Universidad Zaragoza
1542

Trabajo Fin de Grado

Entorno de simulación para control visual de un cuadricóptero con cámara fisheye

Autor

Jorge Villa López

Director

Jesús Bermúdez Cameo

Ponente

José Jesús Guerrero Campo

EINA Universidad de Zaragoza

2014

Entorno de simulación para control visual de un cuadricóptero con cámara fisheye

RESUMEN

Hoy en día, el uso de sistemas de localización y orientación, como pueden ser el GPS o la IMU, están muy extendidos. Pero en lugares donde el GPS no funciona o su señal es muy débil, como por ejemplo, en el interior de un edificio, es necesario el uso de otros sistemas de navegación.

El objetivo de este TFG es el desarrollo de un entorno de simulación para control visual de cuadricópteros en interiores. El trabajo se centra en el uso de cámaras fisheye que ofrecen un campo de vista mucho más amplio que una cámara convencional.

En este simulador, un cuadricóptero (conocidos sus parámetros, modelo dinámico y sistema de control) es capaz de navegar por un escenario desconocido de interiores con la única ayuda de una cámara fisheye colocada en su parte inferior. El proceso es el siguiente: dada una imagen del entorno, se extraen las proyecciones de las rectas de la escena y, asumiendo la existencia de direcciones dominantes, se calculan los puntos de fuga mediante un algoritmo robusto basado en RANSAC. Con dicha información se obtiene la orientación del escenario respecto del cuadricóptero. Seguidamente, se calculan y asignan las consignas al cuadricóptero, de manera que sea capaz de avanzar de un modo coherente por el entorno. Este proceso se repite cada cierto tiempo, tomando imágenes y recalculando la orientación.

El trabajo se ha desarrollado utilizando Matlab, integrando POV-Ray para la generación y renderización de la escena 3D, y Simulink para describir el modelo dinámico y control del cuadricóptero.

En este trabajo se asume como premisa que el entorno de interior contiene direcciones dominantes, característica típica de entornos construidos por el hombre. En este caso, existe paralelismo entre grupos de rectas que comparten puntos de fuga. El cálculo de los puntos de fuga se realiza a partir de los planos de proyección de las rectas, que se describen mediante su vector normal, $\mathbf{n_i}$.

Como ya hemos dicho, una cámara fisheye tiene un campo de vista mucho mayor que una cámara perspectiva convencional, gracias al cual se puede tomar una longitud mayor de las rectas de la escena, hecho que favorece el cálculo de los planos de proyección. Aunque, por otra parte, es necesario utilizar un modelo de proyección no lineal y más complejo que el utilizado para cámaras convencionales.

ÍNDICE

1. Introducción

En los últimos años, los vehículos aéreos no tripulados, entre los que se encuentra el cuadricóptero, han adquirido gran fama y han ganado en autonomía. Este avance es debido a la gran variedad de campos de aplicación de estos pequeños vehículos: búsqueda y rescate, exploración, vigilancia, inspección, en agricultura, en tareas militares… [1] así como a sus ventajas: alta maniobrabilidad, robustez, bajo coste, disminución del riesgo de vidas humanas, etc. [2]

Pero, ¿por qué se ha escogido un cuadricóptero y no otro tipo de vehículo aéreo?

En comparación con vehículos aéreos de ala fija, como podría ser un avión, un cuadricóptero consigue una mayor maniobrabilidad y es capaz de realizar despegues y aterrizajes verticales, lo que lo hace óptimo para entornos de interiores y zonas poco amplias.

En comparación a los helicópteros convencionales con un gran rotor y un rotor de cola, el cuadricóptero es más sencillo de manejar, modelar y controlar; además, en estático, se puede mantener suspendido sin ningún tipo de rotación. Algo que no ocurre en el caso del helicóptero, que para mantenerse estático precisa de un giro continuo en el rotor de cola.

A pesar de todo, el modelo de un cuadricóptero es complejo, altamente inestable y no lineal, por lo que se precisan diversos sensores y controladores para llevar a cabo su control. La mayoría de las aproximaciones de control se basan en dos subsistemas conectados en cascada. Un primer subsistema está formado por el control de la orientación y otro, superpuesto, por el control de la posición [1] [3] [4].

El manejo de un cuadricóptero resulta una tarea relativamente sencilla cuando se considera que el entorno es conocido y cuando se suponen comportamientos ideales de la IMU (Inertial Measurement Unit) y del GPS, que nos proporcionarán los datos necesarios de orientación y posición. El principal inconveniente es que un cuadricóptero no siempre vuela en entornos conocidos y, además, podría hacerlo en lugares donde la señal de GPS sería nula o no tan precisa como se desearía. Por ello, el siguiente paso de la investigación consiste en implementar sistemas de navegación en los que no sea necesario conocer a priori el entorno y no se disponga de señal GPS.

Una vez justificada la elección del cuadricóptero como vehículo, se procede a la justificación de la elección de una fisheye como cámara escogida para la toma de imágenes.

Una cámara fisheye permite la adquisición de imágenes con un gran campo de vista: 360º en un eje y 180º, o incluso más, en el otro. De esta manera, se puede observar una mayor longitud de las rectas de la escena, lo que favorecerá al cálculo de los planos de proyección del entorno. Al igual que los vehículos aéreos no tripulados, en los últimos años las cámaras fisheye también han experimentado un auge destacable. La alternativa a estas cámaras, para conseguir imágenes

con un amplio campo de vista, es un sistema de varias cámaras comunes orientadas en distintas direcciones, una sola cámara amarrada a un brazo móvil o una cámara enfocando hacia un espejo curvado donde se reflejan las imágenes que se desean tomar. El principal inconveniente es que estas alternativas obligan a tener grandes precisiones de posicionamiento y orientación o inutilizan un determinado número de píxeles de la imagen.

En entornos construidos por el hombre suelen existir muchas líneas paralelas, y además, dispuestas en direcciones ortogonales. Estas líneas paralelas intersectan en el infinito, y ese punto de intersección se conoce como punto de fuga. En este trabajo se utilizan dichos puntos de fuga para calcular la orientación del cuadricóptero. La principal ventaja es que esos puntos no dependen de la posición del cuadricóptero, sino únicamente de la orientación [2]. Así, conociendo las tres direcciones principales ortogonales, podremos calcular la rotación del entorno respecto al cuadricóptero. En este caso el escenario será un pasillo que ha sido modelado con POV-Ray, un generador de imágenes en tres dimensiones (ver Anexo D).

1.1 Objetivos y contribuciones

El principal objetivo de este trabajo es implementar un sistema de navegación para un cuadricóptero cumpliendo las dos premisas que se han nombrado en el apartado anterior: no es necesario conocer el entorno a priori ni tampoco es necesaria la utilización de señal GPS.

De esta manera, se ha diseñado un entorno de simulación donde un cuadricóptero, tomando imágenes con una cámara fisheye y procesándolas, es capaz de obtener su orientación respecto del entorno, pudiendo así moverse a lo largo de un escenario sin tener que conocer su posición absoluta ni su entorno.

Para resolver el problema principal que nos atañe en este TFG ha sido necesaria la resolución de problemas más pequeños que se detallan a continuación: comprensión del modelo dinámico de un cuadricóptero, estudio y mejora del control original del cuadricóptero[1], reutilización del código de extracción de planos de proyección en sistemas fisheye [5], cálculo de los puntos de fuga y de la orientación del cuadricóptero respecto de su entorno, programación de un entorno visual en 3D y, finalmente, integración, calibración y ajuste de todos los sistemas.

[1] El control original del cuadricóptero es el realizado por Peter Corke en la toolbox de robótica [7]

1.2 Entorno de trabajo

Para llevar a cabo la ejecución de este trabajo ha sido necesaria la integración de varios entornos de trabajo y programación.

El software principal de trabajo ha sido Matlab, desde el cual, mediante diferentes módulos de interfaz, se ha interactuado con un generador de imágenes sintéticas y con Simulink.

Para el modelado del escenario y la generación de imágenes sintéticas se ha utilizado POV-Ray. Este software de trazado de rayos utiliza un lenguaje algorítmico para la descripción del escenario y la cámara. Esto permite la integración en otros sistemas en los que se pueden generar los ficheros de texto necesarios para configurar la posición y orientación de la cámara. Una vez completada la generación, se obtiene una imagen que corresponde con la captura que adquiere una cámara fisheye situada en la parte inferior de un cuadricóptero.

Esta imagen fisheye es procesada en Matlab, a fin de extraer proyecciones de rectas y sus correspondientes planos de proyección. A partir de éstos, se calculan los puntos de fuga que codifican la orientación del cuadricóptero, utilizada para generar las consignas de posición del cuadricóptero en el esquema de control.

Estas consignas se introducen en Simulink, donde se ejecuta el modelo dinámico y el control del cuadricóptero. De esta manera se actualiza su estado, es decir, se calcula la nueva posición y orientación del mismo. Dicho estado se introduce, nuevamente, en el generador de imágenes POV-Ray, cerrando así el bucle.

1.3 Estructura de contenidos

Para proporcionar una idea más clara del conjunto de este trabajo se muestra la estructura principal del mismo: en el capítulo 2 se presenta el modelo dinámico de un cuadricóptero y se introduce cómo se realiza el control del mismo, utilizando la Robotics Toolbox [7]. El capítulo 3 se centra en explicar cómo se calcula la orientación del entorno respecto del cuadricóptero a partir de las imágenes tomadas por la cámara fisheye. Se explica brevemente la parte de proyección de rectas en sistemas fisheye y detalladamente el cálculo de los puntos de fuga. En el capítulo 4 se expone cómo se calculan las consignas del cuadricóptero y cómo se ha conseguido realizar la integración de todos los sistemas. Los experimentos realizados se presentan en el capítulo 5. Y, finalmente, en el capítulo 6 se exponen las conclusiones obtenidas tras la realización de este trabajo, así como posibles futuras mejoras y aplicaciones. Se incluyen varios anexos, a los que se puede acudir para obtener información más completa y detallada de algunos de los aspectos ya nombrados.

2. Dinámica y control de un cuadricóptero

Los robots aéreos tienen dos principales diferencias respecto a la mayoría de robots terrestres. La primera es el número de grados de libertad: un cuadricóptero tiene 6 grados de libertad, tres para posición y tres para orientación, lo que dificulta notablemente su control. La segunda es su naturaleza inestable, eso hace que se deba gobernar por fuerzas y momentos y no por velocidades, de manera que es necesario utilizar un modelado dinámico y no solamente cinemático [3].

En el Anexo A se amplía el detalle de la nomenclatura que se utiliza de aquí en adelante.

2.1 Modelo dinámico

En la Figura 2.1 se muestra un sencillo esquema del cuadricóptero. El sistema de referencia {B}, de 'body' en inglés, pertenece al cuadricóptero y se halla en su centro de masas. Se puede observar que el sistema de referencia escogido es dextrógiro, con el eje z hacia abajo. Los rotores se suelen designar ya sea con números del 1 al 4 o bien con las direcciones cardinales N, E, S, O. Se representa el empuje de cada uno de los rotores, designado por la letra T, de 'thrust' en inglés. Los rotores 1 y 3 giran en sentido contrario a las agujas del reloj, y los rotores 2 y 4 en el sentido de las agujas del reloj. Esta es la razón por la que un cuadricóptero se mantiene en suspensión (sin rotaciones) si sus cuatro rotores giran a la misma velocidad, ya que los momentos producidos por los rotores 1 y 3 se anulan con los producidos por los rotores 2 y 4.

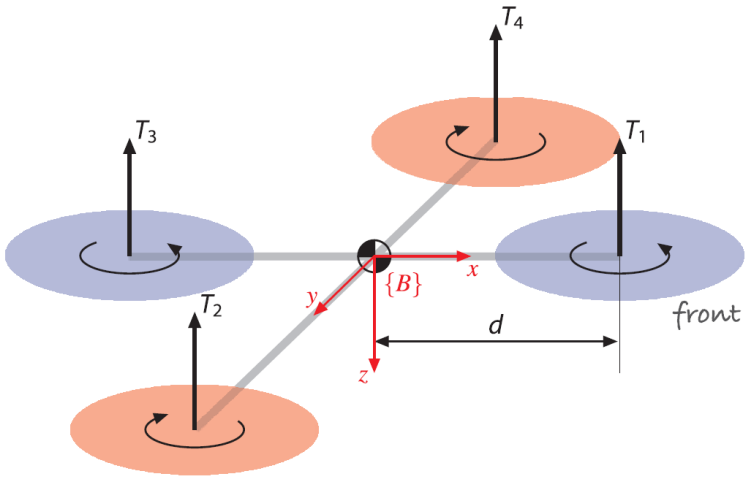

Figura 2.1. Esquema de un cuadricóptero. Fuente: [3]. Se muestra el sentido de giro de los rotores, el sentido del empuje de cada uno de éstos, la longitud del brazo del rotor y el sistema de referencia {B} del propio cuadricóptero.

A continuación se presentan las ecuaciones que determinan el modelo dinámico del cuadricóptero, considerándolo como un sólido rígido. Se calcula el empuje, dos tipos distintos de pares y las derivadas primeras y segundas de la posición y la orientación [3] [6] [7]:

$$\mathbf{T}_i = C_t \rho A r^2 \omega_i^2 \begin{pmatrix} -\sin a_{1_s i} \\ \cos a_{1_s i} \cdot \sin b_{1_s i} \\ \cos b_{1_s i} \cdot \cos a_{1_s i} \end{pmatrix} \tag{2.1}$$

$$\mathbf{Q}_i = -C_q \rho A r^3 \omega_i \left|\omega_i\right| e_3 \tag{2.2}$$

$$\boldsymbol{\tau}_i = \mathbf{T}_i \times \mathbf{D}_i \tag{2.3}$$

$$\dot{\mathbf{r}} = \mathbf{v} \tag{2.4}$$

$$m\dot{\mathbf{v}} = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} - {}^{W}\mathbf{R}_B \begin{pmatrix} 0 \\ 0 \\ \sum\limits_{i=1,2,3,4} T_i \end{pmatrix} \tag{2.5}$$

$$\dot{\mathbf{n}} = \mathbf{W}^{-1}\boldsymbol{\Omega} \tag{2.6}$$

$$\mathbf{I}\dot{\boldsymbol{\Omega}} = -\boldsymbol{\Omega} \times \mathbf{I}\boldsymbol{\Omega} + \sum_{i=1,2,3,4} (\boldsymbol{\tau}_i + \mathbf{Q}_i) \tag{2.7}$$

En (2.1) aparecen dos variables relacionadas con el flapping: $a_{1_s i}$ y $b_{1_s i}$. El flapping es un efecto producido por la traslación horizontal de los rotores en el aire. Se basa en que las hélices del cuadricóptero tienen un comportamiento diferente si se sitúan en la parte delantera o trasera del rotor (según la dirección de vuelo), produciendo una especie de aleteo. Aun así, este tema no se ha planteado como objeto de ese trabajo, así que de aquí en adelante se obviará en las explicaciones.

Cabe destacar también que $\mathbf{T}_i$ (sin tener en cuenta el flapping) siempre toma valores positivos en la ecuación (2.1); tal y como se ha definido el eje z del sistema de referencia {B} del cuadricóptero, observamos que realmente debería tomar valores negativos. En las siguientes ecuaciones observaremos que esta singularidad sí se tiene presente. En (2.2) ω_i se multiplica por su módulo para conservar el signo de rotación de cada rotor; el signo "–" es necesario para que el valor del momento producido resulte coherente con el signo de la velocidad de rotación de cada rotor.

Por definición, el par o momento de una fuerza respecto a un punto viene definido por el producto vectorial de un vector distancia por un vector fuerza $\boldsymbol{\tau} = \mathbf{r} \times \mathbf{F}$. En la ecuación (2.3) se realiza el producto vectorial de esta manera $\boldsymbol{\tau} = \mathbf{F} \times \mathbf{r}$; así se subsana la singularidad que se ha comentado anteriormente acerca del signo de $\mathbf{T}_i$.

Hay que remarcar que la velocidad y la aceleración lineal se expresan en el sistema de referencia absoluto {W}; por eso en (2.5) utilizamos ${}^{W}\mathbf{R}_{B}$, trasladando el empuje T del sistema de referencia {B} del cuadricóptero al sistema absoluto {W}. En dicha ecuación, el signo menos también se debe a la singularidad comentada de $\mathbf{T}_i$.

Para calcular la derivada de los ángulos YPR del cuadricóptero (2.6) es necesario utilizar el wronskiano inverso. Su obtención se detalla en el Anexo B. La aceleración angular $\dot{\boldsymbol{\Omega}}$ (2.7) está expresada en la referencia {B}.

2.2 Control

Para realizar el control del cuadricóptero se ha utilizado la toolbox para Matlab "Robotics, Vision and Control" de Peter Corke [7], en la que se utilizan dos subsistemas en cascada, uno para controlar la orientación, y otro superpuesto para controlar la posición. Esto se hace así porque, a fin de que el cuadricóptero avance según direcciones paralelas al suelo, es necesario variar su orientación.

La única manera posible de actuar sobre el cuadricóptero es variando la velocidad angular de sus rotores; así pues, deberemos conocer cómo se relacionan las fuerzas y pares producidos con las velocidades angulares:

$$T_i = C_t \rho A r^2 \omega_i^2 = c_T \omega_i^2 \tag{2.8}$$

La constante $c_T > 0$ puede ser determinada experimentalmente mediante test estáticos de empuje [4]. La ventaja es que de esta manera los términos del flapping están presentes implícitamente en dicha constante.

$$Q_i = -C_q \rho A r^3 \omega_i \left|\omega_i\right| e_3 = c_Q \omega_i^2 \tag{2.9}$$

La ecuación (2.9) hace referencia al par producido por el giro de los rotores.

$$\tau_i = T_i \times D_i = c_T \omega_i^2 D_i \tag{2.10}$$

La ecuación (2.10) representa el par producido por el empuje, T_i, de cada rotor.

En las tres ecuaciones anteriores falta determinar correctamente los signos pues, por ejemplo, un empuje T_1 positivo produce un τ_y positivo, pero un empuje T_3 positivo produce un τ_y negativo. Así, se muestran las tres ecuaciones de forma matricial con los signos correctamente asignados:

$$\begin{pmatrix} T_\Sigma \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} = \begin{pmatrix} c_T & c_T & c_T & c_T \\ 0 & dc_T & 0 & -dc_T \\ -dc_T & 0 & dc_T & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{pmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \mathbf{A} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} \tag{2.11}$$

El control del cuadricóptero se describe más detalladamente en el Anexo C.

3. Estimación visual de la orientación

El algoritmo utilizado para estimar la orientación se basa en la extracción de puntos de fuga a partir de imágenes tomadas por una cámara fisheye; por ello se parte de ciertas premisas, que se detallan a continuación:

1. En el entorno debe haber suficiente luminosidad para que la cámara fisheye pueda tomar imágenes de calidad aceptable.
2. El entorno debe tener suficientes rectas paralelas en direcciones ortogonales para poder extraer correctamente los puntos de fuga; esto se conoce como escenario de Manhattan.
3. Cuantas más rectas paralelas en direcciones ortogonales existan en el escenario, mayor será la precisión y robustez del sistema.
4. El cuadricóptero debe tener una altura suficiente respecto al suelo, de manera que la cámara fisheye pueda obtener imágenes lo suficientemente amplias para la obtención de los planos de proyección.
5. Asumimos que el cuadricóptero se halla en una situación ambiental de calma, sin fuertes vientos ni otras fuerzas externas; de lo contrario, no bastaría con el control realizado.

3.1 Proyección de rectas en sistemas fisheye

La extracción de la proyección de rectas para cámaras perspectivas convencionales es una tarea relativamente sencilla si se compara con la extracción para cámaras fisheye. En el primer caso, se utiliza un modelo matemático lineal, pues cualquier recta 3D se proyecta sobre el plano de la imagen, resultando una línea 2D. Cuando el sistema de proyección no es perspectivo, como el caso de los sistemas fisheye, el modelo matemático deja de ser lineal y la línea proyectada es una curva.

Para la ejecución de este trabajo se ha utilizado un código de extracción de planos de proyección [5] desarrollado por Jesús Bermúdez. Este mismo autor y otros en [8] [9] describen el modelo matemático utilizado y el funcionamiento del código. Se puede resumir de la siguiente manera: partiendo de un conjunto de al menos dos puntos que pertenezcan a una misma línea de la imagen, se puede construir un sistema lineal homogéneo; la solución a este sistema es un plano que contiene dicha recta 3D. Para detectar los puntos a partir de los cuales se determina la recta,

primeramente se detectan los contornos de la imagen con el algoritmo Canny[2], y se almacenan en componentes conectadas. Para cada componente se lanza un algoritmo RANSAC (ver 3.2). Tras este algoritmo de votación, se determinan cuáles son los planos de proyección, con los que será posible el cálculo de los puntos de fuga del entorno.

3.2 Estimación de la orientación basada en puntos de fuga

Los planos de proyección, nombrados en el apartado anterior, se describen mediante su vector normal. Estos planos sólo contienen dos de los cuatro grados de libertad de cada recta. Sin embargo, si se impone paralelismo entre dos o más rectas, se puede calcular la dirección común de éstas. Asumiendo que el entorno contiene direcciones dominantes, y que por lo tanto existe este paralelismo, se puede plantear un algoritmo robusto para poder calcular los puntos de fuga, y así la orientación del cuadricóptero respecto a su entorno.

Para comprender mejor cómo se han calculado los puntos de fuga o direcciones principales nos centraremos en un caso análogo pero mucho más sencillo. Supongamos un prisma como el que se representa a continuación; en éste se desea obtener la dirección principal 'x':

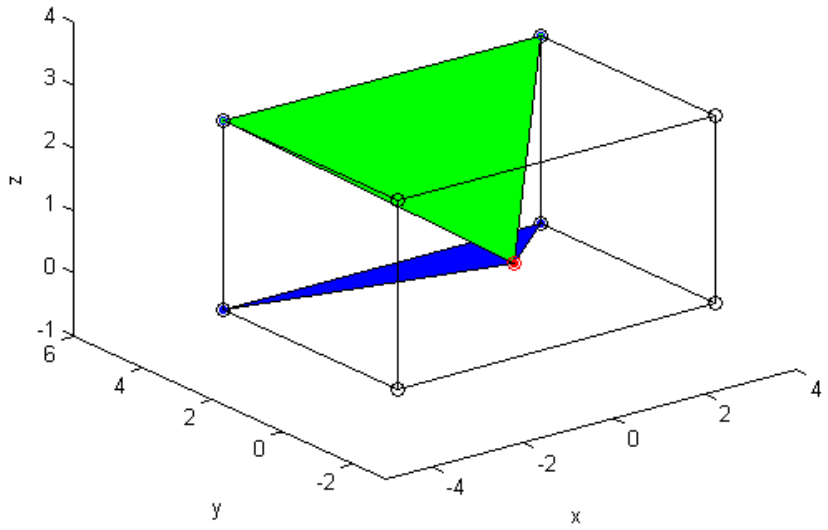

Figura 3.1. Cálculo de los puntos de fuga a partir de las normales de los planos de proyección (ejemplo simplificado).

[2] También se utiliza para extraer el gradiente de cada píxel de la imagen y proporcionar mayor robustez al algoritmo RANSAC.

El origen respecto al que se definen los planos de proyección se ha dibujado con un punto de color rojo. Se pueden observar dos planos, uno azul y otro verde. Éstos son los planos de proyección de dos rectas paralelas, orientadas en dirección 'x' y que, por tanto, constituyen una dirección principal. Al realizar el producto vectorial de las normales 'n' de ambos planos, obtenemos un vector 'v' con la misma dirección que las dos rectas paralelas anteriores.

En un entorno donde existen tres direcciones principales ortogonales, este mismo proceso se puede aplicar a las tres direcciones. De esta manera, se obtienen tres vectores, con las tres direcciones principales $\mathbf{v_1}$, $\mathbf{v_2}$ y $\mathbf{v_3}$.

Esta condición de paralelismo en las rectas del entorno es una simplificación del problema general. Las siguientes razones justifican la integración del modelo geométrico propuesto en un método de extracción robusto:

- El entorno donde se mueve el cuadricóptero puede tener rectas que no pertenecen a ninguna de las tres direcciones principales. Así, estas rectas hay que discriminarlas para que no induzcan errores en el cálculo de la orientación.
- Se pueden encontrar dos planos de proyección paralelos o casi paralelos.
  Dos planos paralelos poseen dos vectores normales $\mathbf{n_1}$ y $\mathbf{n_2}$ también paralelos; su producto vectorial da como resultado un vector de valores nulos. Esto dará problemas a la hora de realizar el cálculo; más adelante se explicará por qué.
- No se conocerá, a priori, qué planos son los que definen la misma dirección principal, de manera que hay que diseñar algún método iterativo que compare planos de proyección hasta obtener una solución correcta.

El método que se usa es el RANSAC, abreviación de "RANdom SAmple Consensus". Se trata de un método iterativo para estimar parámetros de un modelo matemático partiendo de un conjunto de datos que puede tener una cantidad razonable de valores atípicos o espurios.

El ajuste por mínimos cuadrados tiene una clara desventaja: cuando se tiene una considerable cantidad de datos atípicos, se realiza un mal ajuste, pues todos esos datos influyen en el modelo. La ventaja de RANSAC en comparación al ajuste por mínimos cuadrados es que discrimina los valores atípicos, de manera que únicamente realiza el ajuste con valores que se aproximan al modelo. En este trabajo, el algoritmo se aplica al conjunto de rectas normales '**n**', pero para una mejor comprensión se explicará el caso de RANSAC aplicado a puntos.

En la Figura 3.2 pueden observarse una serie de puntos. Los puntos azules se conocen con el nombre de 'inliers' y los puntos rojos como 'outliers'. El proceso iterativo es el siguiente:

1. Se toma un subconjunto de puntos aleatoriamente.
2. Se ajusta un modelo para el subconjunto seleccionado.
3. Se determina y se almacena el número de puntos que se ajustan a dicho modelo.

4. Se repite el proceso un número determinado de veces.
5. Se escoge el modelo al que mayor número de puntos se han ajustado.

De esta manera, discriminando los puntos que están más allá de una determinada distancia, conseguimos que el ajuste sea más preciso que con el método de los mínimos cuadrados.

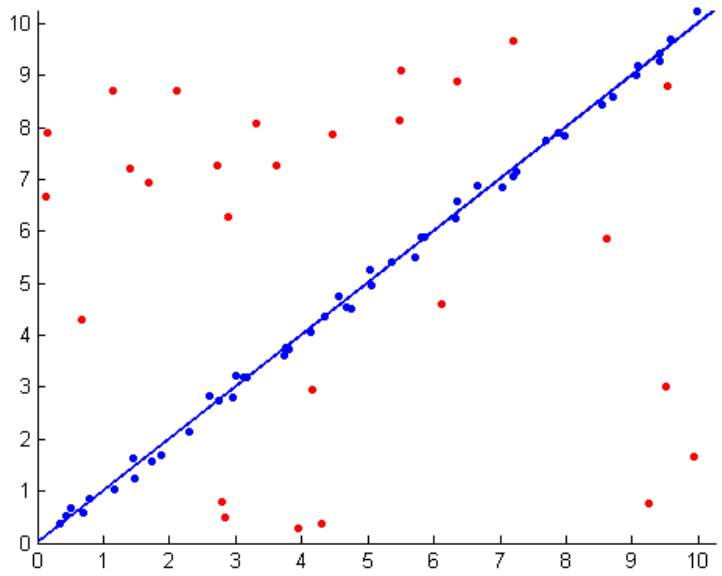

Figura 3.2. Método RANSAC.

La forma de resolver el problema con rectas, en vez de con puntos, es similar. Se parte de un conjunto de vectores, $\mathbf{n}_i$, normales a los planos de proyección.

1. Se escoge, aleatoriamente, una pareja de vectores $\mathbf{n}_1$ y $\mathbf{n}_2$.
2. Se realiza el producto vectorial de ambos, obteniendo el vector **v**, perpendicular a $\mathbf{n}_1$ y $\mathbf{n}_2$; esta dirección **v** es la que interesa obtener, que es la dirección principal.

$$\mathbf{v} = \mathbf{n}_1 \times \mathbf{n}_2 \tag{3.1}$$

   Si el producto vectorial da un vector de valores cercanos a 0, se desecha el resultado de esta iteración. Esto significa que $\mathbf{n}_1$ y $\mathbf{n}_2$ son paralelos o casi paralelos, de manera que sus respectivos planos son también paralelos.
3. Se realiza el producto escalar de cada vector $\mathbf{n}_i$ con **v**:

$$dist_i = \mathbf{n}_i \cdot \mathbf{v} \tag{3.2}$$

4. Si la variable $dist_i$ es menor que un valor umbral[3], ese vector $\mathbf{n_i}$ vota positivo a dicho vector $\mathbf{v}$. Se almacena qué vectores concretos $\mathbf{n_i}$ han votado positivo. Si $dist_i$ es mayor, no se vota.
5. Se repiten los puntos del 1 al 4 un número suficiente de veces. El número mínimo de veces, k, que hay que repetir el algoritmo viene determinado por la siguiente expresión [10]:

$$k = \frac{\log(1-p)}{\log(1-\omega^s)} \tag{3.3}$$

   Donde p es la probabilidad de que el algoritmo seleccione, en alguna iteración, sólo inliers de entre todos los datos de entrada cuando se seleccionan los s puntos (s=2) con los cuales el modelo puede ser estimado. ω es la probabilidad de que un vector dentro del conjunto de datos sea un inlier.
6. Se determina qué vector $\mathbf{v}$ ha obtenido mayor número de votos.
7. Se hace un ajuste por mínimos cuadrados con todos los vectores que han votado positivo al vector $\mathbf{v}$ que ha ganado en la votación. Por eso era necesario saber qué vectores concretos votaban a cada vector **v**. Recordamos que este ajuste por mínimos cuadrados sólo se hace con los vectores que han votado positivo, es decir, ya se han excluido los vectores que no se ajustaban al modelo.
8. Se normaliza el vector resultado del ajuste por mínimos cuadrados $\mathbf{v_{final}}$. Ésta será una dirección principal.
9. Se desechan los vectores $\mathbf{n_i}$ que han votado positivo. Se repite todo el proceso con los $\mathbf{n_i}$ restantes, para sacar la segunda y la tercera dirección principal. A continuación, se muestran dos figuras:

[3] Por esta razón se desechan las iteraciones cuando $\mathbf{n_1} \| \mathbf{n_2}$, pues en tal caso $\mathbf{v} = \mathbf{n_1} \times \mathbf{n_2} \approx (0 \quad 0 \quad 0)$, y entonces, para cualquier vector $\mathbf{n_i}$ se cumpliría: $dist_i = \mathbf{n_i} \cdot \mathbf{v} \approx 0$. Así, todos los vectores $\mathbf{n_i}$ votarían positivo al vector **v**, produciendo un resultado erróneo.

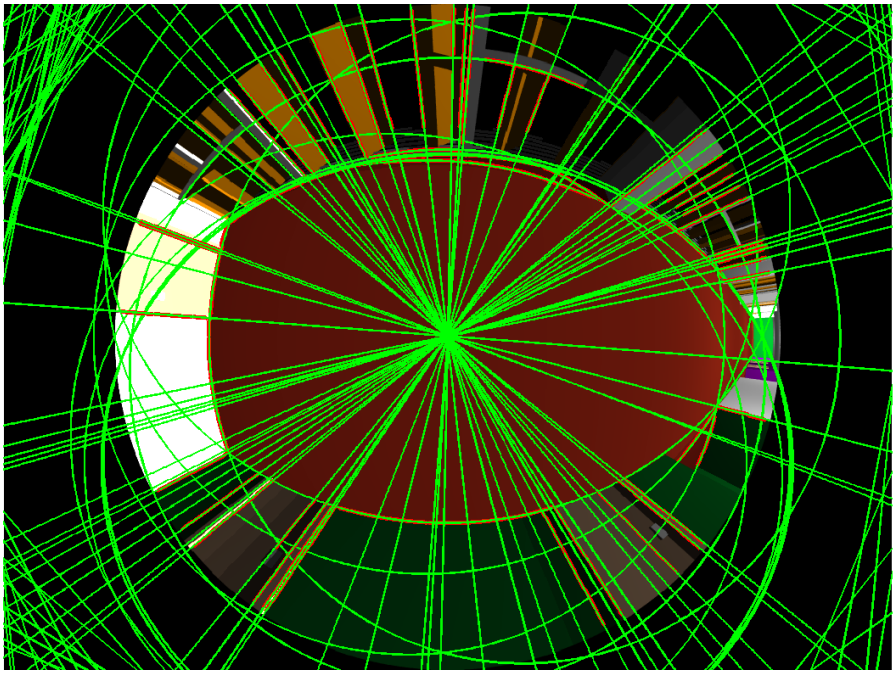

Figura 3.3. Extracción de proyecciones de rectas en una imagen sintética tomada por una cámara fisheye.

En la Figura 3.3 se puede observar el conjunto de planos de proyección que se han obtenido para una posición y orientación concreta.

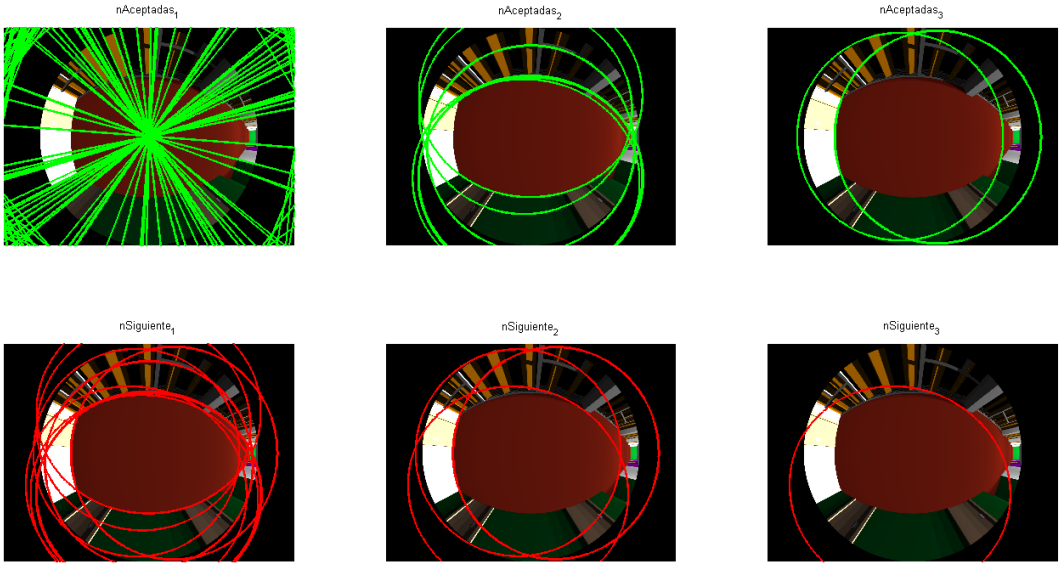

Figura 3.4. Clasificación de los planos de proyección según direcciones dominantes.

En la Figura 3.4, se muestran en color verde, las normales que se han usado para calcular una determinada dirección principal, y debajo, en rojo, las que no se han utilizado y que,

por tanto, se utilizarán para calcular la siguiente dirección. Finalmente, en la sexta figura (abajo a la derecha) se dibujan las que se desechan. En este caso sólo se desecha una.

10. Lógicamente, las tres direcciones principales no tienen correctamente definido el sentido y será necesario ordenarlas para formar la matriz de rotación del cuadricóptero. Así que habrá que seleccionarlas, darles el sentido adecuado y ordenarlas correctamente en la matriz de rotación.

Con el objetivo de obtener la matriz de rotación del cuadricóptero a partir de las 3 direcciones principales calculadas $\mathbf{v_1}$, $\mathbf{v_2}$ y $\mathbf{v_3}$ (se recuerda que con sentido aleatorio y sin ordenar), se ha creado una función a la que se le introducen estos tres vectores y la matriz de rotación calculada en la iteración justamente anterior[4]; se comparan, y se escogen los dos vectores que más se asemejan a la situación anterior; se les establece el sentido correcto, y se disponen de forma ordenada en la matriz de rotación. El tercer vector o dirección principal se calcula como el producto vectorial de los dos anteriores, cerciorándonos de que el sistema obtenido es dextrógiro.

[4] Como excepción, para la primera iteración de la simulación, dicha comparación se realiza con la matriz de rotación obtenida en Simulink.

4. Integración de sistemas para la simulación del control visual

En la Figura 4.1 se muestra un diagrama de bloques, donde se pueden observar los principales módulos que integran el sistema de simulación y con qué programa se ha implementado cada uno:

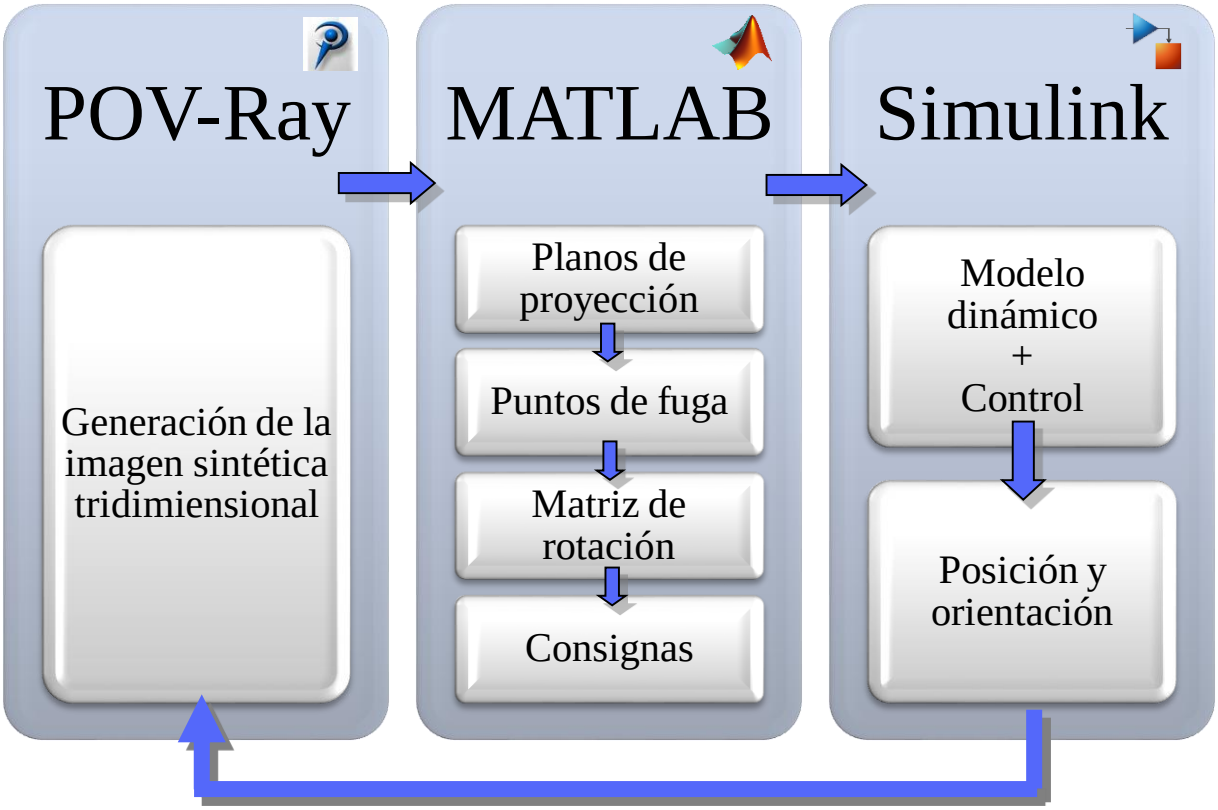

Figura 4.1. Entorno de trabajo. Se muestran los diferentes módulos y el programa con el que se ha implementado cada uno.

En el capítulo 3.2 se ha explicado cómo se calcula la matriz de rotación del pasillo respecto al cuadricóptero. Una vez conocida dicha información, se puede generar la consigna. Debemos señalar que el sistema de control del cuadricóptero se ha diseñado para que las consignas estén expresadas en el sistema de referencia absoluto. Por ello, son necesarias cuatro matrices de rotación que relacionen los sistemas de referencia absoluto, cuadricóptero, cámara y pasillo:

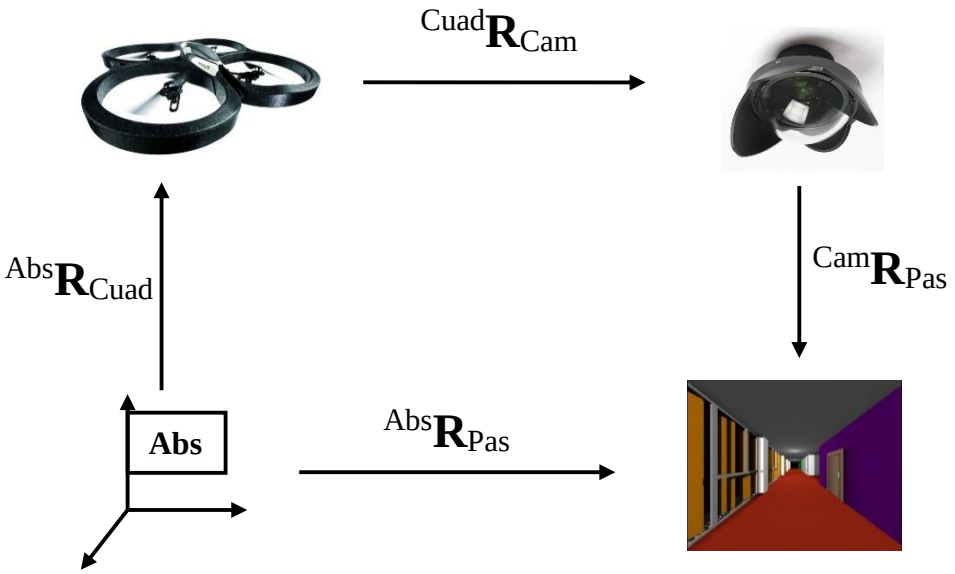

Figura 4.2. Sistemas de referencia y transformaciones necesarias para el cálculo de la consigna.

La matriz de rotación ${}^{Abs}\mathbf{R}_{Cuad}$ se determina por la orientación del cuadricóptero en la simulación en Simulink; la matriz ${}^{Cuad}\mathbf{R}_{Cam}$ relaciona la orientación de la cámara respecto al cuadricóptero, simplemente es una rotación de 180º en el eje 'x'; La matriz ${}^{Cam}\mathbf{R}_{Pas}$ se ha calculado con los planos de proyección y puntos de fuga. Por tanto, la matriz de rotación que no se conoce, ${}^{Abs}\mathbf{R}_{Pas}$, se puede calcular como [11]:

$$ {}^{Abs}\mathbf{R}_{Pas} = {}^{Abs}\mathbf{R}_{Cuad} \cdot {}^{Cuad}\mathbf{R}_{Cam} \cdot {}^{Cam}\mathbf{R}_{Pas} \tag{4.1} $$

Así se consigue relacionar la orientación del pasillo con la referencia absoluta, y posteriormente calcular las consignas del cuadricóptero en el sistema de referencia absoluto. Para que el cuadricóptero avance, por ejemplo, en el eje 'x' del pasillo, es necesario realizar la siguiente operación:

$$ \begin{pmatrix} x^* \\ y^* \\ z^* \end{pmatrix} = {}^{Abs}\mathbf{R}_{pas} \cdot \begin{pmatrix} 1 \\ 0 \\ 0 \end{pmatrix} \tag{4.2} $$

Cada vez que se lanza el generador de imágenes y se calcula la orientación del pasillo respecto al cuadricóptero, se le ordena al cuadricóptero avanzar un metro en dirección x del pasillo. Si quisiéramos que el cuadricóptero avanzara más rápido, simplemente deberíamos modificar el vector de la ecuación (4.2), cambiando el 1 por un número mayor. De la misma manera, si quisiéramos que el cuadricóptero avanzara en la dirección 'y' del pasillo, bastaría con asignar

un valor distinto de cero en la segunda posición de dicho vector, siendo el resto de valores nulos.

Cabe destacar que la consigna z^* se fija en 0. Como ya se ha explicado, la única forma que tiene el cuadricóptero de avanzar (en el eje 'x', por ejemplo) es realizar una rotación (en el eje 'y', ángulo pitch). Si la cámara fisheye tomara una imagen en ese instante, se generaría una matriz de rotación semejante a la que se obtendría al avanzar por un pasillo con una determinada pendiente. Al multiplicar dicha matriz de rotación por el vector $\begin{pmatrix}1 & 0 & 0\end{pmatrix}^T$, se obtendría una consigna z^* no nula, afectando pues a la altura del cuadricóptero sobre el suelo y poniendo en riesgo la correcta extracción de los planos de proyección.

Los valores de las consignas x^*, y^* y z^* se añaden de manera acumulativa a la consigna total del cuadricóptero (ver Figura 4.3).

En el esquema de control mejorado (ver Figura 4.3) se ha añadido un bloque de ruido con el fin de hacer el esquema de control algo más realista.

Otro nuevo bloque que aparece en el esquema de control mejorado (Figura 4.3) y que no aparecía en el esquema de control original (Figura C.1), es el bloque llamado "función completa". Aquí se ejecuta todo el código comprendido desde la creación de la imagen sintética en POV-Ray a partir del estado del cuadricóptero hasta el cálculo de las consignas, pasando por la extracción de los planos de proyección, los puntos de fuga y la matriz de rotación del vehículo. Este bloque se ejecuta continuamente, pero internamente se ha programado que la visión sólo se lance cada cierto tiempo, pues consume bastante potencia de cálculo. Por tanto, los valores de las consignas x^*, y^*, z^* sólo cambian cuando se lanza el algoritmo de visión.

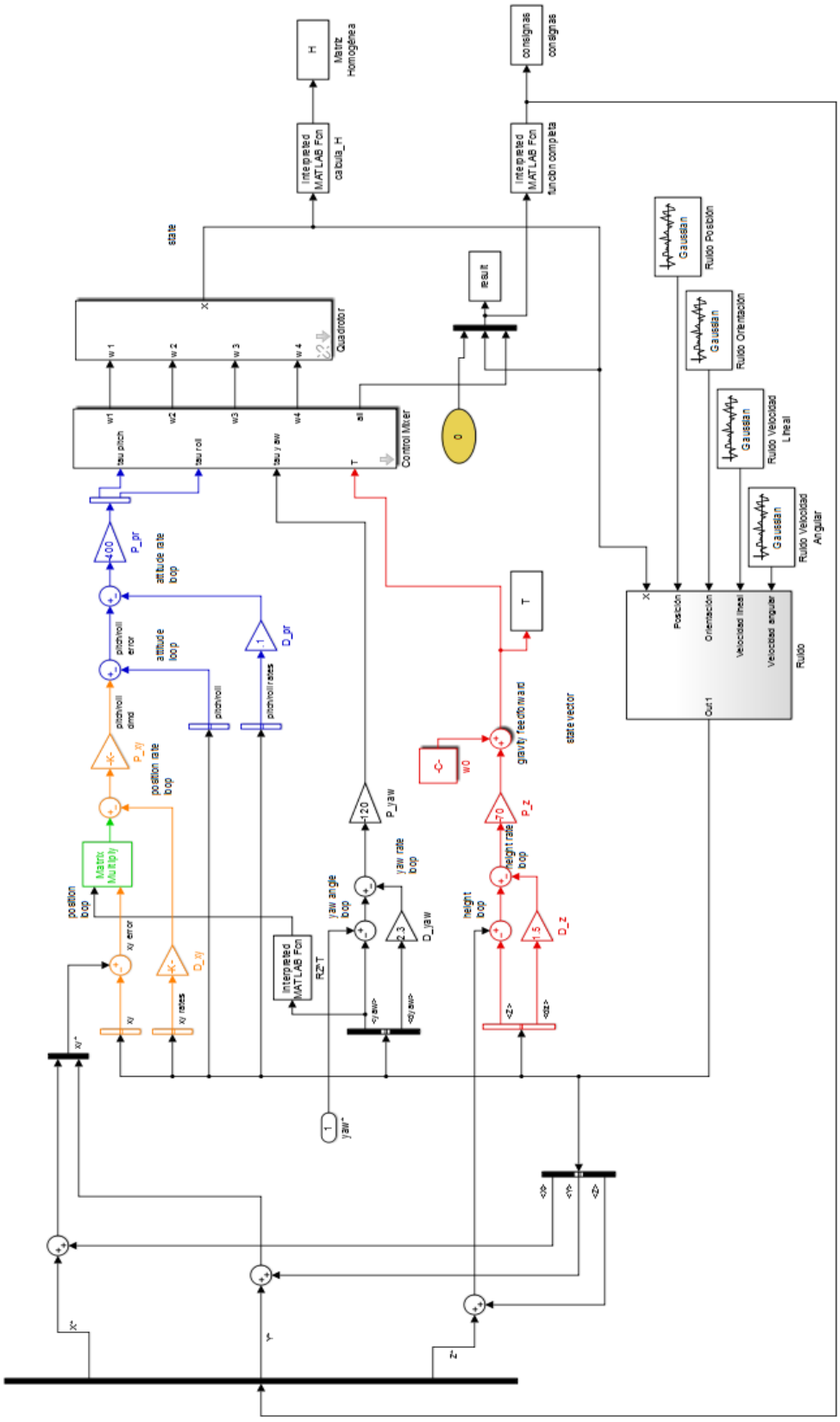

Figura 4.3. Esquema de control mejorado. Se ha mejorado el tiempo de respuesta y la sobreoscilación, se ha añadido un bloque para introducir ruido de medida y se han añadido otras funciones necesarias.

5. Ajustes y experimentos

Principalmente, se han realizado tres experimentos distintos. Uno para mejorar el sistema de control; otro para definir qué umbrales son los más correctos en la diferenciación de los puntos de fuga, de manera que se seleccionen los planos adecuados para cada punto de fuga; y otro, el experimento final, en el que se puede observar cómo el cuadricóptero navega a través de dos pasillos unidos por un giro en ángulo recto. A continuación, se detalla más detenidamente cada uno de estos experimentos.

5.1 Mejora del sistema de control

El esquema de control original obtenido de la toolbox de robótica de Peter Corke [7] se basa en controladores PD. Tal y como estaban definidas las constantes, se obtiene un sistema con cierta sobreoscilación (en algunos casos casi del 40%) y con un tiempo de respuesta mejorable. Por eso se ha intentado mejorar el control variando el valor de las constantes proporcionales del esquema de control. Se han realizado varias simulaciones para distintos valores, y se han determinado como óptimos aquéllos que producen una mayor reducción en los tiempos de respuesta y en la sobreoscilación.

Para analizar el tiempo de respuesta y la sobreoscilación, se han impuesto consignas de 1 metro para ‘x’, ‘y’ y ‘z’ individualmente, y de 1 rad para la rotación en el eje ‘z’, ‘yaw’.

Es deducible que un menor tiempo de respuesta deriva en una mayor acción sobre los motores de los rotores, pero como se explica en el Anexo C, la velocidad angular de los rotores está limitada a 1000 rad/s, por tanto, no supone un problema. En la Figura 5.1 se muestran las gráficas donde se puede observar la mejora conseguida:

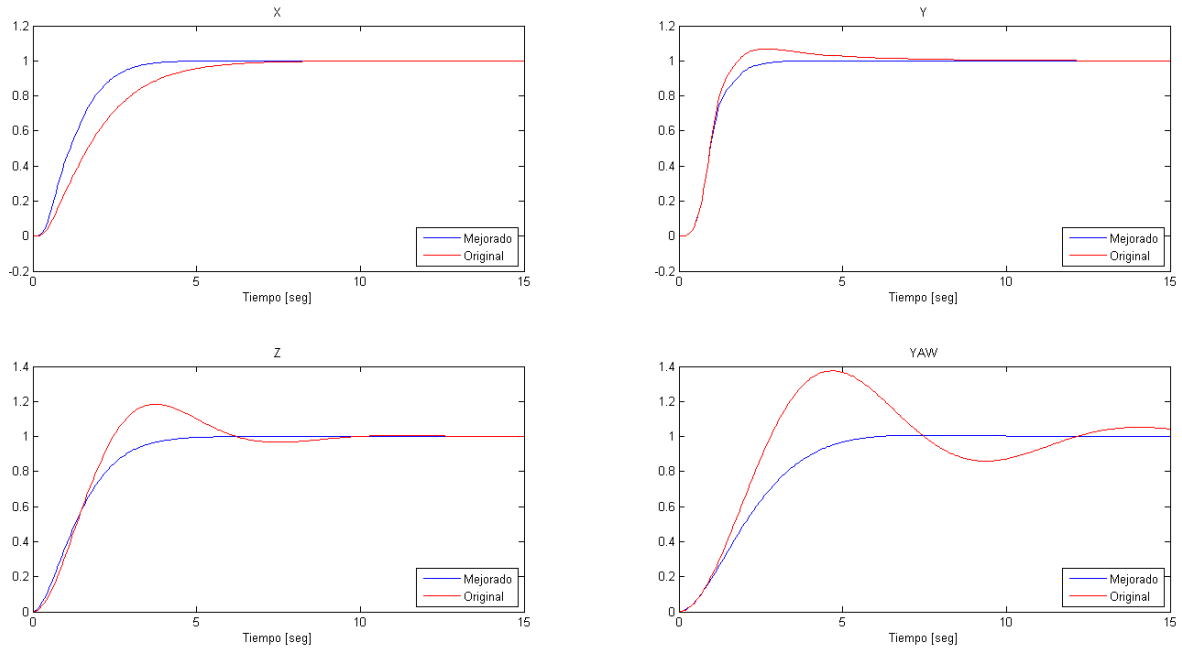

Figura 5.1. Comparación cualitativa entre el sistema de control original y el mejorado. Se muestran en color rojo las respuestas al control original y en color azul las respuestas al control mejorado.

En la Tabla 5.1, se exponen cuantitativamente las mejoras realizadas:

	SO (original)	**SO (mejorado)**	**t_r (original) [seg]**	**t_r (mejorado) [seg]**
X	0 %	0.01 %	4.85	2.9
Y	6.71 %	0.07 %	3.6	2.1
Z	18.36 %	0.01 %	5.55	3.45
yaw, ϕ	37.45 %	0.81 %	14.5	4.7

Tabla 5.1. Comparación cuantitativa entre el sistema de control original y el mejorado

5.2 Calibración del algoritmo RANSAC

A la hora de implementar el algoritmo RANSAC, es necesario definir un umbral. Este umbral será uno de los principales responsables de que funcione bien la extracción de las direcciones principales.

En el caso que nos atañe, el umbral definido hace referencia a la diferencia de orientación entre las distintas rectas de proyección. Por tanto, un plano de proyección votará positivo a aquellas rectas con las que tenga una determinada afinidad (cuando la diferencia de orientación de la dirección principal que representa sea menor que el umbral definido), y no votará a aquéllas en las que la diferencia de orientación sea superior a dicho umbral.

Tras varias pruebas, el umbral más adecuado se ha fijado entre ±0.1 y ±0.22; se trata de un umbral bastante pequeño, lo que resalta la precisión que vamos a obtener en la orientación del cuadricóptero.

La correcta elección de dicho umbral es de suma importancia, pues en caso de escoger un umbral demasiado grande, existirán planos de proyección que votarán positivo a una determinada dirección principal y que, realmente, no tendrán afinidad con la misma. De esta manera, se estarán incluyendo en el cálculo datos espurios (justamente lo que queríamos evitar aplicando el algoritmo RANSAC). Por el contrario, si el umbral es demasiado pequeño, se estará escogiendo un número demasiado pequeño de planos de proyección, desechando un número desmesurado e inadecuado de los mismos y, en consecuencia, se tendrán muy pocos datos para calcular la dirección principal. A continuación se muestran dos imágenes de la aplicación del algoritmo RANSAC para la misma imagen que aparece en la Figura 3.3, primero para un umbral superior y luego para uno inferior al escogido como más adecuado.

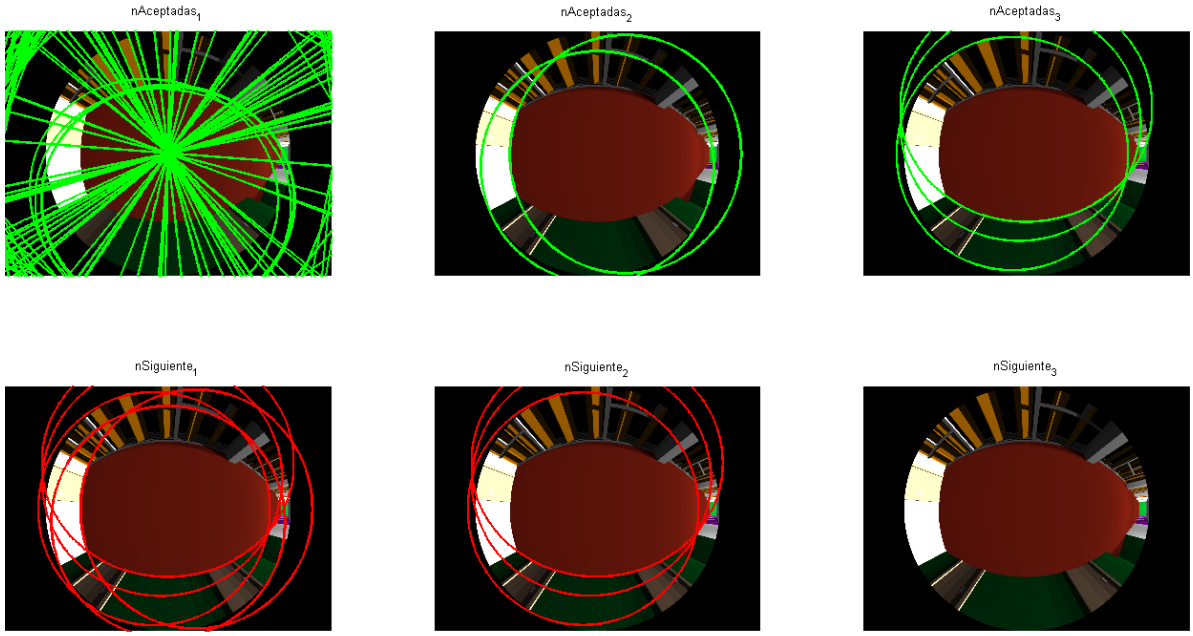

Figura 5.2. Clasificación de los planos de proyección con un umbral superior al adecuado. Se puede observar cómo en la primera imagen (arriba a la izquierda) se escogen planos de proyección que no corresponden a la dirección perpendicular al suelo. Umbral = ±0.6.

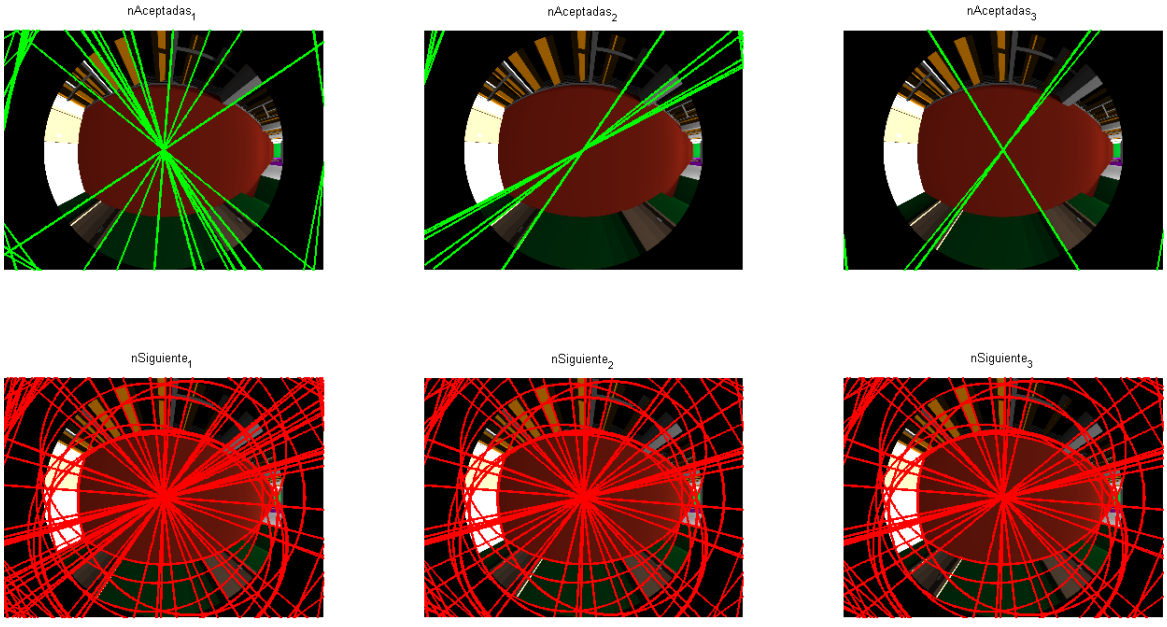

Figura 5.3. Clasificación de los planos de proyección con un umbral inferior al adecuado. Se puede observar cómo se realiza una deficiente clasificación y se desecha, erróneamente, un gran número de planos de proyección. Umbral = ±0.0005.

5.3 Simulación final

Se recuerda que el objetivo principal de este trabajo es la creación de un entorno de simulación, donde un cuadricóptero, gobernado por un sistema de control que simule su comportamiento en la realidad, sea capaz de avanzar por un entorno construido por el hombre, sin más ayuda que las imágenes tomadas por una cámara fisheye unida a la parte inferior del vehículo.

El escenario donde se ha realizado la simulación final está formado por dos pasillos unidos por una curva en ángulo recto. El suelo tiene un color rojizo; a un lado del pasillo aparecen paredes de tres colores distintos: morado, blanco y verde. El otro lado está formado por grandes ventanales con persianas de lama vertical. Se han colocado puertas, marcos, pomos, columnas, y rejillas para el apoyo de las lamas, todo esto con el fin de dotar al escenario de un mayor realismo. En la Figura 5.4 se muestran algunas imágenes de la escena tomadas con una cámara convencional y en la Figura 5.5 las tomadas con una cámara fisheye.

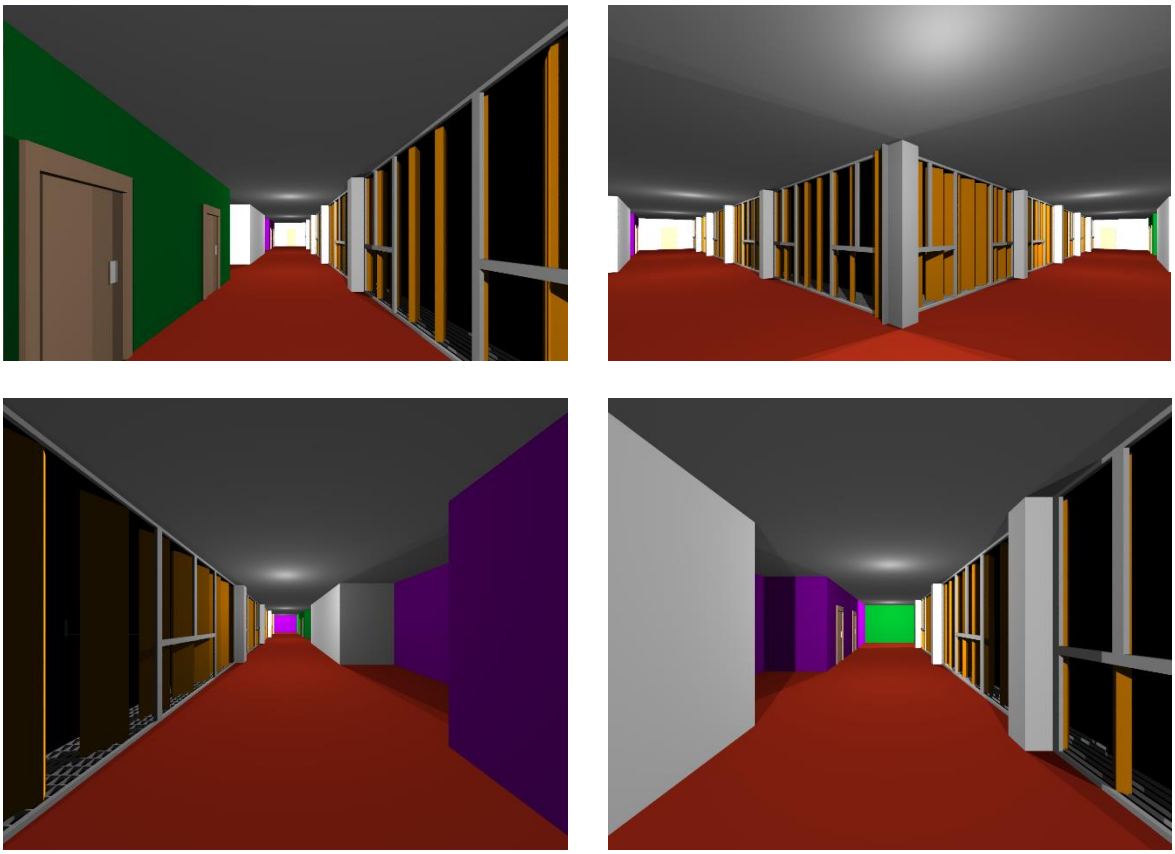

Figura 5.4. Imágenes del entorno tomadas con una cámara convencional

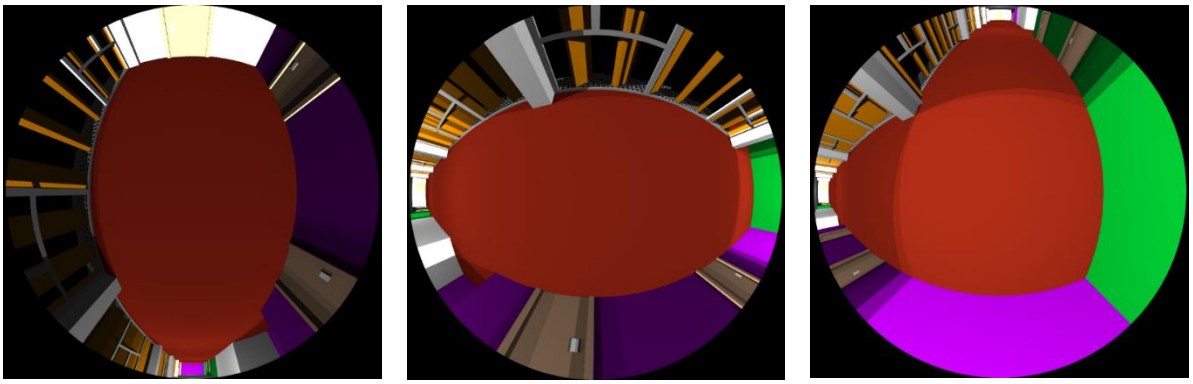

Figura 5.5. Imágenes del entorno tomadas por una cámara fisheye.

En este apartado se muestra el experimento principal, objeto de este trabajo. Se trata de un conjunto de imágenes tomadas por una cámara fisheye, donde se puede observar cómo el cuadricóptero avanza por un pasillo programado en POV-Ray. Primero avanza en línea recta por el primer pasillo, después toma la curva en ángulo recto y continúa avanzando por el segundo pasillo.

Como ya se ha comentado, los puntos de fuga únicamente nos dan información de la orientación, no de la posición, por tanto, debe existir un sistema auxiliar que comunique al cuadricóptero cuándo está aproximándose a la curva, para que así comience a avanzar en la otra dirección. En la realidad, esto sería fácil implementarlo con sensores de distancia, de manera que cuando el sensor detectara que la distancia entre el vehículo y la pared fuera menor de un determinado valor, se recalculara la orientación y tomara la curva.

Dado que en la simulación no tenemos ni un cuadricóptero real ni paredes reales, simplemente se le advierte al vehículo cuándo está a menos de una determinada distancia de la pared, para que cambie la dirección de avance.

El sistema implementado es suficientemente robusto, pero en casos puntuales no es capaz de extraer correctamente la orientación del pasillo respecto al cuadricóptero. Las simulaciones realizadas tienen una duración de 150 segundos; el algoritmo de visión se lanza cada segundo, por tanto se ejecuta 150 veces en una simulación completa. Se ha calculado cuántas veces el algoritmo es incapaz de calcular fielmente dicha orientación, y se ha obtenido un número de errores de entre 12 y 14, lo que supone una tasa de error de aproximadamente un 8,5%.

Se recuerda que para el cálculo de la matriz de rotación del pasillo respecto al cuadricóptero, se realiza una comparación con la obtenida en la iteración anterior, para así poder ordenar las direcciones principales correctamente en dicha matriz; por esta razón es importante que el sistema sea suficientemente robusto, pues si se obtiene un error en alguna iteración, perjudicaría

a las posteriores. Por eso, en los casos en los que se ha detectado error, se asigna directamente la matriz de rotación de la iteración justamente anterior, con lo que se resuelve el problema. Esto se puede hacer porque la visión se lanza cada poco tiempo (un segundo) y por tanto la matriz de rotación varía mínimamente.

Cabe destacar que la extracción de los planos de proyección, cuando el cuadricóptero se aproxima a la curva o cuando se encuentra en ella, no es demasiado exacta debido a la iluminación y a las sombras que se producen en la unión entre los dos pasillos (ver Figura 5.6). Por eso, es en este instante cuando se ha registrado un mayor número de errores (unos 6 errores de media). Así pues, sin tener en cuenta los errores debidos a la curva y recalculando la tasa de error, se obtiene un valor menor al 5%, lo que destaca el nivel de robustez del sistema.

En la Figura 5.6 se presenta una pequeña muestra de la secuencia de imágenes tomada justo en el momento que el cuadricóptero toma la curva en ángulo recto:

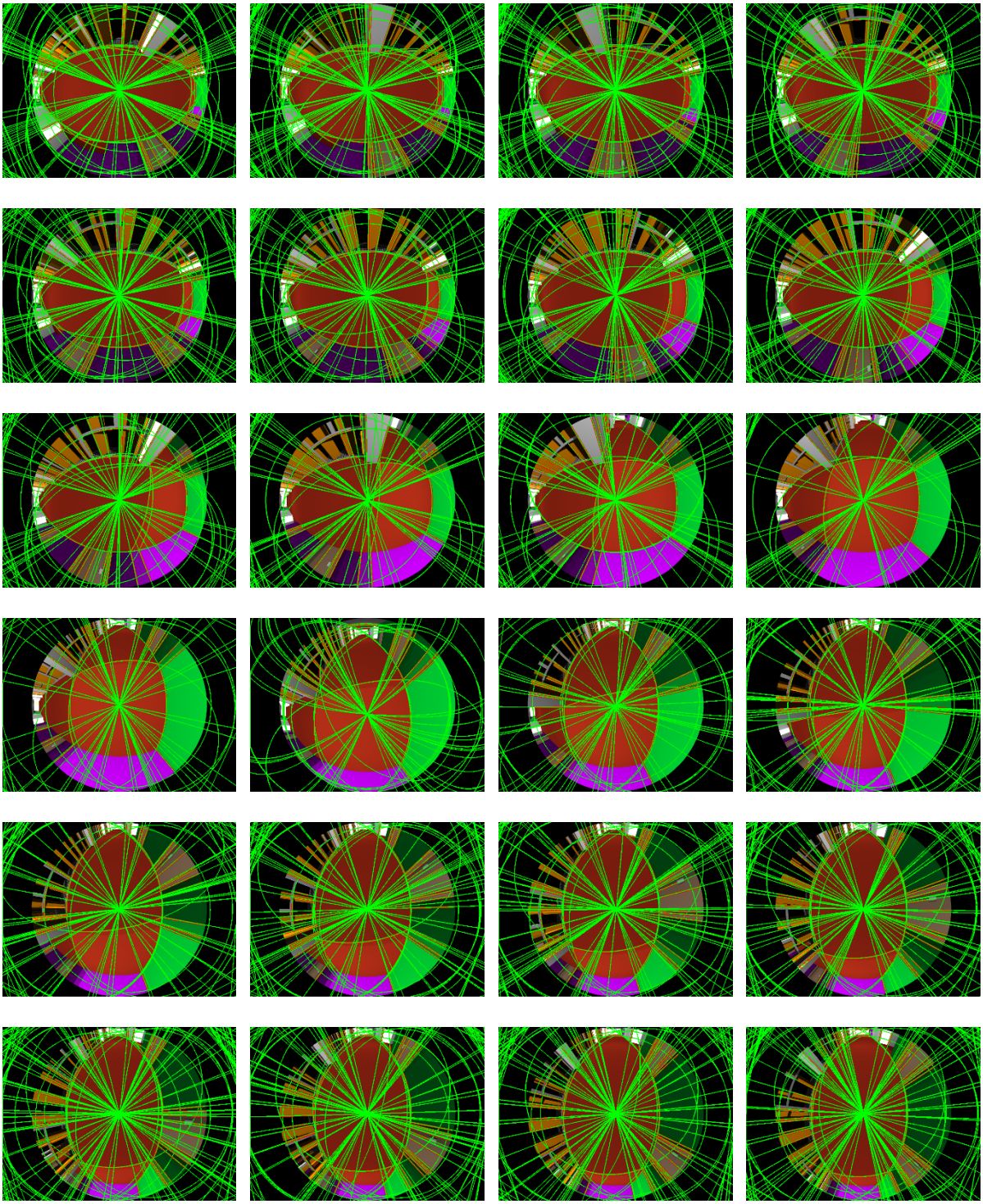

Figura 5.6. Secuencia de imágenes tomadas por la cámara fisheye del cuadricóptero (y los correspondientes planos de proyección) cuando éste toma la curva entre los dos pasillos.

6. Conclusiones

En este trabajo se ha creado un entorno de simulación para un cuadricóptero. Para ello, se han integrado distintos sistemas (modelo dinámico y control de un cuadricóptero [7], toolbox de extracción de rectas en cámaras fisheye [5] y software de rendering POV-Ray) en un proyecto común para el que se han desarrollado las correspondientes interfaces. Además, se han desarrollado los correspondientes módulos para el cálculo de los puntos de fuga, el cálculo de la orientación del cuadricóptero a partir de dichos puntos de fuga, así como la generación de la consigna de este vehículo.

Una importante parte del trabajo ha sido la integración de las distintas partes. En este apartado se puede observar cómo se ha dividido el problema u objetivo principal en pequeños problemas que se han ido resolviendo individualmente, como son: la dinámica y control del cuadricóptero, la programación de una escena en 3D con POV-Ray, la utilización del código de proyección de rectas en sistemas fisheye, el cálculo de los puntos de fuga aplicando el algoritmo RANSAC, la asignación de las consignas, etc. Una vez resueltos todos estos problemas, se han ido integrando en un único sistema, obteniendo el entorno de simulación que, finalmente, se ha presentado.

Se ha cumplido el objetivo central del trabajo: implementar un entorno de simulación para un cuadricóptero con cámara fisheye, así como las principales premisas que se presentaron en la introducción: que no fuera necesaria la señal GPS ni se conociera el entorno con antelación.

El sistema de visión y cálculo de la orientación y consignas es de una precisión destacable. Se han realizado simulaciones de navegación por el interior de un pasillo, llegando a recorrerse 70 metros, obteniendo una desviación máxima en dirección perpendicular a la dirección de vuelo menor a 1 metro. Debe tenerse en cuenta que la única información que se le proporciona al cuadricóptero es la que se obtiene de la cámara fisheye[5]. En un vehículo real, esta desviación podría corregirse con sensores de distancia, al igual que se ha hecho para tomar la curva del pasillo. Pero en nuestro entorno de simulación, el correcto desplazamiento del cuadricóptero se debe al preciso cálculo de la orientación a partir de la visión.

Una clara aplicación del trabajo realizado podría ser la vigilancia o la navegación autónoma en el interior de edificios desconocidos.

En un cuadricóptero real, los ángulos pitch y roll son relativamente fáciles de calcular. Con el uso de la IMU, se puede detectar fácilmente cuál es la dirección de la gravedad y así conocer

[5] Excepto para tomar la curva y cuando llega al final de un pasillo.

dichos ángulos. El ángulo yaw es bastante más complicado de estimar. Por eso, si en un futuro aumentara la potencia de cálculo de los procesadores, y se programara un código para la parte de visión algo más depurado, se podría incluir el sistema de visión en el bucle de control, utilizando la cámara fisheye como un sensor más, con el fin de mejorar la estabilización del cuadricóptero y también la estimación del ángulo yaw.

Anexo A. Detalle de nomenclatura utilizada

A continuación se exponen los principales parámetros y datos físicos de un cuadricóptero, necesarios para llevar a cabo el modelado dinámico del mismo.

Ajeno al cuadricóptero

g Gravedad

ρ Densidad del aire

μ Viscosidad del aire

Cuadricóptero

M Masa total

I Matriz de inercia

h Altura de los rotores sobre el centro de gravedad

d Longitud de los brazos

Rotores

n_b Número de aspas por rotor

r Radio del rotor

C_t Coeficiente adimensional de empuje vertical

C_q Coeficiente adimensional de par o momento

También son necesarios diversos parámetros relativos a las aspas y al flapping, que no son objeto de este trabajo.

Constantes derivadas de las anteriores

A Área del rotor

c_T Coeficiente de empuje vertical producido por el giro de los rotores

c_Q Coeficiente de momento producido por el giro de los rotores

Cabe destacar cómo se obtienen y de qué dependen los coeficientes c_T y c_Q (distintos de C_t y C_q, que son adimensionales), pues serán decisivos a la hora de realizar el control:

$$c_T = C_t \cdot \rho \cdot A \cdot r^2 \tag{A.1}$$

$$c_Q = C_q \cdot \rho \cdot A \cdot r^3 \tag{A.2}$$

$\mathbf{r} = (x \quad y \quad z)$	Posición en {W}, world (ref. absoluta)
$\mathbf{n} = (yaw \quad pitch \quad roll) = (\psi \quad \theta \quad \phi)$	Orientación en {W}
$\mathbf{v} = (\dot{x} \quad \dot{y} \quad \dot{z})_{\{W\}}$	Velocidad en {W}
$\mathbf{\Omega} = (\Omega_1 \quad \Omega_2 \quad \Omega_3)_{\{B\}}$	Velocidad angular en {B}
$\boldsymbol{\omega} = (\omega_1 \quad \omega_2 \quad \omega_3 \quad \omega_4)$	Velocidad angular de los rotores
$\mathbf{e_1}$ = (1 0 0) $\mathbf{e_2}$ = (0 1 0) $\mathbf{e_3}$ = (0 0 1)	Referencia del cuadro fijo del cuadricóptero. e_1 alineado con el rotor Norte o 1, y e_3 con la dirección y sentido de la gravedad. Observar que no es el mismo sistema de referencia que {B}, pues e3 siempre está alineado con la gravedad y en {B} no ocurre así.
${}^{W}\mathbf{R}_{B} = {}^{\text{Abs}}\mathbf{R}_{\text{Cuad}}$	Matriz de rotación del sistema de referencia {B} respecto de {W}
$\mathbf{W}^{-1}$	Wronskiano inverso
I	Matriz de inercia del cuadricóptero
T	Empuje vertical producido por el giro de los rotores
$\boldsymbol{\tau}$	Par producido por el empuje, **T**, de los rotores
Q	Par producido por el giro de los rotores.

Anexo B. Wronskiano

Para realizar la transformación de las velocidades angulares (expresadas en el sistema de referencia {B}) a las derivadas de los ángulos RPY es necesario el uso de lo que P. Corke denomina wronskiano inverso [7] y que se obtiene de la siguiente manera:

Dada la siguiente ecuación[6] (fuente: [4]):

$$\dot{\mathbf{R}} = \mathbf{R} \cdot \boldsymbol{\Omega}_{\times} \tag{B.1}$$

Donde $\boldsymbol{\Omega}_{\times}$ es el tensor velocidad angular; se trata de una matriz antisimétrica tal que $\boldsymbol{\Omega}_{\times} \cdot \mathbf{v} = \boldsymbol{\Omega} \times \mathbf{v}$, así pues $\boldsymbol{\Omega}_{\times}$ toma la siguiente forma:

$$\boldsymbol{\Omega}_{\times} = \begin{pmatrix} 0 & -\Omega_3 & \Omega_2 \\ \Omega_3 & 0 & -\Omega_1 \\ -\Omega_2 & \Omega_1 & 0 \end{pmatrix} \tag{B.2}$$

Se toma la matriz $\mathbf{R}$ y se realiza, para cada elemento, la derivada respecto de los ángulos 'roll', 'pitch' y 'yaw'. Formando así la matriz $\dot{\mathbf{R}}$:

$$\dot{R}_{ij} = \frac{\partial R_{ij}}{\partial \phi}\dot{\phi} + \frac{\partial R_{ij}}{\partial \theta}\dot{\theta} + \frac{\partial R_{ij}}{\partial \psi}\dot{\psi} \tag{B.3}$$

Esta matriz se reordena y se dispone en forma de vector (utilizando el operador *vec* [12]), de manera que se cumpla la siguiente igualdad:

$$vec\left[\dot{\mathbf{R}}\right] = \underset{9x3}{\mathbf{A}} \cdot \begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} \tag{B.4}$$

Por otro lado, se realiza el producto $\mathbf{R} \cdot \boldsymbol{\Omega}_{\times}$; se dispone en columnas al igual que se hizo para $\dot{\mathbf{R}}$, y se reordenan los términos: en la primera columna, los factores que dependen de Ω_1, en la segunda, los que dependen Ω_2, y en la tercera, los que dependen de Ω_3. Se formará la matriz $vec\left[\mathbf{R} \cdot \boldsymbol{\Omega}_{\times}\right]$ de la misma manera que en el caso anterior:

---

[6] Por simplicidad, en este anexo denotaremos $\mathbf{R}$ a la matriz de rotación ${}^{W}\mathbf{R}_{B}$.

$$vec\left[\mathbf{R}\cdot\boldsymbol{\Omega}_{\times}\right] = \underset{9x3}{\mathbf{B}}\cdot\begin{pmatrix}\Omega_1\\ \Omega_2\\ \Omega_3\end{pmatrix} \tag{B.5}$$

Las matrices **A** y **B** tienen dimensión 9x3. Los grados de libertad únicamente son 3, por tanto sabemos que de las 9 filas, 3 de ellas serán independientes y las otras 6 restantes serán dependientes de las anteriores. Así, se escogen 3 filas de ambas matrices, formando las matrices $\tilde{\mathbf{A}}$ y $\tilde{\mathbf{B}}$ respectivamente, asegurándonos de que su rango sea 3, es decir, que sus filas sean independientes y por tanto las matrices sean invertibles.

Finalmente, se realiza la siguiente operación:

$$\tilde{\mathbf{A}}\cdot\begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{pmatrix} = \tilde{\mathbf{B}}\cdot\begin{pmatrix}\Omega_1\\ \Omega_2\\ \Omega_3\end{pmatrix} \Rightarrow \begin{pmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{pmatrix} = \underbrace{\tilde{\mathbf{A}}^{-1}\cdot\tilde{\mathbf{B}}}_{\substack{\mathbf{W}^{-1}\\ \textit{wronskiano}\\ \textit{inverso}}}\cdot\begin{pmatrix}\Omega_1\\ \Omega_2\\ \Omega_3\end{pmatrix} \tag{B.6}$$

De esta manera, se ha obtenido el wronskiano inverso, necesario para poder transformar las velocidades angulares del cuadricóptero (expresadas en el sistema de referencia {B}) a las derivadas de los ángulos RPY.

Anexo C. Control del cuadricóptero

Ya se ha explicado cómo se relacionan las fuerzas y pares producidos con la velocidad angular de cada rotor. Esto es interesante saberlo, pero realmente el control no se realiza en términos de fuerzas y pares, sino en términos de velocidades angulares de los rotores, y a continuación se explica por qué: fijándonos en las ecuaciones expresadas en (2.11), se deduce que para hallar ω_i^2 bastaría con invertir la matriz **A** y multiplicarla por la matriz del empuje y los pares, de manera que se obtendría lo siguiente:

$$\begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \mathbf{A}^{-1} \begin{pmatrix} T_\Sigma \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} = \begin{pmatrix} c_T & c_T & c_T & c_T \\ 0 & dc_T & 0 & -dc_T \\ -dc_T & 0 & dc_T & 0 \\ -c_Q & c_Q & -c_Q & c_Q \end{pmatrix}^{-1} \begin{pmatrix} T_\Sigma \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} \tag{C.1}$$

El objetivo es calcular ω_i a partir de ω_i^2; es aquí donde se presenta el principal problema, pues en algunos casos obtenemos $\omega_i^2 < 0$, siendo irresoluble en el conjunto de los números reales.

En la Figura C.1 se muestra el esquema de control original extraído de la toolbox de Peter Corke [7]:

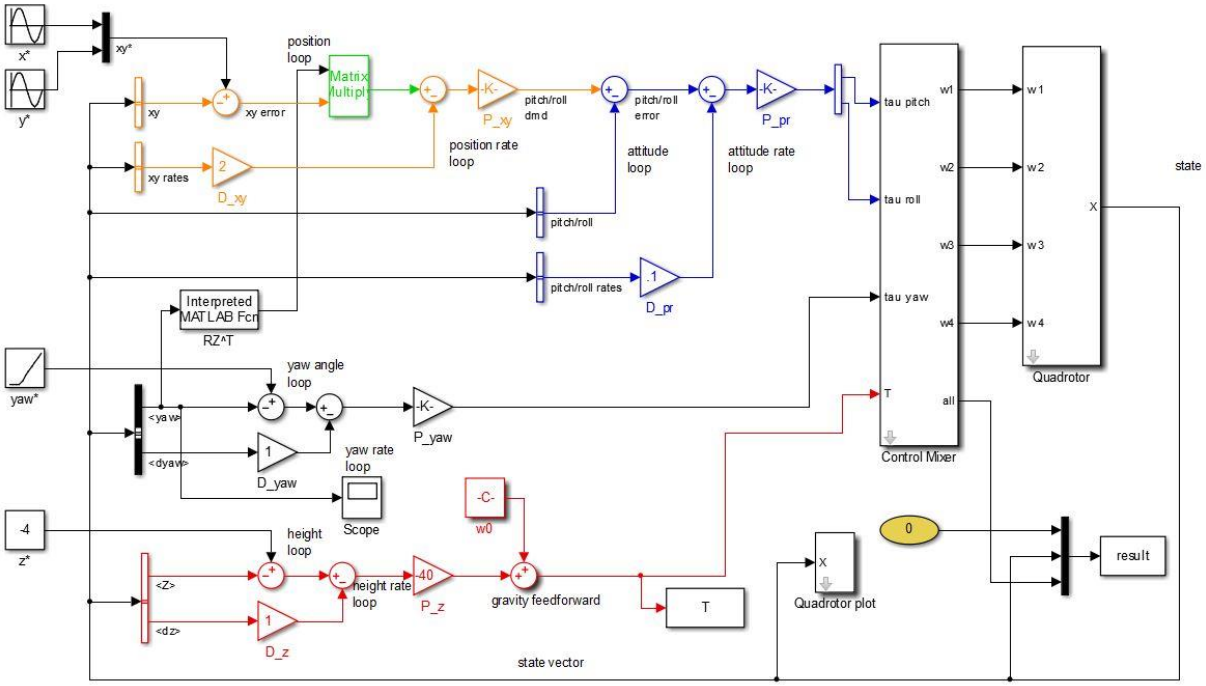

Figura C.1. Esquema de control original. Fuente: [7]

El esquema de control se puede dividir en cuatro partes, comenzando desde abajo hacia arriba (por orden de complejidad) y de izquierda a derecha:

1. Control de la altura del cuadricóptero en el sistema de coordenadas absoluto {W}.
   Se muestra en color rojo, y se rige por la siguiente ecuación (hay que remarcar que estamos hablando de ω_T y no de T, pues se está calculando la velocidad angular de cada rotor que produciría una determinada fuerza de empuje T):

$$\omega_T = K_p(z^* - z) + K_d(\dot{z}^* - \dot{z}) + \omega_0 \tag{C.2}$$

   Normalmente el término $\dot{z}^*$ se puede despreciar [3], y reordenando y agrupando las constantes queda lo siguiente:

$$\omega_T = P_z[(z^* - z) - D_z\dot{z}] + \omega_0 \tag{C.3}$$

   Donde:

$$\omega_0 = \sqrt{\frac{M \cdot g}{4 \cdot c_T}} \tag{C.4}$$

   Que es la velocidad angular que debe tener cada rotor para compensar el propio peso del cuadricóptero.

2. Control del ángulo 'yaw' (eje 'z') del cuadricóptero en el sistema de referencia {W}.
   Se muestra en color negro y, al igual que para la altura, se utiliza un control derivativo-proporcional:

$$\omega_\psi = K_p(\psi^* - \psi) + K_d(\dot{\psi}^* - \dot{\psi}) = P_\psi[(\psi^* - \psi) - D_\psi\dot{\psi}] \tag{C.5}$$

3. Control de la posición 'x' e 'y' del cuadricóptero en el sistema de referencia {V}[7], actuando sobre los ángulos 'pitch' (eje 'y') y 'roll' (eje 'x'). Se realiza en paralelo el control de 'x' e 'y'. Por simplicidad y por evitar redundancia, se explica tan sólo para 'x'.

[7] Sistema de referencia ligado al cuadricóptero, con el mismo origen que {B}, pero con los ejes x e y paralelos al suelo.

Para conseguir desplazar el cuadricóptero en 'x' únicamente se puede actuar sobre el ángulo 'pitch'. Así pues, a partir de la posición actual y la consigna en 'x', se calcula qué ángulo pitch es necesario; este primer cálculo aparece en color naranja (ver Figura C.1). Una vez calculado el ángulo pitch, se calcula qué velocidad angular de los rotores proporcionará un par en el eje 'y' suficiente para producir dicha rotación.
El uso del sistema de referencia {V} está debidamente justificado: supongamos que situamos al cuadricóptero en un pasillo y queremos que se mueva a lo largo del eje 'x' del mismo. Esto produciría una pequeña rotación en el eje 'y', de manera que el cuadricóptero podría avanzar según 'x'. En ese instante, el eje 'x' del pasillo no coincidiría con el eje 'x' del cuadricóptero. Es por ello que hay que definir el nuevo sistema de referencia {V}. Así, se pasa del sistema de referencia fijo o absoluto {W} al sistema de referencia {V}:

$$\begin{pmatrix} x & y \end{pmatrix}_{\{V\}} = \begin{pmatrix} x & y \end{pmatrix}_{\{W\}} \begin{pmatrix} \cos\psi & \sin\psi \\ -\sin\psi & \cos\psi \end{pmatrix} \tag{C.6}$$

A continuación se presenta la ecuación con la que, una vez calculada la consigna en pitch, se obtendrá la velocidad angular necesaria:

$$\omega_\theta = K_p(\theta^* - \theta) + K_d(\dot{\theta}^* - \dot{\theta}) = P_\theta[(\theta^* - \theta) - \mathrm{D}_\theta \dot{\theta}] \tag{C.7}$$

4. Bloques "Control Mixer", "Quadrotor" y "Quadrotor Plot".
   - Control Mixer. Para cada rotor, se suman/restan adecuadamente las velocidades individuales obtenidas $\omega_\phi, \omega_\theta, \omega_\psi$ y ω_T, se saturan a una velocidad angular máxima de 1000 rad/s y se definen los sentidos de rotación de cada rotor. De este bloque, obtenemos las velocidades angulares con signo de cada rotor.
   - Quadrotor. En este bloque se introducen las velocidades angulares de cada rotor en la S-function, *quadrotor_dynamics* [7], donde se halla la dinámica del cuadricóptero. Esta función nos da el estado del cuadricóptero:

$$X = \begin{pmatrix} x & y & z & \psi & \theta & \phi & \dot{x} & \dot{y} & \dot{z} & \dot{\psi} & \dot{\theta} & \dot{\phi} \end{pmatrix} \tag{C.8}$$

   - Quadrotor Plot. Hay que introducir el estado del cuadricóptero; dentro de este bloque está la función quadrotor_plot, que dibuja un esquema muy sencillo del cuadricóptero a través del cual se puede ver qué movimiento está realizando.

Anexo D. Generador de imágenes sintéticas POV-Ray

POV-Ray (Persistence Of Vision Ray-tracer) es un programa de software libre destinado a la creación de imágenes en tres dimensiones.

La escena se describe en un fichero de código que siempre debe contener: la posición y los parámetros de la luz, la cámara y los objetos.

En este trabajo, POV-Ray se ha utilizado para generar, sucesivamente, las imágenes que se verían con una cámara fisheye situada debajo del cuadricóptero. De esta manera, una vez asignadas las luces y el entorno (en este caso, un pasillo), se ejecuta el programa desde Matlab, asignando la posición y orientación adecuada de la cámara fisheye conforme avanza la simulación. Las imágenes tomadas se han utilizado para extraer los planos de proyección, calcular los puntos de fuga y, así, la orientación del cuadricóptero respecto al pasillo, de manera que, generando adecuadamente las consignas, pueda navegar por éste.

Anexo E. Manual del programa

En caso de que terceras personas quisieran comprender, modificar o reutilizar los programas desarrollados para realizar este entorno de simulación, en este anexo se explica de manera general el funcionamiento y la estructura del mismo.

En la Tabla E.1 se muestran las principales funciones utilizadas, los argumentos que se deben introducir y las variables que éstas devuelven.

	Nombre función	**Argumentos**	**Return**
1	funcion_completa	Vector de tiempos de la simulación, estado del cuadricóptero, velocidad angular de los rotores. Todo esto en un vector fila.	x^* y^* z^*
2	quad2pov	${}^{Abs}\mathbf{T}_{Cuad}$	${}^{Abs}\mathbf{T}_{Cam}$
3	imagen_principal	${}^{Abs}\mathbf{T}_{Cuad}$	$\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$ (con sentido aleatorio y sin ordenar)
4	create_scene	${}^{Abs}\mathbf{T}_{Cam}$	$\mathbf{n_i}$ y la imagen que ha tomado la fisheye (también aparecen dibujados los planos de proyección)
6	ransac_anidado	$\mathbf{n_i}$, k (número de intentos), umbral	$\mathbf{v_1}, \mathbf{v_2}, \mathbf{v_3}$ (con sentido aleatorio y sin ordenar), y los planos de proyección clasificados en 3 direcciones.
7	ransac_ind	$\mathbf{n_i}$ (los sobrantes, en caso de que ya se haya realizado alguna iteración), k, umbral.	$\mathbf{v}_{1,2\,ó\,3}$, $\mathbf{n}_{utilizados}$, $\mathbf{n}_{siguiente\_iterac.}$

8	pintar_acepto_rechazo	$\mathbf{n}_{utilizados}$, $\mathbf{n}_{siguiente\_iterac.}$ para las 3 direcciones	Ejecuta y guarda una figura compuesta por 6 'plots', donde se ve la clasificación de los planos (ver Figura 3.4)
9	ordenaRotacion	${}^{Abs}\mathbf{R}_{Cam}$ de la iteración anterior	${}^{Cam}\mathbf{R}_{Pas}$
10	consignas_automatico	${}^{Abs}\mathbf{R}_{Cuad,}$ ${}^{Cam}\mathbf{R}_{Pas,}$ $\mathbf{r}$ (posición quad. respecto abs.), x^* y^* z^* de la iteración anterior.	x^* y^* z^*

Tabla E.1. Principales funciones con los respectivos argumentos y las variables que devuelven.

El esquema de Simulink (donde se ejecuta 'funcion_completa') recibe el nombre de 'sl_quadrotor_n.mdl'. Se ha creado un fichero de Matlab desde el que se lanza dicho esquema de Simulink, con el nombre: 'Lanza_sl_quadrotor_n.m', donde se pueden fijar, entre otros, los siguientes parámetros:

- Periodo de muestreo del algoritmo de visión (en segundos). Variable 'tLimit(2)'
- Duración de la simulación (en segundos).
- Ángulo de giro del cuadricóptero en el eje 'z' (en radianes).

Cabe destacar la existencia de dos ficheros Matlab imprescindibles:

'mdl_quadrotor.m', donde se hallan las variables y constantes físicas del cuadricóptero y del entorno (densidad del aire, gravedad…).

'quadrotor_dynamics.m'; se trata de una 'S-function' donde se ejecuta la dinámica del cuadricóptero. En caso de querer cambiar la situación inicial (posición y orientación) del cuadricóptero es aquí donde debe hacerse.

Si se realizara un cambio en la escena 3D generada en POV-Ray habría que modificar las distancias para tomar la curva o señalar el final de pasillo en 'consignas_automatico'.

El fichero de POV-Ray donde está programado el entorno recibe el nombre de 'CorridorObj.pov', y el fichero donde se carga la posición y orientación actualizadas de la cámara es 'input.pov'. La versión de POV-Ray utilizada es la 3.6.

La función 'create_scene' merece especial atención, pues es desde donde se ejecuta POV-Ray; se genera y se guarda la imagen obtenida por la cámara fisheye, y se realiza la extracción de los planos de proyección. Para su correcto funcionamiento es necesario determinar la ruta donde

se encuentra el ejecutable de POV-Ray, así cada vez que se lance el algoritmo de visión, Matlab será capaz de abrir y cerrar dicho software. También se determinan otros parámetros de configuración de POV-Ray, como son: la dimensión y definición de la imagen obtenida, el nombre de los ficheros '.pov', el nombre del objeto programado en POV-Ray, si el sistema es dextrógiro o levógiro, etc. Otros parámetros que se definen en 'create_scene' son los relativos a la configuración del procesado de la imagen, a la configuración de la cámara, a la extracción de los planos de proyección y al algoritmo RANSAC utilizado para la correcta extracción de dichos planos [5].

En la Figura E.1 se muestra de manera esquemática la jerarquía que siguen algunas de las funciones anteriormente descritas con el fin de obtener una idea global del sistema.

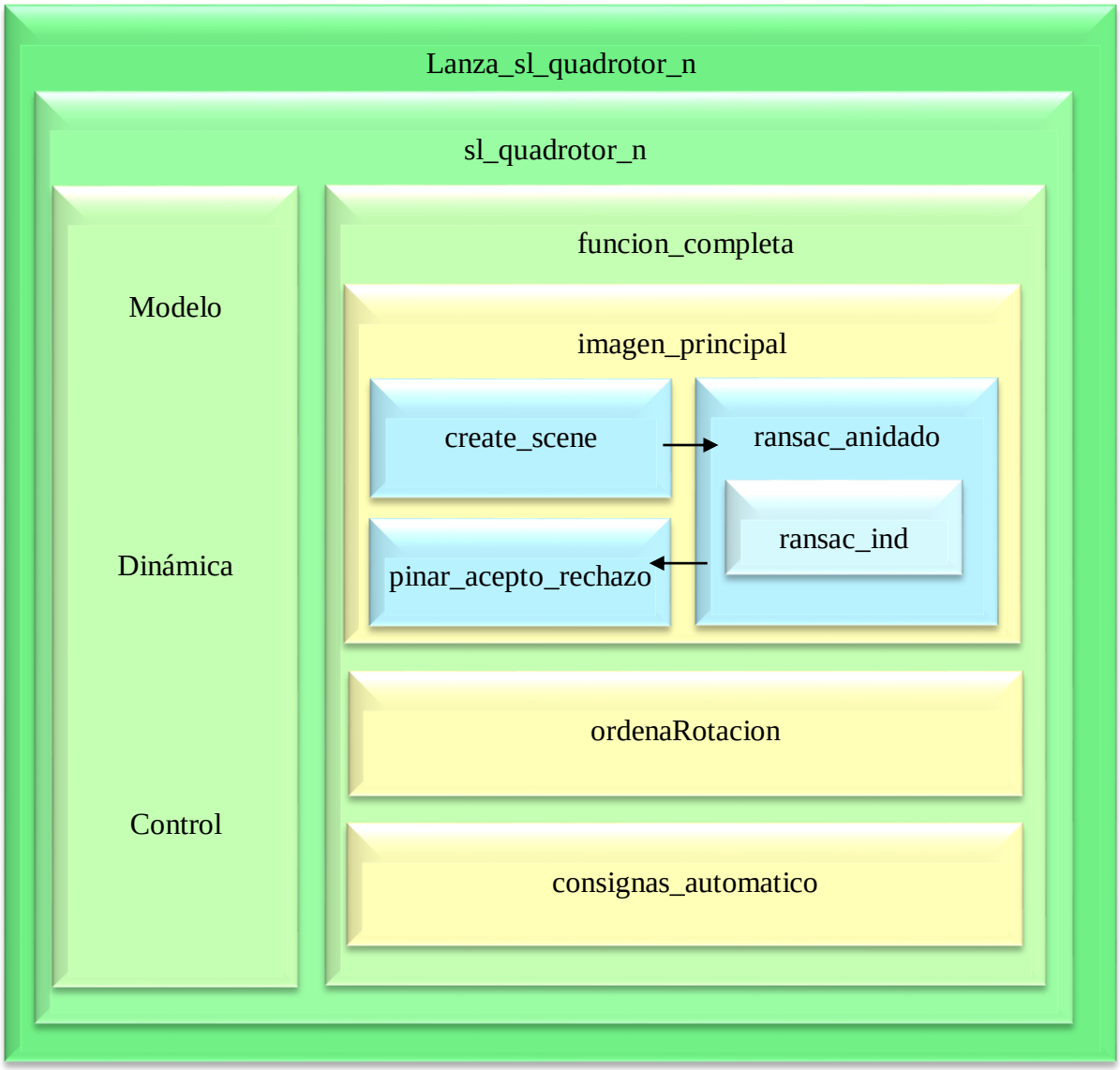

Figura E.1. Jerarquía de las principales funciones que componen el sistema

Bibliografía

[1] M. Blösch, S. Weiss, D. Scaramuzza, R. Siegwart, *Vision Based MAV Navigation in Unknown and Unstructured Environments,* Alaska, 2010.

[2] M. Tarhan, E. Altuğ, *EKF Based Attitude Estimation and Stabilization of a Quadrotor UAV Using Vanishing Points in Catadioptric Images,* publicación online Springer Science+Business Media B.V., 2010.

[3] P. Corke, *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*, Berlin: Springer-Verlag, 2011. Capítulo 4.3 Flying Robots (páginas 78-86).

[4] R. Mahony, V. Kumar, P. Corke, *Multirotor Aerial Vehicles. Modeling, Estimation and Control of Quadrotor*, IEEE Robotics & Automation Magazine, 2012.

[5] J. Bermúdez, *Automatic Line Extraction Toolbox for Uncalibrated Central Systems with Revolution Symmetry v0.5 Alpha*, http://webdiis.unizar.es/~bermudez/toolbox, 2014.

[6] P. Edward, I. Pounds, *Design, Construction and Control of a Large Quadrotor Micro Air Vehicle,* Australian National University, 2007.

[7] P. Corke, *Robotics Toolbox for MATLAB,* www.petercorke.com, 2011.

[8] J. Bermúdez, G. López, J.J. Guerrero, *A Unified Framework for Line Extraction in Dioptric and Catadioptric Caneras*, 11th Asian Conference on Computer Vision, 2012.

[9] J. Bermúdez, G. López, J.J. Guerrero, *Line extraction in uncalibrated central images with revolution symmetry*, 24th British Machine Vision Conference, 2013.

[10] Martin A. Fischler, Robert C. Bolles, *Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography*, Communications of the ACM,1981.

[11] J.J. Guerrero, Apuntes de la asignatura *Automatización Flexible y Robótica* para el Grado en Ingeniería de Tecnologías Industriales, EINA Universidad de Zaragoza, 2013.

[12] J.L. Blanco, *A tutorial on SE(3) transformation parameterizations and on-manifold optimization*, Universidad de Málaga (http://sites.google.com/site/jlblancosite/), 2013.