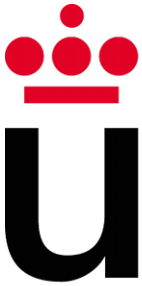


Universidad
Rey Juan Carlos

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.

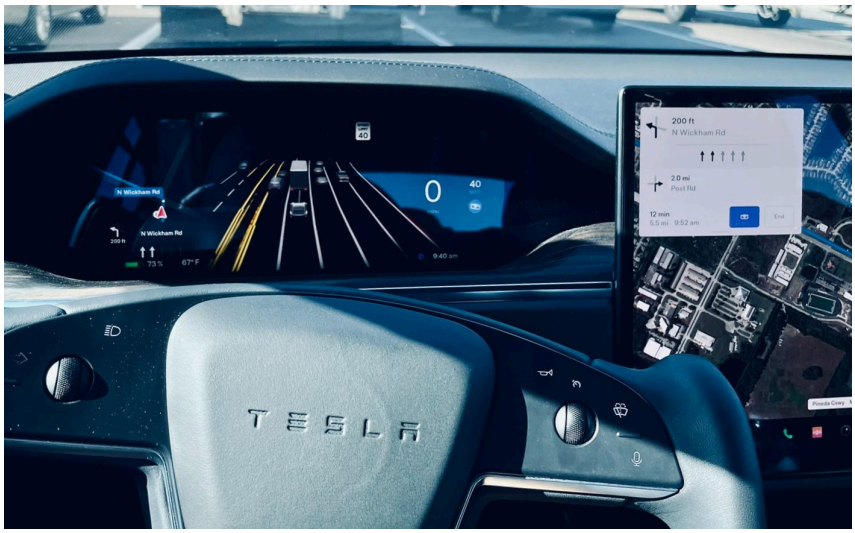


Figura 1.1: Beta de la conducción autónoma total de Tesla

- 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 Roomba.

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, 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 pasa por una zona ya conocida, poder comparar el resultado de la estimación 3D las dos veces, sabiendo que es la misma zona, y por ello deberían ser las mismas coordenadas.

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

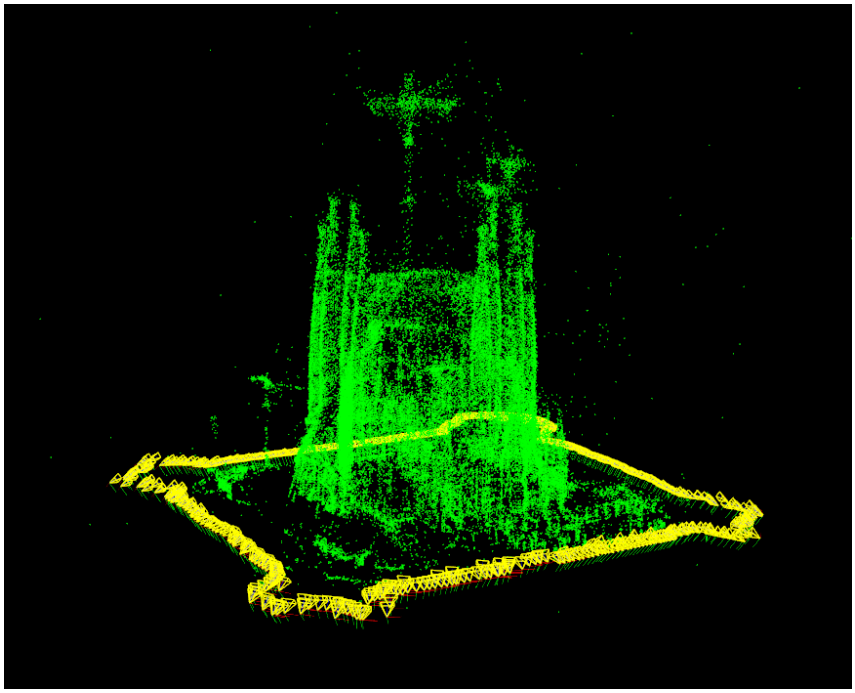


Figura 1.4: Structure from Motion: La sagrada familia

matrices de proyección y los puntos 3D utilizando solo correspondencia entre los puntos 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 [4]. 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)[3] 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) [9] 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

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

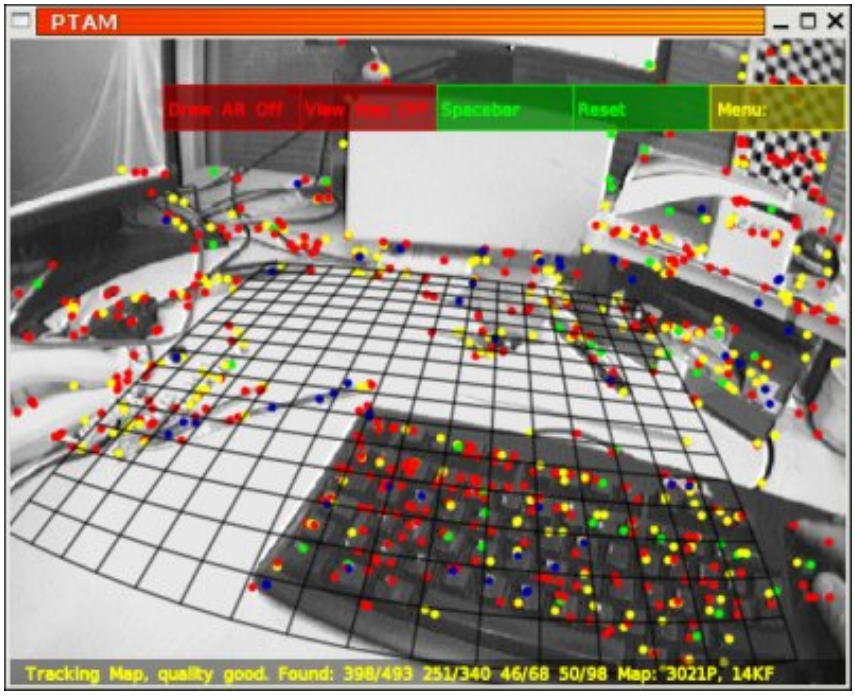


Figura 1.5: PTAM: Funcionamiento sobre un escritorio

vez que se conocen estas marices de rotación y traslación, se puede calcular mediante un 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).

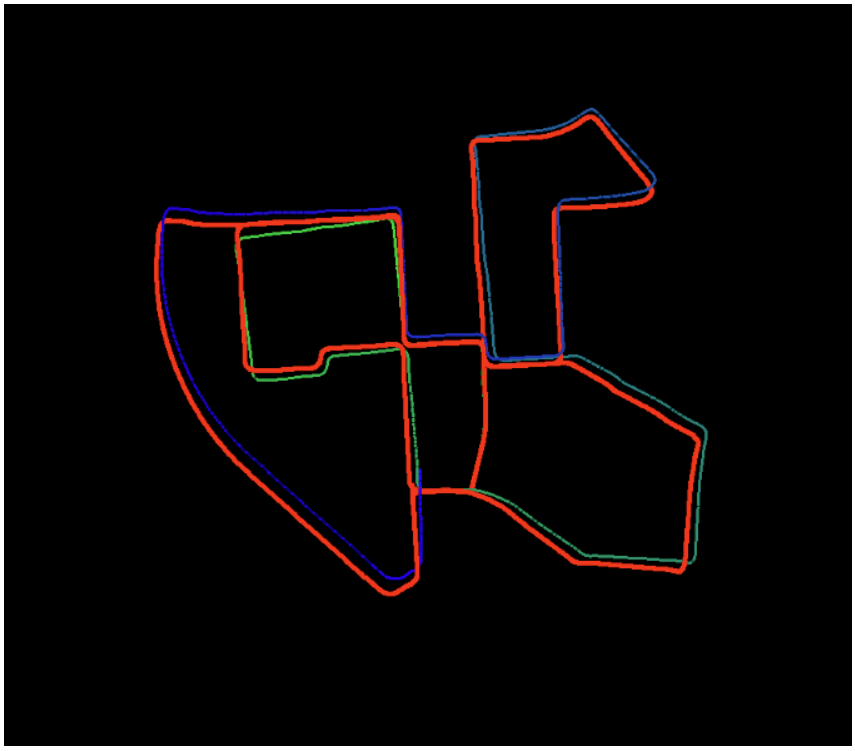


Figura 1.6: Odometría visual 2D. En rojo, la verdad absoluta, frente al resultado de un algoritmo sobre un escenario del dataset de KITTI

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 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, ya que se utiliza como editor del código fuente el navegador web, el cual también es el interfaz gráfico de la ejecución. 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

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

primera es completamente en local, mientras la segunda hace uso de la red para completar los ejercicios.

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 web 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 relacionado con visión artificial son los siguientes:

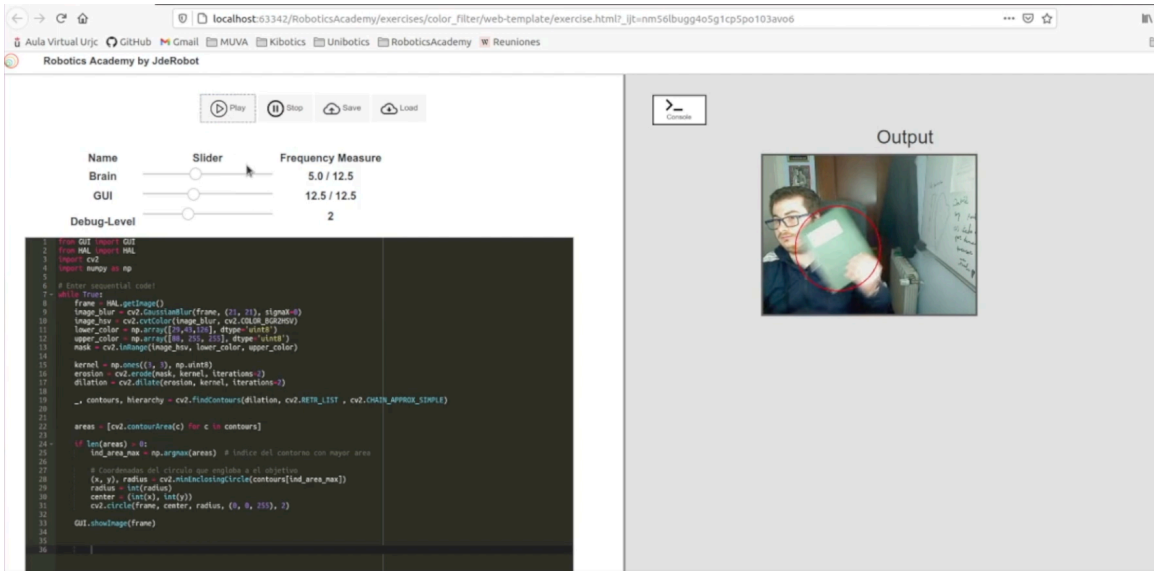


Figura 1.8: Ejercicio *Color Filter* en la plataforma *Unibotics*

- 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. Pincha aquí para ver el vídeo.

- 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. Pincha aquí para ver el vídeo.

- Optical Flow Teleop. En este ejercicio se pretende desarrollar un algoritmo de flujo óptico para teleoperar el robot utilizando las imágenes obtenidas de una cámara web. Pincha aquí para ver el vídeo.

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 presentarán algoritmos de odometría visual, así como técnicas para evaluar su efectividad. Además, se proporcionarán enlaces a sitios web educativos sobre robótica, un campo donde la visión artificial es una herramienta comúnmente utilizada.

En concreto, se explicarán los diferentes tipos de algoritmos de odometría visual, así como algunos algoritmos, se hablará de los *datasets* utilizados y se presentarán métodos para evaluar su rendimiento. También se proporcionarán enlaces a recursos en línea para aprender más sobre el uso de la visión artificial en la robótica.

3.1 Tipos de Algoritmos

La estimación de la posición de un robot móvil puede llevarse a cabo de diversas maneras, una de ellas es mediante la odometría basada en la visión. Esta técnica se puede aplicar de tres formas diferentes:

3.1.1 Enfoque basado en características

Enfoque basado en características. Este enfoque utiliza características específicas en el entorno, como puntos o líneas, para estimar la posición del robot. Estas características son seleccionadas porque son fáciles de detectar y su posición es conocida con bastante precisión.

Para llevar a cabo el seguimiento de estas características, se pueden utilizar diversos algoritmos de detección y seguimiento. Una vez que se han detectado y seguido las carac-

terísticas, se pueden utilizar técnicas de correlación para determinar el desplazamiento del robot en relación con ellas.

El enfoque basado en características tiene varias ventajas. En primer lugar, es relativamente fácil de implementar y requiere poca potencia de procesamiento. Además, es robusto frente a cambios en la iluminación y en la apariencia del entorno. Sin embargo, tiene algunas desventajas. Una de ellas es que solo es posible seguir un número limitado de características, lo que puede dificultar la estimación de la posición del robot en entornos con pocas características distintivas. Además, la precisión de la estimación de la posición puede ser limitada por la precisión de la detección y seguimiento de las características.

Un trabajo de investigación interesante, aunque no se trata de un trabajo de odometría visual, sobre el enfoque basado en características es *ORB-SLAM2: an Open-Source SLAM System for Monocular, Stereo and RGB-D Cameras* de Mur-Artal et al. (2017). En este trabajo, los autores presentan ORB-SLAM2, un sistema de localización y mapeo (SLAM) basado en características para cámaras monoculares, estereos y RGB-D.

ORB-SLAM2 utiliza el algoritmo ORB (Oriented FAST and Rotated BRIEF) para detectar y seguir características en las imágenes capturadas por la cámara. Luego, utiliza estas características para estimar la posición y orientación de la cámara y construir un mapa de la escena. Los autores demuestran que ORB-SLAM2 es capaz de realizar el SLAM de forma precisa y en tiempo real en diversos entornos y con diferentes tipos de cámaras.

3.1.2 Enfoque basado en apariencia

Este enfoque utiliza la apariencia general de las imágenes capturadas por la cámara del robot para estimar su posición. En lugar de depender de características específicas, este enfoque utiliza la información de toda la imagen para estimar el desplazamiento del robot.

Para llevar a cabo esta técnica, se pueden utilizar diversos algoritmos de correlación o de aprendizaje automático. Una vez que se ha comparado la apariencia de la imagen actual con la apariencia de la imagen anterior, se puede utilizar una transformación geométrica para determinar el desplazamiento del robot.

El enfoque basado en apariencia tiene varias ventajas. En primer lugar, es capaz de funcionar en entornos con pocas características distintivas, ya que utiliza toda la información de la imagen. Además, puede ser más preciso que el enfoque basado en características, ya que utiliza una mayor cantidad de información. Sin embargo, tiene algunas desventajas. Una de ellas es que puede ser más difícil de implementar y requerir más potencia de procesamiento. Además, puede ser menos robusto frente a cambios en la iluminación y en la apariencia del entorno.

Un trabajo de investigación interesante sobre el enfoque basado en apariencia es *DeepVO: Towards End-to-End Visual Odometry with Deep Recurrent Convolutional Neural Networks* de Wang et al. (2017). En este trabajo, los autores presentan una red neuronal recurrente con convolutional neural networks (CNNs) para realizar la odometría visual de forma end-to-end, utilizando únicamente imágenes de una cámara monocular como entrada.

La red neuronal recurrente se entrena para predecir el desplazamiento del robot entre dos imágenes consecutivas y utiliza una arquitectura de "encoder-decoder" para procesar las imágenes. Los autores demuestran que su enfoque es capaz de realizar la odometría visual de forma precisa en diversos entornos y supera a otros métodos basados en apariencia en términos de precisión y robustez.

3.1.3 Enfoque híbrido

Este enfoque combina tanto el enfoque basado en características como el enfoque basado en la apariencia para obtener una estimación más precisa de la posición del robot. [Scaramuzza y Fraundorfer, 2011] ; [Valiente García, Montiel y Dorado, 2012]

Para llevar a cabo este enfoque, se pueden utilizar diversos algoritmos de detección y seguimiento de características, así como técnicas de correlación o de aprendizaje automático para comparar la apariencia de la imagen actual con la apariencia de la imagen anterior. Una vez que se han detectado y seguido las características y se ha comparado la apariencia de la imagen, se pueden utilizar técnicas de fusión para combinar ambas fuentes de información y obtener una estimación más precisa de la posición del robot.

El enfoque híbrido tiene varias ventajas. En primer lugar, combina la robustez del enfoque basado en características con la precisión del enfoque basado en apariencia, lo que permite obtener una estimación más precisa de la posición del robot. Además, es capaz de funcionar en entornos con pocas características distintivas, ya que utiliza toda la información de la imagen. Sin embargo, también tiene algunas desventajas. Una de ellas es que puede ser más difícil de implementar y requerir más potencia de procesamiento que el enfoque basado en características. Además, puede ser menos robusto frente a cambios en la iluminación y en la apariencia del entorno que el enfoque basado en características.

Un trabajo de investigación interesante sobre el enfoque híbrido es *Towards a Real-Time Visual Odometry using a Hybrid Approach* de Zou et al. (2018). En este trabajo, los autores presentan un enfoque híbrido que combina el seguimiento de características y la correlación de apariencia para realizar la odometría visual en tiempo real.

Para llevar a cabo el seguimiento de características, los autores utilizan el algoritmo de detección y seguimiento KLT (Kanade-Lucas-Tomasi). Para la correlación de apariencia, utilizan el algoritmo de correlación normalizada. Luego, fusionan los resultados obtenidos

mediante el seguimiento de características y la correlación de apariencia para obtener una estimación más precisa de la posición del robot. Los autores demuestran que su enfoque es capaz de realizar la odometría visual en tiempo real de forma precisa en diversos entornos.

Otro trabajo de investigación interesante sobre el enfoque híbrido es *A Hybrid Approach for Visual Odometry based on Monocular Visual SLAM* de Zhang et al. (2020). En este trabajo, los autores presentan un enfoque híbrido para realizar la odometría visual utilizando tanto características como apariencia.

Para llevar a cabo el seguimiento de características, los autores utilizan el algoritmo ORB-SLAM (Oriented FAST and Rotated BRIEF). Para la correlación de apariencia, utilizan una red neuronal convolutional. Luego, fusionan los resultados obtenidos mediante el seguimiento de características y la correlación de apariencia para obtener una estimación más precisa de la posición del robot. Los autores demuestran que su enfoque es capaz de realizar la odometría visual de forma precisa en diversos entornos y supera a otros métodos basados en características y apariencia en términos de precisión y robustez.

3.2 Algoritmos Fundamentales

A continuación, se presentarán algunos algoritmos utilizados en la odometría visual y que sirven como fundamento para calcular la matriz fundamental (F), que es una matriz de 3x3 que describe la relación entre los puntos de dos imágenes de una misma escena. Antes de entrar en detalle sobre estos algoritmos, se proporcionará una breve descripción del problema al que se enfrentan estos métodos (como se menciona en la referencia [11]). La matriz fundamental se utiliza a menudo en el procesamiento de imágenes y la robótica para estimar la posición y orientación de una cámara en relación con una escena dada.

$$F = \begin{pmatrix} f_{0,0} & f_{0,1} & f_{0,2} \\ f_{1,0} & f_{1,1} & f_{1,2} \\ f_{2,0} & f_{2,1} & f_{2,2} \end{pmatrix} \tag{3.1}$$

Siendo x un punto de la primera imagen, x' el mismo punto en la segunda imagen, F la matriz fundamental y lx la línea epipolar (ver Figura 3.1) que contiene el punto en la segunda imagen, se cumplen las siguientes ecuaciones.

$$lx'_i = Fx_i \tag{3.2}$$

$$lx_i = F^T x'_i \tag{3.3}$$

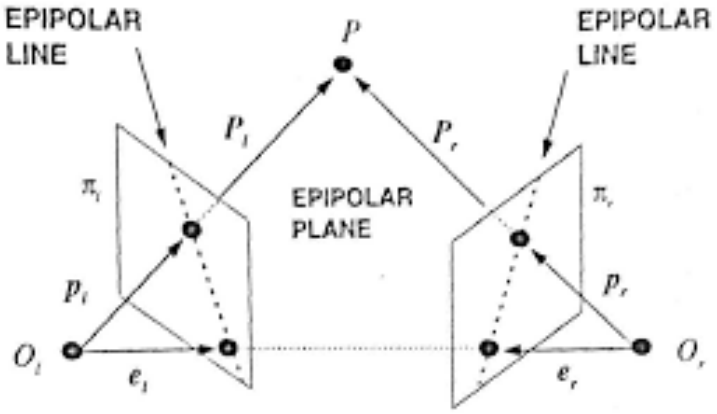


Figura 3.1: Relación entre los puntos de dos imágenes de una misma escena

Las características más importantes de la matriz fundamental son las siguientes [Faugeras, 1992] :

- La multiplicación del punto en la segunda imagen traspuesto por la matriz fundamental y por el punto en la primera imagen da como resultado 0.

$$x_i' F x_i = 0 \tag{3.4}$$

- El rango de la matriz fundamental es 2, lo que significa que tiene siete grados de libertad, pudiendo calcular dos elementos de la matriz a partir de los otros.
- El determinante es 0.

$$\det(F) = 0 \tag{3.5}$$

- La matriz fundamental es homogénea lo que quiere decir que está definida hasta un factor de escala.

Para calcular la matriz fundamental lo primero que hay que hacer es relacionar de alguna forma los puntos de una imagen con los de la otra, es decir realizar un *emparejamiento*, típicamente utilizando el algoritmo de emparejamiento SURF.

Una vez que se tienen los puntos emparejados simplemente hay que desarrollar las ecuaciones y resolver el sistema. La ecuación desarrollada para un punto, siendo i este punto, es la siguiente:

$$\begin{aligned} x_i' x_i f_{0,0} + x_i' y_i f_{0,1} + x_i' f_{0,2} + y_i' x_i f_{1,0} + \\ y_i' y_i f_{1,1} + y_i' f_{1,2} + x_i f_{2,0} + y_i f_{2,1} + f_{2,2} = 0 \end{aligned} \tag{3.6}$$

El sistema se puede resolver de una forma mucho más sencilla si se expresa mediante matrices y se utiliza SVD (*Singular Value Descomposition* por sus siglas en inglés). De esta forma el sistema quedaría definido mediante la siguiente ecuación.

$$Af = 0 \tag{3.7}$$

siendo A una matriz de dimensiones Nx9, cuyas filas se definen así:

$$(x_i'x_i, x_i'y_i, x_i', y_i'x_i, y_i'y_i, y_i', x_i, y_i, 1)$$

y f un vector columna definido por los coeficientes de F:

$$(f_{0,0}, f_{0,1}, f_{0,2}, f_{1,0}, f_{1,1}, f_{1,2}, f_{2,0}, f_{2,1}, f_{2,2})^T$$

Dadas las características de la matriz fundamental no es necesario desarrollar un sistema de nueve ecuaciones, sino que pueden seguirse algunas estrategias para utilizar menos puntos. A continuación se describen los dos algoritmos más utilizados para resolver el sistema de ecuaciones.

3.2.1 Ocho Puntos

Dado que la matriz F sólo está definida hasta un desconocido factor de escala, se impone una restricción adicional sobre la norma de F para eliminar el factor de escala, que consiste en que la norma cuadrática de F sea la unidad:

$$||F||^2 = ||f||^2 = f^t f = 1 \tag{3.8}$$

Para que el sistema de ecuaciones (3.7) admita una solución diferente del vector nulo, la matriz A debería tener un rango máximo de 8. Por lo tanto, el sistema de ecuaciones (3.7) puede ser resuelto mediante la correspondencia entre al menos 8 puntos diferentes en ambas cámaras.

3.2.2 Siete Puntos

Este algoritmo utiliza la restricción del rango de la matriz fundamental. Dado que el rango de esta matriz es 2, se puede introducir una ecuación más que describe la relación entre dos filas de la matriz. De esta forma se pueden calcular dos términos a partir del resto, teniendo que introducir únicamente 7 puntos en las ecuaciones.

Además de estos algoritmos hay otros dos derivados de estos que utilizan sistemas de optimización para encontrar la matriz que mejor se ajusta a los puntos, eliminando a la vez los posibles outliers: RANSAC y LMEDS.

3.2.3 RANSAC

RANSAC, (Fischler y Bolles, 1981)[6], es un algoritmo iterativo en el que en cada iteración se cogen 8 puntos y se calcula la matriz fundamental. Una vez calculada se comprueba qué tal se ajustan el resto de puntos a esta matriz. Si el ajuste es malo la matriz se desecha y si es buena se coge como mejor solución hasta encontrar una mejor o hasta que acabe el algoritmo. El pseudocódigo del método está representado en 1

Algorithm 1 Pseudocódigo de RANSAC

1: Seleccionar aleatoriamente el número mínimo de puntos necesarios para determinar los parámetros del modelo (*inliers*).
2: Resolver para los parámetros del modelo.
3: Determinar cuántos puntos del conjunto de todos los puntos encajan con una tolerancia predefinida E (*conjunto de consenso*).
4: Si la fracción del número de valores inliers sobre el número total de puntos en el conjunto excede un umbral predefinido K, volver a estimar los parámetros del modelo usando todos los valores inliers identificados y terminar.
5: De lo contrario, repita los pasos 1 a 4 (máximo de N veces).

3.2.4 LMEDS

Least Median of Squares (Rousseeuw, 1984)[15] es un algoritmo iterativo que trata de encontrar una matriz fundamental óptima al conjunto de datos dado. Para ello, intenta minimizar el error cuadrático medio entre el modelo (la matriz fundamental calculada en la iteración anterior) y los puntos, mediante el algoritmo de descenso del gradiente de tal forma que según pasan las iteraciones la matriz fundamental calculada se ajusta mejor a los puntos dados.

Este método es muy resistente a las coincidencias falsas, así como a los valores atípicos debido a una mala localización. Sin embargo, es muy difícil definir el algoritmo completo para la mínima mediana de los cuadrados como una fórmula matemática para un conjunto de datos completo, por lo que un algoritmo que genera una solución

$$\min \underset{i}{med}\, r_i^2 \tag{3.9}$$

de la mínima mediana de los cuadrados para un conjunto de datos que se da a continuación:

La estimación de la mínima mediana de los cuadrados es resistente a los valores atípicos debido a su alto valor de descomposición del 50%. Esta es la fracción de valores atípicos

Algorithm 2 Pseudocódigo de LMEDS

1: Se elije m conjuntos aleatorios de puntos con tamaño p del conjunto de datos donde p es el número de parámetros en la ecuación que se está resolviendo.
2: para cada subconjunto se utiliza un método como el de la mínima media de los cuadrados para encontrar una solución para los parámetros de ese conjunto de datos.
3: para cada conjunto de parámetros p obtenidos, se calcula la mediana de los cuadrados de los residuos M con respecto a todo el conjunto de datos:

$$M_J = \underset{i=1,\ldots,n}{med} r_i^2(p_J, m_i) \tag{3.10}$$

4: La solución es p_J para la cual M_J es mínimo entre todos los m M_J .

que se pueden tolerar y al mismo tiempo arrojar una buena solución. Sin embargo, el alto valor de ruptura significa que LMEDS no se adapta bien al ruido gaussiano.

Figura 3.2: Resultado de aplicar LMEDS para generar líneas epipolares

3.3 Conjuntos de Datos

Los datasets de odometría visual son conjuntos de datos que se utilizan para entrenar y evaluar algoritmos de odometría visual, que son sistemas que estiman la posición y orientación de una cámara o de un robot móvil a partir de imágenes o vídeos. Estos sistemas son muy útiles para la navegación autónoma de robots, la realización de mapas en tiempo real y la localización de robots en entornos dinámicos.

Los datasets de odometría visual suelen incluir imágenes o vídeos capturados por una cámara móvil, junto con información de posición y orientación precisa de la cámara en cada momento. Esta información se utiliza como referencia para evaluar la precisión de los algoritmos de odometría visual. Algunos ejemplos de datasets de odometría visual populares son:

- KITTI: Este dataset incluye imágenes y vídeos de un coche equipado con una cámara monocular y un lidar, así como información de posición y orientación precisa obtenida a partir de un sistema de navegación inercial y GPS.
- EuRoC: Este dataset incluye imágenes y vídeos capturados por una cámara monocular y un lidar montados en un robot móvil, junto con información de posición y orientación precisa obtenida a partir de un sistema de navegación inercial y una cinta transportadora.
- TUM-RGBD: Este dataset incluye imágenes y vídeos capturados por una cámara RGB y un sensor de profundidad, junto con información de posición y orientación precisa obtenida a partir de un sistema de navegación inercial y una cinta transportadora.

Los datasets de odometría visual son muy útiles para la investigación y el desarrollo de algoritmos de odometría visual, ya que proporcionan una forma de evaluar la precisión y el rendimiento de estos sistemas de manera objetiva. Además, también pueden ser útiles para la validación de sistemas de navegación autónoma en entornos reales.

3.3.1 KITTI

KITTI (Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago) [7] es un conjunto de datos de imagen y láser para la investigación en visión por computadora y aprendizaje automático. Fue creado por el Karlsruhe Institute of Technology y el Toyota Technological Institute en Chicago para promover la investigación en tecnologías de percepción y localización para sistemas de conducción automatizada.

El dataset KITTI incluye imágenes de cámaras de alta resolución y mediciones de escaneo láser para varias tareas de percepción, incluyendo la detección de obstáculos y la estimación de la profundidad. Las imágenes y las mediciones de escaneo láser se capturaron a bordo de un vehículo en movimiento en diferentes escenarios urbanos y rurales.

El dataset KITTI está dividido en diferentes conjuntos de datos, cada uno de ellos destinado a una tarea específica de percepción. Por ejemplo, el conjunto de datos "object" incluye imágenes y mediciones de escaneo láser para la detección de obstáculos y el conjunto de datos "odometry" incluye imágenes y mediciones de escaneo láser para la estimación de la posición y orientación del vehículo.

El dataset KITTI es ampliamente utilizado en la investigación y el desarrollo de sistemas de conducción automatizada y ha sido utilizado en muchos trabajos de investigación publicados en revistas y conferencias importantes. Si estás interesado en utilizar el dataset KITTI para tus propias investigaciones o proyectos, puedes descargarlo gratuitamente desde el sitio web del proyecto.

3.3.2 EuRoC MAV

EuRoC MAV (Micro Aerial Vehicle) es un conjunto de datos de vuelo de drones que fue recopilado por el European Robotics Challenge (EuRoC) para ser utilizado en investigaciones y desarrollos relacionados con la visión por computadora y la navegación autónoma de drones. El dataset incluye imágenes y datos de sensor de varios drones volando en entornos interiores y exteriores, así como también información de posición y orientación precisas obtenidas mediante un sistema de navegación inercial y un láser.

El dataset EuRoC MAV se compone de dos conjuntos de datos: el conjunto de datos MH_01 y el conjunto de datos MH_05. El conjunto de datos MH_01 fue recopilado en un edificio de oficinas y consiste en 11 vuelos realizados con un drone volando a través de pasillos y habitaciones. El conjunto de datos MH_05 fue recopilado en un edificio de oficinas y en una fábrica, y consiste en 28 vuelos realizados con un drone volando a través de pasillos, habitaciones y patios.

EuRoC MAV es ampliamente utilizado en investigaciones y desarrollos relacionados con la visión por computadora y la navegación autónoma de drones. Por ejemplo, se ha utilizado para evaluar el rendimiento de algoritmos de localización y mapeo de drones, así como para desarrollar y mejorar sistemas de navegación autónoma para drones. Además, el dataset EuRoC MAV ha sido utilizado en la organización de varios desafíos y concursos relacionados con la navegación autónoma de drones.

EuRoC MAV está disponible para su descarga en formato ROS (Robot Operating System).

3.3.3 VKITTI2

VKITTI2 (Virtual KITTI 2) es un dataset de imágenes y vídeos de alta calidad que se ha creado para simular una gran variedad de escenas de conducción en condiciones climáticas y de iluminación variadas. Fue diseñado para ser utilizado en la investigación y el desarrollo de sistemas de visión por computadora y de aprendizaje automático que se utilizan en la robótica móvil y la inteligencia artificial.

El dataset VKITTI2 está compuesto por más de 10.000 imágenes y vídeos, tomados con cámaras de alta definición y un LIDAR 3D. Incluye una amplia variedad de escenas de conducción, desde carreteras de ciudad hasta carreteras de montaña, y cubre diferentes condiciones climáticas, como lluvia, nieve y sol. Además, el dataset incluye imágenes y vídeos tomados en diferentes horas del día y en diferentes condiciones de iluminación, lo que lo hace ideal para el desarrollo y el entrenamiento de sistemas de visión por computadora y de aprendizaje automático que deben funcionar en entornos reales y variados.

El dataset VKITTI2 es una herramienta esencial para la investigación y el desarrollo en el campo de la robótica móvil y la inteligencia artificial, ya que ofrece una gran cantidad de imágenes y vídeos de alta calidad tomados desde un vehículo autónomo en entornos urbanos y suburbanos con una resolución de 1242x375 pixels. Además, incluye información precisa de ubicación y movimiento de la cámara y del vehículo, así como anotaciones detalladas de los objetos presentes en cada imagen o vídeo. Esta información es muy útil para la investigación en áreas como la detección y localización de objetos, la segmentación de imágenes y la percepción de movimiento. También se proporciona información de sensores de velocidad, aceleración y giroscopio, así como datos de localización y orientación precisos en tiempo real.

En resumen, VKITTI2 es una herramienta valiosa para la investigación y el desarrollo en el campo de la robótica móvil y la inteligencia artificial, ya que ofrece una gran cantidad de imágenes y vídeos de alta calidad y anotaciones detalladas que pueden ser utilizadas para entrenar y evaluar sistemas de visión por computadora y de aprendizaje automático.

3.4 Trabajos de Evaluación en Visual SLAM

3.4.1 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 *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.

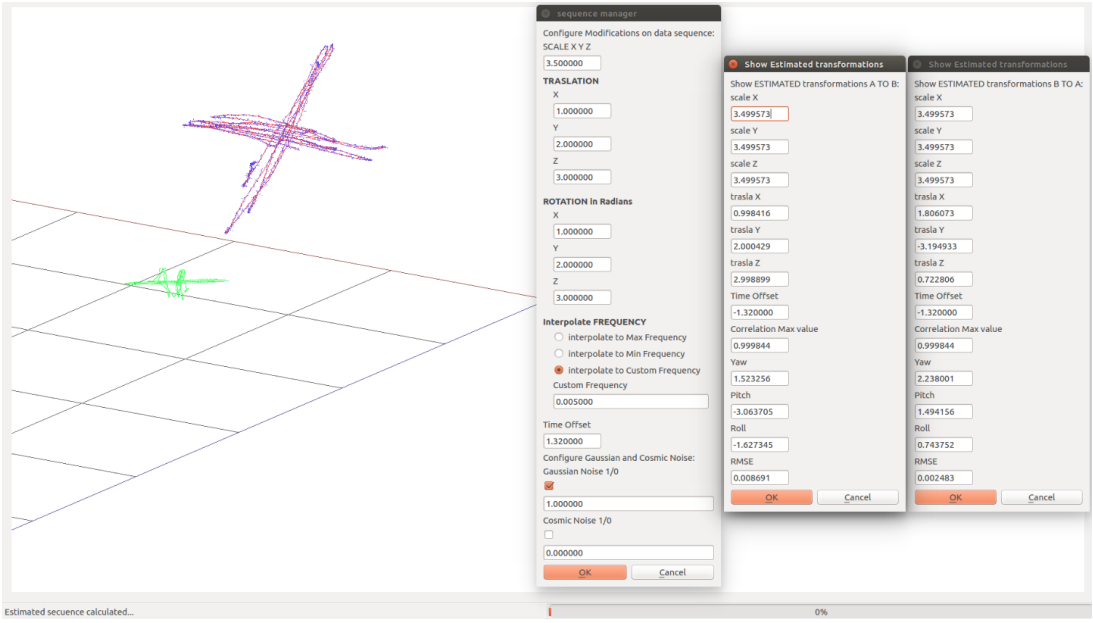


Figura 3.3: SLAMTestbed: Resultados de la estimación de un cambio de escala y traslación, rotación, offset y ruido gaussiano simultáneos.

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.

3.5 Trabajos de Formación en Robótica

3.5.1 TheConstruct

TheConstruct [1] 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.

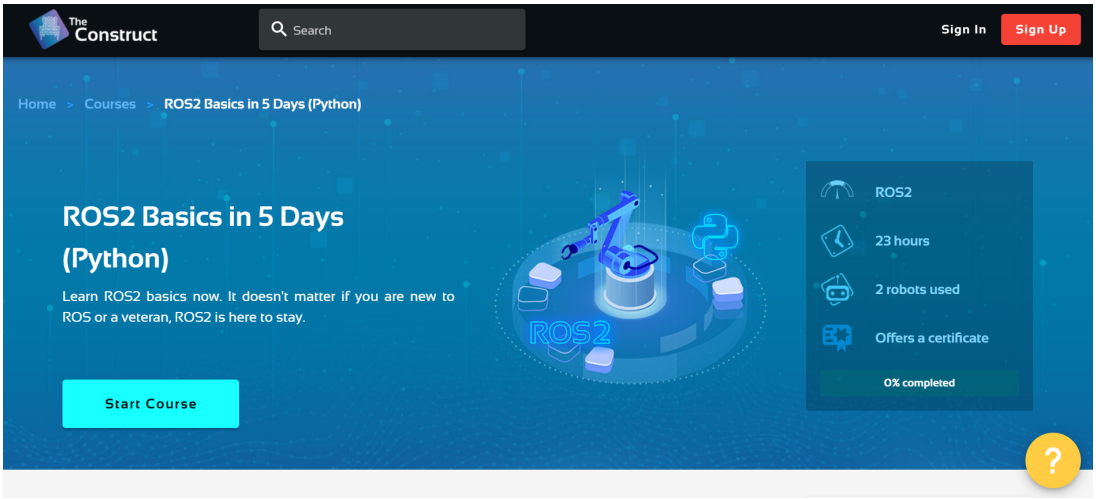


Figura 3.4: TheConstruct

3.5.2 Riders.ai

Riders.ai[2] 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

[1] https://www.theconstructsim.com/
[2] https://riders.ai/

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.

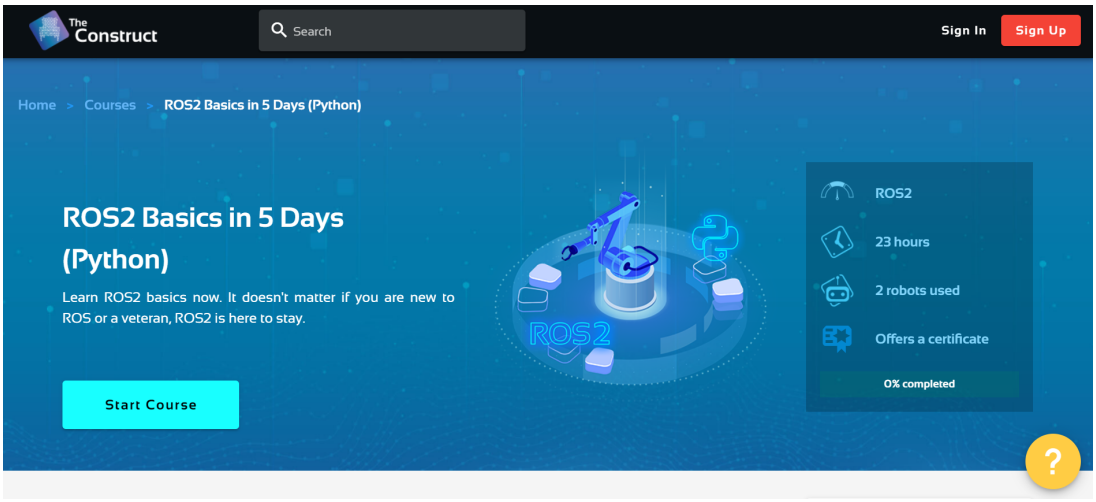


Figura 3.5: Riders.ai

Capítulo 4

Algoritmo de Odometría Visual 3D

Este capítulo presenta un algoritmo de odometría visual 3D desarrollado con una intención educativa, con el objetivo de proporcionar un ejemplo práctico y comprensible de cómo funciona esta técnica de navegación en robots móviles y vehículos autónomos. El algoritmo se basa en la utilización de técnicas de detección y seguimiento de características 3D, así como en la integración de información de diferentes sensores para aumentar la confiabilidad de la estimación de la posición y orientación. Se ha diseñado de manera clara y sencilla para facilitar su uso como herramienta didáctica en el ámbito de la odometría visual y la navegación de robots.

El algoritmo de odometría visual 3D se basa en la detección y seguimiento de características 3D en secuencias de imágenes capturadas por una cámara. La idea principal es estimar la pose (posición y orientación) del robot en cada instante de tiempo a partir de la observación de cómo cambian las características 3D en las imágenes.

La pose del robot en cada instante de tiempo se puede expresar mediante una transformación homogénea $\mathbf{T}_t = \begin{bmatrix} \mathbf{R}_t & \mathbf{t}_t \\ \mathbf{0}^T & 1 \end{bmatrix}$, donde $\mathbf{R}_t$ es la matriz de rotación y $\mathbf{t}_t$ es el vector de translación que representan la orientación y posición del robot, respectivamente.

La posición del robot en cada instante de tiempo se puede obtener a partir del vector de translación $\mathbf{t}_t = \begin{bmatrix} x_t & y_t & z_t \end{bmatrix}^T$.

Para calcular la pose del robot en el instante de tiempo $t + 1$ a partir de la pose en el instante de tiempo t, se utiliza la siguiente ecuación:

$$\mathbf{T}_{t+1} = \mathbf{T}_t \cdot \Delta\mathbf{T}_{t,t+1} \tag{4.1}$$

donde $\Delta\mathbf{T}_{t,t+1} = \begin{bmatrix} \mathbf{R}_{t,t+1} & \mathbf{t}_{t,t+1} \\ \mathbf{0}^T & 1 \end{bmatrix}$ es la transformación que representa el cambio

de pose entre el instante de tiempo t y el $t+1$.

Para obtener la posición absoluta del robot en cada instante de tiempo, es necesario llevar la pose del robot a un sistema de referencia fijo. Esto se puede hacer mediante la concatenación de las poses del robot en todos los instantes de tiempo desde el inicio hasta el instante de tiempo actual.

La pose absoluta del robot en el instante de tiempo t, $\mathbf{T}_t^{abs}$, se puede calcular mediante la siguiente ecuación:

$$\mathbf{T}_t^{abs} = \mathbf{T}_0^{abs} \cdot \prod_{i=1}^{t} \mathbf{T}_i \tag{4.2}$$

donde $\mathbf{T}_0^{abs}$ es la pose absoluta del robot en el instante de tiempo inicial, que se puede establecer de forma arbitraria, y $\mathbf{T}_i$ es la pose del robot en cada uno de los instantes de tiempo desde el inicio hasta el instante de tiempo actual.

La posición absoluta del robot en el instante de tiempo t se puede obtener a partir del vector de translación $\mathbf{t}_t^{abs} = \begin{bmatrix} x_t^{abs} & y_t^{abs} & z_t^{abs} \end{bmatrix}^T$ de la pose absoluta $\mathbf{T}_t^{abs}$.

Otra forma de calcular la posición absoluta del robot de manera incremental es utilizando la siguiente ecuación:

$$\mathbf{t}_{t+1}^{abs} = \mathbf{t}_t^{abs} + \mathbf{R}_t^{abs} \cdot \mathbf{t}_{t,t+1} \tag{4.3}$$

donde $\mathbf{t}_{t+1}^{abs}$ es el vector de translación de la pose absoluta del robot en el instante de tiempo $t+1$, $\mathbf{t}_t^{abs}$ es el vector de translación de la pose absoluta del robot en el instante de tiempo t, $\mathbf{R}_t^{abs}$ es la matriz de rotación de la orientación absoluta del robot en el instante de tiempo t y $\mathbf{t}_{t,t+1}$ es el vector de translación de la pose del robot entre el instante de tiempo t y el $t+1$.

Esta ecuación se basa en la idea de que la posición absoluta del robot en el instante de tiempo $t+1$ es igual a la posición absoluta del robot en el instante de tiempo t más la translación del robot entre el instante de tiempo t y el $t+1$, expresada en el sistema de referencia fijo. La translación entre el instante de tiempo t y el $t+1$ se puede calcular a partir de la observación de cómo cambian las características 3D en las imágenes.

La orientación del robot en cada instante de tiempo se puede obtener a partir de la matriz de rotación $\mathbf{R}_t$ de la pose del robot en ese instante de tiempo, que se puede expresar como una combinación de rotaciones en torno a los ejes x, y y z.

Para obtener la orientación del robot en términos de ángulos de Euler, se pueden utilizar las siguientes ecuaciones:

$$\alpha = atan2(R_{3,2}, R_{3,3}) \quad \beta = asin(-R_{3,1}) \quad \gamma = atan2(R_{2,1}, R_{1,1}) \tag{4.4}$$

donde α, β y γ son los ángulos de Euler en torno a los ejes x, y y z, respectivamente, y $R_{i,j}$ es el elemento de la matriz de rotación $\mathbf{R}_t$ en la fila i y la columna j.

De forma análoga a la ecuación 4.2, la orientación absoluta del robot en el instante de tiempo t se puede obtener a partir de la matriz de rotación $\mathbf{R}_t^{abs}$ de la pose absoluta del robot en ese instante de tiempo, que se puede calcular mediante la siguiente ecuación:

$$\mathbf{R}_t^{abs} = \mathbf{R}_0^{abs} \cdot \prod_{i=1}^{t} \mathbf{R}_i \tag{4.5}$$

donde $\mathbf{R}_0^{abs}$ es la matriz de rotación de la orientación absoluta del robot en el instante de tiempo inicial, que se puede establecer de forma arbitraria, y $\mathbf{R}_i$ es la matriz de rotación de la orientación del robot en cada uno de los instantes de tiempo desde el inicio hasta el instante de tiempo actual.

Asimismo, para calcular la orientación absoluta de forma incremental se puede utilizar la siguiente ecuación:

$$\mathbf{R}_{t+1}^{abs} = \mathbf{R}_t^{abs} \cdot \mathbf{R}_{t,t+1} \tag{4.6}$$

donde $\mathbf{R}_{t+1}^{abs}$ es la matriz de rotación de la orientación absoluta del robot en el instante de tiempo $t+1$, $\mathbf{R}_t^{abs}$ es la matriz de rotación de la orientación absoluta del robot en el instante de tiempo t y $\mathbf{R}_{t,t+1}$ es la matriz de rotación de la orientación del robot entre el instante de tiempo t y el $t+1$.

Una vez obtenida la matriz de rotación $\mathbf{R}_t^{abs}$, se puede utilizar alguna de las ecuaciones que se mencionan en 4.4 para obtener la orientación absoluta del robot en términos de ángulos de Euler.

Se pueden obtener el vector $\mathbf{t}_{t,t+1}$ y la matriz $\mathbf{R}_{t,t+1}$ haciendo uso de la matriz esencial.

La matriz esencial es una matriz de 3x3 que se utiliza para relacionar dos poses de un sistema de referencia de una cámara. Esta matriz se puede utilizar para obtener el vector de translación y la matriz de rotación que relacionan ambas poses.

Para obtener el vector de translación, se puede utilizar la siguiente ecuación:

$$\mathbf{t} = \frac{1}{s} \begin{bmatrix} E_{3,2}E_{2,1} - E_{3,1}E_{2,2} \\ E_{3,0}E_{2,2} - E_{3,2}E_{2,0} \\ E_{3,1}E_{2,0} - E_{3,0}E_{2,1} \end{bmatrix}$$

donde $\mathbf{t}$ es el vector de translación, E es la matriz esencial y s es una escala que se puede establecer de forma arbitraria.

Para obtener la matriz de rotación, se pueden utilizar las siguientes ecuaciones:

$$\mathbf{R} = \frac{1}{s} \begin{bmatrix} E_{2,2}E_{1,1} - E_{2,1}E_{1,2} & E_{2,1}E_{1,0} - E_{2,0}E_{1,1} & E_{2,0}E_{1,2} - E_{2,2}E_{1,0} \\ E_{3,2}E_{1,1} - E_{3,1}E_{1,2} & E_{3,1}E_{1,0} - E_{3,0}E_{1,1} & E_{3,0}E_{1,2} - E_{3,2}E_{1,0} \\ E_{3,1}E_{2,2} - E_{3,2}E_{2,1} & E_{3,2}E_{2,0} - E_{3,0}E_{2,2} & E_{3,0}E_{2,1} - E_{3,1}E_{2,0} \end{bmatrix}$$

Para calcular la matriz esencial, se necesitan las correspondencias entre las características 3D en las dos poses. Una vez que se tienen las correspondencias, se pueden utilizar técnicas de optimización para encontrar la matriz esencial que mejor se ajusta a las correspondencias.

En nuestro caso, utilizaremos el algoritmo RANSAC para calcular la matriz esencial. Este algoritmo se describe en detalle en el capítulo 3.2.3. Según lo explicado en el capítulo 6 de la referencia [10], RANSAC ofrece un buen equilibrio entre velocidad de ejecución y precisión en la estimación de la matriz esencial. Por esta razón, consideramos que es una buena opción para nuestro caso en particular.

A continuación se explicará el código que se ha sido necesario desarrollar antes de llevar a cavo la integración en Unibotics.

4.1 Desarrollo previo a la integración en Unibotics

Es importante explicar la estructura de carpetas del proyecto ya que permite entender la organización y el flujo del código y cómo está dividido en distintas partes y módulos. Además, al describir la estructura de carpetas se puede hacer una introducción a los diferentes archivos y scripts incluidos en el proyecto y su función específica.

También, se presentan los pseudocódigos del backend y el frontend, que muestran de manera resumida y esquemática el flujo de ejecución del código y cómo interactúan entre sí las distintas partes del proyecto. Esto ayudará a comprender cómo se han implementado las

distintas funcionalidades del proyecto y cómo se han integrado en la plataforma educativa.[1]

4.1.1 Estructura de carpetas

La estructura de carpetas consta de dos carpetas principales: *backend* y *frontend*. La carpeta *backend* contiene el código del servidor y todos los módulos necesarios para ejecutar el servidor y el algoritmo de odometría visual 3D. Dentro de esta carpeta se encuentra el archivo *server_app.py*, que es el punto de entrada para el servidor, y el archivo *requirements.txt*, que especifica las dependencias necesarias para ejecutar el servidor. También se incluyen varios scripts en Bash en la raíz de la carpeta *backend* para facilitar la instalación y ejecución del servidor.

La carpeta *backend/src* contiene todos los módulos Python necesarios para ejecutar el algoritmo de odometría visual 3D. Dentro de esta carpeta se encuentran varias subcarpetas, como *maths*, *scales* y *utils*, que almacenan módulos relacionados con las operaciones matemáticas necesarias para el algoritmo, el procesamiento de datos de escala y tiempo y utilidades diversas, respectivamente. La carpeta *test* contiene scripts de prueba para los módulos de la carpeta *src*, concretamente para comprobar que la transformación entre matrices de rotación y ángulos de Euler y viceversa sea correcta.

La carpeta *frontend* contiene el código del cliente, es decir, la interfaz web que se utiliza para visualizar los resultados del algoritmo de odometría visual 3D. Dentro de esta carpeta se encuentran varios archivos y carpetas, como *index.html*, que es la página principal de la interfaz, y *script.js*, que es el archivo principal de JavaScript que se encarga de la lógica de la interfaz. También se incluye un archivo *style.css* para definir el estilo de la interfaz y una carpeta *static* que almacena recursos estáticos, como texturas y gráficos, utilizados por la interfaz.

La carpeta *frontend/src* contiene los módulos JavaScript que se utilizan en la interfaz. Dentro de esta carpeta se encuentra una subcarpeta *3dScene* que almacena el código encargado de generar y controlar la escena 3D que se muestra en la interfaz, y una subcarpeta *vehicles* que almacena el código relacionado con los vehículos que se muestran en la escena. También se incluye una subcarpeta *utils* que almacena módulos de utilidad diversa.

```
.
├── backend
```

[1] El código relacionado con esta sección del trabajo puede encontrarse en el repositorio oficial del proyecto en GitHub, que se puede acceder a través del siguiente enlace: https://github.com/RoboticsLabURJC/2020-tfm-pablo-asensio. Además, si se desea obtener una mayor comprensión de la metodología de trabajo utilizada, se puede consultar el repositorio personal del proyecto en GitHub, que se encuentra disponible en el siguiente enlace: https://github.com/PabloAsensio/MonocularVisualOdometry.
Cave mencionar que dicho código es compatible con los tres datasets que se describen en la sección 3.3.

```
│   ├── app.py
│   ├── find_timestamp.py
│   ├── plot_groundtruth.py
│   ├── requirements.txt
│   ├── server_app.py
│   ├── setup.sh
│   ├── src
│   │   ├── maths
│   │   │   ├── cv_maths.py
│   │   │   ├── quaternions.py
│   │   │   ├── rotation_matrix.py
│   │   │   └── rotation_to_euler.py
│   │   ├── pinhole_camera.py
│   │   ├── scales
│   │   │   ├── find_timestamp.py
│   │   │   ├── grep.py
│   │   │   ├── read_eurocmav_timestamp_groundtruth.py
│   │   │   ├── scales.py
│   │   │   └── true_rotation.py
│   │   ├── utils
│   │   │   ├── load_images.py
│   │   │   └── messages.py
│   │   └── visual_odometry.py
│   └── test
│       └── maths
│           ├── test_rotation_matrix.py
│           └── test_rotation_to_euler.py
├── dataset.cfg
├── download-datasets.sh
├── frontend
│   ├── package-lock.json
│   ├── package.json
│   ├── src
│   │   ├── 3dScene
│   │   │   └── scene.js
│   │   ├── index.html
│   │   ├── script.js
│   │   ├── style.css
│   │   ├── utils
│   │   │   ├── CBuffer.js
│   │   │   └── corrector.js
│   │   └── vehicles
│   │       ├── car.js
```

```
│   │       └── frame.js
│   └── static
│       └── textures
└── run.sh
```

4.1.2 Pseudo códigos / códigos de backend

A continuación se presenta el pseudocódigo correspondiente a la parte de la aplicación que se ejecuta en el backend.

El archivo principal encargado de la lógica del servidor backend es *server_app.py*.

Algorithm 3 Pseudocódigo *server_app.py*

```
1: Leer el fichero de configuración
2: Pre-cargar las imágenes
3: while no sea la última imagen do
4:     Actualizar la pose del vehículo con la imagen actual
5:     Enviar el mensaje al frontend
6: end while
```

Donde el mensaje se compone de la imagen codificada en *base64* del instante t, y la pase calculada y la de referencia.

Sin embargo, el módulo encargado de implementar el algoritmo de odometría visual de forma efectiva es *visual_odometry.py*, ver pseudocódigo 4. Este archivo contiene la clase *VisualOdometry*, encargada de calcular la pose del vehículo en función del dataset con el que se esté trabajando. Cada dataset tiene su propio formato, por lo que es necesario tener en cuenta esta variable a la hora de realizar los cálculos. La clase *VisualOdometry* se encarga de gestionar todas las operaciones necesarias para llevar a cabo el algoritmo de odometría visual de forma efectiva.

Dentro de este pseudocódigo 4 se puede detallar de una mejor forma el proceso de actualización de la pose, ver pseudocódigo 5.

El funcionamiento de la parte del frontend es muy sencillo en este caso, simplemente ha de representar la imagen, y las poses en la escena.

Algorithm 4 Pseudocódigo método *Update* de la clase *VisualOdometry*

```
1: if tamaño de imagen es inválido then
2:     devolver error
3: end if
4: if es el primer frame then
5:     obtener puntos característicos de la imagen
6:     igualar posición actual a 0
7:     guardar matriz de rotación inicial
8: else
9:     if es el segundo frame then
10:        calcular puntos de interés
11:        no actualizar pose
12:    else
13:        calcular puntos de interés
14:        actualizar pose
15:    end if
16: end if
```

Algorithm 5 Pseudocódigo actualización pose

```
1: Realizar el seguimiento de los puntos de interés
2: utilizar findEssentialMat de OpenCV para calcular la matriz esencial
3: utilizar findEssentialMat de OpenCV para calcular la matriz esencial
4: utilizar recoverPose de OpenCV para obtener a partir de la matriz esencial, las matrices
   de rotación y el vector posición
5: calcular la escala
6: if la escala pasa un umbral then
7:     actualizar la posición absoluta con la formula incremental corregida por la escala
8:     actualizar la matriz de rotación con la formula incremental
9: end if
10: if los puntos de interés son menos que un umbral then
11:    calcular nuevos puntos de interés
12: end if
```

4.2 Test y validación

Se han llevado a cabo varios test para validar el correcto funcionamiento del algoritmo de odometría visual 3D. Estos test se han dividido en dos categorías: test de unidad y test de integración.

Los test de unidad se han enfocado en verificar el correcto funcionamiento de cada uno de los módulos y clases del algoritmo. Estos test se han escrito utilizando la librería `pytest` de Python y se han ejecutado mediante el comando `pytest`, creando un *workflow* de GitHub que permite correr estos test antes de realizar un *merge* en un *pull request*.

Por otro lado, los test de integración se han enfocado en verificar el correcto funcionamiento del algoritmo en su conjunto. Para ello, se han utilizado diferentes datasets de prueba, que incluyen imágenes y datos de posición y orientación obtenidos de forma real, y se han comparado los resultados obtenidos por el algoritmo con los datos de verdad.

Además, se ha llevado a cabo una validación experimental mediante el método visual. En este sentido, se ha comparado la trayectoria estimada por el algoritmo con la trayectoria real del vehículo obtenida a partir de los datos de ground truth. Aunque este método no es el más preciso, es suficiente para evaluar si el algoritmo es capaz de seguir de forma correcta la trayectoria del vehículo. Además, no es necesario que el algoritmo sea completamente preciso, sino que simplemente debe ser capaz de seguir de forma razonable la trayectoria del vehículo.

Capítulo 5

Integración en Unibotics

En este capítulo se describe el proceso de integración de la aplicación de odometría visual 3D en la plataforma educativa Unibotics. Se explica cómo se ha adaptado la aplicación para poder ser utilizada por los estudiantes de forma sencilla y cómo se ha integrado en la plataforma de manera que sea accesible desde ella. Además, se describe la experiencia de usuario final y se evalúa el impacto de la integración en el aprendizaje de los estudiantes.

Una vez desarrollado y validado el algoritmo de odometría visual 3D, el siguiente paso es integrarlo en la plataforma educativa Unibotics. Para ello, se ha llevado a cabo una serie de adaptaciones tanto del código del algoritmo como de la interfaz web que se encarga de visualizar los resultados.

La integración se ha llevado a cabo mediante la creación de un nuevo ejercicio dentro de la plataforma, en la que se proporciona un editor integrado para que los alumnos puedan escribir y ejecutar su propio algoritmo de odometría visual 3D a través de una API proporcionada en el enunciado del ejercicio.

Además, se ha añadido un campo en la interfaz de la actividad que ofrece una métrica de evaluación del algoritmo, basada en la distancia en el plano y absoluta entre la trayectoria estimada por el algoritmo y la trayectoria real del vehículo. Esta métrica permite a los alumnos evaluar el rendimiento de su algoritmo y compararlo con el de otros compañeros.

Con esta integración, se espera que los alumnos puedan poner en práctica los conocimientos adquiridos durante el curso y desarrollar sus propias soluciones a problemas reales de odometría visual 3D.

Antes de explicar cómo se realiza la creación de un nuevo ejercicio, se pasa a explicar la arquitectura de Unibotics.

5.1 Arquitectura de Unibotics

La plataforma Unibotics, al igual que Robotics Academy, cuenta con un servidor especialmente diseñado para ofrecer ejercicios de simulación y programación de robots, denominado Robotics Academy Docker Image (RADI). Este servidor permite el uso de varios ejercicios que utilizan Gazebo y STDR, y ofrece un puerto websocket (8765) y un protocolo de comunicación (Robotics Academy Manager Protocol, o RAMP) para solicitar e interactuar con estos ejercicios. Cada ejercicio abre 1, 2 o más websockets para interactuar específicamente con el ejercicio y recibir datos.

El RAMP incluye los siguientes comandos:

- *open* para iniciar un ejercicio especificado en el campo *ejercicio*
- *stop* para detener la simulación
- *resume* para reanudar la simulación
- *reset* para reiniciar la simulación
- *evaluate* para solicitar una evaluación del código enviado en el campo *código*
- *startgz* para abrir el visualizador GZClient
- *stopgz* para cerrar el visualizador GZClient
- *Ping* o *PingDone* para enviar mensajes Ping y comunicar que una orden ha sido ejecutada (después de los comandos resume, reset o stop)

Cada ejercicio está compuesto por un `exercise.html` y un `exercise.py`. El `exercise.py` se ejecuta dentro del RADI, mientras que el `exercise.html` proviene del navegador. Ambos se comunican a través de *websockets*: canales de comunicación bidireccionales que permiten la comunicación entre diferentes lenguajes de programación. Cada websocket de ejercicio (típicamente uno para la GUI y otro para el cerebro del robot) tiene su propio protocolo. Los primeros cinco caracteres se utilizan para identificar el tipo de mensaje.

Comunicación entre el backend y el frontend

La comunicación entre el backend y el frontend consta de estos elementos:

- Websocket manager: se encarga de solicitar los ejercicios y controlar la simulación
- Websocket code: interactúa con el cerebro del robot

- Websocket GUI: recibe datos del robot
- GZClient VNC: interactúa y visualiza la simulación
- Console VNC: muestra mensajes de depuración y salida.

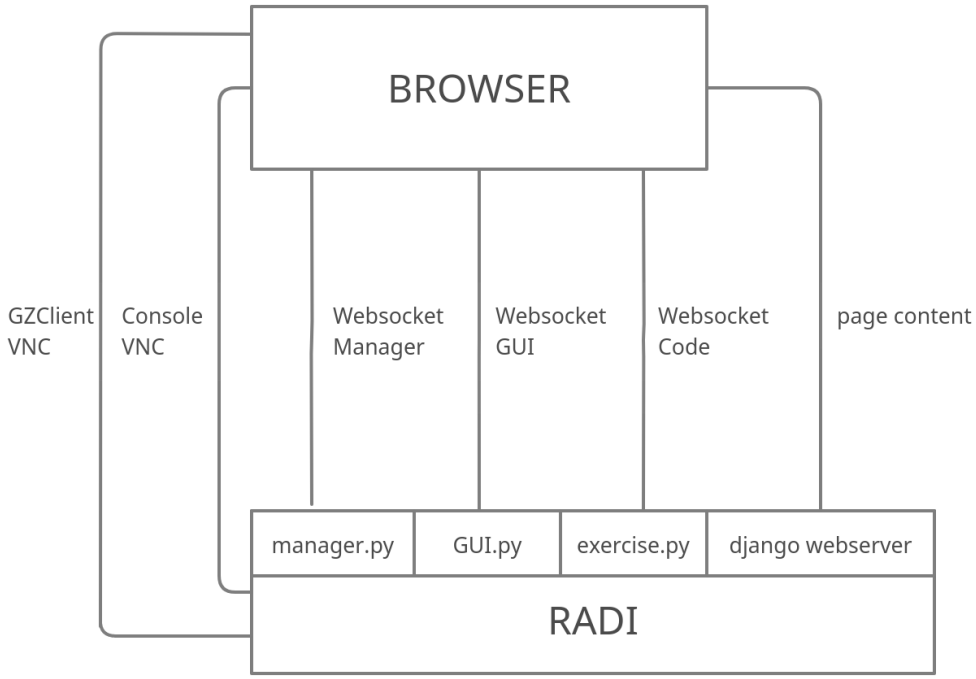


Figura 5.1: Comunicación entre el backend y el frontend

Protocolo entre exercise.py y el navegador

El protocolo de comunicación entre el archivo `exercise.py`, que se ejecuta en el servidor RADI, y el navegador web que utiliza el alumno para interactuar con el ejercicio, incluye dos websockets: el websocket de código, que se utiliza para enviar el código fuente del alumno al ejercicio y controlar su ejecución, y el websocket GUI, que se utiliza para enviar datos y imágenes desde el backend (`exercise.py`) al frontend (navegador web).

El websocket de código tiene un protocolo estandarizado que incluye los siguientes comandos:

- *#freq*: para establecer la frecuencia de ejecución del código del alumno y del GUI
- *#code*: para actualizar el código del alumno en el ejercicio
- *#play*: para iniciar la ejecución del código

- #*stop*: para detener la ejecución
- #*reset*: para detener y reiniciar la ejecución
- #*ping*: para enviar mensajes de ping

El servidor responde con los mensajes #*exec* cuando se ha cargado el último código enviado con #*code*, #*freq* en cada iteración del código con los campos *brain* (frecuencia del código del alumno), *gui* (frecuencia del GUI) y *rtf* (factor de tiempo real), y #*ping* como respuesta a los mensajes de ping.

El *websocket GUI*, por otro lado, se utiliza para enviar datos y imágenes desde el backend al frontend. Este websocket varía según las necesidades de cada ejercicio, pero suele incluir campos como #*image* con imágenes obtenidas de la cámara del robot y #*map* con la posición y rotación del robot.

Protocolo entre manager.py y el navegador

El websocket del manager se encarga de solicitar los ejercicios y controlar la simulación (en el puerto 6080, escuchando al servidor VNC en el puerto 5900). Por ejemplo, inicia/detiene/pausa/reanuda/mata la simulación de Gazebo del ejercicio elegido. También inicia el servidor VNC. El código escrito por el usuario se envía primero desde el navegador al proceso manager.py a través de los websockets del manager. Luego, manager.py comprueba el código con Pylint y devuelve el resultado al navegador. Puede manejar los siguientes comandos:

Open: Mata la simulación anterior y comienza una nueva dependiendo del ejercicio elegido y si la simulación acelerada está habilitada o no. *Resume*: Despausa la física de Gazebo. *Stop*: Pausa la física de Gazebo. *Evaluate*: Comprueba el código del usuario enviado a manager.py y devuelve una matriz vacía si no hay errores. *Evaluate_Style*: Comprueba el código del usuario enviado a manager.py y devuelve una matriz vacía si no hay errores. En este caso, también devuelve advertencias. *Start*: Despausa la física de Gazebo. *Reset*: Tipo de reinicio = predeterminado. Si el ejercicio está incluido en el array DRONE_EX, mata el exercise.py y reinicia el drone. Si el ejercicio está incluido en HARD_RESET_EX, se requiere tirar todo y volver a construirlo. *Soft reset*: Tipo de reinicio = suave. En este caso, ya sea que el ejercicio esté incluido en el array DRONE_EX o en el HARD_RESET_EX, el comportamiento es el mismo: se pausa y se reinicia la física. *Stopgz*: Detiene el cliente de Gazebo. *Startgz*: Configura el ancho y el alto de la pantalla del navegador para el gzclient. También inicia el cliente Gazebo.

Procesamiento de código del usuario

La procesamiento del código del usuario sigue los siguientes pasos:

1. El código es enviado desde el editor ACE del navegador al proceso manager.py del RADI.
2. El código es revisado por Pylint y los errores son devueltos al navegador. Si el navegador recibe un error, éste se muestra en un modal y el código no se envía al cerebro del robot.
3. Si no hay errores, el código es enviado desde el navegador al archivo exercise.py a través del websocket de código. Exercise.py recibe el código fuente del usuario como texto bruto y lo pone en funcionamiento.
4. El código del usuario es separado en dos partes: la parte secuencial (ejecutada una vez) y la parte iterativa (ejecutada cada intervalo de tiempo del cerebro). El código es separado por el primer bucle *while True* encontrado. Dentro de la parte iterativa, el cerebro mide el tiempo después de cada iteración, lo que se llama gestión del código, para no sobrecargar el CPU y mantener un ritmo controlado de iteraciones por segundo para dejar el CPU libre para otras tareas del navegador. La parte iterativa se inserta en otra plantilla junto con código adicional que controla las iteraciones por segundo que se realizan (esqueleto computacional), de modo que este motor computacional está vinculado al código del usuario para garantizar que el código se ejecute a una frecuencia nominal. El código del usuario también se enriquece con algunos elementos de control de ejecución para pausar, reiniciar y cargar un nuevo código en el cerebro del robot.

5.2 Creación de un Nuevo Ejercicio

Para incluir un nuevo ejercicio en la plataforma, sigue los siguientes pasos:

1. Añadir la carpeta con el contenido del ejercicio en `exercises/static/exercises` siguiendo las convenciones de nombres de archivos.
2. Crear una entrada en `db.sqlite3`. Una forma sencilla de hacerlo es a través de la página de Django admin:
   (a) Ejecutar `python3.8 manage.py runserver`.
   (b) Acceder a `http://127.0.0.1:8000/admin/` en un navegador y inicia sesión con "user" y "pass".

   (c) Hacer clic en "add exercise" y rellena los campos: id de ejercicio (nombre de la carpeta), nombre (nombre que se mostrará), estado, idioma y descripción (descripción que se mostrará). Guarda y sal.

3. Hacer commit de los cambios en `db.sqlite3`.

4. Editar el archivo `manager.py` e `instructions.json` si es necesario.

Para realizar la integración del ejercicio en la plataforma Unibotics, es necesario incluir los archivos *hal.py*, *gui.py*, `exercise.py` y `exercise.html` de la ruta *tu_ejercicio/webtemplate/* alojada en `exercises/static/exercises`. Estos archivos son necesarios para que el ejercicio se integre correctamente en la plataforma y pueda ser ejecutado por el alumno a través de la interfaz web.

Es aconsejable basarse en uno de los ejercicios existentes, como ha sido en nuestro caso *3d_reconstruction*, para facilitar la creación del ejercicio. De esta forma, podremos ver cómo están estructurados estos archivos y cómo debemos adaptarlos a nuestro ejercicio.

5.2.1 Archivos principales

hal.py

El archivo *hal.py* contiene la clase HAL, que es la encargada de gestionar los datos como imágenes, posiciones del ground truth, etc. Esta clase cuenta con variables privadas y métodos que pueden ser expuestos a la API del ejercicio. De esta forma, se controla qué información se le da al alumno y cómo puede utilizarla en su algoritmo. Por ejemplo, la clase HAL puede tener una variable privada que almacene el estado del vehículo, y un método que permita al alumno obtener la información de este estado a través de la API. De esta forma, el alumno puede utilizar la información del estado del vehículo en su algoritmo para calcular la odometría visual 3D.

gui.py

El archivo gui.py se encarga de la comunicación entre el backend del ejercicio y la interfaz web del alumno a través de un websocket. Esto se realiza mediante la clase GUI, que se encarga de inicializar y gestionar el websocket, y de enviar y recibir mensajes a través de él. Además, esta clase también se encarga de actualizar la interfaz web del alumno con la información que se envía desde el backend. Es importante tener en cuenta que toda la información que se envía o recibe a través del websocket debe estar en formato JSON.

exercise.py

El archivo `exercise.py` es el encargado de gestionar el servidor del ejercicio, es decir, se encarga de exponer la API del ejercicio a través de la cual el alumno podrá interactuar con el ejercicio. También se encarga de parsear el código del alumno, es decir, de comprender el código escrito por el alumno y hacer que sea ejecutable por el ejercicio.

Además, este archivo se encarga de definir la frecuencia de funcionamiento del ejercicio, es decir, la velocidad a la que se ejecutará el código del alumno. Esto es importante ya que en algunos casos el ejercicio puede consumir demasiados recursos de la máquina del alumno, y así se puede reducir dicho consumo de recursos.

En resumen, el archivo `exercise.py` es el encargado de gestionar el servidor del ejercicio y hacer que el código del alumno sea ejecutable de forma correcta.

exercise.html

El archivo `exercise.html` es el encargado de la parte del frontend que se muestra al alumno. Se encarga de generar la interfaz gráfica a través de la cual el alumno puede interactuar con el ejercicio. Además, este archivo se encarga de calcular el error que hay en el algoritmo del alumno y de realizar la conversión al sistema de puntos. El sistema de puntos es una medida de la calidad del algoritmo del alumno, y se consiguen mejores resultados cuanto menor sea la puntuación obtenida.

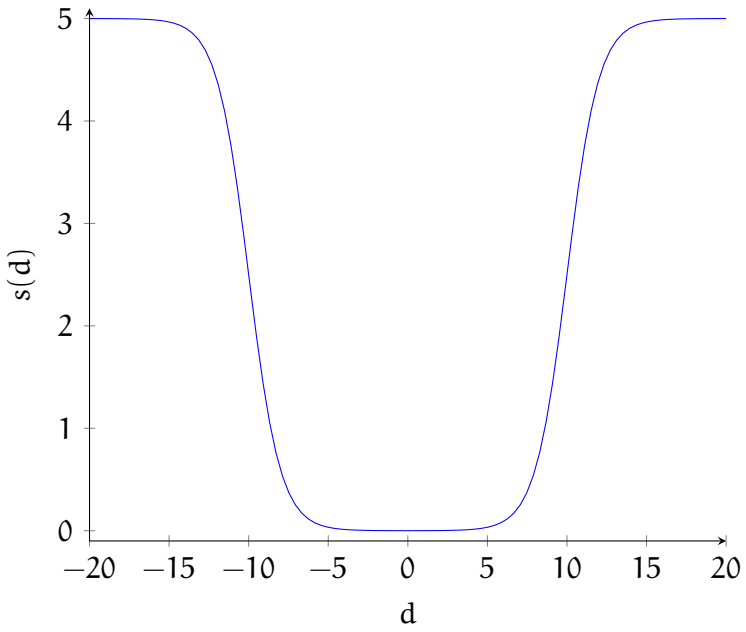
5.3 Sistema de corrección

El sistema de corrección utilizado en este trabajo se basa en la asignación de una puntuación al alumno en función de la distancia entre la posición estimada por el algoritmo del alumno y la posición real del vehículo, obtenida a partir de los datos de ground truth. Esta puntuación se utiliza para determinar el grado de acierto del algoritmo y proporcionar una retroalimentación al alumno sobre el rendimiento de su código. La puntuación se asigna mediante una función sigmoide que penaliza de forma más severa a medida que la distancia entre la posición estimada y la real aumenta. De esta forma, se busca fomentar la implementación de algoritmos precisos por parte del alumno.

Para una distancia d, la puntuación s se rige por la siguiente ecuación:

$$s(d) = \frac{5}{1 + e^{-(|d|-10)}} \tag{5.1}$$

Donde d es la distancia relativa entre la posición calculada y la real, y k es un parámetro que determina la pendiente de la función. Se ha elegido un valor de $k = 1$, lo que hace que la función tenga una pendiente bastante pronunciada cerca del valor de $d = 10$. De esta forma, se consigue que la puntuación caiga rápidamente a medida que la distancia d aumenta.



La corrección del algoritmo del alumno se realiza por cada fotograma, cada vez que se recibe el mensaje en el frontend desde el RADI. De esta forma, se puede evaluar en tiempo real el rendimiento del algoritmo y proporcionar una retroalimentación inmediata al alumno. Esto permite que el alumno pueda ir mejorando su algoritmo de forma progresiva y pueda obtener una mejor puntuación en el ejercicio.

Capítulo 6

Conclusiones y trabajos futuros

En el presente proyecto, se ha desarrollado un ejercicio de odometría visual para la plataforma educativa Unibotics. A través de este ejercicio, se ha pretendido ofrecer a los estudiantes una herramienta de aprendizaje práctica y didáctica, que les permita comprender y aplicar los conceptos teóricos de odometría visual de manera sencilla y amena.

Una vez finalizado el ejercicio, podemos concluir que se han cumplido los objetivos propuestos en su desarrollo. En primer lugar, se ha conseguido implementar correctamente el algoritmo de odometría visual en el ejercicio, permitiendo así a los estudiantes visualizar y entender el funcionamiento de dicho algoritmo de manera clara y concisa. Además, se ha conseguido integrar el ejercicio en la plataforma Unibotics de manera fluida y sencilla, permitiendo a los estudiantes acceder a él de forma rápida y cómoda.

Aunque durante el desarrollo del ejercicio se han encontrado algunos problemas, como la dificultad para sincronizar la odometría visual con la cámara del robot, estos han sido solucionados mediante la implementación de medidas de mitigación adecuadas.

En cuanto a los trabajos futuros, se podría considerar la posibilidad de ampliar el ejercicio con nuevas funcionalidades y opciones de configuración, con el objetivo de hacerlo aún más completo y atractivo para los estudiantes. También se podría explorar la posibilidad de integrar el ejercicio con otras tecnologías y plataformas, con el fin de expandir su alcance y beneficios para la comunidad educativa.

En resumen, la creación de este ejercicio de odometría visual en la plataforma Unibotics ha supuesto un éxito en la consecución de sus objetivos, y abre la puerta a futuros trabajos y mejoras que permitan seguir ofreciendo a los estudiantes una herramienta de aprendizaje de calidad y valor.

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] 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.

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

[5] Olivier D Faugeras. «What can be seen in three dimensions with an uncalibrated stereo rig?» En: *European conference on computer vision*. Springer. 1992, págs. 563-578.

[6] Martin A Fischler y Robert C Bolles. «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography». En: *Communications of the ACM* 24.6 (1981), págs. 381-395.

[7] Andreas Geiger, Philip Lenz y Raquel Urtasun. «Are we ready for Autonomous Driving? The KITTI Vision Benchmark Suite». En: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2012.

[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] 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.

[10] Ignacio San Román Lana. «Odometría visual 3D para autolocalización de una cámara móvil en tiempo real». https://gsyc.urjc.es/jmplaza/students/tfm-visualodometry-isanroman-2015.pdf. Tesis de mtría. Universidad Rey Juan Carlos, 2015.

[11] Quang-Tuan Luong y col. «On determining the fundamental matrix: Analysis of different methods and experimental results». Tesis doct. Inria, 1993.

[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 J. 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.

[14] 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.

[15] Peter J Rousseeuw. «Least median of squares regression». En: *Journal of the American statistical association* 79.388 (1984), págs. 871-880.

[16] D Scaramuzza y F Fraundorfer. «Visual odometry [Tutorial]». En: *IEEE Robotics & Automation Magazine* 18.4 (2011), págs. 80-92.

[17] David Valiente García, J M Montiel y J Dorado. «A review of visual odometry methods for mobile robots». En: *Sensors* 12.12 (2012), págs. 16093-16158.

[18] Sen Wang y col. «Deepvo: Towards end-to-end visual odometry with deep recurrent convolutional neural networks». En: *2017 IEEE international conference on robotics and automation (ICRA)*. IEEE. 2017, págs. 2043-2050.

[19] Y Zhang y col. «A Hybrid Approach for Visual Odometry based on Monocular Visual SLAM». En: *2020 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. IEEE. 2020, págs. 1764-1769.

[20] Q Zou, Y Chen e Y Gao. «Towards a Real-Time Visual Odometry using a Hybrid Approach». En: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, págs. 6373-6379.