



Máster Universitario en Visión Artificial

SD-SLAM+: Mejora de un algoritmo de Visual SLAM
mediante información de profundidad con cámara
RGBD

Memoria del Trabajo Fin de Máster en Visión Artificial

Autor: Omar Garrido Martín

Tutores:

Jose María Cañas Plaza

Diego Martín Martín

Junio 2020

Resumen

Uno de los grandes retos que busca resolver la Visión Artificial y la robótica autónoma es la navegación y el mapeado simultáneos, conocido en inglés por visual SLAM. Para abordarlo se han propuesto numerosos algoritmos a lo largo de los años, pero siempre hay espacio para conseguir mayor precisión, menores tiempos de ejecución o mayor robustez para funcionar en diferentes entornos.

SD-SLAM es un algoritmo que soluciona el problema de SLAM visual con muy buenos resultados, sin embargo no funciona adecuadamente en momentos donde la imagen cuenta con baja textura ya sea por el entorno en sí mismo o por rápidos movimientos de la cámara entre otros. En este TFM se propone una mejora a SD-SLAM que da solución a este problema sin añadir más complejidad computacional, a esta mejora se le denomina SD-SLAM+. Este nuevo algoritmo aprovecha una cámara RGB-D tipo Kinect, que aporta no solo información de color sino también de profundidad en la escena y que hoy en día son sensores muy accesibles por su bajo precio.

SD-SLAM+ consigue muy buenos resultados a la hora de trabajar en entornos con baja textura al combinar el algoritmo original SD-SLAM con el algoritmo DIFODO, que utiliza información de la imagen de profundidad en la estimación de pose. Para caracterizar las prestaciones de SD-SLAM, de DIFODO y para validar SD-SLAM+ se han realizado numerosos experimentos en secuencias internacionales y simulaciones. Los experimentos muestran cómo SD-SLAM+ soluciona este problema satisfactoriamente y consigue una mayor robustez que su antecesor sin perder el funcionamiento en tiempo real.

Índice general

| | |
|---|-----------|
| 1. Introducción | 1 |
| 1.1. Visión artificial | 1 |
| 1.2. Visual SLAM | 4 |
| 1.2.1. Localización visual | 5 |
| 1.2.2. Mapeado | 6 |
| 1.3. Cámaras de profundidad | 7 |
| 1.3.1. Realsense D435 | 9 |
| 1.3.2. Ventajas de la información de profundidad en sistemas SLAM | 13 |
| 1.4. Estructura del documento | 13 |
| 2. Objetivos | 15 |
| 2.1. Descripción del problema | 15 |
| 2.2. Objetivos | 17 |
| 2.3. Metodología | 18 |
| 2.4. Plan de trabajo | 19 |
| 3. Estado del arte | 21 |
| 3.1. Localización con información visual (RGB) | 21 |
| 3.1.1. MonoSLAM | 21 |
| 3.1.2. PTAM | 23 |
| 3.1.3. SVO | 24 |
| 3.1.4. DTAM | 25 |
| 3.1.5. LSD-SLAM | 26 |
| 3.1.6. ORB-SLAM | 27 |

| | |
|---|-----------|
| 3.2. Localización con información visual y/o de profundidad (RGB-D) | 29 |
| 4. Estudio de antecedentes | 35 |
| 4.1. Algoritmo DIFODO | 35 |
| 4.2. Algoritmo SD-SLAM | 37 |
| 4.3. El conjunto de datos para evaluación | 40 |
| 4.3.1. Secuencia rgb_dataset_freiburg1_xyz.bag | 41 |
| 4.3.2. Secuencia rgb_dataset_freiburg1_rpy.bag | 42 |
| 4.3.3. Secuencia rgb_dataset_freiburg1_360.bag | 42 |
| 4.3.4. Secuencia rgb_dataset_freiburg1_floor.bag | 42 |
| 4.3.5. Secuencia rgb_dataset_freiburg1_desk.bag | 43 |
| 4.3.6. Secuencia rgb_dataset_freiburg1_desk2.bag | 44 |
| 4.3.7. Secuencia rgb_dataset_freiburg1_room.bag | 44 |
| 4.3.8. rgb_dataset_freiburg3_nostructure_texture_far.bag | 45 |
| 4.4. Métricas de calidad de algoritmos de autolocalización visual | 47 |
| 4.5. Herramienta odometry_evaluation_file_creator | 48 |
| 4.6. ROSificación de DIFODO | 49 |
| 4.7. Evaluación de DIFODO | 50 |
| 4.7.1. Efecto de la resolución de las imágenes de entrada | 50 |
| 4.7.2. Efecto del nivel de la pirámide | 53 |
| 4.7.3. Conclusiones | 56 |
| 4.8. Comparativa entre DIFODO y SD-SLAM | 56 |
| 4.8.1. Precisión de las estimaciones | 57 |
| 4.8.2. Conclusiones | 61 |
| 5. Autolocalización visual con el algoritmo SD-SLAM+ | 63 |
| 5.1. Introducción | 63 |
| 5.2. Diseño e implementación | 64 |
| 5.2.1. Paso al estado OK DIFODO | 66 |
| 5.2.2. Recuperación de la odometría original | 67 |
| 5.2.3. Relación entre sistemas de referencia | 68 |
| 5.3. Validación experimental | 73 |

| | |
|---|-----------|
| 5.3.1. DIFODO como apoyo a SD-SLAM | 74 |
| 5.3.2. SD-SLAM+ en entornos con zonas de baja textura | 80 |
| 5.3.3. SD-SLAM+ en entornos difíciles | 82 |
| 5.3.4. Evaluación del tiempo de procesamiento | 91 |
| 5.3.5. Conclusiones | 92 |
| 6. Conclusiones | 95 |
| 6.1. Discusión | 95 |
| 6.2. Objetivos conseguidos | 96 |
| 6.3. Trabajos futuros | 97 |
| Bibliografía | 99 |

Índice de figuras

| | |
|--|----|
| 1.1. La visión artificial es una disciplina que engloba a muchas otras. | 2 |
| 1.2. Histórico del error en la competición ImageNet | 3 |
| 1.3. Aplicaciones de realidad aumentada que usan SLAM | 4 |
| 1.4. Emparejamiento de puntos desde dos vistas de una cámara. | 6 |
| 1.5. Cierre de bucle: Comparativa antes y después del cierre de bucle | 7 |
| 1.6. Imagen de profundidad y de color para una cámara realsense | 8 |
| 1.7. Cámara D435 | 10 |
| 1.8. Nube de puntos con textura | 11 |
| 1.9. Cámara D435: (a) emisor infrarrojo activo, (b) emisor infrarrojo desactivado . . | 12 |
| 2.1. Momento en el cual la textura decae en la secuencia rgbd_dataset_freiburg3_floor.bag . Imagen perteneciente a la GUI de SD-SLAM+ | 16 |
| 2.2. Efecto de el movimiento sobre las imágenes capturadas por una cámara. (a) imagen tomada con baja velocidad, (b) es el resultado de un rápido movimiento efectuado sobre la cámara | 17 |
| 2.3. Ciclo de vida de un sprint en metodologías ágiles. | 18 |
| 3.1. Cada uno de los puntos 3D en MonoSLAM, en rojo y amarillo, tiene asociada una distribución gaussiana en función de la incertidumbre sobre ese valor. . . . | 22 |
| 3.2. A la izquierda la reconstrucción densa de DTAM, a la derecha la imagen RGB original | 26 |
| 3.3. Reconstrucción de gran escala usando LSD-SLAM | 27 |
| 3.4. En verde en la imagen izquierda los puntos característicos obtenidos con ORB. A la derecha una imagen con la reconstrucción de esa misma escena. | 28 |

| | | |
|------|---|----|
| 3.5. | Reconstrucción de una oficina con un gran número de <i>surfels</i> en tiempo real. | 31 |
| 3.6. | KinectFusion: A la izquierda la nube de puntos generada por por el sensor. La imagen central y derecha corresponden a la reconstrucción del sistema a partir de la nube de puntos de entrada. | 33 |
| 4.1. | Esquema de una pirámide Gaussiana para dos imágenes [12]. | 36 |
| 4.2. | Escritorio perteneciente a diversas secuencias del conjunto de datos de la universidad técnica de Munich | 42 |
| 4.3. | Fotograma de la secuencia rgbd_dataset_freiburg1_desk.bag . En verde los puntos característicos extraídos por SD-SLAM. | 43 |
| 4.4. | Momento en el cual la textura decae en la secuencia rgbd_dataset_freiburg1_room.bag . Imagen perteneciente a la GUI de SD-SLAM+ | 45 |
| 4.5. | Fotograma de la secuencia rgbd_dataset_freiburg3_nostructure_texture_far.bag que muestra documentos y los puntos característicos extraídos por SD-SLAM en verde. | 46 |
| 4.6. | Final de la secuencia rgbd_dataset_freiburg3_nostructure_texture_far.bag donde se observa poca textura. | 47 |
| 4.7. | Trayectorias 3D de los algoritmos para la secuencia <i>floor</i> . En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de DIFODO y la figura (b) a la estimación de SD-SLAM. | 59 |
| 5.1. | Máquina de estados de SD-SLAM | 64 |
| 5.2. | Máquina de estados de SD-SLAM+ | 65 |
| 5.3. | Cálculo del desplazamiento entre dos poses | 69 |
| 5.4. | Inversa de una pose | 70 |
| 5.5. | Dirección y sentido de los ejes de los sistemas de coordenadas | 71 |
| 5.6. | Adición del desplazamiento a la última pose conocida I | 72 |
| 5.7. | Adición del desplazamiento a la última pose conocida II | 73 |
| 5.8. | Trayectorias 3D estimadas por los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la versión original de SD-SLAM y la figura (b) a SD-SLAM+. | 76 |

| | |
|---|----|
| 5.9. Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM+ en el test 2 y la figura (b) a la estimación de SD-SLAM+ en el test 4. En rojo se encuentran marcadas las posiciones finales para la trayectoria verdadera y estimada. En (a) se aprecia cómo estas están mucho más cercanas que en (b) | 80 |
| 5.10. Comparativa de las trayectorias y reconstrucción del mapa usando los algoritmos SD-SLAM y SD-SLAM+. | 82 |
| 5.11. Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM y la figura (b) a la estimación de SD-SLAM+. Marcado en rojo se encuentran las estimaciones de DIFODO. | 84 |
| 5.12. Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM y la figura (b) a la estimación de SD-SLAM+. Las poses delimitadas por el rectángulo rojo corresponden a las poses estimadas por DIFODO. | 86 |
| 5.13. Trayectorias y reconstrucción del mapa hecha por los algoritmos en la interfaz SD-SLAM. La figura (a) corresponde a SD-SLAM y la figura (b) a SD-SLAM+. En azul se tiene la pose de cada uno de los fotogramas clave, en verde y negro los puntos del mapa, en amarillo la última posición estimada cuando se usa DIFODO. | 87 |
| 5.14. Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM+ para el test 6 y la figura (b) a la estimación de SD-SLAM+ para el test 7. | 89 |
| 5.15. Comparativa de las trayectorias de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. | 90 |
| 5.16. Comparativa del tiempo de procesamiento por imagen para la secuencia <code>rgbd_dataset_freiburg3_flo</code> | |

Índice de tablas

| | |
|---|----|
| 4.1. Tiempos de procesamiento para distintas resoluciones y distintos procesadores . | 51 |
| 4.2. RMSE en metros para el ATE (Absolute Trajectory Error) en distintas resoluciones de DIFODO. | 53 |
| 4.3. Tiempos de procesamiento por fotograma para distintos niveles de pirámide en AMD-FX 8370 | 54 |
| 4.4. Comparación de tiempos de procesamiento para distintos niveles de pirámides y procesadores. | 54 |
| 4.5. RMSE en metros para el ATE (Absolute Trajectory Error) en distintos niveles de pirámide de DIFODO. | 55 |
| 4.6. Comparación del ATE (Absolute Trajectory Error) en metros para DIFODO y SD-SLAM en secuencias de TUM RGB-D SLAM Dataset and Benchmark pertenecientes a la subsección Freiburg 1 del conjunto de datos. | 57 |
| 4.7. Comparación del ATE en metros para DIFODO y SD-SLAM en secuencias de Freiburg 3: Estructura vs Textura. En verde el menor error en cada secuencia. . | 60 |
| 5.1. Comparación del ATE en el Test 1: SD-SLAM vs SD-SLAM+ | 75 |
| 5.2. Comparación del ATE en el test 2: SD-SLAM vs SD-SLAM+ | 77 |
| 5.3. Test 2: Evolución del ATE en SD-SLAM+ | 77 |
| 5.4. Comparación del ATE en el test 3: SD-SLAM vs SD-SLAM+ | 78 |
| 5.5. Test 3: Evolución del ATE en SD-SLAM+ | 78 |
| 5.6. Comparación del ATE en el Test 4: SD-SLAM vs SD-SLAM+ | 79 |
| 5.7. Test 4: Evolución del ATE en SD-SLAM+ | 79 |
| 5.8. Comparación del ATE: SD-SLAM vs SD-SLAM+ | 81 |
| 5.9. Comparación del ATE en el test 5: SD-SLAM vs SD-SLAM+ | 83 |

ÍNDICE DE TABLAS

| | |
|---|----|
| 5.10. Comparación del ATE en el test 6: SD-SLAM vs SD-SLAM+ | 85 |
| 5.11. Test 6: Evolución del ATE en SD-SLAM+ | 88 |
| 5.12. Comparación del ATE en el test 7: SD-SLAM vs SD-SLAM+ | 88 |
| 5.13. Test 8: Evolución del ATE en SD-SLAM+ | 90 |

Capítulo 1

Introducción

En este capítulo se hace un repaso por la historia de la visión artificial. Después se habla de la técnica SLAM (localización y mapeado simultáneo en español), muy utilizada en robótica y que entra dentro de la visión artificial. Finalmente se explicará qué son las cámaras de profundidad y porqué son de importancia en este proyecto. El objetivo de este capítulo es el de introducir contexto y una serie de conceptos básicos al lector para facilitar la lectura.

1.1. Visión artificial

La visión artificial es una disciplina científica que busca la obtención de información a través del análisis de la imagen. La visión artificial es un campo enorme que incluye diversos subcampos, que van desde la parte más física como puede ser: el estudio de la formación de la imagen, el desarrollo de sensores para la adquisición de imagen, sistemas de iluminación, etc.; hasta la parte más cercana a la computación software como es el desarrollo de algoritmos para el procesado y análisis de la imagen.

Es a su vez, un subcampo de la inteligencia artificial, entendiéndose la inteligencia artificial como la idea de que las máquinas sean capaces de pensar como los humanos. La visión artificial está formada o utiliza otros campos como son: la matemática, estadística, informática, aprendizaje máquina, procesamiento de imagen, procesamiento de señal... entre muchos otros, figura 1.1.

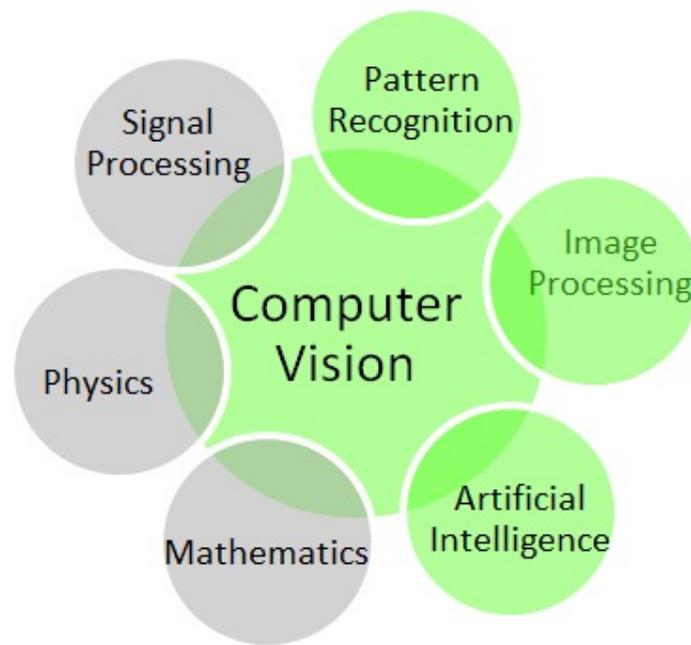


Figura 1.1: La visión artificial es una disciplina que engloba a muchas otras.

La visión artificial nace en la década de los 60 en las universidades como un subcampo de la inteligencia artificial. Ambas tuvieron un fuerte empuje por parte tanto de la comunidad científica como de inversión pública-privada, debido al interés que generó y se consiguieron muchísimos avances que son los fundamentos de la visión artificial que conocemos.

En la década de los 80 este campo se ve empujado por el desarrollo de la ingeniería informática y la creación de procesadores más rápidos, que permiten captar, procesar y reproducir imágenes tomadas por una cámara que podía estar conectada de forma remota. Ambos campos, la visión artificial y la informática, están estrechamente relacionadas pues el procesamiento de imagen es muy costoso computacionalmente, ya que la imagen es una estructura de información de gran tamaño en comparación con otro tipo de señal, por lo que es necesaria la optimización de los algoritmos desarrollados de visión.

La visión artificial cae un poco en el olvido con el denominado invierno de la Inteligencia Artificial hacia la década de los 90. Aunque esto no significa que el campo pare su avance y muchos desarrollos importantes se dieron durante este tiempo, sobre todo en el aprendizaje máquina, un campo del cual se alimenta la visión artificial. Ya por estas fechas se considera un campo mucho más maduro y se utiliza con éxito en el ámbito industrial para automatizar tareas en la industria, como por ejemplo el análisis por imagen de calidad en piezas fabricadas.

2012 es el año en que AlexNet, una red neuronal convolucional profunda, gana la competición *ImageNet Large Scale Visual Recognition Challenge (ILSVRC)*, la mayor competición de reconocimiento de objetos en el mundo de la visión artificial. Sin embargo el mayor logro no fue que ganase la competición, sino de que lograrse tan solo un 15 % de error frente al 25 % del año anterior, como muestra la figura 1.2. Tan solo unos años después, en 2015, se consigue superar la precisión que tiene un ser humano, marcando todo un hito.

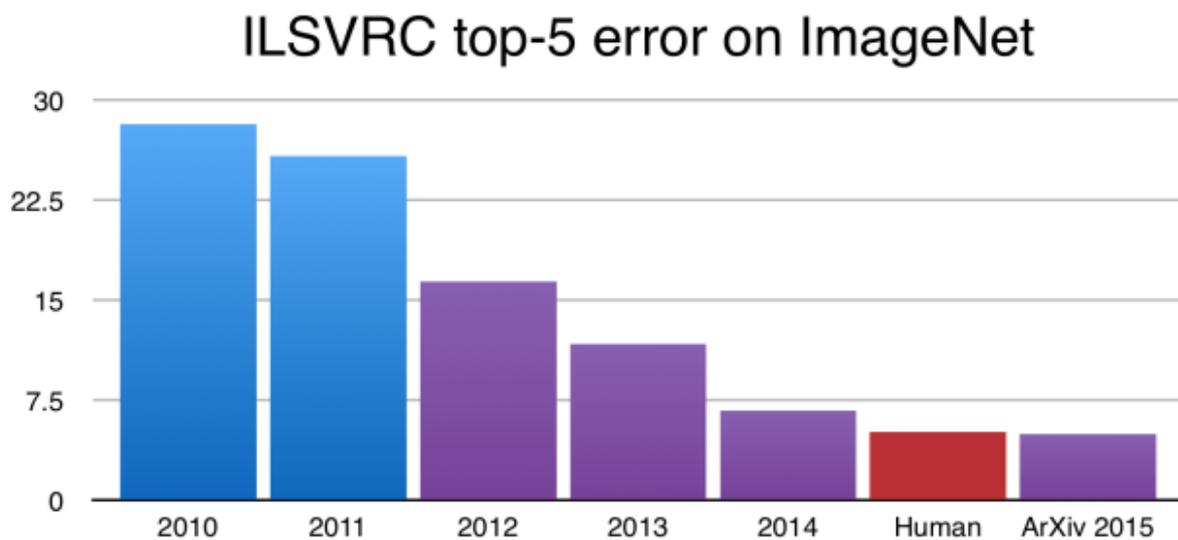


Figura 1.2: Histórico del error en la competición ImageNet

AlexNet supone el inicio de la época dorada de las redes neuronales profundas hasta el día de hoy, con las cuales se han conseguido verdaderos logros en numerosas aplicaciones y campos como: El coche autónomo, avances en imagen de diagnóstico médico, sistemas biométricos como el reconocimiento facial, entendimiento del entorno a nivel semántico, etc., entre muchas otras.

Sin embargo, no todo se soluciona actualmente con *deep learning*, donde queda mucho por explorar, y todavía hay problemas como SLAM, donde se siguen usando los métodos tradicionales o mejor dicho donde el *deep learning* todavía no ha presentado mejoras respecto a estos.

1.2. Visual SLAM

SLAM son las siglas de **Localización y Mapeado Simultáneo** en inglés. Es uno de los problemas abiertos más importantes de la robótica y la visión artificial si hablamos de SLAM visual. Un objetivo con el que nace el SLAM visual es la navegación autónoma de robots. Si se cuenta con un mapa del entorno sobre el que el robot va a navegar, es relativamente sencillo localizar a este en ese entorno y establecer rutas de un punto a otro. El problema llega cuando no existe un mapa existente del entorno sobre el que se pretende navegar. Es entonces cuando nace la necesidad de realizar la tarea de la localización y el mapeado de forma simultáneas. Esto se acaba traduciendo en sistemas que precisan de ejecutarse en tiempo real, añadiendo más complejidad a un problema ya de por sí difícil.

Por el camino los algoritmos de SLAM también han mostrado resultados interesantes en la reconstrucción de entornos 3D, aunque para reconstrucciones muy detalladas se sigue utilizando lo que se conoce como *Structure from Motion* [47] (estructura a partir del movimiento). La diferencia entre el SFM y SLAM es que SFM trabaja con imágenes desordenadas, idealmente distintas vistas del objeto a reconstruir, y es un proceso costoso y que se realiza de forma *offline*, llegando a tardar horas o días, por otro lado visual SLAM trabaja con imágenes consecutivas, ordenadas temporalmente, y precisa de funcionar en tiempo real.

Si bien la navegación robótica fue la que inició el desarrollo de algoritmos de SLAM, en los últimos años campos como la realidad virtual/aumentada y la reconstrucción 3D de objetos han sido los grandes promotores de los últimos avances en SLAM. Distintos sectores como el de los videojuegos ha visto un aumento en las aplicaciones desarrolladas que utilizan SLAM, pero también en sectores más tradicionales como la decoración y sectores industriales, figura 1.3, entre muchos otros.



Figura 1.3: Aplicaciones de realidad aumentada que usan SLAM

1.2.1. Localización visual

La diferencia entre la localización en un mapa conocido con la localización cuando no se cuenta con un mapa previo, es que en el primer caso, se utiliza la información del mapa para determinar la posición y orientación dentro de este y es independiente de posiciones anteriores, mientras que en el caso de no contar con un mapa lo que se calcula típicamente es el desplazamiento relativo con la posición anterior, por lo que hay una dependencia temporal entre estimaciones a diferencia de cuando se cuenta con un mapa.

La parte correspondiente a la localización en SLAM se denomina odometría, cuando la información es visual se le denomina odometría visual. El término de odometría surge en el campo de la robótica, refiriéndose a la estimación incremental o desplazamiento de un robot sobre ruedas, utilizando para este cálculo la geometría del robot. Se habla de odometría visual por primera vez en 2004 [30], en este caso en lugar de la geometría del robot se utilizan imágenes adquiridas por cámaras para esta estimación de pose incremental.

Cuando se utiliza una única cámara para realizar la odometría visual se habla entonces de visión monocular. Para poder realizar la estimación de la pose se precisa de observaciones consecutivas de la misma cámara, obteniendo diferentes perspectivas del mismo entorno, simulándose la captura con numerosas cámaras y perspectivas utilizada en el *Structure from Motion*. Al contar con numerosas perspectivas de la misma escena se puede obtener tanto la reconstrucción 3D como la posición de cada perspectiva, que es el objetivo de los algoritmos de SLAM. El único inconveniente de esta técnica es que no permite obtener una escala real, por lo que los objetos reconstruidos, así como los desplazamientos, se encontrarán en una escala aleatoria, que no sigue el sistema métrico utilizado normalmente. Este problema de la escala se puede solucionar con la inclusión de información de profundidad con los nuevos sensores RGBD de los cuales se habla más adelante. Los algoritmos de odometría visual se pueden dividir en tres categorías:

- **Métodos directos:** Los métodos directos utilizan toda la información de intensidad de los píxeles de la imagen en la estimación del desplazamiento. Esto les permite ser más robustos que los métodos basados en puntos característicos en casos donde hay falta de textura.
- **Métodos basados en puntos característicos:** Estos métodos buscan puntos característi-

cos en la escena, normalmente puntos muy ricos en textura y que son muy representativos y cuentan con mucha información. Se hace un seguimiento de estos puntos a lo largo de las distintas imágenes mediante técnicas de emparejamiento. Con un número suficiente de puntos emparejados, Figura 1.4, se busca estimar la pose de la cámara y la pose 3D de los puntos emparejados mediante la reducción de el error de reproyección. El algoritmo más usado para esta optimización y estimación de los parámetros es *Bundle Adjustment* [46] (ajuste de haces).

- **Métodos híbridos:** Aquellos métodos que combinan tanto métodos directos como basados en características [37].

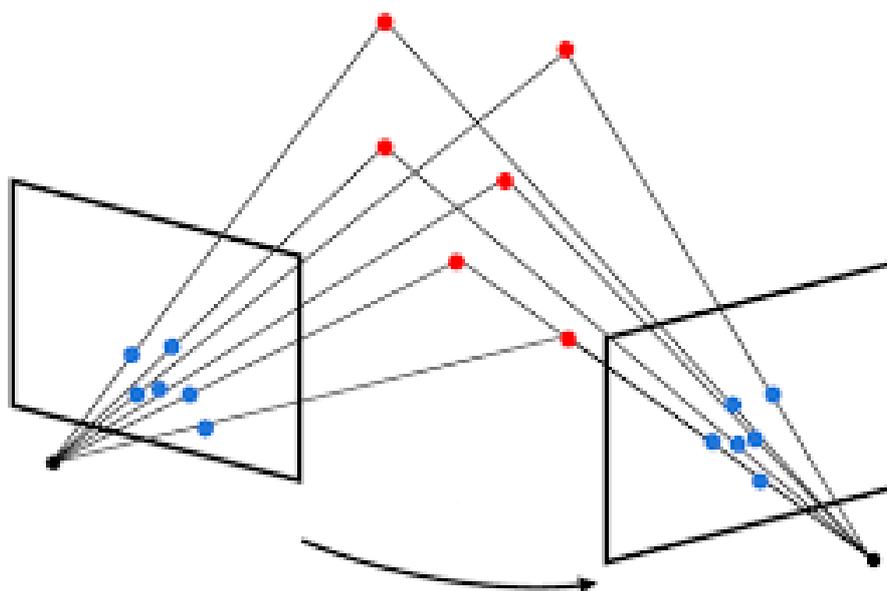


Figura 1.4: Emparejamiento de puntos desde dos vistas de una cámara.

1.2.2. Mapeado

El mapeado consiste en la creación de un mapa tridimensional, orientado a aportar la información necesaria al algoritmo de SLAM para permitirle localizarse. Si bien la odometría visual no precisa de un mapa como tal, el mapeado es una característica de SLAM que le permite utilizar esta información para ser más preciso y robusto en la estimación de la pose. Por un lado, el contar con información de puntos anteriores le permite estimar mejor las nuevas posiciones, reduciendo la deriva incremental del error que se produce típicamente con cada nueva

estimación en la odometría. Por otro lado, una de las características claves de los algoritmos de SLAM es la posibilidad de relocalizarse en caso de perderse y también el cierre de bucle. El cierre de bucle es una técnica que consiste en la optimización global de la trayectoria al pasar por una zona ya visualizada previamente y de la que se tiene información visual en el mapa. Al optimizar la trayectoria también se optimiza al mismo tiempo la reconstrucción 3D por lo que esta técnica es muy útil y favorece la robustez de los algoritmos de SLAM. En la figura 1.5, se aprecia la deriva o acumulación del error en la odometría y como el cierre de bucle optimiza la trayectoria.

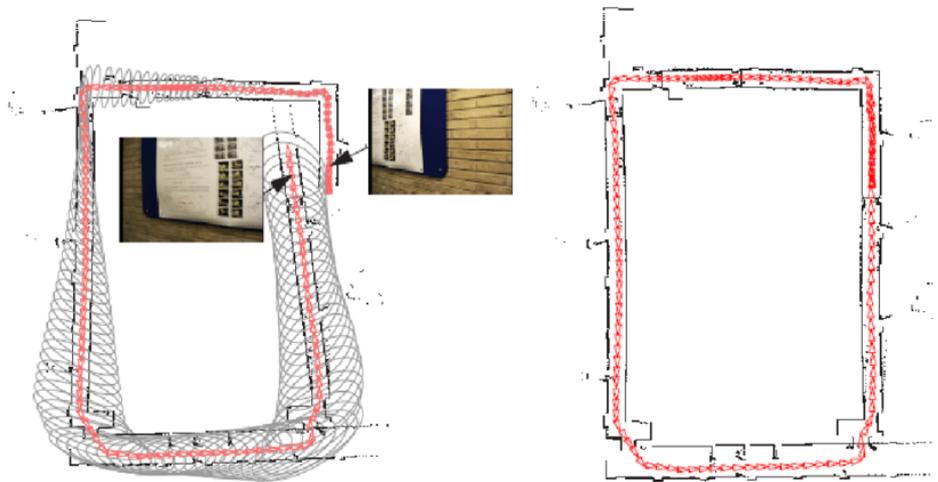


Figura 1.5: Cierre de bucle: Comparativa antes y después del cierre de bucle

1.3. Cámaras de profundidad

En la realización de este trabajo se ha utilizado una cámara que pertenece a la categoría de cámara de profundidad. Una cámara de profundidad es toda cámara que es capaz de obtener una imagen donde el valor de cada uno de sus píxeles corresponde a la distancia entre la cámara y el objeto que se encuentra en ese píxel. Normalmente la representación visual de estas imágenes se hace con colores que van desde los tonos fríos para objetos cercanos, a los tonos calientes para los objetos más alejados, como se puede apreciar en la figura 1.6.

1.3. CÁMARAS DE PROFUNDIDAD

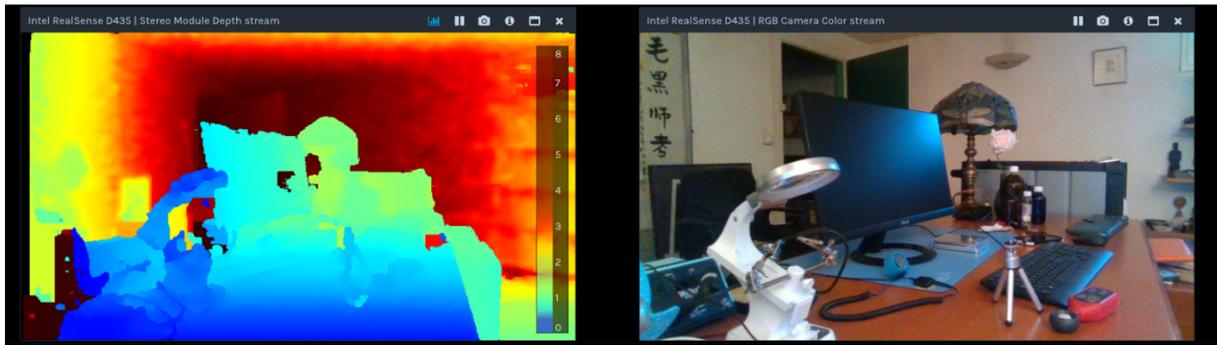


Figura 1.6: Imagen de profundidad y de color para una cámara realsense

Dentro de esta categoría de cámaras de profundidad, se encuentran distintos tipos en función de su funcionamiento físico a la hora de obtener la imagen de profundidad de la escena. Las cámaras de profundidad se agrupan en:

- **Cámaras estereoscópicas:** Las cámaras estereoscópicas se caracterizan por poseer dos lentes, simulando la visión humana. Conocida la geometría entre las dos cámaras se puede crear un mapa de disparidad, que permite conocer a partir de dos imágenes RGB la distancia a cada uno de los píxeles. En esta categoría se puede englobar tanto a las cámaras con dos lentes, como a varias cámaras con captura sincronizada y cuya posición relativa es conocida. También se puede incluir en esta categoría a una única cámara que se desplaza entre tomas, esta última sería el caso de SLAM monocular.
- **Escáneres de luz estructurada:** Este tipo de cámaras forman parte de los sensores activos, ya que a diferencia de los anteriores, estos actúan sobre la realidad, en este caso proyectando un patrón de luz conocido. Este patrón de luz visible o no visible en el caso del infrarrojo, permite que la cámara añada más información a la escena que le permita realizar la reconstrucción 3D. Dependiendo del uso de este patrón de luz encontramos varias categorías:
 1. **Par de cámaras estéreo y luz estructurada visible:** El cómputo del mapa de disparidad se ve mejorado utilizando un patrón de luz estructurada que permite hacer una buena reconstrucción 3D incluso en zonas donde se presentan colores homogéneos, donde los sistemas estéreo basados solo en RGB fallan.
 2. **Triangulación activa:** La proyección de un plano de luz que barre el objeto y la

toma de imágenes permite hacer una reconstrucción 3D del objeto gracias a la deformación del plano de luz al incidir sobre el objeto y la triangulación usando la geometría cámara-emisor de luz.

3. **Una cámara RGB y patrón de luz estructurada infrarroja:** El modelo de cámara representativo de esta categoría es el *Microsoft Kinect™ V1*. Un dispositivo inicialmente pensado para videojuegos pero que permitió a la comunidad científica hacerse con un sensor de bajo coste, con una capacidad de obtener imágenes de profundidad de gran calidad. En este caso, se cuenta con un proyector de luz infrarroja, un receptor de luz infrarroja y una cámara RGB. El patrón de luz estructurada también es conocido y es usado directamente para la creación del mapa de disparidad, sin necesidad de usar un par estéreo pues el proyector de luz estructurada se modela como una lente, sustituyendo a una de las cámaras del par estéreo.

- **TOF (*Time of Flight* o tiempo de vuelo):** Las cámaras de tiempo de vuelo son el último avance en cuanto a imágenes de profundidad. De nuevo, se cuenta con la proyección de un patrón de luz, típicamente un láser infrarrojo, pero en este caso, lo que se mide es el tiempo que tarda la luz en rebotar contra un objeto y volver a la cámara. Conocida la velocidad de la luz y el tiempo que ha tardado ese rayo en volver desde que salió del proyector IR se puede obtener la distancia del objeto contra el que ese rayo rebotó. En este grupo se encuentra la segunda generación del Kinect, *Microsoft Kinect™ V2*.

1.3.1. Realsense D435

La cámara Intel® RealSense™ Depth Camera D435¹ es la cámara de profundidad utilizada a lo largo de este proyecto. Esta cámara cae en la categoría de cámaras de luz estructurada usando un par estéreo. En concreto el dispositivo cuenta con tres cámaras, dos infrarrojas y una de color, y un emisor del patrón de luz estructurada infrarroja.

Las características de este sensor y su bajo coste, así como tamaño, lo hacen ideal para proyectos de robótica, siendo un sensor que podría usarse para vehículos no tripulados como es el caso de los drones.

¹<https://www.intelrealsense.com/depth-camera-d435/>

1.3. CÁMARAS DE PROFUNDIDAD

De entre las características a destacar de este sensor, Figura 1.7, destaca su pequeño tamaño de tan solo 90x25x25mm y un peso de 72g. En el apartado de imagen es capaz de ofrecer hasta 90 imágenes de profundidad por segundo a baja resolución y, de hasta 30 imágenes por segundo en calidad HD. En el apartado de imagen a color la resolución aumenta hasta *full HD*.



Figura 1.7: Cámara D435

El mapa de profundidad se crea a partir de la información del infrarrojo de la escena y del patrón emitido. Por otro lado, se cuenta con la imagen a color, que es independiente de los otros sensores en su funcionamiento. Solo se combina la información de la imagen de profundidad y la de color cuando se registran ambas imágenes, pudiendo obtener una nube de puntos 3D con información de color, como se aprecia en la figura 1.8.

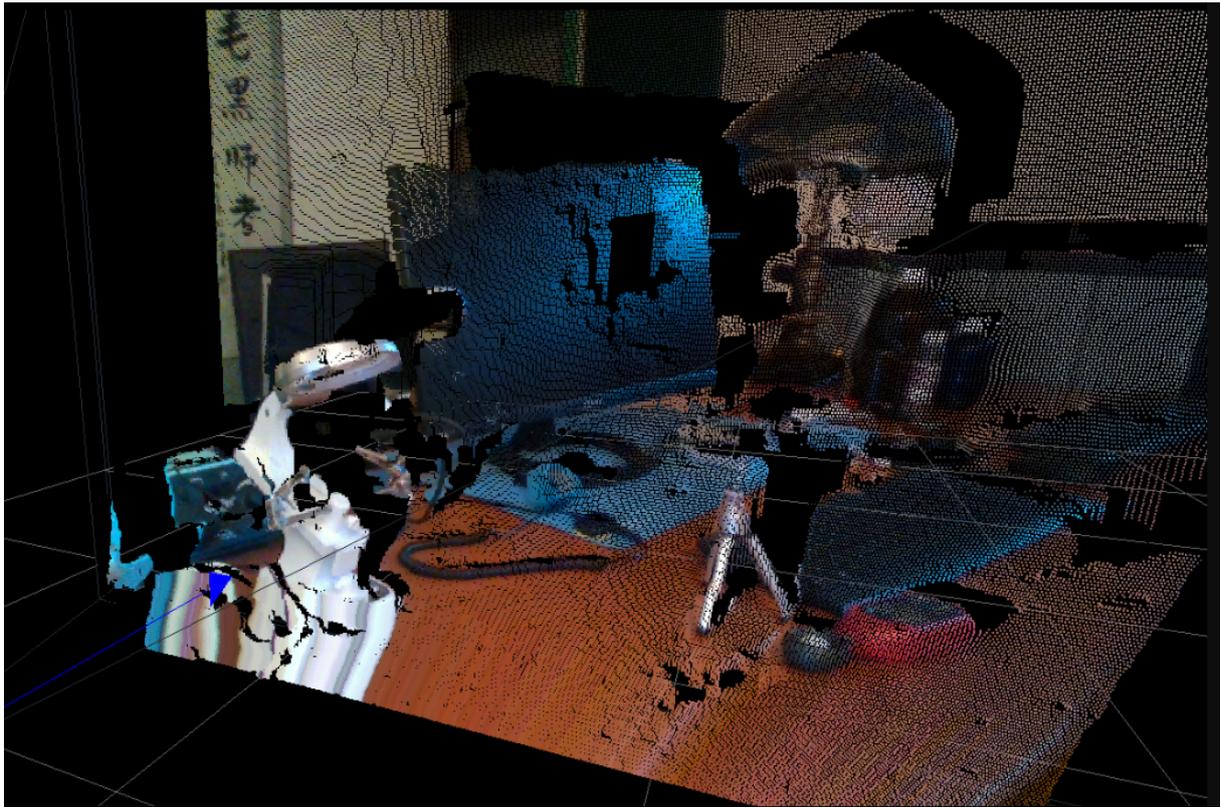
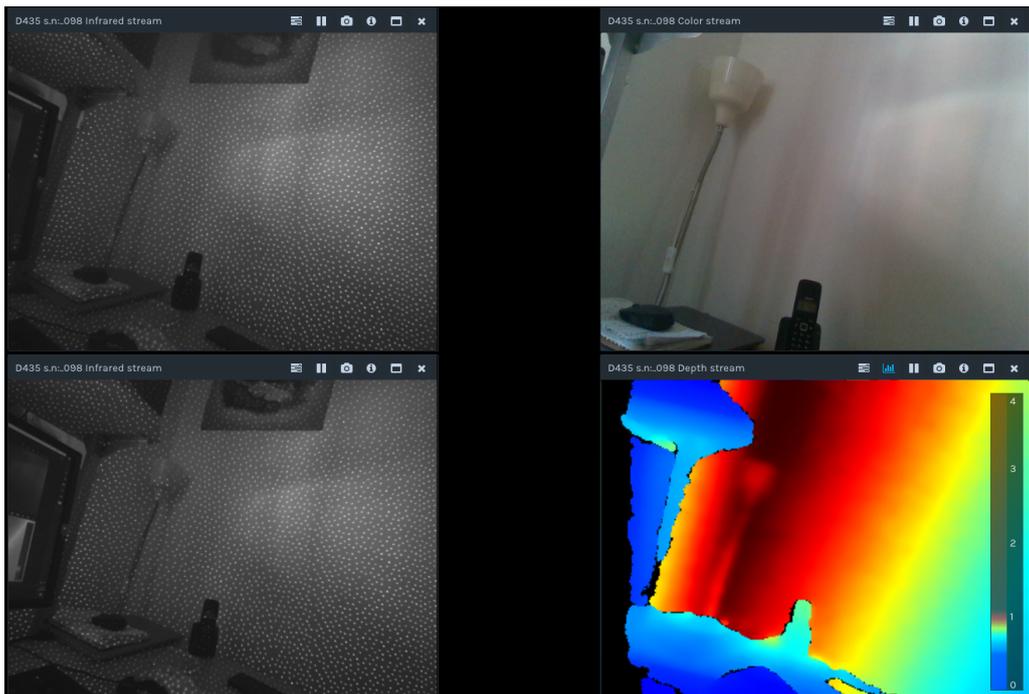


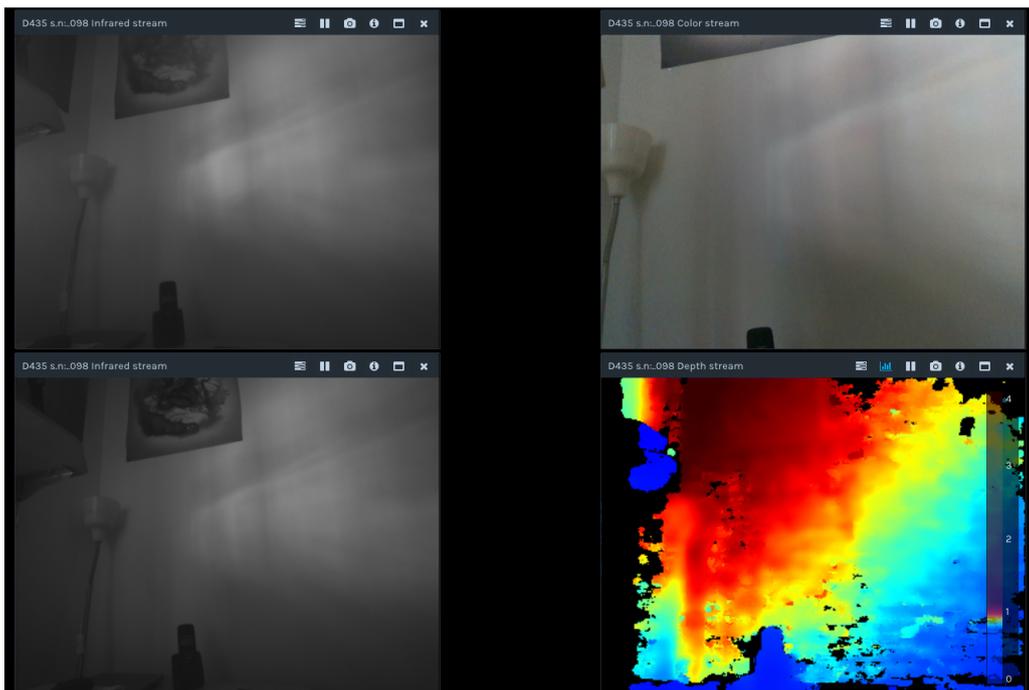
Figura 1.8: Nube de puntos con textura

El efecto de crear un mapa de disparidad, con solo la información de la propia escena y sin aplicar el patrón sobre esta o aplicando de forma activa un patrón, se puede observar en la figura 1.9.

1.3. CÁMARAS DE PROFUNDIDAD



(a)



(b)

Figura 1.9: Cámara D435: (a) emisor infrarrojo activo, (b) emisor infrarrojo desactivado

En (b) se observa cómo la imagen de profundidad presenta alteraciones y zonas donde la profundidad no se ha podido recuperar correctamente. La presencia del patrón de luz infrarroja

sobre la escena, que se aprecia en (a), favorece enormemente la generación de un mejor mapa de profundidad.

1.3.2. Ventajas de la información de profundidad en sistemas SLAM

Las cámaras de profundidad que normalmente aportan también información de color, presentan una serie de ventajas para los algoritmos de SLAM. Entre estas ventajas destaca la posibilidad de contar con escala verdadera. Por otro lado, gracias a esta nueva información se puede optimizar algunos procesos de algoritmos existentes basados tan solo en información RGB, haciendo que estos sean computacionalmente más eficientes. Por ejemplo al utilizar la información de profundidad en el paso del *Bundle Adjustment*(BA). Si en BA se tiene una inicialización en el valor de la profundidad de cada uno de los puntos, se están eliminando parámetros de la ecuación a resolver, permitiendo que el algoritmo pueda converger más rápido. Otra mejora inmediata es la posibilidad de obtener mapas densos, al no tener que triangular cada uno de los puntos de la escena o crear un mapa de disparidad, ambos procesos computacionalmente costosos, sino simplemente recuperando esa información del sensor de forma directa. Gracias a esta nueva información de profundidad también empiezan a surgir algoritmos de SLAM híbridos que utilizan puntos característicos 3D y 2D en el cálculo de la odometría, así como algoritmos basados solo en la información de profundidad.

1.4. Estructura del documento

El presente documento está dividido en capítulos para favorecer su lectura y entendimiento. Tras esta primera introducción para poner en conocimiento del lector conceptos básicos que le permitirán seguir el desarrollo de la memoria con mayor facilidad, se procede a un capítulo donde se enumeran los objetivos planteados. Tras una revisión sobre el estado del arte, se pasa al desarrollo de la mejora del algoritmo SD-SLAM. Finalmente se presentan una serie de experimentos que sirven para validar y demostrar el funcionamiento del nuevo algoritmo desarrollado. El último capítulo corresponde a las conclusiones a las que se han llegado así como posibles mejoras futuras que se pueden explorar para continuar con el perfeccionamiento de SD-SLAM.

Capítulo 2

Objetivos

En esta sección se describe el problema que se busca resolver en este TFM, y se establecen unos objetivos concretos para lograrlo.

2.1. Descripción del problema

Como se ha visto en el anterior capítulo, los algoritmos de visual SLAM buscan la creación de un mapa representativo del entorno que rodea a la cámara así como una estimación de la posición que se ocupa dentro de este. Para lograr esto es necesario utilizar algún tipo de información del entorno que permita representarlo, normalmente esta información es visual. Entre estos algoritmos de visual SLAM se encuentra SD-SLAM [31], desarrollado por el grupo de robótica de la Universidad Rey Juan Carlos, RoboticsLabURJC¹. Este algoritmo a su vez es una mejora sobre otro algoritmo de SLAM, ORB-SLAM2 [26]. Ambos son capaces de funcionar con información únicamente visual, RGB, y también con información de profundidad, RGBD. Sin embargo, como la mayoría de algoritmos de SLAM que utilizan información RGBD, estos solo utilizan la parte de la profundidad para obtener escala real en el mapa y facilitar las operaciones de triangulación al conocer la posición exacta 3D de los puntos. En general incorporar esta información hace a estos algoritmos más eficientes computacionalmente y les proporciona una mejor estimación de la pose.

Sin embargo, existen momentos donde estos algoritmos no funcionan adecuadamente, lle-

¹<http://www.robotica.gsync.es/>

2.1. DESCRIPCIÓN DEL PROBLEMA

gando incluso a no poder dar una estimación de la pose ni de los puntos 3D de la escena en el mapa. Esto ocurre ante la falta de textura en la imagen. La textura en una imagen viene definida por la cantidad de bordes y esquinas que hay en la escena. Si es una escena que presenta homogeneidad en la intensidad de sus píxeles, se dice que hay poca textura. Un ejemplo real donde SD-SLAM se pierde y no consigue estimar la pose se puede apreciar en la Figura 2.1.



Figura 2.1: Momento en el cual la textura decae en la secuencia `rgbd_dataset_freiburg3_floor.bag`. Imagen perteneciente a la GUI de SD-SLAM+

Estas escenas sin textura pueden darse debido a la homogeneidad en el color de los objetos como es el caso anterior, o también se puede producir cuando se hacen movimiento rápidos o bruscos, produciendo un efecto de emborronamiento en la imagen que imposibilita la percepción de los puntos característicos, como se aprecia en la Figura 2.2. Cuando la imagen sufre esta degradación, los bordes de la escena desaparecen, difuminándose la separación entre los distintos objetos de la escena. Otro caso, si bien el más extremo y quizás el menos común donde se puede observar pérdida de textura, se da cuando la iluminación de la escena no es suficiente o directamente no hay luz.



Figura 2.2: Efecto de el movimiento sobre las imágenes capturadas por una cámara. (a) imagen tomada con baja velocidad, (b) es el resultado de un rápido movimiento efectuado sobre la cámara

2.2. Objetivos

En definitiva la pérdida de textura en la imagen puede producirse por diversas causas produciendo el fallo de los algoritmos de SLAM visual como es el caso de SD-SLAM. Este trabajo busca precisamente evitar esto usando cámaras de profundidad. En concreto, el objetivo principal es evitar que la odometría o estimación de la pose de SD-SLAM falle cuando la textura en la imagen es escasa. Independientemente de por qué se haya producido, la falta de textura imposibilita la obtención de puntos característicos sobre la imagen RGB. La imagen de profundidad todavía aporta mucha información que puede ser usada para conseguir la odometría visual. Los objetivos concretos de este trabajo fin de master son:

1. **Explorar algoritmos de odometría visual basados únicamente en cámaras de profundidad.**
2. **Integrar en SD-SLAM el algoritmo elegido en el paso anterior para mejorar la robustez frente a entornos de baja textura.**
3. **Validación experimental de todos los algoritmos involucrados.**

Como único requisito se precisará de que el algoritmo desarrollado funcione en tiempo real ya que SD-SLAM actualmente se ejecuta en tiempo real, una característica que no se puede

sacrificar con la mejora.

2.3. Metodología

La metodología utilizada en este trabajo de fin de Máster ha sido una metodología ágil o *agile* en inglés. Esta metodología se caracteriza por permitir adaptar la forma de trabajo a las condiciones del proyecto. Esto permite una respuesta flexible ante los nuevos descubrimientos durante el desarrollo del proyecto, algo necesario en un proyecto donde se conocen los objetivos pero no se está seguro del camino a seguir para llegar a ellos.

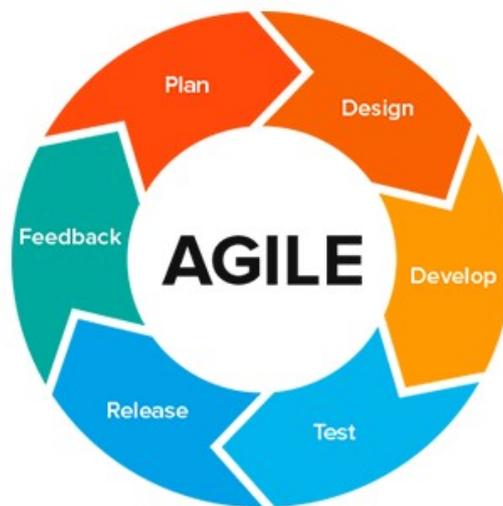


Figura 2.3: Ciclo de vida de un sprint en metodologías ágiles.

Para conseguir esta flexibilidad en el desarrollo, esta metodología se organiza siguiendo *sprints*. Un *sprint*, es un intervalo corto de tiempo, normalmente una o dos semanas, donde se realiza un ciclo entero del desarrollo de un software, Figura 2.3. El comienzo de un esprint empieza por el Plan, la definición de objetivos que se quiere lograr en el intervalo de tiempo que dura el esprint. Con unos objetivos en mente, se pasa a la fase de diseño, *Design*, seguido de la fase de desarrollo o *Develop*, y finalmente las fases de Test y lanzamiento de versión o *Release*. Durante el desarrollo de esta trabajo se han mantenido reuniones semanales con los tutores del trabajo fin de máster, a veces cada dos semanas, en las que se presentaban los progresos de la semana anterior y se planeaban los objetivos para la semana o esprint siguiente. Estas reuniones

son el equivalente a las fases del *Feedback* y el Plan. A lo largo del esprint se procedía al diseño y desarrollo de los objetivos así como Test y lanzamiento de versiones cuando correspondía.

Cada uno de estos sprints se han documentado en forma de entradas en un blog², creado específicamente para este trabajo. Este blog se encuentra en github, una plataforma para albergar repositorios de proyectos de software, donde se encuentra el código³ que acompaña a esta memoria.

2.4. Plan de trabajo

El plan de trabajo establecido y seguido a lo largo del desarrollo de este trabajo de fin de máster consta de las siguientes etapas:

- **Introducción a los algoritmos de SLAM:** El primer será familiarizarse con todas aquellas técnicas y algoritmos característicos de SLAM. Para el desarrollo de esta fase el tutor propondrá una serie de lecturas de trabajos anteriores desarrollados dentro de la Universidad realizados por antiguos alumnos.
- **Instalación y familiarización de software y hardware necesario:** Tras obtener un conocimiento más profundo sobre los algoritmos de SLAM se procederá a la instalación y estudio del paquete de software de SD-SLAM [31] y también se procederá a la instalación y uso de la cámara RGBD Intel® RealSense™ Depth Camera D435.
- **Búsqueda de un algoritmo de odometría basado en profundidad:** Se estudiará el estado del arte de técnicas de SLAM y odometría basadas únicamente en cámaras de profundidad. Se implementará o utilizará un desarrollo existente que después habrá de ser validado experimentalmente.
- **Mejora de SD-SLAM:** Se procederá a la mejora de SD-SLAM mediante la integración del algoritmo de profundidad previamente elegido.

²<https://roboticslaburjc.github.io/2019-tfm-omar-garrido/>

³<https://github.com/RoboticsLabURJC/2019-tfm-omar-garrido>

2.4. PLAN DE TRABAJO

- **Validación experimental:** Se deberá validar experimentalmente el nuevo algoritmo desarrollado SD-SLAM+ durante el transcurso de este trabajo de fin de máster.

Capítulo 3

Estado del arte

En este capítulo se explican varios algoritmos que actualmente solucionan el problema de la localización ya sea con información visual RGB o incluyendo también profundidad RGB-D. Se entrará en detalle en la descripción los algoritmos más importantes o que han tenido más repercusión en los últimos años y se estudiarán las distintas formas de localización disponibles. Esta sección se divide en dos secciones en función de la información de entrada que utilizan estos algoritmos, primero los que usan pura información visual, de color, y segundo los que además usan información de profundidad.

3.1. Localización con información visual (RGB)

Esta sección recorre y detalla los algoritmos más importantes que utilizan imágenes RGB para solucionar el problema de la localización. Son de gran utilidad porque la mayoría de avances en SLAM se han dado con cámaras RGB, ya que son sensores baratos y ligeros que aportan muchísima información y los sensores RGB-D no llegan al mercado hasta mucho más tarde. Es por esto que la mayoría de mecanismos y técnicas se desarrollaron pensando en imágenes de intensidad.

3.1.1. MonoSLAM

El término MonoSLAM viene de “Monocular SLAM” y hace referencia al uso de una única cámara, RGB, para la generación de mapas y localización en estos de forma simultánea. El

3.1. LOCALIZACIÓN CON INFORMACIÓN VISUAL (RGB)

algoritmo original fue propuesto por Andrew Davidson en el año 2002 [6] y mejorado en el año posterior [7], aunque no sería hasta el año 2007 [8] cuando se presentaría de forma oficial.

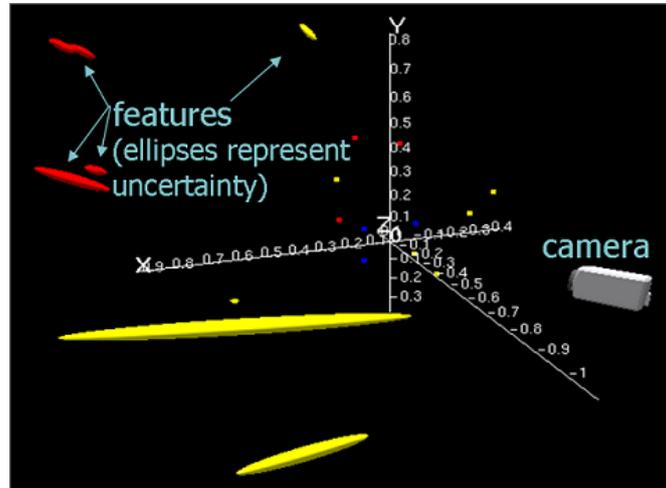


Figura 3.1: Cada uno de los puntos 3D en MonoSLAM, en rojo y amarillo, tiene asociada una distribución gaussiana en función de la incertidumbre sobre ese valor.

MonoSlam utiliza el filtro extendido de Kalman para la estimación de la posición y orientación de la cámara, así como la estimación de la posición de los puntos 3D que pasarán a formar parte del mapa. El filtro de Kalman es un método probabilístico que busca estimar una serie de parámetros los cuales se definen en un vector de estados siguiendo una distribución gaussiana. Cada uno de estos parámetros contará con una media y varianza que se estima, esto se representa visualmente para cada uno de los puntos del mapa como se ve en la figura 3.1. Es un método iterativo, pues la actualización de los estados se produce con la llegada de una nueva medida del sensor, es decir con la llegada de cada nueva imagen. Esto hace que con cada iteración el método converja hacia una solución, en este caso la estimación de los puntos 3D en el espacio y la posición y orientación de la cámara respecto a estos.

Una de las desventajas de este método es que una de las necesidades del filtro extendido de Kalman es que precisa de un estado inicial, por lo que es necesaria una inicialización. Para la inicialización es necesario utilizar al menos cuatro puntos coplanares cuya posición o distancia entre ellos sea conocida, y posteriormente mediante la técnica de *PnP (Perspective-n-Point)*, obtener la posición relativa de la cámara respecto a estos puntos.

Para la actualización o estimación de la posición de cada uno de los puntos es necesario el seguimiento o emparejamiento de estos puntos a lo largo de las sucesivas imágenes, es por ello

que se utiliza *Good Features to Track* de Shi-Tomasi [39].

Debido a la utilización del filtro extendido de Kalman, se tiene un estado en forma de vector de estados de posiciones por todos los puntos 3D del mapa que se desea añadir. Esto supone un problema en cuanto a escalabilidad ya que cuanto mayor sea el vector de estados mayor será el número de operaciones a realizar y por lo tanto mayor el tiempo de cómputo. La capacidad de MonoSLAM de ser tiempo real depende directamente del número de puntos que se tengan en el mapa. El algoritmo demuestra capacidad de funcionar en tiempo real en un entorno de hasta 100 puntos.

3.1.2. PTAM

PTAM (Parallel Tracking and Mapping) llega en el año 2007, desarrollado por Klein y Murray [22], con el objetivo de solucionar el mismo problema que MonoSLAM pero de una forma diferente, intentando dar solución a las desventajas que tenía MonoSLAM.

El mayor de los problemas de MonoSLAM es su baja escalabilidad en cuanto aumenta el número de puntos del mapa, ya que según aumenta el número de puntos también aumenta el tiempo de cómputo. Esto se debe a que en cada iteración se estiman o calculan la posición tanto de la cámara como cada uno de los puntos del mapa. PTAM presenta una idea sencilla, y consigue un gran rendimiento computacional al dividir el algoritmo en dos partes, Localización y Mapeado. La parte de localización es la que es fundamental en un algoritmo de SLAM, por lo que debe funcionar en tiempo real. La parte del mapeado, que es la parte que hace que MonoSLAM no escale bien o no pueda manejar muchos puntos, no precisa de ser ejecutada en cada iteración. PTAM separa de manera muy marcada y eficaz ambas partes en dos hilos asíncronos.

Para la localización se utiliza FAST [33] para extraer puntos característicos de la imagen a distinta resolución usando una pirámide de imágenes. Se actualiza la posición de la cámara con un modelo de velocidad constante, obteniendo la nueva posición tras un instante Δt .

Después se selecciona un subconjunto de los puntos 3D y se proyectan en la nueva imagen con la actualización de la posición de la cámara previa. Se busca emparejar estos puntos en la nueva imagen, alrededor del punto donde se han proyectado, pues se espera que se encuentre cerca. Una vez emparejados, se trata de minimizar el error de reproyección usando el algoritmo iterativo de Gauss-Newton [19].

3.1. LOCALIZACIÓN CON INFORMACIÓN VISUAL (RGB)

Finalmente, se añaden más puntos del mapa y se realiza de nuevo el proceso anterior. Al contar con muchos más puntos, la posición estimada en este caso es mucho más precisa, los puntos se emparejarán mucho más rápido al buscar en un entorno más cercano que antes, debido a la estimación del paso anterior y se vuelve a minimizar el error de reproyección.

El hilo del mapeado por su parte, no se ejecuta constantemente. También requiere de una inicialización, como MonoSLAM, pero en este caso se puede prescindir de una plantilla conocida. Se usa para la inicialización el seguimiento de puntos característicos entre dos imágenes consecutivas con suficiente desplazamiento entre estas. Mediante el algoritmo de cinco puntos [43] calcula el mapa inicial. Tras la inicialización se producen los siguientes pasos:

1. **Se añaden nuevos fotogramas clave o *Keyframes*:** Para que una nueva imagen u observación de la realidad se añada al mapa han de darse ciertas condiciones como que hayan pasado suficientes iteraciones con respecto al último *Keyframe* almacenado y que la localización sea de calidad.
2. **Adición de nuevos puntos 3D:** Se buscan los puntos característicos del nuevo *Keyframe* en los *Keyframes* anteriores. Se emparejan estos puntos y se calcula su posición mediante triangulación.
3. **Refinamiento del mapa:** Con el objetivo de minimizar los errores en la ubicación de los puntos 3D que forman el mapa se utiliza el algoritmo de optimización *Bundle Adjustment* [45]. Este algoritmo es un algoritmo pesado, por lo que solo se realiza con los *Keyframes* cercanos al incluir nuevos *Keyframes* y ejecutando una corrección total con todos los *Keyframes* en pocas ocasiones.

PTAM consigue solventar la principal desventaja de MonoSLAM al ser capaz de presentar un mapa con miles de puntos frente a la poca cantidad de puntos que puede manejar MonoSLAM de apenas unos 100 puntos, obteniendo mapas de mucha mejor calidad.

3.1.3. SVO

SVO (Fast Semi-Direct Monocular Visual Odometry) de Forster et al., 2014 [15], destaca por su gran rapidez, siendo capaz de funcionar en tiempo real incluso en procesadores de bajas prestaciones, por lo que rápidamente se convierte en un algoritmo muy utilizado en sistemas

embebidos y en robótica. El método sigue una estructura muy similar a la empezada por PTAM, separa en dos hilos independientes el algoritmo, localización y mapeado. SVO es un método híbrido pues utiliza métodos directos y basados en características, beneficiándose de las bondades de ambos métodos.

Ambos métodos se usan para la localización. El método directo se aplica para minimizar el error fotométrico entre el fotograma actual y el anterior. Si esto se aplicase sobre todos los píxeles de la imagen el tiempo de cálculo sería demasiado alto, por lo que este cálculo se hace solo sobre un parche 4x4 alrededor de los píxeles donde se han proyectado puntos característicos del fotograma anterior. De esta forma utiliza puntos característicos con un método directo. Tras esta primera estimación de pose, se realiza un ajuste más fino, se usa un parche más grande de 8x8 para conseguir una mejor precisión al estimar el desplazamiento entre las imágenes.

En cuanto al mapeado, es muy parecido a PTAM, también utiliza *Keyframes*. La gran diferencia entre ambos algoritmos es que en este caso no todos los puntos se añaden con cada *Keyframe* sino que se calcula la posible posición de estos puntos en el espacio 3D como una distribución de probabilidad. Al principio, el punto se ha observado solo desde dos posiciones por lo que tiene una alta incertidumbre (alta desviación típica) y solo cuando se ha observado desde suficientes vistas y se ha reducido esta incertidumbre hasta un valor fiable, es cuando se añade ese punto al mapa. Por lo que, en este caso, los puntos tardan más en añadirse al mapa final pero lo hacen de forma más fiable y con una posición más precisa que en PTAM.

SVO es mucho más rápido que PTAM en el tracking y mucho más fiable en los puntos 3D que se añaden al mapa, sin embargo, al necesitar varias vistas del mismo punto para estar seguro y añadirlo al mapa, presenta un mapa menos denso que PTAM. Esto último también puede representar una desventaja para SVO puesto que frente a cambios bruscos en el movimiento puede llegar a perderse al necesitar varias vistas del mismo punto para añadirlo al mapa. Esto es por lo que el algoritmo inicialmente se propuso para cámaras que apunten al suelo, ya que esa condición evita que vayan a producirse variaciones bruscas en la escena.

3.1.4. DTAM

DTAM (Dense Tracking and Mapping), publicado por Richard A. Newcombe et al. en 2011 [29], es un algoritmo de SLAM que a diferencia de los anteriores se centra en la generación del mapa, y en concreto en obtener un mapa denso y detallado, como se puede apreciar en la

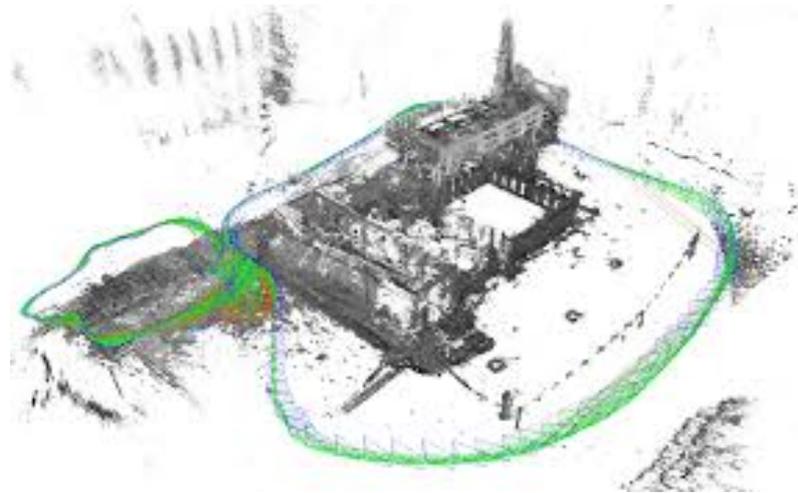


Figura 3.3: Reconstrucción de gran escala usando LSD-SLAM

imágenes, pero normalizado por la varianza. Se incluye una nueva técnica para medir el alineamiento entre imágenes mediante el algoritmo de Gauss-Newton ponderado. Tiene en cuenta el ruido en la estimación de la profundidad que depende del tiempo o número de veces que se ha observado cada uno de los puntos. El hilo de estimación de profundidad se encarga de proyectar en cada nuevo *Keyframe* los puntos del *Keyframe* anterior, y tras un filtrado de valores atípicos se usa como base para la localización de los fotogramas siguientes [11].

El mapeado se encarga de añadir los *Keyframes* que el estimador de profundidad ha dejado de utilizar y optimiza el mapa de forma continua, utilizando para ello una optimización de grafos de posición [23].

3.1.6. ORB-SLAM

Mur et al. presentan ORB-SLAM en 2015 [27] y su versión mejorada ORB-SLAM2 en 2016 [26]. ORB-SLAM ofrece la posibilidad de trabajar con una única cámara, cámaras estéreo incluso con cámaras de profundidad RGB-D. La característica principal de este algoritmo es que utiliza descriptores ORB [36] para extraer y emparejar puntos característicos. Otra característica interesante es la inclusión de un modelo de *bag of words* [16] para almacenar los fotogramas claves y detectar cierre de bucles para su relocalización en caso de necesidad. Un ejemplo del mapa obtenido y de los descriptores para ORB-SLAM se observa a continuación en la Figura 3.4.

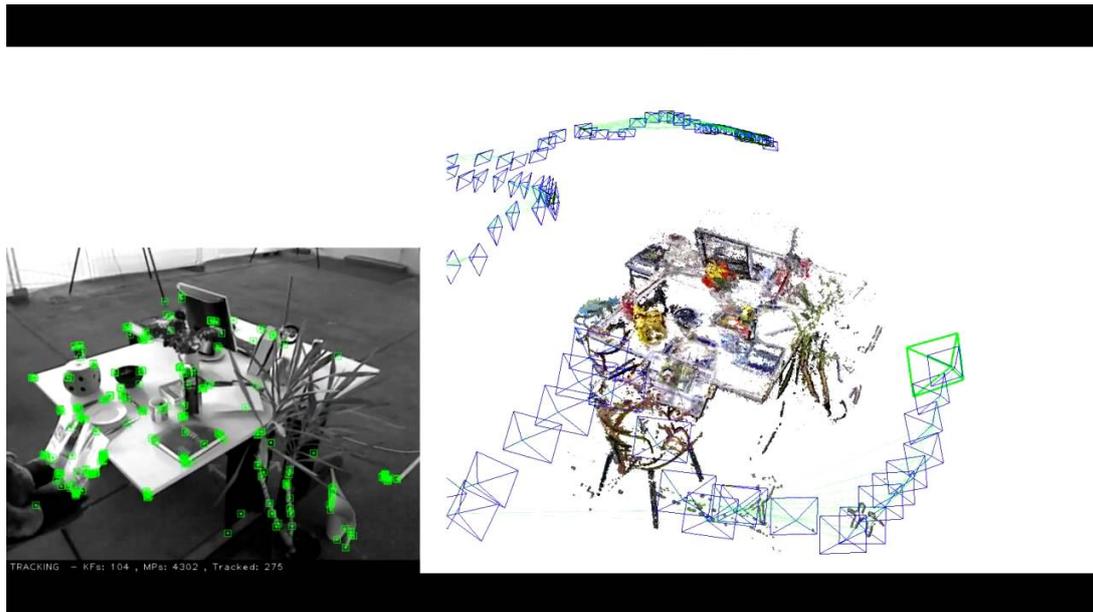


Figura 3.4: En verde en la imagen izquierda los puntos característicos obtenidos con ORB. A la derecha una imagen con la reconstrucción de esa misma escena.

De nuevo ORB-SLAM cuenta con un hilo de localización y otro para el mapeado e incluye un hilo más para la detección del cierre de bucle, ofreciendo también relocalización. Es un algoritmo que funciona en tiempo real con CPU.

Para la localización se extraen puntos característicos de la imagen con ORB, se crean descriptores para los puntos y finalmente se emparejan los descriptores de la imagen actual con la anterior. El modelo de bolsa de palabras se usa en el caso de que el robot se pierda, por ejemplo, por un “secuestro” (se denomina así a la acción de tapar la cámara durante un tiempo y llevarla o no a otra ubicación). Ante esta situación se evalúa el *Keyframe* actual con los *Keyframes* de la bolsa de palabras, y si se encuentran suficientes puntos que se emparejen entre el actual y alguno de los *Keyframes* guardados se reinicializa en esa posición conocida, utilizando el algoritmo de PnP para calcular la posición de la cámara.

La inicialización del mapa se puede realizar de dos formas, mediante un mapa por homografía y mediante el uso de la matriz fundamental. Ambos procesos se realizan en paralelo y obtienen una puntuación de cuánto de fiables son ambos, el que mayor puntuación tenga será el usado para la inicialización. Esto presenta la ventaja frente a otros algoritmos de SLAM de poder inicializarse sin necesidad de un plano principal, usando la matriz fundamental en casos donde no exista un plano principal y la homografía cuando sí se cuente con él. El hilo del

3.2. LOCALIZACIÓN CON INFORMACIÓN VISUAL Y/O DE PROFUNDIDAD (RGB-D)

mapeado se actualiza y optimiza en este caso mediante la técnica de *Bundle Adjustment*. Los puntos 3D se calculan mediante la triangulación de los pares de puntos emparejados mediante ORB o se obtienen directamente del mapa de profundidad en caso de trabajar con un par estéreo o una imagen RGB-D.

Mientras que se genera el mapa, se van guardando los fotogramas claves en la bolsa de palabras y también se genera un grafo donde los vértices son dichos fotogramas claves y las aristas o uniones entre ellos representan que ambos fotogramas contienen puntos en común. Esto permite eliminar fotogramas claves redundantes entre otras cosas.

Finalmente, el hilo del *Looping* es el hilo que se encarga de comprobar si se ha producido un cierre de bucle. Para esto se utiliza la bolsa de palabras y los *Keyframes* con el objetivo de encontrar si el fotograma actual comparte puntos con alguno de los fotogramas claves guardados. En el caso de producirse esto, se detecta un cierre de bucle eliminando puntos 3D duplicados y corrigiendo la posición de *Keyframes* anteriores. Esto ayuda a reducir el error que se ha ido acumulando durante la localización y a obtener mapas de mejor calidad.

ORB-SLAM es un algoritmo que ha tenido un gran éxito y aceptación debido a su robustez y precisión en distintos entornos. Aunque comparado con SVO es más lento, debido al cálculo de los descriptores ORB, es mucho más fiable y obtiene mapas de mejor calidad.

Para funcionar en tiempo real requiere de procesadores con cierta capacidad de cómputo, por lo que todavía no es posible su utilización en ciertos equipos o robots como drones de pequeño tamaño.

3.2. Localización con información visual y/o de profundidad (RGB-D)

Con la llegada al mercado de la cámara *Microsoft Kinect™ VI* a finales de 2010, se populariza el uso de cámaras RGB-D en el ámbito de la investigación en visión artificial debido a su bajo coste.

La localización con información RGB-D presenta una serie de ventajas frente a los métodos RGB al añadir información de la profundidad de la escena a la información de intensidad luminosa. Entre ellas, la más importante es la de no tener que hacer la triangulación puesto que la profundidad de cada píxel se puede recoger directamente de la imagen de profundidad. Otra

3.2. LOCALIZACIÓN CON INFORMACIÓN VISUAL Y/O DE PROFUNDIDAD (RGB-D)

de las ventajas más notorias es la de contar con escala absoluta en el mapa.

En 2011, Steinbrücker, Sturm y Cremers [42] proponen una alternativa usando métodos directos para el registro de dos imágenes mediante la minimización del error fotométrico, que consigue una estimación de pose robusta y permite trabajar en procesadores no muy exigentes. Esta minimización se basa en la foto-consistencia que tienen que mantener los objetos de una escena a otra, una extensión del método de Lukas-Kanade para imágenes 2D [3]. Se apoyan en el uso de una pirámide de imágenes para realizar el cálculo a distintos niveles de escala permitiendo desplazamientos entre imágenes amplios.

Kerl, Sturm y Cremers [21] mejoran dos años después el trabajo original de Steinbrücker et al. [42] al incluir un acercamiento probabilístico que hace el sistema más robusto frente al ruido y movimientos dinámicos. Para esto proponen dos funciones que ponderan el peso que se le da a cada píxel de la escena al calcular la odometría. Una función modela el sensor para disminuir el efecto del ruido y la otra corresponde a un modelo de movimiento que reduce el efecto de objetos dinámicos que se desplazan a distinta velocidad que el resto de la escena.

Henry et al. [17] ofrece un sistema de SLAM cuya estimación de pose está basada en emparejamiento de puntos característicos y ICP. Extrae puntos característicos usando FAST para emparejarlos posteriormente y realizar un SBA, *Sparse Bundle Adjustment*. Incorpora restricciones de ICP al paso de SBA para mejorar la estimación. Uno de los primeros sistemas RGB-D publicados como código abierto que tuvo una gran repercusión fue el trabajo de Endres et al. [9].

Los autores de [21] presentan un algoritmo de SLAM conocido como DVO SLAM, *Dense Visual Odometry SLAM* [20]. Al utilizar la información de todos los píxeles de la escena para estimar la posición se le incluye en el grupo de los métodos directos. Mejoran la odometría de [21], no solo utilizando información RGB para reducir el error fotométrico sino que añaden un método directo para reducir el error de profundidad. Esto les permite obtener un sistema que funciona tanto en entornos con poca textura como con poca estructura, consiguiendo un algoritmo robusto. En este caso para poder funcionar en tiempo real no se realiza una estimación de pose fotograma a fotograma sino que se realiza fotograma contra fotograma clave. Construyen un grafo con los fotogramas claves que les permite realizar cierres de bucle. El algoritmo está disponible como código abierto en la web del grupo de la universidad de Munich¹.

ElasticFusion [48] soluciona el problema de localización y reconstrucción de una forma

¹<https://vision.in.tum.de/data/software/dvo>

3.2. LOCALIZACIÓN CON INFORMACIÓN VISUAL Y/O DE PROFUNDIDAD (RGB-D)

totalmente diferente. El sistema obtiene reconstrucciones de muy alto detalle como se aprecia en la Figura 3.5.



Figura 3.5: Reconstrucción de una oficina con un gran número de *surfels* en tiempo real.

El sistema se basa en crear mapas de *surfels*, elementos de superficie, en lugar de trabajar con puntos en el espacio tridimensional. Sigue un esquema de registro denso de fotograma a modelo para la estimación de la pose y en lugar de un cierre de bucle tradicional mediante grafos utiliza un esquema de comparación del modelo local al modelo. Entre sus desventajas, solo puede trabajar en tiempo real con modelos del tamaño de una habitación. El sistema está implementado sobre GPU ya que el trabajo con superficies es propio de gráficos por computador donde se precisa de unidades de procesamiento gráfico.

El ya mencionado ORB-SLAM2 [26] se beneficia de los sensores RGB-D permitiendo utilizar la información de profundidad disponible para ser más eficiente, evitando el costoso paso de la triangulación y permitiendo reconstrucciones densas del mapa si se requiere. Es un algoritmo preparado tanto para trabajar en RGB como con RGB-D como con un par estéreo.

Por otro lado existen métodos de localización que tan solo utilizan la información de profundidad. Estos métodos se apoyan en técnicas pertenecientes al procesamiento de nubes de puntos o imagen 3D.

Una de las técnicas clásicas en el registro de nubes de puntos es ICP [1]. ICP o *iterative Closest Points* [1] es un algoritmo iterativo empleado para minimizar la distancia entre dos

3.2. LOCALIZACIÓN CON INFORMACIÓN VISUAL Y/O DE PROFUNDIDAD (RGB-D)

nubes de puntos, se puede aplicar tanto en el ámbito 2D como en el 3D. Se pretende encontrar las transformaciones que hay que aplicar sobre una nube de puntos para que esta quede registrada a la otra. Para ello se minimiza una función, normalmente la suma de los cuadrados de la distancia euclídea entre cada par de puntos, con el objetivo de que ambas nubes de puntos queden alineadas tras una serie de iteraciones. Es un problema de optimización. El problema de ICP es que es un método bastante lento y las primeras implementaciones tardaban minutos en registrar nubes de puntos. Sin embargo a día de hoy sigue ofreciendo uno de los mejores resultados en cuanto a precisión en el registro.

Una variación de ICP es *Point-to-Plane* ICP [25] que introduce en la minimización del error entre pares de puntos original, conocido como *Point-to-Point*, información de la normal a la superficie a la que pertenece el punto contra el que se pretende registrar, mejorando la convergencia del algoritmo, especialmente cuando hay estructuras o planos en las nubes de puntos siendo registradas.

GICP [38] es la variante más conocida y la que mejor rendimiento ofrece, siendo su convergencia mucho más rápida y más robusta frente a ruido. GICP también se conoce como *Plane-To-Plane* pues tiene en cuenta la información planar para realizar el registro, esto hace que el ruido no afecte tanto en el proceso. El problema de ICP y sus variantes es que son métodos iterativos, por lo que son sensibles a la inicialización y fallan a la hora de registrar nubes de puntos con grandes desplazamientos.

NDT [2] no tiene este problema. *Normal Distribution Transforms* divide el mapa o nube de puntos sobre la que se quiere registrar en cubos, a los que se le asigna una función de densidad de probabilidad gaussiana que modela la probabilidad de que haya un punto en él. Esta aproximación reduce el efecto del ruido y permite registrar nubes de puntos alejadas, su única desventaja es que es sensible al tamaño del cubo elegido para dividir la nube de puntos.

Con la llegada de la cámara *Microsoft Kinect™ VI*, también se producen avances significativos trabajando con mapas de profundidad. Una de las primeras propuestas que surgen es KinectFusion [28]. El trabajo de Newcombe et al. ofrece una reconstrucción densa en tiempo real de escenas interiores mediante una implementación GPU, Figura 3.6. La localización se consigue con un ICP multiescala usando una pirámide de imágenes. KinectFusion genera estimaciones de superficies a partir de las nubes de puntos mediante TSDF [5]. Su característica principal es que no registra fotograma a fotograma sino que registra la estimación de superficie

3.2. LOCALIZACIÓN CON INFORMACIÓN VISUAL Y/O DE PROFUNDIDAD (RGB-D)

del fotograma recibido con el modelo de superficie que se va generando. Esto proporciona muy poca deriva y mucha precisión en la reconstrucción. La desventaja de registrar contra el modelo de superficie entero es que este puede crecer de forma ilimitada, por lo que su uso en tiempo real es solo factible en escenas del tamaño de una habitación.



Figura 3.6: KinectFusion: A la izquierda la nube de puntos generada por el sensor. La imagen central y derecha corresponden a la reconstrucción del sistema a partir de la nube de puntos de entrada.

Kintinuous [49] soluciona la limitación del tamaño de escena máximo de KinectFusion permitiendo modificar de forma dinámica la región del espacio mapeado que se usa en la localización, de forma que siempre se pueda trabajar en tiempo real y se puedan generar reconstrucciones de larga escala.

Manoj et al. [32], proponen una odometría basada en puntos característicos que extraen de la imagen de profundidad. Es uno de los primeros métodos que utilizan puntos aislados de la nube de puntos, pudiendo realizar el trabajo en CPU, a diferencia de los métodos que utilizan ICP que suelen estar implementados en GPU. Por esta razón se le denomina SDO, *Sparse Depth Odometry*. Los autores proponen dos métodos para la detección de puntos característicos, SURE y NARF que utilizan simultáneamente para obtener una mayor robustez. Se obtiene el desplazamiento entre fotogramas consecutivos y su mayor desventaja reside en el ruido de las imágenes de profundidad que provocan que a veces sea complicado obtener puntos característicos robustos.

DIFODO [18] presenta un método de odometría basado en el flujo de rango [41] para imágenes de profundidad consecutivas. Se caracteriza por ser especialmente rápido incluso en procesadores de bajas prestaciones, consiguiendo incluso 90Hz usando resoluciones de imagen

3.2. LOCALIZACIÓN CON INFORMACIÓN VISUAL Y/O DE PROFUNDIDAD (RGB-D)

pequeñas.

Zhao y Fang [50] combinan DIFODO [18] e ICP [1] consiguiendo una odometría en tiempo real muy robusta que forma parte de DDS, Direct Depth SLAM. DIFODO proporciona la estimación de pose principal mientras que ICP se utiliza para reducir la deriva local en las estimaciones de DIFODO. Como otros métodos de SLAM utiliza un grafo con fotogramas clave que permite el cierre de bucle mediante la asociación de características geométricas que se extraen y emparejan utilizan un mapa NDT [2]. El resultado es un algoritmo que utiliza solo información de profundidad muy eficiente que permite trabajar en tiempo real en procesadores modestos.

La mayoría de métodos RGB-D son métodos RGB que han sido adaptados para aprovechar la información de profundidad disponible y ser más eficientes, pero no se benefician de la información de estructura que aporta la imagen de profundidad y siguen teniendo problemas cuando hay poca textura en la imagen RGB. Por otro lado la mayoría de métodos que utilizan solo información de profundidad utilizan implementaciones en GPU para poder funcionar en tiempo real y suelen tener dificultades para realizar cierres de bucles o en reconstrucciones de gran escala. Además, los métodos que usan solo información de profundidad y funcionan en tiempo real en CPU suelen presentar un mayor error en la estimación de pose en comparación con los métodos que usan información RGB. En este trabajo de fin de Máster se hace patente la necesidad de tener un algoritmo de SLAM que tenga la precisión de los métodos RGB y que sea robusto frente a la falta de textura, incorporando la información geométrica disponible en las cámaras RGB-D. Con este objetivo nace la idea de mejorar SD-SLAM, combinándolo con un algoritmo de odometría que trabaje con información de profundidad y crear un nuevo algoritmo que se denominará SD-SLAM+ que solucione los puntos descritos anteriormente.

Capítulo 4

Estudio de antecedentes

En este capítulo se describen los principales algoritmos de localización 3D visual que han sido utilizados como base durante la realización de este trabajo fin de máster. Se analiza y evalúa DIFODO, un algoritmo de odometría visual basado en cámaras de profundidad. Después se analiza el algoritmo que se busca mejorar, SD-SLAM. Finalmente se hace una comparación experimental entre ambos algoritmos y se discuten los resultados obtenidos. En esta comparativa se ha utilizado un conjunto de datos internacional y las métricas de calidad habituales en los algoritmos de autocalización visual.

4.1. Algoritmo DIFODO

DIFODO [18] es una abreviación de *DIFferential ODOmetry*, que es un método de odometría visual que permite estimar la pose de una cámara mediante información de profundidad únicamente. Ha sido desarrollado por Mariano Jaimez y Javier Gonzalez-Jimenez del departamento de Ingeniería de Sistemas y Automática de la universidad de Málaga. Para la estimación de pose utiliza todos los píxeles de la imagen, por lo que es un método de localización denso o directo. Este algoritmo ha sido diseñado para funcionar con cámaras de profundidad aunque también podría adaptarse para ser usado con otro tipo de sensor de profundidad. Es capaz de funcionar en tiempo real en procesadores no muy potentes pudiendo llegar a funcionar incluso a altas tasa de imágenes, como 60Hz, lo cual le permite trabajar con movimientos rápidos y establecer trayectorias más finas. La principal desventaja de este método es no contar con suficiente información geométrica en la escena, lo cual produce una estimación no precisa, similar

4.1. ALGORITMO DIFODO

al problema que sufren los métodos de odometría visual basados en información fotométrica con la ausencia de textura.

DIFODO estima la velocidad lineal y angular de la cámara entre dos imágenes de rango basándose en la ecuación de restricción del flujo de rango, más conocida por *range flow constraint equation* [40] [41] en inglés. La ecuación de restricción del flujo de rango es el concepto del flujo óptico [3] aplicado a imágenes de rango o profundidad. La ecuación de restricción del flujo de rango asume una escena rígida y formula la velocidad de los puntos en función del movimiento de la cámara usando el modelo *pin-hole* para establecer esta relación. A cada punto de la escena se le puede aplicar esta ecuación de restricción para obtener la velocidad de la cámara. Este es un problema indeterminado para un punto pero con un número suficiente de puntos, al menos seis, se puede resolver y permite llegar a una solución.

Una de las condiciones de la ecuación de restricción del flujo de rango es que la imagen de profundidad debe ser diferenciable, por lo que los puntos en los bordes de objetos se han de evitar pues el campo de profundidad no es diferenciable en el borde de los objetos. Por otro lado, se ha linealizado la expresión mediante una serie de Taylor, donde solo se tienen en cuenta las primeras derivadas, por lo que la ecuación de restricción solo es válida mientras haya pequeños desplazamientos de la cámara entre imágenes consecutivas.

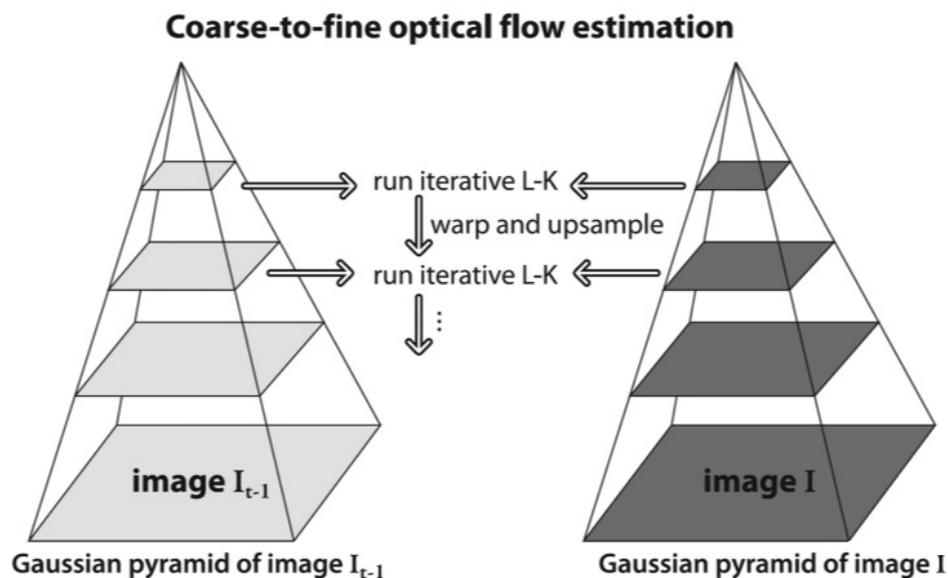


Figura 4.1: Esquema de una pirámide Gaussiana para dos imágenes [12].

La restricción de pequeños desplazamientos se soluciona con la aplicación de un esquema

multiresolución. La creación de una pirámide gaussiana Figura 4.1 permite obtener el flujo de rango a distintos niveles de imagen [4], obteniéndose desplazamientos grandes en los niveles más altos y de menor resolución de la pirámide, y a medida que se desciende por ella y se aumenta la resolución se obtienen desplazamientos más pequeños y locales, solucionando el problema derivado de la linealización. La información obtenida en los niveles más altos de la pirámide se utiliza en los siguiente niveles mediante la deformación y la interpolación.

Para resolver del sistema de ecuaciones creado en cada nivel de la pirámide por la ecuación de restricción del flujo de rango se utiliza una solución de mínimos cuadrados. Aunque tan solo seis puntos se puedan usar para solucionar el sistema, normalmente toda la información de la imagen será usada por lo que se obtendrá un sistema sobre-determinado que se resuelve optimizando por mínimos cuadrados. Se introduce también una forma de ponderar el peso de cada uno de los puntos de la escena, de forma que tengan mayor o menor influencia en la solución por mínimos cuadrados. La función de peso se usa para ajustar la contribución de cada uno de los puntos a la ecuación de restricción y se basa en la medida de error del sensor en la adquisición de imagen y en el error introducido por la linealización de la ecuación de restricción.

Finalmente, mediante el cálculo de la ecuación de restricción del flujo de rango en un esquema multinivel se obtiene una estimación de la velocidad que se utiliza para actualizar la última pose conocida. Este algoritmo cuenta con una implementación en el lenguaje C++, en la librería MRPT (Mobile Robot Programming Toolkit)¹.

4.2. Algoritmo SD-SLAM

SD-SLAM [31] es el resultado de mejorar el algoritmo ORB-SLAM [26], transformándolo en un método híbrido de localización. ORB-SLAM utiliza un método de localización basado en características. La inclusión de un método directo lo convierte en un algoritmo nuevo al que se le denomina SD-SLAM, el cual fue desarrollado por Eduardo Perdices como parte de su tesis doctoral junto a José M. Cañas dentro de la URJC. Al igual que ORB-SLAM, este puede ser usado tanto con una cámara RGB, como RGB-D como con un par de cámaras estéreo.

SD-SLAM ha sido diseñado para ser muy eficiente pudiendo funcionar en tiempo real en

¹<https://www.mrpt.org/>

4.2. ALGORITMO SD-SLAM

procesadores bastante modestos. Como la mayoría de los algoritmos de SLAM y como su predecesor ORB-SLAM, presenta dos hilos diferenciados: uno que estima el movimiento de la cámara y otro que se encarga del mapeo.

Una de las características ventajosas que incorpora SD-SLAM es una nueva forma de representar los puntos 3D, lo que proporciona una serie de mejoras. La mayoría de algoritmos de SLAM representan los puntos 3D utilizando coordenadas cartesianas (X, Y, Z) , un sistema de coordenadas al que estamos acostumbrados, siendo su principal ventaja la sencillez a la hora de trabajar. Un problema de las coordenadas cartesianas cuando se usa una única cámara es que inicialmente los puntos 3D en el espacio tienen una profundidad incierta que se refina con las nuevas observaciones siguientes. Este problema cuando se trabaja con cámaras RGB-D desaparece al poderse extraer directamente de la imagen de profundidad. Otro de los problemas es que es difícil representar con precisión puntos muy alejados de la escena, dificultando el uso de estos.

SD-SLAM introduce una nueva forma de representación para los puntos 3D, utilizando la inversa de la profundidad. Cada punto 3D se representa por la rotación y traslación desde el punto hacia la posición de la cámara que lo detectó por primera vez, el rayo de reproyección en coordenadas respecto de la cámara y la inversa de la profundidad. La ventaja en comparación con las coordenadas cartesianas es que se pueden representar puntos en el infinito, pues la inversa de la profundidad para el infinito es 0. De esta manera los puntos 3D están referenciados a los distintos fotogramas claves donde se detectaron por primera vez y no respecto a un origen arbitrario.

La principal mejora introducida por SD-SLAM se encuentra en el hilo de la localización. Este hilo conserva la extracción de puntos característicos mediante ORB [35], el cual utiliza el algoritmo FAST [34] junto a un pirámide de imágenes para encontrar características a diferentes escalas. Después se utiliza el modelo de movimiento para hacer la primera estimación de la pose de un nuevo fotograma. El modelo de movimiento se actualiza con cada nueva estimación, ofreciendo una velocidad constante en base a los últimos desplazamientos. Esta velocidad estimada se añade a la última posición conocida de la cámara, obteniéndose una primera estimación de la nueva pose. Hasta este punto el funcionamiento es similar al de ORB-SLAM, donde el siguiente paso sería el emparejamiento de puntos y la reducción del error de reproyección. Sin embargo, SD-SLAM añade un módulo previo conocido como alineación de la imagen.

Esta alineación de la imagen realiza una estimación de la pose previa al emparejamiento de puntos, utilizando un método directo para ello, similar al que utiliza SVO [14]. La ventaja principal es que esta alineación previa permite una estimación de la pose muy rápida que luego es refinada por el emparejamiento de puntos y la reducción del error de reproyección. La reducción del error de reproyección es un proceso iterativo por lo que si se proporciona una pose inicial muy cercana a la real el algoritmo converge mucho más rápido, obteniéndose como resultado final un tiempo de procesamiento menor con la combinación del método directo y el método basado en características que con el método basado en características por sí sólo.

Antes de empezar con el emparejamiento de puntos en SD-SLAM se divide la imagen en celdas de tamaño fijo. Todos los puntos 3D del anterior fotograma que son visibles en el nuevo son proyectados sobre el nuevo fotograma. Esto es posible pues la estimación de la pose en el alineamiento de la imagen ofrece una pose muy cercana a la final, lo que permite esta proyección de puntos 3D del anterior fotograma. Tan sólo se emparejará un punto por celda, lo cual reduce mucho el coste computacional. También se establece un número máximo de puntos a emparejar por iteración, lo que reduce aún más el tiempo de procesamiento. Se utiliza RANSAC [13] como mecanismo de detección de puntos atípicos, evitando añadir al mapa puntos que han sido incorrectamente emparejados o que pertenecen a partes dinámicas de la escena.

SD-SLAM cuenta con un mecanismo de relocalización que se activa cuando se detecta que la estimación de la pose presenta mucha incertidumbre. Esta relocalización no siempre funciona en tiempo real si el número de fotogramas clave en el mapa es muy alto ya que se realiza una comparación entre el fotograma actual y todos y cada uno de los fotogramas clave del mapa. La relocalización también ha sido mejorada con respecto a ORB-SLAM de la misma forma que se mejoró la localización, añadiendo el alineamiento de la imagen como paso previo al emparejamiento de puntos.

En lo que respecta al mapeado, la inicialización del mapa cuando se trabaja en el modo RGBD es muy sencilla, a diferencia del modo RGB. Al contar con la profundidad de los puntos de la escena, una única imagen que cuente con un número suficiente de puntos característicos es suficiente para inicializar el mapa. El mapa sólo se actualiza con nuevos *keyframes* y hay un número máximo que puede albergar ya que si no, este podría crecer de forma indefinida haciendo ineficiente al algoritmo. Cuando se llega al límite de fotogramas claves permitidos se procede a la eliminación de los *keyframes* más alejados de la posición actual.

4.3. EL CONJUNTO DE DATOS PARA EVALUACIÓN

Las condiciones para añadir nuevos fotogramas clave es que hayan pasado suficientes iteraciones desde la última inserción y el número de puntos 3D disponga de nuevos puntos 3D. Si no han pasado suficientes iteraciones pero el número de puntos 3D nuevos supera un umbral específico, entonces se añade un nuevo *keyframe* al mapa al presentar mucha información nueva.

Los fotogramas clave y los puntos 3D del mapa se usan para optimizar la pose, corrigiendo la deriva que se produce en la odometría mediante la técnica de Ajuste de Haces [46]. Esta optimización solo se produce de forma local con los fotogramas clave cercanos al último para que no consuma muchos recursos.

SD-SLAM ofrece grandes resultados siempre que la escena presente suficiente textura pero cuando esta carece de textura la estimación de pose se vuelve imprecisa llegando incluso a no poder estimarse. Al ser un algoritmo que funciona no solo con cámaras RGB sino también con cámaras RGB-D, es posible hacer uso de la información de profundidad para tratar de solventar esta debilidad.

4.3. El conjunto de datos para evaluación

El conjunto de datos para evaluar estos algoritmos precisa de unas características específicas, en concreto ha de provenir de un sistema de cámaras con información tanto de color o RGB como de información de profundidad o D de *depth*, profundidad. Estas son las características de la cámara utilizada como apoyo durante el desarrollo de este proyecto, la cámara *Intel® RealSense™ Depth Camera D435*².

El conjunto de datos **TUM RGB-D SLAM Dataset and Benchmark** [44] cumple las características necesarias. Este *dataset* de la universidad técnica de Munich (Technical University of Munich or TUM), contiene más de 50 secuencias de vídeo grabadas con una cámara *Microsoft Kinect™ V1*. Las secuencias están formadas por secuencias de vídeo RGBD con una resolución de 640x480 píxeles a una frecuencia de 30Hz. Las secuencias cuentan con poses consideradas como el *ground-truth*(posiciones verdaderas) que se pueden usar para comparar las estimaciones de los algoritmos usados. También cuenta con información del sensor IMU (unidad de medición inercial) incorporado de la Kinect, aunque esta no es usada en este proyecto, así como parámetros de calibración de la cámara o parámetros intrínsecos en caso de

²<https://www.intelrealsense.com/depth-camera-d435/>

ser necesarios. En el caso de SD-SLAM estos parámetros son necesarios para el proceso de estimación de pose.

El *dataset* se ha grabado en interiores, en habitaciones de oficina, dentro de la propia universidad, y presenta numerosos entornos con características distintas que lo hacen ideal para la comparación de distintos algoritmos. Una de estas características es la de contar con secuencias sin textura, secuencias sin estructura y secuencias que no cuentan con textura ni estructura. Las distintas secuencias están disponibles en dos formatos:

- Como un conjunto de imágenes que pueden ser cargadas desde disco.
- Como *rosbag*, un formato de ficheros para almacenar datos utilizado en el ámbito de ROS.

El formato usado en la realización de estos experimentos ha sido el de ROS ya que SD-SLAM está preparado para trabajar con este formato como entrada. Por otro lado, ROS permite ser independiente del formato de un sensor específico al establecer un formato genérico, lo que facilita el uso de diferentes modelos de cámaras.

A continuación se describe de forma breve las secuencias usadas en los experimentos:

4.3.1. Secuencia `rgbd_dataset_freiburg1_xyz.bag`

En esta secuencia la cámara se apunta hacia un escritorio como el de la figura Figura 4.2. Esta secuencia contiene solo movimientos de traslación a lo largo de los ejes principales de la cámara mientras que la orientación permanece fija en la medida de lo posible. Es una secuencia simple de unos 30 segundos de duración y velocidades bajas.



Figura 4.2: Escritorio perteniente a diversas secuencias del conjunto de datos de la universidad técnica de Munich

4.3.2. Secuencia `rgb_dataset_freiburg1_rpy.bag`

En esta secuencia la cámara apunta al mismo escritorio que en Subsección 4.3.1. En este caso se prueba la rotación sobre los ejes principales evitando movimientos de traslación. De nuevo una secuencia simple que busca probar el funcionamiento de algoritmos. Presenta una duración de 27 segundos.

4.3.3. Secuencia `rgb_dataset_freiburg1_360.bag`

En esta secuencia se realiza un giro de 360 grados mientras que se desplaza hacia arriba y hacia abajo la cámara. El giro se realiza sin que la persona que controla la cámara se desplace del sitio. La duración es de 28 segundos y los movimientos son rápidos.

4.3.4. Secuencia `rgb_dataset_freiburg1_floor.bag`

La cámara se traslada cercana al suelo y apuntando a este. Algunos objetos se aprecian en la escena a lo largo de la grabación y el suelo cuenta con suficiente textura. Por el lado de la estructura, esta escena presenta un único plano durante la mayoría del tiempo que corresponde

al suelo a excepción algún objeto en determinados momentos que entra en escena. Es una escena de complejidad para algoritmos basados solo en información de profundidad. La duración es de unos 45 segundos con una velocidad de desplazamiento moderada.

4.3.5. Secuencia `rgbd_dataset_freiburg1_desk.bag`

Esta secuencia consiste en una trayectoria semicircular a lo largo de dos escritorios de un laboratorio de universidad. En la Figura 4.3 se aprecian ambos escritorios.



Figura 4.3: Fotograma de la secuencia `rgbd_dataset_freiburg1_desk.bag`. En verde los puntos característicos extraídos por SD-SLAM.

La trayectoria empieza en uno de los escritorios, lo recorre hasta llegar al otro escritorio que también se recorre, y después vuelve siguiendo el mismo camino. Es por lo tanto una secuencia que permite relocalizaciones y cierres de bucles en algoritmos de SLAM, al pasar por zonas que ya ha recorrido previamente. La secuencia es de 23.4 segundos de duración, sin embargo, aproximadamente los primeros 4 segundos de esta secuencia y del resto carecen de imágenes. Esto es así para que a la hora de grabar la secuencia, el hardware tenga un tiempo de estabilización o calentamiento en el arranque, por lo que la duración real sería de 20 segundos aproximadamente. Cerca de la mitad de la secuencia se emplea en ir de un escritorio al otro y

4.3. EL CONJUNTO DE DATOS PARA EVALUACIÓN

la otra mitad de la secuencia en volver, por lo que a partir de la segunda mitad de la secuencia se pueden producir cierres de bucles.

4.3.6. Secuencia `rgbd_dataset_freiburg1_desk2.bag`

Secuencia muy similar a Subsección 4.3.5. Corresponde a una segunda grabación del mismo entorno.

4.3.7. Secuencia `rgbd_dataset_freiburg1_room.bag`

Es una secuencia de larga duración, de 49.3 segundos donde los primeros 4 segundos de nuevo no cuentan con imágenes. La secuencia es una trayectoria circular mientras se recorre una habitación con diversos objetos que aportan mucha textura y estructura. Sólo hay un momento donde el aporte de textura decae como se aprecia en la Figura 4.4. Al ser una trayectoria circular, hacia el final de la secuencia se vuelve a visitar una zona ya conocida, buscando producir un cierre de bucle.



Figura 4.4: Momento en el cual la textura decae en la secuencia `rgb_dataset_freiburg1_room.bag`. Imagen perteneciente a la GUI de SD-SLAM+

4.3.8. `rgb_dataset_freiburg3_nostructure_texture_far.bag`

La secuencia consiste, como se puede ver en la Figura 4.5, en una serie de documentos con información en ellos que aportan mucha textura en la escena. Por otro lado, la única estructura que se encuentra en la escena es el plano que forma el suelo. Un plano en estructura 3D es el equivalente de una escena completamente homogénea para la textura para los algoritmos puros de RGB, por lo que las estimaciones de DIFODO en este caso tendrán bastante error.

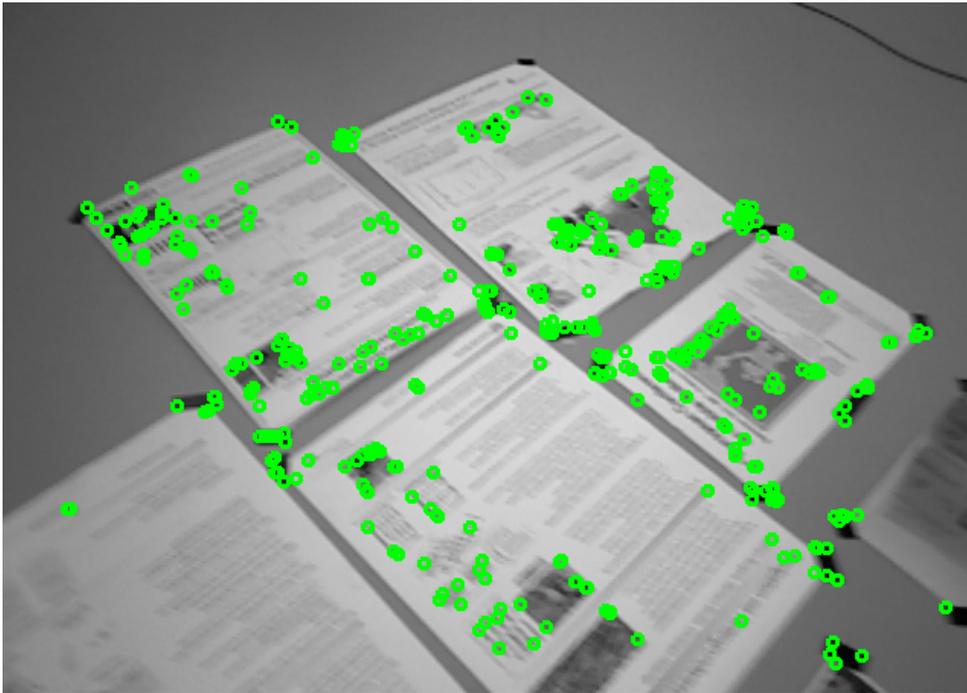


Figura 4.5: Fotograma de la secuencia `rgbd_dataset_freiburg3_nostructure_texture_far.bag` que muestra documentos y los puntos característicos extraídos por SD-SLAM en verde.

La trayectoria seguida en la secuencia es una trayectoria rectilínea, empieza en el punto A y termina en el punto B sin que haya ninguna relocalización ni posibilidad de ello. Hacia el final de la trayectoria, según se aprecia en la Figura 4.6, los documentos se pierden de la vista de la cámara quedando solo un cable y el suelo homogéneo. Esto implica que hacia el final de la secuencia haya una disminución de textura importante, que dificultará a SD-SLAM la estimación de pose. La secuencia es de 16.1 segundos de duración, pero debido a los 4 segundos usados para calentar al sistema de adquisición, solo hay imágenes en los últimos 12 segundos.

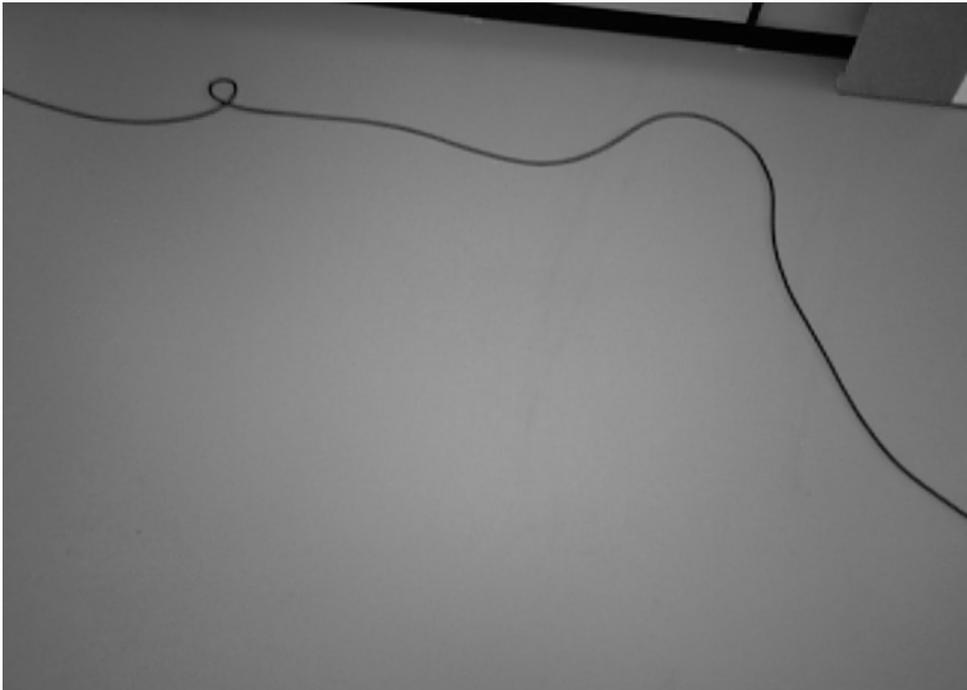


Figura 4.6: Final de la secuencia `rgbd_dataset_freiburg3_nostructure_texture_far.bag` donde se observa poca textura.

4.4. Métricas de calidad de algoritmos de autocalización visual

Entre las distintas formas de evaluar dos trayectorias para medir cual es la diferencia entre ellas hay dos que destacan y que se han popularizado entre los algoritmos de SLAM, son el ATE y el RPE [44] . El ATE o error de trayectoria absoluta está más enfocado a medir el rendimiento de algoritmos de visual SLAM. Por otro lado, el RPE o error de posición relativa está más enfocado a medir la eficacia de algoritmos de odometría visual.

El ATE primero hace un alineamiento de ambas trayectorias en caso de ser necesario, para luego evaluar la distancia geométrica entre cada una de las posiciones de ambas trayectorias. En el cálculo de la diferencia entre poses solo se usa la traslación y no la orientación y, aunque pueda parecer que se está perdiendo la información de la orientación, esta se tiene en cuenta finalmente ya que un error en la orientación se acaba traduciendo con el tiempo en un error en la posición en el espacio.

El RPE calcula diferencias en el movimiento entre dos instantes temporales o sellos temporales

al ser las estimaciones una señal no continua. El RPE puede y suele calcularse usando como unidad las estimaciones con cada nueva imagen. Se puede comparar el movimiento entre imágenes consecutivas, o cada X intervalo de imágenes.

Otro criterio a tener en cuenta es la eficiencia computacional, el tiempo de cómputo, que determina si el algoritmo podrá ejecutarse en robots y dispositivos con capacidad de cómputo limitada. Durante el desarrollo de este trabajo el hardware sobre el que se estaba desarrollando se vio sustituido por uno más actual. Esto ha permitido que se obtengan tiempos de procesamiento de DIFODO tanto para el hardware antiguo como para el nuevo. DIFODO principalmente utiliza el procesador en su implementación y, aunque la memoria RAM y la placa base también afectan al rendimiento en cierta medida, se va a utilizar el procesador como referencia en este estudio. La CPU más antigua era el modelo AMD FX-8370, una CPU del Q3 de 2014 que cuenta con 8 Núcleos, 8 Hilos a una frecuencia de 4.0GHz. La nueva CPU es un AMD Ryzen 5 2600, una arquitectura más nueva del Q2 de 2018 que cuenta con 6 Núcleos, 12 Hilos a una frecuencia de 3.4GHz. El AMD FX-8370 es una CPU que en su día estuvo considerado alta gama mientras que el Ryzen 5 2600 es una gama media. Cuatro años separan a estos procesadores, siendo el Ryzen 5 uno de los más usados actualmente en el ámbito *gaming* por su relación calidad precio. Por otro lado, también se cuenta con información de tiempos en el artículo original de DIFODO [18] así como de la CPU utilizada, un Intel Core i7-3820 del final de 2013 con 4 Núcleos, 8 Hilos y frecuencia de 3.6 GHz. Un procesador cuyo ciclo de vida coincidió con el del AMD-FX 8370 por lo que será útil para comparar estos tiempos. Finalmente, otra característica importante en SLAM es la robustez del sistema, entendiéndose por robustez la capacidad de perderse con menor o mayor frecuencia. Esta característica se reflejará en los experimentos en el error del ATE y de forma cualitativa.

4.5. Herramienta odometry_evaluation_file_creator

Una de las peculiaridades del conjunto de datos elegido es que para cada secuencia se establece un sistema de coordenadas en un punto de la escena que no tiene que coincidir con el inicio de la primera imagen de esta. Por otro lado, las secuencias del conjunto de datos empleado no presentan información visual durante los primeros cuatro segundos. Esto se realiza con el objetivo de que la transmisión de datos sea constante, permitiendo a la cámara un tiempo de

calentamiento. Todo esto produce que la estimación de pose de algoritmos como DIFODO y SD-SLAM no se pueda comparar directamente con la pose verdadera que provee el *dataset*, ya que hay un desplazamiento en translación y rotación entre el origen de coordenadas que usan las poses verdaderas y las de las estimaciones de los algoritmos DIFODO y SD-SLAM.

Por otro lado, a la hora de comparar las estimaciones de pose con la pose verdadera, las distintas herramientas de comparación precisan que los sellos temporales (instante de tiempo preciso en el que se toma cada imagen) sean los mismos entre las poses a comparar. En concreto las secuencias usadas se grabaron en los años 2011-2012, por lo que los sellos temporales datan de esa fecha y no corresponden con el de nuevas estimaciones. En conclusión, las poses verdadera y absoluta precisan de estar registradas tanto temporalmente como espacialmente.

Para poder evaluar el rendimiento de DIFODO y SD-SLAM así como el nuevo SD-SLAM+, se ha desarrollado en este TFM una herramienta³ que permite de forma automática obtener la relación de transformación entre el sistema de coordenadas del conjunto de datos y del algoritmo siendo evaluado y transformar el de este último al del conjunto de datos. Esta herramienta genera un fichero de estimaciones de pose en el mismo formato usado por el conjunto de datos **TUM RGB-D SLAM Dataset and Benchmark** permitiendo la comparación de ambas trayectorias, la estimada y la verdadera.

4.6. ROSificación de DIFODO

La *rosificación* de DIFODO hace referencia al proceso de integración del algoritmo de DIFODO en ROS (Robot Operating System)⁴.

Debido a que el conjunto de datos a usar se encuentra en un formato compatible con ROS y también a que SD-SLAM está *rosificado*, surge la necesidad de *rosificar* DIFODO con el objetivo de evaluarlo y facilitar su posterior integración con SD-SLAM.

Por estas razones y para evaluar DIFODO, se ha creado en este TFM un software que permite usar DIFODO como un nodo de ROS. La implementación es pública y se encuentra disponible en un repositorio en github⁵.

³https://github.com/RoboticsLabURJC/2019-tfm-omar-garrido/tree/master/code/odometry_evaluation_file_creator

⁴<https://www.ros.org/>

⁵https://github.com/RoboticsLabURJC/2019-tfm-omar-garrido/tree/master/code/rosify_difodo

4.7. Evaluación de DIFODO

DIFODO es la piedra angular de la mejora sobre SD-SLAM realizada en este TFM. Para evaluar si era capaz de complementar a SD-SLAM se realizaron una serie de experimentos para analizar su eficacia y tiempos de procesamiento en función de cada uno de los hiperparámetros que permite modificar el algoritmo. Tras la realización de estos experimentos se alcanzaron unas conclusiones que son las que llevaron a integrarlo con SD-SLAM para mejorarlo.

DIFODO cuenta con una serie de hiperparámetros que permiten modificar cómo se comporta el algoritmo [18]. Estos parámetros afectan principalmente a la estimación de la odometría pero también hay algunos parámetros que afectan al tiempo de procesamiento. El parámetro que más afecta a la estimación y al tiempo de procesamiento es el número de imágenes usadas en la pirámide de imágenes. Por otro lado, la resolución de entrada de las imágenes, a pesar de no ser un parámetro directo de DIFODO, afecta en gran medida a la eficacia y al tiempo de procesamiento del algoritmo por lo que también se estudiará en esta sección.

El tiempo de procesamiento es uno de los puntos más importantes de cara a cumplir con el requisito de tiempo real para el nuevo algoritmo desarrollado SD-SLAM+. Es por eso que es clave su estudio en DIFODO. Según el artículo original [18], DIFODO es capaz de funcionar a 60Hz en un procesador de un solo núcleo. Aunque no se especifica en qué resolución, más adelante se observarán los tiempos expuestos en el artículo de DIFODO así como los obtenidos experimentalmente.

4.7.1. Efecto de la resolución de las imágenes de entrada

La resolución afecta al rendimiento de DIFODO y a la estimación de la odometría. Los resultados mostrados en el artículo original [18], muestran cómo a mayor resolución, mayor es la precisión de la estimación así como el tiempo de procesamiento. Se han realizado una serie de experimentos sobre el mismo conjunto de datos pero usando la versión de DIFODO integrada con ROS que es como finalmente se realizará la integración.

La tabla 4.1 muestra los tiempos de procesamiento de DIFODO a distintas resoluciones para los distintos procesadores. En ella se observa cómo se ha conseguido una reducción en el tiempo de procesamiento de casi un factor de 3 para la nueva CPU, el Ryzen 5 2600. Para las resoluciones menores a 320x240 el tiempo de procesamiento es menor que 33.3ms, tanto

para la nueva CPU como la antigua, por lo que DIFODO a esas resoluciones funciona a tiempo real, cumpliéndose el requisito de procesamiento en tiempo real a 30 imágenes por segundo. Esto es de especial importancia pues uno de los requisitos que se persiguen de este trabajo es conseguir un algoritmo que funcione en tiempo real. La resolución VGA presenta tiempos superiores al límite de los 33.3ms por lo que no se podrían considerar en casos donde la CPU no sea especialmente potente.

| Resolución | Ryzen 5 2600 | AMD-FX 8370 | Intel Core i7-3820 |
|------------|--------------|-------------|--------------------|
| 40x30 | - | - | 3.85ms |
| 80x60 | - | - | 5.18ms |
| 160x120 | 3.5ms | 9ms | 10.04ms |
| 320x240 | 12.75ms | 30ms | 28.61ms |
| 640x480 | 55.5ms | 130ms | - |

Tabla 4.1: Tiempos de procesamiento para distintas resoluciones y distintos procesadores

En cuanto a la relación tiempo/resolución, se observa una clara tendencia donde cada vez que se aumenta en dos veces la resolución, cuatro veces si nos basamos en el número de píxeles totales, el tiempo de procesamiento aumenta de forma exponencial. Resoluciones similares o mayores a VGA se tienen que descartar si no se cuenta con un procesador de gran potencia o si se pretende trabajar a tiempo real o 30 imágenes por segundo. Esto es crucial en los algoritmos de visual SLAM ya que trabajar a menor tasa de imágenes por segundo dificulta la tarea de la estimación de la pose al presentar mayor desplazamiento entre imágenes.

En cuanto a los tiempos obtenidos, se puede observar cómo los tiempos expuestos en el artículo original son verosímiles puesto que los procesadores antiguos, AMD-FX 8370 e Intel Core i7-3820 son de un rendimiento similar y se obtienen tiempos parecidos. No se han realizado experimentos de tiempos a resoluciones menores como sí que se hizo en el artículo original, ya que no tiene sentido ir mucho más abajo en resolución pues a 160x120 hay mucho espacio hasta llegar a los 33.3ms. Lo que sí se ha hecho es comprobar que a resoluciones algo más grandes como VGA, el tiempo real deja de ser posible.

Una vez se han observado los tiempos de procesamiento, se puede concluir que DIFODO es un algoritmo que puede integrarse junto a SD-SLAM al menos en cuanto al requisito de

4.7. EVALUACIÓN DE DIFODO

procesamiento en tiempo real. El siguiente paso es observar cómo afecta la resolución a la estimación de la pose.

Lo esperado en base a los resultados reportados en el artículo original [18] es que el algoritmo a mayor resolución tenga un menor error en la estimación de la pose. La tabla 4.2 muestra el error de trayectoria absoluto para distintas resoluciones y para distintas secuencias del conjunto de datos. En rojo se muestra la resolución que arroja el mayor error y en verde la que consigue el menor. La resolución que parece obtener mejores resultados de forma general es la resolución QVGA (320x240). Mientras que los peores resultados se obtienen en la mayor resolución 640x480. Estos resultados son a priori contrarios a lo esperado y a las conclusiones del artículo original, donde se espera que a mayor resolución, menor sea el error. Sin embargo, esto tiene una explicación, que tiene que ver con el diseño de la rosificación de DIFODO que se ha realizado para evaluar el algoritmo. Este software está diseñado para trabajar a la mayor frecuencia de imágenes posible o bien se puede fijar una frecuencia objetivo. En este caso, la frecuencia de las secuencias de vídeo del conjunto de datos es de 30 imágenes por segundo, lo que implica que la máxima frecuencia está limitada por la secuencia. Como se observó en la tabla 4.1, las resoluciones 160x120 y 320x240 pueden trabajar a 30 imágenes por segundo sin problema, incluso a frecuencias mayores, pero la resolución 640x480 apenas llega a las 20 imágenes por segundo. Esto produce que una de cada tres imágenes se pierdan al no poder seguir DIFODO a la resolución de 640x480 la tasa de imágenes por segundo de la secuencia, provocando una pérdida de información que se traduce en un desplazamiento mayor entre las imágenes sobre la que estimar la odometría que afecta de forma negativa a la estimación del algoritmo.

| Secuencia | 160x120 | 320x240 | 640x480 |
|-----------|---------|---------|---------|
| desk2 | 0.2620 | 0.2094 | 0.2026 |
| floor | 0.8663 | 0.8844 | 0.8814 |
| 360 | 0.4496 | 0.4339 | 0.5692 |
| rpy | 0.0793 | 0.1175 | 0.1318 |
| xyz | 0.0805 | 0.0613 | 0.0756 |
| desk | 0.1476 | 0.1441 | 0.1573 |
| room | 0.4131 | 0.3705 | 0.4839 |
| Total | 2.2984 | 2.2211 | 2.5018 |

Tabla 4.2: RMSE en metros para el ATE (Absolute Trajectory Error) en distintas resoluciones de DIFODO.

4.7.2. Efecto del nivel de la pirámide

DIFODO utiliza una pirámide de imágenes donde la imagen de mayor resolución es la imagen original y las imágenes sucesivas se submuestran, obteniendo lo que es conocido como pirámide de imágenes en visión artificial. Este mecanismo se utiliza para aplicar algoritmos de procesamiento a distintos niveles de resolución para después juntar la información obtenida en distintos niveles, obteniendo resultados mucho mejores que en la evaluación a una única resolución. Esta técnica se usó en el cálculo del flujo óptico [3]. DIFODO también la usa pero para el cálculo del flujo de rango [41], técnica derivada del flujo óptico para ser aplicada en imágenes de profundidad en lugar de RGB. Esta pirámide le permite observar tanto movimientos de pequeños objetos como de objetos más grandes, lo que le permite obtener una mejor estimación que usando solo la imagen original. Esto afecta directamente a la precisión de la estimación obtenida y también al tiempo de procesamiento, pues a menor número de imágenes en la pirámide menos son las veces que se tendrá que calcular el *range flow*.

De nuevo se analiza tanto a nivel de tiempos como a nivel de error en la estimación el efecto de este parámetro.

4.7. EVALUACIÓN DE DIFODO

| Resolución | Niveles de Pirámide | Tiempo (ms) |
|------------|---------------------|-------------|
| 160x120 | 1 | 4 |
| 160x120 | 3 | 8.5 |
| 160x120 | 5 | 9 |
| 320x240 | 1 | 15 |
| 320x240 | 3 | 27 |
| 320x240 | 5 | 30 |
| 640x480 | 1 | 80 |
| 640x480 | 3 | 130 |
| 640x480 | 5 | 130 |

Tabla 4.3: Tiempos de procesamiento por fotograma para distintos niveles de pirámide en AMD-FX 8370

En la tabla 4.3 se observa cómo a mayor número de niveles usados en la pirámide mayor es el tiempo de procesamiento. Como se puede apreciar, la diferencia entre calcular 3 o 5 niveles es casi despreciable. La razón es que en cada nuevo nivel se está reduciendo la imagen original a la mitad del tamaño original. Se consigue pues un efecto similar al de elevar al cuadrado de forma continua. Por ejemplo, para la resolución de 640x480 los niveles del resto de la pirámide serían 320x240, 160x120, 80x40 y 40x20 respectivamente. La resolución de los últimos dos niveles de la pirámide es muy pequeña en comparación con el primer nivel por lo que la diferencia entre usar 3 y 5 niveles es computacionalmente despreciable. Sin embargo, usar únicamente la imagen de entrada supone un ahorro computacional cercano al 50%. Finalmente, en la tabla 4.4 se aprecia cómo la disminución de tiempos con la nueva CPU mantiene la proporción de 3 veces inferior, independientemente del número de niveles de la pirámide.

| Resolución | Niveles de Pirámide | AMD-FX 8370 | AMD Ryzen 5 2600 |
|------------|---------------------|-------------|------------------|
| 320x240 | 1 | 15ms | 5.85ms |
| 320x240 | 3 | 27ms | 12.29ms |
| 320x240 | 5 | 30ms | 12.75ms |

Tabla 4.4: Comparación de tiempos de procesamiento para distintos niveles de pirámides y procesadores.

Se ha visto cómo la diferencia entre usar 3 o 5 niveles de pirámide es computacionalmente despreciable, mientras que se ahorra un 50 % de tiempo si solo usamos un nivel. La siguiente duda a resolver es comprobar cómo afecta esto a la precisión en la estimación de la pose.

Para ello se han realizado unos experimentos cuyos resultados se pueden observar en la Tabla 4.5 donde se puede apreciar claramente que a mayor número de imágenes en la pirámide menor es el error cometido en la estimación de la pose. La diferencia es muy notable cuando se usa un solo nivel de pirámide siendo el error total en torno a 2.5 veces mayor que cuando se usan 5 niveles, como era de esperar, pues una única resolución solo es capaz de encontrar movimientos muy locales y no globales en la imagen de profundidad.

Solo hay una secuencia donde los resultados son muy parejos en los 3 casos, en la secuencia titulada *floor*. Esta secuencia además presenta un error muy alto comparado con el resto de secuencias. Esto es así puesto que como la secuencia indica es una secuencia donde la cámara mira al suelo. Esto produce que haya una ausencia de estructura en la imagen, lo que dificulta a DIFODO la estimación. Es el caso equivalente a una escena que es homogénea en los niveles de intensidad de sus píxeles para el cálculo del flujo óptico. Independientemente del nivel de pirámide no se puede hacer una estimación precisa y los nuevos niveles de pirámide no aportan información al no haber estructura ya sea de objetos pequeños o grandes.

| Secuencia | 1 | 3 | 5 |
|-----------|--------|--------|--------|
| desk2 | 1.1880 | 0.2955 | 0.2094 |
| floor | 0.8877 | 0.9473 | 0.8844 |
| 360 | 1.204 | 0.6992 | 0.4339 |
| rpy | 0.3438 | 0.1496 | 0.1175 |
| xyz | 0.1165 | 0.0645 | 0.0613 |
| desk | 0.7288 | 0.2997 | 0.1441 |
| room | 1.2819 | 0.4188 | 0.3705 |
| Total | 5.7507 | 2.8746 | 2.2211 |

Tabla 4.5: RMSE en metros para el ATE (Absolute Trajectory Error) en distintos niveles de pirámide de DIFODO.

4.7.3. Conclusiones

Se ha comprobado cómo DIFODO es un algoritmo capaz de ejecutarse a 30 imágenes por segundo en procesadores no muy exigentes, e incluso a mayor frecuencia en otros procesadores o a resoluciones inferiores sin comprometer la estimación de la pose. Esto es así cuando se trabaja a una resolución menor o igual a 320x240. Cumple por lo tanto con la necesidad de trabajar a tiempo real.

Queda demostrado que el número de niveles de la pirámide afecta a la estimación de la pose y al tiempo de procesamiento, y se llega a la conclusión de que el número óptimo de niveles de pirámide es 5, ya que a 3 niveles de pirámide el tiempo que se reduce es despreciable y se produce una pérdida de precisión en la estimación de la pose.

Por otro lado, se observa cómo las estimaciones de DIFODO son bastante buenas siempre y cuando haya suficiente estructura en la escena. Una comparación más exhaustiva del error cometido se verá en la Sección 4.8, donde se compara con SD-SLAM.

4.8. Comparativa entre DIFODO y SD-SLAM

En esta sección se hace una evaluación cuantitativa del algoritmo DIFODO y de SD-SLAM. Hay que tener en cuenta que DIFODO es un algoritmo únicamente de odometría visual, por lo que se obtiene una estimación con cada nueva imagen en forma de desplazamiento con respecto a la imagen anterior. Sin embargo SD-SLAM es un algoritmo de SLAM completo, no solo hace odometría o *tracking*, sino que hace mapeado también. Sin embargo, lo más importante de cara a esta comparación es que el mapeado le permite hacer un seguimiento de aquellos lugares que se han visto previamente y usarlos para optimizar de forma global las siguientes estimaciones así como las anteriores mediante el mecanismo denominado cierre de bucle. Esto le brinda una mayor robustez en la estimación de pose frente a DIFODO donde solo se utiliza la información del fotograma actual y el anterior en las predicciones. Es de esperar que SD-SLAM tenga unas mejores prestaciones que DIFODO.

4.8.1. Precisión de las estimaciones

Para hacer esta comparación entre SD-SLAM y DIFODO se va a usar el software de SD-SLAM y la versión *rosificada* de DIFODO, junto con la herramienta descrita en la sección 4.5, `odometry_evaluation_file_creator`. El dataset para el estudio de nuevo es **TUM RGB-D SLAM Dataset and Benchmark** [44] detallado anteriormente en la sección 4.3. En cuanto a DIFODO, los parámetros usados durante la realización de estos experimentos son 320x240 de resolución y una pirámide de cinco niveles, pues como se ha demostrado en la sección anterior, que es la mayor resolución que permite tiempo real de procesamiento y la que menor error presenta en la estimación.

| Secuencia | DIFODO RMSE | SD-SLAM RMSE | Mejora relativa del error (%) |
|--------------------|---------------|---------------|-------------------------------|
| desk2 | 0.2094 | 0.1716 | 18.01 |
| floor | 0.8844 | 0.3982 | 54.96 |
| 360 | 0.4339 | 0.3864 | 10.92 |
| rpy | 0.1175 | 0.0281 | 76.02 |
| xyz | 0.0613 | 0.0626 | -2.27 |
| desk | 0.1441 | 0.1235 | 14.26 |
| room | 0.3705 | 0.2909 | 21.46 |
| Valor medio | 0.3173 | 0.2088 | 34.18 |

Tabla 4.6: Comparación del ATE (Absolute Trajectory Error) en metros para DIFODO y SD-SLAM en secuencias de **TUM RGB-D SLAM Dataset and Benchmark** pertenecientes a la subsección **Freiburg 1** del conjunto de datos.

En la tabla 4.6 se compara el error cometido por cada algoritmo en las secuencias de tipo *Freiburg 1*. Como era de esperar el error es menor en casi todos los casos para SD-SLAM en comparación con DIFODO ya que SD-SLAM es un algoritmo completo de SLAM frente a DIFODO que no cuenta con mecanismos para reducir la deriva. SD-SLAM reduce el error en la estimación de la pose en un 34 % de media en comparación con DIFODO para estas secuencias. Sin embargo hay una serie de datos a tener en cuenta. En muchas secuencias como *desk2*, *360*, *xyz*, *desk* la diferencia en la estimación es mucho menor que ese 34 %, en torno a un 15-20 %, siendo muy parejos en ambos casos incluso llegando a ser superior DIFODO en la secuencia de

4.8. COMPARATIVA ENTRE DIFODO Y SD-SLAM

xyz.

De entre todas estas secuencias hay dos valores atípicos claros, que se aprecian para las secuencias *floor* y *rpy*. En la secuencia *floor* la estimación de DIFODO contiene mucho error al no contar con apenas estructura, como ya se discutió anteriormente en la Sección 4.7. Se puede apreciar la mala estimación de pose 3D de DIFODO en la Figura 4.7.

En el caso de la secuencia *rpy* la diferencia apreciada no radica en el hecho de que DIFODO tenga una mala estimación sino que SD-SLAM obtiene una muy buena estimación en este caso, de hecho, es la secuencia donde SD-SLAM presenta un menor error de todas las presentadas al igual que DIFODO.

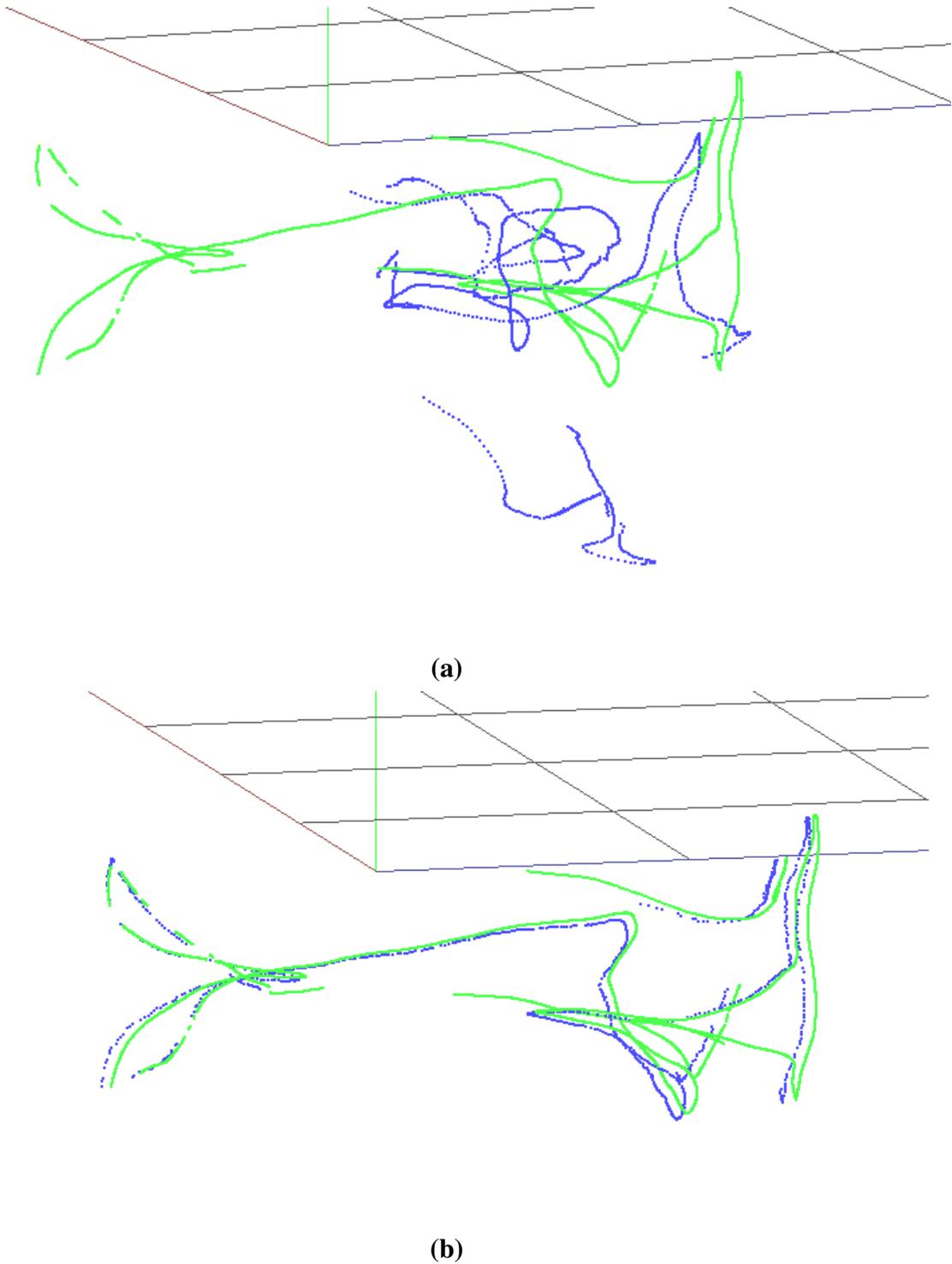


Figura 4.7: Trayectorias 3D de los algoritmos para la secuencia *floor*. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de DIFODO y la figura (b) a la estimación de SD-SLAM.

4.8. COMPARATIVA ENTRE DIFODO Y SD-SLAM

| Tipo | Secuencia | DIFODO | SD-SLAM |
|------------------------|-------------------------------------|--------|---------|
| Sin estructura | nostructure_texture_near | 2.2992 | 0.0665 |
| Sin textura | structure_notexture_near | 0.2903 | 2.005 |
| Sin textura | structure_notexture_far | 0.2708 | 2.6243 |
| Sin estructura/textura | nostructure_notexture_near_withloop | 2.3351 | 2.2091 |
| Sin estructura/textura | nostructure_notexture_far | 1.9627 | 1.7486 |

Tabla 4.7: Comparación del ATE en metros para DIFODO y SD-SLAM en secuencias de Freiburg 3: Estructura vs Textura. En verde el menor error en cada secuencia.

En la Tabla 4.7 se encuentran los resultados del error de la estimación para secuencias de la categoría *Freiburg 3*. Estas secuencias se caracterizan por ser más complejas y presentar diversidad de entornos. En la tabla Tabla 4.7 se han dividido las secuencias por tipos que corresponden a:

- **Sin estructura 3D:** Secuencias donde no hay información de estructura pero sí de textura visual. Por ejemplo una pared lisa pero con dibujos o un folio sobre una superficie plana con texto.
- **Sin textura:** Secuencias donde no hay información de textura pero sí de estructura 3D. Por ejemplo, distintos objetos pero todos monocromos, uniformes visualmente. Un ejemplo serían varias paredes lisas blancas colocadas paralelas entre sí.
- **Sin estructura/textura:** Secuencias donde no hay información ni de estructura ni de textura. Por ejemplo una pared lisa.

Del tipo sin estructura se ha evaluado la secuencia titulada *nostructure_texture_near*. Al no haber estructura el error de DIFODO es muy grande, de más de 2m, mientras que para SD-SLAM se consigue 0.06m de error, apenas unos centímetros.

Del tipo sin textura se han evaluado dos secuencias: *structure_notexture_near* y *structure_notexture_far*. Al no haber textura SD-SLAM obtiene un error de varios metros, ya que en esas secuencias SD-SLAM no consigue ni llegar a inicializar, por lo que no se consigue ni siquiera una estimación de trayectoria. Por otro lado DIFODO permanece en su línea de error, al contar con suficiente información de estructura su estimación es similar a la encontrada en las secuencias anteriores,

Freiburg 1.

Finalmente hay un tipo de secuencia donde no se cuenta con textura y tampoco con estructura. Como es de esperar ambos algoritmos obtienen un error muy alto en torno a los 2m.

En todos los casos el comportamiento y los resultados son los esperados, pues ambos algoritmos dependen de la información visual ya sea en textura o en profundidad.

4.8.2. Conclusiones

A la luz de los experimentos realizados DIFODO ha demostrado ser un algoritmo de odometría visual con un buen rendimiento en entornos donde SD-SLAM es incapaz de estimar pose. SD-SLAM es de media un 20 % aproximadamente mejor que DIFODO en la estimación de la pose. DIFODO solo presenta mejor estimación de pose que SD-SLAM cuando hay poca textura visual. Sin embargo, cuando no hay estructura DIFODO no es capaz de ofrecer una buena estimación. Ambos algoritmos precisan de información visual, en un caso textura y en el otro estructura 3D. El uso de uno cuando el otro falla tiene sentido pues se puede utilizar como mecanismo de apoyo para no perder el *tracking*. Tiene sentido por los resultados vistos usar visual SD-SLAM siempre que haya suficiente información de textura, pues de media la estimación obtenida es mejor y se pueden usar otros mecanismos propios de un algoritmo de SLAM como el cierre de bucle. En aquellos casos donde no haya suficiente textura en la imagen y donde SD-SLAM normalmente se perdería o estimaría una mala posición, se usará DIFODO para suplir esta falta de información.

Se observó también que cuando no hay información ni de textura ni de estructura, el error en ambos algoritmos es muy alto, pero estamos hablando del caso más difícil que pueden afrontar los algoritmos de visual SLAM. La solución en este caso sería utilizar información de otra fuente que no sea visual, como por ejemplo inercial.

Capítulo 5

Autolocalización visual con el algoritmo SD-SLAM+

En este capítulo se describen los detalles de la integración del algoritmo DIFODO junto a SD-SLAM para crear un algoritmo nuevo, denominado SD-SLAM+, cuyas características mejoran las de su predecesor. Finalmente se hace una evaluación experimental del rendimiento y de la eficacia de SD-SLAM+ comparándolo con SD-SLAM.

5.1. Introducción

Una vez se ha comprobado que DIFODO es un algoritmo capaz de funcionar en tiempo real en la sección 4.7 y de comprobar que es capaz de funcionar mejor que SD-SLAM en aquellas secuencias o momentos donde no hay suficiente textura en la sección 4.8, el diseño y desarrollo del nuevo algoritmo SD-SLAM+ comienza.

Los objetivos que se buscan lograr con este nuevo algoritmo combinado es conseguir que SD-SLAM no pierda la estimación de pose cuando no haya suficiente textura en la escena. Para ello se va a integrar DIFODO en SD-SLAM, con el objetivo de que cuando SD-SLAM no sea capaz de estimar una nueva pose, se empiece a usar la odometría de DIFODO hasta que vuelva a haber suficiente textura en la imagen y entonces se pueda recuperar la odometría visual original de SD-SLAM.

5.2. Diseño e implementación

El funcionamiento del algoritmo original SD-SLAM se puede resumir con una máquina de estados como muestra la figura 5.1. El algoritmo original precisa de una inicialización, que para el caso RGBD se realiza con la primera imagen que cuente con al menos 500 puntos característicos o *keypoints*. Una vez el sistema se ha inicializado pasa al estado **OK** donde se utiliza el *tracking* de ORB [35] o estimación de la pose con ORB. En este estado la estimación de la pose se realiza mediante el alineamiento de la imagen y el emparejamiento de puntos característicos y sus descriptores asociados obtenidos mediante el algoritmo ORB. Para que la estimación de pose sea considerada como correcta, existe un límite inferior en cuanto al mínimo número de puntos característicos que han de ser emparejados. Actualmente para SD-SLAM este límite está en 20 pares de puntos emparejados.

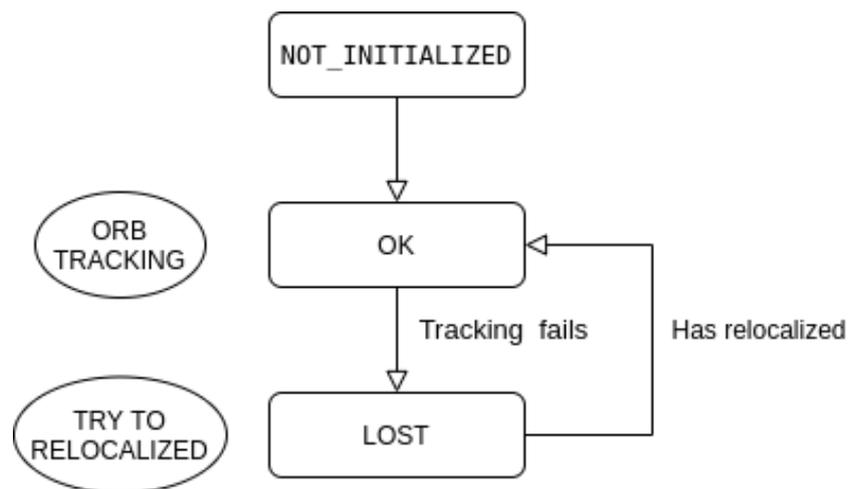


Figura 5.1: Máquina de estados de SD-SLAM

Cuando el número de pares de puntos emparejados no es suficiente, el *tracking* falla y el sistema pasa a un estado **LOST**. En este estado el sistema intentará relocalizarse con la llegada de cada nueva imagen. Este proceso de relocalización es similar al de estimación de pose. Se buscan emparejar puntos característicos extraídos de la nueva imagen con cada uno de los puntos característicos extraídos en cada uno de los *keyframes* o imágenes clave, que son imágenes que se han guardado durante el trayecto ya realizado para poder usarlas en la relocalización, en caso de que el sistema se pierda. De nuevo, al menos 20 pares de puntos han de ser emparejados para que la relocalización se considere válida y se vuelva al estado **OK**.

Una vez visto de forma resumida el funcionamiento lógico del SD-SLAM original, se pasa a ver las diferencias con el algoritmo desarrollado en este TFM denominado SD-SLAM+. La Figura 5.2 corresponde a la máquina de estados que resume el funcionamiento lógico de SD-SLAM+. En ella se aprecian una serie de estados y las condiciones para pasar de uno a otro.

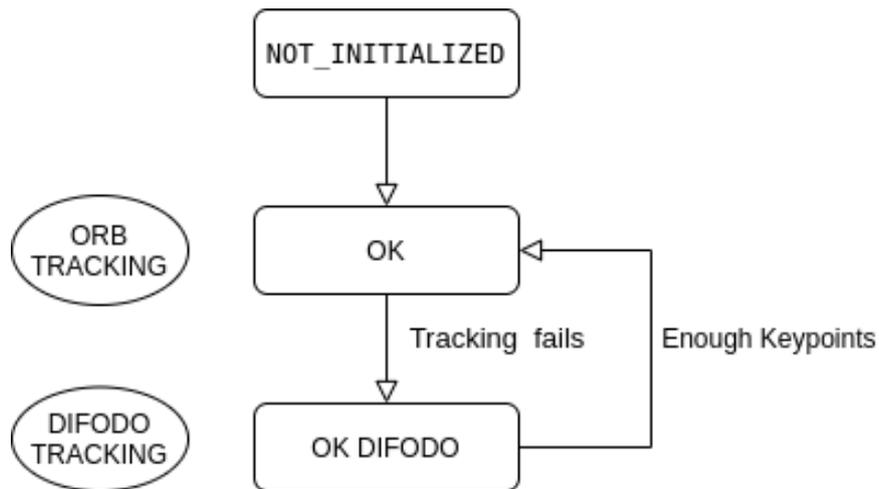


Figura 5.2: Máquina de estados de SD-SLAM+

El estado **NOT_INITIALIZED** es el primer estado al que entra el algoritmo cuando este arranca. Este estado no ha sido modificado de ninguna manera por lo que el paso del estado **NOT_INITIALIZED** al estado **OK** es exactamente el mismo que sigue SD-SLAM.

En el estado **OK** se sigue utilizando la odometría original de SD-SLAM. La única diferencia con el estado del algoritmo original es que ahora cuando la odometría falla al no haber suficiente textura, en lugar de pasar a un estado donde no hay odometría ninguna se pasa a un estado donde se procede a usar la odometría de DIFODO.

En el estado **OK DIFODO** se utiliza la odometría de DIFODO mientras no haya textura disponible en las imágenes que llegan al sistema. Este estado sustituye al antiguo estado **LOST** de SD-SLAM. Para que se vuelva a la odometría original que usa ORB, es necesario que se recupere la textura en las imágenes recibidas.

5.2.1. Paso al estado OK DIFODO

El primer objetivo es establecer cuando se utilizará la odometría de DIFODO, es decir, cuando se considera que la estimación de SD-SLAM no es suficientemente buena o ha fallado. SD-SLAM es incapaz de estimar nuevas posiciones cuando no hay suficiente textura en la imagen.

La primera idea fue utilizar el número de puntos característicos extraídos por imagen y establecer un umbral, sin embargo, el número de puntos característicos no es una métrica totalmente válida, pues el hecho de que haya pocos puntos característicos no implica necesariamente que la imagen no cuente con suficiente textura para estimar la pose con precisión. Por ejemplo una imagen de unos documentos de color blanco con una serie de caracteres escritos en ellos. En esta escena el número de puntos característicos no es muy alto, sin embargo al tener tanto contraste son puntos característicos de gran calidad, con mucha información y podrían emparejarse fácilmente de un fotograma al siguiente, lo cual produciría una estimación correcta de la pose. Otro contraejemplo sería comparar dos escenas completamente distintas con muchos puntos característicos pero que a la hora de emparejarlos, no se emparejan bien puntos característicos y de emparejarse algunos se debe a un error. En conclusión, el número de puntos característicos sobre una escena no es una métrica lo suficientemente buena como para establecerla como umbral.

El siguiente paso fue evaluar usar el mismo criterio que ya estaba siendo usado por SD-SLAM, el número de puntos emparejados entre dos imágenes consecutivas. Ya que SD-SLAM usa este criterio para considerar si la estimación de la pose es correcta o no, tiene sentido que también sea el criterio utilizado para pasar a usar DIFODO. Es por eso que este es el criterio elegido para pasar a usar DIFODO en SD-SLAM+. Quedando la nueva máquina de estados de la siguiente manera, figura 5.2.

En SD-SLAM+ no existe el estado de **LOST**, ya que el algoritmo ya no se pierde, siempre estima pose, ya sea con la estimación original de SD-SLAM o con la odometría de DIFODO. En lugar de este estado, cuenta ahora con un estado denominado **OK DIFODO**, donde la estimación de las nuevas poses se realiza con DIFODO.

5.2.2. Recuperación de la odometría original

El siguiente paso es definir cómo volver de este estado donde se usa DIFODO al *tracking* original de SD-SLAM una vez que se ha recuperado la información de textura. Como se pudo observar en la sección 4.8, la estimación de pose usando el emparejamiento de puntos extraídos con ORB comete menos error de media que la estimación de DIFODO, por lo que siempre que se disponga de suficiente textura en la imagen, se dará preferencia al *tracking* original de SD-SLAM frente al de DIFODO.

El primer impulso para comprobar si hay suficiente textura de nuevo, es usar exactamente la misma lógica de emparejamiento de pares de puntos mediante descriptores ORB y cuando el umbral establecido sea superado, volver al *tracking* de ORB. Aunque esta opción parece la más lógica en un primer momento, esto implicaría intentar el emparejamiento entre imágenes consecutivas en todo momento en el estado de **OK DIFODO**. Esto computacionalmente sería muy costoso pues no solo se estaría estimando la pose con DIFODO sino también se estaría intentando hacer lo mismo con ORB, imposibilitando el funcionamiento en tiempo real, el cual es uno de los objetivos de este trabajo fin de máster.

Descartada esta opción, se evalúa la posibilidad de utilizar el número de puntos característicos en la imagen, que como se vio anteriormente, no era la mejor opción a la hora de definir si se debía pasar a usar DIFODO. Este caso es el inverso del anterior estudiado, pues se pasa de DIFODO al *tracking* de ORB. De hecho, se observa cierta similitud con el paso de **NOT INITIALIZED** a **OK**. En el paso del estado de no inicialización del sistema al estado de *tracking*, lo único que se comprueba es que haya un número suficiente de puntos característicos en la escena, en concreto 500 puntos, un valor obtenido de forma experimental y que ha demostrado ser suficiente como criterio de inicialización. Se opta pues, por usar una lógica similar para volver a la estimación de la pose original de SD-SLAM desde el estado de **OK-DIFODO**. Se crea un mecanismo de reinicialización que es muy similar a la función de inicialización original, con la excepción de que el mapa no se vuelve a establecer sino que se usa toda la información acumulada anteriormente. Mediante los distintos experimentos de la sección 5.3 quedará demostrado que éste es un criterio suficiente para el correcto funcionamiento del nuevo algoritmo, al cambiar de estado solo cuando hay suficiente textura. La ventaja de este método frente al de emparejamiento de pares de puntos es que este método de reinicialización es muy rápido, pues la extracción de estos puntos característicos en la imagen es muy rápida, mientras que

el emparejamiento de puntos característicos es un proceso computacionalmente más costoso, permitiendo el funcionamiento en tiempo real.

El nuevo algoritmo SD-SLAM+ es capaz de pasar de un modo de estimación de la pose a otro eficientemente y por ello el funcionamiento en tiempo real se mantiene.

5.2.3. Relación entre sistemas de referencia

Los criterios para pasar de un estado a otro no son el único reto de esta integración, también se necesita estudiar a más bajo nivel cómo se realiza esta unión. Uno de los problemas encontrados es que cada sistema de odometría, SD-SLAM y DIFODO, tienen su propio sistema de referencia, por lo que añadir la estimación de una nueva pose de uno al otro no es una operación directa sino que precisa de una transformación previa. El objetivo es que el sistema de coordenadas de SD-SLAM sea el principal y que el sistema de referencia de DIFODO se use única y exclusivamente para obtener la nueva el incremento con pose que después habrá de ser llevada al sistema de referencia principal.

Por otro lado, DIFODO es un algoritmo de odometría del cual se obtiene como salida la nueva posición. Sin embargo, en este caso es preferible contar con el desplazamiento entre la anterior pose y la nueva pose, para añadir este desplazamiento a la última pose conocida de SD-SLAM antes de que se perdiera, de una forma sencilla. La Figura 5.3 representa el sistema de referencia de coordenadas de DIFODO, siendo el mundo, *World*, el origen del sistema de referencia de coordenadas, P_t la posición de la cámara en un instante de tiempo t y P_{t+1} la posición de la cámara en un instante de tiempo $t+1$. Este ejemplo representa las poses de dos imágenes consecutivas siendo P_d el desplazamiento entre ambas poses. Este desplazamiento es el que se quiere obtener para luego añadirlo a la última pose conocida por SD-SLAM.

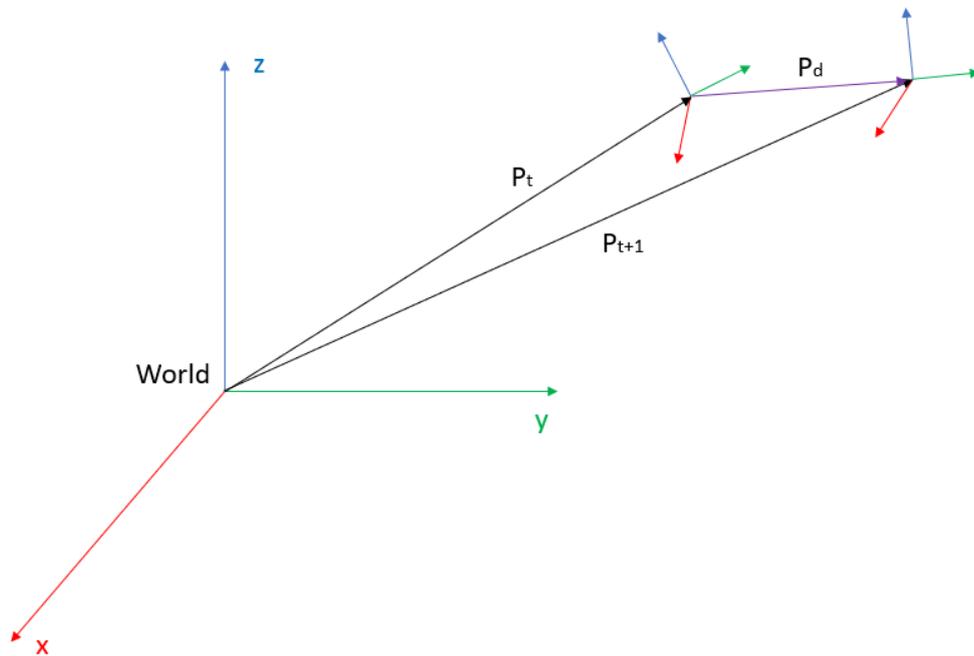


Figura 5.3: Cálculo del desplazamiento entre dos poses

Para obtener P_d siendo conocidas las poses P_t y P_{t+1} hay que llevar la pose P_{t+1} al sistema de referencia de coordenadas de P_t , o bien si se enfoca como un problema de vectores, hay que restar al punto P_{t+1} el punto P_t . Esta operación de sustracción en poses (matrices con orientación y posición) se consigue mediante la aplicación de la inversa:

$$(P_t)^{-1}P_{t+1} = P_d \quad (5.1)$$

La pose es una matriz en coordenadas homogéneas de tamaño total de 4x4 que está formada por una matriz de rotación \mathbf{R} 3x3 y un vector de traslación \mathbf{t} 3x1. La fila inferior de la matriz corresponde a parámetros de escala y perspectiva que no se usan en este contexto. La inversa de una pose se obtiene como se muestra en la Figura 5.4:

$$\left(\begin{array}{c|c} \mathbf{R} & \mathbf{t} \\ \hline \mathbf{0}^T & 1 \end{array} \right)^{-1} = \left(\begin{array}{c|c} \mathbf{R}^T & -\mathbf{R}^T \mathbf{t} \\ \hline \mathbf{0}^T & 1 \end{array} \right)$$

Figura 5.4: Inversa de una pose

De esta forma se consigue el desplazamiento incremental que ha estimado DIFODO con la llegada de la nueva imagen. A continuación, este desplazamiento se habrá de aplicar a la última posición conocida de SD-SLAM, pero para esto es preciso encontrar la relación entre los sistemas de referencia de coordenadas de ambos sistemas primero.

El desplazamiento obtenido depende de la orientación y posición de los ejes de DIFODO con respecto a la imagen. Que se haya detectado un desplazamiento de 2cm en el eje y (hacia la derecha respecto de la imagen) no implica necesariamente que haya que hacer ese desplazamiento en el eje y en el sistema de referencia original de SD-SLAM. Es posible que en SD-SLAM esa dirección esté representada por otro eje distinto al y , incluso puede que este rotado con respecto al de DIFODO, por lo que hay que encontrar la relación de transformación entre ambos sistemas de referencias.

La forma de obtener el sistema de referencias de coordenadas de cada algoritmo es manual. Si un desplazamiento hacia arriba se transmite como un movimiento positivo en el eje z , esa será la dirección y el sentido del eje z . Al menos dos ejes es lo que se necesita saber, pudiendo derivarse el tercero pues sabemos que son sistemas orto-normales.

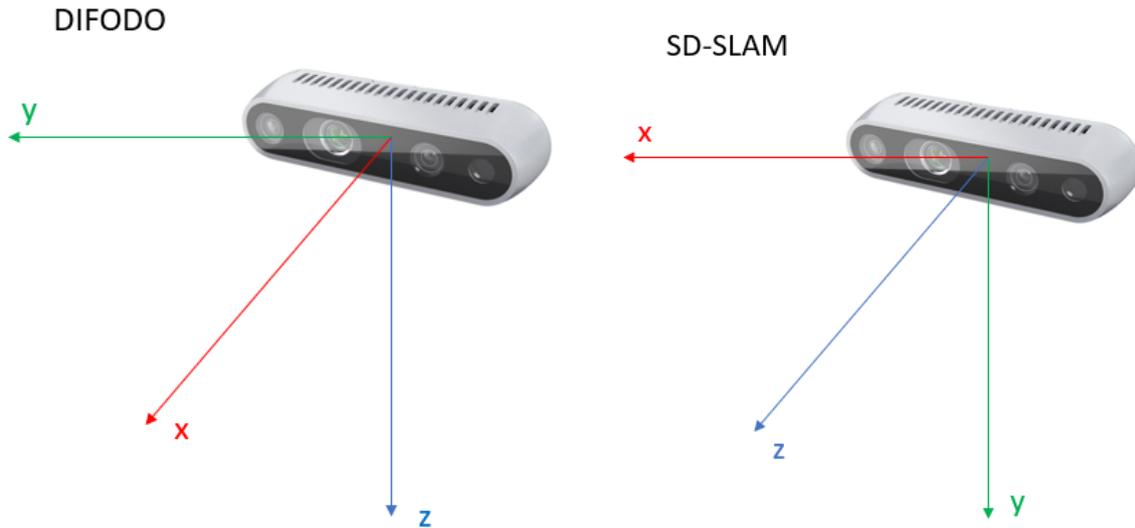


Figura 5.5: Dirección y sentido de los ejes de los sistemas de coordenadas

La Figura 5.5 muestra la dirección y el sentido de cada uno de los ejes de los sistemas de referencia de coordenadas de DIFODO y SD-SLAM. En ella se puede observar cómo ambos sistemas comparten dirección y sentido de sus ejes con respecto a la cámara, solo que existe una relación de rotación entre ellos. Conociendo esta relación ortogonal entre los ejes es fácil agregar el desplazamiento de DIFODO a la última pose de SD-SLAM. Un desplazamiento en el eje y en DIFODO se traducirá a un desplazamiento en el eje x para SD-SLAM. Lo mismo para los otros ejes.

$$X_{DIFODO} = Z_{SD-SLAM}$$

$$Y_{DIFODO} = X_{SD-SLAM}$$

$$Z_{DIFODO} = Y_{SD-SLAM}$$

Como último paso, hay que añadir el desplazamiento ahora según los ejes de SD-SLAM, a la última pose conocida. Este paso es directo salvo por un detalle en la implementación de SD-SLAM. Como se observa en la Figura 5.6, la última posición conocida P_t no se encuentra referenciada al mundo, sino que se conoce la posición del mundo respecto a la pose P_t . Esto aunque en un primer momento pueda parecer extraño, se debe a que la posición de los distintos

5.2. DISEÑO E IMPLEMENTACIÓN

puntos característicos encontrados en la imagen se calculan en primera instancia con respecto a la vista de la cámara y es a posteriori que estos son transformados al sistema de coordenadas mundo. El origen del sistema de coordenadas mundo actúa en este caso como un punto más, visto desde la perspectiva de la cámara. De forma que para añadir el desplazamiento y calcular la nueva pose P_{t+1} es necesario primero cambiar el sistema de referencia de P_d .

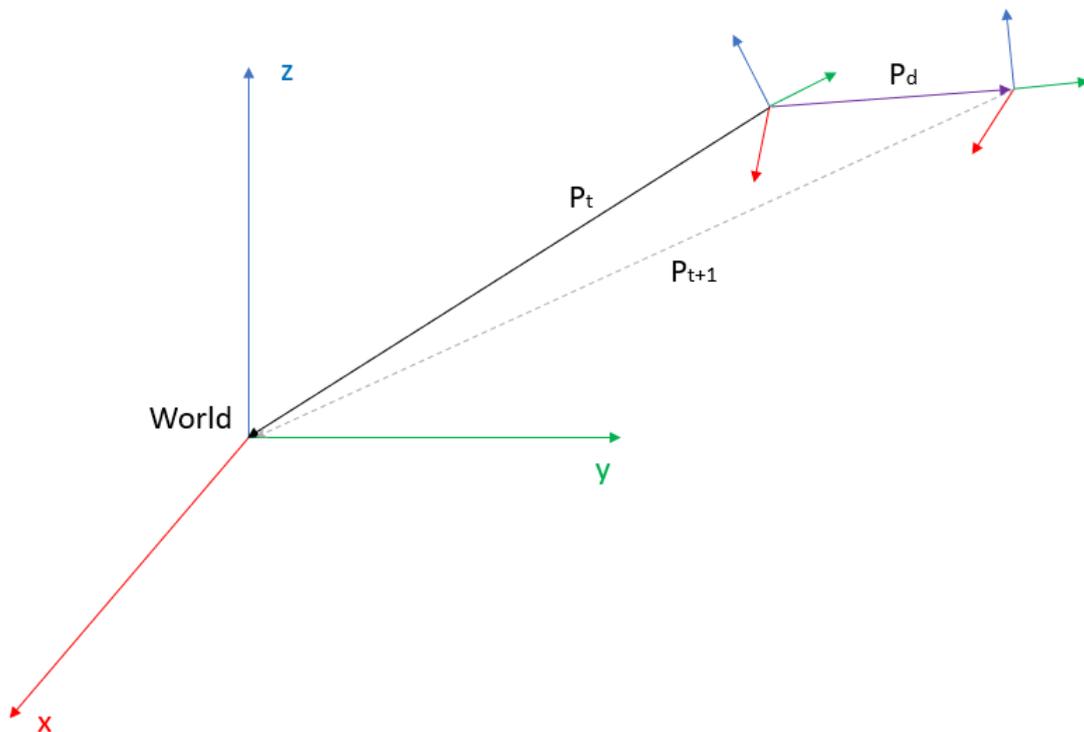


Figura 5.6: Adición del desplazamiento a la última pose conocida I

Como la pose de la cámara en SD-SLAM se expresa de esta manera, lo que se desea conocer no es la pose de P_{t+1} respecto del mundo, sino, la pose del mundo desde P_{t+1} es decir vista desde la cámara. En este caso, en la ecuación 5.2 es el desplazamiento el que se invierte y se aplica a la última pose conocida. Esto se observa geoméricamente en la Figura 5.7.

$$(P_d)^{-1}P_t = P_{t+1} \quad (5.2)$$

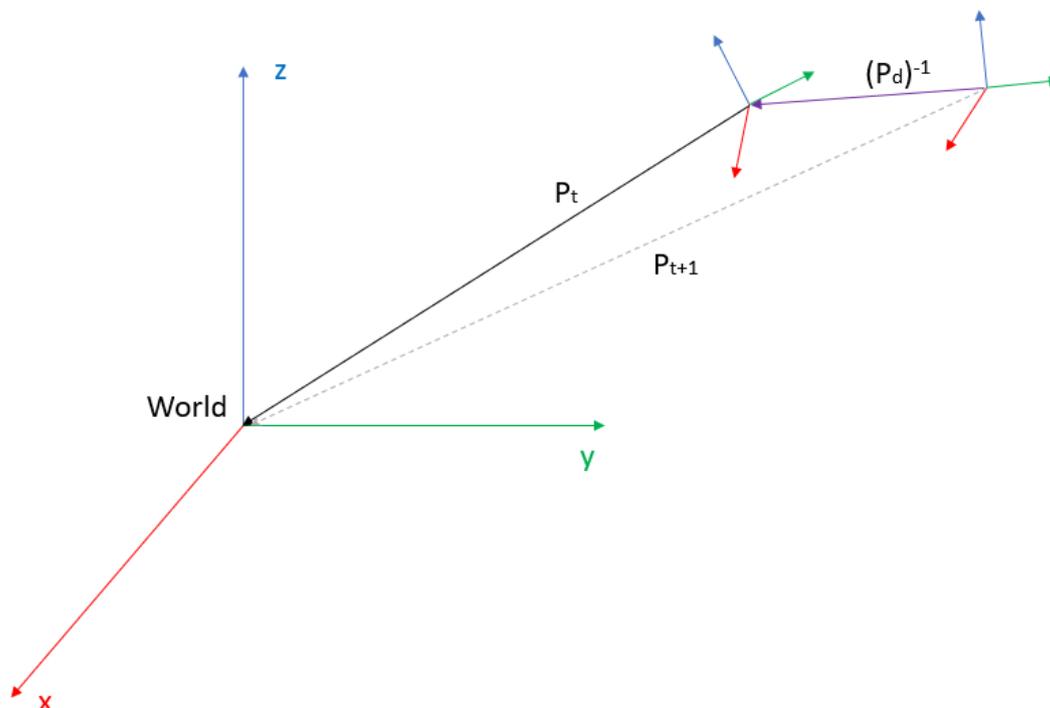


Figura 5.7: Adición del desplazamiento a la última pose conocida II

Finalmente, con esta última operación se obtiene la nueva pose P_{t+1} que representa la nueva pose estimada, habiéndose usado DIFODO para lograrla. Todas estas operaciones se han incorporado en el código de SD-SLAM+ a partir del código ya existente de SD-SLAM.

5.3. Validación experimental

En esta sección se va a evaluar el error en la estimación de la pose obtenido por el algoritmo original, SD-SLAM y el obtenido por su evolución SD-SLAM+.

Una de las características de SD-SLAM+ es que es capaz de funcionar cuando hay poca textura en el entorno gracias a su integración con DIFODO. Con el objetivo de probar el funcionamiento y el rendimiento del nuevo SD-SLAM+, es necesario simular pérdidas de textura en las distintas secuencias del conjunto de datos usado, ya que estas pérdidas no se observan en las secuencias originales salvo por una secuencia donde esto ocurre de forma natural.

Es por ello que se ha creado una versión exclusiva del software para realizar experimentos de ambos algoritmos. Estas versiones cuentan con un nuevo parámetro en el fichero de configura-

ción, donde se pueden configurar los intervalos de tiempo en los cuales se simulará esta pérdida de textura. Esto permite que el algoritmo en los intervalos configurados reciba imágenes RGB completamente negras mediante la creación de una matriz donde el valor de intensidad de todos los píxeles se encuentra a cero. Esta imagen totalmente negra imposibilita la obtención de puntos característicos y por tanto provoca que SD-SLAM no estime nuevas poses.

5.3.1. DIFODO como apoyo a SD-SLAM

Los experimentos que se presentan a continuación pretenden evaluar si la integración de DIFODO con SD-SLAM se ha diseñado correctamente y si SD-SLAM+ consigue una mejor estimación de la pose que SD-SLAM. Se han realizado una serie de test utilizando la misma secuencia de vídeo, **rgb_dataset_freiburg1_desk.bag**. Como esta secuencia no presenta pérdidas de textura naturales se han simulado intervalos donde sí se producen pérdidas de textura, para evaluar diferentes escenarios. En total cuatro tests se han realizado, siendo cada test más exigente que el anterior:

- **Test 1:** Se busca verificar si SD-SLAM+ se ha diseñado correctamente y es capaz de ejecutar DIFODO ante la pérdida de textura en la imagen RGB. Para esto se simula una única pérdida de textura de 1 segundo de duración.
- **Test 2:** Se busca comprobar la capacidad de SD-SLAM+ de utilizar DIFODO en varias ocasiones ante varias pérdidas de textura. Para esto se simulan tres intervalos diferentes donde se pierde la textura de la imagen RGB.
- **Test 3:** Se busca ver cómo afecta un uso prolongado de DIFODO a SD-SLAM+. Para esto se simula una pérdida de textura de larga duración.
- **Test 4:** Se busca crear un entorno complicado añadiendo varios intervalos donde se pierde la textura a lo largo de toda la secuencia. Concretamente se añaden intervalos al final de esta para evaluar cómo afecta DIFODO al cierre de bucle.

A continuación se muestran los resultados obtenidos para cada test:

Test 1

- **Intervalo sin textura simulados (segundos):** [4-5]
- **Descripción:** Se ha simulado una pérdida de textura del segundo 4 al 5 para verificar el correcto funcionamiento de SD-SLAM+ y ver si presenta mejores resultados que SD-SLAM.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 1.2528 | 0.8894 | 0.1415 | 0.8823 | 0.0350 | 2.1653 |
| SD-SLAM+ | 0.1863 | 0.1673 | 0.1199 | 0.0820 | 0.0352 | 0.3153 |

Tabla 5.1: Comparación del ATE en el Test 1: SD-SLAM vs SD-SLAM+

Los resultados de la anterior Tabla 5.1 muestran cómo el error de la nueva versión es mucho más pequeño que el de la versión original. Mientras que la versión original del algoritmo tiene un **RMSE** de 1.25m la nueva versión, SD-SLAM+ tiene un *RMSE* de 0.18m, un error 6 veces más pequeño en este caso. Como se aprecia en la Figura 5.8, en la imagen (a), la trayectoria estimada solo se localiza bien en la primera mitad de la trayectoria. Esto es debido a que a partir del segundo 4 y hasta el 5 desaparece la textura, lo que provoca que el algoritmo de SD-SLAM no pueda continuar con la estimación de la pose y entre en un estado de relocalización. El algoritmo seguirá en este estado hasta la vuelta, donde volverá a pasar por zonas conocidas pudiendo relocalizarse y volviendo a estimar la pose en los últimos tramos. Todo esto se traduce en un alto error ATE, ya que hay zonas durante las cuales no se ha producido estimación alguna. En el caso de la SD-SLAM+, Figura 5.8 (b), se encuentra estimación de la pose a lo largo de toda la trayectoria. Esto es gracias a que durante ese intervalo de 1 segundo donde no se cuenta con textura el nuevo algoritmo pasa a estimar la pose usando DIFODO. Cuando hay textura de nuevo el algoritmo recupera el *tracking* original usando ORB. En el caso de SD-SLAM+ se encuentra un error mucho más pequeño y cercano al que se obtendría en la secuencia original sin este intervalo sin textura.

Este funcionamiento satisfactorio sirve como validación global de que el nuevo algoritmo SD-SLAM+ ha sido bien programado y que su diseño realmente consigue mejorar la calidad de la autolocalización visual conseguida.

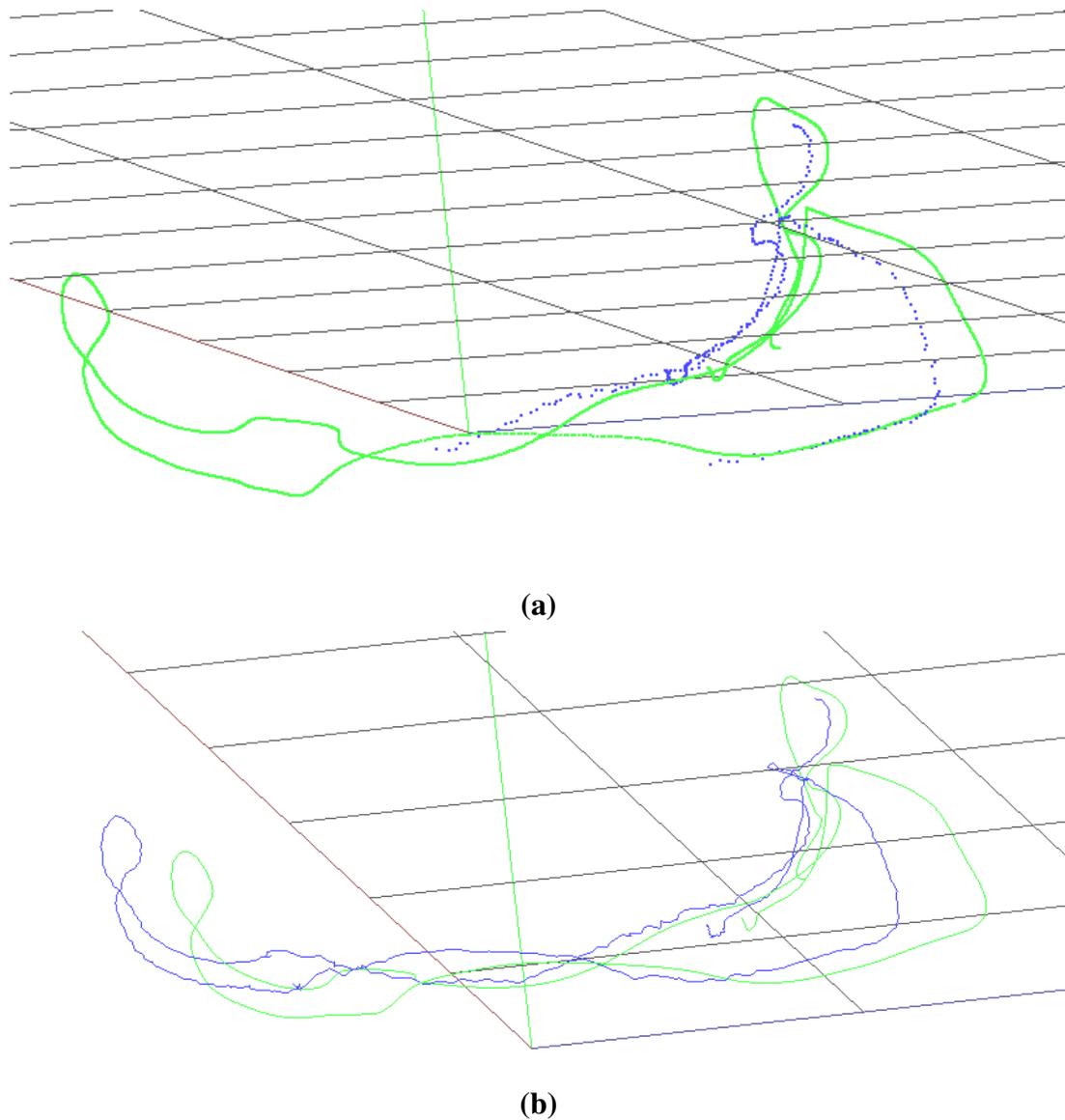


Figura 5.8: Trayectorias 3D estimadas por los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la versión original de SD-SLAM y la figura (b) a SD-SLAM+.

Test 2

- **Intervalos sin textura simulados (segundos):** [4-5, 6-7, 8-9]
- **Descripción:** En esta ocasión se aumenta el número de intervalos sin textura durante la secuencia. El objetivo es ver cómo se comporta SD-SLAM+ cuando se intercambia entre SD-SLAM y DIFODO repetidas veces.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max |
|----------|----------|-----------|-------------|---------|---------|--------|
| SD-SLAM | 1.2528 | 0.8894 | 0.1415 | 0.8823 | 0.0350 | 2.1653 |
| SD-SLAM+ | 0.1960 | 0.1781 | 0.1609 | 0.0819 | 0.0335 | 0.3265 |

Tabla 5.2: Comparación del ATE en el test 2: SD-SLAM vs SD-SLAM+

Los resultados para SD-SLAM siguen siendo los mismos pues desde que deja de hacer estimaciones de pose en el primer intervalo donde se pierde la textura, ya no es capaz de relocalizarse hasta la parte final. El resto de intervalos de pérdida de textura se producen antes de que la relocalización sea posible por lo que no afectan al error de trayectoria absoluto, siendo el de la Tabla 5.1 el mismo que en la Tabla 5.2.

Una vez más, se observa en la Tabla 5.2 cómo el error en el caso de SD-SLAM+ es bastante menor, gracias a que consigue seguir haciendo *tracking* con DIFODO en los momentos donde no hay textura consiguiendo una trayectoria más parecida a la real. Queda validado que SD-SLAM+ no tiene problema en utilizar de forma repetida la estimación de pose de DIFODO.

| Test Número | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|-------------|----------|-----------|-------------|---------|---------|---------|
| Test 1 | 0.1863 | 0.1673 | 0.1199 | 0.0820 | 0.0352 | 0.3153 |
| Test 2 | 0.1960 | 0.1781 | 0.1609 | 0.0819 | 0.0335 | 0.3265 |

Tabla 5.3: Test 2: Evolución del ATE en SD-SLAM+

La Tabla 5.3, muestra cómo ha evolucionado el error desde el test 1 al test 2. Se aprecia cómo a pesar de introducir más intervalos donde el *tracking* de DIFODO es necesario, el error permanece estable y los resultados son prácticamente iguales. Esto es lógico, pues ambos algoritmos DIFODO y SD-SLAM tenían un rendimiento similar para esta secuencia, como se vio en la sección 4.8, por lo que usar un algoritmo u otro no debería alterar el error en gran medida.

Test 3

- **Intervalos sin textura simulados (segundos):** [4-8]
- **Descripción:** En esta ocasión se va a analizar el comportamiento cuando no hay textura de forma prolongada en el tiempo. El objetivo es ver cómo se comporta SD-SLAM+ cuando se necesita acudir a DIFODO de forma prolongada.

5.3. VALIDACIÓN EXPERIMENTAL

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 1.2528 | 0.8894 | 0.1415 | 0.8823 | 0.0350 | 2.1653 |
| SD-SLAM+ | 0.1289 | 0.1117 | 0.0895 | 0.0643 | 0.0330 | 0.3389 |

Tabla 5.4: Comparación del ATE en el test 3: SD-SLAM vs SD-SLAM+

En la Tabla 5.4 se aprecia cómo los resultados para SD-SLAM siguen siendo los mismos, como ya se discutió en el anterior test. También se vuelve a observar cómo los resultados del SD-SLAM+ son mucho mejores que el algoritmo original.

| Test Número | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|-------------|----------|-----------|-------------|---------|---------|---------|
| Test 1 | 0.1863 | 0.1673 | 0.1199 | 0.0820 | 0.0352 | 0.3153 |
| Test 2 | 0.1960 | 0.1781 | 0.1609 | 0.0819 | 0.0335 | 0.3265 |
| Test 3 | 0.1289 | 0.1117 | 0.0895 | 0.0643 | 0.0330 | 0.3389 |

Tabla 5.5: Test 3: Evolución del ATE en SD-SLAM+

La Tabla 5.5 muestra cómo ha evolucionado el error desde el test 1 al test 3. Es curioso observar la relación del error entre el test 2 y el test 3 pues los intervalos de tiempo en ambos donde no hay textura suman un total de 3 y 4 segundos respectivamente durante la primera mitad de la trayectoria de la secuencia. El error es un 30 % menor en el test 3 frente al test 2. y el test 1. Como se pudo observar en la sección 4.8, en esta secuencia el rendimiento de ambos algoritmos, DIFODO y SD-SLAM es similar en cuanto al error total. El test 3 se caracteriza por utilizar de forma prolongada DIFODO, obteniendo mejores resultados que cuando se usaba por intervalos y durante menos tiempo en los test 1 y 2. Esto quiere decir que DIFODO estima la posición con mayor precisión que SD-SLAM durante la primera mitad de la trayectoria de la secuencia.

Test 4

- **Intervalos sin textura simulados (segundos):** [4-5, 8-9, 12-13, 16-17]
- **Descripción:** En esta ocasión se aumenta el número de intervalos sin textura durante la secuencia, a lo largo de toda esta. El objetivo es ver cómo se comporta SD-SLAM+ en las partes finales de la secuencia.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 1.2528 | 0.8894 | 0.1415 | 0.8823 | 0.0350 | 2.1653 |
| SD-SLAM+ | 0.3109 | 0.2845 | 0.2816 | 0.1253 | 0.03501 | 0.4830 |

Tabla 5.6: Comparación del ATE en el Test 4: SD-SLAM vs SD-SLAM+

Se observa de nuevo en la Tabla 5.6 cómo el error en el caso de SD-SLAM+ es bastante menor, sin embargo se aprecia un aumento en su error que merece ser estudiado con más detalle comparándolo con los anteriores tests. Esto se aprecia en la Tabla 5.7:

| Test Número | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|-------------|----------|-----------|-------------|---------|---------|---------|
| Test 1 | 0.1863 | 0.1673 | 0.1199 | 0.0820 | 0.0352 | 0.3153 |
| Test 2 | 0.1960 | 0.1781 | 0.1609 | 0.0819 | 0.0335 | 0.3265 |
| Test 3 | 0.1289 | 0.1117 | 0.0895 | 0.0643 | 0.0330 | 0.3389 |
| Test 4 | 0.3109 | 0.2845 | 0.2816 | 0.1253 | 0.0350 | 0.4830 |

Tabla 5.7: Test 4: Evolución del ATE en SD-SLAM+

En el test 4 se aprecia un aumento del error frente al test 2 de un 50%. De los cuatro intervalos sin textura del test 4, los dos primeros también se simularon en el test 2, por lo que el aumento del error solo se puede deber a los dos intervalos posteriores, a la parte final de la trayectoria. En la Figura 5.9 (b) se puede apreciar cómo en la parte final la estimación de la pose en el test 4 presenta una desviación, comparado con el test 2 en la Figura 5.9 (a). DIFODO en la parte final de la trayectoria estima con mayor error la pose que SD-SLAM, debido al hecho de que SD-SLAM en esta zona final se apoya en los cierres de bucle para mejorar sus estimaciones, algo que DIFODO no puede hacer al ser solo odometría, obteniendo una mejor estimación en el tramo final SD-SLAM frente a DIFODO. Esta es la razón de ver un incremento en el error al precisar usar DIFODO en el tramo final de la trayectoria, en el test 4.

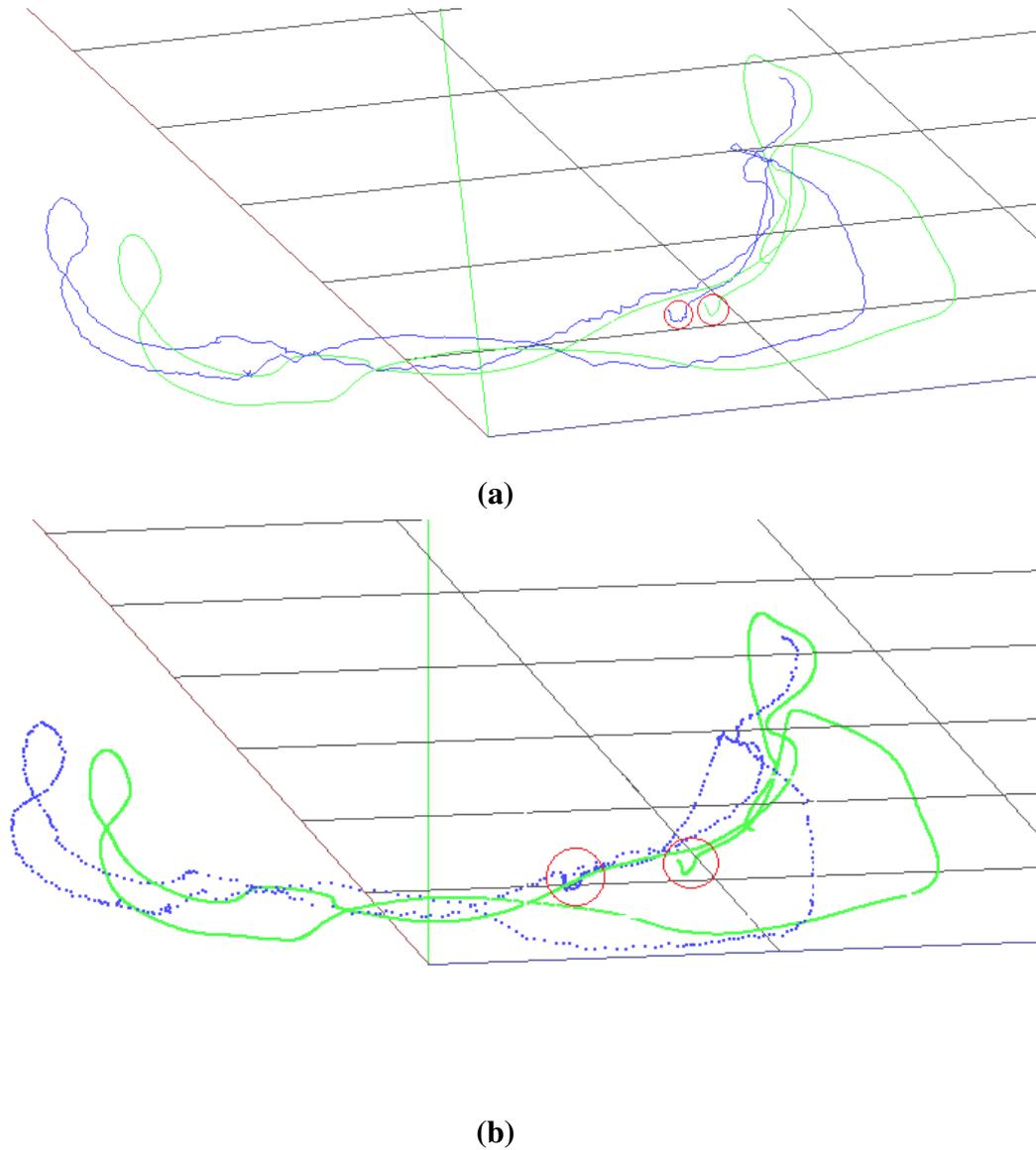


Figura 5.9: Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM+ en el test 2 y la figura (b) a la estimación de SD-SLAM+ en el test 4. En rojo se encuentran marcadas las posiciones finales para la trayectoria verdadera y estimada. En (a) se aprecia cómo estas están mucho más cercanas que en (b).

5.3.2. SD-SLAM+ en entornos con zonas de baja textura

En este experimento se busca evaluar la eficacia de SD-SLAM+ en una escena con una pérdida de textura natural. La secuencia usada para este experimento es la vista en la Subsección 4.3.7. Esta secuencia se caracteriza por tener un intervalo donde hay poca textura, que se

aprecia en el tercer cuarto del vídeo.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 0.2909 | 0.1705 | 0.1022 | 0.2192 | 0.0195 | 1.2202 |
| SD-SLAM+ | 0.1111 | 0.1042 | 0.1011 | 0.0385 | 0.0267 | 0.2188 |

Tabla 5.8: Comparación del ATE: SD-SLAM vs SD-SLAM+

En la Tabla 5.8 se hace una comparación entre las dos versiones de SD-SLAM. Como consecuencia de esta pérdida de textura hacia el final de la secuencia, SD-SLAM tiene un error medio y de mínimos cuadrados mucho más alto que la versión mejorada SD-SLAM+. Sin embargo, si uno se fija en la mediana se obtienen valores casi idénticos. Esto es así puesto que ambos algoritmos utilizan la información o la estimación de pose de SD-SLAM hasta que se pierde la textura. En ese punto, SD-SLAM se pierde, incapaz de hacer una estimación correcta en la pose debido a la ausencia de textura y, empieza a acumular error hasta el momento de relocalizarse hacia el final de la secuencia. Esto se puede observar en el error máximo para SD-SLAM de 1.22m frente a los 0.21m de error máximo de SD-SLAM+.

Por otro lado, SD-SLAM+ mantiene constante su estimación, siendo la media, mediana y el error por mínimos cuadrados muy similares, en torno a los 0.10m de error. Esto lo consigue utilizando la estimación de pose de DIFODO en el momento donde SD-SLAM no es capaz de estimar la pose.

No solo el error se ve afectado, pues mientras que SD-SLAM se encuentra perdido, este no es capaz de hacer el mapeado hasta que se reencuentra de nuevo, por lo que pérdidas de textura momentáneas afectan negativamente a la reconstrucción de entornos 3D también, como se puede ver en la Figura 5.10. Existe una zona recuadrada en rojo en la cual no hay puntos en el espacio puesto que durante esa parte el algoritmo se encontraba perdido intentando relocalizarse.

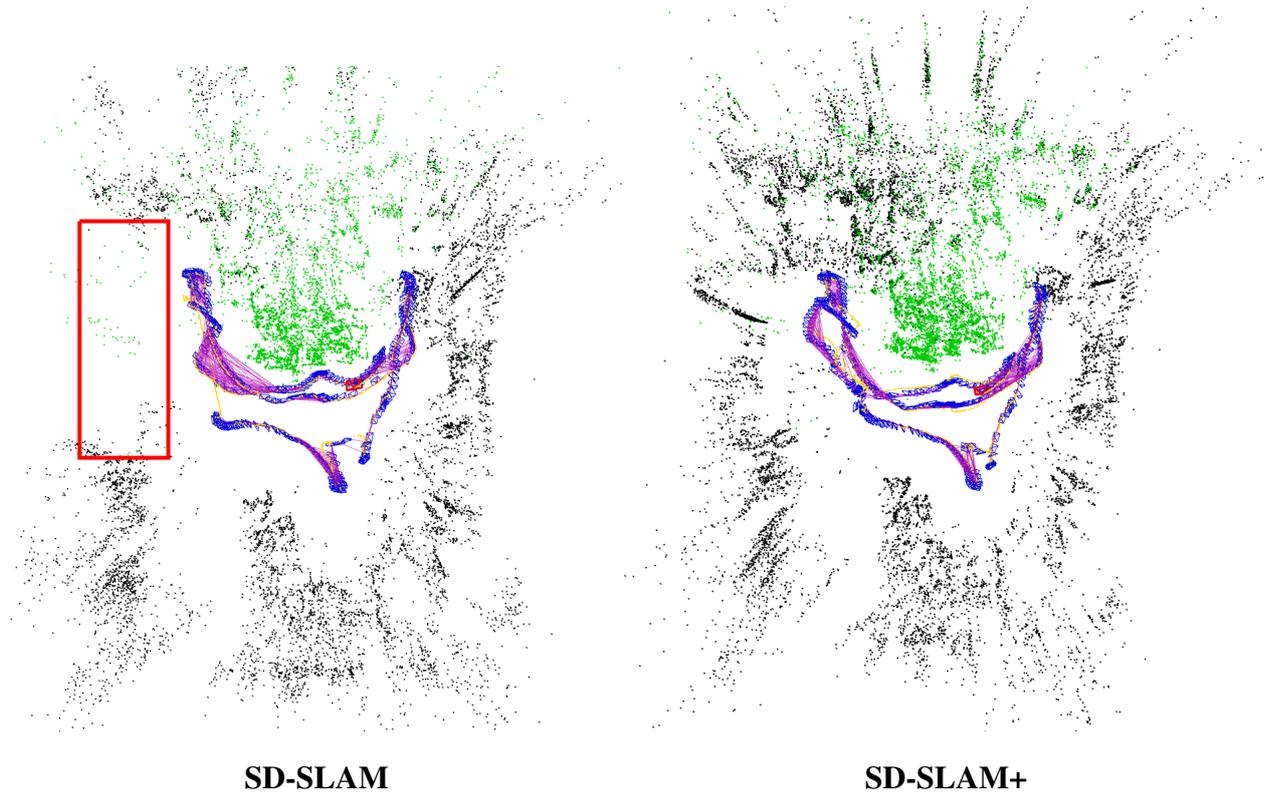


Figura 5.10: Comparativa de las trayectorias y reconstrucción del mapa usando los algoritmos SD-SLAM y SD-SLAM+.

5.3.3. SD-SLAM+ en entornos difíciles

En los experimentos anteriores se ha evaluado SD-SLAM+ en un entorno donde había mucha información de textura y también de estructura o textura de profundidad. Esto daba como resultado que independientemente del método usado para la estimación de la pose, la estimación fuera buena en todos los casos. En concreto en los anteriores tests se comprobaba cómo la falta de textura podía ser suplida por la estructura o información de profundidad de la escena. En este nuevo experimento, se va a comprobar cómo se desenvuelve el nuevo SD-SLAM+ en un entorno mucho más difícil, donde no hay estructura, dificultando las estimaciones de DI-FODO. La secuencia utilizada es `rgbd_dataset_freiburg3_nostructure_texture_far.bag`. Esta secuencia se caracteriza por no presentar textura en la parte final y no contar con posibilidad de cierres de bucle. De nuevo cuatro tests se han realizado, siendo cada test más exigente que el anterior:

- **Test 5:** Se evalúa cómo se comporta SD-SLAM+ hacia el final de la secuencia al no contar con textura ni estructura.
- **Test 6:** Se simula una pérdida de textura al inicio de la secuencia para evaluar cómo afecta esto a SD-SLAM+.
- **Test 7:** Se simulan varias pérdidas de textura para evaluar como afecta esto a SD-SLAM+.
- **Test 8:** Se simula una pérdida de textura de larga duración.

Test 5

- **Intervalos sin textura simulados (segundos):** []
- **Descripción:** No hay intervalos sin textura simulados, ya que la propia secuencia ya presenta una zona hacia el final donde SD-SLAM no encuentra textura suficiente para estimar la posición.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 1.3445 | 0.5417 | 0.0890 | 1.2306 | 0.0136 | 3.9141 |
| SD-SLAM+ | 0.1273 | 0.1072 | 0.0941 | 0.0686 | 0.0102 | 0.3038 |

Tabla 5.9: Comparación del ATE en el test 5: SD-SLAM vs SD-SLAM+

Como se observa en la Tabla 5.9, el error es mucho menor en el caso de SD-SLAM+, sin embargo la estimación no es muy diferente si nos fijamos visualmente en la Figura 5.11. Unos puntos al final de la Figura 5.11 (b) contenidos por un círculo rojo, que representan las poses estimadas por DIFODO cuando los cables desaparecen de escena, son la única diferencia con respecto a la Figura 5.11 (a). Sin embargo hay una gran diferencia numérica en el error entre ambos algoritmos. Esto se debe a que cuando SD-SLAM se pierde debido a la falta de textura, las siguientes posiciones que estima no se encuentran al final de la secuencia, sino que publica como pose su origen de coordenadas. Esto provoca que cuando se calcule al ATE se estén comparando posiciones del final de la secuencia con el inicio por lo que el error crece muchísimo. Esto se aprecia en el valor máximo de 3.9m de la Tabla 5.9, que corresponde a la comparación entre las posición finales y las poses del origen cuando SD-SLAM está perdido.

5.3. VALIDACIÓN EXPERIMENTAL

También se aprecia en cómo el error medio de SD-SLAM es muy pequeño y similar al de SD-SLAM+ pero al contar con estas errores finales tan grandes la media y el RMSE se ven afectados.

Por el contrario, SD-SLAM+ ayuda a seguir estimando posiciones en el final de la trayectoria lo que evita que el error crezca y hace que se mantenga estable. Aunque, como se aprecia en la Figura 5.11 (b), las estimaciones no son precisas debido a la ausencia de estructura de esta secuencia.

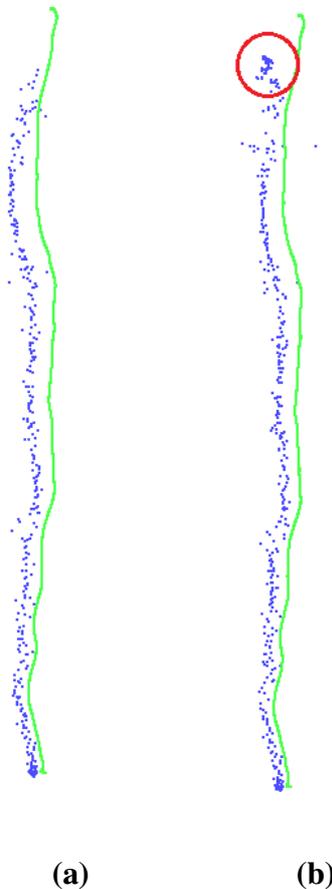


Figura 5.11: Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM y la figura (b) a la estimación de SD-SLAM+. Marcado en rojo se encuentran las estimaciones de DIFODO.

Test 6

- Intervalos sin textura simulados (segundos): [4-5]

- **Descripción:** En este nuevo test se simula un intervalo sin textura al inicio para ver cómo se comporta SD-SLAM+.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 2.5749 | 2.0802 | 2.2577 | 1.5174 | 0.0140 | 4.3386 |
| SD-SLAM+ | 0.3109 | 0.2845 | 0.2816 | 0.1253 | 0.0350 | 0.4830 |

Tabla 5.10: Comparación del ATE en el test 6: SD-SLAM vs SD-SLAM+

De nuevo el error cometido por SD-SLAM+ es mucho menor. En este caso SD-SLAM no ha sido capaz de continuar con la estimación a partir del intervalo en el que la textura se pierde. Como la secuencia no presenta posibles relocalizaciones no se vuelve a estimar bien la posición, como se aprecia en la figura Figura 5.12 y la Figura 5.13 de forma visual.

En el caso de SD-SLAM+ el error cometido es mucho menor, sin embargo se pueden apreciar en la zona resaltada de la Figura 5.12 (b) una serie de estimaciones que se acumulan. Al no contar con más estructura que el plano del suelo, las estimaciones de DIFODO no son muy fiables y las nuevas estimaciones no siguen la trayectoria sino que se acumulan en torno al último punto conocido. Sin embargo, como la pérdida de textura se simula durante un segundo únicamente, cuando se recupera la textura es capaz de seguir utilizando el *tracking* original de SD-SLAM y la estimación global de la trayectoria es mucho mejor que con SD-SLAM.

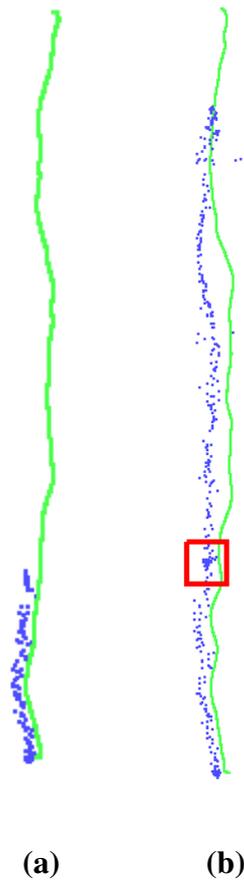


Figura 5.12: Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM y la figura (b) a la estimación de SD-SLAM+. Las poses delimitadas por el rectángulo rojo corresponden a las poses estimadas por DIFODO.

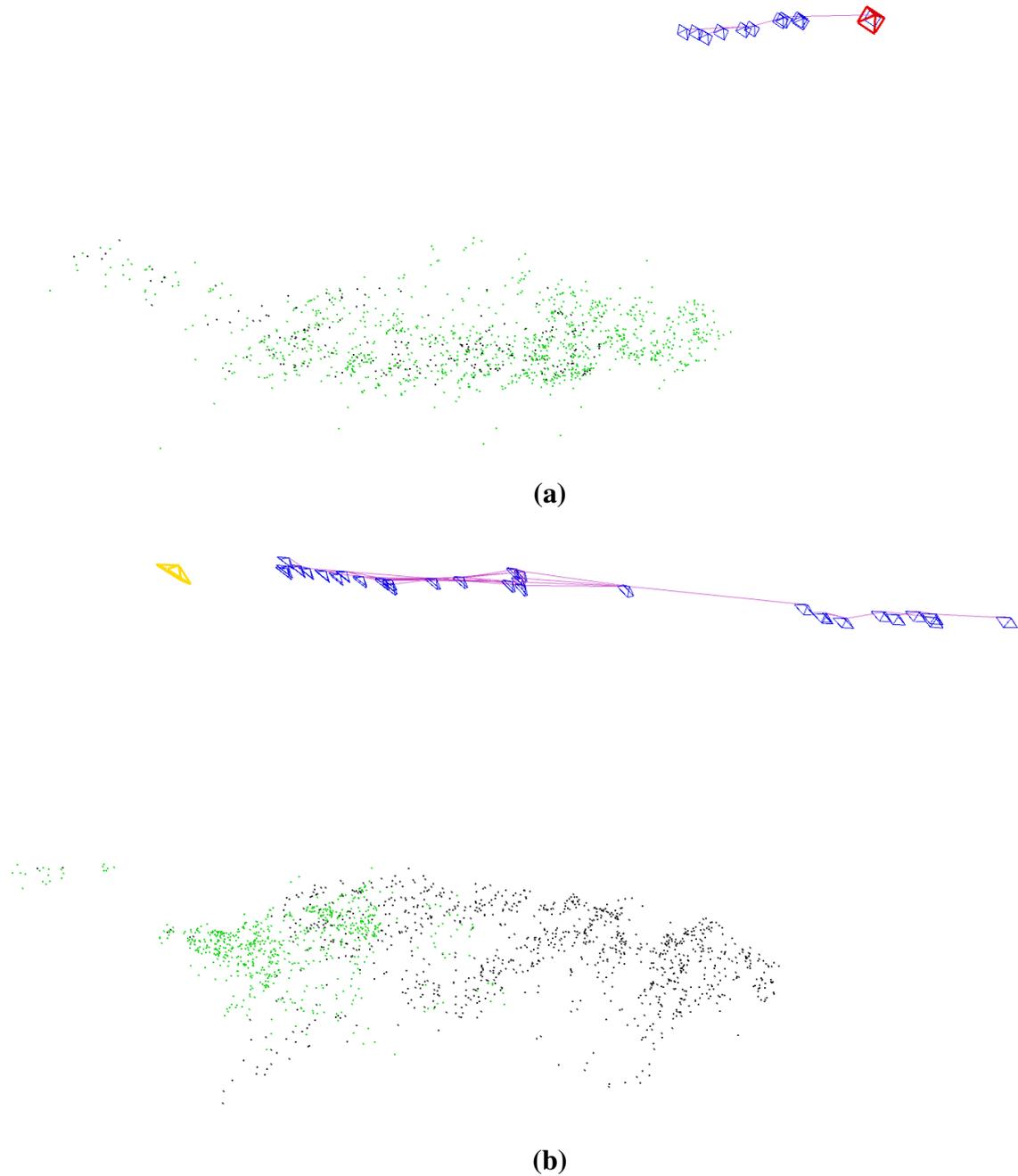


Figura 5.13: Trayectorias y reconstrucción del mapa hecha por los algoritmos en la interfaz SD-SLAM. La figura (a) corresponde a SD-SLAM y la figura (b) a SD-SLAM+. En azul se tiene la pose de cada uno de los fotogramas clave, en verde y negro los puntos del mapa, en amarillo la última posición estimada cuando se usa DIFODO.

En la Tabla 5.11 se compara el rendimiento de SD-SLAM+ entre el test 5 y el test 6. En esta

5.3. VALIDACIÓN EXPERIMENTAL

se observa cómo utilizar tan solo un segundo en el *tracking* de DIFODO aumenta el error en casi un 300%. El error medio pasa a ser de casi 0.30m frente a los 0.10m de la secuencia original en el test 5. Este aumento de 20cm en el error viene dado por ese segundo de intervalo donde la posición casi no avanza debido a la mala estimación de DIFODO, mientras que en la secuencia original durante ese segundo se avanzarán estos 20cm. Esto produce que las estimaciones de SD-SLAM+ vayan por detrás de la secuencia original 20cm para el resto de estimaciones.

| Test Número | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|-------------|----------|-----------|-------------|---------|---------|---------|
| Test 5 | 0.1273 | 0.1072 | 0.0941 | 0.0686 | 0.0102 | 0.3038 |
| Test 6 | 0.3109 | 0.2845 | 0.2816 | 0.1253 | 0.0350 | 0.4830 |

Tabla 5.11: Test 6: Evolución del ATE en SD-SLAM+

Test 7

- **Intervalos sin textura simulados (segundos):** [4-5, 6-7, 8-9]
- **Descripción:** Se añaden hasta 3 intervalos donde no hay textura.

| Versión | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|----------|----------|-----------|-------------|---------|---------|---------|
| SD-SLAM | 2.5749 | 2.0802 | 2.2577 | 1.5174 | 0.0140 | 4.3386 |
| SD-SLAM+ | 0.5854 | 0.4761 | 0.5655 | 0.3406 | 0.0098 | 1.0465 |

Tabla 5.12: Comparación del ATE en el test 7: SD-SLAM vs SD-SLAM+

Las prestaciones de SD-SLAM siguen siendo las mismas que en el test anterior, como era de esperar al no poder relocalizarse. De nuevo SD-SLAM+ lo hace mejor, aunque la precisión no es para nada buena (casi 0.5m de error medio por posición, algo que para algoritmos de SLAM es demasiado). Se visualiza en la Figura 5.14 (b) cómo la trayectoria que se sigue esta vez es mucho más irregular que la que se observaba en el test anterior Figura 5.14 (a). Cada vez que hay un intervalo donde se utiliza DIFODO se observa o bien una acumulación de puntos o un desvío en la trayectoria original. Es interesante observar cómo en el último intervalo se produce una rotación en la orientación por parte de DIFODO que produce que el *tracking* de

SD-SLAM+, una vez recuperada la información visual, estime un desplazamiento rectilíneo pero en una dirección incorrecta, alejando aún más las siguientes estimaciones.

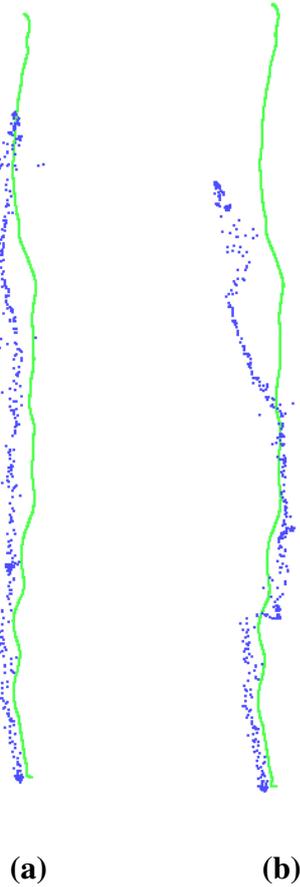


Figura 5.14: Trayectorias estimadas de los algoritmos. En verde se muestra la trayectoria estimada, en azul la trayectoria estimada. La figura (a) corresponde a la estimación de SD-SLAM+ para el test 6 y la figura (b) a la estimación de SD-SLAM+ para el test 7.

Test 8

- **Intervalos sin textura simulados (segundos):** [4-9]
- **Descripción:** Se busca evaluar SD-SLAM+ en una secuencia donde se simula un intervalo sin textura de larga duración.

5.3. VALIDACIÓN EXPERIMENTAL

| Test Número | RMSE (m) | Media (m) | Mediana (m) | STD (m) | Min (m) | Max (m) |
|-------------|----------|-----------|-------------|---------|---------|---------|
| Test 5 | 0.1273 | 0.1072 | 0.0941 | 0.0686 | 0.0102 | 0.3038 |
| Test 6 | 0.3109 | 0.2845 | 0.2816 | 0.1253 | 0.0350 | 0.4830 |
| Test 7 | 0.5854 | 0.4761 | 0.5655 | 0.3406 | 0.0098 | 1.0465 |
| Test 8 | 0.9999 | 0.7783 | 0.7649 | 0.6277 | 0.0131 | 1.6505 |

Tabla 5.13: Test 8: Evolución del ATE en SD-SLAM+

El error que se obtiene en este test 8 es cercano a los 0.8m de media, como se puede ver en la Tabla 5.13. Como era de esperar para esta secuencia, cuanto más tiempo se esté estimando posiciones con DIFODO mayor va a ser el aumento del error, ya que DIFODO en entornos sin estructura no es capaz de estimar con precisión la pose. En la Tabla 5.13 se observa cómo según se ha ido aumentando el tiempo donde se utilizaba DIFODO el error ha ido aumentando.

En la Figura 5.15 se puede visualizar cómo con la inclusión en cada test de un mayor intervalo de tiempo sin textura, los estimaciones han ido degradándose.

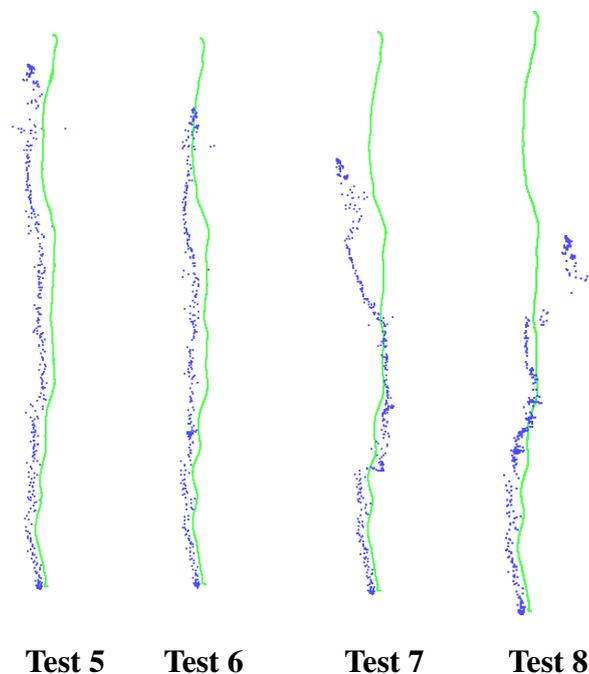


Figura 5.15: Comparativa de las trayectorias de los algoritmos. En verde se muestra la trayectoria verdadera, en azul la trayectoria estimada.

5.3.4. Evaluación del tiempo de procesamiento

En esta última sección se va a evaluar el coste computacional de añadir DIFODO a SD-SLAM, comparando los tiempos de procesamiento de SD-SLAM y SD-SLAM+. Para ello se va a utilizar la secuencia utilizada en el Subsección 5.3.2, **rgbd_dataset_freiburg3_floor.bag**. Para este experimento se ha usado DIFODO con una resolución de 320x240 y cinco niveles de pirámide.

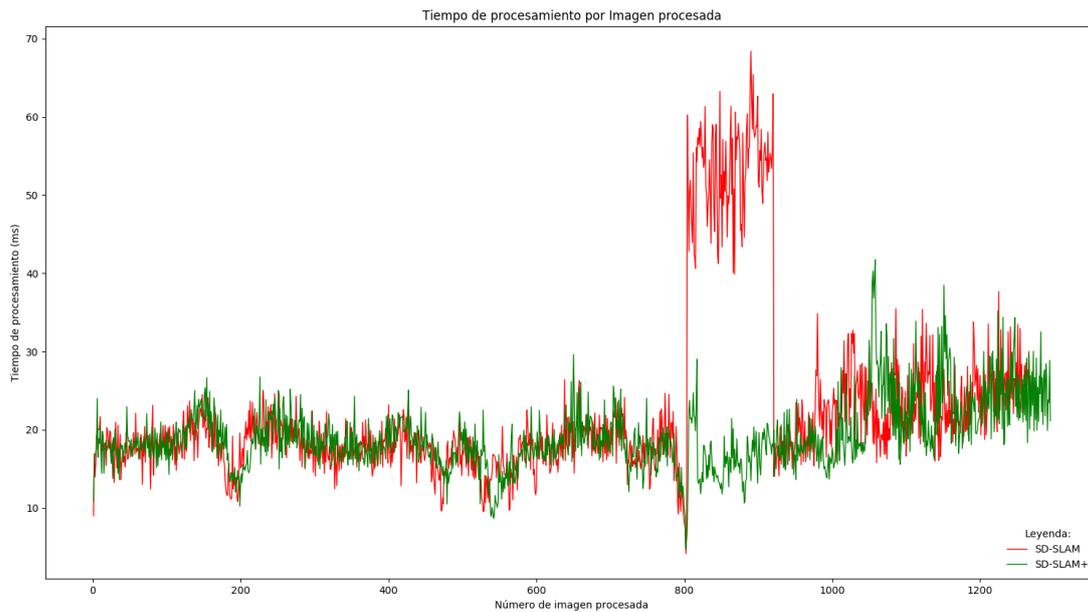


Figura 5.16: Comparativa del tiempo de procesamiento por imagen para la secuencia **rgbd_dataset_freiburg3_floor.bag**

La Figura 5.16 representa el tiempo de procesamiento por imagen utilizando el procesador Ryzen 5 2600, para la secuencia **rgbd_dataset_freiburg3_floor.bag**, con una longitud de 1360 fotogramas. El tiempo de procesamiento por imagen es similar para ambos, SD-SLAM y SD-SLAM+, manteniéndose durante casi toda la secuencia en torno a los 20ms. Sin embargo, se observa un periodo donde hay un aumento importante del tiempo de procesamiento para SD-SLAM, de la imagen 800 a la 950 aproximadamente. Durante estos 5 segundos, SD-SLAM pasa a tener un tiempo de procesamiento de aproximadamente 2.5 veces mayor con respecto al resto de la secuencia, sobrepasando los 30ms que precisa el tiempo real. La razón de este aumento es que SD-SLAM se pierde, imposibilitándole la estimación de nuevas poses, debido a la falta de textura en la escena. Cuando SD-SLAM se pierde, pasa al modo de relocalización, intentando

5.3. VALIDACIÓN EXPERIMENTAL

relocalizarse con las nuevas imágenes que le llegan, utilizando todos los *keyframes* guardados que tiene en el mapa para buscar emparejarlos. Este es un proceso costoso pues si hay muchos *keyframes* con características similares, el proceso de intentar emparejar la imagen actual con cada uno de esos puede llegar a consumir más tiempo del deseado, como en este caso.

Por otro lado SD-SLAM+, al no perderse por usar DIFODO durante ese periodo donde no hay textura, evita entrar en ese modo de relocalización. Esto no solo permite que SD-SLAM+ no se pierda, sino que permite mantener el tiempo de procesamiento por imagen por debajo del límite del *tiempo real*, algo que no se cumple con SD-SLAM cuando se pierde. En concreto se puede apreciar en la Figura 5.16 cómo el tiempo de procesamiento cuando se ejecuta DIFODO, entre la imagen número 800 y la 830 aproximadamente, es de media unos 25ms, un tiempo similar al de la estimación de pose de SD-SLAM con ORB.

Integrar DIFODO junto a SD-SLAM no solo no incrementa el coste computacional, sino que lo reduce al evitar entrar en este modo de relocalización. Además, mientras que la textura se mantenga, el algoritmo original no se verá afectado computacionalmente pues DIFODO solo añade tiempo computacional cuando el *tracking* original de SD-SLAM falla. En la práctica, cuando se usa DIFODO el tracking de SD-SLAM original no se usa, por lo que al sustituir uno con otro el tiempo de procesamiento se mantiene estable.

5.3.5. Conclusiones

En este capítulo se comprobado experimentalmente y de manera exhaustiva cómo DIFODO es un algoritmo que puede integrarse con SD-SLAM para hacer a este último más robusto frente a situaciones donde falta textura visual. Esto es así, pues es un algoritmo de odometría visual que sólo usa información de profundidad y cumple con el requisito de procesamiento en tiempo real.

Se ha comprobado cómo SD-SLAM es mejor que DIFODO en la mayoría de los casos siempre y cuando la imagen de entrada presente información de textura y de estructura suficientes. Sin embargo cuando hay ausencia de textura en la imagen, pero suficiente estructura, DIFODO estima la posición con un menor error que SD-SLAM, el cual directamente no estima posición alguna.

Por otro lado, ambos algoritmos fallan cuando no hay estructura ni textura por lo que SD-SLAM+ precisará de al menos estructura o textura en la escena en todo momento para funcionar

correctamente.

Finalmente, se ha comparado SD-SLAM con SD-SLAM+ viendo cómo en aquellos casos donde hay ausencia de textura en algún momento, SD-SLAM+ tiene un mejor rendimiento al ser capaz de estimar la posición frente a un SD-SLAM que se pierde y solo es capaz de volver a estimar posiciones si se da una relocalización. En contrapartida, cuando se utiliza DIFODO en SD-SLAM+, se pierde la capacidad de realizar un cierre de bucle en esos tramos. El cierre de bucle es similar a la relocalización, pero se realiza en segundo plano y sólo utiliza los fotogramas claves o *keyframes*. Cuando se utiliza DIFODO no se incorporan nuevos *keyframes* al mapa, debido a la falta de textura, por lo que el último *keyframe* no se actualiza y por lo tanto no se intenta realizar un cierre de bucle, proceso que solo ocurre con la disponibilidad de un nuevo *keyframe*. Una futura mejora sobre esto se presenta en la Sección 6.3.

Desde un punto de vista de coste computacional, la integración de DIFODO en SD-SLAM solo trae ventajas y no conlleva ninguna desventaja. SD-SLAM+ ayuda a evitar el elevado tiempo de procesamiento que conlleva la relocalización, al no perderse nunca.

Capítulo 6

Conclusiones

Una vez detallado el estudio de los antecedentes SD-SLAM y DIFODO, así como el nuevo algoritmo resultado de la combinación de ambos que se propone en este TFM, en este capítulo se resumen los objetivos conseguidos a lo largo del desarrollo y los resultados obtenidos con la implementación de SD-SLAM+.

6.1. Discusión

Se han realizado numerosas validaciones experimentales durante el desarrollo llegando a una serie de conclusiones que se resumen a continuación.

DIFODO ha demostrado ser un algoritmo de odometría que es capaz de funcionar en tiempo real con procesadores de prestaciones bajas, algo realmente importante para mantener el tiempo de computación de SD-SLAM+ bajo.

Se ha comprobado experimentalmente como SD-SLAM consigue una mejor estimación de la pose que DIFODO. Sin embargo, DIFODO consigue estimar mucho mejor la pose frente a SD-SLAM en aquellos entornos o escenas donde existe falta de textura, por lo que ambos sistemas pueden complementarse mutuamente.

Se ha validado cualitativamente y cuantitativamente la disminución en el error de la pose estimada tanto en escenas reales como en escenas con simulaciones, presentando en todos los casos mejoras frente a la anterior propuesta. Esto es posible pues ahora SD-SLAM+ es capaz de soportar pérdidas temporales de textura sin que esto afecte a su rendimiento, donde antes SD-SLAM no era capaz de seguir funcionando. Todos los experimentos de la Sección 5.3 muestran

una mejora significativa en la estimación de la pose frente a SD-SLAM.

Se ha demostrado cómo el tiempo de ejecución del nuevo desarrollo es igual o menor frente al original. Esto se consigue gracias al bajo tiempo de cómputo requerido por DIFODO y a la eliminación de la necesidad de relocalización, como se aprecia en la Subsección 5.3.4.

La integración de DIFODO en SD-SLAM supone la pérdida de la capacidad de relocalizarse, pues la relocalización solo se produce cuando la localización no puede realizarse o no es precisa. Como ahora SD-SLAM+ no sufre las implicaciones de entornos de baja textura este siempre puede localizarse. Por otro lado, el cierre de bucle sigue permitiendo optimizar globalmente, aunque solo tiene en cuenta para realizar esta optimización la información de los fotogramas claves, los cuales solo se generan durante la estimación con la información RGB.

Se ha comprobado cómo a pesar de las mejoras, entornos donde no hay ni textura ni estructura en la escena siguen presentando un desafío para este tipo de algoritmos basados en información visual. Este es un problema que todavía no se ha conseguido resolver y es posible que no llegue a hacerse salvo con la inclusión de otros sensores que utilicen otro tipo de información.

6.2. Objetivos conseguidos

Se ha conseguido satisfactoriamente el objetivo principal que era mejorar el algoritmo de visual SLAM conocido como SD-SLAM con la creación de SD-SLAM+. SD-SLAM no es capaz de localizarse o estimar la pose cuando la escena carece de textura. Este problema es solucionado por SD-SLAM+ al utilizar la información de profundidad disponible en cámaras RGB-D para estimar la pose cuando SD-SLAM no es capaz de hacerlo mediante la combinación con el algoritmo DIFODO.

Otro de los objetivos era explorar distintas propuestas del estado del arte que solucionasen el problema de la odometría con información de profundidad. Distintas propuestas se analizan en el Capítulo 3, destacando SDO y DIFODO entre otras. Por las razones descritas durante el Capítulo 4, DIFODO se elige como algoritmo de odometría para integrarse con SD-SLAM. Los detalles de la integración y del funcionamiento de SD-SLAM+ se describen en el Capítulo 5. Finalmente, se realiza una validación experimental del nuevo algoritmo que lo compara con la anterior versión, describiendo las mejoras logradas, consiguiéndose un nuevo algoritmo de

SLAM más robusto y preciso.

Todo lo anterior se ha logrado cumpliendo con el único requisito o restricción de este TFM, que el sistema funcionase en tiempo real. El algoritmo original ya conseguía esto, por lo que la integración no debía suponer un mayor esfuerzo computacional. Se consigue cumplir esta restricción utilizando la nueva estimación de pose solo en los momentos donde es realmente necesaria.

6.3. Trabajos futuros

Durante el desarrollo del TFM se ha visto la posibilidad de explorar algunos caminos que podrían suponer mejoras en el algoritmo desarrollado. Estas posibilidades se detallan a continuación para orientar a trabajos derivados y seguir mejorando SD-SLAM+.

- La estimación de DIFODO también puede no ser lo suficientemente buena, sería útil establecer una forma de determinar cuando la incertidumbre en la estimación de la pose de DIFODO es lo suficientemente grande como para que el sistema se vaya a un estado de perdido (por falta de información de textura y estructura en la escena). Hay un parámetro en DIFODO que ofrece esta incertidumbre ya. El objetivo sería encontrar un valor de forma empírica que indique cuando las estimaciones de DIFODO no son lo suficientemente buenas debido a la falta de estructura.
- Actualmente DIFODO solo permite la estimación de odometría y no añade información al mapa que permita ser usada posteriormente, por ejemplo para cierre de bucles. Sería interesante extraer puntos característicos de la imagen de profundidad con otro algoritmo como se hace en [50].
- Estudiar alternativas de odometría con solo información de profundidad como SDO [32]. Con el diseño actual es fácil intercambiar el algoritmo de odometría DIFODO por otro, permitiendo la posibilidad de realizar un estudio con el que presenta mejores estimaciones.
- Mejorar la estimación de DIFODO, por ejemplo añadiendo ICP de forma que se pueda reducir la deriva del algoritmo utilizando la información de imágenes anteriores [50].

6.3. TRABAJOS FUTUROS

También se puede extraer puntos característicos de la imagen de profundidad con descriptores 3D para buscar hacer un cierre de bucle con la información de profundidad. El objetivo atractivo sería tener dos algoritmos de SLAM uno basado en RGB y otro en D que puedan complementarse.

- Permitir que la primera estimación de pose pueda realizarse con DIFODO si se empieza en una zona sin textura donde ORB no puede inicializarse debido a la falta de puntos característicos.
- Realizar pruebas sobre robots reales, por ejemplo cuadricópteros, desplegando el algoritmo SD-SLAM+ sobre hardware embarcado.

Bibliografía

- [1] P. J. Besl and N. D. McKay. A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256, 1992.
- [2] Peter Biber and Wolfgang Strasser. The normal distributions transform: a new approach to laser scan matching. *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, 3:2743–2748 vol.3, 2003.
- [3] Jean-Yves Bouguet et al. Pyramidal implementation of the affine lucas kanade feature tracker description of the algorithm. *Intel Corporation*, 5(1-10):4, 2001.
- [4] Thomas Brox, Andrés Bruhn, Nils Papenberg, and Joachim Weickert. High accuracy optical flow estimation based on a theory for warping. In Tomás Pajdla and Jiří Matas, editors, *Computer Vision - ECCV 2004*, pages 25–36, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.
- [5] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '96*, page 303–312, New York, NY, USA, 1996. Association for Computing Machinery.
- [6] Andrew J. Davison. Slam with a single camera. *Workshop on Concurrent Mapping and Localization for Autonomous Mobile Robots, ICRA*, 2002.
- [7] Andrew J. Davison. Real-time simultaneous localisation and mapping with a single camera. *ICCV*, 3:1403–1410, 2003.

- [8] Andrew J. Davison, Ian D. Reid, Nicholas Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:1052–1067, 2007.
- [9] F. Endres, J. Hess, J. Sturm, D. Cremers, and W. Burgard. 3-d mapping with an rgb-d camera. *IEEE Transactions on Robotics*, 30(1):177–187, 2014.
- [10] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam: Large-scale direct monocular slam. In *ECCV*, 2014.
- [11] Jakob Engel, Jürgen Sturm, and Daniel Cremers. Semi-dense visual odometry for a monocular camera. *2013 IEEE International Conference on Computer Vision*, pages 1449–1456, 2013.
- [12] Hamed Fathi. Video image sequence super resolution using optical flow motion estimation. 08 2015.
- [13] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981.
- [14] C. Forster, M. Pizzoli, and D. Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.
- [15] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22, 2014.
- [16] Dorian Gálvez-López and Juan D. Tardós. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28:1188–1197, 2012.
- [17] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

-
- [18] M. Jaimez and J. Gonzalez-Jimenez. Fast visual odometry for 3-d range sensors. *IEEE Transactions on Robotics*, 31(4):809–822, 2015.
- [19] C. Kelley. *Iterative Methods for Optimization*. Society for Industrial and Applied Mathematics, 1999.
- [20] C. Kerl, J. Sturm, and D. Cremers. Dense visual slam for rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2100–2106, 2013.
- [21] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In *2013 IEEE International Conference on Robotics and Automation*, pages 3748–3754, 2013.
- [22] Georg Klein and David William Murray. Parallel tracking and mapping for small ar workspaces. *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 225–234, 2007.
- [23] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. G2o: A general framework for graph optimization. *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011.
- [24] Steven Lovegrove and Andrew J. Davison. Real-time spherical mosaicing using whole image alignment. In *ECCV*, 2010.
- [25] Kok-Lim Low. Linear least-squares optimization for point-to-plane icp surface registration. 01 2004.
- [26] Raúl Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, 2017.
- [27] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31:1147–1163, 2015.
- [28] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

- [29] Richard A. Newcombe, Steven Lovegrove, and Andrew J. Davison. Dtam: Dense tracking and mapping in real-time. *2011 International Conference on Computer Vision*, pages 2320–2327, 2011.
- [30] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–I, 2004.
- [31] Eduardo Perdices and José Cañas Plaza. Sdvl: Efficient and accurate semi-direct visual localization. *Sensors*, 19:302, 01 2019.
- [32] S. M. Prakhya, L. Bingbing, L. Weisi, and U. Qayyum. Sparse depth odometry: 3d key-point based pose estimation from dense depth data. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4216–4223, 2015.
- [33] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *ECCV*, 2006.
- [34] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, pages 430–443, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
- [35] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.
- [36] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary R. Bradski. Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*, pages 2564–2571, 2011.
- [37] D. Scaramuzza and R. Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *IEEE Transactions on Robotics*, 24(5):1015–1026, 2008.
- [38] Aleksandr Segal, Dirk Hähnel, and Sebastian Thrun. Generalized-icp. 06 2009.
- [39] Jianbo Shi and Carlo Tomasi. Good features to track. In *CVPR*, 1994.

-
- [40] Hagen Spies, Horst W. Houssecker, Bernd Jähne, and John L. Barron. Differential range flow estimation. In *DAGM-Symposium*, 1999.
- [41] Hagen Spies, Bernd Jähne, and John L. Barron. Range flow estimation. *Computer Vision and Image Understanding*, 85(3):209 – 231, 2002.
- [42] F. Steinbrücker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pages 719–722, 2011.
- [43] Henrik Stewénius, Chris Engels, and David Nistér. Recent developments on direct relative orientation. 2006.
- [44] J. Sturm, W. Burgard, and D. Cremers. Evaluating egomotion and structure-from-motion approaches using the TUM RGB-D benchmark. In *Proc. of the Workshop on Color-Depth Camera Fusion in Robotics at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.
- [45] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment - a modern synthesis. In *Workshop on Vision Algorithms*, 1999.
- [46] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle adjustment — a modern synthesis. In Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, pages 298–372, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.
- [47] S. Ullman. The interpretation of structure from motion. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 203(1153):405–426, 1979.
- [48] Thomas Whelan, Stefan Leutenegger, Renato Moreno, Ben Glocker, and Andrew Davison. Elasticfusion: Dense slam without a pose graph. 07 2015.
- [49] Thomas Whelan, John McDonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J. Leonard. Kintinuous: Spatially extended kinectfusion, July 2012.
- [50] Shibo Zhao and Zheng Fang. Direct depth slam: Sparse geometric feature enhanced direct depth slam system for low-texture environments. *Sensors*, 18:3339, 10 2018.