boldsymbol{y_0}\}$ ϵ D al azar y lo presenta ante la neurona artificial.
- Como se comentó anteriormente, la salida debía ser -1 si el punto pertenecía a C1 o 1 si el punto pertenecía a C2. Si la **neurona clasifica mal este punto**, es decir, si la salida Y = -1 con el punto perteneciendo a C2, o 1 con el punto perteneciendo a C1, se aplica la siguiente correción a los pesos iniciales:

$$w_1 = w_1' + d \cdot x_0 \tag{7.7}$$
$$w_2 = w_2' + d \cdot y_0 \tag{7.8}$$
$$b = b' + d \tag{7.9}$$

Siendo w1, w2 y b los nuevos parámetros a utilizar y w1', w2' y b' los iniciales con valores arbitrarios. El valor de d se obtiene de la siguiente forma:

$$d = \begin{cases} 1 & si\ Y = -1 \quad y \quad \{x_0, y_0\} \ \epsilon \ C_2 \\ -1 & si\ Y = 1 \quad y \quad \{x_0, y_0\} \ \epsilon \ C_1 \end{cases} \tag{7.10}$$

Es decir, si la correspondencia está mal, el valor d toma el valor contrario a la salida.

- **Se repite el análisis de este primer punto tomado**. Si la neurona lo clasifica bien (como debería ser), no se realiza ninguna corrección.
- El procedimiento se repite **pasando a la neurona otro punto del conjunto D y usando los últimos parámetros usados** (es decir, w1, w2 y b, no los arbitrarios del inicio w1', w2' y b'). Si la neurona clasifica mal el punto, de nuevo se aplica la fórmula anterior.
- Esta tarea se repetirá con **todos los puntos del conjunto D**, aplicando la técnica de corrección en caso de mala clasificación.
- El entrenamiento terminará cuando la **neurona clasifique correctamente todos los elementos del conjunto de entrenamiento**. Es decir, este procedimiento converge en un número finito (en el mejor de los casos, sino se aplica ninguna corrección, igual al número de elementos del conjunto D = C1 U C2) obteniéndose finalmente los parámetros finales que elaborarán las recta punto pendiente capaz de separar ambos conjuntos.

Un **modelo neural** usado para la **clasificación**, cuya salida se encuentra determinada por las **ecuaciones** anteriores y que usa el método de **error-corrección** para el ajuste de los parámetros finales se conoce como ***Perceptron***. Un ejemplo de perceptrón simple pero con varias entradas es el siguiente:

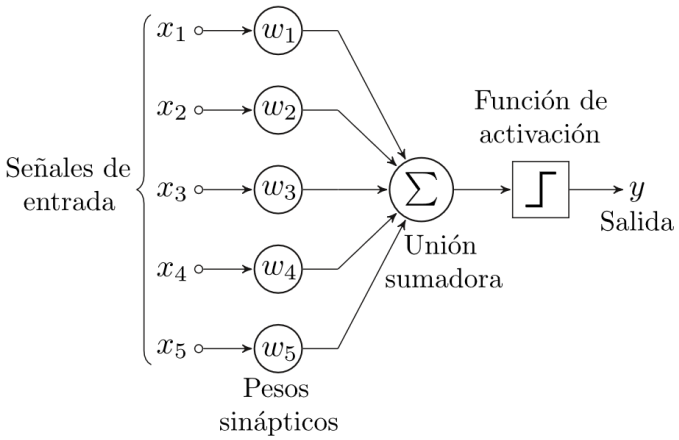


Figura 7.5: *Perceptron* simple

En la Figura 7.5 equivale a una neurona con cinco entradas, con su respectiva unidad sumadora, función de activación y salida. Sin embargo, este modelo, por muchas entradas que tenga, presenta un problema: **No permite resolver problemas que no son linealmente separables** (principal limitación del perceptrón). Es por ello que varias neuronas (o perceptrones) se agrupan en lo que se conoce como **red neuronal artificial (RNA) también conocida como perceptrón multicapa.**

7.1.2. Perceptrón multicapa:

Una **RNA, perceptrón multicapa o perceptrón múltiple** consiste en la unión de forma ordenada de neuronas simples (perceptrones simples) para cumplir alguna función de interés. Un caso particular se puede formar organizando neuronas en sus capas:

- La capa de entrada está formada por las entradas a la red.
- La capa de salida está formada por las neuronas que constituyen la salida final de la red.
- Las capas ocultas están formadas por las neuronas que se encuentran entre los nodos de entrada y de salida.

Aunque **por lo general una RNA tiene varias capas ocultas**, también puede no tener ninguna (en función de la complejidad de la aplicación). Las conexiones sinápticas (flechas que entran y salen de las neuronas) indican el flujo de la señal a través de la RNA, con un peso sináptico correspondiente (es decir, no se pone por cada neurona el símbolo de los pesos, sumatorio y función de activación, sino que se simplifica con las flechas). Si la salida de una neurona se bifurca hacia dos o más neuronas de la siguiente capa, cada neurona de la siguiente capa tendrá como entrada la respectiva combinación de entrada de la capa anterior.

La cantidad de capas de una RNA, denotada como CT (Capas totales) será:

$$CT = CE + COs + CS \tag{7.11}$$

Siendo CE la capa de entrada, COs el número de capas ocultas total y CS la capa de salida. Cuando un sistema presenta, además de la capa de entrada y de salida, capas ocultas (al menos una), en general la RNA pasa a denominarse perceptrón multicapa. Un ejemplo de perceptrón múltiple se muestra a continuación:

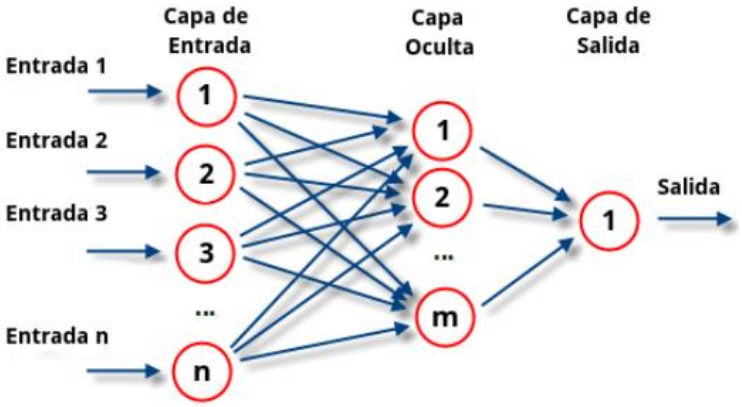


Figura 7.6: *Perceptron* múltiple

Mientras que los perceptrones simples (neuronas simples) eran usados para simples tareas de clasificaciones lineales, el problema habitual que se resuelve con este tipo de redes multicapa es el de, **dado un conjunto de datos ya clasificados** (por ejemplo, con el método de error-corrección con un perceptrón simple), **de los que se conoce la salida deseada, dotar a la red de la pesos adecuados para que se obtenga una aproximación correcta de las salidas que se deben obtener si la red recibe únicamente los datos de entrada.**

Para entrenar estas redes se usa el algoritmo de **Propagación hacia atrás (*Backpropagation*),** el cual aproxima los pesos a partir de los datos objetivo. Este algoritmo de Propagación hacia atrás se puede resumir (brevemente dada su complejidad) en los siguientes pasos:

- Al inicio del proceso, los **pesos sinápticos son arbitrarios** (al azar).
- **Introducir datos en la capa de entrada**, elegidos al azar entre el conjunto de datos de entrada de entrenamiento (D).
- Proceder a que la **red genere un vector de datos de salida**, es decir, dejar que todo fluya hacia delante (*Forward propagation*).
- **Comparar** la salida deseada con la generada por el perceptrón múltiple.
- La **diferencia obtenida** entre la salida deseada y la generada (denominada error) es el dato clave para ajustar los pesos sinápticos de las neuronas de las capas de salidas.
- Este error se propaga hacia atrás (***Back*propagation**) para ir ajustando paso a paso los pesos sinápticos de las sinapsis que unen capas anteriores.
- El proceso continua propagando el error hacia atrás y **ajustando pesos** hasta que se alcanza las capas de entrada.
- Este proceso se repetirá con los diferentes **datos de entrenamiento** (si el perceptrón tiene cinco entradas y hay cinco mil datos de entrenamiento, el proceso iteraría los pasos anteriores mil veces).

Entre las múltiples tareas en las que una RNA puede aplicarse puede haber:

- Clasificación lineal y/o no lineal de una cantidad arbitraria {C1, …, Cn} de clases.
- Regresión estadística lineal y no lineal.
- Análisis de series temporales.
- Control de procesos.
- Optimización de funciones.
- Procesamiento de señales.
- Etc

7.2. Redes neuronales convolucionales:

El sentido de la visión es uno de los grandes prodigios de la naturaleza. En fracciones de segundo, los datos pueden percibir píxeles y el cerebro transformar esa información para identificar líneas, curvas y formas, por ejemplo, cuando se mira el rostro de una persona. Es lógico por tanto inspirarse en una

variante de RNA especializadas en la identificación de imágenes llamada redes neuronal convolucional (***Convolutional Neural Network, CNN***).

Conceptualmente, **una CNN es una RNA donde las neuronas se comportan de manera muy similar a las neuronas biológicas de la corteza visual primaria (V1) de un cerebro biológico**. Este tipo de red es una **variación de un perceptrón multicapa**. Sin embargo, dado que su aplicación es realizada en matrices bidimensionales, es enormemente útil en tareas de *Machine learning* y de visión artificial, por ejemplo en la segmentación y clasificación de imágenes.

Lo que diferencia a una CNN del resto de RNAs es que **explícitamente se supone que los datos de entrada son imágenes**, por lo que la arquitectura generalmente se diseña para esta tarea, ganando en eficiencia (reduciendo la cantidad de parámetros en la red, innecesarios para esta tarea). Una CNN pretende resolver el problema que presenta una RNA ordinaria: Esta última no escala bien las imágenes de mucha definición.

Una CNN trabaja modelando de forma consecutiva pequeñas porciones de información, combinando esta misma información en capas profundas:

- La primera capa tendrá como misión la **detección de bordes y establecer patrones de detección de bordes más complejos.**
- Las capas posteriores tratan de **combinar dichos bordes en formas más simples** y finalmente en patrones de las diferentes posiciones de los objetos, iluminación, escalas, etc…
- Las capas finales intentarán hacer **coincidir una imagen de entrada con todos los patrones** y estimar una predicción final como una suma ponderada de todos ellos.

De estas formas, las CNNs son capaces de modelar complejas variaciones y comportamientos dando unas predicciones bastante precisas.

7.2.1. Fundamentos biológicos:

Los fundamentos de una CNN están basados en el ***Neucognitron*** de Kunihiko Fukushima en 1980. En 1998 se incorpora el método de *Backpropagation* para entrenar al sistema correctamente, y en 2012 fueron refinadas e implementadas en CPU logrando resultados impresionantes.

Biológicamente están basadas en las células responsables de la selectividad de la orientación de bordes en los estímulos visuales dentro de la **corteza visual del cerebro V1**. Dentro de esta corteza se distinguen principalmente dos células: simples y complejas, capaces de tener una mejor respuesta a los estímulos visuales alargados, como líneas y bordes, debido a un campo receptivo más alargado:

- Las **células simples** presentan regiones de excitación e inhibición que **forman patrones elementales alargados, con una dirección, posición y tamaño en particular**. Si un estímulo visual llega con la misma orientación y posición que el patrón de la región excitadora, esta se activa (al mismo tiempo que se evita activar las regiones inhibitorias). Ante tales condiciones, la célula se activa y emite una señal.
- Las **células complejas** operan de forma similar pero, a diferencia de las células simples cuyo patrón excitatorio tenía una orientación en particular, las células complejas presentan **invarianza a la posición**, por lo que el estímulo visual sólo necesita llegar a la célula con la región correcta para activarla.

Por encima de la región V1, están la V2, V4 e IT, donde la complejidad de los estímulos visuales cada vez es mayor. Al mismo tiempo, las activaciones de las células se hacen menos sensibles a la posición y tamaño de los estímulos iniciales. Esto es producto de la activación de las células y propagación de sus propios estímulos a otras células conectadas de la misma jerarquía, gracias a la alternancia entre células simples y complejas.

7.2.2. Arquitectura:

Una CNN consiste en **múltiples capas de filtros convolucionales** de una o más dimensiones. Después de cada capa, generalmente se añade una función para realizar un mapeo casual no-lineal.

Como RNA basadas en la clasificación, estas redes están construidas en torno a una estructura que presenta tres tipos distintos de capas:

1. En primer lugar, una **capa convolucional** que es la que le da nombre a la red.

2. En segundo lugar, una **capa de reducción o *pooling***, la cual reduce la cantidad de parámetros para quedarse con las características más comunes. Entre esta capa y la anterior (capa de convolución) se establece lo que se conoce como fase de extracción de características.

3. Al final de la red se encuentran neuronas de perceptrón sencillas para realizar la **clasificación final** sobre las características extraídas.

Esta fase de extracción de características se asemeja al proceso estimulante en las células de V1. Esta fase se compone en capas alternas de neuronas convolucionales y neuronas de reducción de muestreo, siendo las neuronas en capas lejanas mucho menos sensibles a perturbaciones en los datos de entrada, al mismo tiempo que siendo activadas por características cada vez más complejas.

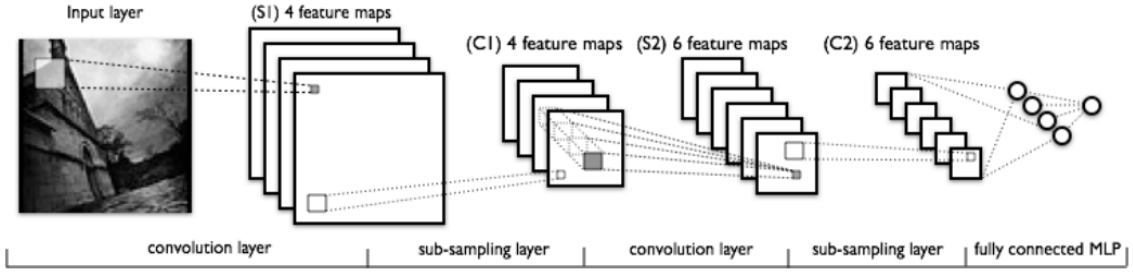


Figura 7.7: Ejemplo simple de CNN

7.2.2.1. Capa convolucional:

Lo que distingue una CNN de una RNA estándar es el uso de la operación convolución en alguna de sus capas, en lugar de la multiplicación de matrices estándar. La salida de cada neurona convolucional se define como:

$$Y_j = g \cdot (b_j + \sum_i K_{ij} * Y_i) \tag{7.12}$$

Donde la salida Y_j de la neurona j es una matriz que se calcula mediante la combinación lineal de las salidas Y_i de las neuronas de la capa anterior, cada una de ellas operadas con el núcleo de convolución (* denota el operador convolución) o kernel K_{ij} correspondiente a dicha conexión. Esto es sumado a la influencia b_j y posteriormente se multiplica por la función de activación no lineal g.

La operación de **convolución** recibe como ***input* la imagen**, luego se aplica sobre ella **un filtro *o kernel*** (núcleo) que devuelve un **mapa de carácterísticas** de la imagen original, logrando la **reducción del tamaño de los parámetros.** La convolución aprovecha tres grandes conceptos que ayudan a mejorar cualquier sistema basado en el aprendizaje de la máquina (*machine learning*):

- **Interacciones dispersas**, ya que al aplicar un filtro de menor tamaño sobre la entrada original se reduce enormemente la cantidad de parámetros y cálculos.
- **Parámetros compartidos**. Esto indica la compartición de ciertos parámetros entre distintos tipos de filtros permite mejorar la eficiencia del sistema.

- **Representación equivalente**. Si las entradas cambias, las salidas deben cambiar de formal similar.

Estos *kernels* presentan habilidades de procesamiento de imágenes específicas, por ejemplo resaltar (y por tanto previamente detectar) bordes, gracias a núcleos que resaltan el gradiente en una dirección en particular, tal y como se vio por ejemplo con el algoritmo de Canny. En general, los ***kernels* entrenados para las convoluciones de una CNN son mucho más complejos para permitir la detección de formas más abstractas**, no sólo bordes.

Por otra parte, la convolución también proporciona un entorno donde se puede trabajar con entradas de tamaño variable, característica muy conveniente.

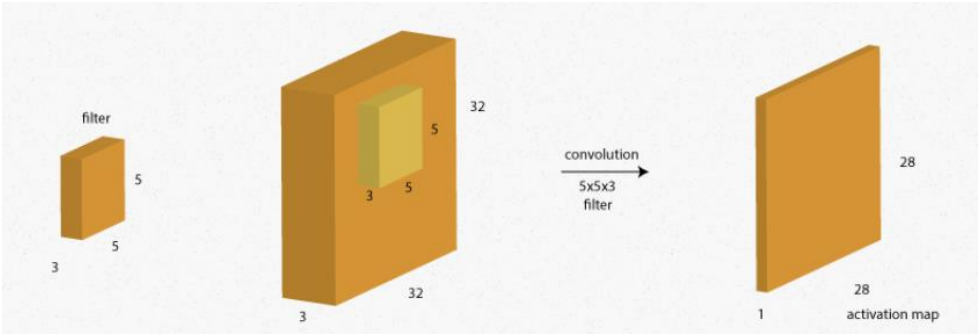


Figura 7.8: Capa de convolución de una CNN

7.2.2.2. Capa de reducción de muestreo:

Una CNN (una RNA en general) cuenta con una determinada **tolerancia ante pequeñas perturbaciones en los datos de entrada**. Por ejemplo, con dos imágenes casi idénticas, que se diferencia sólo en el traslado de algunos píxeles lateralemente, si se analizaran con una CNN, el resultado debería ser el mismo. Esto se debe a la **reducción de muestreo dentro de una CNN**: Al reducir la resolución (reduciendo por tanto información), las mismas características corresponderán a un mayor campo de activación en la imagen de entrada, por tanto incluso aunque hubiera pequeño ruido en la imagen, como no se muestrea hasta el punto de detectar dicho pequeño ruido, este se omite .

Originalmente, las CNNs usaban un proceso de submuestreo para llevar a cabo esta operación. Sin embargo, hoy en día se ha demostrado que hay una técnica mucho más eficaz (y parece ser que biológicamente se asemeja mucho el proceso) para resumir las características de una región. Esta técnica se denomina ***max-pooling.***

El proceso del *max-pooling* **generalmente se coloca después de la capa convolucional**. Su utilidad principal es la **reducción de las dimensiones** (ancho y alto) de la imagen de entrada para la siguiente capa convolucional, **no afectando a la dimensión de profundidad del volumen**. En concreto, la técnica *max*-pooling es una variante mejorada del pooling que **básicamente encuentra el valor máximo** (del pixel) entre una ventana de muestra y denota este valor como el resumen de características sobre esa área. Es por tanto que el tamaño de datos se reduce en un factor igual al tamaño de la ventana:

- A mayor ventana, menor número de datos finales y peor parecido a la imagen original.
- A menos ventana, mayor número de datos finales y más parecido a la imagen original.

Una de las grandes ventajas de esta reducción de muestreo es, al reducir el tamaño, **dotar a las siguientes capas de la red una menor sobrecarga de cálculo.**

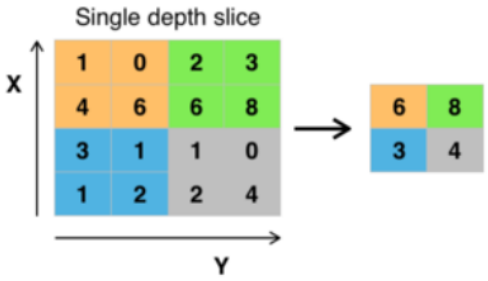


Figura 7.9: Ejemplo de *maxpooling*

7.2.2.3. Capa de clasificación:

Tras las fases de extracción de características (convolución y reducción) llega la clasificación. Tras estas dos primeras fases, **los datos han sido transformados en una serie de características únicas para la imagen de entada**. En esta última fase, según los objetivos de entrenamiento, se clasificarán estas características hacia una clase (etiqueta) u otra.

En esta fase las neuronas funcionan de forma idéntica a las de un perceptrón multicapa:

$$y_j = g \cdot (b_j + \sum_{i=1}^{n} w_{ij} \cdot y_i) \tag{7.13}$$

Donde la salida y_j de una neurona j se calcula mediante la combinación lineal de las salidas y_i de las neuronas de la capa anterior (cada una multiplicada por su peso sináptico w_{ij}), sumándole la influencia b_j y finalmente siendo multiplicada por la función de activación g.

Esta última capa clasificadora tendrá tantas neuronas como número de clases se quiera predecir.

7.2.3. Aplicaciones:

Las aplicaciones principales de una CNN están basadas principalmente en el concepto de convolución. Una CNN es apta para aprender a **clasificar todo tipo de datos distribuidos** de una forma continua a lo largo de un mapa de entrada, siendo similares estadísticamente en cualquier lugar del mapa de entrada. Por esta razón, son **especialmente eficaces en la clasificación de imágenes, por ejemplo para el auto-etiquetado de imágenes** (Objetivo final e importantísimo de este TFG).

Otras aplicaciones son la clasificación de series de tiempo o señales de audio mediante convoluciones 1D o clasificación de datos volumétricos usando convoluciones 3D.

7.2.4. CNN para la detección de cambios a partir de GSV:

Lo que resta de capítulo supone el culmen del proceso secuencial que se ha visto a lo largo de los capítulos, esto es, el **etiquetado automático de la red neuronal convolucional en pares de imágenes sintetizados a partir de enlaces de Google Street View** (objetivo final del trabajo). Ya en la introducción de este trabajo se habló brevemente de las características de la CNN y los objetivos. A lo largo de este apartado se profundizará en este concepto.

La **detección de cambios basado en imágenes** es un problema importante en el campo de la visión artificial y la robótica, desde la identificación de áreas de cambio en una escena hacia otras muchas aplicaciones (por ejemplo, en la navegación autónoma). Por ejemplo, puede ser mejorada la eficiencia del mantenimiento de un mapa tridimensional (3D) mediante una restricción: **Actualización únicamente de las áreas que presentan cambios**, o en su defecto permitir que un sistema aprenda acerca de la naturaleza de los objetos en el entorno mediante su segmentación a medida que cambia, en vez de tener que hacer de nuevo un muestreo total del entorno para elaborar el mapa (**gran ahorro computacional**).

En la detección de cambios urbanos, el cual será el objetivo final de este trabajo, la típica técnica de la detección de estos cambios está separada en dos partes: **Registro y cálculo de similitud**. El registro es requerido porque las bases de datos grandes de zonas urbanas están a menudo escasamente muestreadas y es común tener una referencia de varios metros cuando se compara una imagen con la más cercana a través del tiempo, dando pie a deformaciones en la perspectiva. Por otra parte, generar modelos de superficie lo suficientemente detallados es un problema realmente desafiante, para el que varios autores propusieron soluciones alternativas.

Después de alinear estas imágenes de forma parcial a través de la síntesis (capítulo 4 *Sintetización de imágenes a partir de imágenes panorámicas*), se emplea la segunda parte de la técnica de detección de cambios, esto es, la **determinación de los cambios de interés mediante condiciones de similaridad**, ignorando otros cambios debido al ruido. La definición de qué es un cambio estructural y qué es ruido variará en función de la tarea.

Estos cambios de interés pueden ser puramente geométricos, tales como la aparición o demolición de estructuras urbanas, o incluso de textura, tales como defecto en la superficie, en las fachas, etc… En general, en todo tipo de tareas debe haber invariancia ante ruido acusado por la vegetación (crecimiento de esta en primavera, suelta de hojas en otoño, nieve en invierno, cambio de color a lo largo del año, …) y a los cambios de luminosidad (sombras principalmente).

Estas funciones de similaridad son típicamente una combinación de diferencias absolutas de color, profundidad y distancias entre descriptores *hand-crafted* (descriptores manuales). Una de las primeras **CNN utilizadas en el autoetiquetado de cambios fue la CNN basada en superpixelación [18].** Sin embargo, a pesar de su excelente rendimiento, el **método recae sobre la regularización de superpixeles** (es decir, se analiza en vez de un pixel una región de estos) para delinear los cambios, **técnica que no es lo suficientemente precisa ante ciertos tipos de imágenes complejas**. Es por ello que en el trabajo ***Street-View Change Detection with Deconvolutional Networks*** [17] se propuso el uso de una red deconvolucional (un tipo de red convolucional), técnica también adoptada en este trabajo.

7.2.4.1. Arquitectura de la CNN usada:

Originalmente, la arquitectura de la CNN de este trabajo [17] para la detección de cambios estaba basada en la idea de apilar bloques de contracción y expansión, consistente en aproximadamente 1.4 millones de parámetros, muchísimo menor que otro tipo de CNN con la misma misión (la CNN denotada como FCN-8s que contaba con 134 millones). Su arquitectura era la siguiente:

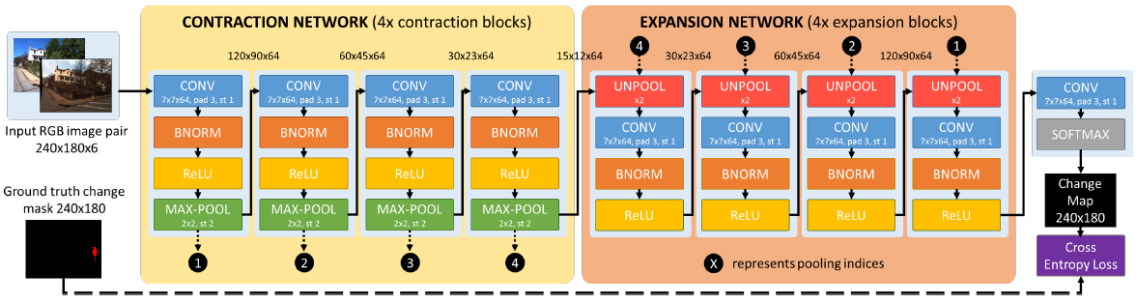


Figura 7.10: Arquitectura original

Para entender la mejora incorporada en la nueva red, primero se debe entender la original a grandes rasgos (no es objetivo de TFG la comprensión total de la red, mayormente porque el autor no ha sido su creador, sino que solo ha interpretado los resultados obtenidos), ya que los procesos y comportamientos son muy similares:

Los cuatro bloques que constituyen la **capa de contracción** sirven para **crear una representación rica que permite el reconocimiento, como ocurre en las CNNs de clasificación estándar**. Por otra parte, los cuatro bloques que forman la **etapa de expansión** son usados para **mejorar la localización y el**

delineamento de las regiones cambiantes. La decisión final de cambio es llevaba a cabo a partir de un clasificador lineal ***softmax*** (de cierta complejidad, por tanto no se tratará en este trabajo) operando densamente por pixel.

Cada bloque de contracción consiste de una **capa de convolución** 7 x 7 con un número fijo de características (64). Sus salidas son normalizadas mediante el uso de un bloque ***batch normalization*** antes de la activación no lineal, con el fin de reducir el desplazamiento covariable interno durante el entrenamiento e incrementar la convergencia de los resultados. En el caso original, los parámetros del proceso *batch normalization* no eran calculados usando estadísticas sino que se calculaban aprendiendo de parámetros extra (es decir, como si fuera un entrenamiento). Esto permite que el proceso *batch normalization* sea fácilmente omitido durante el proceso de test (conceptualmente se usa el término *bypassed* para indicar que se omite durante el proceso de test).

Por otra parte, para la función no lineal de las neuronas de la CNN se usa una función de activación denominada ***ReLU (Rectified Liner Units)***, sin entrar en detalle, este se identifica con la función de activación rampa que presenta la siguiente ecuación: $f(x) = \ln(1 + exp(x))$, siendo x la entrada a la neurona), y finalmente el conjunto de cuatro bloques termina con el proceso de ***max-pooling*** (de 2 x 2 dimensiones, como se se estudió en el apartado anterior) encargado de reducir el número de muestras. En concreto, con un *max-pooling* ***2 x 2*** **se logra reducir el número de muestras a la mitad** (por ejemplo, a partir de una matriz de 4 x 4, con esta reducción de muestreo se queda en 2 x 2):

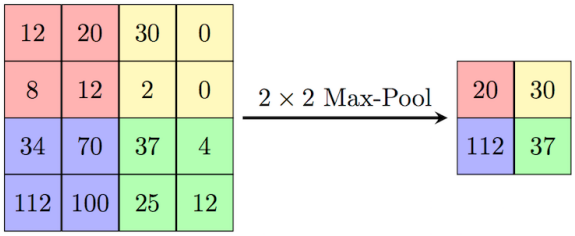


Figura 7.11: Ejemplo de *max-pooling* 2x2

Además, aparte de la reducción de datos para aumentar la eficiencia del proceso, el hecho de almacenar el valor más grande de una región es bueno (para su uso futuro en el correspondiente bloque de expansión) para conseguir los límites bien delimitados.

Respecto a la etapa de expansión, cada bloque comienza por un proceso de ***unpooling*** (concepto contrario al *max pooling*, aumentando el número de datos). Este bloque hace uso de los índices previamente almacenados durante el proceso de *max pooling* para producir una versión sin muestrear de la entrada, que sin embargo ya presenta localización de los bordes y otras características remarcables. Esta operación es seguida por una 7 x 7 convolución de 64 características (igual que antes), se sigue con la clasificación *batch* y finalmente la función de activación no lineal ReLU.

Esta pila de bloques de contracción y expansión hace que la arquitectura de la red sea **altamente simétrica** en términos de características. Sin embargo, este tipo de arquitectura **no presenta capas altamente conectadas**, abriendo la posibilidad a modelos mucho más eficientes.

Es así como surge la nueva arquitectura de la CNN, **más novedosa**, mucho **más eficiente** (aunque conceptualmente tiende a hacer lo mismo, etiquetado de imágenes a partir de un *groundtruth* y dos secciones alineadas que presentan cambios estructurales entre sí) y por tanto más complejo de entender, no entrándose en detalle en este trabajo.

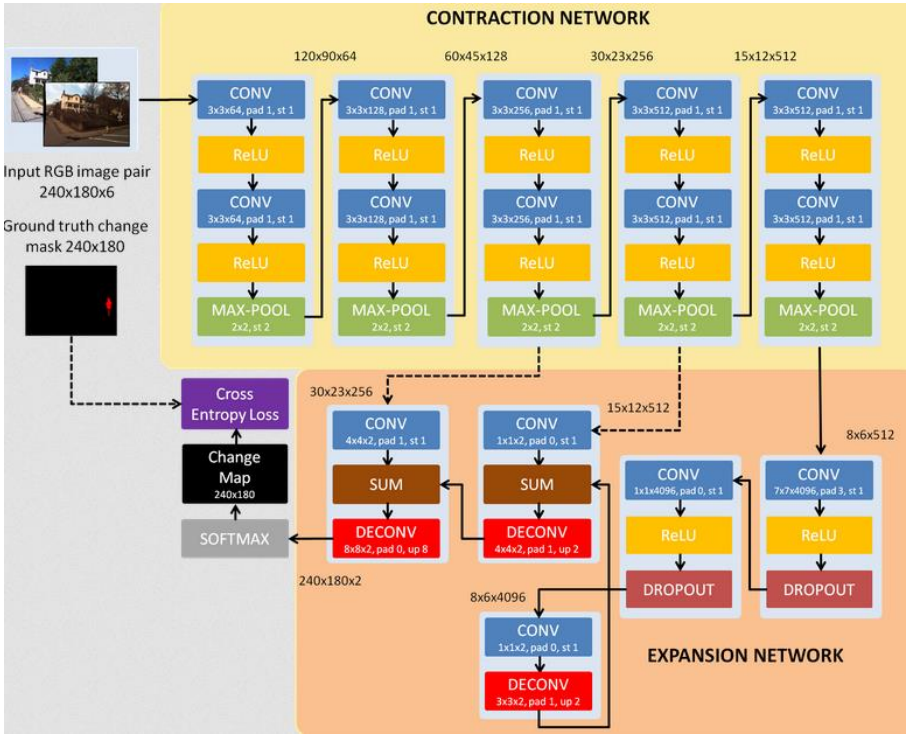


Figura 7.12: Arquitectura mejorada

El entrenamiento es llevado a cabo usando el optimizador de Adam. El algoritmo de **ADAM** (Automated Discovery and Analysis Machine) es un framework capaz de limpiar, analizar y modelar un conjunto de datos proporcionado, de manera automática, extrayendo tanta información como sea posible. Esta elección resulta en un entrenamiento más rápido que en otros procesos. Además, el uso del bloque final ***Cross-entropy loss*** (una función de coste o función de pérdida que relaciona un evento con un número real que representa el coste económico asociado, en este caso computacionalmente, asociado al evento) ponderado es un elemento también clave para aumentar la eficiencia en los resultados que se mostrarán a continuación.

7.2.4.2. Conjunto de entrenamiento:

Como se comentó a lo largo del presente trabajo, el conjunto de entrenamiento son 909 pares de imágenes, generando por tanto 2927 datos de entrenamiento (909 de sección de referencia o *query*, 909 de nueva sección, es decir, del segundo panorama y 909 de etiquetado en blanco y negro, es decir, el *groundtruth*). Un ejemplo se muestra a continuación:

Figura 7.13: Izquierda, sección de referencia. Centro, nueva sección. Derecha, *groundtruth binario*

Siendo a clasificación de clases la siguiente:

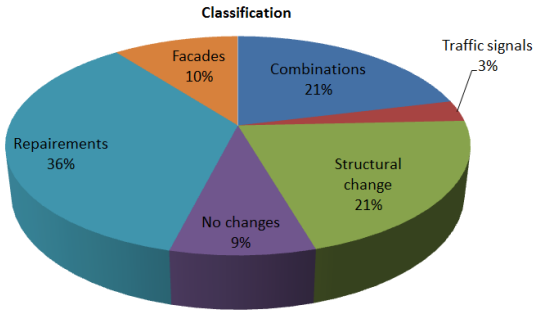


Figura 7.14: Clasificación de cambios etiquetados

Por ejemplo, si se compara el conjunto de entrenamiento obtenido por otros autores:

Dataset	Secuencias	Número de pares	Tipo
Taneja	**4**	**50**	**Perspectiva**
Sakurada	**23**	**92**	**Perspectiva**
Sakurada and Okatani	**-**	**200**	**Panorámica**
VL-CMU-CD (PFA)	**152**		**Perspectiva**
Dataset actual (CGH)	**559 (sólo 354 usadas)**	**909**	**Perspectiva**

Tabla 7.1: Comparativa entre distintos *datasets*

Se puede observar que anteriores *dataset* (evidentemente dentro del mismo ámbito, esto es, la detección de cambios con Google Street View) tienen muchísimas menos secuencias (pudiendo ser de vídeo, geolocalizaciones en este caso de donde posteriormente se seccionarán, etc….) que el actual dataset (CGH = Carlos Gómez Huélamo, autor de este TFG) usado en este trabajo. El número de pares obtenido es 909, pero puede ser muchísimo mayor cuando se mejore el código consiguiendo un seccionamiento total de la panorámica (no sólo una porción del panorama), estimando un aumento del 200 % (de 909 pares a casi 3000 a partir de estas 354 localizaciones usadas), resultando dos órdenes de magnitud superiores respecto de sus predecesores. Es importante notar que tanto el dataset VL-CMU-CD (PFA = Dr. Pablo Fernández Alcantarilla) como el dataset actual, debido a que se encuentran tomados a lo largo de grandes periodos de tiempo, **presentan cambios estructurales y además cambios desafiantes** debido a la iluminación, vegetación, cambio estacional, etc: **Al aumentar la complejidad del análisis, la arquitectura de la CNN se debe mejorar, por tanto con casos más sencillos la CNN será capaz de trabajar de forma extraordinaria.**
Finalmente, se llega al cénit de este TFG, esto es, la muestra de resultados obtenidos a partir de la CNN.

7.2.4.3. Resultados:

En este apartado se examina la **eficiencia de la red deconvolucional** (*Deconvolutional Network, DNN*), **para la detección de cambios usando el actual *dataset***, frente a otros tipos descriptores *hand-crafted* (por ejemplo, el descriptor DAISY [21]) e incluso a la CNN elaborada en el paper de Sakurada y Okatani [18] cuyo *dataset* se constituía del imágenes panorámicas.

Para el entrenamiento final de esta DNN (denotada en las gráficas con CD-Net), **del dataset propio que contiene 909 pares se usarán únicamente 544** (porque el resto presenta *ghosting* o solapamiento del extremo izquierdo del panorama con el extremo derecho, tal y como se reflejó en el tercer capítulo) **más 300 sintéticas a partir de estas 544**, dificultando un poco las condiciones (por ejemplo mediante la aplicación de máscaras, objetos sintéticos, cambios en la iluminación, saturación, ….), haciendo un total de 844 para el entrenamiento (65 % con cambios reales, 35 % cambios sintéticos). Por otra parte, se usarán 233 pares para el test, es decir, el autoetiquetado de las imágenes, teniendo siempre el *groundtruth* de referencia manualmente etiquetado por el usuario en la herramienta para constrastar resultados.

Es obvio que otra de las futuras mejoras de este trabajo es, apoyado en el concepto global, online, fácil de usar y gratuitio de GSV, aumentar en al menos dos orden de magnitud (mínimo 10.000 el número de pares de imágenes) para aumentar el aprendizaje de la red y poder así etiquetar mejor los resultados.

Para entender apropiadamente los resultados, se deben definir los siguientes conceptos: *False positive (negative) rate, True positive (negative) rate* y la curva de *Precision vs Recall*:

- El ***False Positive Rate*** **(FPR)** es la **proporción de todos los negativos que producen resultados positivos**, esto es, la probabilidad condicional de que un evento que no estaba presente de un resultado de prueba positivo. También es denominado **nivel de significancia**, siendo la especificidad de un test igual a 1 – FPR.
- Complementario al FPR está el ***False Negative Rate*** **(FNR),** que indica la **proporción de positivos que resultan en resultados negativos dentro del test**, es decir, la probabilidad condicional de que un evento presente de un resultado de prueba negativo (es decir, que no se esperaba ese resultado, debía ser otro, etc…).
- El ***True Positive Rate*** **(TPR, también llamado sensibilidad o probabilidad de detección)** mide la **proporción de positivos que son correctamente identificados**, por ejemplo, el porcentaje de gente enferma en una población que verdaderamente está enferma.
- El ***True Negative Rate*** **(TNR, también llamado especificidad)** mide la **proporción de negativos que están correctamente identificados** como tal, por ejemplo, el porcentaje de gente sana en una población no porque esté sana, sino porque NO está enferma (de ahí el término True Negative).

Es obvio que en el caso de las figuras que se mostrarán a continuación, estos cuatro parámetros mostrarán los resultados de si una imagen ha sido:

- **Etiquetada sin presentar cambio (Mal)**
- **Etiquetada habiendo cambio (Bien),**
- **No etiquetada habiendo cambio (Mal),**
- **No etiquetada no habiendo cambio (Bien).**

- **Curva *Recall-Precision*:** En tareas de clasificación binaria (relevante o no relevante), la ***precision*** es **la fracción de los datos recuperados que son relevantes**, mientras que la ***recall*** es la fracción de los **datos relevantes que son recuperados**. Esta curva es verdaderamente útil para determinar el rendimiento de un clasificador.

$$Precision = \frac{t_p}{t_p + f_p} \qquad (7.11)$$

$$Recall = \frac{t_p}{t_p + f_n} \qquad (7.12)$$

Siendo t_p el número de positivos verdaderos, f_p el número de positivos falsos y f_n el número de negativos falsos.

Hay que tener en cuenta que la detección de cambios para este TFG se ha llevado a cabo usando un FPR del 10 % (0.1 en escala de 1).

A. Comparación cuantitativa:

Se compara los resultados de tres maneras, TNR vs FNR, TPR (o *Recall*) vs FPR y la curva *Recall-Precision*, siendo en todos los casos la red CDNet mejor que el resto de ejemplos, es decir, es **capaz de identificar cambios cuando verdaderamente hay cambios mejor que ninguna** (Figura 7.16 TPR vs FPR, se puede

observar en los diez primeros ejemplos) y es **capaz de no identificar cambios cuando efectivamente no existen cambios mejor que ninguna** (Figura 7.15 TNR vs FNR, se puede observar en los tres últimos ejemplos). Sin embargo, la **gráfica más importante** de todas es la curva *Precision* vs *Recall* 7.17.

Como se comentó anteriormente, la precisión mide la calidad de las respuestas positivas del clasificador, es decir, mide el porcentaje de los cambios etiquetados que evidentemente son cambios respecto a la suma de todos los cambios etiquetados (ya sean cambios o no, es decir, falsos positivos). Por otra parte, la sensibilidad o *recall* está midiendo cuantos cambios buenos ha etiquetado la red dentro de todos los cambios buenos presentes en la base de datos.

La curva *Precision* vs *Recall* refleja ambos valores para un umbral de decisión dado. El **umbral de decisión** simplemente es un parámetro que indica si la clasificación de la clase es más o menos restrictiva: **Cuanto más alto sea**, más restrictiva será la elección de las muestras (por tanto la precisión será muy buena dado que la mayoría de las muestras serán buenas), mientras que la *recall* será mala porque esa restricción implica coger pocas muestras y lo más seguro es que hubiera muchas más muestras buenas fuera del subconjunto elegido. Al contrario, **si se va disminuyendo la restrictividad del clasificador**, es decir, el umbral de decisión, el número de muestras aumentará y potencialmente la *recall* porque se tomarán más muestras buenas del total de buenas total, mientras que la precisión disminuye porque al estar menos acotada la decisión es más probable que haya más cambios tomados que realmente no lo son (falsos positivos).

En las tres gráficas siguientes (7.15, 7.16 7.17)y se comparan **descriptores *handcrafted* como dense SIFT, DAISY o DASC**, **la red preentrenada del paper de Sakurada-Okatani** basada en superpixelación y la red deconvolucional elaborada en este trabajo **CDNet:**

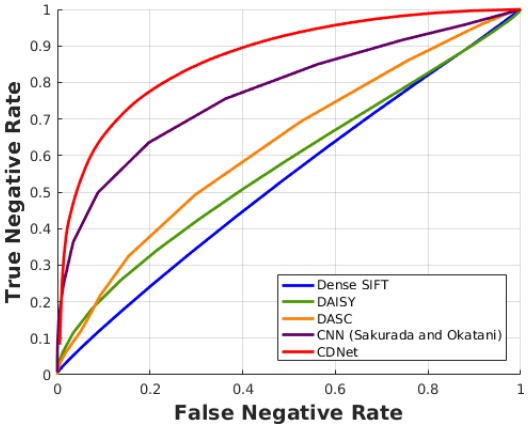


Figura 7.15: TNR vs FNR

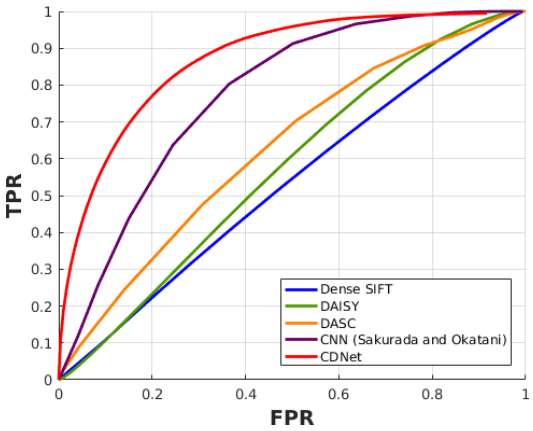


Figura 7.16: TPR vs FPR

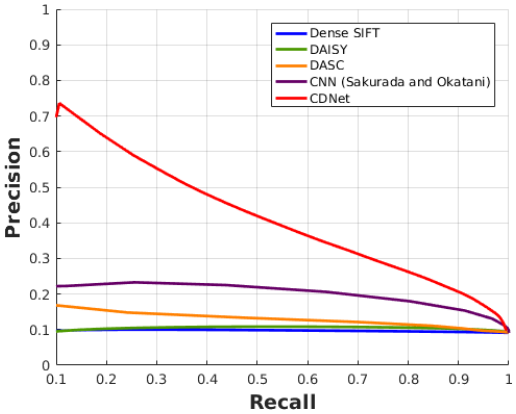


Figura 7.17: *Precision* vs *Recall*

El análisis de la curva 7.17 es muy simple y está basado en la elección de un umbral de decisión en específico para el clasificar (en este caso la DNN): Si se quiere una aplicación en la que el usuario quiere, con la mayor seguridad posible, que su etiqueta sea cierta, deberá escoger un **valor del umbral de decisión alto** (Figura 7.18) para obtener una alta precisión a riesgo de que se obtengan pocas muestras buenas del conjunto de buenas total (*recall* baja).

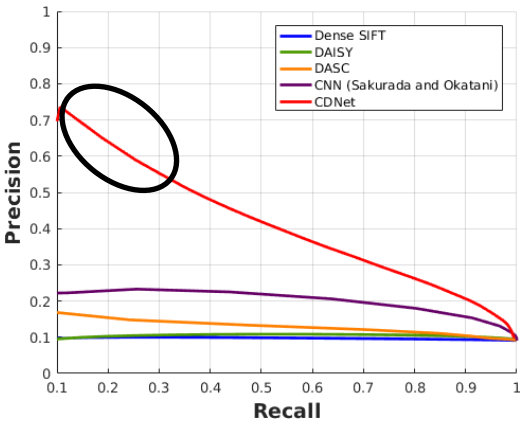


Figura 7.18: Umbral de decisión alto

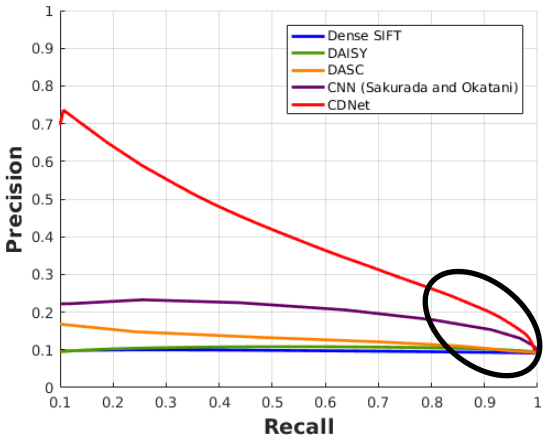


Figura 7.19: Umbral de decisión bajo

Por otra parte, si la aplicación requerida debe etiquetar el número máximo de imágenes, a riesgo de que en algunas la precisión del etiquetado sea mala (ya sea etiquetando cuando no hay, o etiquetando cuando sí lo hay), **el umbral de decisión será bajo** (Figura 7.19).

Siguiendo tales ejemplos, el parámetro *False Positve Rate* (FPR) usado para la detección de cambios en la DNN equivale a 10 % (0.1 en escala de 1).

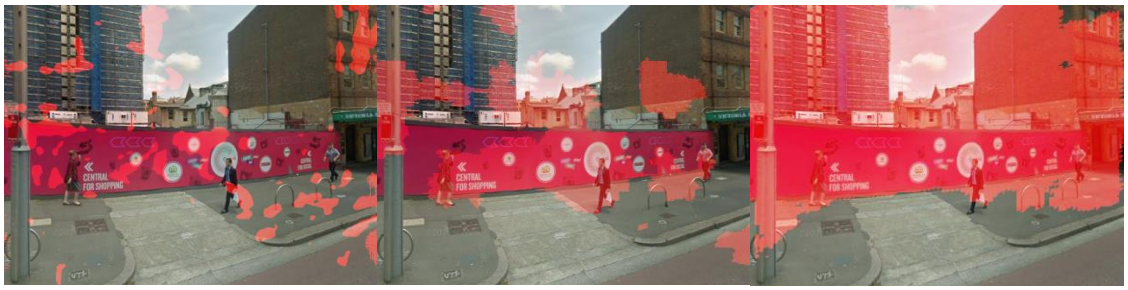
B. Comparación cualitativa:

A continuación se muestran trece ejemplos aleatorios de los 844 pares que se han obtenido, diez de ellos con detección de cambios y los tres últimos para comprobar como la CNN no debe etiquetar aquellas secciones que no presenten cambios entre sí, o equivalentemente, su *groundtruth* sea totalmente negro:

Dentro de cada ejemplo:

- **Fila superior, de izquierda a derecha:** Sección del primer panorama, sección del segundo panorama con antioffset y groundtruth manual elaborado por el usuario de la herramienta.
- **Fila inferior, de izquierda a derecha:** Resultado del autoetiquetado para el descriptor *handcrafted* DAISY, la CNN preentrenada de Sakurada-Otakani y DNN de este TFG.

#1



#2

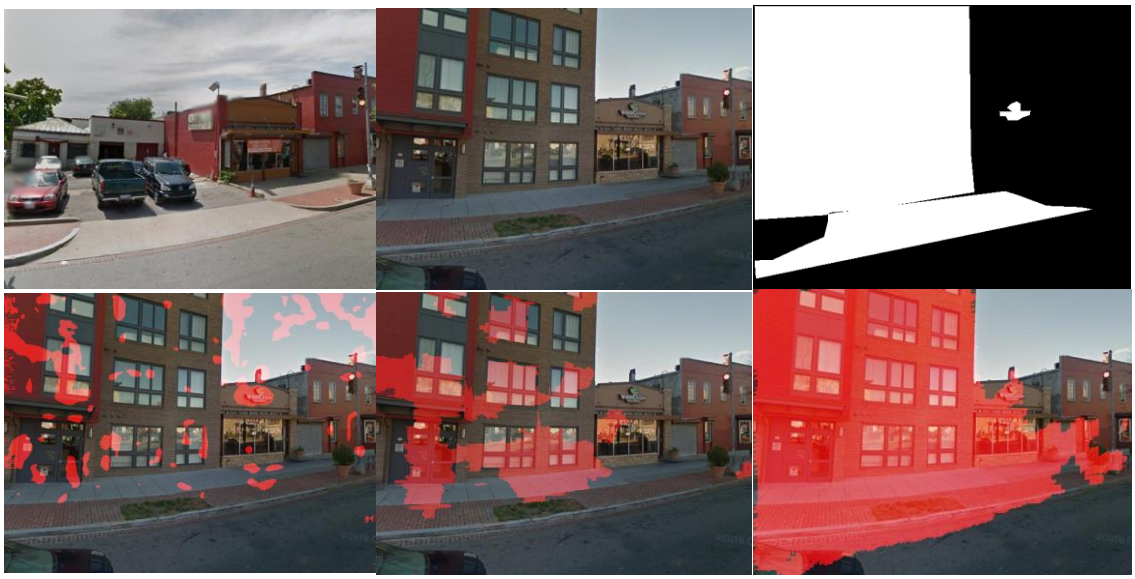
#3

#4

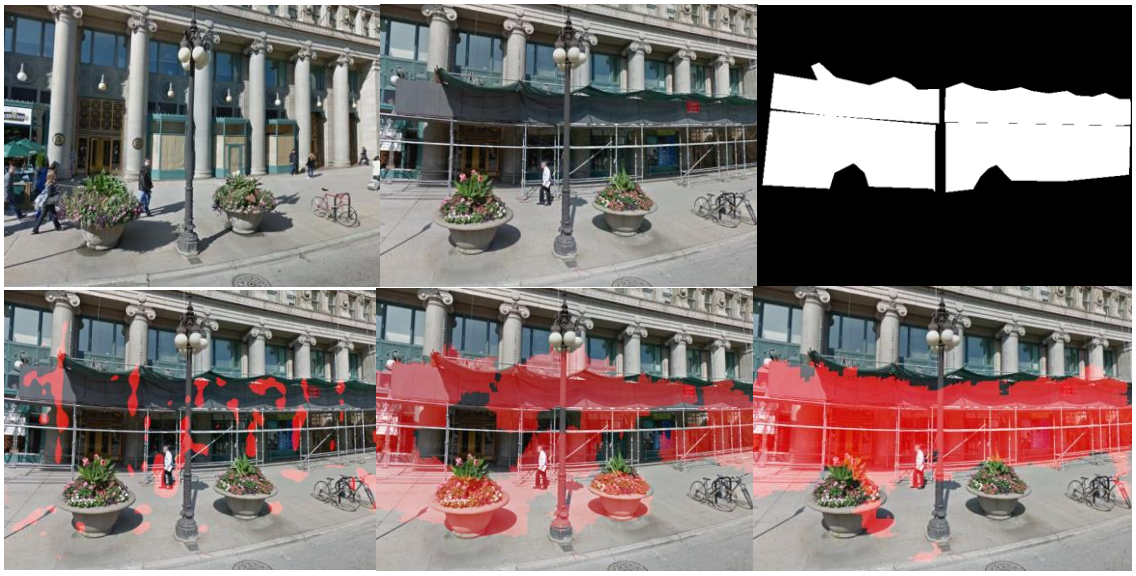
#5

#6

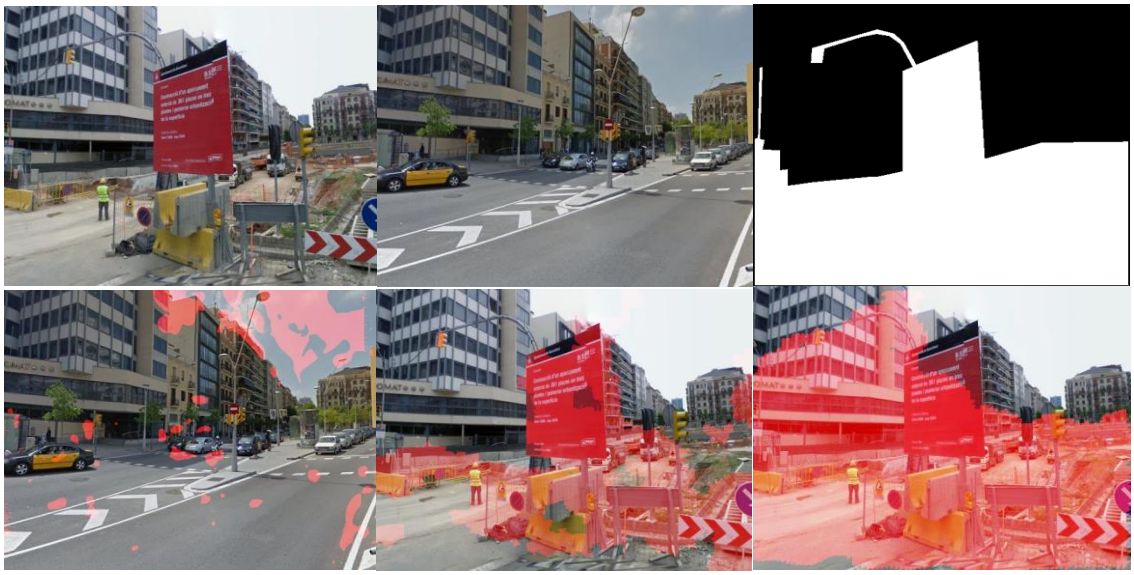
#7



#8



#9

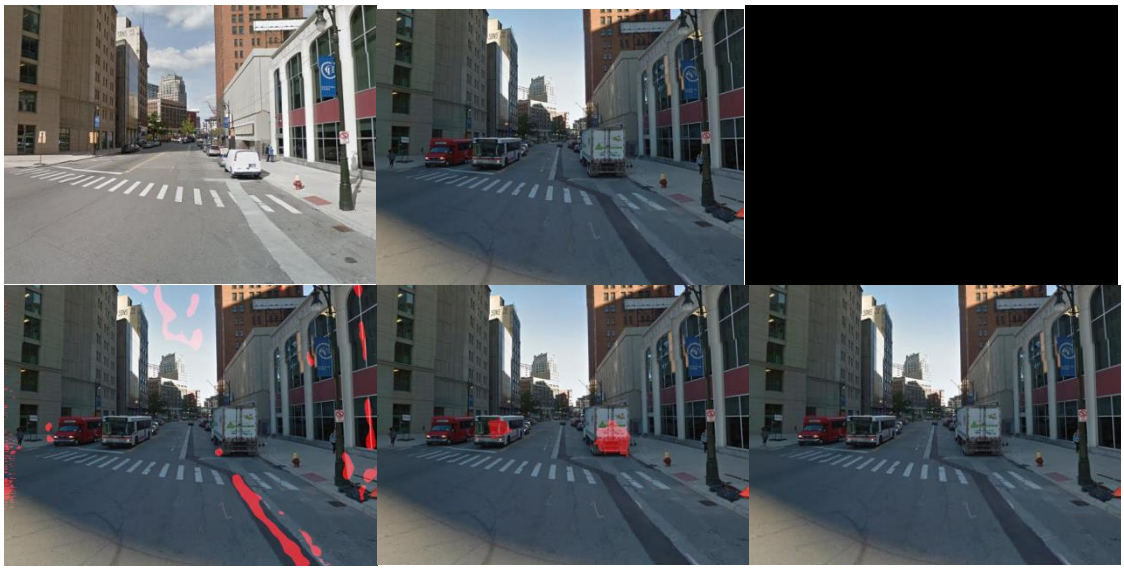


#10

#11

#12

#13



A pesar de que algunas secciones (dos primeras imágenes de cada fila superior) no están perfectamente totalmente alineadas (a diferencia de otros dataset), se puede observar cómo los resultados del autoetiquetado son mucho mejores en la DNN realizada en este trabajo (con la comentada arquitectura de contracción-expansión, destacando los ejemplos #3, #7, #8 y #10 con un etiquetado casi perfecto por parte de la DNN) que el uso de descriptores manuales (de hecho, siendo DAISY el mejor dentro del

conjunto DAISY, DASC y dense SIFT, el resultado es ciertamente malo en comparación) y la CNN basada en superpixelación de Sakurada-Okatani. Por otra parte, los tres últimos ejemplos también demuestran la evidencia de que la DNN es mucho mejor no etiquetando ante presencia de no cambio, alcanzando la perfección en muchos casos, destacando los ejemplos #11 y #13, donde la DNN ha identificado perfectamente la no presencia de cambios y por tanto no se observa ningún *highlight* en rojo destacando dicho cambio, como si ocurre en las otras dos técnicas.

Capítulo 8

Conclusiones y trabajos futuros

Este trabajo se ha basado en el **desarrollo de una aplicación lo más robusta y eficiente posible mediante el entorno de desarrollo de MatLab** que consiste en la creación de las imágenes panorámicas, mapas de profundidad, síntesis de vistas, extracción de características de esas vistas, alineamiento y finalmente un etiquetado de sus diferencias (*groundtruth*). La **herramienta ha sido realizada íntegramente por el autor de este trabajo**, si bien ciertos códigos, como por ejemplo el que genera los mapas de profundidad, la síntesis de vistas o el etiquetado, provienen de otros autores, siendo convenientemente citados y su código mejorados y optimizados, manteniendo la base de su trabajo.

Adicionalmente a la creación de esta herramienta se crearon **dos bases de datos cuyo contenido son geolocalizaciones ordenadas**, así como su posterior procesamiento con la herramienta. Los principales extractores de características para la tarea del alineamiento entre vistas han sido el detector de esquinas de Harris, la técnica SURF y la técnica SIFT, proporcionando estas dos últimas un excelente rendimiento.

Se puede decir que los **datos obtenidos de la herramienta son altamente satifactorios**, pues a pesar de las diferencias en luminosidad o la diferencia en pose de la cámara acoplada en el coche de Google Street View, se consiguieron **909 pares de secciones de los 1062 máximos posibles**, es decir, una eficiencia del 85.6 %. Sin embargo, muchos pares de secciones, alrededor de una tercera parte tenían un marcado *ghosting* (o solapamiento entre el extremo derecho e izquierdo del panorama) por lo que fueron descartados para ser datos de entrenamiento de la red neuronal. Por otro lado, **el desalineamiento entre imágenes es inherente al trabajo a pie de calle**, por tanto, la red neuronal sufre de forma "moderada" este desalineamiento y produce ciertos fallos.

Respecto a la red neuronal, se trabajó con una **red neuronal deconvolucional** DNN cuyo objetivo es la **discriminación entre los verdaderos cambios estructurales y el ruido causado por las diferencias en iluminación, vegetación**, entre otras cosas. Lo bueno de tener a disposición Google Street View es que las bases de datos pueden ser mucho mayores a las bases de datos usadas por otros autores en este ámbito, que hasta el momento siempre tomaban las imágenes en estático en vez de *online*, además de presentar datos desafiantes con diferencia en pose, etc, que realmente presenta una ventaja al realimentar el concepto de DNN y mejorarla.

Los futuros trabajos para mejorar este proyecto serían los siguientes:

1. Obtención de **imagenes panorámicas sin solapamiento de extremos.**

2. Mejora de la estructura planar del mapa de profundidad.

3. Mejora de la síntesis de imágenes incorporando la correción de pose de la cámara en traslación.

4. **Mejora del código de extracción de carácterísticas**, sobre todo en el ámbito de la correción de *outliers*.

5. Posibilidad abierta de **segmentar por clases** (grountruth múltiple) en vez de etiquetar todo bajo la misma clase (etiquetado binario).

6. Mejorar el alineamiento de las imágenes, bien con la herramienta de MatLab, bien desarrollando una nueva arquitectura de red deconvolucional capaz de alinearlas (más complejo), bien uniendo ambas

filosofías (más complejo aún). Con las imágenes perfectamente alineadas los resultados de la detección serían mucho mejores.

7. Aumento de la base de datos de geolocalizaciones a por lo menos una decena de millar para suprimir en la medida de lo posible el ruido introducido en la CNN como causa de los "poco" datos manejados hasta el momento (alrededor de 1.000).

Anexo I

Creación de la herramienta de análisis

Este anexo está íntegramente dedicado al proceso de creación de la herramienta. El entorno donde se ha creado la totalidad de la aplicación ha sido MatLab, así como se ha hecho un uso extensivo de varias toolboxes suyas (y dos externas) y del editor de interfaces GUI.

A. MatLab:

MatLab (acrónimo ingés de MATrix LABoratory, Laboratorio de matrices en español) es una herramienta de **software matemático** con un lenguaje de programación propio (denominado lenguaje M, de ahí que las extensiones de todas las funciones, scripts e interfaces creadas sea .m) además de ofrecer un IDE. Un IDE (Integrated Development Rnvironment, o Entorno de desarrollo inegrado) es una aplicación informática que proporciona servicios al programador para desarrollar software. Algunos IDE contienen su propio compilador, como puede ser el IDE de MatLab basado en en el compilador MEX, el IDE NetBeans, especializado en Java o el IDE Eclipse, especializado en C++).

MatLab es un producto distribuido por la compañía **The MathWorks Inc**. [24], empresa privada estadounidense especializada en las herramientas de software matemático, donde sus dos productos estrella son MatLab y Simulink, ambos enlazados. MatLab está disponible para las principales plataformas como Windows, MAC OS X o GNU/Linux (esta última ha sido la plataforma utilizada dada la limitación de la librería YAEL por tener únicamente compatibilidad con Linux).

Su característica principal (de ahí el nombre) es la expresión de todos sus cálculos en forma de matrices, por lo que en aquellas aplicaciones donde los cálculos principales estén basados en matrices, MatLab será la herramienta natural. Por ello, es ampliamente utilizada por personas del ámbito científico/ingenieril.

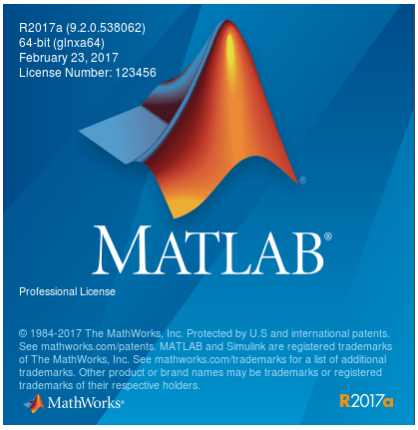


Figura I.1: Logo MatLab

MatLab cuenta con múltiples prestaciones: Aparte de la **manipulación de matrices**, es capaz de representar datos y funciones mediante **plots o gráficos**, lo que facilita su análisis y permite una mayor comprensión dada la visualización de estos mismos. Por otra parte, también se permite el **desarrollo de algoritmos** (en forma de scrips y funciones) e **incluso aplicaciones** (con las interfaces GUI) que pueden integrarse en otros lenguajes (de hecho el lenguaje M está basado en Java).

Este producto contiene una gran libería de ***toolboxes*** (en términos informáticos una *toolbox* es una librería, es decir, un conjunto de funciones con un propósito en específico que permiten que este software pueda emplearse en cientos de aplicaciones):

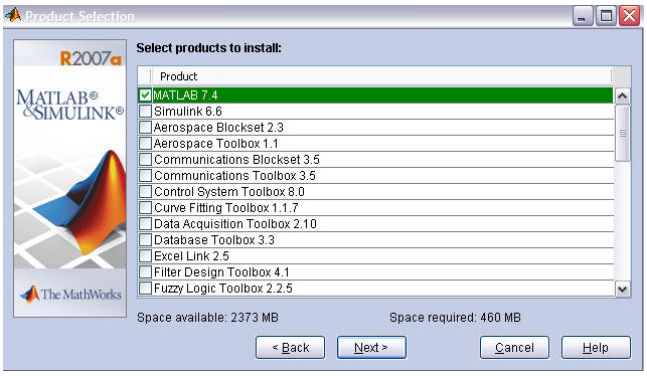


Figura I.2: Opción para instalar o no las toolboxes en el momento de instalación de MatLab

Sin embargo, MatLab no es estático y permite la **incorporación y/o actualización de toolboxes** previas sin necesidad de reinstalar de nuevo el producto.

Además, existen dos herramientas preinstaladas:

- ***Simulink*: Plataforma que permite la **simulación dinámica de sistemas** mediante diagramas de bloques. Es posible añadir otros paquetes de bloques (al igual que se podía añadir *toolboxes*) para ampliar sus posibilidades.
- ***Guide*: Editor de interfaces de usuario.** Es muy útil para el desarrollo de aplicaciones y será el principal paquete empleado en este TFG. Una interfaz gráfica de usuario, conocida también como **GUI** (del inglés graphical user interface), es un programa informático que actúa de interfaz de usuario, utilizando un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz. Su principal uso, consiste en **proporcionar un entorno visual sencillo para permitir la comunicación entre el usuario y el sistema operativo de una máquina o computador**.

A.1. Toolboxes empleadas:

Estas toolboxes incluyen funciones específicas dentro de diversos campos. En el caso de este TFG se han utilizado fundamentalmente tres toolboxes (dos internas a MatLab y una externa) además de dos externas adicionales para el cómputo del mapa de profundidad y síntesis de imágenes respectivamente. Las toolboxes internas han sido las siguientes:

- ***Computer vision toolbox*** (Librería de visión artificial): Conjunto de funciones, algoritmos y aplicaciones para el **diseño y simulación de sistemas de visión artificial y procesamiento de vídeo**. Esta toolbox permite, entre otras cosas, el reconocimiento de objetos, calibración de cámara, reconstrucción 3D o procesamiento de nubes de puntos. En el caso de este TFG, el principal uso ha sido la extracción de características para el alineamiento de imágenes. Las principales funciones usadas se describen brevemente a continuación:

[points] = detectHarrisFeatures(I): Detecta los puntos característicos a partir del algoritmo de Harris con la imagen en escala de grises.

[points] = detectSURFFeatures(I): Detecta los puntos característicos a partir de la técnica SURF con la imagen en escala de grises.

[features,validPoints] = extractFeatures(I,points): Obtiene los vectores de características (descriptores) a partir de una imagen I en escala de grises y los puntos obtenidos en la detección.

[indexPairs] = matchFeatures(features1,features2): Correspondencia entre los descriptores de dos imágenes (features 1 respecto de la imagen 1 y features 2 respecto de la imagen 2). IndexPairs contiene los índice de los pares buenos de keypoints.

[tform,inlierpoints1,inlierpoints2]=estimateGeometricTransform(matchedPoints1,matchedPoints2,transformType): Estima la matriz de transformación y los inliers definitivos a partir de las teóricas correspondencias matchedPoints1(2) y un tipo de transformación (similarity, affine o projective). Está basada en el filtro MSAC para el descarte de *outliers*.

showMatchedFeatures(imagen1, imagen2, matchedfeatures1, matchedfeatures2, method): Muestra las correspondencias entre imágenes, con círculos rojos los *inliers* de la primera sección y con cruces verdes los *inliers* de la segunda sección. El method (método) usado en este TFG ha sido *montage* , esto es, colocar una a continuación de la otra en la horizontal.

[B]=imwarp(A,tform): Aplica la matriz de transformación calculada con la función *estimateGeometricTransform* a la matriz A (en este caso sería la segunda sección) para obtener la matriz B (en este caso la segunda sección referenciada a la primera).

overlay=imshowpair(im1, im_aligned, method, 'Colorchannels',choice): Crea un overlay o superposición entre la sección 1 y la sección 2 tras aplicarle la matriz de transformación. El método usado en este caso es *falsecolor,* el cual crea dicho overlay con un modelo RGB, siendo los canales disponibles red-cyan o Green-magen (respectivamente primera y segunda sección). En general, el overlay red-cyan ofrece una gran vision del resultado para etiquetar.

- ***Image processing toolbox*** (Librería de procesamiento de imágenes): Conjunto de funciones, algoritmos y aplicaciones para el **procesamiento, análisis y visualización de imágenes**. Esta toolbox permite, entre otras cosas, analizarlas, segmentarlas, mejorarlas, etc Las principales funciones usadas se describen brevemente a continuación:

[I]=imread(filename): Guarda en la variable I una matriz correspondiente con la información de la imagen real filename (extensión .png, .jpg, etc…). En caso de estar la imagen original en escala de grises, I será una matriz M x N (anchora y altura en píxeles respectivamente), mientras que si presenta color, la matriz adoptará el modelo RGB, siendo M x N x 3, una matriz para cada plano Rojo Verde y Azul.

Imshow(I), image(I): Son funciones recíprocas a imread. Muestra por pantalla la matriz obtenida con *imread* de forma que se pueda visualizar.

B=rgb2gray(A): Convierte la imagen original en modelo RGB a escala de grises. Elimina la información del tono y la saturación pero conserva la luminancia.

imwrite(data, folder): Guarda el archivo data, el cual es una matriz de datos (obtenida a partir de imread u otras operaciones), dentro del directorio especificado por folder con su extensión pertinente (por ejemplo .jpg o .png)

Por otra parte, la principal toolbox externa es ***VLFEAT***. Esta es una librería de código abierto muy popular dentro de los algortimos de visión artificial, especializada en el **análisis de imágenes y la extracción de características y alineamiento de estas mismas**. Está escrito en C por eficieincia y

compatabilidad, adaptada a lenguaje M para facilitar su uso. Está soportada por las plataformas Windows, MAC OS X y Linux. Las principales funciones de esta *toolbox* han sido referentes al extractor de características SIFT:

[F,D]=vl_sift(I): Extrae las características, las localiza, les asigna orientación y finalmente computa los descriptores SIFT. Cada columna de D es un descriptor correspondiente a un marco o frame (representa un objeto geométrico, como puede ser un punto, círculo o elipse que representa la localización y forma de uno de estos keypoints) de F. El descriptor es un vector fila de 128 elementos.

[matches]=vl_ubcmatch(D1, D2): Devuelve las correspondencias entre *keypoints* teniendo como argumento de entrada los descriptores respectivos de cada imagen (D1 y D2) obtenidos con la función *vl_sift*.

No se detallarán las otras dos *toolboxes* externas puesto que el autor de este TFG no ha hecho uso de ellas, si bien son requeridas en ciertas partes del código:

- Para la descompresión del archivo .json que proporciona la API de GSV en lo referente al mapa de profundidad se utiliza la librería **JSONLAB**. JSONLAB es una toolbox especializada en convertir estructuras de datos .m (es decir, de MatLab, como por ejemplo un array o una estructura) en formatos cadena .json / .ubjson, o al revés (como es este caso), decodificar un archivo .json / .ubson en una estructura de datos en MatLab.
- Para la síntesis de imágenes se utiliza la librería **YAEL**, incorporada en el código del paper 24/7 Tokyo. Esta librería está especializada en operaciones de kmeans o búsqueda del vecino más cercano, usada en el apartado de la síntesis. Sin embargo, actualmente este paquete no es compatible con Windows, sino sólo para Linux y MAC OS X, de ahí que el desarrollo de la herramienta y la obtención de resultados se hayan tenido que hacer en Ubuntu 16.04 (por comodidad, el autor de este TFG incorporó un dual boot a su computadora para poder tener tanto Windows 8.1 como Ubuntu 16.04).

Finalmente, se explican otras funciones importantes de carácter general:

[num, txt, raw]=xlsrread(Excel_file, rango_lectura): Obtiene la información de un documento excel (extensión .xlsx) con un rango de lectura determina (por ejemplo el recuadro A13:D21), almacenando en num los valores númericos, en txt las cadenas de texto y en raw ambas cosas. La lectura general realmente es raw, escribiendo en la matriz resultante en aquellos huecos donde no haya ni número ni texto un NaN (Not a number = desconocido).

[B,k] = sort(A): Ordena la matriz A de mayor a menos (la matriz A contiene formatos numéricos únicamente), dando lugar a la matriz ordenada B y las posiciones de donde estaban originalmente esos números, almacenadas en k.

str=sprintf(formatSpec, A1, …., An): Convierte los datos en el formato especificado por formatSpec. Por ejemplo: sprintf('A13:L%d', length(raw)-1)).

A=lenght(B): Determinar la longitud de la dimensión más larga de la matriz B;

[filas, columnas, profundidad, etc…]=size(B): Determina las dimensiones de la matriz B. Los argumentos de salida dependerán del número de dimensiones con que cuente la matriz.

Str=strjoin(C): Construye un único texto, str, uniendo cada elemento del array C el cual cuenta con un único espaciado. El dato C puede ser una celda que representa un vector de caracteres o un dato en formato cadena.

tf = strcmp(s1,s2): Compara dos cadenas, devolviendo un 1 si son idénticas y un 0 en caso contrario.

Estructura try/catch I.3: Esta estructura ha sido enormemente útil en este TFG. Cuenta con la siguiente estructura:

```
try
        statements
catch Me
        statements
end
```

En primer lugar se ejecuta las sentencias entre try y catch. Si no hay ningún error, se ejecutan y termina la función. En caso de que ocurra algún error, se ejecutan las sentencias entre catch Me y end. Si no hay ningún error, se ejecutan y termina la función, pero en caso haber error, la excepción Me almacenará la causa del fallo para que el usuario la pueda utilizar. Es lo que se denomina un "chivato" de código para localizar errores y además bifurcar códigos.

Al igual que las sentencias for e if se pueden anidar varios try catch.

Figura I.3: Estructura *try-catch*

B=str2num(A): Convierte el formato cadena (string) de A a formato numérico de tipo doublé (doble precisión). Observación: str2num no opera sobre arrays de tipo cadena o arrays de celdas. Para convertir un array de cadenas o una array de celdas con vectores de caracteres, usar B=str2double(A).

K=strfind(str, pattern): Almacena en un vector k las posiciones de los caracteres que cumplen con un determinado patrón. Por ejemplo, cuando se disecciona la URL de GSV, angulos1 = strfind(pan1,'@'), en angulos1 se almacena la posición del carácter dentro de la cadena pan1 que se identifica con el carácter @.

isempty(A): Devuelve un 1 si la matriz tiene todos sus elementos a 0 o bien devuelve un 0 si existe al menos un elemento distinto de 0.

A.2. Clases en MatLab:

MatLab está basada en la **programación con objetos**. Este tipo de programación a su vez está basado en una combinación de datos y métodos (acciones) desembocando en estructuras lógicas (objetos) cuyo objetivo es el aumento de la capacidad para manejar software complejo, esencial en el desarrollo de aplicaciones y grandes estructuras de datos.

El uso de objetos en MatLab sirve para elaborar aplicaciones complejas con mayor rapidez que otros lenguajes como C++ o Java (entre otras cosas porque el lenguaje .m es más sencillo y flexible que los dos anteriormente citados). La definición de clases permitirá reutilizar código, encapsulamiento, herencia y comportamiento de referencia sin prestar atención a tareas de bajo nivel como en otros lenguajes.

Para la programación de objetos con MatLab es requerido:

- **Archivos de definición de clases:** Permiten la definición de propiedades, métodos y eventos.
  - Propiedades: Posibilita el almacenamiento de datos para posteriormente emplearlo en una clase.
  - Métodos: Funciones que implementan operaciones dentro de la clase.
  - Eventos: Mensajes definidos por las clases ejecutados cuando se produce una acción.
- **Clases con comportamiento de referencia:** Ayudan a la creación de estructuras de datos como listas vinculadas.
- **Eventos y oyentes:** Permiten monitorizar estas acciones y cambios acerca de las propiedades de los objetos.

- Oyentes: Objetos que responden ante un evento ejecutando una función de devolución de llamada.

Tras explicar los requisitos básicos para la programación orientada a objetos con MatLab, se dará paso a la descripción de una interfaz de usuario y proceso de creación.

B. Interfaz de usuario GUI:

MatLab incorpora una opción de creación de interfaces gráficas de usuario (*Graphic User Interface*) cuyo objetivo es la **interacción mutua y sencilla entre el usuario y la computadora** para un control sencillo de aplicaciones software mediante menús, botones, listas, *checkboxes* o incluso controles deslizantes, sin la necesidad de escribir en el momento líneas de código o correr manualmente una función desde la ventana de comandos, ya que se puede enlazar más fácilmente la invocación de una función al pulsar un botón, por ejemplo.

Para crear este tipo de interfaces, MatLab dispone del **entorno GUIDE**. Éste es un entorno de programación visual para realizar y ejecutar GUIs, generan de forma automática el código necesario .m para construir la interfaz, siendo el usuario el responsable de rellenar las funciones de cada objeto dentro de la interfaz. Cuando un objeto se llame, es decir, se clicke sobre él (u otra acción), el programa ejecutará el código que está dentro de la *Callback* de dicho objeto (botón, *checkbox*, etc…).

Para **ejecutar este entorno** se puede hacer de dos formas:

1. Tecleando (y ejecutando) en la *command window* el comando *guide*.

2. Ejecutar los siguientes pasos: *Home →New →GUI.*

En ambos casos, se abrirá una ventana que dará opción al usuario de comenzar a trabajar desde un ejemplo, abrir una interfaz existente o crear una desde el principio (opción por defecto).

Tras elegir una de las tres opciones se abrirá un archivo *.fig* (básicamente porque es un figura) donde se podrá empezar a trabajar (pudiendo colocar ejes, botones, etc…).

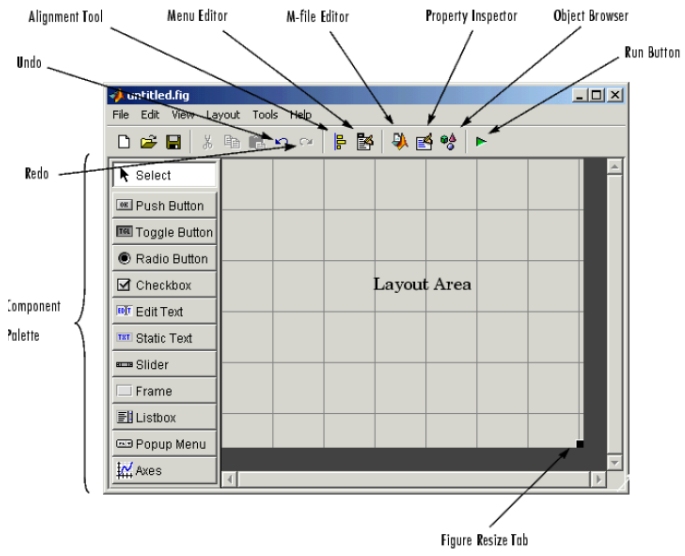


Figura I.4: Archivo *.fig* vacío

Este **área de trabajo** GUIDE puede dividirse en:

1. Barra de herramientas:

Botón	Descripción
Alignment tool	**Ajuste la posición de un objeto respecto a otro (alineamiento tanto vertical como horizontal).**
Menu editor	**Diseño de barras de herramientas en las GUIs (*Menu Bar*) y los menús que aparecen en los objetos (*Context Menus*).**
M-file editor	**Edición del código creado por la interfaz en un archivo .m.**
Propoerty inspector	**Asignación y modificación de las propiedades de cada objeto.**
Object browser	**Muestra los objetos que se encuentran en la figura en forma de árbol, permitiendo su selección.**
Run button	**Creación de la GUI y ejecución.**

Tabla I.1: Barra de herramientas GUIDE

2. Paleta de componentes: Componentes que se pueden agregar al área de trabajo *.fig* . Estos reciben el nombre de *uicontrols.*

Tipo	Descripción
Push button	**Al pulsarlo lanza su *Callback***
Toggle button	**Conmuta del estado on al off y viceversa.**
Radio button	**Indica una opción que puede seleccionarse o no.**
Check box	**Asignación y modificación de las propiedades de cada objeto.**
Edit text	**Muestra los objetos que se encuentran en la figura en forma de árbol, permitiendo su selección.**
Static text	**Muestra un texto estático, sin posibilidad de modificación.**
Slider	**Representa un rango de valores desde un máximo a un mínimo**
List box	**Lista deslizable con varias opciones a elegir.**
Axes	**Permite insertar ejes donde se podrán mostrar posteriormente figuras y/o funciones en la interfaz.**
Pop-up menu	**Crea un menú desplegable con diferentes opciones.**

Tabla I.2: Paleta de componentes GUIDE

En todos los ***uicontrol***, tras hacer su correspondiene acción (clickar, escribir y presionar enter, arrastrar, etc…) se **ejecutará el código de su *Callback* correspondiente en el .m**. Al clickar sobre uno de estos controles con *click derecho* se puede acceder a su conjunto de opciones, donde modificar su etiqueta, color, etc…. Además de ver su función asociada en ***View Callbacks***, donde se puede ver el código que se ajecutará al cumplir con su "función de activación".

Es importante tener en cuenta que todos los valores de las variables de la interfaz GUI se almacenan en una **estructura general** identificada con ***handles***. Por ejemplo, si se crea la variable bandera dentro del proceso y toma el valor 1, dentro de la estructura habrá un campo que sea *handles.bandera=1.* Para asegurar que cualquier cambio o asignación de propiedades o variables quedan guardados, se debe usar la sentencia *guidata*, generada automáticamente por la herramienta al final de cada *Callback:*

```
97    handles.output = hObject;
98
99    % Update handles structure
100   guidata(hObject, handles);
```

Para finalizar la introducción a las interfaces GUI, para **asignar** u **obtener valores de estos controles**, por ejemplo el índice de la ciudad que se ha clickado en la lista de la interfaz *Datachoice.m* (se verá a continuación en el manual de usuario), se utilizan las sentencias *get* y *set:*

- ***set(id, 'propiedad', 'valor'):*** Establece un nuevo 'valor' para la 'propiedad' del objeto identificado con id.
- ***get(id, 'propiedad'):*** Obtiene el valor de la propiedad seleccionada en el objeto identificado con id.

Anexo II

Manual de usuario:

En este anexo se va a detallar el **funcionamiento del sistema a través de la GUI desarrollada**, mostrando las posibilidades y funcionalidad de cada elemento de interacción que compone la aplicación general. Además, en cada paso se explicará como proceder con cada botón de cada interfaz y lo que genera en el momento.

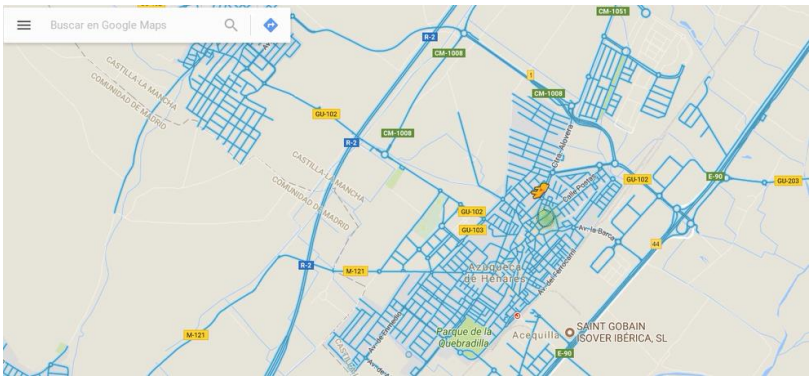
Por motivos de extensión, no se expondrá todo el código desarrollado por el autor, aunque se han ido mostrando ciertos códigos de interés a lo largo del presente TFG. La aplicacación ha sido íntegramente creada por el autor de este TFG, en su gran mayoría elaborando código él mismo y en menor medida, aunque no por por ello menos importante, mejorando o incorporando códigos de trabajos previos. La aplicación cuenta con un total de 4.584 líneas, distribuidas en doce funciones, dentro de las cuales cinco de ellas son las interfaces gráficas GUI con las que el usuario puede interaccionar. Estas son, en orden de aparición:

- ***Inicio_CGH***
- ***DataChoice_CGH***
- ***Initial Information_CGH***
- ***Recommended_Alignment_CGH***
- ***Matching_CGH***

Aunque muchos de los procesos (e imágenes) que se verán a continuación ya han sido reproducidos a lo largo del trabajo, para elaborar un manual de usuario no hay nada mejor que mostrar un análisis entero, sin interrupciones, **desde la elección de la geolocalización a analizar en la interfaz hasta la elaboración del groundtruth**, con sus variantes y problemas, sin entrar en procesos físico matemáticos o conclusiones que ya se sacaron en capítulos anteriores.

1. En primer lugar para llevar a cabo un análisis, el usuario debe contar con una conexión estable a Internet. Posteriormente clickará en la barra del buscador, escribirá ***Google Maps*** y se introducirá en la página.

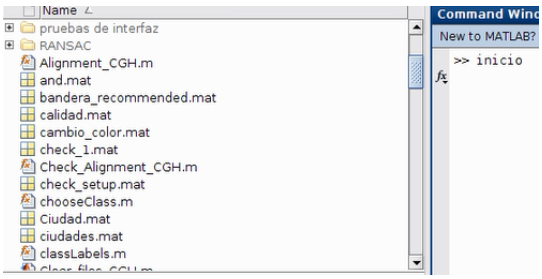
2. A continuación, tiene dos posibilidades: Bien **escribiendo la zona** de donde se quiere buscar una geolocalización, bien **arrastrando al muñeco PEGMAN** hasta una zona, siempre y cuando esté en azul que indica que al mismo hay una pasada del coche de GSV.



3. Ya estando dentro del servicio a pie de calle, se debe **comprobar que en la esquina superior izquierda aparece un relojito** indicando que en esta geolocalización hay dos o más pasadas. En caso contrario, se deberá buscar otra localización.

4. Se **buscarán dentro de la máquina del tiempo** (desplegable que se abre al pulsar sobre el reloj) **dos localizaciones con aparentemente mínimo cambio en rotación y traslación del punto de vista**, poca vegetación, pocos transeuntes, pocos vehículos, etc Se extraen ambas URLs de Internet y se almacenan en la base de datos de Excel. De izquierda a derecha, para cada línea de análisis se deberá escribir la zona concreta de cambio (por ejemplo calle y número), ciudad, país, mes y año del primer panorama, mes y año del segundo panorama, URL1, URL2, máxima profundidad, tipo de cambio y si se quiere el futuro panorama en alta o baja calidad.

5. Con el análisis dentro del Excel, el usuario arrancará la aplicación **MatLab**. EL usuario se deberá colocar en el directorio donde se encuentran todos los ***scripts***, funciones e interfaces correspondientes a la aplicación. Escribirá ***inicio*** en la ventana de comandos o bien la correrá con el botón play estando en ese momento en su código:



A partir de este momento hasta el cierre de la aplicación, todo funcionará a base de interfaces.

6. En la pantalla de *inicio* hay cinco botones:

Figura II.1: Interfaz *Inicio*

El botón **Warning** proporciona las citaciones requeridas si se usa este código:

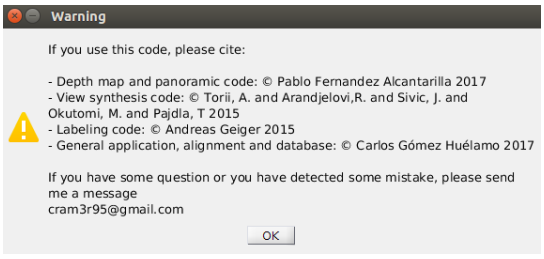


Figura II.2: Botón *Warning*

El botón **Information** muestra los objetivos que persigue la herramienta (no el TFG en su conjunto pues son aún más profundos):

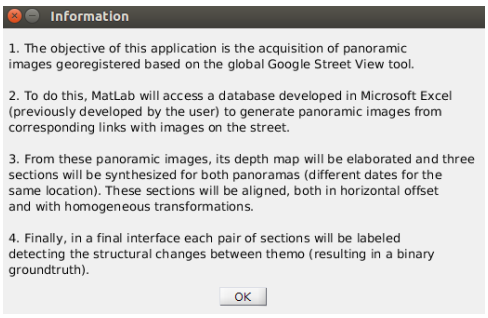


Figura II.3: Botón *information*

El botón **Requirements** indica los requisitos de software que debe cumplir el sistema:

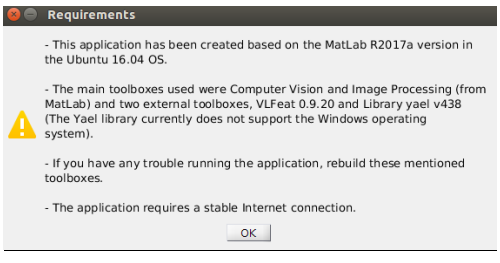


Figura II.4: Botón *Requirements*

El botón **Help** muestra un flujograma acerca de como proceder en la aplicación. Obviamente, el flujograma empieza donde pone START.

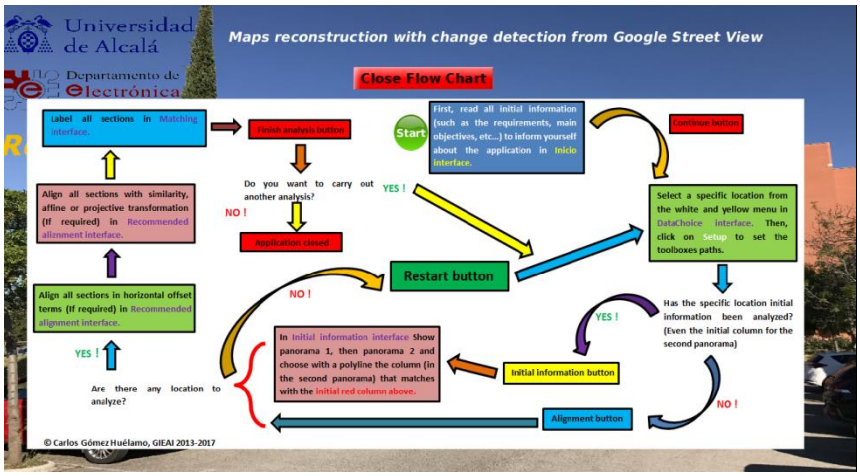


Figura II.5: Flujograma de la aplicación

Finalmente, el botón **Continue** da paso a la interfaz de elección de geolocalizaciones.

7. En la interfaz de elecciones de localizaciones *DataChoice* se deberá **clickar en la primera lista**, donde se encuentran las veinticinco **ciudades más importantes del mundo**, más otra opción de ***Others***. Dentro de cada ciudad habrá una **serie de localizaciones, linkadas al archivo Excel** que debe encontrarse en el mismo directorio que donde se encuentran todas las funciones de MatLab. Clíckese en una de ellas. El programa analizará automáticamente si dicha localización cuenta con su respectiva **información inicial** (mapas de profundiad, imágenes panorámica y demás datos de las estructuras) o no, y si se ha **etiquetado alguna sección de esa localización** (en caso afirmativo, evidentemente implicaría que ese está repitiendo este análisis y la información inicial está disponible).

A continuación, clíckese en el botón de **SETUP** para instalar los directorios pertinentes. Si la información inicial está disponible, se podrá clickar tanto el botón **INITIAL INFORMATION** como el de **ALIGNMENT**. En caso contrario, si el usuario clicka en **ALIGNMENT** sin estar la información inicial disponible, le saldrá un error obliándole a calcular primer la información inicial (**INITIAL INFORMATION**).

Finalmente aparecerá un GIF en el centro de la pantalla inidcando la continuidad del proceso.

OBS: Opcionalmente se incorpora un botón de **RESTART** por si el usuario en algún momento quiere reiniciar la aplicación porque se ha equivocado de localización escogida.

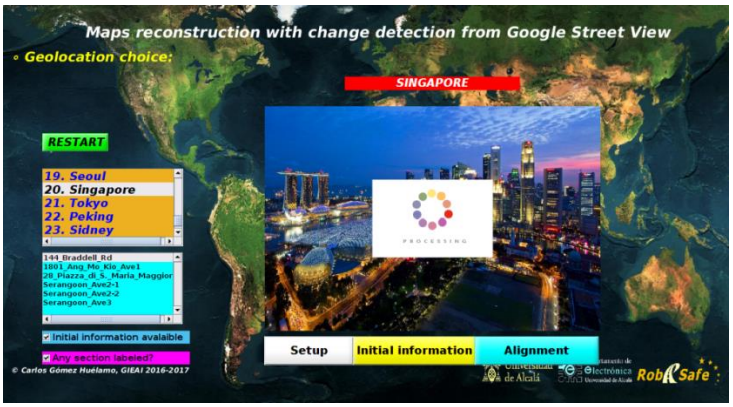


Figura II.6: Interfaz *DataChoice* (arriba, al arranque; abajo tras elegir localización)

(8). Este paso únicamente se ejecutará si se presiona sobre el botón de **INITIAL INFORMATION** dentro de la interfaz *DataChoice*. Aquí se calculan los mapas de profundidad, las imágenes panorámicas y demás parámetros. Además, la interfaz proporciona seis botones, aunque realmente realmente van por pares: **SHOW INFORMATION** (1/2) indica los parámetros de la estructura, tales como los ángulos de inclinación, la latitud o la longitud, **SHOW PANORAMA** (1/2) muestra las imágenes panorámicas y SHOW **DEPTH MAP** muestra los mapas de profundidad.

Figura II.7: Interfaz *Initial information* (arriba, al arranque; abajo tras escoger la columna del segundo panorama para el alineamiento global)

En esta interfaz, el usuario presionará sobre **SHOW PANORAMA 1**, y al presionar sobre **SHOW PANORAMA 2** le aparecerá un cursor para elaborar una polilínea. **Dibújese** a continuación, **en línea vertical lo más recta posible, la columna que corresponde a la información contenida en la columna roja del panorama 1** (el de arriba). Esto permite un **alineamiento global** de las vistas sintetizadas.

9. Primera fase de la realimentación, esto es, **alineamiento individual de offset**. El programa mostrará por defecto las tres mejores correspondencias de *inliers* (puede ser dos, una o ninguna en cuyo caso no se mostraría la interfaz). Supervise manualmente cuales correspondencias son más acordes. **Si la necesidad de correción de offset es imperante** (hay una rotación notable en el punto de vista), clíckese **CALCULATE OFFSET**. Si la diferencia es mínima o ninguna, clíckese **KEEP CURRENT OFFSET**.

Aplíquese esta filosofía a tantas veces como aparezca la interfaz *Recommended alignment* con esa estructura de botones (hasta un máximo de 3).

También presenta un botón de *Restart*.

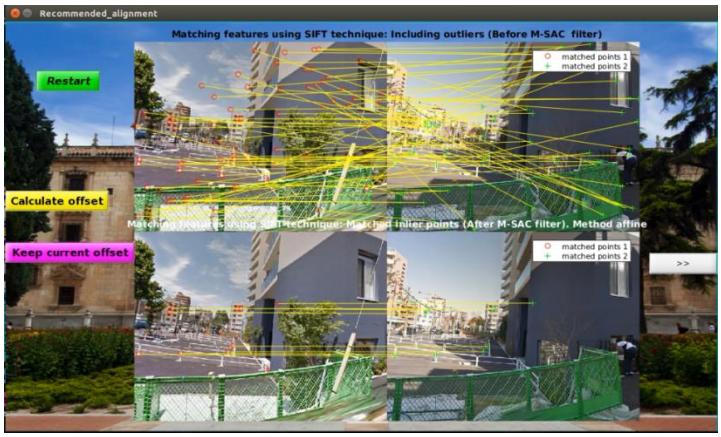


Figura II.8: Alineamiento individual no recomendado

En este ejemplo (Figura II.8) no se debería calcular el offset aunque no sea totalmente perfecto, puede generar una segunda sección peor que la que más que aceptable que había inicialmente.

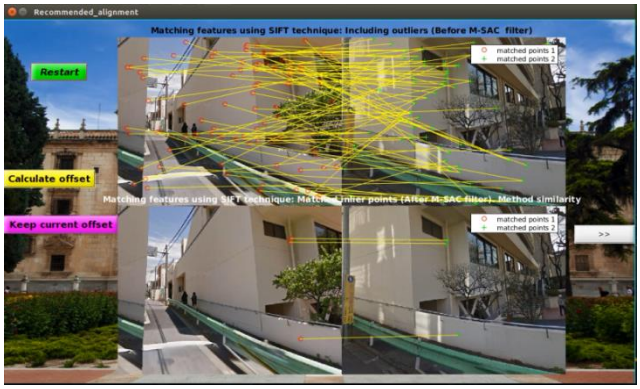


Figura II.9 Alineamiento individual recomendado

Este caso sí sería conveniente aplicar el cálculo del offset

10. Segunda fase de la realimentación, esto es, **alineamiento mediante transformación homogénea**. La interfaz es la misma que el paso anterior, *Recommended alignment*, pero ahora el aspecto cambia. Se puede observar un botón **Check recovered image**, donde se muestra la segunda sección con la transformación homogénea aplicada que pretende alinearla lo más posible a la primera. Cobra enorme importancia en este paso el botón **>>** y **<<**, porque aunque la primera correspondencia es la que más inliers tiene, no tiene porque ser las mejores correspondencias (aunque generalmente sí). Tres ejemplos:

Figura II.10: Transformación homogénea recomendada

Se observa que la mejor transformación es la similarity en este caso a pesar de ser la segunda correspondencia recomendada. **Es por ello que se recomienda el usuario siempre recorrer todas las recomendaciones y contrastarlas.**

En caso de estar de acuerdo con una recomendación, se pulsará sobre el botón **I AGREE** y para continuar con el proceso **CONTINUE.** En caso de querer elegir otra recomendación, **I DO NOT AGREE.**

Pueden existir casos donde todas las **transformaciones recomendadas sean aberraciones**, ya sea por gran distorsión, excesivo escalado, excesiva rotación, etc En ese caso, se elegirá cualquiera de las recomendaciones con I AGREE y posteriormente CONTINUE para proceder en la interfaz siguiente (*matching*) con un etiquetado manual.

Aplíquese esta filosofía tantas veces como aparezca la interfaz Recommended alignment con esa estructura de botones (hasta un máximo de 3).

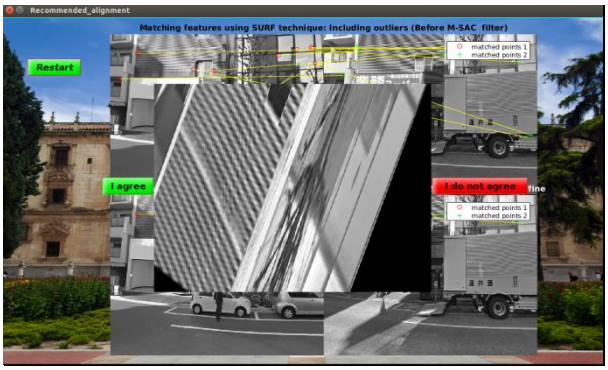


Figura II.11: Ejemplo de aberración

También presenta un botón de *Restart*.

11. Último paso del proceso, aquí se **etiquetarán los cambios** elaborándose el ***groundtruth***. En primer lugar, clíckese sobre el botón **START ANAYLISIS**.

Figura II.12: Interfaz *Matching images*

Aparecerán a la izquierda el ***overlay* (superposición) de la sección 1 con la sección 2** (la cual presenta correción de offset y transformación homogénea), a la derecha arriba la sección 1 y abajo la sección 2 simplemente con la correción de offset. Mediante el botón CHANGE COLOR se puede cambiar el color del overlay.

Figura II.13: A la izquierda, interfaz con *overlay red-cyan;* A la derecha, interfaz con *overlay green-magenta*

A continuación, clíckese arriba **LABELING**. Con esto aparecen tres botones nuevos: Con **ADD LABEL** se permite al usuario etiquetar un cambio, con **REMOVE LABEL** elimarlo y con **SHOW BINARY GROUNDTRUTH** generar un mapa de segmentación a partir de estos polígonos. El código creado en la herramienta está creado de tal forma para que se busque en la base de datos si este *overlay* de secciones fue etiquetado o no. En caso afirmativo, cargará el archivo .mat en el *overlay* mostrando las etiquetas que se hicieron previamente. En caso negativo, no habrá ningún polígono inicial sobre el *overlay*.

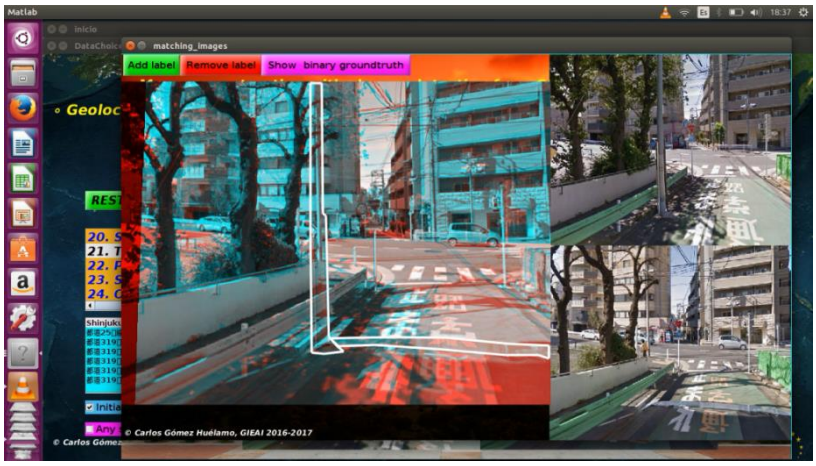


Figura II.14: Ejemplo de etiquetado

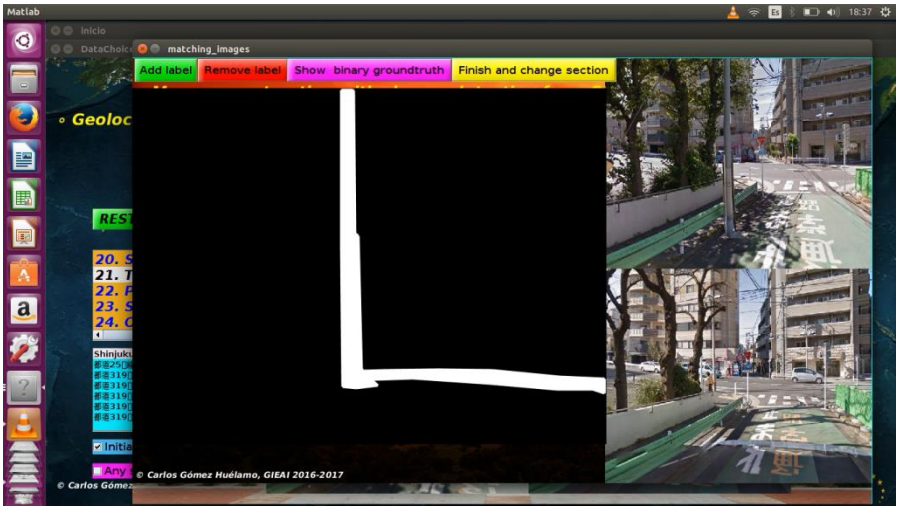


Figura II.15: Ejemplo de *groundtruth*

Tras pulsar en **SHOW BINARY GROUNDTRUTH** aparecerá el botón **FINISH AND CHANGE SECTION** o bien el botón **FINISH ANALYSIS** si únicamente había un par de secciones por etiquetar.

Escójase en la lista amarilla la sección que aun no haya etiquetado (inicialmente está la primera, estando en el mejor de los casos quedarán la segunda y tercera por etiquetar) y repítase este proceso hasta que al pulsar sobre SHOW BINARY GROUNDTRUTH aparezca el botón FINISH ANALYSIS.

12. Al pulsar el último botón **FINISH ANALYSIS** al usuario se le mostrará una pregunta:

***Do you want to carry out another analysis?* (¿Desea llevar a cabo otro análisis?)**

Púlsese **YES en caso de querer hacer otro análisis**, con lo cual **la aplicación se reiniciará**, volviendo a la interfaz *Datachoice*, o **NO en su defecto para cerrar la herramienta.**

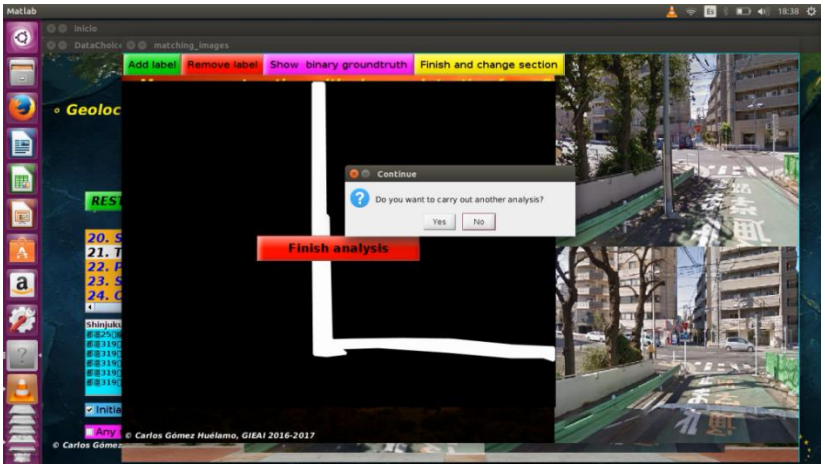


Figura II.16: Final del proceso

Anexo III

Pliego de condiciones:

A. Requisitos:

A.1. Requisitos Hardware:

- Ordenador Asus Intel Core i5-4210U

  - CPU de 1.70 GHz
  - Memoria RAM de 4 GB DDR3 1333 MHz
  - Disco duro de 500 GB
  - Tarjeta gráfica NVIDIA GeForce 320 M con Tecnología CUDA

- Conexión estable a Internet, principalmente en las etapas de búsqueda de geolocalizaciones y procesamiento de estas mismas con la herramienta.

A.2. Requisitos de Software:

- Sistema operativo Linux distribución Ubuntu 16.04

- MatLab R2017a (principalmente las librerías de procesamiento de imagen, *Image Processing*, y la de visión artificial, *Computer Vision*, actualizadas)

- Librería externa de codificación y decodificación JSONLAB

- Librerías externas de visión artificial YAEL y VLFEAT 0.9.20

- Microsoft Excel

- Microsoft Word

Anexo IV

Presupuestos:

A continuación se desglosan los costes estimados para este proyecto en lo que a mano de obra, software y hardware se refiere.

B.1. Coste de ejecución material:

B.1.1. Costes de hardware:

Concepto	Precio unidad	Unidades	Subtotal
Ordenador Asus Intel Core i5	549 €	1	549 €

Tabla III.1: Costes de *hardware*

B.1.2. Costes de software:

Durante la realización de este trabajo se han usaso varias aplicaciones para generar la documentación y el código. Aunque para el alumno ha tenido costo cero, se debe tener en cuenta que la licencia de MatLab R2017a es pagada por la UAH, imprescindible porque incorpora funciones actualizada sólo para esta versión (como por ejemplo la librería SURF).

Concepto	Coste de licencia [€]
MatLab R2017a	6.745,45
Ubuntu 16.04	0
Librería JSON	0
Librería VLFEAT	0
Librería YAEL	0
Microsoft Excel	0
Microsoft Word	0

Tabla III.2: Costes de *software*

Aunque Excel y Word presentan un coste de licencia, estas aplicaciones están incorporadas en el ordenador tras la compra, por tanto no hace falta adquirirlas de nuevo para este TFG.

B.1.3. Costes de personal:

Para estimar el coste del personal se debe desglosar las horas del proyecto con un sueldo medio de 15 euros por hora para un Ingeniero Industrial:

Concepto	Precio [€]/hora	Horas	Subtotal [€]
Comprensión del estado del arte	15	120	1.800
Creación de la herramienta	15	240	3.600
Búsqueda de geolocalizaciones	15	100	1.500
Procesamiento de las geolocalizaciones	15	150	2.250
Modificación de CNN y resultados	15	60	900
Redacción de la memoria	15	170	2.250

Tabla III.3: Costes de personal

B.1.4. Costes de ejecución material totales:

Los costes totales están basados en la **suma de los costes de hardware, software y personal:**

Concepto	Subtotal [€]
Costes de hardware	549
Costes de software	6.745,45
Costes de personal	12300
Subtotal final	19.594,45

Tabla III.4: Costes de ejecución material totales

B.2. Gastos generales y beneficio industrial:

El beneficio industrial y los gastos generales son desembolsos obligatorios que vienen tipificados por la utilización de las instalaciones de trabajo e ingresos generados a nivel de manufactura. Se estimará un 16 % sobre el coste de ejecución material:

Concepto	Subtotal [€]
Costes de ejecución material	19.594,45
Porcentaje estimado	16 %
Subtotal final	3135,11

Tabla III.5: Gastos generales y beneficio industrial

B.3. Presupuesto de ejecución por contrata:

Este presupuesto es igual a la **suma de los costes de ejecución material y gastos generales y el beneficio industria:**

Concepto	Subtotal [€]
Costes de ejecución material	19.594,45
Gastos generales y beneficio industrial	3135.11
Subtotal final	22.729,56

Tabla III.6: Presupuesto de ejecución por contrata

B.4. Importe total del presupuesto:

Para terminar el presupuesto final del TFG, este se debe calcular a partir de la **aplicación de un IVA** (por normal general en este ámbito es de un **21 %**) a partir del presupuesto de ejecución por contrata:

Concepto	Subtotal [€]
Presupuesto de ejecución por contrata	22.729,56
Porcentaje estimado	21 %
Subtotal final	4.773,21
Importe final	27.202,77

Tabla III.7: Importe total del presupuesto

El importe final de proyecto asciende a: 22.729,56 € + 4.773,21 € = 27.202,77 € (**Veintisiete mil dosciente dos con setenta y siete euros**).

Bibliografía:

[1] Página oficial de Google Street View (© Google)

https://www.google.es/intl/es/streetview/

[2] API de Google Street View (© Google)

https://developers.google.com/maps/documentation/streetview/?hl=es-419

[3] Edward R. Dowski, Jr., and W. Thomas Cathey (2015), "Extended depth of field through wave-front coding", Osapublishing. Disponible en:

https://www.osapublishing.org/DirectPDFAccess/7306D4F7-C659-BA4A-FECD0B2D5E41F03E_45359/ao-34-11-1859.pdf?da=1&id=45359&seq=0&mobile=no

[4] Marco Cavallo, (2016), "3D City Reconstruction From Google Street View" University Of Illinois At Chicago. Disponible en:

https://www.evl.uic.edu/documents/3drecomstrictionmcavallo.pdf

[5] API Google Maps Rest para generación del azulejo en alta resolución (© Google):

http://cbk0.google.com/cbk?output=tile&panoid=%s&zoom=5&x=%d&y=%d

[6] API Google Maps Rest para generación del azulejo en baja resolución (© Google):

https://geo0.ggpht.com/cbk?cb_client=maps_sv.tactile&authuser=0&hl=en&panoid=%s&output=tile&x=%d&y=%d&zoom=3

[7] API Google Maps Rest para generación del mapa de profundidad (© Google):

https://maps.google.com/cbk?output=json&cb_client=maps_sv&v=4&dm=1&pm=1&ph=1&hl=en&panoid=%s

[8] Akihiko Torii, Relja Arandjelovic, Josef Sivic, Masatoshi Okutimi, Tomas Pajdla (2015), "24/7 place recognition by view synthesis", Okutomi & Tanaka Laboratory. Disponible en:

http://www.ok.ctrl.titech.ac.jp/~torii/project/247/download/Torii-CVPR-2015-final.pdf

[9] T. Hermosilla, E. Bermejo, A. Balaguer, L.A. Ruiz (22-23 de Noviembre de 2006), "Detección de bordes con precisión subpíxel en imágenes digitales: Interpolación lineal frente a esquemas de tipo no lineal", Universidad Politécnica de Valencia, VII Jornadas de Matemática Aplicada , páginas 9–20. Disponible en:

http://cgat.webs.upv.es/BigFiles/Charla_Hermosilla_JMA_06.pdf

[10] Jorge Valverde Rebaza (2010): "Detección de bordes mediante el algoritmo de Canny". Researchgate. Disponible:

https://www.researchgate.net/profile/Jorge_Valverde-Rebaza/publication/267240432_Deteccion_de_bordes_mediante_el_algoritmo_de_Canny/links/548dd1ae0cf225bf66a5f636/Deteccion-de-bordes-mediante-el-algoritmo-de-Canny.pdf

[11] Chris Harris, Mike Stephens (1998) "A COMBINED CORNER AND EDGE DETECTOR", Plessey Research Roke Manor, United Kingdom, © The Plessey Company pic. Disponible en:

http://courses.daiict.ac.in/pluginfile.php/13002/mod_resource/content/0/References/harris1988.pdf

[12] Rafael Aracil López (2012), "Desarrollo de un sistema cognitivo de visión para la navegación robótica", Trabajo de Fin de Grado, Escola Tècnica Superior d'Enginyeria Informàtica, Universitat Politècnica de València. Disponible en:

https://riunet.upv.es/bitstream/handle/10251/17010/memoria.pdf?sequence=1

[13] A. Vedaldi and B. Fulkerson (2008) "{VLFeat}: An Open and Portable Library", VLFEAT. Disponible en:

http://www.vlfeat.org/

[14] Roberto García García (2007), "Sistema de Odometría Visual para la Mejora del Posicionamiento Global de un Vehículo", Universidad de Alcalá. Disponible en:

https://www.researchgate.net/profile/MA_Sotelo/publication/237674176_DETECCION_Y_RECONOCIMIENTO_DE_PANELES_DE_TRAFICO_MEDIANTE_TECNICAS_DE_PROCESADO_DE_IMAGEN/links/55d1b8ec08ae3dc86a4f2c56.pdf

[15] Larry Zitnic, CSE P 576 "Geometric Transformations"

https://courses.cs.washington.edu/courses/csep576/11sp/pdf/Transformations.pdf

[16] Andreas Geiger (2014), "3D Traffic Scene Understanding from Movable Platforms". Pattern Analysis and Machine Intelligence (PAMI). Disponible en:

http://www.cvlibs.net/software/liblabel/

[17] Pablo F. Alcantarilla, Simon Stent, German Ros, Roberto Arroyo, Riccardo Gherardi (2016" "Street-View Change Detection with Deconvolutional Networks". iRobot Corporation, London UK. Department of Engineering, University of Cambridge. Computer Vision Center, UAB, Barcelona, Spain. Department of Electronics, Univeraity of Alcaá, Madrid, Spain. Toshiba Research Europe Ltd., Cambridge, UK. Disponible en:

https://pdfs.semanticscholar.org/fd21/0d083941f438934e3009d4399030eafa6b26.pdf?_ga=2.231654488.658083681.1505573766-341240629.1505573766

[18] Ken Sakurada, Takayuki Okatani (2015), "Change Detection from a Street Image Pair using CNN Features and Superpixel Segmentation", Tohoku University. Disponible en: CNN basada en superpixelación

http://www.vision.is.tohoku.ac.jp/files/9814/3947/4830/71-Sakurada-BMVC15.pdf

[19] Fernando Sancho Caparrini (2017) "Redes Neuronales: Una visión superficial". Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla. Disponible en:

http://www.cs.us.es/~fsancho/?e=72

[20] Raul E.López Briega (2016) "Redes Neuronales Convolucionales con Tensor Flow". Disponible en;

https://relopezbriega.github.io/blog/2016/08/02/redes-neuronales-convolucionales-con-tensorflow/

[21] Engin Tola, Pascal Fua, Vincent Lepetit (2010) "Daisy: An efficient dense descriptor applied to wide baseline stereo". Disponible en:

https://www.researchgate.net/publication/42345292_Daisy_An_Efficient_Dense_Descriptor_Applied_to_Wide_Baseline_Stereo

[22] Derek Hoiem (2011) "Projective Geometry and Camera Models". Computer Vision. CS 543 / ECE 549. University of Illinois. Disponible en:

https://courses.engr.illinois.edu/cs543/sp2011/lectures/Lecture%2002%20-%20Projective%20Geometry%20and%20Camera%20Models%20-%20Vision_Spring2011.pdf

[23] Antonio R. Franco (1999) "Características de las Coordenadas UTM y descripción de este tipo de coordenada". Elgps. Disponible en:

http://www.elgps.com/documentos/utm/coordenadas_utm.html

[24] Página oficial de MathWorks (© MathWorks Inc)

https://es.mathworks.com/