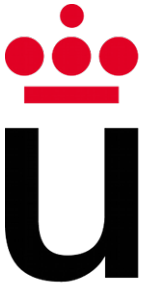


Universidad
Rey Juan Carlos

MÁSTER EN VISIÓN ARTIFICIAL

Curso Académico 2021/2022

Trabajo Fin de Máster

ODOMETRÍA VISUAL TRIDIMENSIONAL EN LA PLATAFORMA EDUCATIVA UNIBOTICS

Autor/a : Pablo Asensio Martínez
Tutor/a : Dr. JoseMaria Cañas Plaza

Trabajo Fin de Máster

Título del Trabajo con Letras Capitales para Sustantivos y Adjetivos

Autor/a : Nombre del Alumno/a
Tutor/a : Dr. Nombre del profesor/a

La defensa del presente Proyecto Fin de Grado/Máster se realizó el día 3 de de 20XX, siendo calificada por el siguiente tribunal:

Presidente:

Secretario:

Vocal:

y habiendo obtenido la siguiente calificación:

Calificación:

Móstoles/Fuenlabrada, a de de 20XX

Aquí normalmente
se inserta una dedicatoria corta

Agradecimientos

Aquí vienen los agradecimientos...

Hay más espacio para explayarse y explicar a quién agradeces su apoyo o ayuda para haber acabado el proyecto: familia, pareja, amigos, compañeros de clase...

También hay quien, en algunos casos, hasta agradecer a su tutor o tutores del proyecto la ayuda prestada...

Resumen

Aquí viene un resumen del proyecto. Ha de constar de tres o cuatro párrafos, donde se presente de manera clara y concisa de qué va el proyecto. Han de quedar respondidas las siguientes preguntas:

- ¿De qué va este proyecto? ¿Cuál es su objetivo principal?
- ¿Cómo se ha realizado? ¿Qué tecnologías están involucradas?
- ¿En qué contexto se ha realizado el proyecto? ¿Es un proyecto dentro de un marco general?

Lo mejor es escribir el resumen al final.

Summary

Here comes a translation of the “Resumen” into English. Please, double check it for correct grammar and spelling. As it is the translation of the “Resumen”, which is supposed to be written at the end, this as well should be filled out just before submitting.

Índice general

Índice de figuras

Índice de fragmentos de código

Capítulo 1

Introducción

En este primer capítulo se propone dar una visión general del contexto en que se encuadra el proyecto fin de máster, que es la visión artificial en el ámbito de las plataformas educativas en línea. Dentro de ésta se abordará el problema al que vamos a hacer frente: creación de un ejercicio de autolocalización visual, o dicho de otro modo, la estimación de la posición y orientación 3D de una cámara haciendo uso de algoritmos de odometría visual. Concretamente se implementará un algoritmo de odometría visual 3D para la plataforma educativa Unibotics.

1.1 Visión Artificial

La visión artificial es un campo de la inteligencia artificial que pretende obtener información del mundo a partir de una o varias imágenes, que normalmente vienen dadas de forma de matriz numérica. La información relevante que se puede obtener a partir de las imágenes puede ser el reconocimiento de objetos, la recreación en 3D de la escena que se observa, el seguimiento de un objeto, etc.

El inicio de la visión artificial se produjo en 1961 por parte de Larry Roberts, quien creó un programa que podía ver una estructura de bloques, analizar su contenido y reproducirla desde otra perspectiva, utilizando para ello una cámara y procesando la imagen desde un ordenador. Sin embargo, para obtener el resultado las condiciones de la prueba estaban muy controladas. Otros muchos científicos también han tratado de solucionar el problema de conectar una cámara a un ordenador y hacer que este describa lo que ve. Finalmente, los científicos de la época se dieron cuenta de que esta tarea no era sencilla de realizar, por lo que se abrió un amplio campo de investigación, que tomó el nombre de visión artificial.

En este campo se persigue, por ejemplo, que el ordenador sea capaz de reconocer en

una imagen distintos objetos al igual que los humanos lo hacemos con nuestra visión. Se ha demostrado que este problema es muy complejo y que algo que para nosotros resulta automático puede que se tarde mucho tiempo en que lo resuelva, con la misma robustez y versatilidad, una máquina. Por otra parte, a pesar del alto precio computacional que se paga por utilizar cámaras como sensor, si se consigue analizar correctamente la imagen, es posible extraer *mucha* información de ella que no podría obtenerse con otro tipo de sensores.

En los años noventa empezaron a aparecer los primeros ordenadores capaces de procesar las imágenes lo suficientemente rápido. Además se comenzó a dividir los posibles problemas de la visión artificial en otros más específicos.

A continuación se presentan algunas aplicaciones y escenarios de aplicación, no solo en el área académica sino también en la vida cotidiana, que hacen uso de los descubrimientos y avances logrados por diversos investigadores en visión artificial.

- Robótica. Unos de los procesos más complejos que tiene que realizar un robot es interpretar el mundo a su alrededor a través de la información que adquiere mediante los sensores, como puede ser una cámara de vídeo. Esta información puede usarse para la localización o reconocimiento de objetos o incluso el seguimiento de los mismos. Los coches autónomos de Tesla o las aspiradoras robóticas de gama alta son ejemplos de robots que usan visión para comprender la escena, detectar objetos de interés o autolocalizarse.

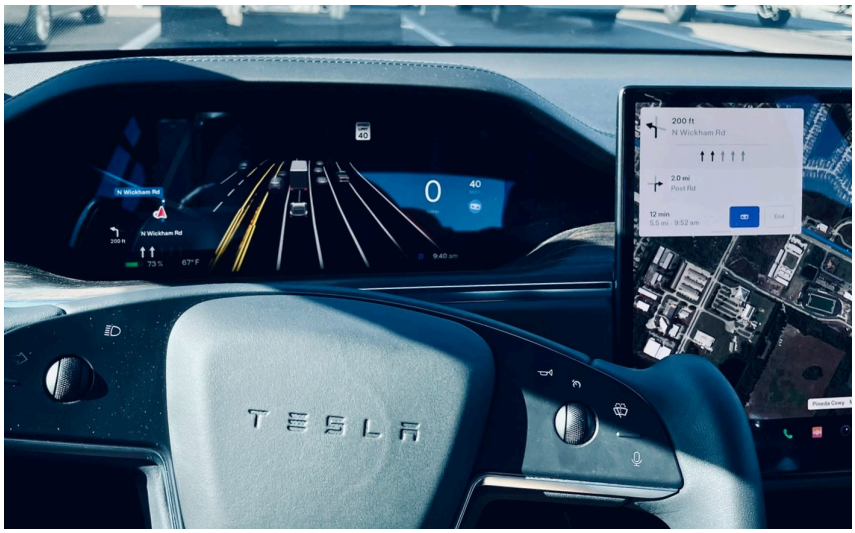


Figura 1.1: Beta de la conducción autónoma total de Tesla

- Videojuegos. Este sector ha contribuido notablemente al desarrollo de la visión artificial. Ejemplos claros de su uso son el dispositivo Kinect desarrollado por Microsoft (cámara RGBD), y Eye Toy, desarrollado por PlayStation para el reconocimiento de los gestos realizados por los jugadores. En la nueva generación de consolas de videojuegos se está avanzando en mejorar la interacción jugador-consola.

- Medicina. Uno de los objetivos de la visión artificial, en este contexto, es el tratamiento y análisis de imágenes, detección de patrones y reconstrucción 3D para ayudar al correcto diagnóstico por parte del especialista clínico. Aplicaciones comunes aquí son la detección y caracterización automática de tumores, mejora de la imagen de microscopio análisis hiperespectral para extraer la composición de los tejidos orgánicos contenidos en una imagen.

- Biometría. La biometría estudia los métodos automáticos para el reconocimiento único de humanos basado en uno o más rasgos conductuales o rasgos físicos intrínsecos. Dentro de estos sistemas se encuentran los sistemas de reconocimiento facial. Estos son muy usados en seguridad, como ejemplo de ello se encuentra el control de fronteras en aeropuertos, el sistema de vigilancia de la ciudad de Londres que realiza un reconocimiento facial mediante cámaras distribuidas por la ciudad; o el reconocimiento de caras y posterior emborronado automático de caras de Google Street View.

Figura 1.2: Control de frontera en el aeropuerto

1.2 Autolocalización Visual

Dentro de la visión artificial se encuentra el problema de la autolocalización visual que consiste en conocer la localización 3D de la cámara en todo momento solamente con las imágenes capturadas y sin disponer de ninguna información extra. Debido al gran abanico de posibilidades que abre resolver este problema, es uno de los retos más importantes dentro del campo de la robótica.

Esta técnica se plantea en los sistemas de navegación automáticos náuticos, terrestres y aéreos. Actualmente numerosas empresas están invirtiendo en este tipo de sistemas en el que apuestan por una navegación total o parcialmente autónoma, un ejemplo puede ser Rumba.

La autolocalización visual es una técnica que permite a aplicaciones de realidad aumentada, que es el término que se usa para definir una visión directa o indirecta de un entorno físico del mundo real, combinar el entorno real con elementos virtuales generados por ordenador para la creación de una realidad mixta en tiempo real. En la imagen 1.3 se puede observar un pokemon que parece que se encuentra en la calle, de frente de quien toma la fotografía, pero realmente ha sido el uso de la realidad aumentada lo que ha permitido una integración fidedigna del pokemon con el entorno en el contexto de la imagen.

Figura 1.3: Realidad aumentada del videojuego para móvil Pokemon Go

Las técnicas de autolocalización han suscitado gran interés por los investigadores en los últimos años. El problema ha sido abordado por dos comunidades distintas. Por un lado la de visión artificial que denominó al problema como *structure from motion* (SfM), donde la información es procesada por lotes, capaz de representar un objeto 2D a 3D con solo unas cuantas imágenes desde diferentes puntos de vista. Y por otro lado la comunidad robótica denominó al problema SLAM (*Simultaneous Localization and Mapping*) que trata de resolver el problema de una manera más ágil adaptando el funcionamiento de los sistemas en tiempo real.

Algunos de los conceptos más interesantes de conocer en la autolocalización visual son:

- Localización absoluta. Esta técnica pretende situar la cámara en una posición cuyo sistema de referencia sea común para todos. Podría usarse el sistema de referencia GPS, o el inicio de la escena, por ejemplo.
- Localización incremental. A diferencia de la localización absoluta, esta técnica hace

uso de un sistema de referencia horizonte local, siendo origen de referencia el *frame* inmediatamente anterior. Haciendo la suma de estos incrementos se podría calcular la posición absoluta.

- Localización desde un mapa conocido. Una técnica que permite, conociendo el escenario de trabajo, poder estimar de una mejor forma, la posición de la cámara.
- Localización sin mapa. Aquella técnica de localización que no hace uso de información conocida previa del escenario.
- Error acumulativo. Es un tipo de error que está en la naturaleza de este problema, ya que al haber pequeños errores de cálculo numérico al computar las matrices de rotación en cada iteración, a la hora de actualizar la pose, se va acumulando este pequeño error.
- Cierre de bucle. Es un mecanismo por el cual se podría utilizar para reajustar los cálculos y reducir en mayor medida el error acumulativo. Por ejemplo, cuando se ha pasado por una zona ya conocida, poder comparar el resultado de la estimación 3D las dos veces.

1.2.1 Structure from Motion (SfM)

Dentro de la visión artificial el Structure from Motion (SfM) es la línea de investigación que toma como entrada únicamente un conjunto de imágenes, y pretende conocer de manera totalmente automática la estructura 3D de la escena vista y las ubicaciones de las cámaras desde donde las imágenes fueron captadas. El SfM es una de las áreas más atractivas de investigación en la última década. Ha llegado a un estado de madurez donde alguno de los algoritmos tienen aplicación comercial.

El SfM surge en la segunda mitad del siglo XIX, denominado fotogrametría, que tuvo como objetivo extraer información geométrica de las imágenes a partir de un conjunto de características manualmente identificadas por el usuario. La fotogrametría hace uso de técnicas de optimización no lineales como el ajuste de haces para reducir al mínimo error de retroproyección. El problema abordado por la comunidad de visión artificial ha sido en su mayoría lograr la completa automatización del proceso. Esto provocó avances en tres aspectos: en primer lugar, restricciones impuestas al movimiento bajo el supuesto de rigidez de la escena; en segundo lugar, la detección de características y descriptores [Canny, 1986], [Harris, Stephens y col., 1988]; y por último, la eliminación de espúreos.

Dadas múltiples vistas, un punto tridimensional puede ser reconstruido mediante triangulación. Un prerrequisito importante es determinar la calibración de la cámara. Se puede representar por una matriz de proyección. La teoría geométrica de SfM permite calcular las matrices de proyección y los puntos 3D utilizando solo correspondencia entre los puntos

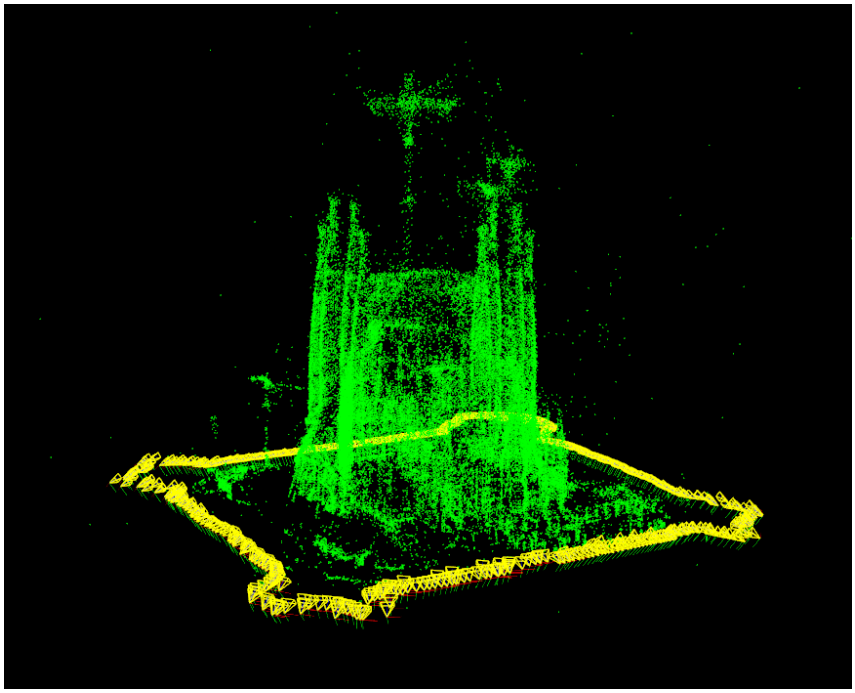


Figura 1.4: Structure from Motion: La sagrada familia

de cada imagen. Para mejorar el rendimiento del SfM se puede aprovechar el conocimiento sobre la escena, con el fin de reducir el número de grados de libertad. Por ejemplo, las restricciones que imponen el paralelismo y la coplanaridad, pueden utilizarse para reconstruir formas geométricas simples como líneas y polígonos planos, a partir de sus posiciones proyectas en vistas simples.

1.2.2 Visual SLAM

La cuestión conocida como Simultaneous Localization and Mapping (SLAM) busca resolver los problemas que plantea colocar un robot móvil en un entorno y una posición desconocidas, y que él mismo se encuentre, sea capaz de construir incrementalmente un mapa de su entorno consistente y a la vez utilizar dicho mapa para determinar su propia localización.

La solución a este problema junto con un mecanismo de navegación haría que el sistema se encuentre con la capacidad para saber a dónde desplazarse, ser capaz de encontrar obstáculos y reaccionar ante ellos de manera inteligente. Esto conseguiría hacer sistemas de robots completamente autónomos.

La resolución al problema SLAM visual ha suscitado un gran interés en el campo de la robótica y se han propuesto muchas técnicas y algoritmos para darle solución, como es el caso del artículo de Durrant-Whyte y Bailey [5]. Y aunque algunas de ellas han obtenido

buenos resultados, en la práctica siguen surgiendo problemas a la hora de buscar el método más rápido o el que genere un mejor resultado con menos índice de fallo. La búsqueda de algoritmos y métodos que resuelvan completamente estos problemas sigue siendo una tarea pendiente.

Uno de los trabajos más importantes en el ámbito es el de monoSLAM de Davison [1] (Andrew J. Davison y Stasse, 2007)[4] que propone resolver este problema con una única cámara RGB como sensor y realizar el mapeado y la localización simultáneamente. El algoritmo propuesto por Davison utiliza un filtro extendido de Kalman para estimar la posición y la orientación de la cámara, así como la posición de una serie de puntos en el espacio 3D. Para determinar la posición inicial de la cámara es necesario a priori dotar de información sobre la posición 3D de por lo menos 3 puntos. Después el algoritmo es capaz de situar la cámara en el espacio tridimensional y de generar nuevos puntos para crear el mapa y servir como apoyo a la propia localización de la cámara. En la Figura 1.8 se pueden ver unas capturas de pantalla sobre uno de los experimentos realizados.

Es importante destacar también la trascendencia que ha tenido el trabajo PTAM (Klein y Murray, 2007) [11] que viene a solucionar uno de los principales problemas que tienen los algoritmos monoSLAM; el tiempo de cómputo, ya que aumenta exponencialmente con el número de puntos (Figura 1.9). Para ello se aborda el problema separando el mapeado de la localización, de tal modo que solo la localización deba funcionar en tiempo real, dejando así que el mapeado trabaje de una manera asíncrona en segundo plano. Este algoritmo parte de la idea de que solo la localización es necesaria que funcione en tiempo real. PTAM hace uso de *keyframes*, es decir, fotogramas clave que se utilizan tanto para la localización como para el mapeado y también de una técnica de optimización mediante ajuste de haces, como en SfM.

1.2.3 Odometría Visual

Dentro de las familias de técnicas pertenecientes a Visual SLAM se encuentra la odometría visual, que es una parte que abordaremos en este trabajo. Consiste en la estimación del movimiento 3D *incremental* de la cámara en tiempo real. Es decir, el cálculo de la rotación y traslación tridimensionales de la cámara a partir de imágenes consecutivas. Se trata de una técnica incremental ya que se basa en la posición anterior para calcular la nueva.

En este tipo de algoritmos se suelen utilizar técnicas de extracción de puntos de interés, cálculos de descriptores y algoritmos para el emparejamiento. Normalmente el proceso es: una vez calculados los puntos emparejados se calcula la matriz fundamental o esencial y descomponerlas mediante SVD para obtener la matriz de rotación y traslación (RT). Una vez que se conocen estas marices de rotación y traslación, se puede calcular mediante un

[1] http://www.doc.ic.ac.uk/~ajd/

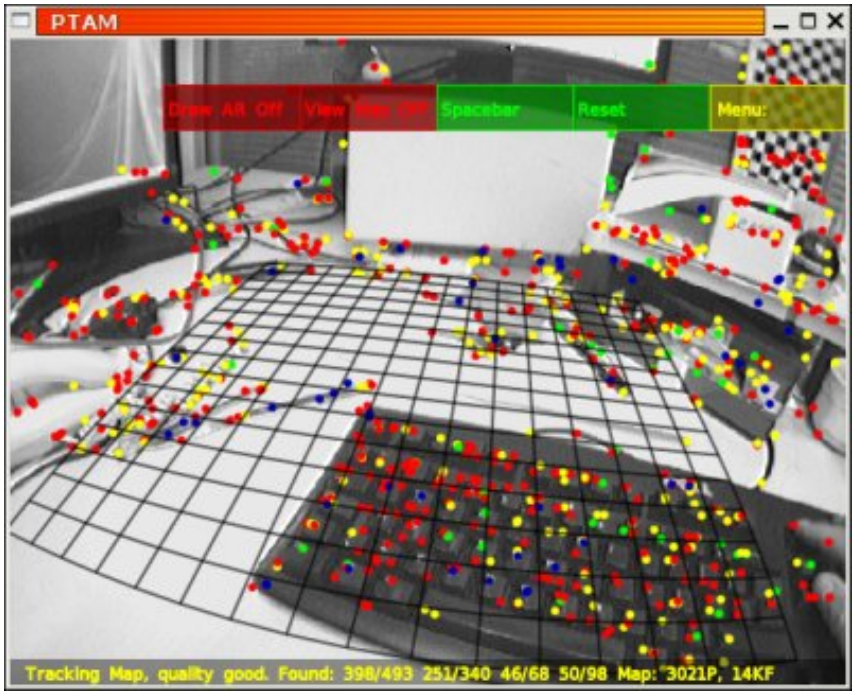


Figura 1.5: PTAM: Funcionamiento sobre un escritorio

algoritmo, la posición que ocupa nuestra cámara en el espacio.

Dentro de la odometría visual podemos diferenciar diferentes tipos:

- Monocular y estéreo. Según la configuración de la cámara, la odometría visual se puede clasificar como odometría visual monocular (cámara única), odometría visual estéreo (dos cámaras en configuración estéreo).
- Método directo y basado en características. La información visual tradicional de la odometría visual se obtiene mediante el *método basado en características*, que extrae los puntos característicos de la imagen y los rastrea en la secuencia de imágenes. Los desarrollos recientes en la investigación en este campo proporcionaron una alternativa, llamada *método directo*, que utiliza la intensidad de píxeles en la secuencia de imágenes directamente como entrada visual. También hay métodos híbridos.
- Odometría inercial visual. Si se utiliza una unidad de medida inercial (IMU) dentro del sistema, esto denomina comúnmente *odometría inercial visual* (VIO).

1.3 Introducción a Unibotics

En los últimos años el uso de plataformas web educativas ha ido incrementando debido a la pandemia del COVID-19, que ha sido un factor por el cual tanto el trabajo como en

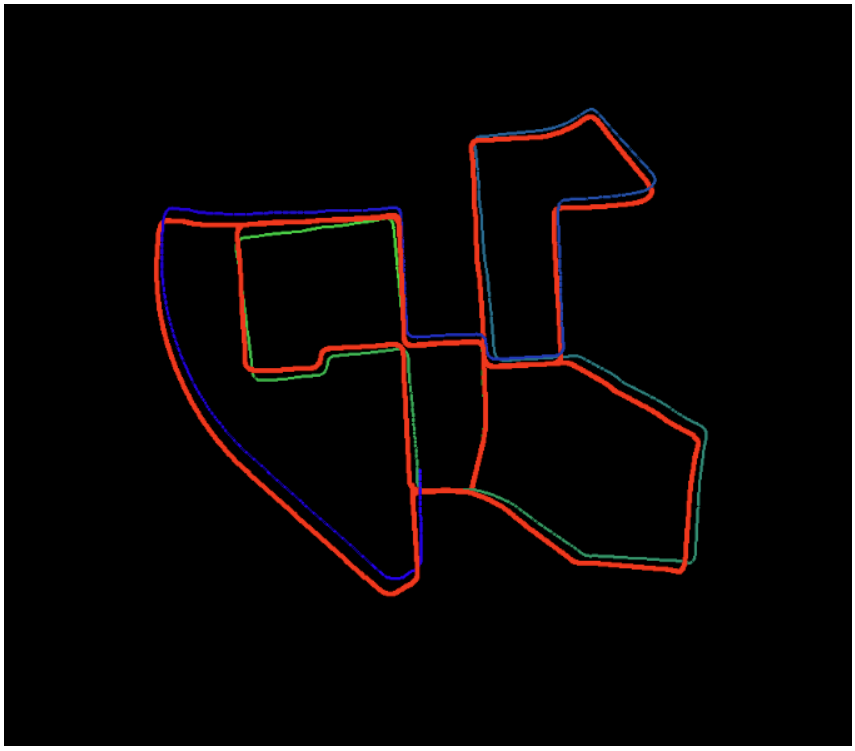


Figura 1.6: Odometría visual 2D. En rojo, la verdad absoluta, frente al resultado de un algoritmo sobre un escenario del dataset de KITTI

estudio a distancia ha aumentado considerablemente. Y, como consecuencia directa el uso de estas plataformas *online* es más demandado. En el área robótica esto no es una excepción. Se está haciendo especial hincapié en el desarrollo de plataformas online que permitan el aprendizaje sobre la programación de robots. Y aunque Unibotics no nació a causa de la pandemia, sí es una de las pioneras.

Unibotics nace como extensión natural de *Robotics Academy*[2]. Es un entorno docente de robótica universitaria. Este entorno tiene una orientación muy práctica en cuanto al aprendizaje de la programación de la inteligencia de los robots. La plataforma posee una gran colección de ejercicios muy variados que abarcan muchas de las aplicaciones robóticas que han surgido recientemente: drones, robots aspiradores, coches autónomos, asistente de aparcamiento, control de robots móviles, etc. La diferencia entre Robotics Academy[2] y Unibotics[3] es la forma en la que se ejecutan los ejercicios. La primera es completamente en local, mientras la segunda hace uso de la red para completar los ejercicios.

[2] http://jderobot.github.io/RoboticsAcademy/
[3] https://unibotics.org/

Figura 1.7: Logo de Unibotics

Toda esta colección de ejercicios son de código abierto lo que proporciona la posibilidad de compartir, modificar y estudiar el código fuente, además de colaborar entre usuarios. Unibotics es un entorno multiplataforma pudiendo implementarse en sistemas operativos tales como Linux, Windows y MacOS. La plataforma hace uso del simulador Gazebo y el lenguaje principal con el que se programa es interpretado y multiplataforma, Python.

Algunos de los ejercicios que podemos encontrar en Unibotics son los siguientes:

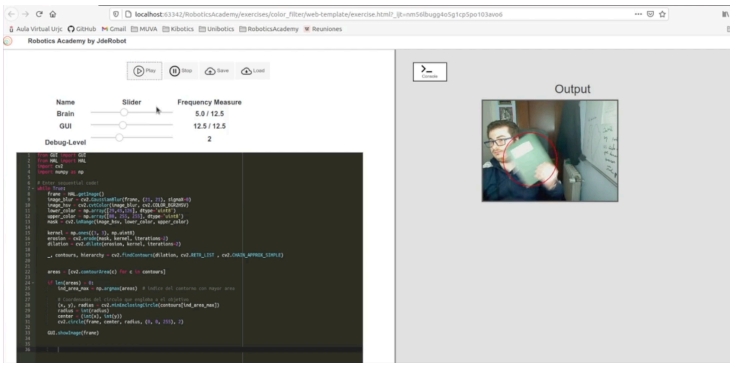


Figura 1.8: Ejercicio *Color Filter* en la plataforma *Unibotics*

- Follow line. El objetivo de *Follow line* es realizar un control reactivo PID capaz de seguir la línea pintada en el circuito de carreras. Utilizando como sensor exclusivamente la cámara.
- 3D Reconstruction. En este ejercicio se busca reconstruir una escena 3D a partir de un par estéreo.
- Color Filter. *Color Filter* es uno de los últimos ejercicios añadidos a la plataforma. Persigue el desarrollo un filtro de color para segmentar algún objeto en la imagen y rastrearlo.

Capítulo 2

Objetivos

2.1 Planificación temporal (Falta)

Es conveniente que incluyas una descripción de lo que te ha llevado realizar el trabajo. Hay gente que añade un diagrama de GANTT. Lo importante es que quede claro cuánto tiempo has consumido en realizar el TFG/TFM (tiempo natural, p.ej., 6 meses) y a qué nivel de esfuerzo (p.ej., principalmente los fines de semana).

2.2 Estructura de la memoria (Falta y cambiar a parrafo final del capitulo 1 Intro)

Por último, en esta sección se introduce a alto nivel la organización del resto del documento y qué contenidos se van a encontrar en cada capítulo.

- En el primer capítulo se hace una breve introducción al proyecto, se describen los objetivos del mismo y se refleja la planificación temporal.
- En el siguiente capítulo se describen el estado del arte así como las tecnologías utilizadas en el desarrollo de este TFM (Capítulo 3).
- En el capítulo **??** Se describe la arquitectura y el proceso de desarrollo de la herramienta.
- En el capítulo **??** Se presentan las principales pruebas realizadas para validación del ejercicio, incluyendo resultados de los experimentales efectuados.

- Por último, se presentan las conclusiones del proyecto así como los trabajos futuros que podrían derivarse de éste (Capítulo 6).

Capítulo 3

Estado del arte

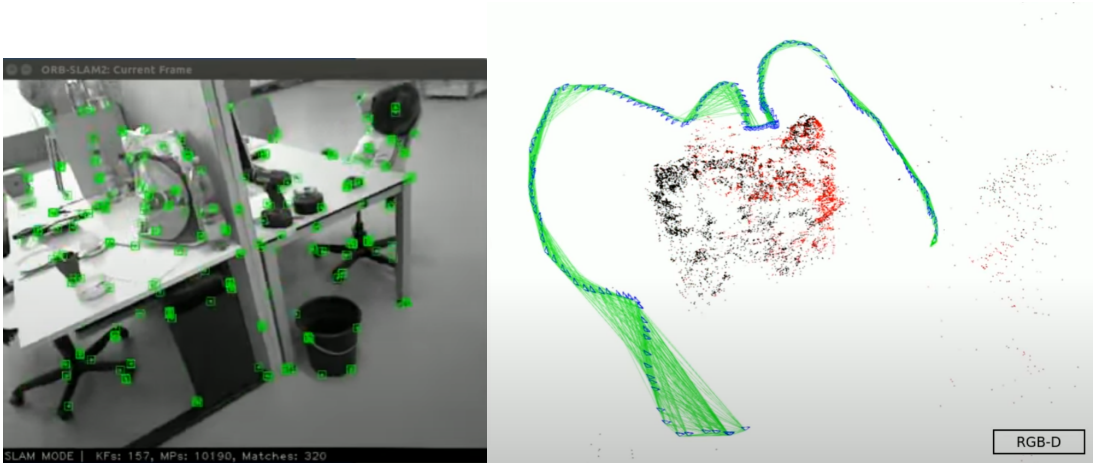
En este capítulo se explicará varios de los algoritmos de odometría visual y de SLAM, así como algoritmos para la evaluación de estos mismos, además de una serie de sitios web con caracter educativo sobre robótica y el entorno que rodea a esta.

3.1 Trabajos de Odometría Visual

3.1.1 ORB-SLAM

ORB-SLAM es un algoritmo desarrollado en la Universidad de Zaragoza por Raúl Mur et al. presentado en el año 2015 [13] y mejorado en el año 2017 [14]. Es un método basado en características capaz de ser empleado mediante sistemas monoculares, estéreo o cámaras de profundidad RGB-D. El algoritmo toma este nombre debido a que hace uso de descriptores ORB [17] para la detección y emparejamiento de píxeles de interés. Una de sus características principales es que, junto a los hilos de ejecución de Tracking y Mapping, añade un tercer proceso dedicado a la detección de cierres de bucle o Looping. Haciendo uso de estos tres hilos, el algoritmo puede funcionar en tiempo real siempre y cuando el procesador tenga cierta capacidad de cómputo, dado que, pese a que los descriptores ORB son más robustos que los empleados por los anteriores algoritmos, requieren de un mayor tiempo de cómputo. Por otra parte, el proceso de inicialización se realiza empleando el método de homografía y, por otra parte, mediante la matriz fundamental, haciendo uso del algoritmo de los ocho puntos [9]. Cada uno de los métodos generará un mapa diferente y se determinará cuál de ellos se usará. Si la escena consta de un plano principal se escogerá el mapa obtenido por homografía, y en caso contrario, el obtenido mediante la matriz fundamental. Este proceso permite obtener una mayor robustez en la generación del mapa inicial que será determinante en los futuros procesos.

Tanto el proceso de Mapping como el de Tracking son muy similares a los implementados en PTAM. El hilo de Mapping se encarga de crear KeyFrames cuando sea necesario, añadir nuevos puntos 3D al mapa y optimizar el mismo haciendo uso del algoritmo de Boundle Adjustment. Sin embargo, en este caso, el emparejamiento de los puntos 3D se realiza mediante los descriptores ORB. Además, ORB-SLAM construye un grafo de co-visibilidad que relaciona unos KeyFrames con otros en función de cuántos puntos 3D son observados de forma común.



(a) (b)

Figura 3.1: ORB-SLAM. Localización de los puntos característicos detectados con ORB en color verde junto al mapa generado.

Esto ayuda, entre otras cosas, a eliminar KeyFrames redundantes. Al igual que en los casos anteriores, el proceso de Tracking realiza en cada iteración una primera estimación de la posición de la cámara empleando un modelo de velocidad constante y empareja los puntos 3D visibles en el fotograma actual y anterior calculando la posición final a partir de los emparejamientos obtenidos por triangulación. Además, incorpora un mecanismo para recuperar la posición de la cámara ante pérdidas (oclusiones, baja calidad visual, secuestros, etc.). Esto se consigue buscando KeyFrames cuya información visual concuerde con la del fotograma actual. Por tanto, es necesario que la cámara vuelva a pasar por un lugar previo almacenado en el mapa para poder reiniciarse. Para reducir el coste computacional de la obtención de los posibles KeyFrames candidatos se utiliza un modelo de bolsa de palabras [6]. En la Figura 3.1 se pueden observar puntos característicos detectados mediante ORB y el mapa reconstruido por ORB-SLAM.

Por último, la función del hilo de Looping es la detección de posibles cierres de bucle. Funciona igual que el proceso de recolección, haciendo uso del modelo de bolsa de palabras junto con el grafo de co-visibilidad. En el caso de detectar un cierre de bucle, se corrige la posición del fotograma actual para hacerla coincidir con el KeyFrame en cuestión, se eliminan posibles puntos 3D duplicados, y finalmente se corrige la posición de los KeyFrames restantes para hacer coincidir la trayectoria con la nueva posición. La estructura propues-

ta por ORB-SLAM otorga una gran robustez tanto en entornos pequeños como extensos, sumado al hecho de estar liberado como un proyecto de código abierto[1], ha hecho que sea uno de los algoritmos de Visual SLAM más conocidos y utilizados hoy en día.

3.1.2 SD-SLAM

El algoritmo SD-SLAM, que es el acrónimo de Semi-Direct SLAM, fue desarrollado en la Universidad Rey Juan Carlos por Eduardo Perdices como parte de su tesis doctoral [7] en el año 2017. SD-SLAM es el resultado de mejorar ORB-SLAM transformándolo, de un método basado en características a uno híbrido, debido a la adición de métodos directos.

Al ser una extensión de ORB-SLAM consta de todas las características que este disponía, como el uso de 3 hilos de procesamiento (Tracking, Mapping y Looping), soporte para sistemas monoculares, estéreo y cámaras de profundidad RGB-D, uso de descriptores ORB y módulo de recolección. Pero además, añade una serie de nuevas características como son el uso de métodos directos en el proceso de estimación de la pose de la cámara y una nueva forma de representar los puntos 3D.

Donde encontramos la principal diferencia es en el hilo de Tracking. En ORB-SLAM este proceso consistía, en primer lugar, en una estimación de la posición basada en un modelo de velocidad constante para posteriormente realizar una corrección basada en el emparejamiento de puntos ORB minimizando el error de retro-proyección. Este último proceso es iterativo y, por tanto, si se proporciona una pose inicial cercana a la real el algoritmo convergerá en un menor tiempo.

Con el objetivo de proporcionar una mejor estimación inicial antes de comenzar el proceso de emparejamiento, SD-SLAM añade un método directo basado en la minimización del error fotométrico, similar al realizado por SVO, que denomina *alineamiento de imágenes*. Este proceso consiste en la proyección de los puntos 3D del fotograma anterior visibles desde el fotograma actual y construye, para cada uno de ellos, parches de tamaño 4x4 píxeles para ser comparados. Además, mientras que en SVO estos parches se transforman aplicando una matriz afín asociada al movimiento antes de ser comparados, SD-SLAM evita este paso dado que presupone que el desplazamiento entre fotogramas será pequeño y se podrá asumir que no existe una gran variación en las poses y, por consiguiente, en los parches. Finalmente, se minimiza el error fotométrico (problema no lineal de mínimos cuadrados) con el algoritmo de Gauss-Newton [10].

Este proceso de alineación de imágenes se realiza haciendo uso de una pirámide de imágenes para detectar rápidamente el desplazamiento en las imágenes de menor tamaño, para posteriormente ser refinado a medida que se avanza por la pirámide. El resultado

[1] https://github.com/raulmur/ORB_SLAM2

es una segunda estimación de pose muy rápida y cercana a la final, agilizando el proceso final de corrección (emparejamiento y minimización del error de retro-proyección). De este modo se logra una reducción en el tiempo de cómputo respecto a ORB-SLAM, resultando especialmente útil para incorporar este algoritmo en sistemas con capacidad de cómputo limitada, como por ejemplo en los drones.

Los módulos de relocalización y detección de cierres de bucle (Looping) también han sido mejorados para hacer uso del alineamiento de imágenes. Estas dos operaciones se basan en la comparación del fotograma actual con un subconjunto de KeyFrames potencialmente similares, y por tanto, el tiempo de ejecución es dependiente del número de KeyFrames presentes en el mapa, dificultando su ejecución en tiempo real cuando este número crece. Al ser modificadas del mismo modo que se hizo en el proceso de Tracking permite reducir el tiempo de cómputo y tolerar así un mayor número de KeyFrames.

Por otra parte, el hilo de Tracking trabaja tal y como lo hacía en el algoritmo de ORBS-LAM exceptuando el modo en el que se representan los puntos 3D del mapa. La mayoría de algoritmos de Visual SLAM representan los puntos 3D mediante las coordenadas de su posición espacial en el mundo (X; Y;Z) desde un origen arbitrario (normalmente el primer KeyFrame del mapa). La posición 3D es calculada mediante triangulación. Este proceso es dependiente de que el ángulo que forman los rayos de retro-proyección sea suficientemente amplio; es decir, el desplazamiento (paralaje) entre las observaciones debe ser suficiente. A medida que este ángulo disminuye, el punto se aleja y su incertidumbre crece, provocando potenciales errores. De hecho, existen objetos para los cuales nunca se logrará un desplazamiento suficiente para poder estimar su posición. Por ejemplo, las estrellas se podrían considerar en el "infinito" y, por tanto, no será posible obtener un paralaje suficiente para estimar su profundidad real.

SD-SLAM hace uso de la inversa de la profundidad [3] para representar los puntos 3D. Además, la posición de estos puntos se representa como una matriz de transformación (rotación y traslación) respecto de la cámara donde fueron detectados por primera vez. De este modo, los puntos 3D están referenciados a los distintos KeyFrames donde se detectaron por primera vez en lugar de a un origen arbitrario. La ventaja de esta representación en comparación a la parametrización clásica es que es posible representar puntos en el infinito, dado que la inversa de la profundidad en el infinito es 0.

3.1.3 VIO-SDSLAM

El algoritmo VIO-SDSLAM, que es el acrónimo de Visual Inertial Odometry SD-SLAM, fue desarrollado en la Universidad Rey Juan Carlos por Javier Martínez del Río como parte de su TFM [16] en el año 2020. VIO-SDSLAM es el resultado de mejorar SD-SLAM, añadiendo un método basado en estimación de la posición a través de una unidad de medida inercial, la cual redunda la información captada por el sensor óptico. Permitiendo que en

caso de no tener información de la cámara, poder seguir generando KeyFrames nuevos, llamados Fake-KeyFrames. Logrando una navegación "a ciegas".

Los principales problemas que se resuelven en este trabajo son:

- **Ambigüedad en la escala.** Está asociado al proceso de construcción del mapa inicial, donde no es posible conocer la distancia real a la cual se encuentran los objetos. Este es un problema inherente de los sistemas monoculares debido a la pérdida de información 3D al momento de ser proyectada sobre el plano imagen (2 dimensiones). Es decir, un objeto en el mundo real de un determinado tamaño a una cierta distancia proyecta sobre el plano imagen la misma información que un objeto del doble de tamaño al doble de distancia. Debido a esto, los sistemas monoculares de SLAM fijan una escala arbitraria en cada proceso de inicialización de la cámara.

- **Modelo de movimiento.** El coste computacional de determinar la nueva posición de la cámara en cada instante de tiempo es elevado. En los algoritmos basados en características este proceso suele ser iterativo, basándose en la retro-proyección y emparejamiento de los puntos visibles del mapa sobre cada nueva imagen. Con el objetivo de acelerar este proceso se emplean técnicas como la estimación de la nueva posición mediante modelos de movimiento o alineaciones de imágenes.

- **Recuperación ante pérdidas.** Cuando la calidad visual disminuye, debido a oclusiones, bajas texturas, movimientos bruscos o secuestros entre otros motivos, provoca que el algoritmo entre en un estado de pérdida del cual no es capaz de recuperarse directamente.

  La incorporación de una IMU puede complementar el proceso de la estimación de posición y orientación de la cámara. Sin embargo, este sensor sufre de derivas espaciales debido al error acumulativo proveniente de sus medidas. Estos errores pueden proceder de la calidad del sensor, proporcionando datos no muy fiables o muy ruidosos. Además el acelerómetro está influenciado por la fuerza de la gravedad y se necesita una estimación de la orientación para eliminar este efecto. Cualquier pequeño error producido bien por las medidas del sensor o por el modelo de odometría implementado será acumulado de una estimación a la siguiente, es decir, el error aumenta a medida que el tiempo avanza.

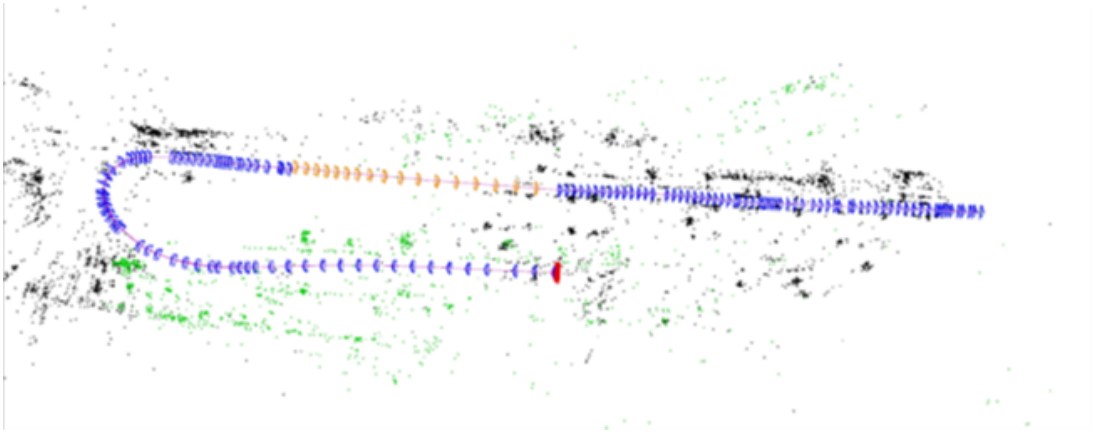


Figura 3.2: VIO SD-SLAM: Trayectoria estimada. En azul los KeyFrames estimados por visión y en naranja los Fake-KeyFrames estimados por odometría inercial durante el estado de pérdida.

Por todas las características descritas que dan forma al algoritmo de VIO-SDSLAM se ha decido hacer uso de este para el desarrollo del proyecto. Los principales motivos para la elección de este algoritmo son los siguientes:

- Código abierto[2].
- Soporta sistemas monoculares.
- Capacidad de trabajar en tiempo real.
- Inicialización robusta (homografía y matriz fundamental).
- Incorpora un módulo para detectar cierres de bucle.
- Emplea hilos asíncronos para los módulos de Tracking, Mapping y Looping.
- Es capaz de representar puntos en el infinito.
- Posibilidad de realizar navegación a estima[3] con la IMU.

3.2 Trabajos de Evaluación en Visual SLAM

3.2.1 SLAMTestbed

SlamTestbed [12] es una aplicación diseñada y creada para comparar cuantitativamente algoritmos SLAM. El diseño de esta herramienta, que por sencillez se tratará como una

[2] https://github.com/JdeRobot/SDslam/tree/vio-sdslam

[3] Navegación a estima es aquella que se efectúa según el rumbo y la distancia navegada.

caja negra que cuenta en la entrada con dos secuencias de puntos 3D orientados. Uno de los datasets será la verdad absoluta, y el segundo dataset será la posición y orientación en 3D obtenidos tras aplicar un algoritmo Visual SLAM, correspondiendo cada registro del dataset con una posición de la cámara. Una vez procesados los dos datasets por SlamTestbed, se obtiene como salida las transformaciones estimadas por la herramientas entre la verdad absoluta y las posiciones y orientaciones calculadas por el algoritmo de SLAM. Además, se mostrará un conjunto de estadísticas que miden el error cometido en las estimaciones de SLAM qué caracteriza la precisión de los algoritmos.

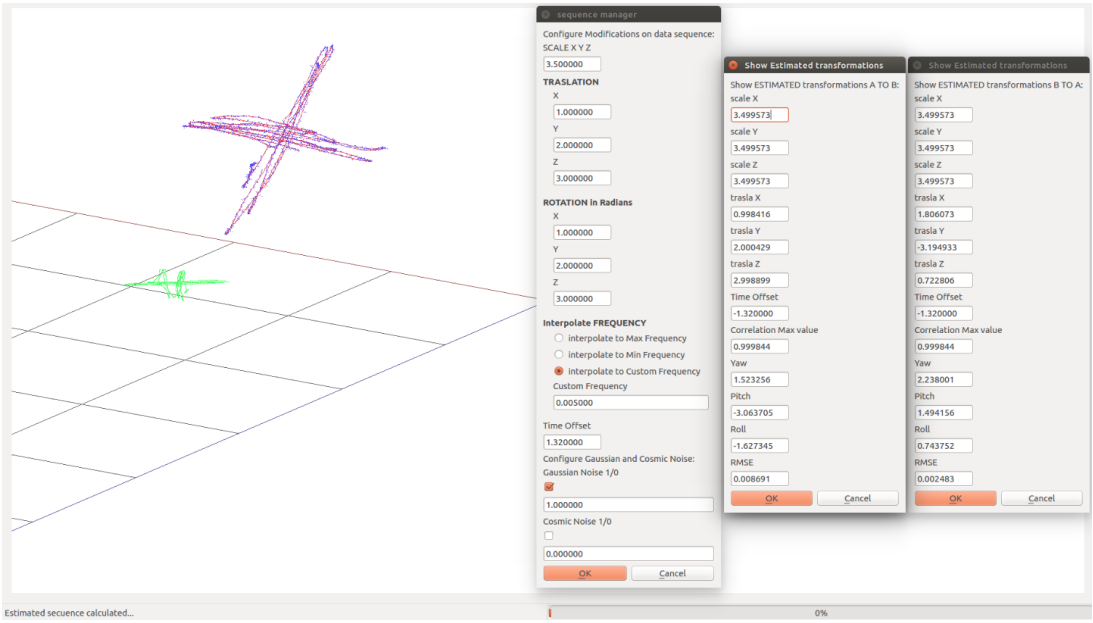


Figura 3.3: SLAMTestbed: Resultados de la estimación de un cambio de escala y traslación, rotación, offset y ruido gaussiano simultáneos.

El objetivo principal de la herramienta desarrollada es calcular el error existente entre una secuencia con las posiciones y orientaciones 3D verdaderas (dataset A) y la trayectoria calculada por el algoritmo de SLAM (dataset B). Para que esto sea posible necesitaremos antes eliminar algunas variables que no permiten comparar directamente las dos trayectorias, como son la escala, el offset temporal y la transformación en 3D entre ellas. Por ello, se necesita calcular un nuevo dataset (estimado), que sea comparable con el dataset A.

Las principales funcionalidades utilizadas para obtener el dataset estimado son:

- **Cálculo de PCA**. El análisis de componentes principales (o PCA), permite reducir los dos datasets a sus componentes principales, lo que posibilita estimar a continuación la escala y el offset existente entre ellos.
- **Estimación de escala**. Estima la diferencia de escala entre los dos datasets a partir de los datos proporcionados en el cálculo de componentes principales.

- **Estimación de offset temporal**. Con este módulo podremos hallar la diferencia entre marcas de tiempos de los 2 datasets, ya que pueden haber comenzado en periodos de tiempo distintos.

- **Interpolación para igualar frecuencias de muestreo**. Se pueden igualar en frecuencia los dos datasets, en caso de que estas sean distintas.

- **Operaciones de registro para estimar la Rotación y Traslación**. Permite estimar la traslación y rotación existentes entre el dataset A y el dataset B y llevarlas así al mismo sistema de referencia espacial, donde ya son directamente comparables.

3.3 Trabajos de Formación en Robótica

3.3.1 TheConstruct

TheConstruct [4] es una plataforma web que enseña sobre robótica, ROS e inteligencia artificial. Tiene una versión gratuita en la que se ofrecen tres cursos: Linux para robótica, Python3 para robótica y C++ para robótica, si se quiere puedes acceder a todos los cursos con su versión de pago. Está desarrollado para que puedan utilizarlo tanto principiantes como profesionales. No requiere de la instalación de ROS. Además, una de sus ventajas es que tiene una gran comunidad donde se puede establecer contactos y aprender nuevas formas de programar robots. TheConstruct cuenta con robots reales que se pueden alquilar por un determinado tiempo, dando la posibilidad de poder conectarse a ellos y programarlos desde cualquier lugar. Una vez se selecciona un curso, se tiene en la parte izquierda la teoría para realizar el curso. También tiene un interfaz de usuario dónde se escribe el código, un terminal para escribir comandos y una simulación del robot que se va a programar.

[4] https://www.theconstructsim.com/

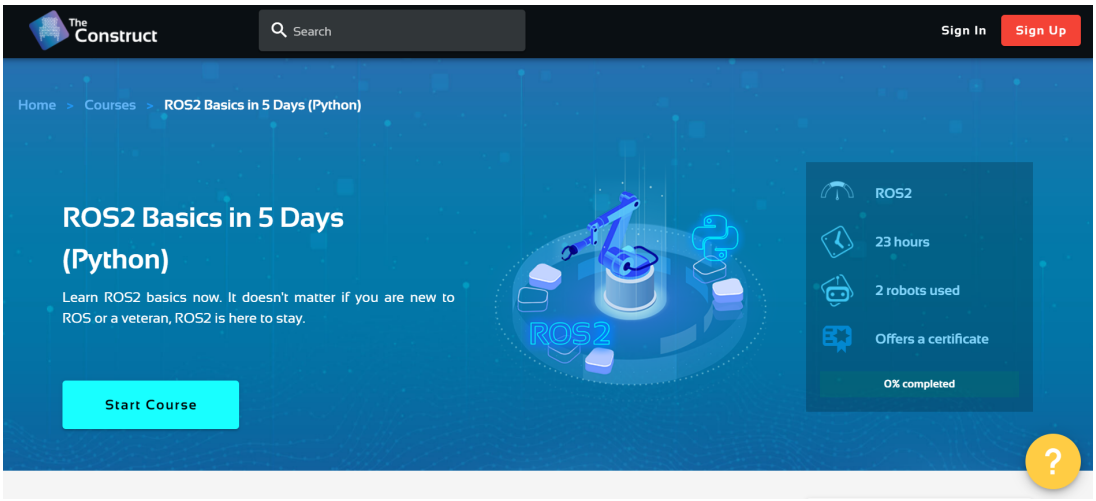


Figura 3.4: TheConstruct

3.3.2 Riders.ai

Riders.ai[5] es una plataforma sobre robótica de simulación, educación y competiciones basada en la nube, desarrollada por Acrome Robotic Systems. Es una plataforma de pago, solamente es gratis la primera lección del curso. Consta de otros dos cursos, los cuales son la continuación del curso mencionado anteriormente. Además, una vez finalizados los cursos se obtiene un certificado. Se enseña a programar en Python o C++ los robots. Las lecciones están formadas por la teoría, el interfaz de usuario y el simulador Gazebo, todo ello en la misma pestaña, como se muestra en la figura 3.5, Riders dispone de dos ligas para competir con otros programadores. Una liga consiste en programar drones voladores y otra en programar robots móviles.

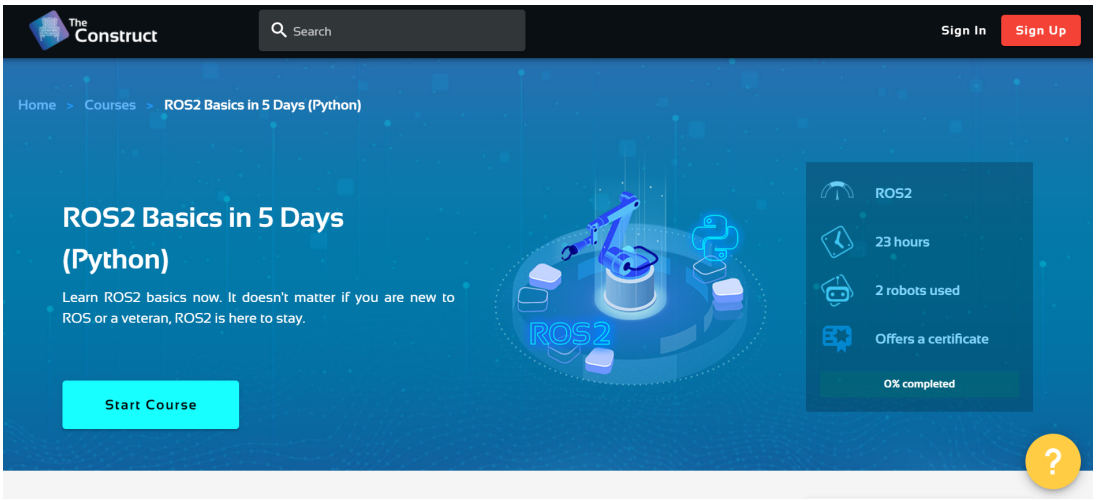


Figura 3.5: Riders.ai

[5] https://riders.ai/

3.4 Entorno de desarrollo: PyCharm

PyCharm es un Integrated Development Enviroment (Entorno de Desarrollo Integrado) (IDE) dedicado concretamente a la programación en Python y desarrollado por la compañía checa JetBrains.

Proporciona análisis de código, un depurador gráfico, una consola de Python integrada, control de versiones y, además, soporta desarrollo web con Django. Todas estas características lo convierten en un entorno completo e intuitivo, idóneo para el desarrollo de proyectos académicos como el que nos ocupa.

Capítulo 4

Algoritmo de Odometría Visual 3D

Atención: Este capítulo se introdujo como requisito en 2019.

Describe los experimentos y casos de test que tuviste que implementar para validar tus resultados. Incluye también los resultados de validación que permiten afirmar que tus resultados son correctos.

4.1 Incorporación de código en la memoria

4.1.1 Pseudo códigos / codigos de backend

4.1.2 Pseudo códigos / codigos de frontend

4.2 Test y validación

4.2.1 Validación Experimental

Capítulo 5

Integración en Robotics Academy

Capítulo 6

Conclusiones y trabajos futuros

6.1 Consecución de objetivos

Esta sección es la sección espejo de las dos primeras del capítulo de objetivos, donde se planteaba el objetivo general y se elaboraban los específicos.

Es aquí donde hay que debatir qué se ha conseguido y qué no. Cuando algo no se ha conseguido, se ha de justificar, en términos de qué problemas se han encontrado y qué medidas se han tomado para mitigar esos problemas.

Y si has llegado hasta aquí, siempre es bueno pasarle el corrector ortográfico, que las erratas quedan fatal en la memoria final. Para eso, en Linux tenemos aspell, que se ejecuta de la siguiente manera desde la línea de *shell*:

```
aspell --lang=es_ES -c memoria.tex
```

6.2 Trabajos futuros

Ningún proyecto ni software se termina, así que aquí vienen ideas y funcionalidades que estaría bien tener implementadas en el futuro.

Es un apartado que sirve para dar ideas de cara a futuros TFGs/TFMs.

Siglas

IDE Integrated Development Enviroment (Entorno de Desarrollo Integrado). 22

Referencias

[1] John Canny. «A computational approach to edge detection». En: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1986), págs. 679-698.

[2] José M Cañas y col. «A ROS-based open tool for intelligent robotics education». En: *Applied Sciences* 10.21 (2020), pág. 7419.

[3] Javier Civera, Andrew J Davison y JM Martinez Montiel. «Inverse depth parametrization for monocular SLAM». En: *IEEE transactions on robotics* 24.5 (2008), págs. 932-945.

[4] Andrew J Davison y col. «MonoSLAM: Real-time single camera SLAM». En: *IEEE transactions on pattern analysis and machine intelligence* 29.6 (2007), págs. 1052-1067.

[5] Hugh Durrant-Whyte y Tim Bailey. «Simultaneous localization and mapping: part I». En: *IEEE robotics & automation magazine* 13.2 (2006), págs. 99-110.

[6] Dorian Gálvez-López y Juan D Tardos. «Bags of binary words for fast place recognition in image sequences». En: *IEEE Transactions on Robotics* 28.5 (2012), págs. 1188-1197.

[7] Eduardo Perdices García. «Técnicas para la localización visual robusta de robots en tiempo real con y sin mapas». https://burjcdigital.urjc.es/handle/10115/14747. Tesis doct. Universidad Rey Juan Carlos, 2017.

[8] Chris Harris, Mike Stephens y col. «A combined corner and edge detector». En: *Alvey vision conference*. Vol. 15. 50. Citeseer. 1988, págs. 10-5244.

[9] R Hartley y A Zisserman. «Multiple View Geometry in Computer Vision, Cambridge Uni». En: *Pr., Cambridge, UK* 1 (2000), pág. 2.

[10] Carl T Kelley. *Iterative methods for optimization*. SIAM, 1999.

[11] Georg Klein y David Murray. «Parallel tracking and mapping for small AR workspaces». En: *2007 6th IEEE and ACM international symposium on mixed and augmented reality*. IEEE. 2007, págs. 225-234.

[12] Elías Barcia Mejias. «Herramienta de evaluación cuantitativa de algoritmos Visual SLAM». https://gsyc.urjc.es/jmplaza/students/tfm-visualslam_slamtestbed-elias_barcia-2019.pdf. Tesis de mtría. Universidad Rey Juan Carlos, 2019.

[13] Raúl Mur-Artal, J. M. M. Montiel y Juan D. Tardós. «ORB-SLAM: a Versatile and Accurate Monocular SLAM System». En: *IEEE Transactions on Robotics* 31.5 (2015), págs. 1147-1163. DOI: 10.1109/TRO.2015.2463671.

[14] Raúl Mur-Artal y Juan D. Tardós. «ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras». En: *IEEE Transactions on Robotics* 33.5 (2017), págs. 1255-1262. DOI: 10.1109/TRO.2017.2705103.

[15] Victor Arribas Raigadas. «Análisis de algoritmos de VisualSLAM: un entorno integral para su evaluación». https://gsyc.urjc.es/jmplaza/students/tfm-visualslam_evaluation-victor_arribas-2016.pdf. Tesis de mtría. Universidad Rey Juan Carlos, 2016.

[16] Javier Martínez del Río. «Algoritmo de autolocalización basado en visión y sensor inercial: VIO-SDSLAM». https://gsyc.urjc.es/jmplaza/students/tfm-slam-vio_sdslam-javier_martinez-2020.pdf. Tesis de mtría. Universidad Rey Juan Carlos, 2020.

[17] Ethan Rublee y col. «ORB: An efficient alternative to SIFT or SURF». En: *2011 International conference on computer vision*. Ieee. 2011, págs. 2564-2571.