uc3m | Universidad Carlos III de Madrid

Grado en Ingeniería de Sistemas Audiovisuales

2017-2018

Trabajo Fin de Grado

"UAV pose estimation based on visual information"

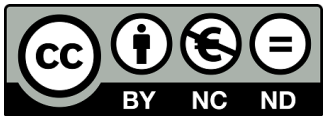
Rocío Sieiro Alfonsín

Tutor/es

Abdulla Hussein Abdulrahman Al Kaff

Francisco Miguel Moreno Olivo

10 de Junio de 2018, Leganés (Madrid)



Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento - No Comercial - Sin Obra Derivada**

Se hace difícil agradecer en pocas líneas a todas las personas que me han apoyado y me han dado las fuerzas necesarias para llegar hasta aquí.

En primer lugar me gustaría agradecerle a Adbdulla Al Kaff por darme la oportunidad de realizar este trabajo y a Francisco Moreno Olivo por su paciencia y disponibilidad en todo momento. Sin su ayuda este trabajo no habría salido adelante.

En segundo lugar, a mis compañeros de clase, en especial a Eva mi tercer brazo desde que llegué a Madrid y Alba, que en los tres últimos años se ha convertido en una gran amiga. Gracias por todas las veces que nos hemos "salvado el culo.[en]tre nosotras.

A mis amigos, los de aquí y los de allí, los que se han quedado atrás en mi vida y los que aún siguen en ella. A todos aquellos que han conseguido sacarme una sonrisa en esos momentos en los que uno lo necesita. Sara, Marta, SaraPé, Agus, María Arjona, Javi, Virgi, Juan, a mi querido Guille y a todos los demás. Gracias.

Si nunca hubiera empezado esta aventura nunca hubiese tenido la oportunidad de conocer a ese país tan maravilloso que es Brasil (al menos estos años) y con el llevarme a una de las personas más increibles que he conocido, María. A ti, gracias por aguantarme estos últimos años y ser mi familia en ese año loco por el extranjero.

No puedo dejar de mencionar también, a mi tío Javi, quién ha sido mi vínculo más cercano con mi tierra durante estos años. Gracias por esos cocidos de domingo y todas tus palabras que me han dado fuerzas para continuar.

Y para finalizar, gracias a mi hermana María y sobretodo a mis padres por haber sido exigentes conmigo y haberme enseñado que todo esfuerzo tiene su recompensa.

Os quiero.

RESUMEN

El trabajo presentado a continuación trata de optimizar el desarollo e implementación de un sistema de localización para vehículos autónomos, particularmente para VANTs (Vehículos Aéreos No Tripulados, del inglés *UAV, Unmanned Aerial Vehicle*) más comunmente conocidos como drones, basado en odometría visual monocular desarollado en el lenguaje de programación C++. Para ello, integraremos una arquitectura de cálculo paralelo conocida como CUDA (*Compute Unified Device Architecture*) que nos permitirá sustituir el procesamiento central en la CPU (*Central Processing Unit*) de nuestro computador por el coprocesamiento repartido entre la CPU y la GPU (*Graphics Processing Unit*), es decir, su tarjeta gráfica. Como veremos a lo largo del trabajo, el uso de unidades de procesamiento gráfico para resolución de problemas de tipo visual resulta muy interesante al dar unos resultados mejores que la ejecución del sistema solo sobre la CPU. Además, se presentará una alternativa de desarollo del algoritmo sustituyendo la técnica de procesamiento de imágenes utilizada para la detección de puntos característicos de la imagen, conocidos como *keypoints*. Una vez obtenidos los resultados bajo las dos soluciones presentadas, se hará una comparación para ver la eficacia de cada una y analizar sus ventajas a la hora de implementar en una aplicación de navegación autónoma real.

Palabras clave: odometría, visual, monocular, estimación, homografía, drone, orientación, CUDA, OpenCV, SIFT, SURF, procesamiento, digital, imágenes.

ABSTRACT

The following work that will be presented tries to optimize the development and implementation of the location system for an unmanned vehicle, specifically for UAVs (Unmanned Aerial Vehicles), which are commonly known as drones, based on molecular visual Odometry implemented in the programming language C++. For this, we will employ a parallel calculus architecture known as CUDA (Compute Unified Device Architecture) that will allow us to substitute the central processing done by the CPU (Central Processing Unit) of our computer, by a co-processing performed by the CPU and the GPU (Graphics Processing Unit), in other words, its graphic card. As we will observe along this work, the use of graphic process units for solving visual problems turns into a very interesting tool, since it gives better results than executing the system just with the CPU control. In addition, an algorithm development alternative was presented replacing the image processing technique used for the detection of characteristic points of the image, known as keypoints. Once obtained the results under the two presented solutions, a comparison was made for the effectiveness of each one and analyze its advantages for the time to implement in a real autonomous access application.

Key words: odometry, visual, monocular, estimation, pose, homography, drone, orientation, CUDA, OpenCV, SIFT, SURF, image processing, digital, images.

ACRÓNIMOS

GCS = Ground Control Station

GPU = Graphics Processing Unit

CPU = Central Processing Unit

CUDA = Compute Unified Device Architecture

UAV = Unmanned Aerial Vehicle

VANT = Vehículo Aéreo No Tripulado

INS = Inertial Navigation System

GPS = Global Positioning System

SNS = Sistema de Navegación por Satélite

GNSS = Global Navigation Satellite System

INS = Inertial Navigation System

IFF = Identification Friend or Foe

SLAM = Simultaneous Localization and Mapping

PTAM = Parallel Tracking and Mapping

VSLAM = Visual Simultaneous Localization and Mapping

MonoSLAM = Monocular Simultaneous Localization and Mapping

EASA = European Aviation Safety Agency

AESA = Agencia Estatal de Seguridad Aérea

SIFT = Scale-Invariant Feature Transform

FREAK = Fast Retina Keypoint

RPA = Remotely Piloted Aircrafts

VLOS = Visual Line of Sight

EVLOS = Extended Visual Line of Sight

BVLOS = Beyond Visual Line of Sight

MTOW = Maximum TakeOff Weight

FVP = First Person View

OpenCL = Open Computing Language

OpenCV = Open Source Computer Vision Library

PDI = Processing Digital Image

FPS = Frames Per Second

JIT = Just In Time

RANSAC = Random Sample Consensus

Índice

Índice de figuras

Índice de Tablas

1. INTRODUCCIÓN

Desde la aparición de la robótica ha resultado muy interesante la búsqueda de la autonomía de cualquier tipo de vehículos, tanto tripulados como no. Es decir, conseguir la máxima navegación autónoma posible de un vehículo en un entorno totalmente desconocido.

Resulta realmente interesante cuando trabajamos con UAVs (*Unmanned Aerial Vehicles*) o en español VANTs (Vehículos Aéreos No Tripulados), más conocidos coloquialmente como "drones", pues han permitido la aparición de múltiples aplicaciones de uso diario como es la fotografía aérea u otras aplicaciones más especializadas como la monotorización del tráfico o exploración de ciertas áreas geográficas. Los sistemas de navegación usados por este tipo de vechículos pueden ser manejados con piloto, desde una torre de control terrestre o sin piloto. La última tendencia debido a las necesidades de obtener cada vez drones de menor tamaño, resulta en la automatización del vuelo usando ciertos sensores incorporados al vehículo. A lo largo de los años, se han presentado múltiples soluciones a este problema desde diferentes persperctivas, estudiando la respuesta a distintos sensores y algortimos.

Las técnicas más extendidas son aquellas basadas en el uso del GPS (*Global Positioning System*) que recogen información directa sobre la posición global del vehículo enviada a través de una constelación de satélites. Los sistemas de navegación obtenidos bajo esta ténica resultan muy interesantes pues presentan buenos resultados tanto a para distancias largas como cortas y además no son muy sensibles a factores metereológicos y pueden ser usados a cualquier hora del día, sin importar la luz solar disponible. Sin embargo, resultan muy poco robustos en ciertas áreas como zonas urbanas, bosques o lugares donde hay baja visibilidad de satélites, llegando a no funcionar directamente en algunas áreas interiores. Han sido muchas las soluciones propuestas para resolver este problema. Una de las técnicas más famosas es el uso del GPS con otro tipo de información, como puede ser datos obtenidos del INS (*International Navegation System*). Sin embargo todas ellas siguen manteniendo la dependencia de visibilidad de los satélites para la transmisión de señales al receptor GPS.

Otra técnica para obtener sistemas de navegación robustos en aéreas interiores es el uso de sistemas de localización externos pero resultan realmente costosos. Por ello, con el fin de encontrar un equilibrio entre buenos resultados tanto para áreas exteriores e interiores y el coste de los sistemas de navegación aparecen los algoritmos de visión por computador. Este tipo de algoritmos pueden ser de uso exclusivo o pueden ser usados conjuntamente con otros sensores para mejorar su precisión como en el caso mostrado anteriormente para el GPS.

Estos algoritmos se basan en información extraída de imágenes y por ello tienen un bajo coste y además una baja potencia de consumo por lo que resultan ideales para los drones con tamaño reducido. Sin embargo, depende de muchos otros factores como la

resolución de las imágenes, el tiempo de captura, el ángulo de visión, la iluminación o los datos tomados como referencia. Por ello, siguen siendo un área abierta de investigación, pues existen también múltiples perspectivas de trabajo. Cabe mencionar que dado que la información con la que se trabaja son las imágenes, éstas pueden ser obtenidas dos maneras esenciales dando aparición a dos sistemas de navegación diferentes: los sistemas de navegación estéreo, que utilizan dos cámaras o una cámara estéreo para obtener las imágenes y los sistemas de navegación de visión monocular, que únicamente disponen de una cámara incorporada al vehículo.

Aunque los sistemas de navegación estéreo presentan inicialmente múltiples ventajas frente a los sistemas de navegación monocular, éstos últimos resultan muy interesantes cuando trabajamos con drones debido a su poco peso y su bajo consumo de energía, cosa que los sistemas estéreos no pueden ofrecer.

Este trabajo se centrará por tanto en un sistema de navegación de visión monocular en tiempo real basado en odometría visual, entendiendo por odometría visual al proceso por el cual obtenemos la posición y orientación de un objeto o vehículo con los datos capturados por una cámara o varias en diferentes instantes del tiempo.

Es de clara evidencia que este tipo de algoritmos conllevan una carga computacional muy alta, por lo que a veces estos sistemas no sólo necesitan de un software de alto nivel, sino que el tiempo de ejecución puede llegar a ser elevado, lo que se intentará mejorar en el presente trabajo.

1.1. Aplicaciones

Centrándonos en los vehículos aéreos no tripulados (VANT, en inglés UAV), las aplicaciones más extendidas desde hace ya un par de años son la fotografía aérea u aplicaciones de alta demanda para la filmación de ciertos deportes, conciertos u otro tipo de eventos pues permiten conseguir diferentes ángulos de captura que antes de la existencia de este tipo de vehículos no eran posibles, o al menos muy difíciles de conseguir. Este tipo de aplicaciones necesitan el acoplamiento de cámaras a los vehículos aéreos, lo que despertó mucho interés de cientos de investigadores para el desarollo de aplicaciones más especializadas como militares, civiles o científicas. A continuación enumeraremos algunas de ellas que hoy en día tienen un papel importante en nuestra comunidad:

- Fotografía y filmación.
- Adquisición de información espacial. Entre infinitudes de aplicaciones, resulta muy interesante el último desarollo de la NASA, el /textitExtreme Access Flyer [1] que permite la exploración de otros planetas y asteroides.
- Inspección de edificaciones, como en el trabajo presentado para inspeccionar la iglesia de la Mercerd en Burgos para posteriormente llevar a cabo su rehabilitación

[2] o para realizar el levantamiento tridimensional de la iglesia de San Miguel de Ágreda en Soria [3]. También pueden servir para el control de instalaciones industriales

- Obtención de datos sobre áreas inaccesibles o peligrosas, gracias al pequeño tamaño de los drones.
- Exploración geográficas. Son aplicaciones geomáticas [4] generalmente enfocadas al estudio urbanístico del terreno como por ejemplo el trabajo [5]. También pueden ser usados para el estudio de ciertas áreas geográficas, como por ejemplo para en la inspección de diques en Bélgica, como se muestra en la figura 1

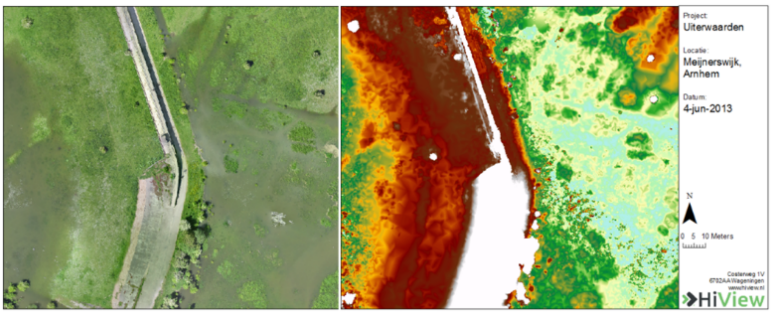


Fig. 1. Ejemplo de control de inundaciones.

- Transporte y manipulación. Resulta una propuesta muy atractiva para las empresas dedicadas al comercio electrónico donde los drones son los encargados del transporte, distribución y entrega [6].
- Protección animal y control de la fauna y el hábitat. Gracias a las imágenes obtenidas podemos identificar el tipo de flora y fauna y llevar un control de su evoluación [7] además de llevar un control de las especies que viven en ella.
- Navegación por simple ocio.
- Monotorización del tráfico. Para llevar un control del tráfico en carreteras, como en los trabajos [8] y [9].
- Monotorización de cultivos agrícolas. Son muchos las aplicaciones existentes para el control de cultivos agrícolas, como los trabajos [10] para el reconocimiento de plagas en el cultivo de la caña de azúcar, [11] para realizar un estudio fotopatológico, es decir, para detectar enfermedades y plagas en cualquier tipo de planta, o el [12] para la detección de algun tipo de maleza en plantaciones de quinoa. Normalmente, este tipo de aplicaciones se basan en detectar anomalías en la absorción de radiación solar para la fontosíntesis de las plantas como se muestra en la figura 2

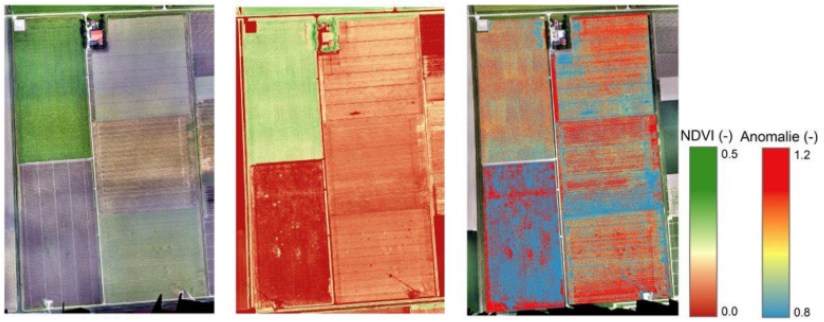


Fig. 2. Ejemplo de monotorización de cultivo agrícola.

- Búsqueda de personas perdidas. Actualmente en España y otros paises como Andorra, existe un sistema integrado de drones usado por los bomberos que permite conectar éstos con teléfonos móviles sin señal para informar de su posición [13]. También puede resultar interesante la información gráfica obtenida por las cámaras para llevar a cabo un reconocimiento facial.

1.2. Justificación

Dado un sistema de navegación basado en odometría visual monocular se pretende optimizar la eficacia del mismo mejorando los tiempos de repuesta del algortimo desarollado. La odometría visual consiste en la estimación de la posición y orientación de un vehículo o robot basándose principalmente en la detección de ciertas características de la imagen y con ellas el cálculo de la homografía de correspondencia entre imágenes consecutivas. La detección de características, que no es otra cosa que localizar ciertas partes de la imagen con un cierto interés por poder ser distingidas del resto a través del procesamiento de imágenes supone un alto coste computacional, lo que conlleva a altos tiempos de ejecucción. Esto supone que este algoritmo no pueda aplicarse en situaciones en tiempo real, por lo que en el presente trabajo se plantean dos soluciones para obtener unos tiempos de ejeccución menores.

1.3. Problema a resolver

Se presentan dos alternativas con el fin de reducir el tiempo de compilación del algoritmo manteniendo la robustez del mismo: el uso de la arquitectura CUDA (*Compute Unified Device Architecture*) sobre el algoritmo bajo estudio y la sustitución del algoritmo de detección de características SIFT (*Scale-invariant Feature Transform*) por el algoritmo SURF (*Speeded-Up Robust Features*) proporcionado por la libería de visión artificial OpenCV (*Open Source Computer Vision*) que opera directamente sobre CUDA y se realizará una comparativa con los resultados obtenidos.

1.4. Objetivos

Cualquier sistema de navegación autónomo debería ser capaz de realizar las siguientes tareas:

- Estimar la posición del vehículo. Cuando hablamos de VANTs se trata de la capacidad de estimar su posición y orientación.
- Detectar obstáculos y tener la capacidad de evitarlos, desviándose en una trayectoria alternativa.
- Gestionar y controlar toda la navegación.

En este trabajo el objetivo principal será el de mejorar el tiempo de compilación del algoritmo reduciendo de alguna manera su coste computacional. Se presentarán dos alternativas posibles, pensadas las dos para introducir la arquitectura CUDA en el código, que nos permite acelerar drásticamente los tiempos de ejecución de ciertas operaciones aprovechando la potencia de nuestra tarjeta gráfica: la transformación del algoritmo de detección de características SIFT introducciendo la programación CUDA y la sustitución de este mismo algoritmo por el algoritmo SURF desarollado ya por la libería OpenCV para trabajar sobre CUDA. Para ello, se hará primero un repaso a todos los conceptos de la imagen digital, su procesamiento, la odometría visual y el funcionamiento de las distintas arquitecturas de compilación en paralelo así como el funcionamiento de la libería OpenCV y el funcionamiento del algoritmo SURF y el algoritmo SURF basado en CUDA.

Para analizar los resultados, se hará una comparación de los tiempos de ejecucción obtenidos por el algoritmo original, la implementación en CUDA y los obtenidos con el uso del algoritmo SURF sobre CUDA. Por último, estudiaremos las posibles mejoras que se podrían desarollar en un futuro.

1.5. Marco regulador para Drones

En este apartado se presentará el marco regulador que afecta tanto a la regulación que proporciona la seguridad necesaria para el uso de drones en sus diversas aplicaciones como el marco regulador presente para salvaguardar los derechos fundamentales de los ciudadanos particularmente en cuestiones de protección de datos e intimidad regidos tanto como por la Unión Europea (UE) como por España.

Dado que la normativa aplicada por la Unión Europea afecta a diferentes niveles frente a la española, es necesario hacer una distinción entre los distintos tipos de drones a los que se aplica. La regulación europea afectará a todo tipo de drones, conocidos técnicamente como "VANTs"(Vehículos Aéreos No Tripulados) mientras que la española afecta sólo

a una pequeña parte de ellos denominados RPAs (*Remotely Piloted Aircrafts*), es decir, aquellos vehículos no tripulados que son pilotados remotamente. Además, la legislación en vigor de la Unión Europea deja bajo la jurisdición de las naciones miembro el uso de drones en el campo militar, drones de uso civil de uso fuerzas del estado, ciertos drones con características especiales o con un cierto diseño limitado además de aquellos drones donde su Masa Máxima de Despegue (MTOW, *Maximum TakeOff Weight*) no supera los 150 kg denominados de *Clase I* [14].

1.5.1. Regulaciones y leyes en la Unión Europea.

La primera normativa en aparecer sobre la aviación civil aplicados a drones en la Unión Europea fue en el año 2008 al comenzar la demanda de este tipo de vehículos fue el Reglamento Número 216/2008 por el cual se daba la formación de la EASA (Agencia Europea de Seguridad Aérea, del inglés *Europe Aviation Safety Agency*) encargada de juramentar la seguridad y protección del medio ambiente en el sector del transporte aéreo en todo territorio Europeo y donde se daban las primeras directrices sobre la regulación en la navegación de los drones.

Sin embargo este Reglamento quedó bastante obsoleto con el gran desarollo para este tipo de vehículos en los últimos años, dado que sólo ofrecía una regulación general sobre aviación civil de todo tipo de aeronaves. Por ello, no es de extrañar que en la Resolución del Parlamento del 20 de Octubre de 2015 [15] se incluyera de forma más específica una cierta normativa para los drones RPA mencionados con anterioridad, que en su mayoría, se usan con fines recreativos. Es decir, aparece una distinción sobre la legislación dependiendo del tipo de aplicación final de los drones: profesional o de tipo recreativo.

Además, los drones no sólo deben acatar la legislación de aviación civil, sino que deben ceñirse a lo que respecta en relación a la protección de los datos personales y la libre circulación de datos regidas en 1995 en la Directiva 95/46/CE del Parlamento Europeo y del Consejo y que se completaba en el 2016 en la Directiva 2016/680, así como el reciente Reglamento de Protección de Datos aplicable desde el pasado 25 de mayo[15].

La última noticia obtenida hasta el momento en el que se realiza el presente trabajo es la aprobación de una nueva legislación en junio de 2018 para la regulación del vuelo de drones y que entrará en vigor tras publicarse en el Diario Oficial Europeo.

1.5.2. Regulaciones y leyes en España.

En el territorio español aparece el 2014 las primeras normativas en relación con navegación aeronaves profesionales no tripuladas aparecen Real Decreto 552/2014 en el que se destacan su regualción para la protección de personas y personal, previsión de colisiones, vuelos regidos por condiciones meterológicas, clasificación del espacio aéreo o para ciertos vuelos especiales como los nocturnos a diferentes alturas dependiendo del entorno, que fue complementada posteriormenete con la Ley 18/2014 en su apartado de “Infraes-

tructura y transporte" en el capítulo de "Aviación civil". Dentro de este capítulo quedan reflejados los requisitos necesarios para pilotar un VANT con fines profesionales que son tener los 18 años de edad cumplidos, tener licencia de piloto, acreditar conocimientos básicos de la propia aeronave a navegar y de sus sitemas y disponer de un certificado médico que variará dependiendo del peso del drone en cuestión:

"2.º Los pilotos que operen aeronaves de hasta 25 kilos de masa máxima al despegue deberán ser titulares, como mínimo, de un certificado médico que se ajuste a lo previsto en el apartado MED.B.095 del anexo IV, Parte MED, del Reglamento (UE) número 1178/2011 de la Comisión, de 3 de noviembre de 2011, por el que se establecen requisitos técnicos y procedimientos administrativos relacionados con el personal de vuelo de la aviación civil en virtud del Reglamento (CE) n.º 216/2008 del Parlamento Europeo y del Consejo, en relación a los certificados médicos para la licencia de piloto de aeronave ligera (LAPL).
3.º Los pilotos que operen aeronaves de una masa máxima al despegue superior a 25 kilos deberán ser titulares como mínimo de un certificado médico de Clase 2, que se ajuste a los requisitos establecidos por la sección 2, de la subparte B, del anexo IV, Parte MED, del Reglamento (UE) n.º 1178/2011 de la Comisión, emitido por un centro médico aeronáutico o un médico examinador aéreo autorizado."

(Ley 18/2014, BOE, 17 de octubre).

Además de dar una primera normativa para la aviación civil profesional, esta ley legislaba la obligación de indentificar el vehículo mediante una placa que facilitara el nombre del fabricante. tipo y modelo del vehículo, su número de serie, el nombre de la operadora y sus datos de contacto.

En el último Real Decreto del 29 de Diciembre de 2017 publicado por la BOE (Boletín Oficial del Estado) sobre normativa de utilización civil de aeronaves se recogen las leyes en vigor actualmente que afectan a los RPAs de actividad profesional y de uso recreativo por separado. En lo referente a los drones pilotados por control remoto destinados a actividades de uso recreativo cabe destacar los siguientes aspectos recogidos en el Real Decreto 1036/2017:

- Ha de volarse siempre fuera del espacio aéreo controlado, además de a una distancia mínima de 8 km de aeropuertos o aérodromos no no producir colisiones.
- Todos los drones con un peso menor a los 2kg deben volar por debajo de los 120 metros de altura mientras que los drones menores de 250g deben volar por debajo de los 20 metros.
- El drone siempre ha de volar dentro del alcance visual del piloto (VLOS, *Visual Line of Sight*), aunque se use FVP (*First Person View*), es decir, un controlador visual de navegación.

- Los vuelos en ciudad o sobre aglomeraciones quedarán reducidos a aquellos donde el drone no supere los 250g de peso y los 20 metros de altura.
- Volar queda restringido a ciertas condiciones meteorológicas por cuestiones de seguridad ciudadana.
- Los vuelos además, quedarán restringidos a diurnos excepto si el drone pesa menos de 2kg que serán permitidos vuelos nocturnos siempre que no superen 50 metros de altura.

Por otra parte, en lo referente a los drones pilotados por control remoto destinados con fines profesionales cabe destacar los siguientes aspectos recogidos en el Real Decreto 1036/2017:

- Siempre que el drone supere los 25Kg de MTOW (Máximo Peso de Despegue) es obligatorio disponer de un certificado de aeronavegabilidad y estar incrito en la Matrícula de Aeronaves Civiles.
- En lo que respecta al uso de drones profesionales dentro de ciudades o sobre algomeraciones habrá varios requisitos a cumplimentar: disponer de la autorización correspondiente proporcionada por la AESA, volar siempre dentro del alcance visual del piloto (VLOS), no superar los 10kg de MTOW y cumplir con los requisitos de seguridad de distancia mínima de vuelo de 50 metros de altura, mantener una distancia horizontal entre las diferentes estructuras que puedan aparecer de al menos 50 metros también, además de disponer de un sistema de seguridad intregrado en el vehículo limitador de la energía del impacto como pueden ser los airbags o un paracaídas.
- Los vuelos nocturnos serán autorizados siempre que se disponga del permiso requerido por la AESA además de la visibilidad de la nave.
- Se podrá navegar dentro de un espacio aéreo controlado siempre el drone se encuentre dentro del aérea de alcance visual del piloto cuando este no supere los 25Kg de MTOW y disponga del permiso necesario facilitado por la AESA o cuando se disponga de la calificación de radiofonista y de un transpondedor en modo S cuando supere los 25Kg.
- Una de las normativas más novedosas es la permisión de vuelos fuera del alcance visual del piloto (BVLOS, *Beyond Visual Line of Sight*) para drones con su MTOW mayor a 2kg y con ciertos permisos proporcionados por la AESA y la de vuelos dentro del alcande visual aumentado (EVLOS, *Extended Visual Line of Sight*) a través de unos observadores intermedios que se comunicaran con el piloto a través de radio.

Cabe destacar al margen de lo anteriormente mencionado, en lo que respecta directamente a las aplicaciones del trabajo presente que en el territorio español no existe una normativa vigente concreta sobre los vehículos no tripulados con navegación autónoma, de hecho en la actualidad no están permitidos:

"Este real decreto, en coherencia con la convención internacional en la materia y las normas de derecho comparado no regula el uso de aeronaves civiles no tripuladas que no permiten la intervención del piloto en la gestión del vuelo, las denominadas aeronaves autónomas, cuyo uso en el espacio aéreo español y en el que España es responsable de la prestación de servicios de tránsito aéreo no está permitido."

(BOE, Real Decreto 1036/2017, 29 de Diciembre.)

Usos	2016	Real Decreto 1036/2017
Vuelos nocturnos	✖ No se permite	✔ En las condiciones establecidas en el Real Decreto
Zonas fuera de aglomeraciones de personas y poblaciones	✔ Limitado	✔ En las condiciones establecidas en el Real Decreto
Sobrevuelo de zonas urbanas y sobre aglomeraciones de personas	✖ No se permite	✔ En las condiciones establecidas en el Real Decreto
Vuelos en espacio aéreo controlado	✖ No se permite	✔ En las condiciones establecidas en el Real Decreto
Operaciones de policía, aduanas, CNI y tráfico	✖ No se contempla un régimen específico	✔ En las condiciones establecidas en el Real Decreto

Fig. 3. Comparativa de la normativa de 2016 con el nuevo Real Decreto. Fuente: Plan Estratégico para Drones.

1.6. Entorno socio-económico

Como se ha comentado ya más de una vez en este trabajo, en los últimos años la aparición de drones no ha dejado de crecer gracias a la posibilidades que estos ofrecen tanto en aplicaciones de ocio como de carácter científico o de investigación. Es por ello que no es de sorprender que los costes en relación con la fabricación y venta de este tipo de vehículos se hayan reducido un 90 % en los últimos cinco años según el Plan Estratégico de Drones del Ministerio de Fomento de España así como que cada vez aparezcan más modelos a nivel internacional(Figura 4). Tampoco lo es que los principales países benefactores de esta nueva tecnología sean Estados Unidos, China y en general Europa.

En Europa los países con mayores ingresos son Reino Unido, Alemania y seguido por Francia donde se encuentra la sede de la famosa empresa Parrot. A pesar de que España no se encuentra en el pódium es destacable el crecimiento de sus ingresos en los últimos años obteniendo, según la AESA, cuantías de 20000 a 50000 €anuales.

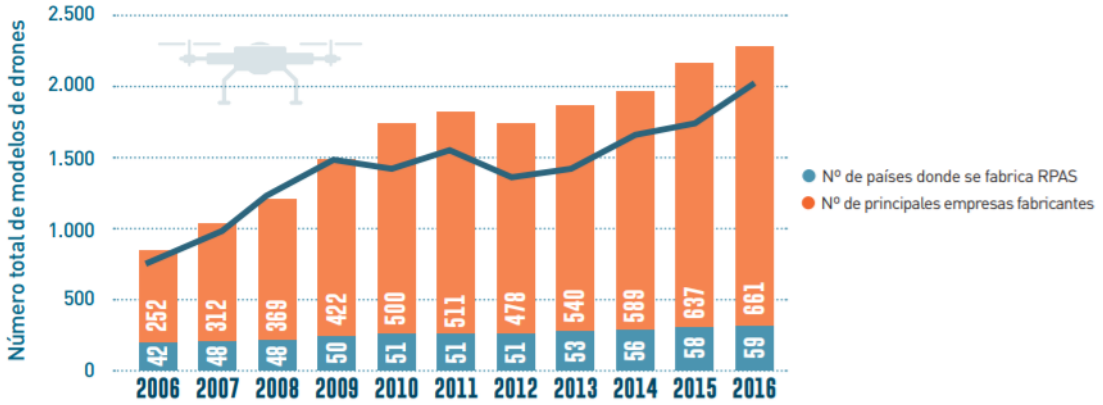


Fig. 4. Estadísticas internacionales del sector de drones (2006-2016). Fuente: Plan Estratégico, UVS International.

Tanto a nivel internacional como nacional el 95 % de drones fabricados se destinan a actividades de uso recreativo como la fotografía o la filmación (Figura 5, Figura 6). Sin embargo, ciertas asociaciones como la Droniberia (Asociación de Empresas de Drones de España) como el propio Gobierno presentan algunas iniciativas para poder introducir el uso de drones en otros escenarios.

	2016	2017
Recreativos	2,00	2,80
Profesionales	0,10	0,17
Total unidades	2,10	2,97
Crecimiento interanual	60%	39%

Fig. 5. Fabricación de drones recreativos y profesionales (millones de unidades). Fuente: Plan Estratégico, Gartner (2017).

Cabe destacar la Compra Pública Innovadora y el CDTI (Centro para el Desarrollo Tecnológico Industrial) en cuestiones de financión para impulsar este sector, además del Plan Estratégico de Drones 2018-2022 presentado por el Ministerio de Fomento de España que trata de conseguir el crecimiento de esta nueva tecnología para hacer del país un competencia digna en Europa. Este plan estratégico se basa principalmente en 4 ejes o focos:

- "Eje Estratégico 1: Implantación del marco normativo actual y desarrollo de la normativa futura"

  Se centra en la implantación de nuevas leyes para perservar la seguridad y la privadad de la ciudadanía.

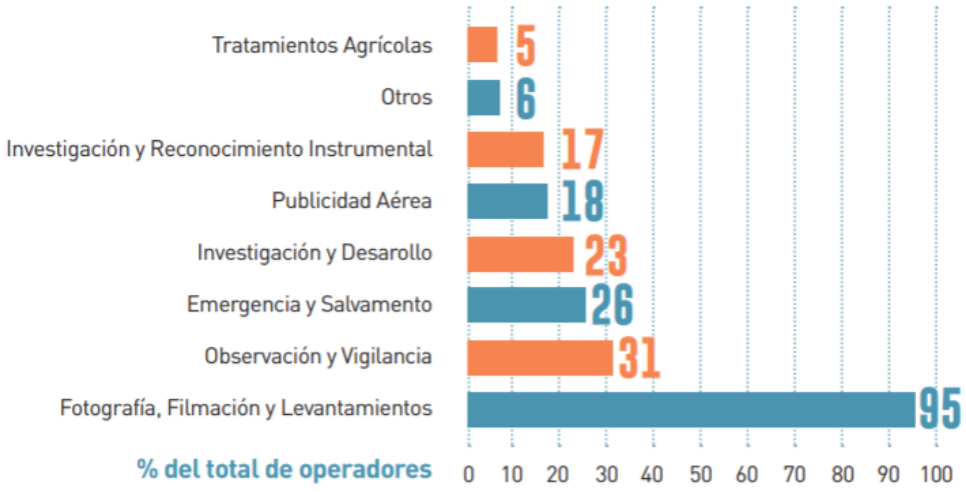


Fig. 6. Estadísticas sobre el uso de drones en 2018. Fuente: Plan Estratégico, AESA.

- "Eje estratégico 2: Impulso al desarrollo empresarial y a la I+D+i del sector"

  Se centra en realizar un estudio del mercado para poder impulsar ciertas iniciativas con el fin de integrar los drones en todo tipo de actividades y no solo en aquellas de tipo recreativo.

- "Eje estratégico 3: Divulgación de información sobre el sector"

  Se centra en desarrollar tácticas para aprender el funcionamiento correcto de los drones y para la difusión de información sobre la utilidad del sector en nuestra sociedad.

- "Eje estratégico 4: Coordinación entre Administraciones"

Todo este tipo de iniciativas permitirá en cuestión de tiempo la aparición cada vez más de drones con fines profesionales. La aparición de nuevos modelos y nuevas aplicaciones generará nuevos puestos de trabajo y nuevos servicios como por ejemplo el servicio de paquetería a través de aeronaves, y por lo tanto riqueza al país. Según datos proporcionados por el Plan Estratégico se estima que en 2035 Europa alcanzará las 400000 aeronaves con fines profesionales, como para los que está pensado el presente trabajo además de que se estima un impacto de 1220 millones de euros en el país gracias a este sector.

Es por ello que este proyecto puede resultar más que interesante teniendo en cuenta las nuevas tendencias de introducir drones tanto en la vida cotidiana como profesional. Además, más adelante en este trabajo se estudiará el presupuesto que implicaría llevar a cabo este proyecto y se reflejará la compensación de inversión.

1.7. Fases de desarollo y planificación del proyecto

- **Fase 1. Planteamiento del proyecto.**

  En esta etapa se recogen las primeras reuniones con el tutor para definir cuál es el objetivo del proyecto y dar una idea general del planteamiento y organización del mismo.

- **Fase 2. Instalación del entorno requerido.**

  Las primeras semanas se han dedicado a la instalación del entorno de desarollo, en concreto:

  - Instalación de Ubuntu 16.04.
  - Instalación de OpenCV 3.3.1.
  - Instalación de Nvidia GeForce GTX 1050.
  - Instalación de ROS Kinetic.
  - Instalación de CUDA 9.1.
  - Instalación de LaTeX.

- **Fase 3. Estudio de diferentes tecnologías.**

  Esta fase se ha centrado en el estudio o repaso de las tecnologías necesarias para poder llevar a cabo el trabajo. A modo de resumen:

  - Repaso del lenguaje de programación C++.
  - Tutorial de OpenCV.
  - Instalación de Nvidia.
  - Instalación de ROS Kinetic.
  - Instalación de CUDA.
  - Instalación de LaTeX.

- **Fase 4. Desarollo del código planteado.**

  Como es lógico esta fase, con la anterior, serán las más extensas del proyecto. Esta fase es en la que desarollaremos todo las mejoras para el algoritmo dado. La primera parte del desarollo se centrará en refactorizar el lenguaje de programación C++. La siguiente parte a adaptar el código para poder compilar con CUDA. Por último, debemos desarollar el algoritmo de detección de características SURF para realizar la sustitución del algoritmo SIFT.

- **Fase 5. Pruebas y control de errores.**

  Esta etapa se dedica para repasar el algoritmo y hacer un pequeño control de errores para comprobar que todo funciona como era previsto.

- **Fase 6. Conclusiones.**

  En esta fase se realizaron las comparaciones entre las diferentes propuestas de mejora, se extrajeron las conclusiones finales del proyecto, y se analizaron las posibles líneas de trabajo futuras basadas en este trabajo.

- **Fase 7. Escritura de la memoria final.**

  La última fase, antes de la entrega, es recoger todos los resultados y las conclusiones finales de lo trabajado en las fases anteriores en la memoria presente.

- **Fase 8. Entrega de la memoria final.**

  Entrega de la memoria final a través de la plataforma Aula Global de la UC3M.

FASES	Fecha de inicio	Fecha de final	Duración (días)
Planteamiento del proyecto	02/10/2017	09/10/2017	7 días
Instalación del entorno requerido	09/10/2017	23/10/2017	14 días
Estudio de diferentes tecnología	23/10/2017	04/04/2018	130 días
Desarollo del código planteado	04/04/2017	22/06/2018	80 días
Pruebas y control de errores	22/06/2017	01/07/2018	9 días
Conclusiones	01/07/2017	11/07/2018	10 días
Escritura final de la memoria	11/07/2017	24/09/2018	75 días
Entrega de la memoria	25/07/2017	25/09/2018	1 día

TABLA 1. PLANIFICACIÓN INICIAL.

1.8. Estructura del documento

En este apartado se hará una breve descripción del contenido de cada seción del presente trabajo:

- **Capítulo 1. Introducción.** En esta introducción se hace un pequeño resumen de los temas que se van a tocar durante el trabajo además de un estudio sobre el marco regulador vigente por el posible impacto que podría tener este trabajo para el entorno socioeconómico.

- **Capítulo 2. Estado del arte.** En esta sección se presentarán los conceptos básicos para poder entender a continuación las tecnologías empleadas a lo largo del trabajo. Podríamos separarla en varias subsecciones:

  1. Revisión de los conceptos básicos sobre la imagen y el procesamiento digital.
  2. Revisión del funcionamiento de las cámaras digitales, el modelo Pinhole, el sistema de coordenadas usado por las cámaras y su calibración.

3. Revisión de algoritmos de sistemas de navegación autónoma de VANT's y revisión de los conceptos de odometría visual.
4. Presentación del algoritmo sobre el que vamos a trabajar y las técnicas utilizadas.
5. Presentación de las arquitecturas CUDA y OpenCL.

- **Capítulo 3. Solución propuesta.** En esta sección se presentarán las propuestas para mejorar la eficacia de nuestro algoritmo centrándose en:

  1. Compilación sobre tarjetas gráficas (*GPU computing*) mediante la arquitectura CUDA del algoritmo SIFT manual.
  2. Algoritmo de detección de características SURF en CUDA directamente.

- **Capítulo 4. Resultados obtenidos.** Esta sección es la encargada de recoger los resultados obtenidos tras los cambios aplicados y realizar las comparaciones pertinentes entre las dos propuestas con el fin de observar la optimización del sistema de navegación autónoma basada en odometría visual monocular, principalmente gracias al aumento de velocidad en tiempo de compilación del programa. También se recogerá los escenarios, plataformas y materiales usados para su ejecucción.

- **Capítulo 5. Presupuesto del proyecto.** En esta sección se presenta una estimación del presupuesto real del proyecto.

- **Capítulo 6. Conclusiones y trabajo futuro.** Esta sección se presenta las conclusiones finales del trabajo una vez analizados los diferentes resultados y el posible trabajo que podría surgir en relación con el proyecto presentado.

- **Apéndices.** Se adjunta en esta sección una breve descripción y funcionamiento de los procesadores, y de otro tipo de técnicas mencionadas en el trabajo pertinentes por si fueran de ayuda para el lector en el entendimiento de los cambios aplicados.

2. ESTADO DEL ARTE

En la primera parte de este capítulo de este capítulo daremos un breve repaso a los conceptos básicos sobre la imagen y el procesamiento digital de las mismas. Estos conocimientos serán requeridos para entender los algoritmnos y técnicas basadas en visión por computador.

A continuación, se revisarán las aplicaciones UAV más conocidas basadas en visión por computador, así como los algoritmos usados en cada una de ellas, centrándonos para finalizar en el algoritmo proporcionado que ha sido desarollado para la estimación de la posición de un UAV basado en odometría visual monocular, sobre el que trabajaremos en el resto del trabajo con el fin de mejorar su eficiencia.

2.1. Conceptos iniciales

2.1.1. Naturaleza de luz

La Luz es la parte de la radiación electromagnética que puede ser percibida por el ojo humano. Esta tiene una naturaleza doble, es decir, su comportamiento puede ser explicado por dos modelos teóricos coexistentes: el modelo ondulatorio y el modelo corpuscular.

El primer modelo defiende que la luz es una radiación electromagnética que se propaga en forma de ondas caracterizadas por una frecuencia (f) y un cumplimiento de onda λ. Sin embargo, solo una pequeña franja de frecuencias dentro del espacio electromagnético (fig.7), denominada luz visible, puede ser detectada por el ojo humano. Como puede observarse en la figura 8, cada longitud de onda corresponde a un color determinado captado por un individuo.

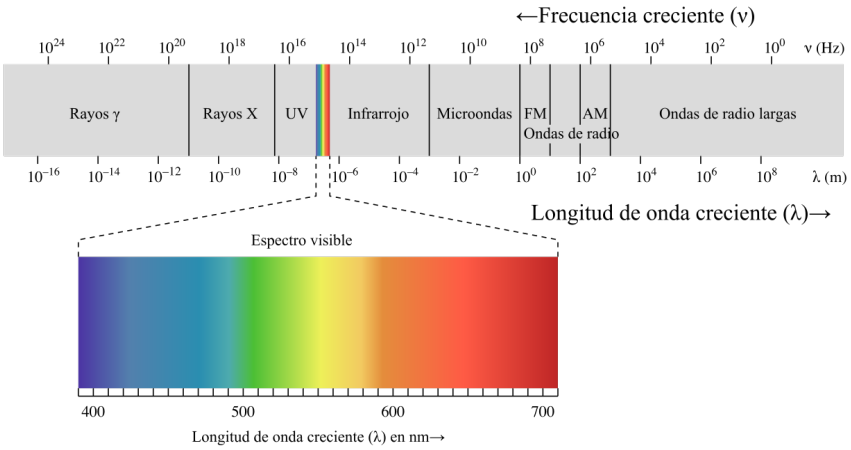


Fig. 7. Espectro electromagnético. Fuente: http://www.areaciencias.com

Luz visible

Color	Frecuencia	Longitud de onda
Violeta	668–789 THz	380–450 nm
Azul	631–668 THz	450–475 nm
Ciano	606–630 THz	476–495 nm
Verde	526–606 THz	495–570 nm
Amarillo	508–526 THz	570–590 nm
Naranja	484–508 THz	590–620 nm
Rojo	400–484 THz	620–750 nm

Fig. 8. Longitudes de onda y frecuencias para cada color. Fuente: http://e-ducativa.catedu.es

2.1.2. El ojo humano: formación de la imagen

El funcionamiento de las cámaras fotográficas está basado en el funcionamiento del ojo humano debido a su clara analogía (figura 9), como hemos explanado en el apartado anterior.

La luz atraviesa inicialmente la córnea, la parte anterior transparente y protectora del ojo que con su forma cóncava funciona como un lente convergente. Esa luz penetra en el ojo a través de un pequeño orificio ubicado en la parte central del iris denominado pupila. La pupila es la encargada de transportar esa luz hasta el cristalino, otra lente convergente, que localiza la imagen sobre una película situada en el fondo del ojo conocida como retina. Es en ella donde se forma la imagen invertida que es enviada al cerebro a través del nervio óptico para interpretarla.

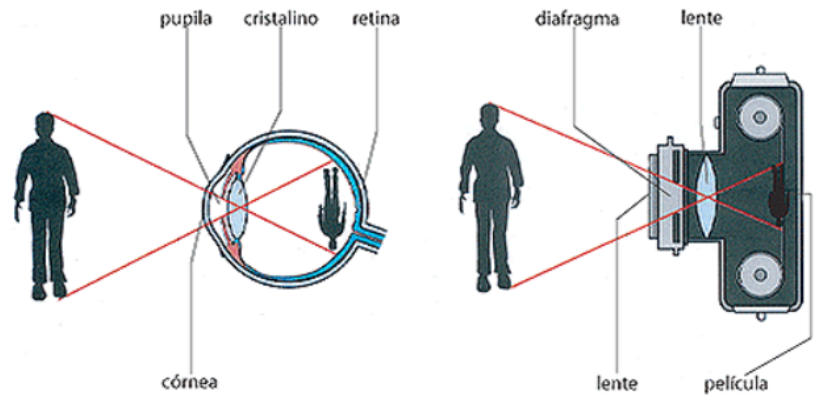


Fig. 9. Formación de la imagen en el ojo y comparación con una cámara fotográfica. Fuente: http://www.medic.ula.ve

2.1.3. La imagen digital

Una imagen digital es la representación de un escenario por medio de un conjunto de elementos discretos y de tamaños finitos denominados píxeles. Los píxeles son distribuidos en una matriz bidimensional como se muestra en la figura 10, que representará las dimensiones de la imagen, es decir, la imagen tendrá N x M píxeles de resolución, siendo N el número de filas y M el de columnas. Como podemos prever, cuanto mayor sean el número de filas y columnas, mayor será el detalle de la imagen (resolución) pero mayor tamaño ocupará el archivo en la memoria.

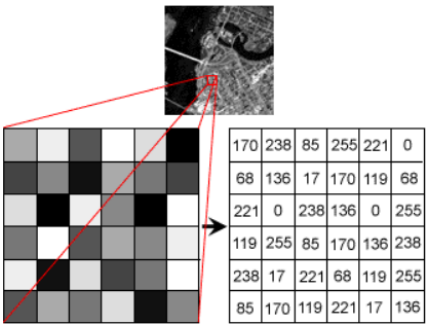


Fig. 10. Extracción de píxeles de una imagen.

En escala de grises, cada píxel contiene un valor númerico entre 0 (correspondiente al negro puro) y normalmente, 255 (correspondiente al blanco puro) en formato binario, que representa la cantidad de luminosidad de ese píxel. Cuando la imagen es a color, el píxel pasa a ser un vector cuyos elementos representan los valores del sistema RGB (*"Red, Green and Blue"*), por lo que una imagen a color puede ser entendida como la suma de tres imágenes monocromáticas (fig.11).

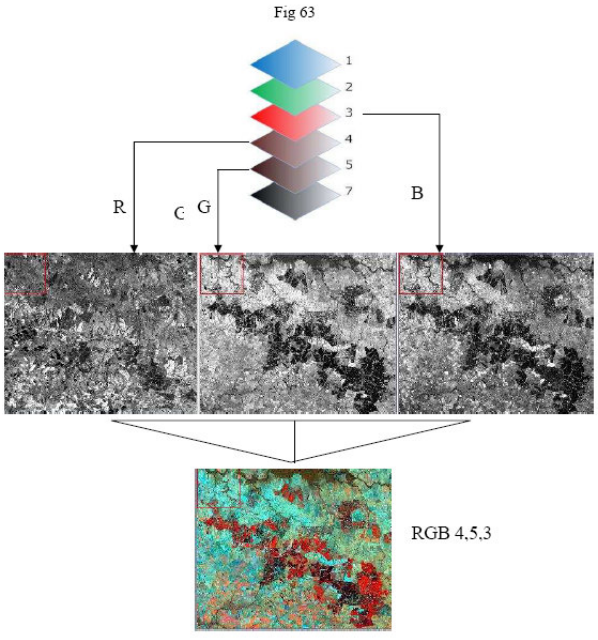


Fig. 11. Imagen RGB compuesta por tres imagenes monocromáticas.

2.1.4. Formación del vídeo

Un vídeo es conjunto de imágenes que se sucen una a otra en el tiempo. Estas imágenes son conocidas como fotogramas o en inglés, *frames*. (Figura 12). Estos fotogramas o *frames* se caracterizan por los denominados fps *("Frames Per Second")* que no es otra cosa que la frecuencia del vídeo, es decir, la cantidad de imágenes que suceden en un segundo.

Fig. 12. Ejemplo de fotogramas que conforman un vídeo. Fuente: Google Sites

Dependiendo de la finalidad del vídeo se puede grabar a diferentes *frames por segundo*. Por convenio lo valores estándares de FPS son:

- Formato televisión o vídeo PAL $\rightarrow 25 fps$
- Formato de televión o vídeo NTSC $\rightarrow 29{,}97 fps$

El vídeo digital que es el que se maneja practicamente en el mercado, es un tipo de sistema de grabación que usa una señal de digital en vez de analógica, es decir, se representa a través de un flujo de bits, denominado *bit rate*. Al igual que los FPS, dependiendo de la finalidad del vídeo, este puede transmitir menos o más bits. Cuando el vídeo exige mucho movimiento o cambios de plano, los píxeles de las imágenes difieren mucho unas de otras por lo que se precisa de un flujo variable de bits, mientras que para otro tipo de escenas el flujo resulta constante.

Este vídeo digital se capturan mediante cámaras digitales aunque muchas veces se graban sobre cintas que se distribuyen después en discos ópticos.

2.1.5. Procesamiento digital de las imágenes

Llamamos procesamiento digital de imágenes (PDI, *"Processing Digital Image"*) al uso de algoritmos computacionales que realizan cierto procesamiento digital en las imágenes, es decir, transformaciones, por ejemplo, para realzar ciertas características dentro de ellas [16].

El procesamiento digital de una imagen se puede divivir en varias tareas o etapas interconectadas (figura 13) que explicaremos brevemente a continuación [**?**]

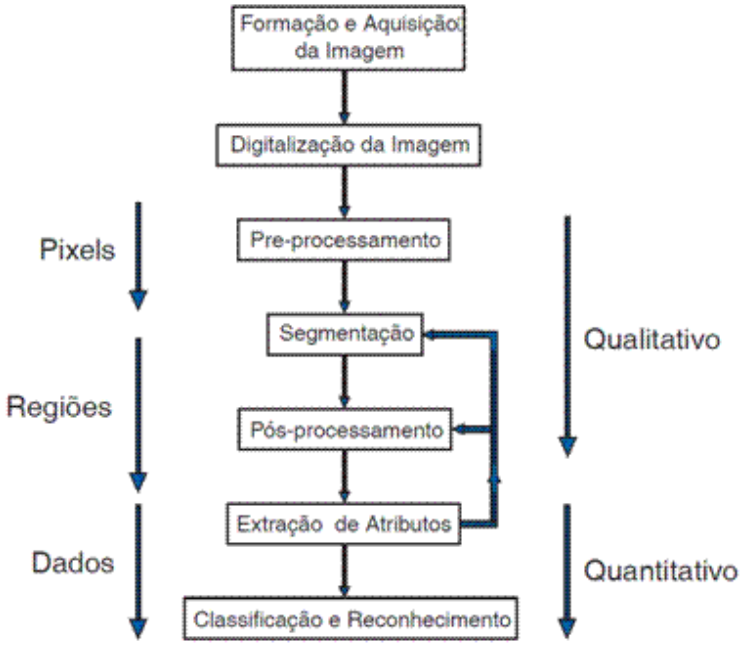


Fig. 13. Etapas del procesamiento digital de las señales. Fuente: http://www.cbpf.br

1. **Adquisición y digitalización: Muestrear y cuantificar**

   Esta es la primera etapa de cualquier procesamiento de imagen. Las imagenes digitales no son más que la discretización de imágenes analógicas que se representan mediante una matriz bidimensional como ya hemos comentado anteriormente. Es en esta etapa donde la imagen es capturada por un sensor y transformada en una imagen digital. A la transformación de discretización en coordenadas espaciales la denominados muestreo, mientras que cuando discretizamos los valores de brillo para cada coordenada lo denominamos cuantificación.

2. **Preprocesamiento**

   Esta es una de las etapas más importantes, pues se encarga de la modificación de ciertas características de la imagen con la finalidad de facilitar su procesamiento posterior digital. Podemos distinguir 4 categorías:

- Transformaciones de brillo (modificación del histograma):

  Existen dos transformaciones dentro de esta categoría: las correcciones del brillo, que depende de la posición del píxel, y las tranformaciones del nivel de grises, que no tienen dependencia ninguna.

- Transformaciones geométricas (distorsiones):

  Son aquellas que modifican la relación espacial entre los píxeles.

- Métodos basados en vecinidad local, es decir, usan los N píxeles vecinos para calcular un nuevo valor de luminosidad para un píxel determinado. Podríamos distinguir dos tipos de transformaciones: el suavizado (*smoothing*) y las operaciones de gradiente.

- Restauración de imágenes.

Transformaciones de brillo:

El histograma es una representación gráfica de la imagen donde se representa la cantidad de píxeles que tienen un mismo nível de gris (Figura 14).

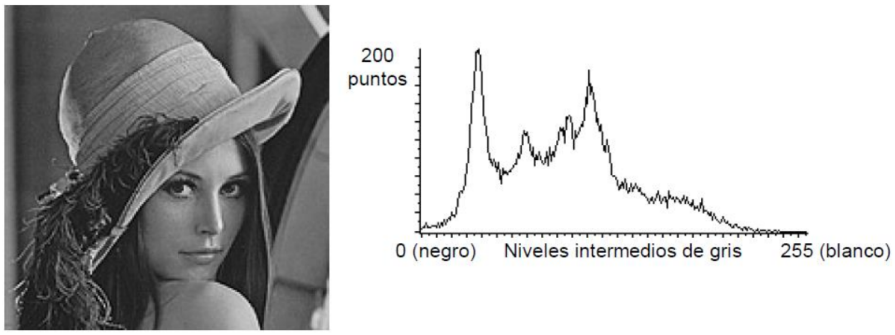


Fig. 14. Histograma de una imagen. Fuente: http://acodigo.blogspot.com

Existen muchas transformaciones puntuales de imagenes que se basan en la modificación de su histograma, como por ejemplo:

- Inversión de los niveles de grises: consiste en obtener el negativo da la imagen positiva y viceversa (Figura 15).
- Expansión del contraste: consiste principalmente en la redistribución de los tonos de nivel de grises con el fin de elevar el contraste, muy usado en aplicaciones que necesitan reconocer ciertos patrones (Figura 16).
- Ecualización del histograma: Consiste en conseguir un histograma con distribución uniforme para realzar las diferencias de tonalidad.

Fig. 15. Inversión de la escala de grises en una imagen.

Fig. 16. Expansión del contraste de una imagen.

Transformaciones geométricas:

Las transformaciones geométricas constan de dos operaciones básicas:

- La transformación espacial que cambia la posición de los píxeles dentro de la imagen. Son operaciones como cambio de escala, rotación por un ángulo...
- Interpolación de los niveles de grises con los N píxeles vecinos.

Métodos en el dominio espacial:

Siendo la imagen la funcion $f(x, y)$, estas transformaciones espaciales se pueden expresar según la siguiente ecuación (1):

$$g(x, y) = T\{f(x, y)\} \tag{1}$$

Donde el operador T está definido sobre alguna vecindad del punto *(x,y)* [17], como se muestra en la figura 17.

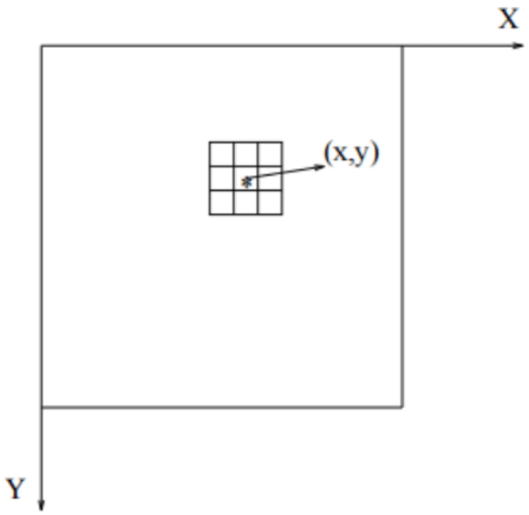


Fig. 17. Ejemplo de la vecinidad 3x3 sobre un punto determinado (x, y) en una imagen.

Estas técnicas normalmente usan matrices bidimensionales a las que denominamos máscaras que actúan sobre cada píxel de la imagen multiplicando sus elementos, que son escogidos con anterioridad con el fin de detectar o resaltar una característica de la imagen, por todos los píxeles que conforman el área de la vecindad (fig. (17)). Para calcular el nuevo valor de cada píxel de la imagen de salida $g(x, y)$, el centro de la máscara se va desplazando por la imagen.

- Suavizado de imágenes:

  El suavizado es una técnica que se usa para tratar de eliminar cualquier efecto no deseado que aparece en la imagen normalmente después del proceso de adquisión, como por ejemplo el ruído.

  Este proceso obtiene el valor de cada píxel como valor medio de N píxeles vecinos. Otra técnica de suavizado consiste en la aplicación de filtros de mediana sobre cada píxel eliminando los picos de intensidad que aparecen en ciertas zonas de la imagen [17].

- Contraste de la imagen:

  Se trata de técnicas usadas principalmente para resaltar los bordes de la imagen. Estos bordes pueden ser obtenidos mediante varios métodos, pero uno de los más usados es el método por gradiente, pues permite realizar esta transformación en el dominio del tiempo [17].

  Se entiende por gradiente de una imagen *f(x,y)* como el vector *G(f(x,y))* tal que (2):

$$G(f(x,y)) = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \frac{\partial(x,y)}{\partial y} \\ \frac{\partial(x,y)}{\partial x} \end{bmatrix} \tag{2}$$

  Donde lo que realmente interesa, el módulo, se pude aproximar a (3) para

facilitar los cálculos.

$$G(f(x,y)) = |G_x| + |G_y| \tag{3}$$

El gradiente tomará valores altos cuando detecte cambios bruscos de luminosidad, que serán interpretados como bordes y valores bajos cuando no, es decir, cuando el nivel de grises sea casi constante.

Métodos en el dominio de frecuencia:

Como sabemos, una imagen puede ser representada como una función sinusoidal (4) de dos dimensiones debido a su naturaleza ondulatoria:

$$f(m,n) = \sin\{2\pi(Um + Vn)\} \tag{4}$$

Donde n y m serán las coordenadas espaciales (en píxeles) y U y V sus frecuencias correspondientes (ciclos/píxel). Sin embargo, es más común trabajar con las frecuencias normalizadas dada la dimensión de cada imagen NxM, lo que permite reescribir la fórmula de la siguiente manera (5):

$$f(m,n) = \sin\{2\pi(uMm + vNn)\} \tag{5}$$

Donde ahora las frecuencias U y V tienen un periodo de unidad y siempre se cumple la siguiente propiedad: *0 ciclos/pixel <U,V <1 ciclos/pixel* regido por el Teorema de *Nyquist* [17].

a) **La Transformada de Fourier:**

Cualquier señal puede descomponerse en una suma de funciones seno y coseno a diferentes frecuencias, por ello, es lógico que cualquier imagen pueda ser representada en sus componentes de frecuencia mediante la *Transformada de Fourier* [18]. Siendo una imagen de tamaño MxN, la transformada de Fourier será (6) y (7) con la frecuencia normalizada:

$$F(U,V) = \sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-j2\pi(Um+Vn)} \tag{6}$$

$$F(U,V) = \frac{1}{MN}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1} f(m,n)e^{-j2\pi(\frac{u}{M}m+\frac{v}{N}n)} \tag{7}$$

Y la Transformada Inversa de Fourier (8) y (9) respectivamente:

$$f(m,n) = \int_0^1\int_0^1 F(U,V)e^{j2\pi(Um+Vn)}dUdV \tag{8}$$

$$f(m,n) = \frac{1}{MN}\sum_{u=0}^{M-1}\sum_{v=0}^{N-1} F(u,v)e^{j2\pi(\frac{u}{M}m+\frac{v}{N}n)} \tag{9}$$

Sin embargo, como el rango dinámico de los sistemas de visualización en general es bastante menor que el de la transformada, se suele representar gráficamente la función *D(u,v)* (10) y un único periodo M x N, pues al ser periódica su comportamiento se repite.

$$D(u, v) = \log_2(1 - |F(u, v)|) \tag{10}$$

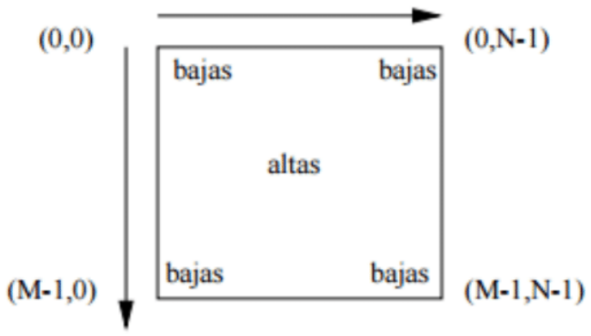


Fig. 18. Interpretación de la Transformada de Fourier de una imagen. Fuente: https://www.researchgate.net Elias Silva

b) **Filtros de baja, alta y frecuencias medias.**

Podemos clasificar los filtros por los que puede pasar una imagen dependiendo de como afectan a sus frecuencias. Denominamos filtros de baja frecuencia a aquellos que dejan pasar las bajas y atenúan las altas, filtros de alta frecuencia a los que dejan pasar las altas y atenúan las bajas y filtros de frecuencias medias a aquellos que dejan pasar las medias y atenúan el resto.

Este tipo de transformaciones se consiguen multiplicando la imagen, que no es otra cosa que una señal como ya hemos comentado repetidas veces, por otra señal, es decir, realizando una operación de convolución de la transformada de Fourier de la imagen con la transformada de la otra señal.

Cabe destacar que los filtros de bajas frecuencias son usados normalmente para eliminar el ruido de alta frecuencia o variaciones suaves de los niveles de grises y los filtros de altas frecuencias para sustraer ciertas características de interés en la imagen donde los niveles de iluminación cambian bruscamente como veremos a continuación en las técnicas para detectar bordes de una imagen.

Detección de bordes de una imagen:

Los bordes de una imagen no son otra cosa que transiciones entre dos regiones de nivel de grises significativamente distintos. Esa diferencia se puede detectar con la primera y segunda derivada de la imagen en escala de grises (figura 12):

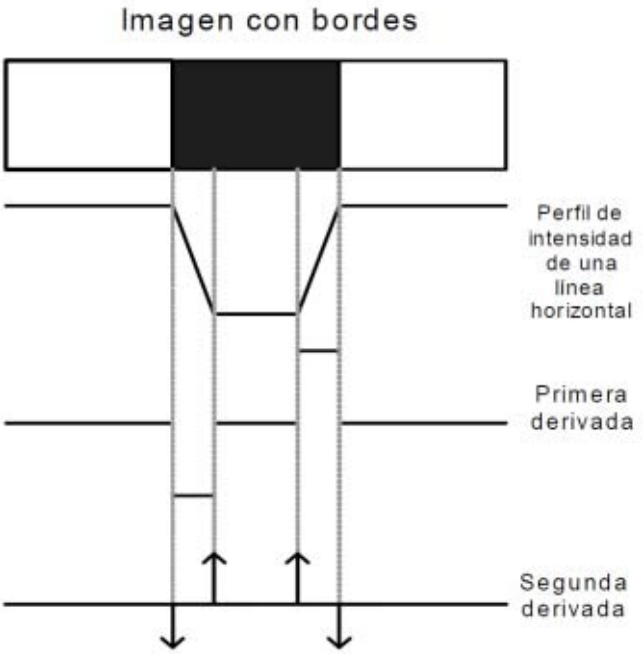


Fig. 19. Detección de bordes mediante derivadas. Fuente: https://www.researchgate.net Jorge Valverde-Rebaza

- **Métodos basados en la primera derivada:**

  Como ya hemos comentado, el gradiente permite detectar la presencia de un borde dentro de una imagen bidimensional. Un píxel para ser clasificado como tal debe tener un nivel de luminosidad mayor que un cierto umbral. Existen varios operadores de gradiente que se usan para la detección de bordes. A continuación se nombran los más conocidos:

  Operador de Roberts:

  Este operador tiene muy buenas prestaciones pero sin embargo una extremada sensibilidad al ruído y una calidad de detección bastante pobre.

  Operadores de Prewitt, Sobel y Frei-Chen:

  Los tres operadores mostrados a continuación se pueden conseguir mediante la aplicación de las siguiente máscaras:

  Llamamos operador Prewitt cuando K = 1, operador Sobel cuando K = 2 y operador Frei-Chen cuando K = $\sqrt{2}$.

- **Métodos basados en la segunda derivada:**

Gradiente fila

0	0	0
0	0	1
0	-1	0

Gradiente columna

-1	0	0
0	1	0
0	0	0

Fig. 20. Operadores de Roberts.

Gradiente fila

$\frac{1}{2+K}$

1	0	-1
K	0	-K
1	0	-1

Gradiente columna

$\frac{1}{2+K}$

-1	-K	-1
0	1	0
1	K	1

Fig. 21. Operadores de Prewitt, Sobel y Frei-Chen, dependiendo del valor de la constante K.

Los métodos basados en la segunda derivada permiten también determinar si un píxel clasificado ya como borde se encuentra en el lado oscuro o claro del borde. El operador más conocido es el Laplaciano aplicando previamente una gaussiana para resolver su gran sensibilidad al ruído.

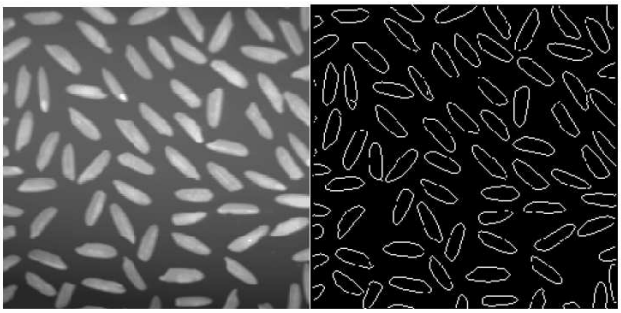


Fig. 22. Ejemplo de detección de bordes en una imagen.

3. Segmentación

Las etapas siguientes del DPS pertenecen al análisis de la imagen. La primera parte de esta etapa corresponde con la segmentación de la imagen, es decir, su subdivisión en regiones con características similares o objetos.

Los algoritmos de segmentación se aplican sobre imágenes monocromáticas y se basan en detetectar dos propiedades de este tipo de imágenes: la discontinuidad y similaridad de los niveles de luminosidad. Los algoritmos de discontinuidad son los encargados en segmentar la imagen basándose en los cambios abruptos del nivel de luminosidad mientras que los de similaridad en la limiarización (*thresholding*) y en la existencia de regiones.

Dentro de los algoritmos de detección de discontinuidad podemos encontrar algoritmos que son capaces de detectar puntos, líneas o bordes.

a) Procesamiento local:

Es aquel algoritmo que detecta bordes simplemente analizando la similitud entre píxeles vecinos. Consideramos que un píxel es similar a otro cuando:

$$|\nabla f(x,y) - \nabla f(x,y)| \leq T$$

Siendo T un umbral positivo y $|\alpha(x,y) - \alpha(x,y)| < A$ un ángulo límite.

b) Procesamiento global mediante la Transformada de Hough:

La transformada de Hough es una herramienta usada para detectar figuras en una imagen que pueden ser expresadas matemáticamente como son las rectas, círculos o elipses.

Sabiendo que la ecuación de un recta en su forma polar es: $\rho = x\cos(\theta) + y\,\text{sen}(\theta)$, la Transformada de Hough se encargará de encontrar los puntos alineados que pueden existir dentro de la imagen, es decir, que cumplan dicha ecuación para los distintos valores de ρ y θ.

La Transformada de Hough de un círculo se basa en la siguiente ecuación:

$$(x-a)^2 + (y-a)^2 = r^2 \tag{11}$$

Donde *(a,b)* es el centro del círculo y r su radio.

c) Segmentación por limiarizaicón (*thresholding*):

Esta técnica consiste en agrupar los píxeles que pertenecen a los diversos objetos de una imagen diferenciandolos del fondo. Esta umbralización se basa normalmente en el histograma de la imagen, donde la elección del umbral permite agrupar los píxeles de la imagen en distintas regiones con características de luminosidad parecidas.

d) Segmentación por regiones:

Las técnicas basadas en regiones dividen la imagen en distinas partes convexas donde cada una de ellas tiene propiedades distintas que hacen que puedan distinguirse unas de otras. Las técnicas más cononocidas son el crecimiento, la división y fusión de regiones.

4. **Post-procesamiento**

   El procesamiento es la etapa del PDI donde se corrigen los defectos o imperfecciones producidos por la segmentación mediante normalmente lo que conocemos por operaciones morfológicas.

   Estas operaciones se basan en operadores booleanos de conjunto y las más conocidas son:

   - Erosión: Permite separar objetos que se tocan ligeramente.
   - Dilatación: Permite rellenar agujeros en el interior de un objeto o juntar objetos que se tocan ligeramente.
   - Apertura: Consiste en aplicar primero la operación de erosión y después la de dilatación para eliminar pequeñas partículas de la imagen y suavizar el contorno de los objetos.
   - Estrechamiento: Consiste en aplicar primero la operación de dilatación y después la de erosión para cerrar ciertos canales que separan a los objetos y eliminar agujeros que puedan aparecer en su interior.

5. **Parametrización y extración de atributos.**

   Los descriptores son una forma de representar la imagen matemáticamente y permiten definir patrones que caracterizan los objetos de interés en ella para posteriormente clasificarlos.

   Esquemas de representación externa:

   - Código de cadena: Es un tipo de estructura para representar el contorno de un objeto dentro de una imagen binaria mediante la secuencia de números que indican las orientaciones de segmentos de entornos de 4 o 8 vecinos (figura 23), conectados consecutivamente, comenzando por cualquier punto del contorno y siguiendo el sentido de las agujas del reloj.

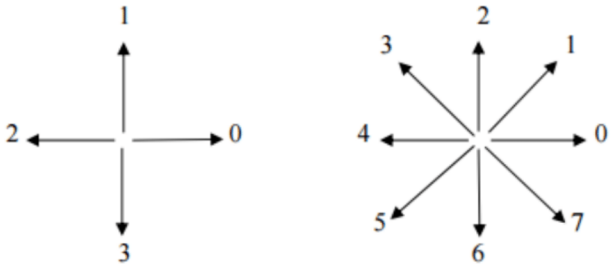


Fig. 23. Dirección para entornos de 4 y 8 vecinos.

- Signatura: Es una representación del contorno de una imagen mediante una función unidimensional real. La forma más simple de definir esta función es através de la distancia de un punto interior (normalmente el centro del contorno), a cada uno de los puntos del mismo como se muestra en la figura (24).

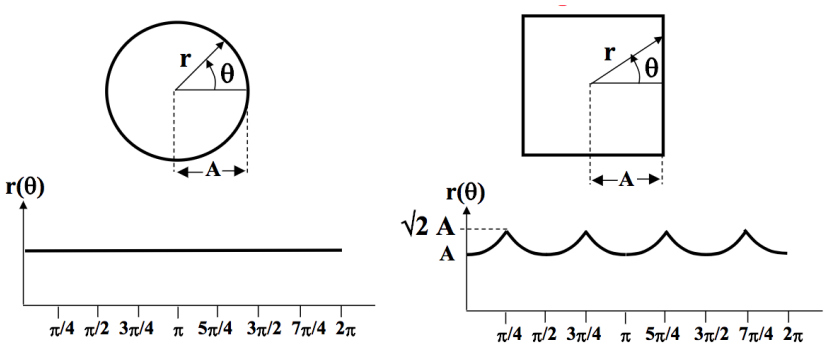


Fig. 24. Ejemplos de signaturas de un círculo y un rectángulo.

- Descriptores de Fourier: Este tipo de descriptores son usados para representar el contorno de una imagen cuando la curva es cerrada o cuando existe simetría en los objetos de interés. Representan pequeñas y grandes variaciones en la tendencia del contorno. Las altas frecuencias representan los detalles más finos, mientras que las bajas determinan la forma global de este.
- Curvas poligonares: Son una aproximación del contorno de un objeto a los vértices de una curva de partes lineares que constituye un polígono. El criterio más usado para evaluar la calidad del ajuste es el de lo mínimos cuadrados.

Esquemas de representación externa:

a) Descriptores de contornos:

Se basan en la forma geométrica de los contornos de las regiones.

- Perímetro: Número total de píxeles que configuran el entorno.
- Diámetro: Distancia euclídea entre los dos píxeles más alejados pertenecientes al contorno. La recta que atraviesa esos dos puntos se conoce como recta mayor de la región.
- Rectángulo base: Rectángulo menor que contiene al contorno.
- Excentricidad: Cociente entre la mayor longitud y la longitud del lado menor del rectángulo.
- Centro de gravedad: Punto *(x,y)* determinado por un cierto conjunto de píxeles $\{(xi, yi)\ i = 1, 2...N\}$ tal que (12) y (13):

$$x = \frac{\sum_{i=1}^{N} Xi}{N} \tag{12}$$

$$y = \frac{\sum_{i=1}^{N} Yi}{N} \tag{13}$$

- Curvatura: Tasa de cambio en la pendiente de un contorno que se obtiene, normalmente, mediante la diferencia entre las pendientes de segmentos adjacentes que pertenecen a él.
- Momentos estadísticos: Representación del contorno mediante una función real g(x) descrita por sus momentos estadísticos como son la media, varianza o otro tipo de momentos de orden mayor.

b) Descriptores de regiones:

1) Parámetros geométricos:
Como el área, es decir, la región dada por el número de píxeles que la componen. Se puede obtener a partir de su código de cadena.

2) Parámetros topológicos:
   - Compacidad (o circularidad): Parámetro que no depende de tamaño de la región y viene dado por el coeficiente entre el área y el perímetro al cuadrado.
   - Rectangularidad: Se define como el coeficiente entre el área de la región y el de su rectángulo base.
   - Alargamiento:Coeficiente entre la longitud del lado mayor y el lado menor de su rectángulo base.

3) Textura de la región:
Existen dos formas principales de estudiar la textura de una región: mediante momentos estadísticos o mediante la transformada de Fourier. Los primeros se aplican sobre el histograma en escala de grises de los píxeles que conforman la región mientras que la textura espectral se usa para descriminar entre patrones de textura periódica y no periódica además de que permite cuantificar las diferencias entre patrones periódicos.

6. **Clasificación y reconocimiento**

   Denominamos patrón a un conjunto de características que permiten describir un objeto o algún área de interés de la imagen. Estos patrones están formados por uno o más descriptores que se recogen en un vector denominado como vector de características. Una clase de patrones es un conjunto de patrones con características similares. Por tanto, el objetivo de realizar una clasificación y reconocimiento consistirá en asignar un patrón a la clase a la que con más probabilidad pertenezca.

   El proceso de clasificación consta por tanto, de los siguientes pasos:

Paso 1. Escoger un grupo de descriptores apropiado y determinar el número de clases que existen en la clasificación a realizar.

Paso 2. Asignar a cada elemento a clasificar una de las clases.

   Existen dos tipos de clasificadores en función de los procedimientos usados para determinar las clases existentes: los clasificadores supervisados, donde se dispone de un conjunto de aprendizaje para entrenar el clasificador y los clasificadores no supervisados, donde simplemente usamos un procedimiento estadístico para clasificar los patrones dentro del las diferentes clases. La asignación de cada elemento a una de las clases puede realizarse por procedimientos puramente estadísticos, algoritmos basados en inteligencia artifial como las redes neuronales o por otro tipo de procedimientos como los árboles de decisión o el cálculo de mínima distancia.

   En la clasificación no supervisada la técnica más conocida es la clasificación K-medias que clasifica los elementos a cada clase mediante técnicas de mínima distancia y la clasificación Isodata.

   La clasificación supervisada, a su vez, se subdivide en paramétrica que asume una distribución Gaussiana de los datos y no paramétrica. En la clasificación no paramétrica destacan el clasificador de máxima verosimilitud, el clasificador K-vecinos (*k-NN Nearest Neighboury*) que se basa en la idea de atribuír cada elemento a la clase a la cual pertenecen la mayor cantidad de vecinos más cercanos del conjunto de aprendizaje y el uso de redes neuronales sensitivas con un número de neuronas igual al de los parámetros de entradas y un número de efectoras igual al de clases deseadas.

2.2. La cámara: Modelo Pinhole y el sistema de coordenadas

La cámara consta principalmente de dos elementos: el sensor y la lente. La lente es la encargada de captar la luz y de enviarla al sensor, donde se transforma en impulsos electrónicos.

El modelo más simple que explica el funcionamiento de una cámara es el modelo Pinhole (fig. (25)). En este modelo suponemos que la lente es el único punto atravesado por la luz y permite calcular las coordenadas de cualquier objeto de la imagen respecto a las coordenadas del mundo real.

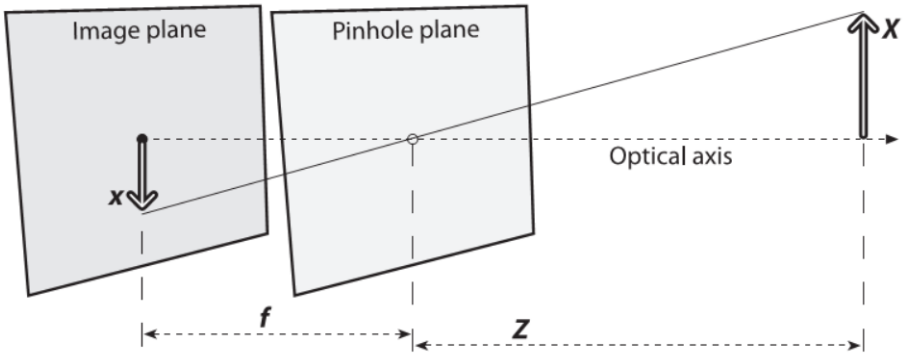


Fig. 25. Representación gráfica del modelo *Pinhole*. Fuente: https://dsp.stackexchange.com Learning OpenCV

La transformación de las coordenadas 3D a coordenadas 2D se realiza por semejanza de triángulo mediante:

$$(x, y, z) \rightarrow (f\frac{X}{Z}, f\frac{Y}{Z})$$

Siendo más específicos, en la figura (26) se muestran los puntos más importantes del modelo de proyección perspectiva. El punto C es el denominado centro de la cámara o origen del sistema de coordenadas de ella. El eje z es el eje principal. El plano de la imagen se forma cuando $z = f$ y la intersección de este plano con el eje principal se denomina punto principal (p).

Se utilizan matrices homogéneas para facilitar la conversión. Siendo la matriz homogénea $x = \begin{bmatrix} x \\ y \\ w \end{bmatrix}$ la correspondiente a las coordenadas 2D de un punto cualquiera de la imagen, y la matriz homogénea $X = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ la del mismo punto de la escena 3D, la transfor-

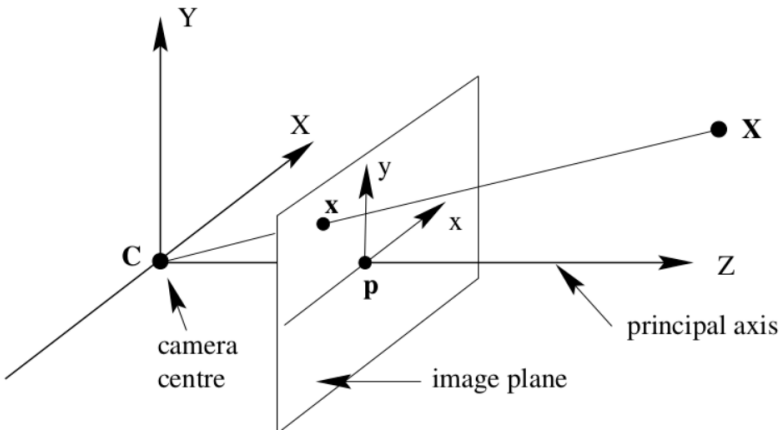


Fig. 26. Representación gráfica del modelo *Pinhole*. Fuente: https://www.researchgate.ne Aldo Von Wangenheim

mación puede ser representada por:

$$\begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & f & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ w \end{bmatrix} = x \rightarrow (f\frac{X}{Z}, f\frac{Y}{Z}) \tag{14}$$

2.2.1. Parámetros intrínsecos de una cámara.

En la ecuación anterior (14), se considera el origen de coordenadas para el plano de la imagen el punto principal p. Puede ser que esta suposición no se cumpla en ciertas aplicaciones, por lo que deberemos incluír un posible dislocamiento sabiendo que el punto principal será $p = [p_x, p_y]$

El otro problema que hay que corregir es las unidades de medida. La ecuación anterior (14) está pensada para el sistema de coordenadas de la cámara, pero para el futuro procesamiento computacional de la imagen capturada, necesitamos expresar ella en el sistema de coordenadas de la imagen, osea, en píxeles. Para ello, necesitamos introducir dos factores de conversión s_x y s_y. Además, muchas veces por defectos de la lente, se producen irregularidades en la proyección de la imagen, denominadas aberraciones o distorsiones que también han de tenerse en cuenta.

Todos estos valores que se introducen en la ecuación 14 (también aquellos que corrigen la distorsión) se denominan parámetros intrínsecos de la cámara. Queda por tanto la ecuación 14 expresada en coordenadas de imagen como:

$$\begin{bmatrix} \frac{-f}{s_x} & 0 & p_x & 0 \\ 0 & \frac{-f}{s_y} & p_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x_{im} \\ y_{im} \\ w \end{bmatrix} = X_{im} \tag{15}$$

Que equivale a:

$$\begin{bmatrix} \frac{-f}{s_x} & 0 & p_x \\ 0 & \frac{-f}{s_y} & p_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} x_{im} \\ y_{im} \\ w \end{bmatrix} \tag{16}$$

Donde la matriz $K = \begin{bmatrix} \frac{-f}{s_x} & 0 & p_x \\ 0 & \frac{-f}{s_y} & p_y \\ 0 & 0 & 1 \end{bmatrix}$ es la denominada matriz de calibración o matriz de parámetros intrínsecos.

2.2.2. Parámetros extrínsecos de la cámara.

Todos los puntos en el mundo real están descritos por un sistema de coordenadas global. Por eso, cuando representamos un punto de la escena 3D mediante la matriz X, debemos transformarla primero para que su representación este también en el sistema de coordenadas global o "del mundo"(X_w). La relación entre las coordenadas es la siguiente:

$$X = RX_w + T \tag{17}$$

Siendo $R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$ la matriz de rotación que relaciona los dos sistemas de coordenadas y $T = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$ la matriz de translación que indica cuánto se translada el punto de referencia. Los valores que componen ambas matrices son los denominadas parámetros extrínsecos de la cámara y normalmente se recogen en la denominada matriz de parámetros extrínsecos para poder rescribir la relación entre coordenadas (17) de la siguiente manera:

$$K[R|T]X_w \tag{18}$$

Reescribiendo la ecuacion 16 con la relación 18:

$$K[R|T]X_w = \begin{bmatrix} x_{im} \\ y_{im} \\ W \end{bmatrix} \tag{19}$$

$$PX_w = \begin{bmatrix} x_{im} \\ y_{im} \\ W \end{bmatrix} \tag{20}$$

Donde la matriz $P = K[R|T]$ es la matriz de proyección y es la que realmente relaciona un punto cualquiera$X$ de la escena real *3D* con el mismo punto x representado en la imagen.

2.2.3. Calibración de la cámara

La calibración de una cámara consiste principalmente en calcular los parámetros intrínsecos y extrínsecos de ella. Existen distintas técnicas de calibración o de autocalibración, pero la mayoría de ellas se basan en el modelo de cámara *Pinhole* mencionando con anterioridad. Este modelo generalmente no parametriza algunos aspectos como la profundidad del campo, la apertura de la cámara, la distancia de enfoque, la deslinearización entre el plano de la imagen y la lente, o la distorsión de las imágenes, que de a si serlo, será modelado de forma muy simplificada.

Dentro de los métodos de calibración podemos distinguir varios tipos:

- Métodos de calibración explítica e implícita:

  En la calibración explítica se obtienen los valores de cada uno de los parámetros que forman el modelo mientras que en la calibración implícita normalmente se obtienen matrices de transformación que contienen todos los parámetros sin permitir conocer el valor exacto de cada uno de ellos.

- Métodos de calibración de computación lineal y no lineal:

  La calibración de computación lineal usa técnicas de resolución de sistemas de ecuaciones lineales mientras que la calibración no lineal usa métodos iterativos.

- Métodos de calibración en un paso o multipasos:

  Denominamos calibración en un sólo paso cuando en cada ciclo del proceso de resolución se actualizan todos los parámetros en un mismo tiempo. En la multipasos, en cada fase se obtiene un conjunto de parámetros usando aproximaciones de los primeros consiguiendo tras varias iteraciones un valor ajustado.

- Métodos de calibración con patrón en un plano o múltiplo plano:

  Cuando todos los puntos del patrón se encuentran en un mismo plano se denomina calibración en un plano. En la calibración en múltiplos planos, se puede conocer la relación entre los planos o se obtienen adquisiciones sobre el mismo patrón variando la posición del mismo como por ejemplo moviendo la cámara.

Uno de los métodos de calibración más usado es el basado en la matriz de homografía H presentado por *Z.Zhang* [19], es decir una matriz de correspondencia entre puntos. Sabiendo que el modelo *Pinhole* relaciona las coordenadas globales de un punto X_w de una escena *3D* con las coordenas del mismo punto representado en el plano de la imagen X mediante la siguiente relación:

$$K[R|T]X_w = sX \tag{21}$$

Y suponiendo que todos los puntos en las coordenadas *3D* son coplanares en el modelo, se selecciona un sistema de coordenadas globales tal que el plano Z es cero, permitiendo

reescribir la ecuacion 20 como:

$$K\begin{bmatrix} r_1 & r_2 & r_3 & T \end{bmatrix}\begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} = sX \tag{22}$$

$$K\begin{bmatrix} r_1 & r_2 & T \end{bmatrix}\begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = sX \tag{23}$$

Por tanto, el punto $X = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}$ en la imagen se puede relacionar con la escena *3D* mediante una matriz de homografía:

$$H = K\begin{bmatrix} r_1 & r_2 & T \end{bmatrix} \tag{24}$$

Si definimos ahora un mínimo de cuatro puntos no colineares en el modelo, la homografía H puede ser determinada hasta cierto factor de escala λ:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \lambda K\begin{bmatrix} r_1 & r_2 & T \end{bmatrix} \tag{25}$$

Una vez obtenida la matriz de parámetros intrínsecos K, podemos obtener los parámetros extrínsecos mediante las siguientes relaciones:

$$r_1 = \lambda K^{-1} h_1 \tag{26}$$

$$r_2 = \lambda K^{-1} h_2$$

$$r_3 = r_2 x r_2$$

$$T = \lambda K^{-1} h_3$$

Donde el factor de escala λ es:

$$\lambda = \frac{1}{|K^{-1}h_1|} = \frac{1}{|K^{-1}h_2|} \tag{27}$$

2.3. Aplicaciones, algoritmos y desarollos

Una vez dado el repaso a los conceptos necesarios para entender el siguiente trabajo, en esta subsección se mencionará brevemente las principales aplicaciones de uso de los VANT'S, así como las técnicas utilizadas para conseguirlas. En última instancia se dará paso a explicar más detalladamente el algoritmo basado en odometría visual sobre el cual trabajaremos en adelante.

2.3.1. Aplicaciones autónomas para UAV's basadas en visión por computación y procesamiento de imágenes

Dentro de esta sección nos encontramos con varias aplicaciones muy interesantes como las enumeradas a continuación:

1. Aplicaciones de aterrizaje autónomo, comportamiento que puede resultar muy útil para conseguir que el drone descienda cuando la batería sea insuficiente en plataformas, por ejemplo, de carga por inducción, como en [20] resolviendo los problemas de corta duración de la batería de este tipo de vehículos. También puede resultar interesante conseguir este aterrizaje autónomo cuando el drone haya terminado de realizar una tarea concreta.

2. Aplicaciones de vigilancia como el trabajo [21], son muy interesantes debido a las ventajas que ofren frente a otro tipo de aplicaciones de videovigilancia ya que además de poder acceder a ciertas zonas poco accesibles gracias al tamaño reducido de los drones, proporcionan un amplio campo de visión y pueden moverse a una velocidad elevada en caso de necesitar algún tipo de seguimiento. Son muy populares el uso de aplicaciones de vigilancia de uso civil.

3. Aplicaciones de inspección. Este tipo de aplicaciones ya se mencionaron en este trabajo con anterioridad. Pueden ser de múltiples tipos como los trabajos de: inspección y supervisión en tareas industriales [22], inspección de infrastructuras eléctricas [23], inspección termográfica [24] o inspección agrícola [25].

2.3.2. Estimación de la posición de un VANT

Estas aplicaciones se basan en la estimación de la posición y orientación del vehículo durante su movimiento mediante uno o varios sensores integrados en el mismo, que pueden funcionar independientemente o fusionados para obtener la información necesaria para realizar esta tarea.

Dependiendo de los sensores utlizados aparecen distintas técnicas. A continuación se mostraran las más importantes o las más populares.

1. **GPS (*Global Positioning System*).** Son sistemas de navegación basados en satélites (SNS, del inglés GNSS, *Global Navigation Satellite System*). Este tipo de sistema son los más extendidos para cualquier tipo de vehículo autónomo, ya sea terreste, aéreo u subacuático. El sensor GPS normalmente se usa en sistemas de navegación como único sensor independiente [26] aunque también existen muchas otras aplicaciones que usan sistemas de navegación de GPS junto a otro tipo de sensor, como puede ser el visual [27]. A estos últimos se les denomina sistemas asistidos.

2. **Sistemas asistidos.** Este tipo de sistemas aparecen para solventar los problemas y errores que generan los GPS funcionando independientemente, es decir, son aquellos sistemas de navegación que usan el GPS junto a otro (u otros) tipo de sensores.

   Uno de los sistemas de navegación más populares de este tipo son los que usan los datos proporcionados por un sistema de nagevación inercial (INS, *Inertial Navigation System*) que usan sensores de movimiento y de rotación, junto a la información obtenida através del GPS [28]. Han sido muchas las propuestas para mejorar la eficiencia y robusted de esta técnica como el uso a mayores de la información de la cámara o el uso de algún tipo de sensor de altitud.

   Otro sistema de navegación asistido bastante conocido dentro del campo militar es el que usa el sensor "identificador amigo-enemigo" (IFF, *Identification Friend or Foe*) que no es más que un sistema de identificación criptográfica [29], junto al GPS como se explica en el trabajo [30].

3. **Sistemas basados en visión.** Estos sitemas de navegación se basan en la información obtenida por sensores visuales que se encuentran en las cámaras. Dependiendo tanto de la información visual con la que se trabaja (detección de horizontes [31], rastreo de puntos [32] o detección de bordes en la imagen [33]) como del sistema de visión utlizado (cámara monocular ([34], [35]), binocular [36] o una cámara estéreo [37], trinocular [38] o omnidireccional [39]) aparecen distintos sistemas de navegación que dan posibilidad a una infunitud de aplicaciones.

   Los primeros enfoques de este tipo de sistemas se basaron en la determinación del horizonte. Sin embargo, estas técnicas no daban buenos resultados en ambientes interiores o cuando el VANT se encontraba a un distancia bastante baja debido a la complejidad de detectar el horizonte bajo estas circustancias.

   Dos enfoques aparecieron años después para solventar este problema y conseguir un buen resultado para todo tipo de entornos. Hoy en día, estos dos enfoques siguen siendo la tendencia más popular al desarollar sistemas basados en visión por computador y se explicarán brevemente a continuación.

   a) **SLAM (*Simultaneous Localization and Mapping*).**

   Denominamos SLAM a los algoritmos de localización y mapeo simultáneo capaces de realizar la construcción de un mapa y la estimación de la posición del vehículo a la vez.

A lo largo de los años se han planteado diferentes enfoques como los mencionados a continuación:

- Algoritmo PTAM (*Parallel Tracking and Mapping*) basado en el uso de *bundle adjustment* (BA), como se explica en el artículo [40], que separa los procesos de localización y generación de mapas permitiendo ejecutarlos en tiempos real.
- MonoSLAM (*Monocular Simultaneous Localization and Mapping*) para generalmente conocido como VSLAM (*Visual Simultaneous Localization and Mapping*) cuando se trata de vehículos aéreos [41]. Como su mismo nombre indica están basados en el uso de cámaras monoculares, por lo que a diferencia de otro tipo de algoritmos precisa del movimiento de la cámara para conseguir percibir la profundidad de la escena *3D*.
- Algoritmos basados en el seguimiento de patrones y reconocmiento de lugares. Esta técnica no da buenos resultados cuando la escena el vehículo se desplaza por una escena *3D* que no se ha visitado con anterioridad al basarse en el reconocimiento de lugares, pero sin embargo es uno de los enfoques que más han triunfado a la hora de estimar el ángulo de rotación y la velocidad de translación del vehículo.
- Algoritmos de Laser-SLAM [42], que permiten obtener mapas en interiores con una precisión bastante elevada.
- LSD-SLAM (*Large- Scale Direct Monocular SLAM*) que se caracteriza por carecer de las fases de detección y decripción de características y por tanto, de la de "matching.ªl basarse en mapas de profundidad en vez de mapas de puntos en el espacio. [34] y [43].
- Algoritmo ORB-SLAM (*Oriented FAST and Rotated BRIEF SLAM*) [44] que se basa en la detección de características que serán representadas por descriptores ORB, que se explicarán más adelante.

El principal problema de este tipo de algoritmos es que son computacionalmente costos y complejos, además de la frecuente aparición de valores atípicos al obtener los datos a procesar, afectando muy negativamente en la estimación. Es por ello que han comenzado a aparecer, cada vez más, trabajos basados en el segundo enfoque a mencionar.

b) **ODOMETRÍA VISUAL.**

Estos algoritmos son aquellos que intentan estimar tanto la posición 3D del vehículo como su orientación.

La información obtenida para llevar a cabo este tipo de procedimiento puede ser obtenida por cámaras monoculares o múltiples, como se comentó con

anterioridad. Sin embargo, a diferencia de el enfoque SLAM no necesitan información de poses anteriores para realizar la estimación en tiempo real.

Dependiendo del número de cámaras usadas, distinguimos dos tipos de odometría:

- Odometría visual estereoscópica:
  Es la técnica más usada y se basa en la visión estetoscópica del ser humano (véase apéndice A), es decir se hace uso de dos imágenes capturadas por dos cámaras distintas separadas cierta distancia o por una cámara estereoscópica, que consta de dos objetivos. Con estas imágenes y detectados los puntos de correspondecia entre ellas, se realiza la técnica de triangulación para calcular su profundidad (véase apéndice B).

- Odometría visual monocular:
  Usa únicamente una cámara para capturar las imágenes, pero presenta problemas en la estimación de la posición al precisar del conocimiento previo del ángulo de la cámara en relación con el suelo. Es por esto que en este tipo de técnicas todas las distancias son relativas al primer movimiento de la cámara, osea, se toma como referencia el primero que sucede y se calculan los sucesivos a partir de este.
  Dentro de este grupo encontramos odometría visual que se basa en el uso la información de luminosidad de los píxeles o de la luminosidad en una región, odometría visual basada en características que usan la correspondencia de puntos entre imágenes consecutivas y odometría visual híbrida que mezcla las dos anteriores.

El primer trabajo de este tipo de enfoque fue presentado por Nister [45] usando una cámara estéreo para capturar dos imágenes a la cual se les aplicaba la técnica de detección de bordes mediante el uso de un filtro Harris. Una vez capturado los bordes en cada frame o imagen consecutiva, este algoritmo estimaba la posición mediante el algoritmo de 5 puntos y RANSAC (Random Sample Consensus). Este trabajo marcó en la odometría visual la distinción de varios procesos:

1) Obtención de las imágenes. Se obtienen las imágenes capturadas por la cámara o cámaras incorporadas en el vehículo del cual queremos estimar su posición.
2) Detección de características. Se detectan ciertas características de la imagen mediante técnicas de procesamiento digital de imágenes, explicadas en capítulos anteriores.
3) *Matching* de las características. Se buscan las mismas características en distintas imágenes, osea, se empareja un punto en una imágen con ese mismo punto en una imagen capturada en un instante del tiempo sucesivo.

4) Rechazo de los *outliers*. Esta etapa es la encargada de eliminar el ruído o descartar *"matches"* incorrectos, a los que denominamos *outliers*.
5) Estimación del movimiento. Esta etapa es la más importante de todo el proceso, pero depende de lo eficientes que hayan resultado las anteriores para dar buenos resultados. Se calcula el movimiento usando los puntos de *matching* y se estimará para cada par de imágenes consecutivas.

2.4. Estimación de la posición de VANTs basados en visión monocular

El algoritmo presentado se basa en el cálculo de homografías, que no son otra cosa que transformaciones que establecen una correspondencia entre elementos. Es decir, a un punto le corresponde otro punto y a una recta otra. Por ejemplo, las matrices de translación o rotación vistas en la calibración de las cámaras, no es otra cosa que matrices homográficas. Así, cualquier punto en una escena real *3D* $X_{x,y}$ puede ser proyectado en el plano de la imagen en el punto $X_{i,j}$ mediante la siguiente relación:

$$X_{i,j} = H_{0w} X_{x,y} \tag{28}$$

Donde H_{0w} es la homografía inicial conocida por homografía *world-to-frame*.

A modo de resumen, a continuación se mostrará un diagrama de flujo del algoritmo sobre el que vamos a trabajar que realiza las siguientes tareas:

Paso 1 Cálculo de la homografía inicial *world-to-frame* $\rightarrow H_{0w}$.

Paso 2 Obtener el frame anterior $\rightarrow F_{i-1}$.

Paso 3 Hacer correciones de luz y distorsión a la imagen obtenida (F_{i-1}).

Paso 4 Obtener el frame anterior $\rightarrow F_i$.

Paso 5 Hacer correciones de luz y distorsión a la imagen obtenida (F_i).

Paso 6 Detección y descripción de las características mediante el algoritmo SIFT (*Scale-Invariant Feature Transform*).

Paso 7 Cálculo de los *match points*, es decir, los puntos de correspondencia entre las dos imágenes, mediante el algoritmo *Brute-Force* $\rightarrow q$.

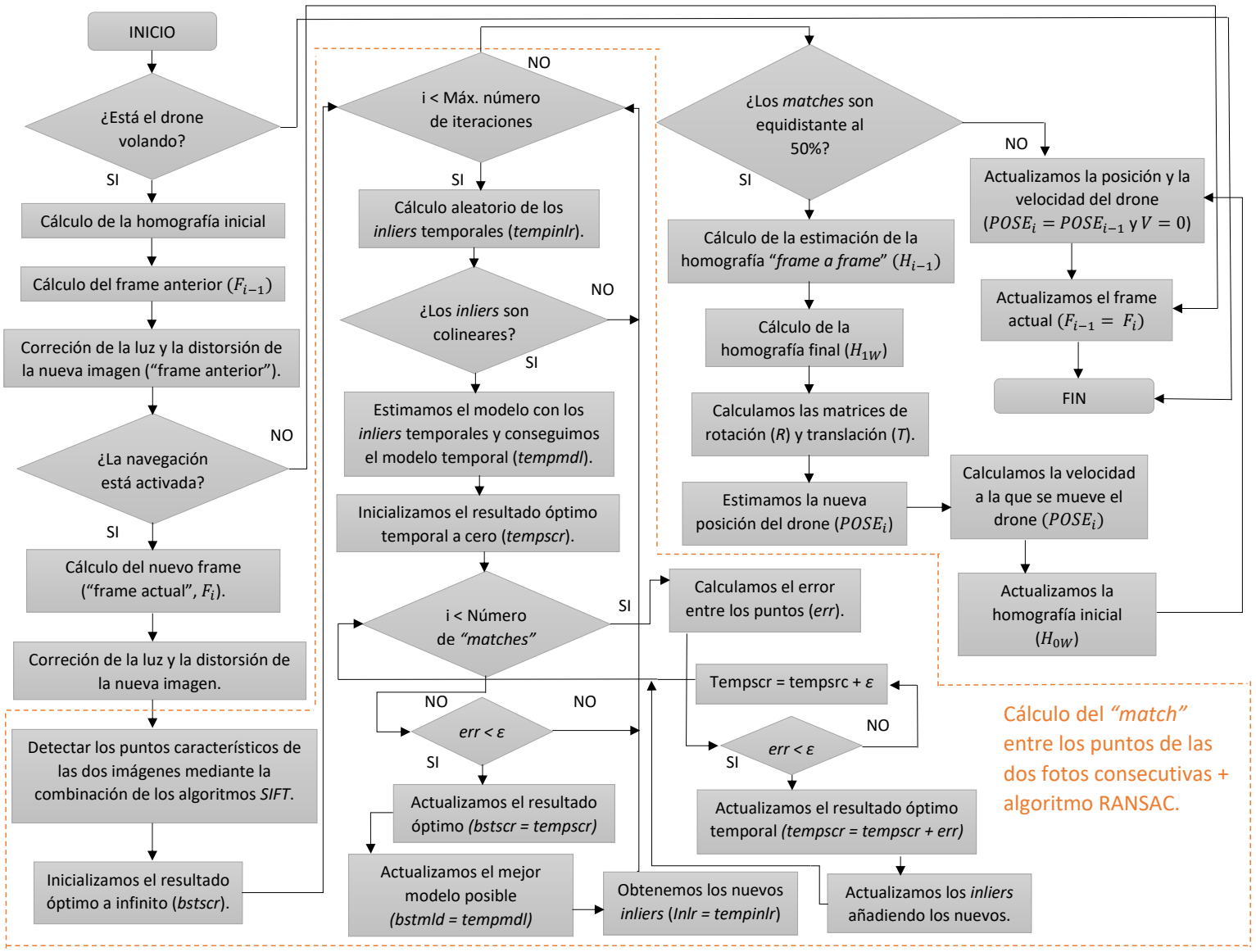
Paso 8 Filtración de los puntos de correspondencia eliminando los valores atípicos mediante el algoritmo RANSAC (*Random Sample Consensus*).

Paso 9 Cálculo de la homografía *frame-to-frame* $\rightarrow H_{i-1}$.

Paso 10 Cálculo de la homografía final *frame-to-world* $\rightarrow H_{1w}$.

Paso 9 Cálculo de las matrices de rotación y translación mediante la homografía obtenida en el paso anterior $\rightarrow R, T$.

Paso 10 Estimación de la posición y la velocidad del VANT.


INICIO
¿Está el drone volando?
SI
Cálculo de la homografía inicial
Cálculo del frame anterior (F_{i-1})
Correción de la luz y la distorsión de la nueva imagen ("frame anterior").
¿La navegación está activada?
NO
SI
Cálculo del nuevo frame ("frame actual", F_i).
Correción de la luz y la distorsión de la nueva imagen.
Detectar los puntos característicos de las dos imágenes mediante la combinación de los algoritmos *SIFT*.
Inicializamos el resultado óptimo a infinito (*bstscr*).
i < Máx. número de iteraciones
NO
SI
Cálculo aleatorio de los *inliers* temporales (*tempinlr*).
¿Los *inliers* son colineares?
NO
SI
Estimamos el modelo con los *inliers* temporales y conseguimos el modelo temporal (*tempmdl*).
Inicializamos el resultado óptimo temporal a cero (*tempscr*).
i < Número de *"matches"*
SI
NO
err < ε
NO
SI
Actualizamos el resultado óptimo *(bstscr = tempscr)*
Actualizamos el mejor modelo posible *(bstmld = tempmdl)*
Obtenemos los nuevos *inliers* (*Inlr = tempinlr*)
Calculamos el error entre los puntos (*err*).
Tempscr = tempsrc + ε
err < ε
NO
SI
Actualizamos el resultado óptimo temporal *(tempscr = tempscr + err)*
Actualizamos los *inliers* añadiendo los nuevos.
¿Los *matches* son equidistante al 50%?
NO
SI
Cálculo de la estimación de la homografía "*frame a frame*" (H_{i-1})
Cálculo de la homografía final (H_{1W})
Calculamos las matrices de rotación (*R*) y translación (*T*).
Estimamos la nueva posición del drone ($POSE_i$)
Calculamos la velocidad a la que se mueve el drone ($POSE_i$)
Actualizamos la homografía inicial (H_{0W})
Actualizamos la posición y la velocidad del drone ($POSE_i = POSE_{i-1}$ y $V = 0$)
Actualizamos el frame actual ($F_{i-1} = F_i$)
FIN
Cálculo del *"match"* entre los puntos de las dos fotos consecutivas + algoritmo RANSAC.

2.4.1. Cálculo de la homografía inicial

Lo primero que debemos calcular para poder obtener una referencia entre la relación entre el mundo real *3D* y el plano de la imagen, es la homografía inicial H_{w0}, denominada homografía "*world to reference frame*":

$$H_{w0} = \lambda C \begin{bmatrix} r_1^0 & r_2^0 & r_3^0 & t^0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \zeta \end{bmatrix} \tag{29}$$

Donde el símbolo ζ representa la altitud inicial obtenida por algún sensor o la estimada a partir del tamaño de un patrón predefinido en la posición inicial de la navegación usando el modelo de *Pinhole* (véase Apéndice A), y $\lambda = \frac{1}{s}$, siendo s un factor de escala homogéneo.

Para facilitar los cálculos, suponemos que el punto genérico q_w se encuentra en el plano en el que: $z_w = 0$, por lo que la matriz R se reduce a $R = \begin{Bmatrix} r_1^i & r_2^i \end{Bmatrix}$ y la homografía, ahora llamada homografía plana, pasa a ser:

$$\lambda C \begin{bmatrix} r_1^i & r_2^i & t^i \end{bmatrix} \tag{30}$$

Las homografías que relacionan los puntos detectados en las imágenes con sus correspondientes en el mundo real, es decir, las homografías "*frame to world*"podrán descomponerse mediante la siguiente escuación:

$$H_{wi} = H_{i-1i} H_{i-2i-1} \dots H_{01} H_{w0} \tag{31}$$

2.4.2. Correción de la luz y la distorsión

El primer paso para facilitar los cálculos es pasar las imágenes obtenidas en cada frame a escala de grises. Como ya se comentó en capítulos anteriores, una imagen en escala de grises no es más que una matriz donde el valor de cada uno de sus elementos representa la luminosidad de cada píxel.

Debido a que en ocasiones en estas imágenes el contraste de la imagen es bajo, es decir, el rango de nivel de grises se concentra en una zona del intervalo de posibles valores, detectar puntos clave (*keypoints*) que no es otra cosa que detectar aquellos que difieren del resto, por ejemplo, en niveles de luminosidad, puede resultar muy complicado. Para resolver este problema, se puede expandir el histograma de la imagen o realizar un ecualización del mismo. En este algoritmo se ha optado por expandir el rango dinámico de la intesidad de las dos imágenes antes de pasar a la detección de puntos característicos realizando una ecualización del histograma.

Es decir, dada una imagen con dimensiones MxN, se ecualizará su histograma realizando la siguiente operación:

$$I_{equhist} = \frac{L(cdf_i - cdf_{min}) - (NM)}{NM} \tag{32}$$

Siendo L el número de niveles de grises de la imagen (normalmente 256), cdf_i la frecuencia acumulada y cdf_{min} el mínimo valor de valores no nulos acumulados en la distribución, la ecualización resulta en una dispersión del histograma en un rango mayor dentro del intervalo $[0, L-1]$ de forma automática, repartiendo los valores de nivel de grises en el histograma de forma que se obtiene una distribución lo más uniforme posible.

Una vez realizada la ecualicación, se hace una corrección de la distorsión para evitar la aparición de saturación en los extremos del rango de valores de la escala de grises mediante una corrección *Gamma* sobre la imagen ecualizada.

2.4.3. Detección y descripción de los puntos característicos

La etapa de detección y descripción de los puntos característicos de la imagen no es otra cosa que la detección de aquellos puntos en una imagen que difieren de sus vecinos y que son fáciles de detectar en cualquier otra imagen pues no les afecta ciertas transformaciones como la rotación, translación, contración, expansión o algún tipo de distorsión, y una vez obtenidos, describirlos mediante un descriptor que extrae un "parche"local alrededor de ellos permitiendo que los puntos clave o *keypoints* puedan distinguirse unos de otros y se puedan comparar en las distintas imágenes.

Existen múltiples algoritmos de detección de características como son SIFT (*Scale-Invariant Feature Transform*) [46], SURF (*Speeded-Up Robust Features*) [47], FAST [48] o BRISK (*Binary Robust Invariant Scalable Keypoints*) [49] entre otros, como múltiples detectores como ORB (*Detector Oriented FAST and rotated BRIEF*) [50] o FREAK (*Fast Retina Keypoint*) [51].

Algunos algoritmos solo realizan la fase de detección de los *keypoints* como los algoritmos FAST o BRISK, otros solo realizan la fase de descripción como ORB, mientras que otros consiguen realizar las dos tareas como es el caso de los algoritmos SURF y SIFT.

A continuación se describen resumidamente los algoritmos SURF, SIFT y FREAK.

- SURF (*Speed-Up Robust Features*). Es el algoritmo más rapido debido a que usa imágenes integradas y que los *keypoints* contienen menos descriptores, pero sin embargo es menos robusto, aunque presenta buenos resultados frente a rotaciones, cambios de escala y cambios de iluminación. Se basa principalmente en dos pasos:

  1. Detección del *keypoint*: Se aplica el *Laplaciano de Gauss (LoG)* en las imágenes y después se calcula la matriz Hessiana y se usa su determinante para la localización y cálculo de la escala de los *keypoints*.
  2. Descripción del *keypoint*: Consiste en un descriptor de 64 valores obtenidos por la respuesta de las orientaciones de los puntos de interés a un filtro Haar y a la ponderación de una gaussiana con el fin de conseguir una mayor robustez frente a errores de orientación.

- SIFT (*Scale Invariant Feature Transform*). Este algoritmo transforma la imagen como un vector característico robusto frente a cambios de escala, rotaciones, traslaciones y de manera parcial a cambios de ilumninación. A diferencia de SURF, este algoritmo es lento y se basa en cuatro pasos:

  1. Detección extrema de la escala del espacio: usa para ello la diferencia de gaussianas (*Difference of Gaussians* , DoG) para identificar los puntos clave de una pirámide de escalas.
  2. Localización de Keypoint.
  3. Asignación de orientación.
  4. Descriptor de keypoint.

- FREAK (*Fast Retina detector*). Este descriptor se basa en el conocimiento del sistema visual humano, en particular, en el funcionamiento de la retina, de ahí su nombre. Este algoritmo realiza la función de descripción y describe los *keypoints* mediante una cascada de cadenas binarias. Estas cadenas se obtienen comparando las intesidades de la imagen sobre un patrón de muestreo de la retina.

En el trabajo sobre el que trabajamos se optó por el uso del algoritmo SIFT considerando que presentaba buenos resultados, proporcionando un modelo de detección invariante, es decir, robusto frente a las transformaciones de las imágenes entre los distintos frames como pueden ser la rotación, la translación, cambios de escala o cambios de iluminación.

Una vez detectados los puntos de interés y obtenidos sus descriptores, a estos se les aplica el algoritmo de *Brute-Force* para realizar la tarea de *"matching"* o emparejamiento. Este algoritmo empareja los puntos característicos entre la imagen del primer *frame* y del segundo, calculando la distancia entre el descriptor de un *keypoint* que se encuentra en la primera imagen y todos lo que pertenecen a la segunda, dando como resultado de *"match"* el más cercado.

Para calcular estas distancias, este algoritmo calcula los vecinos más cercanos a los keypoints lo que puede producir errores de *match*. Para conseguir una mayor fiabiliad en los resultados, los puntos de *match* se filtran mediante la siguiente ecuación (33) para eliminar aquellos donde la relación entre la distancia más cercana al primer descriptor de la segunda imagen y al segundo más cercano es mayor que cierto umbral, donde en este trabajo se ha definido ese valor como 0.28.

$$q = \begin{cases} (x, y, s) \quad si \quad relacion \leq 0{,}28 \\ \qquad\qquad\qquad\qquad\qquad\qquad \forall n \in N \\ 0 \qquad\qquad\qquad resto \end{cases} \tag{33}$$

Donde q representará entonces los puntos de *"match"* filtrados, s representa su tamaño, en particular, su diámetro, y N el número total de *matches"* obtenidos sin ajustar.

2.4.4. Algoritmo de ajuste RANSAC (*Random Sample Consensus*)

Dada a distintos efectos no deseados en las fotos a la hora tanto de la adquisición como de la digitalización como son: cambios de intesidad, desenfoque o borrosidad provocan la aparición de valores atípicos (fig. 27) que se arrastran a la hora de detectar *keypoints* y acaban produciendo errores a la hora de realizar la estimación de la posición. Con el objetivo de eliminar estos errores se usa el algoritmo de ajuste RANSAC ([52], [53]).

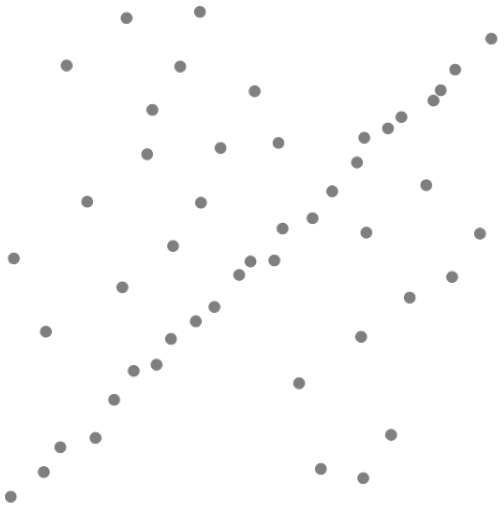


Fig. 27. Ejemplo de un modelo de datos con valores atípicos. Fuente: Wikipedia

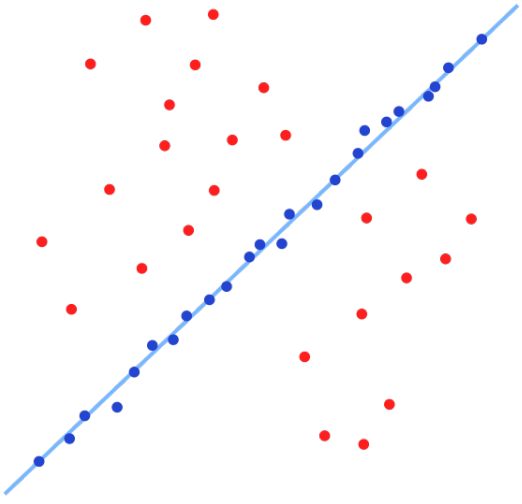


Fig. 28. *Inliers* seleccionados por el algoritmo RANSAC. Fuente: Wikipedia

Este algoritmo iterativo proporciona un modelo de datos matemático eliminando los *outliers* (fig. 28), es decir, los valores atípicos que contiene otro modelo inicial (fig. 27).

Siguiendo el diagrama de flujo que se presentó al principio de este capítulo, se puede resumir el funcionamiento del algoritmo RANSAC mediante los siguientes pasos:

1. Selección de un subconjunto de *inliers hipotéticos*.

   Se seleccionan la menor cantidad de puntos posibles dentro de todos los puntos de correspondencia encontrados q. Dado que estos puntos característicos son usados para calcular la homografía de correspondencia *"frame to frame"* H_{i-1}, se sortean aleatoriamente cuatro puntos (k) dentro de todos ellos.

   $tempinlr = Random(q, k)$

   Se evalúan si estos puntos aleatorios son colineares. De no ser así, se vuelven a calcular otros puntos aleatorios hasta que éstos lo sean.

2. Abstracción de un modelo de datos mediante los *inliers hipotéticos*.

   Dado los inliers aleatorios obtenidos, se abstrae un modelo de datos.

   $tempmdl = EstimateModel(tempinlr)$

3. Evaluación de los *keypoints*.

   El resto de los puntos de correspondencia se evalúan para saber si se ajustan al modelo estimado mediante el cálculo de la distancia de cada punto al modelo, es decir, calculando la desviación de cada punto respecto al modelo y evaluando si es menor a un umbral ε.

   $err = distance(tempmdl, q[i])$

   Si la condición se cumple se añade el punto característico correspondiente al conjunto *inliers* totales.

   $tempinlr = Add(q[i])$

4. Evaluación del modelo de datos estimado.

   Se recalcula el error con los nuevos *inliers*. Si ya se han evaluado todos los puntos y este error es menor que el mejor resultado obtenido hasta el momento, actualizamos el modelo y el resultado con los últimos obtenidos. Sino, se toma por modelo de datos el más óptimo obtenido antes de acabar de evaluar todos los puntos.

   $bstscr = tempscr$

   $bstmdl = tempmdl$

   $Inlr = tempinlr$

5. De forma opcional, se puede realizar una mejora del modelo de datos estimado.

El error de *match* entre dos puntos se realiza mediante la siguiente ecuación:

$$d^2_{transfer} = d(q_i, H^-1q_{i-1})^2 + d(q_{i-1}, Hq_i)^2 \tag{34}$$

2.4.5. Cálculo de las homografías

Una vez obtenidos los puntos de correspondecia podemos calcular la homografía que las relaciona H_{i-1} (Véase diagrama de flujo). Hemos de tener en cuenta que esta matriz puede ser modificada por escalar diferente de cero sin alterarla al que denominaremos factor de escala s.

$$sq_i = H_{i-1}q_{i-1} \tag{35}$$

Que equivale a:

$$s\begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} = \begin{bmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{bmatrix}\begin{bmatrix} x_{i-1} \\ y_{i-1} \\ 1 \end{bmatrix} \tag{36}$$

Que si multiplicamos el factor de escala s por el vector columna q_i y el vector $qi - 1$ por la homografía H, la ecuación 36 queda simplificada como:

$$\begin{bmatrix} sx_i \\ sy_i \\ s \end{bmatrix} = \begin{bmatrix} h_{11}x_{i-1} & h_{12}y_{i-1} & h_{13} \\ h_{21}x_{i-1} & h_{22}y_{i-1} & h_{23} \\ h_{31}x_{i-1} & h_{32}y_{i-1} & h_{33} \end{bmatrix} \tag{37}$$

De donde se puede deducir el valor de s:

$$s = h_{31}x_{i-1} + h_{32}y_{i-1} + h_{33} \tag{38}$$

Y sustituyendo el valor de s en las otras dos ecuaciones no homogéneas, obtenemos el siguiente sistema de ecuaciones:

$$\left.\begin{aligned} sx_i &= h_{11}x_{i-1} + h_{12}y_{i-1} + h_{13} \\ sy_i &= h_{21}x_{i-1} + h_{22}y_{i-1} + h_{23} \end{aligned}\right\} \tag{39}$$

A pesar de que estas ecuaciones no son homogéneas si que lo son los elementos de la matriz de homografía *"frame to frame"* H_{i-1} por lo que se puede representar estas dos ecuaciones de la siguiente manera:

$$\left.\begin{aligned} a_x^T h &= 0 \\ a_y^T h &= 0 \end{aligned}\right\} \tag{40}$$

Donde:

$$\begin{aligned} a_x^T &= \begin{bmatrix} x_{i-1} & y_{i-1} & 1 & 0 & 0 & 0 & -x_{i-1}x_i & -y_{i-1}x_i & -x_i \end{bmatrix} \\ a_y^T &= \begin{bmatrix} 0 & 0 & 0 & x_{i-1} & y_{i-1} & 1 & -x_{i-1}y_i & -y_{i-1}y_i & -y_i \end{bmatrix} \\ h &= \begin{bmatrix} h_{11} & h_{12} & h_{13} & h_{21} & h_{22} & h_{23} & h_{31} & h_{32} & h_{33} \end{bmatrix}^T \end{aligned}$$

Entonces para n puntos, conseguimos transformar la correspondencia en la siguiente ecuación:

$$AH = 0; H \neq 0 \tag{41}$$

Donde H representa la matriz de homografía para todos los puntos, es decir, contiene los elementos h_{ij} y A será una matriz de dimensiones $2n$ x 9 defina tal que:

$$A = \begin{bmatrix} x_{i-1_1} & y_{i-1_1} & 1 & 0 & 0 & 0 & -x_{i-1_1}x_{i_1} & -y_{i-1_1}x_{i_1} & -x_{i_1} \\ 0 & 0 & 0 & x_{i-1_1} & y_{i-1_1} & 1 & -x_{i-1_1}y_{i_1} & -y_{i-1_1}y_{i_1} & -y_{i_1} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{i-1_n} & y_{i-1_n} & 1 & 0 & 0 & 0 & -x_{i-1_n}x_{i_n} & -y_{i-1_n}x_{i_n} & -x_{i_n} \\ 0 & 0 & 0 & x_{i-1_n} & y_{i-1_n} & 1 & -x_{i-1_n}y_{i_n} & -y_{i-1_n}y_{i_n} & -y_{i_n} \end{bmatrix}$$

2.4.6. Método de SVD (*Singular Value Decomposition*)

Puede observarse que ahora el problema puede reducirse a la resolución de un sistema matricial $AX = 0$. Sabiendo que la matriz A puede descomponerse en sus valores singulares, método SVD (Véase Apéndice D.):

$$A = U\Sigma V^T = \sum_{i=1}^{9} \sigma_i u_i v_i^T \tag{42}$$

Y que el núcleo de la matriz A que es el conjunto de todos los vectores que cuya imagen bajo esta sea el vector nulo, es decir, corresponden con la homografía H, el sistema matricial anterior, puede ser planteado como un problema de autovalores de la matriz A^tA, dado que el autovector que se obtiene del menor autovalores de esta, corresponde con el núcleo de la matriz A. Es decir, el núcleo de A, y por tanto, la homografía H, corresponderá al autovector de la última columna de la matriz de autovectores V:

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

Es por ello, que la homografía se ajusta perfectamente será determinada, por lo que $\sigma_9 = 0$,

2.4.7. Estimación de la posición del drone

Siendo q_w un punto cualquiera en la escena real *3D*:

$$q_w = (x_w, y_w, z_w, 1)^T \tag{43}$$

Mediante una matriz de proyección P_e de 3x4, obtenemos la relación entre ese punto y su correspondiente en la imagen q_i:

$$sq_i = p^i q_w = C[R^i|t^i]q_w = C\begin{bmatrix} r_1^i & r_2^i & r_3^i & t^i \end{bmatrix} q_w \tag{44}$$

Siendo C la matriz de calibración de la cámara, como se explicó con anterioridad.

Con la ecuación (44) es fácil evidenciar que la homografía de correspondencia *"frame to world"* será:

$$H_{wi} = \lambda C\begin{bmatrix} r_1^i & r_2^i & r_3^i & t^T \end{bmatrix} \tag{45}$$

Donde los elementos de la matriz de rotación $R = \begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$ y los de la matriz de translación $T = t$:

$$r_1 = \lambda C^{-1} h_1$$
$$r_2 = \lambda C^{-1} h_2$$
$$r_3 = r_1 x r_2$$
$$t = \lambda C^{-1} h_3$$

Donde:

$$\lambda = \frac{1}{\|C^{-1}h_1\|} = \frac{1}{\|C^{-1}h_2\|} \tag{46}$$

- **Cálculo de la matriz de rotación *R***:

  Dado que la matriz R es ortogonal, aplicando el método de *SVD*, esta puede descomponerse en la siguiente ecuación (47) y la matriz Σ corresponderá a la matriz identidad I (48).

$$R = U\Sigma V^T \tag{47}$$

$$R' = \begin{Bmatrix} r'_1 & r'_2 & r'_3 \end{Bmatrix} = UIV^T \tag{48}$$

  Como se observa en la Figura 29, el drone puede rotar sobre los tres ejes, es decir, puede realizar tres tipos de rotaciones: rotación sobre el eje x medido a través del ángulo ϕ_{roll} denominada inclinación, rotación sobre el eje y medido a través del ángulo θ_{pitch} denominada balanceo y la rotación sobre el eje z medida por el ángulo ψ_{yaw}, por lo que la matriz R puede descomponerse en sus componentes de x,y,z.

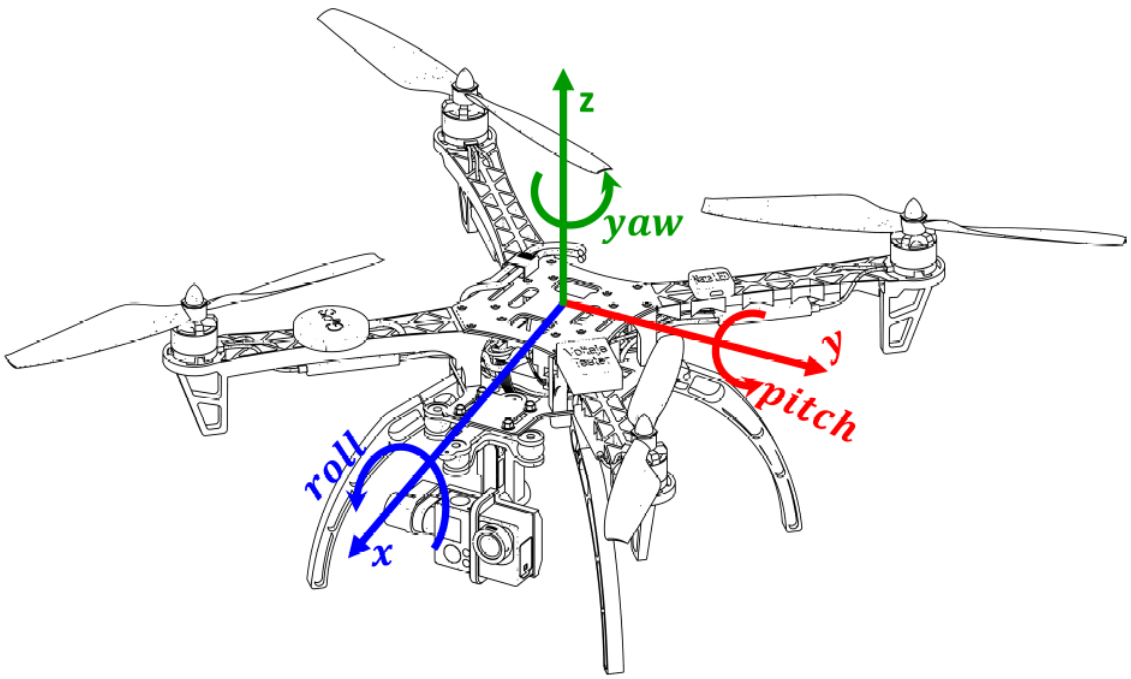


Fig. 29. Movimienos posibles en la rotación y sus respectivos ángulos.

$$R_x(\phi_{roll}) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi_{roll}) & \sin(\phi_{roll}) \\ 0 & -\sin(\phi_{roll}) & \cos(\phi_{roll}) \end{bmatrix} \tag{49}$$

$$R_y(\theta_{pitch}) = \begin{bmatrix} \cos(\theta_{pitch}) & 0 & -\sin(\theta_{pitch}) \\ 0 & 1 & 0 \\ \sin(\theta_{pitch}) & 0 & \cos(\theta_{pitch}) \end{bmatrix} \tag{50}$$

$$R_z(\phi_{yaw}) = \begin{bmatrix} \cos(\psi_{yaw}) & \sin(\psi_{yaw}) & 0 \\ -\sin(\psi_{yaw}) & \cos(\psi_{yaw}) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{51}$$

Por lo que la matriz de rotación final R_{xyz} quedará como (52) pudiendo obtener de forma aislada los ángulos implicados:

$$R_{xyz} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\phi\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\phi\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\phi\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\phi\cos\psi & \cos\phi\cos\theta \end{bmatrix} \tag{52}$$

$$\begin{aligned} \phi_{roll} &= \text{atan}_2(\sin\phi\cos\theta, \sin\phi\cos\theta) \\ \theta_{pitch} &= \text{atan}_2(-\sin\theta, \cos\phi) \\ \psi_{yaw} &= \text{atan}_2(\cos\theta\sin\psi, \cos\theta\cos\psi) \end{aligned}$$

La matriz de rotación se aproximará a (53) cuando los valores de $\cos\theta\sin\psi$ y $\cos\theta\cos\psi$ se acercan a 0 y el valor de $\cos\theta$ cae por debajo del umbral $\varepsilon = 1{,}9073e-6$:

$$R_{xyz} = \begin{bmatrix} 0 & 0 & -(\pm 1) \\ \pm\sin\phi\cos\psi - \cos\phi\sin\psi & \pm\sin\phi\sin\psi + \cos\phi\cos\psi & 0 \\ \pm\cos\phi\cos\psi + \sin\phi\sin\psi & \pm\cos\phi\sin\psi - \sin\phi\cos\psi & 0 \end{bmatrix} \tag{53}$$

En este caso resulta muy fácil obtener el ángulo ϕ_{roll} dado que los ángulos ψ_{yaw} y actuan como un único parámetro. Por ello, si igualamos cualquiera de estos dos últimos a cero la matriz de rotación se reduce a (54) y podremos obtener el ángulo de roll mediante (55). Esto es una gran ventaja a la hora de presentar muy poca sensibilidad frente a cambios abruptos en la rotación, pues cualquier cambio brusco en los valores de ϕ_{roll} se puede corregir mediante el ángulo ψ_{yaw}.

$$R_{xyz} = \begin{bmatrix} 0 & 0 & -(\pm 1) \\ \pm\sin\phi & \cos\phi & 0 \\ \pm\cos\phi & -\sin\phi & 0 \end{bmatrix} \tag{54}$$

$$\phi_{roll} = atantwo(-\sin\phi, \cos\phi) \tag{55}$$

En contraposición, cuando estas condiciones no se cumplen, los ángulos se comportan de forma independientes y los cambios abruptos pueden producir errores graves el cálculo. Para evitarlo el procedimiento más común para calcular los ángulos pertiinentes será extraer primero los ángulos de roll y pitch para proceder a calcular por último el ángulo de yaw, que es el que realmente nos interesa al ser el que nos permite observar la ruta que sigue el drone, como se indica a continuación:

$$R' = \left[\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix}\begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}\right]^T \begin{bmatrix} r'_{11} & r'_{12} & r'_{13} \\ r'_{21} & r'_{22} & r'_{23} \\ r'_{31} & r'_{32} & r'_{33} \end{bmatrix} \tag{56}$$

$$R' = \begin{bmatrix} \cos\psi & \sin\psi\sin\theta & \cos\phi\sin\theta \\ 0 & \cos\psi & -\sin\phi \\ -\sin\theta & \sin\phi\cos\theta & \cos\phi\cos\theta \end{bmatrix}\begin{bmatrix} r'_{11} & r'_{12} & r'_{13} \\ r'_{21} & r'_{22} & r'_{23} \\ r'_{31} & r'_{32} & r'_{33} \end{bmatrix} \tag{57}$$

$$R' = \begin{bmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{58}$$

Que teniendo en cuenta las ecuaciones 57 y 58, equivale a:

$$\psi_{yaw} = \text{atan}_2(\sin\psi, \cos\psi) \tag{59}$$

$$\psi_{yaw} = \text{atan}_2((\sin\psi r'_{31}, \cos\psi r'_{21}), (\sin\psi r'_{22}, \cos\psi r'_{32})) \tag{60}$$

2.5. CUDA Y OpenCL

La aparición del modelo de computación sobre una tarjeta gráfica (la GPU, denominado como device) conjuntamente con una CPU (denominado como host) de manera que forman un modelo de computación heterogéneo, aparece la necesidad de desarollar arquitecturas que permitan hacer que las tarjetas gráficas sea completamente programables para aplicaciones con caracter científico realizando la parte más costosa del cálculo mientras que la parte secuencial sobre la CPU. Además, esta arquitectura debe soportar lenguajes de alto nivel como C++, con el que trabajaremos en el presente proyecto, pues en sus comienzos era necesario el uso de lenguajes específicos para gráficos como OpenGL para usarlas. Es por ello que la empresa NVIDIA desarolla en 2009 la arquitectura CUDA (*Compute Unified Device Architecture*) para sus tarjetas gráficas que permite la computación en paralelo además de proporcionar un soporte para lenguajes de alto nivel. Otra alternativa que aparerece desarollada por Apple es la interfaz OpenCL.

2.5.1. CUDA (*Compute Unified Decive Architecture*)

La arquitectura de cálculo paralelo desarollado por Nvidia proporciona denominado CUDA permite el desarollo de algoritmos ejecutables en sus tarjetas gráficas mediante el uso de lenguajes de programación como C++, C o Fortran u otros tipo de lenguajes como Java o Python mediante un software que permite integrarlo. Esta arquitectura permite que gracias a las GPU utilizando el paralelismo que ofrecen sus múltiples núcleos (véase Apéndice D) permiten acelerar la ejecución de diferentes operaciones mediante el lanzamiento de múltiples hilos (Apéndice D) de forma simultánea que realizan tareas independientes. La forma de proceder de CUDA es:

1. Se asigna memoria en la GPU, denominado *Device*, copiando los datos de la memoria principal, CPU o *Host*, al ser estas dos independientes.
2. La CPU envía órdenes a la GPU para que realice la operación requerida.
3. La GPU ejecuta estas operaciones en cada núcleo a través de distintos hilos.
4. Los resultados se transfieren de nuevo a la memoria de la CPU y se libera la de la GPU.

2.5.2. OpenCL (*Open Computing Language*)

Siendo las plataformas heterogéneas aquellas que están formadas por una CPU y una GPU, OpenCL es una alternativa a CUDA que permite el desarollo de programas que pueden ser ejecutados en ellas, ya que cuenta con un compilador adaptativo que permite ejecutar el código desarollado en el lenguaje de programación C o C++ tanto en la CPU

como en la GPU. También puede desarollarse programas en otros lenguajes de programación pero se necesita de APIs de terceros. La mayor ventaja frente a la arquitectura CUDA es que no está diseñado para una tarjeta gráfica en concreto, sino que es compatible con un gran número de fabricantes.

3. ALGORITMO PROPUESTO

En esta sección se presentan los algoritmos propuestos para entender su funcionamiento y cómo se ha desarollado las propuestas.

3.1. Programación en CUDA

En la primera propuesta se mantiene el algoritmo de detección y decripción SIFT (*Scale Invariant Feature Transform*).

El algoritmo SIFT se basa en la diferencia entre el suavizado que se realiza a una secuencia de imagenes por filtros de gaussianas como se muestra en la Figura (30).

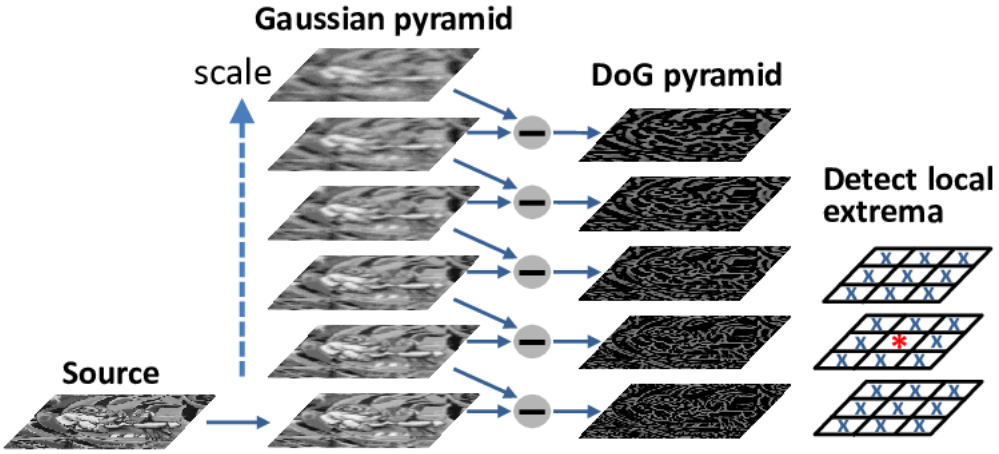


Fig. 30. Diagrama del detección SIFT. Fuente: https://www.researchgate.net Guohui Wang

Teniendo la imagen original (*source*) en escala de grises se suaviza mediante la aplicación de una función gaussiana con una varianza σ relativamente pequeña a la que denominamos escala. Esta escala cuanto mayor sea permitirá detectar regiones de interés más grandes. Al contrario, cuanto menor sea detectará cambios locales de intesidad. Una vez obtenida la primera imagen suavizada, se aumenta el valor de la escala y se vuelve a suavizar la imagen original, y así sucesivamente hasta obtener la secuencia de gaussianas requerida (primera columna de la figura (30).

El siguiente paso será realizar lo que se denomina DoG, es decir, la diferencia entre las imagenes suavizadas consecutivas. Estas imágenes destacan aquellas zonas de la imagen con irregularidades, es decir, partes con cambios bruscos de intesidad, otorgando valores negativos si se trata de una región clara frente a un fondo oscuro, valores positivos si se trata de una región oscura y valores aproximados a 0 cuando se trata de una región con valores de nivel de grises constante. Una vez obtenida la diferencias de gaussianas se pueden calcular sus máximos (oscuros) y mínimos (claros) dentro de una vecinidad de 3x3 teniendo en cuenta que para poder considerar un punto como máximo ha de serlo en su imagen suavizada a escala predecesora y en la de su sucesora. Estos máximos y mínimos serán los puntos de interés buscados, llamados *keypoints*.

Para conseguir que el algoritmo SIFT sea robusto frente a cambios de escala se introduce el concepto de pirámide de imágenes. Consiste en generar secuencias de imágenes cada vez más pequeñas y detectar en cada una de ellas los máximos y mínimos para las que tienen mayor dimensión sean las encargadas en detectar las regiones más pequeñas mientras que las imágenes más pequeñas las regiones más amplias.

El procedimiento es el siguiente:

1. Aplicamos la gaussiana a la imagen original aumentando secuencialmente la escala.
2. Se aplica la diferencia de gaussianas (DoG).
3. Se calculan los máximos y mínimos.
4. La última imagen suavizada, es decir, a la que se ha aplicado la mayor σ se reescala a la mitad para obtener el siguiente nivel.

Con la imagen reescalada se repiten los pasos anteriores hasta que no sea posible seguir disminuyendo la imagen. Además, se refina la selección de los *keypoints* mediante distintas técnicas como RANSAC.

Con este algoritmo cada punto de interés tiene asociado el nivel de pirámide y el valor de escala en el que fue detectado. Con ello, se selecciona la imagen correspondiente y se divide la región de interés detectada en nxn celdas (en nuestro caso n=4). Para cada celda se calcula el gradiente de cada uno de sus píxeles (como se observa en la Figura 31), que se puede calcular fácilmente a partir de la diferencias de sus intesidades:

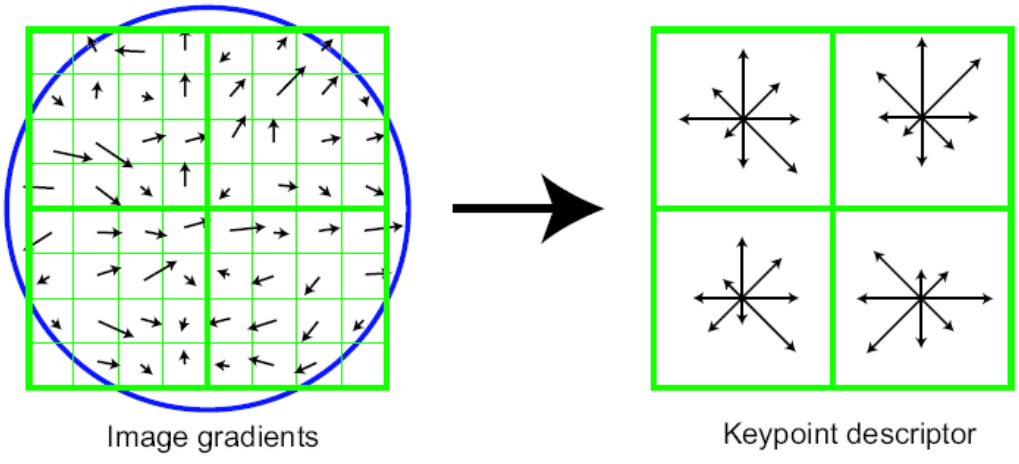


Fig. 31. Descriptor SIFT. Fuente: https://www.researchgate.net Antoine Manzanera

$$d_x = I(x+1, y) - I(x-1, y)$$

$$d_x = I(x, y+1) - I(x, y-1) \tag{61}$$

Recordemos que un gradiente es un vector que nos proporcionará la dirección donde se produce el cambio de intesidad además de la magnitud con la que sucede. Una vez obtenida las direcciones de cada píxel en esa región, se debe escoger la dirección predominante para cada celda. Para ello se divide el rango posible de grado de orientaciones (de 0 a 360) en diferentes intervalos (de 30 o 45 grados) y se asigna el gradiente de cada píxel a la que pertenezca y se calcula el histograma de orientaciones totales. Es decir, el histograma recoge la cantidad de gradientes que aparecen en cada intervalo. Para que los resultados finales sean más ajustados debemos darle más importancia a los píxeles que se encuentren más centrados dentro de la región de interés por lo que se multiplica toda la región por una gaussiana centrada en ella (círculo azul de la figura 31). Se escogerá como orientación dominante de la región al máximo global del histograma y orientaciones dominantes alternativas a aquellos máximos locales de al menos un 80 % del máximo global.

El descriptor final concatena el histograma de cada celda, que en el caso de n=4, serán 4x4 histrogramas que contiene a su vez 8 intervalos (si los intervalores escogidos para divir el rango de orientaciones fue de 45 grados) será un vector de 128 valores (Figura 31 resultado a la derecha). Además, estos histogramas normalmente se normalizan para evitar producir errores frente a cambios de contraste.

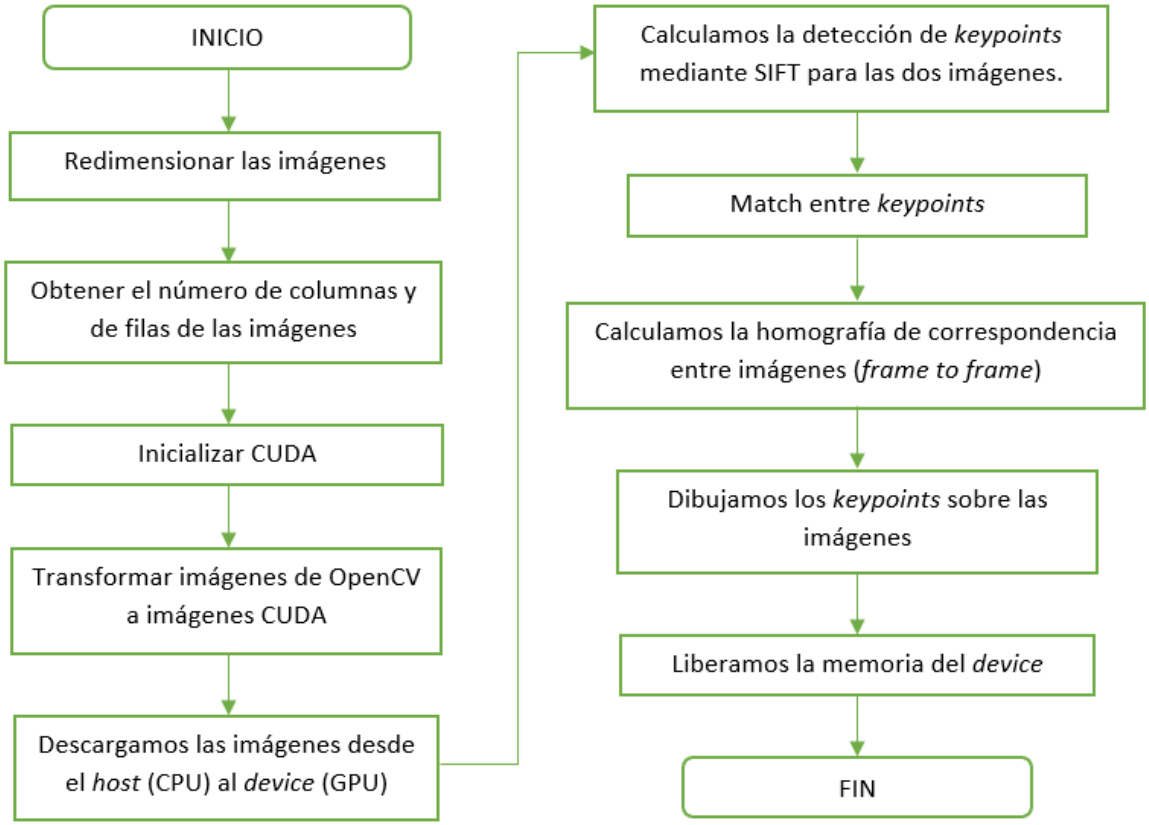


Fig. 32. Diagrama de flujo de SIFT en CUDA.

OpenCV no proporciona un herramienta para utilizar el algoritmo directamente sobre CUDA por lo que se tendrá que programar manualmente. Los pasos a seguir son los siguientes:

1. REDIMENSIONAR LAS IMÁGENES

   Se redimensionan las imágenes para poder adaptarlas a la forma necesaria de las imágenes CUDA mediante el método de OpenCV "convertTo":

   ```
   cv::Mat imgCPU;
   frame1.convertTo(imgCPU,CV_32FC1);
   ```

2. INICIALIZAR CUDA

   Es lo primero que ha de realizarse si queremos trabajar en CUDA sobre la GPU. En este caso se ha realizado bajo el método "InitCuda":

   ```
   InitCuda(devNum);
   ```

3. TRANSFORMAR IMÁGENES

   Se declaran las imágenes CUDA (de tipo CudaImage) y se transforman aquellas imágenes ya redimensionadas correspondientes a cada frame gracias a la función "Allocate" de CUDA que nos permite asignar memoria para la imagen en la GPU:

   ```
   CudaImage imgCUDA;
   imgCUDA.Allocate(w, h, iAlignUp(w, 128), false, NULL, (float*)limg.data);
   ```

4. DESCARGAMOS LAS IMÁGENES AL DEVICE

   Una vez asignada la memoria en la GPU, simplemente debemos "descargarla" para transferirla del host (CPU) al device (tarjeta gráfica). Para ello CUDA ofrece el método "Download":

   ```
   img1.Download();
   ```

5. ALGORITMO SIFT

   Se aplica manualmente el algoritmo SIFT. Este algoritmo ha sido desarollado bajo los métodos *InitSiftData* y *ExtractSift*

6. CÁLCULO DEL *MATCHING*

   Se calcula el *matching* entre los puntos de interés de los frames sobre la GPU. Se ha realizado en el método *MatchSiftData* manualmente.

7. CÁLCULO DE LA HOMOGRAFÍA

   El siguiente paso, una vez obtenidos los puntos de correspondecia ha sido calcular la homografía de correspondencia *frame to frame*. Se ha realizado mediante el método "FindHomography" y después se ha ajustado eliminando los *outliers* en el método "ImproveHomography".

```
float* homography = new float[9];
int numMatches;
FindHomography(siftData1, homography, &numMatches, 10000, 0.98f, 1.0f, 5.0);
int numFit = ImproveHomography(siftData1, homography, 5, 0.98f, 1.0f, 3.0);
```

8. REPRESENTACIÓN DE LOS *KEYPOINTS*

   A continuación se han dibujado los *keypoints* sobre la imagen original en la CPU. Para ello ha sido necesario volver a redimensionar la imagen:

   ```
   clonedFrame1.convertTo(clonedFrame1,CV_8UC1);
   ```

9. LIBERAMOS LA MEMORIA DEVICE

   Como ya no trabajaremos más con CUDA en el algoritmo, debemos de liberar la memoria del device. Para ello, se ha utilizado la función "FreeSiftData":

   ```
   FreeSiftData(siftData1);
   ```

Aunque no se muestra en el diagrama de flujo, este algoritmo presentado devuelve la homografía final obtenida *frame to frame* para calcular a continuación la homografía *frame to world*:

```
return homography;
```

3.2. SURF en CUDA

El algoritmo SURF(*Speeded-Up Robust Features*) está basado en SIFT y hace uso de la matriz hessiana (ecuación 62) ya que con la segunda derivada podemos obtener el suavizado de la imagen y los máximos y míninos a la vez.

$$H_{x,y} = \begin{bmatrix} L_{x,x} & L_{x,y} \\ L_{x,y} & L_{y,y} \end{bmatrix} \tag{62}$$

Siendo la transformación L el Laplaciano, $L_{x,x}$ representa los cambios de intesidad en el eje horizontal, $L_{y,y}$ en el eje vertical y $L_{x,y}$ en las diagonales, mientras que para obtener una información general sobre los cambios locales de un punto x se calcula el determinante de la matriz hessiana $H_{x,y}$.

Calcular el Laplaciano para cada píxel tiene una gran carga computacional por lo que con el fin de reducir tiempos de ejecución, este algoritmo utiliza derivadas aproximadas. Es por ello que para este algoritmo se utilizan imágenes integradas, que no es otra cosa que las imágenes transformadas para que el valor de cada uno de sus píxeles sea la suma de todos los píxeles en la imagen original situados en su esquina superior izquierda. Esta transformación es rápida porque sólo necesita un único recorrido por cada fila:

$$I(x, y-1) + s(x, y) \tag{63}$$

Siendo $s(x, y)$ los valores de la fila actual a la izquierda del píxel.

La gran particularidad de este tipo de imágenes es que permiten calcular la suma de intesidad de cualquier región con forma rectangular en tiempos de ejecucción reducidos. Siendo la vecinidad rectangular R y p_1, p_2, p_3 y p_4 los píxeles que la definen, la suma de las intesidades que abarca su área se puede calcular mediante:

$$R = I(p_4) - I(p_2) - I(p_3) + I(p_1) \tag{64}$$

Por tanto el algoritmo SURF permite encontrar los puntos de interés en tan solo cuatro pasos:

1. Se transforma la imagen original a su correspondiente imagen integrada.
2. Se calcula la convolución entre la imagen integrada y un filtro que simula el resultado de un filtro gaussiano para suavizar la imagen. Esto se consigue binarizando el gaussiano, donde los valores más oscuros se corresponden a valores positivos, los valores claros a valores negativos y los más grisáceos a 0 al ser los valores de transición.
3. A diferencia del algoritmo SIFT, en este caso se reescalará el filtro utilizado y se irá realizando la convolución con los filtros con dimensión cada vez mayor como se muestra en la Figura 33. Por lo general se calculan 3 niveles.

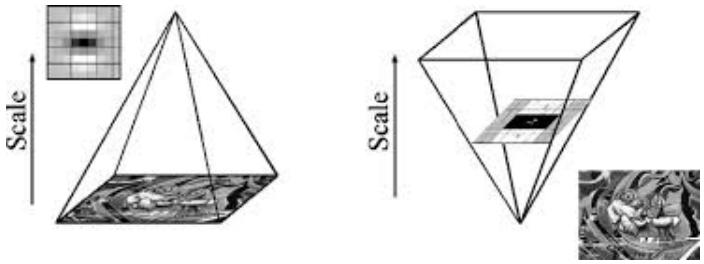


Fig. 33. Escalado en SURF. Fuente: https://dsp.stackexchange.com

4. Calculamos el determinante de la matriz Hessiana ya explicada para obtener los puntos de interés. Se considerarán *keypoints* aquellos puntos máximos que lo sean en los 3 niveles de escala.
5. Una vez obtenidos los puntos de interés se usan los filtros Haar para obtener la orientación dominante. Para calcular la orientación de cada [keypoint] se selecciona la imagen con la escala σ donde se detectó y tras aplicar estos filtros binarios seleccionamos alrededor de este una circunferencia de tamaño 6σ (Figura 34). En esta vecinidad circular realizamos para cada intervalo de 60 grados la suma de todos los vectores de respuesta a los filtros Haar que lo forman para obtener un único vector por intervalo. La orientación dominante para esta vecinidad será aquel que tenga mayor magnitud entre todos los vectores de los intervalos.

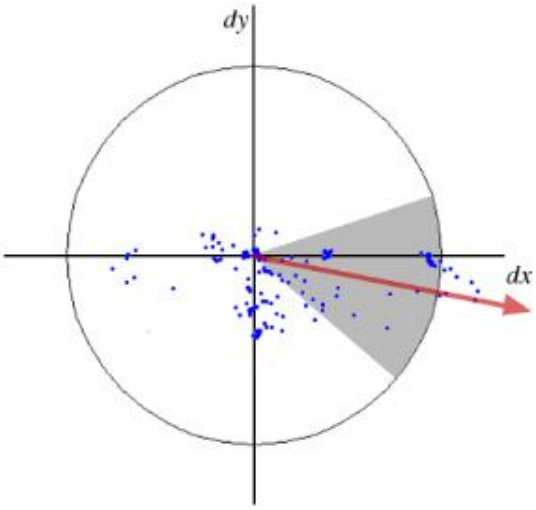


Fig. 34. Circunferencia para definir las orientaciones SURF. Fuente: https://docs.opencv.org

Tras obtener la dirección de la orientación dominante, se centra ahora una región cuadrada centrada en el punto de interés de tamaño 30σ y se divide en 16 celdas sobre las que se aplicaran de nuevo los filtros Haar para obtener los valores sobre el eje x (h_x) y eje y (h_y) y además de sus valores absolutos. Así obtenemos 64 valores que corresponderán al descriptor: 16 valores h_x + 16 $|h_x|$ + 16 valores h_y y 16 $|h_y|$ (35). Se realiza las sumas h_x y h_x por separado y se normalizan los valores para que esta sea la unidad.

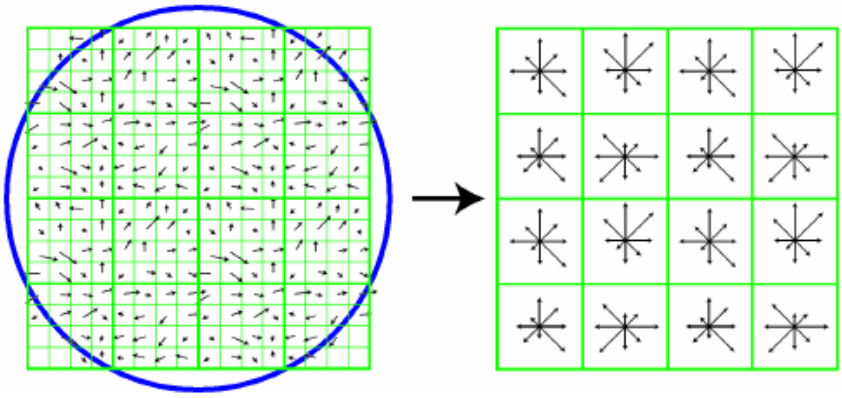


Fig. 35. Descriptor SURF. Fuente: https://www.researchgate.net Nguyen-Khang Pham

A diferencia con el algoritmo SIFT, OpenCV si proporciona una clase para trabajar con el algoritmo SURF sobre CUDA. Esta es la **cv::cuda::SURF_CUDA**.

Plara incluir esta clase en nuestro código el esquema seguido fue el siguiente (Figura 36) que se explicará paso a paso:

1. CONVERSIÓN DE IMÁGENES

   Como vamos a trabajar sobre CUDA debemos realizar la conversión de las imágenes OpenCV (almacenadas en la CPU, es decir, es el *host*) a imágenes que tra-

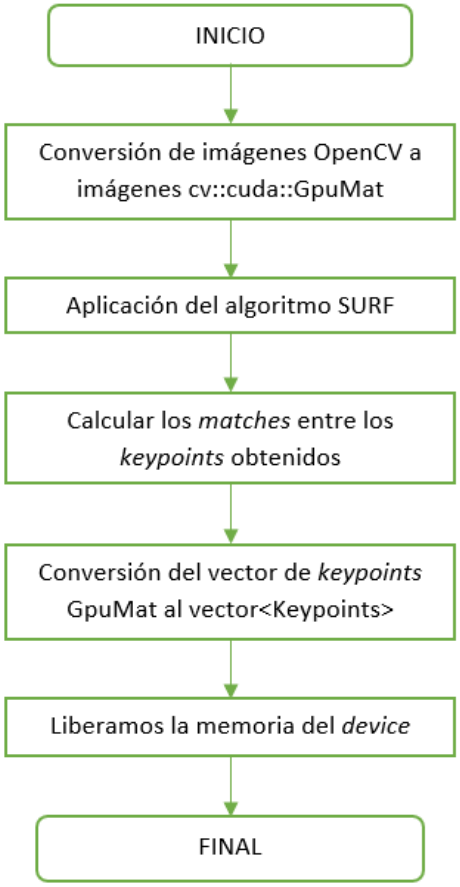


Fig. 36. Diagrama de flujo de SURF bajo CUDA.

bajen sobre CUDA (almacenadas sobre la GPU, es decir, el *device*). Las imágenes OpenCV son de tipo cv::Mat y las otras cv::cuda::GpuMat. La clase de OpenCV SURF_CUDA proporciona un método para realizar esta transformación de subir las imágenes desde la CPU a la GPU:

```
GpuMat imgGPU;
img1.upload(imgOpenCV);
```

2. ALGORITMO SURF

   Una vez realizada la conversión de las imágenes correspondientes a cada frame consecutivo, aplicamos el algoritmo surf. Este nos devolverá los puntos de interés (*keypoints*) y sus descriptores de una vez.

```
SURF_CUDA surf;
GpuMat keypoints1GPU, keypoints2GPU;
GpuMat descriptors1GPU, descriptors2GPU;
surf(imgGPU, GpuMat(), keypointsGPU, descriptorsGPU);
```

3. CÁLCULO DE *MATCHING*

Se calcula la correspondencia entre los puntos de interés obtenidos en cada imagen (*matching*).

```
Ptr<DescriptorMatcher>matcher = DescriptorMatcher::createBFMatcher(surf.defaultNorm());
vector<DMatch>matches;
matcher->match(descriptors1GPU, descriptors2GPU, matches);
```

Con estos matches son los que se calculará posteriormente la homografía de correspondencia.

4. CONVERSIÓN DE VECTORES

   Así como al principio tuvimos que convertir las imágenes CPU en imágenes GPU, ahora debemos de transformar los datos que trabajan sobre la GPU (GpuMat) a sus equivalentes en la CPU (vector<KeyPoint>, vector<float>).

```
vector<KeyPoint>keypoints1;
vector<float>descriptors1;
 surf.downloadKeypoints(keypoints1GPU, keypoints1);
surf.downloadDescriptors(descriptors1GPU, descriptors1);
```

5. LIBERACIÓN DE MEMORIA

   Como no vamos a usar más los datos sobre la GPU debemos liberar la memoria:

```
releaseMemory();
```

4. EXPOSICIÓN DE SOLUCIONES

4.1. Escenarios

Para obtener los resultados de este trabajo se han utilizado X archivos de extensión .bag, es decir, un formato de archivo en ROS (*Robot Operating System*), SDK (*Software Development Kit*) que se presentará más adelante, que contendrán las imágenes de prueba que simulan un vuelo de un drone a través de mensajes de ROS.

4.2. Plataformas y materiales

Este trabajo ha sido desarollado en lenguaje de programación C++. A continuación se presenta las plataformas más importantes además de la ya comentada arquitectura CUDA que han permitido su ejecucción además de los materiales utilizados.

4.2.1. Libería OpenCV

La libería OpenCV (*Open Source Computer Vision Library*) es una libería de código abierto desarollada por Intel en 1999 de visión por computador usado para diferentes aplicaciones tanto comerciales al permitir realizar procesamientos de CV (*Compute Vision*) como reconocimiento de patrones o calibración de cámaras en tiempo real.

Es una libería multiplataforma al ser compatible con diferentes sistemas operativos como GNU/Linux, Mac OS X y Microsoft Windows e incluso Android, escrita en los lenguajes de programación C y C++ además de disponer de interfaces que permiten su compatibilidad con Python, MATLAB y Ruby entre otros.

OpenCV trabajando con imágenes:

Esta libería permite trabajar con los siguientes formatos de imágenes:

- JPEG (.jpeg, .jpg, .jpe).
- Portable Network Graphics (.png).
- Windows bitmaps (.bmp, .dib).
- Sun Rasters (.ras, .sr).
- JPEG 2000 (.jp2).
- Portable image formats (.pbm, .ppm, .pgm).
- WebP (.webp).
- TIFF (.tif, .tiff).

Estas imágenes son representadas mediante la clase *cv::Mat* que permite almacenar y manipular los píxeles de una imagen. Dado que la imagen digital no es más que una matriz de M filas y N columnas de píxeles, el constructor de esta clase es el siguiente:

```
cv::Mat::Mat(int filas, int columnas, int tipo)
```

Donde se indican las dimensiones de la imagen y el tipo tendrá la forma

$$\text{CV\_tamañoBit}[U|S|F]\text{CNumeroCanales}$$

Donde las letras *U, S y F* indican el tipo de datos:

$$U \rightarrow \text{sin signo (0, 255)}$$
$$S \rightarrow \text{con signo (-127, 127)}$$
$$F \rightarrow \text{dato flotante}$$

En este proyecto se ha trabajo con la **versión OpenCV 3.1.0**.

4.2.2. ROS (*Robot Operating System*)

ROS (*Robot Operating System*) es un SDK de código abierto desarollado en 2007 por el Laboratorio de Inteligencia Artificial de Stanford y que en 2008 continuó desarrollándose en un instituto de investigación robótico en Willow Garage, que a pesar de no ser un sistema operativo ofrece los servicios más comunes de estos. Además presenta diferentes herramientas que permiten escribir, compilar y ejecutar código comunicando varios computadores. Actualmente, pertenece a la *Open Source Robotics Foundation* y funciona bajo licencia BSD (*Berkeley Software Distribution*). ROS está pensado para funcionar en un sistema UNIX, aunque se está adaptando a otro tipo de sistemas operativos.

ROS funciona bajo el modelo de "publicación/subscripción", es decir, se compone de múltiples nodos independientes que son los encargados de realizar la operaciones pertinentes que proporcionan servicios que permiten interactuar con otros nodos. Para ello, el nodo que deseamos que interactue con otros se publica en los denominados tópicos ROS que estarán subscritos a otros nodos. Esta comunicación se realiza mendiante mensajes ROS, que no es más que un conjuto de datos compuestos.

4.2.3. Materiales

- La CPU utilizada: Procesador Intel® Core™ i7-7700HQ CPU @ 2.80GHz
- La GPU utilizada: Nvidia Copyrigth 2005-2017 GeForce GTX 1050
- Sistema operativo Ubuntu 16.04.

- CUDA 9.1 v9.1.85: CUDA Driver CUDART, CUDA Driver Version 9.2, CUDA Runtime Version 9.1, NumDevs 1.
- OpenCV 3.3.1
- ROS Kinetic

4.3. Resultados

Existen por tanto dos diferencias notables a tener en cuenta a la hora de trabajar con SIFT o SURF sobre CUDA:

- OpenCV proporciona el algoritmo SURF_CUDA para poder trabajar con SURF sobre CUDA directamente mientras que no proporciona ningún algoritmo para trabajar con SIFT directamente sobre CUDA, por lo que ha de programarse manualmente.
- El cálculo de la homografía de correspondencia *frame to frame* se realiza sobre la GPU en el algoritmo SIFT programado en CUDA mientras que cuando trabajamos con el algoritmo SURF_CUDA sólo se realizan sobre la tarjeta gráfica la detección, descripción y *matching* de los puntos de interés, lo cual puede afectar también a los tiempos de ejeccución. Esto afectará principalmente cuando se comparen los resultados obtenidos para el algoritmo completo basado en odometría visual monocular.

Es por ello que los tiempos de ejecucción obtenidos varian bastante de un método a otro a pesar que los dos trabajan sobre la GPU y de que que teóricamente el algoritmo SURF debería ser más rápido.

Además, debemos tener claro dos cuestiones. En primer lugar los resultados obtenidos dependen directamente de la tarjeta gráfica y el procesador central unitario utilizado dado que no todos tienen el mismo rendimiento. También dependerá de la eficiencia de escritura del código. En segundo lugar, todos los procesadores se suelen encontrar en un estado de *standby* si no se hace uso de ellos por lo que las primeras ejecuciones suelen conllevar a tiempos de ejecución más elevados, como se puede comprobar en la Tabla 2. Es por ello que normalmente para obtener unos resultados definitivos se suelen realizar un par de ejecuciones previas de calentamiento. En nuestro caso, se han realizado 10 ejecuciones para obtener un resultado final (Tabla 3).

Los datos recogidos en estas dos primeras tablas (Tabla 2, Tabla 3) hacen referencia a los tiempos de ejecucción de media en relación con las operaciones de detección y descripción de características (algoritmos SIFT o SURF) y el *matching* de estos *keypoints* entre frame y frame consecutivo. Es decir, recoge el tiempo que nos interesa para saber cuál es el algoritmo más eficiente en cuestión de tiempos de ejucción sin tener en cuenta la robustez y los resultados que presentan para la estimación de la posición final.

Procedimiento	Tiempo en ms
SIFT CPU	887.9ms
SIFT en CUDA	156.8ms
SURF_CUDA	223.5ms

TABLA 2. COMPARACIÓN DE TIEMPO DE EJECCUCIÓN PRIMERA EJECUCIÓN.

Procedimiento	Tiempo en ms
SIFT CPU	881.2ms
SIFT en CUDA	137.3ms
SURF_CUDA	196.8ms

TABLA 3. COMPARACIÓN DE TIEMPO DE EJECCUCIÓN TRAS 10 EJECUCIONES.

A diferencia con el algoritmo procesado únicamente sobre la CPU, debemos tener en cuenta que se requiere de cierto tiempo de carga de las imágenes sobre la GPU por lo que no obtendremos la misma eficiencia para un recorrido largo del drone, donde se necesitará el procesamiento de un número elevado de imágenes que para un recorrido muy pequeño dado que no compensa asumir el tiempo de carga. Esto se traduce de forma general de la siguiente manera (Figura 37):

- Procesamiento sobre CPU: Aumento en el tiempo de ejecución de manera exponencial al aumentar el número de imágenes a procesar.
- Procesamiento sobre CUDA: Tanto al tratarse de SIFT o SURF bajo CUDA, estos algoritmos que trabajan sobre la GPU aumentan su tiempo de ejecución de manera progresiva al aumentar el número de imágenes a procesar.

Para analizar como afectan estas propuesta de manera general en el algoritmo de odometría visual se observan los frames por segundo que presenta cada algoritmo (Tabla 4)

Procedimiento	Frames por segundo
Algoritmo original	5fps
SIFT en CUDA	15fps
SURF_CUDA	11fps

TABLA 4. COMPARACIÓN DE FRAMES POR SEGUNDOS FINAL.

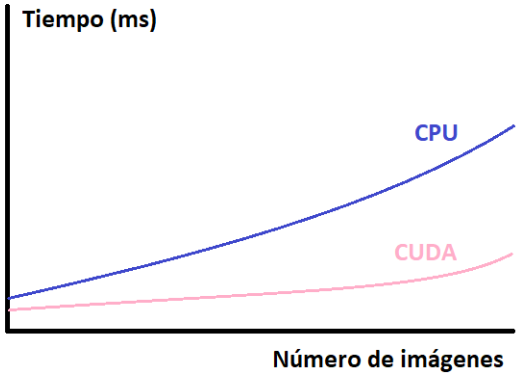


Fig. 37. Tendencia en los tiempos de ejecución sobre cada procesador.

Como era de esperar si el algoritmo SIFT en CUDA era el que producía unos tiempos de ejecucción menores, también es el que permite trabajar con mayor número de frames por segundo (fps), es decir, en cada segundo maneja 15 imágenes frente a 11 que maneja el algoritmo SURF en CUDA y 5 que manejaba el algoritmo original con SIFT sobre la CPU, lo que se traduce en que el algoritmo final basado en odometría visual terminará de ejecutarse antes y por lo tanto, será más rápido.

5. PRESUPUESTO DEL PROYECTO

En esta sección se presentará el presupuesto estimado para realizar este proyecto, teniendo en consideración los siguientes requisitos:

- Se ha tomado como referencia el suelo medio de un ingeniero de telecomunicaciones junior de 18 €/hora, para estimar cuánto sería el presupuesto si el trabajo fuese realizado para una aplicación real.
- Se ha tomado como referencia el sueldo de un ingeniero senior de 45 €/hora, para dar a estimar el coste del tiempo que invertido por el tutor.
- Las horas invertidas en este proyecto han sido estimadas aproximadamente y desplegadas según las diferentes tareas realizadas en la Tabla 5.
- Se ha usado un ordenador portátil Gaming- Asus GL553VD-DM065T, 15.6"Full HD Intel® Core™ i7-7700HQ, 8 GB RAM, 1 TB HDD + 128 GB SSD, GTX1050 con sistema operativo Windows 10 Original de 64 bits integrado con valor de 1149 €(Fuente: MediaMarkt).
- Se ha tomado como referencia el precio de la tarjeta gráfica Nvidia GeForce GTX 1050 (Fuente Amazon).
- Tomamos como referencia el drone OOFAY Drone con Cámara X350 Quadcopter 3D sin escobillas de seis canales con FPV Drone remoto aéreo vivo (Fuente Amazon).
- Tanto el sistema operativo Ubuntu 16.04 LTS, como OpenCV 3.3.1, ROS Kinect (licencia BSD), CUDA 9.1 y el editor de texto LaTex son gratuitos.
- Se han calculado de forma aproximada las horas de consumo de energía por uso de un ordenador pórtatil teniendo como referencia el modelo de cargador “Original Slim Asus PA-1121-28 ROG G550 G550JK Adaptador Cargador” que consume 120W.
- Para poder calcular la factura de energía que supone este proyecto, se ha tenido en cuenta que los precios por kWh de la empresa Endesa es de 0,146632 €/kWh.
- Se han calculado de forma aproximada los costes indirectos y se han desplegado en la Tabla 6.
- La suma total del presupuesto se muestra en la Tabla 7.

Teniendo en cuenta de que entre el total de las horas trabajadas, un 95 % han sido haciendo uso de un ordenador portátil de 120 W:

FASES	Trabajo personal	Trabajo tutor
Planteamiento del proyecto	8 horas	4 horas
Instalación del entorno requerido	8 horas	14 horas
Estudio de diferentes tecnología	130 horas	6 horas
Desarollo del código planteado	80 horas	10 horas
Pruebas y control de errores	15 horas	4 horas
Conclusiones	12 horas	1 hora
Escritura final de la memoria	100 horas	2 horas
Entrega de la memoria	1 hora	0 horas
Total de horas	354 horas	41 horas

TABLA 5. HORAS TRABAJADAS.

$$\frac{120W}{1000} \times [(354 + 41) \times 0{,}95)]horas = 45{,}03kWh$$
$$45{,}03kWh \times 0{,}146632€/kWh = 6{,}60284 \approx 7€$$

*Internet: Consideramos un tarifa de Internet de 30 €/mes.

*Alquiler de un local: Consideramos el alquiler en Madrid de un local por 600 €/mes. El proyecto dura 9 meses.

*Materiales: Folios, bolígrafos, lápices, gomas, fotocopias...

Factores	Precio aproximado
Energía	7 €
Alquiler de local*	5400 €
Internet*	270 €
Materiales*	10 €
Otros	10 €
TOTAL	5697 €

TABLA 6. COSTES INDIRECTOS.

Y los sueldos de cada ingeniero y las horas trabajadas estimadas:

$$354 horas \times 10€/hora = 3540€$$

$$41 horas \times 50€/hora = 2050€$$

Factores	**Precio final**
Ingeniero Junior	6372 €
Ingeniero Senior	1845 €
Ordenador portátil	1149 €
GPU	180 €
Drone prestado	370 €
Ubuntu 16.04 LTS	Gratuito
OpenCV 3.3.1	Gratuito
ROS Kinect	Gratuito
CUDA 9.1	Gratuito
LaTex	Gratuito
Costes indirectos	5697 €
TOTAL	15613 €

TABLA 7. PRESUPUESTO FINAL.

6. CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

Una vez analizados los resultados se puede decir que en cuestión de tiempos de ejecución el algoritmo más eficiente es el algoritmo SIFT programado bajo CUDA. Sin embargo, el algoritmo SURF_CUDA tampoco genera tiempos elevados, por lo que podría ser una solución totalmente factible.

Hay que tener en cuenta que estas dos propuestas están pensadas para reducir el tiempo de ejecucción de la parte de detección y descripción de características y *matching* únicamente, sin tener en cuenta cómo afecta sus resultados a la estimación de posición y orientación final del VANT.

Teóricamente, SURF omite ciertos detalles en la imagen que sí que son detectados por el algoritmo SIFT y por ello procesa menos puntos característicos generando unos tiempos de ejeccución menores. Sin embargo, nuestros resultados muestran lo contrario. Esto puede deberse a varios factores, entre otros, a la resolución de las imágenes. Como es lógico, cuanto menor sea la resolución de la imágen, menor número de píxeles que la componen lo que conlleva a menor número de *keypoints* detectados y por tanto menor tiempo de ejecucción, pero sin embargo, los algoritmos pueden funcionar mejor dentro de un rango de resolución en concreto que no tienen porque coincidir. Además también puede afectar el número de imágenes utilizadas. Aunque se presentó el gráfico 37 a modo de idea general, dentro de la línea CUDA no funcionan exactamente igual el algoritmo SURF que el algoritmo SIFT.

Además, por lo general el algoritmo SURF funciona mejor frente a rotaciones y desenfoques pero es menos robusto frente a cambios de iluminación. Teniendo esto en cuenta, quedará a cargo de la persona a cargo de la aplicación final a la que se podría aplicar este proyecto de escoger un algoritmo u otro atendiendo a lo que más le interesa en cada caso.

6.2. Trabajo futuro

- Reescribir todo el código introduciendo la arquitectura CUDA y no sólo de forma aislada para el algoritmo SIFT además de el matching de las características y la homografía frame-to-frame.
- Introducir la arquitectura OpenCL (Open Computing Language) como alternativa a CUDA, que permite el desarollo de programas que pueden ser ejecutados en cualquier unidad de procesamiento, ya que cuenta con un compilador adaptativo que permite ejecutar el código desarollado en el lenguaje de programación C o C++ tanto en la CPU como en la GPU. La mayor ventaja frente a la arquitectura CUDA es que no está diseñado para una tarjeta gráfica en concreto, sino que es compatible con un gran número de fabricantes.

- Sustituir el algoritmo de detección de características SIFT por el algoritmo FREAK (Fast Retina Keypoint) que se basa en el conocimiento del sistema visual humano, en particular, en el funcionamiento de la retina, de ahí su nombre. Este algoritmo realiza la función de descripción y describe los *keypoints* mediante una cascada de cadenas binarias. Estas cadenas se obtienen comparando las intesidades de la imagen sobre un patrón de muestreo de la retina. Este algoritmo, implementado también en OpenCV, al necesitar menor carga de memoria es más rápido pero sin embargo presenta peores resultados que los algoritmos SIFT o SURF.

- Reemplazar la librería OpenCV, usada en el procesamiento de las imágenes necesarias para llevar a cabo la estimación de posición y orientación del vehículo, por la librería ArrayFire. Está libería adaptada para el desarrollo de programas en lenguajes de programación C++, C, Fortran, Python, Java o R, es además compatible con CUDA y OpenCL y permite por tanto, trabajar tanto sobre la CPU como la GPU. Esta librería contiene la tecnología *GFOR* que permite realizar sobre la GPU iteraciones en bucle, lo que resulta muy interesante a la hora de realizar cualquier tipo de procesamiento de imagen dado que estas se almacen en ellas. Proporciona además, funciones que permiten ejecutar operaciones aritméticas, de álgebra lineal, estadísticas con mayor rapidez y la manipulación sencilla de vectores o matrices sin afectar sus dimensiones. Otra alternativa sería usar esta libería de forma complementaria a OpenCV, al menos en lo que respecta a la parte de detección y descripción de características además del matching al permitir realizar operaciones algebraicas en tiempos de ejecucción mucho más pequeños.

- Realizar mejoras en la escritura del código C++. Aunque no afectará de manera drástica a los tiempos de ejecucción del programa existen ciertas herramientras en lenguaje de programación C++ que sirven para optimizarlo. Entre ellas cabe destacar el uso del modificador de acceso register para almacenar ciertas variables locales como los contadores en registros para acelerar la obtección de su valor o el uso de funciones inline con operaciones simples para evitar la llamada a funciones que realentizan el programa.

7. APÉNDICES

7.1. Ápendice A: Esteroscopia.

La técnica de estereoscopía consiste en la recomposición de la visión binocular del ser humano, es decir, en la creación de imágenes *3D* a partir de dos imágenes *2D* separadas a cierta distancia.

Nuestros ojos al estar situados en posiciones diferentes, captas dos imágenes ligeramente distantes entre sí. A este fenómeno se le conoce como disparidad binocular. Cuando las imágenes llegan al cerebro, este interpreta esta separación denominada técnicamente como disparidad y calcula la distancia real a la que se encuentran los objetos de la escena *3D* generando un efecto de profundidad, es decir, creando la visión tridimensional.

Esta percepción de tercera dimensión se puede lograr mediante el tamaño relativo de los objetos, las sombras o mediante la perspectiva de visión, pero sólo la convergencia relativa de los ejes óptimos la que realmente genera la aparición de profundidad de los objetos. La intersección de esta convergencia de los ejes genera un ángulo denominado paralático cuyo valor depende directamente de la distancia entre el objeto real de la escena y la persona que se encuentra observándola, como se muestra en la figura (38).

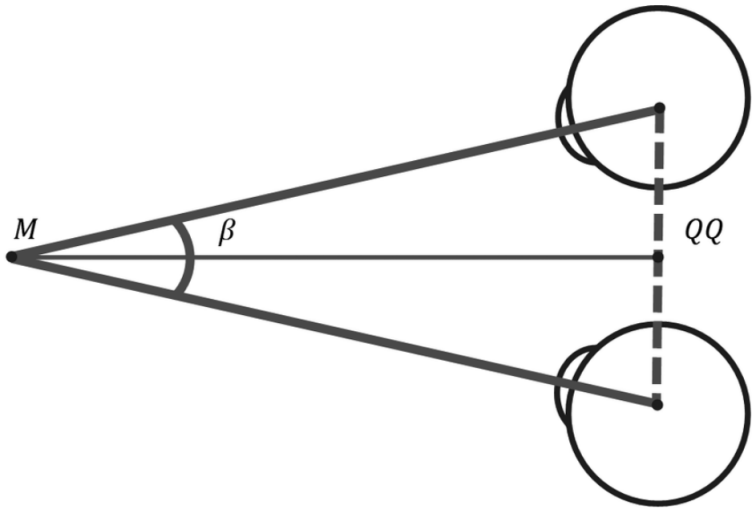


Fig. 38. Generación del ángulo paraláctico por la convergencia de los ejes ópticos. Fuentes: https://www.researchgate.net Andrés Ussa Caycedo

7.2. Ápendice B: Geometría epipolar.

La geometría epipolar es la encargada en explicar la visión binocular matemáticamente. Suponemos que dos cámaras sacan una foto del mismo punto p de la escena *3D*. Este punto estará proyectado en cada imagen como se muestra en la figura (39) Se puede afir-

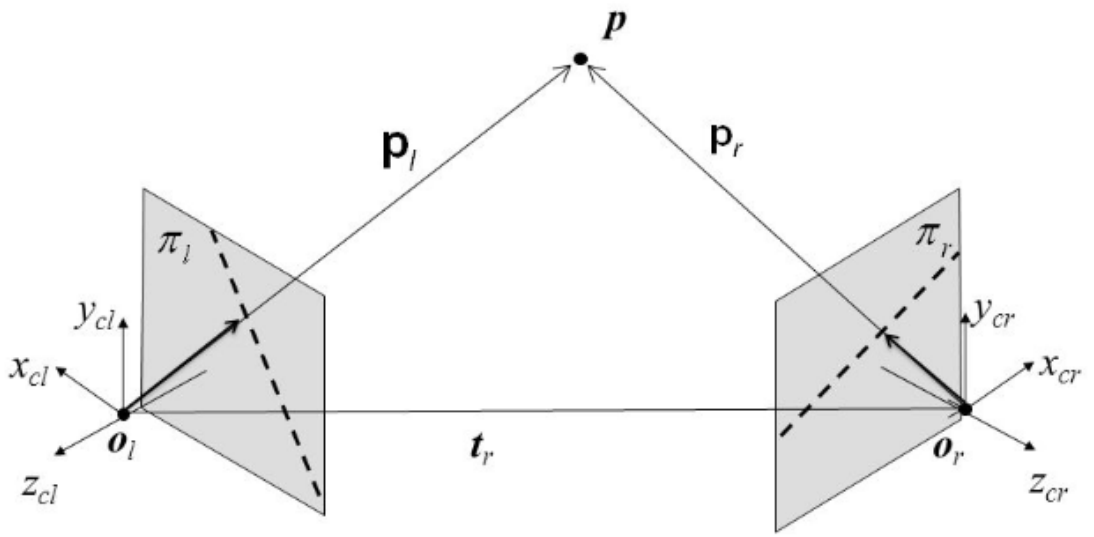


Fig. 39. Geometría epipolar.

mar que el punto p pertenece a la recta que nace en el centro óptico O_l para formar P_l en el plano de la imagen (π_l). En esta recta son muchos los puntos de la escena real que puede formar el punto p_l. Proyectando en la siguiente imagen los posibles obtenemos la denominada línea eipipolar (línea discontinua) en el plano de la imagen de la segunda cámara (π_r). Esta línea es la que contiene todos los posibles puntos que pueden formar la proyección de p_l, incluyendo la proyección del punto de la escena real P, denominado p_r.

Por lo tanto, para cada punto de la imagen existe una línea en la otra imagen en la que está incluído, osea, existe una correspondencia entre un punto de la imagen y una línea en la otra. Esta correspondencia es representada mediante la matriz fundamental F de dimesiones 3x3.

7.3. Ápendice C: Reconstrucción *3D*: Técnicas de triangulación.

Una de las formas de reconstruir una escena *3D* a partir de dos imágenes es siguiendo los siguientes pasos:

1. Calcular la matriz F con los puntos de correspondencia obtenidos.

   Esta matriz puede ser calculada con un mínimo de 7 puntos de correspondecia, sabiendo que está definida por la siguiente ecuación:

   $$x'^T F x = 0 \tag{65}$$

   Existen varios algoritmos que resuelven esta ecuación, entre los que destacan el algoritmo de los 8 puntos normalizados, el basado en la distancia geometría y un algoritmo de estimación robusta basado en RANSAC (*Random Sample Consensus*).

2. Calcular las matrices de las cámaras o de proyección, con la matriz previamente calculada F.

3. Calcular el punto de la escena *3D* que corresponde a cada par de puntos en correspondencia.

7.4. Ápendice D: Descomposición en Valores Singulares (SVD)

Cualquier matriz A de dimensiones mxn, donde el número de filas sea mayor o igual al de columnas, es decir, $m \geq n$, puede descomponerse en:

$$A = U\Sigma V^T$$

Donde U es una matriz de dimensiones mxn con columnas ortogonales, Σ una matriz diagonal nxn y V una matriz ortogonal nxn, es decir, matriz cuya inversa (V^{-1}) coincide con su transpuesta (V^T).

A esta forma de expresar la matriz A se le denomina Descomposición en Valores Singulares (DVS, o *SVD, Singular Value Decomposition*, en inglés).

Dado que la matriz A tiene unas dimensiones de mxn, es decir, es asimétrica, calcularemos los autovalores (o valores singulares) y autovectores para la matriz simétrica A^TA, es decir, encontrar los valores de λ y v que satisfacen la siguiente relación:

$$A^TAv = \lambda v$$

Resolviendo esta ecuación, obtenemos que los autovalores de la matriz A serán las raíces cuadradas de los de la matriz simétrica A^tA ($\sqrt{\lambda_i}$) y los autovectores de la simétrica aquellos que satisfacen:

$$v_i^T A^T A v_i = \lambda_i$$

Es así que la matriz Σ estará compuesta por una matriz diagonal D de rxr (siendo r el rango de la matriz A) y tres matrices ceros $\emptyset$ de la siguiente manera:

$$\Sigma = \begin{bmatrix} D & \emptyset \\ \emptyset & \emptyset \end{bmatrix}$$

Donde la matriz D contiene los autovalores σ_i de A:

$$D = \begin{bmatrix} \sigma_1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & \sigma_r \end{bmatrix}$$

Es fácil de observar que el rango de la matriz A siempre será $r \leq n$. En el caso que $r = n$, la matriz Σ sólo estará formada por la matriz diagonal D ($\Sigma = D$).

La matriz ortogonal V nxn será por tanto aquella compuesta por los autovectores de la matriz A^TA:

$$V = \begin{bmatrix} v_1 & v_2 & \dots & v_n \end{bmatrix}$$

La matriz U, la más díficil de conseguir, se obtendrá mediante el conocimiento de los autovalores no nulos ya calculados de la matriz A (σ_i), y la relación $\sigma_i = \|Av_i\|$.

Con esto, se obtiene una matriz:

$$U = \begin{bmatrix} u_1 & u_2 & \dots & u_r & u_{r+1} & \dots & u_n \end{bmatrix},$$

tal que cada uno de sus elementos es obtenido mediante la siguiente relación de normalización:

$$u_i = \frac{Av_i}{\sigma_i}$$

En el caso de q $r = n$, la matriz será:

$$U = \begin{bmatrix} u_1 & u_2 & \dots & u_r \end{bmatrix}$$

7.5. Apéndice E: Los procesadores.

Se definen a continuación, a modo de resumen, los distintos conceptos para entender el funcionamiento de cada procesador:

- CPU (*Central Processing Unit*). Es el principal circuito electrónico de un ordenador que dispone de varios núcleos que realizan todas las tareas que se precise, que pueden ser tanto operaciones lógicas, algebraícas o de otro tipo.
- Núcleos. Son un conjunto de microprocesadores que contiene el circuito electrónico que forma la CPU. Las CPU más potentes tienen un número mayor de núcleos (hasta 32) permitiendo realizar un mayor número de instrucciones más rápidas.
- GPU (*Graphics Processing Unit*). La tarjeta gráfica de un ordenador o de una forma más técnica, unidad de procesamiento gráfico es un circuito electrónico desarollados para la creación y procesamiento de imágenes. Sin embargo, dado que contiene una gran cantidad de núcleos se presenta como una gran alternativa para realizar tareas de cálculo con gran carga computacional.
- Kernel. LLamamos Kernel al código desarollado para ejecutarse diferentes hilos (*theread*) de ejecucción de forma paralela dentro de la GPU de un ordenador.
- Hilo de ejecucción (*theread*). Cada hilo de ejecucción es el conjunto de instrucciones que se deben ejecutar en el procesador seleccionado de forma secuencial.
- Memoria. La memoria es el circuito electrónico que necesita cualquier tipo de procesador para poder funcionar al ser el encargado de almacenar de forma temporal los datos utilizados en las diferentes operaciones que realiza.

Referencias

[1] Extreme access flyer to take planetary exploration airborne.
URL https://www.nasa.gov/feature/extreme-access-flyer-to-take-planetary-explo

[2] R. d. Barrio Tajadura, et al., Uso de drones en la inspección para la rehabilitación del patrimonio. la iglesia de la merced de burgos.

[3] Z. J. P. Checa, A. Fernández, L. A. Hernández, Combinación de fotogrametría terrestre y aérea de bajo coste: el levantamiento tridimensional de la iglesia de san miguel de ágreda (soria), Virtual Archaeology Review 5 (10) (2014) 51–58.

[4] D. P. Prado, Drones en espacios urbanos: Caso de estudio en parques, jardines y patrimonio edificado de cuenca, Estoa. Revista de la Facultad de Arquitectura y Urbanismo de la Universidad de Cuenca 6 (11) (2017) 159–168.

[5] P. M. Fernandez, D. O. Martínez, E. P. García, F. B. Lorenzo, Vehículos aéreos no tripulados (vant) en cuba, aplicados a la geomática. estado actual, perspectivas y desarrollo.

[6] E. CASTRO MARTÍNEZ, Transporte, tiempo, distancia y entregas: El futuro del dron en méxico.

[7] S. Mandujano, M. Mulero-Pázmany, A. Rísquez-Valdepeña, Drones: Una nueva tecnología para el estudio y monitoreo de fauna y hábitats, Agro Productividad 10 (10).

[8] J. M. Castell Barahona, Sistema de vigilancia global para gestión del tráfico aéreo de drones, B.S. thesis, Universitat Politècnica de Catalunya (2018).

[9] G. R. Zambrano, V. E. Senti, Marco de trabajo para el diseño de una arquitectura its en ecuador que mejore la interoperabilidad y el despliegue de los sistemas de control de tráfico vehicular/[framework for designing an its architecture in ecuador that improves the interoperability and deployment of vehicular traffic control systems], International Journal of Innovation and Applied Studies 14 (4) (2016) 886.

[10] C. A. C. Flórez, O. L. R. Sandoval, D. A. Hurtado, Procesamiento de imágenes para reconocimiento de daños causados por plagas en el cultivo de begonia semperflorens (flor de azúcar), Acta Agronómica 64 (3) (2015) 273–279.

[11] A. D. Antivero Bazalar, D. L. Gomero Milla, J. J. Montalvan Espinoza, C. O. Villar Uribe, Proyecto profesional monitoreo fitopatológico de cultivos de uva: Golden eagle drone services.

[12] V. Ponce, G. Rey, Implementación de un sistema de procesamiento digital de imágenes para la detección de malezas en el cultivo de quinua.

[13] D. N. Brito, C. F. Nunes, F. L. Padua, A. Lacerda, Evaluation of interest point matching methods for projective reconstruction of 3d scenes, IEEE Latin America Transactions 14 (3) (2016) 1393–1400.

[14] J. S. Esteve, El régimen jurídico de la utilización de drones en españa.

[15] T. O. Molina Moscoso, Reglamento que regule el registro, certificación, control y operación del sistema de aeronaves no tripuladas (rpas–drones), B.S. thesis, Quito: UCE (2017).

[16] J. E. Elizondo, L. P. Maestre, Fundamentos de procesamiento de imágenes, Documentación Universidad Autónoma de Baja California, Unidad Tijuana.

[17] W. P. González, R.C, Capítulo 3. Filtrado de imágenes.

[18] L. Á. Ruis Fernández, La transformada de fourier. aplicación al filtrado de imágenes, Ph.D. thesis, ETSI. Geodésica, Cartográfica y Topográfica. Universidad de Valencia.

[19] Z. Zhang, A. R. Hanson, 3d reconstruction based on homography mapping, Proc. ARPA96 (1996) 1007–1012.

[20] M. E. Gil Fernández, Desarrollo de un sistema de aterrizaje autónomo sobre una plataforma de carga para un uav tipo parrot ar. drone 2.

[21] A. F. S. Bohórquez, L. E. Mendoza, C. A. P. Cortés, Sistema de inspección y vigilancia utilizando un robot aéreo guiado mediante visión artificial, ITECKNE: Innovación e Investigación en Ingeniería 10 (2) (2013) 190–198.

[22] A. Romero Ordas, et al., Inserción de drones en el campo de las inspecciones.

[23] R. E. Sepúlveda Cossio, I. A. Agudelo Correa, J. C. Casas Pamplona, Modelo metodológico para realizar mantenimientos predictivo y preventivo por medio de drones en el municipio de guatapé., Ph.D. thesis (2017).

[24] M. A. Charfen Garcia, Recomendaciones para la aplicación de los drones en el mundo de la arquitectura, Master's thesis, Universitat Politècnica de Catalunya (2015).

[25] G. F. A. Henríquez, C. M. M. Torres, Aplicaciones de los drones en la agricultura.

[26] J. G. Jiménez, A. O. Baturone, Estimación de la posición de un robot móvil, Automática (29) (1996) 3–18.

[27] E. Statello, R. Verrastro, B. Robino, J. C. Gomez, S. Tapino, Navegación por visión estereoscópica asistida por gps.

[28] M. K. Martin, D. A. Vause, New low cost avionics with ins/gps for a variety of vehicles, IEEE aerospace and electronic systems magazine 13 (11) (1998) 41–46.

[29] C. F. Rivers, T. H. Powell, Identification friend or foe (iff) system, uS Patent 8,325,081 (Dec. 4 2012).

[30] M. Asencio, L'utilisation civile des drones: problèmes techniques, opérationnels et juridiques, Sécurité globale (4) (2008) 109–118.

[31] I. Bondar, Seismic horizon detection using image processing algorithms 1, Geophysical Prospecting 40 (7) (1992) 785–800.

[32] S. Gonzalez, E. Gonzlez, M. Nitsche, P. De Cristforis, Odometrıa visual para robots móviles utilizando smartphones como unidad de sensado y procesamiento, Jornadas Argentinas de Robótica.

[33] J. C. Rodríguez-Sánchez, V. M. Landassuri-Moreno, J. M. F. Albino, Reconocimiento de patrones numéricos para vuelo controlado de un ar drone utilizando redes neuronales artificiales., Research in Computing Science 107 (2015) 61–71.

[34] J. Engel, T. Schöps, D. Cremers, Lsd-slam: Large-scale direct monocular slam, in: European Conference on Computer Vision, Springer, 2014, pp. 834–849.

[35] N. Díaz Mesa, Sistema de odometría visual: estimación de la localización de un vehículo.

[36] L. A. P. Dias, Odometria visual usando imagens rgb-d, Master's thesis (2013).

[37] F. Díaz Lorenzo, Odometría con cámaras estereoscópicas en robótica.

[38] A. G. Aparicio, O. Reinoso, L. Paya, M. Ballesta, Assessing the influence in the parameters of a rao-blackwellised particle filter to solve the slam problem, IEEE Latin America Transactions 6 (1).

[39] L. Fernández, L. Payá, M. Ballesta, F. Amorós, O. Reinoso, Odometría visual y construccion de un mapa topologico a partir de la apariencia global de imagenes omnidireccionales.

[40] E. Fernández-Moral, J. González-Jiménez, V. Arévalo, Partición de mapas para slam monocular en gran escala.

[41] A. J. Davison, I. D. Reid, N. D. Molton, O. Stasse, Monoslam: Real-time single camera slam, IEEE Transactions on Pattern Analysis & Machine Intelligence (6) (2007) 1052–1067.

[42] J. D. Carlson, M. Mittek, S. A. Parkison, P. Sathler, D. Bayne, E. T. Psota, L. C. Pérez, S. J. Bonasera, Smart watch rssi localization and refinement for behavioral classification using laser-slam for mapping and fingerprinting, in: Engineering in Medicine and Biology Society (EMBC), 2014 36th Annual International Conference of the IEEE, IEEE, 2014, pp. 2173–2176.

[43] D. Caruso, J. Engel, D. Cremers, Large-scale direct slam for omnidirectional cameras., in: IROS, Vol. 1, 2015, p. 2.

[44] R. Mur-Artal, J. M. M. Montiel, J. D. Tardos, Orb-slam: a versatile and accurate monocular slam system, IEEE Transactions on Robotics 31 (5) (2015) 1147–1163.

[45] D. Nistér, O. Naroditsky, J. Bergen, Visual odometry, in: Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on, Vol. 1, Ieee, 2004, pp. I–I.

[46] A. Vedaldi, An open implementation of the sift detector and descriptor, UCLA CSD.

[47] H. Bay, A. Ess, T. Tuytelaars, L. Van Gool, Speeded-up robust features (surf), Computer vision and image understanding 110 (3) (2008) 346–359.

[48] L. A. A. Caicedo, Aplicación del algoritmo fast marching en ambientes tridimensionales: Caso de estudio en imágenes médicas, Trabajo de Grado.

[49] S. Leutenegger, M. Chli, R. Y. Siegwart, Brisk: Binary robust invariant scalable keypoints, in: Computer Vision (ICCV), 2011 IEEE International Conference on, IEEE, 2011, pp. 2548–2555.

[50] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, Orb: An efficient alternative to sift or surf, in: Computer Vision (ICCV), 2011 IEEE international conference on, IEEE, 2011, pp. 2564–2571.

[51] P. Vandergheynst, R. Ortiz, A. Alahi, Freak: Fast retina keypoint, in: 2012 IEEE Conference on Computer Vision and Pattern Recognition, Ieee, 2012, pp. 510–517.

[52] M. A. Fischler, R. C. Bolles, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM 24 (6) (1981) 381–395.

[53] R. Hartley, A. Zisserman, Multiple view geometry in computer vision, Cambridge university press, 2003.

*** ANEXO: EXTENDED ABSTRARCT.**

Since the birth of robotics, the search for autonomy of any type of vehicle, both manned and unmanned, has been a very interenting topic. Therefore, achieving the maximum possible autonomous navigation of a vehicle in a completely unknown environment.

It is specially interesting working with UAVs (Unmanned Aerial Vehicles) or in Spanish, VANTs (Vehículos Aéreos No Tripulados), better known as "drones", since this has allowed the appearance of many day-to-day applications as aerial photography or other more specific applications such as monitoring of traffic or the exploration of certain geographical areas. Navigation systems used by this kind of vehicles can be operated using a pilot from a ground control tower, or even without pilot. The latest trend due to the need to obtain smaller drones, results in the automation of the flight using certain sensors incorporated into the vehicle.

Despite being a reality in some countries, in Spain, this type of drones known as autonomous, are not allowed according to the last Real Decreto 1023/17, which focuses in the regularization of the so-called RPA (Remotely Piloted Aircrafts).

Throughout the years, multiple possibilities have been studied to offer an autonomous flight using the different types of sensors incorporated into the vehicle. The most widespread techniques are those based on the use of GPS (Global Positioning System) that collect direct information about the global position of the vehicle sent through a constellation of satellites. However, this presents serious problems when working in interiors or areas of low visibility of satellites. Therefore, other kind of techniques have appeared to supplement it (called assisted systems) or simply to replace it as navigation systems based on computer vision.

These algorithms are based on information extracted from images and therefore, they have low cost and low power consumption, so they are ideal for drones with reduced size. However, it depends on many other factors such as the resolution of the images, the capture time, the angle of vision, the illumination or the data taken as a reference. Therefore, they are still an open area of research since there are also multiple work perspectives. Depending on the visual information used (horizon detection [31], point tracking [32] or edge detection in the image [33]) and the vision system used (monocular ([34], [35]) , binocular [36] or stereo [37], trinocular [38] or omnidirectional [39] camera), different navigation systems appear that enable the possibility of a wide range of applications.

To obtain good results regardless the type of environment in which drones move, the two most popular techniques among the algorithms based in computer vision: SLAM (simultaneous localization and mapping) and visual odometry. SLAM refers to the mapping algorithms and simultaneous mapping capable of performing the construction of a map and the estimation of the vehicle's position at the same time. The main problem with this type of algorithm is that they are computationally expensive and complex, besides the appearance of outliers when obtaining the data to be processed, affecting negatively in the estimate. In contrast with SLAM, visual odometry are those algorithms that try to estimate

both the 3D position of the vehicle and its orientation, and they do not need information from prior positions to compute the actual estimate, so they are not so sensitive to the appearance of outliers.

As already mentioned before, depending on the number of cameras used we distinguish two types of odometry: stereoscopic visual odometry, which is the most used technique and is based on the stereoscopic vision of humans, that is, the use of two images captured by two different cameras separated by a certain distance or by a stereoscopic camera, which consists of two objectives and the monocular visual odometry that uses only one camera to capture the images, but this presents some problems in the estimate of the position when since it needs the previous knowledge of the camera's angle with respect to the ground. Therefore, in this type of techniques all distances are relative to the first movement of the camera, meaning that the first one is taken as reference and the successive ones are calculated from it. Within this group we find visual odometry based on the use of the luminosity information of the pixels or of the luminosity in a region, visual odometry based on characteristics that use the correspondence of points between consecutive images and hybrid visual odometry, which mixes the two previous types.

This paper presented the improvements made to a navigation system based on monocular visual odometry based on characteristics to achieve better response times of the developed algorithm. An algorithm based on visual odometry could be divided into the following subtasks:

1. Obtaining the images. The images captured by the camera or cameras incorporated in the vehicle from which we want to estimate its position are obtained.

2. Characteristics detection. Certain characteristics of the image are detected by means of digital image processing techniques.

3. Matching of the characteristics. The same characteristics are searched in different images, that is, a point in an image is paired with that same point in an image captured at an instant of successive time.

4. Rejection of the outliers. This stage is responsible for eliminating the noise or ruling out incorrect matches, which we call outliers.

5. Estimate of movement. This stage is the most important one of the whole process, but it depends on the efficiency of the previous steps to obtain good results. The movement is calculated using the matching points and will be estimated for each pair of consecutive images.

It is evident that the detection of characteristics, which is nothing more than locating parts of the image with a certain interest for being able to be distinguished from the rest through the image processing, supposes a high computational cost, which leads to high compilation times. In this paper, two alternatives were presented with the aim of reducing

the compilation time of the algorithm while maintaining its robustness: the use of the CUDA (Compute Unified Device Architecture) architecture and the replacement of the SIFT (Scale-invariant Feature Transform) feature detection algorithm by the SURF (Speeded-Up Robust Features) algorithm provided by the OpenCV (Open Source Computer Vision) artificial vision library that operates directly on CUDA. This parallel calculation architecture developed by Nvidia in 2009 to cover the need for the appearance of a tool that allows the possibility of making video cards (called "devices") programmable through the use of programming languages such as C ++, together with a CPU (known as host), allows us to drastically speed up the execution times of certain operations, especially those related to image processing.

In this work, two solutions are proposed to obtain shorter execution times. In the first proposal, the detection and decription algorithm SIFT (Scale Invariant Feature Transform) is maintained. This algorithm is based on the difference between the smoothing performed on a sequence of images by Gaussian filters.

The procedure is the following:

1. We apply the Gaussian to the original image by sequentially increasing the scale.
2. The Gaussian difference (DoG) is applied.
3. The maximum and minimum are calculated.
4. The last smoothed image, the one with the highest σ applied, is rescaled in half to get the next level.

With the rescaled image, the previous steps are repeated until it is not possible to continue decreasing the image. In addition, the selection of keypoints is refined using different techniques such as RANSAC.

With this algorithm each point of interest has associated the level of pyramid and the scale value in which it was detected. With this, the corresponding image is selected and the region of interest detected in nxn cells is divided (in our case n = 4). For each cell the gradient of each of its pixels is calculated, which can be easily calculated from the differences in its intensities.

Once the addresses of each pixel have been obtained in that region, the predominant direction must be chosen for each cell by calculating a histogram of orientations. The global maximum of the histogram and alternative dominant orientations to those local maximums of at least 80 % of the global maximum will be chosen as the dominant orientation of the region.

OpenCV does not provide a tool to use the algorithm directly on CUDA so it will have to be programmed manually. The steps to follow are these:

1. RESCUE THE IMAGES

Images are resized to be able to adapt them to the necessary form of CUDA images.

2. INITIALIZE CUDA

   It is the first thing that has to be done if we want to work in CUDA on the GPU.

3. TRANSFORM IMAGES

   The CUDA images (of type CudaImage) are declared and those images already resized corresponding to each frame are transformed thanks to the .Allocate"function of CUDA that allows us to allocate memory for the image in the GPU.

4. DOWNLOAD THE IMAGES FROM THE HOST TO THE DEVICE

   Once the memory is assigned to the GPU, we simply have to " download " it to transfer it from the host (CPU) to the device (graphic card). For this, CUDA offers the " Download " method.

5. SIFT ALGORITHM

   The SIFT algorithm is applied manually.

6. MATCHING CALCULATION

   The matching is calculated between the points of interest of the frames on the GPU.

7. HOMOGRAPHY CALCULATION

   The next step, once the correspondence points have been obtained, has been to calculate the correspondence homography frame to frame. It has been done using the " FindHomography " method and then it has been adjusted by removing the outliers in the " ImproveHomography " method.

8. KEYPOINTS REPRESENTATION

   Next the keypoints have been drawn on the original image on the CPU. For this it was necessary to re-size the image again.

9. BREAK FREE THE DEVICE MEMORY

   As we will no longer work with CUDA in the algorithm, we must break free the memory of the device.

The second proposal is to use the SURF algorithm (Speeded-Up Robust Features). This algorithm is based on SIFT and makes use of the Hessian matrix because from the second derivative we can obtain the smoothing of the image and the maximum and minimum at the same time.

Calculate the Laplacian for each pixel has a large computational load so in order to reduce execution times, this algorithm uses approximate derivatives. Integrated images are used for this algorithm, which are just transformed images so the value of each of its pixels is the sum of all the pixels in the original image located in its upper left corner.

This algorithm allows you to find points of interest in just four steps:

1. The original image is transformed to its corresponding integrated image.
2. The convolution is calculated between the integrated image and a filter that simulates the result of a Gaussian filter to smooth the image.
3. The used filter is rescaled and the convolution is carried out with the filters with increasing size. Usually 3 levels are calculated.
4. We calculate the determinant of the Hessian matrix already explained to obtain points of interest. Keypoints will be considered those maximum points that are in the 3 levels of scale.
5. Once the points of interest are obtained, the Haar filters are used to obtain the dominant orientation.

After obtaining the direction of the dominant orientation, a square region centered on the point of interest of size 30σ is now centered and divided into 16 cells on which the Haar filters will be applied again to obtain the values on the x-axis (h_x) and y-axis (h_y) and in addition to their absolute values. Thus we obtain 64 values that correspond to the descriptor: 16 values h_x + 16 $|h_x|$ + 16 values h_y and 16 $|h_y|$. The sums h_x and h_x are made separately and the values are normalized so that this is the unit.

Unlike with the SIFT algorithm, OpenCV does provide a class to work with the SURF algorithm over CUDA. This is the cv::cuda::SURF_CUDA to be able to store both data in the GPU and in the CPU. The steps to follow are these:

1. IMAGES CONVERSION

   Since we are going to work on CUDA, we must convert the OpenCV images (stored in the host) to images that work on CUDA (stored on the device). The OpenCV images are of type cv::Mat and the other cv::cuda::GpuMat. The conversion is done using the upload"method of the OpenCV SURF_CUDA class.

2. SURF ALGORITHM

   Once the conversion of the images corresponding to each consecutive frame has been made, we apply the SURF algorithm. This will return the keypoints and their descriptors at the same time.

3. MATCHING CALCULATION

   The correspondence between the points of interest obtained in each image (matching) is calculated. With these matches we will subsequently calculate the correspondence homography (frame to frame homography).

4. VECTOR CONVERSION

   Just as at the beginning we had to convert the CPU images into GPU images, now we have to transform the data that works on the GPU (GpuMat) to its CPU equivalents using the "downloadDescriptors.ªnd "downloadKeypoints"methods.

5. MEMORY RELEASE

   As we are not going to use more data about the GPU, we must rescue the memory.

Once the results have been analyzed, it can be said that in terms of execution times, the most efficient algorithm is the SIFT algorithm programmed with CUDA. However, the SURF_CUDA algorithm also doesn't generate long times, so it could be a totally feasible solution.

Keep in mind that these two proposals are designed to reduce the execution time of the part of detection and description of features and matching only, without taking into account how it results affect the position estimation and final orientation of the VANT.

Theoretically, SURF omits certain details in the image that are detected by the SIFT algorithm, and therefore, processes fewer characteristic points, generating shorter execution times. Nevertheless, our results show the opposite. This may be due to several factors, among others, the resolution of the images. As it is logical to believe, the lower the resolution of the image, the smaller the number of pixels that compose it, which leads to a lower number of keypoints detected and therefore shorter execution time, but in despite of this last fact, the algorithms can work better within a specific resolution range that don't have to match. In addition, it can also affect the number of images used. Although the graph 37 was presented as a general idea, within the CUDA line the SURF algorithm doesn't work exactly the same as the SIFT algorithm.

In addition, the SURF algorithm generally works better against rotations and blurs but is less robust against changes in lighting. Taking this into account, it will be the responsibility of the person in charge, for the final application to which this project could be applied, choosing one algorithm or another, considering what is most important for each case.

To conclude, as future work we could study several alternatives that could reduce the execution time of the program such as the following mentioned:

- Rewrite all the code introducing the CUDA architecture and not just for the SIFT algorithm in addition to the matching characteristics and the frame-to-frame homography.
- Introduce the OpenCL (Open Computing Language) architecture as an alternative to CUDA, which allows the development of programs that can be executed in any processing unit, since it has a compiler adaptive that allows executing the code developed in the programming language C or C ++ in both the CPU and the GPU. The

biggest advantage versus to the CUDA architecture is that it is not designed for a graphics card in concrete, but it is compatible with a large number of manufacturers.

- Replace the SIFT feature detection algorithm with the algorithm FREAK (Fast Retina Keypoint) that is based on the knowledge of the system human visual, in particular, in the retina functioning. This algorithm performs the description function and describes the keypoints through a cascade of binary chains. These chains are obtained by comparing the intensities of the image on a retinal sampling pattern. This algorithm, also implemented in OpenCV, needing less load of memory, is faster but nevertheless presents worse results than the SIFT or SURF algorithms.

- Replace the OpenCV library, used in the processing of images necessary to carry out the estimation of position and orientation of the vehicle, by the ArrayFire library. It is adapted for the development of programs in C ++, C, Fortran, Python, Java or R programming languages. It is also compatible with CUDA and OpenCL and therefore allows to work both about the CPU as the GPU. This library contains the GFOR technology that allows you to loop iterations over the GPU, which is very interesting when carrying out any type of image processing, given that these are stored on the graphics cards. It also provides functions that allow executing arithmetic operations, linear algebra, statistics more quickly and the simple manipulation of vectors or matrices without affecting their dimensions. Another alternative would be to use this library in a complementary way to OpenCV, at least with respect to the feature detection and description part in addition to matching by allowing algebraic operations to be implemented in smaller execution times.

- Make improvements in the writing of the C ++ code. Although it will not drastically affect the execution times of the program, there are certain tools in the C ++ programming language that could serve to optimize them. These include the use of the register access modifier to store certain local variables such as counters in registers to accelerate the acquisition of their value or the use of inline functions for simple operations to avoid calling functions that slow down the program.