UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA HARDWARE-IN-THE LOOP SOBRE PLATAFORMA RASPBERRY PI PARA LA SIMULACIÓN DEL VEHÍCULO AÉREO KADETT 2400 DE GRAUPNER

AUTOR: IVÁN LUQUE NAVARRO

TUTOR: JUAN MANUEL HERRERO DURÁ

COTUTOR: SERGIO GARCÍA-NIETO RODRÍGUEZ

Curso Académico: 2016-17

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA HARDWARE-IN-THE LOOP SOBRE PLATAFORMA RASPBERRY PI PARA LA SIMULACIÓN DEL VEHÍCULO AÉREO KADETT 2400 DE GRAUPNER

Documento 1: Memoria

AUTOR: IVÁN LUQUE NAVARRO

TUTOR: JUAN MANUEL HERRERO DURÁ

COTUTOR: SERGIO GARCÍA-NIETO RODRÍGUEZ

Curso Académico: 2016-17

Resumen

El presente proyecto consiste en el desarrollo de un sistema Hardware-in-the-Loop para la simulación en tiempo real del vehículo aéreo Kadett 2400 de Graupner realizado sobre plataforma de bajo coste Raspberry Pi. El desarrollo de la aplicación se ha realizado con lenguaje de programación científica Python.

Se incluye la simulación de las ecuaciones dinámicas y cinemáticas de la aeronave con el objetivo de determinar la posición y orientación del avión en el espacio. Los datos obtenidos de la simulación son enviados vía protocolo UDP al software de visualización FlightGear que se encarga de representar gráficamente el comportamiento de la aeronave.

Además, las superficies de control de la aeronave pueden ser fijadas de forma manual mediante un joystick o un sistema de control automático.

Índice

1. OBJETO DEL TFG

El objeto del presente TFG es el desarrollo de un sistema Hardware-In-the-Loop (HIL) sobre plataforma Raspberry Pi mediante el lenguaje de programación científica Python para la simulación del vehículo aéreo Kadett 2400 de Graupner.

La plataforma computará las ecuaciones dinámicas y cinemática básicas de la aeronave, determinando la orientación y posición en el espacio. Dicha información será enviada vía protocolo UDP al software de visualización FlightGear, el cual representará gráficamente el comportamiento de la aeronave. Asimismo, las superficies de control de la aeronave (empuje, deflectores de cola, profundidad y alabeo) podrán ser fijadas de manera manual (mediante un joystick similar al de una aeronave estándar) o mediante un algoritmo de control automático.

El grupo CPOH ai2 de la UPV está investigando en estrategias de control para el Kadett para pilotaje autónomo. El simulador HIL que se desea implementar fomentará el desarrollo de los sistemas de control automático para la aeronave.

El cliente impone especificaciones técnicas a las que el proyecto debe atenerse.

- La utilización de Raspberry Pi para la implementación del sistema HIL.
- El lenguaje de programación utilizado debe ser Python 2.x o Python 3.x
- Uso del software Flight Gear para la visualización del comportamiento del vehículo.

2. INTRODUCCIÓN

2.1 Antecedentes

La simulación de modelos físicos reales, utilizados para la predicción y control del comportamiento del sistema, es una práctica habitual en algunos ámbitos industriales como en la industria automotriz o en la aeroespacial. Sin embargo, las plataformas que se utilizan normalmente para esta finalidad tienen un elevado coste y además suelen tener una portabilidad muy limitada. Por tanto, no siempre será rentable realizar un sistema Hardware-in-the-loop, y será necesario un estudio de viabilidad previo.

2.2 Motivación

Se pretende desarrollar un dispositivo de bajo coste que tenga como objetivo principal la simulación del comportamiento del vehículo aéreo Kadett 2400 en tiempo real. El software de simulación desarrollado en Python irá implementado en una Raspberry Pi, sistema empotrado que se encargará también de transferir las variables resultantes que consideremos de importancia, éstas son las variables necesarias para la visualización del avión. A esto se le denomina simulador HIL, y permite el análisis y control del sistema real al ser conectado a éste.

El análisis de la simulación en tiempo real de sistemas complejos nos permite obtener una gran información sobre éste y capacita el correcto diseño de los modelos de control.

El bajo coste de la Raspberry Pi, así como la utilización del lenguaje de programación Python de licencia gratuita y con multitud de funciones implementadas en librerías, dan la posibilidad de obtener un sistema de simulación HIL económico y con una alta portabilidad.

3. DESCRIPCIÓN DE LA SOLUCIÓN ADOPTADA

3.1. Hardware in the Loop

La simulación “Hardware in the Loop”, en adelante simulación HIL, consiste en una técnica utilizada en el desarrollo en tiempo real de sistemas empotrados complejos. Un sistema empotrado, o sistema embebido, es un sistema de computación diseñado para cubrir unas necesidades específicas, a diferencia de los ordenadores personales que están diseñados para cumplir multitud de necesidades. Se trata de una plataforma efectiva y funcional, pues incluye toda la complejidad del sistema físico controlado por el sistema embebido. Se realiza un modelo matemático que engloba todos los aspectos y características que afectan a la dinámica del elemento de estudio con el que el sistema embebido interactúa.

La simulación HIL debe tener en cuenta además del modelo físico, la simulación de los sensores y actuadores que intervienen en el éste, ya que éstos sirven como la unión entre el modelo y el sistema embebido. Los valores de los sensores se ven afectados por el modelo de estudio, y son leídos por el sistema empotrado. Una vez simulada la respuesta del sistema controlado, un algoritmo de control manda las señales necesarias a los actuadores. Este proceso constituye por tanto un bucle de control cerrado.

Esta tecnología es adecuada en los casos donde un fallo en planta puede ser crítico, o bien el proyecto no es económicamente viable. Los factores que hacen adecuada la utilización de la simulación HIL son:

-Ciclos cortos de desarrollo:

Las limitaciones de tiempo de diseño y producción que requieren algunas empresas para ser competitivas pueden hacer inviable la espera a la disponibilidad de un primer prototipo para el testeo del sistema empotrado. Se hace necesaria por tanto la realización de una simulación HIL de forma paralela al desarrollo de la planta. Este caso se da principalmente en la industria aeroespacial y automotriz.

-Modelo de planta fiable y económico:

En proyectos costosos, la simulación HIL juega un gran papel ya que permite realizar las pruebas necesarias para el desarrollo del proyecto con una reducción de costes importante.

- Implementación de la interacción usuario-sistema de control:

La simulación HIL permite analizar la relación del usuario con el sistema de control. Puede ser capaz de simular en tiempo real el comportamiento del usuario en el control de planta. Es una gran alternativa en la prueba de procesos que puedan resultar peligrosos para el usuario. Además, permite una reducción de costes, ya que el usuario sería reemplazado por el sistema embebido.

En este caso, se realizará una simulación HIL de una aeronave sobre la cual se pueda ensayar y sirva de base para el diseño de los sistemas de control de una forma más económica y segura, que con el uso del avión real.

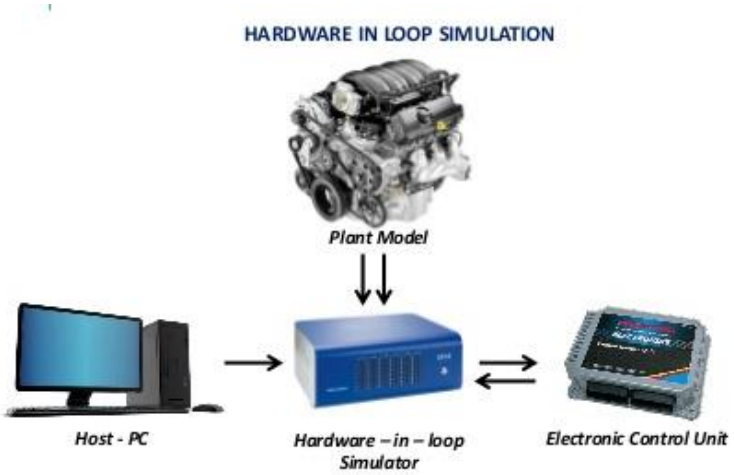


Figura 1. Simulación HIL. (https://www.slideshare.net/GowthamSubramanian6/hardware-in-loop-simulation)

3.2. Modelo

En aviación se emplean diferentes sistemas de coordenadas para definir la posición y velocidades del avión. Tendremos dos sistemas a los que referir el avión: (Rivas, 2015)

- Los ejes de tierra que, asumiendo una hipótesis de Tierra plana se puede suponer un sistema inercial. La hipótesis de Tierra plana se puede asumir para alturas de vuelo pequeñas en comparación con el radio terrestre y a velocidades de vuelo pequeñas comparadas con la velocidad orbital. El origen de coordenadas se toma en un punto cualquiera de la superficie terrestre. Los ejes X e Y forman el plano horizontal, normalmente X apunta al norte e Y hacia el Este. El eje Z es perpendicular al plano formado y se toma positivo hacia el centro de la Tierra.
- Los ejes locales del avión, que son solidarios a éste y por tanto se mueven a la misma velocidad. El origen de coordenadas se toma en el centro de gravedad de la aeronave. El eje X se sitúa en el eje de simetría del vehículo aéreo siendo positivo hacia adelante, el eje Z forma junto al eje X el plano de simetría y es positivo hacia abajo en la posición normal del avión. El eje Y completa el triedro.

Las ecuaciones del modelo dinámico de la aeronave se pueden encontrar en el documento de (Velasco – Carrau, 2016) referenciado en la bibliografía.

Todas las variables del modelo, así como los parámetros se tomarán en S.I.

El modelo dinámico del avión es el siguiente:

$$\dot{u} = rv - qw + \frac{\bar{q}S}{m} Cx\left(\delta_{[e,a,r]}\right) - g\sin\theta + \frac{T}{m} \quad (1)$$

$$\dot{v} = pw - ru + \frac{\bar{q}S}{m} Cy\left(\delta_{[e,a,r]}\right) + g\cos\theta\sin\phi \quad (2)$$

$$\dot{w} = qu - pv + \frac{\bar{q}S}{m} Cz\left(\delta_{[e,a,r]}\right) + g\cos\theta\cos\phi \quad (3)$$

$$\dot{p} - \frac{I_{xz}}{I_x}\dot{r} = \frac{\bar{q}Sb}{I_x} C_l\left(\delta_{[e,a,r]}\right) + \frac{I_z - I_y}{I_x} qr + \frac{I_{xz}}{I_x} qp \quad (4)$$

$$\dot{q} = \frac{\bar{q}S\bar{c}}{I_y} C_m\left(\delta_{[e,a,r]}\right) - \frac{I_x - I_z}{I_y} pr - \frac{I_{xz}}{I_y}(p^2 - r^2) - I_p\Omega_p r \quad (5)$$

$$\dot{r} - \frac{I_{xz}}{I_z}\dot{p} = \frac{\bar{q}Sb}{I_z} C_n\left(\delta_{[e,a,r]}\right) - \frac{I_y - I_x}{I_z} pq + \frac{I_{xz}}{I_z} qr - I_p\Omega_p p \quad (6)$$

Siendo las *u, v* y *w* las velocidades lineales en X, Y y Z en los ejes locales del avión respectivamente, tomadas en S.I. y por tanto en m/s, y sus derivadas por tanto las aceleraciones lineales, en m/s^2. Las variables p, q y r son las velocidades angulares en los ejes X, Y y Z respectivamente, medidas en rad/s, y de igual forma sus derivadas son las aceleraciones angulares en los susodichos ejes locales, en rad/s^2.

Las ecuaciones cinemáticas del vehículo aéreo:

$$\dot{\phi} = p + \tan\theta\,(q\sin\phi + r\cos\phi) \quad (7)$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \quad (8)$$

$$\dot{\psi} = \frac{q\sin\phi + r\cos\phi}{\cos\theta} \quad (9)$$

Donde $\phi, \theta\ y\ \psi$ son los ángulos de Euler de roll, pitch y yaw respectivamente. Por consiguiente, sus derivadas son las velocidades angulares de dichos ángulos. Los ángulos de Euler se tomarán en rad.

Los parámetros C_i son coeficientes aerodinámicos que dependen de la posición relativa de la aeronave respecto al viento, la velocidad del viento y el estado en las superficies de control del avión. Pueden obtenerse con las siguientes ecuaciones:

Ecuaciones del modelo aerodinámico longitudinal:

$$C_D = C_{D_0} + C_{D_{Vair}}\frac{1}{V_0}\Delta V_{air} + C_{D_\alpha}\Delta\alpha + C_{D_{\alpha^2}}\Delta\alpha^2 + C_{D_q}\frac{\bar{c}}{2V_0}q + C_{D_{\delta_e}}\Delta\delta_e \quad (10)$$

$$C_L = C_{L_0} + C_{L_{Vair}}\frac{1}{V_0}\Delta V_{air} + C_{L_\alpha}\Delta\alpha + C_{D_{\alpha^2}}\Delta\alpha^2 + C_{L_{\dot{\alpha}}}\frac{\bar{c}}{2V_0}\dot{\alpha} + C_{L_q}\frac{\bar{c}}{2V_0}q + C_{D_{\delta_e}}\Delta\delta_e \quad (11)$$

$$C_m = C_{m_0} + C_{m_{Vair}}\frac{1}{V_0}\Delta V_{air} + C_{m_\alpha}\Delta\alpha + C_{m_{\alpha^2}}\Delta\alpha^2 + C_{m_{\dot{\alpha}}}\frac{\bar{c}}{2V_0}\dot{\alpha} + C_{m_q}\frac{\bar{c}}{2V_0}q + C_{m_{\delta_e}}\Delta\delta_e \quad (12)$$

Ecuaciones del modelo aerodinámico lateral:

$$C_Y = C_{Y_0} + C_{Y_\beta}\Delta\beta + C_{Y_p}\frac{b}{2V_0}p + C_{Y_r}\frac{b}{2V_0}r + C_{m_{\delta_a}}\Delta\delta_a + C_{m_{\delta_r}}\Delta\delta_r \quad (13)$$

$$C_l = C_{l_0} + C_{l_\beta}\Delta\beta + C_{l_p}\frac{b}{2V_0}p + C_{l_r}\frac{b}{2V_0}r + C_{l_{\delta_a}}\Delta\delta_a + C_{l_{\delta_r}}\Delta\delta_r \quad (14)$$

$$C_n = C_{n_0} + C_{n_\beta}\Delta\beta + C_{n_p}\frac{b}{2V_0}p + C_{n_r}\frac{b}{2V_0}r + C_{n_{\delta_a}}\Delta\delta_a + C_{n_{\delta_r}}\Delta\delta_r \quad (15)$$

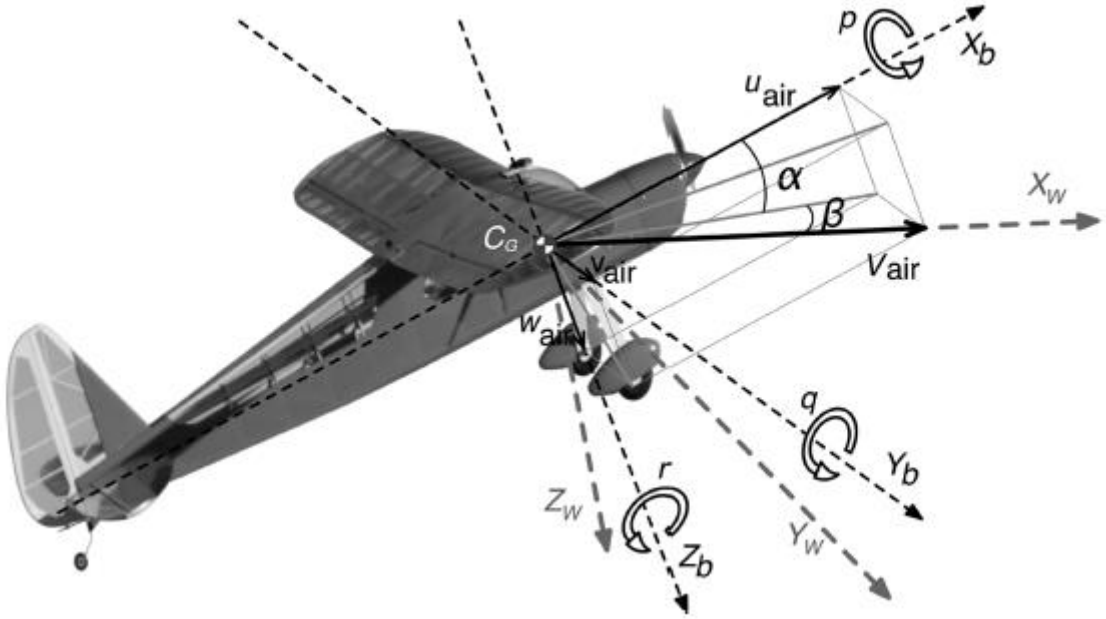


Figura 2. Ejes locales, y velocidad del viento. (J. Velasco-Carrau)

Donde α y β son los ángulos de ataque y deslizamiento en rad, respectivamente, y pueden calcularse según la Figura 2:

$$\alpha = \arctan(^{w_{air}}/_{u_{air}}) \quad (16)$$

$$\beta = \arctan(^{u_{air}}/_{V_{air}}) \quad (17)$$

Siendo V_{air} el módulo del vector velocidad del viento, y sus componentes (u_{air} , v_{air} , w_{air}).

Por último, los coeficientes C_L y C_D son los coeficientes de propulsión y arrastre respectivamente, calculados en (10) y (11), y se relacionan con los coeficientes C_X y C_Z:

$$C_L = -C_z \cos\alpha + C_X \sin\alpha \quad (18)$$

$$C_D = -C_X \cos\alpha - C_Z \sin\alpha \quad (19)$$

Para el caso del Kadett, se muestran a continuación los valores de los parámetros utilizados en el modelo, tanto las propiedades físicas como la masa o momentos de inercia del avión como los coeficientes aerodinámicos empleados. Todos los parámetros de la aeronave, así como los valores de los coeficientes aerodinámicos han sido proporcionados por el Departamento de Ingeniería de Sistemas y Automática (DISA).

Nombre de la variable	Significado	Valor (Unidades en S.I)
m	Masa	6.3 Kg
g	Intensidad del campo gravitatorio	9.81 m/s^2
b	Envergadura	2.38 m
$\bar{c}$	Cuerda del ala	0.4 m
S	Superficie aerodinámica	0.9 m^2
I_x	Momento de inercia en eje X	0.5789 Kg/m^2
I_y	Momento de inercia en eje Y	1.48 Kg/m^2
I_z	Momento de inercia en eje Z	1.9437 Kg/m^2
I_{xz}	Producto de inercia	0.05654 Kg/m^2
ρ	Densidad del aire	1.1617 Kg/m^3

A continuación, se detallan los valores de los coeficientes aerodinámicos tomados:

- Parámetros del coeficiente de sustentación C_L

Parámetro	Valor
C_{L_0}	0.329299
$C_{L_{Vair}}$	-0.31377
C_{L_α}	1.8854
$C_{L_{\alpha^2}}$	-2.7011
$C_{L_{\dot{\alpha}}}$	-54.6756
C_{L_q}	70.9757
$C_{L_{\delta_e}}$	0.410701

- Parámetros del coeficiente de resistencia C_D

Parámetro	Valor
C_{D_0}	0.126887
$C_{D_{Vair}}$	-0.028686
C_{D_α}	-0.4719
$C_{D_{\alpha^2}}$	2.7030
$C_{D_{\dot{\alpha}}}$	0
C_{D_q}	-18.3649
$C_{D_{\delta_e}}$	-0.226548

- Parámetros del coeficiente de momento de cabeceo C_m

Parámetro	Valor
C_{m_0}	0
$C_{m_{Vair}}$	0.0281592
C_{m_α}	0.162735
$C_{m_{\alpha^2}}$	-0.136745
$C_{m_{\dot{\alpha}}}$	9.95444
C_{m_q}	-15.3993
$C_{m_{\delta_e}}$	-0.77304

- Parámetros del coeficiente de fuerza lateral C_Y

Parámetro	Valor
C_{Y_0}	0
C_{Y_β}	-0.39393
C_{Y_p}	0.6121
C_{Y_r}	0.38049
$C_{Y_{\delta_a}}$	0.30623
$C_{Y_{\delta_r}}$	0.023562

- Parámetros del coeficiente de momento de alabeo C_l

Parámetro	Valor
C_{l_0}	0
C_{l_β}	-0.021969
C_{l_p}	-0.22666
C_{l_r}	0.02506
$C_{l_{\delta_a}}$	-0.11659
$C_{l_{\delta_r}}$	-0.0065928

- Parámetros del coeficiente de momento de guiñada C_n

Parámetro	Valor
C_{n_0}	0
C_{n_β}	0.049286
C_{n_p}	-0.18266
C_{n_r}	-0.1627
$C_{n_{\delta_a}}$	-0.058881
$C_{n_{\delta_r}}$	-0.085319

3.3. Proceso

Al iniciar la simulación, el programa primero asigna todas las constantes del sistema y lee las entradas del joystick. Una vez leídas las entradas, se introducen en el modelo y son integradas las ecuaciones dando como resultado las velocidades y posición del avión y los ángulos de Euler, así como otras variables necesarias para el cálculo de la siguiente iteración.

Tras obtener los resultados, se envía un datagrama (paquete de datos) en protocolo UDP al software de visualización FlightGear en el que se simula el comportamiento real del avión.

Con el objetivo de garantizar que la simulación sea en tiempo real, objetivo prioritario en la simulación HIL, el bucle no vuelve a comenzar hasta que se realizan todas las operaciones anteriormente mencionadas. De no realizarse controlando el tiempo de ejecución del algoritmo, se obtendrían resultados erróneos.

Para el cálculo del tiempo empleado en cada una de las iteraciones del bucle, se emplean funciones de la librería time con las que se calcula el tiempo requerido en la iteración, siendo la diferencia entre el tiempo al finalizar los cálculos y el tiempo de inicio.

Cabe destacar que el programa lleva a cabo dos líneas de ejecución simultáneamente con el objetivo de realizar el proceso en tiempo real. Una línea se encarga de la actualización de las variables de entrada provenientes de la lectura del joystick, mientras que la segunda línea de ejecución realiza el cálculo de las variables de estado del modelo y envía los resultados.

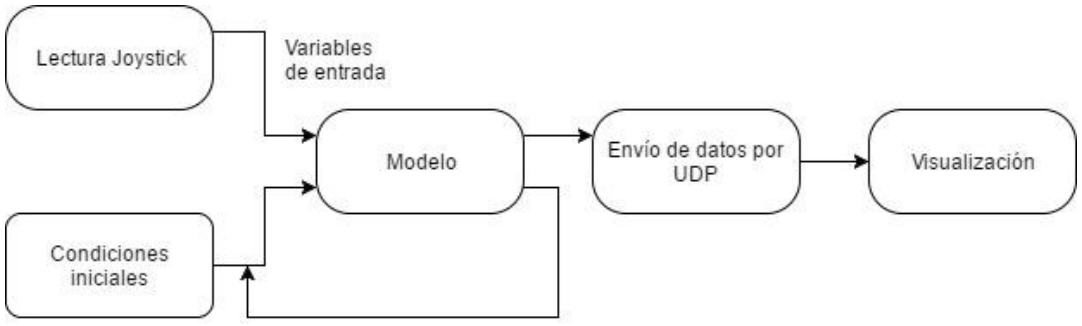


Figura 3. Flujograma del proceso

3.4. Integración

El comportamiento dinámico de la aeronave viene dado por las ecuaciones diferenciales expuestas previamente, donde se contemplan las características geométricas y físicas. Para obtener las soluciones del sistema de ecuaciones es necesario utilizar algún algoritmo de integración. Python dispone de la librería Scipy, en la cual se pueden encontrar multitud de métodos de integración. Tras realizar una prueba con diferentes métodos, (vode, zvode, lsoda, dopri5, dop853) y basándonos en el tiempo que requiere llevar a cabo un paso de integración con cada uno de los métodos, se ha escogido el método vode por ser el más rápido, lo que permite adaptarse de mejor manera en los márgenes del tiempo real que se nos plantea. El método vode es una optimización de los métodos de Runge-Kutta. muy utilizado en resolución de problemas con ecuaciones diferenciales, en especial para problemas de valor inicial.

Los métodos de Runge-Kutta permite solucionar ecuaciones diferenciales de la forma $y'(t) = f(t, y(t))$ con $f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ donde Ω es un conjunto abierto, siendo el valor inicial $(t_0, y_0) \in \Omega$. Entonces el método Runge-Kutta tiene la expresión siguiente:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i$$

Donde h es el paso por iteración y los coeficientes k_i son términos de aproximación intermedios.

$$k = f(t_n + hc_i\,, y_n + h \sum_{j=1}^{s} a_{ij} k_j$$

3.5. Protocolo UDP

El protocolo UDP (User Datagram Protocol) permite el intercambio de datagramas a través de una red sin ser necesario el establecimiento de conexión previo, ya que el datagrama incluye en su cabecera la información de direccionamiento. No tiene control de flujo ni confirmación, por lo que los paquetes enviados pueden variar el orden y además no hay certeza de si el paquete ha llegado correctamente. Sin embargo, gracias a esto se carga menos la red que con el uso del protocolo TCP (Transmission Control Protocol) que sí garantiza el envío de los paquetes ordenados. UDP es utilizado principalmente para transmisión de video y voz en tiempo real a través de una red, ya que, aunque algunos paquetes no se envíen correctamente, es más importante en estas tareas un envío rápido de datos que volver a enviar los datagramas que no han alcanzado el destino.

El requisito del tiempo es una cuestión principal para el funcionamiento del sistema HIL, por lo que el protocolo UDP nos proporcionará una buena velocidad de envío. Como se explica anteriormente es prioritario la fluidez del tráfico de datos que la recepción de todos los paquetes.

Para el testeo al utilizar envíos por UDP o por TCP se recomienda el uso de Packet Sender, un programa que permite entre otras opciones comprobar el envío y recepción de paquetes de datos por un puerto definido. Puede descargarse desde la página oficial de Packet Sender de manera gratuita. https://packetsender.com/

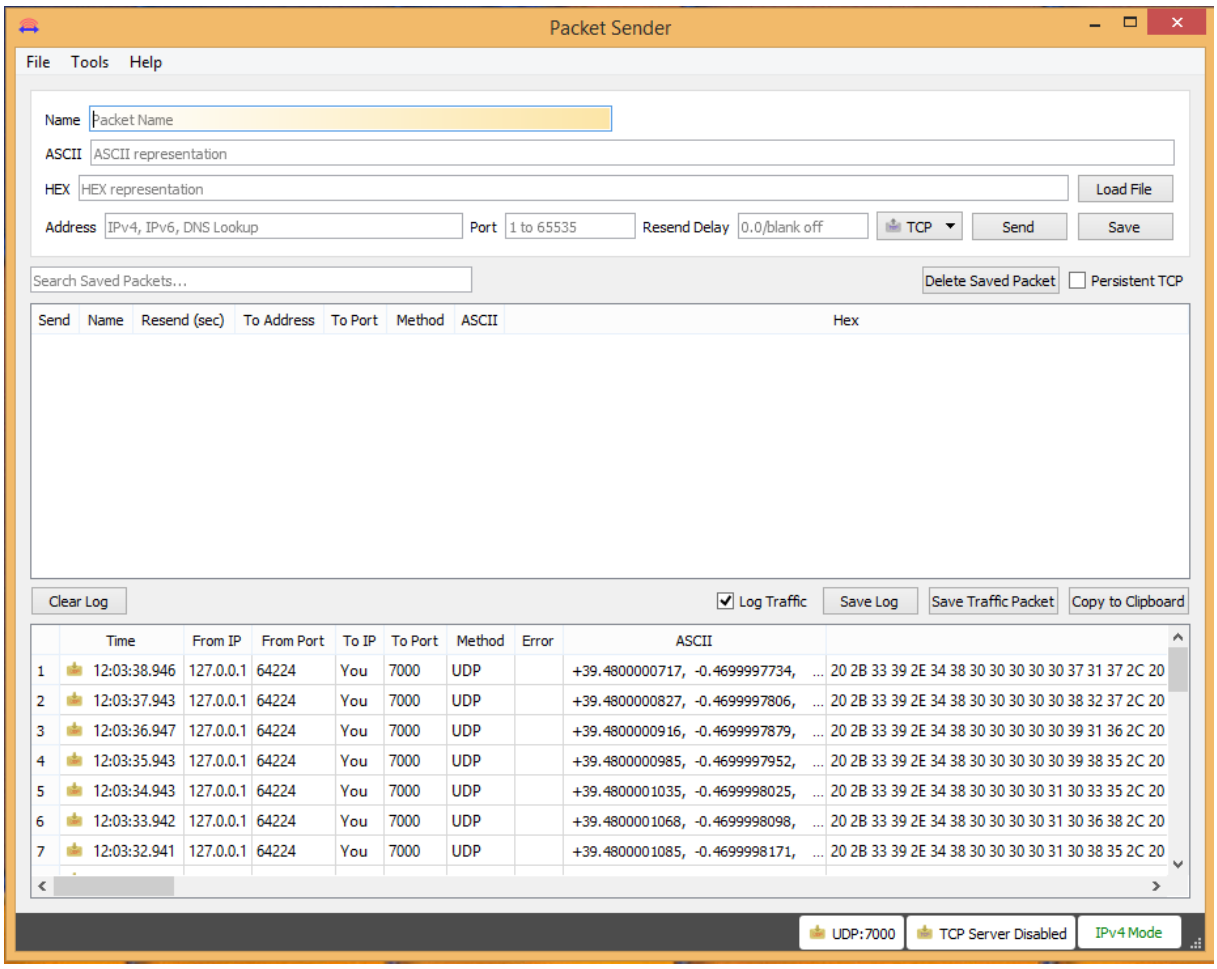


Figura 4. Packet Sender

La comunicación que se establece entre el simulador HIL y FlightGear lleva impuesto el formato para que los datos enviados y recibidos sean interpretados de la misma forma.

Se ha preparado un archivo llamado “input_external_simulation” que servirá de protocolo para la lectura de FlightGear, de forma que si los datos son recibidos en el formato indicado serán leídos correctamente. Este archivo debe copiarse en la carpeta “Protocol” donde está instalado FlightGear.

El archivo contiene lo siguiente:

```
<?xml version="1.0"?>
<PropertyList>
<generic>
   <input>
      <line_separator>newline</line_separator>
      <var_separator>,</var_separator>


      <chunk>
         <name>/position/latitude-deg</name>
         <node>/position/latitude-deg</node>
         <type>float</type>
         <format>%+15.10f</format>
       </chunk>

      <chunk>
         <name>/position/longitude-deg</name>
         <node>/position/longitude-deg</node>
         <type>float</type>
         <format>%+15.10f</format>
      </chunk>

      <chunk>
         <name>/position/altitude-ft</name>
         <node>/position/altitude-ft</node>
         <type>float</type>
         <format>%+15.5f</format>
       </chunk>

     <chunk>
         <name>/orientation/roll-deg</name>
         <node>/orientation/roll-deg</node>
         <type>float</type>
         <format>%+010.5f</format>
      </chunk>

      <chunk>
         <name>/orientation/pitch-deg</name>
         <node>/orientation/pitch-deg</node>
         <type>float</type>
         <format>%+010.5f</format>
      </chunk>

      <chunk>
         <name>/orientation/yaw-deg</name>
         <node>/orientation/yaw-deg</node>
         <type>float</type>
         <format>%+010.5f</format>
      </chunk>


   </input>

</generic>
</PropertyList>
```

Como puede observarse Flight Gear recibirá latitud y longitud en grados, altitud en pies y los ángulos de Euler en grados para la representación gráfica. Los formatos son los indicados en cada una de las variables.

4. HARDWARE

4.1. Raspberry Pi

Raspberry Pi es un sistema embebido de bajo coste económico formado por una única placa base. Su diseño proporciona una plataforma con elevada portabilidad, compatibilidad y de fácil manejo sobre la cual es posible desarrollar proyectos de diversos ámbitos tecnológicos, con útiles que, en otros casos son de un elevado precio.

La Raspberry Pi funciona como cualquier PC tradicional. Es posible reproducir videos, trabajar con programas de ofimática, navegar por Internet o cualquier otra tarea básica. Ofrece múltiples posibilidades de entrada y salida además de los GPIO pins que permite al usuario definir cada uno de ellos como entrada o salida según su necesidad.

Para el desarrollo del proyecto se ha utilizado la Raspberry Pi 3, un modelo nuevo con mejores prestaciones respecto al modelo anterior. Principalmente interesa la mejora del procesador, que permitirá realizar la simulación de forma más rápida, acercando el sistema al tiempo real.

- Procesador 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LAN
- Bluetooth 4.1
- Bluetooth Low Energy (BLE)
- 1GB RAM
- 4 puertos USB
- 40 GPIO pins. (General Purpose Input/Output, son pines sin función de entrada salida determinada, pueden ser programadas a voluntad por el usuario)
- Puerto HDMI
- Puerto Ethernet
- Combined 3.5mm audio jack and composite video
- Camera interface (CSI)
- Display interface (DSI)
- Micro SD card slot
- VideoCore IV 3D graphics core

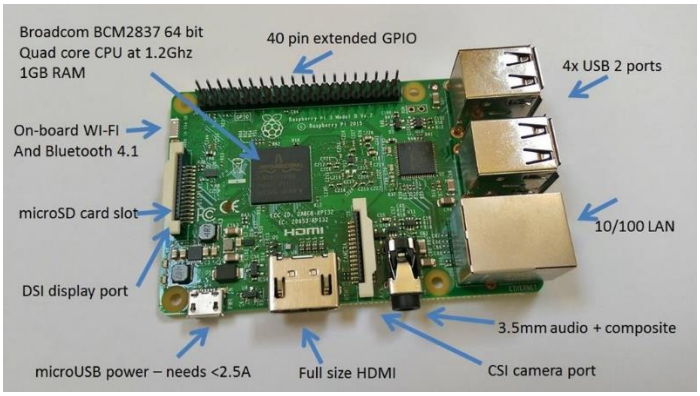


Figura 5. Raspberry Pi, elementos. (http://www.microsolution.com.pk/raspberry-pi3-pakistan/)

4.2. Joystick

El joystick utilizado ha sido el Saitek Cyborg Evo Force, periférico con conexión USB con el que se controlaran las superficies de control de la aeronave, de elevadores, cola y alerones del avión simulado. La palanca también simulará la palanca del motor. Los drivers se deben instalar automáticamente al conectar el joystick. En caso de no funcionar correctamente se puede consultar el manual en la página web oficial de Saitek, referenciada en la bibliografía.

Figura 6. Joystick Saitek Cyborg Evo Force. (http://www.saitek.com)

5. SOFTWARE

5.1. Instalación

Para el correcto funcionamiento del simulador en la Raspberry Pi es necesaria la instalación de los siguientes elementos de software, entre ellos el sistema operativo, Python y las librerías utilizadas para llevar a cabo la programación.

- S.O.: **Raspbian Wheezy** (Linux kernel)

- Python 3.5

- Librerías:

  - **Numpy:** Tratamiento de vectores y matrices, funciones matemáticas de alto nivel.
  - **Scipy:** Herramientas y algoritmos matemáticos para uso científico, tales como optimización, integración numérica, interpolación o resolución de ODEs.
  - **Matplotlib:** Representación gráfica, tablas, etc.
  - **Pygame:** Desarrollo de videojuegos para Python. Se utiliza principalmente para el control de la entrada de joystick.
  - **Time:** Manejo de fechas y tiempos. Su utilización será para el control del tiempo empleado entre las iteraciones.
  - **Socket:** Permite el envío de paquetes de datos con diferentes protocolos. Permitirá la comunicación entre el intérprete de Python y el visualizador Flight Gear.

5.1.1. Sistema operativo. Raspbian Debian Wheezy.

El sistema operativo Raspbian es una distribución de Linux libre, basado en Debian Wheezy optimizado para el uso en Raspberry Pi. Su lanzamiento inicial fue en 2012. Raspbian permite el cálculo en coma flotante y también herramientas de programación en Python.

Raspbian fue creado con el objetivo de dotar a la Raspberry Pi de un S.O. de fácil instalación y con alta funcionalidad.

Para instalar Raspbian en la Raspberry Pi necesitaremos los siguientes elementos:

- Una Raspberry Pi.
- Una tarjeta SD. Deberá tener suficiente memoria como para guardar el sistema operativo y los demás elementos, como mínimo el intérprete de Python, el IDLE y los demás programas que instalemos. El Flight Gear no es necesario, ya que la visualización en éste puede darse en otro dispositivo, tal y como se plantea en nuestro caso.
- Un cable de alimentación con conexión microUSB. Se puede utilizar un cable microSD similar a los cargadores de los teléfonos móviles.
- Monitor: Si queremos visualizar el s.o., puede conectarse con un cable HDMI o por conexión remota a otro ordenador.
- Teclado y Ratón USB: Aportan funcionalidad al sistema. Con teclado, ratón y monitor dispondríamos de un ordenador completo.
- Cable de red o adaptador Wifi. Necesario para tener conexión a Internet y poder actualizar la Raspberry Pi con las librerías directamente.

Para instalar el sistema operativo en la tarjeta SD podemos descargarlo de la página web oficial de Raspberry Pi (htttp://www.raspberrypi.org/downloads/). Aquí se nos ofrecen dos posibilidades para instalar correctamente Raspbian en la tarjeta SD.

El método más sencillo es utilizar el asistente de instalación NOOBS (New Out Of Box Software) que grabará Raspbian en la SD. Sólo es necesario insertar la tarjeta SD en el lector del ordenador (o en un adaptador que lo permita) y formatearla. Puede utilizarse la opción que nos da Windows o utilizar el programa SD Card Formatter o similar.

El segundo método es descargar de la página la imagen del sistema operativo. Para ello necesitaremos un programa que permita la grabación de imágenes. El equipo técnico de Raspberry Pi recomienda en su página web, el uso del programa Win32 Disk Imager para Windows o Etcher por su simplicidad, recomendado en la página oficial de Raspberry Pi para realizar esta tarea. Puede descargarse en (https://etcher.io/).

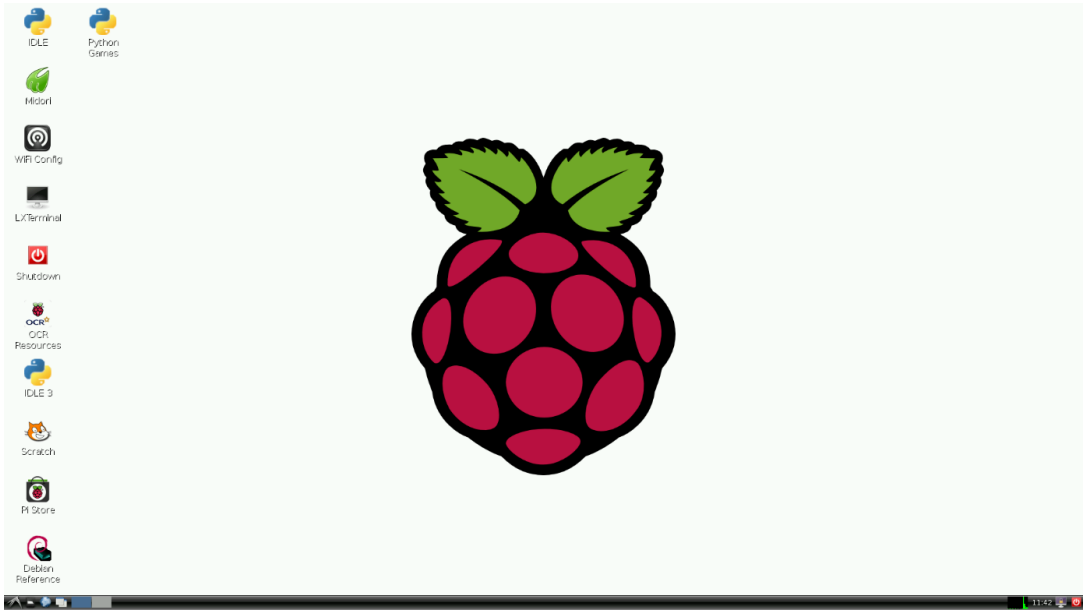


Figura 7. Escritorio básico de Raspbian.

5.1.2. Instalación de las librerías.

Una vez instalado Raspbian podemos acceder a la Raspberry Pi. Será necesario instalar las librerías utilizadas para que el intérprete de Python pueda ejecutar el código.

Para la instalación se abre la terminal del sistema (LXTerminal) y se teclea los comandos siguientes.

Figura 8. Terminal LX de Rasbian.

- Instalación de Numpy:

>> sudo apt-get install python-numpy

- Instalación de Scipy:

>> sudo apt-get install python-scipy

- Instalación de Matplotlib:

>> sudo apt-get install python matplotlib

- Instalación de Pygame:

Para instalar pygame se debe descargar la librería de la página web (https://www.pygame.org/download.shtml) y seleccionar la opción para el sistema operativo que se requiera. En este caso escogeremos la opción optimizada para Debian. Una vez descargado se instalara desde la terminal introduciendo la siguiente sentencia.

>> pip install pygame –user

Se utiliza en este caso el instalador pip. Para instalar pip o actualizarlo será necesario escribir en la consola:

En Linux o macOS:

>> pip install -U pip

En Windows:

>> python -m pip install -U pip

5.1.3. IDE Spyder

Un IDE (Integrated Development Environment), entorno de desarrollo integrado en español, es una aplicación informática que proporciona servicios para facilitar la creación, modificación, depuración e implementación de código. Normalmente tienen un editor de código fuente, un depurador, un compilador y un intérprete. Pueden ser para un único lenguaje de programación o puede ser multiplataforma y soportar diferentes lenguajes.

Se ha escogido el IDE Spyder 3 para el desarrollo del código del simulador.

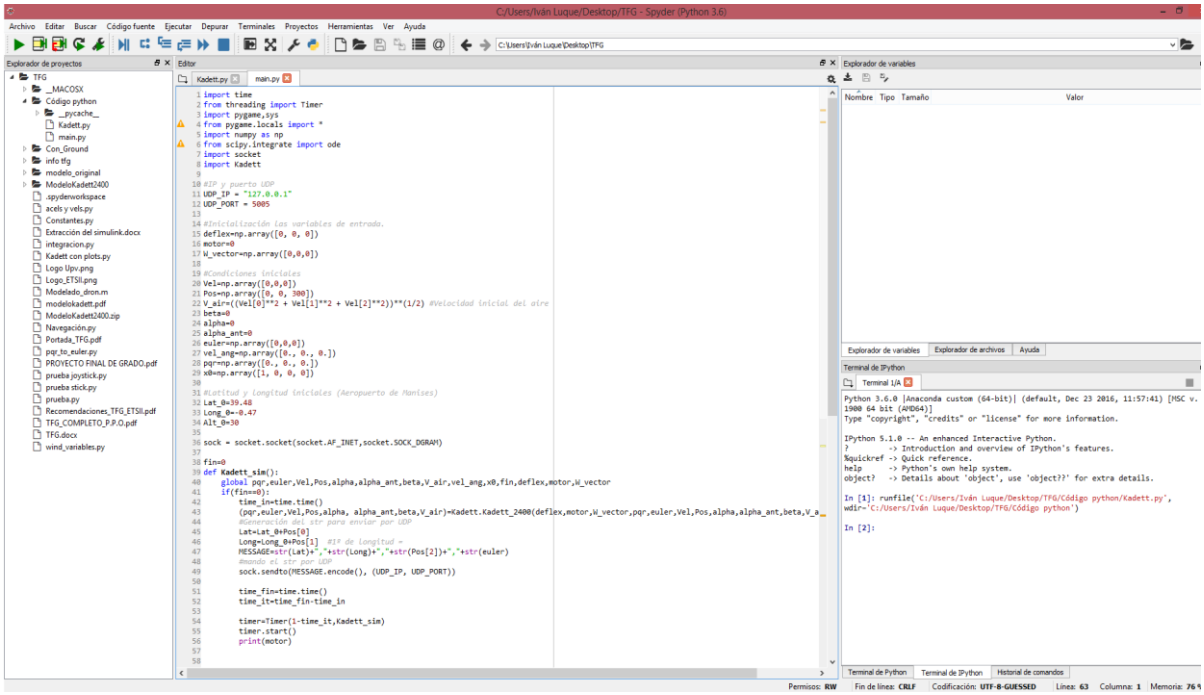


Figura 9. Interfaz de Spyder.

Spyder permite la personalización de la interfaz con múltiples herramientas. En el caso de la figura se puede ver a la izquierda un explorador de proyectos, en el centro el editor de código fuente y a la derecha un explorador de variables y la consola de Python.

5.1.4. Flight Gear

Para la visualización de los resultados obtenidos de la integración del modelo físico de la aeronave se utilizará el software Flight Gear. Se trata de un simulador de vuelo multiplataforma y libre. Este software se puede descargar gratuitamente desde la página web oficial (http:// www.flightgear.org).

El simulador dispone de una extensa base de datos, del mapeado terrestre y de multitud de modelos de aeronaves. Además, permite un sistema abierto y flexible de modelado de aviones. Es posible cargar un modelo físico generado externamente y simular su comportamiento en una interfaz gráfica.

Figura 10. Interfaz gráfica de FlightGear. (http://www.flightgear.org)

Para cargar correctamente los escenarios es necesario instalar y habilitar TerraSync una herramienta que FlightGear utiliza automáticamente en la carga inicial del programa.

Se ha preparado un Script que se encarga de iniciar FlightGear con la configuración necesaria para el caso que nos ocupa. El archivo se llama FGConfig.bat y si lo iniciamos desde la terminal del sistema operativo abrirá FlightGear. Si lo abrimos con un editor de texto podemos encontrar lo siguiente:

C:

cd C:\Program Files\FlightGear

SET FG_ROOT=C:\Program Files\FlightGear\data

.\\bin\fgfs --aircraft=c172p --fdm=external --enable-terrasync --generic=socket,in,1000,localhost,7000,udp,input_external_simulation --fog-fastest --disable-clouds --start-date-lat=2004:06:01:09:00:00 --disable-sound --in-air --enable-fuel-freeze --enable-freeze --airport=KSFO --altitude=7224 --heading=113 --offset-distance=4.72 --offset-azimuth=0

Principalmente el Script abre FlightGear e impone una serie de configuraciones. Habilita Terrasync, selecciona que el avión visualizado será el Cessna 172p e indica que las variables serán tomadas de forma externa por UDP por el puerto 7000 según "input_external_simulation", explicado previamente en el apartado 3.5.

5.2. Código del simulador

El código del simulador HIL del Kadett se compone de dos archivos desarrollados en Python. El archivo llamado main.py asigna los valores de los parámetros del sistema y gobierna las dos líneas de ejecución. Por un lado, una de las líneas, realiza la lectura de los valores de las variables de entrada provenientes del joystick y las actualiza periódicamente. En la otra línea de ejecución se hace una llamada a la función Kadett.py, que tiene implementado el modelo físico de la aeronave, y da como variables de salida las necesarias para su visualización en Flight Gear. La conexión entre el intérprete de Python y el programa de representación se realiza también en la función main.py gracias a la librería que permite la comunicación UDP.

5.2.1. Función main

La función main se encarga de controlar las dos líneas que se dan paralelamente. Un bucle se encarga del tratamiento de las señales de entrada del joystick mediante funciones de control de eventos de la librería Pygame. Dicho bucle además de la lectura, crea una ventana que sirve como método para la detención del programa. En la ventana se muestran el estado de las variables de importancia. El otro bucle toma las variables de entrada y las introduce en el modelo, calculando así la posición y velocidad de la aeronave que serán enviadas por formato UDP al software de visualización.

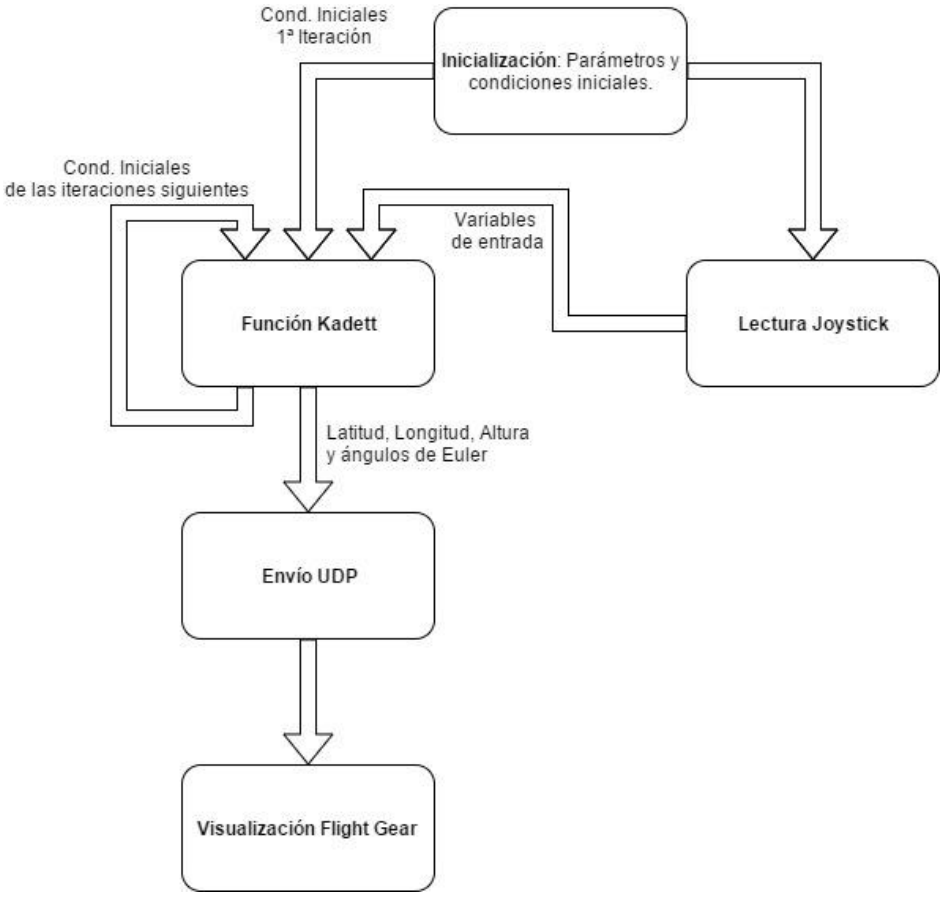


Figura 11. Flujograma función main

5.2.1.1. Importación de módulos y librerías.

En primer lugar, el programa importa los módulos y librerías necesarios para el funcionamiento adecuado del resto de código. También se importa la función Kadett, la encargada del cálculo de las variables del modelo.

```
1 import time
2 from threading import Timer
3 import pygame,sys
4 from pygame.locals import *
5 import numpy as np
6 import socket
7 import Kadett
```

Como puede observarse, no ha sido importada la librería Matplotlib. Esto se debe a que, para evitar tiempos mayores de ejecución, en la versión final del programa ha sido eliminada la creación de gráficas. No es estrictamente necesario para el funcionamiento del simulador HIL, aun así, el uso de la librería Matplotlib fue útil en la fase de diseño para el testeo del código, pero una vez comprobado ha sido eliminado.

5.2.1.2 Definición de las condiciones iniciales.

A continuación, se definen valores iniciales de las entradas y las condiciones iniciales necesarias para efectuar las iteraciones siguientes. Obviamente las condiciones iniciales pueden ser modificadas por el usuario en caso de requerir una simulación diferente a la aquí realizada.

```
#IP y puerto UDP
UDP_IP = "127.0.0.1"
UDP_PORT = 5005

#Inicialización las variables de entrada.
deflex=np.array([0, 0, 0])
motor=0
W_vector=np.array([0,0,0])

#Condiciones iniciales
Vel=np.array([0,3,0])
Pos=np.array([0, 0, 0])
V_air=((Vel[0]**2 + Vel[1]**2 + Vel[2]**2))**(1/2) #Velocidad inicial del aire
beta=0
alpha=0
alpha_ant=0
euler=np.array([0,0,0])
vel_ang=np.array([0., 0., 0.])
pqr=np.array([0., 0., 0.])
x0=np.array([1, 0, 0, 0])

#Latitud y longitud iniciales (Aeropuerto de Manises)
Lat_0=39.48
Long_0=-0.47

Lat_fin=Lat_0+10/111.12
Long_fin=Long_0+10/111.12

Lat=Lat_0
Long=Long_0
```

Se ha seleccionado el aeropuerto de Manises como punto de inicio para la visualización en Flight Gear.

Utilizando la hipótesis de Tierra plana se calcula la latitud y longitud en cada momento a través de una interpolación.

5.2.1.3. Bucle de simulación

Antes de ejecutar el bucle, se crea un objeto socket con la ayuda de la librería de mismo nombre. El objeto creado tiene una subrutina que permite el envío de un mensaje a una IP y puerto definidos.

Se define una función llamada Kadett_sim donde se pasarán todas las variables definidas anteriormente. Para ello se utiliza la función global, que pasa las variables globales del código a variables locales de la función Kadett_sim.

Al ejecutar el código, se toma el tiempo de inicio, previo al cálculo del modelo y del envío de las señales. Una vez realizadas las tareas se vuelve a tomar el tiempo con la ayuda de la función time, y se calcula el tiempo empleado en una iteración. Dicho tiempo debe ser lo más reducido posible para lograr el requisito de tiempo real, al menos debe lograrse que el tiempo necesario en una iteración sea menor que el paso de integración de nuestro modelo.

La función Timer realiza una nueva ejecución de la función Kadett_sim, una vez haya transcurrido un tiempo equivalente al paso de integración. En este modelo se ha adoptado un paso de 1 ms. Por tanto, el tiempo que debe esperar la función Timer a hacer la llamada recursiva será la diferencia entre el paso de integración del modelo y el ya transcurrido en la ejecución.

Es necesario testear el tiempo utilizado en el cálculo del modelo y el envío de los datos por UDP, en el testeo realizado se cumple obteniendo un tiempo empleado en la iteración del orden de 0,3 ms, siendo menor que el paso de integración. Si no se cumpliese el susodicho requisito del tiempo real habría que conseguir reducirlo para el correcto funcionamiento del sistema HIL. Puede lograrse de diferentes formas:

- Mejorando el código, retirando cálculos que se repiten en todas las iteraciones y reemplazándolos por los resultados.
- Cambiar el hardware por otro con mejores prestaciones. No realizable en el presente TFG pues la utilización de la Raspberry Pi es uno de los requisitos marcados por el cliente.
- Aumentar el paso de integración. Aunque no es posible aumentarlo en gran medida debido a que el comportamiento dinámico del vehículo aéreo es muy inestable.

```
45 def sprintf(buf, format, *args):
46     buf.write(format % args)
47
```

Se define una función, de nombre sprintf que se encarga de escribir en un buffer con un formato específico. La recepción de datos en Flight Gear ha sido preparada para que los datos lleguen en un orden y formato específico por lo que debe mandarse de tal forma para que el funcionamiento de la visualización sea correcto.

```
fin=0
def Kadett_sim():
    global pqr,euler,Vel,Pos,alpha,alpha_ant,beta,V_air,vel_ang,x0,fin,deflex,motor,W_vector,Lat,Long
    if(fin==0):
        time_in=time.time()
        (pqr,euler,Vel,Pos,alpha, alpha_ant,beta,V_air)=Kadett.Kadett_2400(deflex,motor,W_vector,pqr,euler,Vel,Pos,alpha,alpha_ant,beta,V_air,vel_ang,x0)

        #Generación del str para enviar por UDP a Flight Gear
        Lat=Lat_0+Pos[0]*(Lat_fin-Lat_0)/10000
        Long=Long_0+Pos[1]*(Long_fin-Long_0)/10000
        Alt=Pos[2]*3.28084  #Altura en pies
        euler_deg=euler*(180/np.pi)

        buf=io.StringIO()
        MESSAGE=sprintf(buf,'%+15.10f'','%+15.10f'','%+15.5f'','%+010.5f'','%+010.5f'','%+010.5f', Lat,Long,Alt,euler_deg[0],euler_deg[1],euler_deg[2])
        s=buf.getvalue()

        #mando el str por UDP
        sock.sendto(s.encode(),(UDP_IP, UDP_PORT))

        time_fin=time.time()
        time_it=time_fin-time_in

        timer=Timer(1-time_it,Kadett_sim)
        timer.start()
        print(s)
```

La variable fin que se observa en el código se utiliza para poder salir de la ejecución de forma segura. Cuando se termina la ejecución, fin se actualiza desde la otra línea de ejecución y en la siguiente iteración se para el programa.

Como puede observarse, en la función Kadett_sim además de calcular el modelo físico se adecúan los resultados para enviarlos al programa Flight Gear. Tal y como ha sido preparada la recepción de datos, se deben enviar: Latitud y Longitud en grados, altura en pies y los ángulos de Euler en grados.

5.2.1.4. Bucle de lectura de las entradas

En primer lugar, se inicia el módulo joystick de Pygame y se busca e inicia el joystick. Se crea también una ventana que tiene como función la visualización de las lecturas de los ejes del joystick. Además, sirve para la gestión del evento QUIT, definido como pulsar la cruz de la ventana, al hacerlo el programa cierra toda la simulación, cierra los módulos abiertos y finaliza el programa saliendo de los 2 bucles que se estuviesen ejecutando.

Mientras el programa se está ejecutando se realiza la lectura de los ejes del joystick y, tras adecuar con una interpolación los valores a los de las señales de entrada se guardan en los vectores deflex y motor. Las lecturas se dan entre -1 y 1 mientras que, en el modelo las señales van entre -0.4 y 0.4 radianes para las deflexiones y entre 0 y 1 para la palanca de motor.

```
pygame.joystick.init()
count=pygame.joystick.get_count()
joysticks = pygame.joystick.get_count()
ventana=pygame.display.set_mode((500,500))
pygame.display.set_caption("Simulador Kadett")
joystick = pygame.joystick.Joystick(0)
joystick.init()
numaxes=joystick.get_numaxes()
fuente = pygame.font.Font(None, 20)


while True:
    for event in pygame.event.get():
        if event.type == QUIT:  #Evento QUIT es pulsar la cruz de la ventana.
            fin=1
            pygame.quit()   #El programa en consecuencia se detiene.
            sys.exit()
    if joysticks:
        axes0=joystick.get_axis(0)#Lateral (Alerones) -1/1 izq/der
        axes1=joystick.get_axis(1)#Elevadores -1/1  abajo/arriba
        axes2=joystick.get_axis(2)#Palanca de motor -1/1 max/min
        axes3=joystick.get_axis(3)#Cola -1/1 izq/der


    #Adecuación de los valores obtenidos a las señales del modelo.
    aleron=axes0*0.4
    elev=-axes1*0.4
    motor=(axes2-1)/-2
    cola=axes3*0.4
    deflex=np.array([aleron,elev,cola])
    time.sleep(0.1)

    pygame.display.update() #Actualiza la ventana de pygame

```

5.2.2 Función Kadett

La función Kadett se encarga del cálculo de las variables de estado necesarias como las fuerzas y momentos, las velocidades angulares y lineales y la posición del avión. Para ello se ha utilizado programación funcional. Al realizar el código en funciones conseguimos diversos beneficios.

- Este paradigma de programación da como resultado un lenguaje limpio y elegante, que ayuda a una cómoda lectura y fácil comprensión.
- Las variables intermedias que no sean utilizadas serán borradas, permitiendo una mayor liberación del espacio en disco.

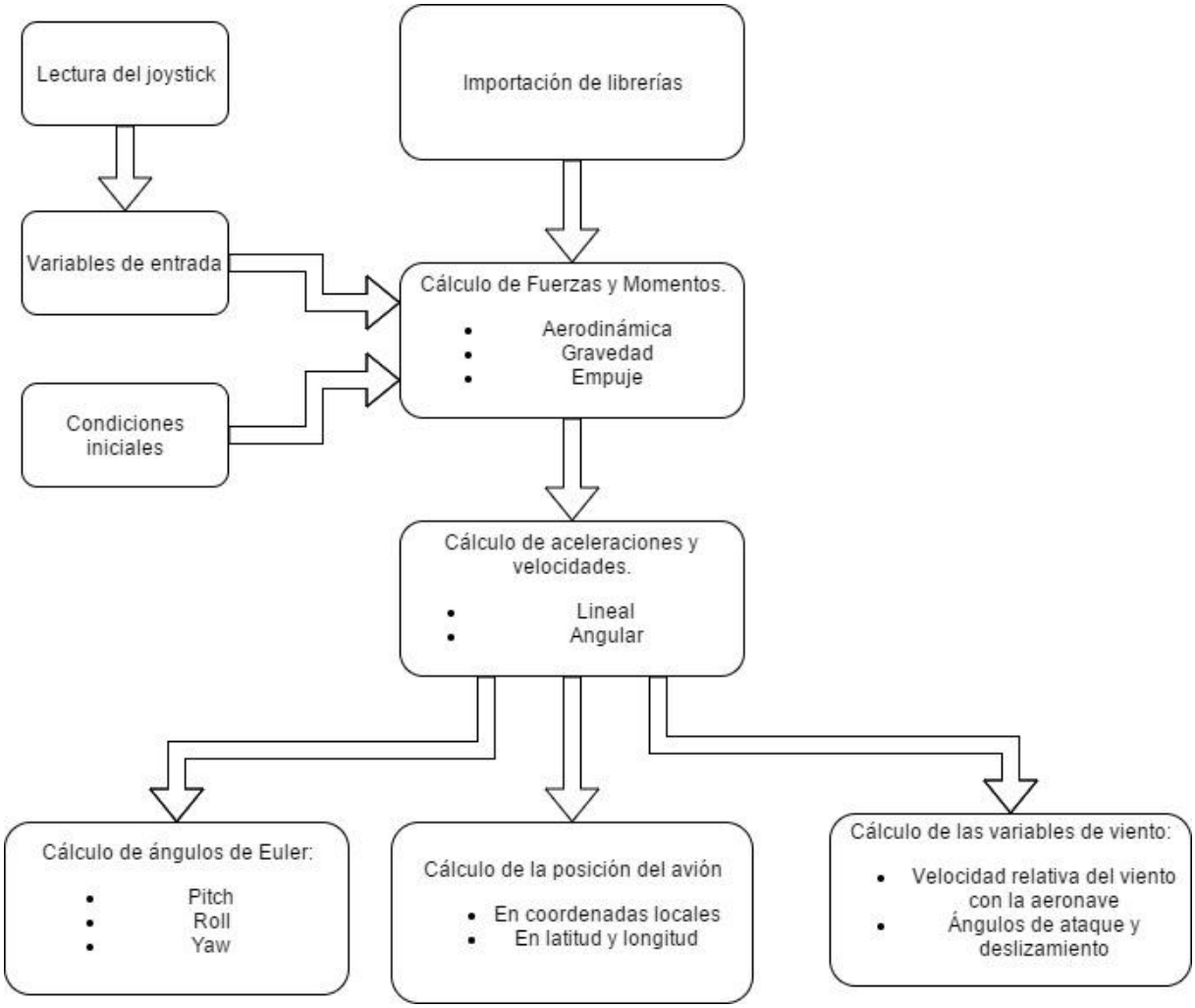


Figura 5. Flujograma de la función Kadett

Las primeras líneas de código realizan la importación de las librerías Numpy y Scipy que serán necesarias para el funcionamiento de las funciones.

```
1 import numpy as np
2 from scipy.integrate import ode
```

Se define la función Kadett_2400 que engloba el resto de funciones que constituyen el modelo. Las variables de entrada de la función son todas las necesarias para el modelo. Las variables de entrada del sistema: deflex (vector de orden 3, con las deflexiones de alerones, elevador y cola), motor (posición de la palanca de motor en tanto por uno) y W_vector (vector del viento). El resto de variables introducidas en la función toman inicialmente los valores indicados en la función main.py y son recalculadas tras cada iteración para ser reintroducidas en el modelo.

```
def Kadett_2400(deflex, motor, W_vector,pqr,euler,Vel,Pos,alpha,alpha_ant,beta,V_air,vel_ang,x0):
    #Definimos las funciones utilizadas y los parámetros del modelo.
    #Constantes
    m=6.3
    g=9.81
    S=0.9
    b=2.38
    c=0.4 #cuerda del ala
    dens_aire=1.1617
    Ix=0.5789;
    Iy=1.48;
    Iz=1.9437;
    Ixz=0.05654;
    dt=0.01

```

Además, se definen los parámetros constructivos de la aeronave. En caso de cambiar el avión podrían modificarse sus características desde aquí.

5.2.2.1. Cálculo de las fuerzas y momentos.

```
    def regressor(deflex, Vel ,beta, alpha, alpha_dot, vel_ang):
        V0 = 18.16
        alpha_dot=(alpha-alpha_ant)/dt
        var_lat=np.array([beta, vel_ang[0]*b/(2*V0), vel_ang[2]*b/(2*V0), deflex[0], deflex[2]])
        var_long=np.array([1, (Vel-V0)/V0, alpha, alpha**2, alpha_dot*c/(2*V0), vel_ang[1]*c/(2*V0), deflex[1]])
        return(var_lat,var_long)

```

La función regressor crea dos vectores con los términos de las ecuaciones del modelo aerodinámico lateral y longitudinal y las guarda en dos variables var_lat y var_long que serán introducidas en la siguiente función.

```
#Cálculo de fuerzas y momentos
def F_M(var_lat, var_long, Vel):
    #   Parámetros del coeficiente de sustentación
    CL0      =  0.329299
    CLV      =  -0.31377
    CLa      =  1.8854
    CLa2     =  -2.7011
    CLa_rate =  -54.6756
    CLq      =  70.9757
    CLde     =  0.410701
    CL_pol=np.array([CL0, CLV, CLa, CLa2, CLa_rate, CLq, CLde])
    #   Parámetros del coeficiente de resistencia
    CD0      = 0.126887
    CDV      = -0.0286686
    CDa      = -0.4719
    CDa2     = 2.7030
    CDa_rate = 0
    CDq      = -18.3649
    CDde     = -0.226548
    CD_pol=np.array([CD0, CDV, CDa, CDa2, CDa_rate, CDq, CDde])
    #   Parámetros del coeficiente de momento de cabeceo
    CM0      = 0
    CMV      = 0.0281592
    CMa      = 0.162735
    CMa2     = -0.136745
    CMa_rate = 9.95444
    CMq      = -15.3993
    CMde     = -0.77304
    CM_pol=np.array([CM0, CMV, CMa, CMa2, CMa_rate, CMq, CMde])

    #   Parámetros del coeficiente de fuerza lateral
    CYb  =  -0.39393
    CYp  = 0.6121
    CYr  = 0.38049
    CYda = 0.30623
    CYdr = 0.023562
    CY_pol=np.array([CYb, CYp, CYr, CYda, CYdr])
    #   Parámetros del coeficiente de momento de alabeo
    Clb  = -0.021969;
    Clp  = -0.22666;
    Clr  =  0.02506;
    Clda = -0.11659;
    Cldr = -0.0065928;
    Cl_pol=np.array([Clb, Clp, Clr, Clda, Cldr])
    #   Parámetros del coeficiente de momento de guiñada
    CNb  =  0.049286
    CNp  = -0.18266
    CNr  = -0.1627
    CNda = -0.058881
    CNdr = -0.085319
    CN_pol=np.array([CNb, CNp, CNr, CNda, CNdr])

```

Se crean los polinomios con los coeficientes aerodinámicos, se guardan en vectores que posteriormente son multiplicados por las variables obtenidas en la función regressor. Se obtienen así las ecuaciones del modelo aerodinámico completas.

```
        d_CA_CG=-0.05
        alpha=var_long[2]
        pres_din=(dens_aire*V_air**2)/2
        #Ecuaciones laterales
        Cy=CY_pol.dot(var_lat)
        Cl=Cl_pol.dot(var_lat)
        Cn=CN_pol.dot(var_lat)
        #Ecuaciones longitudinales
        CL=CL_pol.dot(var_long)
        CD=CD_pol.dot(var_long)
        Cm=CM_pol.dot(var_long)
        AC=np.transpose(np.array([CD,CL,Cm,Cy,Cl,Cn]))
        aux=(np.linalg.inv(np.array([[np.sin(alpha),-np.cos(alpha)],[-np.cos(alpha),-np.sin(alpha)]]))).dot(np.array([CL,CD]))
        Cx=aux[0]
        Cz=aux[1]
        AC_extra=np.array([[Cx],[Cz]])
        if -Vel >=-2:
            FA=np.array([0,0,0])
            M=np.array([0,0,0])
        else:
            FA=S*pres_din*np.array([Cx,Cy,Cz])
            M=S*pres_din*np.array([b*Cl,c*Cm-d_CA_CG*Cz,b*Cn+d_CA_CG*Cy])
        return(M,FA,AC,AC_extra)
```

Como puede observarse si la velocidad de la aeronave es menor a 2 m/s se desprecia las fuerzas y momentos aerodinámicos. La matriz aux se crea con el objetivo de obtener Cx y Cz a partir del resto de coeficientes aerodinámicos.

A continuación, se definen las funciones F_motor y F_gravedad que calculan las fuerzas de empuje y gravitatoria, respectivamente. De la misma forma que con las fuerzas y momentos aerodinámicas, se calculan respecto a los ejes locales de la aeronave.

```
    #Cálculo de la fuerza del motor
    def F_motor(motor):
        T=(-0.0292*(motor*100)**2+36.681*(motor*100)-107.5)*g/1000
        if T<=0:
            T=0
        FT=np.array([T,0,0])
        return(FT)

    #Cálculo de fuerza de la gravedad
    def F_gravedad(euler):
        FG=m*g*(np.array([-np.sin(euler[1]),np.sin(euler[0])*np.cos(euler[1]),np.cos(euler[0])*np.cos(euler[1])]))
        return(FG)
```

5.2.2.2. Aceleración y velocidad

El vector de aceleraciones lineales se calculará a partir de la fuerza resultante dividida entre la masa de la aeronave.

```
    #Cálculo de la aceleración lineal del avión
    def a_lin(FA,FT,FG):
        F=FA+FT+FG
        a_lineal=F/m
        return(a_lineal,F)
```

Se definen ahora las funciones v_angulares y v_lin, que calculan tras ser integradas las velocidades angulares y lineales respectivamente. La sintaxis correcta para el correcto funcionamiento de la integración numérica con Scipy es la mostrada aquí. Las funciones deben tener como variables de entrada: la variable respecto a la que se va a integrar, la variable a integrar y por último los parámetros que se pasan a la función. Como return de la función definida debe quedar la derivada de la variable integrada. Es decir que, en el caso de la integración de la aceleración, el tiempo y la velocidad deben ser las entradas junto al resto de parámetros que modifican el cálculo de ésta tras cada iteración, mientras que la aceleración aparecerá en el return de la función. Sabiendo esto las funciones quedan definidas como se muestran a continuación.

```
#Cálculo de las aceleraciones angulares del avión
def v_angulares(t,v_ang,M):
    A_aux=np.array([[1, (-Ixz/Ix)],[-Ixz/Iz, 1]])
    B_aux=np.array([[(M[0]-v_ang[1]*v_ang[2]*(Iz-Iy)+ v_ang[1]*v_ang[0]*Ixz)/Ix],
                    [(M[2] - v_ang[0]*v_ang[1]*(Iy-Ix)- v_ang[1]*v_ang[2]*Ixz)/Iz]])
    aux=np.linalg.inv(A_aux).dot(B_aux)
    p_dot=aux[0]
    q_dot=(M[1]-v_ang[0]*v_ang[2]*(Ix-Iz)-(v_ang[0]**2-v_ang[2]**2)*Ixz)/Iy
    r_dot=aux[1]
    return np.array([p_dot,q_dot,r_dot])

def v_lin(t,v_lin,a_lineal,pqr):
    C=np.cross(pqr,v_lin)
    V_dot=a_lineal+C
    return V_dot
```

5.2.2.3. Variables de viento

Se define una función que calcula, a partir de los vectores velocidad del avión, velocidad del viento y ángulos de Euler, los ángulos α y β y la velocidad relativa del viento con la aeronave. Cx, Cy y Cz son las matrices de rotación, multiplicando dichas matrices de rotaciones simples obtenemos la matriz de rotación compuesta en los tres ejes. Al multiplicar un vector por la matriz resultante se puede pasar de las coordenadas globales a las coordenadas locales del avión.

```
#Cálculo de las variables del viento. (Ángulos de incidencia y velocidad)
def wind_variables(V_vector, euler,W_vector):
    roll=euler[0]
    pitch=euler[1]
    yaw=euler[2]
    Cx=np.array([[1, 0, 0],[0, np.cos(roll), np.sin(roll)],[0, -np.sin(roll), np.cos(roll)]])
    Cy=np.array([[np.cos(pitch), 0, -np.sin(pitch)],[0, 1, 0],[np.sin(pitch), 0, np.cos(pitch)]])
    Cz=np.array([[np.cos(yaw), np.sin(yaw), 0],[-np.sin(yaw), np.cos(yaw), 0],[0, 0, 1]])
    W_vector_local=(Cx.dot(Cy.dot(Cz))).dot(W_vector)
    airspeed_vector=V_vector-W_vector_local
    u=airspeed_vector[0]
    v=airspeed_vector[1]
    w=airspeed_vector[2]
    airspeed=np.sqrt(u**2+v**2+w**2)
    alpha=np.arctan2(w,u)
    beta=np.arcsin(v/airspeed)
    return(alpha,beta,airspeed)
```

5.2.2.4. Posición

Para obtener la posición del avión, se requiere realizar una nueva integración de la velocidad lineal. Ya que queremos obtener la posición respecto a los ejes globales con el origen definido en nuestro origen de coordenadas (Pos_0 = [0, 0, 0]), será necesario multiplicar por la matriz de transformación de igual forma que se ha realizado para la obtención de los ángulos α y β.

```
156     def nav(t,Pos,Vel,euler):
157         roll=euler[0]
158         pitch=euler[1]
159         yaw=euler[2]
160         Cx=np.array([[1, 0, 0],[0, np.cos(roll), np.sin(roll)],[0, -np.sin(roll), np.cos(roll)]])
161         Cy=np.array([[np.cos(pitch), 0, -np.sin(pitch)],[0, 1, 0],[np.sin(pitch), 0, np.cos(pitch)]])
162         Cz=np.array([[np.cos(yaw), np.sin(yaw), 0],[-np.sin(yaw), np.cos(yaw), 0],[0, 0, 1]])
163         NAV_dot=np.linalg.inv(Cx.dot(Cy.dot(Cz))).dot(np.array([Vel[0],Vel[1],Vel[2]]))      #Velocidad en X,Y,Z
164         return(NAV_dot)
165
```

5.2.2.5. Ángulos de Euler

La obtención de los ángulos de Euler se realiza con dos funciones. La primera función quat_sol(t, r, pqr), que se ejecutará antes, crea un quaternión con el vector de las velocidades angulares pqr y lo multiplica por otro quaternión que da la función a integrar. El resultado de la integración de esta función se introduce en la función Euler_from_quat(q) que transforma de nuevo el resultado del dominio de los quaterniones al de los números reales. De dicha transformación se obtienen los ángulos de Euler.

El motivo por el cual se realiza el cambio de dominio a quaterniones es, salvar las discontinuidades que presentan las funciones arctan2, y arcsin. En las discontinuidades, la integración numérica puede producir un error, por lo que se recurre a esta extensión de los números reales con el objetivo facilitar el cálculo de la integración numérica. Los quaterniones son una extensión de los números reales, generada de manera similar a los números imaginarios. Se añaden las unidades imaginarias *i, j* y *k* a los números reales de forma que las bases de un quaternión son 1, *i, j* y *k.*

Presentan varias ventajas en el cálculo en comparación con la rotación por matrices. Concatenar quaterniones requiere menos operaciones y exige menos espacio de almacenamiento.

```
166     def euler_from_quat(q):
167         phi=np.arctan2(2*(q[2]*q[3]+q[0]*q[1]),(q[0]**2-q[1]**2-q[2]**2+q[3]**2))
168         theta=np.arcsin(-2*(q[1]*q[3]-q[0]*q[2]))
169         psi=np.arctan2(2*(q[1]*q[2]+q[0]*q[3]),(q[0]**2+q[1]**2-q[2]**2-q[3]**2))
170         euler=np.array([psi,theta,phi])
171         return(euler)
172
173     def quat_sol(t,r,pqr):
174         q=0.5*np.array([0,pqr[2], pqr[1], pqr[0]])
175         quat_dot=np.array([r[0]*q[0]-r[1]*q[1]-r[2]*q[2]-r[3]*q[3],
176                            r[0]*q[1]+r[1]*q[0]-r[2]*q[3]+r[3]*q[2],
177                            r[0]*q[2]+r[1]*q[3]+r[2]*q[0]-r[3]*q[1],
178                            r[0]*q[3]-r[1]*q[2]+r[2]*q[1]+r[3]*q[0]])
179         return[quat_dot[0],quat_dot[1],quat_dot[2],quat_dot[3]]
180
181
```

5.2.2.6. Ejecución del modelo

Una vez que el intérprete de Python ha cargado las funciones que se van a utilizar, sigue con la lectura del código y ejecuta las funciones previamente definidas.

```
    #Cálculo de las fuerzas y momentos.
    (var_lat,var_long)=regressor(deflex, V_air,beta, alpha, alpha_ant, pqr)
    (M,FA,AC,AC_extra)=F_M(var_lat, var_long, V_air)
    FT=F_motor(motor)
    FG=F_gravedad(euler)

    #Aceleración lineal
    (a_lineal,F)=a_lin(FA,FT,FG)

    #Integración de la aceleración angular.
    v_ang_integ = ode(v_angulares).set_integrator('vode', method='bdf')
    v_ang_integ.set_f_params(M)
    v_ang_integ.set_initial_value(pqr)
    pqr=v_ang_integ.integrate(v_ang_integ.t+dt)

    #Cálculo de la velocidad lineal del avión.
    v_integ = ode(v_lin).set_integrator('vode', method='bdf')
    v_integ.set_f_params(a_lineal,pqr)
    v_integ.set_initial_value(Vel)
    Vel=v_integ.integrate(v_integ.t+dt)

    #Cálculo de las variables del viento.
    alpha_ant=alpha
    (alpha,beta,V_air)=wind_variables(Vel, euler, W_vector)

    #Cálculo de la posición de la aeronave.
    integ_nav = ode(nav).set_integrator('vode', method='bdf')
    integ_nav.set_f_params(Vel,euler)
    integ_nav.set_initial_value(Pos)
    Pos=integ_nav.integrate(integ_nav.t+dt)

    #Obtención de los ángulos de Euler.
    Eul_quat=ode(quat_sol).set_integrator('vode', method='bdf')
    Eul_quat.set_f_params(pqr)
    Eul_quat.set_initial_value(x0)
    quat=Eul_quat.integrate(Eul_quat.t+dt)

    euler=euler_from_quat(quat)

    return(pqr,euler,Vel,Pos,alpha,alpha_ant,beta,V_air)
```

La función Kadett finalmente devuelve las variables necesarias para el funcionamiento correcto del código. Las funciones Pos y euler son las variables de estado que necesitamos enviar a Flight Gear para visualizar el comportamiento del avión, mientras que el resto de variables serán las entradas de la siguiente iteración bien como entradas de las funciones o bien como condiciones iniciales de las integraciones.

6. CONCLUSIONES Y TRABAJOS FUTUROS

Con el desarrollo del presente TFG se pone en manifiesto la flexibilidad que aporta el uso de un sistema embebido como la Raspberry Pi con lenguaje de programación Python. Estas herramientas nos permiten la creación de un sistema Hardware-In-The-Loop sólido y con buenas prestaciones a un coste mucho más reducido a lo acostumbrado en la industria actual.

Los simuladores HIL son realmente útiles para situaciones en las que se requiere ciclos cortos de desarrollo, reducir costes o en procesos que puedan llegar a ser peligrosos. Demostrar que puede llevarse a cabo la simulación de modelos complejos, como es el caso de la aeronave, en tiempo real y con un gasto en el hardware empleado realmente bajo puede originar una expansión de esta tecnología a industrias en las que actualmente no son utilizadas. Podría suponer una buena herramienta para mejorar la competitividad de una empresa a medio – largo plazo.

En cuanto al futuro del proyecto, hay múltiples posibilidades para mejoras y continuaciones futuras para el simulador HIL desarrollado. La primera posibilidad y más obvia sería diseñar un controlador automático, ya que el proyecto se realiza con el objetivo de conseguir desarrollar un sistema de control para el pilotaje autónomo del Kadett. Se podría lograr bien utilizando un hardware independiente, como un autómata industrial o bien se podría implementar en versión software, configurando un control PID según las necesidades del cliente. El controlador podría sustituir el joystick en su función actual de proporcionar las entradas del sistema.

También es posible añadir una función que genere gráficas con los resultados obtenidos en la integración del modelo. Cabe decir que, esto sí que ha sido realizado en la fase de diseño del código, pero por motivos de ahorro en el tiempo de ejecución ha sido eliminado del código definitivo una vez se había testeado el modelo.

El modelo del simulador es válido únicamente en vuelo y la simulación debe empezar desde el aire. Sería posible considerar las fuerzas y momentos que aparecen en las situaciones de despegue y aterrizaje para tener un simulador completo.

BIBLIOGRAFÍA

Manual del joystick Cyborg Evo Force:
http://www.saitek.com/manuals/Cyborg%20Evo%20Force%20Manual%20EFGIS.pdf

Documentación de Spyder 3:

https://pythonhosted.org/spyder/

Ayuda con Flight Gear:

http://www.flightgear.org/

wiki.flightgear.org

Manual Flight Gear:

http://flightgear.sourceforge.net/getstart-en/getstart-en.html

Modelo físico del Kadett 2400 de Graupner:

J. Velasco-Carrau, S. García-Nieto y J. V. Salcedo (2016). *Multi-Objective Optimization for Wind Estimation and Aircraft Model Identification.* Journal of Guidance Control and Dynamics Vol. 39

Manual Debian:

Hertzog, R., & Mas, R. (s.f.). *The Debian Administrator's Handbook.*

Documentación sobre Python:

Lutz, M. (2001). *Programming Python.* O'Reilly.

Raúl González Duque. (s.f). *Python para todos.*

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA HARDWARE-IN-THE LOOP SOBRE PLATAFORMA RASPBERRY PI PARA LA SIMULACIÓN DEL VEHÍCULO AÉREO KADETT 2400 DE GRAUPNER

Documento 2: Manual de Usuario

AUTOR: IVÁN LUQUE NAVARRO

TUTOR: JUAN MANUEL HERRERO DURÁ

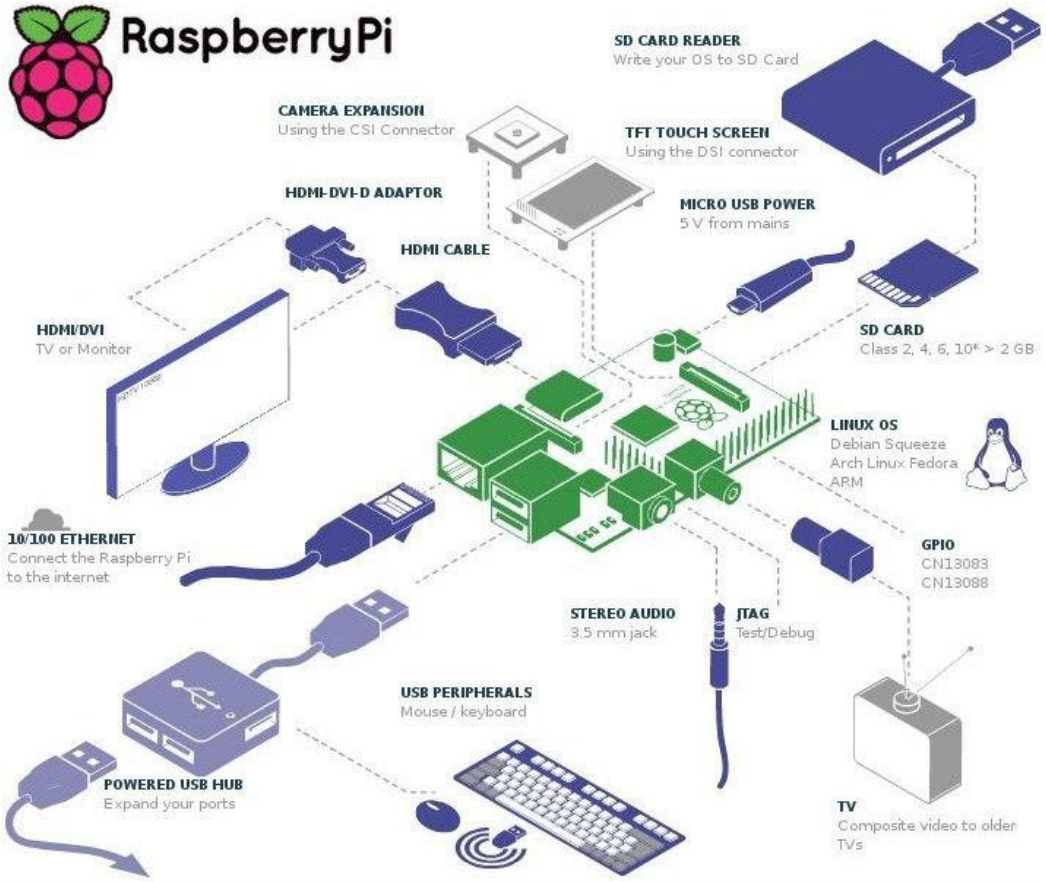
COTUTOR: SERGIO GARCÍA-NIETO RODRÍGUEZ

Curso Académico: 2016-17

MANUAL DE USUARIO

1. Conexión e instalación

Es posible conectar la Raspberry Pi de dos formas distintas: Con un monitor con puerto HDMI, y teclado y ratón USB o bien se puede realizar una conexión remota a un ordenador mediante un cable Ethernet.



Una vez logrado esto, será necesario alimentar la Raspberry Pi con un cargador MicroUSB de 5V similar a los utilizados en multitud de teléfonos móviles.

Windows 7 y superiores cuenta con una aplicación para configurar la conexión remota, aunque también es posible establecer la conexión SSH utilizando programas como PuTTY (www.putty.org/). Para poder establecer la conexión necesitaremos conocer la dirección IP de la Raspberry Pi.

Hay dos formas de obtener la dirección IP una vez conectada.

- Entrar en el router de la red que está la Raspberry Pi. Generalmente esto se puede lograr desde el navegador de cualquier ordenador tecleando en la barra de direcciones 192.168.1.1. Aquí debe aparecer la dirección IP asignada a todos los dispositivos vinculados a la red.

- Conectando la Raspberry con teclado y pantalla. En el terminal es posible teclear el comando *ifconfig* en el que aparece la dirección IP.

En la Conexión a Escritorio remoto se pide el Equipo y Usuario. En Equipo debemos poner la dirección IP obtenida y el Usuario predeterminado para Raspberry Pi es “Pi”.

Además de realizar la conexión, es necesario disponer del sistema operativo grabado en la tarjeta SD. Se recomienda la utilización de Raspbian wheezy cuya instalación esta detallada en la memoria del presente trabajo.

Para la visualización gráfica del comportamiento de la aeronave simulada se requiere la instalación de FlightGear. Puede descargarse de manera gratuita desde su página oficial (www.flightgear.org) hay versión para Windows, Mac y Linux. Para visualizar correctamente los escenarios se debe instalar la herramienta TerraSync que se ocupa de descargar el escenario próximo a las coordenadas de inicio del vuelo de forma automática al cargar FlightGear.

2. Utilización del simulador

Para iniciar la simulación, abrimos primero FlightGear. Se ha preparado un script con nombre FGconfig.bat que contiene la configuración inicial para hacer la visualización más sencilla. Iniciamos el programa desde éste para que FlightGear inicie.

El script contiene la habilitación de TerraSync así como el aeropuerto que debe cargar, por defecto el aeropuerto de San Francisco (KSFO). También se impone aquí que el modelo sea tomado de forma externa a partir de un protocolo UDP y el puerto que se toma. Estas opciones son importantísimas para la correcta visualización, por tanto, si se cambia algo de aquí debe cambiarse del código del simulador HIL y viceversa. El resto de opciones no son tan importantes para la simulación solo configuran ciertos aspectos de FlightGear como deshabilitar las nubes o bloquear el consumo de combustible.

Una vez iniciado FlightGear, abrimos el simulador desde la terminal. Se debe abrir el archivo main.py teniendo el archivo Kadett.py en la misma carpeta. Cuando el programa empiece a funcionar podremos visualizar el comportamiento del avión y su respuesta dinámica según vayamos posicionando el joystick. FlightGear empieza la visualización por defecto en pausa, para quitarla se debe presionar la tecla “P” además podemos cambiar entre múltiples vistas con la tecla “V”.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

TRABAJO FIN DE GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

DESARROLLO DE UN SISTEMA HARDWARE-IN-THE LOOP SOBRE PLATAFORMA RASPBERRY PI PARA LA SIMULACIÓN DEL VEHÍCULO AÉREO KADETT 2400 DE GRAUPNER

Documento 3: Presupuesto

AUTOR: IVÁN LUQUE NAVARRO

TUTOR: JUAN MANUEL HERRERO DURÁ

COTUTOR: SERGIO GARCÍA-NIETO RODRÍGUEZ

Curso Académico: 2016-17

PRESUPUESTO

CÓDIGO	UD.	DESCRIPCIÓN	RDTO	PRECIO	IMPORTE
CAP. 1		ANÁLISIS Y PLANIFICACIÓN DEL PROYECTO			
1.1.	Ud.	Reunión inicial con el cliente			
		Definición de los requerimientos del cliente. Establecimiento de las lineas generales.			
	h	Graduado en Ingeniería en Tecnologías Industriales	5	25	125
	%	Costes directos complementarios	0,02		2,5
				Coste total:	**127,5**
1.2.	Ud.	Estudio previo al inicio del proyecto			
		Recopilación de información y búsqueda de bibliografia.			
		Análisis de las soluciones disponibles.			
	semana	Graduado en Ingeniería en Tecnologías Industriales	2	1000	2000
	%	Costes directos complementarios	0,02		40
				Coste total:	**2040**
CAP. 2		DESARROLLO DEL PROYECTO			
2.1.	Ud.	Desarrollo del software			
		Selección e instalación del software auxiliar. Desarrollo del simulador HIL.			
	semana	Graduado en Ingeniería en Tecnologías Industriales	4	1000	4000
	%	Costes directos complementarios	0,02		80
				Coste total:	**4080**
2.2.	Ud.	Implementación del Hardware			
		Elementos físicos utilizados.			
	Ud.	Raspberry Pi 3 Model B	1	30	30
	Ud.	Joystick Cyborg Evo Force de Saitek	1	80	80
	h	Graduado en Ingeniería en Tecnologías Industriales	1	25	25
	%	Costes directos complementarios	0,02		2,7
				Coste total:	**137,7**
2.3.	Ud.	Testeo del producto.			
		Pruebas de funcionamiento, representación e interpretación de los resultados.			
	h	Graduado en Ingeniería en Tecnologías Industriales	8	25	200
	%	Costes directos complementarios	0,02		4
				Coste total:	**204**
CAP. 3		ELABORACIÓN DE DOCUMENTOS			
3.1.	Ud.	Redacción de los documentos del proyecto			
	h	Graduado en Ingeniería en Tecnologías Industriales	40	25	1000
	%	Costes directos complementarios	0,02		20
				Coste total:	**1020**

CAP. 1	ANÁLISIS Y PLANIFICACIÓN DEL PROYECTO	2167,50 €
CAP. 2	DESARROLLO DEL PROYECTO	4421,70 €
CAP. 3	ELABORACIÓN DE DOCUMENTOS	1020,00 €
PRESUPUESTO EJECUCIÓN MATERIAL (PEM)		7609,20 €
GASTOS GENERALES (13% del PEM)		989,20 €
BENEFICIO INDUSTRIAL (6% del PEM)		456,55 €
PRESUPUESTO DE EJECUCIÓN POR CONTRATA		9054,95 €
IVA (21%)		1901,54 €
PRESUPUESTO BASE DE LICITACIÓN		10956,49 €

El presupuesto asciende a la cantidad de:

DIEZ MIL NOVECIENTOS CINCUENTA Y SEIS CON CUARENTA Y NUEVE CÉNTIMOS