Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Aplicación de Técnicas de Detección y Cruce de Ventanas con Vehículos Aéreos no Tripulados

Autor: Víctor Quesada Conejero

Tutor: Jesús Capitán Fernández

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Aplicación de Técnicas de Detección y Cruce de Ventanas con Vehículos Aéreos no Tripulados

Autor:
Víctor Quesada Conejero

Tutor:
Jesús Capitán Fernández
Profesor Contratado Doctor

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2020

Trabajo Fin de Grado: Aplicación de Técnicas de Detección y Cruce de Ventanas con Vehículos Aéreos no Tripulados

Autor: Víctor Quesada Conejero
Tutor: Jesús Capitán Fernández

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

A mi familia, por permitirme cumplir este sueño;
A mi tío Lolo, por su infinito apoyo;
Jesús Capitán, por su paciencia y dedicación;
Raúl, por los cafés tomados y los que quedan por tomar;
Clara, por acompañarme en este bonito viaje.

Resumen

La robótica es un campo en auge que se extiende a casi todos los ámbitos, desde el entretenimiento al laboral. A favor de la inclusión de la robótica se plantea el uso de vehículos aéreos no tripulados en operaciones de rescate; reduciendo el riesgo humano del personal de emergencia en estas situaciones. En concreto, la robótica aérea permite la inspección de superficies en altura sin la necesidad de exponer a riesgos de caídas al personal de rescate.

Este trabajo de fin de grado presenta un análisis de distintas técnicas para la detección de ventanas aplicando técnicas de extracción de *features* y el cruce de las mismas realizando un control en velocidad con un UAV *(Unmanned Aerial Vehicle)*. Para ello, se hace uso de un robot aereo equipado con una cámara estereoscópica y un LIDAR *Light Detection and Ranging* para la recolección de datos de su entorno. A través de la simulación se comprueba la efectividad del algoritmo y los puntos a desarrollar para conseguir un sistema más robusto, como la inclusión de técnicas SLAM *Simultaneous Localization And Mapping* que permiten generar mapas del entorno y mejorar el posicionamiento.

Abstract

Robotics is a field on the rise with many applications in almost every domain, from entertainment to work-related issues. Amongst the examples supporting Robotics-integration, the use of unmanned aerial vehicles (UAVs) in rescue operations is considered, as it can reduce personal risk for emergency services staff in these situations. Specifically, aerial robotics allows inspecting at-height areas while avoiding exposing rescue service personnel to the risk of a fall.
The aim of this project is to present an analysis of different techniques for detecting and crossing windows applying feature detection techniques and UAV velocity control for each of these ends, respectively. Accordingly, an UAV equipped with a stereo camera and a LIDAR instrument for data collection in the surrounding environment has been employed. This simulation enables for the effectiveness of the algorithm to be assessed and to determine the actions to be taken in order to develop a more robust method, such as including SLAM techniques which allows map generation of the surrounding environment as well as improving positioning.

Índice Abreviado

Índice

Notación

$\sum$	Sumatorio
$\approx$	Aproximadamente igual
$\frac{\partial x}{\partial y}$	Derivada parcial de x respecto a y
$det(A)$	Determinante de la matriz cuadrada A
$trace(M)$	Traza de la matriz cuadrada A
mín(a,b)	Valor mínimo entre a y b
máx(a,b)	Valor máximo entre a y b
$\leq$	Menor o igual
$\geq$	Mayor o igual
:	Tal que
$\in$	En
$\bar{c}$	Valor booleano negado

Introducción

La inclusión de los UAVs (*Unmanned Aerial Vehicle* o vehículo aéreo no tripulado) en nuestro día a día es una realidad cada vez más cercana. Los robots aéreos, están ya instaurados como una herramienta indispensable en muchos sectores, tanto profesionales como lúdicos, desde sistemas de grabación en el cine hasta herramientas de mantenimiento para superficies en altura. Mientras que en otros, como transporte de mercancías o situaciones de riesgo, se convertirán en herramientas imprescindibles en los años venideros.
Este trabajo de fin de grado presenta un algoritmo para la detección y cruce de ventanas para UAVs, poniendo especial énfasis en la detección de ventanas a través de imágenes captadas por una cámara estereoscópica, llamadas así debido a la visión estereoscópica humana, teniendo la capacidad de capturar pares de imágenes a partir de las que se pueden obtener medidas de profundidad.

Figura 1.1 Ilustración de un UAV comercial.

Motivación del trabajo

La principal motivación del presente trabajo de fin de grado está basada en la inclusión de los UAVs en situaciones de emergencia, en concreto, como una herramienta de apoyo a bomberos en incendios en edificios altos. Minimizando el riesgo para el cuerpo de bomberos y mejorando la velocidad de actuación bajo estas circunstancias.
Los trabajos en altura han supuesto un problema para la sociedad, teniendo que poner en riesgo la vida de trabajadores para realizar tareas que en un futuro podrán ser realizadas por robots autónomos.
Esta necesidad se vio reconocida en la competición MBZIRC (*Mohamed Bin Zayed International Robotics Challenge*) celebrada su última edición en marzo de 2020 en Abu Dhabi, donde grupos de investigación ligados a la robótica compiten en una serie de retos. Siendo uno de ellos, en esta última edición, la colaboración de UAVs y un UGV *(unmanned ground vehicle)* para la extinción de un incendio en un edificio de forma autónoma, situándose los focos de ignición tanto en el interior del edificio como en su fachada e inmediaciones.

Los UAVs deberán estar equipados con lo necesario para la extinción de incendios, ya sea mediante extinción por agua a presión, uso de agentes químicos u otros métodos explorados.

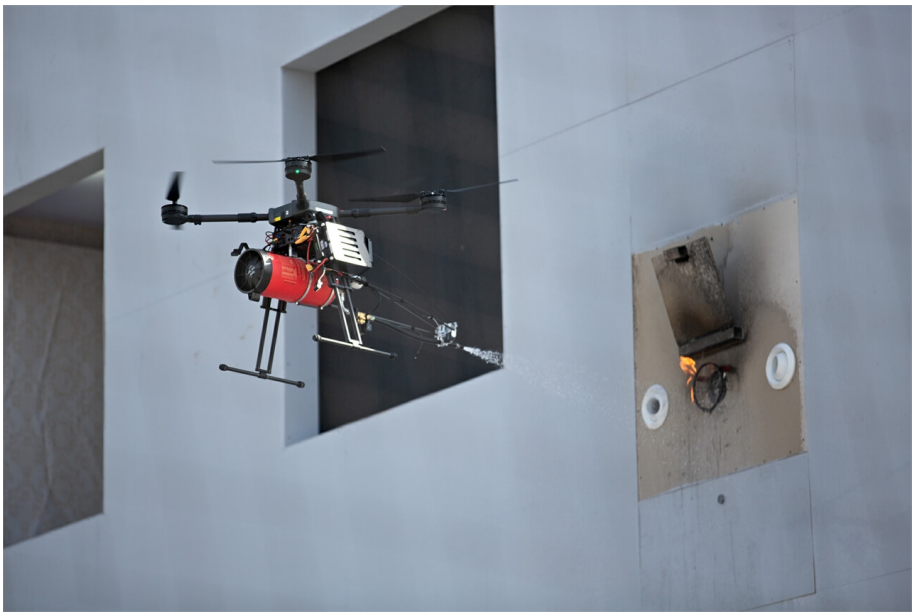


Figura 1.2 UAV durante el challenge 3 de MBZIRC 2020.

Estudios relacionados

En la actualidad se han publicado algunos artículos relacionados con la actuación de los UAVs frente a situaciones de riesgo, en concreto podemos encontrar artículos relacionados con la acción frente a incendios en espacio aéreo urbano [22] donde se realiza un uso de los UAVs sin adentrarse en los edificios transportando agua desde una zona cercana y humedeciendo la zona de una forma similar a como operan los hidroaviones en la actualidad, el uso de esta herramienta para incendios forestales [18][30] o rescates marítimos [11].
Otro ejemplo de la investigación en la extinción de incendios basados en la actuación de agentes químicos aplicados por UAVs, en el artículo explican los componentes utilizados y los resultados obtenidos tras los ensayos experimentales asegurando una reducción en el tiempo de extinción de incendios en un 95 % [5].

Figura 1.3 Imagen visual e infrarroja de un incendio tomada por un UAV [18].

Objetivo

El objetivo del algoritmo propuesto a lo largo del trabajo es la identificación, mediante el uso de sensores 3D, de ventanas en edificios y pudiendo cruzar las mismas de forma segura y eficaz. Es por ello que se ha desglosado los siguientes hitos en el proyecto:

1. Desarrollo método de detección de ventanas.
2. Desarrollo método de cruce de ventanas.
3. Simulación detección de ventanas.
4. Simulación conjunta detección de ventanas y cruce de ventanas.

Estructura

La estructura de este trabajo de fin de grado se basa en seis capítulos. El capítulo 2 versa sobre el estado del arte, describiendo las distintas técnicas relacionadas con el proyecto. El capítulo 3 define las herramientas empleadas para llevar a cabo del proyecto. En el capítulo 4 se encuentra el método propuesto para el cruce de ventanas y la identificación. El capítulo 5 muestra las simulaciones realizadas y los experimentos llevados acabo y por último, el capítulo 6 cierra el texto mostrando las conclusiones y trabajos futuros.

Estado del arte

Introducción

En este capítulo se expondrá el estado del arte de visión artificial y se determinará el marco teórico y las demostraciones matemáticas de distintos algoritmos de los que se hará uso en los capítulos posteriores de la memoria.

Obtención de esquinas

Como primera sección comenzaremos comentando los distintos algoritmos que se encuentran actualmente para la detección de esquinas en imágenes. Para este proyecto ha sido necesario implementar uno de estos algoritmos para detectar las cuatro esquinas de las ventanas y poder identificar la posición en un espacio tridimensional.

Harris corner detector

Harris Coner Detector desarrollado por Chris Harris y Mike Stephens en 1988 [13] como mejora del *Moravec corner detection*, es un algoritmo de obtención de *features* ampliamente utilizado.
Para entender su funcionamiento supondremos una imagen en escala de grises I y vamos a recorrer la imagen con una ventana calculando la variación de intensidad

$$E(u,v) = \sum_{x,y} \underbrace{w(x,y)}_{\text{window function}} \; [\underbrace{I(x+u,y+v)}_{\text{shifted intensity}} - \underbrace{I(x,y)}_{\text{intensity}}]^2 \tag{2.1}$$

Siendo u, v, x y z coordenadas en píxeles. Nosotros estamos buscando ventanas en las que se encuentren esquinas por lo que queremos maximizar la variación de intensidad en la ecuación anterior. Usando el desarrollo de Taylor se puede expresar como:

$$E(u,v) \approx \sum_{x,y} [I(x,y) + uI_x + vI_y - I(x,y)]^2 \tag{2.2}$$

Siendo:

$$I_x = \frac{\partial I}{\partial x}(x,y) \quad I_y = \frac{\partial I}{\partial y}(x,y) \tag{2.3}$$

Cuya expresión matricial quedaría de la siguiente forma:

$$E(u,v) \approx \begin{bmatrix} u & v \end{bmatrix} \left(\sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \right) \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.4}$$

A partir de la cual podemos definir el tensor M como:

$$M = \sum_{x,y} w(x,y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix} \tag{2.5}$$

A través del siguiente parámetro se puntúan cada una de las ventanas de la imagen:

$$R = det(M) - k(trace(M))^2 = \lambda_1\lambda_2 + k(\lambda_1 + \lambda_2) \tag{2.6}$$

Siendo k un valor que se ha de determinar empíricamente y λ los autovalores de la matriz .
Si el valor de R supera el umbral indicado se puede afirmar que contendrá una esquina en el interior de la ventana. Para indicar la posición de la esquina es habitual calcular el máximo local de la ecuación (1.1) dentro de la región delimitada.
Respecto a este algoritmo se han planteado distintas alternativas para mejorarlo, como establecer un valor límite adaptativo [12].

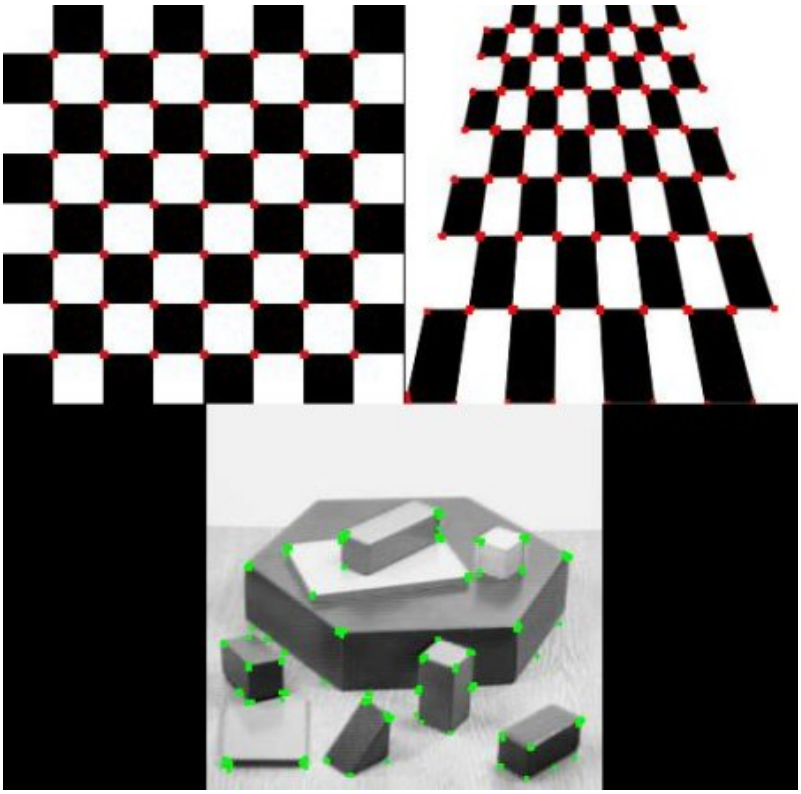


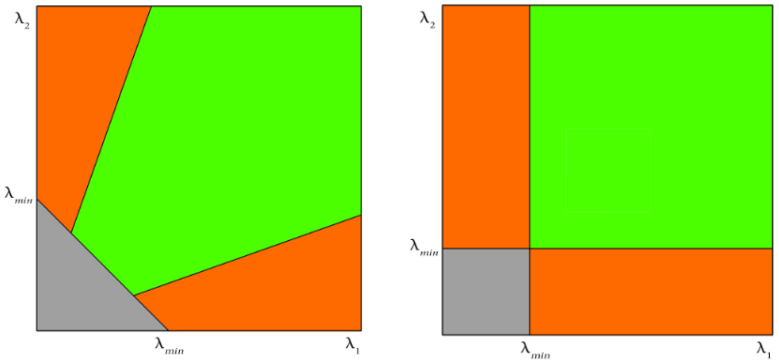
Figura 2.1 Ilustración resultado del algoritmo Harris Corner Detector.

Shi-Tomasi corner detector

En el 1994, J. Shin y C.Tomasi publicaron un *paper* llamado **Good Features to Track**, donde propusieron una modificación en el algoritmo *Harris Corner*, en concreto con la forma de puntuar los pixel, lo cual se demuestra experimentalmente una mejora respecto al anterior [16].
Substituyendo la ecuación 2.6 por la siguiente:

$$R = \text{mín}(\lambda_1, \lambda_2) \tag{2.7}$$

Si el valor de R supera un umbral, se considera que hay una esquina. Representado en un espacio $\lambda_1 - \lambda_2$ las dos ecuaciones anteriores quedaría de la siguiente forma:



(a) Harris Corner. **(b)** Shi-Tomasi Corner .

Figura 2.2 Representación en un espacio $\lambda_1 - \lambda_2$ de los algoritmos de Harris y Shi-Tomasi .

Como podemos observar en la imagen las zonas verdes son consideradas esquinas, mientras que las zonas naranjas son bordes y por último la zona gris se considera que no hay ningún punto de interés.

Features from Accelerated Segment Test

Se han expuesto varios algoritmos para la detección de esquinas, que pueden llegar a ser demasiados lentos para ser usados en programas que necesiten un procesado en tiempo real.
Como solución se presenta el algoritmo *Features from Accelerated Segment Test* (FAST), publicado en 2006 por Edward Rosten y Tom Drummond en *"Machine learning for high-speed corner detection"*[27] y su revisión realizada en el 2010 [26].
Primero comenzaremos explicando como funciona el detector de *features* y luego nos centraremos en como, mediante técnicas de *machine learning*, podemos convertirlo en un detector de esquinas:

Detección de *Features* mediante FAST

Para realizar la detección de *features* se seguirán los siguientes pasos.

1. Seleccionamos un píxel de la imagen P que será identificado como punto de interés o no. Denominamos la intensidad como I_p.

2. Seleccionamos un valor límite t, este valor sera determinado empíricamente o mediante técnicas.

3. Consideramos un círculo de 16 píxeles alrededor de P [8].

4. N píxeles contiguos deben tener mayor intensidad frente al valor establecido como $t + I_p$ o menor que $t - I_p$ para que P sea considerado un punto de interés. Los autores indican como valor óptimo $N = 12$ en la primera versión del algoritmo.

5. Para conseguir hacer el algoritmo más eficiente se comprueban primero los píxeles 1, 9, 5 y 13 numerados como se muestra en la Figura 2.3 . Si P es un punto de interés al menos tres de los cuatro deben de ser más brillantes o más oscuros que los valores indicados.

6. Si se ha cumplido el criterio anterior se comprueba finalmente el resto de píxeles o se descarta el punto.

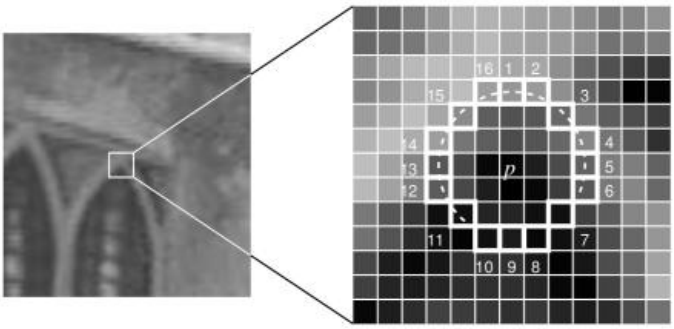


Figura 2.3 Imagen que muestra los 16 píxeles seleccionados alrededor del píxel de interés en el algoritmo FAST.

Este algoritmo presenta distintos problemas o limitaciones. Primero, si el valor de $N < 12$ el algoritmo no funciona correctamente ya que detecta demasiados puntos de interés, el orden de los 16 píxeles determina la velocidad del algoritmo [26] y varios puntos de interés pueden determinarse juntos. Las dos primeras limitaciones pueden ser suplidas añadiendo *machine learning* como veremos a continuación, mientras que la última mediante la técnica *Non-maximal Suppression* (**NMS**).

Machine Learning

Para poder mejorar el rendimiento del algoritmo y enfocarlo a un detector de esquinas usaremos técnicas de *machine learning*, para ello seguiremos los siguientes pasos:

1. Primero necesitamos tener un set de imágenes para el entrenamiento. Debemos seleccionar dicho set para que sea lo más parecido posible a las imágenes que vamos a aplicar el algoritmo.
2. Ejecutamos el algoritmo FAST visto anteriormente a todo el set y almacenamos los 16 píxeles que conforman la circunferencia siempre que sea valorado el píxel interior como punto de interés.
3. Cada uno de los píxeles x en el vector de 16 se le puede asignar uno de los siguientes estados:

$$S_{p\to x} = \begin{cases} d, & I_{p\to x} \leq I_p - t \quad \text{(darker)} \\ s, & I_p - t < I_{p\to x} < I_p + t \quad \text{(similar)} \\ b, & I_p + t \leq I_{p\to x} \quad \text{(brighter)} \end{cases} \tag{2.8}$$

   Siendo $S_{p\to x}$ el estado, $I_{p\to x}$ el valor de intensidad del pixel y t el valor límite.

4. Dependiendo de los estados, el vector sera dividido en tres subsets, P_d, P_s, P_b siendo P el set completo:

$$\begin{aligned} P_b &= \{p \in P : S_{p\to x} = b\} \\ P_s &= \{p \in P : S_{p\to x} = s\} \\ P_d &= \{p \in P : S_{p\to x} = d\} \end{aligned} \tag{2.9}$$

5. Definimos una variable boleana K_p, que sera verdadera si el punto es una esquina o falsa si no lo es.
6. Mediante el algoritmo ID3, un clasificador por árbol de decisiones, lo lanzaremos a los tres subsets indicados anteriormente en busca de las preguntas necesarias para clasificar un punto como esquina. El objetivo de dicho algoritmo es buscar el mínimo de comprobaciones posibles para indicar si consideramos el punto como esquina.
7. ID3 funciona bajo el principio de mínima entropía:

$$\begin{aligned} H(P) &= (c+\bar{c})\log_2(c+\bar{c}) - c\log_2 c - \bar{c}\log_2 \bar{c} \\ c &= |\{p|K_p \textit{ es verdadero}\}| \\ \bar{c} &= |\{p|K_p \textit{ es falso}\}| \end{aligned} \tag{2.10}$$

   Por lo que finalmente selecciona los píxeles que tienen más información para determinar si el punto es una esquina.

8. Repetimos este proceso hasta conseguir igualar la entropía a cero.

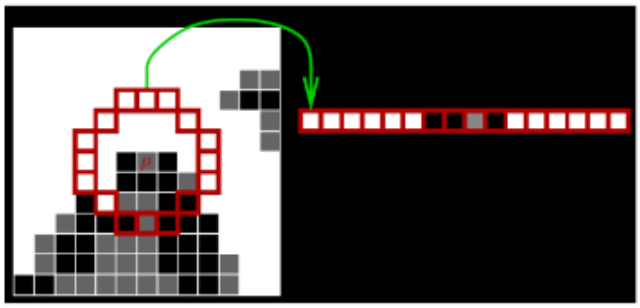


Figura 2.4 16 píxeles alrededor de un p guardados como vector.

Non-maximal suppression

La detección de puntos de interés seguidos supone otro problema al planteamiento de la primera versión del algoritmo. Esto puede solucionarse aplicando el algoritmo *Non maximal suppression* después de haber detectado los puntos de interés. A continuación se expone una breve descripción del algoritmo:

1. Calculamos un valor para cada uno de los puntos de interés mediante una función V. Esta función está definida como "la suma del valor absoluto de la diferencia entre los píxeles en la circunferencia y el píxel central".

2. Considerando dos píxeles que han sido determinados como puntos de interés, descartaremos aquel que tenga menor valor V.
Quedando esto expresado de forma matemática de la siguiente forma:

$$V = max \begin{cases} \sum(pixel\ value - p), & (value - p) > t \\ \sum(p\text{-}pixel\ value), & (p - value) > t \end{cases} \tag{2.11}$$

Donde p es el píxel central, t es el valor límite elegido en el detector, y *pixel values* corresponde a los valores correspondientes de los N píxeles contiguos del circulo [14].

Algoritmo 1: Non-maximal Suppression

```
Funcion NMS (P,c);
input  : Lista de valores de interés, P y radio de vecindad c
output : Lista de valores procesado P_nms
P_nms ← 0 ;
para p_i ∈ P hacer
    Descartar ← false;
    para p_j ∈ P hacer
        si distancia(p_i,p_j) < Distancia_minima entonces
            si V(c,p_j) > V(c,p_i) entonces
                Descartar ← True
    si no Descartar entonces
        P_nms → P_nms ∪ P_i
```

Imagen 3D

Para el desarrollo del proyecto es necesario utilizar técnicas de imágenes en 3D. Las imágenes 3D tienen como objetivo recuperar una estructura tridimensional de una escena usando dos o más imágenes tomadas desde distinto ángulo. Es por ello que en esta sección se expondrán los modelos matemáticos para poder obtener una imagen 3D a través de un par de imágenes estereoscópicas. Cabe destacar que en los últimos años se ha valorado por los investigadores la creación de mapas de profundidad a través de técnicas de inteligencia artificial pero este tema no se abordará ya queda fuera del alcance de este trabajo [6].

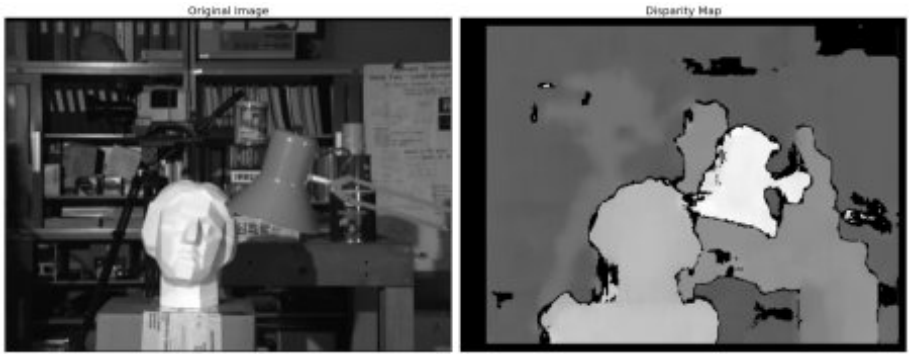


Figura 2.5 Imagen de profundidad [1].

Triangulación

El modelo de cámara estenopeica describe la relación entre las coordenadas de un punto en un espacio tridimensional y su proyección en un plano ideal de una cámara estenopeica.

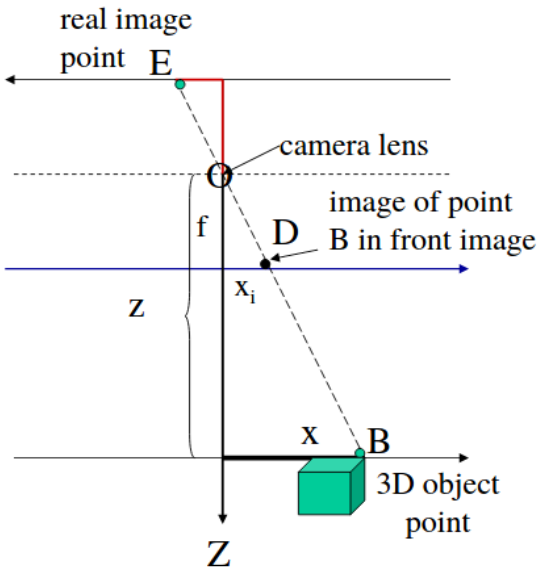


Figura 2.6 Geometría de una cámara estenopeica visto desde el eje y.

Mediante triángulos semejantes podemos obtener la siguiente relación:

$$\frac{x_i}{f} = \frac{x}{z} \tag{2.12}$$

Siendo O el punto de apertura de la cámara, f la distancia local , B es el punto en el espacio tridimensional y E su proyección en el plano de la imagen.

De este modelo podemos pasar al modelo más simple de dos cámaras en el plano epipolar para cámaras paralelas:

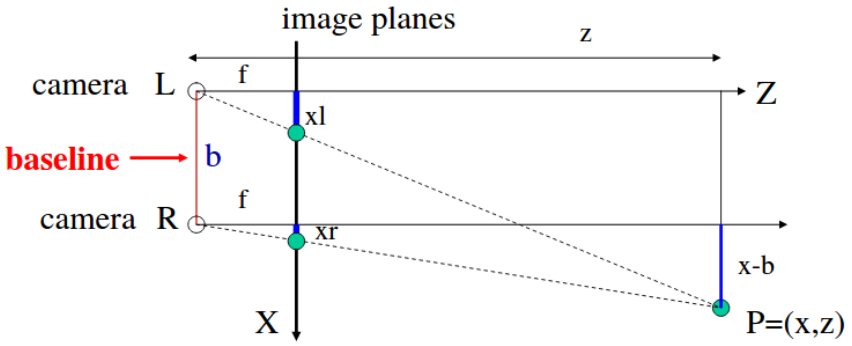


Figura 2.7 Geometría de dos cámaras paralelas en el plano epipolar.

Denominando b como la distancia entre las dos cámaras. Haciendo un razonamiento mediante triángulos semejantes, como en el modelo anterior, llegamos a las siguientes ecuaciones:

$$\frac{z}{f} = \frac{x}{x1} \tag{2.13}$$

$$\frac{z}{f} = \frac{x-b}{xr} \tag{2.14}$$

$$\frac{z}{f} = \frac{y}{y1} = \frac{y}{yr} \tag{2.15}$$

Para cámaras estero con los ejes paralelos, distancia focal f, distancia entre lentes b, la relación entre los puntos de las imagenes y las coordenadas del espacio tridimensional son:

$$\begin{array}{rcl} z & = & \frac{f*b}{x_l - x_r} = \frac{f*b}{d} \\ x & = & \frac{x_l * z}{f} = b + \frac{x_r * z}{f} \\ y & = & \frac{y_l * z}{f} = \frac{y_r * z}{f} \end{array} \tag{2.16}$$

Definiendo la disparidad d como la distancia entre la proyección del mismo punto en las dos imágenes, siendo este valor inversamente proporcional a la profundidad del objeto. El método para determinar la profundidad a través de la disparidad, llamado triangulación, no esta exento de dificultades como la necesidad de conocer la distancia focal o la distancia entre las lentes, lo cual puede ser resuelto mediante calibración, o la necesidad de obtener la correspondencia de los puntos en ambas imágenes.

Correspondencia

El problema de correspondencia se puede atajar de varias maneras, utilizando técnicas de correlación de ventanas entre imágenes o técnicas de *features* como las expuestas en los apartados anteriores. Gracias a la geometría descrita en la sección anterior podemos mejorar el coste computacional de las técnicas de correspondencia, observando la figura 2.8 los puntos $C1$, $C2$ y P forman un plano epipolar que corta los planos de las imágenes de ambas lentes por lo que, como se observa en la Figura 2.9 se puede asegurar que el punto de correspondencia de $P1$ sobre el plano de la lente $C2$ se encontrará contenido en la recta epipolar mostrada.

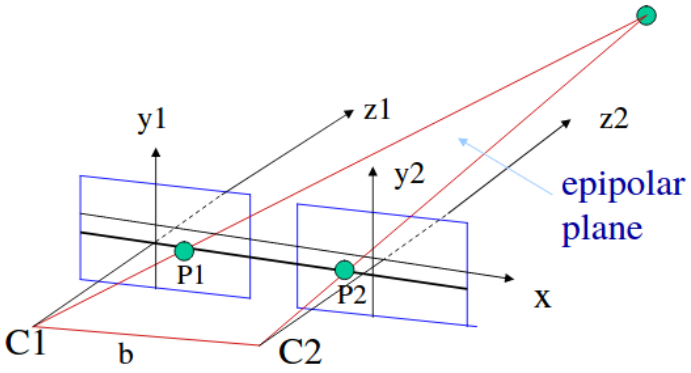


Figura 2.8 Plano epipolar.

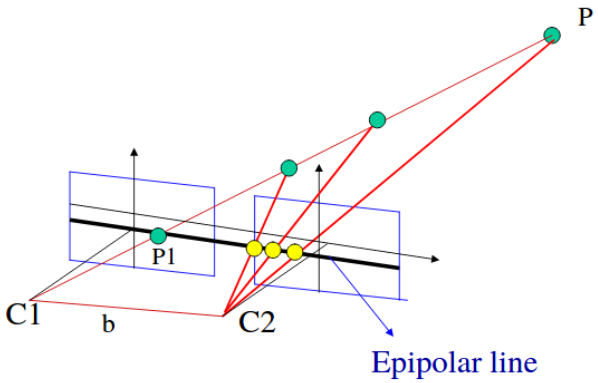


Figura 2.9 Recta epipolar.

Campo de visión

El campo de visión o **FOV** (*field of view*) se define como la zona se solapamiento entre las imágenes tomadas por ambas cámaras.

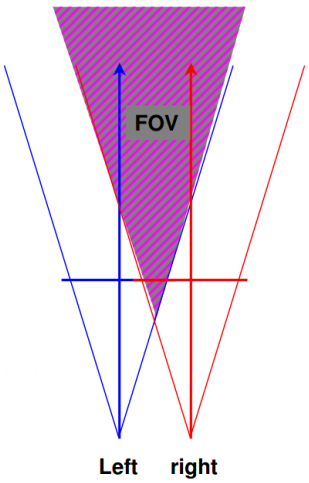


Figura 2.10 Representación del campo de visión de una camara estéreo.

Este parámetro puede modificarse alterando la distancia entre las dos cámaras lo cual tiene las siguientes consecuencias: si aumentamos la distancia entre las dos cámaras tendremos una mejor estimación de la profundidad, sin embargo, reduciremos el FOV y complicaremos la correspondencia debido al aumento de la posibilidad de producirse oclusión.

Coordenadas y transformaciones homogéneas

Las coordenadas homogéneas son un sistema usado en la geometría proyectiva. En concreto, en la robótica, nos permite la localización de un objeto respecto a otro mediante cambios de sistemas de coordenadas. Suponiendo un ejemplo en el cual disponemos de un brazo articulado y un objeto de interés, conocidas las posiciones de la base del brazo y del objeto respecto a un eje de coordenadas fijo, podemos proyectar la posición del objeto respecto de los ejes de coordenadas del actuador del brazo.
Las coordenadas homogéneas han sido una herramienta clave en el proyecto, gracias a ellas se ha podido situar en el espacio las ventanas detectadas y conocer su posición y orientación respecto al UAV. En esta sección se detallarán los principios matemáticos de estas coordenadas.

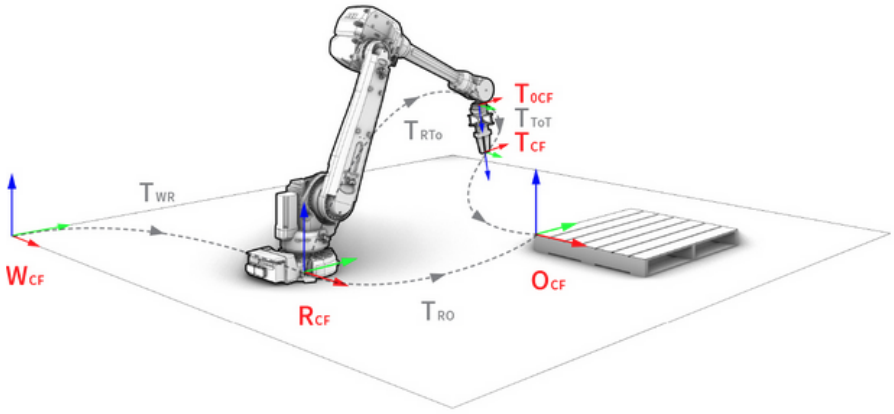


Figura 2.11 Representación 3D brazo articulado.

Punto

Se define un punto como una posición en el espacio. Los puntos serán representados mediante un vector columna:

$$p_1 = [x_1 y_1 z_1]^T \tag{2.17}$$

Operador

Un operador es cualquier proceso por el cual convertimos un punto del espacio en otro.

$$p_2 = \begin{bmatrix} x_2 \\ y_2 \\ z_2 \end{bmatrix} = T p_1 = \begin{bmatrix} t_{xx} & t_{xy} & t_{xz} \\ t_{yx} & t_{yy} & t_{yz} \\ t_{zx} & t_{zy} & t_{zz} \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} = \begin{bmatrix} t_{xx}x_1 + t_{xy}y_1 + t_{xz}z_1 \\ t_{yx}x_1 + t_{yy}y_1 + t_{yz}z_1 \\ t_{zx}x_1 + t_{zy}y_1 + t_{zz}z_1 \end{bmatrix} \tag{2.18}$$

Con este operador podemos hacer transformaciones como el escalado, reflexión, rotación y proyección.

Coordenadas homogéneas

Aunque el operador es bastante general, no llega a ser lo suficientemente amplio para muchas aplicaciones de la robótica ya que no permite representar una traslación. Dicha transformación puede ser representada como la suma de dos vectores:

$$p_2 = p_1 + p_3 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} + \begin{bmatrix} x_3 \\ y_3 \\ z_3 \end{bmatrix} \tag{2.19}$$

Esta transformación no puede representarse mediante una matriz porque, en el caso general, no puede expresarse como una combinación lineal de p_1.
Sin embargo, esto puede resolverse mediante las coordenadas homogéneas. Añadimos un elemento extra a los puntos para representar un factor de escala.

$$p_1 = [x_1 y_1 z_1 w_1]^T \tag{2.20}$$

Para poder proyectar este factor en un espacio 3D se divide cada componente por el valor de escala.

$$p_1 = [\frac{x_1}{w_1} \frac{y_1}{w_1} \frac{z_1}{w_1}]^T \tag{2.21}$$

$$p_1 = [x'_1 y'_1 z'_1 1]^T \tag{2.22}$$

Transformaciones homonogéneas

Mediante las transformaciones homogéneas podemos realizar las traslaciones mediante operaciones matriciales.

$$p_2 = p_1 + p_3 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} + \begin{bmatrix} x_3 \\ y_3 \\ z_3 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & x_3 \\ 0 & 1 & 0 & y_3 \\ 0 & 0 & 1 & z_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} = Trans(p_3) p_1 \tag{2.23}$$

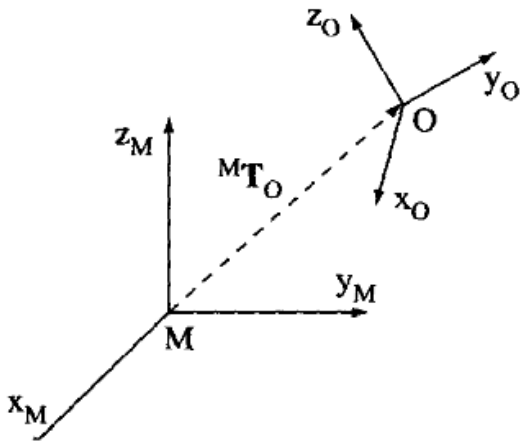


Figura 2.12 Representación transformación homogénea.

Transformaciones homogéneas como eje de coordenadas

Las coordenadas homogéneas no solo sirven para representar puntos, también pueden usarse para representar una dirección:

$$i = [1\,0\,0\,0]^T \quad j = [0\,1\,0\,0]^T \quad k = [0\,0\,1\,0]^T \tag{2.24}$$

Dichas tres columnas pueden agruparse con las coordenadas homogéneas origen para formar la matriz identidad:

$$[i\ j\ k\ o] = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.25}$$

Transformaciones homogéneas como transformaciones coordinadas

Supongamos un punto p expresado en los ejes de coordenadas b:

$$p^b = [x^b y^b z^b 1] \tag{2.26}$$

Para poder representar ese punto respecto a otras coordenadas definimos la matriz T_b^a que transforma desde los ejes b a los ejes a:

$$p^a = T_b^a p^b \tag{2.27}$$

Siendo la matriz T:

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix} \tag{2.28}$$

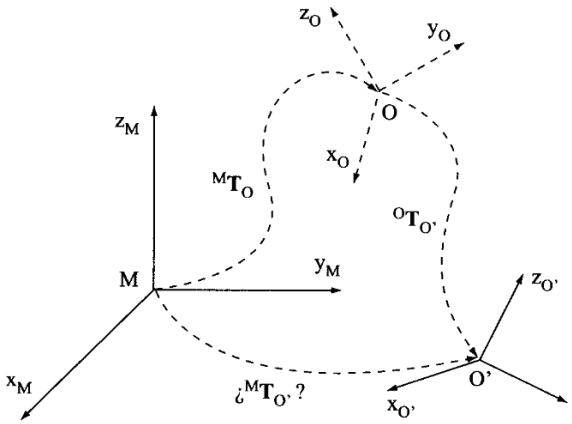


Figura 2.13 Representación transformación coordinada.

Herramientas utilizadas

Introducción

Antes de exponer los distintos métodos y algoritmos desarrollados como parte de la realización de este trabajo cabe destacar las herramientas de las que he hecho uso durante el desarrollo del proyecto. En este capítulo se describirá de una forma breve el software utilizado; desde el framework de desarrollo de robótica ROS al control de versiones realizado con Git.

ROS

ROS es un sistema operativo robótico de código abierto que proporciona un entorno estructurado y robusto para el desarrollo de software para robots. Se puede definir como una colección de herramientas, librerías y convenciones que facilitan el desarrollo de software enfocado a la robótica [23].

Figura 3.1 Logo ROS [25].

¿Por qué ROS?

ROS ha sido un pilar base en el desarrollo de este trabajo, pero antes de decidirme por utilizar este framework indagué en los distintos pros y contras que tienen desarrollar en ROS. Los siguientes puntos sintetizan la razón de su uso:

1. **Estándar en la industria:** En los últimos años ROS puede considerarse un estándar en el campo de la robótica, contando con una gran cantidad de *papers* publicados sobre investigación y una gran comunidad de desarrollo.
2. **Integración con OpenCV:** Para la parte de visión del proyecto se consideró desde un principio el uso de OpenCV ya que disponía de experiencia previa.
3. **Multilenguaje:** La posibilidad de poder programar tanto en Python como en C++ lo considero un punto a favor, dando la libertad de poder intercalar código de ambos lenguajes dependiendo de las necesidades del momento.
4. **Código abierto:** He tenido preferencia por hacer uso de software de código abierto en el desarrollo de este proyecto.

Durante el desarrollo del proyecto he tenido multitud de razones más para el uso de ROS, como la integración con Gazebo o la estructuración en nodos, reafirmando la decisión de utilizar el sistema operativo.

Gazebo

Para comprobar el proyecto realizado necesitaba un entorno donde simular el vuelo del UAV y poder obtener métricas. Gazebo suple todas las necesidades del proyecto, un entorno de simulación 3D especializado en robótica, brindando la posibilidad de generar datos de sensores [17].

Figura 3.2 Logo Gazebo [10].

OpenCV

Para el tratamiento de imágenes me decanté por el uso de OpenCV. Necesitaba una librería que me permitiera implementar distintos algoritmos de detección de *features* y realizar tratamiento de imágenes estéreo, al comprobar que OpenCV cumplía con todos mis requisitos la decisión fue sencilla.

Figura 3.3 Logo OpenCV.

A modo de resumen OpenCV es una librería libre de visión artificial originalmente desarrollada por *Intel*. Utilizada para aplicaciones como reconocimiento facial, reconocimiento de gestos, robótica movil o realidad aumentada [4].

Unmanned aerial vehicle abstraction layer

Uno de los objetivos del proyecto es el movimiento del UAV para realizar el cruce de la ventana, para ello he utilizado el UAL (*Unmanned aerial vehicle abstraction layer*), una librería desarrollada en la Universidad de Sevilla por el Grupo de Robótica, Visión y Control. La librería se encuentra alojada en *Github* *https://github.com/grvcTeam/grvc-ual* donde se puede acceder a todo lo necesario para realizar su implementación.

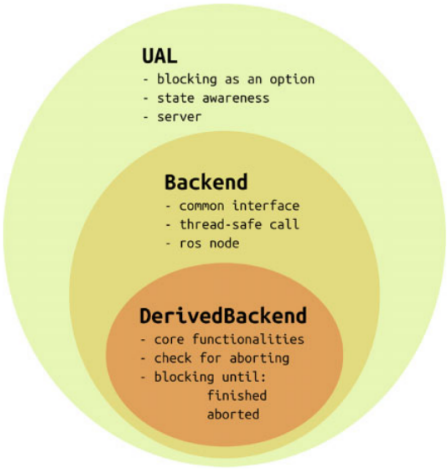


Figura 3.4 Esquema en forma de capas del funcionamiento de UAL.

Como se puede observar en la imagen, UAL permite comandar el movimiento de un UAV sin necesidad de utilizar la interfaz del *Backend*, teniendo un sistema más flexible a cambios en el autopiloto.
UAL viene incorporado con dos formas de acceder a los comandos: mediante servicios o una clase que se puede instanciar en un objeto de C++. En el caso de este proyecto se hace uso de los comandos mediante clase ya que proporciona una respuesta más rápida que su implementación mediante servicios [24].

Git

Por último he querido mencionar el uso del *software* de control de versiones durante la totalidad del proyecto. La decisión de usar Git y no otro *software* es debido a que ya tenía experiencia con el mismo y lo considero una herramienta importante a la hora de desarrollar un proyecto.

Figura 3.5 Logo Github.

Todo el proyecto que se presenta en esta memoria esta alojado en *https://github.com/jilker/detector_tfg*, lugar donde se puede observar la evolución del proyecto.

Librería Tf2

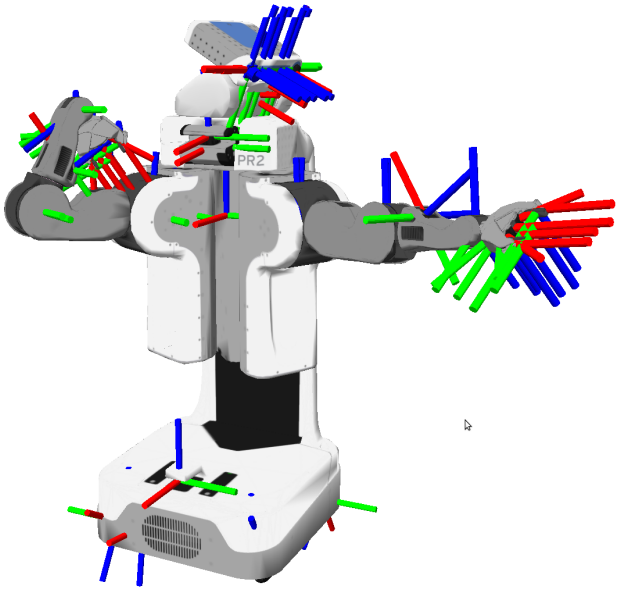


Figura 3.6 Representación de los ejes de coordenadas de un robot.

Mediante la librería tf2 en ROS tenemos la posibilidad de realizar todas las transformaciones que se han expuesto en este capítulo mediante código, ademas de herramientas de visualización [9]. En el Apéndice B se incluyen diagramas en forma de árbol que representan la relación entre los ejes que componen el sistema.

Método propuesto

Introducción

En este capítulo se plantea el método propuesto para el cruce de ventanas y los módulos que lo componen. Como se ha indicado en los capítulos anteriores se ha definido como objetivo poder detectar y realizar el cruce de forma autónoma, para ello se han definido dos nodos ROS: uno de ellos se encargará del tratamiento de imágenes 3D mientras que el otro se encargá del movimiento del UAV. En el Apéndice A se incluye un diagrama de bloques de los nodos ROS que componen la totalidad del sistema.

Método de visión

En esta sección del capítulo se expondrá el método diseñado para realizar el tratamiento de imágenes en el proyecto, comenzando por un esquema general del método al cuál seguirán secciones que analizarán en detalle los puntos más destacados.

Esquema general de visión

Este nodo se encargará del tratamiento de las imágenes capturadas por la cámara estéreo, dichas imágenes se utilizan produciendo mapas de profundidad a los cuales se les aplica los algoritmos de detección de esquinas. De esta forma estamos buscando en un mapa de profundidad si se encuentra alguna variación que pueda llegar a ser considerada ventana. Para realizarlo se ha definido un flujo de trabajo para cada imagen:

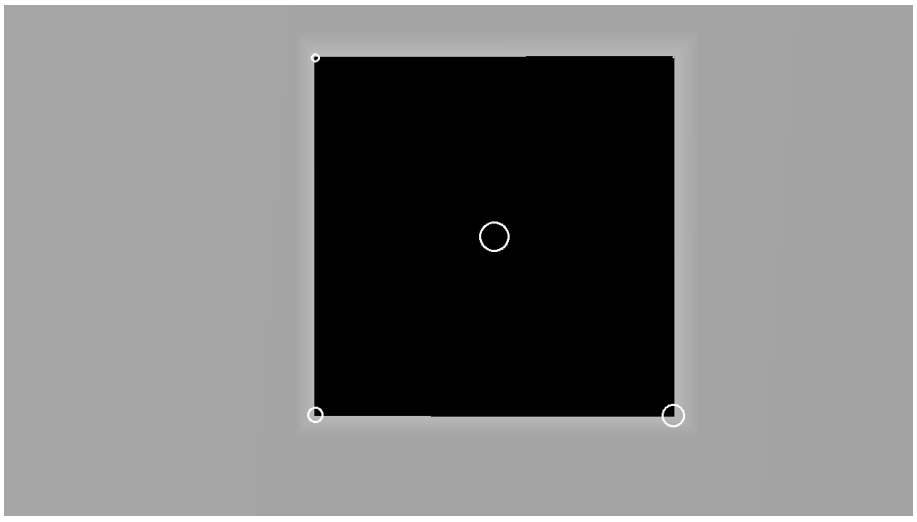


Figura 4.1 Simulación detección de ventana.

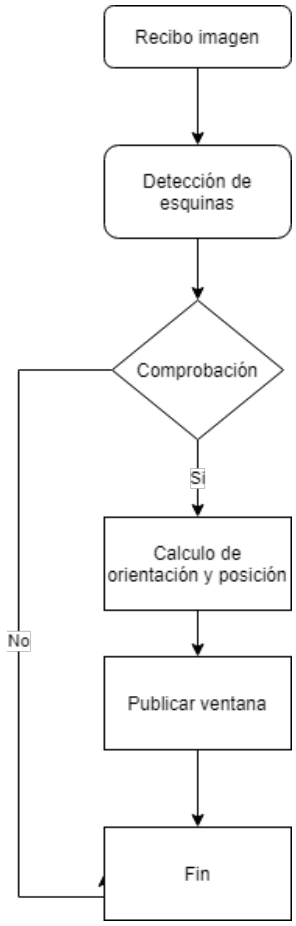


Figura 4.2 Diagrama de flujo detección de ventanas.

Algoritmo 2: Detección de ventanas

Detección de ventanas (M);
input : Imagen de profundidad M
output : Posición y orientación de la ventana P
Inicio;
Algoritmo de detección en M;
si *Esquinas cumplen los requisitos impuestos* **entonces**
    Algoritmo de orientación y posición;
    Publicar la ventana;

Este nodo estará constantemente revisando las imágenes publicadas por la cámara equipada por lo que se ha intentado minimizar el coste computacional del mismo para no entorpecer al resto de tareas.

Recibir imágenes

Toda la comunicación del proyecto se ha realizado dentro de ROS, lo cual ha facilitado mucho el trabajo y para recibir las imágenes, únicamente el programa se subscribirá al *topic* correspondiente. Un sensor cámara deberá estar publicando en dicho *topic* los mapas de profundidad que vaya captando, asumiéndose que la cámara ya realiza el tratamiento de las imágenes estereoscópicas.
Una vez recibidas las imágenes he utilizado la librería *cv_bridge* para convertirlas del formato mensaje de ROS al formato utilizado por OpenCV.

Detección de esquinas

Se plantea la necesidad de identificar las ventanas en las imágenes de profundidad estableciendo como metodología la detección de las cuatro esquinas correspondientes, es por ello que se han implementado

tres algoritmos: *Harris corner*, *Shi-Tomasi* y FAST, todos descritos en el Capítulo 2. Durante las pruebas preliminares al desarrollo del algoritmo se comprobó que todos suplían las necesidades del proyecto por lo que durante la etapa de experimentos se valorará la eficacia de cada uno, aunque como se ha descrito en el estado del arte FAST deberá tener un mejor rendimiento.

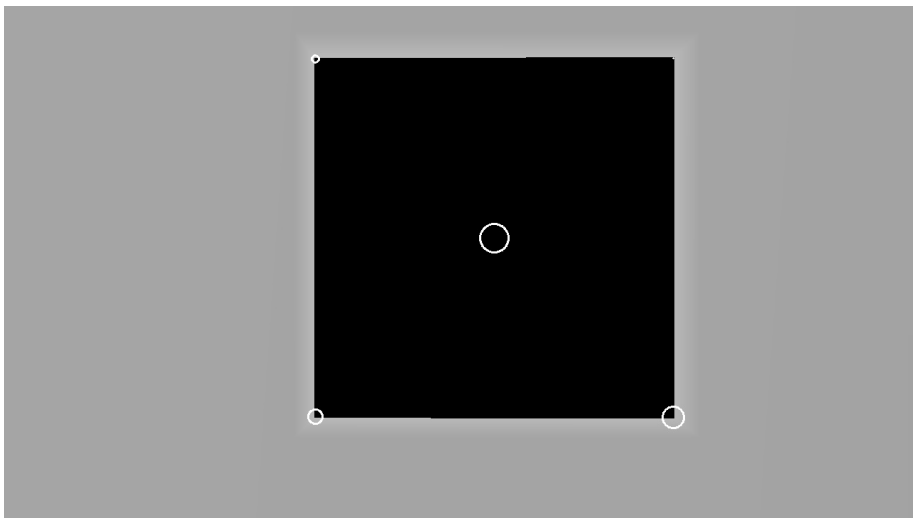


Figura 4.3 Simulación detección de ventana.

Comprobación de ventana

Esta etapa del algoritmo es crucial que sea robusta, ya que un error en la comprobación de la existencia de una ventana podría provocar una colisión del UAV, es por ello que se han definido dos métodos distintos para la comprobación.

Verificación por dimensiones

En esta solución es necesario el conocimiento previo de las dimensiones de las ventanas a cruzar, esto puede traducirse como una reducción de la operatividad del algoritmo pero permite un alto nivel de robusted. El funcionamiento es sencillo y se expone en el siguiente pseudocódigo:

Algoritmo 3: Verificación por dimensiones

Verificación por dimensiones (P);
input : Vector esquinas de las ventana ordenado P
output : Variable booleana
para p_i *en* P **hacer**
  para p_j *en* P **hacer**
    si $!verificacion\_distancia(p_i, p_j)$ **entonces**
      Return False;
Return True

La verificación de distancia únicamente comprueba que la distancia entre los puntos es igual a la distancia medida, es por ello que es necesario tener el vector de esquinas ordenados según la regla que se defina previamente, en mi caso se ha definido la esquina superior derecha como primer punto y el resto se han ordenado siguiendo las agujas del reloj

Verificación por histograma

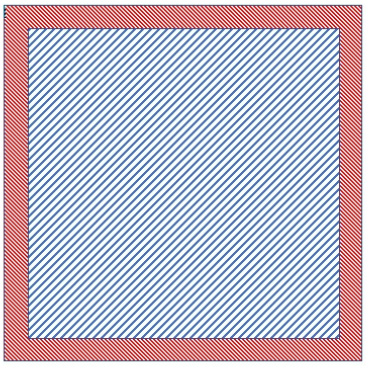


Figura 4.4 Ilustración técnica de comprobación por histograma.

Para este caso se realizará una comparación mediante el histograma de la imagen. El histograma es una representación gráfica que refleja la cantidad de pixels en una imagen que tienen una determinada luminosidad, para nuestro caso la luminosidad de la imagen de profundidad nos indica la distancia a la que se encuentra el punto proyectado del objetivo de la cámara. Bajo esta premisa se diseñó el siguiente algoritmo:

Algoritmo 4: Verificación por histograma

Verificación por histograma (P);
input : Vector esquinas de las ventana ordenado P
output : Variable booleana
$N \rightarrow$ *Expansión del polígono* ;
Calculo de histograma del área definida en el polígono P ;
Calculo de histograma del área definida en el polígono P y $P+N$;
si *Hist(P)< Hist(P+N)* **entonces**
    Return True;
en otro caso
    Return False;

Este método tiene la ventaja de no necesitar conocer previamente las dimensiones de las ventanas, como contra, la necesidad de establecer varios valores empíricamente: como el valor de expansión del área a comprobar, la distancia entre ambos histogramas o la inclusión de alguna restricción geométrica para asegurar que la ventana tiene las dimensiones seguras para realizar el cruce.
Durante la etapa de experimentos se valorarán ambas soluciones para poder definir que solución tomar finalmente.

Calculo de posición y orientación

Para esta sección se ha hecho uso del modelo de cámara estenopeica, por ello para determinar la posición en el espacio respecto de los ejes cámara se utiliza las siguientes expresiones matemáticas:

$$x = \frac{(x^* - c_x)}{f_x} * z \quad y = \frac{(y^* - c_y)}{f_y} * z \quad z = z \tag{4.1}$$

Siendo x^* , y^* la posición del píxel en la imagen, c_x , c_y el centro de la imagen y f_x , f_y la distancia focal.[20]

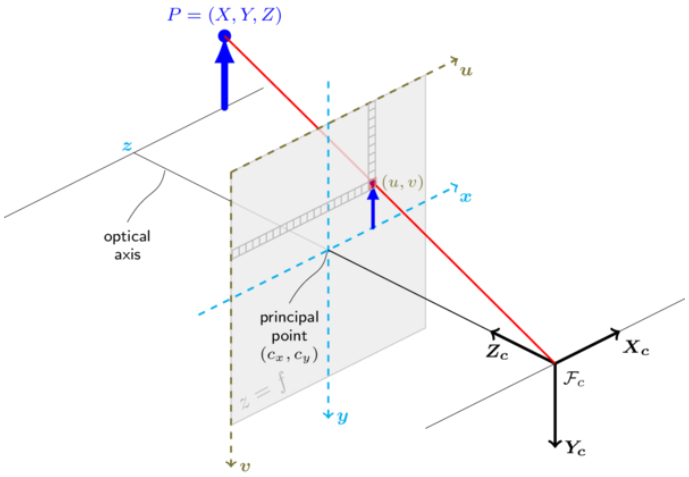


Figura 4.5 Ilustración reconstrucción de la imagen.

Una vez obtenidos los puntos en el espacio es relativamente fácil sencillo obtener la orientación de la ventana respecto de los ejes del UAV, a través de los cuales podemos realizar trasformaciones hasta obtener la posición y orientación respecto de los ejes fijos. Para ello definimos unos ejes de referencia formando dos vectores perpendiculares mediante la unión de los puntos y a través de una multiplicación vectorial conseguimos el vector normal al plano de la ventana.
Para obtener la orientación solo tendremos que resolver la siguiente ecuación matricial:

$$D = T * V \tag{4.2}$$

Siendo D los ejes de coordenadas del UAV, T la matriz de transformación homogénea y V los ejes asociados a la ventana.

Publicar ventana

Una vez detectada la ventana y posicionada en el espacio 3D se publican unos ejes de referencia en el centroide de la ventana. Estos son publicados respecto a unos ejes fijos globales, facilitando la tarea de compartir la información recabada con el UAV con el resto del equipo.

Visual Servoing

Tras realizar el módulo de detección de ventanas se plantearon las distintas posibilidades de cara a realizar el cruce de la ventana. *Visual servoing* se denomina al conjunto de técnicas que usa la información que recibe la cámara para determinar el movimiento del robot, existen varios modelos de estimación pero para este proyecto los más adecuados son los siguientes [15].

Basado en imagen

Esta técnica es la más clásica de las dos, se basa en controlar el movimiento del robot para obtener en la visión una posición determinada del objeto identificado.

Basado en posición

Esta técnica, por el contrario, convierte los objetos de interés detectados en la imagen a puntos en el espacio, de esta forma el robot no se mueve para satisfacer una imagen objetivo si no una posición objetivo.
Se ha optado por esta técnica debido a que de esta forma la identificación de una ventana por parte de un UAV permite la estimación de la posición para el resto. Se realiza un movimiento en base al espacio 3D, por contra partida aumentamos la posibilidad de tener errores debido a la necesidad de conocer en todo momento la posición en el espacio de los UAVs

Servidor UAL

Como se indicó en el capítulo anterior se ha hecho uso de un servidor UAL para manejar el movimiento del UAV bajo la librería en C++, es por ello que se ha podido hacer uso de funciones de alto nivel, agilizando el trabajo. A continuación se define un flujo de trabajo para el servidor:

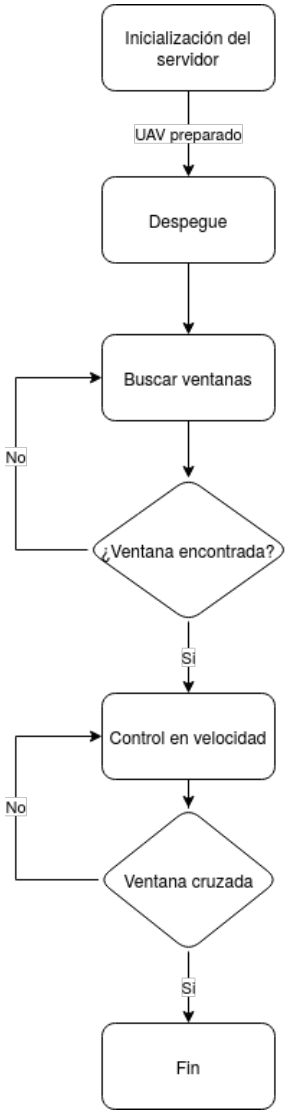


Figura 4.6 Diagrama de flujo servidor UAL.

A continuación se desglosará como se ha llevado a cabo cada una de las acciones indicadas en el diagrama de flujo, en especial el control en velocidad integrando el módulo LIDAR para corregir errores provenientes de la odometría.

Inicialización y despegue

Gracias a la aportación del servidor UAL esta etapa del sistema de navegación se sintetiza en la definición de la clase UAL dentro del código destinado al movimiento y la utilización de los comandos de alto nivel para la orden de despegue.

Búsqueda de ventanas y detección de ventanas

Para la búsqueda de ventanas se ha implementado un sistema de seguimiento de puntos previamente definidos. Esta implementación se realizó de dicha manera para facilitar la repetibilidad de los experimentos y disminuir

el tiempo de los mismos. Queda pendiente en el apartado de trabajo adicional la inclusión de algún algoritmo de cálculo de rutas, ya sea mediante técnicas SLAM o utilizando un sencillo sistema de seguidor de pared gracias al LIDAR incorporado.

Control en velocidad

Nos encontramos en la situación la cual el UAV a detectado una ventana y dispone de los ejes publicados por el sistema de visión, en ese instante se inicia el siguiente algoritmo de control de velocidad:

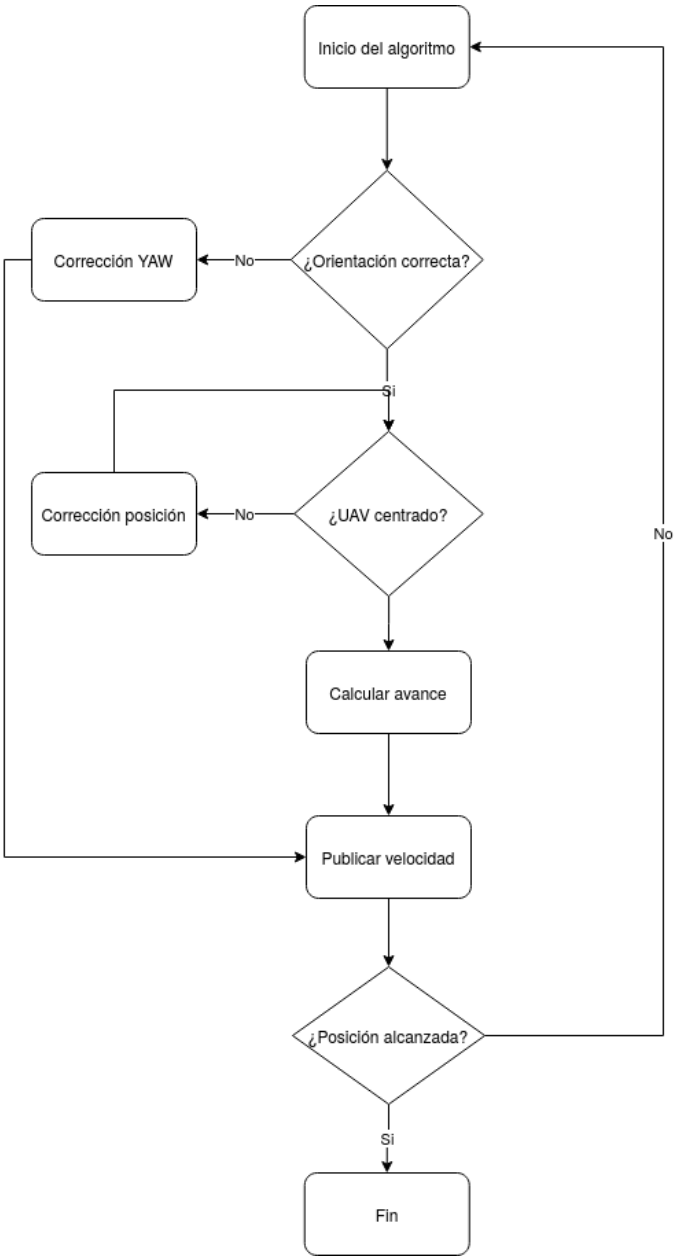


Figura 4.7 Diagrama de flujo del control de velocidad.

Calculo del error

Para comprobar si el robot se encuentra orientado o centrado respecto de la ventana se ha definido el error como:

$$e = s_d - s_r \tag{4.3}$$

Siendo s_d el estado deseado y s_r el estado real. Definiendo el estado de la siguiente forma:

$$s = \begin{bmatrix} \text{posición} \\ yaw \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \theta \end{bmatrix} \tag{4.4}$$

De esta manera la actuación de la corrección se efectúa según el error calculado.
Durante el capítulo de experimentos se detallarán las distintas ecuaciones planteadas para gobernar el control en velocidad, pasando desde algunas más conservadoras y seguras a otras más agresivas y rápidas.

Posicionamiento

Una de la penalizaciones en la competición MBZIRC era el uso de GPS para el posicionamiento. Es por ello que en este proyecto solo se ha hecho uso de odometría para el posicionamiento del UAV. La inclusión del láser solo aparece durante la realización del cruce de ventanas. Gracias al LIDAR incorporado al robot aéreo durante la maniobra de cruce, en paralelo, se esta midiendo con el láser la distancia a los laterales de la ventana ,es decir, al marco izquierdo y derecho de la misma. De esta forma tenemos la posibilidad de actuar realizando una acción de control tal que minimice la diferencia de distancia del UAV al marco.

$$d_{diff} = d_{izq} - d_{der} \tag{4.5}$$

Siendo d_{izq} la distancia al punto más cercano del lado izquierdo de la ventana y d_{der} el homólogo al lado derecho. Definiendo entonces la acción de control tal que minimice el valor de la diferencia.

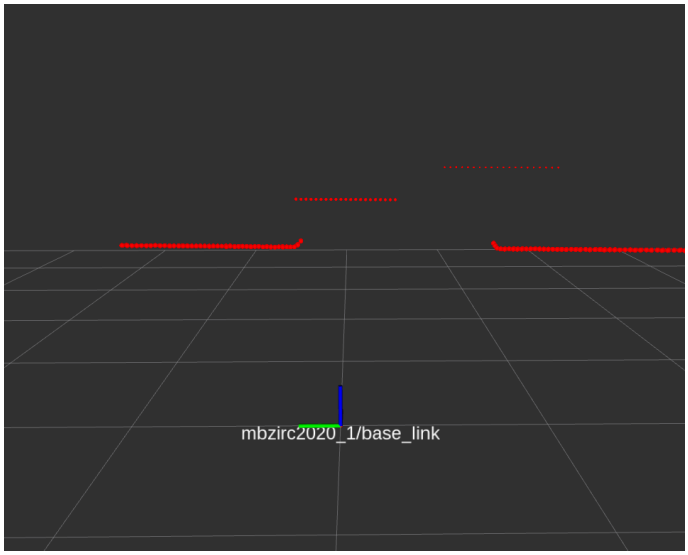


Figura 4.8 Representación de la información percibida por el láser en *Rviz*.

Simulación y experimentos

Introducción

En este penúltimo capítulo se expondrán las simulaciones y experimentos realizados para comprobar el funcionamiento del programa descrito en los apartados anteriores. Cabe destacar que también servirán dichos experimentos para fijar varios planteamientos que se han realizado, como la selección del algoritmo de detección de ventanas.
La estructura del capitulo será de una sección para cada experimento, donde se indicarán: descripción general del experimento, la motivación del experimento, las condiciones concretas, resultados y conclusiones extraídas.

Simulación

La simulación se ha realizado utilizando un mundo diseñado en Gazebo por el equipo de la Universidad de Sevilla para la competición MBZIRC. Dicho mundo ha sido modificado para poder implementar el programa diseñado, como sustituyendo el servidor UAL o el programa de detección de ventanas. Las simulaciones constan de un UAV equipado con la instrumentación y un edificio con distintas ventanas de 2x2 metros:

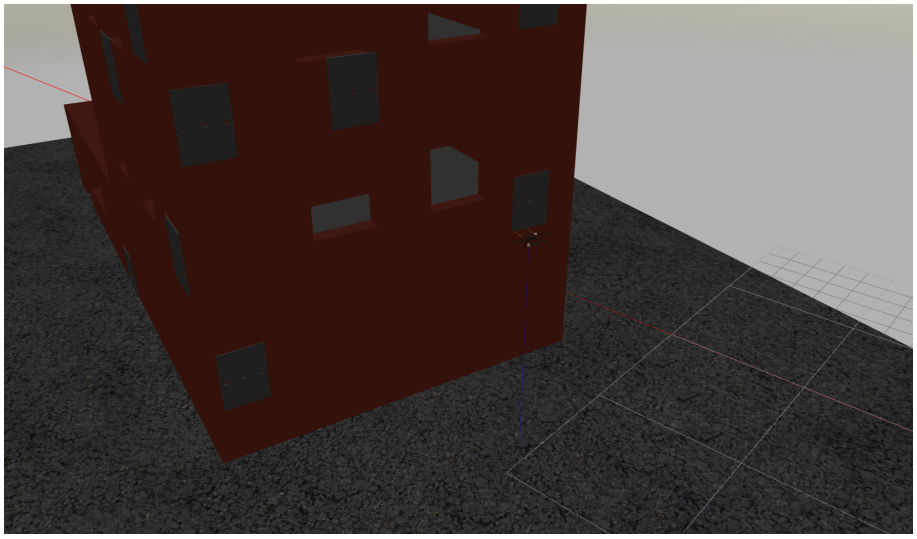


Figura 5.1 Ilustración simulación en gazebo.

Todas las simulaciones descritas en este capítulo se han realizado bajo las siguientes condiciones:

Tabla 5.1 Condiciones del equipo de simulación.

S.O.	Ubuntu 16.04 LTS 64-bit
CPU	Intel® Pentium(R) CPU G4560 @ 3.50GHz × 4
GPU	AMD Radeon RX 470 Graphics (POLARIS10 / DRM 3.23.0 / 4.15.0-112-generic, LLVM 6.0.0)
RAM	8 GB
ROS	Kinetic Kame
OpenCV	2.4.9
Gcc	5.4.0
C++	11
Python	2.7.12

Todo los datos de software no indicados en la tabla se supondrán bajo la última versión compatible con el equipo definido a fecha de agosto de 2020.

Algoritmo de detección de esquinas

En este experimento se comparará el desempeño de los tres algoritmos descritos en la teoría para la detección de esquinas para la detección de una ventana. Este experimento tiene como objetivo poder determinar cual de los algoritmos tienen mejor desempeño para la detección. Se asume que los tres algoritmos tienen la capacidad de detectar las esquinas de las ventanas, como se ha podido comprobar a priori, por lo que se valorará el gasto computacional de cada uno. Según lo expuesto en el capitulo 2 deberíamos obtener unos resultados muy favorables del algoritmo FAST.

Condiciones

Se han utilizado las implementaciones de los algoritmos realizadas en la librería OpenCV. Para realizar este ensayo se ha situado el UAV a una distancia de 4 metros frente a una ventana y se ha registrado el tiempo de ejecución de cada algoritmo. Se han seleccionado los siguientes valores para las variables de cada algoritmo:

Harris corner

A través de ensayos preliminares se han fijado los valores indicados, con los cuales, el algoritmo cumple su función correctamente:

Tabla 5.2 Valores para Harris Corner.

Variable	Valor	
Tamaño ventana	2	píxel
K	0.04	
Valor límite	150	

Repasando lo indicado en el Capitulo 2, K es un valor empírico utilizado para determinar la puntuación de las esquinas, siendo $K \in [0.04,0.06]$, valor límite establece el valor mínimo de puntuación para determinar una esquina y tamaño ventana establece el tamaño de la ventana de búsqueda del algoritmo, en este caso una ventana de 2x2 píxeles.

Shi-Tomasi

Para la implementación de *Shi-Tomasi* se ha hecho uso de la función *good features to track* perteneciente a OpenCV que ademas añade la capacidad de establecer una distancia mínima entre los puntos encontrados. Se presenta también la variable de calidad la cuál caracteriza la mínima calidad admisible de una esquina. Dicho valor es multiplicado por la mejor esquina detectada en la imagen, el cual corresponde con el valor mínimo de los autovalores, y todas las esquinas que tengan un valor inferior a este son descartadas [3].

Variable	Valor	
Calidad	0.01	
Mínima distancia	100	píxel
Ventana	2	píxel
K	0.04	

Repasando nuevamente las variables: Mínima distancia determina la distancia mínima que debe haber entre dos esquinas encontradas, ventana establece el tamaño de la ventana de búsqueda del algoritmo y K es el valor empírico.

FAST

A través de ensayos preliminares se han fijado los siguientes valores para un correcto funcionamiento del algoritmo:

Tabla 5.3 Valores para FAST.

Variable	Valor
nonmaxSuppression	True
Valor límite	100

Para la implementación de dicho algoritmo se ha utilizado la función perteneciente a OpenCV [2].

Resultados y conclusiones

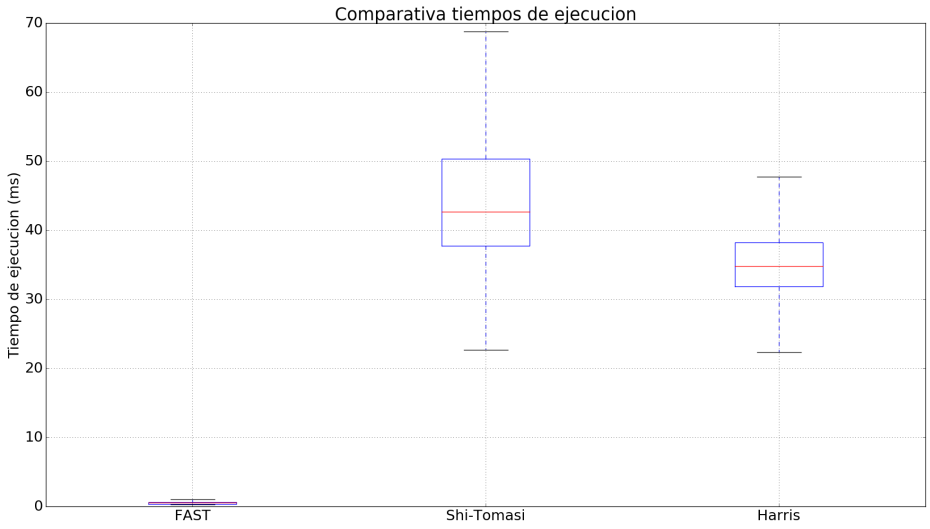


Figura 5.2 Comparativa de los tiempos de ejecución de los algoritmos de detección de esquinas.

Se han tomado 1000 medidas del tiempo de ejecución para cada uno de los algoritmos de distintos *frames* pertenecientes a la fachada del edificio simulado. Como se puede comprobar en la gráfica confirmamos que la velocidad del algoritmo FAST es muy superior a los demás algoritmos.
Con los resultados obtenidos se determina el uso del algoritmo FAST para la detección de esquinas. El resto de experimentos se asumirá que se usa dicho algoritmo.

Fusión láser y odometría en cruce de ventanas

En este experimento se determinará si la inclusión del láser supone una mejora en el cruce de las ventanas. Se ha observado que mediante el uso de odometría para el posicionamiento del robot respecto de la ventana

puede ocurrir en un bajo porcentaje choques con los laterales de las ventanas. Es por ello que mediante la inclusión del láser se desea aumentar la precisión con la que situamos el robot.

Condiciones

Se realizarán diez cruces de ventana sin el uso del láser, reiniciando el sistema entre cada uno de ellos, y se realizarán otros diez con el láser. La distancia entre el centroide de la ventana al centroide del UAV se utilizará como medida para determinar el grado de precisión. Se ha optado por una velocidad de cruce conservadora para este experimento.

Resultados y conclusiones

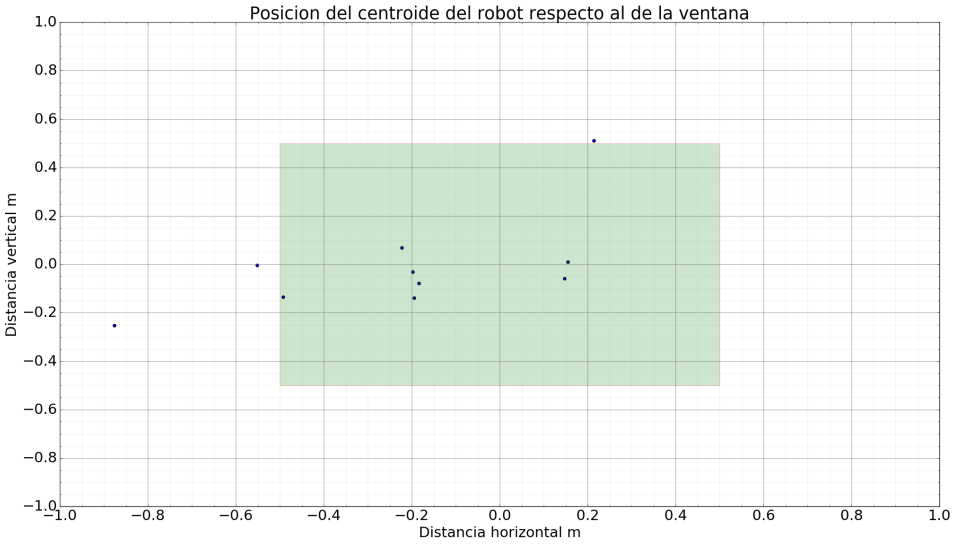


Figura 5.3 Cruce mediante odometría.

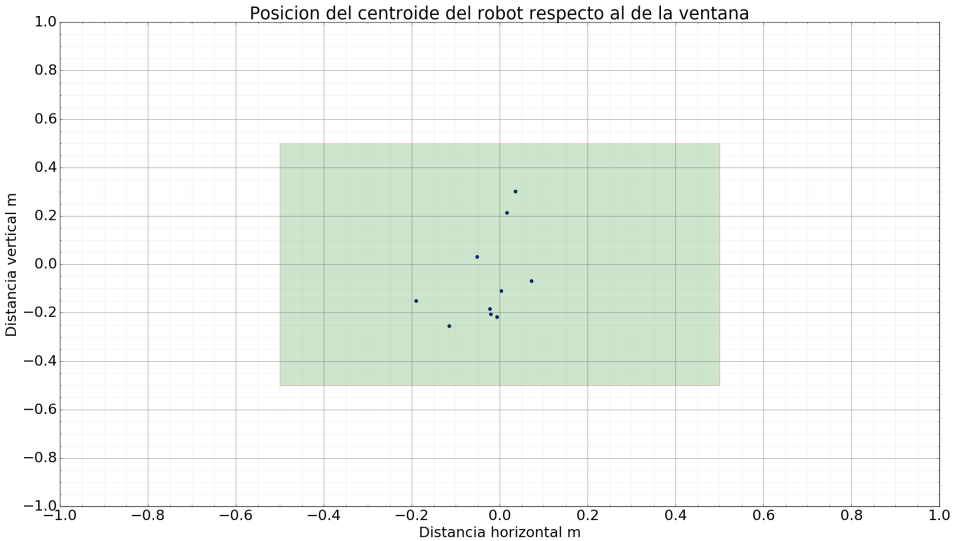


Figura 5.4 Cruce mediante odometría y correcciones láser.

Se ha fijado un área de color verde en las Figuras 5.4 y 5.3, dicha zona se ha considerado con el UAV cruza de forma segura a través de la ventana, cualquier valor fuera de ese área puede corresponder con un choque y no son admisibles.
Como se podía intuir el uso del láser ha mejorado notablemente el cruce de ventana, realizando de manera más robusta y segura. De ahora en adelante todos los experimentos realizados se supondrá que se ha utilizado el láser en la etapa de cruce.

Comparativa velocidad

Con este experimento comprobaremos cuanto podemos aumentar la velocidad de cruce sin perder la robustez que se ha obtenido en el ensayo anterior. Con este ensayo se busca obtener un método más rápido. La velocidad de cruce se calcula de la siguiente forma:

$$V_x = e(\text{posición}) * K + C \tag{5.1}$$

Siendo $e(\text{posición})$ el error entre la posición del UAV y la posición objetivo, K es la variable que modificaremos y C es una constante que para este experimento valdrá siempre 0.1.

Condiciones

Las condiciones del experimento se basaran en ir realizando cruces de ventana a cada vez más velocidad, registrando la distancia al centro como en el experimento anterior e intentar inferir una velocidad optima a través de los datos.

Resultados y conclusiones

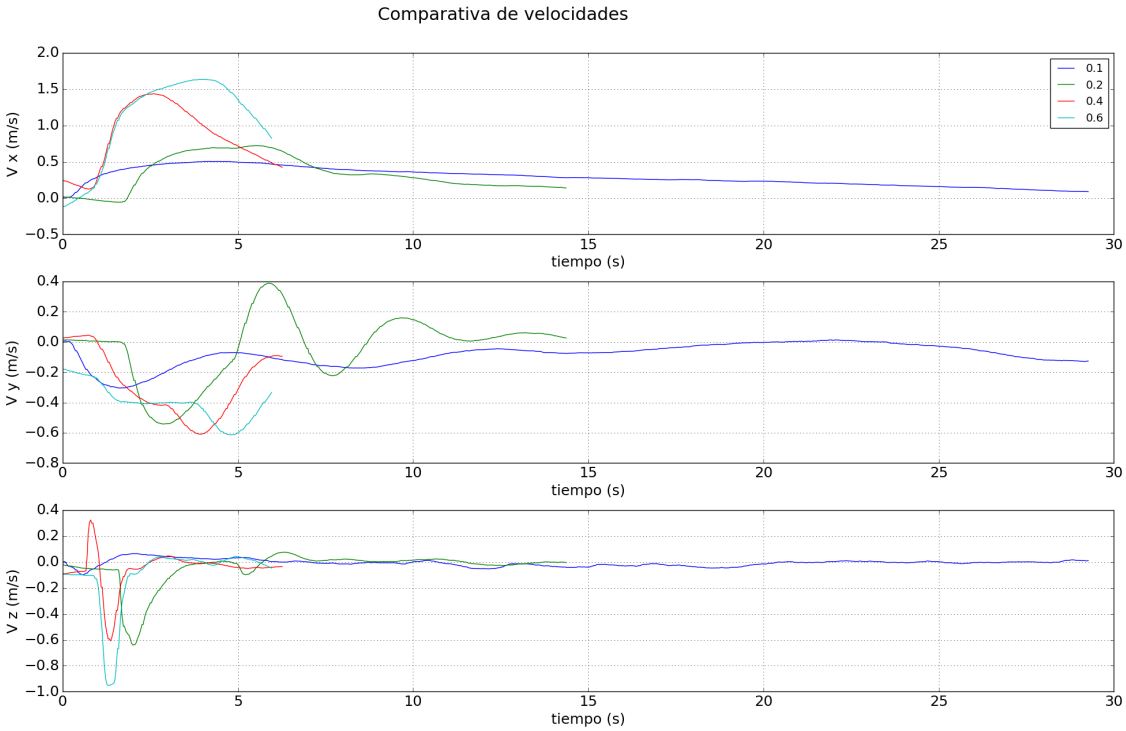


Figura 5.5 Comparativa de velocidades para distintos valores de K durante el cruce de una ventana.

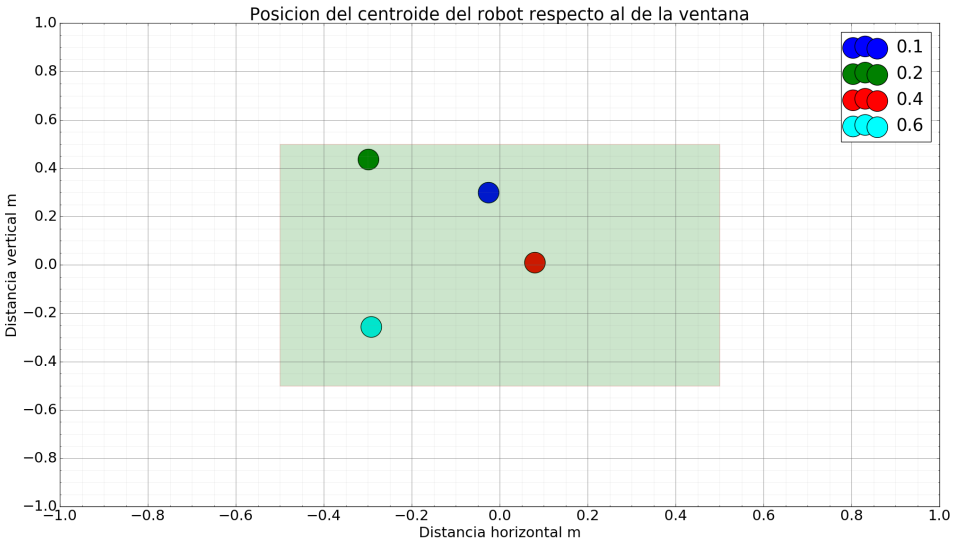


Figura 5.6 Comparativa de desviación para distintos valores de K durante el cruce de una ventana.

En las gráficas se muestran el comportamiento de las velocidades para distintos valores de K y la desviación del centroide correspondiente.
Como se puede observar en las Figuras 5.5 y 5.6 al aumentar la velocidad reducimos el tiempo para realizar el cruce a costa de aumentar el error, aun así, no se consideró el experimento concluyente para determinar un valor óptimo de K pero si una buena orientación para su selección de cara a las necesidades de implementación.

Cruce con dos UAVs

En este experimento se intentará realizar el cruce de ventanas mediante dos UAVs, para este caso se ha añadido un nuevo UAV idéntico al anterior pero se han modificado el comportamiento de ambos. En este caso el primer UAV se acercará a una ventana, la identificará, publicará su posición y tomará tierra dejando paso al segundo robot aéreo. Este segundo UAV despegará y realizará el cruce de ventana sin identificarla, eliminando por completo el módulo de visión, unicamente utilizando odometría para el posicionamiento y el láser para la etapa de cruce de ventana.
La motivación de este experimento radica en aprovechar la decisión de utilizar *visual servoing* basado en posición y no en imagen, pudiendo de esta forma posicionar objetos en un espacio tridimensional.

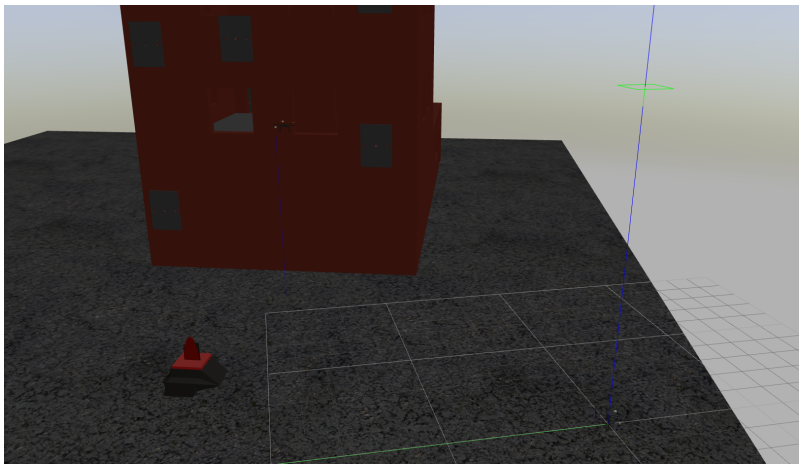


Figura 5.7 Simulación del experimento con dos UAVs.

Condiciones

El segundo UAV no despegará hasta el que el primero no haya aterrizado, no se contempla ninguna interacción física entre ambos. En el Apéndice B se muestra un esquema sobre la relación de dependencia entre ejes.

Resultados y conclusiones

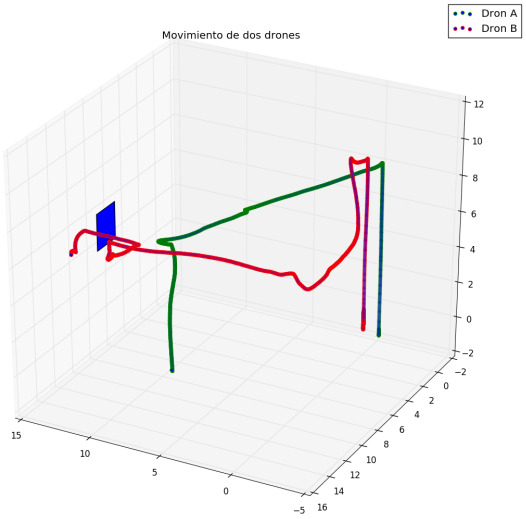


Figura 5.8 Simulación del experimento con dos UAVs.

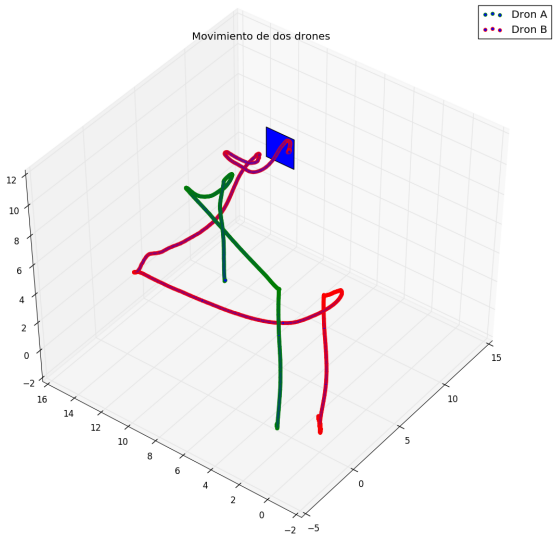


Figura 5.9 Simulación del experimento con dos UAVs.

En la gráfica se puede observa la ruta tomada por cada uno de los UAV y una representación de la imagen en el espacio.
Después de realizar varias repeticiones se puede afirmar que el sistema no es lo suficientemente robusto para esta situación, sería necesario reducir el error de posicionamiento a través de técnicas de mapeo o la inclusión de GPS.

Conclusiones y ampliaciones

Introducción

En este capítulo se exponen las conclusiones obtenidas durante el desarrollo del proyecto y se indicarán varías vías de posibles ampliaciones futuras. Se propondrá la inclusión de técnicas de mapeo para mejorar el posicionamiento sin necesidad del uso de técnicas GPS y la inclusión de algoritmos de rutas para aumentar la autonomía del sistema.

Conclusiones

En esta sección se detallarán las conclusiones extraídas durante el desarrollo del proyecto, se repasarán los hitos establecidos en el Capítulo 1 y el desempeño obtenido del método.

Desarrollo del proyecto

Durante el desarrollo del proyecto se tomaron varias decisiones como el uso de ROS o el uso de *visual servoing*, las cuales han resultado cruciales para concluir satisfactoriamente el objetivo del documento, definir el método de detección de ventanas y realizar su cruce mediante el uso de UAVs.
El uso de ROS ha facilitado la tarea que supone el desarrollo de los módulos que conforman el método y su librería *tf2* ha sido crucial para el posicionamiento de los distintos elementos internos del robot aéreo y del entorno que lo rodea.
Gracias al *visual servoing* se puede cumplir la misión de forma autónoma, sin la intervención de un piloto. Además, al determinar una implementación basada en posición no se a limitado el proyecto al cruce de las ventanas, obteniendo la capacidad de posicionarlas en un espacio tridimensional.

Repaso hitos del proyecto

Repasando los hitos definidos para el proyecto, se han cumplido todos satisfactoriamente, consiguiendo realizar un cruce de ventana y su detección mediante el uso de un robot aéreo de forma segura. Ademas se ha explorado la posibilidad de realizar la integración con varios robots aéreos.

Desempeño del método

Como se puede inferir del Capítulo 5 mediante la fusión del láser y la odometría se ha conseguido un sistema robusto bajo las condiciones de simulación. Durante los ensayos las incidencias de colisión se vieron reducidas a una porción marginal bajo los ensayos con un único sistema UAV mientras que, utilizando dos UAVs, será necesario la integración de nuevas técnicas de posicionamiento.

Ampliaciones y próximos trabajos

En esta sección se detallarán posibles ampliaciones al proyecto para mejorar algún aspecto concreto o añadir nuevas cualidades al mismo. De esta forma se abre la posibilidad de continuar dicho proyecto.

Ensayos con UAV reales

Queda fuera del alcance de este proyecto la realización de ensayos sobre un UAV real, pudiendo determinar mejor la robustez del algoritmo y contrastar los resultados expuestos en este documento.

Inclusión técnicas de localización y modelado simultáneos

Como se ha indicado reiteradas veces en el desarrollo de este documento la inclusión de técnicas de mapeo supondrían una gran mejora al proyecto, reduciendo el error en posición inherente a la odometría y por ende, reducir el error de detección de las ventanas. En concretó se expone la inclusión de SLAM (*simultaneous localizacion and mapping*), dicha técnica permite construir un mapa de un entorno desconocido para el robot, a la vez que estima una trayectoria al desplazarse en dicho entorno. Esta técnica supondría la solución también a la inclusión de un algoritmo para el cálculo de rutas [29] [28].
En la actualidad existen un gran número de formas de implementar SLAM en un proyecto, en concreto para este se expone la opción de implementar VSLAM (*Visual SLAM*) la cual consiste en el uso de una cámara para desarrollar el mapeado y localización simultáneos. Este algoritmo esta compuesto por 5 módulos diferentes: inicialización, localización, mapeado, re-localización y optimización del mapeado. Se presentan en los siguientes artículos citados distintos algoritmos para su implementación [19] [7] [21].

Apéndice A
Esquema de nodos ROS

En este apéndice se incluye un esquema de los nodos ROS que forman el sistema diseñado para este proyecto. Cabe destacar que gracias a esta estructura es muy sencillo reutilizar trabajo ya realizado o implementar módulos diseñados por la comunidad como es el caso la librería UAL en este proyecto.
Como se puede observar en las Figuras A.1 y A.2, son estructuras complejas con una gran cantidad de información compartida. ROS ofrece una arquitectura para poder llevar a cabo proyectos de gran envergadura por lo que su popularidad en la industria esta justificada.

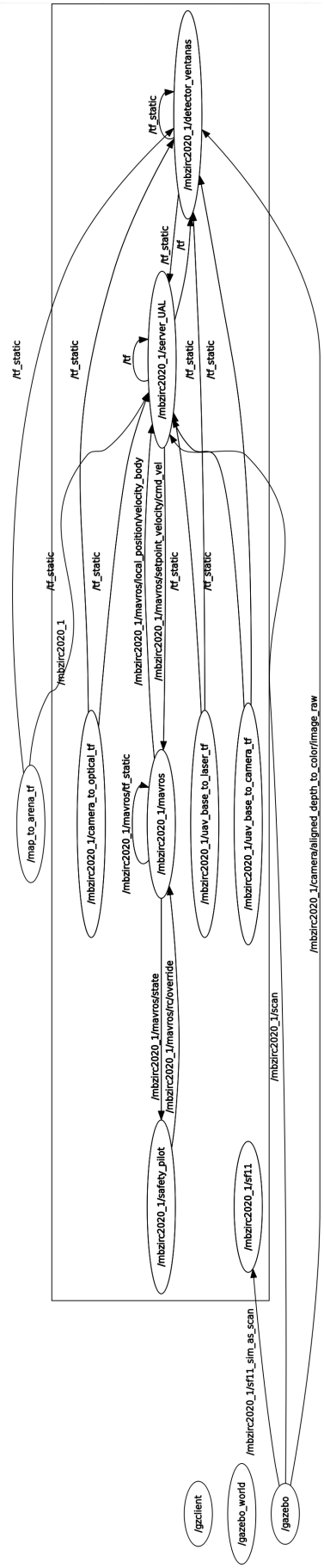


Figura A.1 Esquema de nodos ROS para un solo UAV.

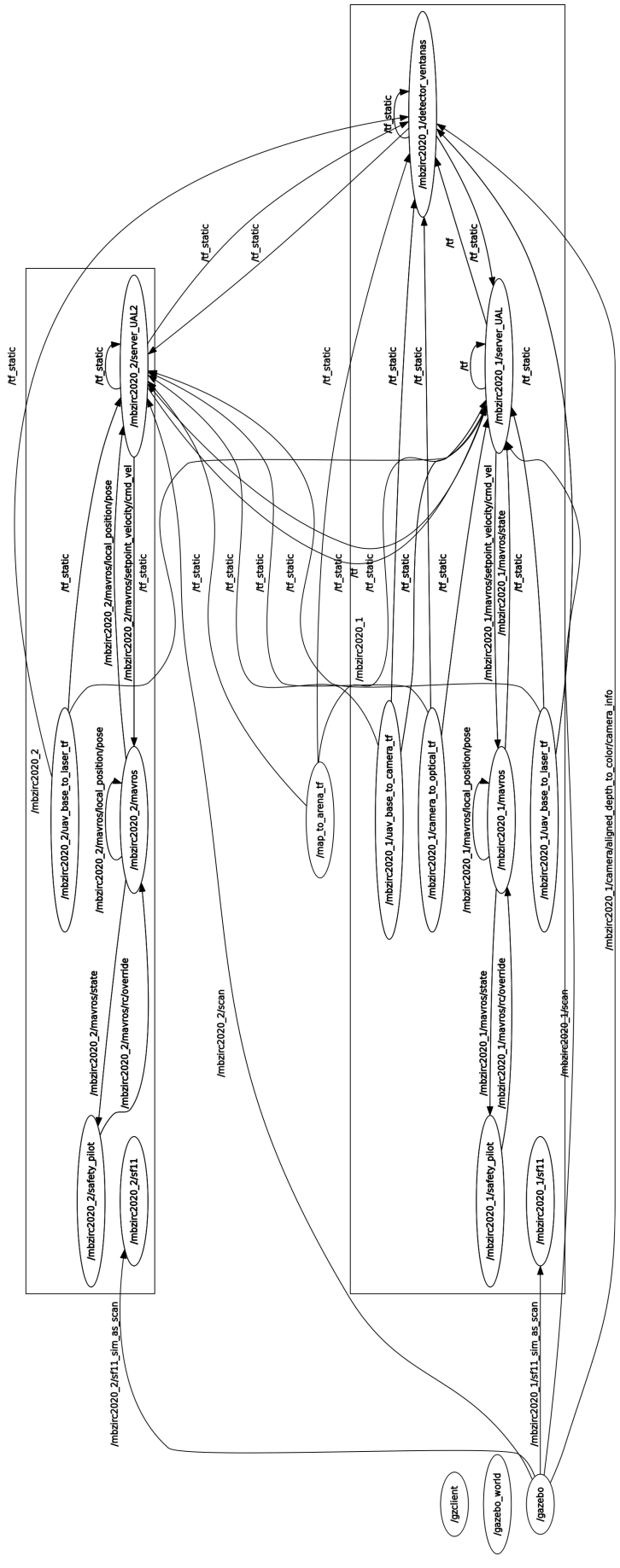


Figura A.2 Esquema de nodos ROS para dos UAVs.

Apéndice B

Ejes de coordenadas del sistema

En este último apéndice se quiere hacer mención a las relaciones de transformaciones homogéneas que se realizan en el proyecto. Como ya se indicó en el Capítulo 3 se ha hecho uso de la librería *tf2* de ROS para la implementación de la transformaciones y en este apéndice se mostrarán dos estructuras en árbol de ellas.

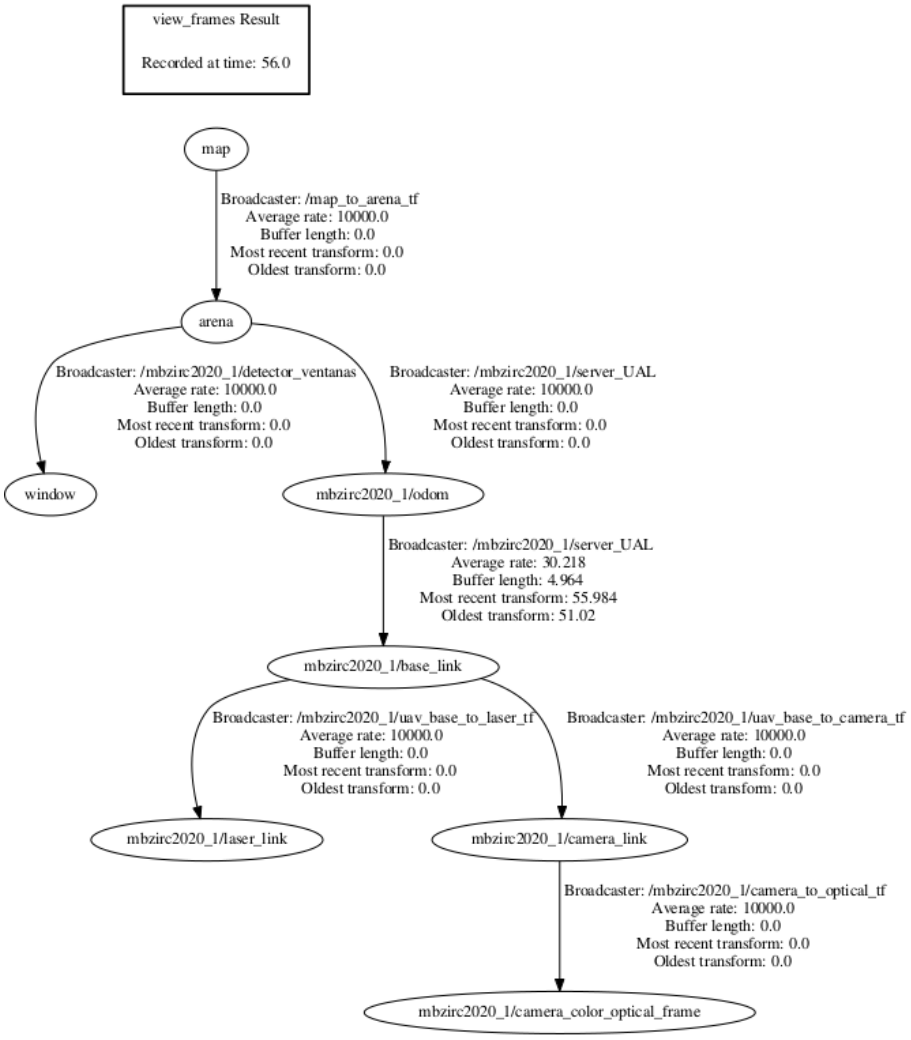


Figura B.1 Esquema en árbol de relación entre ejes de coordenadas para un UAV.

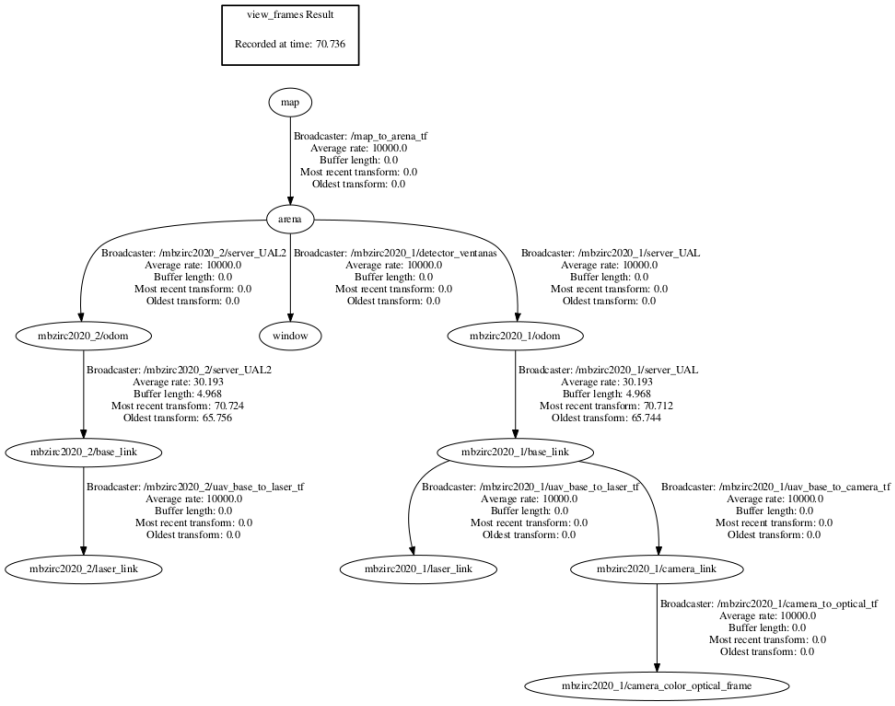


Figura B.2 Esquema en árbol de relación entre ejes de coordenadas para dos UAVs.

Como se puede observar en las figuras la relación de los ejes de la ventana a un sistema de referencia fijo no ha sido una decisión aleatoria, de esta forma mantenemos la información de la posición de las ventanas incluso tras perder el contacto con el sistema que la haya detectado.

Índice de Figuras

Índice de Tablas

Índice de algoritmos

Bibliografía

[1] *Depth map from stereo images*, https://docs.opencv.org/master/dd/d53/tutorial_py_depthmap.html, 2020.

[2] *Implementación fast*, https://docs.opencv.org/2.4.13.7/modules/features2d/doc/feature_detection_and_description.html#fast, 2020.

[3] *Implementación good feactures to track*, https://docs.opencv.org/2.4/modules/imgproc/doc/feature_detection.html#goodfeaturestotrack, 2020.

[4] Gary Bradski and Adrian Kaehler, *Learning opencv: Computer vision with the opencv library*, " O'Reilly Media, Inc.", 2008.

[5] R. Chen, H. Cao, H. Cheng, and J. Xie, *Study on urban emergency firefighting flying robots based on uav*, 2019 IEEE 4th Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), vol. 1, 2019, pp. 1890–1893.

[6] David Eigen, Christian Puhrsch, and Rob Fergus, *Depth map prediction from a single image using a multi-scale deep network*, 2014.

[7] Jakob Engel, Thomas Schöps, and Daniel Cremers, *Lsd-slam: Large-scale direct monocular slam*, European conference on computer vision, Springer, 2014, pp. 834–849.

[8] Colin Flanagan, *The bresenham line-drawing algorithm*, https://www.cs.helsinki.fi/group/goa/mallinnus/lines/bresenh.html, 1999.

[9] Tully Foote, *tf: The transform library*, Technologies for Practical Robot Applications (TePRA), 2013 IEEE International Conference on, Open-Source Software workshop, April 2013, pp. 1–6.

[10] GAZEBO, http://gazebosim.org/, 2020.

[11] S. N. A. M. Ghazali, H. A. Anuar, S. N. A. S. Zakaria, and Z. Yusoff, *Determining position of target subjects in maritime search and rescue (msar) operations using rotary wing unmanned aerial vehicles (uavs)*, 2016 International Conference on Information and Communication Technology (ICICTM), 2016, pp. 1–4.

[12] S. Han, W. Yu, H. Yang, and S. Wan, *An improved corner detection algorithm based on harris*, 2018 Chinese Automation Congress (CAC), 2018, pp. 1575–1580.

[13] C. Harris and M. Stephens, *A combined corner and edge detector*, Alvey Vision Conference, 1988.

[14] Jan Hosang, Rodrigo Benenson, and Bernt Schiele, *Learning non-maximum suppression*, 2017.

[15] S. Hutchinson, G. D. Hager, and P. I. Corke, *A tutorial on visual servo control*, IEEE Transactions on Robotics and Automation **12** (1996), no. 5, 651–670.

[16] Jianbo Shi and Tomasi, *Good features to track*, 1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition, 1994, pp. 593–600.

[17] N. Koenig and A. Howard, *Design and use paradigms for gazebo, an open-source multi-robot simulator*, 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566), vol. 3, 2004, pp. 2149–2154 vol.3.

[18] Luis Merino, Fernando Caballero, J Ramiro Martínez-De-Dios, Iván Maza, and Aníbal Ollero, *An unmanned aircraft system for automatic forest fire monitoring and measurement*, Journal of Intelligent & Robotic Systems **65** (2012), no. 1-4, 533–548.

[19] Raul Mur-Artal, J. M. M. Montiel, and Juan D. Tardos, *Orb-slam: A versatile and accurate monocular slam system*, IEEE Transactions on Robotics **31** (2015), no. 5, 1147–1163.

[20] OpenCV, *Camera calibration and 3d reconstruction*, *https://docs.opencv.org/2.4/modules/calib3d/doc/camera_calibration_and_3d_reconstruction.html*, 2020.

[21] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles, *S-ptam: Stereo parallel tracking and mapping*, Robotics and Autonomous Systems **93** (2017), 27–42.

[22] H. Qin, J. Q. Cui, J. Li, Y. Bi, M. Lan, M. Shan, W. Liu, K. Wang, F. Lin, Y. F. Zhang, and B. M. Chen, *Design and implementation of an unmanned aerial vehicle for autonomous firefighting missions*, 2016 12th IEEE International Conference on Control and Automation (ICCA), 2016, pp. 62–67.

[23] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Ng, *Ros: an open-source robot operating system*, vol. 3, 01 2009.

[24] Fran Real, Arturo Torres-González, Pablo Ramón Soria, Jesús Capitán, and Anibal Ollero, *Unmanned aerial vehicle abstraction layer: An abstraction layer to operate unmanned aerial vehicles*, International Journal of Advanced Robotic Systems **17** (2020), no. 4, 1–13.

[25] ROS, *https://www.ros.org/*, 2020.

[26] E. Rosten, R. Porter, and T. Drummond, *Faster and better: A machine learning approach to corner detection*, IEEE Transactions on Pattern Analysis and Machine Intelligence **32** (2010), no. 1, 105–119.

[27] Edward Rosten and Tom Drummond, *Machine learning for high-speed corner detection*, vol. 3951, 07 2006.

[28] Randall Smith, Matthew Self, and Peter Cheeseman, *Estimating uncertain spatial relationships in robotics*, 2013.

[29] Randall C. Smith and Peter Cheeseman, *On the representation and estimation of spatial uncertainty*, The International Journal of Robotics Research **5** (1986), no. 4, 56–68.

[30] C. Wang, P. Liu, T. Zhang, and J. Sun, *The adaptive vortex search algorithm of optimal path planning for forest fire rescue uav*, 2018 IEEE 3rd Advanced Information Technology, Electronic and Automation Control Conference (IAEAC), 2018, pp. 400–403.

Glosario

FAST Features from Accelerated Segment Test. 7, 8, 23, 30, 31, 45, 47

LIDAR Light Detection and Ranging. 26–28

MBZIRC Mohamed Bin Zayed International Robotics Challenge. 1, 28, 29

OpenCV Open computer vision. 17, 18, 22, 30, 45

ROS Robotic operative system. 17, 20–22, 45

SLAM Simultaneous localization and mapping. 27, 37

UAL Unmanned aerial vehicle Abstraction Layer. 18, 19, 26, 29, 45

UAV Unmanned Aerial Vehicle. VIII, X, 1, 2, 13, 18, 19, 21, 23, 25–30, 32, 34–37, 45, 46

UGV Unmanned Ground Vehicle. 1

vSLAM Visual simultaneous localization and mapping. 37