



FACULTAD DE INFORMÁTICA
UNIVERSIDAD POLITÉCNICA DE MADRID

TESIS DE MÁSTER
MÁSTER UNIVERSITARIO EN INTELIGENCIA ARTIFICIAL

SIMULACIÓN DE ENTORNOS URBANOS PARA EL
APRENDIZAJE DE DESCRIPTORES LOCALES DE
APARIENCIA

AUTOR: Néstor Rafael Audante Ramos
TUTOR: Xoan Iago Suárez Canosa
DIRECTOR: Luis Baumela Molina

Madrid, 17 de julio de 2019

Resumen

En este trabajo se presenta “RUF3DM” una base de datos de: **(1)** imágenes urbanas realistas, que contienen información de profundidad, obtenidas desde un simulador y **(2)** porciones pequeñas de imágenes (patches) correspondientes a esquinas, manchas y segmentos (características locales) provenientes de las imágenes previamente mencionadas. Esta base de datos puede ser usada en distintos tipos de problemas, como pueden ser la estimación de puntos de fuga, la reconstrucción 3D, la estimación de profundidad, etc. Asimismo, se detalla paso a paso el procedimiento seguido para la generación de la base de datos, que básicamente consiste en: (1) obtener las imágenes y su información de profundidad, (2) alinear los sistemas de referencia para expresar las coordenadas de la cámara y su orientación en términos del mundo, (3) la detección de las características locales, (4) la orientación y el recortado de los patches y (5) la evaluación de la eficacia de los patches generados en el entrenamiento de un descriptor. Los experimentos realizados demuestran que el dataset sintético generado es tan parecido al real, que los descriptores de imágenes locales pueden ser entrenados a partir de este dataset y obtener resultados comparables con los obtenidos con imágenes reales.

Agradecimientos

Gracias ante todo a Dios.

Gracias a mis padres, María y Néstor, por darme lo mejor de ellos todo este tiempo.

Gracias a mis hermanos Luis, Lesli, Sandro y Mitchel, por estar siempre para mí cuando los necesito, por sus consejos y su aliento.

Gracias a todas las personas que, de una u otra manera, contribuyeron a la elaboración de este trabajo: A todo el laboratorio de Percepción Computacional y Robótica por su constante ayuda; a los profesores de la Facultad, por los conocimientos impartidos; y a mis compañeros de estudios, de los cuales siempre pude aprender mucho.

Gracias a todas las personas valiosas que me rodean y que me motivan a intentar ser cada día mejor.

Índice general

1. Introducción	1
1.1. Objetivos	2
1.2. Organización del documento	2
2. Trabajos previos	3
2.1. Características Locales	3
2.1.1. Detección de características	3
2.1.2. Descripción de características	4
2.2. Bases de datos de imágenes generadas usando Structure from Motion	6
2.3. Base de datos de patches	6
2.4. Motores Gráficos Sintéticos	7
3. Generación de la Base de datos de patches	9
3.1. Entorno de Simulación: Carla	9
3.1.1. Módulos de Carla	9
3.1.2. Principales características de Carla	10
3.1.2.1. Sensores	10
3.1.2.2. Cambio de condición climática	12
3.1.2.3. Selección de Mapas	12
3.2. Generación de la base de datos de imágenes	13

3.2.1.	Formato de la base de datos de imágenes	13
3.2.2.	Alineamiento de los Sistemas de Referencia	17
3.3.	Construcción de las bases de datos de patches	20
3.3.1.	Construcción de BD de patches de Esquinas	20
3.3.1.1.	Detección de esquinas en las imágenes	21
3.3.1.2.	Cálculo de los puntos 3D usando los mapas de profundidad. . . .	21
3.3.1.3.	Generación de patches de esquinas.	23
3.3.2.	Construcción de BD de patches de Manchas	25
3.3.2.1.	Detección de manchas en las imágenes	26
3.3.2.2.	Cálculo de las manchas 3D usando los mapas de profundidad. . .	27
3.3.2.3.	Generación de patches de manchas.	27
3.3.3.	Construcción de BD de patches de Segmentos	29
3.3.3.1.	Detección de segmentos en las imágenes	29
3.3.3.2.	Cálculo de los segmentos 3D usando los mapas de profundidad. .	29
3.3.3.3.	Generación de patches de segmentos.	33
4. Experiments		37
4.1.	Experimentos en patches de Esquinas	37
4.1.1.	Primer experimento: Prueba en dataset “notredame”.	37
4.1.2.	Segundo experimento: Prueba en dataset “liberty”.	38
4.1.3.	Tercer experimento: Prueba en dataset “RUF3D_corners”.	39
4.2.	Experimentos en patches de Manchas	41
4.2.1.	Primer experimento: Prueba en dataset “notredame”.	41
4.2.2.	Segundo experimento: Prueba en dataset “liberty”.	42
4.2.3.	Tercer experimento: Prueba en dataset “RUF3D_blobs”.	42
5. Conclusiones		45

5.1. Trabajos Futuros	46
Bibliografía	47

Índice de figuras

1.1.	Patches similares de la Estatua de la Libertad [44].	1
2.1.	Funcionamiento de BELID. Se calculan los valores medios de gris de los píxeles de las cajas rojas y azules. Luego, por cada par de cajas rojas y azules (weak learner) se restan los valores medios de gris, esto da como resultado $f(x)$. Se aplican una serie de umbrales para obtener las respuestas $h(x)$ y, finalmente, se multiplica por la matriz B para producir el descriptor $D(x)$ [17].	5
2.2.	Imágenes obtenidas desde MegaDepth[24].	6
2.3.	En la izquierda, se muestra la reconstrucción 3D obtenida del mapa de profundidad y de la imagen RGB (que están en el lado derecho) del dataset MegaDepth.	8
3.1.	Módulos del simulador Carla.	10
3.2.	Imágenes capturadas por el sensor de tipo Cámara. Zona Superior: Imagen tomada con la Cámara RGB. Zona media: Imagen tomada con la Cámara de Profundidad. Zona inferior: Imagen tomada con la Cámara de Segmentación.	11
3.3.	Información almacenada en el archivo .h5. Zona superior: Mapa de profundidad. Zona Inferior: Información relativa a la profundidad de cada uno de los píxeles (las imágenes son 1280x720 píxeles).	12
3.4.	En cada una de las filas se muestran imágenes con distintas condiciones climáticas, tomadas desde distintas perspectivas de una misma escena.	13
3.5.	Distintas ciudades disponibles en Carla. Zona Superior: Mapa de “town01”. Zona media: Mapa de “town02”. Zona inferior: Mapa de “town03”	14
3.6.	Estructura de directorios de base de datos de imágenes generada usando el simulador Carla.	15
3.7.	Relación entre ángulos de visión (fov), la distancia focal y el tamaño del sensor. . .	16
3.8.	Cálculo de la focal	16

3.9. La cámara que capta la escena se encuentra físicamente en el vehículo, el cual tiene una dimensión, por tanto su sistema de referencia (de centro “C”) puede ser diferente al sistema de referencia del mundo (de centro “O”).	18
3.10. Transformación Euclídea entre las coordenadas del mundo y de la cámara[16].	19
3.11. Ángulos de rotación[30]	20
3.12. Ejemplo de patches de esquinas.	21
3.13. Esquinas detectadas con ORB.	21
3.14. Zona superior izquierda: Imagen original en formato RGB. Zona superior derecha: Mapa de profundidad de la imagen anterior. Zona inferior: Imagen 3D reconstruida a partir de la información de profundidad.	22
3.15. Puntos seleccionados.	23
3.16. Un mismo punto 3D detectado en 2 perspectivas distintas.	23
3.17. Recortado y orientado de patches.	24
3.18. Patches de esquinas generados.	26
3.19. Ejemplo de patches de manchas.	26
3.20. Puntos detectados con SIFT	27
3.21. Patches de manchas generados.	28
3.22. Ejemplo de patches de segmentos.	29
3.23. Segmentos detectados con FSG.	30
3.24. Zona superior izquierda: Imagen original en formato RGB. Zona superior derecha: Mapa de profundidad de la imagen anterior. Zona inferior: Imagen 3D reconstruida con la información de profundidad. En esta imagen también se muestran los segmentos -que han sido detectados previamente- reproyectados en el mundo 3D.	30
3.25. Segmentos seleccionados.	31
3.26. Un mismo segmento 3D detectado en 2 perspectivas distintas.	32
3.27. Segmentos mezclados.	33
3.28. Recortado y orientado de patches de segmentos.	33
3.29. Generación de patches de segmentos.	34

3.30. En la sección superior, se encuentran patches sin orientar, con el segmento ubicado en el medio del patch, resaltado. En la sección inferior, se muestran los mismos patches orientados en base al $\cos \Theta$	35
3.31. Patches de segmentos generados.	36
4.1. Curva ROC obtenida entrenando el descriptor BELID en el dataset de RUF3D_corners200000 y probando en el dataset “notredame”.	38
4.2. Curva ROC obtenida, probando en el dataset “liberty”.	39
4.3. Curva ROC obtenida probando en el dataset “RUF3D_corners”.	40
4.4. Curva ROC obtenida, para las manchas, en el dataset “notredame”.	41
4.5. Curva ROC obtenida probando en el dataset “liberty”.	43
4.6. Curva ROC obtenida probando en el dataset “RUF3D_blobs”.	44

Capítulo 1

Introducción

La capacidad de comparar porciones pequeñas de imágenes (patches) ha sido la base de muchos enfoques empleados para dar solución a problemas de visión comunes, como son: SLAM, odometría visual, recuperación de imágenes y un amplio etcétera. Dichas secciones de imágenes usualmente suelen representar esquinas, manchas o segmentos (características locales) obtenidas de una imagen de mayor tamaño y son de gran utilidad en visión, pues los modelos con ellas construidos son invariantes a rotación, a cambios de iluminación, robustas a occlusiones parciales, etc. La figura 1.1 muestra un ejemplo de patches que son similares entre sí.



Figura 1.1: Patches similares de la Estatua de la Libertad [44].

Sin embargo, para la obtención de dichas bases de datos, se han empleado imágenes reales, lo que supone muchas restricciones dado que su generación es muy costosa, pues se requiere que se hayan etiquetado algunos elementos (que pueden ser píxeles, puntos, segmentos, etc) en las imágenes, esta tarea generalmente es muchas veces manual y demanda mucho tiempo y recursos. Otra opción es generar dichas correspondencias de forma automática, pero en este caso es necesaria una tarea de validación de que las correspondencias obtenidas sean correctas, algo que tampoco es trivial y supone un esfuerzo intensivo. Esto hace que las bases de datos reales tengan muy pocos ejemplos y que los patches generados no sean muy diversos, sino más bien muy parecidos entre si, etc. Una forma de automatizar este proceso es asumir que la estructura de la escena es sencilla, generalmente plana, lo cual implica no representar todas las configuraciones geométricas posibles que pueden darse en una escena real. En consecuencia, las bases de datos reales actuales resultan limitadas en el número de imágenes, variedad geométrica de las escenas consideradas y, por tanto, en los patches generados.

En este trabajo, se va a explorar la forma de generar una base de datos de patches a partir de imágenes sintéticas, obtenidas desde un simulador y se va a evaluar si al emplearlas se encuentran resultados similares a los obtenidos usando patches de imágenes reales.

1.1. Objetivos

Los principales objetivos de este trabajo son:

- **O1:** Generar un programa que permita obtener imágenes sintéticas y su información geométrica 3D -de forma automática- usando para esto un motor gráfico.
- **O2:** Generar bases de datos de secciones de imágenes (patches) correspondientes a esquinas, manchas y segmentos (características locales) a partir de la base de datos de imágenes sintéticas generada en el punto anterior.
- **O3:** Demostrar que se pueden usar las bases de datos de porciones pequeñas de imágenes sintéticas -obtenidas en el punto anterior-, como reemplazo de bases de datos de imágenes reales. Para esto se evaluará la eficacia de un descriptor entrenado usando la bases de datos de porciones de imágenes sintéticas y probándolo en bases de datos de imágenes reales.

1.2. Organización del documento

Este documento está organizado de la siguiente forma: En el capítulo 2 revisaremos el estado del arte y realizaremos una revisión de las técnicas más usadas en la generación de bases de datos de patches. En el capítulo 3 mostraremos el procedimiento seguido para la generación de las bases de datos de patches de esquinas, manchas y segmentos, a partir de la imágenes sintéticas obtenidas desde el simulador Carla[10]. En el capítulo 4 se muestran los resultados obtenidos entrenando el descriptor BELID[17] con la base de datos de patches generada en este trabajo y probando su efectividad en las bases de datos patches de Brown[44]. Por último, en el capítulo 5 se presentan las conclusiones y trabajos futuros que se desprenden de este trabajo.

Capítulo 2

Trabajos previos

En el campo de la visión por computador, un problema recurrente es el de obtener una base de datos con la que entrenar y validar los algoritmos con los que se trabaja. Este Trabajo de Fin Máster (TFM) se dedica al emparejamiento de imágenes, para esto vamos a usar características locales: esquinas, manchas y segmentos. Por tanto, es necesario definir previamente qué son las características locales y cómo nos pueden ser útiles, algo que pasamos a describir a continuación.

2.1. Características Locales

Supongamos que pensamos en una imagen en términos de datos (como una matriz de píxeles), las características de la imagen es la información que se extrae de ella y que la representa en términos de valores numéricos empleando una menor dimensión. Las ventajas de las características locales es que son distintivas, que son robustas a cambios de punto de vista, a occlusiones parciales y muy eficientes. Su representación puede, por tanto, ayudarnos a reconocer la misma característica en imágenes distintas.

Dos tareas fundamentales, al hablar de características locales, consisten en su detección y su descripción, lo que pasamos a detallar.

2.1.1. Detección de características

Se encarga de ubicar aquellas regiones de interés, en una imagen, que presenten contenido relevante, suelen ser: esquinas, manchas, segmentos, etc; que son identificados porque presentan un pronunciado cambio en los niveles de gris de los píxeles que lo rodean. En la actualidad, podemos agrupar dichos detectores en 2 tipos: (1) Detectores ingenieriles (o handcrafted) y (2) detectores entrenados. Entre los detectores ingenieriles encontramos: HARRIS, SIFT y SURF. Entre los detectores entrenados encontramos a FAST y ORB. Una aproximación común para detectar dichas características es usar métodos basados en el gradiente (pues no son muy costosos desde el punto de vista computacional), como HARRIS [15], que usa el operador de Sobel, SIFT[26] que está

basado en la diferencia de gaussianas, SURF [7] que intenta aproximar la diferencia de gaussianas -que usa SIFT- usando un filtro de caja (con lo que consigue mejor rendimiento), FAST[34] que se basa en la comparación de cada píxel de la imagen con un anillo de píxeles a su alrededor, ORB [35] que es una fusión del detector FAST, con una variación de la descripción de BRIEF[9]. En este trabajo vamos a usar a ORB y SIFT para detectar esquinas y manchas respectivamente. Hemos seleccionado a ORB por ser el descriptor más eficiente que existe en la literatura y SIFT por el ser el más ampliamente utilizado, siendo un compromiso razonable entre precisión y eficiencia computacional. A continuación pasamos a describirlos.

- **SIFT[26]: Scale Invariant Feature Transform.**

Es un detector y descriptor de puntos de interés (patentado) que transforma datos de una imagen en coordenadas invariantes a escalas. Es invariante a rotación, es robusto a transformaciones afines, a cambios en los puntos de vistas 3D, es robusto al ruido y a cambios en las condiciones de iluminación. Los principales pasos que usa SIFT para generar las características de la imagen son:

- **Detección de extremos en el espacio de escalas:** Usa la diferencia de gaussianas para ubicar potenciales puntos de interés que sean invariantes tanto a escala como a orientación.
- **Localización de puntos de interés:** Consiste en localizar de forma adecuada los puntos de interés, en base a un modelo, dichos puntos serán seleccionados en base a su estabilidad.
- **Asignación de orientación:** Una o más orientaciones son asignadas a los puntos de interés, basados en el gradiente de la imagen local.
- **Descripción de los puntos de interés:** Los gradientes de la imagen son medidos, en la escala seleccionada, en la región que rodea al punto de interés. Dichas medidas, son luego representadas como un vector de alta dimensionalidad.

En nuestro caso, usaremos SIFT para las tareas orientadas a la detección de manchas.

- **ORB[35]: Oriented FAST and Rotated BRIEF.** Es un detector y descriptor de puntos de interés, basado en BRIEF, que consigue invarianza a rotación usando la orientación de esquinas de FAST, y que ordena los puntos de interés basado en el score de Harris. Este procedimiento, lo hace en cada nivel de una pirámide de escalas de imágenes[21]. En nuestro caso, usaremos ORB para las tareas orientadas a la detección de esquinas.

A continuación pasamos a detallar la descripción de características.

2.1.2. Descripción de características

Una vez que se han detectados los puntos de interés en una imagen, el siguiente paso es la descripción de dichos puntos y su entorno, empleando una representación compacta y robusta, de forma tal que estos puedan compararse con otros puntos característicos y así evaluar cuán similares son. En la actualidad, podemos agrupar dichos descriptores en 2 tipos: (1) Descriptores ingenieriles (o

handcrafted) y (2) Descriptores entrenados. Entre los descriptores ingenieriles encontramos: SIFT, SURF, BRIEF, BRISK, ORB. Entre los descriptores entrenados encontramos a BinBoost, VGG, BELID.

Algunos de los descriptores ingenieriles más usados son los descriptores binarios, como por ejemplo: BRIEF[9], que propone una codificación binaria, que se genera comparando directamente pares de puntos en las vecindades del punto característico; y versiones mejoradas del mismo para hacerlo invariante a rotación, escalado e iluminación como pueden ser ORB y BRISK[22]. Otro tipo de descriptores son los denominados “Spectra Descriptor”[21], llamados así porque requieren más cálculo y recursos, a menudo involucrando cálculos en punto flotante como son: SIFT y SURF (que -como se mencionó en la sección anterior- son detectores y descriptores a la vez). Entre los descriptores entrenados tenemos: BinBoost[41] que usa boosting, un paradigma de optimización bastante conocido, para realizar una correspondencia no lineal entre los patches de entrada y el espacio de características, con el empleo de “weak learners” basados en el gradiente, VGG[37] que usa la optimización convexa para aprender las posiciones de la característica detectada en las que debe realizar las mediciones, BELID[17] que es un descriptor entrenado, recientemente creado en el Grupo de Percepción Computacional y Robótica de la UPM, el cual hace uso de la imagen integral para calcular eficientemente la diferencia entre la media de los valores de gris presente en un par de regiones cuadradas. Se basa en BoostedSCC (una modificación de Adaboost) para de forma discriminante seleccionar un conjunto de características, las cuales combina para generar una descripción fuerte. A muy alto nivel, su funcionamiento se ilustra en la figura 3.12.

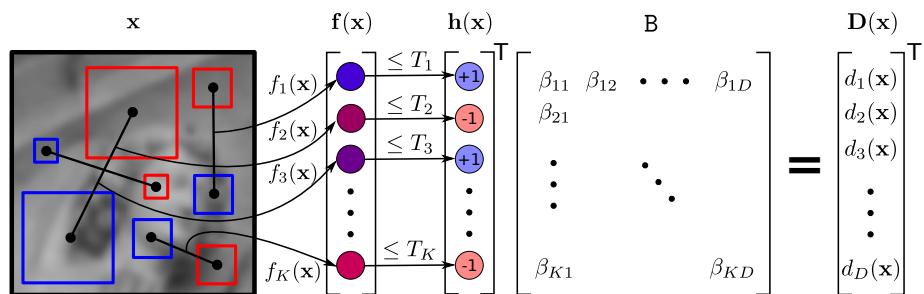


Figura 2.1: Funcionamiento de BELID. Se calculan los valores medios de gris de los píxeles de las cajas rojas y azules. Luego, por cada par de cajas rojas y azules (weak learner) se restan los valores medios de gris, esto da como resultado $f(x)$. Se aplican una serie de umbrales para obtener las respuestas $h(x)$ y, finalmente, se multiplica por la matriz B para producir el descriptor $D(x)$ [17].

Una vez entendidas las características locales, aún nos queda el problema de la construcción de la base de datos que nos va a servir para el entrenamiento de nuestros algoritmos. Una primera aproximación -para obtener estas bases de datos- son las propuestas de algunos autores que las han generado usando Structure from Motion (sfm). A continuación procedemos a describir estas aproximaciones.

2.2. Bases de datos de imágenes generadas usando Structure from Motion

Structure from Motion (sfm) consiste en reconstruir la estructura 3D de una escena a partir de un conjunto de imágenes 2D, mediante la estimación del movimiento relativo entre las cámaras que capturaron dichas imágenes. Algunas bases de datos relevantes que se han construido usando sfm son: Photo tourism[38] que genera un explorador de fotos, que permite realizar un tour virtual de escenas específicas (como por ejemplo de la catedral de Notre Dame), usando para su entrenamiento imágenes obtenidas a partir de escenas controladas (por ejemplo una persona con una única cámara y un único lente captando imágenes de Praga) y escenas diversas de internet (fotos de Flickr por ejemplo); Dubrovnik6K[23] y Rome16K[4] que generan un dataset a partir de colecciones de imágenes de internet buscadas a partir de una consulta específica (por ejemplo Roma) para elaborar un modelo 3D, usando también “image matching” y Megadepth[24] que presenta un gran dataset de imágenes con su mapa de profundidad, obtenidas a partir de imágenes de internet (específicamente Flickr) y usando además técnicas de MVS (multi-view stereo). La figura 2.2 muestra un ejemplo de dichas imágenes y mapa de profundidad.

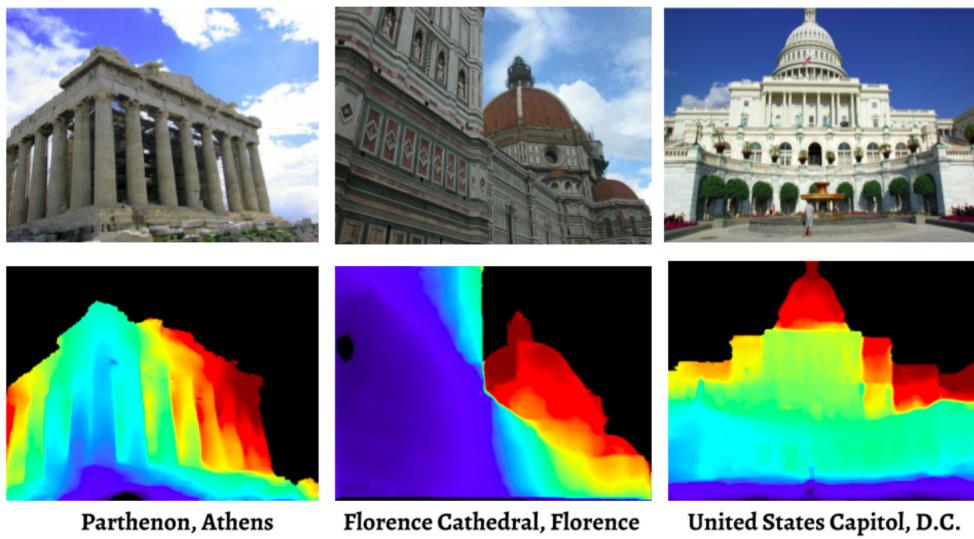


Figura 2.2: Imágenes obtenidas desde MegaDepth[24].

Sin embargo, el examinar porciones de imágenes que contienen las características locales (patches) -en lugar de las imágenes completas- puede suponer una ventaja para algunos algoritmos (en términos de precisión y de cómputo), por lo que algunos autores han preferido la generación de bases de datos de patches. A continuación, se presentan algunas de dichas aproximaciones.

2.3. Base de datos de patches

Una de las primeras referencias para la creación de base de datos de patches es la base de datos de Mikolajczyk[28] que presenta un dataset de 48 escenas planas (8 secuencias de 6 imágenes),

cuyo “ground truth” está basado en homografías. Los 6 tipos de transformaciones aplicadas sobre dichas imágenes son: rotación, cambio de escala, cambio de perspectiva, empañado de imágenes, compresión JPEG e iluminación. Aquí, si bien es cierto, no se genera una base de datos de patches “per se”, pero la idea de generación del ground truth y el uso de homografías sirvió como base para las bases de datos posteriores. En Oxford building dataset[31] se presenta un método de recuperación de imágenes a gran escala a partir de una consulta sobre una región específica de una imagen, para esto propone un método de cuantización basado en árboles aleatorios (random trees) para crear un vocabulario de características de imágenes. En Brown[44] se presenta un dataset de patches cuadrados (de 64x64) que se obtienen proyectando los puntos 3D de las reconstrucciones de Photo Tourism[38] (yosemite, notredame, liberty) en sus imágenes originales. Para la identificación de los puntos de interés se usa SIFT, descartando algunos de dichos puntos usando simetría y utilizando RANSAC para estimar las matrices fundamentales. En HPatches[6] se presenta un método para comparar algoritmos de extracción de descriptores, dichos descriptores son evaluados en el dataset de patches que los autores generan a partir de imágenes de distintas fuentes (como por ejemplo [19] y [3]), seleccionando puntos de interés con los siguientes detectores : Hessian, Harris y DoG, y quedándose con los puntos 3D que pueden ser reproyectados en la imagen que les dio origen. Finalmente, AMOS[32] presenta un método para generar patches a partir de imágenes obtenidas desde cámaras web estacionarias ubicadas al aire libre (dataset AMOS[18]).

Sin embargo las bases de datos de imágenes -completas y de patches- obtenidas a partir de imágenes reales presentan algunos inconvenientes, como son:

- No son suficientemente precisas.
- No contienen toda la información que muchas veces se necesita.
- Su generación es extremadamente costosa (en términos de cómputo).

Una consecuencia de que la calidad de la información de profundidad no sea del todo buena, es que las reconstrucciones 3D que se obtienen de dichos dataset sean muy mejorables, tal y como se muestra en la figura 2.3 que corresponde a una reconstrucción obtenida a partir de la base de datos MegaDepth.

Vemos entonces, que es demasiado complejo, generar buenas bases de datos a partir de imágenes reales. Sin embargo, gracias al problema de conducción autónoma, han aparecido nuevas fuentes de datos (motores gráficos sintéticos) a partir de los cuales podríamos obtener imágenes sintéticas que nos pueden servir para la generación de una base de datos de patches. A continuación, pasamos a describir algunos de dichos motores.

2.4. Motores Gráficos Sintéticos

Entre los motores gráficos más conocidos tenemos los siguientes: Virtual Kitti[20] que presenta un dataset sintético de vídeos foto-realistas, que se encarga de generar mundos sintéticos a partir de vídeos reales obtenidos de Kitti Benckmark [14], usando para esto el motor de videojuegos “Unity”. SYNTHIA[33] (SYNTHetic collection of Imagery and Annotations), que es un dataset

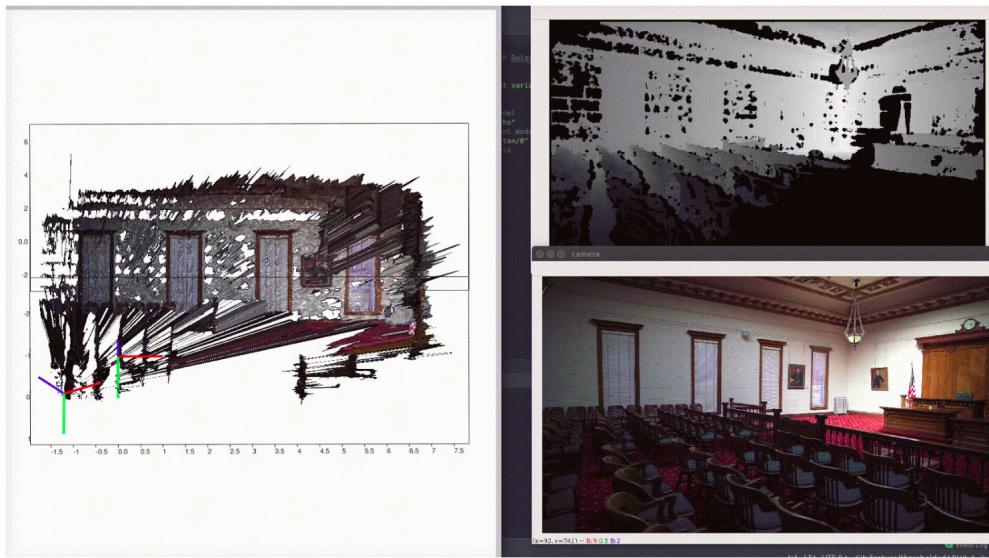


Figura 2.3: En la izquierda, se muestra la reconstrucción 3D obtenida del mapa de profundidad y de la imagen RGB (que están en el lado derecho) del dataset MegaDepth.

de imágenes sintéticas (obtenido de una ciudad virtual generada en Unity) que se usa para tareas relacionadas a segmentación semántica y problemas relacionados al entendimiento de escenas en el escenario de la conducción autónoma. Finalmente, encontramos a Carla[10] que es un simulador, también usado para la conducción autónoma, que presenta un mundo virtual generado completamente desde cero, usando el motor Unreal Engine 4[13]. Carla es el simulador desde el que se van a generar las bases de datos de patches sintéticos que servirán para el entrenamiento del descriptor BELID en esquinas, manchas y segmentos, y se describirá a detalle en la sección 3.1, por esta razón aquí sólo se menciona brevemente. En el capítulo siguiente pasamos a describir el procedimiento para la generación de las bases de datos de patches sintéticos previamente mencionadas.

Capítulo 3

Generación de la Base de datos de patches

En este capítulo se describirá los pasos realizados para generar nuestra base de datos de patches a partir de imágenes sintéticas obtenidas de un simulador. Un patch es una porción pequeña de una imagen (generalmente de forma rectangular). Para ilustrar la idea tomemos de ejemplo un patch de 16x16, lo que indica es que se trata de una región cuadrada de 256 píxeles de una imagen más grande. El simulador que se va a utilizar para obtener las imágenes sintéticas, para generar nuestra base de datos, es Carla, el cual pasamos a describir a continuación.

3.1. Entorno de Simulación: Carla

Carla[10] es un simulador de código abierto cuya finalidad es ayudar a la investigación en tareas relacionadas a la conducción autónoma, sin embargo, en este trabajo nos hemos valido de facilidad que nos brinda para obtener imágenes sintéticas desde distintas perspectivas de una escena, además de interactuar con los sensores del vehículo que está capturando la escena y poder cambiar las condiciones climáticas, esto beneficiará la generación de nuestra base de datos de patches. La versión de Carla que se ha usado para nuestra simulación es la 9.0.2.

3.1.1. Módulos de Carla

En la Figura 3.1 se muestran los principales módulos de Carla[1], en ella se puede apreciar que cuenta con: **(a)** Un módulo de simulación (“simulator”), que es donde se hace el trabajo pesado, que es el renderizado, la definición de la lógica, la definición de los mapas, etc; y **(b)** un módulo de API’s de Python que nos permite interactuar mediante programación con el simulador, y de esta forma controlar el vehículo, obtener datos del mismo, seleccionar los sensores, cambiar el clima, etc. En este trabajo, se va a interactuar principalmente con esta API, mediante scripts de Python.

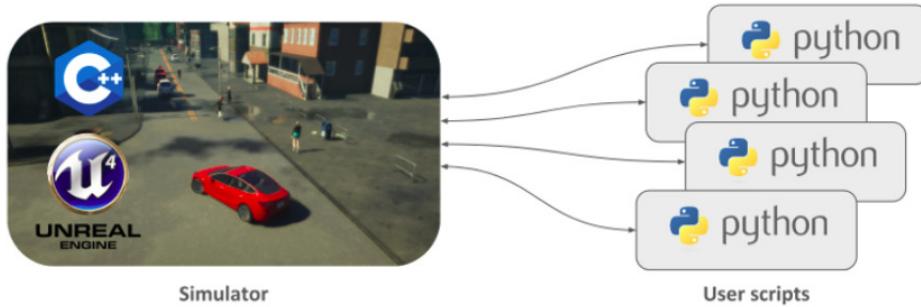


Figura 3.1: Módulos del simulador Carla.

3.1.2. Principales características de Carla

Se va a listar las características principales que hemos aprovechado de este entorno de simulación.

3.1.2.1. Sensores

Carla presenta una diversidad de sensores, entre ellos: sensores de cámara, sensores de colisión, sensores de obstáculos, sensores de invasión de carril, etc. En este trabajo nos centraremos en los sensores de cámara.

a) Sensores de Cámara

Los sensores de cámara proveen de información valiosa, entre otras la imagen, la posición de la cámara en coordenadas del mundo, la orientación de la cámara, etc. Los sensores de cámara pueden ser de 3 tipos: sensores de cámara RGB, sensores de profundidad y sensores de segmentación semántica.

La figura 3.2. nos muestra como se ve una misma escena captada con distintos tipos de sensores de cámara.

En nuestro caso, se va a trabajar con los sensores RGB y sensores de profundidad de la cámara. Para cada imagen se guardará su imagen en RGB (en formato png) y su mapa de profundidad (se realizó una modificación en la funcionalidad de Carla, para guardar la información relativa a la distancia de los píxeles en formato de 32 bits en puntos flotantes, contenido en un archivo “hdf5”), tal y como se muestra en la figura 3.3 (Para la visualización se usó el software HDfCompass[2]).

La profundidad de las imágenes devueltas por Carla se encuentra codificada en el espacio de colores RGB (va desde el bit más significativo al menos significativo, $R - > G - > B$), entonces para expresar la profundidad de cada pixel en metros (dist_en_metros) se debe usar la ecuación:

$$\text{dist_en_metros} = 1000(R + 256G + 265^2B)/(256^3 - 1) \quad (3.1)$$

Carla ofrece también la ”pose de la cámara”, algo que ellos llaman ”transform”, en nuestro caso, la información relevante es la siguiente:

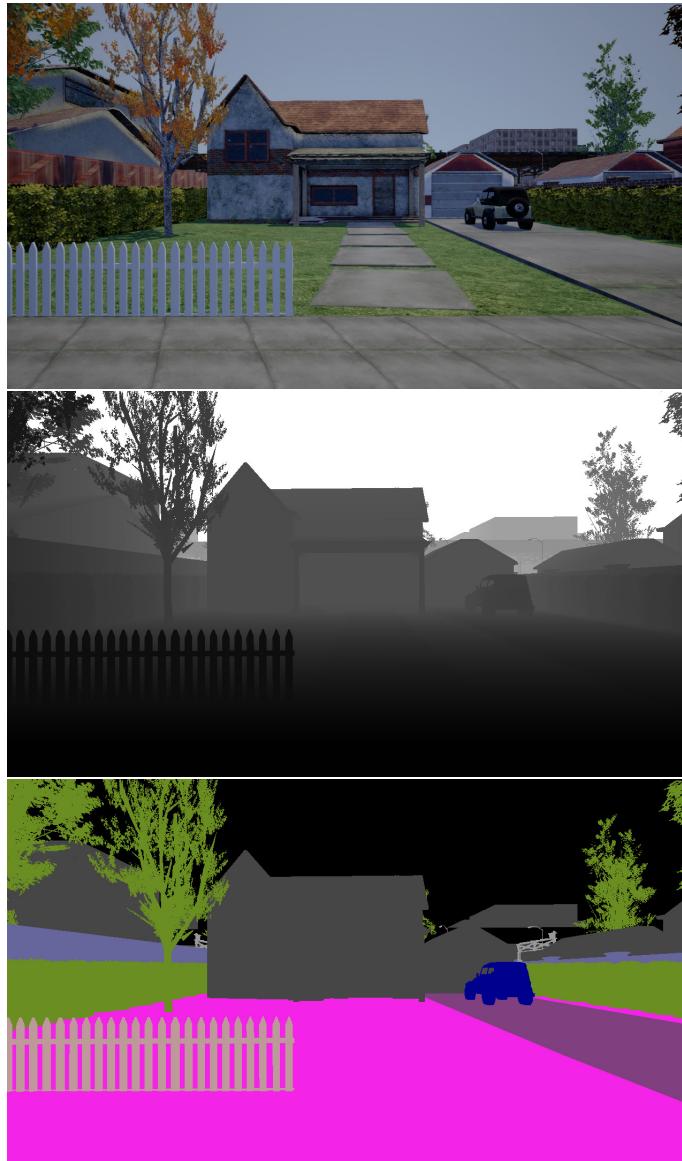


Figura 3.2: Imágenes capturadas por el sensor de tipo Cámara. **Zona Superior:** Imagen tomada con la Cámara RGB. **Zona media:** Imagen tomada con la Cámara de Profundidad. **Zona inferior:** Imagen tomada con la Cámara de Segmentación.

- Las posiciones de la cámara en metros (coordenadas x, y, z).
- Los ángulos de rotación de la cámara: roll, pitch y yaw, en grados.
- El ángulo de visión horizontal **fov**, en grados.

Mas adelante, explicaremos el uso que le damos a estos datos. A continuación se explica otra característica importante, que es el cambio de las condiciones climáticas.

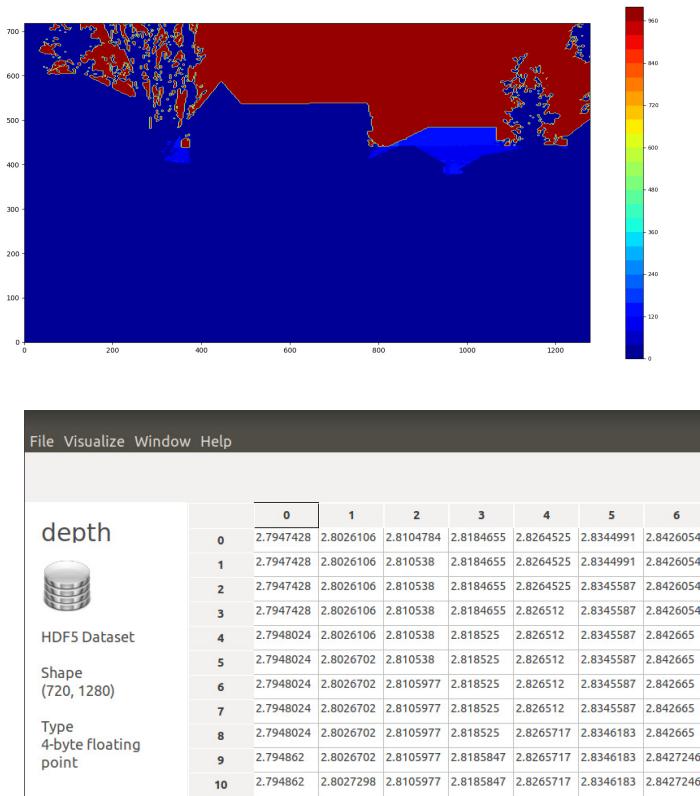


Figura 3.3: Información almacenada en el archivo .h5. **Zona superior:** Mapa de profundidad. **Zona Inferior:** Información relativa a la profundidad de cada uno de los píxeles (las imágenes son 1280x720 píxeles).

3.1.2.2. Cambio de condición climática

Carla nos permite cambiar las condiciones climáticas, lo que posibilita capturar una misma escena con diferentes climas y, diferentes condiciones de iluminación. Se puede capturar, por ejemplo, una imagen de una escena con un día soleado, otra con un día nublado y otra con un día lluvioso, etc. Esto es muy valioso, para brindarle diversidad al dataset que se quiere generar. En cada fila de la figura 3.4 se puede apreciar imágenes de una misma escena, en distintas condiciones climáticas.

3.1.2.3. Selección de Mapas

En la versión de Carla con la que trabajamos (v9.0.2), se dispone de mapas, correspondientes a 3 ciudades distintas para la ejecución de la simulación. Dichos mapas se muestran en la figura 3.5.



Figura 3.4: En cada una de las filas se muestran imágenes con distintas condiciones climáticas, tomadas desde distintas perspectivas de una misma escena.

3.2. Generación de la base de datos de imágenes

A continuación se procede a describir los pasos para generar la base de datos de imágenes. Debido a que el sistema de referencia del simulador Carla es distinto al sistema de referencia empleado típicamente en visión por computador, es necesario realizar un alineamiento de los sistemas de referencia. Adicionalmente, dado que la base datos de imágenes va ha ser comparada con otras bases de datos (como por ejemplo, con la base de datos de MegaDeph[25]), se ha decidido seguir un formato estándar (adecuado principalmente al formato Megadepth) para guardar la información referente a los parámetros extrínsecos e intrínsecos de la cámara. Procedemos a describir este formato en el siguiente punto.

3.2.1. Formato de la base de datos de imágenes

El formato empleado en la generación de nuestra base de datos de imágenes, es similar que se usó en las bases de datos de Megadepth[25], que se muestra en la figura 3.6.

Se puede apreciar que existe un directorio para las imágenes en RGB (“**images**”) y un directorio para los mapas de profundidad (“**depths**”). En el directorio “**0**”, de la ruta “**sparse→manhattan→0**” se encuentran 2 archivos importantes: “**cameras.txt**” e “**images.txt**”

- “**cameras.txt**”: Permite almacenar información referente a las cámaras que se usan. En nuestro caso sólo usamos una. Recordemos que la matriz de intrínsecos “K” [16] se calcula



Figura 3.5: Distintas ciudades disponibles en Carla. **Zona Superior:** Mapa de “town01”. **Zona media:** Mapa de “town02”. **Zona inferior:** Mapa de “town03”.

de la siguiente forma:

$$K = \begin{pmatrix} fk_u & s & i_0 \\ 0 & fk_v & j_0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} \alpha & s & i_0 \\ 0 & \beta & j_0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.2)$$

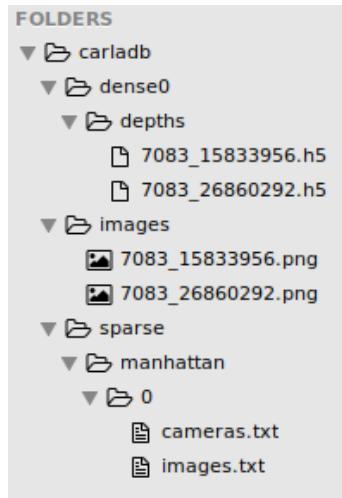


Figura 3.6: Estructura de directorios de base de datos de imágenes generada usando el simulador Carla.

Donde s representa el sesgo de la cámara, α y β representan las focales horizontal y vertical e i_0 y j_0 representan el punto principal.

La información que se almacena en el archivo es: (1) un identificador de cámara, (2) el modelo de cámara (en nuestro caso es una cámara radial), (3) la resolución de la misma (alto y ancho) y (4) los parámetros intrínsecos de la cámara; tal y como se ve en el listado siguiente:

```

1 # Camera list with one line of data per camera:
2 #   CAMERA_ID, MODEL, WIDTH, HEIGHT, PARAMS[]
3 # Number of cameras: 1
4 1 SIMPLE_RADIAL 1280 720 640 640 360 0

```

Para obtener el valor de los parámetros de las focales y del punto principal impondremos las siguientes restricciones:

- El valor del sesgo es 0 (dado que es un modelo sintético, es decir, nuestra cámara es perfecta).
- El punto principal está centrado en la imagen.
- No existe distorsión óptica.
- Las focales fk_u y fk_v tienen el mismo valor (píxeles cuadrados).

En la figura 3.7 se muestra que se puede relacionar el ángulo de visión Θ (fov, que en nuestro caso es 90°), con la distancia del centro de proyección al plano imagen: f (focal).

Fijándonos sólo en el fov horizontal Θ_x y, dado que el punto principal está centrado en el medio, esta relación se puede simplificar aún más y quedaría expresada con la siguiente ecuación (dicha relación también se muestra en la figura 3.8):

$$f = \frac{w_{pix}}{2 \tan(\Theta_x/2)}$$

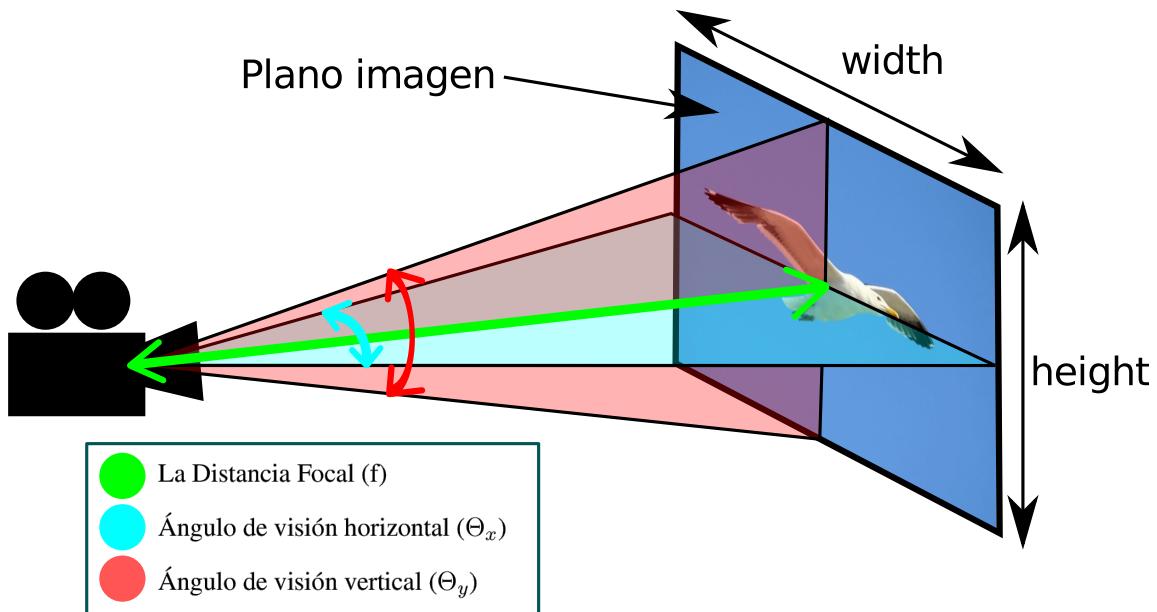


Figura 3.7: Relación entre ángulos de visión (fov), la distancia focal y el tamaño del sensor.

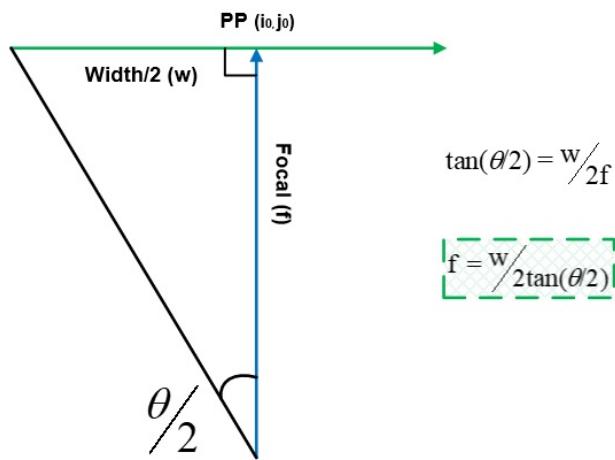


Figura 3.8: Cálculo de la focal

Con lo que nuestra matriz de intrínsecos \mathbf{K} quedaría como:

$$K = \begin{pmatrix} 640 & 0 & 640 \\ 0 & 640 & 360 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.3)$$

- “**images.txt**”: : Almacena la información referente a la posición de la cámara en cada una de las imágenes. La información que se almacena se muestra en el siguiente listado:

```

1 # Image list with two lines of data per image:
2 # IMAGE_ID, QW, QX, QY, QZ, TX, TY, TZ, CAMERA_ID, NAME
3 1_120 0.205 0.001 -0.978 0.002 -84.341 0.954 -120.499 1 1_120.png

```

Donde QW, QX, QY, QZ representan la rotación de la cámara expresada en cuaternios y TX, TY, TZ representan el vector de traslación t o lo que es lo mismo $[t_x, t_y, t_z]^T$. En la sección siguiente, explicaremos cómo obtener el vector de traslación t y el vector de Rotación R (dado que hay un paso previo de alineamiento de sistemas de referencia), sin embargo, este fichero necesita las rotaciones de la cámara expresada como cuaternios, no como matriz de rotación. Para obtener los cuaternios a partir de la matriz R se puede usar la función **mat2quat** del módulo **quaternions** de la librería **transform3d**[27]. La forma de usarla es la siguiente.

```
q = quaternions.mat2quat(R)
```

Por lo tanto, el problema se reduce a hallar los vectores t y R , algo que se pasa a describir a continuación.

3.2.2. Alineamiento de los Sistemas de Referencia

En la parte superior de la figura 3.9 se muestra una escena capturada por el sensor de tipo cámara del auto. Dicha imagen es similar a la que tendríamos si nos encontráramos en el interior del auto. En la parte inferior de dicha figura, se muestra una toma desde el exterior, de la misma escena. Es importante precisar que, dado que la cámara está físicamente situada en el auto (el cual tiene una dimensión, forma, etc), que el auto puede moverse, y que el centro de la misma (en la gráfica lo hemos representado con la letra “C”) no se encuentra en el centro del sistema de referencia del mundo (que lo hemos representado con la letra “O”). Adicionalmente, dicha cámara puede estar rotada con respecto al centro de referencia del mundo. La rotación y la translación, con respecto al mundo, la hemos representado en la figura con las letras “R” y “t”, donde R_{3x3} es una matriz de rotación y t_{3x1} es un vector de traslación.

La figura 3.11 muestra la transformación Euclídea entre las coordenadas del mundo y de la cámara de manera más simplificada. La ecuación que relaciona el vector de traslación, la matriz de rotación y el centro de la cámara viene dada por: $t = -RC$. Los valores devueltos por el sensor de la cámara nos permiten encontrar los valores de R y C .

1. Cálculo del centro de la cámara C



(a) Escena capturada por la cámara del auto.



(b) Vista de la misma escena desde el exterior del auto.

Figura 3.9: La cámara que capta la escena se encuentra físicamente en el vehículo, el cual tiene una dimensión, por tanto su sistema de referencia (de centro “C”) puede ser diferente al sistema de referencia del mundo (de centro “O”).

La posición de la cámara es un dato retornado por el sensor, sin embargo la dirección de los ejes en el sistema de coordenadas de Carla es distinto a la dirección de los ejes del sistema de coordenadas usado en visión por computador. Esto es debido a que el sistema de coordenadas de carla está orientado a “derecha” (Right-handed coordinate system) mientras que el sistema de referencia de visión por computador está orientado a “izquierdas” (Left-handed coordinate system). Las diferencias son las siguientes:

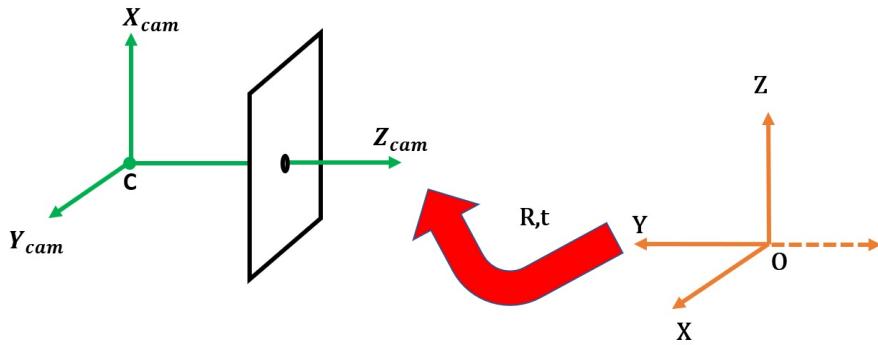


Figura 3.10: Transformación Euclídea entre las coordenadas del mundo y de la cámara[16].

- En el sistema de coordenadas de carla, la dirección positiva del eje X, corresponde a la dirección positiva del eje Z del sistema de visión por computador.
- En el sistema de coordenadas de carla, la dirección positiva del eje Y, corresponde a la dirección positiva del eje X del sistema de visión por computador.
- En el sistema de coordenadas de carla, la dirección positiva del eje Z, corresponde a la dirección negativa del eje Y del sistema de visión por computador.

Por lo tanto, para que los ejes sean equivalentes tanto en el sistema de Carla como en el de visión por computador, es necesario multiplicar el valor de las coordenadas del centro de la cámara, devuelto por Carla, por la siguiente matriz:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{pmatrix} \quad (3.4)$$

De esta manera ya se tiene calculado el centro C .

Ahora se pasa a describir la forma de encontrar la matriz de Rotación R .

2. Cálculo de la matriz de Rotación R

Carla devuelve los ángulos que se encuentra rotada la cámara en notación de ángulos de Euler: roll, pitch y yaw. Para ilustrar las orientaciones que siguen dichos ángulos se muestra la figura 3.11.

Sea $\Theta_x = \text{pitch}$, $\Theta_y = \text{yaw}$ y $\Theta_z = \text{roll}$, las ecuaciones para obtener la matriz de Rotación R son:

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\Theta_x) & -\sin(\Theta_x) \\ 0 & \sin(\Theta_x) & \cos(\Theta_x) \end{pmatrix} \quad (3.5)$$

$$R_y = \begin{pmatrix} \cos(\Theta_y) & 0 & \sin(\Theta_y) \\ 0 & 1 & 0 \\ -\sin(\Theta_y) & 0 & \cos(\Theta_y) \end{pmatrix} \quad (3.6)$$

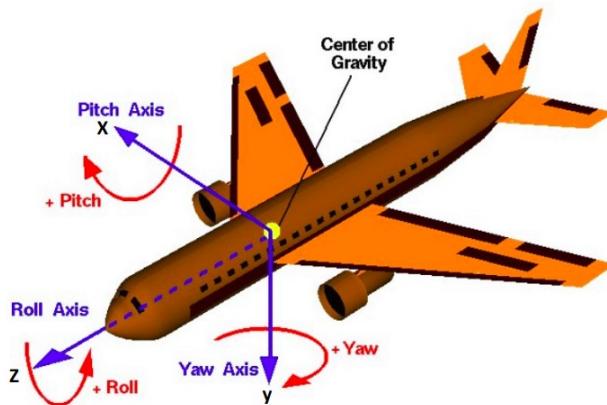


Figura 3.11: Ángulos de rotación[30]

$$R_z = \begin{pmatrix} \cos(\Theta_z) & -\sin(\Theta_z) & 0 \\ \sin(\Theta_z) & \cos(\Theta_z) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (3.7)$$

$$R = R_z^T \cdot R_x^T \cdot R_y^T = (R_y \cdot R_x \cdot R_z)^T \quad (3.8)$$

De lo dicho anteriormente, el cálculo del vector t se reduce a efectuar la siguiente multiplicación: $-RC$.

3.3. Construcción de las bases de datos de patches

En este trabajo se van a generar 3 bases de datos de patches, referentes a: patches de esquinas, patches de manchas, patches de segmentos. A continuación se pasa a detallar cada uno de estos puntos.

3.3.1. Construcción de BD de patches de Esquinas

La finalidad de esta sección es generar las mismas características 3D representadas desde distintas perspectivas. En nuestro caso, dichas características van a ser esquinas (corners) existentes en distintas imágenes y que van a ser almacenadas como patches. Un ejemplo de patches de esquinas se muestra en la figura 3.12

Los pasos a seguir para la generación de patches, son los siguientes:

- Detectar esquinas en las imágenes.



Figura 3.12: Ejemplo de patches de esquinas.

- Calcular los puntos 3D (esquinas) usando la imagen de profundidad.
- Detectar si la misma esquina ha sido detectada en varias imágenes.
- Recortar el patch alrededor de la imagen y exportarlo a un formato común.

3.3.1.1. Detección de esquinas en las imágenes

Entre las opciones que tenemos para detectar esquinas las más conocidas son: **Harris**[15], **FAST**[29], **Shi-Tomasi**[36] y **ORB**[35]. Como se mencionó en secciones anteriores en este trabajo emplearemos **ORB**. En la figura 3.13 se muestra un ejemplo de esquinas detectadas por ORB.

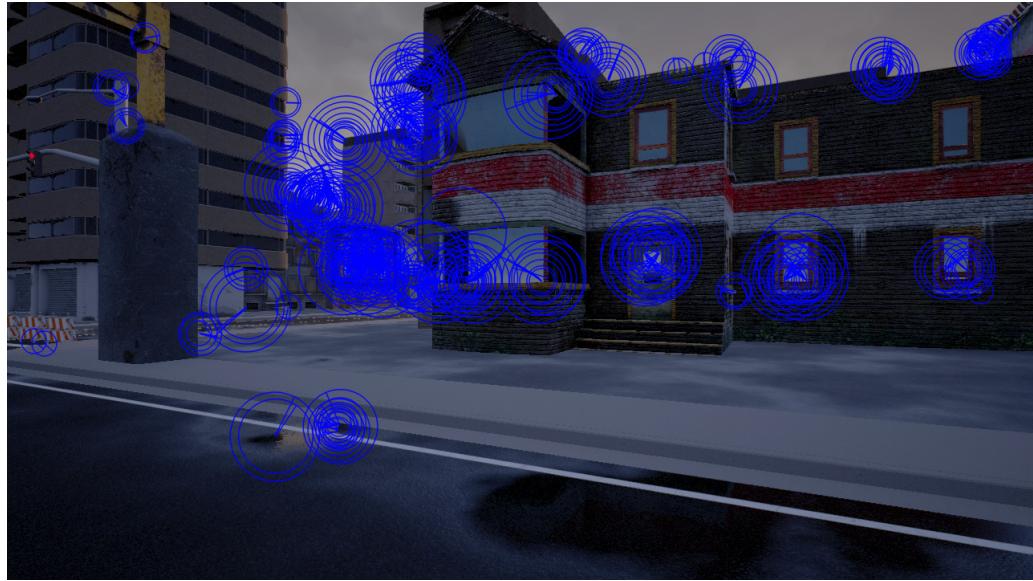


Figura 3.13: Esquinas detectadas con ORB.

Se puede notar que ORB, detecta esquinas a diversas escalas, es decir, que las características (Key-points) detectadas por ORB presentan un punto central, un radio y también una orientación. Estas características se usarán mas adelante en la orientación y recortado de los patches.

3.3.1.2. Cálculo de los puntos 3D usando los mapas de profundidad.

Es posible generar una representación 3D a partir de las imágenes 2D y de los mapas de profundidad (pues esta contiene información de la distancia a la que se encuentra cada uno de los píxeles).

En nuestra base de datos, se tiene esta información almacenada, por lo que es posible obtener una representación 3D de cada escena. En la figura 3.14, se muestra la reconstrucción 3D de una escena, y la imagen y el mapa de profundidad que dieron origen a la misma.

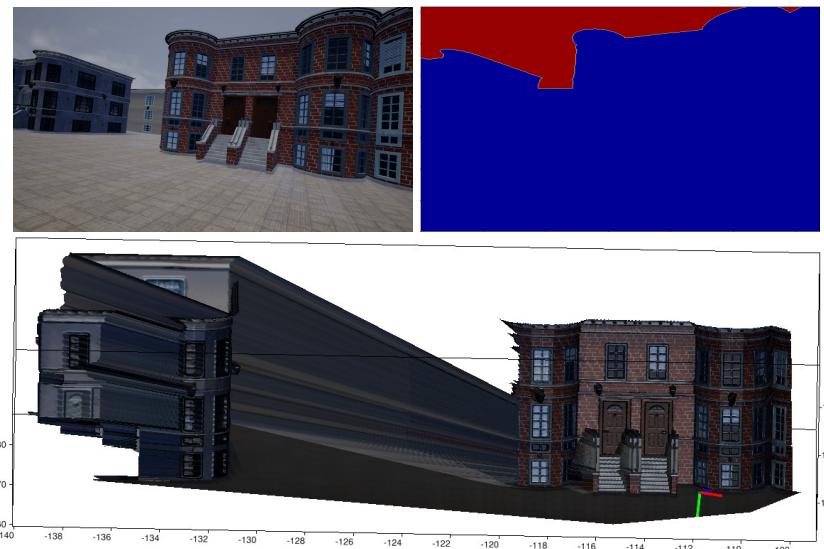


Figura 3.14: **Zona superior izquierda:** Imagen original en formato RGB. **Zona superior derecha:** Mapa de profundidad de la imagen anterior. **Zona inferior:** Imagen 3D reconstruida a partir de la información de profundidad.

El detector de esquinas de ORB encuentra un conjunto esquinas en 2D. De ese conjunto de puntos, sólo nos interesa quedarnos con aquellas que puedan ser reproyectadas de manera robusta al mundo 3D.

El procedimiento del cálculo de puntos 3D a partir de la imagen 2D y de la información de profundidad se conoce como “lifting”. El Lifting, consiste en transportar un punto 2D, que existe en el plano de la imagen (expresado en píxeles), hacia el mundo 3D.

En la figura 3.15 se muestra, en color rojo, el subconjunto de puntos seleccionados del total de puntos detectados en la escena anterior.

Dado que, del conjunto de puntos 3D iniciales (puntos 3D detectados), nos hemos quedado con un sub-conjunto (puntos 3D seleccionados), se procede a generar una estructura que contenga la información de todos los puntos 3D seleccionados, en cada una de las vistas de una determinada escena, algo hemos llamados “mapa de puntos 3D”. La finalidad de generar este mapa, es que podamos discernir si el punto 3D que acabamos de obtener -en una determinada perspectiva de la escena-, se corresponde a otro punto previamente detectado -desde otra perspectiva de la misma- lo que se ilustra en la figura 3.16

Para saber si un punto 3D encaja con algún otro punto existente, se calcula la distancia euclídea entre ambos, y si esta es menor a un umbral (que en nuestro caso ha sido fijado en 0.1), se considera que es el mismo.

La lógica de generación del “mapa de puntos 3D” es la siguiente: Recorrer cada uno de los puntos

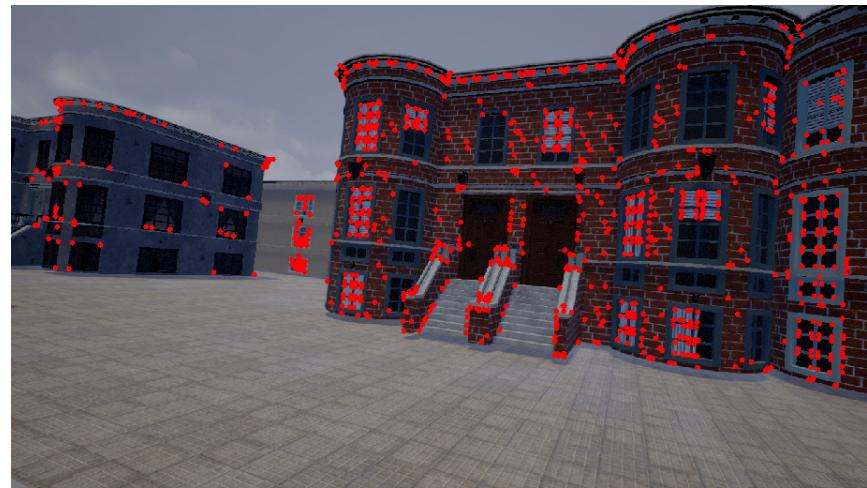


Figura 3.15: Puntos seleccionados.

Detectar que un mismo punto 3D ha sido detectado en varias imágenes

Por cada tipo de características (en este caso puntos) se debe definir una medida de distancia que nos permita mezclar características detectadas en distintas imágenes

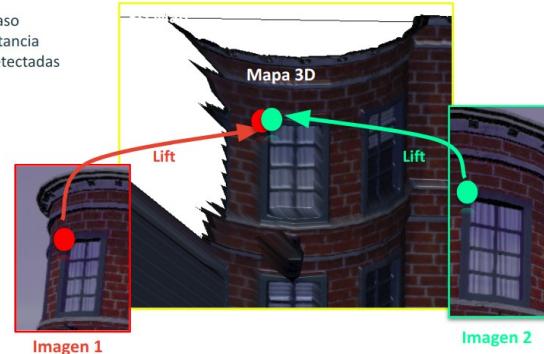


Figura 3.16: Un mismo punto 3D detectado en 2 perspectivas distintas.

3D seleccionados y si alguno de dichos puntos encaja con alguno que se encuentra en el mapa, estos se deben mezclar. El punto resultante (p_n) es la media ponderada de los 2 puntos iniciales: p_1, p_2 y se calcula con la siguiente fórmula:

$$p_n = \frac{w_1 \cdot p_1 + w_2 \cdot p_2}{w_1 + w_2} \quad (3.9)$$

donde, w_1 y w_2 son los pesos (las veces que dichos puntos han aparecido en las diferentes escenas) de los puntos p_1 y p_2 .

A continuación se pasa a describir la lógica para el recortado de los patches y su generación.

3.3.1.3. Generación de patches de esquinas.

Hay 2 pasos fundamentales para la generación de patches:

- Recortado del patch.
- Orientado del patch.

Dichos pasos están mostrados en la figura 3.17. En nuestro caso se van a generar patches de 64x64 y los patches serán orientados en base a la orientación devuelta por ORB (que en este caso será Θ grados)

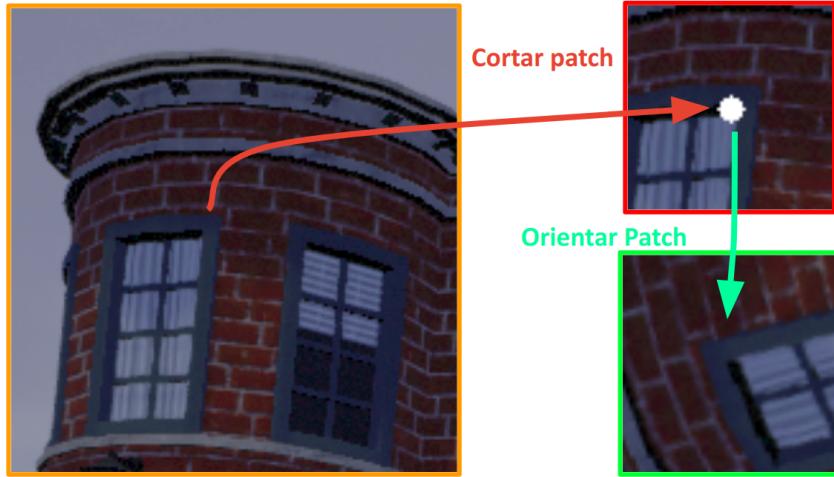


Figura 3.17: Recortado y orientado de patches.

Para recortar y orientar el patch, es necesario encontrar una transformación lineal \mathbf{H} , que obtenga el patch a partir de la imagen. Dicha transformación, debe:

- Trasladar el punto central del patch (p_x, p_y) a la esquina superior izquierda de la imagen (origen de su sistema de coordenadas).
- Rotarla en base a la orientación devuelta por ORB (Θ grados).
- Devolver los puntos (p_x, p_y) al centro del patch.

El cálculo de dicha transformación viene dado por la fórmula que se muestra a continuación:

$$H = \begin{pmatrix} 1 & 0 & pSize_x/2 \\ 0 & 1 & pSize_y/2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.10)$$

Donde: $pSize_x$ y $pSize_y$ son las dimensiones horizontales y verticales del patch. En nuestro caso, se trata de un patch cuadrado, por tanto dichos valores van a ser iguales. (p_x, p_y) son las coordenadas del centro del patch y Θ es la orientación devuelta por ORB.

Para obtener el patch, es necesario aplicar la transformación afín \mathbf{H} usando la función **warpPerspective** de OpenCV[8], como se muestra en el siguiente código:

```
1 result_img = cv2.warpPerspective(original_img, H, patch_size)
```

Finalmente, se procede a generar la base de datos con los patches obtenidos e información referida a los mismos. La estructura de nuestra base de datos es la siguiente:

- Archivo **points.txt**, el cual contiene la información de los patches y de los puntos 3D que la originaron en el siguiente formato:

```
1 patch_idx ptn_3d_idx proj_id n_file idx_in_file norm_reproj_err
```

donde:

- **patch_idx**: El índice del patch, que identifica únicamente un patch en la base de datos.
- **ptn_3d_idx**: el identificador del punto 3d al que corresponde el patch.
- **proj_id**: el identificador de proyección, habrá tantos como patches existan.
- **n_file**: el número de archivo donde se encuentra el patch.
- **idx_in_file**: el índice del patch dentro del archivo n_file.
- **norm_reproj_err**: el error de re-proyección normalizado. Es decir, la distancia entre el punto detectado en la imagen y la proyección del punto estimado en 3D.
- Varios archivos .txt con formato de nombre **m50_n1_n1_0.txt**, los cuales contienen información de **n1** matches verdaderos y **n1** matches falsos de los puntos 3D. Esto servirá para la evaluación de nuestra base de datos.
- Varias imágenes que contienen 256 patches cada uno (16 filas cada una de 16 patches). Un ejemplo de una de esas imágenes se muestra en la figura 3.18.

3.3.2. Construcción de BD de patches de Manchas

La finalidad de esta sección es generar las mismas características 3D representadas desde distintas perspectivas. En nuestro caso, dichas características van a ser manchas (blobs) existentes en una determinada escena y que van a ser almacenados en forma de patches. Un ejemplo de patches se muestra en la figura 3.19

Los pasos a seguir para la generación de patches, son los siguientes:

- Detectar manchas en las imágenes.
- Calcular los manchas 3D usando la imagen de profundidad.
- Detectar si la misma mancha ha sido detectada en varias imágenes.
- Recortar el patch alrededor de la imagen y exportarlo a un formato común.

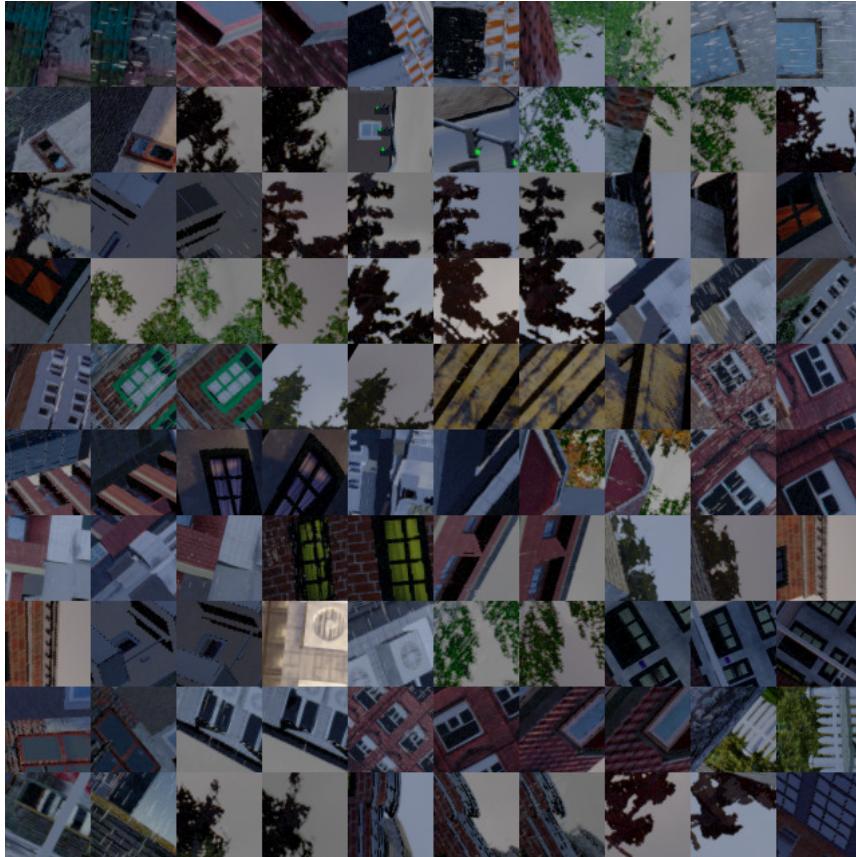


Figura 3.18: Patches de esquinas generados.



Figura 3.19: Ejemplo de patches de manchas.

3.3.2.1. Detección de manchas en las imágenes

Entre las opciones que tenemos para detectar manchas las más conocidas son: **SIFT**[26] (Scale-Invariant Feature Transform) y **SURF**[7] (Speeded-Up Robust Features). Con la finalidad de que los patches que generemos sean comparables con los obtenidos usando la base de datos de Brown [39], se procedió a usar SIFT. En la figura 3.20 se muestra un ejemplo de manchas detectadas por SIFT.

Se puede notar que SIFT, detecta manchas a diversas escalas, es decir, que las características detectadas por SIFT presentan un punto central, un radio y también una orientación. Estas características se usarán más adelante en la orientación y recortado de los patches.

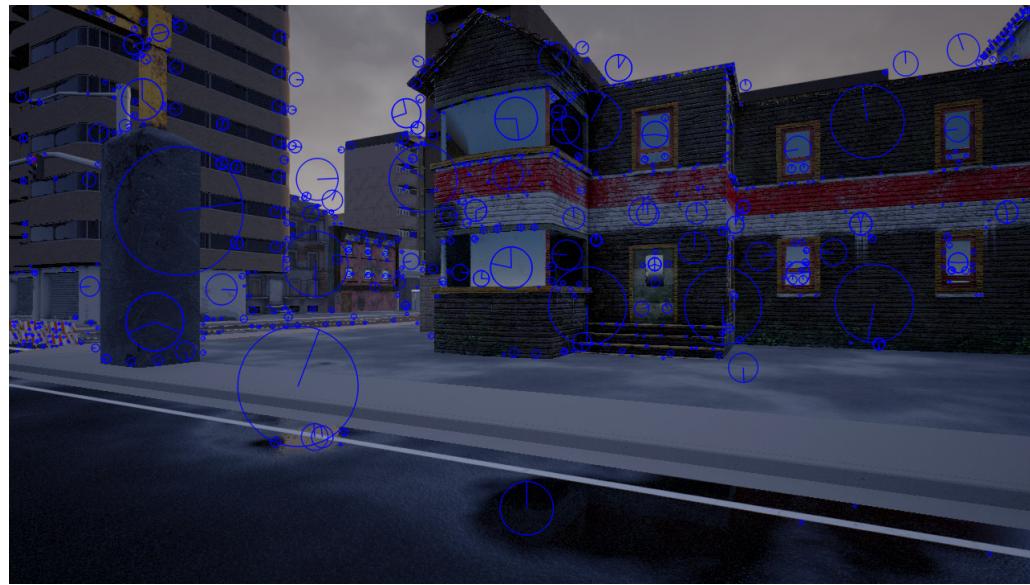


Figura 3.20: Puntos detectados con SIFT

3.3.2.2. Cálculo de las manchas 3D usando los mapas de profundidad.

Una mancha, a diferencia de un punto, posee una forma, una escala y una orientación. En este trabajo, a fin de simplificar los cálculos para la generación del mapa 3D, es decir, para efectos de calcular si dos manchas detectadas son las mismas, o para mezclar las manchas, hemos tomado la aproximación de tratar una mancha como un punto, es decir, obviamos su escala y orientación. Por lo tanto, los pasos y los criterios para la generación del mapa de manchas 3D son los mismos que se describieron en el apartado anterior. Sin embargo, es en la generación de los patches, en donde se va a tener en cuenta la orientación de la mancha (para orientar el patch) y la escala (para recortar el patch), tal y como veremos en el apartado siguiente.

3.3.2.3. Generación de patches de manchas.

Al igual que en los casos anteriores, se deben realizar 2 acciones para generar los patches: (1) Recortado del patch, y (2) Orientado del Patch. En este caso se van a generar también patches de 64x64, pero estos patches se orientarán en base a la orientación devuelta por SIFT (que en este caso será Θ grados) y serán re-escalados (dado que el patch tiene una forma).

Para recortar, orientar y escalar el patch, es necesario encontrar una transformación lineal \mathbf{H} , que obtenga el patch a partir de la imagen. Dicha transformación, debe:

- Trasladar el punto central del patch (p_x, p_y) a la esquina superior izquierda de la imagen (origen de su sistema de coordenadas).
- Rotarla en base a la orientación devuelta por SIFT (Θ grados).
- Aplicarle la escala al patch: s .

- Devolver los puntos (p_x, p_y) al centro del patch.

El cálculo de dicha transformación está dado por la fórmula que se muestra a continuación:

$$H = \begin{pmatrix} 1 & 0 & pSize_x/2 \\ 0 & 1 & pSize_y/2 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} s & 0 & 0 \\ 0 & s & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos \Theta & -\sin \Theta & 0 \\ \sin \Theta & \cos \Theta & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & -p_x \\ 0 & 1 & -p_y \\ 0 & 0 & 1 \end{pmatrix} \quad (3.11)$$

Donde: $pSize_x$ y $pSize_y$ son las dimensiones horizontales y verticales del patch. (p_x, p_y) son las coordenadas del centro de la mancha, s es la escala, que en nuestro caso es el factor de escala sugerido por SIFT ($SIFT_SCALE_FACTOR = 6,75$) y Θ es la orientación devuelta por SIFT. Para obtener el patch, es necesario aplicar la transformación afín \mathbf{H} , descrita en el punto anterior. La estructura de la base de datos generada, también es similar a la descrita en el apartado de puntos. Un ejemplo de las imágenes formadas a partir de los patches se muestra en la figura 3.21.

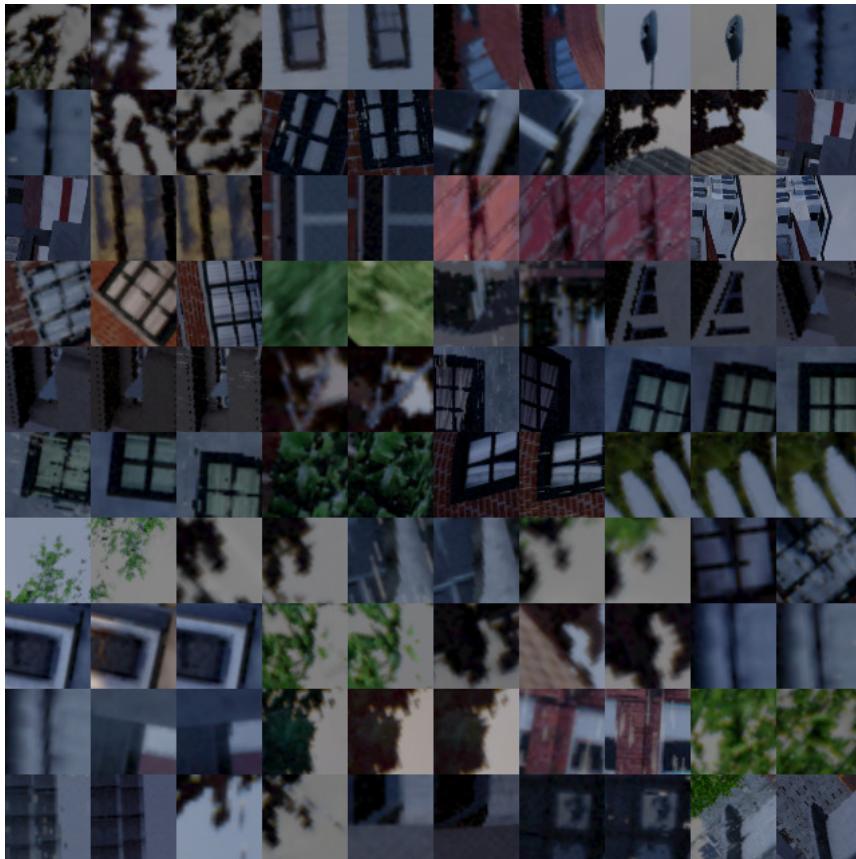


Figura 3.21: Patches de manchas generados.

3.3.3. Construcción de BD de patches de Segmentos

La finalidad de esta sección es generar las mismas características 3D representadas desde distintas perspectivas. En nuestro caso, dichas características van a ser segmentos existentes en una determinada escena y que van a ser almacenados en forma de patches. Un ejemplo de patches se muestra en la figura 3.22



Figura 3.22: Ejemplo de patches de segmentos.

Los pasos a seguir para la generación de patches, son los siguientes:

- Detectar segmentos en las imágenes.
- Calcular los segmentos 3D usando la imagen de profundidad.
- Detectar si el mismo segmento ha sido detectada en varias imágenes.
- Recortar el patch alrededor del segmento y exportarlo a un formato común.

3.3.3.1. Detección de segmentos en las imágenes

Entre las opciones que tenemos para detectar segmentos, las más conocidas son: **LSD**[43] y **ED-Lines**[5], las cuales están basadas en el agrupamiento local de píxeles, para de esta manera evitar el alto costo computacional que supondría el uso de la transformada de Hough[11]. Sin embargo, hace apenas unos meses, el grupo de Robótica y percepción de la UPM, presentó el detector de segmentos **FSG**[40], el cual es un algoritmo rápido y robusto para la detección de segmentos, que consta de 2 componentes: (1) un componente que agrupa los segmentos de forma voraz y sugiere los posibles segmentos candidatos para formar una línea y (2) un modelo probabilístico, que se encarga de determinar si una determinada agrupación de segmentos corresponde a una linea. En este trabajo vamos a emplear el detector FSG en las tareas relacionadas a la detección de segmentos.

En la figura 3.23 se muestra un ejemplo de segmentos detectados por FSG.

3.3.3.2. Cálculo de los segmentos 3D usando los mapas de profundidad.

Es posible generar una representación 3D a partir de las imágenes 2D y de los mapas de profundidad (pues esta contiene información de la distancia a la que se encuentra cada uno de los píxeles). En nuestra base de datos, se tiene esta información almacenada, por lo que es posible obtener

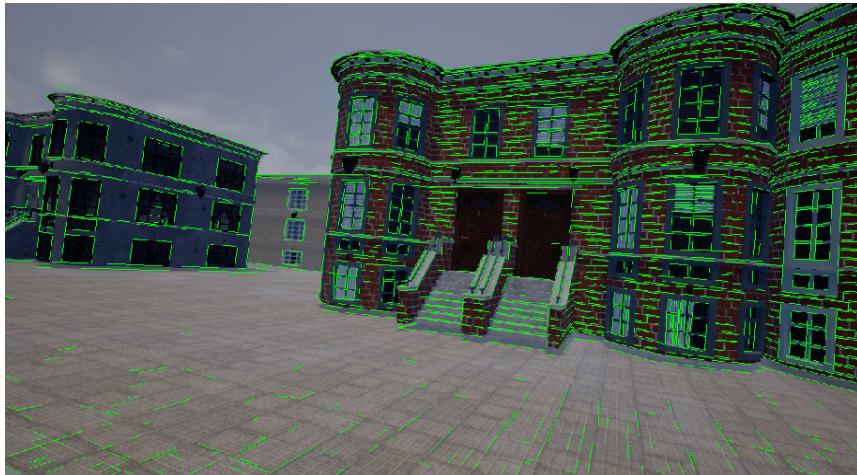


Figura 3.23: Segmentos detectados con FSG.

una representación 3D de cada escena. En la figura 3.24, se muestra la reconstrucción 3D de una escena, y la imagen y el mapa de profundidad que dieron origen a la misma.

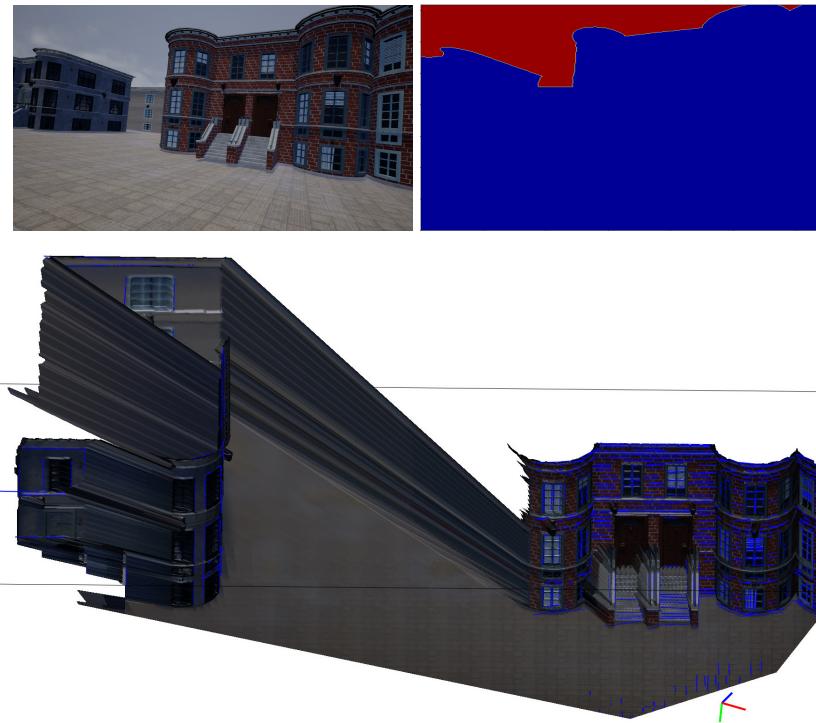


Figura 3.24: **Zona superior izquierda:** Imagen original en formato RGB. **Zona superior derecha:** Mapa de profundidad de la imagen anterior. **Zona inferior:** Imagen 3D reconstruida con la información de profundidad. En esta imagen también se muestran los segmentos -que han sido detectados previamente- reproyectados en el mundo 3D.

El detector de segmentos FSG encuentra un conjunto de segmentos 2D. De ese conjunto de segmentos sólo nos interesa quedarnos con los segmentos que:

- Sean lo suficientemente largos.
- Puedan ser reproyectados de manera robusta al mundo 3D.

El procedimiento del cálculo de segmentos 3D a partir de la imagen 2D y de la información de profundidad se conoce como “lifting”. El Lifting, consiste en transportar un segmento 2D que existe en el plano de la imagen (expresado en píxeles) hacia el mundo 3D. Para calcular el segmento 3D, se requiere una aproximación por mínimos cuadrados de la línea 3D tomando como base la profundidad de todos los píxeles que conforman la línea 2D original. Los pasos que se siguen son los siguientes:

1. Calcular los píxeles de cada uno de los puntos que conforman el segmento en la imagen 2D.
2. Obtener la profundidad de cada uno de los píxeles, usando la información de su mapa de profundidad.
3. Proyectar cada uno de los píxeles válidos como punto 3D.
4. Realizar una estimación robusta de la línea 3D, a partir de los puntos 3D usando el algoritmo RANSAC[12]
5. Descartar aquellas líneas 3D que no sean soportadas por una cantidad suficiente de puntos (el threshold es decidido por el usuario).
6. Calcular los extremos del segmento 3D en base al método de vecinos más cercano, tomando como referencia los extremos iniciales del segmento 2D original.

En la figura 3.25 se muestra, en color rojo, el subconjunto de segmentos seleccionados del total de segmentos detectados en la escena anterior.

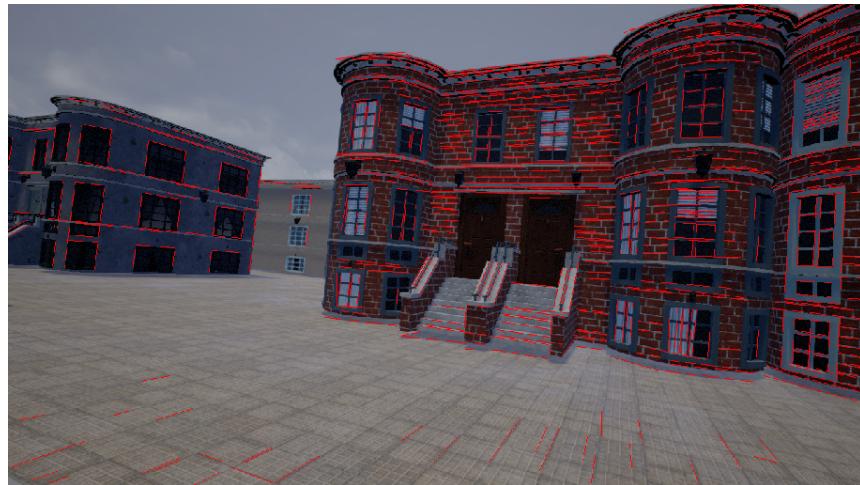


Figura 3.25: Segmentos seleccionados.

Dado que del conjunto de segmentos 3D iniciales (Segmentos 3D detectados), nos hemos quedado con un sub-conjunto (segmentos 3D seleccionados), se procede a generar una estructura que contenga la información de todos los segmentos 3D seleccionados, en cada una de las vistas de una

determinada escena, algo hemos llamados "mapa de segmentos 3D". La finalidad de generar este mapa, es que podamos discernir si el segmento 3D que acabamos de obtener -en una determinada perspectiva de la escena-, se corresponde a otro segmento previamente detectado- desde otra perspectiva de la misma, lo que se ilustrará en la figura 3.26

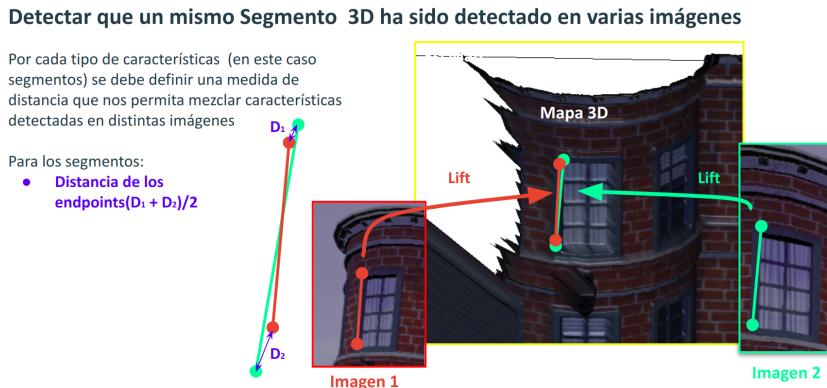


Figura 3.26: Un mismo segmento 3D detectado en 2 perspectivas distintas.

Para saber si un segmento 3D encaja con algún otro segmento existente, se compara la media de la distancia de los extremos de ambos segmentos y si dicho valor es menor a un umbral definido por el usuario (distance_threshold, en nuestro caso es "0.02"), los segmentos son los mismos. El cálculo de la distancia entre segmento se ilustra en el siguiente código:

```

1  @staticmethod
2  def segments_distance(s1, s2):
3      """
4          Calculates the distance between two segments as the mean of the
5              endpoints distance.
6          :param s1: The first segment.
7          :type s1: megadepth.Segment3D.Segment3D
8          :param s2: The second segment.
9          :type s2: megadepth.Segment3D.Segment3D
10         :return: The mean of the segments endpoints distance
11         :rtype: float
12         """
13         d_ss = np.linalg.norm(s1.start_p - s2.start_p)
14         d_se = np.linalg.norm(s1.start_p - s2.end_p)
15         d_es = np.linalg.norm(s1.end_p - s2.start_p)
16         d_ee = np.linalg.norm(s1.end_p - s2.end_p)
17         # Return the mean of the endpoints distance
18         return 0.5 * (min(d_ss, d_se) + min(d_es, d_ee))

```

La lógica de generación del "mapa de segmentos 3D" es la siguiente: Recorrer cada uno de los segmentos 3D y si alguno de dichos segmentos encaja con alguno que se encuentra en el mapa, estos se deben mezclar. El segmento resultante es la mezcla de los otros 2. En la figura 3.27 se muestran resultados, de color azul, aquellos segmentos que han sido mezclados en la escena anterior.

A continuación se pasa a describir la lógica para el recortado de los patches y su generación.

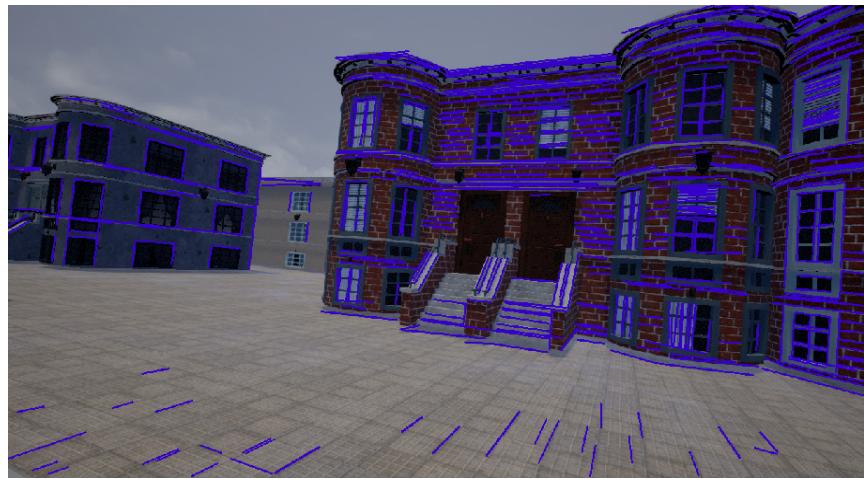


Figura 3.27: Segmentos mezclados.

3.3.3.3. Generación de patches de segmentos.

Al igual que en los casos anteriores, se debe realizar 2 acciones para generar los patches: (1) Recortado del patch, y (2) Orientado del Patch, sin embargo, a diferencia de los casos anteriores, en esta oportunidad se está trabajando con segmentos, por lo que el proceso de recortado y orientación es muy distinto. Los patches que se van a generar ya no van a ser cuadrados, sino rectangulares(en nuestro caso, de 50x100 píxeles), y la orientación va a estar guiada por el gradiente de los píxeles alrededor de la zona central del segmento 2D como se muestra en la figura 3.28.

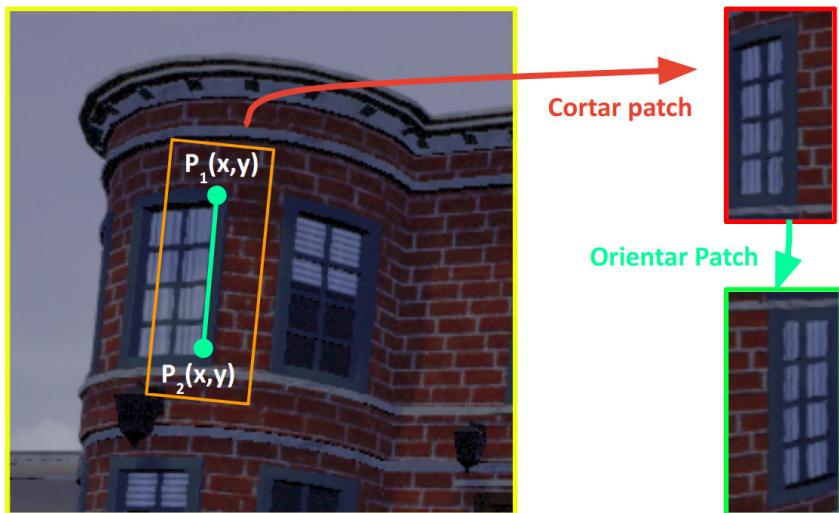


Figura 3.28: Recortado y orientado de patches de segmentos.

A continuación paso a detallar cada uno de estos puntos.

- **Recortado del patch** Los pasos a seguir para el recortado del patch son los siguientes:

- Calcular la longitud del segmento $sHeight$ que une los extremos del segmento detectado: $P_1(x, y)$ y $P_2(x, y)$, dicha longitud se calcula como: $\|\overrightarrow{P_1P_2}\|$
- Calcular el largo ($wHeight$) y el ancho ($wWidth$) de la ventana de píxeles a recortar, alrededor del segmento, que viene dado por:

$$wHeight = hMargin * sHeight$$

$$wWidth = wHeight * pSize_x / pSize_y$$

donde, $hMargin$, es un margen que se va a dejar en los extremos del segmento antes de recortar la ventana de píxeles.

- Una vez calculado los márgenes de la ventana que contiene al segmento, el siguiente paso es hallar una transformación \mathbf{H} que nos pase de dicha ventana a un patch rectangular (en nuestro caso, de 50x100 píxeles), en cuyo centro se encuentre el segmento, tal y como se muestra en la figura 3.29.

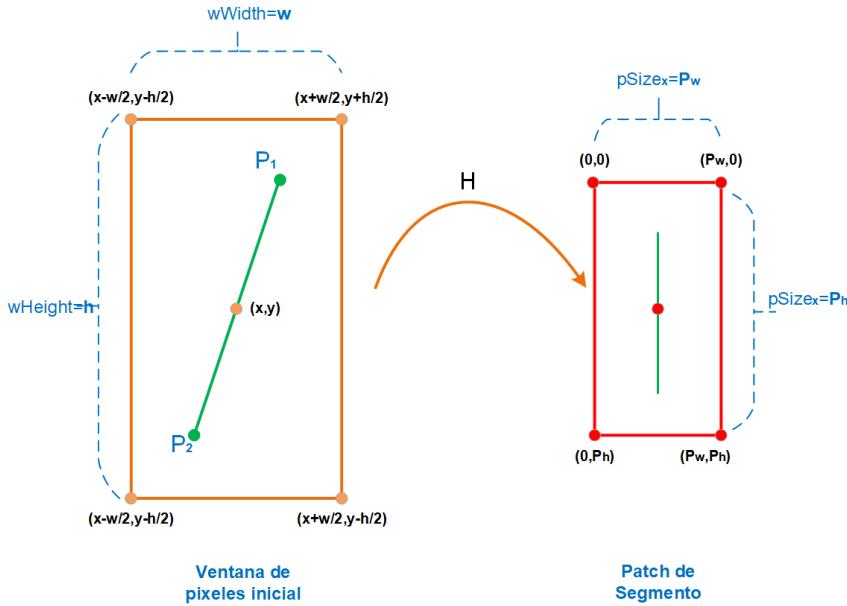


Figura 3.29: Generación de patches de segmentos.

Para hallar la transformación \mathbf{H} se va a usar la función de OpenCV: **findHomography**. Pero primero hay que rotar el segmento hacia la posición vertical, para eso hay que calcular el ángulo Θ de rotación del segmento y después aplicarle una matriz de rotación \mathbf{R} , las fórmulas a emplear son las siguientes:

$$\Theta = \arctan \frac{P_2(y) - P_1(y)}{P_2(x) - P_1(x)} \quad (3.12)$$

$$R = \begin{pmatrix} \cos \Theta & -\sin \Theta \\ \sin \Theta & \cos \Theta \end{pmatrix} \quad (3.13)$$

El código aplicado para hallar la transformación \mathbf{H} es:

```
1 H, _ = cv2.findHomography( srcPts , dstPts )
```

donde: $srcPts$ son las esquinas de la ventana de píxeles inicial y $dstPts$ son las esquinas del patch de segmentos final. Para aplicar dicha homografía a la imagen y obtener el patch, es necesario aplicar una transformación afín. Para este fin se usó la función **warpPerspective** de OpenCV, tal y como se comentó en las secciones anteriores. Con esto ya se tiene recortado el patch, con el segmento en la línea del medio del mismo.

- **Orientado del patch** Para el orientado del patch, sólo nos fijamos en el gradiente de los píxeles centrales del segmento. Se calcula la orientación del gradiente de los píxeles de la línea central, en las direcciones horizontal (G_x) y vertical (G_y) con la función “**Sobel**” de OpenCV. El ángulo que forman dichas gradientes es la orientación de la línea y viene dada por:

$$\Theta = \arctan \frac{G_y}{G_x} \quad (3.14)$$

Por lo tanto, se considerará que el patch está bien orientado si $\cos \Theta \geq 0$, caso contrario se girará 180 grados el patch. La figura 3.30 muestra ejemplos de patches normales y reorientados.

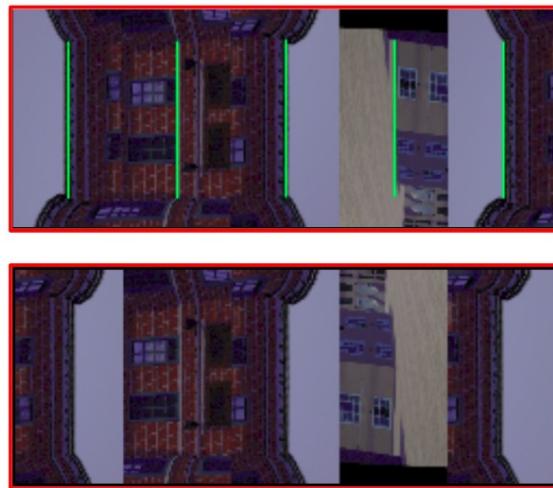


Figura 3.30: En la sección superior, se encuentran patches sin orientar, con el segmento ubicado en el medio del patch, resaltado. En la sección inferior, se muestran los mismos patches orientados en base al $\cos \Theta$.

Una vez recortado y orientado el patch, se procede a generar nuestra base de datos de patches. La estructura de la base de datos generada es similar a la descrita en los apartados anteriores. Un ejemplo de las imágenes formadas a partir de los patches se muestra en la figura 3.31.



Figura 3.31: Patches de segmentos generados.

Capítulo 4

Experimentos

Con el objetivo de evaluar la calidad de la base de datos de características generada “RUF3DM” se van a desarrollar diversos experimentos, en los que se van a comparar los resultados obtenidos entrenando con nuestra base de datos contra los resultados obtenidos entrenando con las bases de datos del dataset de Brown[39]: “liberty, notredame y yosemite”. Esta evaluación sólo considerará los patches obtenidos desde esquinas y desde manchas, quedando como un trabajo futuro evaluar el resultado obtenido con patches de segmentos. Para esta evaluación se van a utilizar diversos descriptores existentes en el estado del arte como son: SIFT, ORB, LBGM[42] y la versión U-512 de BELID (unoptimized BELID o BELID-U). Como métrica de comparación se usa el indicador “Área bajo la curva ROC” (AUC: Area Under the ROC Curve).

A continuación se pasa a describir los experimentos realizados:

4.1. Experimentos en patches de Esquinas

4.1.1. Primer experimento: Prueba en dataset “notredame”.

En este experimento se van a aplicar las siguientes configuraciones:

- Entrenamiento del descriptor BELID-U (“BoostedSSC 512 GradientBasedWL”) con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de esquinas de la base de datos RUF3DM (RUF3D_corners200000).
- Entrenamiento del descriptor BELID-U con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de esquinas de la base de datos “liberty”.
- Prueba de los descriptores SIFT, ORB, LBGM y BELID-U en un dataset balanceado de 100K patches (50K positivos y 50K negativos) de la base de datos “notredame”.

En la figura 4.1 se muestran las curvas ROC obtenidas. Se puede notar, que los resultados obtenidos con BELID-U entrenado con nuestra base de datos de esquinas (curva de color rojo, AUROC

0.971) y con “liberty” (curva de color verde, AUROC 0.972) son similares, lo que indicaría que es equivalente entrenar en nuestra base de datos sintética a entrenar en una base de datos de esquinas reales. Esto es una muy buena noticia pues con un dataset sintético se abren innumerables oportunidades dado que se tiene data ilimitada, se puede obtener de manera semi-automática, el dataset resultante es más variado (presenta mayores cambios de iluminación, diversas condiciones climáticas, etc). Para el caso de LBGM, SIFT y ORB se han usado las implementaciones disponibles en OpenCV. Se puede notar que BELID-U entrenado con un dataset sintético, obtiene un resultado similar al obtenido por LBGM. Sin embargo, BELID-U entrenado en sintético obtiene mejores resultados que SIFT y mucho mejores que ORB (ORB es el descriptor que peores resultados obtiene en términos de accuracy).

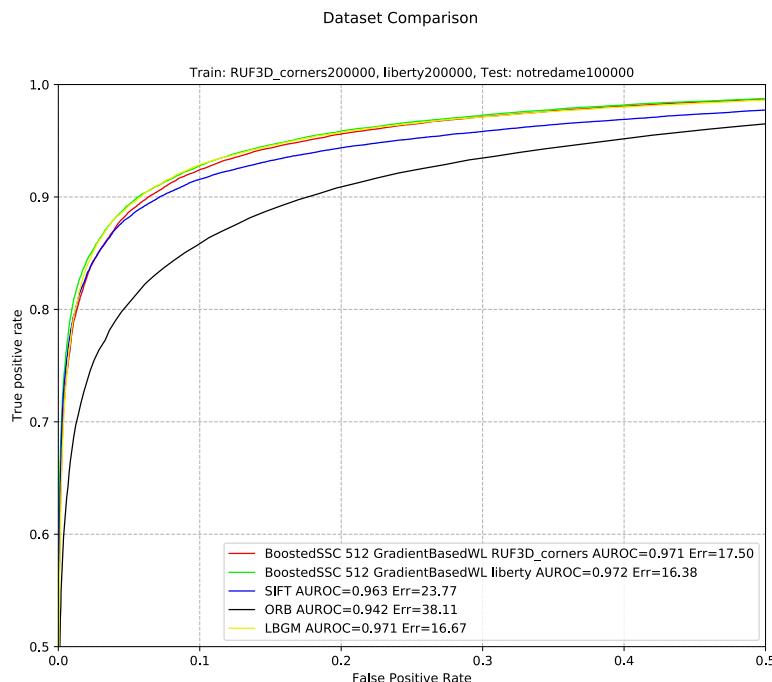


Figura 4.1: Curva ROC obtenida entrenando el descriptor BELID en el dataset de RUF3D_corners200000 y probando en el dataset “notredame”.

4.1.2. Segundo experimento: Prueba en dataset “liberty”.

En este experimento se van a aplicar las siguientes configuraciones:

- Entrenamiento del descriptor BELID-U (“BoostedSSC 512 GradientBasedWL”) con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de la base de datos RUF3DM(RUF3D_corners200000).
- Entrenamiento del descriptor BELID-U con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de esquinas de la base de datos “liberty”.

- Prueba de los descriptores SIFT, ORB, LBGM y BELID-U en un dataset balanceado 100K patches (50K positivos y 50K negativos) de la base de datos “liberty”.

En la figura 4.2 se muestran las curvas ROC obtenidas. Se puede notar que el mejor resultado se obtiene entrenando BELID-U con “liberty” (curva de color verde, AUROC 0.973) algo que era de esperarse, dado que la prueba es también en la base de datos “liberty”). Sin embargo, el entrenamiento con “RUF3D_corner” obtiene un resultado aceptable (curva de color rojo, AUROC 0.964) pues el mismo es ligeramente superior al obtenido por LBGM (curva de color amarillo, AUROC 0.960), mejor que el alcanzado por SIFT (curva de color azul, AUROC 0.953) y mucho mejor que el alcanzado por ORB (curva de color negro, AUROC 0.929), que es el que obtiene peores resultados en términos de accuracy.

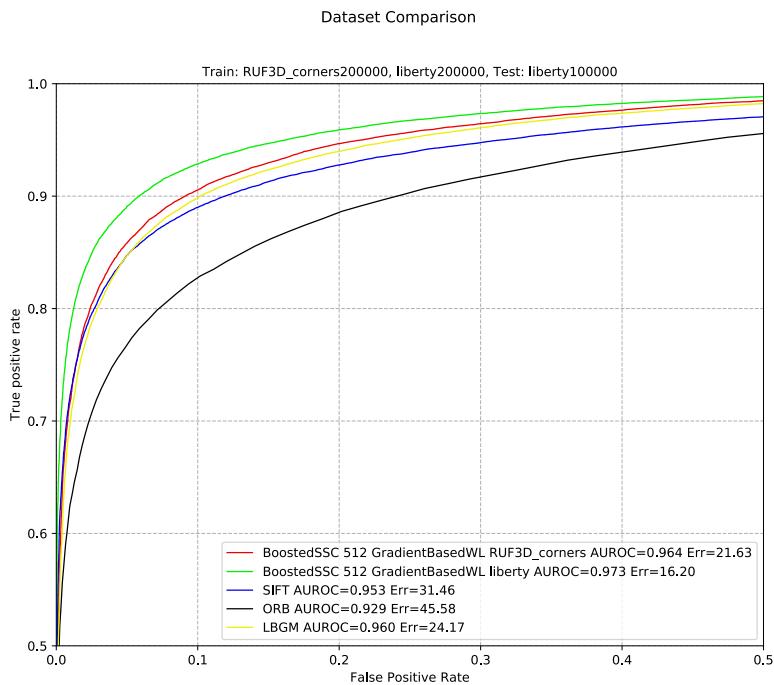


Figura 4.2: Curva ROC obtenida, probando en el dataset “liberty”.

4.1.3. Tercer experimento: Prueba en dataset “RUF3D_corners”.

En este experimento se van a aplicar las siguientes configuraciones:

- Entrenamiento del descriptor BELID-U (“BoostedSSC 512 GradientBasedWL”) con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de la base de datos RUF3DM(RUF3D_corners200000).
- Entrenamiento del descriptor BELID-U con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de esquinas de la base de datos “liberty”.

- Prueba de los descriptores SIFT, ORB, LBGM y BELID-U en un dataset balanceado de 100K patches (50K positivos y 50K negativos) de la base de datos “RUF3D_corners”.

En la figura 4.3 se muestran las curvas ROC obtenidas. Se puede notar que el mejor resultado se obtiene entrenando BELID-U con “RUF3D_corners” (curva de color rojo, AUROC 0.891), algo que era de esperarse dado que la prueba es también en la base de datos “RUF3D_corners”), sin embargo las AUC obtenidas no son muy buenas, llegando a tener en el peor de los casos un AUC de 0.784 (como es el caso de ORB). Esto parece indicar que nuestra base de datos es muy complicada, mas complicada incluso que liberty y notredame. Esto se debe a diversos factores:

- Nuestras imágenes (en consecuencia nuestros patches) presentan un mayor cambio en las condiciones de iluminación: Día, noche, atardecer, puesta del sol, etc.
- Nuestras imágenes presentan una mayor diversidad de clima: Nublado, lluvioso, nieve, etc.
- Nuestras imágenes presentan una geometría más complicada de aprender, pues nuestra base de datos no está limitada a homografías planas o transformaciones geométricas simples.
- Nuestras imágenes son más diversas dado que, como se posee de entrada la información de profundidad, no se deben hacer simplificaciones para estimarla (como buscar escenas simples, interiores de oficinas por ejemplo.)

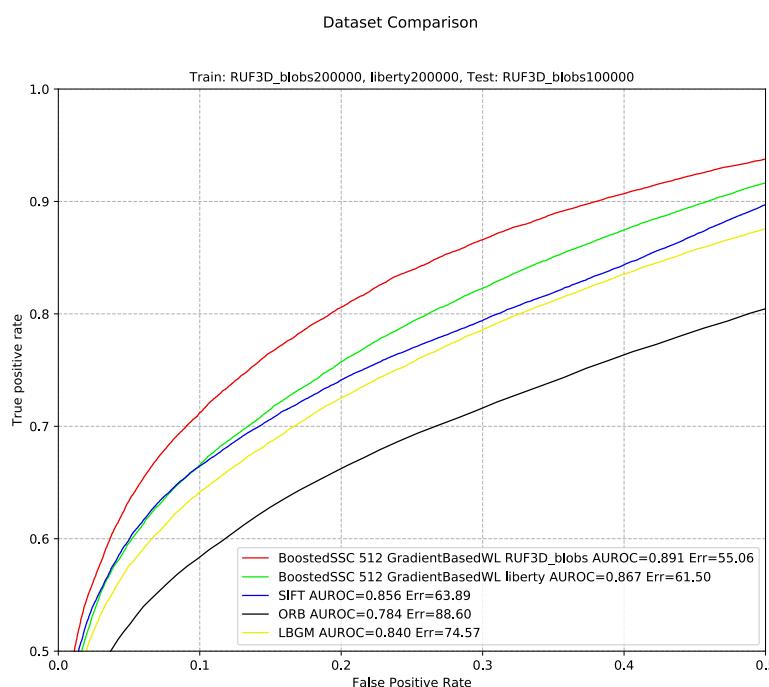


Figura 4.3: Curva ROC obtenida probando en el dataset “RUF3D_corners”.

4.2. Experimentos en patches de Manchas

4.2.1. Primer experimento: Prueba en dataset “notredame”.

En este experimento se van a aplicar las siguientes configuraciones:

- Entrenamiento del descriptor BELID-U (“BoostedSSC 512 GradientBasedWL”) con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de manchas de la base de datos RUF3DM(RUF3D_blobs200000).
- Entrenamiento del descriptor BELID-U con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de manchas de la base de datos “liberty”.
- Prueba de los descriptores SIFT, ORB, LBGM y BELID-U en un dataset balanceado de 100K patches (50K positivos y 50K negativos) de la base de datos “notredame”.

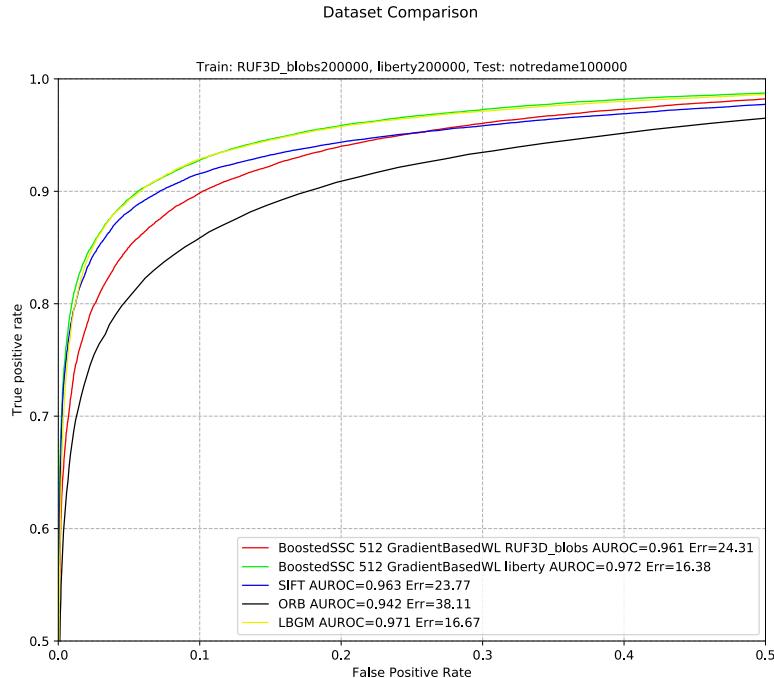


Figura 4.4: Curva ROC obtenida, para las manchas, en el dataset “notredame”.

En la figura 4.4 se muestran las curvas ROC obtenidas. Se puede notar, que los resultados obtenidos con BELID-U entrenando con la base de datos “liberty” (curva de color verde, AUROC 0.972) son mejores que los obtenidos con nuestra base de datos de manchas (curva de color rojo, AUROC 0.961). SIFT y LBGM también nos superan en cuanto a accuracy, pero sin embargo obtenemos mejores resultados que los de ORB (curva de color negro, AUROC 0.942). El menor desempeño en el entrenamiento con RUF3D_blobs puede deberse a lo siguiente:

- La escala de SIFT (diámetro en píxeles) que hemos usado en el escalado de las manchas (valor sugerido por OpenCV) no se asemeja al que usan en la base de datos “notredame”.
- Es necesario afinar el proceso de generación del mapa 3D de las manchas, pues estamos tomando la simplificación de tratar una mancha como si fuera un punto, pero una mancha tiene una escala, y la forma de medir el error entre dos manchas detectadas en distintas imágenes (es decir, detectar si se trata de la misma mancha) no debería de ser la misma que se usó en las esquinas (en cuyo caso se trata de puntos).

4.2.2. Segundo experimento: Prueba en dataset “liberty”.

En este experimento se van a aplicar las siguientes configuraciones:

- Entrenamiento del descriptor BELID-U (“BoostedSSC 512 GradientBasedWL”) con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de la base de datos RUF3DM(RUF3D_blobs200000).
- Entrenamiento del descriptor BELID-U con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de manchas de la base de datos “liberty”.
- Prueba de los descriptores SIFT, ORB, LBGM y BELID-U en un dataset balanceado de 100K patches (50K positivos y 50K negativos) de la base de datos “liberty”.

En la figura 4.5 se muestran las curvas ROC obtenidas. Se puede notar que el mejor resultado se obtiene entrenando BELID-U con “liberty” (curva de color verde, AUROC 0.973) algo que era de esperarse, dado que la prueba es también en la base de datos “liberty”. Sin embargo, el entrenamiento con “RUF3D_blob” obtiene un resultado aceptable (curva de color rojo, AUROC 0.959) pues el mismo es ligeramente inferior al obtenido por LBGM (curva de color amarillo, AUROC 0.960), mejor que el alcanzado por SIFT (curva de color azul, AUROC 0.953) y mucho mejor que el alcanzado por ORB (curva de color negro, AUROC 0.929), que es el que obtiene peores resultados en términos de accuracy.

4.2.3. Tercer experimento: Prueba en dataset “RUF3D_blobs”.

En este experimento se van a aplicar las siguientes configuraciones:

- Entrenamiento del descriptor BELID-U (“BoostedSSC 512 GradientBasedWL”) con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de la base de datos RUF3DM (RUF3D_blobs200000).
- Entrenamiento del descriptor BELID-U con un dataset balanceado de 200K patches (100K positivos y 100K negativos) de manchas de la base de datos “liberty”.
- Prueba de los descriptores SIFT, ORB, LBGM y BELID-U en un dataset balanceado de 100K patches (50K positivos y 50K negativos) de la base de datos “RUF3D blobs”.

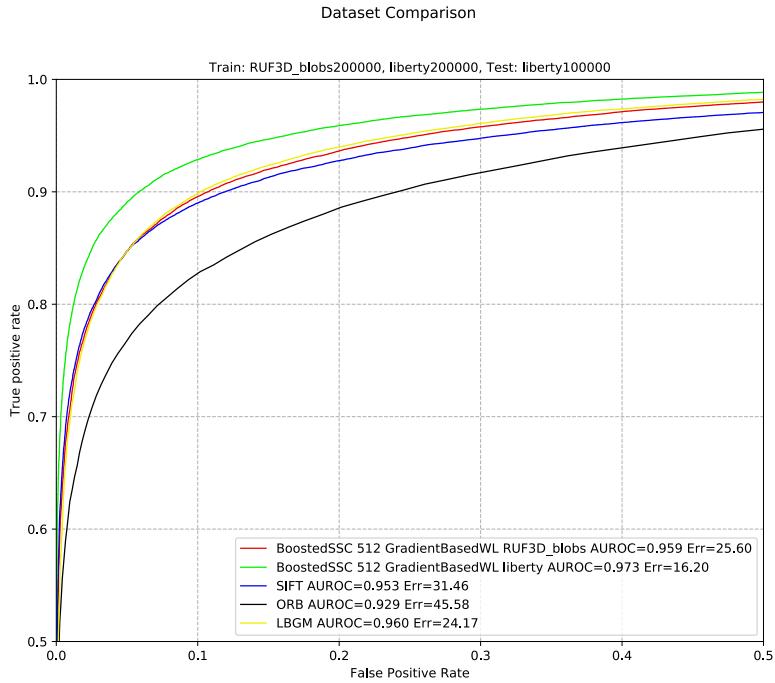


Figura 4.5: Curva ROC obtenida probando en el dataset “liberty”.

En la figura 4.6 se muestran las curvas ROC obtenidas. Se puede notar que el mejor resultado se obtiene entrenando BELID-U con “RUF3D_blobs” (curva de color rojo, AUROC 0.936), algo que era de esperarse dado que la prueba es también en la base de datos “RUF3D_blobs”), sin embargo las AUC obtenidas no son muy buenas, llegando a tener en el peor de los casos un AUC de 0.850 (como es el caso de ORB). Esto parece indicar que nuestra base de datos es muy complicada, mas complicada incluso que liberty y notre dame. Esto se debe a diversos factores:

- Nuestras imágenes (en consecuencia nuestros patches) presentan un mayor cambio en las condiciones de iluminación: Día, noche, atardecer, puesta del sol, etc.
- Nuestras imágenes presentan una mayor diversidad de clima: Nublado, lluvioso, nieve.
- Nuestras imágenes presentan una geometría más complicada de aprender, pues nuestra base de datos no está limitada a homografías planas o transformaciones geométricas simples.
- Nuestras imágenes son más diversas dado que, como se posee de entrada la información de profundidad, no se deben hacer simplificaciones para estimarla (como buscar escenas simples, interiores de oficinas por ejemplo.)

Sin embargo, si comparamos las bases de datos de RUF3D_blobs y RUF3D_corners, vemos que la base de datos RUF3D_blobs es la menos difícil de aprender (AUROC=0.936 vs AUROC=0.891). Nótese que cuando se prueba en una base de datos externa (como en notre dame o liberty), la base de datos de esquinas “RUF3D_corners” es la que obtiene mejores resultados (cercanos incluso a

patches reales), es decir, generaliza mejor pues sus ejemplos son más variados, más diversos. Nos damos cuenta que la aproximación de usar puntos en el espacio para representar manchas no es del todo válida (como sí lo es en el caso de las esquinas). Es necesario entonces cambiar la forma en qué se reproyectan las manchas (considerando la escala de la mancha) y afinar los parámetros de nuestro detector de manchas (SIFT), entre ellos el valor de la desviación estándar ($\sigma = 1,6$ por defecto) dado que este valor es para cámaras que no tienen muy buena resolución (mientras menos resolución tenga la cámara menor valor de σ) y como nuestra cámara es sintética, es perfecta, por tanto aumentando este valor se debería apreciar una mejora considerable (esto quedaría como un trabajo futuro)

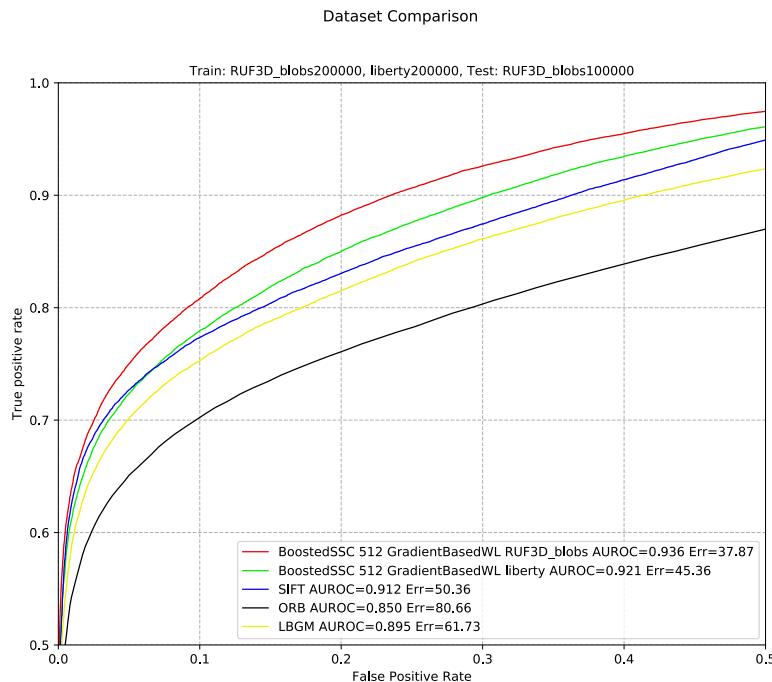


Figura 4.6: Curva ROC obtenida probando en el dataset “RUF3D_blobs”.

Capítulo 5

Conclusiones

En este trabajo se presentó RUF3DM, una base de datos tanto de imágenes (con su información de profundidad) como de patches sintéticos, generada a partir del simulador Carla. Esta base de datos sintéticos ha obtenido:

- Resultados similares al entrenamiento con patches reales (en nuestro caso los patches de Brown) para el caso de esquinas (como se muestra en la sección 4.1) obteniendo un AUROC de 0.971 vs un AUROC de 0.972 de liberty. Se mostró además que el dataset sintético generado es más difícil de aprender que los datasets reales, dado que sus imágenes presentan una geometría más complicada, cambios de iluminación, cambios de clima, etc; lo que hace que esta base de datos presente un buen nivel de generalización.
- Resultados un poco peores al entrenamiento con patches reales (en nuestro caso los patches de Brown) para el caso de manchas (como se muestra en la sección 4.2) obteniendo un AUROC de 0.961 vs un AUROC de 0.972 de liberty. Esto parece indicarnos que la aproximación de tratar una mancha como un punto (para efectos de validar si dos manchas son en realidad las mismas y -si lo son- proceder a mezclarlas), y solamente tener en cuenta la orientación de la mancha para orientar el patch no es una aproximación correcta. Hay muchas maneras de mejorar este resultado como son: (1) considerar la escala de la mancha (en consecuencia el radio de la misma) para medir el error entre y decidir si 2 manchas se deben mezclar, (2) Aumentar el valor de la desviación estándar que usa SIFT para obtener las manchas, pues el valor por defecto ($(\sigma = 1,6)$) es para cámaras que no tienen una buena resolución, pero -dado que nuestra cámara es virtual- es una cámara perfecta.

Además, se ha abordado la totalidad de objetivos planteados en la sección 1.1 de la siguiente forma:

- En la sección 3.2 se ha abordado el objetivo **O1**, mostrando la forma de generar una base de datos de imágenes, con su correspondiente información de profundidad y con información de la posición y orientación de la cámara que capturó dichas imágenes, todo esto a partir de un motor de imágenes virtuales (que en nuestro caso fue Carla).
- En la sección 3.3 se ha abordado el objetivo **O2**, detallando la forma de generar las bases de

datos de patches de esquinas, manchas y segmentos a partir de la base de datos de imágenes sintéticas.

- En la sección 4 se ha abordado el objetivo **O3**, mostrando que para el caso de esquinas es equivalente entrenar con imágenes reales o sintéticas, lo cual abre la puerta a contar con datos infinitos. En el caso del entrenamiento con patches de manchas sintéticas se consiguieron resultados aceptables aunque sin llegar a igualar el resultado obtenido con patches reales, pero con un menor esfuerzo y con una fuente de datos infinita, identificando además el esfuerzo requerido para optimizar los resultados alcanzadas en este trabajo.

5.1. Trabajos Futuros

A pesar de lo mencionado en la sección anterior, hay algunas líneas de mejora, que sería interesante abordar en un futuro, las cuales paso a describir:

- Afinar el proceso de generación del mapa 3D de las manchas para considerar su escala, tanto al momento de discernir si 2 manchas son iguales, como al momento de mezclarlas.
- Automatizar aún más la captura de imágenes sintéticas para la base de datos. La idea es que se le pueda indicar al simulador las características de la escenas que se necesitan capturar y el simulador las obtenga, considerando variaciones en los niveles de iluminación, variaciones de las condiciones climáticas, variación de la perspectiva con que se han capturado las fotos, etc.
- Evaluar el rendimiento de nuestra la base de datos de patches de segmentos para entrenar un descriptor de segmentos.
- Aumentar el tamaño de las bases de datos de patches generadas para poder usarlas en el entrenamiento de descriptores basados en deep learning y evaluar sus resultados en bases de datos reales.

Bibliografía

- [1] Documentación carla. https://carla.readthedocs.io/en/latest/getting_started.
- [2] Hdfcompass. <https://support.hdfgroup.org/projects/compass/>.
- [3] Henrik Aanæs, Anders Lindbjerg Dahl, and Kim Steenstrup Pedersen. Interesting interest points. *International Journal of Computer Vision*, 97(1):18–35, 2012.
- [4] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. Building rome in a day. *Communications of the ACM*, 54(10):105–112, 2011.
- [5] Cuneyt Akinlar and Cihan Topal. Edlines: A real-time line segment detector with a false detection control. *Pattern Recognition Letters*, 32(13):1633–1642, 2011.
- [6] Vassileios Balntas, Karel Lenc, Andrea Vedaldi, and Krystian Mikolajczyk. Hpatches: A benchmark and evaluation of handcrafted and learned local descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5173–5182, 2017.
- [7] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer, 2006.
- [8] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.
- [9] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *European conference on computer vision*, pages 778–792. Springer, 2010.
- [10] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*, pages 1–16, 2017.
- [11] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. Technical report, Sri International Menlo Park Ca Artificial Intelligence Center, 1971.
- [12] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.
- [13] Epic Games. Unreal engine 4. <https://www.unrealengine.com>.

-
- [14] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
 - [15] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Citeseer, 1988.
 - [16] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.
 - [17] J. M. Buenaposada y L. Baumela I. Suárez, G. Sfeir. Boosted efficient local image descriptor. *IbPRIA 2019: 9th Iberian Conference on Pattern Recognition and Image Analysis*, 2019.
 - [18] Nathan Jacobs, Walker Burgin, Nick Fridrich, Austin Abrams, Kylia Miskell, Bobby H Braswell, Andrew D Richardson, and Robert Pless. The global network of outdoor webcams: properties and applications. In *Proceedings of the 17th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 111–120. ACM, 2009.
 - [19] Nathan Jacobs, Nathaniel Roman, and Robert Pless. Consistent temporal variations in many outdoor scenes. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–6. IEEE, 2007.
 - [20] Matthew Johnson-Roberson, Charles Barto, Rounak Mehta, Sharath Nittur Sridhar, Karl Rozaen, and Ram Vasudevan. Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks? *arXiv preprint arXiv:1610.01983*, 2016.
 - [21] Scott Krig. *Computer vision metrics: Survey, taxonomy, and analysis*. Apress, 2014.
 - [22] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.
 - [23] Yunpeng Li, Noah Snavely, Dan Huttenlocher, and Pascal Fua. Worldwide pose estimation using 3d point clouds. In *European conference on computer vision*, pages 15–29. Springer, 2012.
 - [24] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2041–2050, 2018.
 - [25] Zhengqi Li and Noah Snavely. Megadepth: Learning single-view depth prediction from internet photos. In *Computer Vision and Pattern Recognition (CVPR)*, 2018.
 - [26] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.
 - [27] Matthew-Brett. quaternions. <https://matthew-brett.github.io/transforms3d/reference/transforms3d.quaternions.html>.
 - [28] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. 2005.

-
- [29] Marius Muja and David G Lowe. Fast matching of binary features. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 404–410. IEEE, 2012.
 - [30] NASA. <https://www.grc.nasa.gov/www/k-12/airplane/rotations.html>.
 - [31] James Philbin, Ondrej Chum, Michael Isard, Josef Sivic, and Andrew Zisserman. Object retrieval with large vocabularies and fast spatial matching. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.
 - [32] Milan Pultar, Dmytro Mishkin, and Jiří Matas. Leveraging outdoor webcams for local descriptor learning. *arXiv preprint arXiv:1901.09780*, 2019.
 - [33] German Ros, Laura Sellart, Joanna Materzynska, David Vazquez, and Antonio M. Lopez. The synthia dataset: A large collection of synthetic images for semantic segmentation of urban scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
 - [34] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *Computer Vision–ECCV 2006*, pages 430–443. Springer, 2006.
 - [35] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.
 - [36] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Cornell University, 1993.
 - [37] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(8):1573–1585, 2014.
 - [38] Noah Snavely, Steven M Seitz, and Richard Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM transactions on graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.
 - [39] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.
 - [40] Iago Suárez, Enrique Muñoz, José M Buenaposada, and Luis Baumela. Fsg: A statistical approach to line detection via fast segments grouping. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 97–102. IEEE, 2018.
 - [41] Tomasz Trzcinski, Mario Christoudias, and Vincent Lepetit. Learning image descriptors with boosting. *IEEE transactions on pattern analysis and machine intelligence*, 37(3):597–610, 2015.
 - [42] Tomasz Trzcinski, Mario Christoudias, Vincent Lepetit, and Pascal Fua. Learning image descriptors with the boosting-trick. In *Advances in neural information processing systems*, pages 269–277, 2012.

-
- [43] Rafael Grompone Von Gioi, Jeremie Jakubowicz, Jean-Michel Morel, and Gregory Randall. Lsd: A fast line segment detector with a false detection control. *IEEE transactions on pattern analysis and machine intelligence*, 32(4):722–732, 2008.
 - [44] Simon AJ Winder and Matthew Brown. Learning local image descriptors. In *2007 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2007.