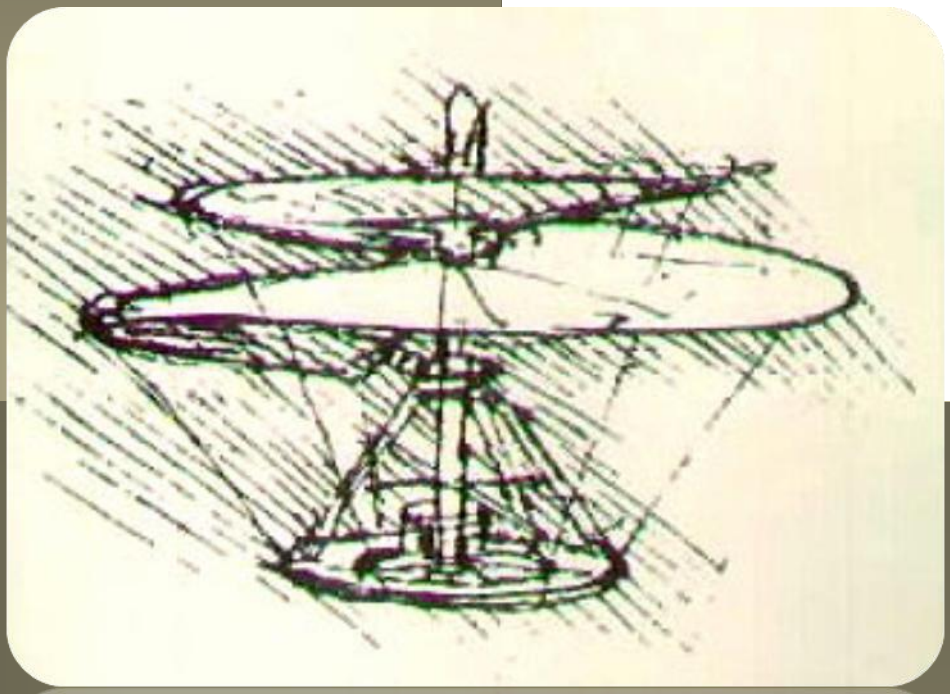
S.S.I.I. 08/09 - Control de un vehículo aéreo no tripulado



Autores:

Daniel Garijo Verdejo
Jesús Ismael López Pérez
Isaac Pérez Estrada

Directores de proyecto:

José Jaime Ruz Ortiz
José Antonio López Orozco

Universidad Complutense de Madrid

Facultad de informática

Resumen

Un Vehículo Aéreo no Tripulado (UAV: Unmanned Aerial Vehicle) es un vehículo controlado autónomamente o desde tierra utilizando planes de vuelo programados. Las aplicaciones de este tipo de vehículos es cada día mayor en tareas que implican algún tipo de dificultad o riesgo para vehículos convencionales tripulados por personas, como son la detección de incendios, la identificación de manchas de petróleo en el mar, el seguimiento del tráfico, la inspección de líneas de tendido eléctrico, etc.

Para el curso 2008/2009 en la asignatura de S.I. nos proponemos controlar automáticamente, desde un computador, un vehículo aéreo cuatrimotor para que realice rutas pre programadas. Para ello será necesario resolver dos problemas principales:

1) Identificación de la posición espacial 3D y el ángulo de orientación del cuatrimotor en tiempo real.

2) Actuación sobre el cuatrimotor en función de su posición y de la ruta programada.

En este proyecto planteamos realizar la tarea de actuación para controlar el cuatrimotor a través de una emisora de radiofrecuencia conectada al computador. Cada uno de los cuatro canales de la emisora se controla independientemente desde el computador a través de una conexión serie RS-232 según el esquema de la siguiente figura:

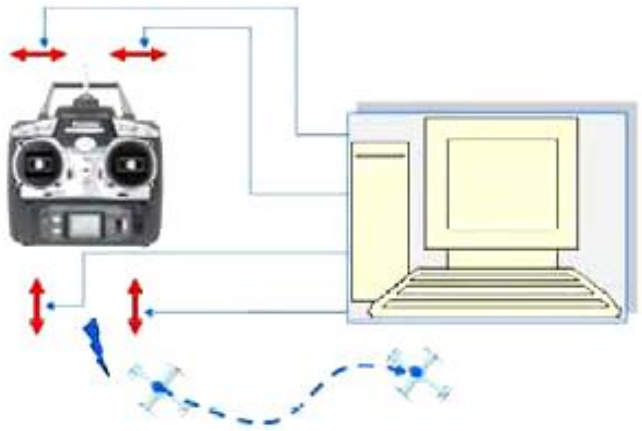


Figura 1: conexión PC - emisora

Para obtener un control fiable del cuatrimotor será necesario un examen y análisis de las señales que se envían a la emisora y del efecto que surte en el cuatrimotor, es decir, una calibración de cada una de las señales. Así se puede crear un sencillo modelo del funcionamiento del cuatrimotor que se aproxime al funcionamiento real del que se está probando.

Este modelo permite que, dadas unas señales enviadas al vehículo y su posición anterior, estime la posición actual del cuatrimotor. Esta posición se utilizará junto con la ruta programada para dar la siguiente consigna y así sucesivamente hasta finalizar la ruta escogida.

Además se programarán actuaciones repetitivas o de emergencia, como por ejemplo el despegue a una altura determinada o el aterrizaje en caso de pérdida de la posición. La ruta realizada se aproximará a la deseada tanto más cuando mejor sea la estima de la posición real del vehículo. El modelo permitirá una demostración de que el control y actuación sobre el cuatrimotor es correcta y se observará que el cuatrimotor realiza físicamente las trayectorias solicitadas. Evidentemente si, en lugar de utilizar una posición estimada, se realimenta con la posición real del cuatrimotor las trayectorias serán más precisas y podrán realizarse maniobras y rutas más complejas.

Palabras clave

UAV, posición espacial, emisora, ángulo de orientación, tiempo real, cuatrimotor, modelo, RS-232, consigna.

Abstract

An UAV, (Unmanned Aerial Vehicle), is an autonomous aerial vehicle which can also be controlled from land with programmed flight plans. The importance of this kind of vehicles in tasks which involve any difficulty or risk for human beings is becoming greater nowadays, tasks like fire detection, oil stain identification on the sea, traffic tracking, etc.

Our purpose for this year is to develop an automatic control, which can order an UAV to follow programmed routes from a computer. To achieve our goal, two main problems have to be solved:

1) Identification of the 3D position and the orientation angle of the UAV.

2) Acting over the UAV depending on its position and programmed route.

The UAV will be controlled thanks to an emitter connected to the computer. Each of its four channels is controlled independently by the computer through an RS-232 connection according to this figure:

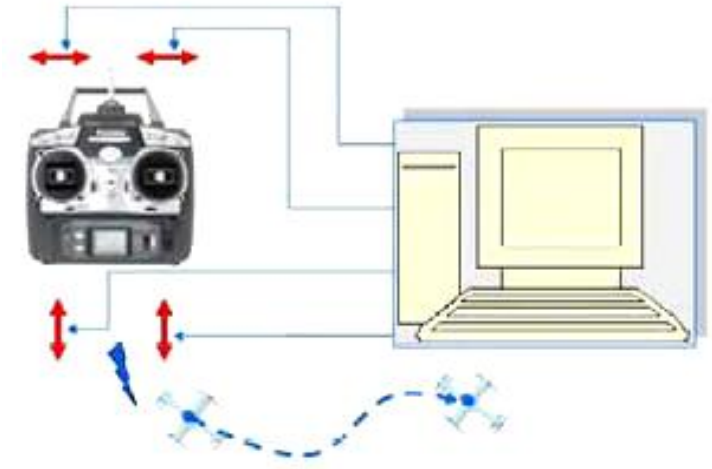


Figura 2: PC - controller connection

To obtain a reliable control of the UAV, it is necessary to calibrate each of the signals sent to the emitter. As a result, the model of the tested UAV's behaviour is generated.

This model allows us to estimate the current position of the vehicle using given signals and its last position. This position is used with the programmed route to calculate the next position, repeating the process until the UAV finishes the chosen route.

Emergency protocols will also be programmed, such as taking off until certain height is reached, or landing if the controller looses the track of the position. The followed route will be closer to the desired route when estimation of the real position improves. The model will allow a demonstration proving that the control over the UAV is correct, and we will check that the UAV follows the chosen routes. Of course, if we use the real position of the UAV instead of the estimated position, we will be able to execute more complex movements and routes.

Keywords

UAV, 3D position, emitter, orientation angle, real time, model, RS-232, next position.

AUTORIZACIÓN A LA UCM

Por la presente se autoriza a la Universidad Complutense de Madrid a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto esta memoria, como el código, la documentación y el prototipo desarrollado.

Los autores:

Daniel Garijo Verdejo

Jesús Ismael López Pérez

Isaac Pérez Estrada

ÍNDICE

1 INTRODUCCION Y OBJETIVOS

1.1 Introducción

En este proyecto se ha diseñado e implementado un sistema automático de control que mantiene un helicóptero (o un cuatrimotor) volando en una posición fija. El sistema recibe como entrada el error respecto a la posición objetivo, y genera, como salida, las señales de control que corrigen su posición.

El error se obtiene de un sistema de visión independiente formado por dos cámaras, que observan la superficie donde se mueve el helicóptero y calculan la diferencia entre la posición objetivo, y la posición real. Esto permite transformar el control manual representado en la Figura 3:

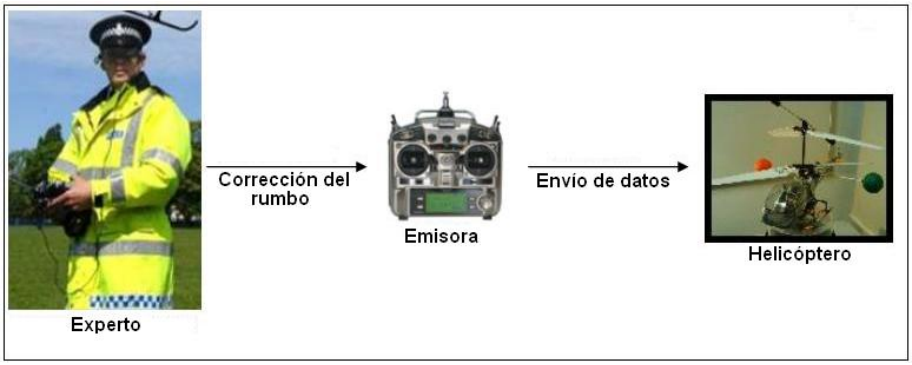


Figura 3: Control manual

En uno automático tal y como muestra la Figura 4:

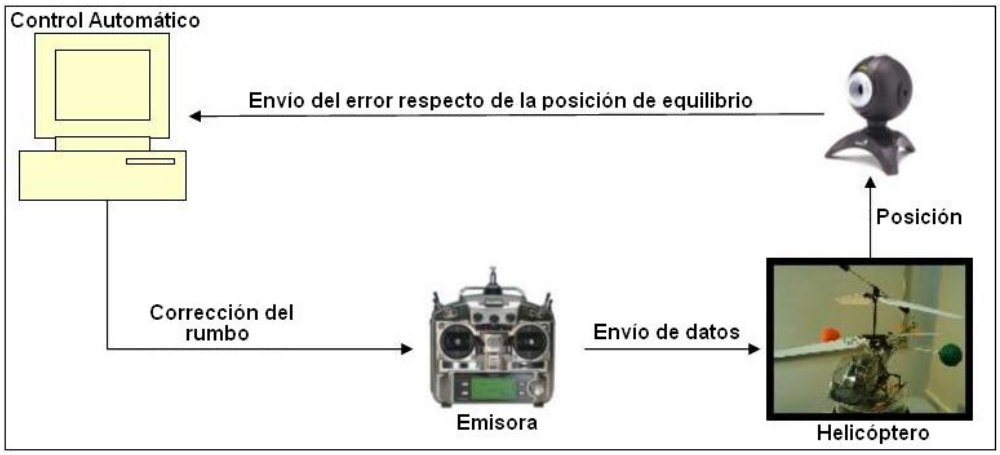


Figura 4: Control automatizado

Se ha desarrollado un simulador para ajustar los parámetros del controlador. Esto permite realizar las pruebas necesarias para validar el comportamiento del sistema sin dañar el helicóptero.

El controlador reacciona rápidamente a los cambios efectuados por el helicóptero. Recibe el error respecto a la posición que debería ocupar, calcula las señales de control correspondientes y las envía bien a la emisora o bien al simulador.

En cuanto al simulador, se comporta de forma similar al sistema real. Se comunica con el controlador, y ofrece una visualización adecuada de los resultados de las simulaciones. Además, es independiente del controlador para favorecer la velocidad de éste, (Figura 3).

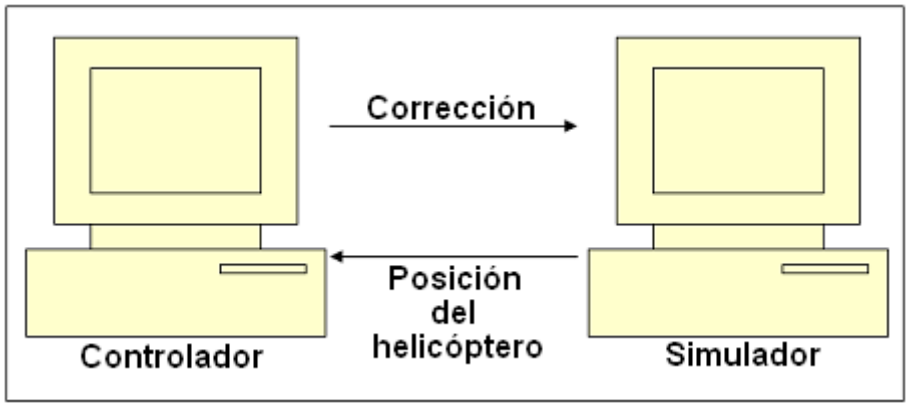


Figura 5: Comunicación Controlador - Simulador

Para el desarrollo de ambas aplicaciones se ha decidido utilizar el lenguaje C#, así como el motor gráfico gratuito TV3D. Se tomó esta decisión por considerar que ambas herramientas son gratuitas, están bien documentadas y han sido utilizadas con éxito en proyectos anteriores.

1.2 Objetivos

A continuación, detallamos las fases de desarrollo del proyecto, señalando los objetivos de cada una de ellas:

1) Fase inicial: **generador de escenarios**.
    - Desarrollo de un generador de escenarios con el que crear mapas para el simulador de manera sencilla. Se definen cómo han de ser los escenarios y se sientan las bases del simulador.
2) Primera fase de desarrollo: **implementación del simulador**.
    - Estudio de la dinámica de vuelo para que sea realista.
    - Implementación de un control sencillo para comenzar a realizar las pruebas del comportamiento simulado del helicóptero.
    - Desarrollo de una aplicación desde el sistema simulado para realizar seguimiento de contornos terrestres a partir de las imágenes captadas por una cámara a bordo.
3) Segunda fase de desarrollo: **implementación del controlador**.
    - Utilización de las pruebas realizadas en la fase anterior para diseñar un controlador que guie de manera automática el helicóptero y lo mantenga estable en un punto.
    - Comunicación del controlador con el simulador y el programa de identificación óptica de la posición.
    - Realización de algoritmos de movimientos predefinidos para el helicóptero.
    - Ajuste de los parámetros del controlador mediante pruebas con el simulador.
4) Fase de pruebas con el helicóptero: **ajuste del controlador**.
    - Ajuste de los parámetros del controlador para conseguir que el helicóptero permanezca estable en un punto.
    - Realización de un sistema para visualizar los parámetros enviados y recibidos por el controlador en las pruebas reales y poder corregir los errores.

La fase inicial ha servido también como introducción al lenguaje C# y al uso del motor TrueVision3D. Las fases 2 y 3 han sido las de mayor tiempo de desarrollo, y en ellas además se ha realizado un estudio acerca de posibles ampliaciones para el simulador. En la fase 4 es cuando se ha culminado la redacción de la documentación.

1.3 Estado del arte

En esta sección veremos algunos trabajos relacionados con el proyecto que hemos desarrollado, y posteriormente explicaremos otros sistemas de control alternativos al que hemos utilizado.

1.3.1 Sistemas relacionados

Existe un interés general por el control no tripulado de vehículos, ya que en los últimos años se han desarrollado muchos proyectos relacionados con este tema. Algunos incluso se ayudan de un sistema de localización, dependiendo de las necesidades del sistema.

El tipo de control empleado también puede llegar a ser muy distinto. Podemos optar desde un control PID, (Proporcional, Integral, Derivativo), usado mucho en aviación y basado en la realimentación de sus parámetros, hasta un control lineal con movimiento constante. En cualquier caso, se suelen complementar con algoritmos de planificación de trayectorias.

El sistema relacionado más simple es una aspiradora robot, sin localización en el espacio, con un control lineal simple y varias rutinas. Se trata de un sistema independiente muy básico. Un sistema algo más avanzado es el de un robot para extinguir incendios:

Un grupo de investigadores de una Universidad en Alemania, desarrolló un robot para apagar incendios forestales. Conocido como "OLE", el robot, semejante a un escarabajo, puede recorrer entre 20 y 30 kilómetros por hora en busca de áreas que podrían generar incendios forestales. Además, en caso de peligro, éste puede retraer sus patas y protegerse con el cuerpo (que funciona como un escudo). El robot "OLE" funciona por biosensores, los cuales detectan fuentes que causan fuego. De acuerdo a los investigadores, existen 30 robots que podrían proteger alrededor de 4,300 kilómetros de áreas verdes, dentro de un bosque.

Figura 6: OLE, robot apaga incendios

Anna Konda mide 3 metros y pesa 75 kilos, pero no es la "Anaconda" de las películas, es una imitación de una serpiente a nivel de robótico. Este robot no solo puede lanzar agua, sino que basa su funcionamiento en el agua, posee 20 articulaciones compuestas por válvulas hidráulicas y cilindros capaces de manejar hasta 101 kg de presión de agua.

Figura 7: Anna Konda: robot apaga incendios

Estos sistemas no integran aun un sistema de localización, o al menos no se basan en él, aunque si tienen algoritmos de control avanzados.

El control PID es más utilizado en aviación, pues es la mejor opción para controlar sistemas que se mueven en 3 dimensiones. Tener vehículos no tripulados aéreos puede facilitar labores de búsqueda y rescate en condiciones extremas, acompañando el sistema de algoritmos de búsqueda y reconocimiento.

El IAI Heron, también conocido como Machatz-1 es un UAV (vehículo aéreo no tripulado) israelí, desarrollado por Malat, división de la empresa Israel Aerospace

Industries. Su modo de operar también le ha dado el calificativo de MALE (Medium Altitude Long Endurance; traducido: Altitud Media Larga Duración). Este tipo de operaciones tienen una duración aproximada de 52 horas a una altitud de 35.000 pies. Aunque ha demostrado realizar 52 horas de vuelo continuo, la duración operacional máxima del vuelo es menor, debido al esquema de vuelo y la carga del avión.

Este último sistema ya integra control PID para estabilizar el vuelo, pero necesita un piloto en tierra que lo dirija en cada momento. Los siguientes sistemas poseen un control más avanzado que les permite ser autónomos al realizar una tarea:

El INRIA, en Francia, está desarrollando vehículos con la capacidad de seguir una ruta predeterminada y resolver los problemas que puedan presentarse en el camino.

(EOL/Liliana Toledo).- El Instituto Nacional de Investigación en Informática y Automática de Francia (INRIA), está trabajando en soluciones para enfrentar las necesidades de transporte de calidad. Utiliza un control PID para controlar las diferentes velocidades que puede alcanzar un vehículo en ciudad, y podría estabilizar el sistema a una velocidad constante, facilitando el ahorro de energía.

Los denominados Cybercars son vehículos de carretera totalmente automatizados con capacidad de conducción. Requieren la localización del vehículo dentro de la ciudad y la capacidad de éste para saber dónde puede y podría moverse, considerando los obstáculos y desniveles que se encuentren en el camino. Todo ello gracias a un sistema denominado SLAMMOT, que permite localizar y detectar objetos y sus movimiento que el automóvil pueda crear un mapa en tiempo real.

La Universidad de Maryland, en College Park, EEUU, fue el escenario de la presentación del modelo AD-150, un vehículo aéreo no tripulado desarrollado por American Dynamics Flight Systems. El AD-150 utiliza alta tecnología de propulsión que le permite despegar verticalmente, así como un control PID para realizar la transición hacia vuelo horizontal, y mantener una capacidad de velocidad aérea muy alta.

Figura 8: AD-150 de American Dynamics Flight Systems

El MIT, (Boston), está desarrollando el proyecto UAV SWARM Health Management Project, cuyo objetivo es la posibilidad de ejecutar misiones de larga duración con una flota de UAVs en un entorno dinámico. Dado que la batería de cada vehículo es limitada, se han de coordinar relevándose para ir a repostar sin descuidar la misión que estén llevando a cabo. Los vehículos operan de forma autónoma, y el sistema está controlado por un ser humano.

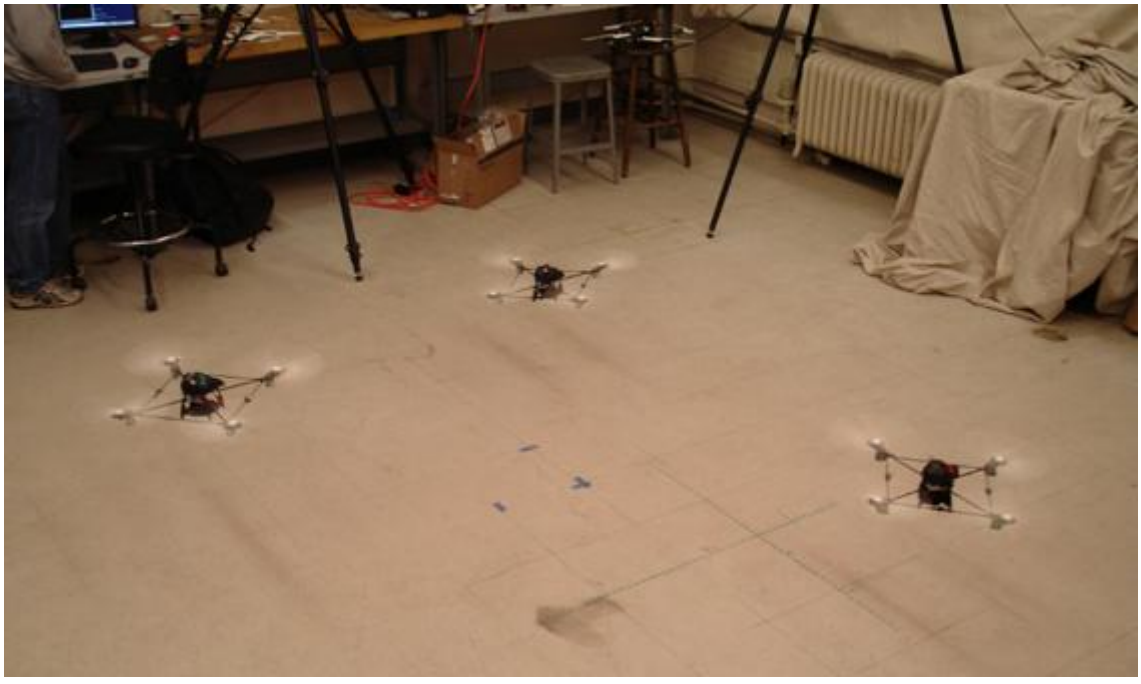


Figura 9: UAV SWARM Health Management Project del MIT de Boston

1.3.2 Otros sistemas de control

La alternativa al control PID es la lógica borrosa o difusa, basada en lo relativo a lo observado. Este tipo de lógica toma dos valores aleatorios, pero contextualizados y referidos entre sí. Así, por ejemplo, una persona que mida 2 metros es claramente una persona alta, si previamente se ha tomado el valor de persona baja y se ha establecido en 1 metro. Ambos valores están contextualizados a personas y referidos a una medida métrica lineal.

La lógica difusa se adapta mejor al mundo real en el que vivimos, e incluso puede comprender y funcionar con nuestras expresiones, del tipo "hace mucho calor", "no es muy alto", "el ritmo del corazón está un poco acelerado", etc.

La clave de esta adaptación al lenguaje, se basa en comprender los cuantificadores de nuestro lenguaje (en los ejemplos de arriba "mucho", "muy" y "un poco").

En la teoría de conjuntos difusos se definen también las operaciones de unión, intersección, diferencia, negación o complemento, y otras operaciones sobre conjuntos (ver también subconjunto difuso), en los que se basa esta lógica.

Para cada conjunto difuso, existe asociada una función de pertenencia para sus elementos, que indican en qué medida el elemento forma parte de ese conjunto difuso. Las formas de las funciones de pertenencia más típicas son trapezoidales, lineales y curvas.

Se basa en reglas heurísticas de la forma SI (antecedente) ENTONCES (consecuente), donde el antecedente y el consecuente son también conjuntos difusos, ya sea puros o resultado de operar con ellos. Sirvan como ejemplos de regla heurística para esta lógica (nótese la importancia de las palabras "muchísimo", "drásticamente", "un poco" y "levemente" para la lógica difusa):

SI hace muchísimo calor ENTONCES disminuyo drásticamente la temperatura.

SI voy a llegar un poco tarde ENTONCES aumento levemente la velocidad.

Los métodos de inferencia para esta base de reglas deben ser simples, veloces y eficaces. Los resultados de dichos métodos son un área final, fruto de un conjunto de áreas solapadas entre sí (cada área es resultado de una regla de inferencia). Para escoger una salida concreta a partir de tanta premisa difusa, el método más usado es el del centroide, en el que la salida final será el centro de gravedad del área total resultante.

Las reglas de las que dispone el motor de inferencia de un sistema difuso pueden ser formuladas por expertos, o bien aprendidas por el propio sistema, haciendo uso en este caso de redes neuronales para fortalecer las futuras tomas de decisiones.

Los datos de entrada suelen ser recogidos por sensores, que miden las variables de entrada de un sistema. El motor de inferencias se basa en chips difusos, que están aumentando exponencialmente su capacidad de procesamiento de reglas año a año.

Un esquema de funcionamiento típico para un sistema difuso podría ser de la siguiente manera:

Figura 10: Entorno físico de control

La lógica difusa se utiliza cuando la complejidad del proceso es muy alta y costaría mucho esfuerzo desarrollar un control matemático. Pero no es recomendable para sistemas que dispongan de un modelo conocido y sus resultados son eficientes.

Por tanto, para desarrollar el control de UAV será mucho más eficiente y exacto el control PID que la lógica difusa.

2 DISEÑO HARDWARE

2.1 Introducción

En la siguiente figura podemos observar la interacción del simulador y el sistema real con el controlador:

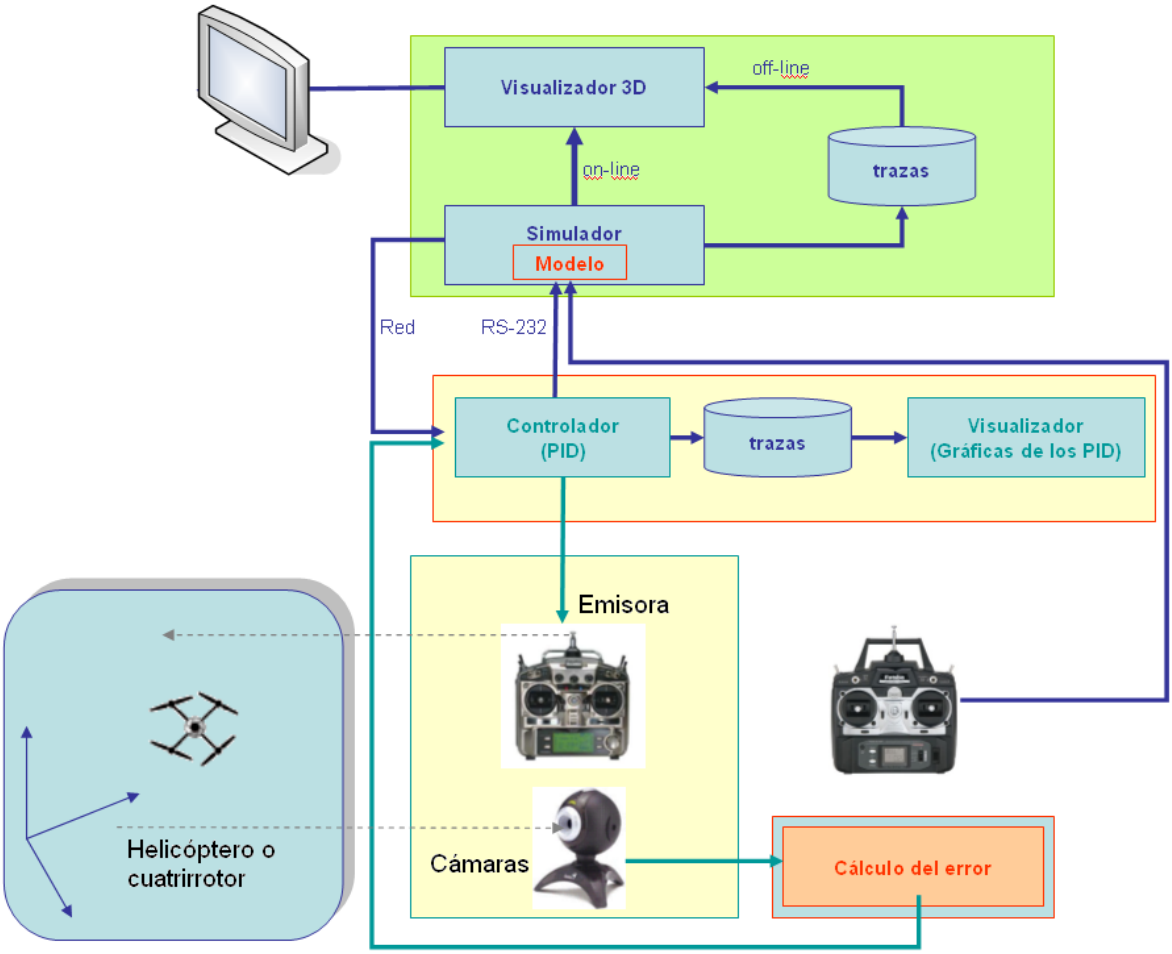


Figura 11: Interacción entre los módulos del sistema

En esta sección hablaremos del helicóptero y cómo controlarlo, (bloque inferior izquierdo en la figura), haremos un repaso acerca del control PID, cómo influye en cada uno de los canales de la emisora y por último presentaremos el entorno virtual y real en el que se han realizado las pruebas de simulación.

2.2 Helicóptero o cuatrimotor

2.2.1 Ángulos de navegación

Los ángulos de navegación son una forma de ángulos Eulerianos utilizados para movimientos y posicionamiento de objetos en el espacio. Sirven para saber la posición de un sistema móvil en un momento dado respecto del espacio con sistema de coordenadas fijo.

Se basan en describir la forma de alcanzar la posición final desde la inicial con tres rotaciones, llamadas yaw, (guiñada), pitch, (cabeceo) y roll, (alabeo) ,y el resultado final dependerá del orden en que se apliquen: primero el yaw, luego el pitch y por último el roll. En la siguiente figura se han representado sobre un helicóptero:

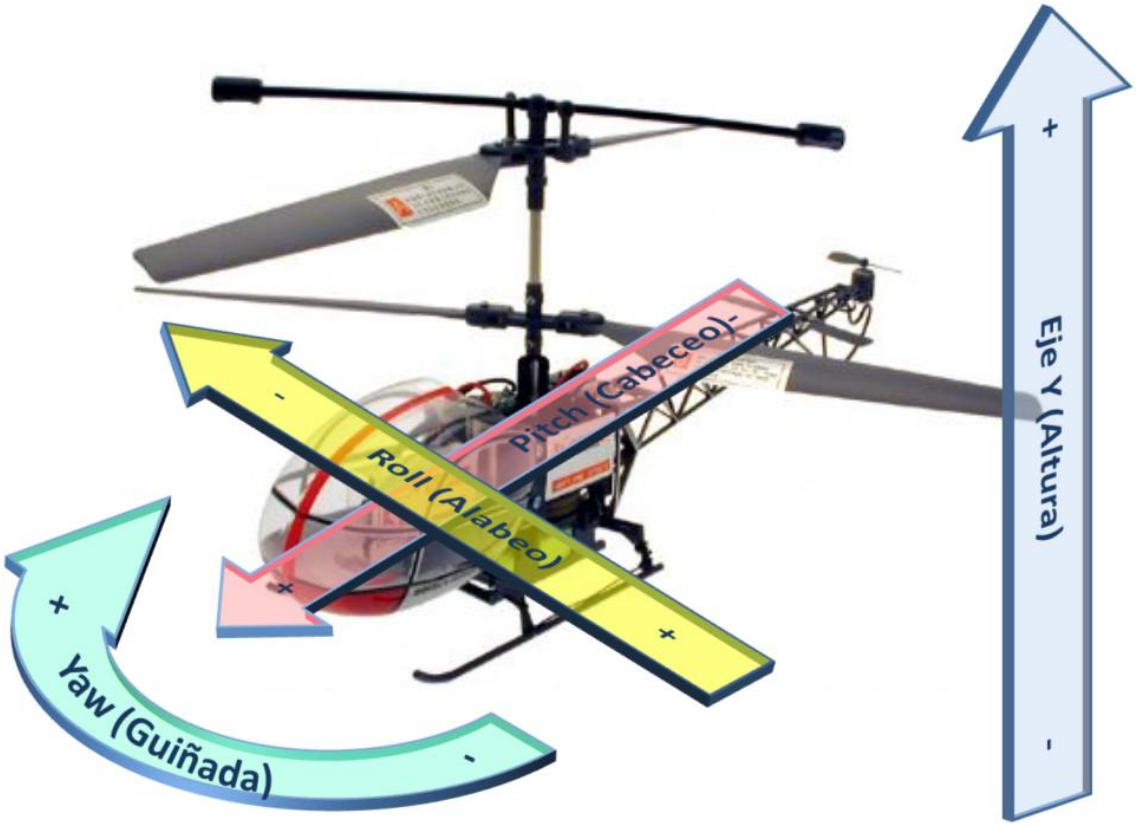


Figura 12: Ejes del helicóptero

Explicación detallada de cada ángulo:

- **Pitch** (Cabeceo): inclinación del morro del avión, o rotación respecto al eje ala-ala.
- **Roll** (Alabeo): rotación intrínseca alrededor del eje longitudinal del avión
- **Yaw** (Guiñada): movimiento del avión respecto del eje imaginario vertical que pasa por el centro de gravedad de la aeronave. Este eje es perpendicular al de cabeceo y al de balanceo, está contenido en un plano que pasa por el morro y la cola del aparato y que normalmente divide a este en dos partes simétricas.

2.2.2 Calibración del helicóptero

Se trata de un proceso necesario, porque si vamos a manejar el helicóptero mediante un sistema automático o manualmente, y no está correctamente calibrado, se vuelve inmanejable.

En primer lugar, nos hemos de asegurar que las palas del helicóptero están equilibradas. Para ello hemos comparar los pesos de cada par viendo que están compensados.

A continuación aplicaremos el proceso conocido como la maniobra de Hover:

- Revolucionaremos el motor hasta que los patines se separen ligeramente del suelo. Nos situaremos a unos 4 metros del helicóptero.
- Cuando las dos palas vayan al compás parecerá como si las puntas se solaparan visto desde un lado del rotor. Si las palas no van al compás, se ha de ajustar el varillaje que conecta con el brazo del rotor principal.
- Como el comportamiento del helicóptero aún puede ser inestable, para evitar que se balancee y caiga facialmente, habría que estabilizarlo. Estabilizarlo consiste en modificar ligeramente los valores que la emisora envía al helicóptero como punto de equilibrio del canal.

- En caso de no ser un valor adecuado para el helicóptero, la razón puede deberse a múltiples causas: el nivel de batería del helicóptero, la batería de la emisora o las interferencias que pueda haber en el entorno. Si no se corrige puede provocar balanceos no deseados del helicóptero impidiendo así un control regular.

Esta estabilización se realiza generalmente en la emisora, pero como en nuestro caso el controlador le envía directamente los valores, se puede efectuar directamente en la ventana de configuración del mismo.

Si el helicóptero se balancea hacia la izquierda o hacia la derecha usaremos el control de estabilidad de movimiento lateral para compensar. Operaremos igual con el movimiento frontal y rotacional, en caso de que el helicóptero avance o retroceda en el equilibrio, o rote sobre su eje.

Estabilizar en altura normalmente no será necesario, pero lo podemos ajustar para hacer que el helicóptero despegue con mayor rapidez o que tarde más en despegar. Si el helicóptero está bien estabilizado facilitará el proceso de control.

2.3 PID

Un PID (Proporcional Integral Derivativo) es un mecanismo de control por realimentación. Un controlador PID corrige el error entre un valor medido y el valor que se quiere obtener calculándolo y sacando una acción correctora que puede ajustar al proceso acorde.

El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error, asegurándonos que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero. El Derivativo determina la reacción del tiempo en el que el error se produce.

La suma de estas tres acciones es usada para ajustar al proceso vía un elemento de control como la posición de una válvula de control o la energía suministrada a un calentador, por ejemplo. Ajustando estas tres constantes en el algoritmo de control del PID, el controlador puede proveer un control diseñado para lo que requiera el proceso a realizar.

Esta figura muestra las ecuaciones que componen un control PID y la relación entre ellas:

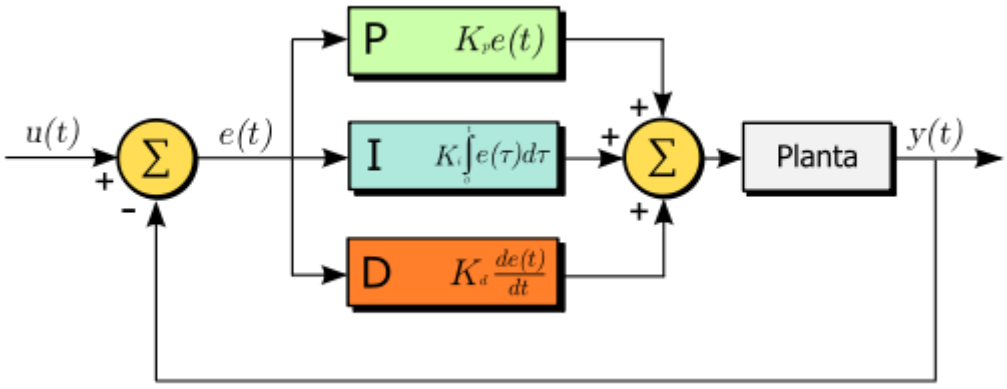


Figura 13: Diseño de un controlador PID

La siguiente figura representa la estructura del sistema del controlador bajo el punto de vista del control PID, desde la entrada al sistema generada por el software de visión, (Cámaras), hasta la salida enviada a la emisora:

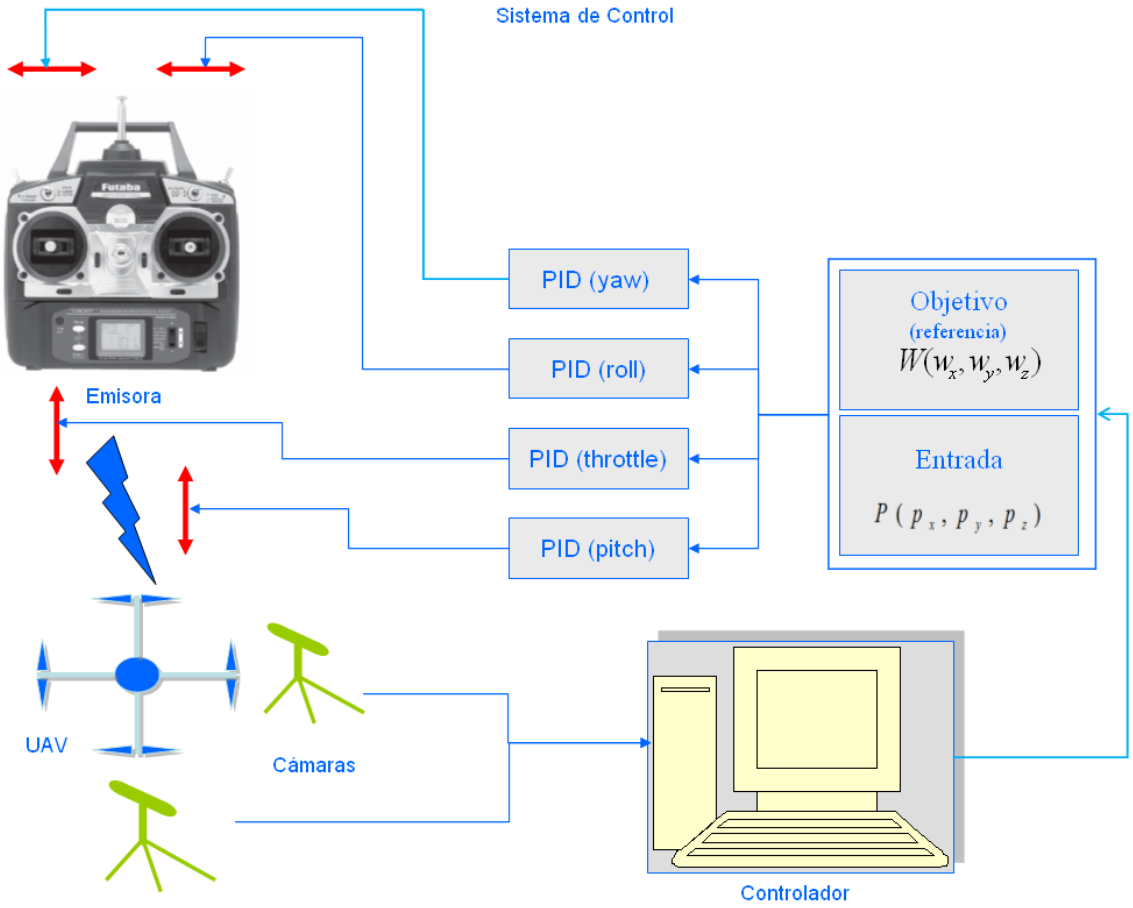


Figura 14: Sistema completo con conexión de los PID´s

El objetivo es que el UAV, situado en cierta posición, alcance la posición objetivo (o de referencia). Para ello recibe un vector de 4 bytes enviado por la red, los cuales representan la diferencia entre la posición actual del UAV, y la posición objetivo (referencia).

Una vez recibidos, cada uno de los 4 errores de posición es tratado mediante un PID, haciendo que necesitemos 4 controladores para cada uno de los grados de libertad del UAV. Los controladores PID calcularán la señal de control de cada grado de libertad necesaria para que el UAV se posicione en las coordenadas objetivo. Estas 4 señales de

control se envían por un puerto RS-232 a la emisora, que automáticamente reenviará los valores equivalentes por radiofrecuencia al UAV. Esto resultará en una nueva posición del UAV que, en caso de que siga sin ser la posición Objetivo, será corregida nuevamente siguiendo el mismo proceso; y, en caso contrario, los controladores PID reaccionarán dejando una señal estable que conseguirá que el UAV no altere su posición.

2.4 Entornos de pruebas

Como ya hemos explicado anteriormente, un helicóptero es un vehículo muy sensible a los cambios de su entorno. Cualquier mínima variación que no esté contemplada o no se corrija a tiempo puede ocasionar que choque con algún elemento y caiga.

Por lo tanto, las primeras pruebas se hicieron con el simulador, hasta conseguir un sistema fiable. Más adelante se siguió utilizando el simulador para las probar los nuevos cambios de la aplicación, y el sistema real para ir viendo el resultado obtenido.

Además se utilizaron pequeñas aplicaciones auxiliares para probar funcionalidades básicas, como el envió y recepción de datos, o la visualización de la salida por la emisora.

1.-Simulador

El simulador ofrece un espacio virtual de vuelo limitado, (de unos 500 x 500 pixeles), con una dinámica realista para el helicóptero, y la simulación de un sistema localizador que actúa como queremos. Permite hacer numerosas pruebas rápidamente debido a que con solo pulsar un botón se reiniciarán las condiciones iníciales de las que partíamos. También podemos grabar las pruebas que hagamos, para reproducirlas posteriormente y buscar los errores. Así determinaremos si el control es como se espera.

Una vez realizadas las pruebas en el simulador y comprobado su funcionamiento, se pueden comenzar las pruebas en el sistema real con mayor seguridad.

En la siguiente figura podemos observar el simulador en una de las pruebas, (en este caso viendo cómo responde a los controles de teclado):

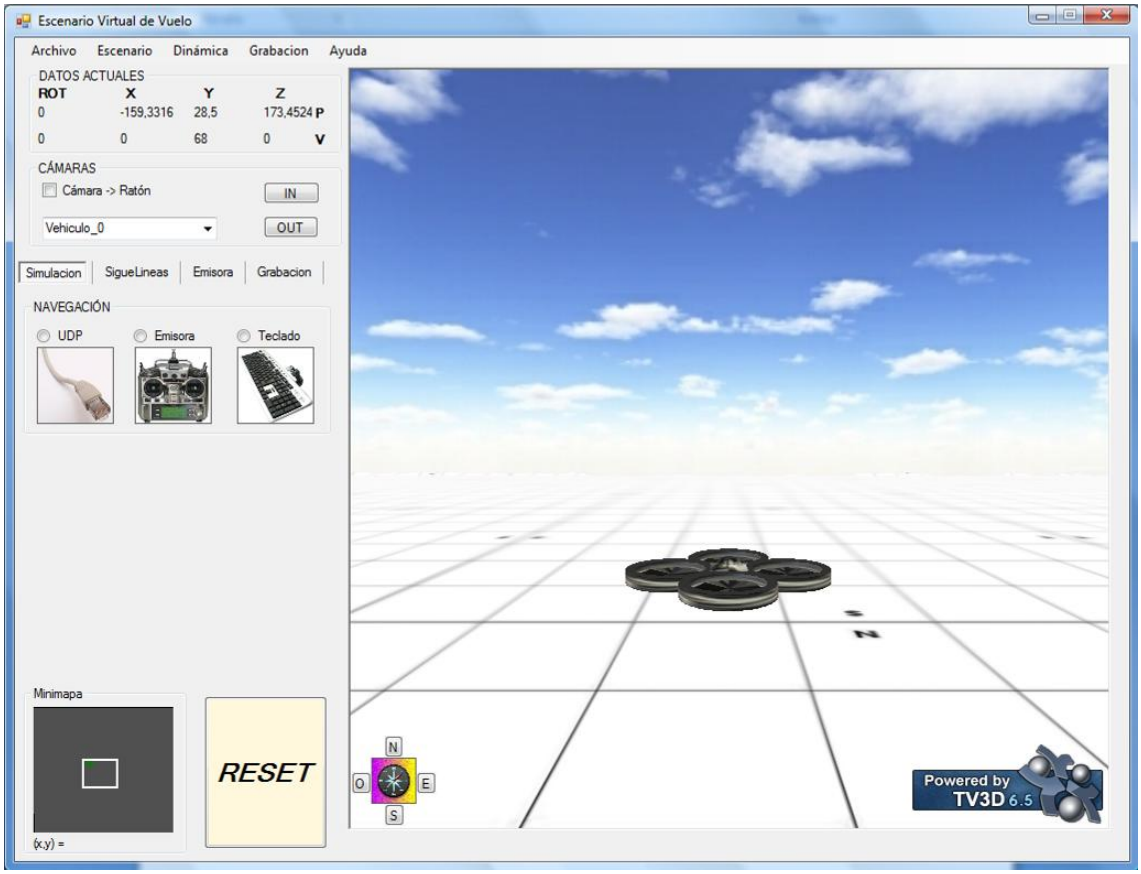


Figura 15: Prueba de movimiento en simulador

2.-Entorno del laboratorio

En este entorno se realizaran las pruebas reales de control, teniendo como objetivo principal estabilizar el helicóptero en un punto del espacio, para después moverlo. Llevará más tiempo realizar estas pruebas que las pruebas en el simulador, y los errores en el sistema real pueden ser catastróficos. Por este motivo las pruebas reales serán menos, y sólo cuando estemos seguros del sistema.

En la siguiente figura se muestra la disposición del helicóptero respecto a las cámaras y los ejes fijos:

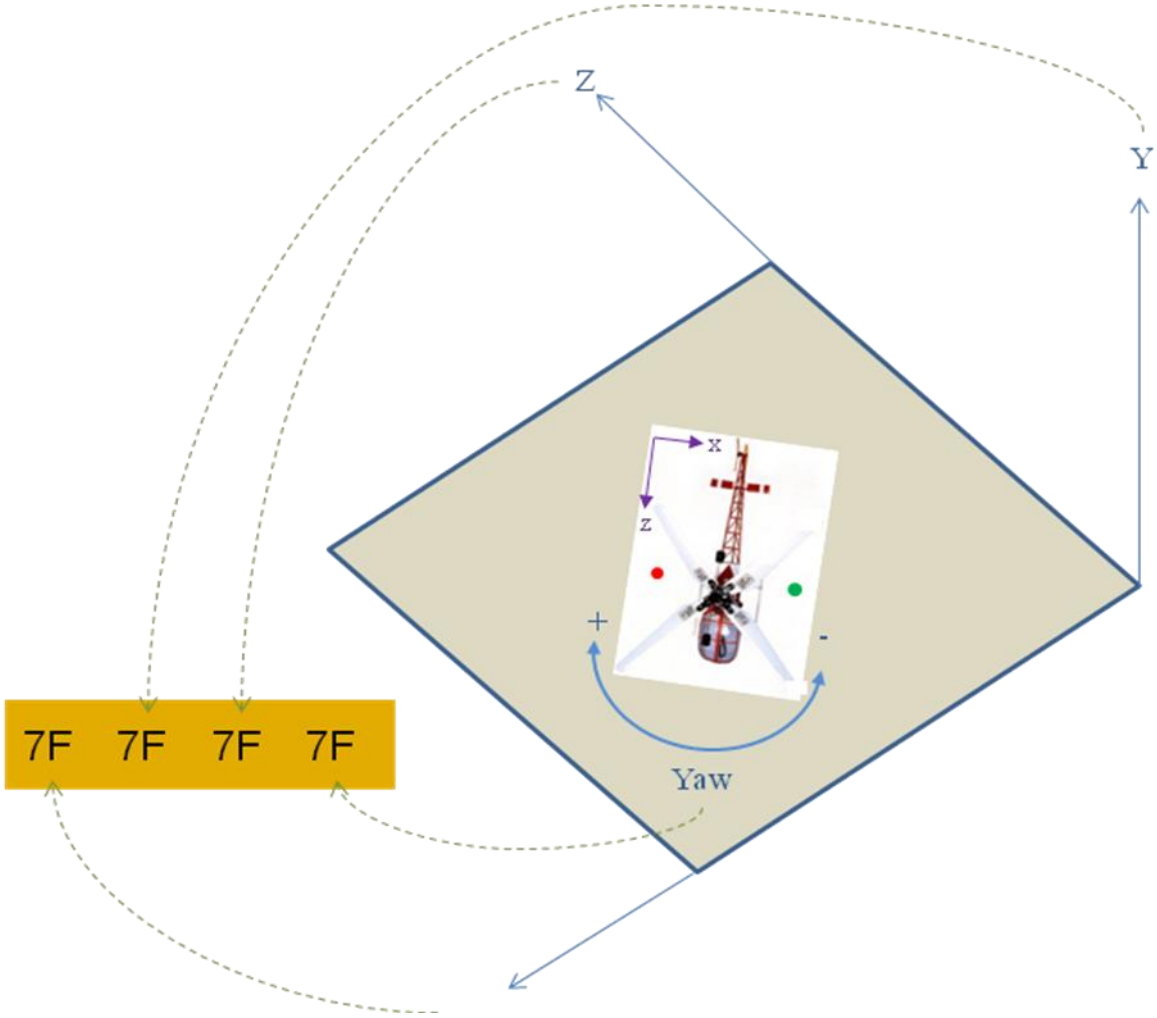


Figura 16: Captura del movimiento del helicóptero

Por esta razón, tenemos que saber interpretar los resultados que se pueden producir, ya que los ejes Y del helicóptero y los ejes fijos coinciden, pero el resto no. El

error recibido por el controlador es respecto a los ejes fijos, mientras que las modificaciones que se envían a la emisora son respecto a los propios ejes del helicóptero. Es decir, que si el helicóptero se mueve frontalmente incrementando su eje Z, el controlador recibirá un error respecto al eje X, (que se habrá incrementado), y al eje Z, (que se habrá decrementado).

Entorno de pruebas real:

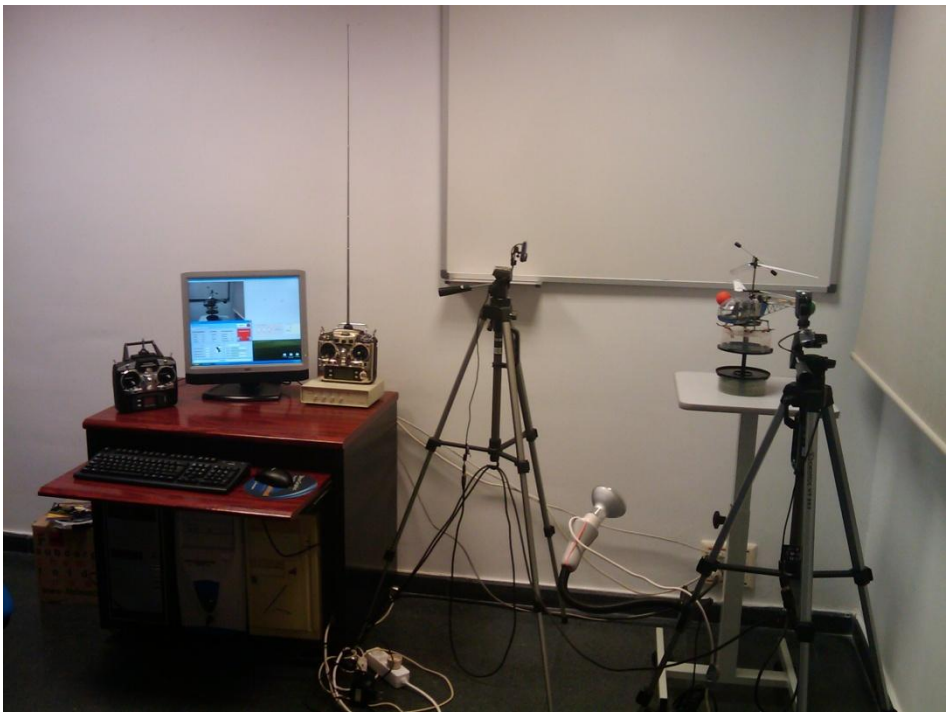


Figura 17: Laboratorio de pruebas reales

3 DISEÑO SOFTWARE

3.1 Introducción

En este capítulo presentaremos en detalle el sistema desde el punto de vista software. En la figura 18 se muestra la relación entre todos los módulos que componen el proyecto, unos desarrollados por nosotros, (como son el selector, el controlador, el generador de escenarios y el simulador), y otros preexistentes, (sistemas de visión por cámaras, helicóptero y emisora):

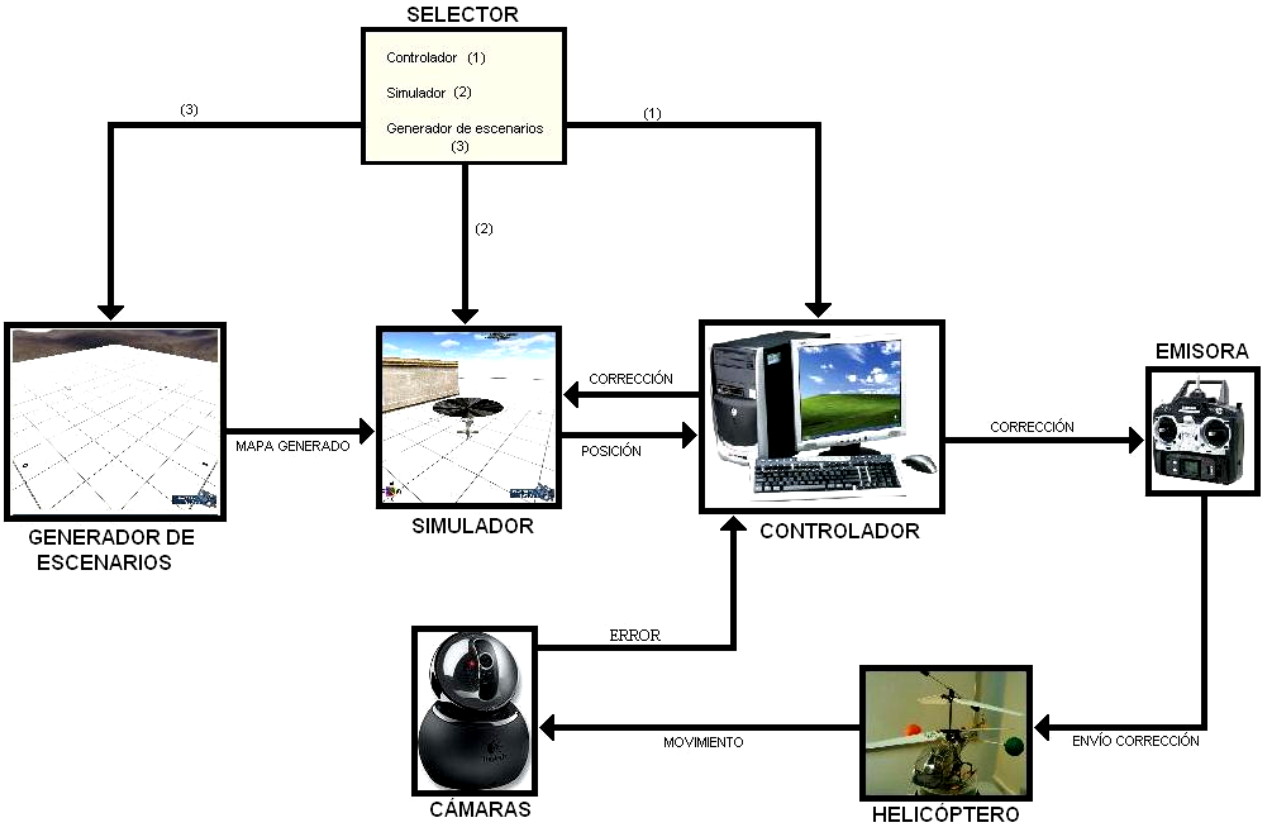


Figura 18: Módulos del sistema

El **selector** es una aplicación auxiliar que sirve para iniciar el simulador, controlador o el generador de escenarios según la elección del usuario. Si lo cerramos, cerrará cualquiera de las tres aplicaciones que tengamos abiertas y finalizará el programa.

El **generador de escenarios** es nuestro gestor de mapas para el **simulador**. Mediante éstos, el simulador realiza vuelos de los vehículos virtuales sin control o

comunicándose con el **controlador**, que a su vez puede corregir el sistema real recibiendo la entrada del programa de visualización.

A continuación detallamos la estructura y funcionamiento de cada nuestros módulos: el simulador y el generador de escenarios en la sección 3.2, y el controlador en la sección 3.3

3.2 Desarrollo del simulador

3.2.1 Motivación

Uno de los riesgos importantes del proyecto era no tener un sistema fiable de localización para el UAV, lo que limitaría la capacidad de hacer pruebas del controlador. El entorno real de trabajo, aún con un sistema fiable de localización, no es productivo a la hora de realizar pruebas, pues lleva mucho tiempo prepararlo y los resultados no son claros, (influyen condiciones externas a la aplicación que no se pueden controlar). La idea de crear un simulador de un UAV también tuvo origen en el riesgo de usar un controlador de prueba sobre el UAV real, ya que no nos podíamos arriesgar a que un fallo del software terminase en el deterioro o el mal funcionamiento del UAV que se usó para las pruebas, debemos recordar que un pequeño fallo de control, o incluso de visión en pleno vuelo del UAV, puede acabar con el UAV estrellado y estropeado.

Como previsión a este riesgo, se ha desarrollado un entorno que simule el sistema real, en el que poder controlar todas las variables que influyen en el sistema y poder realizar las pruebas de forma más eficiente. Gracias a él, el desarrollo del controlador fue mucho más rápido y efectivo permitiendo a los miembros del grupo modificar el controlador y realizar diversas pruebas en cualquier máquina.

El simulador no se limita solo a las pruebas en sustitución del sistema real, sirve también para desarrollar funciones más avanzadas, como el tratamiento de imagen o labores de coordinación entre varios UAVs.

3.2.2 Funcionamiento del simulador

Simula el comportamiento virtual del helicóptero en un entorno 3D, y visualiza por pantalla los resultados. Para ello existen dos modos de simulación, uno a través de una entrada humana, (a través de una emisora conectada mediante un puerto RS-232, o a través de entrada por teclado), y otro a través de los datos recibidos por la red desde el controlador.

Si los datos son recibidos desde el controlador, el simulador los aplica al UAV virtual generando una nueva posición, que es representada en pantalla para que la pueda

visualizar el usuario. La posición es enviada por la red al controlador, que volverá a calcular y enviar la nueva señal de control al simulador.

Para simular el comportamiento del UAV en el simulador, se hace uso de cuatro dinámicas que simulan cada uno de sus grados de libertad. Reciben la entrada por red, y generan una posición como resultado de aplicar la entrada al estado actual que tengan, es decir, la velocidad, la posición y la rotación actuales. El esquema queda por lo tanto de la siguiente forma:

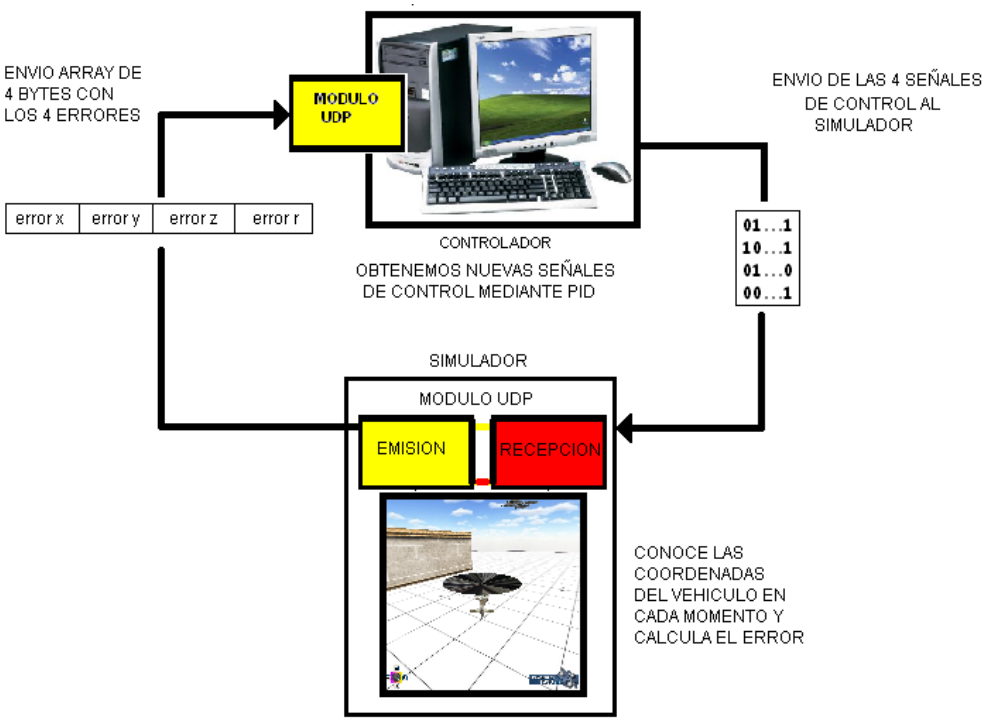


Figura 19: Esquema de conexión del simulador con el controlador

Cuando la entrada es el propio usuario, el vector de entrada al sistema no es el recibido por red, sino un vector interno de la aplicación que se irá actualizando conforme se maneje la emisora conectada, o se pulse alguna tecla del teclado destinada a ello. El programa será un bucle infinito cuya actuación consistirá en mirar si se ha pulsado alguna tecla o eje de la emisora, actualizar los vectores de entrada, y aplicárselos como entrada a la dinámica. La equivalencia entre las teclas y la emisora son:

- 7 / 9: Eje de rotación - / +
- 2 / 8: Eje frontal - / +
- 4 / 6: Eje lateral - / +
- 0 / 5: Eje de altura - / +

El esquema queda por lo tanto:

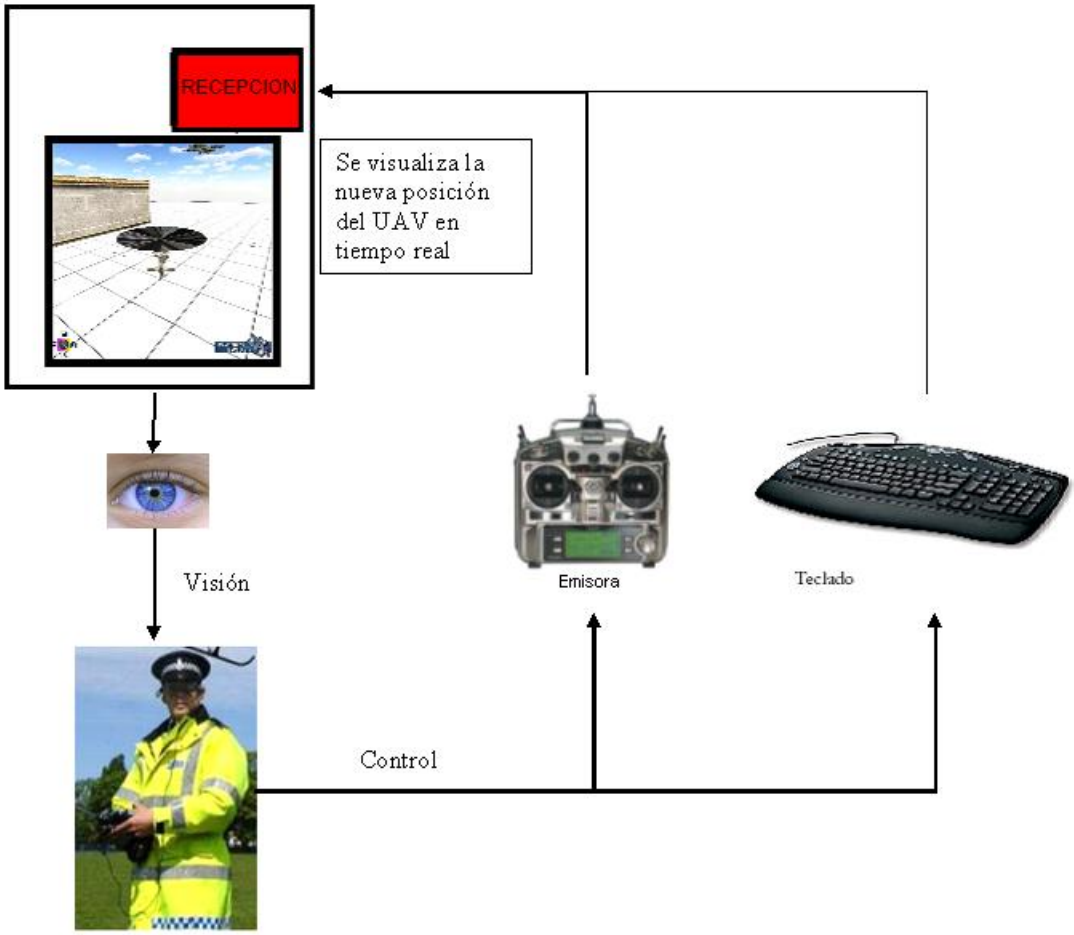


Figura 20: Esquema de captura de la posición

En las siguientes dos figuras se detalla el esquema de funcionamiento que sigue el simulador, tanto si recibe la entrada del controlador:

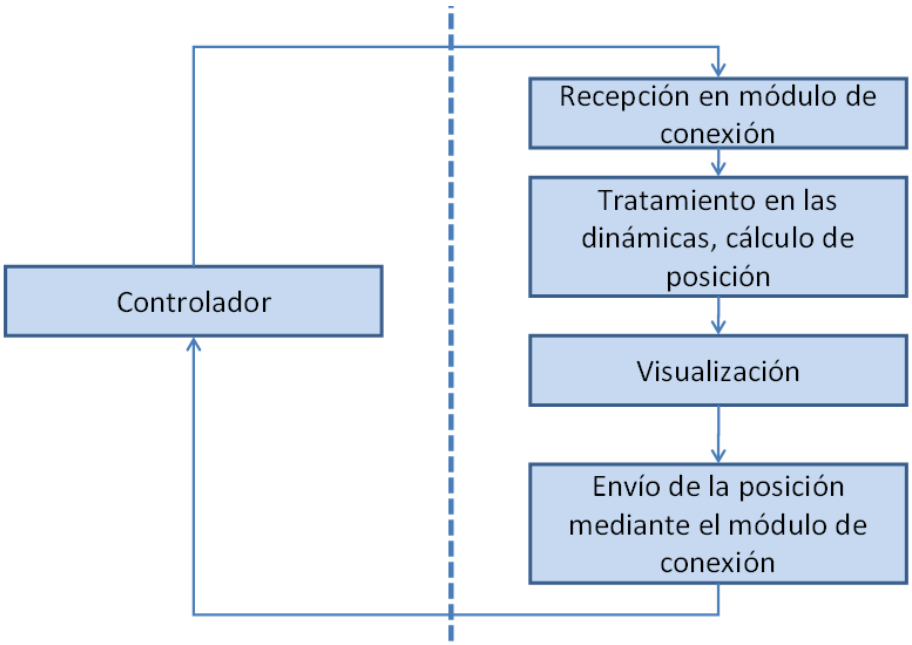


Figura 21: Esquema de secuencia del simulador funcionando con el controlador

Como si el control proviene del propio usuario, ya sea por teclado o enchufando una emisora al ordenador:

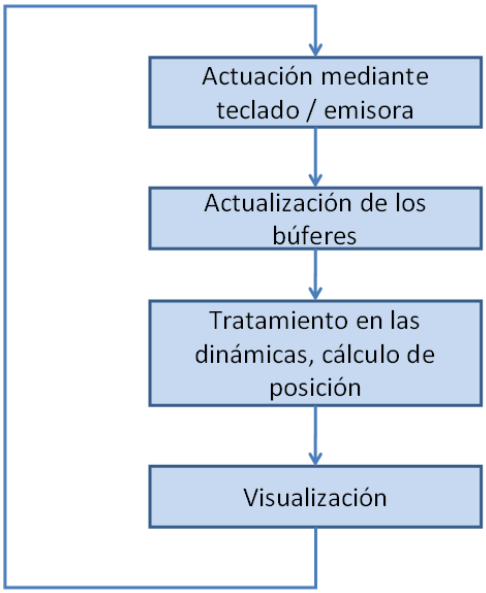


Figura 22: Esquema de secuencia del simulador funcionando con el usuario

A continuación entraremos en detalle en cada una de las fases que aparecen en las figuras, y haremos hincapié en la comunicación y configuración del simulador. También veremos cómo se configuran los escenarios, y cómo se realiza el tratamiento de imagen.

Obtención de la información

La información relativa a la dinámica en altura del UAV se obtiene mediante un fichero XML que contiene las cuatro matrices A, B, C y D que componen el sistema, así como la frecuencia de muestreo, usada posteriormente para el timer del simulador encargado de actualizar la posición según la dinámica en altura.

Para mayor comodidad, se ha creado un script en Matlab que convierte una dinámica cualquiera de Matlab en forma continua a su equivalente discreta y la guarda en el formato XML usado por el simulador, junto a la frecuencia de muestreo. El script se llama 'Planta_a_XML.m', y la forma de llamarla es la siguiente: Planta_a_XML (función de transferencia, muestreo, path de destino), por ejemplo:

- Planta_a_XML(g , 0.01 , 'C:\Users\Pepito\ejemplo.XML')

El formato del fichero XML resultante, y leído por el simulador, tiene la estructura que se puede apreciar en el siguiente ejemplo:

```
<?xml version="1.0" ?>
- <Planta>
    <Muestreo>0.01</Muestreo>
  - <A>
      <filas>2</filas>
      <columnas>2</columnas>
    - <F1>
        <C1>2</C1>
        <C2>-1</C2>
      </F1>
    - <F2>
        <C1>1</C1>
        <C2>0</C2>
      </F2>
    </A>
  - <B>
      <filas>2</filas>
      <columnas>1</columnas>
    - <F1>
        <C1>0.015625</C1>
      </F1>
    - <F2>
        <C1>0</C1>
      </F2>
    </B>
  - <C>
      <filas>1</filas>
      <columnas>2</columnas>
    - <F1>
        <C1>0.0045714</C1>
        <C2>0.0045714</C2>
      </F1>
    </C>
  - <D>
      <filas>1</filas>
      <columnas>1</columnas>
    - <F1>
        <C1>0</C1>
      </F1>
    </D>
</Planta>
```

Figura 23: Estructura XML de dinámicas del simulador

La dinámica de los otros tres ejes es una compartida, y está implementada en el propio programa debido a las dependencias que existen entre ellas. Para ello se usan algunas variables internas, como el punto medio, que se encargan de actuar en el algoritmo.

Comunicación con el controlador

La comunicación con el controlador se realiza a través del envío y recepción de paquetes UDP.

El protocolo UDP (Protocolo de datagrama de usuario) es un protocolo del nivel de transporte basado en el intercambio de datagramas. Permite el envío de datagramas a través de la red sin que se haya establecido previamente la conexión. No hay confirmación de llegada ni control de flujo, por lo que unos paquetes pueden adelantarse a otros. Tampoco se verifica si la llegada del paquete es correcta, debido a que no hay confirmación de entrega o recepción.

Sin embargo, la ventaja por la que se ha elegido este protocolo frente a otros es su velocidad. Lo que importa es que llegue la información lo más rápido posible, siendo aceptable la pérdida de algún paquete por el camino. Si hay alguna incoherencia, el controlador la tratará, pero si los datos no llegan con suficiente fluidez, no se producen las señales de control a tiempo y el helicóptero probablemente se desestabilice.

El módulo de comunicaciones incluido en el simulador escucha constantemente el puerto asignado para leer las señales de control que llegan desde el controlador. Cada vez que recibe el vector con las cuatro señales de control, el módulo de comunicaciones actualiza los búferes internos del programa para que actúen sobre el UAV virtual en uso. A su vez, envía las cuatro componentes de la posición del UAV por UDP hacia el controlador.

Dinámica

La dinámica del simulador se basa en otras dos bien diferenciadas: una dinámica independiente de las demás, dedicada a simular el movimiento en altura, y una dinámica de movimiento dedicada a simular el movimiento del UAV en su eje frontal, lateral y de rotación. Esta dinámica controla los ejes debido a la dependencia que tienen entre sí, es decir, puesto que el algoritmo utilizado por la dinámica devuelve en todo momento la posición absoluta del UAV, (sobre los ejes terrestres), el valor devuelto por ésta dependerá en cada momento de la rotación que tenga en ese momento el UAV. Por

ejemplo, si el UAV está orientado sobre el eje X, y se le aplica una fuerza en el eje frontal, éste avanzará sobre el eje X, sin cambiar su posición en el eje Z. Sin embargo, si el UAV se encuentra orientado sobre el eje Z, y se le aplica la misma fuerza que antes, ahora el UAV sólo se moverá por el eje Z, y no sobre el X anterior, y esto es sólo posible saberlo conociendo en todo momento la rotación del UAV y calculando la salida a partir de la misma.

a) Dinámica en altura

La dinámica se ha implementado respecto a la siguiente función:

$$X(t+1) = AX(t) + Bu(t)$$
$$Y(t) = CX(t) + Du(t)$$

Donde 'X' representa el estado en el que está el sistema, 'u' representa la entrada al mismo, e 'y' representa la salida del sistema. Así, conociendo las matrices A, B, C y D que identifican el sistema, podemos simularlo de manera totalmente fiable.

A través del script de Matlab 'Planta_a_XML.m', se puede pasar cualquier dinámica en continuo en Matlab a un XML con las cuatro matrices que identifican el sistema. De esta forma, cualquier dinámica que se haya probado y estudiado mediante Matlab podrá ser usada en el simulador.

El simulador accede al XML generado, y, mediante un algoritmo de lectura de XML saca las cuatro matrices y las usa en la dinámica. El algoritmo es muy sencillo, para una entrada 'u', calcula el nuevo vector de estado X (t+1) mediante cálculo de matrices, y, de la misma manera resuelve la salida Y (t) en el instante t.

Esta dinámica se actualiza constantemente mediante un timer (temporizador) con período de muestreo igual al que está definido en el XML de la dinámica, que el usuario puede elegir cuando llama al script de Matlab.

La dinámica posee una función de reinicio (o reset), a través de la cual se inicializan sus valores internos, tales como el vector de estado X, a su estado inicial.

Esta función es especialmente útil al inicializar el sistema, o al llegar el UAV a los límites del escenario.

b) Dinámica del movimiento frontal, lateral y de rotación

El funcionamiento de esta dinámica se basa en la teoría de que la posición de un objeto en el instante t+1 es la posición en el instante t más la velocidad multiplicada por el tiempo, es decir:

$$p(t+x) = p(t) + v(t) * x$$

Para simular dicha dinámica, el vector de estado del sistema se compone de las posiciones en 'x' y en 'y', así como del ángulo de rotación del UAV, y de las velocidades en cada eje, es decir, velocidad de desplazamiento frontal, velocidad de desplazamiento lateral y velocidad de rotación.

El algoritmo que sigue calcula la posición siguiente del UAV en función de la velocidad de los ejes y del ángulo de rotación. De esta manera, se puede obtener la posición en el instante t+1 mediante sencillos cálculos trigonométricos en función de la rotación del UAV. También se encarga de actualizar las velocidades de cada eje en función de la entrada, sumándole o restándole velocidad en caso de que sea positiva o negativa.

En esta dinámica también existe una función de inicialización como en la dinámica de la altura, que, igualmente, pondrá a cero todas las velocidades del UAV para simular que éste se encuentra en un estado estable de reposo.

Visualización

Para la visualización del simulador de hace uso del motor gráfico TrueVision3D. Podemos representar en el espacio todo tipo de figuras tridimensionales, así como controlar la visualización mediante cámaras, y modificar la intensidad de luz del escenario.

Para representar todo esto, el escenario se compone de algunos objetos creados desde el creador de escenarios, así como el UAV con el que se está simulando. Cada objeto tiene asignada una posición en 3D en el escenario, es decir, en X, Y y Z.; se pueden modificar cambiando los atributos internos de cada objeto. De esta manera, cambiando en cada período de muestreo la posición del objeto que representa el UAV, por los valores obtenidos de la salida de la dinámica, podemos visualizar la trayectoria que sigue el UAV en el espacio a lo largo del tiempo. Para ello es necesario refrescar la imagen cada cierto tiempo, lo que se realiza mediante un contador del programa.

Mediante este método de visualización, es importante que el UAV nunca se salga de nuestro rango de visión. Para ello, se ha incorporado un sistema de cambio dinámico de cámaras muy útil, a través del cual se puede elegir la cámara más adecuada para cada momento, dándonos la opción de usar dos cámaras pegadas al UAV (exterior o interior), o una cámara libre que se puede mover u girar mediante el uso del teclado y del ratón.

Cálculo de la entrada

El simulador posee diversos sistemas de entrada. Se puede usar recibiendo la entrada mediante el módulo de conexión UDP, para usarlo con el controlador; o puede recibir una entrada del usuario, bien desde el teclado, o bien desde una emisora real conectada a la máquina.

En el caso de usar el controlador, los valores que reciba el simulador son los que se escribirán como entrada en las dinámicas, resultando esto en una nueva posición del UAV en el espacio.

En caso de usar la emisora o el teclado, el simulador lleva un buffer interno que se encarga de almacenar el valor de entrada al sistema. En el primer caso, la emisora modifica directamente el buffer, cambiando sus valores por los equivalentes a las posiciones de sus palancas de control. En caso del teclado, el buffer se incrementa o decrementa según la tecla pulsada.

En todo caso, siempre se actúa sobre los búferes internos del programa, y éstos, a su vez, actúan como entrada a las dos dinámicas del sistema.

3.2.3 Generación de escenarios

Con el controlador funcionando con el entorno de simulación desarrollado, comienzan las pruebas en el sistema real. Sin embargo el simulador sigue siendo útil puesto que con él se pueden desarrollar nuevas funciones e investigar el comportamiento del sistema en nuevas situaciones, como el control de varios UAV a la vez, el comportamiento en un espacio con obstáculos o los algoritmos sigue-líneas.

Dentro de la idea del simulador, surgió el problema de tener un escenario de simulación que representase una realidad y unas condiciones equivalentes a las que tuviese el escenario real de las pruebas. Para ello nos vimos obligados a ofrecer al usuario una aplicación a través de la cual éste pudiese crear diversos tipos de escenarios con condiciones totalmente diferentes, y poder acceder a ellos desde el simulador; incluso poder alternar entre los diversos escenarios de manera totalmente dinámica e instantánea dentro del simulador.

Funcionamiento del generador de escenarios

Se trata de una herramienta cuya función principal es la de crear escenarios que puedan ser utilizados posteriormente por el simulador. Además permite editar creaciones anteriores y posee una vista previa que hace que todo nuevo cambio realizado sea visible inmediatamente.

Todos los mapas que se generen están en formato XML, siendo posible, incluso, editar el propio archivo e introducir los valores correspondientes a mano.

El flujo de datos sigue el siguiente esquema:

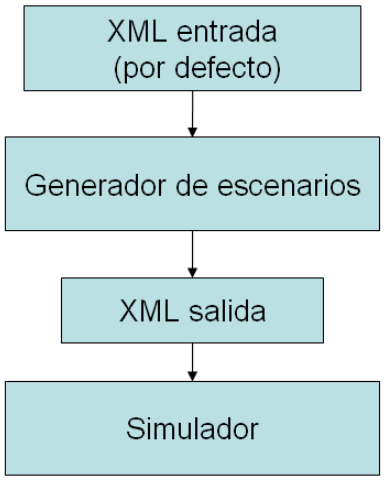


Figura 24: Ciclo de vida de un escenario

Para ver más información acerca de la implementación del generador, basta con dirigirse al anexo 8.3.4.

Configuración de un escenario

A la hora de guardar un mapa en formato XML, se usa la siguiente estructura:

- Escenario: es la raíz del fichero XML
- Texturas: nodo donde se definirán todos las texturas que van a ser cargadas por el simulador. Cada textura se define de la manera siguiente:
    - Path: ruta de donde se cargará la textura. Está referido de manera relativa a la carpeta "data" del proyecto
    - Nombre: el nombre que le daremos a la textura para trabajar con ella en el simulador o en el generador de escenarios.
- Terreno: nodo donde se guarda la información necesaria para cargar el relieve del mapa. Al igual que con las texturas, necesitamos conocer su ruta, (path), y la textura que va a tener, (por eso se cargan antes).

- Objetos: nodo donde se explican las características de todos los objetos que encontraremos en el mapa. Como debemos saber numerosos detalles, Se tiene la siguiente estructura:

    - Modelo: ruta del archivo de donde cargaremos el modelo del objeto.
    - Textura: nombre de la textura que usará el objeto. Obviamente se ha cargado anteriormente porque el fichero se lee en orden.
    - Coordenadas: posición en coordenadas x, y y z del objeto en el mapa.
    - Rotación: giro del objeto respecto a los ejes x, y y z.
    - Escala: representa el tamaño que ocupará el objeto en el escenario.
    - Tipo: aporta información acerca del tipo de objeto con el que estamos tratando: si es vehículo, pared, cámara, etc.

Cabe destacar que si el objeto es una pared, hay un campo adicional situado entre "Modelo" y "Textura", el campo "Medidas". Dicho campo específico el alto, ancho y profundo de la pared en cuestión.

A continuación se presenta un ejemplo simple de la formación de un escenario:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
- <escenario>
  - <texturas>
    - <textura>
        <path>Cielo\PuestaSol\Up.jpg</path>
        <nombre>SkyTop</nombre>
      </textura>
    - <textura>
        <path>Cielo\PuestaSol\Down.jpg</path>
        <nombre>SkyBottom</nombre>
      </textura>
    - <textura>
        <path>Cielo\PuestaSol\Left.jpg</path>
        <nombre>SkyLeft</nombre>
      </textura>
```

```
- <textura>
    <path>Cielo\PuestaSol\Right.jpg</path>
    <nombre>SkyRight</nombre>
  </textura>
- <textura>
    <path>Cielo\PuestaSol\Front.jpg</path>
    <nombre>SkyFront</nombre>
  </textura>
- <textura>
    <path>Cielo\PuestaSol\Back.jpg</path>
    <nombre>SkyBack</nombre>
  </textura>
- <textura>
    <path>Texturas\10.bmp</path>
    <nombre>pared</nombre>
  </textura>
- <textura>
    <path>Texturas\sand.jpg</path>
    <nombre>Arena</nombre>
  </textura>
- <textura>
    <path>Texturas\3.bmp</path>
    <nombre>SueloGris</nombre>
  </textura>
- <textura>
    <path>Texturas\4.bmp</path>
    <nombre>SueloArenoso</nombre>
  </textura>
- <textura>
    <path>Texturas\13.bmp</path>
    <nombre>Pavimento</nombre>
  </textura>
- <textura>
    <path>Texturas\9.bmp</path>
    <nombre>Ladrillos</nombre>
  </textura>
- <textura>
    <path>Texturas\Floor_3.bmp</path>
    <nombre>Baldosas</nombre>
  </textura>
- <textura>
    <path>Texturas\rendTexture.bmp</path>
    <nombre>Piedras</nombre>
  </textura>
- <textura>
    <path>Texturas\terrain.bmp</path>
    <nombre>Hierba</nombre>
  </textura>
- <textura>
    <path>Texturas\grass9.jpg</path>
    <nombre>Hierba2</nombre>
  </textura>
```

```
- <textura>
    <path>Texturas\dirt.jpg</path>
    <nombre>Seco</nombre>
  </textura>
- <textura>
    <path>Texturas\sun.jpg</path>
    <nombre>Sun</nombre>
  </textura>
- <textura>
    <path>Texturas\cobra.BMP</path>
    <nombre>cobra</nombre>
  </textura>
- <textura>
    <path>Texturas\suelo.jpg</path>
    <nombre>suelo</nombre>
  </textura>
- <textura>
    <path>Texturas\suelo1.JPG</path>
    <nombre>suelo1</nombre>
  </textura>
- <textura>
    <path>Texturas\chapapote.JPG</path>
    <nombre>chapapote</nombre>
  </textura>
- <textura>
    <path>Texturas\suelo2.JPG</path>
    <nombre>suelo2</nombre>
  </textura>
 </texturas>
- <terreno>
   <path>Mapas\mapa_plano.jpg</path>
   <textura>Seco</textura>
 </terreno>
- <objetos>
  - <objeto>
     <modelo>Modelos\x-ufo.x</modelo>
     <textura>cobra</textura>
    - <coordenadas>
       <x>-159,3316</x>
       <y>111,5</y>
       <z>173,4524</z>
     </coordenadas>
    - <rotacion>
       <rx>0</rx>
       <ry>270</ry>
       <rz>0</rz>
     </rotacion>
     <escala>20</escala>
     <tipo>Vehiculo</tipo>
   </objeto>
 </objetos>
</escenario>
```

Figura 25: Estructura XML de un escenario guardado

3.2.4 Módulo de tratamiento de imagen

Este módulo tiene como tarea principal implementar el comportamiento sigue-líneas del helicóptero. Se ha decidido investigar este comportamiento para explotar los usos del simulador, y porque una vez se tuviera el sistema real controlado, una de las primeras aplicaciones que tendría el sistema sería precisamente el de seguir líneas, formas o contornos, (como manchas en el mar o el tendido eléctrico).

Para su implementación, se ha optado por construir un autómata de 3 estados, que sigue el siguiente diagrama:

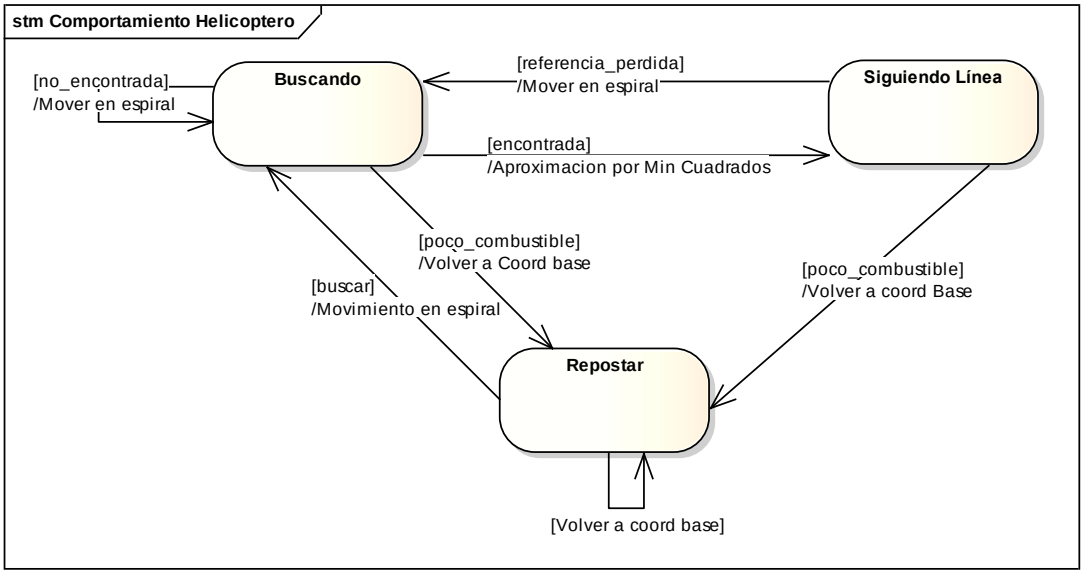


Figura 26: Autómata de funcionamiento del modulo de tratamiento de imagen

Descripción de los estados:

- **Buscando**: No se ha encontrado un punto al que dirigirse. Se procede por tanto a un movimiento en espiral hasta que se encuentre uno o el combustible del helicóptero sea insuficiente para continuar.

- **Siguiendo línea**: Hemos encontrado un punto al que dirigirnos, realizando una aproximación por mínimos cuadrados de la imagen tomada con la cámara interna del cuatrimotor.

- **Repostar**: Nos dirigiremos hacia la base a repostar.

Para explicar en detalle el funcionamiento de los 2 primeros estados, necesitaremos completar las siguientes tareas:

1. Aproximación de la recta por mínimos cuadrados:

Se recuperan los puntos de una imagen y se realiza la aproximación por mínimos cuadrados, devolviéndonos los parámetros de la recta resultante.

Aproximación por mínimos cuadrados: (Regresión lineal).

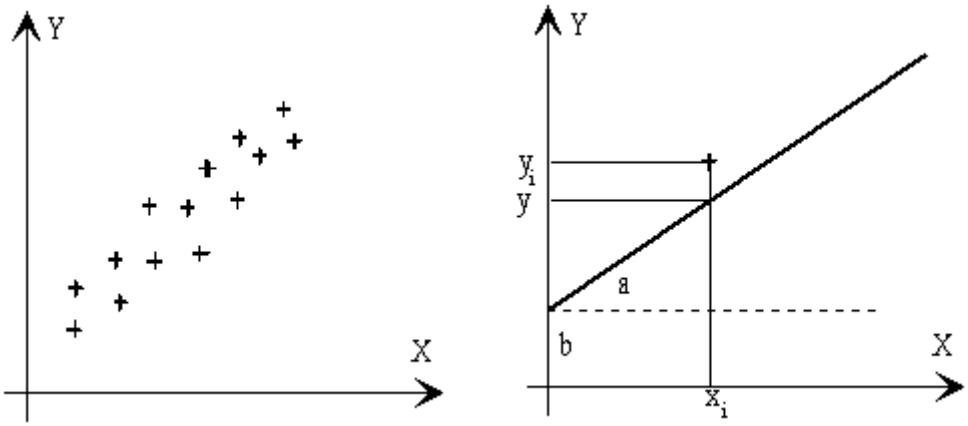


Figura 27: Calculo de trayectoria por mínimos cuadrados

Se trata de aproximar la nube de puntos que tenemos en la imagen de la izquierda mediante una recta como la que tenemos en la imagen derecha. De esta manera, si los puntos representan el borde de la figura, la recta nos indicará hacia dónde debemos dirigir el cuatrimotor a continuación.

Se denomina error e_i a la diferencia y_i-y, entre el valor observado y_i, y el valor ajustado $y= ax_i+b$, tal como se ve en la figura inferior. El criterio de ajuste se toma como aquél en el que la desviación cuadrática media sea mínima, es decir, debe de ser mínima la suma.

$$s = \sum_{1}^{N} e_i^2 = \sum_{1}^{N} (y_i - (ax_i + b))^2$$

Ecuación 1: Calculo de la desviación cuadrática media mínima

Los extremos de una función se obtienen cuando las derivadas de s respecto de a y de b sean nulas. Lo que da lugar a un sistema de dos ecuaciones con dos incógnitas del que se despeja a y b.

$$\frac{\partial s}{\partial a} = 0 \ldots\ldots a = \frac{N\sum x_i y_i + \sum x_i \sum y_i}{N\sum x_i^2 - \left(\sum x_i\right)^2}$$

$$\frac{\partial s}{\partial b} = 0 \ldots\ldots b = \frac{\sum y_i - a\sum x_i}{N}$$

Ecuación 2: Calculo de los extremos de la desviación cuadrática

La imagen que procesamos ya está tratada para que los bordes estén resaltados y sólo nos tengamos que fijar en el valor de los píxeles para realizar la aproximación. Para ello ha sido necesario binarizar la imagen, pasándola por un filtro detector de bordes.

Imagen antes y después del filtro:

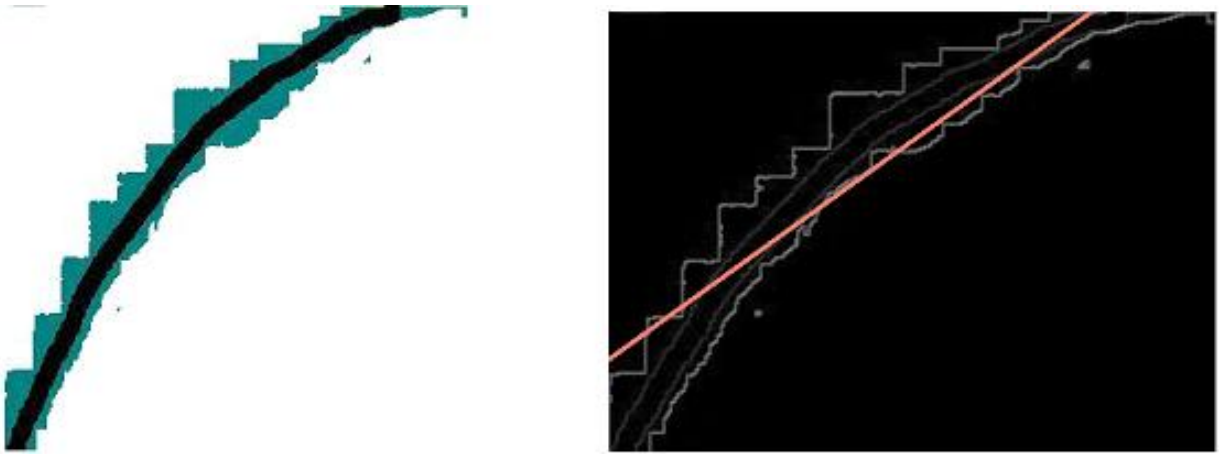


Figura 28: Imagen a reconocer y resultado obtenido, la recta color salmón es la aproximación

Para acceder a los puntos de la matriz, necesitamos un acceso eficiente a la misma, dado que vamos a tener que realizar el cálculo numerosas veces por segundo. Por ello, usamos el modo unsafe (no seguro) de C#, que nos permite acceder a los propios bits de la imagen en vez de utilizar las funciones que el lenguaje aporta por defecto, (mucho más lentas).

2. Cálculo del ángulo de giro:

Ya tenemos la recta que debemos seguir, pero ¿cuánto hemos de girar el cuatrimotor para dirigirnos a nuestro destino? Para responder esta pregunta, se calcula el ángulo de corrección de giro respecto al centro de la imagen, gracias a la recta obtenida mediante regresión lineal:

Figura 29: Esquema de cálculo del ángulo de giro

En este caso, unos 40 º, (La recta obtenida por regresión lineal es la roja, y el ángulo de 0º coincide con la línea blanca vertical).

También podemos obtener las coordenadas reales hacia donde se dirigirá el cuatrimotor a partir de las coordenadas que ocupa en el mapa, la escala del mismo y la altura a la que se encuentra.

3. Movimiento en espiral:

En caso de que el helicóptero no encuentre una línea, barrerá el área buscando una que seguir. El movimiento que utilizará será el de una espiral cuadrada, de la que podremos modificar su radio y ángulo si queremos ser más precisos.

El movimiento sería como el que se refleja en la imagen, siendo el recorrido del cuatrimotor la estela verde:

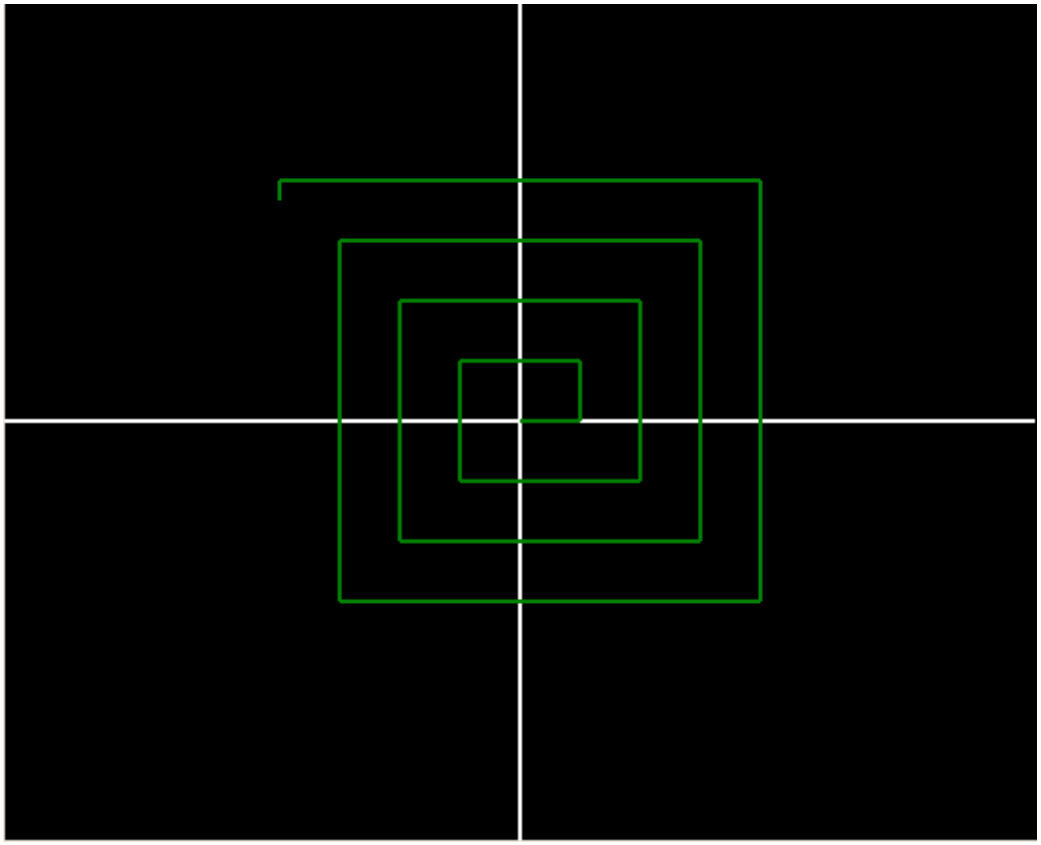


Figura 30: Trayectoria generada para el movimiento en espiral

Para ver más detalles acerca de la implementación del módulo de tratamiento de imagen, véase el anexo 8.3.3

3.2.5 Grabación de un vuelo

Hay veces que no basta con ver un vuelo una vez para descubrir las causas de una desestabilización. Por ello, se decidió añadir al simulador la opción de guardar una sesión de vuelo que grabara todo el recorrido del UAV hasta que paremos la grabación. Así se puede visualizar una sesión de vuelo y estudiar como ha sido el funcionamiento del control.

Para guardar y leer trayectorias anteriores en un escenario cualquiera, se ha optado por un fichero XML con la siguiente estructura:

- **Coord y rot:** nodo raíz del fichero, en donde anotaremos tanto las coordenadas como las rotaciones correspondientes.

- **Coordenadas:** nodo que indica el comienzo de las coordenadas. Se trata de una colección de hijos (de nombre "pos"), con la siguiente estructura:

- **x, y, z:** coordenadas donde se encuentra el objeto en este instante.

Rotaciones: nodo donde almacenaremos la rotación respecto del eje 'y' en cada instante. Se realiza de manera parecida a las coordenadas, delimitándose cada hijo por el nombre 'rot'.

Cabe destacar que se guardan tantas rotaciones como coordenadas, y que el par rotación, coordenada no se repite.

Ejemplo de una trayectoria grabada:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
- <coord_y_rot>
  - <coordenadas>
    - <pos>
        <x>-159,3316</x>
        <y>111,5</y>
        <z>173,4524</z>
      </pos>
    - <pos>
        <x>0</x>
        <y>0</y>
        <z>0</z>
      </pos>
    - <pos>
        <x>-5,735938</x>
        <y>3,045</y>
        <z>5,463751</z>
      </pos>
   </coordenadas>
  - <rotaciones>
      <rot>270</rot>
      <rot>0</rot>
      <rot>0</rot>
   </rotaciones>
 </coord_y_rot>
```

3.2.6 Diseño del simulador

Para la implementación de esta herramienta se ha intentado descentralizar al máximo la arquitectura, ofreciendo una serie de módulos en los que tenemos distinta funcionalidad. De esta manera conseguimos detectar, antes, cualquier error que se produzca y, en caso de querer introducir nuevos cambios, no nos tenemos que preocupar por interferir con otros módulos. El esquema que se sigue viene detallado en la siguiente figura:

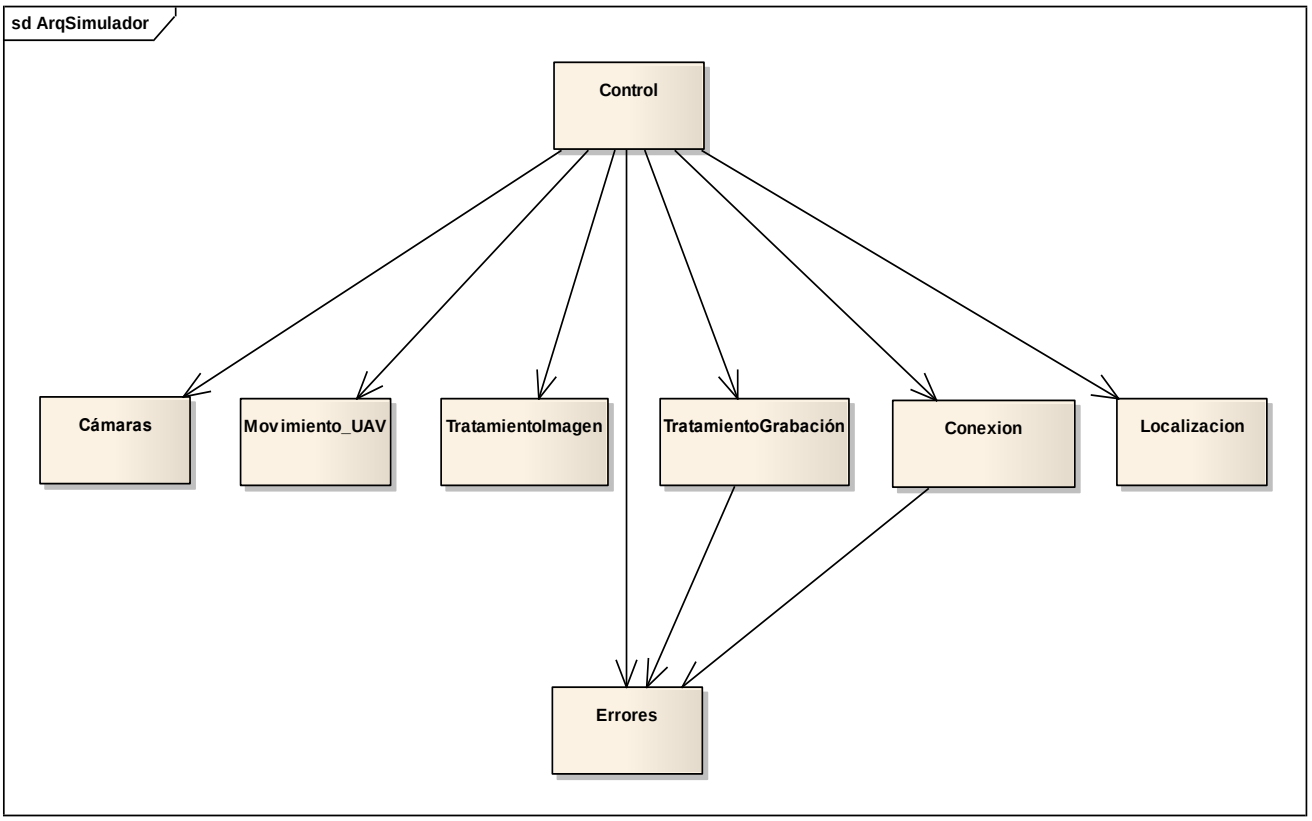


Figura 31: Esquema de la arquitectura del simulador

De la coordinación de todos los módulos se encarga el Control, que realiza la función de árbitro entre todos los módulos minimizando su interacción.

A la hora de la implementación, tal y como veremos en el diagrama de clases posterior, nuestra clase controladora se denomina Form1. De la gestión del movimiento del UAV se encargarían Planta y PlantaMovimiento, mientras que de las cámaras la responsable es la clase Piloto. El resto de las clases es fácilmente identificable con el

esquema de la arquitectura: GestorMinimapa se encarga de la localización, ModuloConexion de las conexiones, ModuloGrabación de la reproducción y grabación de los distintos movimientos del UAV y ModuloTratamientoDeImagen de los cálculos asociados al movimiento sigue-líneas del UAV. La clase GestorErrores será la encargada de almacenar y reproducir los errores que se han producido a lo largo de la ejecución.

En último lugar, cabe resaltar que la visualización implementada en el controlador hace uso de dos formularios auxiliares: el de carga y el de ayuda.

En la siguiente figura podemos ver la relación que tienen las clases entre sí:

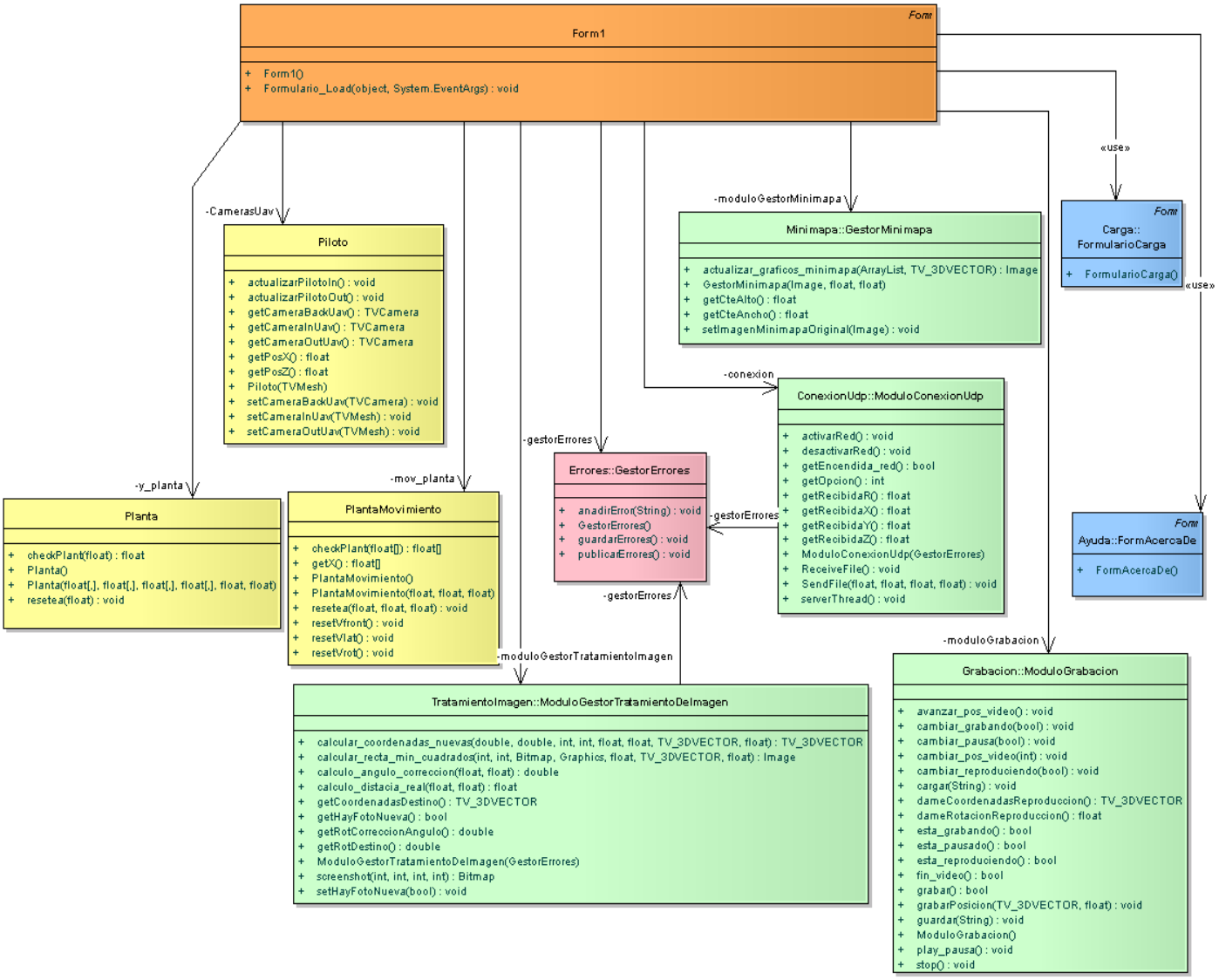


Figura 32: Diagrama de clases del simulador

Como Form1 es la clase principal y controladora, tiene referencias a todos los módulos que la componen y que aportan algo al simulador:

- **Piloto, Planta y PlantaMovimiento:** para definir cómo se mueve el UAV
- **GestorErrores:** Encargado de tratar los errores que se produzcan en cualquier módulo.
- **GestorMiniMapa:** Centrado en gestionar el minimapa de la interfaz principal

- **ModuloGestorTratamientoImagen:** Usado para el seguimiento de líneas con el UAV

- **ModuloConexiónUDP:** Necesario para comunicarse con el cliente remoto.

- **ModuloGrabación:** Dedicado a la grabación de trayectorias del UAV

Si se quiere ver una explicación más detallada de las clases que componen el simulador, véase el anexo 8.3.2.

3.3 Desarrollo del controlador

3.3.1 Funcionamiento del controlador

La figura 29 muestra el esquema del funcionamiento del controlador conectado a la aplicación de posicionamiento, controlando el UAV real a través de la emisora. La aplicación de posicionamiento envía al controlador un vector de 4 bytes representando el error en cada posición del UAV. El controlador genera las señales necesarias para llevar al UAV a su punto de equilibrio, y las copia a la emisora por puerto serie RS-232. La emisora envía al UAV las señales mediante radiofrecuencia, y recoge posteriormente el vector de 4 bytes que indica el error respecto al punto de equilibrio.

El envío de los bytes se realiza, al igual que en el simulador, mediante el protocolo UDP (User Datagram Protocol).

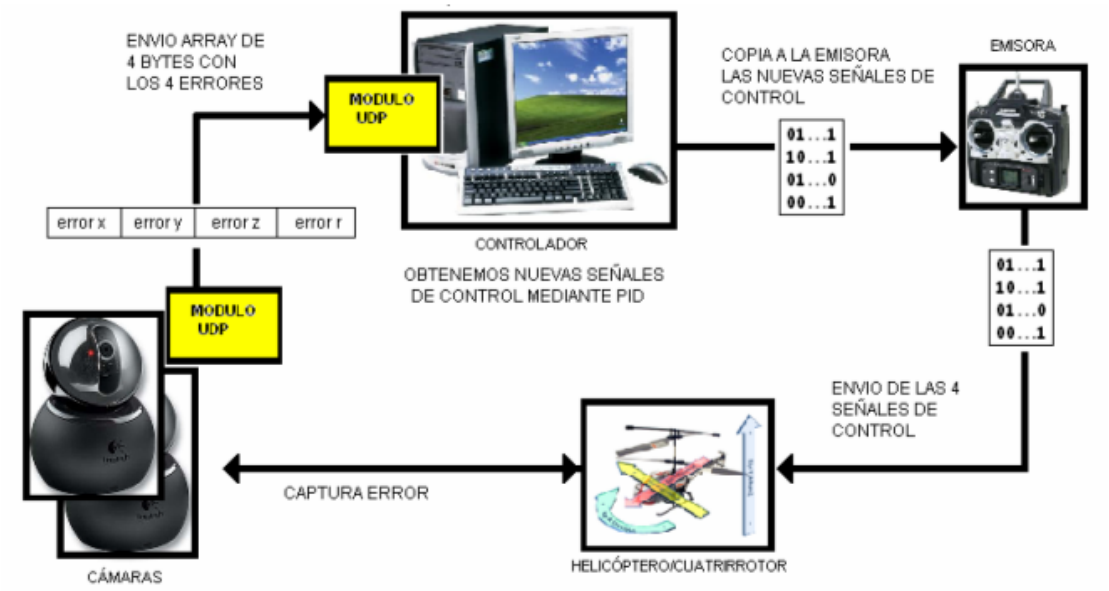


Figura 33: Esquema de funcionamiento del controlador y camino de los bytes

En la figura 29 podemos ver el esquema del funcionamiento del controlador conectado al simulador. Desde el simulador llega al controlador un vector de 4 bytes mediante un paquete UDP, indicando, en este caso, no el error respecto al punto de equilibrio, sino la posición absoluta del UAV en el simulador. Como el controlador es el que fija la posición de punto de equilibrio, calculará la señal de control necesaria para llevar al UAV a ésta, y las enviará mediante un paquete UDP al simulador. Éste se

encargará de representar de forma virtual la reacción del UAV, la mostrará por pantalla. Asimismo, enviará de nuevo un paquete UDP con la nueva posición al controlador.

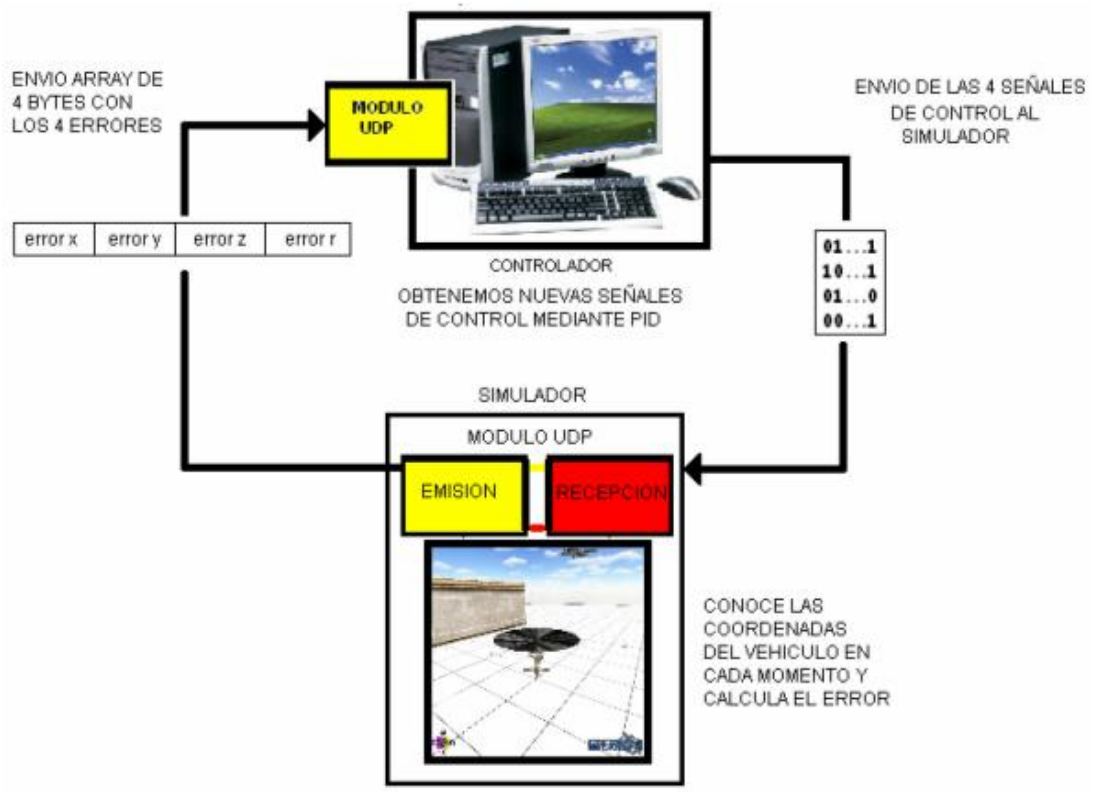


Figura 34: Funcionamiento del controlador comunicándose con el simulador

Tanto si se recibe la entrada del sistema de posicionamiento real como del simulador, el esquema de funcionamiento seguido es el de la siguiente figura:

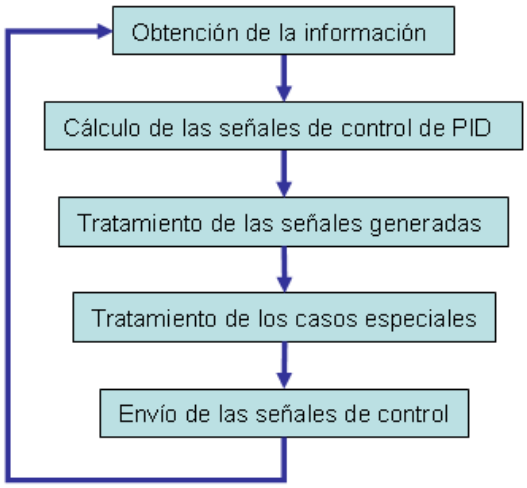


Figura 35: Diagrama de secuencia de funcionamiento del controlador

3.3.1.1 Obtención de la información

El primer paso para controlar el sistema es obtener los datos necesarios para configurar el controlador, y mantener una comunicación estable con el sistema real o simulado.

Al abrir la aplicación, ésta cargara de un archivo XML la configuración de los parámetros del PID, así como los valores medios de los cuatro canales y el modo en que estos funcionan, normales o invertidos. Siempre cargará un archivo de configuración por defecto, dejando al usuario la posibilidad de cargar otro diferente una vez iniciada la aplicación.

Al iniciar el sistema de control, se establece una conexión mediante un módulo UDP integrado en el controlador que realiza la comunicación entre el controlador y el sistema real o simulado, y se mantendrá durante toda la ejecución del programa.

El controlador hace uso de esta conexión para recibir en cada ciclo el error del UAV que está intentando corregir, utilizándolo para el cálculo de nuevas señales de control.

3.3.1.2 Cálculo de las señales de control de PID

Utilizamos la función CheckPID, perteneciente a la clase PID. Actúa de la siguiente manera:

- Cálculo del error acumulado total

```
integral += error;
```

- Cálculo de cuánto se ha reducido el error entre la posicion anterior y la actual

```
derivative = error - lasterror;
lasterror = error;
```

- Obtención de la señal de control mediante los parámetros de configuración del PID y los valores de error calculados, de la siguiente forma

```
resultado = ((P * error) + (I * integral) + (D * derivative));
```

Existen cuatro objetos de la clase PID diferentes asociados a cada uno de los grados de libertad del helicóptero, diferenciados por sus parámetros de configuración. Se realiza una llamada a la función CheckPid por cada uno de los errores recibidos, con lo que se obtienen las cuatro señales de control para corregir la posición del UAV.

Cada objeto de la clase PID está configurado con los valores P, I y D cargados mediante XML al abrir la aplicación y serán modificables desde la interfaz de configuración. Además de estos tres parámetros, el PID contendrá dos variables internas que son integral, que guarda el error acumulado por la ejecución y lasterror, que contiene el último error recibido.

3.3.1.3 Tratamiento de las señales generadas

De las señales obtenidas sólo podemos afirmar que están entre los valores mínimos y máximo de la emisora (normalmente entre 40 y 255), por lo que será necesario comprobar que sean válidas.

Una vez generadas las señales de control, comparamos éstas con las correspondientes señales del ciclo anterior, con el fin de evitar picos muy rápidos que puedan descontrolar el sistema o dañar la emisora. Si el error entre una señal del ciclo anterior y una nueva señal es mayor que un rango de seguridad constante, la nueva señal se sustituirá por la señal anterior más el rango de seguridad en dirección hacia l nuevo valor. De esta forma se consigue que la curva de fuerza de la señal no tenga una inclinación excesiva, haciendo el control más estable.

Tratamiento en casos especiales

Existen situaciones especiales en las cuales el control PID será inhibido por una rutina que generara valores de control predefinidos. Estos casos serán los siguientes:

- **Rutina de despegue**: Tiene lugar al iniciar el sistema, las señales de control valdrán el valor medio de la emisora para los desplazamientos horizontales así como para la rotación. En el caso de la altura, será una progresión de valores que irán aumentando en una medida igual al rango de seguridad desde el mínimo del canal que controla el

movimiento vertical, hasta el valor medio, en que se supone ya está el helicóptero en el aire.

- **Rutina de aterrizaje**: Ocurre al cerrar el sistema, las señales de control valdrán el valor medio de la emisora para los desplazamientos horizontales así como para la rotación. En el caso de la altura, se irá reduciendo el valor de la última señal de control válida en una medida igual al rango de seguridad con cada ciclo de reloj hasta llegar al valor mínimo del canal, en el cual se detendrá el helicóptero.

- **Abortar control**: Igual a la rutina de despegue, esta situación puede ocurrir bien por el usuario, si ve que el sistema no se comporta correctamente y pulsa el botón "Abortar", o bien por el mismo controlador, si detecta alguna situación de error.

3.3.1.4 Envío de la señal de control

Una vez llegados a este punto, ya tenemos las señales de control libres de errores y nos disponemos a enviárselas al sistema receptor.

En caso de estar trabajando con el simulador, se compondrá un vector de bytes con las cuatro señales de control, colocadas de la forma establecida en la que el simulador pueda reconocerlas (primero byte X, segundo byte Y, tercero byte Z y cuarto Byte Yaw). El envío se realizara mediante el módulo de conexión UDP con el método send.

Por otro lado, si trabajamos con el sistema real, el envió se realiza a través de la emisora. Las señales de control sustituyen el valor de sus mandos. La emisora se conecta mediante un PIC al puerto de comunicaciones rs232, y el envío desde el controlador consiste en escribir en este puerto las 4 señales de control en el orden establecido por el PIC para los cuatro canales.

Tras la escritura, el controlador lee del puerto el valor de 'eco', que son los valores que efectivamente se han copiado en la emisora. En el caso de que el eco no coincida en algún canal con el valor escrito, se identificara el error y saltara la rutina de abortar.

3.3.2 Ciclo de transformación de la información

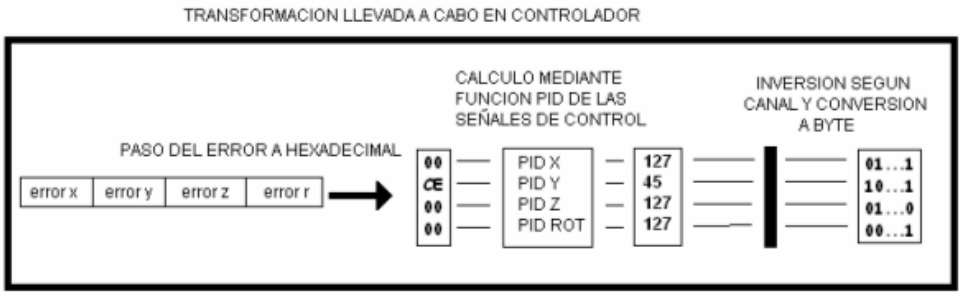


Figura 36: ciclo de tratamiento del error

En el apartado siguiente se explica de forma detallada los pasos que sigue el controlador para obtener las señales de error, tal y como se muestra en la figura 36.

Al comienzo del ciclo el controlador lee los datos que envía el sistema localizador real, o simulado, mediante el método receiveBytes del módulo UDP.

Del vector de bytes recibido, el controlador identifica los 4 primeros como las cuatro señales de error, descarta el resto del vector y transforma la parte resultante en 4 errores diferentes, previa conversión a hexadecimal.

En el caso de estar trabajando con el simulador, la señal de error será la posición absoluta del UAV en el espacio, y conocido el setPoint, el controlador calculara el error.

Cada uno de los cuatro errores es tratado mediante la llamada al método checkPID de su PID asociado, que aplica el algoritmo PID diseñado. El resultado de esta conversión son las cuatro señales de control de tipo entero necesarias para dirigir el UAV hacia las coordenadas objetivo.

A continuación, éstas señales son modificadas para conferir fiabilidad al sistema, en caso de que pudieran provocar situaciones de riesgo. También se modifican en función de cómo funcionan los canales de la emisora, es decir, si están invertidos o tienen algún valor medio diferente al común.

En el último paso, tenemos dos situaciones diferentes, el caso de trabajar con el simulador, y el caso de trabajar con el sistema real.

Si el receptor de las señales es el simulador, el controlador transforma las señales en un vector de bytes ocupando las primeras cuatro posiciones y se envían a través del módulo UDP mediante el método send(envío).

Si por el contrario el receptor de las señales es el sistema real, el controlador las convierte a bytes independientes y los escribe en el puerto rs-232, donde la emisora los identifica, por lo que deberán escribirse en un orden determinado.

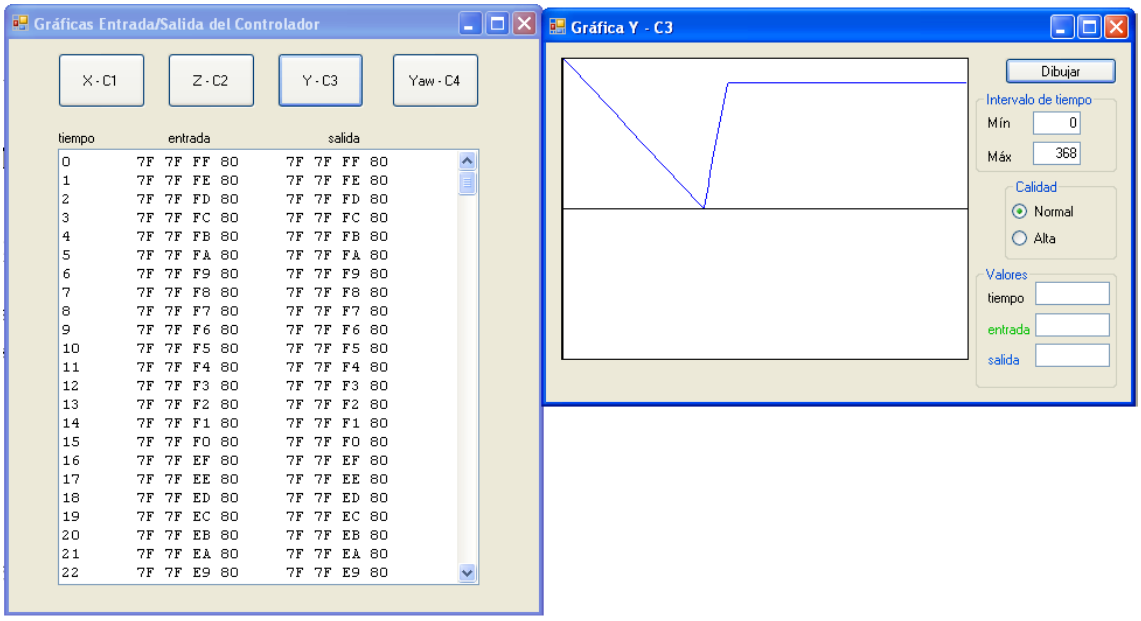
Una vez llegado a este punto el controlador está listo para tratar las nuevas señales de error.

3.3.3 Grabación de los resultados obtenidos

Para poder ver y analizar los resultados del control en cada simulación, se guardan en un fichero el instante, la señal de referencia y la señal emitida por el controlador cada ciclo en cada uno de sus cuatro canales. El fichero resultante tiene la siguiente forma:

```
0       7F 7F 7F 7F     80 7F FF 80
1       7F 7F 7F 7F     80 7F FE 80
2       7F 7F 7F 7F     80 7F FD 80
3       7F 7F 7F 7F     80 7F FC 80
4       7F 7F 7F 7F     80 7F FB 80
5       7F 7F 7F 7F     80 7F FA 80
6       7F 7F 7F 7F     80 7F F9 80
7       7F 7F 7F 7F     80 7F F8 80
8       7F 7F 7F 7F     80 7F F7 80
9       7F 7F 7F 7F     80 7F F6 80
10      7F 7F 7F 7F     80 7F F5 80
11      7F 7F 7F 7F     80 7F F4 80
12      7F 7F 7F 7F     80 7F F3 80
13      7F 7F 7F 7F     80 7F F2 80
14      7F 7F 7F 7F     80 7F F1 80
15      7F 7F 7F 7F     80 7F F0 80
16      7F 7F 7F 7F     80 7F EF 80
17      7F 7F 7F 7F     80 7F EE 80
18      7F 7F 7F 7F     80 7F ED 80
19      7F 7F 7F 7F     80 7F EC 80
20      7F 7F 7F 7F     80 7F EB 80
21      7F 7F 7F 7F     80 7F EA 80
22      7F 7F 7F 7F     80 7F E9 80
23      7F 7F 7F 7F     80 7F E8 80
24      7F 7F 7F 7F     80 7F E7 80
25      7F 7F 7F 7F     80 7F E6 80
26      7F 7F 7F 7F     80 7F E5 80
27      7F 7F 7F 7F     80 7F E4 80
28      7F 7F 7F 7F     80 7F E3 80
29      7F 7F 7F 7F     80 7F E2 80
30      7F 7F 7F 7F     80 7F E1 80
```

El instante, (primera columna en la figura), se codifica en formato decimal, mientras que la salida corregida, (columnas cuatro a 7), y la señal de referencia o error (columnas 8 a 11), en formato hexadecimal. Gracias a un sencillo programa visualizador, podemos ver las gráficas de cada uno de los parámetros del UAV:



3.3.4 Configuración XML

La configuración del controlador se realiza cargando datos de un archivo XML. En el inicio de la sesión, se carga un XML por defecto, llamado default, el cual contiene los datos del entorno normal de trabajo, y una vez iniciada la sesión se puede cargar cualquier otro XML con un formato que corresponda al de configuración que contenga nuevos datos.

La información que contiene el XML será la siguiente:

- Variables P, I D para los cuatro controladores PID.
- Una variable booleana para cada canal que informa si el canal esta invertido o no.
- El valor medio ‘trimado’(estabilizado) de cada canal.

A continuación podemos ver un ejemplo de un archivo XML de configuración:

```
<?xml version="1.0" encoding="utf-8" standalone="no" ?>
- <Config>
  - <PID>
    - <Frontal>
        <P>1</P>
        <I>0</I>
        <D>0</D>
      </Frontal>
    - <Altura>
        <P>1</P>
        <I>0</I>
        <D>0</D>
      </Altura>
    - <Lateral>
        <P>1</P>
        <I>0</I>
        <D>0</D>
      </Lateral>
    - <Rotacional>
        <P>1</P>
        <I>0</I>
        <D>0</D>
      </Rotacional>
  </PID>
  - <INVERSION_EMISORA>
      <Invertir_Frontal>False</Invertir_Frontal>
      <Invertir_Lateral>False</Invertir_Lateral>
      <Invertir_Altura>False</Invertir_Altura>
      <Invertir_Rotacion>True</Invertir_Rotacion>
    </INVERSION_EMISORA>
  - <TRIM_EMISORA>
      <Medio_X>127</Medio_X>
      <Medio_Y>127</Medio_Y>
      <Medio_Z>127</Medio_Z>
      <Medio_R>127</Medio_R>
    </TRIM_EMISORA>
</Config>
```

3.3.5 Conexión UDP

Para transmitir los datos a través de la red y comunicar el controlador con el sistema de posicionamiento mediante cámaras o el simulador, se usa el protocolo UDP.

La información enviada son 4 bytes con los valores que se han de transmitir a cada uno de los cuatro canales de la emisora.

Al pulsar el botón "Activar la red" de la interfaz del controlador se establece un puerto de entrada, (puerto 11000), por el que recibirá la señal desde el sistema de localización, y un puerto de salida, (puerto 11001), si se está usando el simulador.

Para que el sistema funcione en tiempo real, sin perder el control del helicóptero, la lectura del error y escritura de las señales de control se realiza mediante un temporizador que actúa cada 20ms.

3.3.6 Arquitectura del controlador

Hemos utilizado un modelo vista controlador, desacoplando los elementos del formulario y tratándolos en la clase de control, que es la que actúa como intermediario de las demás.

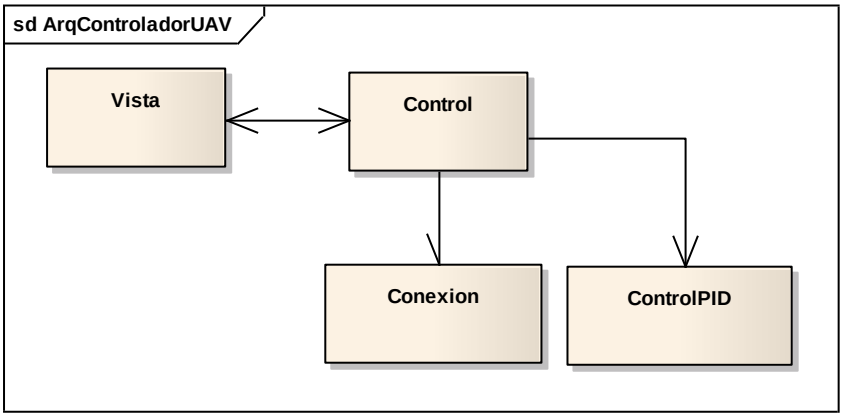


Figura 37: Arquitectura del controlador

El módulo de conexión es el encargado de recibir por la red los cambios producidos en el UAV, y transmitírselos al control para que tome las decisiones convenientes.

El Módulo de Control PID se encarga de calcular las señales que se envían al UAV a medida que se van recibiendo los datos, mientras que la Vista visualiza todas las señales emitidas y recibidas.

En cuanto a la implementación de cada módulo, tanto la vista, como el control y el módulo de conexión están compuestos por una sola clase, (incluso reutilizada del simulador en el caso del módulo de conexión).

El control PID también está formado por 2 clases. Una que se encarga del cálculo de las señales de control en caso de que hayamos seleccionado la opción de recibir de las cámaras y otra que se encarga en caso de que queramos usar el simulador. De esta manera no se mezclan métodos comunes y se facilitará el seguimiento de las trazas.

En la siguiente figura podemos ver un esquema de la arquitectura:

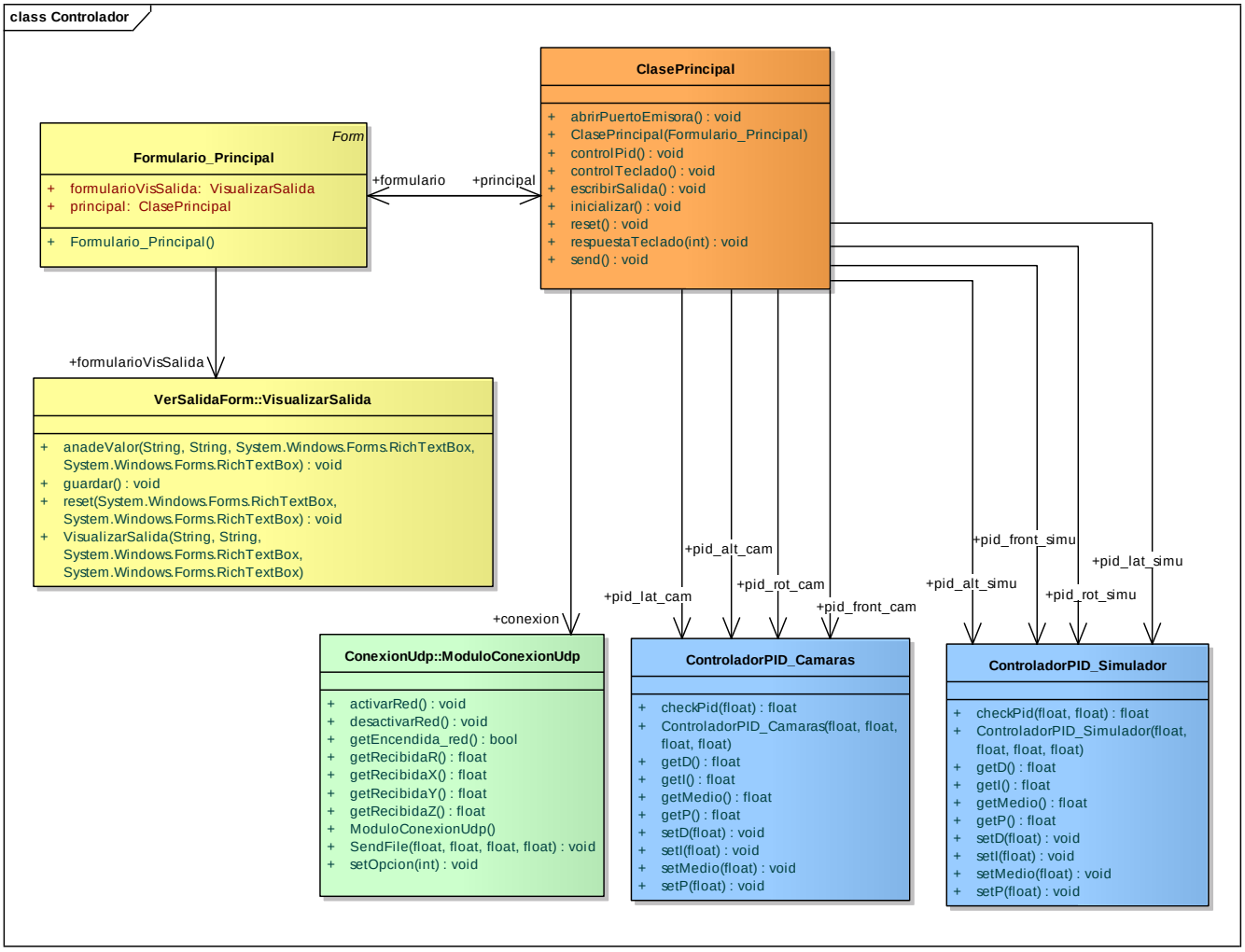


Figura 38: Diagrama de clases del controlador

La clase raíz de nuestra aplicación es ClasePrincipal. Posee las siguientes dependencias:

- **FormularioPrincipal:** se encargan de la visión de la aplicación. Muestran todos los resultados que se van obteniendo en la ejecución y notifica a la clase principal acerca de las acciones del usuario sobre la interfaz, (si ha introducido datos nuevos, si ha tocado algún botón, etc.).

- **ModuloConexionUDP**: se encarga de las comunicaciones con los programas de las cámaras o el simulador.

- **ControladorPID_Cámaras y ControladorPID_Simulador**: se encargan de generar las señales de control adecuadas según estemos realizando el control con un helicóptero de verdad o con uno virtual. Hay cuatro de cada tipo, uno por cada parámetro del UAV.

Para ver información más detallada acerca de los métodos de las clases, véase el anexo 8.3.1.

3.4 Inicialización y terminación del sistema

A continuación describiremos cómo se inicia y termina el controlador y el simulador:

Inicio del controlador: Viene detallado en el siguiente diagrama de secuencia:

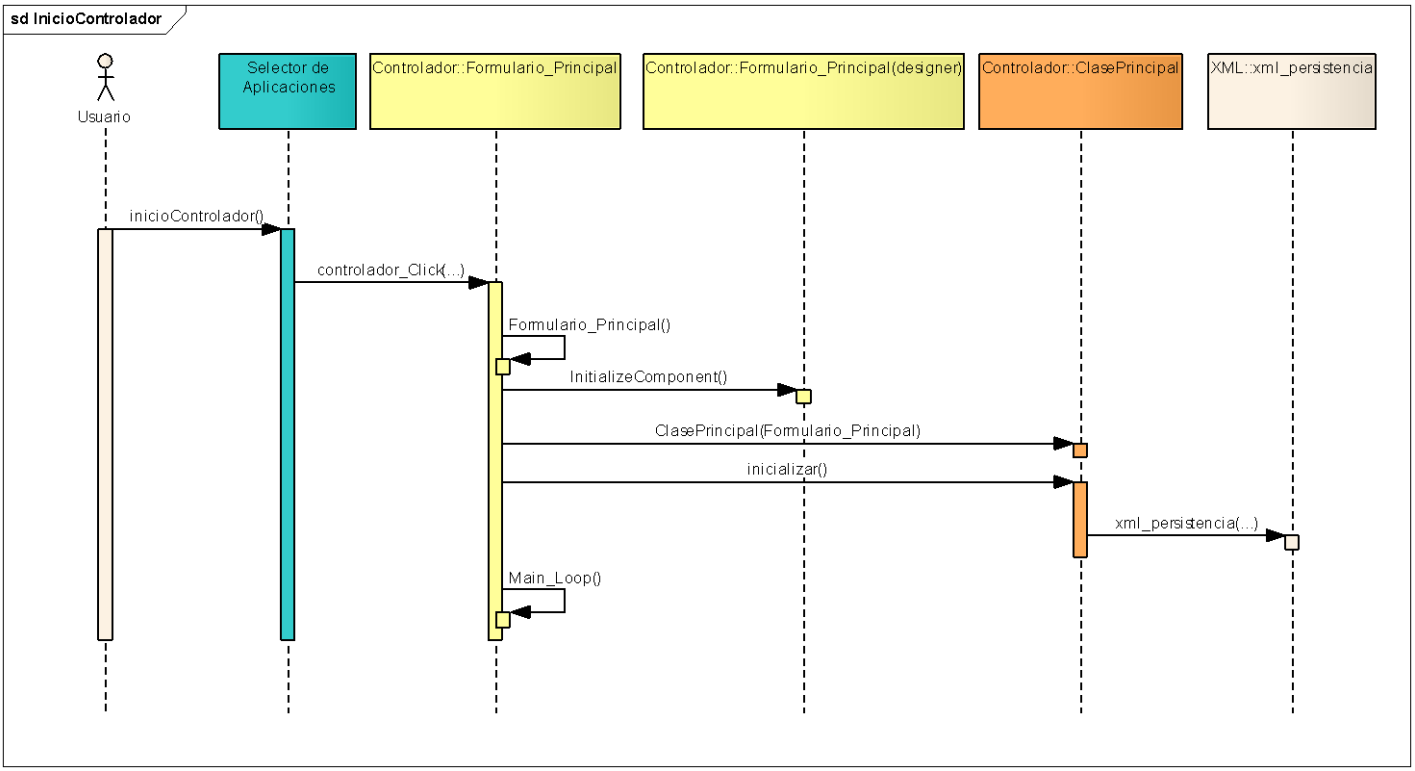


Figura 39: Diagrama de secuencia de inicio del controlador

Cuando un usuario hace clic en el selector eligiendo el controlador, se lanza un proceso que inicia una instancia del mismo. Para ello, primero se llama a su constructora, Formulario_Principal (), y posteriormente se inician todos los elementos del formulario, (InitializeComponent ()), y el controlador, (que a su vez realizará la carga de la configuración previamente guardada).

En último lugar, se inicia el contador que hará que se ejecute el bucle principal, (Main_Loop), cada 10 milisegundos.

Finalización del controlador:

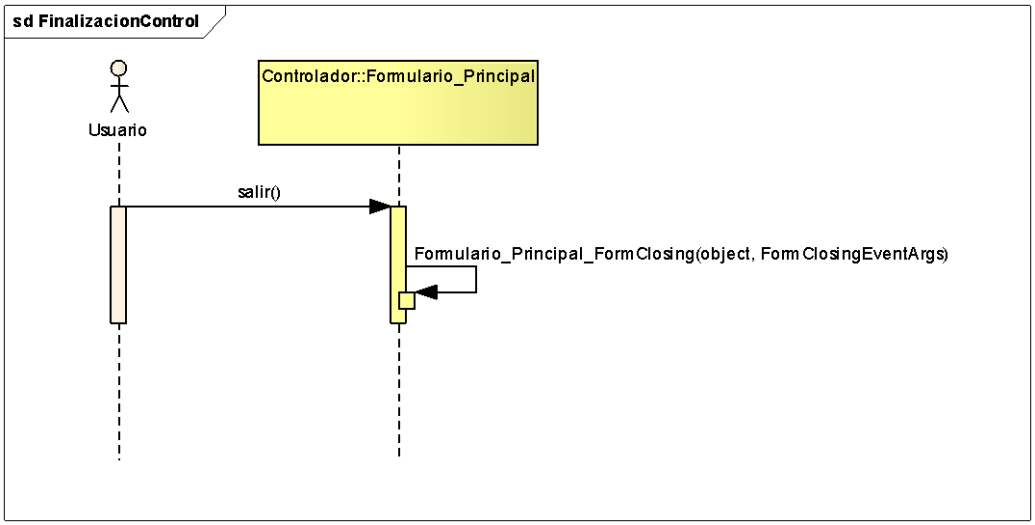


Figura 40: Diagrama de secuencia de finalización del controlador

Se trata de un proceso más simple: cuando el usuario decide terminar el sistema, se llama al método que destruye el formulario y se cierra. No se guarda la configuración actual ni los resultados de las pruebas realizados.

Inicio del simulador:

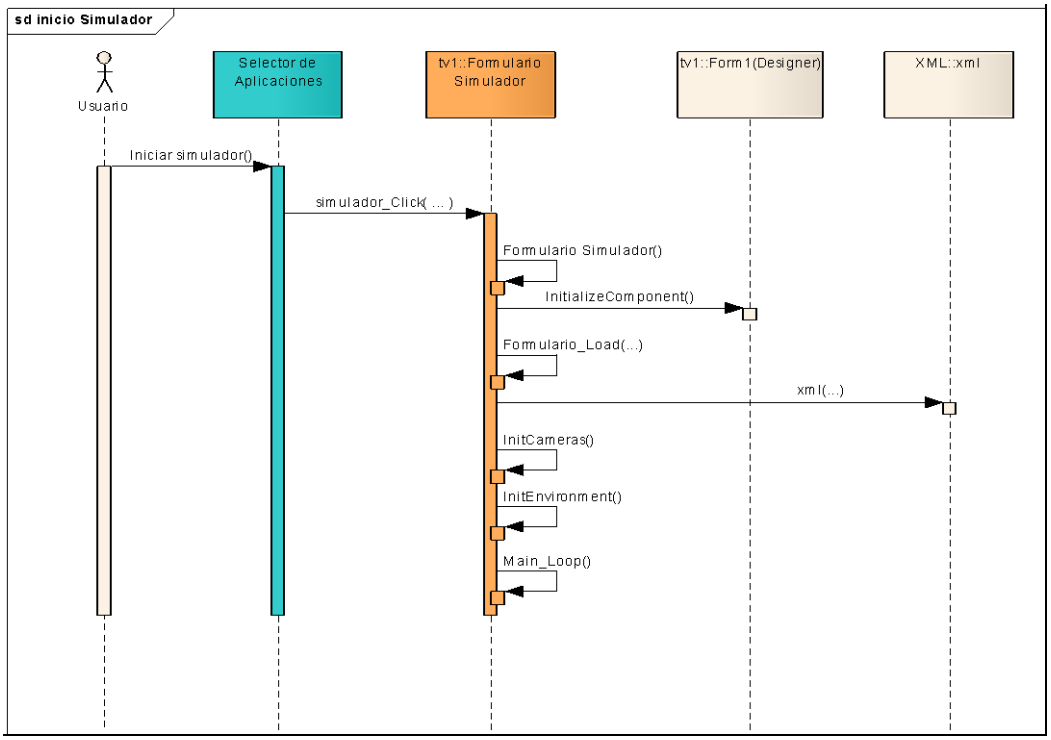


Figura 41: Diagrama de secuencia de inicio del simulador

En este caso, después de que el usuario seleccione el simulador en el selector, se inicia la creación de la instancia correspondiente: se llama a la constructora, (FormularioSimulador), se inicializan las componentes del escenario y formulario, (InitializeComponent, FomrLoad()), se cargan del fichero XML por defecto los objetos y texturas del escenario, (XML()), y se inicializan las cámaras,(InitCameras()) y el entorno, (Init_Environment()). Por último, al igual que en el controlador, se inicia el Timer que se encargará de hacer que el sistema responda con rapidez a las peticiones del usuario.

Finalización del simulador:

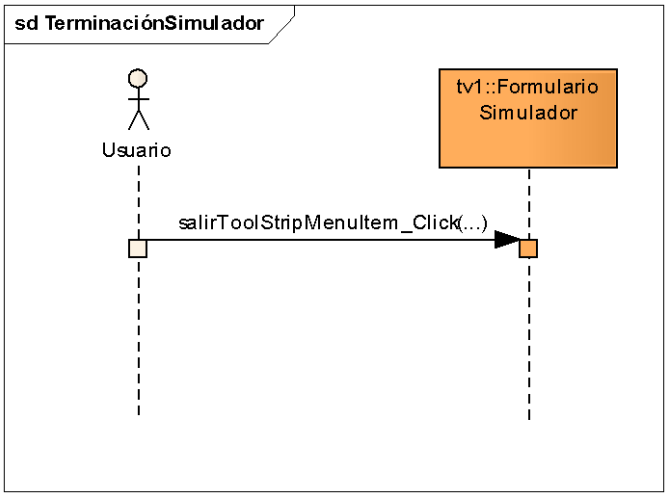


Figura 42: Diagrama de secuencia de finalización del simulador

Si el usuario presiona sobre el botón de salida, se ejecuta el método correspondiente que cierra los recursos clave antes de dejar de hacer visible el formulario.

4 DESCRIPCION DE LAS INTERFACES

A continuación se muestra de forma más detallada como es la composición de las interfaces de usuario del simulador, controlador y generador de escenarios, con el fin de que un posible usuario de la aplicación aprenda a usarla.

4.1 Interfaz de usuario del controlador

El controlador es el encargado de hacer el control del UAV, con el se controla el comportamiento, del UAV real o del UAV virtual del "simulador", a través de su conexión UDP con el resto de aplicaciones y su conexión con la emisora a través del puerto RS-232.

En esta sección de detalla la interfaz del controlador y cuáles son las funciones de sus controles

Visión global de la interfaz de usuario:

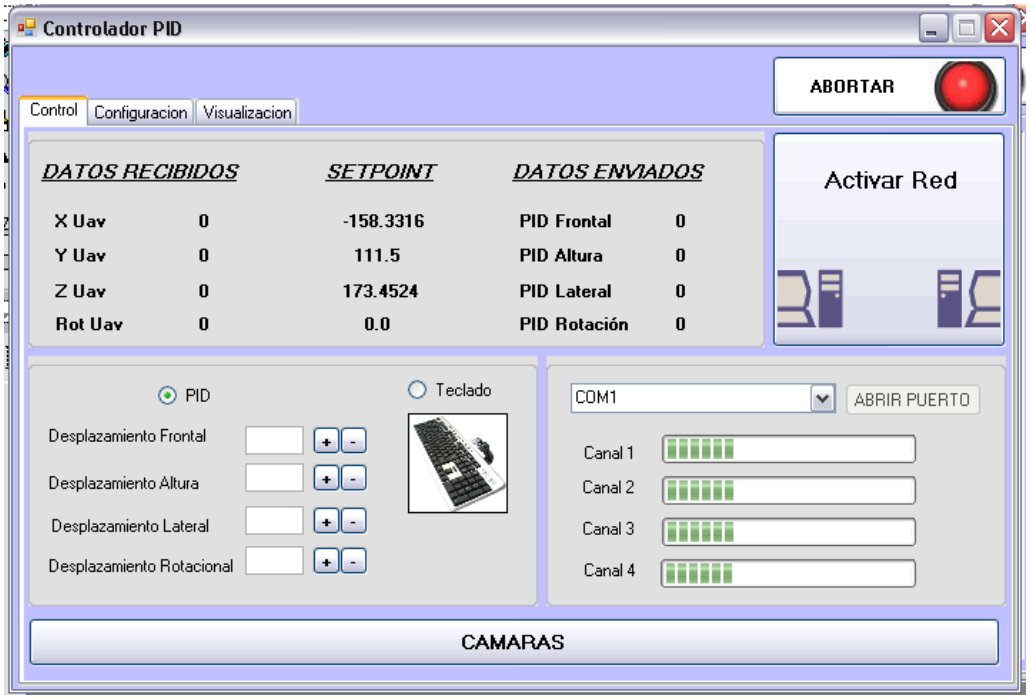


Figura 43: Interfaz de usuario del controlador

Esta es la interfaz inicial del controlador que se muestra al arrancar la aplicación, esta es la interfaz de control, donde se establecen las conexiones con el sistema remoto por UDP y se inicia o finaliza el control, nos ofrece una visión en tiempo real de las señales de control y de error.

Visión detallada de la interfaz de usuario:

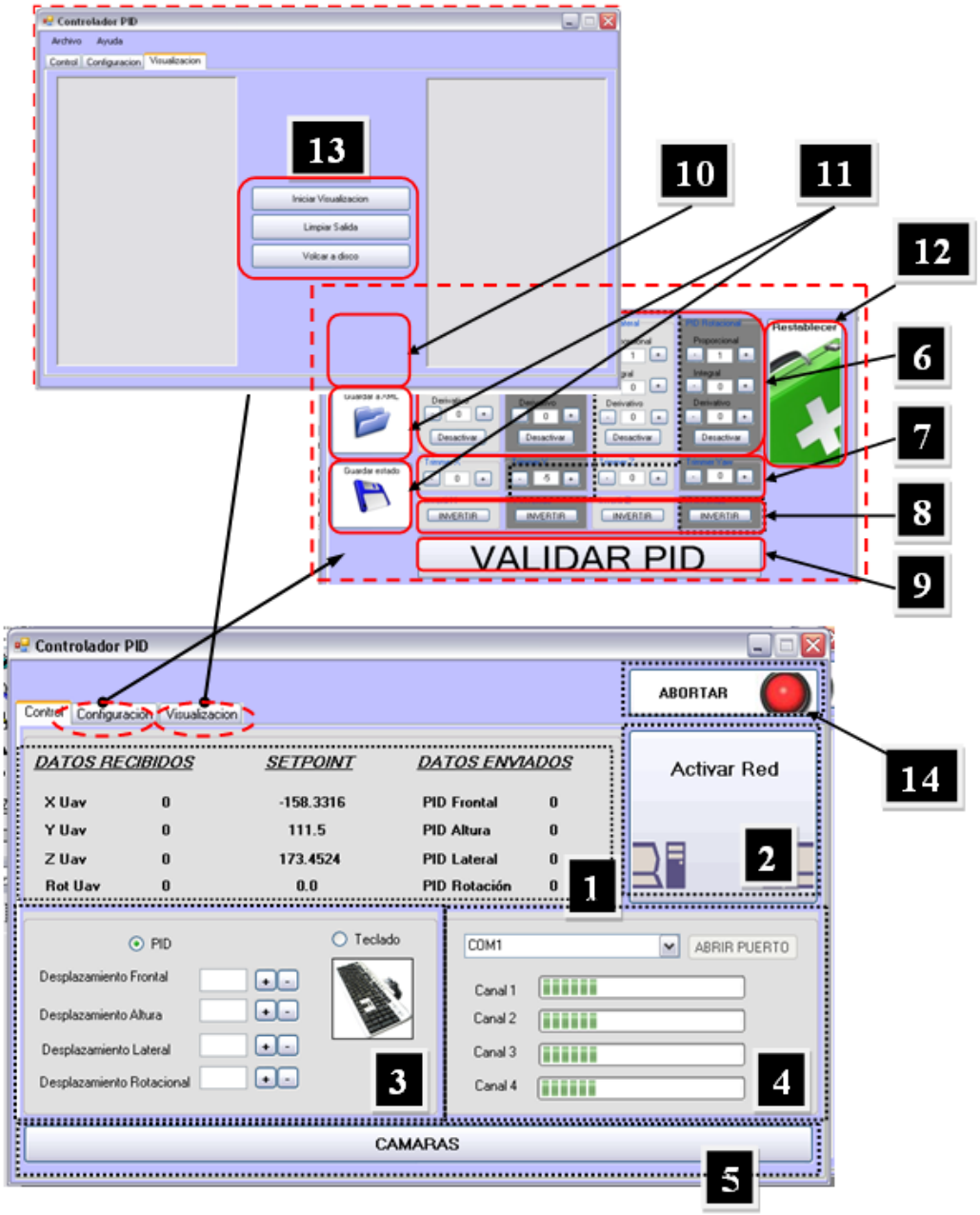
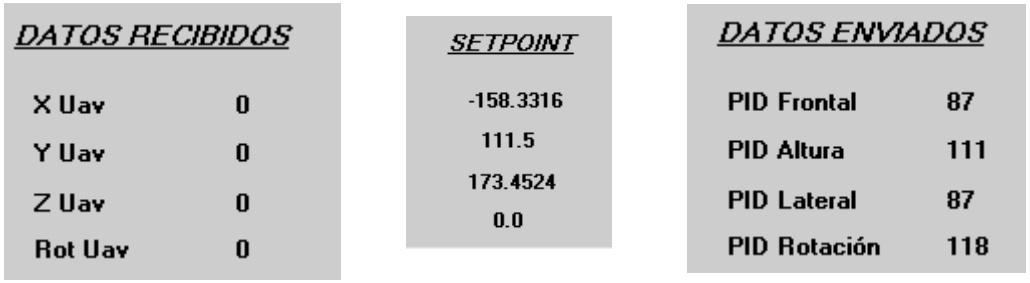


Figura 44: Detalles de la interfaz de usuario del controlador

Partiendo de la visión detallada, se explica ahora la función de los controles de la interfaz

1.- Variables de estado

Son un conjunto de variables de estado que muestra el comportamiento del controlador y la posición de UAV en tiempo real.



La primera columna, DATOS RECIBIDOS, muestra la posición del UAV recibida por la conexión a través del módulo UDP.

Trabajando con el simulador, estos datos componen la posición del UAV en el espacio, pues el simulador trabaja con direcciones en el espacio completo.

En el caso del sistema real, los datos recibidos como ya sabemos serán los errores en la posición de UAV con el setPoint, si queremos conocer la posición, solo tenemos que restarle a los datos recibidos el setPoint.

La segunda columna, SETPOINT, corresponde con el punto de consigna hacia el que tiene que desplazarse el UAV, comparándolo con la columna anterior, se ve si el helicóptero a ha llegado a su objetivo o no.

La última columna, DATOS ENVIADOS, contiene los datos obtenidos por el control PID, estos datos serán los mismos que se envían a través de módulo UDP al simulador y por el puerto RS-232 a la emisora, cuando corresponda.

2.- Botón de acción: red

Este botón tiene la función de activar la red e iniciar la comunicación. Al pulsarlo crea una nueva instancia del módulo UDP y entra en un ciclo de espera hasta recibir una respuesta. En ese momento la conexión está establecida, y en el caso del simulador ya comienza a enviar los valores de control por UDP. Con el sistema real, aún no comienza el control.

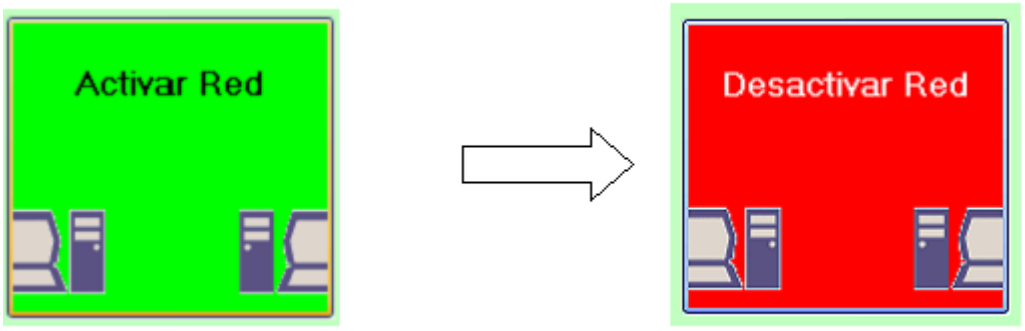


Figura 45: Activar/Desactivar red

3.- Movimiento del UAV

Los controles de este cuadro permiten controlar el desplazamiento del UAV.

Vemos dos opciones, la primera correspondiente a PID, define que se realiza el control PID hacia el setPoint, y con los 4 controles de debajo podemos variar el setPoint, para realizar un movimiento.

La segunda opción permite controlar el UAV directamente desde el teclado del PC como si este fuera una emisora, sin aplicar ningún tipo de control.

Figura 46: Cambio del setPoint

4.- Salida emisora

El siguiente control activa el envió a través de la emisora, permite elegir el puerto de la lista de puertos disponibles en nuestro PC, en el que este enchufada la emisora.

Para el caso en que trabajamos con el sistema real, en este momento es cuando comienza el control PID.

Por motivos de seguridad, antes de iniciar el control se mirara la posición del UAV, si este en tierra, en lugar de iniciar inmediatamente el control, se lanzara la rutina de despegue.

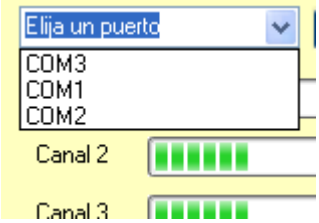


Figura 47: Selección del puerto de la emisora

En las barras de progreso se muestran los valores copiados a la emisora en tiempo real.

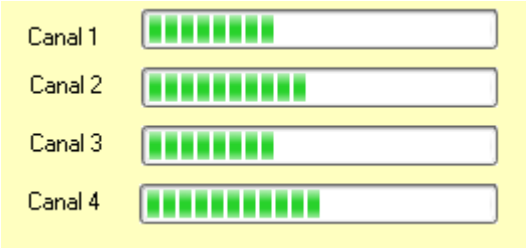


Figura 48: Valores enviados a la emisora

5.- Switch: cámaras / simulador

El botón (o switch) cámaras, indica al programa si se está trabajando con el simulador o con el sistema real. El nombre que muestra en su texto indica con que sistema está funcionando.

Este botón es necesario debido a que el control es diferente para el simulador y para el sistema real

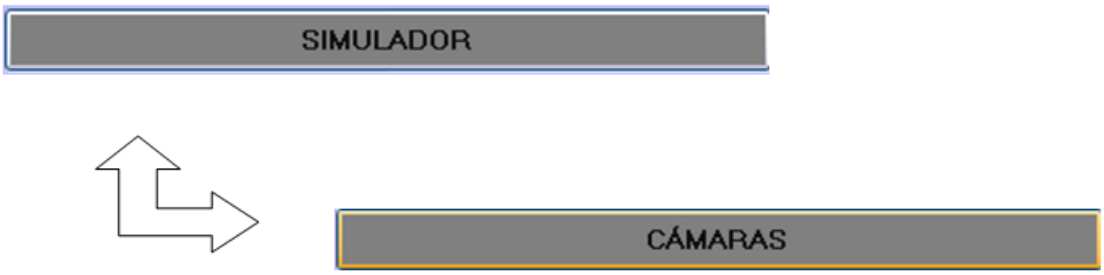


Figura 49: Cambio entre simulador y sistema real

Menú Configuración

A continuación se muestra el menú configuración, mediante el cual se pueden configurar los parámetros de control que actúan en la aplicación:

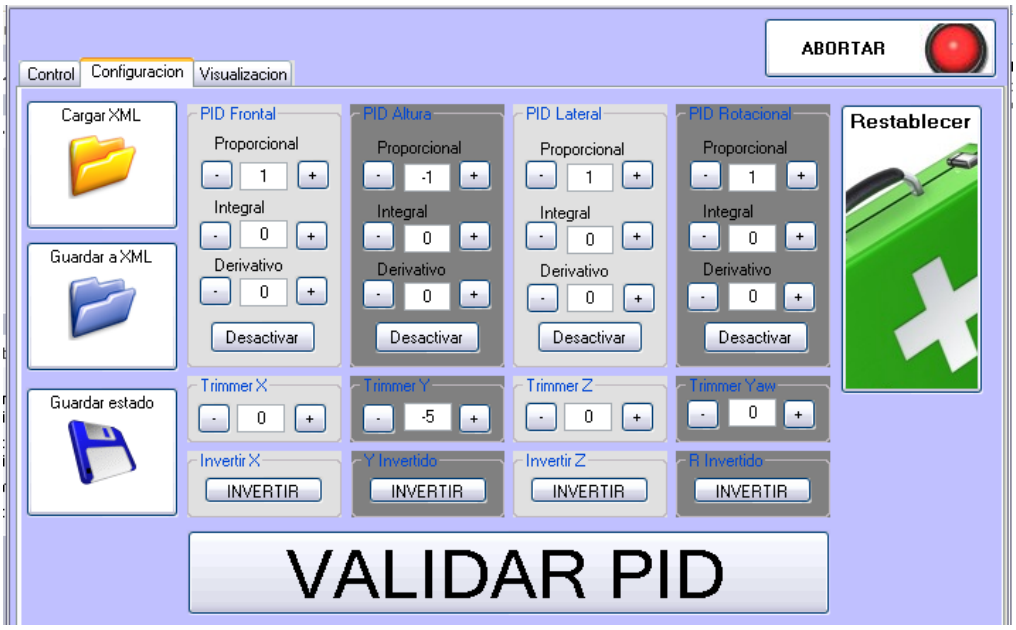
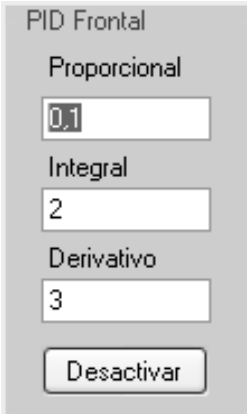


Figura 50: Menú de configuración del controlador para el sistema

6.- Variables de control: PID



Este dialogo sirve para cambiar los parámetros P, I y D de cada uno de los controles PID asociados a los cuatro grados de libertad del UAV, este cambio será inmediato en los controles.

También incorpora un botón para desactivar temporalmente el control PID, esto puede ser útil cuando se estudia la respuesta del controlador en un solo eje, para verlo más claro sin la interferencia de los demás.

7.- Variables de control: Estabilizador (Trimer)

La emisora tiene un valor medio para cada canal, que será el punto donde retorna el control de cada eje si nosotros lo soltásemos el mando de la emisora. Este punto debe enviar al canal un valor que no siga desplazando al UAV en ese eje. Con el valor de estabilidad podemos transformar el valor medio enviado por el controlador en otro valor que alcance ese punto de estabilidad en el eje a configurar.

Para el eje de altura, el valor medio elegido representa la fuerza que queremos que lleve el UAV en el aire, cuando está estable en un punto.

8.- Variables de control: Invertir

Figura 51: Valores de inversión de los canales

Este conjunto de cuatro botones sirve para invertir el envío de la señal de control a la emisora. En el comportamiento normal se entiende que el valor más alto del rango de control (255) será la máxima potencia, mientras que en el comportamiento invertido, el valor mínimo de control será la máxima potencia en ese eje.

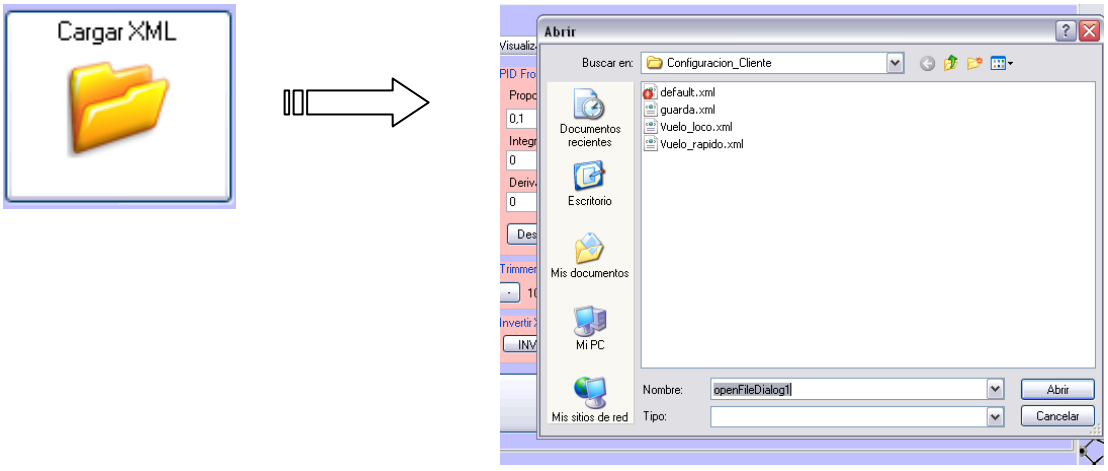
La inversión se realiza una vez obtenidas las señales de control, en el momento de escribirlas en la emisora.

9.- Botón de acción: validar

Figura 52: Botón para que la nueva configuración sea efectiva

Sirve para forzar el cambio de los parámetros de configuración en el controlador, en el caso de que por algún error no se reflejara el cambio al escribirlos

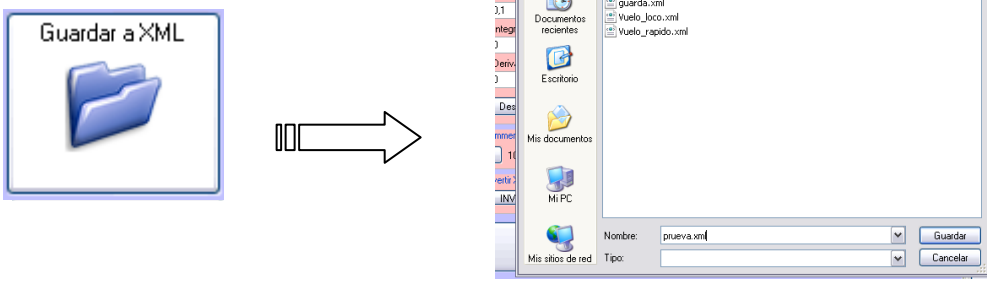
10.- Botón de acción: cargar



Este botón de acción, mediante un nuevo dialogo con el usuario, ofrece la posibilidad de cargar un archivo XML el cual debe tener todos los parámetros de configuración del controlador.

Se utiliza en el caso de tener varias configuraciones para distintas situaciones de vuelo

11.- Botón de acción: guardar



Mediante diálogo con el usuario, podremos guardar la configuración actual del controlador en un archivo que podremos elegir, de esta forma, solo tendremos que configurar una vez el controlador para nuestro UAV.

Este botón realiza la misma acción que el botón guardar, pero sin diálogo con el usuario, escribiendo sobre el archivo default.XML. La función de este botón es guardar los cambios según se hacen con mayor rapidez.

12.- Botón de acción: restablecer

En caso de que la configuración actual no funcione bien para el control, mediante el botón restablecer se carga automáticamente el archivo XML llamado default, que tendrá una configuración fiable de control.

Menú Visualización

Tercera pestaña, corresponde al menú mediante el cual podremos ver la respuesta externa del controlador; En la columna izquierda veremos los datos que la aplicación envía a la emisora (eco de emisora), y en la columna derecha vemos los datos recibidos mediante el módulo UDP.

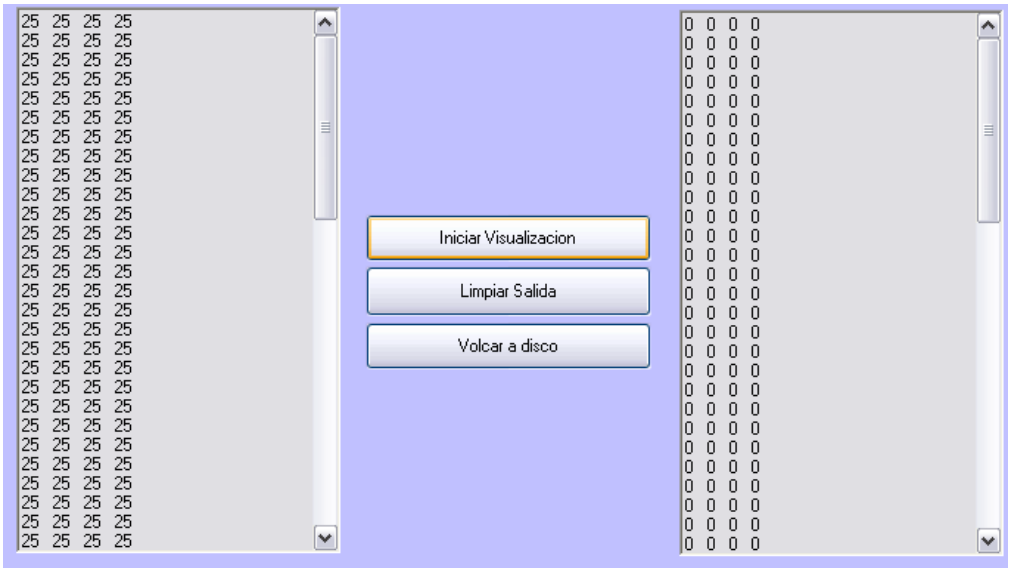


Figura 53: Visualización de la salida

13.- Botones de acción: visualizar

Mediante estos botones de acción, podremos comenzar la visualización cuando nos sea necesario, reiniciarla mediante el botón de limpiar, y guardar las trazas visualizadas, permitiendo generar las graficas de comportamiento para las pruebas del sistema.

14.- Botón de acción: Abortar

El botón abortar realiza un aterrizaje rápido y seguro para el UAV en el momento en que se pulsa. Resulta muy útil en situaciones en las que se pierde el control del sistema o cuando se realizan pruebas en busca de errores.

Configurar por primera vez tu vehículo RC

Como ya hemos visto en la parte del controlador, éste tiene una configuración predeterminada o "default", que se carga por XML al arrancar la aplicación, y además vimos que podríamos crear y guardar nuevas configuraciones.

En este apartado vamos a ver cómo podríamos crear una nueva configuración optimizada para nuestro vehículo en concreto.

Requisitos previos:

- Asegurarnos de tener la emisora conectada al PC mediante un puerto serie de comunicaciones

- Arrancar la aplicación
- Disponer de algo de espacio diáfano para colocar el UAV, preferiblemente en el suelo, para evitar caídas.

Configuración PID:

El tipo de controlador PID recibe su nombre porque se sintoniza mediante sus 3 parámetros (Proporcional, Integrativo y Derivativo). La configuración de estos parámetros suele ser difícil, pero esta tarea puede ser más cómoda gracias a una herramienta de Matlab llamada Simulink, aunque para ello debemos conocer la dinámica del UAV.

En caso de tener la dinámica de nuestro sistema, para ajustarlo con Simulink, deberíamos seguir los siguientes pasos:

1. Abrir Matlab y definir el entorno de trabajo
2. Abrir Simulink y crea un nuevo tapiz

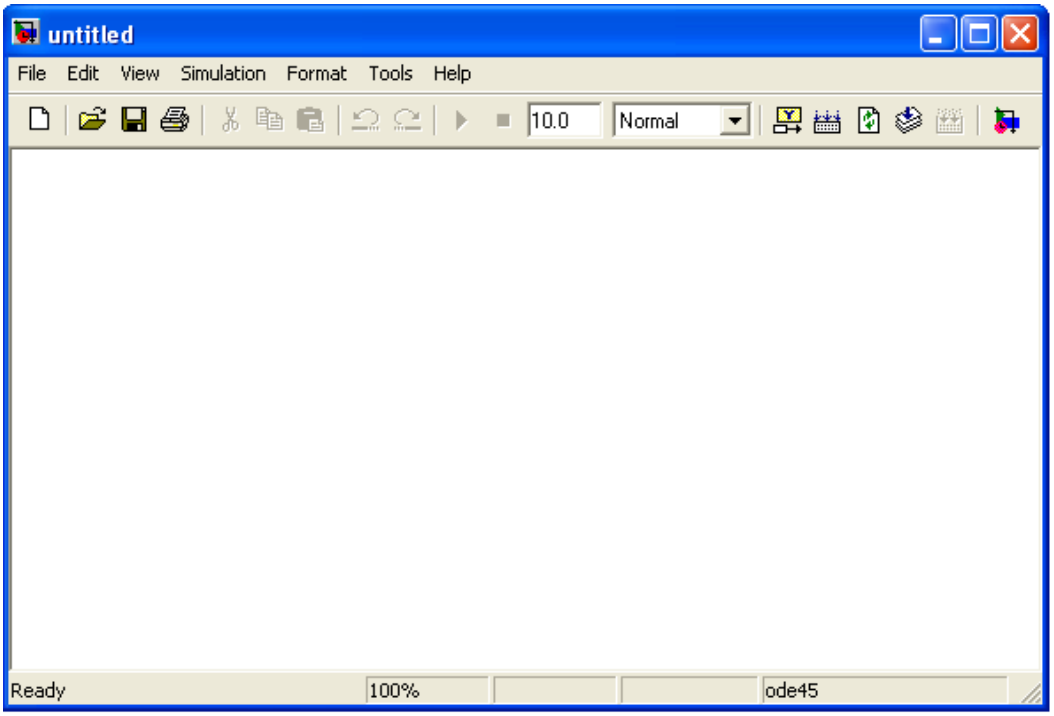


Figura 54: Superficie de diseño en blanco de Simulink

3. Mediante los componentes de la biblioteca, forma un circuito controlador con la dinámica que desees.

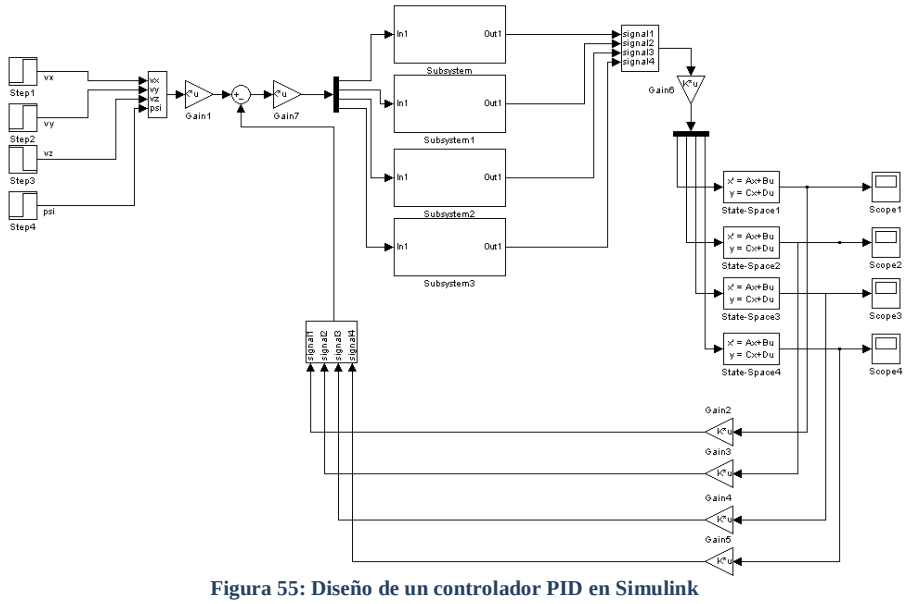


Figura 55: Diseño de un controlador PID en Simulink

4. Configurar los parámetros de la dinámica mediante el dialogo y las matrices A, B, C, D

5. Simular con diferentes parámetros para PID hasta conseguir una curva de respuesta similar a la siguiente.

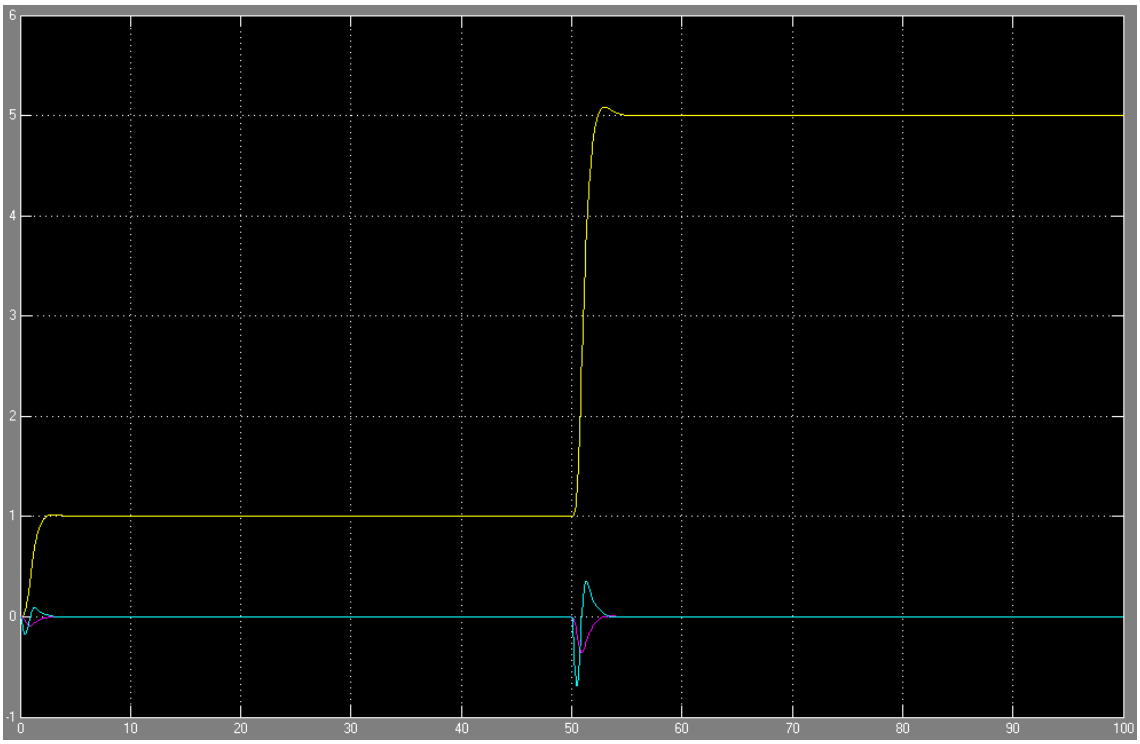


Figura 56: Respuesta en Simulink del controlador PID

Configuración estabilizadores (Trim):

1. Como primer paso, deberíamos poner a 0 el estabilizador de altura, para así parar el motor del UAV, e inmediatamente después iremos subiendo poco a poco, viendo como aumenta la velocidad de giro de las hélices. para los siguientes pasos, no escogeremos ahora la velocidad media final de este eje, sino una más baja, la velocidad recomendada será una en la que el helicóptero comience a levantarse del suelo.
2. A continuación, elegiremos uno de los otros 3 canales a controlar. Tenemos que variar el valor de estabilización de ese canal hasta que no oscile el helicóptero sobre el eje de movimiento asociado al canal.
3. Haremos lo mismo que en el paso anterior para los dos ejes restantes.

Configuración de la inversión de los canales:

Si vemos que el helicóptero no responde como queremos al configurar los estabilizadores, probablemente sea porque alguno de los canales esta invertido en la emisora, como por ejemplo, si con un valor muy bajo de altura vemos que tiene mucha fuerza, o que al aumentar el valor de trim de rotación vemos que se acelera en sentido horario. Bien, en este caso podremos invertir el canal para solucionar el problema, a continuación se muestra una tabla con la configuración original del controlador y su comportamiento, a partir de la cual deberemos invertir los canales para que coincidan con la configuración de nuestra emisora.

	Canal Y	**Canal X**	**Canal Z**	**Canal rotación**
Valores altos	subir	Adelante	Derecha	giro horario
Valores bajos	bajar	Atrás	Izquierda	giro anti horario

Por último, deberemos guardar la nueva configuración mediante el botón de guardar, con el nombre deseado, aconsejablemente, un nombre que relaciones el archivo con nuestro vehículo.

4.2 Interfaz de usuario del simulador

El simulador es una herramienta diseñada para probar, ajustar y calibrar los movimientos del UAV sin tener que usar el UAV real. De esta forma, podemos probar si el funcionamiento del Cliente UAV es el correcto sin necesidad de poner el UAV real en funcionamiento. El simulador, a su vez, nos permite familiarizarnos con el uso de un UAV real de estas características, ya que dispone de una interfaz de conexión con una emisora real para controlar los movimientos del UAV simulado; pudiendo sustituir el control de la emisora, por un control similar al que se realiza desde ésta, pero simulado mediante el teclado.

Visión global de la interfaz de usuario:

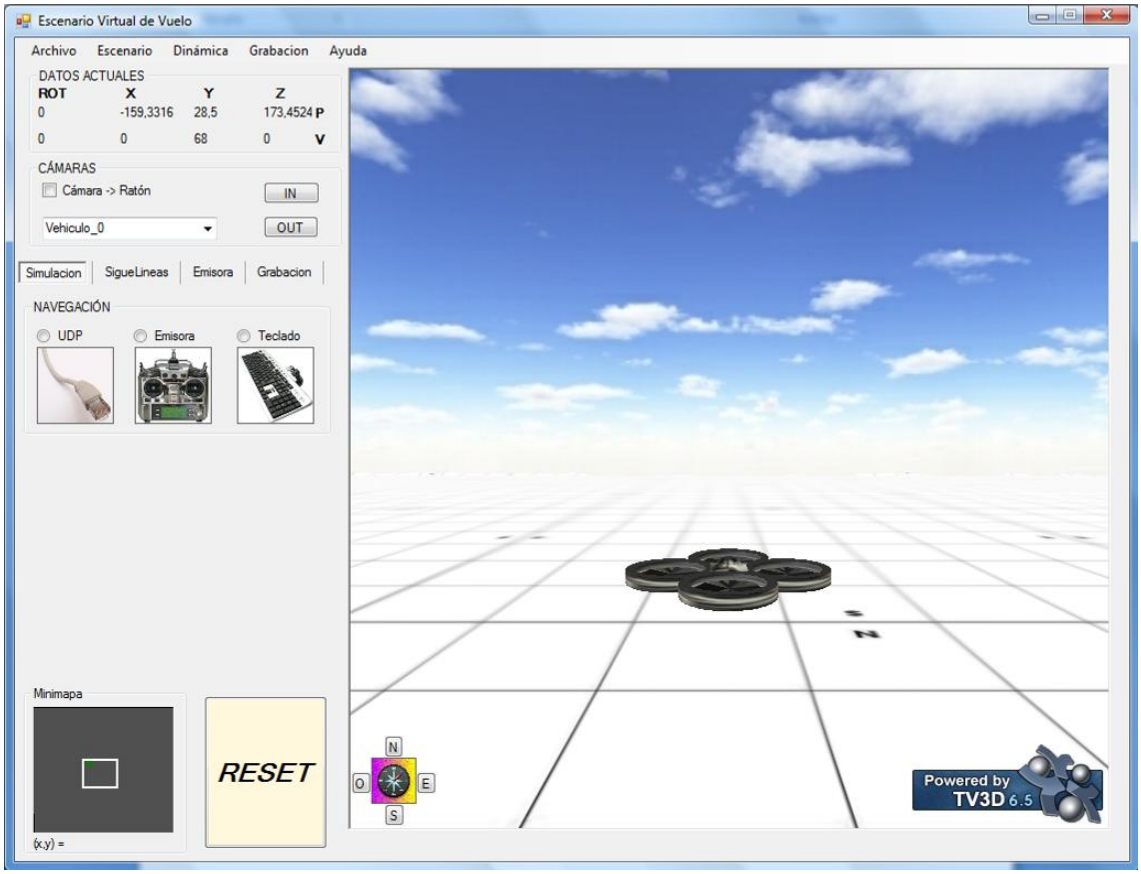


Figura 57: Interfaz de usuario del simulador

Como se puede ver en la imagen, se dispone de tres partes en la interfaz de usuario bien diferenciadas.

La primera es el menú de configuración, con el cual podremos acceder a distintas opciones como cambiar de escenario, o cambiar los parámetros de la dinámica que simula el UAV.

La segunda es la columna de control del UAV, a través de la cual controlaremos el UAV y visualizaremos sus parámetros en todo momento.

La tercera es la zona de dibujado, donde se visualiza el UAV sobre el escenario.

Menú de configuración:

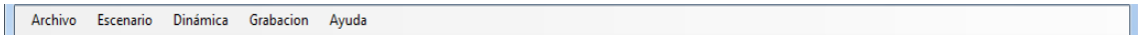


Figura 58: Barra de tareas del simulador

Archivo:

Tenemos las siguientes opciones:

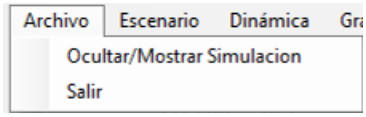


Figura 59: Menú archivo del simulador

La opción 'Ocultar/Mostrar Simulación' ocultará/mostrará la zona de dibujado, liberando, de esta forma, mucho trabajo de la CPU. Esto es útil cuando queremos probar el comportamiento del UAV controlado mediante el programa 'Cliente UAV o mediante la emisora, sin querer sobrecargar de trabajo a la máquina. Esto es posible gracias a que en todo momento se visualiza la posición y parámetros del UAV en la columna de control del UAV. El resultado de activar esta opción es el siguiente:

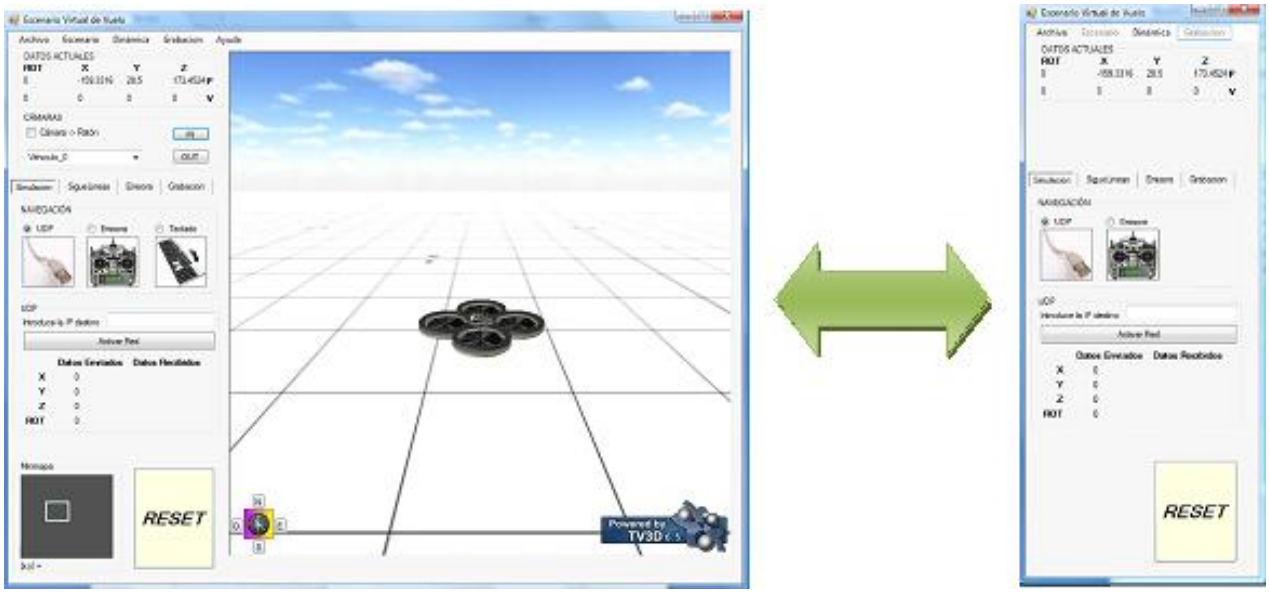


Figura 60: Simulador extendido / Simulador embebido

La opción salir, cerrará la aplicación.

Escenario:

Tenemos las siguientes opciones:

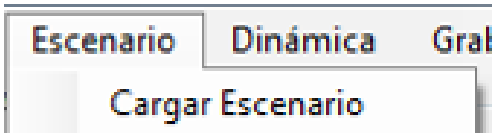


Figura 61: Menú escenario del simulador

La opción 'Cargar Escenario' mostrará un diálogo para abrir un fichero de escenario .XML creado a través de la aplicación 'Generador de escenarios'.

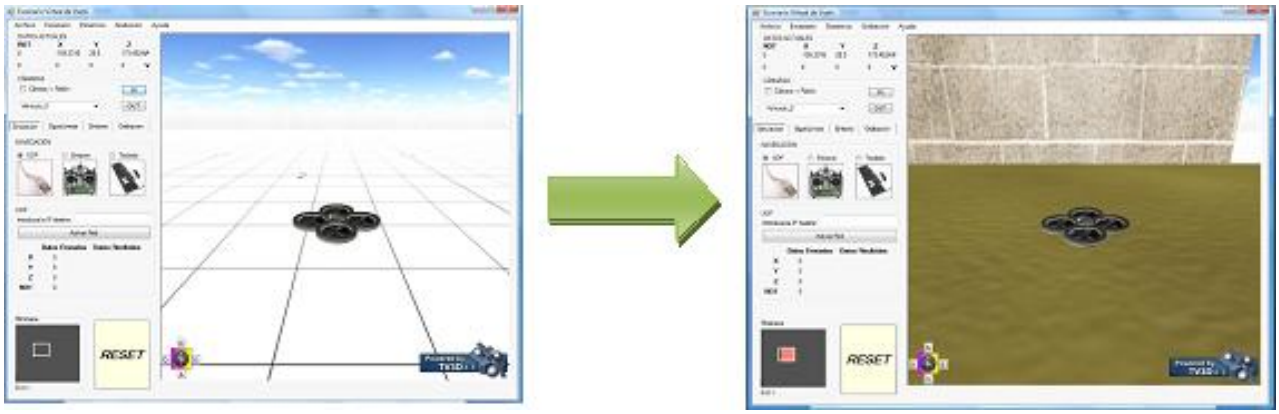


Figura 62: Ejemplo de cambio de escenario en simulador

Grabación:

El menú grabación nos permitirá grabar la trayectoria que ha seguido el UAV en un intervalo de tiempo determinado. De esta forma, se podrá guardar en un fichero XML para reproducirlo más tarde. Tenemos las siguientes opciones:

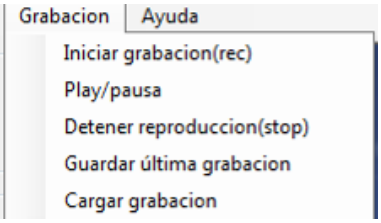


Figura 63: Menú grabación del simulador

La opción 'Iniciar grabación (rec)' comenzará a grabar la trayectoria del UAV en el simulador. Esta opción quedará desactivada hasta que paremos la grabación mediante el botón 'Detener reproducción (stop)'. Podremos pausar/reanudar la grabación mediante la opción 'Play/pausa', por si queremos que no se grabe alguna parte del movimiento, pero que, al continuar, la grabación continúe en el punto en el que pausamos la grabación.

Mediante la opción 'Guardar última grabación', se guardará la última grabación realizada en formato XML; lo que nos permitirá la carga de ésta tras cerrar el simulador, mediante la opción 'Cargar grabación'.

Columna de control del UAV:

Figura 64: Barra de control del simulador

Datos actuales:

Figura 65: Dialogo donde se muestran las características del UAV, posición y velocidad en cada eje

Esta parte de la interfaz mostrará en todo momento la posición del UAV mediante la fila P, en las coordenadas x, y, z, así como su rotación en grados mediante notación decimal. También, mediante la fila V, se mostrará en todo momento las velocidades del UAV en x (frontal), y (altura), z (lateral), y rot (velocidad de rotación).

Cámaras:

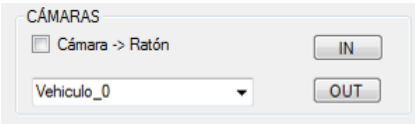


Figura 66: Dialogo selector de la cámara a usar

Mediante esta parte de la interfaz podremos controlar todo lo relacionado con la posición de la cámara en la zona de dibujado.

Para el control de las cámaras, se usan las siguientes teclas del teclado:

- Q: Bajar cámara
- E: Subir cámara
- W: Cámara hacia adelante
- S: Cámara hacia atrás
- A: Cámara hacia la izquierda
- D: Cámara hacia la derecha

Si se activa la opción 'Cámara -> Ratón', se podrá controlar la inclinación y la rotación la cámara mediante el ratón.

El botón 'IN' cambiará la cámara a una posición en tercera persona respecto al UAV, mientras que el botón 'OUT' cambiará la cámara a una colocada en la parte inferior del UAV, enfocada hacia el suelo.

Mediante la lista desplegable se podrá elegir el vehículo en uso en el simulador. Esto sólo será aplicable en el caso de que el mapa contenga varios UAV.

Navegación:

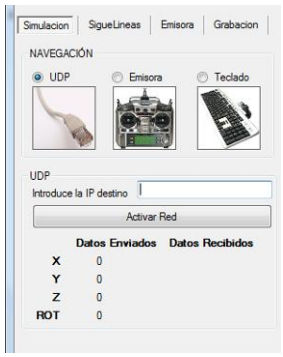
Figura 67: Diálogo selector del control de movimiento del UAV en el simulador

En el menú navegación existen cuatro opciones diferentes:

- Simulación:

Esta opción es la que permite controlar el UAV para simular movimientos, o para probar el programa 'Cliente UAV'. Por lo tanto, existen las siguientes posibilidades:

- UDP:



Esta opción será la que hará uso del programa 'Cliente UAV'. Primero, se introduce la dirección IP donde se está ejecutando el cliente. Una vez hecho esto, se pulsa el botón 'Activar Red', iniciando así, el proceso de control PID. En todo momento se visualizan tanto los datos enviados al cliente, como los recibidos por éste, en la tabla debajo del botón de activar red.

- Emisora:

Esta opción ofrece la posibilidad de controlar el UAV en el simulador, haciendo uso de una emisora real, conectada a un puerto COM de la máquina. Para ello, habrá primero que seleccionar el puerto COM al que se encuentra conectada, y pulsar el botón 'ABRIR PUERTO'. En caso de que sea necesario, habrá que calibrar sus valores máximos y mínimos, pulsando el botón 'CALIBRAR', y, desde la emisora, mover todos los ejes a su valor máximo y mínimo.

El programa ya está preparado para controlar el UAV desde la emisora. En todo momento se visualizará, en las barras Canal 1...Canal 4, la posición del eje en la emisora, haciendo más fácil la visualización del valor que se está enviando.

- Teclado:

Esta opción permitirá un control igual al que se hace desde la emisora, pero emulándola desde el teclado, para ello se hace uso de las telas del teclado numérico. El mapeado de las teclas es el siguiente:

- 7/9: Rotación izquierda/derecha.
- 8/2: Avanzar hacia adelante/atrás
- 4/6: Movimiento lateral hacia izquierda/derecha
- 5/0: Aumentar/disminuir altura

Minimapa:

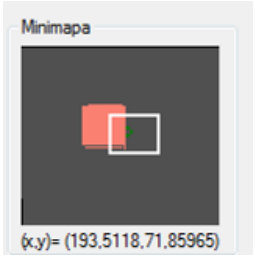


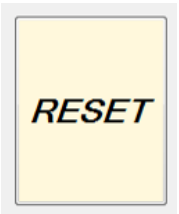
Figura 68: Diseño del mini mapa

El minimapa es una parte muy útil del simulador, ya que en ella se ofrece una vista de pájaro del escenario de simulación. El UAV se muestra de color verde, y todas las paredes en color salmón. El recuadro blanco indica dónde está colocada la cámara en ese instante. También podemos saber donde está colocada la cámara ya que en todo momento se muestra su posición (x, y) abajo.

Si se pulsa en cualquier posición del minimapa, la cámara cambiará al lugar en que se ha pulsado, afectando esto a la zona de dibujado, en la que se obtendrá una vista de pájaro de la zona delimitada por el recuadro blanco.

Botón RESET:

Mediante este botón se podrá reinicia la posición del UAV a la posición de inicio del programa, y reiniciar el estado de su dinámica. Por lo tanto, el UAV se encontrará en una posición de reposo, en la posición inicial indicada por el escenario.



4.3 Interfaz de usuario del generador de escenarios

El generador de escenarios es una herramienta desarrollada para crear escenarios de manera simple e intuitiva. Se apoya en el motor TrueVision3D, gracias al cual podemos mostrar una visión panorámica de cómo quedará nuestra creación.

Visión global de la interfaz de usuario:

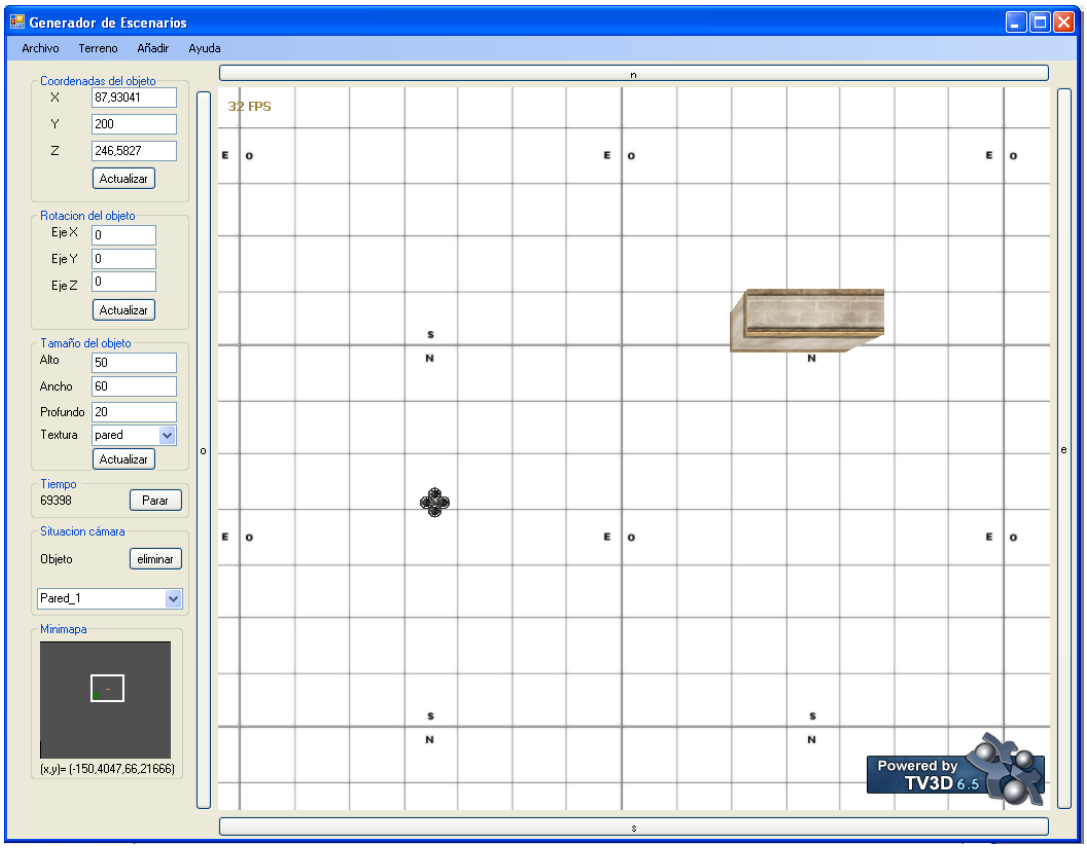


Figura 69: Interfaz de usuario del generador de escenarios

En la parte izquierda y superior de la interfaz, están los menús y opciones de configuración de los objetos y del terreno, mientras que en la parte derecha podemos observar todos los cambios que vamos realizando.

Los cuatro botones N, S, E y O sirven para desplazarnos por el escenario, aunque también podríamos movernos con las flechas de nuestro teclado convencional.

En último lugar, en la parte inferior izquierda encontramos el minimapa, que nos ayuda a tener una visión global de todos los objetos que hay en nuestro escenario así

como un acceso directo a ellos seleccionando sobre él. El cuadrado blanco indica en qué parte del escenario en concreto nos encontramos.

A continuación veremos una descripción detallada de cada parte.

Descripción:

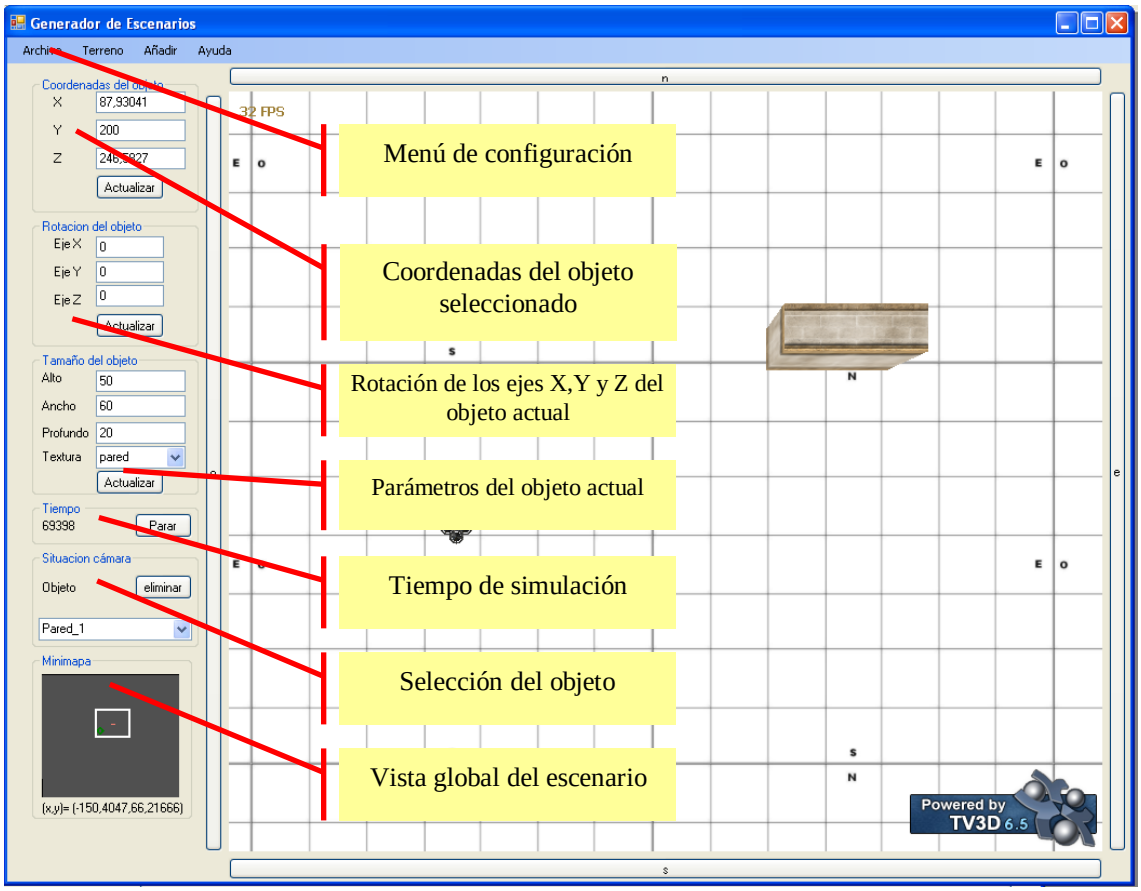


Figura 70: Descripción de la interfaz de usuario del generador de escenarios

Menú de configuración:

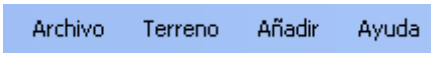


Figura 71: Barra de herramientas del generador de escenarios

Archivo:

Tenemos las siguientes opciones:

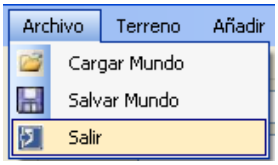


Figura 72: Menú archivo del generador de escenarios

La opción "Cargar mundo" lanzará una ventana para que seleccionemos el archivo que se desea cargar:

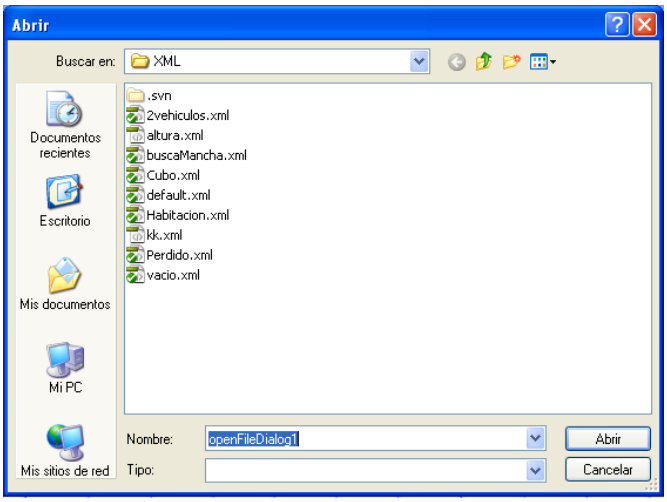


Figura 73: Dialogo de carga XML con datos de un escenario

La opción "Salvar mundo" abrirá una ventana parecida a la anterior en donde nos pedirá dónde deseamos guardar el mapa actual.

Por último, la opción "Salir" sale de la aplicación. Si no se han guardado los cambios, se perderán.

Terreno:

Esta pestaña presenta la siguiente forma:

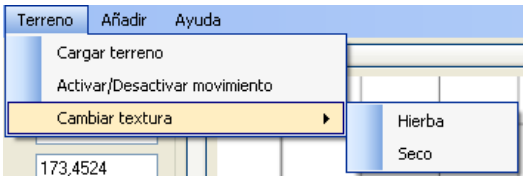


Figura 74: Menú terreno del generador de escenarios

Cargar terreno lanza una ventana para que seleccionemos una imagen en blanco y negro para dar al mapa el relieve que queramos.

Activar/desactivar movimiento realiza una carga de imágenes de manera que parezca que el suelo está en movimiento.

Por último, cambiar textura permite que cambiemos la apariencia del terreno, permitiendo dejar el original o cambiarlo por hierba.

Añadir:

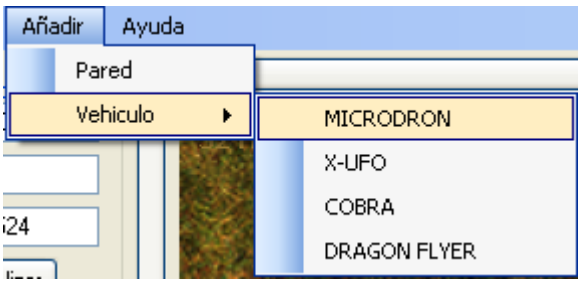


Figura 75: Menú añadir del generador de escenarios

Es la pestaña más importante del menú de configuración debido a que es la que nos permite añadir objetos u obstáculos como queramos. Simplemente hay que seleccionarlo y aparecerá sobre el terreno para que lo desplacemos y giremos como queramos.

Ayuda:

Ofrece ayuda acerca de la aplicación en general.

Menú de coordenadas:

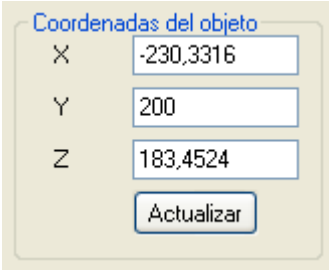


Figura 76: Diálogo de coordenadas de un objeto en el escenario

Permite cambiar las coordenadas al objeto seleccionado. Al pulsar "Actualizar", el objeto se situará en la nueva posición. Cabe destacar que un objeto no se tiene que cambiar estrictamente con este menú: podemos seleccionarlo con el botón izquierdo del ratón y desplazarlo libremente manteniendo el derecho.

A medida que lo movamos por el mapa, las coordenadas se reflejarán automáticamente en el menú.

Menú de rotación:

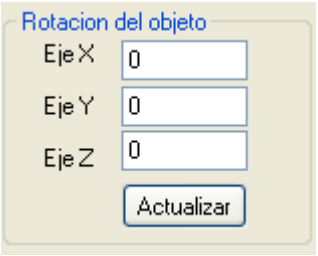


Figura 77: Diálogo de rotación de un objeto en el escenario

Permite rotar el objeto respecto a los ejes de coordenadas. El Y es el eje vertical, (no el Z, como es costumbre).

Menú de parámetros de objeto:

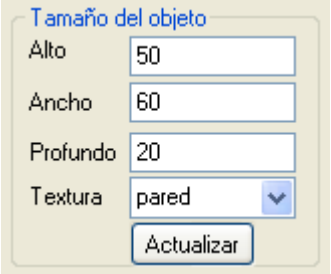


Figura 78: Diálogo de tamaño de un objeto en el escenario

Este submenú aparecerá únicamente si seleccionamos un objeto de tipo pared, (obstáculo). Podremos definir tanto su anchura como su longitud y profundidad, así como la textura que poseerá.

Tiempo de simulación:

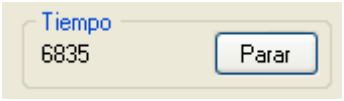


Figura 79: Diálogo de tiempo de simulación editando un escenario

Menú que permite parar la simulación en cualquier momento, pulsando sobre el botón "Parar".

Menú de selección:

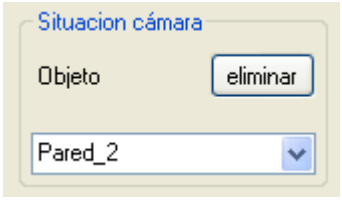


Figura 80: Diálogo de selección de objetos del escenario

Este menú permite elegir los objetos que hay en el escenario, (centrando la cámara automáticamente en ellos), así como eliminar el objeto seleccionado actualmente.

Minimapa:

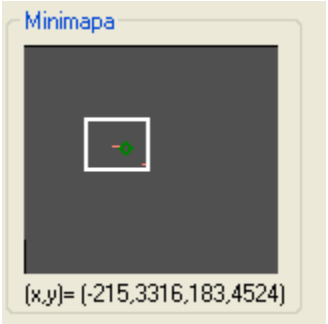


Figura 81: Mini mapa en el generador de escenarios

En este simple recuadro se muestra el mapa completo. Lo que en la imagen es el recuadro de color gris oscuro equivaldría al todo el escenario, mientras que lo que

aparece dentro del rectángulo blanco es la zona a la que estamos enfocando actualmente.

Los pequeños dibujos color salmón y verde indican dónde están los objetos que hay en el escenario. Si el objeto es de color salmón, indica que es una pared, mientras que si es de color verde oscuro es un vehículo.

Las coordenadas de la parte inferior muestran el punto donde estamos enfocando la vista con el rectángulo blanco.

En caso de querer desplazarnos a una zona del mapa en concreto, basta con hacer clic en el minimapa y automáticamente la cámara saltará a las coordenadas que correspondan. Se sitúa siempre a vista de pájaro sobre las mismas, para que tengamos una visión global de lo que estamos haciendo.

5 PRUEBAS DEL PRODUCTO

5.1 Pruebas en el simulador

Resultado de las pruebas sobre el simulador:

Se han realizado 57 pruebas con el simulador. De ellas aproximadamente 20 han sido para calibrar los parámetros de control del UAV en el entorno virtual, y el resto para probar que el UAV se comporta adecuadamente cambiando los puntos de consigna de los distintos ejes.

Para simplificar los resultados, hablaremos de las 3 pruebas más significativas y las explicaremos en detalle, mostrando además los resultados obtenidos en cada una. En cada gráfica podremos ver los resultados del control en azul, y en verde el error recibido del simulador.

PRUEBA 1:

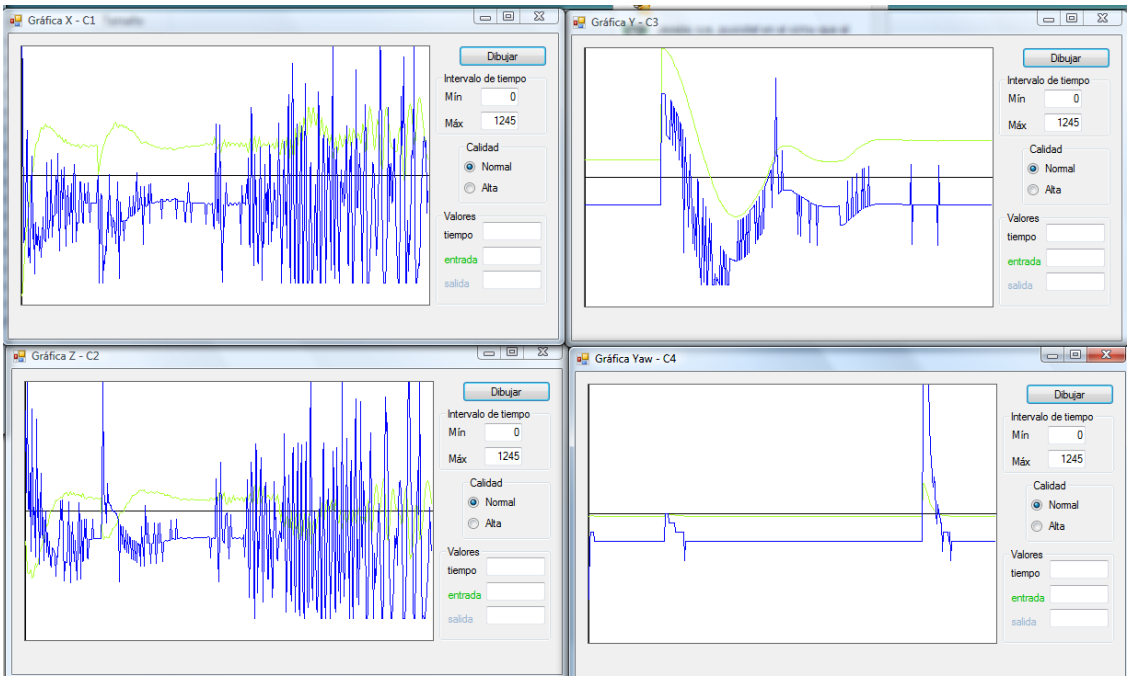


Figura 82: Gráfica de resultados de la prueba 1 en simulador

En esta prueba se aprecia claramente cómo, en el eje X (frontal, gráfica situada arriba a la izquierda) y Z (lateral, gráfica situada abajo a la izquierda), la señal de

control reacciona de forma totalmente contraria a la señal de error. Los cambios en la señal de control, (los picos que aparecen en la gráfica), son debidos al control derivativo que usan los controladores PID y a la dinámica del sistema.

En las gráficas que representan la altura, (arriba a la derecha), y el yaw, (abajo a la derecha), los canales están invertidos, por lo que la señal de control no es complementaria a la de error. Esto se debe a la configuración de la emisora y no afecta a la estabilización del UAV. En ambas gráficas, vemos cómo el sistema controlador es capaz de estabilizar el UAV al final, obteniendo como resultado una señal de error constante, de apenas pocas unidades. También se aprecia la oscilación existente en la altura como resultado de cambiar el setPoint de ésta, (pico verde casi al inicio de la simulación). Sin embargo, la sobreelongación y el tiempo de estabilización son muy pequeños por lo que no afectan a la estabilidad del sistema.

PRUEBA 2

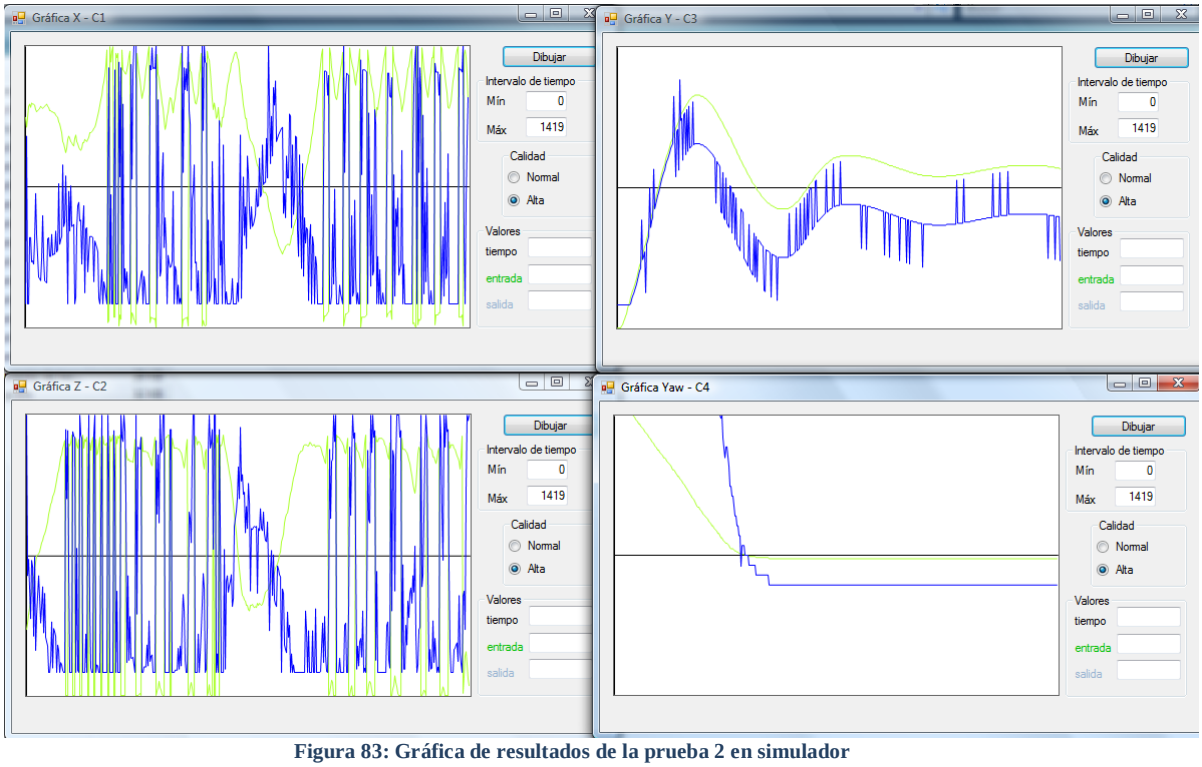


Figura 83: Gráfica de resultados de la prueba 2 en simulador

En este caso, al igual que la anterior, se aprecia cómo el UAV se queda estable alrededor de un punto fijo en los ejes X y Z, (gráficas superior izquierda e inferior

izquierda). La oscilación con la que se queda el UAV sobre su punto de consigna es despreciable puesto que es muy pequeña no se ha desestabilizado a pesar de continuar la simulación durante varios minutos.

Nuevamente, en las gráficas de los ejes de altura, (arriba a la derecha) y yaw, (abajo a la derecha) el canal está invertido, lo que no impide que se aprecie cómo la señal de control actúa de forma equivalente a la de error, estabilizando la altura a partir de la mitad de la gráfica. El yaw se estabiliza antes incluso de ese punto, lo que supone unos tiempos de estabilización muy rápidos, y un sistema rápido, fiable y seguro.

PRUEBA 3

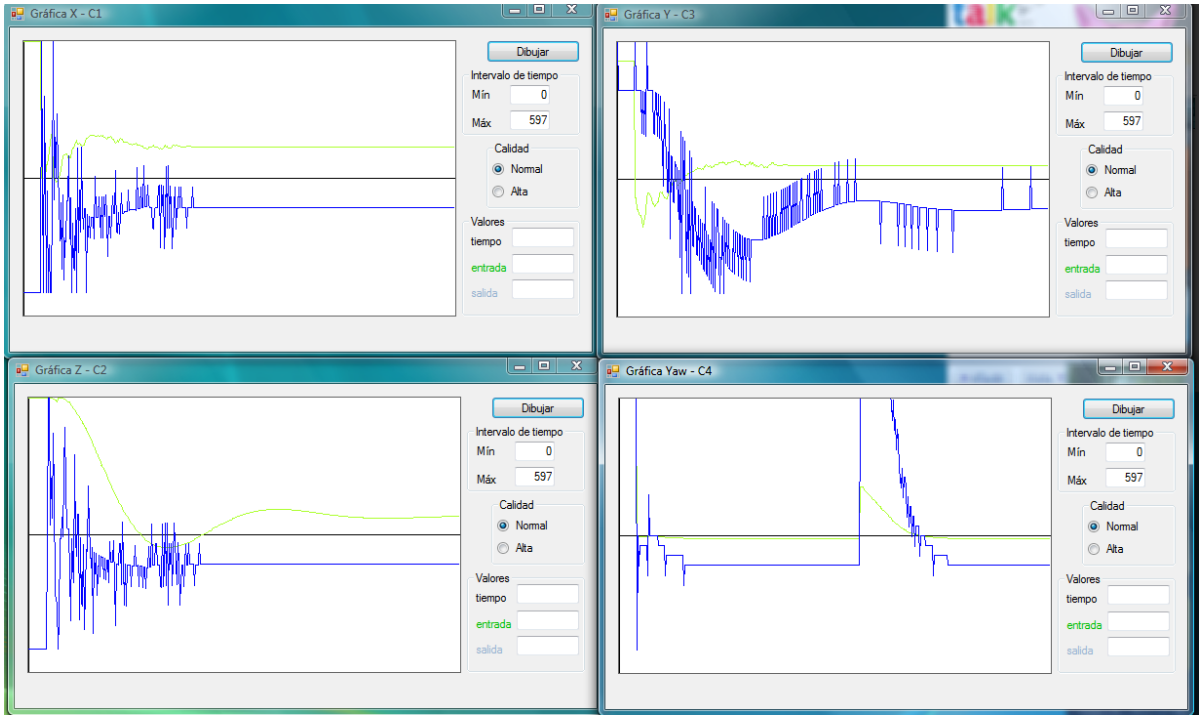


Figura 84: Gráfica de resultados de la prueba 2 en simulador

En la gráfica de los ejes X y Z, (de nuevo arriba a la izquierda y abajo a la izquierda, respectivamente), se puede apreciar cómo el sistema empieza con un error de magnitud bastante alta y, gracias al control que se aplica al UAV, se llegan a estabilizar ambos ejes. El primero, el eje X, se estabiliza alrededor de t=1/3 del tiempo simulado, sin nada de sobreelongación; mientras que el segundo, el eje Z, tarda algo más, se estabiliza en t=1/2 del tiempo simulado, y con algo más de sobreelongación. El

resultado es una estabilización perfecta con una ínfima diferencia entre el punto de consigna y el la posición del UAV en un corto período de tiempo.

El eje de altura, (arriba a la derecha), también se estabiliza alrededor de t=1/3 del tiempo simulado, aunque con algo más de sobreelongación que el eje X.

En el eje del yaw, (abajo derecha), se ha aplicado una señal para desestabilizarlo alrededor de la mitad del tiempo simulado, resultando en la disminución de la señal de control (ya que se encuentra invertido el canal), que consigue estabilizar de nuevo el UAV en su punto de consigna.

Resultado de las pruebas de sigue-líneas:

A continuación detallamos las pruebas más significativas que hemos realizado en el simulador en el modo sigue-líneas. Se trata de 4 pruebas, donde se ha verificado si el helicóptero era capaz de seguir las líneas de circuitos de diferente complejidad, incluso estando perdido.

Todas las pruebas se han realizado con el simulador, en mapas de 2300x2300 pixeles ya binarizados, para tener una visión nítida del circuito.

- Circuito número 1:

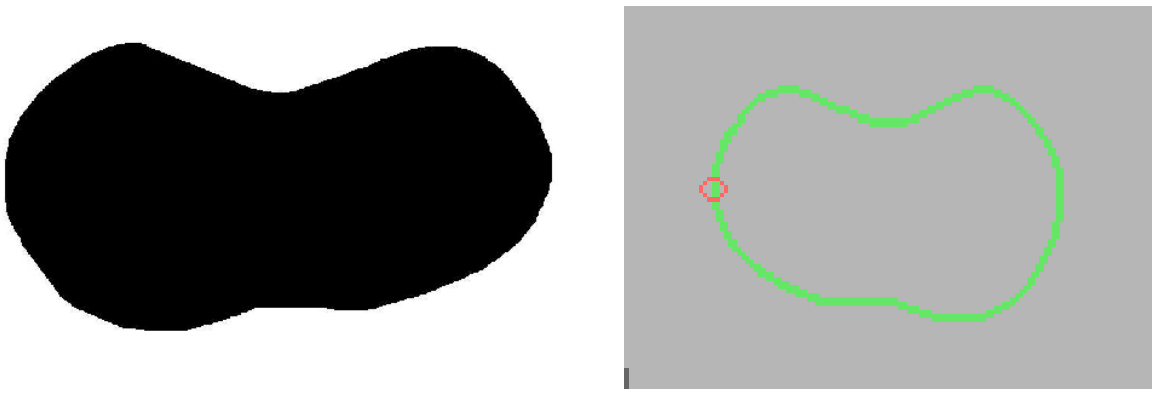


Figura 85: Pruebas de circuito 1

Como primer circuito, presentamos uno de poca dificultad, con curvas suaves, redondeadas y sin brusquedades para seguir.

La trayectoria seguida por el helicóptero es la línea verde, y su posición inicial está marcada en rojo. Podemos comprobar que el circuito se ha seguido fielmente, y hemos comprobado tras 10 minutos de simulación que el helicóptero seguía avanzando encima de la línea.

- Circuito número 1: Helicóptero perdido

En este caso repetimos el circuito anterior, con la diferencia de que el helicóptero comienza la búsqueda enfocando el centro de la mancha en lugar de empezar sobre la línea como en la primera prueba. La trayectoria que obtenemos en este caso es:

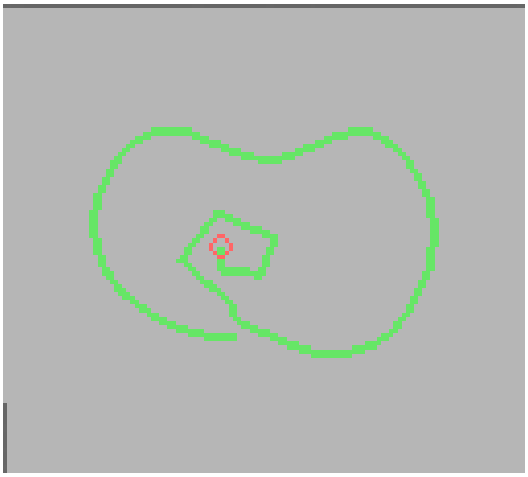


Figura 86: Pruebas en circuito 1 modificado

El helicóptero no encuentra ninguna línea al empezar a buscar, luego detecta que está perdido. Comienza por tanto a moverse en espiral, hasta que encuentra la línea y sigue el circuito.

La espiral que se sigue no es cuadrada exacta porque hemos modificado el ángulo de giro en los vértices a uno ligeramente superior a 90 grados. De esta manera conseguimos una espiral más amplia que recorre un área mayor en menos tiempo.

Circuito número 2:

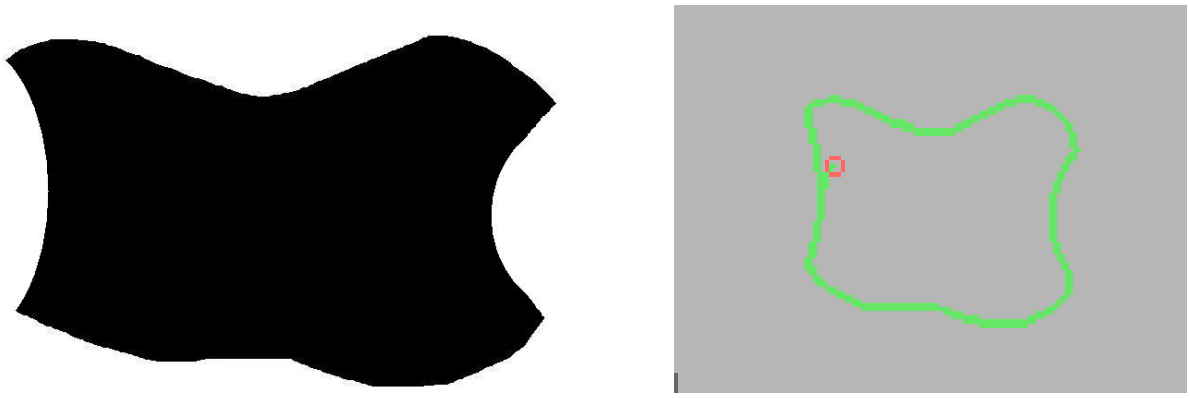


Figura 87: Pruebas en circuito 2

Se trata de una modificación sobre el circuito 1 incrementando la dificultad. Bordes que antes eran suaves ahora son puntiagudos, generando cambios bruscos que deben ser seguidos por el vehículo.

El resultado es satisfactorio: aunque no sigue el circuito hasta los bordes más agudos, mantiene la forma del mismo sin extraviarse, por lo que se puede considerar que ha englobado y reconocido la mancha con éxito.

- Circuito número 3:

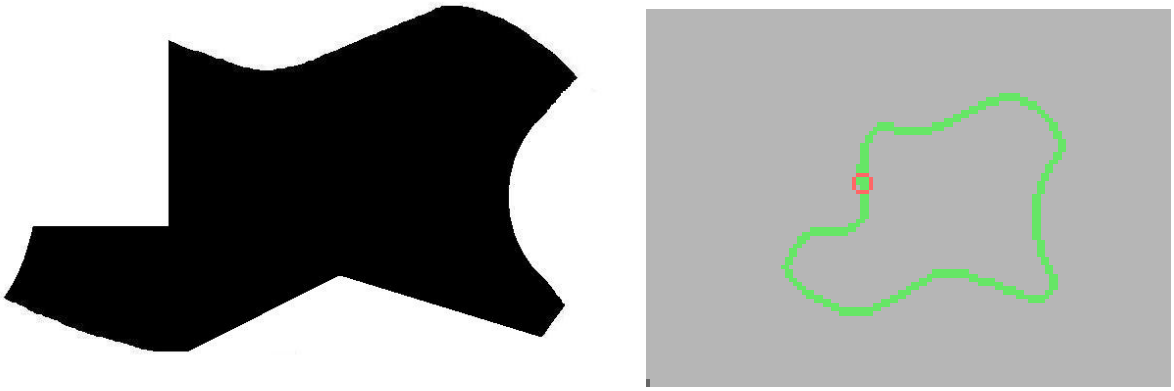


Figura 88: Pruebas en circuito 3

En esta última prueba se ha vuelto a modificar el circuito original, no solo poniendo picos y bordes bruscos, sino también giros de 90 grados para conseguir confundir al helicóptero.

Los resultados vuelven a ser positivos. El helicóptero, al igual que ocurría con el circuito anterior, suaviza las brusquedades y completa un perímetro de la mancha volviendo a su posición original sobre la misma y completando su objetivo.

5.2 Pruebas en el sistema real

A continuación mostraremos las pruebas más significativas cuando el controlador recibe el error del sistema real. La entrada, representada en verde manzana, indica el error respecto a la posición inicial que registran las cámaras a través del software de visión. A su vez, la salida, representada en color azul, representa la señal de salida que el software controlador envía a la emisora, es decir, el resultado de los controladores PID respecto a la entrada.

Las gráficas representan la evolución de ambas señales a lo largo del tiempo de simulación, permitiéndonos así conocer la reacción tanto de los controladores a la señal de error, como la reacción del UAV a las señales de control. La línea en color negro que cruza cada gráfica representa el error cero, es decir, siempre que la línea verde esté sobre la negra, significa que está en la posición en la que debería estar. Cuanto más alejadas estén de éstas, más error tiene el UAV respecto a la posición deseada.

Sin embargo, no nos hemos de dejar engañar por todo lo que vemos representado. Si recordamos la figura que explicamos previamente en el entorno de pruebas:

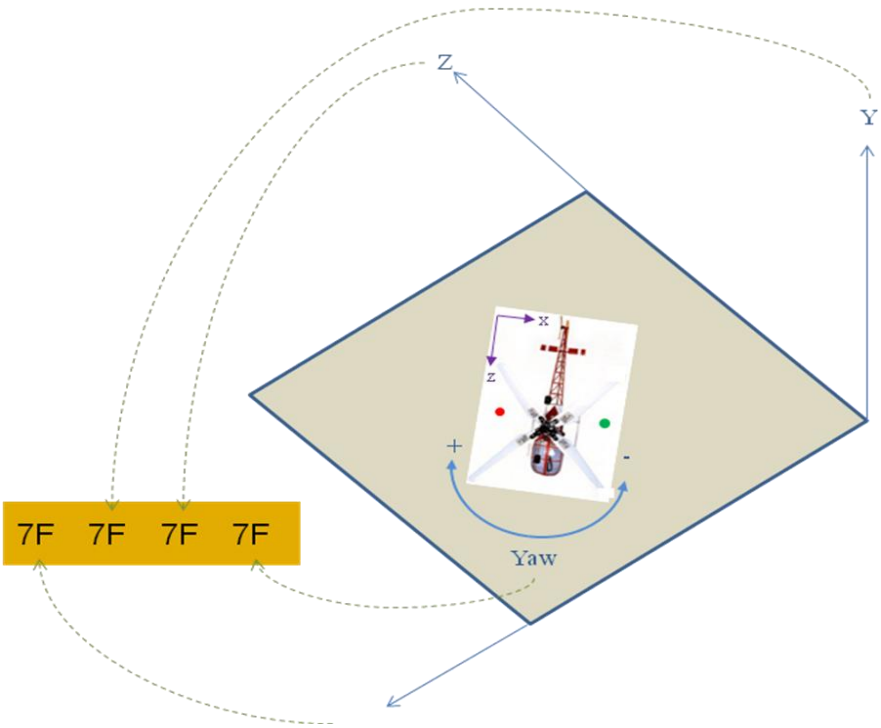
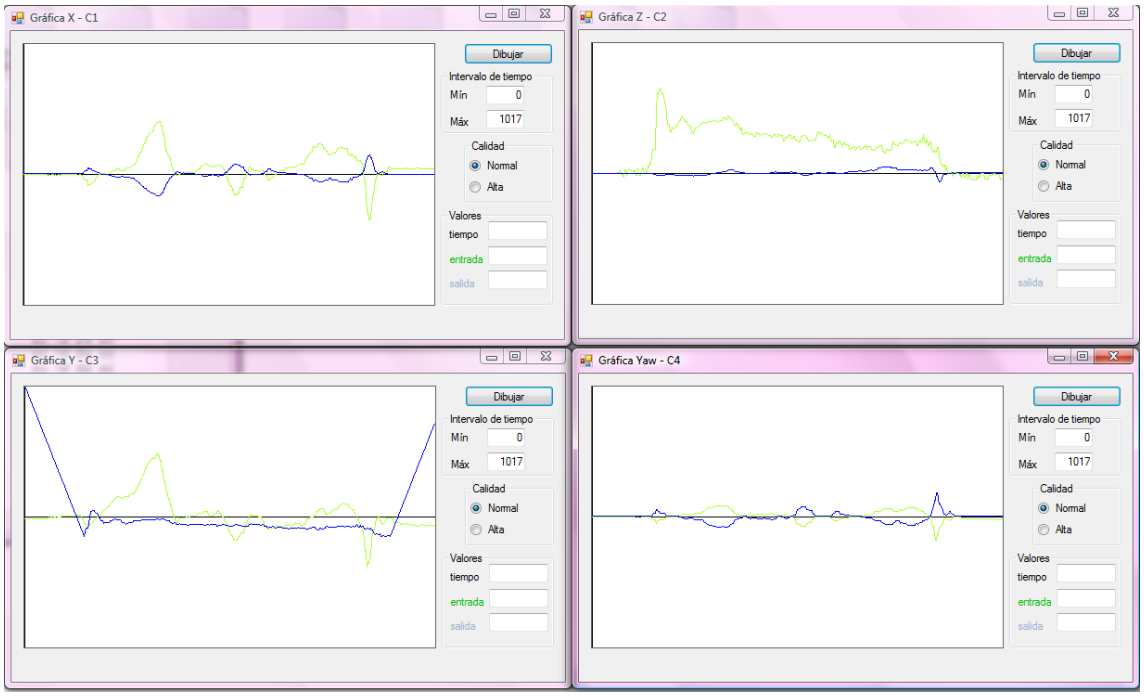


Figura 89: Gráfico de captura del error mediante las cámaras

Los errores que proporcionan las cámaras se ven respecto a los ejes fijos, mientras que la respuesta del controlador, en azul, viene dada respecto a los ejes del propio helicóptero.

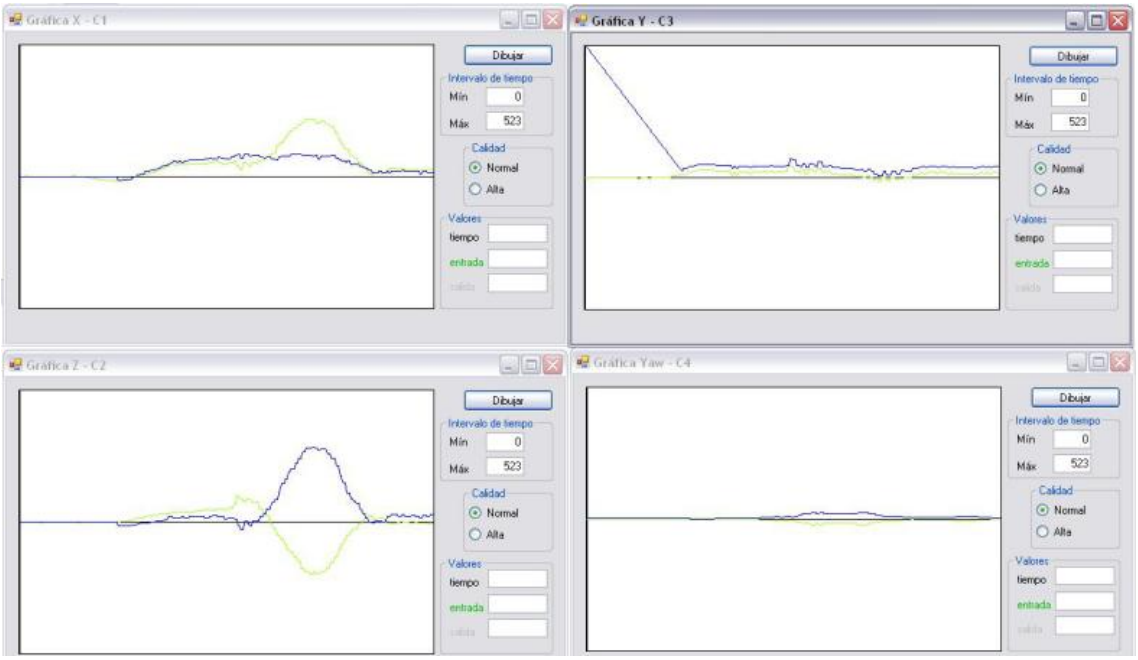
Se han elegido las dos pruebas de control más significativas entre un grupo de diez, en donde más claro se puede observar la función del controlador.

PRUEBA 1



En esta prueba, se aprecia cómo el control reacciona adecuadamente al error del UAV en los ejes X(arriba izquierda), Y(abajo izquierda) y Yaw(abajo derecha,pero no así en Z(arriba derecha). En el eje Y, se ve primero un aumento de la señal (está invertido) y al final un descenso, que corresponde a la rutina de despegue y aterrizaje implementada en el sistema de control.

PRUEBA 2



En esta ocasión se controla bien en todos los ejes, ya que el error en X que captan las cámararas y que aparentemente no se corrige es debido a un desplazamiento frontal del UAV. Como hemos explicado anteriormente, dicho desplazamiento se traduce en una corrección en el eje Z del helicóptero, pero no en el eje X, mientras que las cámaras lo captan tanto en X como en Z.

6 CONTROL DE CALIDAD

6.1 Control de calidad del producto

Con el fin de que el sistema sea fiable, y teniendo en cuenta que estamos trabajando en un entorno muy sensible, en el que cualquier error puede causar daños UAV, se han llevado a cabo los siguientes controles de calidad:

Rutinas de despegue y aterrizaje, al iniciar el sistema se lanzara una rutina de despegue que llevara de forma progresiva el buffer de altura hasta su valor medio, una vez alcanzado, comenzaran a funcionar los controles PID. Al cerrar el sistema, o ante un comportamiento inesperado, el foco de control tomará la rutina de aterrizaje que irá reduciendo el buffer de altura hasta parar los motores

Truncamiento de los valores de control en el propio PID, de esta forma cualquiera de los controladores PID no generara valores fuera del rango admisible por el UAV.

Control de picos en el envío, se tendrán en cuenta los valores generados con anterioridad en un controlador PID al generar el valor actual. Si el salto es muy grande, no se aplicará el PID, sino que se procederá a incrementar/decrementar la señal del ciclo anterior en dirección a la nueva señal de control, en una medida segura para el UAV y la emisora. De esta forma también se evita un fallo simple en la señal recibida desde el sistema de posicionamiento.

Lectura de eco de la emisora tras la escritura de los valores de control en ella. Estos valores deben coincidir con el eco, ya que de no ser así, sabremos que hay un error en el envió a la emisora, y procederemos a abortar la ejecución.

6.2 Control de calidad del código

Para asegurarnos de que entre los 3 miembros del grupo creamos un código legible y funcional, hemos seguido una serie de pautas a lo largo del desarrollo del proyecto:

Tratar de dar nombres representativos a las variables, para que alguien que no haya desarrollado esa parte del proyecto pueda saber a qué se refiere.

Los nombres de los métodos comenzarán por minúsculas, mientras que los de las clases por mayúsculas.

Separar en la medida de lo posible las constantes comunes que vayan a ser reutilizadas por otras clases, declarándolas en la clase de constantes correspondiente.

Independizar en la medida de lo posible las partes en las que se trabaje, haciendo que repercuta lo mínimo en lo escrito por los demás: desarrollo en módulos, (como podemos ver en la arquitectura de cualquiera de las 3 aplicaciones).

Que el código enviado no solo compile, sino que no haya repercutido negativamente en cualquiera de las funcionalidades del proyecto. Es decir, que antes de subir cualquier mejora, se prueben a fondo los cambios.

6.3 Riesgos

Al iniciar el proyecto, se estudiaron los posibles riesgos que se podrían encontrar y cómo solucionarlos. A continuación se detalla la lista resultante:

Riesgo	Probabilidad	Impacto
Incompatibilidad de horario ente los miembros del grupo	Muy alta	Tolerable
Discrepancias en el grupo	Baja	Serio
Mala estimación en el tiempo necesario para realizar las funciones principales del proyecto	Alta	Catastrófico
Falta de tiempo para realizar características secundarias	Alta	Tolerable
El personal no es capaz de adquirir los conocimientos.	Baja	Catastrófico
Bajas permanentes de los miembros del proyecto, (dejar el proyecto, abandono del mismo, etc.)	Baja	Catastrófico
Falta de coordinación entre los distintos miembros del equipo	Alta	Tolerable
El grupo con el que se va a integrar el controlador no entrega su parte a tiempo	Moderada	Catastrófico
La parte del grupo con el que se va a integrar el controlador falla	Moderada	Catastrófico

Incompatibilidad de horario ente los miembros del grupo:

Probabilidad	Muy alta
Impacto	Tolerable
Descripción	Debido a que los 3 miembros del grupo coincidimos en pocas clases, la única vez en donde discutimos presencialmente acerca del proyecto es en las reuniones semanales.
Consecuencias	No poder trabajar conjuntamente, y tener que realizar cosas por separado
Cómo evitarlo	Haciendo que no sea necesario que estén todos para seguir trabajando
Cómo tratarlo	Repartiendo bien el trabajo y mediante las reuniones semanales.
Eliminado	Desde el inicio del proyecto, fue de los primeros riesgos que se eliminaron

Discrepancias en el grupo:

Probabilidad	Baja
Impacto	Serio
Consecuencias	Mal ambiente de trabajo y resultados mediocres de la aplicación. Incluso si hay grandes discrepancias podría resultar en aplicaciones diferentes.
Cómo evitarlo	A lo largo del desarrollo del software, dar libertad de decisión a todos los miembros del grupo.
Cómo tratarlo	Cada miembro del grupo ha hecho hincapié en una parte del proyecto, acordando previamente cómo tenían que ser los resultados, y obteniendo libertad para desarrollar su parte.
Eliminado	Debido a la organización que hemos llevado a cabo, no ha llegado a surgir este riesgo

Mala estimación en el tiempo necesario para realizar las funciones principales del proyecto:

Probabilidad	Alta

Impacto	Catastrófico
Cómo evitarlo	Tener los pies en la tierra y no añadir características excesivamente complejas al programa.
Cómo tratarlo	Planificar con cuidado las tareas a realizar para
Eliminado	Gracias a las reuniones semanales, hemos eliminado este riesgo proponiendo cada semana los objetivos para la siguiente.

Falta de tiempo para realizar tareas secundarias:

Probabilidad	Alta
Impacto	Tolerable
Descripción	Debido a que nuestro proyecto es muy amplio, admite muchas mejoras en muchos campos, y con el tiempo que tenemos no es posible terminarlas todas
Consecuencias	El proyecto puede parecer incompleto
Cómo evitarlo	Una buena planificación y definición de los objetivos a alcanzar
Cómo tratarlo	Acordar fechas de entrega para ir revisando la evolución general del proyecto

El personal no es capaz de adquirir los conocimientos:

Probabilidad	Baja
Impacto	Catastrófico
Consecuencias	El sujeto no puede progresar, no siendo útil y no pudiendo desempeñar el trabajo necesario. El proyecto se detiene, y en el peor caso, se anula.
Cómo evitarlo	Reparto adecuado de las partes del proyecto teniendo en cuenta los conocimientos previos de cada miembro
Cómo tratarlo	Asignar al encargado de la tarea actual otra que sea de utilidad para el proyecto, y a su vez que otro miembro le releve de la suya.

Eliminado	Todos los desarrolladores han conseguido familiarizarse con su parte del trabajo, así como a involucrarse en las de los demás en caso de que hubiera sido necesario.

Bajas permanentes de los miembros del proyecto:

Probabilidad	Baja
Impacto	Catastrófico
Consecuencias	Siendo tan pocas personas para desarrollar el proyecto, el abandono de una de ellas supondría aumentar la carga de trabajo de las otras 2 enormemente, llegando incluso a hacerles abandonar también
Cómo evitarlo	Concienciar a todos de que somos un grupo y que hay que actuar como tal. Intentar organizar el trabajo de manera que una baja afecte lo menos posible al proyecto.
Cómo tratarlo	Reorganizar el trabajo entre los miembros posibles del grupo
Eliminado	No se ha producido, puesto que nadie ha abandonado.

Falta de coordinación entre los distintos miembros del grupo:

Probabilidad	Alta
Impacto	Tolerable
Descripción	Al tener bastante carga de trabajo y ser pocas personas, cuando se integra el conjunto es probable que si no se coordinan los resultados de cada miembro, haya mucha confusión
Consecuencias	Confusión acerca de cuál es la versión estable del proyecto, desarrollo sobre versiones antiguas en vez de las más recientes y entorpecimiento en general del proyecto.
Cómo evitarlo	Mediante reuniones donde quede claro qué está haciendo cada miembro, así como un control de versiones para conocer cuál es la última versión estable.
Cómo tratarlo	Mejorando la comunicación entre los miembros del grupo, llegando a un acuerdo acerca de cómo y qué desarrollar

Eliminado	Aunque llegamos en un principio a tener problemas de este tipo, el riesgo se eliminó mediante las reuniones semanales y el control de versiones.

El grupo con el que se va a integrar el controlador no entrega su parte a tiempo:

Probabilidad	Moderada
Impacto	Catastrófico
Consecuencias	El proyecto conjunto no podría ser implementado y no se podrían realizar las pruebas con el controlador.
Cómo evitarlo	Coordinar las actividades y exigir resultados para poder realizar pruebas a menor escala
Cómo tratarlo	Desarrollando por nuestra parte un software que permita simular el controlador en un sistema más o menos real
Eliminado	Se ha encontrado este riesgo, y se ha eliminado realizando la herramienta del simulador, que permite conectar por UDP al controlador y realizar vuelos con una dinámica realista.

La parte del grupo con el que hay que integrar el controlador falla:

Probabilidad	Moderada
Impacto	Catastrófico
Consecuencias	El proyecto conjunto no podría ser implementado y no se podría integrar el controlador
Cómo evitarlo	Coordinar con el otro grupo los resultados que se van obteniendo, así como las entradas que requiere el controlador, para ver que no es desmesurado
Cómo tratarlo	Utilizar nuestro software de prueba para que se ajuste de manera máxima al resultado real. Así una vez no ocurran los fallos la integración será inmediata

7 CONCLUSIONES

Los resultados obtenidos se pueden resumir en los siguientes puntos:

Se ha construido un controlador que permite conectarse tanto a un programa virtual de simulación como a un sistema de monitorización que observe la superficie del helicóptero y envíe los errores reales del vuelo. El controlador calcula en ambos casos las correcciones correspondientes basándose en sus parámetros de configuración, que se pueden ajustar por el usuario según convenga.

Se ha desarrollado un simulador que transmite a través de la red la posición del helicóptero y recibe las correcciones necesarias en función de la propia posición. Posee una dinámica realista, en donde se simula la fuerza de la gravedad sobre el cuatrimotor.

Se ha implementado un subsistema accesible desde el simulador que permite analizar la imagen que se obtiene desde el UAV para determinar si hay una línea o no; y en caso de haberla, seguirla. En caso contrario, se busca dicha línea con un movimiento en espiral.

Se ha desarrollado un generador de escenarios para crear mapas en los que realizar simulaciones, (introduciendo obstáculos o vehículos). Además, permite editar los que ya habíamos guardado, y pre visualiza los cambios en todo momento gracias al motor TrueVision3D.

Se ha desarrollado dos subsistemas de gestión de pruebas. Uno de ellos para el simulador, permitiendo guardar cualquier vuelo del helicóptero para un análisis posterior. El otro para el controlador, que muestra de forma gráfica los resultados del control para cada uno de los cuatro parámetros del helicóptero.

Ahora bien, ¿Hemos conseguido que el UAV se mantenga estable de forma completamente desatendida?

Tras un mes de pruebas, hemos comprobado que el helicóptero se estabiliza correctamente en todas las direcciones posibles, si bien siempre se comporta mejor en

altura y rotación. Por otra parte, no es un resultado extrapolable a cualquier entorno, dado que las pruebas se han hecho en un lugar cerrado, donde no se tenían en cuenta factores como el viento. Dichas situaciones podrían considerarse objeto de un trabajo futuro partiendo del modelo existente.

Trabajo futuro:

Con el controlador capaz de estabilizar el helicóptero en un punto, el siguiente paso sería desplazarlo en el espacio. Esta tarea resultaría sencilla debido a como está implementado el controlador, simplemente cambiando el punto de destino a un lugar cercano al entorno, lograríamos un movimiento del helicóptero.

Hay que tener en cuenta que la investigación se ha realizado en un principio para estabilizar el helicóptero en un punto, y que el comportamiento de helicóptero en movimiento será diferente al de este caso, pues la dinámica se comporta de manera distinta.

Lo conveniente sería realizar un vuelo de prueba e ir grabando todos los valores, con un control manual estable. Después, mediante la herramienta Matlab y con la serie de valores obtenidos de error, respuesta y posición real, se podría obtener la dinámica real del helicóptero en movimiento. Con esta dinámica y la herramienta Simulink de Matlab, deberíamos sintonizar los valores de los PIDs del controlador, hasta que consiguiéramos controlar el vehículo en movimiento de manera autónoma.

8 ANEXOS

8.1 Herramientas utilizadas

8.1.1 C#

C# es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.

La sintaxis deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET similar al de Java, aunque incluye mejoras derivadas de otros lenguajes (entre ellos Delphi).

C#, como parte de la plataforma.NET, está normalizado por ECMA desde diciembre de 2001 (ECMA-334 "Especificación del Lenguaje C#"). El 7 de noviembre de 2005 salió al mercado la versión 2.0 del lenguaje, que incluía mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables. El 19 de noviembre de 2007 salió la versión 3.0 de C# destacando entre las mejoras los tipos implícitos, tipos anónimos y el LINQ (Language Integrated Query).

C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma Visual Studio, aunque ya existe un compilador implementado que provee el Framework de GNU que genera programas para distintas plataformas como Win32, UNIX y Linux.

8.1.2 Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado (IDE, por sus siglas en inglés) para sistemas operativos Windows. Soporta varios lenguajes de programación tales como Visual C++, Visual C#, Visual J#, ASP.NET y Visual Basic .NET, aunque actualmente se han desarrollado las extensiones necesarias para muchos otros.

Visual Studio permite a los desarrolladores crear aplicaciones, sitios y aplicaciones Web, así como servicios Web en cualquier entorno que soporte la

plataforma .NET (a partir de la versión net 2002). Así se pueden crear aplicaciones que se intercomuniquen entre estaciones de trabajo, páginas Web y dispositivos móviles.

Desde el lanzamiento de Visual Studio 5.0 en 1997 han surgido en total 7 versiones incluyendo la versión beta 2010 que utilizara el framework 4.0 de reciente lanzamiento junto con el nuevo sistema operativo de Microsoft Windows 7, pero para el desarrollo del proyecto fueron elegidas las versiones 2005 y 2008 con total compatibilidad entre ellas, que utilizaran el Framework 3.5.

Elegimos este entorno porque es de los más rápidos y fiables para desarrollo en C#, necesario en el desarrollo porque permite llegar a niveles más bajos de la arquitectura del equipo que otros idiomas de programación como C o Java, y nosotros teníamos la necesidad de poder acceder directamente al puerto COM del equipo con el menor retardo posible, para el envió a la emisora.

8.1.3 TrueVision3D

TrueVision3D, también abreviado TV3D, es un motor escrito en Visual Basic6 y C++ que soporta Direct X. Es accesible desde un gran número de lenguajes de programación, como C++, C# o Delphi.

Se trata de un motor enfocado a la construcción de entornos de manera precisa pero con un precio razonable. Precisamente por esta razón se decidió optar por usar esta alternativa en el proyecto, ya que aunque no se ha llegado a comprar la licencia, se ofrece una versión gratuita en la página oficial.

Características:

Sistema de renderizado completo.

Soporte Shader HLSL (para el cálculo de la reacción de una superficie ante la luz).

Sistema de terreno y paisaje, con soporte para generar relieves a partir de una imagen.

Sistema de soporte de objetos, siendo el máximo soportado 16 millones de triángulos y vértices.

Objetos animados, que pueden ser importados de otras herramientas populares, como Maya, 3DS Max o MilkShape 3D. Además se incluye un sistema de colisiones y detección de los "clics" de ratón muy acertado.

- Materiales y sistema de luces.
- Sistemas de partículas, con múltiples emisores y atractores soportados.
- GPGPU, (soporte de punto flotante incluso para 64 y 128 bits).

Motor de físicas Newton, entre otras cosas, necesario para simular la integración directa entre objetos.

Ventajas:

Aparte de todo lo mencionado en las características, cabe destacar que hay mucha documentación en Internet sobre cómo manejar la herramienta, con lo que hemos aprendido a manejarlo relativamente rápido.

Hemos de hacer hincapié sobre todo en los tutoriales y la Wiki que se promocionan en la página oficial de la herramienta, ya que se relatan numerosos problemas sufridos por los usuarios que nos han ayudado a mejorar nuestra aplicación

Carencias:

La única carencia que hemos encontrado es que al tener la versión gratuita, aparece un pequeño logo representativo de la compañía, y que nos ha entorpecido a la hora de realizar cálculos basados en tratamiento de imagen.

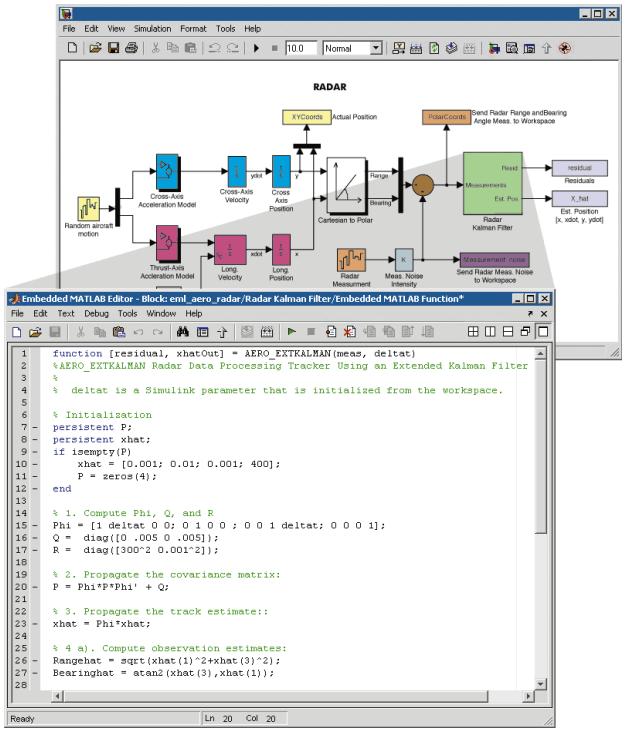
8.1.4 Matlab y Simulink

Simulink es un entorno de programación visual, que funciona sobre el entorno de programación Matlab.

Es un entorno de programación de más alto nivel de abstracción que el lenguaje interpretado Matlab (archivos con extensión .m). Simulink genera archivos con extensión .mdl (de "model").

Simulink viene a ser una herramienta de simulación de modelos o sistemas, con cierto grado de abstracción de los fenómenos físicos involucrados en los mismos. Se hace hincapié en el análisis de sucesos, a través de la concepción de sistemas (cajas negras que realizan alguna operación).

Se emplea arduamente en Ingeniería Electrónica en temas relacionados con el procesamiento digital de señales (DSP), involucrando temas específicos de ingeniería biomédica, telecomunicaciones, entre otros. También es muy utilizado en Ingeniería de Control y Robótica



Aplicación

El rango de aplicación de Simulink es casi ilimitado, podremos simular gracias a él, cualquier sistema continuo o discreto, como por ejemplo:

Esta imagen muestra un diagrama en bloques de un Radar, en el cuál se muestra que uno de sus bloques de procesamiento de señal, es un filtro Kalman realizado en un script de Matlab.

En las siguientes imágenes podemos apreciar un sistema de control automático, junto a su modelización y finalmente un sistema de un automóvil, vinculando la simulación a un entorno de realidad virtual.

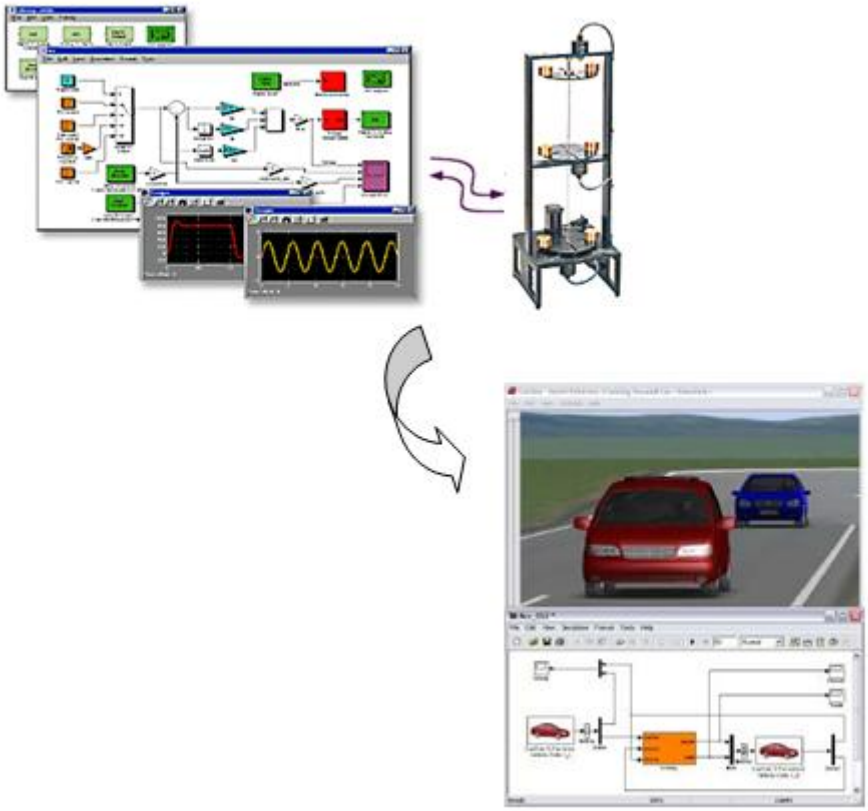


Figura 90: Simulación en Simulink de un automóvil

Gracias a Simulink hemos podido simular el comportamiento de nuestro sistema, mediante un sencillo modelo donde hemos podido cambiar los parámetros del PID con seguridad para el UAV real, y sintonizar estos de una manera fiable y bastante precisa. Siempre queda la pega de que el comportamiento simulado nunca podrá ser como el real, puesto que los parámetros que influyen en el modelo nunca serán los mismos de una sesión a otra.

8.1.5 Tortoise SVN

Definición:

Subversión es un software de sistema de control de versiones diseñado específicamente para reemplazar al popular CVS. Es software libre bajo una licencia de tipo Apache/BSD y se le conoce también como SVN por ser ese el nombre de la herramienta de línea de comandos. Una característica importante de Subversión es que, a diferencia de CVS, los archivos versionados no tienen cada uno un número de revisión independiente. En cambio, todo el repositorio tiene un único número de versión que identifica un estado común de todos los archivos del repositorio en cierto punto del tiempo.

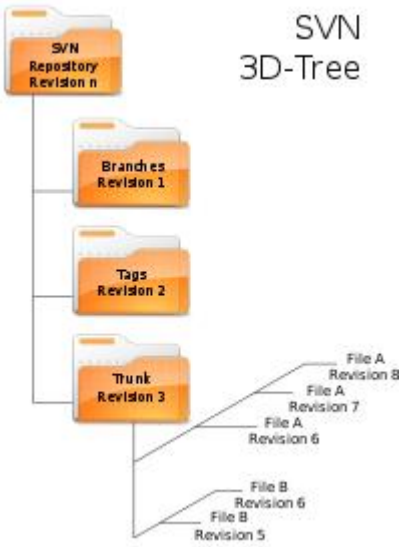


Figura 91: Árbol de control de versiones de SVN

Ventajas:

- Se sigue la historia de los archivos y directorios a través de copias y renombrados.
- Las modificaciones (incluyendo cambios a varios archivos) son atómicas.
- La creación de ramas y etiquetas es una operación más eficiente; Tiene costo de complejidad constante (O(1)) y no lineal (O(n)).
- Se envían sólo las diferencias en ambas.
- Puede ser servido mediante Apache, sobre WebDAV/DeltaV. Esto permite que clientes WebDAV utilicen Subversión en forma transparente.

- Maneja eficientemente archivos.
- Permite selectivamente el bloqueo de archivos. Se usa en archivos binarios que, al no poder fusionarse fácilmente, conviene que no sean editados por más de una persona a la vez.
- Cuando se usa integrado a Apache permite utilizar todas las opciones que este servidor provee a la hora de autentificar archivos (SQL, LDAP, PAM, etc.).

Subversión es muy conocido en la comunidad de software libre y se utiliza en muchos proyectos, incluyendo la fundación del software de Apache, KDE, GNOME, Free Pascal, FreeBSD, GCC, Python, Django, Ruby, Mono, SourceForge.net, ExtJS y Tigris.org. El servicio Google Code también proporciona almacenamiento Subversión para sus proyectos de software libre. Los sistemas de BountySource lo utilizan exclusivamente. Codeplex ofrece a acceso tanto para Subversión como para otros tipos de clientes. Subversión también está siendo adoptando en el mundo corporativo. En un informe 2007 de Forrester Research, reconocía a Subversión como el único líder en la categoría de sistema de control de versiones.

Gracias al sistema de subversión, hemos podido llevar un completo control de versiones del proyecto, pudiendo en cualquier momento volver a una versión anterior; Además, gracias a el sistema de diferencias que implementa, por el cual puedes ver dos versiones de un mismo archivo, marcando las diferencias entre ellas, hemos podido salvar algunos obstáculos y problemas, como los producidos al compilar en diferentes arquitecturas (por ejemplo entre x86 y x64 debido a las tablas de rutas de estas).

8.2 Ampliación teórica

8.2.1 Control PID

Un PID (Proporcional Integral Derivativo) es un mecanismo de control por realimentación que se utiliza en sistemas de control industriales. Un controlador PID corrige el error entre un valor medido y el valor que se quiere obtener calculándolo y luego sacando una acción correctora que puede ajustar al proceso acorde. El algoritmo de cálculo del control PID se da en tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional determina la reacción del error actual. El Integral genera una corrección proporcional a la integral del error, esto nos asegura que aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero. El Derivativo determina la reacción del tiempo en el que el error se produce. La suma de estas tres acciones es usada para ajustar al proceso vía un elemento de control como la posición de una válvula de control o la energía suministrada a un calentador, por ejemplo. Ajustando estas tres constantes en el algoritmo de control del PID, el controlador puede proveer un control diseñado para lo que requiera el proceso a realizar. La respuesta del controlador puede ser descrita en términos de respuesta del control ante un error, el grado el cual el controlador llega al "set point", y el grado de oscilación del sistema. Nótese que el uso del PID para control no garantiza control óptimo del sistema o la estabilidad del mismo. Algunas aplicaciones pueden solo requerir de uno o dos modos de los que provee este sistema de control. Un controlador PID puede ser llamado también PI, PD, P o I en la ausencia de las acciones de control respectivas. Los controladores PI son particularmente comunes, ya que la acción derivativa es muy sensible al ruido, y la ausencia del proceso integral puede evitar que se alcance al valor deseado debido a la acción de control.

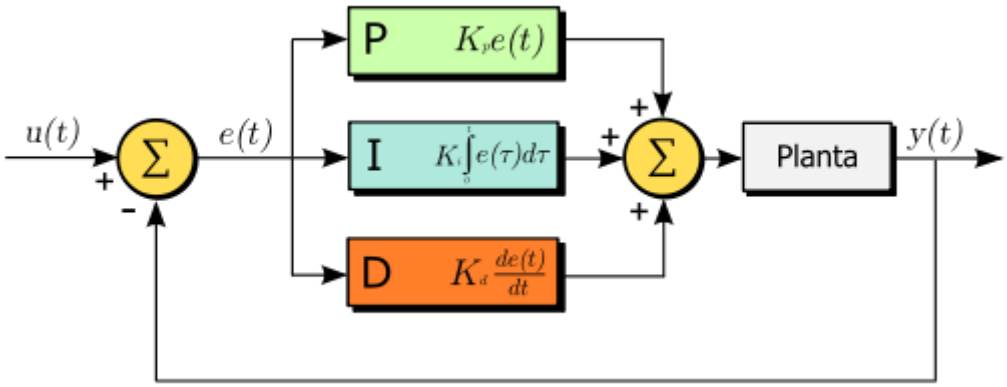


Figura 92: Diseño de un controlador PID

Funcionamiento

Para el correcto funcionamiento de un controlador PID que regule un proceso o sistema se necesita, al menos:

1. Un sensor, que determine el estado del sistema (termómetro, caudalímetro, manómetro, etc.).
2. Un controlador, que genere la señal que gobierna al actuador.
3. Un actuador, que modifique al sistema de manera controlada (resistencia eléctrica, motor, válvula, bomba, etc.).

El sensor proporciona una señal analógica o digital al controlador, la cual representa el punto actual en el que se encuentra el proceso o sistema. La señal puede representar ese valor en tensión eléctrica, intensidad de corriente eléctrica o frecuencia. En este último caso la señal es de corriente alterna, a diferencia de los dos anteriores, que son con corriente continua.

El controlador lee una señal externa que representa el valor que se desea alcanzar. Esta señal recibe el nombre de punto de consigna (o punto de referencia), la cual es de la misma naturaleza y tiene el mismo rango de valores que la señal que proporciona el sensor. Para hacer posible esta compatibilidad y que, a su vez, la señal pueda ser entendida por un humano, habrá que establecer algún tipo de interfaz (HMI - Human Machine Interface), son pantallas de gran valor visual y fácil manejo que se usan para hacer más intuitivo el control de un proceso.

El controlador resta la señal de punto actual a la señal de punto de consigna, obteniendo así la señal de error, que determina en cada instante la diferencia que hay entre el valor deseado (consigna) y el valor medido. La señal de error es utilizada por cada uno de los 3 componentes del controlador PID. Las 3 señales sumadas, componen la señal de salida que el controlador va a utilizar para gobernar al actuador. La señal resultante de la suma de estas tres se llama variable manipulada y no se aplica directamente sobre el actuador, sino que debe ser transformada para ser compatible con el actuador que usemos.

Las tres componentes de un controlador PID son: parte Proporcional, acción Integral y acción Derivativa. El peso de la influencia que cada una de estas partes tiene en la suma final, viene dado por la constante proporcional, el tiempo integral y el tiempo derivativo, respectivamente. Se pretenderá lograr que el bucle de control corrija eficazmente y en el mínimo tiempo posible los efectos de las perturbaciones.

Proporcional

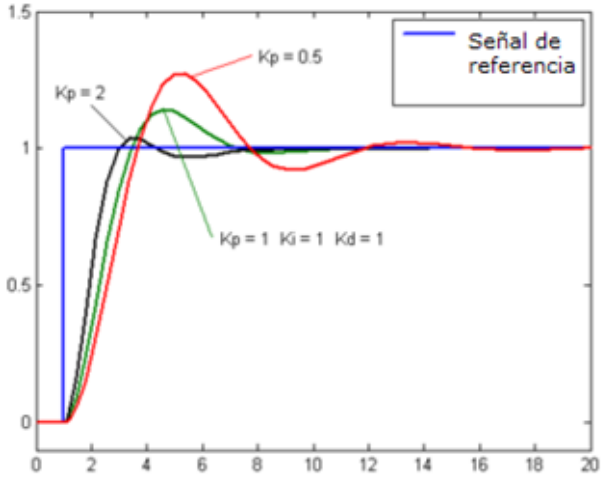


Figura 93: control proporcional de un sistema

La parte proporcional consiste en el producto entre la señal de error y la constante proporcional como para que hagan que el error en estado estacionario sea casi nulo, pero en la mayoría de los casos, estos valores solo serán óptimos en una determinada porción del rango total de control, siendo distintos los valores óptimos para cada porción del rango. Sin embargo, existe también un valor límite en la constante

proporcional a partir del cual, en algunos casos, el sistema alcanza valores superiores a los deseados. Este fenómeno se llama sobre oscilación y, por razones de seguridad, no debe sobrepasar el 30%, aunque es conveniente que la parte proporcional ni siquiera produzca sobreoscilación. Hay una relación lineal continua entre el valor de la variable controlada y la posición del elemento final de control (la válvula se mueve al mismo valor por unidad de desviación). La parte proporcional no considera el tiempo, por lo tanto, la mejor manera de solucionar el error permanente y hacer que el sistema contenga alguna componente que tenga en cuenta la variación respecto al tiempo, es incluyendo y configurando las acciones integral y derivativa.

La fórmula del proporcional está dada por: $P_{\mathbf{sal}} = K_p\, e(t)$

El error, la banda proporcional y la posición inicial del elemento final de control se expresan en tanto por uno. Nos indicará la posición que pasará a ocupar el elemento final de control.

Ejemplo: Cambiar la posición de la una válvula (elemento final de control) proporcionalmente a la desviación de la temperatura (variable) respeto al punto de consigna (variable deseada).

Integral

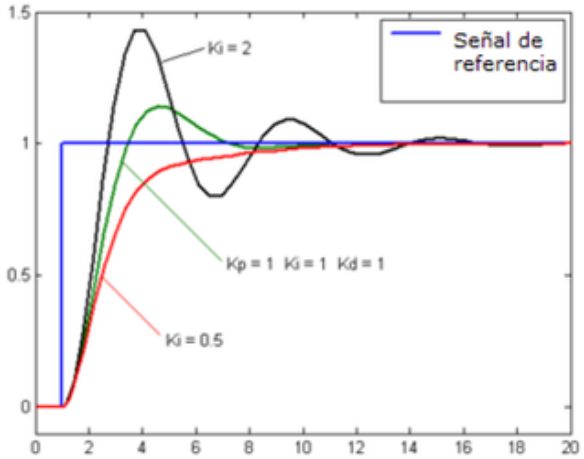


Figura 94: Control integral de un sistema

El modo de control Integral tiene como propósito disminuir y eliminar el error en estado estacionario, provocado por el modo proporcional. El control integral actúa cuando hay una desviación entre la variable y el punto de consigna, integrando esta desviación en el tiempo y sumándola a la acción proporcional. El error es integrado, lo cual tiene la función de promediarlo o sumarlo por un periodo de tiempo determinado; Luego es multiplicado por una constante **I**. **I** representa la constante de integración. Posteriormente, la respuesta integral es adicionada al modo Proporcional para formar el control P + I con el propósito de obtener una respuesta estable del sistema sin error estacionario.

El modo integral presenta un desfasamiento en la respuesta de 90º que sumados a los 180º de la retroalimentación (negativa) acercan al proceso a tener un retraso de 270º, luego entonces solo será necesario que el tiempo muerto contribuya con 90º de retardo para provocar la oscilación del proceso. <<< la ganancia total del lazo de control debe ser menor a 1, y así inducir una atenuación en la salida del controlador para conducir el proceso a estabilidad del mismo. >>> Se caracteriza por el tiempo de acción integral en minutos por repetición. Es el tiempo en que delante una señal en escalón, el elemento final de control repite el mismo movimiento correspondiente a la acción proporcional.

El control integral se utiliza para obviar el inconveniente del offset (desviación permanente de la variable con respeto al punto de consigna) de la banda proporcional.

La fórmula del integral está dada por: $I_{sal} = K_i \int_0^t e(\tau)\, d\tau$

Ejemplo: Mover la válvula (elemento final de control) a una velocidad proporcional a la desviación respeto al punto de consigna (variable deseada).

Derivativo

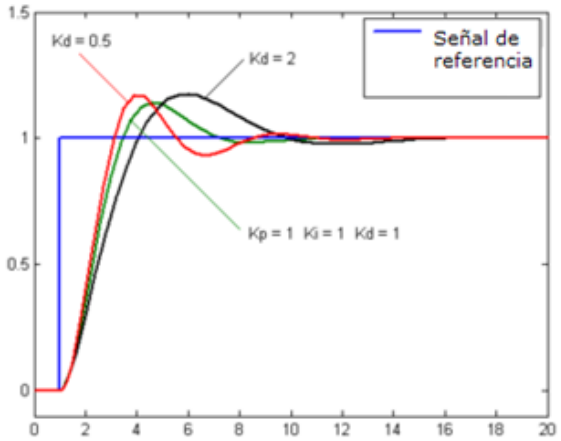


Figura 95: Control derivativo de un sistema

La acción derivativa se manifiesta cuando hay un cambio en el valor absoluto del error (si el error es constante, solamente actúan los modos proporcional e integral).

El error es la desviación existente entre el punto de medida y el valor consigna, o "Set Point".

La función de la acción derivativa es mantener el error al mínimo corrigiéndolo proporcionalmente con la misma velocidad que se produce; de esta manera evita que el error se incremente.

Se deriva con respecto al tiempo y se multiplica por una constante **D** y luego se suma a las señales anteriores (P+I). Es importante adaptar la respuesta de control a los cambios en el sistema ya que una mayor derivativa corresponde a un cambio más rápido y el controlador puede responder acordemente.

La fórmula del derivativo está dada por:

$$D_{\text{sal}} = K_d \frac{de}{dt}$$

El control derivativo se caracteriza por el tiempo de acción derivada en minutos de anticipo. La acción derivada es adecuada cuando hay retraso entre el movimiento de la válvula de control y su repercusión a la variable controlada.

Cuando el tiempo de acción derivada es grande, hay inestabilidad en el proceso. Cuando el tiempo de acción derivada es pequeño la variable oscila demasiado con

relación al punto de consigna. Suele ser poco utilizada debido a la sensibilidad al ruido que manifiesta y a las complicaciones que ello conlleva.

El tiempo óptimo de acción derivativa es el que retorna la variable al punto de consigna con las mínimas oscilaciones

Ejemplo: Corrige la posición de la válvula (elemento final de control) proporcionalmente a la velocidad de cambio de la variable controlada.

La acción derivada puede ayudar a disminuir el rebasamiento de la variable durante el arranque del proceso. Puede emplearse en sistemas con tiempo de retardo considerables, porque permite una repercusión rápida de la variable después de presentarse una perturbación en el proceso.

Significado de las constantes

P constante de proporcionalidad: se puede ajustar como el valor de la ganancia del controlador o el porcentaje de banda proporcional. Ejemplo: Cambia la posición de la válvula proporcionalmente a la desviación de la variable respecto al punto de consigna. La señal P, mueve la válvula siguiendo fielmente los cambios de temperatura multiplicados por la ganancia.

I constante de integración: indica la velocidad con la que se repite la acción proporcional.

D constante de derivación: hace presente la respuesta de la acción proporcional duplicándola, sin esperar a que el error se duplique. El valor indicado por la constante de derivación es el lapso de tiempo durante el cual se manifestará la acción proporcional correspondiente a 2 veces el error y después desaparecerá. Ejemplo: Mueve la válvula a una velocidad proporcional a la desviación respeto al punto de consigna. La señal I, va sumando las áreas diferentes entre la variable y el punto de consigna repitiendo la señal proporcional según el tiempo de acción derivada (minutos/repetición).

Tanto la acción Integral como la acción Derivativa, afectan a la ganancia dinámica del proceso. La acción integral sirve para reducir el error estacionario, que

existiría siempre si la constante Ki fuera nula. Ejemplo: Corrige la posición de la válvula proporcionalmente a la velocidad de cambio de la variable controlada. La señal d, es la pendiente (tangente) por la curva descrita por la variable.

La salida de estos tres términos, el proporcional, el integral, y el derivativo son sumados para calcular la salida del controlador PID. Definiendo u (t) como la salida del controlador, la forma final del algoritmo del PID es:

$$\mathrm{u(t)} = \mathrm{MV(t)} = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de}{dt}$$

Ecuación 3: salida de un controlador PID

Usos

Por tener una exactitud mayor a los controladores proporcional, proporcional derivativo y proporcional integral se utiliza en aplicaciones más cruciales tales como control de presión, flujo, fuerza, velocidad, en muchas aplicaciones química, y otras variables. Además es utilizado en reguladores de velocidad de automóviles (control de crucero o cruise control), control de ozono residual en tanques de contacto.

Ajuste de parámetros del PID

El objetivo de los ajustes de los parámetros PID es lograr que el bucle de control corrija eficazmente y en el mínimo tiempo los efectos de las perturbaciones; se tiene que lograr la mínima integral de error. Si los parámetros del controlador PID (la ganancia del proporcional, integral y derivativo) se eligen incorrectamente, el proceso a controlar puede ser inestable, por ejemplo, que la salida de este varíe, con o sin oscilación, y está limitada solo por saturación o rotura mecánica. Ajustar un lazo de control significa ajustar los parámetros del sistema de control a los valores óptimos para la respuesta del sistema de control deseada. El comportamiento óptimo ante un cambio del proceso o cambio del "setpoint" varía dependiendo de la aplicación. Generalmente, se requiere estabilidad ante la respuesta dada por el controlador, y este no debe oscilar ante ninguna combinación de las condiciones del proceso y cambio de "setpoints". Algunos procesos

tienen un grado de no-linealidad y algunos parámetros que funcionan bien en condiciones de carga máxima no funcionan cuando el proceso está en estado de "sin carga". Hay varios métodos para ajustar un lazo de PID. El método más efectivo generalmente requiere del desarrollo de alguna forma del modelo del proceso, luego elegir P, I y D basándose en los parámetros del modelo dinámico. Los métodos de ajuste manual pueden ser muy ineficientes. La elección de un método dependerá de si el lazo puede ser "desconectado" para ajustarlo, y del tiempo de respuesta del sistema. Si el sistema puede desconectarse, el mejor método de ajuste a menudo es el de ajustar la entrada, midiendo la salida en función del tiempo, y usando esta respuesta para determinar los parámetros de control. Ahora describimos como realizar un ajuste manual.

Ajuste manual

Si el sistema debe mantenerse “online”, un método de ajuste es el de primero fijar los valores de I y D a cero. Incrementar P hasta que la salida del lazo oscile, luego P debe ser configurada a aproximadamente la mitad del valor configurado previamente. Ahora incrementar D hasta que el proceso se ajuste en el tiempo requerido aunque subir mucho D puede causar inestabilidad. Finalmente, incrementar I, si se necesita, hasta que el lazo sea lo suficientemente rápido para alcanzar su referencia luego de una variación brusca de la carga. Un lazo de PID muy rápido tiene como ventaja alcanza su “setpoint” de manera veloz, aunque algunos sistemas no son capaces de aceptar este disparo brusco; en estos casos se requiere de otro lazo con un P menor a la mitad del P del sistema de control anterior.

Limitaciones de un control PID

Mientras que los controladores PID son aplicables a la mayoría de los problemas de control, puede ser pobres en otras aplicaciones. Los controladores PID, cuando se usan solos, pueden dar un desempeño pobre cuando la ganancia del lazo del PID debe ser reducida para que no se dispare u oscile sobre el valor del "setpoint". El desempeño del sistema de control puede ser mejorado combinando el lazo cerrado de un control PID con un lazo abierto. Conociendo el sistema (como la aceleración necesaria o la inercia) puede ser avanaccionado y combinado con la salida del PID para aumentar el

desempeño final del sistema. Solamente el valor de avanacción puede proveer la mayor porción de la salida del controlador. El controlador PID puede ser usado principalmente para responder a cualquier diferencia o "error" que quede entre el setpoint y el valor actual del proceso. Como la salida del lazo de avanacción no se ve afectada a la realimentación del proceso, nunca puede causar que el sistema oscile, aumentando el desempeño del sistema, su respuesta y estabilidad.

Por ejemplo, en la mayoría de los sistemas de control con movimiento, para acelerar una carga mecánica, se necesita de más fuerza (o torque) para el motor. Si se usa un lazo PID para controlar la velocidad de la carga y manejar la fuerza o torque necesaria para el motor, puede ser útil tomar el valor de aceleración instantánea deseada para la carga, y agregarla a la salida del controlador PID. Esto significa que sin importar si la carga está siendo acelerada o desacelerada, una cantidad proporcional de fuerza está siendo manejada por el motor además del valor de realimentación del PID. El lazo del PID en esta situación usa la información de la realimentación para incrementar o decrementar la diferencia entre el setpoint y el valor del primero. Trabajando juntos, la combinación avanacción-realimentación provee un sistema más confiable y estable.

Otro problema que posee el PID es que es lineal. Principalmente el desempeño de los controladores PID en sistemas no lineales es variable. También otro problema común que posee el PID es, que en la parte derivativa, el ruido puede afectar al sistema, haciendo que esas pequeñas variaciones, hagan que el cambio a la salida sea muy grande. Generalmente un Filtro pasa bajo ayuda, ya que removería las componentes de alta frecuencia del ruido. Sin embargo, un FPB y un control derivativo pueden hacer que se anulen entre ellos. Alternativamente, el control derivativo puede ser sacado en algunos sistemas sin mucha pérdida de control. Esto es equivalente a usar un controlador PID como PI solamente.

Conclusiones

Los sistemas PID pueden ser (y son) utilizados en aproximadamente un 80 % de los controles de procesos, debido a su suficiente flexibilidad como para alcanzar excelentes resultados a un precio bajo con respeto a sus competidores.

El uso de los modos de control, es siempre conforme a las características del proceso, lo cual significa que debemos entender bien la operación del proceso antes de automatizarlo y de proceder a las rutinas de los algoritmos de control, veamos unos ejemplos:

El modo **On-Off**, es un caso especial del modo proporcional aplicable solamente a un proceso estático, ya que la ganancia del On-Off es infinita (B.P.=0). Cuando solo deseamos dar estabilidad al proceso, el modo proporcional es suficiente.

Ahora sabemos que el modo proporcional tiene la desventaja de producir un error estacionario (stand by response), para corregirlo es necesario hacerlo integrando el error y ésta es una función del tiempo f (t) = 1 / Integrale (t). Siendo I la denominada constante de integración que representa la ganancia con la que el modo integral contribuye.

¿Cuándo se debería aplicar la acción derivativa? Cuando exista un cambio de carga y no podamos esperar a que la acción integral corrija el error por si sola, entonces medimos la velocidad con la que se produce el error y el controlador responderá con la rapidez necesaria para evitar que el error aumente.

Por último, los parámetros de diseño por lo general son totalmente rígidos y basados en objetivos y criterios de diseño, con los que muchas veces no se pueden cumplir o que se requiere de soluciones extremadamente complejas, por lo tanto se debe saber y tener muy en cuenta entre lo que se quiere hacer y lo que los recursos tecnológicos nos pueden ofrecer.

8.2.2 Dinámica

La dinámica es la parte de la física que describe la evolución en el tiempo de un sistema físico en relación a las causas que provocan los cambios de estado físico y/o

estado de movimiento. La forma en que nos interesa la dinámica es para plantear ecuaciones de movimiento o ecuaciones de evolución para dicho sistema, para posteriormente,

Cálculo en dinámica

A través de los conceptos de desplazamiento, velocidad y aceleración es posible describir los movimientos de un cuerpo u objeto sin considerar cómo han sido producidos, disciplina que se conoce con el nombre de cinemática. Por el contrario, la dinámica es la parte de la mecánica que se ocupa del estudio del movimiento de los cuerpos sometidos a la acción de las fuerzas.

El cálculo dinámico se basa en el planteamiento de ecuaciones del movimiento y su integración. Para problemas extremadamente sencillos se usan las ecuaciones de la mecánica newtoniana directamente auxiliados de las leyes de conservación. La ecuación esencial de la dinámica es F=m*a donde F es la resultante de las fuerzas aplicadas, el m la masa y la a la aceleración.

Leyes de conservación

Las leyes de conservación pueden formularse en términos de teoremas que establecen bajo qué condiciones concretas una determinada magnitud "se conserva" (es decir, permanece constante en valor a lo largo del tiempo a medida que el sistema se mueve o cambia con el tiempo). Además de la ley de conservación de la energía las otras leyes de conservación importante toman la forma de teoremas vectoriales. Estos teoremas son:

El **teorema de la cantidad de movimiento**, que para un sistema de partículas puntuales requiere que las fuerzas de las partículas sólo dependan de la distancia entre ellas y estén dirigidas según la línea que las une. En mecánica de medios continuos y mecánica del sólido rígido pueden formularse teoremas vectoriales de conservación de cantidad de movimiento.

El **teorema del momento cinético**, establece que bajo condiciones similares al anterior teorema vectorial la suma de momentos de fuerza respecto a un eje es igual a la variación temporal del momento angular.

Conceptos relacionados con la dinámica

Inercia:

La inercia es la dificultad o resistencia que opone un sistema físico o un sistema social a posibles cambios.

En física se dice que un sistema tiene más inercia cuando resulta más difícil lograr un cambio en el estado físico del mismo. Los dos usos más frecuentes en física son la inercia mecánica y la inercia térmica. La primera de ellas aparece en mecánica y es una medida de dificultad para cambiar el estado de movimiento o reposo de un cuerpo. La inercia mecánica depende de la cantidad de masa y del tensor de inercia. La inercia térmica mide la dificultad con la que un cuerpo cambia su temperatura al estar en contacto con otros cuerpos o ser calentado. La inercia térmica depende de la cantidad de masa y de la capacidad calorífica.

Las llamadas fuerzas de inercia son fuerzas ficticias o aparentes que un observador en un sistema de referencia no-inercial....

La masa inercial es una medida de la resistencia de una masa al cambio en velocidad en relación con un sistema de referencia inercial. En física clásica la masa inercial de partículas puntuales se define por medio de la siguiente ecuación, donde la partícula uno se toma como la unidad (m1 =1): donde mi es la masa inercial de la partícula i, y ai1 es la aceleración inicial de la partícula i, en la dirección de la partícula i hacia la partícula 1, en un volumen ocupado sólo por partículas i y 1, donde ambas partículas están inicialmente en reposo y a una distancia unidad. No hay fuerzas externas pero las partículas ejercen fuerza las unas en las otras.

8.2.3 Helicóptero

8.2.3.1 Evolución Histórica

El **helicóptero** es una aeronave sustentada, al contrario de los aviones, por un conjunto de aspas giratorias, más conocido como hélice o rotor, situado en la parte superior del aparato. Esta aeronave es propulsada horizontalmente mediante la inclinación del rotor y la variación del ángulo de ataque de sus aspas. La palabra helicóptero proviene de las griegas helix (hélice) y pteron (ala), y fue acuñado en 1863 por el pionero de la aviación Gustave Ponton d'Amécourt, por lo que deriva del francés hélicoptère («ala en hélice»).

La idea del helicóptero es muy anterior a la del autogiro, inventado por el español Juan de la Cierva, aeronave con la que tiene sólo cierta similitud externa. Sin embargo, los primeros helicópteros pagaron patente y derechos de utilización del rotor articulado, original del ingeniero español. También se tomaron ideas del genio italiano Leonardo da Vinci, pero el inventor del primer helicóptero pilotado y motorizado fue el eslovaco Jan Bahyl. El primer aparato controlable totalmente en vuelo y producido en cadena fue fabricado por Igor Sikorsky en 1942.

Comparado con otros tipos de aeronave como el avión, el helicóptero es mucho más complejo, tiene un mayor coste de fabricación, uso y manutención, es relativamente lento, tiene menos autonomía de vuelo y menor capacidad de carga. No obstante, todas estas desventajas se ven compensadas por otras de sus características, como su gran maniobrabilidad y la capacidad de mantenerse estático en el aire, girar sobre sí mismo y despegar y aterrizar verticalmente. Si no se consideran aspectos tales como la posibilidad de repostaje o las limitaciones de carga y de altitud, un helicóptero puede viajar a cualquier lugar y aterrizar en cualquier sitio que tenga la suficiente superficie (dos veces la ocupada por el aparato).

Los orígenes

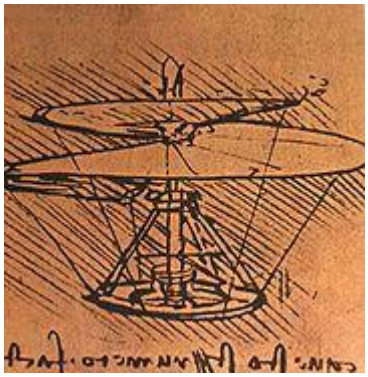


Figura 96: Máquina voladora de Leonardo DaVinci

Existe una historia que dice que en el año 500 A. C., técnicos chinos ya diseñaron un "trompo volador", juguete que consistía en un palo con una hélice acoplada a un extremo que, al girar entre las manos, se elevaba a la vez que giraba rápidamente; sería el primer antecedente del fundamento del helicóptero.

Hacia el año 1490, Leonardo da Vinci fue la primera persona que diseñó y dibujó en unos bocetos un artefacto volador con un rotor helicoidal, pero hasta la invención del avión motorizado en el siglo XX no se iniciaron los esfuerzos dirigidos a lograr una aeronave de este tipo. Personas como Jan Bahyl, Enrico Forlanini, Oszkár Asbóth, Louis Breguet, Paul Cornu, Emile Berliner, Ogneslav Kostovic, Federico Cantero, Stepanovic e Igor Sikorsky desarrollaron este tipo de aparato, a partir del autogiro de Juan de la Cierva, inventado en 1923. El primer vuelo de un helicóptero medianamente controlable fue realizado por el argentino Raúl Pateras de Pescara en 1916 en Buenos Aires, Argentina.[1] En 1931 los ingenieros aeronáuticos soviéticos Boris Yuriev y Alexei Cheremukhin comenzaron sus experimentos con el helicóptero TsAGI 1-EA, el primer aparato conocido con un rotor simple, el cual alcanzó una altitud de 605 metros el 14 de agosto de 1932, con Cheremukhin en los controles.

Primeros tiempos

La Alemania nazi usó el helicóptero a pequeña escala durante la Segunda Guerra Mundial. Modelos como el Flettner FL 282 Kolibri fueron usados en el Mar Mediterráneo. La producción en masa del Sikorsky XR-4 comenzó en mayo de 1942

gracias a la armada de los Estados Unidos. El aparato fue usado para operaciones de rescate en Birmania. También fue utilizado por la Royal Air Force. La primera unidad británica en ser equipada con helicópteros fue la escuela de entrenamiento para Helicópteros (Helicopter Training School, en inglés) constituida en enero de 1945 en Andover, con nueve helicópteros Sikorsky R-4B Hoverfly I.

Figura 97: Flettner FL 282 Kolibri

El Bell 47, diseñado por Arthur Young, se convirtió en el primer helicóptero en ser autorizado para uso civil (mayo de 1946) en los Estados Unidos y veinte años más tarde el Bell 206 llegó a ser el más exitoso helicóptero comercial jamás fabricado y el que más récords industriales estableció y rompió.

Los helicópteros capaces de realizar un planeo estable de forma fiable fueron desarrollados décadas más tarde que el avión de alas fijas. Esto se debió en gran parte a la mayor necesidad de potencia en el motor de los primeros respecto a los segundos (Sikorsky, por ejemplo, retrasó sus investigaciones en los helicópteros a la espera de que hubiera mejores motores disponibles en el mercado). Las mejoras en combustibles y motores durante la primera mitad del siglo XX fueron un factor decisivo en el desarrollo de los helicópteros. La aparición de los motores de turbo eje en la segunda mitad del siglo XX condujo al desarrollo de helicópteros más rápidos, mayores y capaces de volar a mayor altura. Estos motores se usan en la gran mayoría de los helicópteros excepto, a veces, en modelos pequeños o con un coste de fabricación muy bajo.

8.2.3.2 Funcionamiento

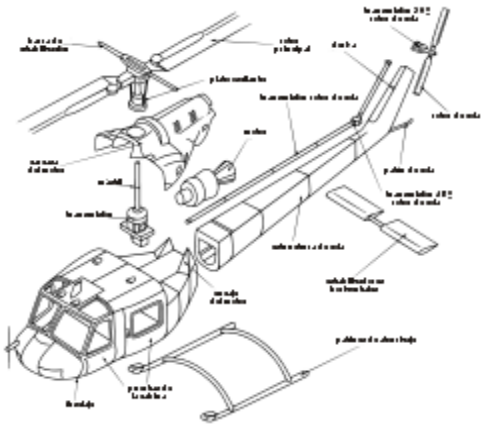


Figura 98: Despiece helicóptero militar americano

Rotor

Las palas del rotor tienen una forma aerodinámica similar a las alas de un avión, es decir, curvadas formando una elevación en la parte superior, y lisas o incluso algo cóncavas en la parte inferior (perfil alar). Al girar el rotor esta forma hace que se genere sustentación, la cual eleva al helicóptero. La velocidad del rotor principal es constante, y lo que hace que un helicóptero ascienda o descienda es la variación en el ángulo de ataque que se da a las palas del rotor: a mayor inclinación, mayor sustentación y viceversa.

Una vez en el aire, el helicóptero tiende a dar vueltas sobre su eje vertical en sentido opuesto al giro del rotor principal. Para evitar que esto ocurra, salvo que el piloto lo quiera, los helicópteros disponen en un lado de su parte posterior de una hélice más pequeña, denominada rotor de cola, dispuesta verticalmente, que compensa con su empuje la tendencia a girar del aparato y lo mantiene en una misma orientación.

Hay helicópteros que no tienen rotor de cola vertical, sino dos grandes rotores horizontales. En este caso, los rotores giran en direcciones opuestas y no se necesita el efecto "antipar" del rotor de cola como en los helicópteros de un solo rotor.

El rotor principal no sólo sirve para mantener el helicóptero en el aire (estacionario), así como para elevarlo o descender, sino también para impulsarlo hacia

adelante o hacia atrás, hacia los lados o en cualquier otra dirección. Esto se consigue mediante un mecanismo complejo que hace variar el ángulo de incidencia (inclinación) de las palas del rotor principal dependiendo de su posición.

Imaginemos un rotor, que gira a la derecha con velocidad constante. Si todas las palas tienen el mismo ángulo de incidencia (30º por ejemplo), el helicóptero empieza a subir hasta que se queda en estacionario. Las palas tienen durante todo el recorrido de los 360º, el mismo ángulo y el helicóptero se mantiene en el mismo sitio.

Movimiento

Si hacemos que las palas, únicamente al pasar por el sector 0º a 180º aumenten ligeramente su ángulo de incidencia y luego vuelvan a su inclinación original, el empuje del rotor será mayor en el sector de 0º a 180º y el helicóptero en vez de mantenerse parado, tiende a inclinarse hacia el sector donde las palas tiene menor empuje por la diferencia de sustentación que existe entre los sectores (en este caso, a la izquierda) produciendo así que el empuje total se realice de manera inclinada pudiendo desplazar en aparato en función del coseno del ángulo del vector de la tracción de las palas del helicóptero. Si las palas aumentan el ángulo de incidencia en el sector de 270º a 90º, el empuje será mayor por la parte trasera y el helicóptero tiende a inclinarse para adelante, bajando el morro y subiendo la cola, produciendo el movimiento hacia adelante, al igual que en el caso anterior.

Los helicópteros no varían la velocidad de las palas ni inclinan el eje del rotor para desplazarse. Lo que hacen es variar ligeramente y de forma cíclica el paso (inclinación) de las palas con respecto al que ya tienen todas (el colectivo de las palas). Ese aumento cíclico en un sector, hace que el helicóptero se desplace hacia el lado opuesto. Ahora se entenderá mejor porqué el mando de dirección de un helicóptero se llama cíclico y el mando de potencia se llama colectivo.

Además de estos controles de vuelo, el helicóptero usa los pedales para girar cuando está en estacionario. Esto se logra aumentando o disminuyendo el paso de las palas del rotor de cola, con lo que se consigue que el rotor de cola tenga más o menos empuje y haga girar al helicóptero hacia un lado u otro.

Los helicópteros también planean, y de hecho es lo que hacen en caso de necesidad para aterrizar en caso de emergencia. El rotor se comporta como una cometa y el helicóptero se transforma en un autogiro.

Durante el descenso, el flujo de aire hace girar a las palas que se transforman en una especie de "ala", y al llegar cerca del suelo, la velocidad de las palas se aprovecha para obtener sustentación y así disminuir la velocidad de descenso hasta posarse en el suelo suavemente. Esto se llama autorrotación.

Es importante saber también que el sonido característico de los helicópteros no suele ser por el motor, es hecho por el rotor. Al girar a una velocidad muy alta, hace que los extremos de las aspas vayan más rápido que la velocidad del sonido, o sea, supersónicas, por lo que el sonido producido son pequeños choques subsónicos contra el aire.

Figura 99: Diagrama de los ejes de movimiento

8.2.4 Emisora

Existen emisoras desde 2 canales hasta 14 canales.

Para pilotar un helicóptero con motor necesitamos un mínimo de 3 canales (para un velero sin motorización pueden bastar con dos canales), uno para la altura, otro para la profundidad y otro para el yaw pero lo normal es necesitar de 4 canales para controlar también el pitch. A la hora de comprar una emisora si nuestra economía nos lo permite es recomendable comprarla de al menos 6 canales para disponer de dos canales adicionales para accionar por ejemplo un tren de aterrizaje retráctil o unos flaps por ejemplo. Cuantos más canales dispongamos más operaciones podremos realizar.

Cuando compramos un equipo de radio para nuestro helicóptero éste ya se compone de todo lo necesario para poder pilotar un vehículo de 4 canales, emisora, receptor, interruptor de encendido de la parte del equipo instalada en el vehículo, una caja para las pilas y 4 servos. Es aconsejable no utilizar pilas sueltas en el avión con la caja suministrada (porta pilas) ya que a veces puede dejar de hacer contacto una de las pilas dejando de responder el receptor. Para evitar esto prepararemos un pack de baterías uniendo en serie las cuatro pilas (en el caso de alimentar el receptor con 4,8 V) con trenza de cobre (como la que se usa en los contactos de los coches de scalextric) soldada con estaño, no obstante últimamente los equipos de radio ya se suelen suministrar con un pack de baterías para la emisora y para el receptor.

Una de las características principales de un equipo de radio es la forma en la que emite, es decir, como codifica la información a enviar en la señal que emite. Podemos distinguir los siguientes tipos:

Amplitud modulada (AM)

La modulación de amplitud es un tipo de modulación lineal en la que se hace variar la amplitud de la onda portadora. Los cambios en la onda portadora se ajustan con las variaciones de nivel de la señal moduladora que es la que contiene la información a transmitir. La ventaja de AM es que su demodulación es muy simple con lo que los receptores son muy sencillos y baratos. Este método de transmitir es poco fiable y su uso queda relegado al automodelismo y al modelismo naval.

Frecuencia modulada (FM)

La frecuencia modulada (FM) o la modulación de frecuencia es una modulación angular que transmite información a través de una onda portadora variando su frecuencia. La emisión se realiza en banda estrecha con lo que los equipos de FM son menos sensibles a las interferencias.

Ya hemos visto que para aeromodelismo se emplea la FM y ahora hemos de diferenciar entre los dos tipos de modulación para la FM:

Modulación por Posición de Pulsos (PPM)

Este tipo de modulación es el más extendido y se caracteriza porque la señal emitida es analógica, tiene la ventaja de que podemos emplear receptores y emisoras de distintas marcas ya que la compatibilidad es 100% (siempre que sean ambos PPM y trabajen en la misma banda y frecuencia). Tiene el inconveniente de que el número de canales es limitado. En cuanto a la decodificación de la señal en el receptor podemos encontrarnos con "**single conversion**" o "**dual conversion**". Los que emplean "single conversion" son más sensibles con la recepción de señales de las frecuencias adyacentes que estén usando otros pilotos.

Figura 100: Emisora de 4 canales Futaba Skysport (FM/PPM)

Modulación por Impulsos Codificados (PCM)

La Modulación por Impulsos Codificados (MIC o PCM por sus siglas inglesas de Pulse Code Modulation), es un procedimiento de modulación utilizado para transformar una señal analógica en una secuencia de bits. Con esta modulación la comunicación entre emisora y receptor está más libre de interferencias por ser digital. Tanto la emisora como el receptor están provistos de microprocesadores, son pues equipos de radio de media-alta gama. En las transmisiones se envía siempre un código de comprobación para descartar posibles interferencias. Es por esto que no hay compatibilidad entre distintas marcas ya que cada una emplea su código propio. Muchos receptores de este tipo disponen de failsafe que entrará en juego si otro piloto comienza a utilizar nuestro canal evitando que el avión se vuelva incontrolable, lo que hace el **failsafe** es dejar el avión con la trayectoria actual hasta que ese otro piloto apague su emisora o cambie de canal, aun así hay que reaccionar muy pronto o perderemos el avión.

Figura 101: Emisora de 6 canales Futaba F-6 EXHP (FM/PCM)

Cuando decimos sin más que tenemos un equipo de radio FM en realidad tenemos un equipo **FM/PPM** y si decimos que tenemos un equipo PCM en realidad se trata de un equipo **FM/PCM**.

Otra característica a destacar es si un equipo de radio es **sintetizado** o no. Los equipos no sintetizados emplean cristales de cuarzo para funcionar en una determinada frecuencia. Los sintetizados no emplean cristal, cuando se enciende el receptor éste adopta la frecuencia de la emisora más cercana hasta que lo apaguemos y la frecuencia de la emisora se puede cambiar.

Hace tiempo se usaban las bandas de 35, 40 o 72 Mhz para la FM pero ahora mismo solamente es legal para aeromodelismo la banda de emisión de 35 Mhz que va desde los 35.030 a 35.200 Mhz. En esta banda tenemos disponibles 17 canales con una separación de 10 Khz.

Equipos de radio de 2,4 Ghz.

Esta última tecnología de radio evita todo tipo de interferencias. Con las emisoras de 2,4 Ghz nos olvidamos de asignar un canal a la emisora y al receptor. Evita el problema que teníamos con las emisoras FM si otro piloto ocupaba nuestro mismo canal. También se evitan las interferencias causadas por los canales adyacentes (otros pilotos pueden usar emisoras que emitan en un canal con suficiente desviación como para interferir en el nuestro).

Los equipos de radio de 2,4 Ghz utilizan tecnología **DSS** (distribución dinámica de espectro). Aunque Futaba ya llevaba tiempo haciendo pruebas con esta tecnología fué Spektrum quien se lanzó primero al mercado con DSS. DSS es una tecnología en la que no se transmite en una única frecuencia sino que emplea varias frecuencias de forma alternada.

Figura 102: Emisora Spektrum DX6i 2,4GHZ

Existen dos formas de llevar a cabo la tecnología DSS:

DSSS (Espectro Ensanchado por Secuencia Directa): Tanto DSSS como FHSS están definidos por la IEEE en el estándar 802.11 para redes de área local inalámbricas WLAN. En esta modalidad la señal se mezcla con el ruido empleando un algoritmo matemático. Solo el receptor que conoce dicho algoritmo será capaz de interpretar dicha señal. Para el resto de receptores solo será ruido. La frecuencia de emisión se va alternando pero en cada frecuencia se transmite por completo la información a transmitir.

FHSS (Espectro ensanchado por salto de frecuencia): La tecnología de espectro ensanchado por salto en frecuencia consiste en transmitir una parte de la información en una determinada frecuencia durante un intervalo de tiempo muy breve para continuar transmitiendo dicha información en otra frecuencia. De esta forma cada tramo de información se va transmitiendo en una frecuencia distinta durante un

intervalo muy corto de tiempo. El orden de salto en frecuencia ha de ser conocido tanto por la emisora como por el receptor.

Figura 103:Receptor Spektrum AR6250 2,4 GHZ

Vemos que con DSS las señales son casi indetectables por otro equipo de radio pues se confunden con el ruido. Para transmitir con esta tecnología se necesita un espectro amplio, por ello no se puede emplear en 35 Mhz ya que el ancho de banda disponible es pequeño. Para ello hay que desplazarse a frecuencias más altas que no están tan demandadas.

Se ha elegido la frecuencia de 2,4 Ghz porque es libre y gratuita. Sin embargo se trata de una frecuencia muy ineficiente ya que coincide con la frecuencia de resonancia del agua, por lo que la señal se atenúa enormemente en entornos con humedad. Aun así pese a que la señal se ve afectada por la humedad del ambiente no es tanto problema para pilotar un avión que se va a encontrar a 300 o 500 metros. El problema que existe es que al ser libre puede ser utilizada por otros aparatos electrónicos que no utilicen DSS e invadan gran parte de los canales con mucho ruido. Bastará pues con que haya varios pilotos volando con FPV (vista en primera persona) y alguna casa con un repetidor de video inalámbrico para abarcar todo el espectro dejando inoperativo nuestro equipo de radio.

8.3 Descripción detallada de la implementación

8.3.1 Controlador

En esta sección veremos con detalle las clases y funciones más importantes del controlador.

Clase ClasePrincipal:

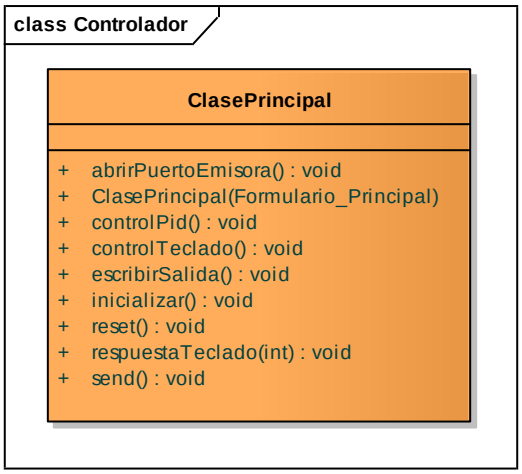


Figura 104: Controlador - clase principal

La clase principal es la clase sobre la que se sustenta la aplicación, conteniendo así todas las variables globales del programa, como los buffers de la emisora, los setpoints de los controladores PID, los valores medios de la emisora, los controladores PID, los valores de salida de la aplicación, y algunas variables auxiliares. Para ello, hace uso de algunas funciones, que serán descritas a continuación.

- Funciones importantes:
  - `public void controlPid()`

Ésta es la función encargada de usar los controladores PID para obtener la salida necesaria para controlar el UAV, ya sea a través del simulador o el UAV real. Para ello, llama a los PID de altura y de rotación, teniendo en cuenta la posición actual del UAV y el setpoint correspondientes. Posteriormente, y según sea el ángulo de rotación que tiene el UAV, usa los controladores PID de movimiento frontal y lateral para calcular la

salida de éstos, intercambiando sus parámetros de entrada según si el UAV tiene una rotación de 90º ó 270º o no.

- `public void respuestaTeclado(int tecla)`

Esta función es la encargada de cambiar el valor del buffer de la emisora, aumentando o disminuyendo en 1 respecto a su valor anterior, según si la tecla ordenaba aumentar o disminuir. También comprobará que el valor alcanzado no sobrepase los valores máximos y mínimos de la emisora, consiguiendo así un control igualitario entre teclado y emisora.

- `public void actualizarBuffer()`

Es la función encargada de actualizar el buffer de envío con los valores que han sido preparados para enviar. También se encarga de comprobar que los valores estén en el rango de los valores que puede dar la emisora.

- `private void copiaEmisora()`

Es la función encargada de escribir los valores de salida en el puerto COM dónde está conectada la emisora, para así enviárselos al UAV. También se encarga de recibir el eco que genera la emisora para comprobar que los valores que realmente envía la emisora son los que queremos que se envíen.

- `public void reset()`

Reinicia todas las variables y el estado de la aplicación a su estado inicial.

Clase Formulario_Principal:

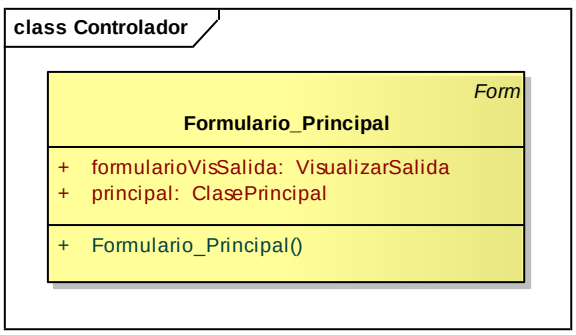


Figura 105: Controlador - Formulario principal

Esta clase contiene todas las funciones asociadas a los eventos y botones del formulario principal.

Funciones importantes:

- `private void Main_Loop()`

Esta función se ejecuta por cada vez del ciclo de reloj del timer del programa. Su función es la de diferenciar si estamos usando los controladores PID o el teclado, y llamar a las funciones necesarias en cada caso, 'controlPid()' en caso de que estemos controlando el UAV mediante controladores PID, o 'controlTeclado()' en caso de que queramos manejar el UAV mediante el teclado. Posteriormente actualizará las etiquetas de la interfaz con los valores nuevos, así como las barras de progreso que representan la salida del programa. En caso de que tengamos activado el formulario de salida, se actualizará con los nuevos valores de salida de la aplicación.

- `private void Camaras_Simu_Click(object sender, EventArgs e)`

Esta función es la encargada de alternar entre el modo simulador y el modo real, adaptando sus funciones (como los controles PID) al tipo de uso que le estemos dando. Esto es necesario ya que desde las cámaras que siguen al UAV real nos llega el error que tiene respecto al setpoint, mientras que desde el simulador lo que llega es la posición del UAV, debiéndola nosotros comparar con el setpoint definido en el cliente.

- `private void validar()`

Esta es la función encargada de validar los cambios efectuados en el control desde la pestaña 'Configuración', estos cambios afectan tanto a los parámetros de los controladores PID, como a los canales de la emisora.

Clase VisualizarSalida:

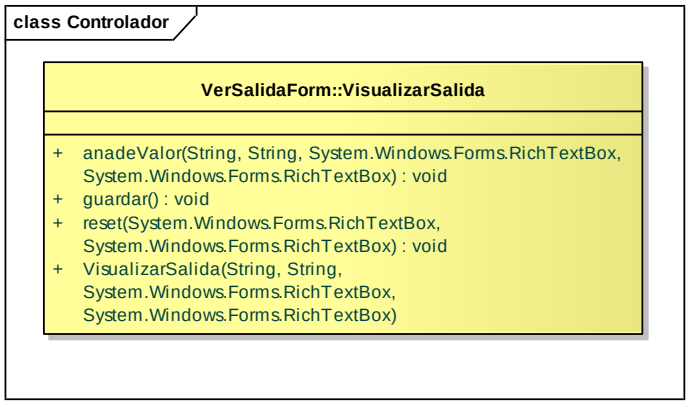


Figura 106: Controlador - Formulario de visualizar salida

Ésta es la clase encargada de mostrar la salida a través de dos cuadros de texto, en uno de ellos la salida de la aplicación, y en el otro el eco recibido de la emisora. También es capaz de guardar la salida en un fichero para analizarla posteriormente, pudiendo así tener presente en todo momento la evolución de la aplicación para evitar posibles errores.

- Funciones importantes:

  - ```
    public VisualizarSalida(String sE, String sR,
    System.Windows.Forms.RichTextBox
    textoSalida,System.Windows.Forms.RichTextBox
    textoSalidaRecibido)
    ```

    Su labor es mostrar por pantalla el valor de salida actual.

  - ```
    public void anadeValor(String sE, String sR,
    System.Windows.Forms.RichTextBox
    textoSalida,System.Windows.Forms.RichTextBox
    textoSalidaRecibido)
    ```

    Esta función concatena la salida actual a la anterior, separadas por un fin de línea, y lo muestra por pantalla.

  - ```
    public void guardar()
    ```

    Esta función guarda los valores de salida en un fichero '.txt'.
    ```

Clase ModuloConexionUDP:

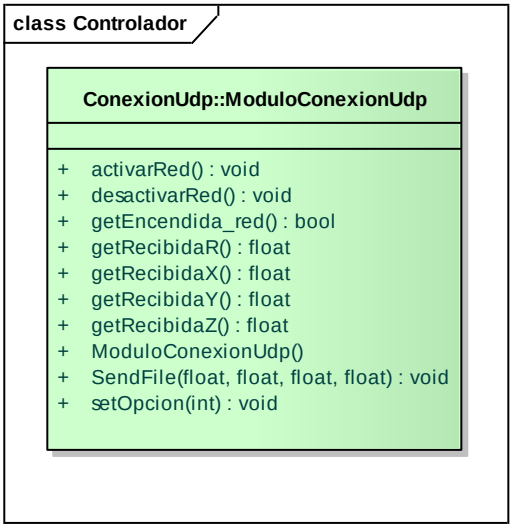


Figura 107: Controlador - Modulo de conexion UDP

Esta clase es la que se encarga de realizar las comunicaciones entre el controlador y la aplicación de captura de datos (simulador o cámaras). Implementa métodos de recepción y envió de datos, así como las funciones necesarias para establecer la conexión.

Funciones importantes:

- public void activarRed()

Se encarga de establecer la conexión a través mediante dos puertos UDP, uno de envío y otro de recepción

- public void SendFile(float x, float y, float z, float r)

Envía un array de bytes

- public float getRecibidaY()

Leemos la componente Y recibida de la otra aplicación

Clase ControladorPID_Camaras y COntroladorPID_Simulador:

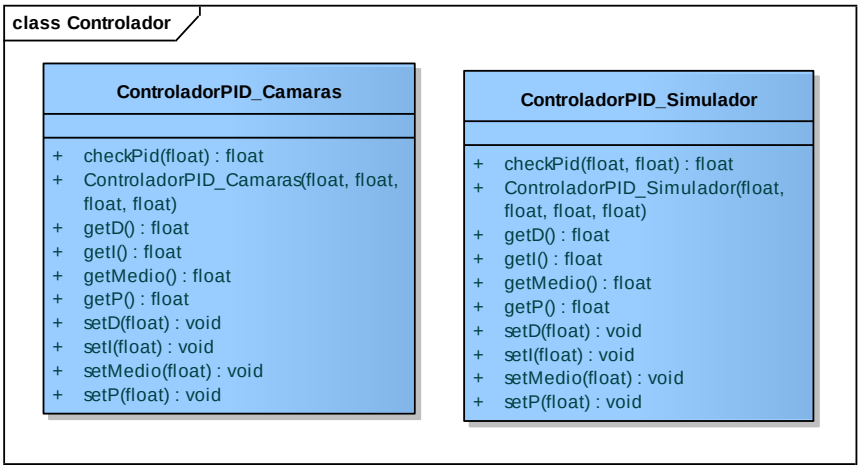


Figura 108: Controlador - Controladores PID

Estas clases son las encargadas de generar la señal de control a partir de los datos recibidos a través del modulo de conexión UDP, son dos clases especificas, una para cuando trabajamos con el simulador, y otra para el sistema de cámaras.

Esta distinción es necesaria puesto que cuando trabajamos con las cámaras reales solo recibimos el error del UAV respecto al SetPoint, mientras que trabajando con el simulador recibiremos la posición absoluta del UAV en el espacio de vuelo.

Funciones importantes

- public float checkPid(float error)

Se encarga de calcular las señales de control respecto al error.

Clases auxiliares:

Clase constantes

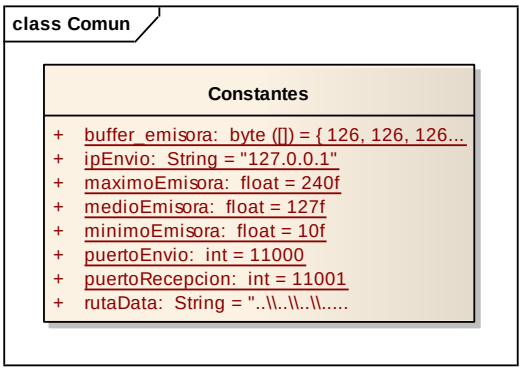


Figura 109: Controlador - clase que contiene las constantes del sistema

Sirve para declarar todas las constantes que utiliza el sistema durante su ejecución, permitiendo realizar cambios con mayor facilidad. Algunas de estas constantes serán, por ejemplo, los valores extremos de la emisora o el número de puerto de envió y recepción.

Clase XMLpersistencia

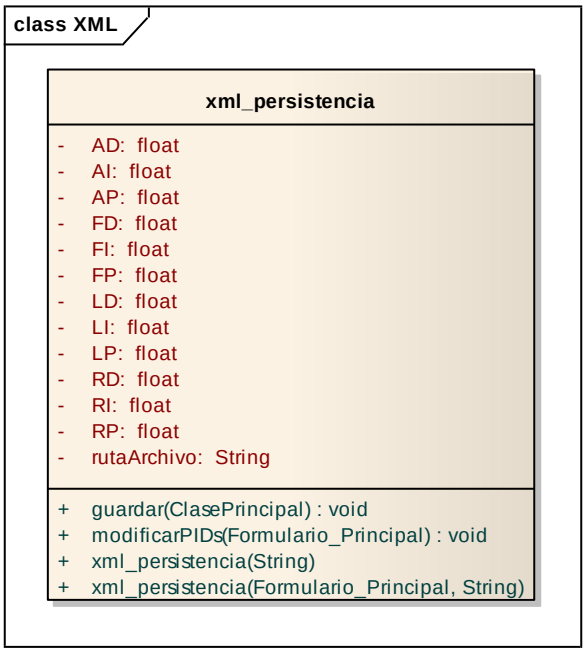


Figura 110: Controlador - Clase que genera el XML de datos guardados

Será la encargada de guardar las configuraciones del sistema que deseemos en un archivo XML con la estructura predeterminada que el sistema es capaz de leer.

8.3.2 Simulador

En esta sección veremos las clases y funciones más importantes del simulador.

Clase Form1:

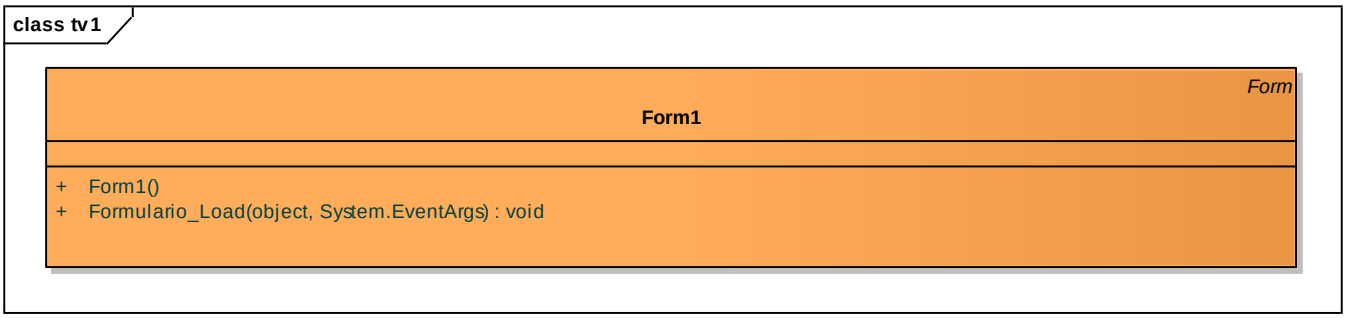


Figura 111: Simulador - Formulario principal

Se trata del formulario principal, desde donde recibimos y controlamos los eventos que hacen funcionar la aplicación. Lleva una referencia a todos los módulos que se usan, mandándoles las instrucciones adecuadas para su funcionamiento, y gestionando los errores que se produzcan en cualquiera de ellos.

A lo largo de estos meses de trabajo se ha intentado realizar una jerarquía de clases descentralizada, de manera que si se producía un error en un módulo o se quisiera cambiar no afectase al funcionamiento de los que no dependiesen de él. El papel que jugaría el Form1 dentro de esta jerarquía es el de controlador, un árbitro entre todo el resto de módulos que los coordina para un funcionamiento óptimo.

Dado que los métodos de control son privados, de cara al usuario solo vemos la constructora.

- Funciones importantes:
  - `private void Main_Loop()`

Ésta es la función principal de la aplicación. Es la que se encarga de actualizar a cada momento el estado y la posición del UAV, según en el estado que se encuentre la

aplicación. Para ello está constantemente actualizando la escena del motor gráfico, y la posición del UAV, utilizando un timer (timer1). También se encargará de llamar a la función CheckInput(), que será la encargada de leer.

- `private void CheckInput()`

Esta función es la encargada de la entrada al sistema del UAV. Para ello distingue tres casos, el caso en que el UAV está siendo controlado por teclado, el caso en que está siendo controlado a través de la emisora, y el caso que esté siendo controlado mediante el controlador por UDP. Para el teclado y la emisora, actualiza el bufer de entrada, y, para las tres, llama a las funciones de movimiento del UAV, que serán las encargadas de actualizar la posición del UAV.

También gestiona el movimiento de las cámaras, actualizando su posición en el espacio respecto a las teclas pulsadas, y su rotación respecto al movimiento del ratón.

- `private void timer_Muestreo_Tick(object sender, EventArgs e)`

Esta función es la encargada de actualizar la dinámica y la posición del UAV respecto a la entrada al sistema mediante la llamada a las cuatro funciones checkPlant() correspondientes a las cuatro dinámicas que controlan el UAV. Además, comprueba que la posición resultante del UAV no sobrepase los límites del escenario; en cuyo caso inicializa las variables internas de la dinámica mediante la función resetea() de las dinámicas, y mantiene su posición.

- `private void envio_recepcion()`

Es la función encargada de enviar los datos que representan la posición del UAV por UDP. También se encarga de mostrar en todo momento los datos enviados y los recibidos.

Clases Piloto, Planta y PlantaMovimiento:

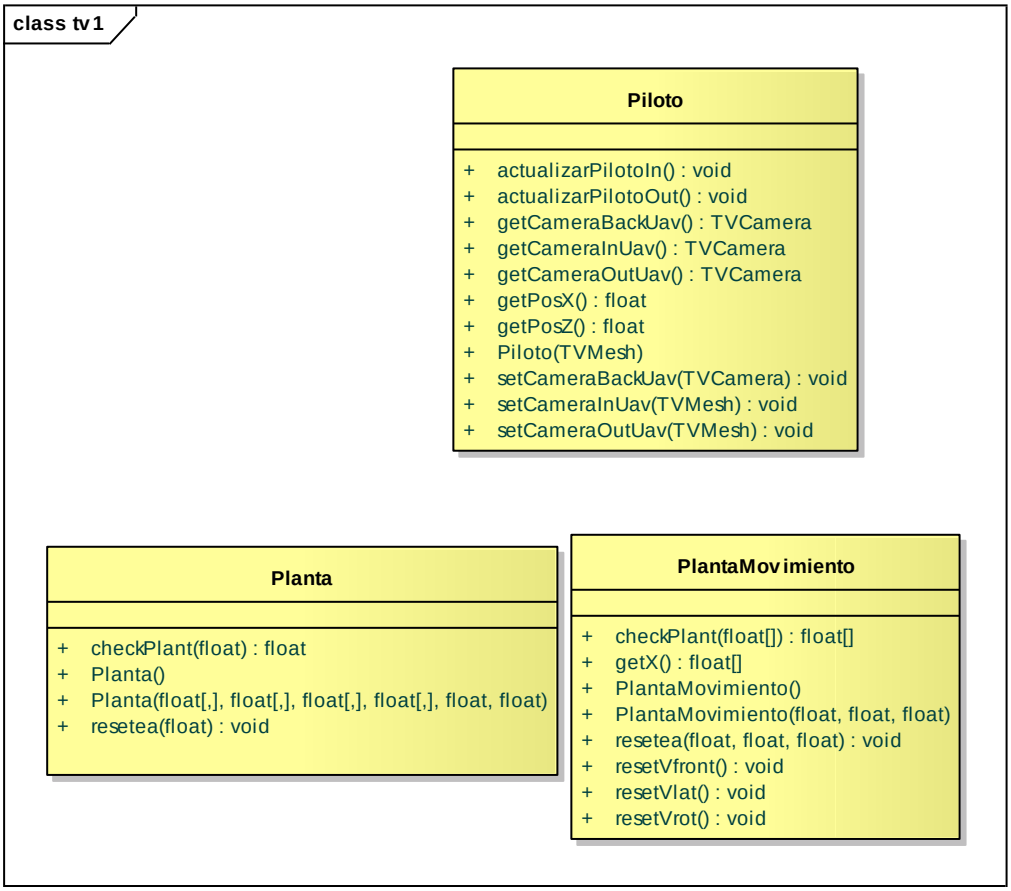


Figura 112: Simulador - Clases Piloto, planta y planta Movimiento

La clase **Piloto** es la encargada de gestionar y cambiar las cámaras del simulador, así como de actualizar la posición de éstas en el simulador.

La clase **Planta** es la clase que simula el movimiento en altura del UAV. Es una clase genérica que se basa en las matrices A, B, C y D para describir el comportamiento de un sistema. Para ello, carga de un XML los valores de dichas matrices, así como el intervalo de muestreo; previamente inicializado mediante el Script Matlab 'Planta_a_XML.m', creado por nosotros e incluido en el proyecto. Tiene algunas funciones importantes:

- `public float checkPlant(float u)`

Esta función es la encargada de actualizar la planta del sistema como resultado de la introducción de una señal de entrada 'u'. Para ello primero actualiza la salida, que depende del estado anterior, y, posteriormente, actualiza el vector X de estado en función de la entrada 'u'. El resultado es la devolución del nuevo valor de salida, en este caso, la posición del UAV en altura.

- `public void resetea(float estadoInicial)`

Esta función se encarga de reiniciar el estado de la planta a un estado inicial dado por el parámetro 'estadoInicial'. Para ello se calcula el valor del vector de estado X necesario para que la salida del sistema, es decir, la posición, sea la dada por estadoInicial.

La clase **PlantaMovimiento** simula los otros tres grados de libertad del UAV, es decir, el movimiento frontal, el movimiento lateral, y la rotación. Es necesario que estos tres grados de libertad estén contenidos en la misma clase, puesto que la salida de cada uno depende de los otros. Por ejemplo, la salida en el eje x al aplicarle un movimiento frontal dependerá de la rotación de UAV en ese momento, es decir, del estado del UAV, al igual que el movimiento lateral. Esto se consigue con un vector de estado de seis componentes, en el que las tres primeras se corresponden con la posición en x, la posición en y, y el ángulo de rotación; y las tres restantes se corresponden con la velocidad frontal, la velocidad lateral, y la velocidad de rotación actuales. Algunas funciones importantes son:

- `public float[] checkPlant(float[] mov_in)`

Esta función es la encargada de actualizar la planta del sistema como resultado de la introducción de una señal de entrada en forma de vector, 'mov_in'. La señal de entrada representa el incremento de la entrada a los ejes de movimiento frontal, movimiento lateral y rotación, respectivamente. En esta función se calcula la salida del sistema, la cual no depende de la entrada, si no solamente del estado anterior del sistema. Ésta salida se calcula con cálculos trigonométricos basados en senos y cosenos.

Posteriormente, se calcula la velocidad del próximo estado del sistema, en relación con el vector de entrada del sistema, teniendo en cuenta un coeficiente de rozamiento dad por la constante 'rozamiento'.

- `public void resetea(float posx, float posz, float rot)`

Esta función es la encargada de reiniciar el estado del sistema; para ello, asigna al vector de estado las posiciones indicadas por los parámetros de entrada, y pone también, en el vector de estado, las velocidades de cada grado de libertad a cero.

Módulo de tratamiento de imagen:

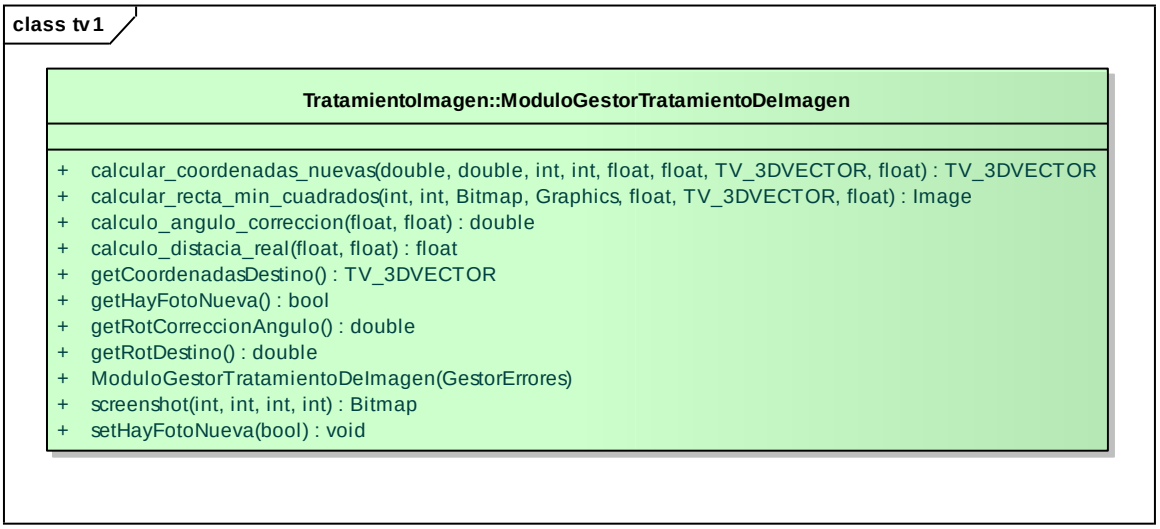


Figura 113: Simulador - Modulo de tratamiento de imagen

Gestiona todo el apartado asociado al tratamiento de imagen en el simulador, es decir, realiza la captura de pantalla desde el interior del UAV, analiza la imagen binarizándola y obteniendo los bordes, hace la aproximación por mínimos cuadrados de la binarización y por último calcula las coordenadas hacia donde se debe dirigir el UAV a continuación.

En caso de producirse un error, lo recoge e informa al controlador de cuál ha sido para que lo gestione como es debido.

Funciones importantes:

- `public Bitmap screenshot(int coorX, int coorY, int anchura, int altura)`

Función encargada de devolver la foto de lo que ve el UAV. Debido a que la función para sacar fotos que otorgaba el API de Truevision era muy lenta, decidimos implementar nosotros mismos una más veloz. Se limita a sacar un a foto de todo lo que se esté ejecutando en la pantalla y recortarla de modo adecuado para coger únicamente lo que nos interesa.

- `public Image calcular_recta_min_cuadrados(int width, int height, Bitmap screen, Graphics graficos_tratam_img, float alturaMesh, TV_3DVECTOR coordenadasMesh, float rotacionInicial)`

Se encarga de calcular la recta con la que se obtendrán las coordenadas de destino gracias a la aproximación de mínimos cuadrados que saca de la imagen "screen". Se presupone que la imagen que se le pasa es la de un circuito verídico, porque en otro caso la aproximación se realizará de manera incorrecta.

- `public TV_3DVECTOR calcular_coordenadas_nuevas(double a, double b, int height, int width,float alturaMesh,float cteMultiplicativa,TV_3DVECTOR coorIniciales,float rotInicial)`

Calcula las coordenadas objetivo del UAV a partir de la recta de la aproximación. Además se estima la distancia real y se calcula el ángulo real de giro.

Como el módulo de tratamiento posee dependencias de varias clases inferiores en jerarquía, lo definiremos en detalle más adelante.

Módulo de grabación:

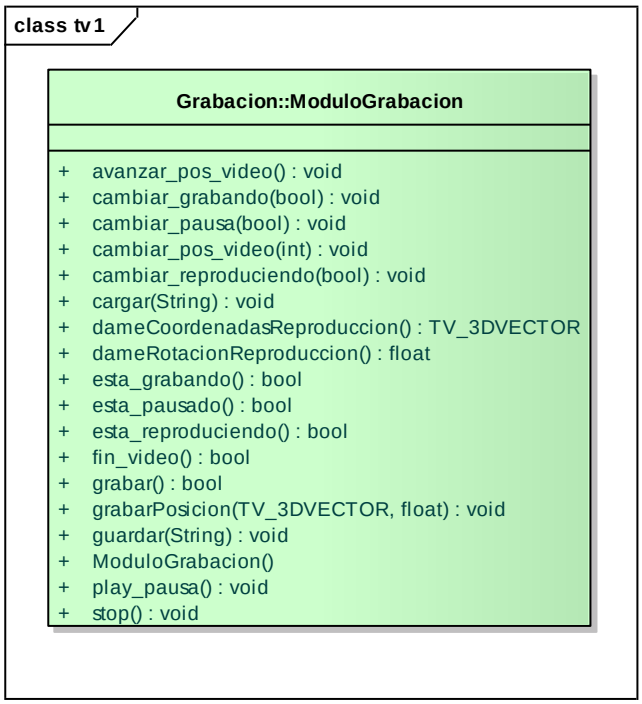


Figura 114: Simulador - Modulo de grabación de la trayectoria

Se trata de un módulo que graba los movimientos del UAV cuando se lo indicamos para ver fallos incorrecciones en las pruebas realizadas en el simulador.

Permite asimismo guardar las grabaciones en un fichero, cargarlas y reproducirlas cuando queramos, (pudiendo pausar, para y reanudar la reproducción cuando queramos). Cabe destacar, no obstante, que no se carga el entorno donde se grabaron las pruebas: únicamente se reproduce el movimiento con el vehículo disponible en el mapa.

- Funciones importantes:

  - ```
    public void grabarPosicion(TV_3DVECTOR posicionAGuardar,
    float rotacion)
    ```

Graba la posición actual si es distinta a la anterior. De esta manera no grabamos cuando el UAV se encuentra parado
    ```

- funciones guardarTrayectoria y cargar:

Según elijamos guarda en el disco la última trayectoria grabada o carga en el simulador la que queramos para poder observarla

El Gestor de errores:

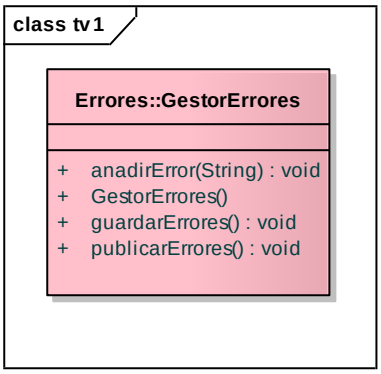


Figura 115: Simulador - Gestor de errores

Su tarea es simple: guardar los errores que le notifican, sea el módulo que sea. Dichos errores se podrán volcar a disco o mostrar por pantalla para facilitar las trazas y corregir fallos en la aplicación. En general todos los módulos comparten la misma instancia del gestor de errores, luego sólo encontraremos uno por ejecución.

Módulo de conexión por UDP:

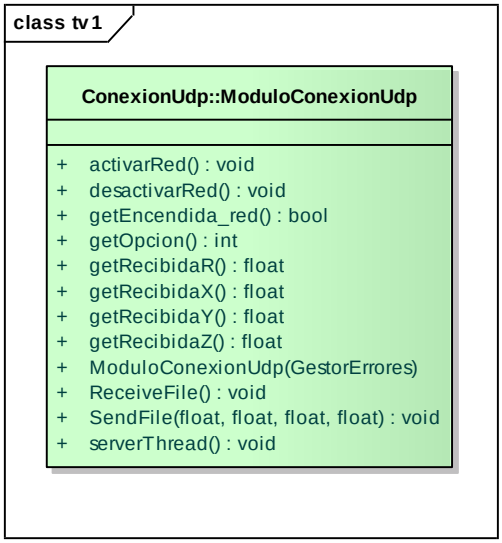


Figura 116: Simulador - Modulo de conexión UDP

Mediante este módulo se realizan las conexiones a través de la red con el controlador cuando el simulador juega el papel de sistema real para las pruebas con el controlador. Utilizando la conexión UDP el simulador enviara al controlador la posición del UAV en cada momento, el controlador generara las señales de control teniendo en cuenta la posición recibida y el setPoint definido, que reenviara mediante un modulo similar a este y recibiremos en el simulador para mover el vehiculo.

- Funciones importantes:
  - `public void activarRed()`

Establece una conexión de red de doble sentido con el módulo del controlador a través de dos puertos UDP (11000 para envío y 11001 para recepción) predefinidos.

La función desactivarRed`()` hará lo contrario.

  - `public void SendFile(float, flota, flota, float)`

Envia a traves del puerto de salida 4 valores en coma flotante que representan la posición en el espacio de vuelo del UAV, estos son enviados en el orden siguiente que el controlador reconocerá: posición X – posición Y – posición Z – valor del Yaw, transformadas previamente en un array de Bytes

  - `public void ReceiveFile()`

Esta función leerá el contenido en ese instante del puerto de recepción, en el cual encontrara un array de bytes, del cual obtendrá las 4 primeras posiciones, correspondientes a las 4 señales de control necesaria, y los asignara a las variables globales del módulo (recibidasX, recibidaY, recibidaZ y recibidaR) para que sean accesibles desde el simulador mediante sus 4 accesoras

Localización: el Gestor del Minimapa

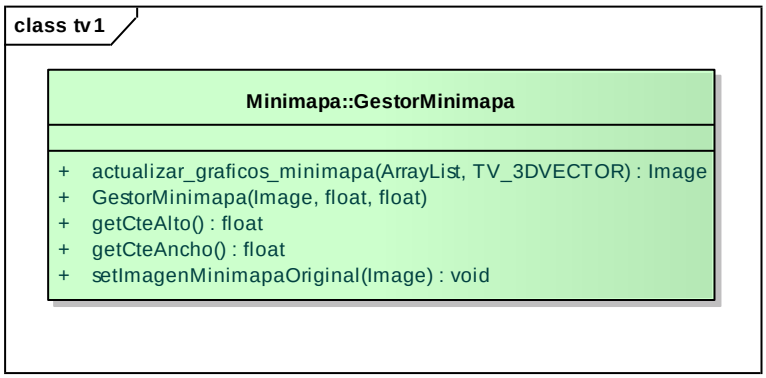


Figura 117: Simulador - Gestor del mini mapa y sistema de localización

Debido a que en el entorno 3D es fácil perderse con la cámara, hemos desarrollado un pequeño mapa que se despliega abajo a la izquierda en la aplicación y que se encarga de ofrecer una vista en miniatura de los objetos que podemos encontrar en el escenario. Así sabemos dónde está cada cosa, y nos podremos situar rápidamente sobre ella.

Este gestor es el que se encarga de que funcione correctamente mediante la función actualizar_gráficos_minimapa, que en cada ciclo chequea cada pequeño cambio y lo dibuja para que estemos informados

Formularios auxiliares:

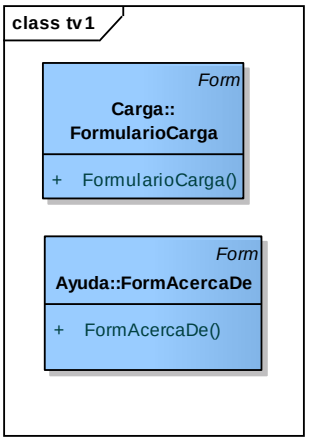


Figura 118: Simulador - Formularios de carga y de ayuda

No todo se puede mostrar en el formulario principal porque quedaría confuso y engorroso a la hora de utilizarlo. Por esta razón nos hemos ayudado de dos pequeños formularios: Uno se utiliza mientras se carga el programa, (aunque quizá en la versión final no se incluya por eficiencia), mientras que el otro muestra por pantalla información acerca de la versión y los desarrolladores del programa.

Dependencias:

De las clases MovimientoEspiral, para saber a dónde nos debemos dirigir si estamos perdidos, y CalculoMinimosCuadrados, para calcular la recta correspondiente.

La dependencia de GestorErrores sirve para controlar los posibles errores y guardarlos.

8.3.3 Módulo de tratamiento de imagen

En esta sección ampliaremos el detalle sobre la implementación del módulo de tratamiento de imagen. En la siguiente figura podemos ver la arquitectura del mismo:

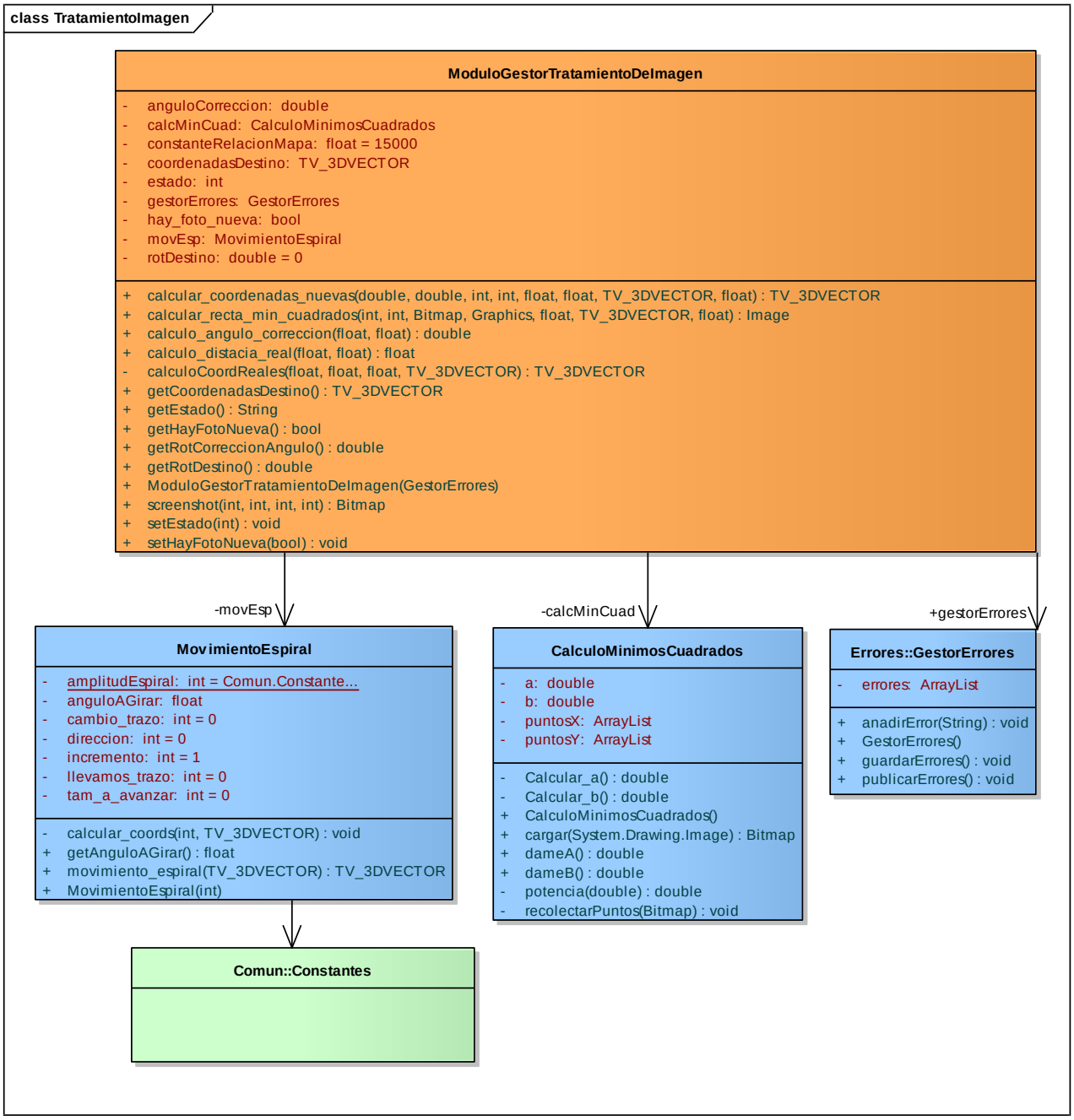


Figura 119: Grafo de dependencias del modulo de tratamiento de imagen

Explicación detallada de las clases:

MovimientoEspiral:

Clase que permite obtener las coordenadas hacia donde nos debemos dirigir en el instante actual. Ofrece 2 operaciones, dependiendo de cómo prefiramos realizar el movimiento

getAnguloAGirar(), que nos devuelve el ángulo en grados que debe girar el vehículo en el instante actual.

Movimiento_espiral(TV_SDVector), que dado un vector de coordenadas inicial, calcula las coordenadas hacia donde se debe dirigir el vehículo.

CalculoMinimosCuadrados:
Clase que se encarga de realizar la aproximación por mínimos cuadrados de la imagen tomada por el cuatrimotor en el simulador. Para ello utiliza las siguientes funciones:

RecolectarPuntos(Bitmap), que dada una imagen binarizada calcula los puntos que vamos a necesitar para el cálculo de la recta.

calcular_a() y calcular_b(), encargados de obtener los parámetros a y b de la recta de aproximación.

GestorErrores:
Si se produce algún error en todo el proceso, el gestor se encarga de archivarlo para que posteriormente el módulo encargado los muestre por pantalla o los guarde en un archivo.

ModuloGestorTratamientoDeImagen:

Clase principal del módulo. Se encarga de calcular las siguientes coordenadas dependiendo del estado en que se encuentre el vehículo, bien sea moverse en espiral o dirigirse al punto que indique la recta resultante de aproximar por mínimos cuadrados la imagen tomada.

Tiene accesoras y mutadoras para saber cuáles son las coordenadas nuevas, así como el estado en que se encuentra el vehículo.

Constantes:
Clase donde están establecidas todas las constantes que usan los módulos del proyecto. En este caso, se usa para saber cuál es la amplitud que queremos que tenga la espiral.

8.3.4 Generador de escenarios

En esta sección explicaremos con detalle cómo se ha implementado el generador de escenarios.

La arquitectura que sigue la herramienta se puede ver en la siguiente figura:

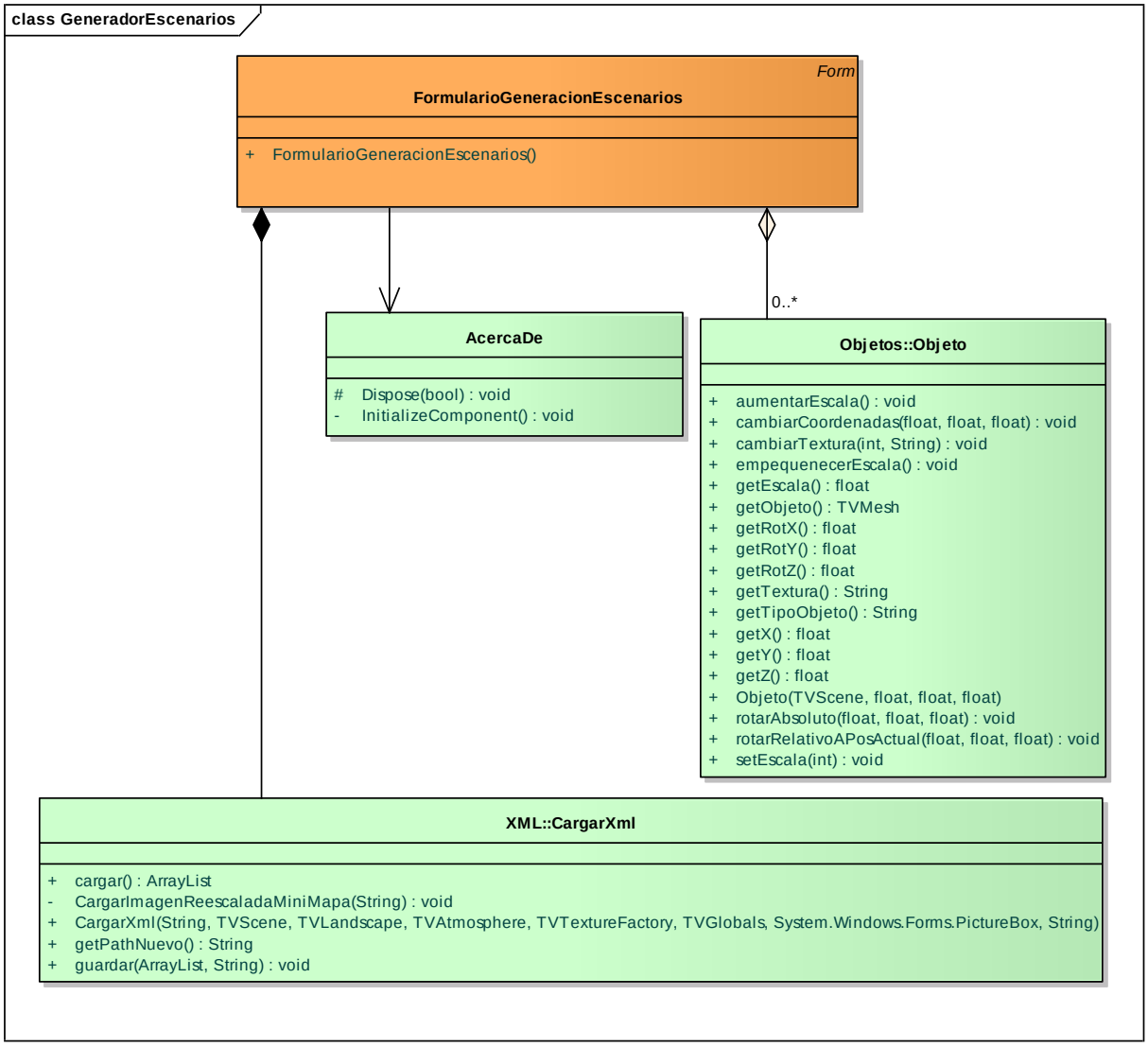


Figura 120: Grafo de dependencias de clase del generador de escenarios

La clase principal es FormularioGeneracionEscenarios, que se encarga de visualizar la herramienta de generación propiamente dicha. Presenta el Formulario principal con todas las opciones disponibles, y se encarga de gestionar todos los posibles errores que ocurran durante la generación, avisando adecuadamente al usuario.

La clase CargarXML carga un escenario mediante el método CargarXML. En caso de que ocurriesen errores, informa al FormularioGeneracionEscenarios para que los trate.

El formulario AcercaDe simplemente muestra datos acerca de los creadores de la aplicación.

Por último, los objetos son los distintos obstáculos y vehículos que encontraremos en nuestro escenario. Como casi todos tienen las mismas características, hemos optado por realizar la siguiente jerarquía:

Jerarquía de la clase Objeto:

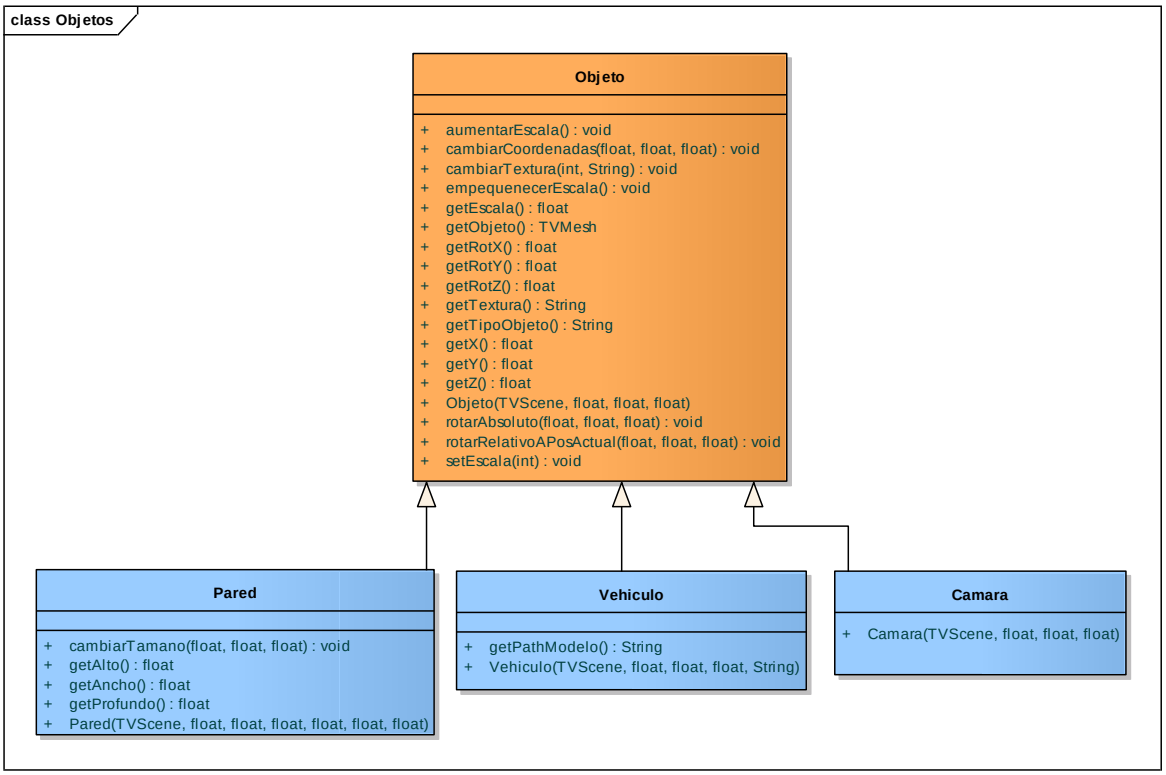


Figura 121: Grafo de dependencias de las clases que heredan de Objeto

La clase Objeto es la principal. Posee las características principales del objeto, como su escala, rotación de los ejes, coordenadas en el escenario, etc., así como sus mutadoras y accesoras.

De ella heredan las clases Pared, Vehículo y Cámara.

- Pared: Representa un obstáculo en el escenario. Por lo tanto tiene sus atributos de ancho, alto y profundo, aparte de mutadoras y accesoras para poder cambiarlos como queramos desde fuera.

- Vehículo: Representa a un vehículo en el escenario. Como tiene un modelo asociado, guarda su ruta para poder ir a la carpeta adecuada a cargarlo cuando se inicia el escenario.

- Cámara: inicialmente se ideó este objeto para que representase una cámara que se pudiera dejar en donde quisiéramos en el escenario, pero al final se optó por no usarla. No se ha borrado por si en futuras modificaciones resulta necesario.

9 GLOSARIO

- **Ángulo de orientación: á**ngulo que esta girado el helicóptero sobre el eje Y del sistema de coordenadas fijo.
- **Punto de consigna:** se trata del punto a donde queremos que se dirija el UAV.
- **Cuatrimotor:** el vehículo aéreo de 4 hélices que vamos a intentar controlar de manera autónoma.
- **Dinámica:** La dinámica es la parte de la física que describe la evolución en el tiempo de un sistema físico en relación a las causas que provocan los cambios de estado físico y/o estado de movimiento. El objetivo de la dinámica es describir los factores capaces de producir alteraciones de un sistema físico, cuantificarlos y plantear ecuaciones de movimiento para dicho sistema.
- **Eco:** Al escribir en la emisora, se pueden leer los valores escritos, es lo que llamamos eco.
- **Emisora:** instrumento que sirve para enviar al cuatrimotor las señales de control a distancia. Es lo que comúnmente se conoce como control remoto.
- **Modelo:** conjunto de las dinámicas que actúan sobre un sistema.
- **Pitch (Cabeceo):** inclinación del morro del avión, o rotación respecto al eje ala-ala
- **Posición espacial:** se refiere a la posición del UAV en un espacio de coordenadas fijo.
- **Radiofrecuencia:** cada una de las frecuencias de las ondas electromagnéticas empleadas en la radiocomunicación.
- **Roll (Alabeo):** rotación intrínseca alrededor del eje longitudinal del avión.
- **Setpoint:** se trata del punto a donde queremos que se dirija el UAV.

- **PIC:** Los '**PIC**' son una familia de microcontroladores tipo RISC. El PIC usa un juego de instrucciones tipo RISC, cuyo número puede variar desde 35 para PICs de gama baja a 70 para los de gama alta. Las instrucciones se clasifican entre las que realizan operaciones entre el acumulador y una constante, entre el acumulador y una posición de memoria, instrucciones de condicionamiento y de salto/retorno, implementación de interrupciones y una para pasar a modo de bajo consumo llamada sleep.

- **Puerto RS-232:** Es una interfaz que designa una norma para el intercambio serie de datos binarios entre un DTE (Equipo terminal de datos) y un DCE (Data Communication Equipment, Equipo de Comunicación de datos), aunque existen otras en las que también se utiliza la interfaz RS-232.El RS-232 consiste en un conector tipo DB-25 (de 25 pines), aunque es normal encontrar la versión de 9 pines (DE-9), más barato e incluso más extendido para cierto tipo de periféricos.

- **Tiempo real:** Un Sistema en Tiempo Real (STR) es aquel sistema digital que interactúa activamente con un entorno con dinámica conocida en relación con sus entradas, salidas y restricciones temporales, para darle un correcto funcionamiento de acuerdo con los conceptos de predictibilidad, estabilidad, controlabilidad y alcanzabilidad. Un STR tiene tres condiciones básicas: interactúa con el mundo real (proceso físico), emite respuestas correctas y cumple restricciones temporales.

- **Trim:** hablamos de "estabilizar" el helicóptero cuando ajustamos los valores medios de la emisora para que éste se quede estable en el aire.

- **UAV:** Unmanned Aerial Vehicle (vehículo aéreo no tripulado), que en este proyecto será nuestro cuatrimotor o helicóptero.

- **UDP:** Protocolo de red orientado a conexión y sin confirmación

- **Yaw (Guiñada):** movimiento del avión respecto del eje imaginario vertical que pasa por el centro de gravedad de la aeronave. Este eje es perpendicular al eje de cabeceo y al de balanceo, está contenido en un plano que pasa por el morro y la cola del aparato y que normalmente divide a este en dos partes simétricas.

10 BIBLIOGRÁFIA

Método mínimos cuadrados:

www.wikipedia.org

http://www.sc.ehu.es/sbweb/fisica/cursoJava/numerico/regresion1/regresion1.htm

Motor gráfico y ejemplos:

http://www.truevision3d.com/

http://wiki.truevision3d.com/ (ejemplos)

Ajustes en el cuatrimotor:

http://www.radiocontrol.com.es/helicopteros_rc/helicopteros_electricos/ajustes_vuelo_helicoptero.html

ESA España:

http://esamultimedia.esa.int/esaCP/SEM5VKNFGLE_Spain_0.html

Introducción acerca de los sistemas de control:

http://www.isa.uma.es/C11/Ingenier%C3%ADa%20de%20Sistemas/Document%20Library/CONTROL_DE_SISTEMAS.pdf

http://www.monografias.com/trabajos6/sicox/sicox.shtml

Proyectos de aviones no tripulados israelíes:

http://www.israeli-weapons.com/weapons/aircraft/UAV/heron/Heron.html

Conceptos fundamentales de la lógica difusa:

http://www.tesisenxarxa.net/TESIS_UPC/AVAILABLE/TDX-0207105-105056//04Rpp04de11.pdf