Trabajo de Fin de Grado

Odometría con cámaras estereoscópicas en robótica

Odometry with stereoscopic cameras in robotics

Fabián Díaz Lorenzo

La Laguna, 5 de mayo de 2016

D. **Jonay Tomás Toledo Carrillo**, con N.I.F. 78.698.554-Y profesor contratado Doctor del Departamento de Ing. Informática y de Sistemas de la Universidad de La Laguna, como tutor

C E R T I F I C A (N)

Que la presente memoria titulada:

"Odometría con cámaras estereoscópicas en robótica."

ha sido realizada bajo su dirección por D. **Fabián Díaz Lorenzo**, con N.I.F. 43.838.583-P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 5 de julio de 2016.

Agradecimientos

Especialmente me gustaría agradecer el conocimiento, apoyo y paciencia a mi tutor Jonay Tomás Toledo Carillo

También me gustaría agradecer a mi familia y amigos por el apoyo ofrecido.

Licencia

Resumen

El objetivo de este trabajo es el realizar una comparación de los distintos algoritmos para la estimación de la odometría a partir de cámaras.

Para ello se ha usado ROS, un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo.

Explicaremos los distintos algoritmos para la extracción de características de una imagen y nos centraremos en la realización pruebas desarrolladas en distintos entornos. Se utilizarán los algoritmos VISO2 y el nodo Stereo Odometry perteneciente a la librería RtabMap para realizar dichas pruebas. Dichas pruebas se realizarán con la cámara estéreo PS4 Eye.

Todas las pruebas se realizarán utilizando la silla perenquén, la cual tiene implementada odometría laser y mecánica que nos servirán para comparar con los resultados de los algoritmos de odometría visual.

Palabras clave: ROS, Odometría Visual, PS4 Eye, extracción de características, estimación.

Abstract

The objective of this work is make a comparison of different algorithms for estimating the odometry from cameras.

We will use ROS, a framework for the development of software for robots that provides the functionality of an operating system.

We explain the different algorithms for extracting features of an image and focus on the tests to be developed in different environments. the VISO2 algorithms and Stereo Odometry RtabMap node belonging to the library is used for these tests. These tests are performed with the stereo camera PS4 Eye.

All tests were made using the perequen chair, which has implemented odometry and laser mechanics that will help us to compare the results of algorithms visual odometry

Keywords: *ROS, Visual Odometry, PS4 Eye , extracting features, estimation*

Índice General

Índice de figuras

Índice de tablas

Capítulo 1. Antecedentes y Estado Actual

1.1 ¿Qué es la odometría?

La palabra odometría se compone de las palabras griegas odos (que significa "camino") y metron (que significa "medida").

Se denomina odometría al uso de los datos de los sensores para estimar los cambios en la posición de un objeto en el tiempo, es decir, estimar lo que se ha movido un robot en el tiempo.

Este método no es preciso, pues no se basa en medias absolutas, por eso se usa como sistema de estimación. A corto plazo, los errores de medida suelen ser despreciables, pero si espera el tiempo suficiente, la acumulación de todos los errores de medida llega a ser muy considerable, por lo que se hace necesaria la intervención de algún otro sistema corrector que restablezca el sistema y especifique de algún modo cuál es su posición real con respecto al punto de referencia inicial.

Una de las odometría más comunes es la denominada odometría Mecánica que gracias a los encoders, se puede monitorizar eléctricamente la posición de un eje giratorio (en este caso las ruedas) y gracias a eso saber hacia dónde se ha movido el robot. El problema de esta odometría viene dado a problemas de deslizamientos de las ruedas que producen estimaciones de movimiento erróneos. Este error es aun mayor cuando el vehículo no opera en superficies uniformes ya que las lecturas de la odometría se vuelven menos fiables con el tiempo.

Por ello se suele usar en conjunto con otra odometría como puede ser la odometría laser que utiliza láseres para medir la distancia con los obstáculos y mediante los ángulos que forme se puede estimar de forma más precisa hacia donde se ha movido el robot. El problema de esta odometría es que los sensores necesarios para poder

implementarla en un sistema tienen un coste bastante elevado.

Otra opción es la odometría visual que es el proceso de determinar la posición y orientación de un robot mediante el análisis de las imágenes capturadas por una o varias cámaras que incorpore.

1.2 Tipos de Odometría visual

Dependiendo del número de cámaras podemos referirnos a ella como:

- Odometría visual monocular: Este tipo son menos usados a pesar de que puede haber cámaras con bastantes grados de visión presentan el problema de que para estimar la posición a través de una sola cámara es necesario saber perfectamente el ángulo de la posición de la cámara con respecto al suelo y hacer un mapeado. Esto produce más ruido que la odometría visual estereoscópica.
- Odometría visual Estereoscópica: Basada en la visión humana, es aquella visión que utiliza dos imágenes que provienen de dos cámaras distintas separadas a una cierta distancia la una de la otra. A partir de obtener las imágenes esto se buscan puntos de interés en las imágenes y se establece una correspondencia entre los puntos que corresponden a un solo punto en la escena tomada, de esta forma mediante una sencilla triangularización puede hallarse la distancia de este punto a las cámaras. Con esto podemos generar una imagen 3D de lo que se observa en las imágenes.

  En vez de usar dos cámaras independientes se usan las cámaras estereoscópicas que constan de dos objetivos idénticos y en perfecta sincronía dos imágenes separadas a una cierta distancia.

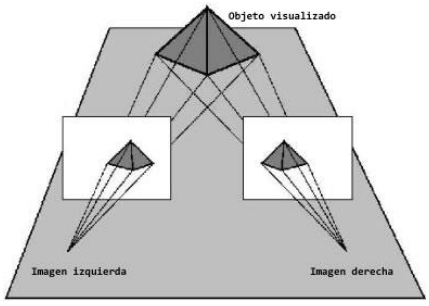

Figura 1.1. Visión Estereoscópica.

1.3 ROS

Es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo. ROS provee servicios estándar del sistema operativo como abstracción de hardware, control de dispositivos de bajo nivel, implementación de las funciones utilizadas normalmente. Es de código libre y se basa en un modelo de paso de suscripción y envió de mensajes. Esto lo hace realmente flexible y adaptable para las necesidades del usuario.

Las partes principales son: el núcleo de framework y ros-pkg, que es un conjunto de paquetes aportados por la contribución de los usuarios que implementan funcionalidades como localización y mapeo, planificación, simulación, etc.

Organiza el software en una arquitectura de grafos en la que los nodos son porciones ejecutables de código y esos se comunican entre sí a través de 'topics'. Todos estos nodos son capaces de encontrarse y comunicarse gracias al nodo ROS master que permite la intercomunicarse entre ellos.

1.4 RTABMAP

RTAB-Map (Real-Time Appearance-Based Mapping, Mapeado basado en apariencia en tiempo real) es una librería, en su momento independiente y ahora parte del sistema ROS, basado en un enfoque SLAM RGB-D fundamentado en un detector de cierre de bucle global (Consiste en la detección de terreno nuevo descubierto como localizaciones pasadas) con restricciones de tiempo real. Nos permite generar nubes de puntos en 3D del entorno, crear mapa 2D para la navegación, etc.

1.4.1 SLAM

SLAM (Simultaneous Localization and Mapping) Es una técnica usada por robots y vehículos autónomos para construir un mapa de un entorno desconocido en el que se encuentra, a la vez que estima su trayectoria al desplazarse dentro de este entorno.

Capítulo 2. Odometría Visual Estereoscópica

Para poder entender mejor las pruebas realizadas y el propósito de este trabajo, en este apartado, hablaremos de los aspectos teóricos en los que se basan los algoritmos de odometría visual estereoscópica. En estos apartados se explicarán los algoritmos de extracción de características de las imágenes fundamentales para poderse situar en el espacio, así como la obtención de la profundidad y como se realiza la estimación del movimiento.

2.1 Algoritmos para extracción de las características en imágenes

La extracción de característica de una imagen consistes en buscar y extraer puntos característicos de una imagen. Esto es necesario para poder de extrapolar esos puntos a las imágenes posteriores y así estimar el movimiento realizado.

Denominamos característica a una parte interesante o critica de una imagen. Dependiendo de lo eficiente que sea el algoritmo de detección de características influirá en la calidad del algoritmo de visión.

Debido a que este algoritmo es una de las primeras etapas en realizar en el desarrollo del algoritmo de visión, existe una amplia variedad según el tipo de características que detecta y la complejidad computacional. Los podemos clasificar en tres grupos:

- Detector de regiones (Blobs detectors).
- Detector de esquinas (Corners detectors).
- Detector de bordes (Edge detectors)

2.1.1 Detector de regiones

Están basados en implementar técnicas de detección de punto o regiones clara oscuras de una imagen. La

implementación de este tipo de detectores es bastante relevante ya que aportan información sobre regiones que otros detectores no capta. Suelen ser usado para identificación y seguimiento de objetos. Estas técnicas están implementadas en multitud de aplicaciones como la detección de rostros o sonrisas en imágenes, análisis de objetos en movimiento para sistemas de vigilancia.

Dentro de estos detectores destacamos los siguientes algoritmos:

- SIFT (Scale-Invariant Featura Transform): El algoritmo SIFT es un método para extraer puntos característicos invariantes y distintivos de una imagen que pueden ser usados para mejorar la correspondencia entre dos vistas diferentes de un objeto o una escena.

  Los puntos característicos obtenidos por SIFT son invariantes a escala y rotación de la imagen, y son mostradas para proporcionar una comparación robusta a pesar de que exista un amplio rango de distorsiones afines, cambios en la vista 3D, adición de ruido a la escena y cambios en la iluminación. Los puntos se distinguen claramente, en el sentido en que un único punto se puede corresponder correctamente con una alta probabilidad, contra una gran base de datos de puntos de muchas imágenes. Se puede dividir la estructura de SIFT en cuatro módulos:

  - ❖ Detección de máximos y mínimos escala-espacio: El primer paso es la búsqueda de puntos en la imagen que puedan ser puntos claves. Se implementa eficientemente al emplear la función de diferencias gaussianas para identificar los puntos de interés que son invariantes a escala y orientación.
  - ❖ La localización de los puntos claves: En cada localización obtenida en el apartado anterior se desarrolla un modelo cuadrático detallado para determinar la localización y la escala. Los puntos claves se seleccionan basándose en su estabilidad.
  - ❖ Asignación de orientación: Una o más orientaciones se asignan a cada localización de los puntos claves, basándose en la dirección del gradiente de la imagen. Todas las futuras operaciones se ejecutan con los datos de la imagen que ha sido transformada de acuerdo con la orientación, escala y localización

asignada para cada característica, de este modo se proporciona invariancia a estas transformaciones.

- ❖ Descriptores de los puntos claves: Los gradientes locales de la imagen son medidos en la escala seleccionada en la región alrededor de cada punto clave. A partir de esto se crea un histograma por cada subregión.

- SURF (Speeded Up Robust Features): Este algoritmo guarda cierta similitud con la filosofía del algoritmo SIFT, pero presenta notables diferencias. Las dos grandes mejoras, según los autores, son:
  - ❖ Velocidad de cálculo considerablemente superior sin ocasionar perdida del rendimiento.
  - ❖ Mayor robustez ante posibles transformaciones de la imagen.

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

  Las diferencias más características con el algoritmo SIFT son:
  - ❖ La normalización o longitud de los vectores de características de los puntos de interés es considerablemente menor, concretamente se trata de vectores de dimensionalidad 64, mientras que los de el algoritmo SIFT son de 128.
  - ❖ El descriptor SURF utiliza siempre la misma imagen (original)
  - ❖ Utiliza el determinante de la matriz Hessiana para calcula tanto la posición como la escala de los puntos de interés.
  - ❖ SIFT guarda la posición, la escala y la orientación, puesto que es posible que en una misma posición encontremos varios puntos de interés., mientras que SURF, en un mismo punto solamente aparece un único punto de interés, por lo que son guarda la escala y la orientación.

- RANSAC (RANdom SAmple Consensus): es un algoritmo de estimación robusta que permite hallar un modelo

matemático a partir de datos contaminados con numerosos valores que no se ajustan al modelo. Presenta gran capacidad para proporcionar un buen ajuste a partir de datos contaminados.

Busca el mejor modelo considerando todos los pixeles de contorno incluidos aquellos que no se ajunta al modelo buscado. Para ello, selecciona aleatoriamente muestras de s pixeles, siendo s los puntos necesarios para establecer los parámetros del modelo: (2 para una línea, 3 para una circunferencia, 5 para una elipse, …). Una vez calculados los parámetros para cada muestra, se evalúa el conjunto de consenso que es el número de pixeles de la imagen de contornos original próximos al modelo calculado dentro de una tolerancia preestablecida. Si el nuevo resultado es mejor, entonces se reemplaza el resultado almacenado por el nuevo.

2.1.2 Detector de esquinas

Una esquina puede definirse como la intersección de dos líneas que forman un ángulo. También puede definirse como la arista o el ángulo que resulta del encuentro de dos superficies, como puede ser las paredes.

Los puntos de interés, en este tipo de algoritmos son puntos en la imagen que tiene una posición bien definida. Estos pueden ser una esquina, pero también puntos aislados de intensidad local máxima o mínima como pueden ser el final de una línea o un punto de una curva que es locamente máximo.

Este tipo de algoritmos no son muy robustos y en muchos casos requiere de una comprobación posterior o la implementación de métodos de repetitividad para reducir el número de errores en la tarea de reconocimiento.

Una forma de determinar la calidad de este tipo de detectores es su capacidad para descubrir la misma esquina en múltiples imágenes similares, pero en condiciones diferentes de iluminación, rotación, traslación, etc.

Algunos de los algoritmos que implementa este tipo de detectores son:

- Harris Corner Detection

- Shi-Tomasi Corner Detector
- FAST (Features form Accelerate Segment Test)

2.1.3 Detectores de bordes

Este tipo de detectores trata de localizar puntos en una imagen en la que esta cambia drásticamente. Estos cambios suelen ser cambios bruscos en el brillo de la imagen que pueden deberse a cambios en el escenario, movimientos de objetos, giros de cámaras, etc.

Los resultados de aplicar un detector de borde a una imagen pueden conducir a un conjunto de curvas conectas que indican las fronteras de los objetos, las fronteras de las marcas en las superficies de estos objetos y curvas que corresponden a discontinuidades en la orientación de las superficies. Sin embargo, no siempre se obtiene bordes ideales a partir de imágenes reales. Un algoritmo que implementa un detector de bordes es el denominando Canny Edge Detector.

2.2 Obtención de la profundidad

A partir de la obtención de las características de las dos imágenes se utilizan algoritmos de emparejamiento para encontrar los puntos obtenidos de cada una de las imágenes que conforman en par con la otra imagen de dicho par.

2.2.1 Imagen de Disparidad

Cuando se han localizados los diferentes puntos característicos en las dos imágenes, lo siguiente es calcular la geometría de las cámaras para obtener la rotación y la traslación. Estos cálculos están basados en el concepto de geometría epipolar. Desarrollando estos cálculos se genera lo que se denomina imagen de disparidad que representa la diferencia de pixeles entre el mismo objeto en una imagen y la otra.

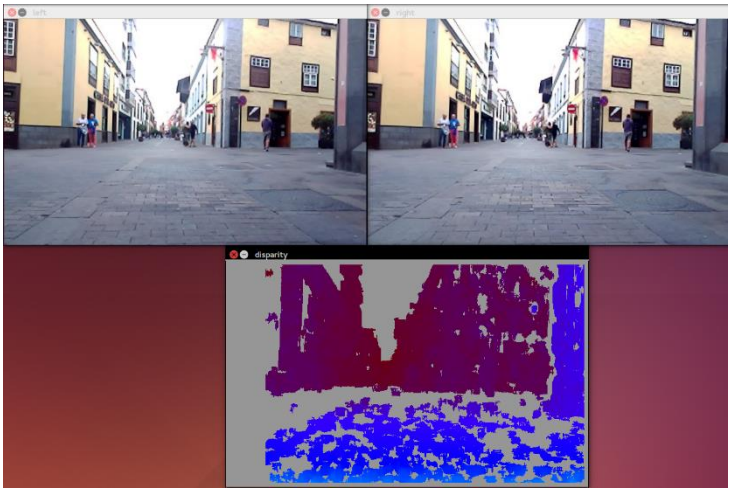

Figura 2.1. Imagen de Disparidad.

2.2.2 Nube de puntos

A partir de las imágenes rectificadas, obtenidas de las cámaras, y de la imagen de disparidad se desarrolla lo que se denomina nube de puntos.

La nube de puntos es una representación 3D del entorno en el que navega el robot, es decir, nos proporciona una visión de la superficie y de la profundidad del entorno.

Figura 2.2. Nube de Puntos.

2.3 Estimación de movimiento

A partir de los conceptos anteriores se ha explicado cómo obtener un modelo 3D de un par de imágenes. Para obtener una estimación de movimiento el proceso es muy igual. Aparte de buscar los puntos característicos del par de imágenes actual, buscas también puntos característicos del par de imágenes anterior.

Mirando la diferencia entre puntos característicos del par de imágenes anteriores con el par de imágenes actual se puede realizar una estimación de la distancia que ha recorrido el robot.

El problema en este tipo de odometría puede surgir cuando se va navegando por entornos homogéneos debido que, al tener los mismos puntos característicos en la estimación de movimiento, se observa que el robot "piensa" que no se ha movido de su sitio.

Capítulo 3. Desarrollo

3.1 Introducción al proyecto

En el siguiente apartado se expondrá la investigación sobre los diferentes algoritmos de visión estéreo que se encuentra dentro de ROS. Por ello empezaremos con una explicación sobre las diferentes implementaciones de odometría visual y en que algoritmos teóricos están basados. Posteriormente se mostrarán una serie de pruebas en diferentes entornos y se realizara una comparación entre las diferentes implementaciones para finalizar cada prueba con la comparación con la odometría mecánica y laser. Para comprobar la efectividad de la odometría se observará todo el recorrido, pero se pondrá énfasis en los puntos finales.

Todas las bolsas de datos tratados en la investigación han sido capturadas utilizando la cámara ps4, una cámara estereoscópica de tamaño 186mm x 27mm x 27mm que tiene una velocidad de captura de 120 fps con una resolución de 640 x 400 píxeles. También en estas bolas se ha capturado la odometría implementada en la silla perenquén. Esta odometría está conformada por:

- odometría mecánica que utiliza encoders colocados en la parte inferior de la silla de ruedas que consta de dos receptores desfasados 90°. Con este desfase y utilizando dos bits se puede saber hacia dónde se ha movido. Para saber el recorrido que ha realizado utiliza π*d*nc siendo d el diámetro de la rueda y nc el número de cuentas. Se denomina cuenta a examinar los impulsos se los encoders y ver observar hacia donde se ha movido. Con esto se consigue una constante π/cuentas y sabemos lo que se ha recorrido. Para calculo el recorrido de ambas ruedas se utiliza esta sencilla formula:

$$Desplazamiento = \frac{avance\ derecho + avance\ izquiedo}{2}$$

  Para saber que ángulo se ha movido usamos esta otra fórmula:

$$Ángulo = \frac{avance\ izquierdo + avance\ dercjpo}{R(separacion\ entre\ ruedas)}$$

El gran problema de esta odometría son las variaciones de peso que puede causar cambios en los radios de las ruedas, el tamaño, los deslizamientos, … que hacen que la estimación no sea acercada a la realidad. El tópico que usa en la silla es la que viene como controlador y se denomina perenquén.

- odometría laser que utiliza un escáner laser 2D a cada lado de la silla. Esto realiza un barrido que busca puntos singulares y que puedan servirle de ayuda para posicionarse. Luego realizamos otro barrido y observamos los puntos singulares y los comparamos con el anterior barrido. El problema que puede surgir en esta odometría es cuando circula por entornos muy homogéneos como pasillos en los que no tiene puntos singulares en los que fijarse. El tópico que usa la silla es la que se denomina como Laser_scan_matched.

La combinación de estas odometrías se realiza mediante la covarianza de la odometría laser (calidad de la medida. Si es cercana a 0 es una medida fiable, si es cercana a uno no es nada fiable). Se llama Filtrado de extendido de calma a la mescla de la odometría mecánica y odometría laser usando la covarianza de la odometría laser para saber que odometría es más fiable en cada momento y utilizarla para estimar su movimiento.

Debido a que los resultados de una prueba también dependen de las características del ordenador es necesario exponerlas para hacernos una idea de su comportamiento con ordenadores similares:

Portátil MSI:

- Procesador Intel I7-4720HQ
- Nvidia GeForce GTX 950M
- 8 Gb DRAM

3.2 Librerías para la odometría visual

Existen librerías que implementan la odometría visual. Algunas de ellas eran independientes, pero han pasado a ser parte de ROS. Estoy nos permite realizar un estudio de ello para ver cuál de ellos nos viene mejor a la hora de realizar un proyecto que necesite odometría visual.

3.2.1 Stereo odometry

Este nodo está incluido dentro de la librería RTAB-MAP y usando las imágenes estéreo, calcula la odometría usando las características visuales extraídas de la imagen izquierda con su información de la profundidad calculada mediante la búsqueda de las mismas características en la imagen derecha. Usando las correspondientes características, un enfoque basados en el algoritmo RANSAC calcula la transformación entre las consecutivas imágenes izquierdas.

3.2.2 VISO2

Es un algoritmo desarrollado en C++, multiplataforma muy rápida. La versión estéreo está basada en minimizar los errores de reproyección de las coincidencias características dispersas y es bastante general (sin modelos de movimientos o restricciones de configuración, excepto las imágenes que tienen que estar rectificadas y los parámetros de calibración son conocidos).

La entrada de este algoritmo de odometría son las características capturadas entre 4 imágenes, es decir, dos imágenes consecutivas de cada lado. Con el fin de encontrar la localización de características estables, primero se realizar un filtrado de las imágenes entrantes con una región de 5x5 y unas mascara de esquina. Posteriormente se aplicamos una no máxima supresión y una no mínima supresión en las imágenes filtradas (Es un tipo de filtrado en los que solo te quedas con los máximos del gradiente, se usan pocos pixeles para eliminar ruido), dando como resultados características candidata en una de las cuatro clases (cuadrado máximo, cuadrado mínimo, esquina máxima y esquina mínima). Para reducir el esfuerzo computacional solo se compara las características dentro de esas clases. En contrastes con otros métodos que reconstruyen a partir de una colección de imágenes desordenadas, este método asume

que una suave trayectoria de la cámara, reemplazando computacionalmente una intensa rotación de la misma y la escala de descriptores invariantes como SURF.

Dado dos puntos característicos, simplemente se compara en una ventada de bloque de 11x11 de la respuesta de filtro de Sobel usando cada uno una suma de diferencias absolutas (SAD).

3.2.3 FOVIS

Este algoritmo de odometría visual ha sido desarrollado basado en un standard de odometría visual estéreo en tubería. Este algoritmo se basa en desarrollar los siguientes pasos:

1. Preprocesamiento de imagen: Se recibe un par de imágenes de cámara estéreo y el componente RGB de las imágenes es convertido a escala de grises y suavizada. Se construye una pirámide Gaussiana para permitir más robustez en la detección de características a diferentes escalas. Cada nivel de la pirámide corresponde a un octavo en el espacio de la escala. Las características en las escalas más grande generalmente corresponden a estructuras de imagen largas en la escena, que generalmente hacen más repeticiones y más robustez al desenfoque del movimiento.
2. Extracción de características: Las características son extraídas en cada nivel de la pirámide gaussiana usando el detector FAST. La profundidad correspondiente a cada característica es también extraída de la profundidad de la imagen. Las características que no asociadas a una profundidad son descartadas. Para mantener una distribución más uniforme de características, cada nivel de la pirámide es discretizando en 80 x 80 cubos de pixeles y en cada cubo, las 25 características con mayor puntuación se mantienen.
3. Estimación de rotación inicial: Se usa una técnica propuesta por Mei para calcular la rotación inicial por la minimización directa de la suma de los errores de los pixeles cuadrado entre las versiones submuestreadas de la versión actual y la versión anterior.

4. Comparación de características: A cada característica se le asigna 80 bytes que consiste en valores de brillo de 9 x9 pixeles alrededor de la característica, normalizando a 0 y omitiendo el pixel inferior izquierdo. Las características son comparadas a través de marcos para comparar sus descriptores de características usando chequeo de mutua consistencia. El resultado de la comparación entre dos características son la suma absoluta de diferencias (SAD)de sus descriptores característicos. Una comparación de características es declarada cuando 2 características tienen baja puntuación de SAD entre si. Una vez se encuentra una coincidencia, la localización de la característica en la nueva imagen es distinguida obteniendo una comparación sub-pixel.
5. Deteccion inlier: A pesar de las limitaciones impuestas por la estimación de la rotación inicial sustancialmente reducida por la tasa de comparaciones de características incorrectas entre imágenes, es necesario quitar las malas comparaciones.
6. Estimación de movimiento: se calcula a partir de las características coincidentes en tres pasos.

   En el primer paso, el método de orientación absoluta de Horn provee una estimación inicial para minimizar las distancias euclídeas entre las características captadas.

   Seguidamente, la estimación de movimiento es refinada para minimizar el error de reproyección de características usando un solucionador de mínimos cuadrados no lineal.

   Por último, la comparación de características que excedan un límite de error de reproyección son descartadas.

3.3 Pruebas realizadas

3.3.1 Primera prueba (pasillo Facultad Física y Matemáticas)

- Rtabmap Stereo odometry

En las primeras pruebas con este algoritmo vemos que, para las características de este ordenador, es incapaz de realizar las transformadas y situarse en el espacio en tiempo real. Probando la capacidad del algoritmo multiplicamos la velocidad de frame por 0.8, es decir, reducimos la velocidad en un 20%. Con esto conseguimos que el algoritmo tenga más tiempo para desglosar las imágenes e intentar situarse. A partir de esta prueba conseguimos que realiza parte del recorrido, pero al final pasa lo mismo que ejecutándolo en tiempo real.

Reduciéndole la velocidad en un 40% podemos observar que se efectúa todo el recorrido sin que haya problemas de computo. El recorrido de la siguiente imagen hay que destacar que se realiza en el mismo pasillo (ubicado en la planta baja de la Facultad de Física y Matemáticas) en el cual el punto de inicio que será el (0,0,0) y el punto final estará en el siguiente rango (-2-0,-1-0,0). Sabiendo esto podemos observar que el recorrido, según este algoritmo, no concuerda con el que se ha realizado quedando a 1,2 metros en el eje x y a 0.3 metros en el eje y del punto inicial al que empezó. Observando esto podemos deducir que el algoritmo ha fallado por varios metros en su estimación de la posición a la cual ha llegado.

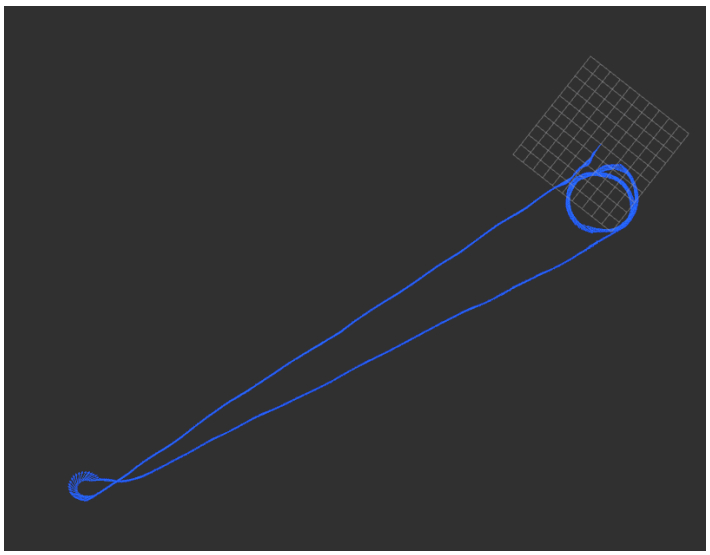

Figura 3.1. Recorrido primera prueba Stereo Odometry Posición Final *(1,245, 0,338, 0,000)*.

Este algoritmo no es capaz de resolver cambios en el tiempo, es decir, no es capaz de soportar saltos en el

tiempo como los que puede producir la reproducción en bucle de una bolsa de datos.

- VISO2

Como podemos observar en la primera prueba con este algoritmo se ve una gran diferencia de rendimiento ya que al reproducir la bolsa a velocidad normal es capaz de ejecutar el algoritmo sin problemas de rendimiento. También observamos una mejora perceptible en comparación con el algoritmo anterior. Esto lo podemos observar tanto en el recorrido como en la posición final, la cual por ejemplo la coordenada y entra dentro del rango anteriormente mencionado.

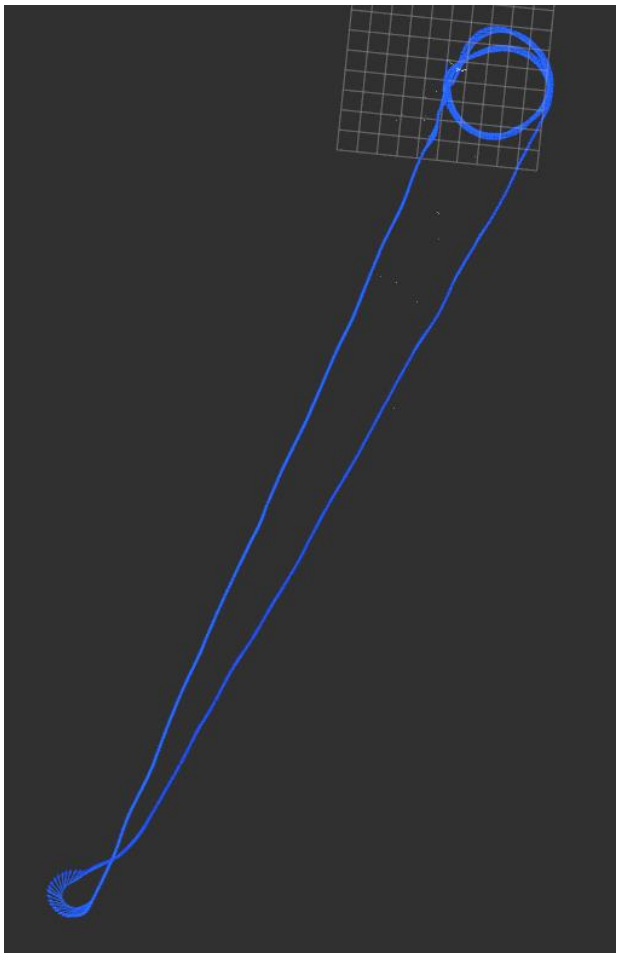

Figura 3.2. Recorrido primera prueba VISO2 Posición Final *(0,432, -0,874, 0,000)*.

Incrementando el parámetro max_features (el cual permite definir el número máximo de características de los bloques) de 2, que es el valor por defecto, a 5 y desactivando el parámetro multi_stage (hace comparaciones en múltiples etapas, lo cual permite que sea más denso y rápido la ejecución del algoritmo) se ha conseguido una

leve mejora en el punto final como se puede observar en la siguiente imagen.

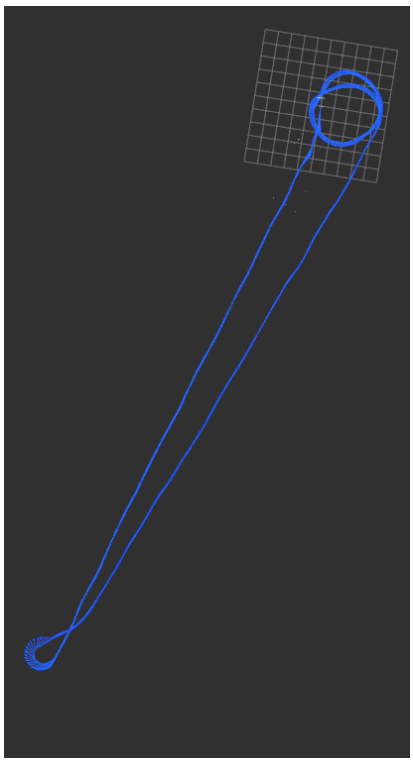

Figura 3.3. Recorrido primera prueba VISO2 primera mejora Posición Final *(-0,568, -0,162, 0,000)*.

Una segunda prueba se llevó a cabo cambiando más notablemente en los parámetros, pero siguiendo la línea de la primera incrementación. Por ello, aparte de incrementar la cantidad de características máximas de 5 a 10, se disminuyó la distancia máxima entre pixeles para la no máxima supresión. También se disminuyó de 5 a 2 la tolerancia a la disparidad para eliminar características que no están cercanos al resto de las observaciones (outliers).

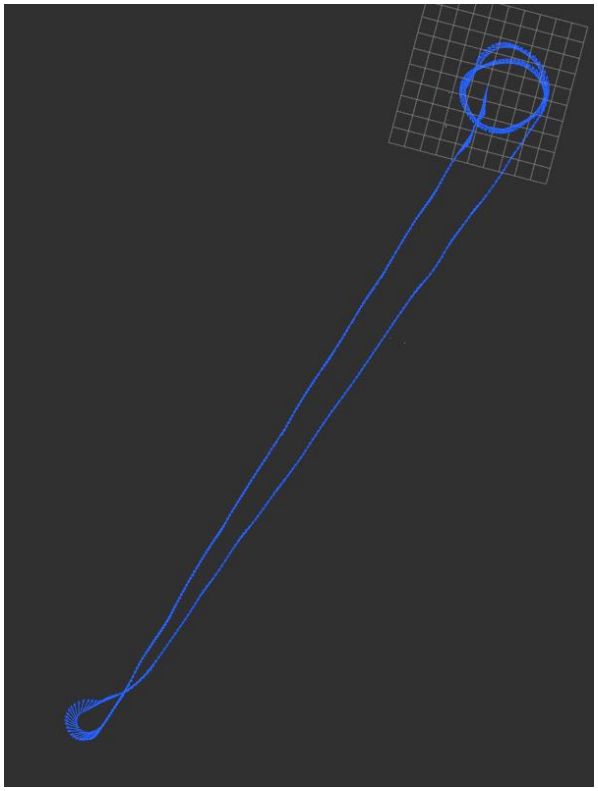

Figura 3.4. Recorrido primera prueba VISO2 segunda mejora Posición Final *(-0,463, -0,981, 0,000)*.

A posteriori se realizaron más cambios, pero ninguno dio un resultado que significara una mejora a las que ya hemos comentado

- FOVIS

Se produjo un gran error con la odometría FOVIS debido a que la resolución de las imágenes con las que trabaja tiene que ser de 640x480 y las imágenes que capturada por la cámara ps4 son de 640x400.

Se intentó introducir cambios en el código fuente de dicha odometría para conseguir que la imagen capturada por la cámara se copiara a una imagen de mayor tamaño para que el algoritmo pudiera trabajar con ella. Esta es una representación de la modificación en las imágenes.

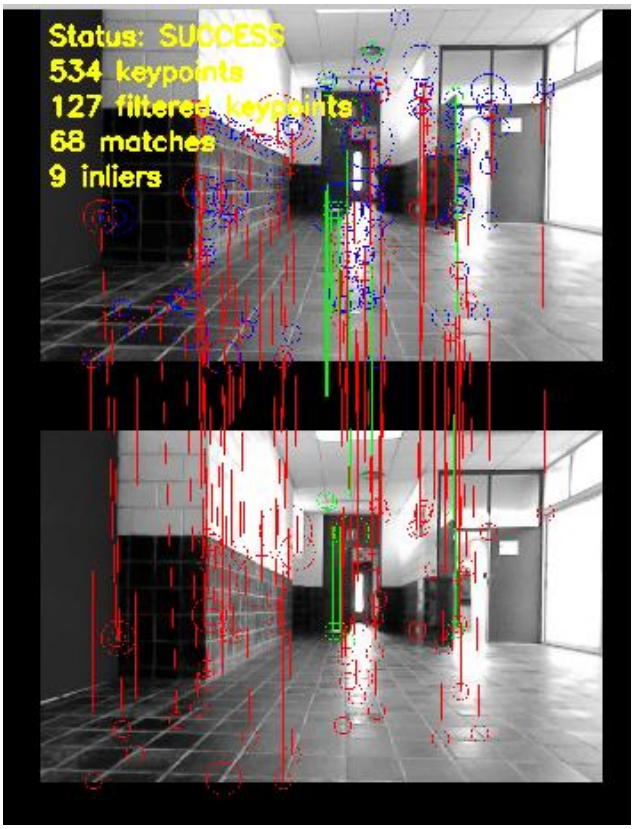

Figura 3.5. FOVIS Problema imágenes.

A parte de la modificación del código fuente se cambiaron los valores de configuración por defecto, pero ninguno de los cambios fue exitoso para obtener una odometría fiable. Por ello en las siguientes pruebas no expondrá ningún caso con esta odometría. Cabe decir que para bolsas de datos donde las imágenes son del tamaño adecuado, este algoritmo funciona razonablemente bien.

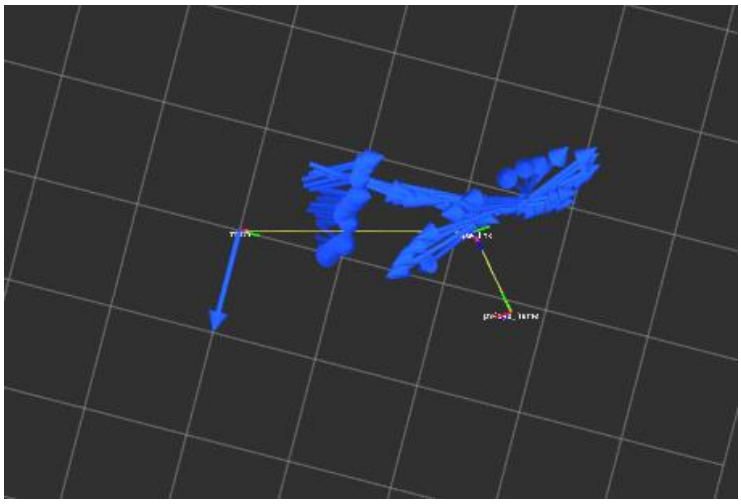

Figura 3.6. FOVIS problema Recorrido.

- Odometría visual estereoscópica vs odometría mecánica

Para mostrar la efectividad de los algoritmos de las odometrías visuales estereoscópicas la compararemos con la odometría mecánica que incorpora la silla perenquén obtenemos la siguiente imagen.

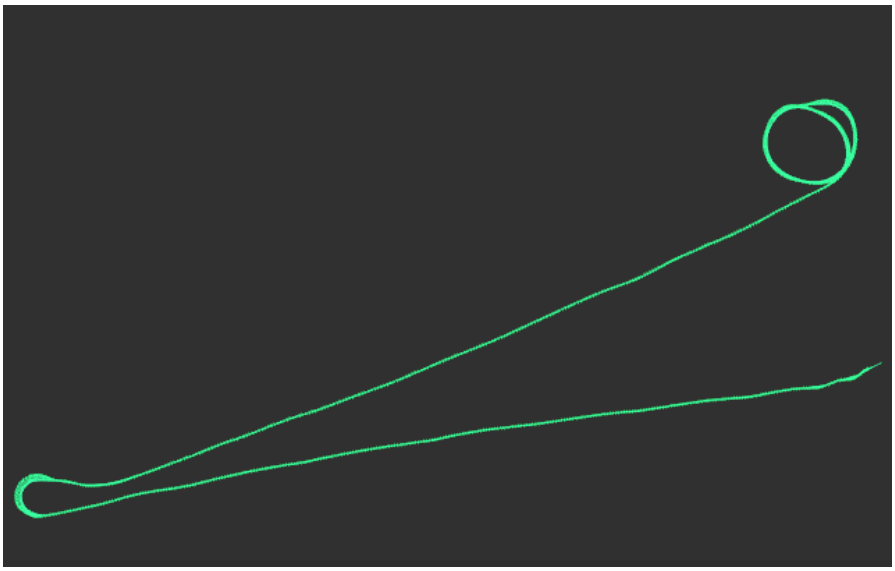

Figura 3.7. Recorrido primera prueba Odometría mecánica segunda mejora Posición Final (-45,665, 11,602, 0,000) Posición Inicial (-31,298, 17,376, 0,000).

Se puede visualizar que en la primera vuelta la silla no da la vuelta completa por lo que va alejándose cada vez más del punto inicial en vez de intentar en un rango como lo hacen los algoritmos de visión. Esto conlleva que el fallo de la posición final no es de uno o dos metros, sino que se ven una gran diferencia ya que en el eje *x* conlleva un error de más de 10 metros y en el eje *y* se ve un error de alrededor de 6 metros.

3.3.2 Segunda Prueba (Calle Herradores)

- Rtabmap stereo odometry

Como se puede observar en la siguiente imagen este es el resultado de uno de los recorridos grabados en la calle Herradores. Se puede observar que se desarrolla en línea recta y que la odometría visual de la librería Rtabmap realiza un recorrido en una misma línea. La posición final

(285,032, 87,441, 0,000) nos da una estimación que se acerca bastante a la realidad.

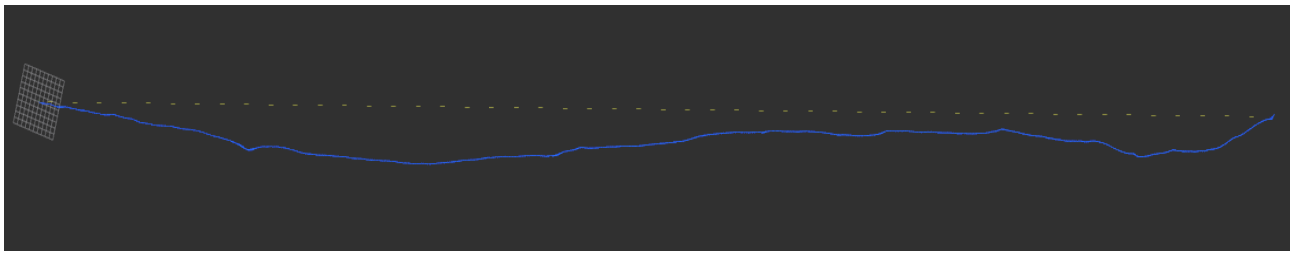

Figura 3.8. Recorrido segunda prueba stereo odometry.

Esta vez el rendimiento del algoritmo fue bastante bueno ya que se pudo realizar el procesamiento de imágenes en tiempo real. Esto puede venir dado a que hay características muy distinguibles en las imágenes por lo que no es necesario un procesamiento tan pesado como el anterior, el cual era un pasillo y el número de características diferente es limitado porque es un recorrido monótono con un conjunto de características limitado.

- VISO2

Este algoritmo se ha ejecutado utilizando las mejoras aplicadas para la anterior prueba y hemos conseguido unos buenos resultados como se puede ver en la siguiente imagen. En recorrido lo ha desarrollado bastante más suave que el algoritmo anterior. Lo que se puede observar es que el final es un poco raro seguramente a los objetos en movimiento se detectaron al final (243,996, 26,208, 0,000) de la bolsa de datos.

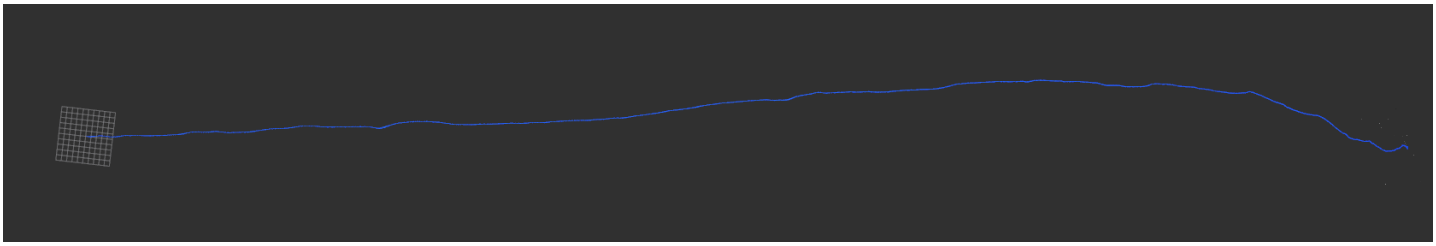

Figura 3.9. Recorrido segunda prueba VISO2.

A pesar de que los dos algoritmos se desarrollan de una manera bastante similar y los dos comienzan en el punto

0,0,0 la posición final de cada uno es bastante diferente y se produce un desfase de en unos 40 metros en el eje x y unos 60 metros en el eje y.

- Odometría combinada

En la siguiente imagen se puede visualizar la odometría combinada. Esta odometría es la unión de la odometría laser con la odometría mecánica que esta implementada en la silla. Debido a diversos problemas esta odometría realizar múltiples que muestra que se ha perdido y vuelto a localizar en el entorno. Por ello, y en este caso, no nos sirve de comparación para verificar la eficacia de la odometría visual estéreo.

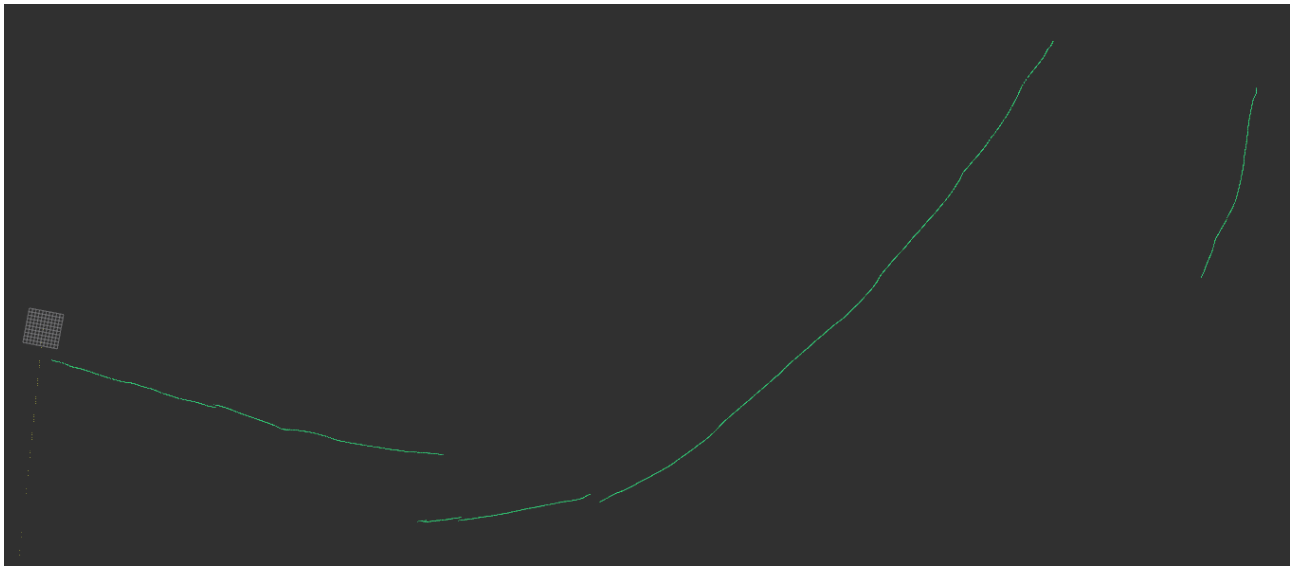

Figura 3.10. Recorrido segunda prueba odometría combinada.

3.3.3 Tercera prueba (Avenida Trinidad)

Como podemos visualizar en las dos imágenes siguientes tanto la odometría de la librería Rtabmap como la odometría viso 2 tiene un recorrido muy similar, pero, al igual que en el caso anterior, vemos una gran diferencia en la estimación porque supera los 100 metros de diferencia entre una odometría y la otra.

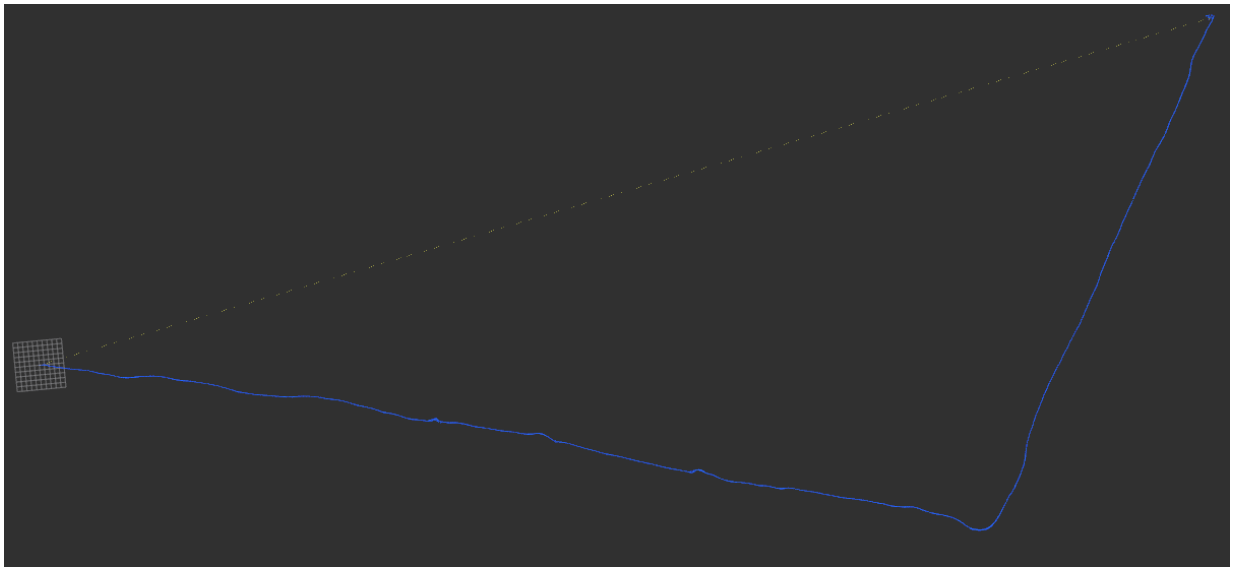

Figura 3.11. Odometria Rtabmap stereo odometry tercera prueba Punto final (245,572, 47,640, 0,000).

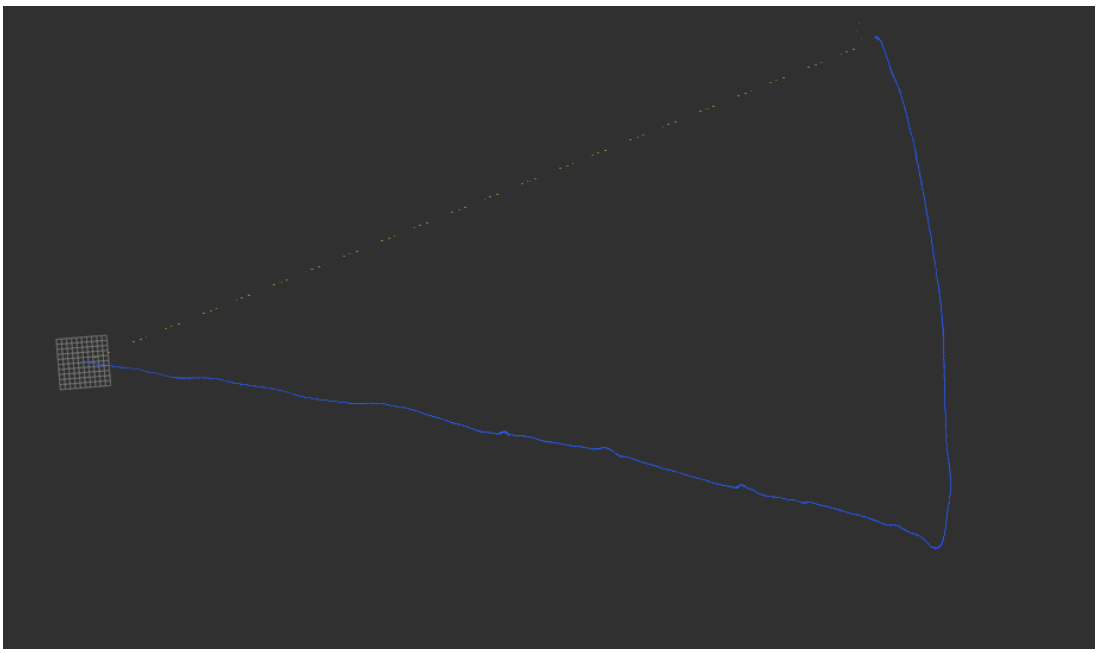

Figura 3.12. Recorrido segunda prueba Odometría VISO2 Punto final (162,691, 50,644, 0,000).

- Odometria Combinada

Si observamos la odometria conbinada de la laser con la mecanica nos da como punto final (190.145,200.400,0). Por lo que podemos observar se parece mas a la odometria VISO2 que a la stereo pero aun asi sigue habiendo grandes diferencias entre ellas.

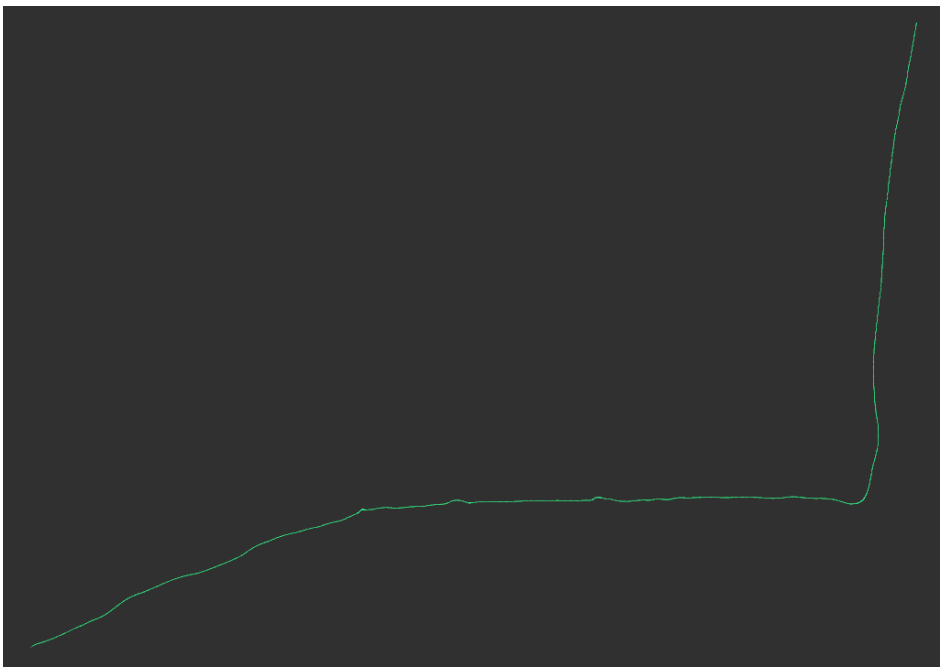

Figura 3.13.Tercera prueba odometría combinada.

La diferencia es de alrededor de unos 30 metros si la comparamos con la odometria VISO2. Si la comparamos con la odometria de Rtabmap son mas de 60 metros.

3.3.4 Cuarta Prueba (Pasillo Fisica y Matematicas)

- Rtabmap odometria visual estereo

En esta prueba hemos vuelto al pasillo de Fisica porque hemos obtenido una buena odometria combinada para poder realizar una comparacion con cada uno de las odometrias visuales estereo.

En esta prueba podemos observar lo que se ha comentado de que en lugares con caracteristicas muy similares la vision es incapaz de localizarse en el entorno.En la imagen podemos visualizar una buena union de laser con mecanica que sera la linea verde . En la linea azul observamos el algoritmo de vision estereo de rtabmap que es incapaz de se realizar todo el recorrido y tambien es incapaz de recuperarse despues de haberse perdido. A pesar que para exteriores tiene un rendimiento aceptable debido a la cantidad de caracteristicas que son distinguibles unas de otras, para interiores, sobre todos entornos en los que las caractericticas no varien, es incapaz de localizarse o de recuperase despues de haberse caido.

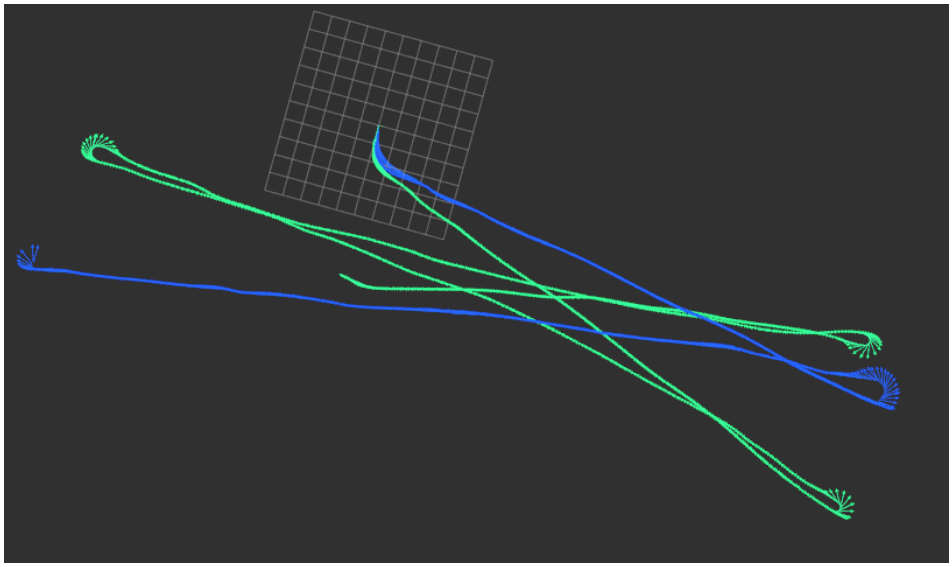

Figura 3.14. Recorrido cuarta prueba stereo odometry y odometría combinada.

- VISO2

Con esta otra odometria podemos observar que, a pesar de que se perdio en el mismo sitio que el anterior, fue capaz de recuperarse e intentar volver a estimar su posicion.En el momento que se pierde y se recupera se fue distanciando de la estimacion que llevaba acorde a la otra odometria

Si observamos los puntos finales (VISO2 (6,558, 6,842, 0,000) ,odometria combinada (8,461, 1,105, 0,000)) vemos que las diferentes odometrias no difieren sino en varios metros con respecto al resto de las pruebas.

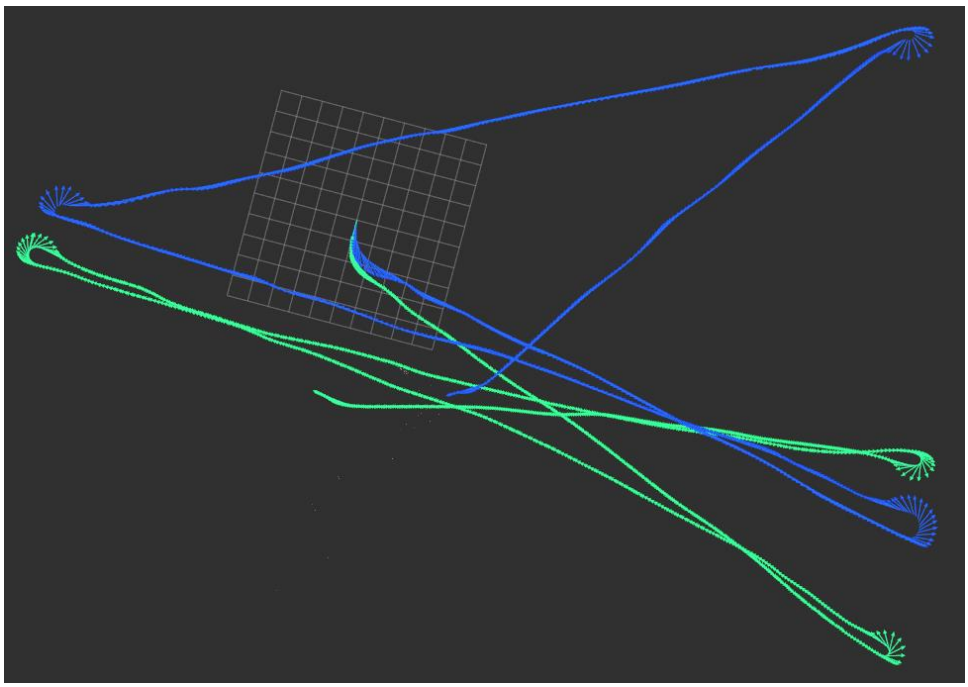

Figura 3.15. Recorrido cuarta prueba VISO2 y odometría combinada.

Capítulo 4. Conclusiones y líneas futuras

En este trabajo se ha hecho un estudio sobre la odometría visual estereoscópica y se ha desarrollado pruebas para la comparación con otro tipo de odometría que en este caso ha sido la odometría mecánica unida a la odometría laser.

Como se ha podido observar en todas las pruebas la odometría visual estereoscópica VISO2 es la que mejor se adapta y se combina con las odometrías. La odometría visual Rtabmap, a pesar de que las pruebas no fueron demasiado malas, su rendimiento y su eficacia queda por debajo de la odometría VISO2.

Ha sido problemático no poder contar con el algoritmo Fovis para poder hacer un estudio completo de las odometrías existentes. En líneas futuras se podría modificar todo el algoritmo para que se pueda adaptar a este tipo de situaciones como lo hacen las otras dos odometrías que han sido objeto de estudio.

A partir de aquí se puede seguir estudiando el mapeado, localización y búsqueda de obstáculos.

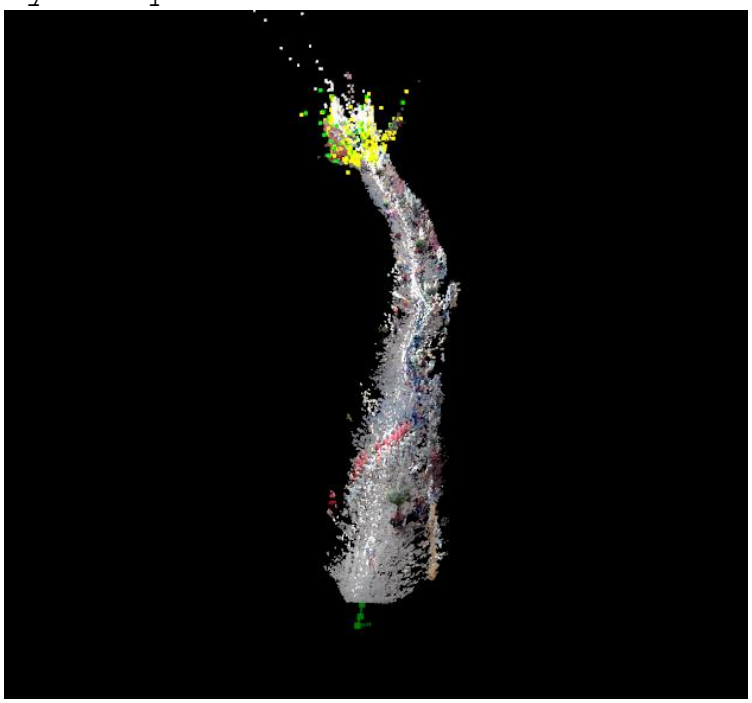

Figura 4.1. Mapeado.

Capítulo 5.
Summary and Conclusions

In this work has made a study of stereoscopic visual odometry and has developed tests for comparison with other odometry which in this case has been the mechanical odometry attached to the laser odometry.

As has been observed in all tests stereoscopic visual odometry VISO2 is best adapted and merges with the other odometrías. The visual odometry Rtabmap, although tests were not too bad, its performance and its effectiveness is below the VISO2 odometry.

It has been problematic not to have the algorithm Fovis to make a thorough study of existing odometrías. In future lines you could change the entire algorithm so that it can adapt to this kind of situation as do the other two odometrías that have been studied.

From here you can continue to explore the mapping, location and search for obstacles.

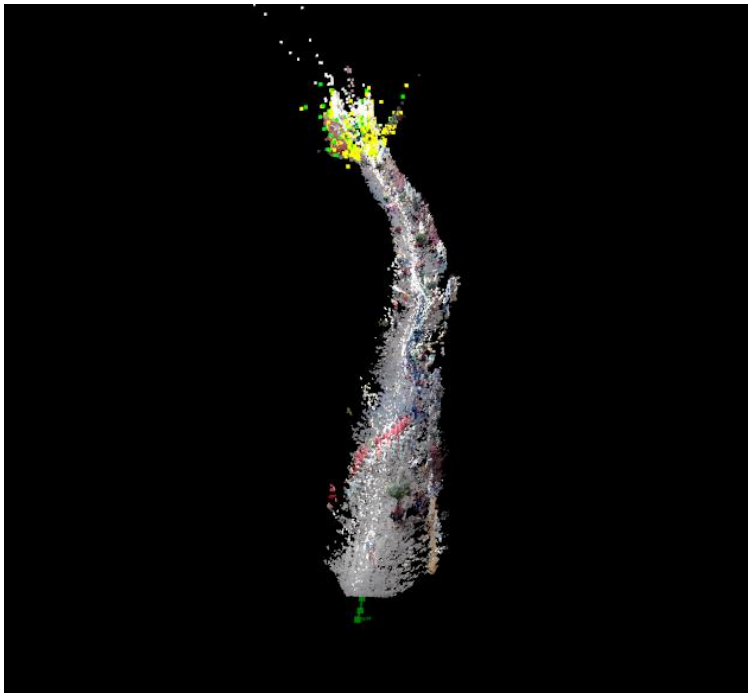

Figura 5.1. Mapping.

Capítulo 6. Presupuesto

En cuestión de software el presupuesto ha sido de 0 debido a que todos los programas, documentación y demás elementos usados en este proyecto han sido software libre.

El coste asociado a este proyecto viene dado por los componentes hardware que se muestran en la siguiente tabla:

Descripción	Coste
PS4 EYE	54,90 €
Adaptador USB	3,00 €
Presupuesto:	**57,90 €**

Tabla 6.1. Presupuesto.

Apéndice A.
Título del Apéndice 1

A.1. Silla.launch

```
  /*****************************************************************************
  *
  * Fichero .h
  *
  *****************************************************************************
  *
  * Fabian Díaz Lorenzo
  *
  * 22/06/2016
  *
  * Este fichero contiene todos los parámetros y la configuración de los distintos
  * algoritmos utilizados en las pruebas, aparte de toda la configuración básica para
  * que todo el sistema ROS funcione de manera correcta
  *****************************************************************************/

  <launch>

  <!-- Choose visualization -->
  <arg name="rtabmapviz"              default="true" />
  <arg name="rviz"                    default="false" />

  <param name="use_sim_time" type="bool" value="True"/>

  <!-- Corresponding config files -->
  <arg    name="rtabmapviz_cfg"                                    default="-d    $(find
Script_local)/launch/config/rgbd_gui.ini" />
  <arg   name="rviz_cfg"                                            default="-d   $(find
Script_local)/launch/config/rgbd.rviz" />
  <arg name="frame_id"                default="/base_link"/>
  <arg name="time_threshold"          default="0"/>
  <arg name="optimize_from_last_node" default="false"/>
  <arg name="visual_odometry1"         default="true"/>
  <arg name="visual_odometry2"         default="false"/>
```

```
   <arg name="visual_odometry3"         default="false"/>
   <arg           if="$(arg           visual_odometry1)"           name="database_path"
default="~/.ros/silla1.db"/>
   <arg           if="$(arg           visual_odometry2)"           name="database_path"
default="~/.ros/silla2.db"/>
   <arg           if="$(arg           visual_odometry3)"           name="database_path"
default="~/.ros/silla3.db"/>
   <arg name="rtabmap_args"            default=""/>
   <arg name="stereo_namespace"        default="/stereo"/>
   <arg     name="left_image_topic"                                          default="$(arg
stereo_namespace)/left/image_rect_color" />
   <arg     name="right_image_topic"                                         default="$(arg
stereo_namespace)/right/image_rect_color" />
   <arg name="left_imageraw_topic"      default="$(arg stereo_namespace)/left/image_raw"
/>
   <arg name="right_imageraw_topic"    default="$(arg stereo_namespace)/right/image_raw"
/>


   <arg name="left_camera_info_topic"  default="$(arg stereo_namespace)/left/camera_info"
/>
   <arg                  name="right_camera_info_topic"                   default="$(arg
stereo_namespace)/right/camera_info" />
   <arg name="approximate_sync"        default="true"/>
   <arg name="compressed"              default="true"/>
   <arg name="localizacion"            default="false"/>
   <arg name="subscribe_scan"          default="false"/>
   <arg name="scan_topic"              default="/right_laser"/>
   <arg name="odom_topic"              default="/perenquen/odom"/>
   <arg name="namespace"               default="rtabmap"/>
   <arg name="wait_for_transform"      default="0.1"/>
   <!-- Odometry parameters: -->
   <arg name="strategy"           default="0" <arg name="feature"            default="6"
   <arg name="estimation"          default="1" />
   <arg name="nn"                 default="3" />
   <arg name="max_depth"           default="0" />
   <arg name="min_inliers"         default="20" />
   <arg name="inlier_distance"     default="0.1" />
   <arg name="local_map"           default="1000" />
   <arg name="odom_info_data"      default="true" />
   <arg name="variance_inliers"    default="true"/>


   <!-- Nodes -->
   <!-- TF -->
```

```
  <node pkg="tf" type="static_transform_publisher" name="transformada" args="0.0 0.0 0.85 -1.5707 0 -1.5707 base_link ps4eye_frame 100" />

  <node pkg="tf" type="static_transform_publisher" name="identidad" args="0.0 0.0 0 0 0 0 map odom 100" />

  <!--node pkg="tf" type="static_transform_publisher" name="camera_transform" args="0.84 0 0.05 -1.57 0.0 -1.70 base_link /ps4eye_frame 100" / -->

  <!-- Descomprimir las imagenes de la bolsa de datos -->

  <node if="$(arg compressed)" name="republish_left" type="republish" pkg="image_transport" args="compressed in:=$(arg left_imageraw_topic) raw out:=$(arg left_imageraw_topic)"/>

  <node if="$(arg compressed)" name="republish_right" type="republish" pkg="image_transport" args="compressed in:=$(arg right_imageraw_topic) raw out:=$(arg right_imageraw_topic)"/>


  <!-- node if="$(arg compressed)" name="republish_left" type="republish" pkg="image_transport" args="compressed in:=$(arg left_image_topic) raw out:=$(arg left_image_topic)" / -->

  <!-- node if="$(arg compressed)" name="republish_right" type="republish" pkg="image_transport" args="compressed in:=$(arg right_image_topic) raw out:=$(arg right_image_topic)" / -->


  <!-- Stereo_image_proc -->

  <node pkg="stereo_image_proc" type="stereo_image_proc" name="stereo_image_proc" args="__ns:=stereo" />

  <!-- Odometry -->

  <group ns="$(arg namespace)">

  <node if="$(arg visual_odometry1)" pkg="rtabmap_ros" type="stereo_odometry" name="stereo_odometry" output="screen">

  <remap from="left/image_rect"        to="$(arg left_image_topic)"/>

  <remap from="right/image_rect"       to="$(arg right_image_topic)"/>

  <remap from="left/camera_info"       to="$(arg left_camera_info_topic)"/>

  <remap from="right/camera_info"      to="$(arg right_camera_info_topic)"/>


  <param name="frame_id"                  type="string" value="$(arg frame_id)"/>

  <param name="wait_for_transform_duration"             type="double"     value="$(arg wait_for_transform)"/>

  <param name="approx_sync"             type="bool"   value="$(arg approximate_sync)"/>


  <param name="Vis/Strategy"            type="string" value="$(arg strategy)"/>

  <param name="Vis/FeatureType"         type="string" value="$(arg feature)"/>

  <param name="OdomF2M/NNType"          type="string" value="$(arg nn)"/>

  <param name="Vis/EstimationType"      type="string" value="$(arg estimation)"/>

  <param name="Vis/MaxDepth"            type="string" value="$(arg max_depth)"/>

  <param name="Vis/MinInliers"          type="string" value="$(arg min_inliers)"/>

  <param name="Vis/InlierDistance"      type="string" value="$(arg inlier_distance)"/>
```

```
   <param name="OdomF2M/LocalHistorySize" type="string" value="$(arg local_map)"/>
   <param name="Vis/FillInfoData"        type="string" value="true"/>
   <param      name="Vis/VarianceFromInliersCount"      type="string"      value="$(arg
variance_inliers)"/>
   </node>
   </group>
   <node    if="$(arg    visual_odometry2)"    pkg="viso2_ros"    type="stereo_odometer"
name="stereo_odometer" output="screen">
   <remap from="stereo" to="$(arg stereo_namespace)"/>
   <remap from="image" to="image_rect"/>
   <remap from="/stereo_odometer/odometry" to="rtabmap/odom" />
   <param name="max_features" value="40" />
   <param name="multi_stage" value="0" />
   <param name="bucket_width" value="100.0" />
   <param name="outlier_flow_tolerance " value="2" />
   <param name="bucket_height" value="100.0" />
   <param name="nms_n" value="2" />
   <param name="outlier_disp_tolerance" value="1" />
   <!-- param name="odom_frame_id"  type="string" value="stereo_image_proc"/ -->
   <!-- param name="base_link_frame_id" value="$(arg frame_id)"/ -->
   <param name="publish_tf" value="true"/>
   <param name="ref_frame_change_method" value="1"/>
   <param name="queue_size" value="30"/>
   </node>
   <node  if="$(arg  visual_odometry3)"  pkg="fovis_ros"  type="fovis_stereo_odometer"
name="stereo_odometer" >
   <remap from="stereo" to="$(arg stereo_namespace)"/>
   <remap from="image" to="image_rect"/>
   <param name="publish_tf" value="true"/>
   <remap from="stereo_odometer/odometry" to="rtabmap/odom" />
   <!-- <param name="fast_threshold_adaptive_gain" type="string" value="0.001"/>

   <param name="use-image_normalization" type="string" value="false"/>
   <param name="min-features-for-estimate" value="2"/>
   <param name="ref-frame-changethreshold" value="2"/>
   <param name="fast-threshold" value="20"/>
   <param name="max-keypoints-perbucket" value="100" />
   <param name="odom_frame_id"  type="string" value="/odom"/>
   <param name="base_link_frame_id" value="$(arg frame_id)"/>
   </node>
   <!-- Visualizacion RVIZ -->
   <node if="$(arg rviz)" pkg="rviz" type="rviz" name="rviz" args="$(arg rviz_cfg)"/>
```

```
   <node    if="$(arg    rviz)"    pkg="nodelet"    type="nodelet"    name="points_xyzrgb"
args="standalone rtabmap_ros/point_cloud_xyzrgb">
   <remap from="left/image"        to="$(arg left_image_topic)"/>
   <remap from="right/image"       to="$(arg right_image_topic)"/>
   <remap from="left/camera_info"  to="$(arg left_camera_info_topic)"/>
   <remap from="right/camera_info" to="$(arg right_camera_info_topic)"/>
   <remap from="cloud"             to="points2" />

   <param name="decimation"  type="double" value="2"/>
   <param name="voxel_size"  type="double" value="0.02"/>
   <param name="approx_sync" type="bool"   value="$(arg approximate_sync)"/>
   </node>
   </launch>
```

Bibliografía

[1] ROS. http://www.ros.org/.

[2] Repositorio ROS versión indigo.
http://repositories.ros.org/status_page/ros_indigo_default.html?s=2.

[3] RTABMAP. http://introlab.github.io/rtabmap/.

[4] Stereo imagen proc.
http://wiki.ros.org/stereo_image_proc.

[5] Odometría Fovis. http://wiki.ros.org/fovis_ros.

[6] Odometría VISO2 http://wiki.ros.org/viso2_ros.

[7] Algoritmo RANSAC. http://slideplayer.es/slide/1737326/

[8] Extracción de características.
http://www.sc.ehu.es/ccwgrrom/transparencias/pdf-vision-1-transparencias/capitulo-8.pdf.

[9] Algoritmo SURF.
http://oa.upm.es/20480/1/INVE_MEM_2012_135438.pdf.

[10] Algoritmo SIFT.
http://iie.fing.edu.uy/investigacion/grupos/gti/timag/trabajos/2011/keypoints/PaginaWeb.html.

[11] OpenCV. http://opencv.org/.

[12] Algortimo SIFT.
http://41jaiio.sadio.org.ar/sites/default/files/6_EST_2012.pdf.

[13] LibFovis. http://fovis.github.io/.

[14] LibVISO2. http://www.cvlibs.net/software/libviso/

[15] Repositorio LibVISO2.
http://www.cvlibs.net/software/libviso/.

[16] Rtabmap ROS. http://wiki.ros.org/rtabmap_ros.