UCLV
Universidad Central "Marta Abreu" de Las Villas

FIE
Facultad de Ingeniería Eléctrica

Departamento de Automática y Sistemas Computacionales

TRABAJO DE DIPLOMA

Título: Diagnosticador de Fallos fuera de línea aplicado al Quadcopter X4 GARP

Autor: Richard Hernández Ruiz

Tutores: Ing. Ailet Abreu López

Dr. C. Eduardo Izaguirre Castellanos

Dr. C. José Rafael Abreu García

Santa Clara, junio de 2018

PENSAMIENTO

La ciencia, esta echa de errores, pero de errores útiles de cometer, pues poco a poco conducen a la verdad.

Julio Verne

DEDICATORIA

A mis padres,

por la educación que me han dado y los sacrificios de toda una vida en especial estos cinco años.

A mi hermana,

por darme el mínimo de dolores de cabeza en su primer año en la UCLV.

A mi familia,

por todo el apoyo brindado en los momentos más difíciles.

A mi novia,

por su amor, apoyo y cariño todo este tiempo.

AGRADECIMIENTOS

A mi mamá por todos los momentos de preocupación que puedo haberle dado y por su comprensión, cariño y entrega.

A mi padre que gracias a su ejemplo y sacrificio, me he convertido en la persona que soy hoy en día.

A mis abuelos que siempre han estado pendientes y me han ayudado en todo cuanto han podido.

A mi hermana por todo su apoyo.

A mis tías, mi tío y mis primos por toda la ayuda dada en todos estos años, en especial estos últimos tiempos difíciles.

A mi novia porque en el transcurso de dos años se ha convertido en una de las personas más importantes de mi vida.

A mis suegros por toda la preocupación y la ayuda que me han dado todo este tiempo.

A mis antiguos y a mis actuales compañeros de cuarto por ser mi segunda familia y haber compartido tanto en estos años.

A mis compañeros de grupo por todo este tiempo juntos y en especial a los monos, por todos estos años secándose unos a otros. Mi hermano Ale a quien le debo la mitad de mis notas, a Carlitos compañero de viajes, A JJ por no dejarme dormir en sus borracheras y griterías del dota, a Maddiel, Osvany, Ernest y Victor por tan buenos resúmenes para las pruebas, a Abel por disociarme cuando la cabeza no me daba más estudiando al Guille por sus ocurrencias y sus tan efectivas soluciones para cualquier problema con la computadora, A Cambell y el maestro por alegrarme el día con sus grandes ideas. A Ilianni y Annalie por ser tan buenas compañeras en todo este tiempo y en especial a Adrián de la Paz que cuando se recupere completamente podremos reunirnos y compartir todos juntos de nuevo algún días.

A mis profesores durante los cinco años, a mis tutores en especial a Ailet que todavía le debe estar doliendo la vista de tanto revisar esta cosa a la que hoy en día se le puede llamar tesis.

A todas las personas que me han apoyado a lo largo de mi vida y han hecho posible que llegue este momento.

RESUMEN

En la actualidad el Diagnóstico de Fallos se ha convertido en un foco importante de interés debido al aumento gradual de la complejidad de los sistemas automatizados. Los vehículos aéreos no tripulados gracias a los avances tecnológicos en esta rama pueden realizar disímiles tareas como vuelos de reconocimiento y vigilancia, servicio de entrega, estudios de los suelos entre otros; todo esto ligado a la ocurrencia de fallos provoca que se comprometa la misión asignada, se provoquen daños en sensores, actuadores y hasta la pérdida del vehículo, por lo que ha sido necesario emplear sistemas de diagnóstico que realicen una temprana detección de la ocurrencia de estos eventos. En esta investigación se propone un Diagnosticador de Fallos fuera de líneas basado en un modelo de Redes de Petri para el Quadcopter X4 GARP con el objetivo de la detección de los fallos más comunes que se presentan en el mismo. Para el desarrollo del programa se hace un estudio profundo sobre los tipos de vehículos aéreos no tripulados existentes y sus diferentes componentes, fundamentos y metodologías del diagnóstico de fallos y se realiza un análisis de Python con el cual se programa el software diagnosticador debido a las ventajas que tiene sobre los lenguajes de programación clásicos. Por último, se concluye con una serie de experimentos que validan el funcionamiento del programa, en los cuales el mismo realiza un procesamiento de los archivos descargados posteriores al vuelo del vehículo, teniendo como resultado un grupo de registros que describen los fallos ocurridos y muestran la evolución del modelo en Redes de Petri del Quadcopter X4 GARP.

TABLA DE CONTENIDOS

INTRODUCCIÓN

En la actualidad los Vehículos Aéreos no Tripulados (UAV por sus siglas en inglés) han experimentado un elevado crecimiento, debido a la diversidad de aplicaciones en los que estos pueden ser utilizadas, como es el caso de misiones de exploración de terreno, incendios forestales, revisión de cableado eléctrico en zonas intrincadas, seguimiento de objetos, entrega de paquetes, detección de plagas en zonas agrícolas, entre otras. Muchas de estas aplicaciones son de vital importancia y la ocurrencia de complicaciones se convierte en graves pérdidas que van desde el vehículo que realiza la tarea, hasta el cumplimiento de la misión asignada. Por consiguiente, es necesario que este tipo de sistemas (UAVs), posean cierta fiabilidad que asegure su correcto funcionamiento.

La seguridad y fiabilidad de los procesos son la principal preocupación de los ingenieros, por tanto, desde un punto de vista económico el objetivo principal es reducir el costo del mantenimiento, el tiempo de inactividad y las pérdidas causadas por los fallos. Debido a esto el diagnóstico de fallos se ha convertido en un foco importante de interés para los investigadores de todo el mundo e ingenieros de la industria. En este contexto, un fallo se definirse como una variación no permitida de al menos una propiedad característica o variable de un sistema; de forma que éste ya no pueda satisfacer la función para la cual fue creado. La existencia de estos puede estar causada por el efecto de la temperatura, por desgastes debido a fricción mecánica, por desviaciones en los sensores, entre otras causas. El temprano diagnóstico de fallos en este tipo de sistema evita grandes daños al dispositivo, así como influye en el cumplimiento de su misión por lo tanto el diagnosticador de fallos constituye una parte muy importante del sistema del UAV. Las técnicas de diagnóstico de fallos se han aplicado en diferentes campos de la investigación como: inserción a sistemas de ingeniería y control, inteligencia artificial, aplicaciones matemáticas y estadísticas, y en

campos de aplicación como Química, Eléctrica, Mecánica en Ingeniería Aeroespacial (Martínez, 2016).

Para la realización de un Diagnosticador de Fallos es necesario tener un profundo conocimiento del sistema y los posibles fallos, además, hay que desarrollar un modelo que represente el funcionamiento del vehículo. Existen diferentes técnicas para el modelado de procesos tales como máquinas de estado, árboles de decisión y Redes de Petri (RdP) (Martínez, 2016). Las Redes de Petri (RdP) son una herramienta de modelado gráfico y matemático aplicado a muchos sistemas. Esta posee mucha proyección en el campo de la automática, en la cual se puede estudiar y describir información de sistemas de procesamiento que son caracterizados por ser concurrentes, paralelos, asíncronos, distribuidos, no determinísticos. Como herramienta gráfica las RdP son usadas como una ayuda de comunicación visual, similar a las cartas de flujo, diagramas de bloques y redes. En adición las marcas son usadas en estas redes para simular la dinámica y actividades de concurrencia de sistemas. Como herramienta matemática es posible hacer ecuaciones de estado, ecuaciones algebraicas y otros modelos que representan el comportamiento de sistemas (Rodríguez, 2006, Canales, 2005, Martínez, 2016).

Los Vehículos Aéreos no Tripulados (UAV por sus siglas en inglés) poseen un grupo de sensores y componentes propensos a sufrir roturas o desviaciones y esto puede llegar a comprometer la misión asignada. En este campo se han realizado investigaciones relacionadas a los fallos en UAVs por universidades y centros de investigación de todo el mundo como están los casos de (Martínez, 2016, Johry and Kapoor, 2016, Dybsjord, 2013, Ducard, 2007, Alain and Nicolas, 2010), en los cuales se realizan análisis de fallos en UAV y se aplican varias herramientas para el diagnóstico de fallos como es el caso de los Sistemas de Expertos, Neurofuzzy, Redes de Petri, entre otras. En otras investigaciones como (Oncu and Yildiz, 2014, Arrabito et al., 2010), se determinan los factores causales probables en los accidentes de UAV basados en el análisis del factor humano.

En Cuba, el estudio de fallos en los UAVs es un tema en el cual se han realizado investigaciones, entre las que se encuentran; (Aréchaga, 2017) donde se hace una caracterización de fallos en el Vehículo Autónomo Aéreo Quadcopter X4 GARP, existe también (Cortés, 2017) el cual realiza un modelado de un Quadcopter utilizando las Redes

de Petri, el Grupo de Automatización, Robótica y Percepción (GARP) de la Universidad Central “Marta Abreu” de Las Villas ha estado trabajado en los últimos años en el diseño de estrategias de control de vuelo en Vehículos Aéreos No Tripulados (Ávila, 2008) (Martínez, 2009) (Perez, 2014) entre otras. La empresa GEOCUBA ha empleado UAVs en los levantamientos aerofotográficos mediante técnicas de Control Fotográfico lo que agiliza en gran medida el proceso de levantamiento topográfico (Eloy Pérez García, 2016) (Ricaño, 2016).

A pesar de la actualidad en estas investigaciones no se han encontrado referencias sobre el desarrollo de un Diagnosticador de Fallos fuera de línea aplicado al Quadcopter X4 GARP.

Por tal motivo el **objeto de estudio** de esta investigación se centra en el empleo de las Redes de Petri como herramientas para la construcción de un Diagnosticador de Fallos.

Mientras que el **campo de aplicación** engloba el empleo de las Redes de Petri como herramientas para la construcción de un Diagnosticador de Fallos en los sistemas antes mencionados de vehículos UAV.

En estos momentos el Vehículo Autónomo Aéreo Quadcopter X4 GARP no presenta un Diagnosticador de Fallos que permita la detección y diagnóstico de desviaciones de las diferentes variables o roturas ya sea en el sistema de comunicación con la estación en tierra, motores o en su Sistema de Posicionamiento Global y Unidad de medida Inercial (GPS+IMO por sus siglas en inglés) por lo que se concibe el siguiente **problema científico**:

No existencia de un diagnosticador de fallos fuera de línea capaz de detectar los fallos que puedan afectar al Quadcopter X4 GARP.

Con el desarrollo de un Diagnosticador de Fallos dirigido al Quadcopter X4 GARP basado en el modelo en RdP de (Cortés, 2017), se determinan los posibles fallos con la adquisición y procesamiento de las variables en tierra y en un futuro implementado en línea (*online*) sobre el vehículo, alertando de forma parcial cualquier tipo de accidente que pueda afectar o dejar en desuso el vehículo.

Con el fin de solucionar el problema existente se plantea el siguiente **objetivo general**:

Desarrollar un software Diagnosticador de Fallos fuera de línea haciendo uso del modelo obtenido con Redes de Petri para el Quadcopter X4 GARP.

Para satisfacer este objetivo general se plantean a seguir los siguientes **objetivos específicos**:

- Analizar los fundamentos teóricos conceptuales relacionados con la construcción del Diagnosticador de Fallos fuera de línea empleando modelo basado en Redes de Petri (RdP).
- Estudiar a fondo la operación del Quadcopter X4 GARP haciendo hincapié en la identificación de los parámetros de funcionamiento y principales fallos, así como el modelo en RdP existente para el UAV.
- Diseñar del Diagnosticador de Fallos fuera de línea empleando el modelo basado en RdP aplicado al Quadcopter X4 GARP.
- Realizar la simulación del Diagnosticador de Fallos fuera de línea.

Tareas de investigación

- Revisión de la literatura especializada dentro del objeto de estudio.
- Estudio de la operación del UAV Quadcopter GARP X4. Identificación de los parámetros de funcionamiento. Principales fallos. Fallos en los actuadores y sensores.
- Análisis de los fundamentos teóricos conceptuales relacionados con el modelado de fallos en sistemas basados en RdP.
- Estudio del lenguaje de programación Python y las bibliotecas correspondientes para el desarrollo del software de diagnóstico basado en modelo de RdP.
- Realización de una propuesta de un software Diagnosticador de Fallos en RdP aplicado al Quadcopter X4 GARP.
- Validación de la simulación del Diagnosticador de Fallos fuera de línea.

Organización del informe

El informe de investigación está compuesto por la Introducción, tres Capítulos, Conclusiones, Bibliografía y Anexos.

En la Introducción del trabajo se evidencia la importancia del tema abordado, los avances más recientes en la tecnología en este campo y la necesidad de su uso. Además, se especifica el objeto y el campo de investigación y los objetivos del trabajo.

Capítulo 1: Se realiza la introducción al tema mediante un estudio en profundidad del Diagnóstico de Fallos y los fallos más comunes y del uso de las RdP aplicado a los UAV, quedando expuestas las características, ventajas, desventajas, métodos de implementación.

Capítulo 2: Se exponen las distintas formas de realizar un Diagnosticador de Fallos aplicado a los UAV, además se explican los conceptos principales para la utilización de la herramienta RdP en el diagnóstico de fallos y el uso de un lenguaje de programación para la realización de una aplicación de diagnóstico para la investigación. En este capítulo se realiza el desarrollo del Diagnosticador de Fallos y se mostrará un análisis de los principales fallos a modelar para la realización del diagnosticador.

Capítulo 3: En este se realizan las pruebas, validaciones y análisis de los resultados obtenidos a partir del diagnosticador desarrollado en el capítulo anterior. Además, se hace un análisis económico y medioambiental.

Conclusiones

Recomendaciones

Referencias bibliográficas

Anexos.

CAPÍTULO 1. FUNDAMENTOS DEL DIAGNÓSTICO FALLOS

En este Capítulo 1 se exponen los conceptos, principios, características y metodología para la construcción del Diagnosticador de Fallos utilizando las Redes de Petri, así como los fundamentos teóricos y matemáticos en los que se sustenta esta última herramienta. Se presentan además diferentes tipos de UAVs centrándose en sus principales características y fallos.

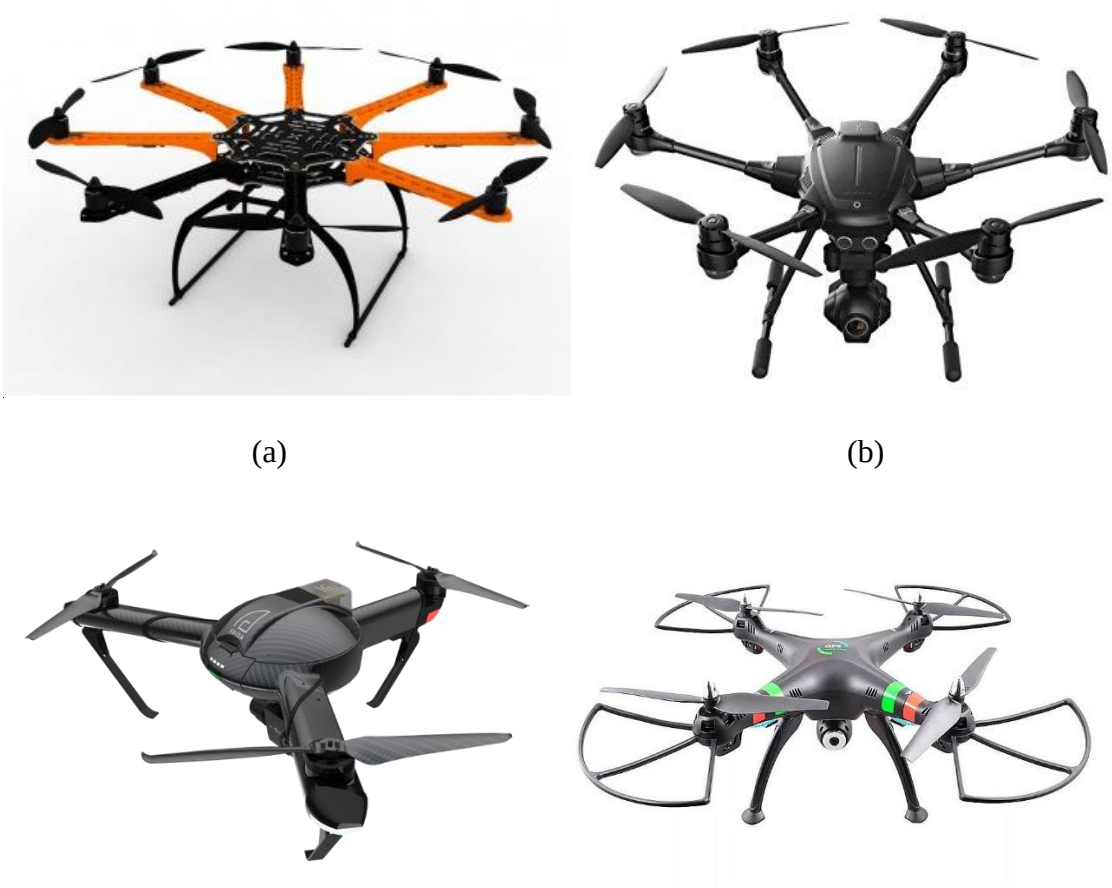
1.1. Vehículos Aéreos no Tripulados

Un vehículo aéreo no tripulado (UAV), es una aeronave que tiene la posibilidad de realizar vuelos sin humanos a bordo. Este es parte de un sistema de aeronave no tripulada que incluye módulos de comunicación que permite el control remoto y el intercambio de video y otros datos con una estación de control en tierra, casi siempre formado por una persona o una computadora con un dispositivo de radio control.

1.1.1 Tipos de UAVs

En la actualidad existen diversas arquitecturas de vehículos aéreos no tripulados debido a que no existe un estándar aceptado (Aréchaga, 2017) teniendo en cuenta diferentes parámetros y criterios. Estos se pueden clasificar en dos grupos, Vehículos de ala fija y vehículos multirotores.

Los primeros son más eficientes a la hora de realizar vuelos más largos debido a la gran autonomía y logran mayor velocidad, son silenciosos permitiendo misiones de vigilancia y por último soportan mejor las condiciones meteorológicas ya sea viento lluvia y temperatura.



(a) (b)

(c) (d)

Figura 1.1. Octocóptero (a), Hexacóptero (b), Tricótero (c), Cuadricóptero (d)

Por otro parte, los multirotores son vehículos de fácil manejo permitiendo realizar despegues en área reducidas debido a que estos se mantienen en el aire gracias a la fuerza de sustentación producida por las hélices, lo que le permite también realizar vuelo estacionario y según la cantidad de motores y la potencia de estos le permite trabajar con cargas más pesadas que los de ala fija. Según la cantidad de motores y hélices se agrupan en 4 tipos fundamentales tricóteros, cuadricópteros, hexacópteros, octocóptero Figura 1.1.

1.2. Diagnóstico de Fallos

En la actualidad la fiabilidad de los sistemas son la principal preocupación de los ingenieros y grupos de investigación debido al daño que causa la ocurrencia de un fallo o avería desde el punto económico, parada del sistema, mantenimientos e incluso peligro a los operadores, por lo cual el diagnóstico de fallos se convierte en un proceso primordial en cualquier

sistema. El funcionamiento confiable y seguro de los sistemas técnicos es de gran importancia para la protección de las vidas humanas, la salud, el ambiente y las inversiones económicas. La temprana detección de fallos es crítica para evitar una degradación del comportamiento, daño de la maquinaria o vidas humanas. Un diagnóstico de fallos preciso ayuda a tomar decisiones correctas en acciones de emergencia y reparación.

1.2.1 Conceptualización del Diagnóstico de Fallos

El Diagnóstico de Fallos (FD por sus siglas en inglés), se basa en inteligencia computacional, análisis estadístico, y conocimiento experto, además debe ser capaz de recopilar, combinar, analizar, y comparar datos históricos de toda la planta para detectar puntualmente avisos sobre deterioros en sensores, componentes y procesos.

En este campo existen dos términos de suma importancia, ellos son; los fallos y las averías, los cuales son usados para describir diferentes grados de desgaste de sistemas que puede disminuir las facultades del mismo.

Un fallo es una desviación no permitida de al menos una propiedad característica del sistema de la condición aceptable, usual y estándar (Johry and Kapoor, 2016).

En resumen, un fallo es un funcionamiento no deseado del sistema, que puede no afectar el comportamiento total del mismo, aunque es posible que conduzca a una avería. La ocurrencia de estos en ocasiones no es visible debido a que se manifiesta como menor que el valor detectable o puede estar oculto y por lo tanto ser difícil de detectar y corregir. Por otro lado una avería es una interrupción permanente de la capacidad de un sistema para realizar una función requerida bajo condiciones de funcionamiento específicas (Johry and Kapoor, 2016).

Debido a la ocurrencia de fallos continuos pueden presentarse averías, este es un suceso que impide el funcionamiento total o parcial del sistema donde ha ocurrido, por lo tanto, la avería es una condición más severa que el fallo. Cuando ocurre un fallo en un dispositivo, por ejemplo; en un actuador, continúa funcionando, aunque existe la posibilidad de que presente una respuesta más lenta o se vuelva menos eficaz, sin embargo, cuando ocurre una avería, este deja de producir el efecto deseado.

En relación con el control general, la avería es interpretada como un fallo (Dybsjord, 2013). En el caso de un UAV de ala fija una avería se puede producir en uno de los alerones dejando

esta superficie de control inutilizable, y a pesar de esto, mediante el uso del resto de las superficies de control de manera diferente la aeronave es todavía controlable (Aréchaga, 2017), por lo tanto, se concluye que, en algunos sistemas ciertas averías no impiden el funcionamiento total.

Los fallos y las averías clasifican también en términos de tiempo. Los fallos y averías abruptas presentan cambios rápidos e inesperados en el sistema, por ejemplo, un actuador atascado. Los fallos incipientes conducen a respuestas lentas de sensores y actuadores, son más sutiles y por lo tanto son menos obvios pero la presencia de estos por largos períodos de tiempo puede traer consigo averías y tener resultados catastróficos (Ducard, 2007).

1.2.2 Clases de Fallos en un sistema

En general en este tipo de procesos se presentan tres clases de fallos o averías (Martínez, 2016).

Cambios bruscos en los parámetros del modelo

En la actualidad existen sistemas los cuales no pueden se modelados exactamente debido a complejas interacciones entre sus variables. A la hora del modelado, para simplificar el procesamiento no se tiene en cuenta parte de la dinámica del sistema, por lo tanto, estos procesos los cuales no son modelados, son agrupados como parámetros e incluyen interacciones de los límites del sistema. Todo lo anterior provoca que existan variaciones en los coeficientes, incrementándose cuando hay entradas perturbadoras en el proceso desde el ambiente a través de una o más variables independientes.

Cambios estructurales

Los cambios estructurales se refieren a variaciones en el propio proceso, ellos se presentan por cuenta de fallos en el equipamiento. Para resolver este tipo de avería en un sistema de diagnóstico habría que hacer un levantamiento y reestructuración de las ecuaciones del modelo en función de remodelar la situación actual del proceso (Martínez, 2016).

Fallos en sensores y actuadores

Los sensores y actuadores son propensos a ser víctimas de errores graves debido a fallos continuos en el tiempo y desviaciones de los valores. Estos instrumentos otorgan importantes datos de variables o realizan acciones sobre el sistema, por lo que un fallo en alguno de ellos

siempre va a tener efectos negativos como; variaciones de los valores permitidos de las variables de estado de la planta, a menos, que sean detectados a tiempo estos fallos y se realicen acciones de corrección en un tiempo determinado. Este sería el propósito del diagnóstico, detectar cualquier fallo en los instrumentos que pudiera afectar el funcionamiento del sistema de control (Aréchaga, 2017, Martínez, 2016).

1.2.3 Fallos en UAVs

Los fallos y averías pueden afectar a sensores, actuadores o al sistema de cómputo que administra las acciones de vuelo. Estos son los fallos más comunes que se encuentran en este tipo de sistemas.

Fallos en sensores

Estos afectan grandemente el funcionamiento general del sistema ya que según el nivel de afectación serán las alteraciones en las mediciones requeridas por los controladores y por consiguiente existirán errores en el sistema en lazo cerrado. En (Aréchaga, 2017) se referencian algunos de los fallos en sensores como, desviación, pérdida de exactitud o error de calibración, deriva, congelación.

Fallos en actuadores

Estos dispositivos son un componente esencial en cualquier sistema de control y han sido utilizados en todo tipo de aplicaciones industriales. Su trabajo es convertir la energía externa en movimiento. Un fallo causa desde un consumo elevado de energía, baja eficiencia de lazo de control y hasta pérdida parcial o total del control. Este tipo de fallos en UAVs de ala fija provoca una reducción en el rendimiento de la aeronave y un aumento de la inestabilidad pudiendo provocar un accidente que traiga consigo la pérdida del vehículo. En el caso de los multirotores si existe una afectación en los motores esto produce inestabilidad debido al empuje insuficiente o por otra parte, si se produce la pérdida de los motores ante la imposibilidad de mantenerse en vuelo esta caerá al suelo(Aréchaga, 2017).

Fallos de comunicación

Los UAV presentan dispositivos receptores/transmisores que son los encargados de transmisión y recibo de paquetes de datos con la estación en tierra, sin embargo este medio no es totalmente confiable debido a las diferentes interferencias ambientales que afectan las

señales y bloquean sus trayectorias introduciendo ecos, ruidos e interferencias (Johry and Kapoor, 2016).

Fallos en baterías

Un fallo en la batería ya sea por baja carga o por perforación provoca problemas en el hardware además, las altas temperaturas producen daños en esta posibilitando su ruptura y por consiguiente representando un paro del sistema, haciendo que caiga la aeronave. Existen muchas investigaciones donde se trata este aspecto en los UAV (Olondo et al., 2015, Hobbs and Herwitz, 2006).

Averías estructurales

Son muchas las fuentes posibles que provocan daños estructurales, estos posibilitan que cambien los coeficientes dinámicos y el centro de gravedad del vehículo constituyendo un cambio en la dinámica del sistema y afectando la mayoría de los lazos de control existentes en el UAV.

Averías electrónicas

Los componentes electrónicos más comunes de estos vehículos son: el controlador de vuelo, radio telemetría, sensores, batería, GPS, entre otros. Esta electrónica es bastante sofisticada y, en consecuencia, hay varias oportunidades para un fallo electrónico, en ellos ocurren averías producto de descargas electroestáticas, sobretensión eléctrica, altas temperaturas, vibraciones que causan daños en los conectores y las soldaduras (Caswell and Dodd, 2014). Debido a la presencia de estas averías, se deben conocer los rangos ambientales que los UAV pueden soportar (temperaturas, presiones, vibraciones en el vuelo, choque en el aterrizaje, turbulencia) (Whitlock, 2014) sin incurrir en una avería eléctrica, para evitar accidentes.

Factor Humano

El manejo de estos vehículos requiere una vasta experiencia. La ausencia del piloto del vehículo minimiza la necesidad humana y maximiza la tasa de éxito para las misiones (Aréchaga, 2017). A pesar de esto los vehículos pueden desviarse de su trayectoria o colisionar debido al error del piloto o a interferencias en el UAV por lo que se tiene que tener mucha atención a comportamientos inesperados (Johry and Kapoor, 2016).

Efectos ambientales

El mal tiempo y otros efectos ambientales posibilitan la degradación del rendimiento de los diferentes sensores y actuadores del UAV. Los fenómenos naturales (lluvia, fuertes vientos, calor, humedad, sequías, incendios, tormentas de arena, tormentas de nieve), estructuras artificiales, animales, sustancias tóxicas incluso otros vehículos aéreos influyen en la pérdida de estos dispositivos.

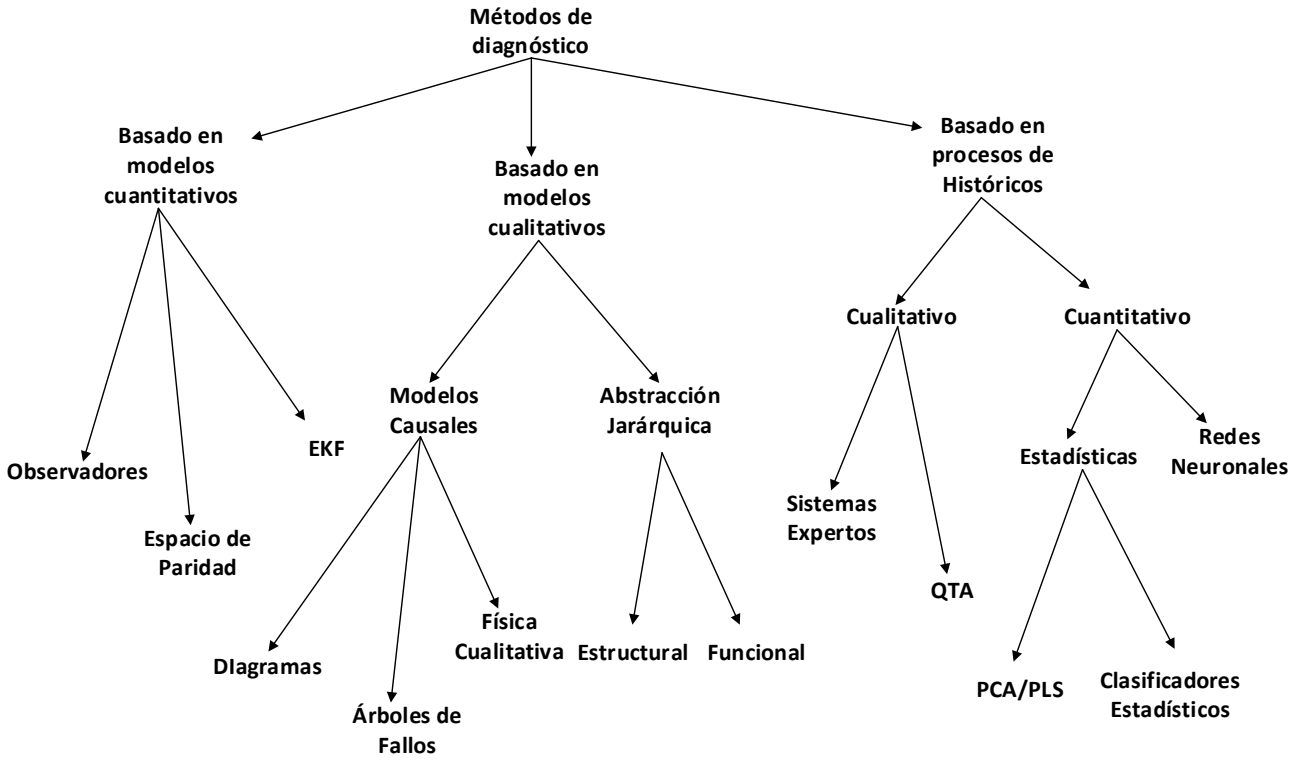


Figura 1.2. Algoritmos de Diagnóstico

1.3. Tipos de Diagnóstico de Fallos

Debido al amplio alcance de los problemas del diagnóstico de fallos y las dificultades de estos en soluciones en tiempo real, varias investigaciones basadas en ordenador han sido desarrolladas en los últimos años. Ellas cubren una amplia variedad de técnicas tales como los esfuerzos tempranos usando árboles de fallos (Vizcaya et al., 2011) y grafos (Alfie et al., 2010), investigaciones analíticas, sistemas basados en conocimiento (sistemas expertos), máquinas de estados finitos (El-Fakih et al., 2003), redes neuronales y RdP entre otros.

Entre los métodos ya mencionados se encuentran las Redes de Petri las cuales son muy utilizadas en el modelado de disímiles procesos. Esta investigación se basa en el modelo propuesto en (Cortés, 2017) donde se refleja el funcionamiento normal y con fallos del Quadcopter X4 GARP el cual servirá como base para la construcción del Diagnosticador de Fallos propuesto en esta investigación.

1.3.1 Redes de Petri

Las RdP fueron introducidas en 1962 por Carl Adam Petri, en su trabajo de tesis doctoral titulado "*Kommunikation mit Automaten*" (traducido al español: Comunicación con máquinas expendedoras), presentado en la Facultad de Matemáticas de la Universidad de Bonn, en Alemania (Brauer and Reisig, 2009). Las RdP son una herramienta de modelación gráfica y matemática que se aplica en el estudio de muchos sistemas. Con estas se puede modelar y estudiar sistemas concurrentes, asíncronos, distribuidos, paralelos y no deterministas. Como una herramienta gráfica, las RdP se utilizan como un apoyo de comunicación visual similar a diagramas de flujo, diagramas de bloques y redes. Además, las marcas son utilizadas en estas redes para simular el comportamiento dinámico y concurrente de los sistemas.

Como una herramienta matemática, es posible establecer ecuaciones de estado, ecuaciones algebraicas y algunos otros modelos matemáticos que denoten el comportamiento del sistema modelado (Murata, 1989).

Conceptos Redes de Petri aplicadas al diagnóstico de fallos

Una RdP tiene dos tipos de nodos (lugares y transiciones). Un lugar (P) representado por un círculo y una transición (T) por una barra. Los lugares y transiciones son conectados por arcos. El número de lugares y transiciones son finitos y diferentes de cero. Un arco es la conexión que existe de lugares de salida a transiciones y de transiciones a lugares de entrada. Es decir, en una RdP los lugares y las transiciones alternan en un camino hecho de arcos consecutivos. Por lo cual es obligatorio que cada arco tenga un nodo en cada una de sus terminaciones, haciendo referencia al hecho de identificar transiciones de entrada y salida, asociadas a un lugar de entrada o salida en particular de la red (Martínez, 2016, Murata, 1989).

Marcado. Cada lugar contiene un entero (positivo o cero) de marcas. El número de marcas contenidas en un lugar P_i es llamado M (P_i) o M_i. La red marcada, M, es definida por el vector de ese marcado, es decir, $M = (M_1, M_2, \dots, M_n)$.

El marcado en un cierto momento define el estado de la RdP, o más precisamente el estado del sistema descrito por la RdP. La evolución del estado por lo tanto corresponde a la evolución del marcado, causada por el disparo de transiciones. Una transición puede solo ser disparada si cada uno de los lugares de entrada de esta transición contiene al menos una marca. Una RdP se dice que es segura para un marcado inicial M_0, si para todos los marcados alcanzables, cada lugar contiene cero o una marca. Una transición T_j se considera viva por un marcado inicial M_0, si para cada marcado alcanzable $M_i \in \mu(M_0)$, existe una secuencia de disparo s desde M_i que contiene a T_j. Una RdP es viva para el marcado inicial M_0, si todas sus transiciones son vivas para M_0. (Murata, 1989, Martínez, 2016).

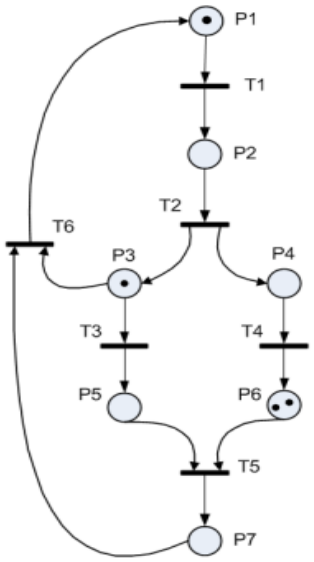


Figura 1.3. Esquema de una RdP

Álgebra lineal en Redes de Petri

Una RdP Ordinaria no marcada o una estructura de RdP, es un grafo bipartito representado por la cuádrupla $Q = < P, T, P_{re}, P_{ost} >$ tal que: $P = \{P_1, P_2, ..., P_n\}$ es el conjunto de lugares finito y no vacío; $T = \{T_1, T_2, ..., T_M\}$ es el conjunto de transiciones finito y no vacío; $P \cap T = \emptyset$, es decir que los conjuntos P y T son disjuntos; $P_{re} : P \times T \rightarrow \{0, 1\}$ es la aplicación de la incidencia de entrada; $P_{ost} : T \times P \rightarrow \{0, 1\}$ es la aplicación de la

incidencia de salida. P_{re} (P_i, T_j) es el peso del arco $P_i \rightarrow T_j$. El peso es 1 si el arco existe y 0 si no. P_{ost} (P_i, T_j) es el peso del arco $T_j \rightarrow P_i$. P_{re} y P_{ost} así relaciona a la transición T_j del par (P_i, T_j) (Martínez, 2016).

Una RdP no marcada generalizada es definida como una RdP ordinaria no marcada, excepto P_{re}: $P \times T \rightarrow \mathbb{N}$, P_{ost}: $P \times T \rightarrow \mathbb{N}$.

Las siguientes notaciones serán usadas:

- ${}^{o}T_j = \{P_i \in P \mid P_{re}\ (P_i, T_j) > 0\}$ =el conjunto de lugares de entrada de T_j;
- $T = \{P_i \in P \mid P_{ost}\ (P_i, T_j) > 0\}$ =el conjunto de lugares de salida de T_j;
- ${}^{o}P_i = \{T_j \in T \mid P_{re}\ (P_i, T_j) > 0\}$ = el conjunto de transiciones de entrada de P_i;
- $P_0 = \{T_j \in T \mid P_{ost}\ (P_i, T_j) > 0\}$ = el conjunto de transiciones de salida de P_i.

Una RdP marcada es un par $R = < Q, M_0 >$ en el cual Q es una RdP no marcada y M_0 es el marcado inicial (Cortés, 2017, Martínez, 2016). Las condiciones de validación pueden ser expresadas como sigue: una transición T_j es habilitada para un marcado M_k si $M_k\ (P_i) \geq P_{re}$ (P_i, T_j) para cada $P_i \in {}^{o}T_j$.

La matriz de incidencia de entrada es $W_- = [w_{ij-}]$ donde $w_{ij-} = P_{re}\ (P_i, T_j)$; la matriz de incidencia de salida es $W_+ = [w_{ij+}]$] donde $w_{ij+} = P_{ost}\ (P_i, T_j)$; entonces la matriz de incidencia es: $W = W_+ - W_- = [w_{ij}]$ (Cortés, 2017, Martínez, 2016, Murata, 1989).

La ecuación fundamental es tratada en (Cortés, 2017, Martínez, 2016, Murata, 1989, Ruiz-Beltrán et al., 2005), describe la función de marcado de los diferentes lugares de la RdP. Sea S una secuencia de transiciones, la cual se realiza desde el marcado M_i, la cual puede ser escrita como $M_i \rightarrow S$. El vector característico de secuencia S, escrito como s, es el vector de M componentes cuyo componente número j corresponde al número de disparos de la transición T_j en la secuencia S. Si el disparo de la secuencia S es tal que $M_i \rightarrow M^s k$, entonces la ecuación fundamental es obtenida (Martínez, 2016, Murata, 1989):

$$M_k = M_i + W \cdot s \tag{1.1}$$

1.3.2 Tipos de Redes de Petri

Redes de Petri Autónomas y no-autónomas

Cuando una RdP describe el funcionamiento de un sistema evolucionando de una manera autónoma, es decir, cuyos instantes de disparo están entre no conocidos o no indicados, se puede decir que esto es una RdP autónoma. Una RdP No-Autónoma describe el funcionamiento de un sistema cuya evolución es condicionada por eventos externos y/o tiempo. Una RdP No-Autónoma es sincronizada y/o temporizada (Martínez, 2016, Cortés, 2017).

Las RdP No-autónomas

Las RdP No-autónomas habilitarán el sistema a ser modelado cuyos disparos están sincronizados con eventos externos y/o cuyas evoluciones son dependientes del tiempo. En una RdP sincronizada, un evento es asociado con cada transición, y el disparo de esta transición ocurrirá si la transición es habilitada, cuando los eventos asociados ocurren. Los eventos externos corresponden a un cambio en el estado de la palabra externa (incluyendo el tiempo), por el contrario, una variación en el estado interno o en el marcado, podría ser llamado un evento interno. En (Cortés, 2017) se referencia que una RdP interpretada exhibe características tales como sincronización, lugares temporizados (*P-timed*) y tiene una parte para procesamiento de datos además que las entradas están asociadas con las transiciones y las salidas están asociadas con los lugares como se muestra en la Figura 1.4. Las RdP temporizadas son útiles para evaluar comportamiento y habilitar un sistema a ser descrito cuando su funcionamiento es dependiente del tiempo. Por ejemplo, el tiempo que puede durar entre el inicio y el final de una operación.

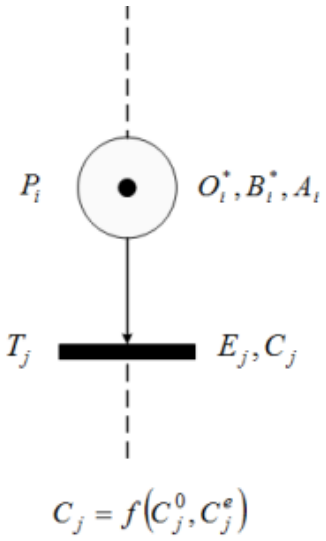


Figura 1.4. Red de Petri Interpretada de Control

En la tesis de doctorado (Martínez, 2016) queda conceptualizado que una RdP interpretada de control presenta las siguientes características:

- Es sincronizada a eventos externos y estables.
- Es segura.
- Es determinística.
- Tiene una parte de procesamiento de datos cuyo estado es definido por un conjunto de variables $V = \{V_1, V_2, ..., V_n\}$. Este estado es modificado por operaciones O_i* las cuales son asociadas a los lugares. Este determina el valor de los predicados C_j0.
- Recibe información booleana C_e del ambiente. Envía acciones de nivel A_j y acciones impulso B_j, asociadas con los lugares, al ambiente (Murata, 1989).

Redes de Petri Autónomas

La Red de Petri continua es un modelo en el cual el número de marcas en el lugar es representado por números reales en vez de números enteros.

Una RdP continua autónoma es una quíntupla $R = < P, T, P_{re}, P_{ost}, M_0 >$ tal que (Martínez, 2016):

- $P = \{P_1, P_2, . . ., P_n\}$ es el conjunto de lugares finito y no vacío;

- $T = \{T_1, T_2, ..., T_M\}$ es el conjunto de transiciones finito y no vacío;
- $P \cap T = \emptyset$, es decir que los conjuntos P y T son disjuntos;
- $P_{re}: P \times T \rightarrow Q_+$ es la aplicación de la incidencia de entrada;
- $P_{ost}: T \times P \rightarrow Q_-$ es la aplicación de la incidencia de salida;
- $M_0: P \rightarrow R_+$ es el marcado inicial.

Una RdP híbrida está compuesta por lugares y transiciones discretas y continuas. Es una RdP autónoma marcada que a diferencia de la red continua vista anteriormente $R = < P, T, P_{re}, P_{ost}, M_0, h >$ se le agrega el vector h que representa la función híbrida $h: P \cup T \rightarrow \{D, C\}$, que indica si los nodos son discretos $P^D \; y \; T^D$ o continuos $P^C \; y \; T^C$.

Una transición discreta en una RdP híbrida es habilitada si cada lugar P_i en ^oT_j encuentra la condición $M(P_i) \geq P_{re}(P_i, T_j)$. Una transición continua en una RdP híbrida es habilitada si cada lugar P_i, en ^oT_j encuentra la condición $m(P_i) \geq P_{re}(P_i, T_j)$, si P_i es un lugar discreto (D), o $m(P_i) > 0$, si P_i es un lugar continuo (C) (Martínez, 2016).

Figura 1.5. Elementos de una RdP híbrida

La Matriz de Incidencia de una RdP Híbrida puede ser escrita como:

$$W = \begin{bmatrix} W^D & o \\ W^{CD} & W^C \end{bmatrix} \tag{1.2}$$

Donde W^D corresponde a los arcos entre nodos discretos, W^C a los arcos entre nodos continuos, y W^{CD} a los arcos entre los lugares C y transiciones D. Los arcos entre los lugares D y transiciones C corresponden a la sub-matriz 0. Para este tipo de RdP existe una condición que aparece en (Martínez, 2016), la cual establece que un arco debe unir a una transición C a un lugar D tan pronto como un arco recíproco exista. Esto asegura que el marcado de un lugar D, es un entero para cualquiera evolución que ocurra. Sea S una secuencia de disparo y s sea el vector característico de S. La dimensión del vector s es igual al número M de

transiciones. El j-ésimo componente de M representa el número de disparos de transiciones T_j y será denotado por n_j. Si T_j es una transición D, entonces n_j es un entero y si T_j es una transición C, entonces n_j es un número real (Murata, 1989). Un marcado M puede ser deducido de un marcado M_0 debido a una secuencia S, usando la relación fundamental:

$$M = M_0 + W \times S \tag{1.3}$$

La relación fundamental de una RdP Híbrida es idéntica con la relación fundamental de una RdP Discreta. Entonces, se deduce que cada propiedad de RdP Discretas resultante de esta relación puede ser transpuesta a RdP Híbridas (Cortés, 2017, Martínez, 2016).

1.3.3 Metodología para la construcción de un diagnosticador de fallos

En el sistema en cuestión se necesita hacer un análisis de las interacciones de los procesos, los cuales tienen diferentes características. En estos se generan varias señales mezcladas que dependen de variables como el tiempo y eventos externos e internos de una manera asincrónica.

La complejidad de este sistema permite que se tome la decisión de dividir el modelo de RdP en subsistemas para un mejor análisis de los diferentes fallos trabajando en conjunto y compartiendo diferentes funcionalidades(Cortés, 2017).

Debido a lo anterior en (Trigos et al., Cortés, 2017, Martínez, 2016) se proponen los siguientes pasos para la realización del modelo:

Primero se divide el modelo en sub-sistemas, después se construye el Modelo de RdP de los componentes de cada subsistema teniendo en cuenta el funcionamiento normal y de fallo. Como tercer paso se realiza la integración de los diferentes subsistemas, donde además de integrar el comportamiento normal del sistema en general, se precisan los diferentes fallos, así como los lugares y marcados iniciales de cada subsistema definiendo el comportamiento normal y de fallos de cada uno de estos lugares y transiciones. Y por último se realiza un refinamiento del modelo general (Cortés, 2017) donde se realiza la identificación de sensores y la tabla de integración estos, que es la productoria de las salidas de las lecturas sensoriales, comparadas con las lecturas esperadas del lugar. Cuando el subsistema está en cualquier lugar de funcionamiento normal, los sensores pueden entregar medidas diferentes a las esperadas,

indicando la presencia de fallo. La tabla de integración de sensores se construye individual para cada subsistema.

1.4. Conclusiones Parciales del Capítulo 1

Luego de revisar la literatura especializada en diagnóstico de fallos y Redes de Petri se arriban a las siguientes conclusiones.

Las Redes de Petri constituyen una herramienta eficaz para la representación de sistemas de eventos discretos, continuos e híbridos, gracias a su potencialidad en el modelado y poderoso basamento matemático que poseen.

La construcción del diagnosticador de fallos para el Quadcopter X4 GRAP con RdP se debe efectuar a partir del conocimiento profundo sobre el funcionamiento del sistema a modelar, sobre todo en lo relacionado a los parámetros de funcionamiento y sus especificaciones funcionales asociadas.

Los fallos en los UAVs llegan a alterar las mediciones requeridas por el controlador, pudiendo verse comprometido el comportamiento del sistema en lazo cerrado, además el desgaste operacional de actuadores es un aspecto de vital importancia que puede traer consigo inestabilidad en el vehículo o la pérdida del control.

CAPÍTULO 2. PROGRAMACIÓN DEL DIAGNOSTICADOR DE FALLOS

En el Capítulo 2 se expondrán las principales características del lenguaje de programación Python utilizado para el desarrollo del Diagnosticador de Fallos, resaltando las principales funciones y algoritmos implementados, la metodología para el desarrollo de la programación del modelo en RdP del Quadcopter, sus características principales, así como el sistema de detección de fallos y el procesamiento de las variables obtenidas de los archivos de vuelo que realiza el software.

2.1 Python como lenguaje de programación

Python es un lenguaje de programación extremadamente fácil y entretenido que en la última década se ha hecho muy popular. Desarrollado por Guido Van Rossum inicialmente para la enseñanza y posteriormente para resolver problemas reales, presentando una amplia variedad de características de lenguaje de programación como C++ y Java. La diferencia más notable que hace de este lenguaje tan popular es su productividad ligada a lo divertido de su uso. Python es clasificado como un lenguaje interpretado, de alto nivel, multiplataforma que a diferencia de otros lenguajes de programación posee una serie de reglas de estilos, a fin de poder escribir un código fuente más claro y de manera estandarizada. En un lenguaje de código abierto que permite realizar revisiones a los diferentes módulos y cambiarlos a conveniencia según la aplicación que se desee desarrollar. (Allen Downey, 2002)

Como en todos los lenguajes de programación Python posee variables que no son más que espacio para almacenar datos modificables en la memoria de un ordenador. Entre las variables que más se utilizan en el software Diagnosticador de Fallos se encuentran las constantes, que pueden ser de diferentes tipos como son cadenas de texto (*string*), números

ya sean enteros, reales, hexadecimal, booleanos (*true or false*); y otros datos más complejos como tuplas, listas y diccionarios. Una de las ventajas que tiene Python sobre otros lenguajes de programación es que no es necesario inicializar las variables para un tipo específico, este lenguaje automáticamente detecta el tipo de la variable y le asigna el espacio de memoria determinando.

Las estructuras de control de flujo utilizadas en este lenguaje ya sean iterativas o condicionales (*for*, *while, if* ,etc), funcionan de manera similar a los demás lenguajes de programación con la diferencia que en este se utiliza la identación para el control de que código es procesado en estas estructuras. La identación es lo mismo que la sangría en el lenguaje humano, en Python esta regla es obligatoria otorgándole mayor organización al código y por consiguiente mayor legibilidad. Otra estructura muy importante en todos los lenguajes de programación y que revolucionó en su momento toda la perspectiva acerca del como programar, son la funciones. Esta es la forma de agrupar algoritmos que realizan determinadas acciones, pero solo se ejecutan cuando son llamadas sin la necesidad de volver a escribir el código nuevamente.

Como todos los lenguajes de alto nivel de hoy en día la programación orientada a objetos se ha convertido en una necesidad. Como en la vida diaria un objeto en una entidad que tiene una serie de atributos que lo diferencian de los demás objetos, pueden realizar diferentes acciones según su diseño y estos pueden estar conformados por otros y existe la posibilidad que objetos diferentes realicen las mismas funciones y más. En la programación se denomina de una forma particular pero la explicación es similar.

Entre los elementos de la programación orientada a objetos encontramos las clases. Estas son los modelos sobre los cuales se construyen los objetos. En Python, una clase se define con la instrucción *class* seguida de un nombre genérico para el objeto y (:).

Estas variables que se incluyen dentro de la clase representan características intrínsecas del objeto, se denominan en términos de programación “propiedades”. Se puede decir que las clases son el razonamiento abstracto de un objeto, mientras que el objeto es su materialización. A la acción de crear objetos se le llama instanciar una clase y dicha instancia, consiste en asignar la clase, como valor a una

Para acceder a cualquier de las diferentes estructuras dentro del objeto en este lenguaje de programación es necesario poner el nombre del objeto creado, un punto y el nombre de la estructura a la que se quiere acceder. A igual que con cualquiera variable esta es la forma de modificarlas o usar los métodos que presente el objeto creado. Otra propiedad que tienen los objetos es la herencia que no es más que hacer que un objeto comparta las mismas características de uno ya creado, teniendo la posibilidad de no tener que volver a crear el objeto desde cero y agregarle también nuevas propiedades que no tenías el anterior. Cuando una clase no hereda de ninguna otra, debe hacerse heredar de *object*, que es la clase principal de Python, que define un objeto; de esta forma se puede evitar algunos errores en compilación.

Otras de las posibilidades que brinda este lenguaje de programación es el trabajo con archivos, en específico de texto, siendo esto útil para crear los registros necesarios para la recopilación de la información de los fallos y los estados de la RdP. En Python el objeto **file** es el que nos permite el trabajo con este tipo de archivos entre otros y posee diferentes métodos para su manejo. Para asignar a una variable un valor de tipo **file**, solo es necesario recurrir a la función integrada **open()**, la cual está destinada a la apertura de un archivo: Esta función recibe dos parámetros la ruta hacia el archivo que se desea abrir o crear y el segundo, la forma en el cual abrirlo.

2.2 Programación del modelo en RdP

Basado en el modelo en RdP desarrollado en (Cortés, 2017) se propone una programa en el lenguaje de programación Python que simula el funcionamiento de todos los estados de la RdP; este es un lenguaje de programación muy versátil altamente utilizado en la actualidad por muchas compañías de gran prestigio que se dedican al desarrollo de software.

Para el correcto funcionamiento del programa es necesario tener instalado las bibliotecas *numpy.py* que es uno de los módulos más importantes utilizados en este lenguaje de programación, diseñado principalmente por Jim Hugunin, inicialmente llamado *Numeric* y su objetivo era para dotar a Python de funcionalidades de procesamiento matemático de softwares como *MATLAB* y luego de incorporársele una serie de nuevos métodos y funciones toma el nombre que tiene en la actualidad. Por otro lado, es necesario la utilización del paquete *scipy.io* que extiende la funcionalidad de *numpy.py*, implementando módulos que

realizan tareas de optimización, integración, agrupamientos, álgebra lineal etc; y dan acceso a funciones especiales, distribuciones de probabilidad y constantes científicas, entre otros. Este módulo es el que realiza las lecturas de los archivos (.mat) que contienen toda la información de los parámetros del vuelo para el procesamiento del software Diagnosticador de Fallos.

Los estados de la RdP incluyen el funcionamiento normal del Quadcopter y los fallos que pueden presentarse en tierra y en vuelo. Para el desarrollo de la programación se utiliza el programa *Spyder* en su versión 3.0 que facilita el trabajo con auto-completamiento de código, descripción de las principales partes y módulos del lenguaje e imprime mensajes de error cuando ocurre algún problema en las líneas de código o en la compilación del programa.

Como se mencionó en el Capítulo 1, la RdP se compone de lugares, transiciones y arcos que unen a estos. En la programación, estas estructuras se interpretan en forma de objetos. Se crea un objeto de este tipo para cada uno de los 30 lugares presentes en la RdP. En la variable **nombre (**tipo *string***)** se cargan lo nombres de los lugares de la RdP que van desde P1, P2, P3, …P30. Las marcas equivalen a las marcas o *token* de los lugares de la red y en el estado_lugar se cargan con la descripción de cada uno de los lugares; ejemplo: **P15.estado_lugar** = "Quadcopter en vuelo con fallos". Ejemplo la creación en el código de las clases de los objetos tipo lugar tenemos:

```
class lugar(object):
    nombre = "default"
    marca = 0
    estado_lugar = "estado por defecto"
    count = 0
```

Por otro lado las transiciones, que hereda de la clase lugar, cuenta con las mismas propiedades que este, además se le agrega una variable del tipo bandera para asociarlas con el procesamiento de las señales externas, dos variables de la misma clase que **lugar()** para indicar el estado de salida y el de entrada de la transición y un método o función que es el encargado del cambio de estado o el movimiento de las marcas por los lugares asociados, simulando así la evolución de la red de Petri. Al igual que para los lugares se crean 71 objetos de tipo transición. En el atributo **nombre** y **estado_lugar** heredado de la clase lugar, se llenan

con los nombres que van desde T1, T2, T3, … T71 y las descripciones de las transiciones; ejemplo: **T23.estado_lugar** = "IMU lista para vuelo". Como ejemplo de la creación en el código de las clases de las transiciones tenemos:

```
class trans_simp(lugar):
    ext_signal = 0
    lug_asoc = ""
    lug_sal = lugar()
    lug_ent = lugar()
    def trans(self):
```

El modelo en RdP (Cortés, 2017) en cuestión se compone de 30 lugares y 71 transiciones que describen el funcionamiento del Quadcopter X4 GARP en modo normal y los fallos más comunes. Cada uno de estos se inicializa en el código con los objetos explicados. Se agrega una transición nueva correspondiente al cumplimiento de la misión sin fallos. Ejemplo de la inicialización en el código de los objetos correspondientes a los lugares y transiciones tenemos:

```
P1 = lugar()
P1.nombre = "P1"
P1.estado_lugar = "Quadcopter apagado"
P1.marca = 1

T1 = trans_simp()
T1.nombre = "T1"
T1.lug_sal = P1
T1.lug_ent = P2
T1.estado_lugar = "Mando encender Quadcopter (manual)"
```

Dentro de la clase correspondiente a las transiciones es válido destacar la función que realiza el paso de estado de la red moviendo las marcas de los lugares. Ejemplo de parte del código de la función:

```
def trans(self):
    if self.nombre != "default":
    if self.lug_sal.marca >= 1 and self.ext_signal == 1:
    self.lug_sal.marca = self.lug_sal.marca - 1
    self.lug_ent.marca = self.lug_ent.marca + 1
```

El algoritmo de esta función opera de la forma que cuando se activa una señal externa (producto al procesamiento de las variables) en una de las transiciones y el lugar de salida

contiene la cantidad de marcas previstas, automáticamente cuando esta función es llamada (realizado en cada ciclo de procesamiento) quita las marcas en el lugar en salida y pone una marca en el lugar de entrada correspondiente, representando la evolución de la RdP. Debido a la complejidad de la red fue necesario la creación de otras clases de transiciones específicas que difieren en el modo en que se realiza el cambio de las marcas entre los lugares de entrada y de salida, ejemplo cuando existen múltiples lugares de entrada y uno de salida y viceversa.

2.3 Procesamiento de las variables

Para la activación de las transiciones al igual que en las RdP interpretadas de control Figura 1.3, es necesario que se cumpla el marcaje de los lugares de salida (**lug_sal**) y se activen las señales externas que habilitan las transiciones asociadas a estos. Para la activación de las banderas (**T**xx**.ext_signal** = **1**) correspondientes a las transiciones es necesario realizar un procesamiento de las variables externas relacionadas a estas.

Las variables a procesar se encuentran en un archivo obtenido después de realizado el vuelo, descargado y convertido a extensión .mat por el software Mission Planner. Esta extensión del archivo es reconocida por *MATLAB*. En este archivo se encuentran todas las variables organizadas según van siendo procesadas por el controlador de vuelo Figura 2.1. Se compone de varias matrices que en sus columnas guardan la información de las variables y los comandos de la estación en tierra.

Para el procesamiento en Python, como ya se menciona en un acápite anterior se utiliza la biblioteca *scipy.io* para el trabajo con los archivos (.mat), traduciéndolos a un tipo de variable compleja de Python nombrada “diccionarios”, que guardan una serie de arreglos o como se le conoce en términos de este lenguaje de programación “llaves”, que contienen los datos de las variables del vuelo.

Variables - IMU

CURR
CURR_label
ATT
EV
GPS
IMU
POWR
AHR2
AHR2_label

13613x13 double

	3	4	5	6	7	8	9	10
1	2.2258e-04	-2.2285e-04	-2.9530e-04	0.6647	0.8308	-9.9064	0	0
2	-3.4946e-04	-2.5357e-04	-4.3878e-04	0.6727	0.8344	-9.9015	0	0
3	-1.4496e-04	1.2095e-04	-2.1887e-04	0.6835	0.8207	-9.8890	0	0
4	-1.5600e-04	-3.4002e-04	-0.0011	0.6736	0.8332	-9.8958	0	0
5	-3.8001e-04	6.0565e-04	1.7662e-05	0.6913	0.8319	-9.8664	0	0
6	6.5590e-04	-6.0321e-04	-8.2245e-04	0.6730	0.8373	-9.8920	0	0
7	8.7909e-05	-4.4463e-04	9.0143e-04	0.6672	0.8213	-9.8814	0	0
8	2.6552e-04	-5.3713e-04	-2.7376e-04	0.6635	0.8293	-9.8906	0	0

Figura 2.1. Variables

Dentro de las llaves, similares en su organización a las matrices, se guardan los datos de la misión, la primera columna corresponde a la línea del procesamiento general (formada por números enteros de menor a mayor), en la segunda se encuentra el tiempo transcurrido desde el encendido del sistema (en microsegundos). En cada una de las filas se presentan las lecturas de las diferentes variables de los componentes del Quadcopter ya sea IMU, GPS, Barómetro, Batería entre otros; así como los mensajes con los cual trabaja el programa durante la misión. El Diagnosticador de Fallos hace un barrido de estas líneas y procesa las diferentes variables por separado en cada una de las llaves mediante algoritmos que buscan la ocurrencia de fallos. En este procesamiento no solo se analizan los valores que correspondan al fallo, sino también se habilitan las transiciones del comportamiento normal y de fallo del sistema cambiando la variable bandera de estas a 1.

2.4 Diagnóstico de Fallos

Mediante el procesamiento de las variables se pueden encontrar parámetros no deseados que pueden constituir fallos del sistema. En el archivo (.mat) de registro, para el módulo GPS se muestran parámetros como el número de satélites encontrados (NSat), el HDop y Status Figura 2.2. Si el GPS no detecta un mínimo de 7 satélites puede verse afectada la exactitud de la estimación de la posición global. El valor HDop representa la exactitud de la estimación de posición horizontal (latitud / longitud) del GPS, un valor menor 1.5 unidades se considera muy bueno y por encima de 2 ya existe un error considerable en la estimación de la posición (Hulbert and French, 2001). En la columna de Status se pueden presentar 4 estados

enumerados desde el 0 al 3, donde el estado 0 es la no existencia o el estado de apagado del módulo GPS, el 1 representa el encendido, pero no realiza ninguna de sus funciones, 2 es que proporciona las coordenadas 2D, es decir latitud y longitud y el estado 3 informa que el GPS aporta las coordenadas 2D y la altura (3D).

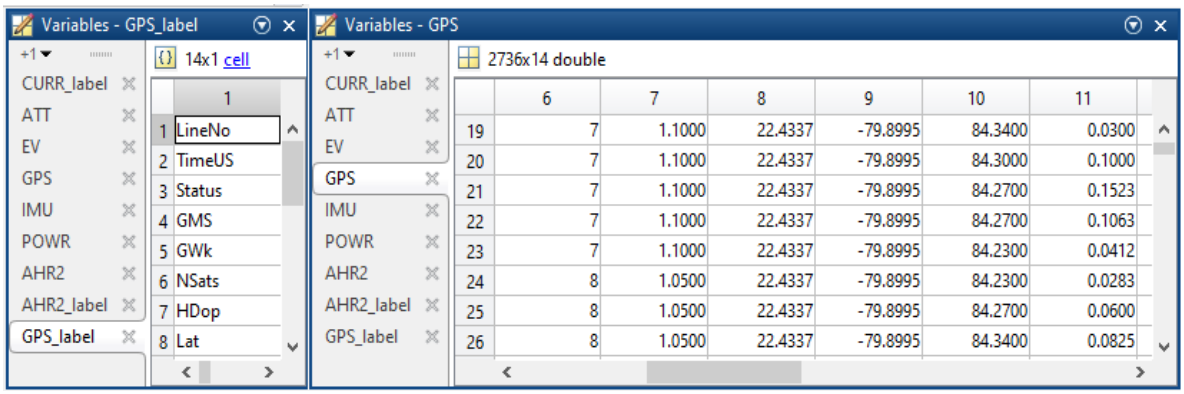


Figura 2.2. Parámetros de GPS

Uno de los fallos más importantes son las que se ocasionan en los motores o actuadores del vehículo, los más comunes son valores de temperatura altas (mayores de 40 °C) y fallo por atascamiento (que se presenta por valores menores a los 6720 RPM que corresponde a un voltaje de alimentación de 7V). Debido a que el Quadcopter X4 GARP no presenta sensores de temperatura o de RPM en los motores se utiliza otro método para inferir el fallo. Los fallos mecánicos más comunes en el motor hacen que aparezca en el registro un repentino alejamiento en el balanceo (*roll*) y cabeceo (*pitch*) deseado contra el balanceo y cabeceo real del vehículo. Esta discrepancia es visible en los registros de control de datos graficando el Roll-In vs Roll y Pitch-In vs Pitch del mensaje de la llave ATT, esto no se nota tanto en los ángulos Yaw-In versus Yaw. Por lo tanto para diagnostica un posible fallo en los motores se analizaran estos datos y si existe una discrepancia superior a la permitida se diagnostica un fallo en este sistema (Nahum, 2016).

El fallo en el suministro de energía puede detectarse a través de la llave CURR, esta guarda los parámetros del voltaje en la batería, la corriente total procedente de esta, y la que llega al controlador de vuelo. Si el parámetro del voltaje de la batería es inferior a 12V existe una condición de fallo de la batería ya sea por baja carga o por otros problemas. Existen vuelos donde esta llave no existe y es necesario entonces tomar en cuenta la llave POWR, que presenta valores relativos a voltajes en el controlador de vuelo, se realiza el mismo

procesamiento, pero esta vez teniendo en cuenta que el límite de voltaje superior de este es 5V, un voltaje inferior a este (menor que 4.5V) ya representaría un comportamiento no deseado de la fuente de energía y un fallo crítico debido al módulo donde este se produce, un error de este tipo en el controlador de vuelo impide todas las funciones que puede realizar el Quadcopter.

2.5 Obtención del registro del vuelo

El Quadcopter X4 GARP después de cada vuelo permite analizar los registros de las variables durante la misión DataFlash log en la memoria MicroSD a bordo del Pixhawk, de la cual se pueden descargar los datos de vuelo después de culminada la misión. Estos ficheros es necesario convertirlos a archivos (.mat), esta operación se puede realizar con el programa Missión Planner que permite en la ventana principal Figura 2.3 en la pestaña *Create Matlab File*, de ahí permite buscar la ubicación del archivo y automáticamente en la misma ubicación genera con un archivo .mat con el mismo nombre del archivo fuente.

Figura 2.3. Conversión de los archivos de vuelo

2.6 Utilización del programa

El software Diagnosticador de Fallos es un programa de fácil utilización y no necesita grandes recursos de procesamiento para su trabajo. En un principio después de su ejecución con una de las terminales de Python (mayor o igual a la versión 3.4 y con permisos de administración) el programa procede a pedir la ubicación del archivo (.mat) del vuelo Figura 2.4; luego se introduce donde se desea que se generen los registros, en este paso hay que tener en cuenta que se debe poner el símbolo (\) al final de esta dirección y se debe tener una

carpeta predeterminada para guardar estos registros, esto ayuda a que no existan conflictos con registros anteriores. Luego del procesamiento el programa en función de la evolución de la red programada, se generan una serie de registros. Estos registros contienen la evolución de la RdP programada, todos los fallos detectados incluyendo los valores de las variables, un fichero que contiene la información de los fallos separados por cada uno de los módulos donde se producen y un registro llamado Chequeos que muestra cuando se comienza y termina el vuelo incluyendo los tiempos respectivos.

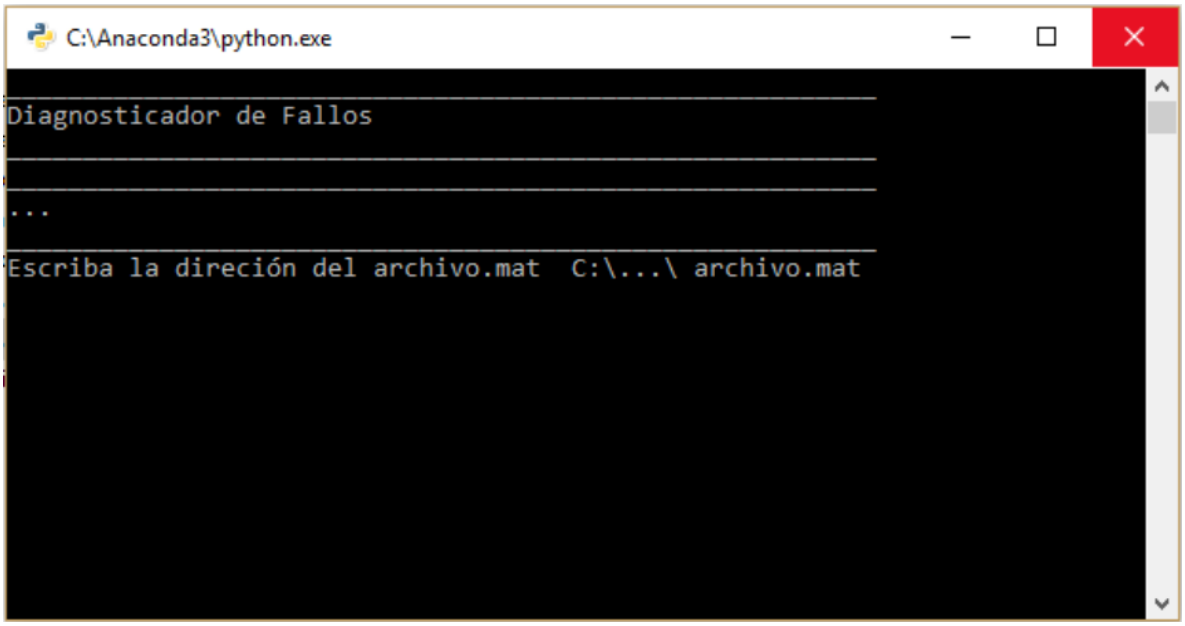


Figura 2.4. Software Diagnosticador de Fallos

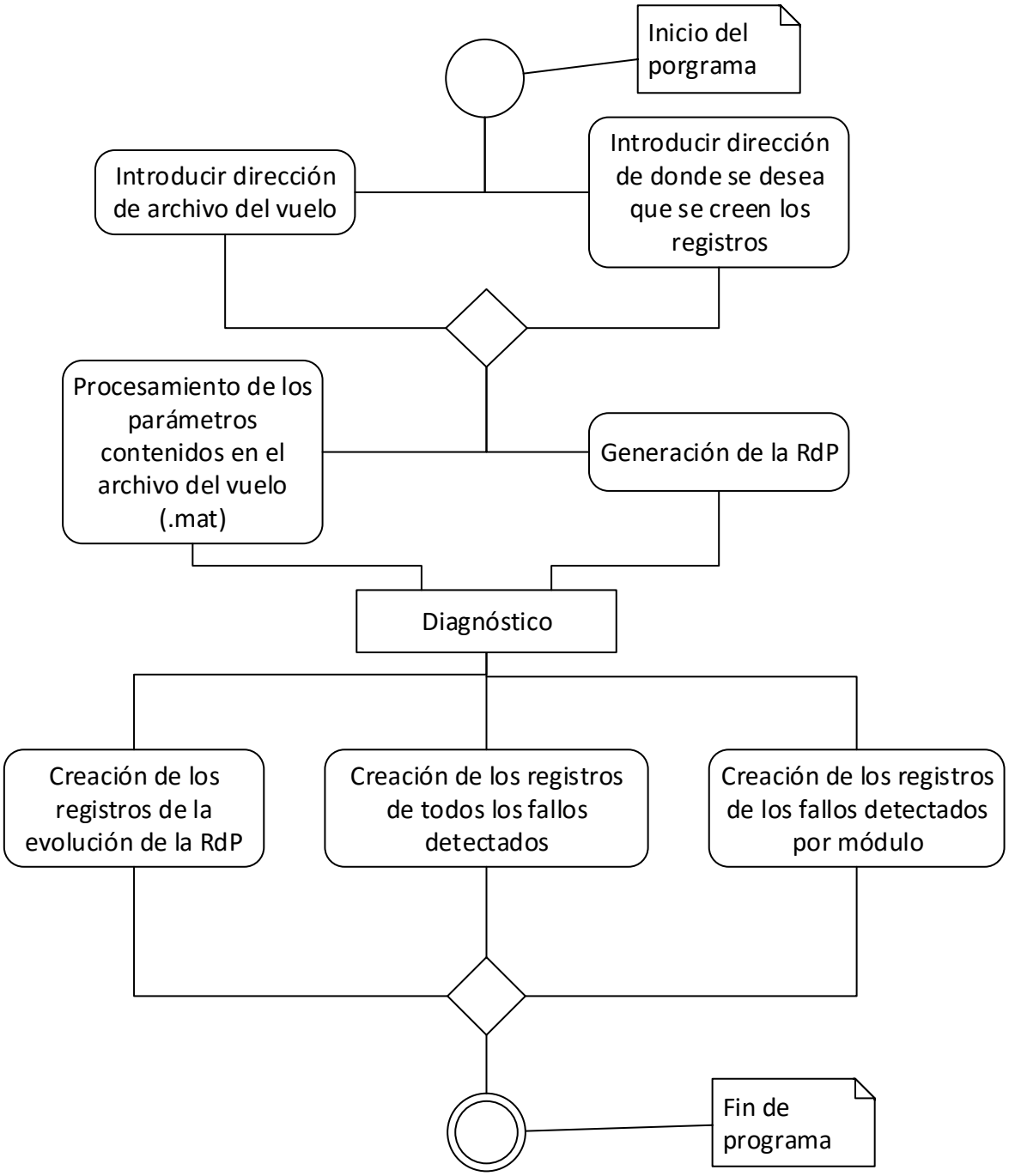


Figura 2.5. Funcionamiento del Diagnosticador de Fallos

2.7 Conclusiones Parciales del Capítulo 2

Python es un lenguaje de programación muy versátil con muchas ventajas sobre los lenguajes clásicos de programación debido a las diferentes reglas de identación que otorgan limpieza y

legibilidad al código, además de las facilidades en el momento de la creación de las diferentes variables.

El correcto procesamiento de las variables del *Dataflash log* descargado posterior a los vuelos es de vital importancia para el diagnóstico de los diferentes fallos modelados en la RdP.

El programa Diagnosticador de Fallos es un software de fácil utilización y no necesita altos recursos de procesamiento para realizar su trabajo y debido a que está programado en Python es un software multiplataforma que puede ser corrido sobre la mayoría de los sistemas operativos actuales.

CAPÍTULO 3. ANÁLISIS DE LOS RESULTADOS

En el capítulo 3 se realiza una serie de pruebas al programa con el objetivo de demostrar el correcto diagnóstico de fallos fuera de línea en el Quadcopter X4 GARP, así como analizar la evolución de la Red de Petri programada para el mismo al presentarse fallos.

3.1 Análisis de fallos en Motores y ESC

En el primer experimento se pretende realizar un análisis de los fallos en el módulo de motor y ESC, este es uno de los fallos más críticos que pueden ocurrir en este tipo de sistema debido a que están directamente relacionados a los actuadores del vehículo, para eso se utilizan los datos obtenidos en un vuelo con presencia de fallos asociados a los motores y ESC.

Al ejecutar el programa con este vuelo la evolución de la Red de Petri en el archivo de registro de las transiciones se muestra que se realizan todos los chequeos de los módulos correctamente (lugares: P4 (motores), P6 (ESC), P8 (Batería), P9 (IMU), P11 (GPS)) y se llega a los estados de listo para vuelo, pero luego del mando de vuelo (T29 llegando al lugar P14 correspondiente a “Quadcopter en vuelo normal”) se activa la alarma de “Posible condición de fallo de motor y ESC” (lugares P16 y P19).

En la Figura 3.1 las transiciones T17 correspondiente a “Posible condición de fallo de motor (alarma)” debido a la detección de amplios errores en el seguimiento de los ángulos *roll* y *pitch* deseados, por consiguiente la evolución de la red activa las transiciones T30, T31 y T37 correspondientes a “Condición de vuelos con Fallos”, “Posible condición de Fallo de motor (en vuelo)” y “Posible condición de Fallo Atascamiento de Motor (en vuelo).

```
SE PRODUCE TRANSICION T17 : Posible condición de fallo motor (alarma) / Sun Jun 10 0
P4 --|--> P16

-------Estado actual--------
SE PRODUCE TRANSICION T18 : Posible condición de Fallo ESC (Vout < 7V) / Sun Jun 10
P6 --|--> P19

-------Estado actual--------
SE PRODUCE TRANSICION T30 : Condición Vuelo con Fallos / Sun Jun 10 02:40:45 2018
P14 --|--> P15

-------Estado actual--------
SE PRODUCE TRANSICION T31 : Posible condición de Fallo en el Motor (vuelo) / Sun Jun
P15 --|--> P23

-------Estado actual--------
SE PRODUCE TRANSICION T37 : Posible condición de Fallo Atascamiento Motor (vuelo) /
P23 --|--> P25
```

Figura 3.1. Registro de Transiciones

Por consiguiente, se generan los registros de fallos en el módulo de motor y ESC como se muestra en la Figura 3.2. Donde se observa los valores de error en los ángulos de navegación contra los ángulos de navegación deseados. Como se puede observar los errores en cada caso superan los 9 grados por lo que son detectados por el programa y presentados en este registro.

```
Posible fallo en motores o ESC; Error ROLL PICH  en línea |  | 5182.0 |
pich error = 9.66  roll error = 4.78

Posible fallo en motores o ESC; Error ROLL PICH  en línea |  | 5214.0 |
pich error = 11.3  roll error = 5.93

Posible fallo en motores o ESC; Error ROLL PICH  en línea |  | 5248.0 |
pich error = 11.61  roll error = 6.8

Posible fallo en motores o ESC; Error ROLL PICH  en línea |  | 5277.0 |
pich error = 11.65  roll error = 7.61
```

Figura 3.2. Registro de fallo en actuadores o ESC

De acuerdo con los resultados arrojados por el programa se muestran en la Figura 3.3 que observan grandes errores en el seguimiento de los ángulos de navegación (errores mayores a

9º) roll y pitch contra roll y pitch deseados, además de vibraciones fuera de los parámetros normales (vibraciones mayores a $\pm 3G$) Figura 3.4.

Figura 3.3. Ángulos de navegación roll y pitch contra roll y pitch deseado

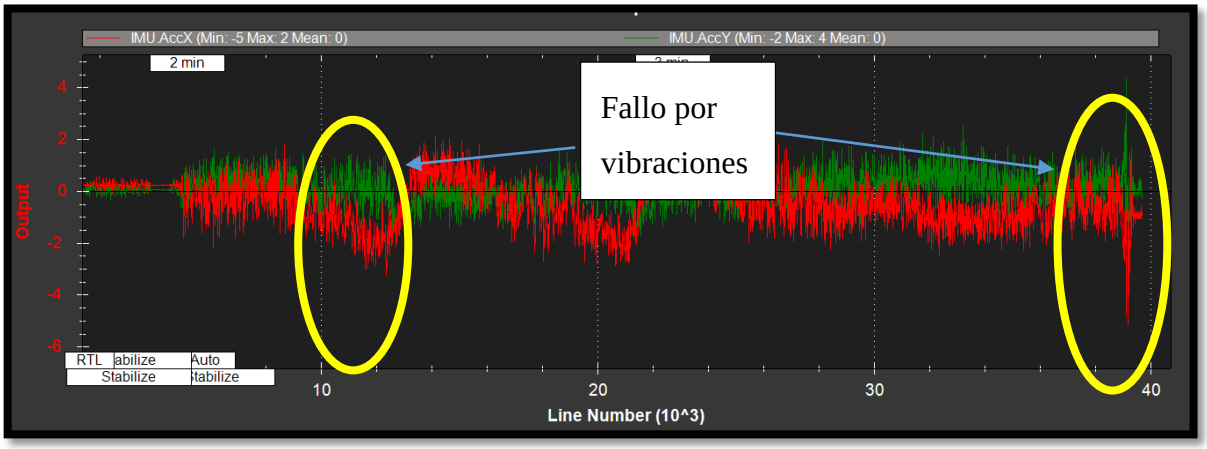


Figura 3.4. Vibraciones en ejes X, Y.

3.2 Analisis de fallos en módulo GPS

Para el segundo experimento se analizó el funcionamiento del módulo GPS. Entre los vuelos del UAV Quadcopter X4 GARP no existe ningún vuelo donde ocurra este tipo de fallos. Con el fin de analizar el funcionamiento del diagnosticador, se realizan modificaciones a los ficheros de los datos del vuelo para poder hacer las simulaciones pertinentes y probar el correcto análisis de detección de fallos. En este archivo se introducen valores fuera de los rangos de comportamiento normal ya sea en el número de satélites (NSat) y el que representa la exactitud de la estimación de posición horizontal (HDop). Estos parámetros se relacionan con la exactitud con que el GPS estima la posición global del sistema; en estos se introducen valores que incumplen lo establecido para un correcto funcionamiento, en el caso del número de satélites se incluyen valores menores que 7 satélites encontrados, y en el HDop mayores que 1.9 unidades. Con este objetivo utilizando la herramienta *MATLAB* se cargan los archivos del vuelo y se cambian los campos antes mencionados (NSat y HDop) apareciendo estos en forma de matrices. A la hora del procesamiento el archivo resultante se guarda con otro nombre y se le pasa al programa Diagnosticador de Fallos.

El vuelo a utilizar es un vuelo que no presenta fallo alguno y completa la misión satisfactoriamente. Al ejecutar el programa con estos datos se obtienen los siguientes resultados.

```
-------Estado actual--------
SE PRODUCE TRANSICION T6 : Chequeo GPS correct. conectado / Tue Jun 12 09:10:28 2018
P2 --|--> P11

-------Estado actual--------
SE PRODUCE TRANSICION T24 : GPS listo para vuelo / Tue Jun 12 09:10:28 2018
P11 --|--> P13

-------Estado actual--------
SE PRODUCE TRANSICION T26 : Condición de fallo GPS (GPS) / Tue Jun 12 09:10:28 2018
P11 --|--> P22

-------Estado actual--------
SE PRODUCE TRANSICION T20 : Motor listo para vuelo / Tue Jun 12 09:10:28 2018
P4 --|--> P13

-------Estado actual--------SE PRODUCE TRANSICION T29 : Mando de vuelo del Quadcopter
P13 --|--> P14
```

Figura 3.5. Registro de Transiciones

En el registro de transiciones Figura 3.5 se observa que todos los chequeos se realizan correctamente al igual que el ejemplo anterior, pero se activa la transición T26 (Condición de fallo GPS) que permite que se llegue al estado del lugar P22 correspondiente a “Fallo en medición de altura”. Se llega a este estado debido a la incapacidad del GPS de no poder hacer una estimación correcta de la altura ante la presencia de menos de 7 satélites detectados.

Todos estos eventos ocurren debido a la presencia de esos valores fuera de los parámetros introducidos en el vuelo, presentados en el registro de fallos en GPS mostrado en la Figura 3.6. Como se puede observar los valores de NSat y HDop son de 6 satélites detectados y un valor de 2 unidades respectivamente.

```
Error en línea = 901.0  fallo en GPS Thu Jun 14 14:46:40 2018
NSat = 6.0  HDop = 2.0
tiempo de la misión  230.578813 seg

Error en línea = 966.0  fallo en GPS Thu Jun 14 14:46:40 2018
NSat = 6.0  HDop = 2.0
tiempo de la misión  230.778725 seg

Error en línea = 1031.0  fallo en GPS Thu Jun 14 14:46:40 2018
NSat = 6.0  HDop = 2.0
tiempo de la misión  230.978856 seg

Error en línea = 1096.0  fallo en GPS Thu Jun 14 14:46:40 2018
NSat = 6.0  HDop = 2.0
tiempo de la misión  231.178936 seg
```

Figura 3.6. Registro de fallas en GPS

3.3 Análisis de fallos en módulo Batería

Al igual que el experimento anterior para el análisis en batería es necesario introducir valores en el archivo de vuelo que representen fallos en este módulo. En el vuelo de muestra existe la llave CURR que contiene los valores de voltajes en la batería por lo que el programa puede realizar un análisis directo con estos parámetros sin tener que inferir el fallo en la llave POWR, que solo contiene el voltaje de la *board* (placa). Se realiza el mismo procedimiento con el *MATLAB* para el cambio de los valores (por valores de voltaje menores a 12V) que se

encuentran en la columna número 3 de la llave CURR. Estos valores se introducen en un vuelo que no presenta fallos para un mejor enfoque en análisis de la evolución de la RdP.

Al ejecutar el programa con este archivo se generan los registros observándose en el de las transiciones Figura 3.7, que se realizan todos los chequeos previos y el vehículo se encuentra listo para vuelo, pero después de darse el mando de vuelo (transición T29 llegando al estado P14 correspondiente a “Quadcopter en vuelo normal”) se activan las transiciones T19, T32, T33 relacionadas a un fallo en el módulo de batería.

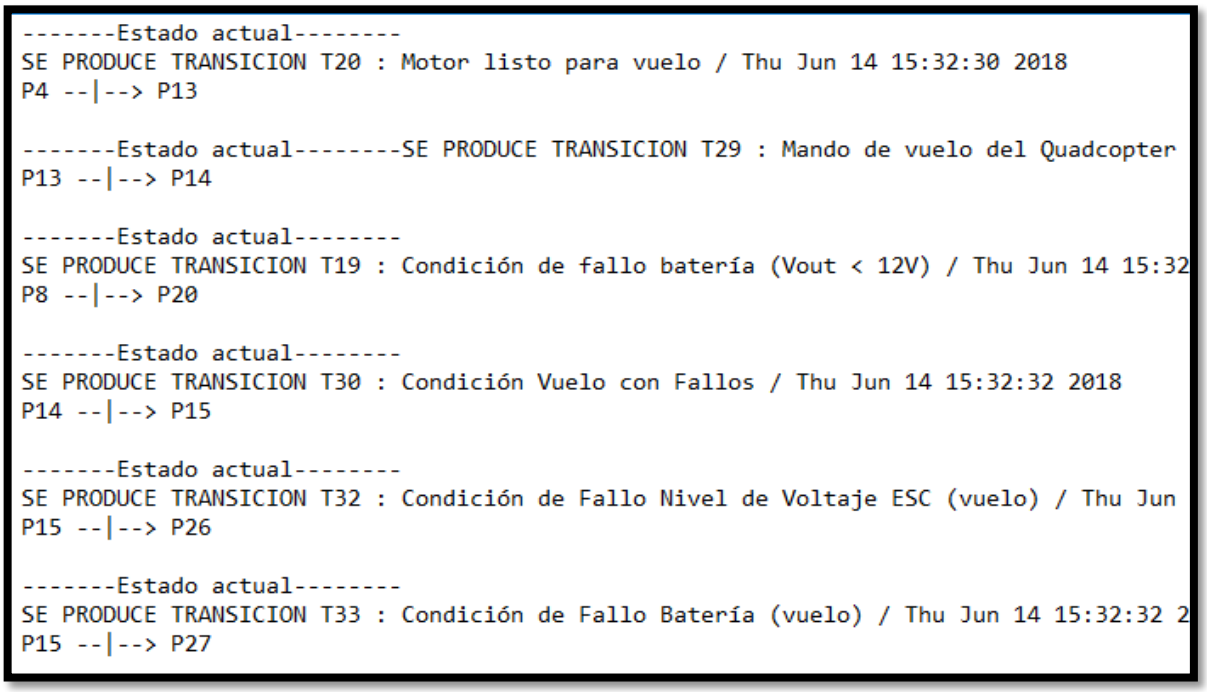


Figura 3.7. Registro de Transiciones

Esta evolución de la red es debida a la detección por el programa de los valores de fallo insertados. En los registros de fallo en la batería se puede observar que el software Figura 3.8 detecta los valores de voltaje por debajo de los límites de correcto funcionamiento.

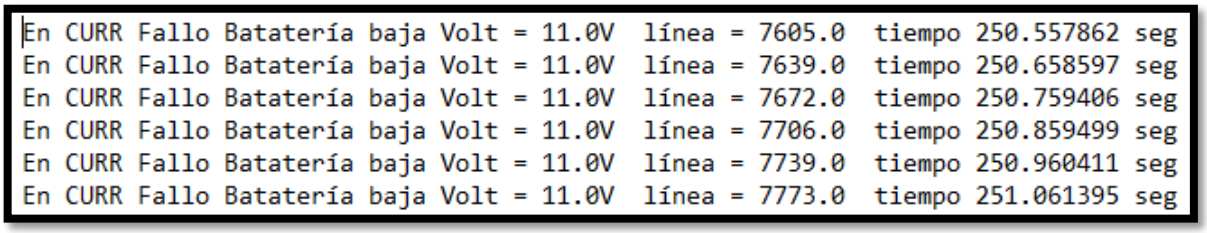


Figura 3.8. Registro de Fallos en Batería

Al realizar un experimento con uno de los vuelos que solo presenta la llave POWR, utilizando el mismo procedimiento de inserción de parámetros de fallos, se muestran resultados similares, generándose registros de fallos similares y mostrándose la misma evolución de la RdP vista anteriormente, aunque el programa detecta la no existencia de la llave CURR y pasa al análisis de los datos en POWR Figura 3.9.

```
En POWR Fallo Batería baja Volt = 4.0V línea = 59075.0  tiempo 402.620706 seg
En POWR Fallo Batería baja Volt = 4.0V línea = 59109.0  tiempo 402.721289 seg
En POWR Fallo Batería baja Volt = 4.0V línea = 59143.0  tiempo 402.822035 seg
En POWR Fallo Batería baja Volt = 4.0V línea = 59176.0  tiempo 402.922267 seg
En POWR Fallo Batería baja Volt = 4.0V línea = 59210.0  tiempo 403.02239 seg
En POWR Fallo Batería baja Volt = 4.0V línea = 59243.0  tiempo 403.123394 seg
```

Figura 3.9. Registro de Fallos en Batería

3.4 Otros Registros importantes

Otra de las funcionalidades del programa, utilizada para observar cuando comienza y termina el vuelo se muestra en el registro de "Chequeo.log". En ese se almacena cuando comienza y termina el vuelo, colocando en este registro *DATA_ARMED* y *DATA_DISARMED* respectivamente con los tiempos y líneas de ejecución. Esto es sacado de la llave nombrada "*EV*" donde se encuentran algunos de los estados del Quadcopter ya sean los chequeos a los componentes, el mando de vuelo y el fin de misión entre otros. Y existe otro registro de fallos el cual hace una recopilación de todos los detectados en vuelo incluyendo los tiempos y las líneas de ejecución en donde se presentan.

3.5 Análisis económico

La realización de un Diagnosticador de Fallos fuera de Línea para el Quadcopter X4 GARP tiene una gran importancia desde el punto de vista económico ya que la ocurrencia de fallos en este sin la debida atención, se pueden convertir en averías para componente como sensores actuadores e incluso la pérdida del vehículo en sí, representando todo esto una inversión innecesaria para la reposición de las partes. Desde un punto de vista medio ambiental, el hecho de que se tenga la certeza de la realización de misiones exitosas gracias a este diagnóstico en tareas de reconocimiento en zonas intrincadas donde se hagan trabajos de

preservación de flora y fauna, mapeo de cultivos para riego inteligente y cálculo de estrés de los suelos y prevención de incendios, permite que se atribuya mucha más importancia a este proceso. En la Tabla 3.1 se muestran los precios de los componentes del Quadcopter X4 GARP.

Tabla 3.1. Precios de los componentes del Quadcopter X4 GARP

Componentes	**Precios**
Estructura	17.70 USD
Módulo de 4 motores brushless 2212920Kv + 4 ESC Simonk 30 A	41.31 USD
Controlador de vuelo Pixhawk	100 USD
Módulo GPS+Compass	20 USD
Módulo de Radio Telemetría	30 USD
Batería LIPo 4S 6000 mAh 14.8 V 35C	98.98 US
Total	308.09 USD

3.6 Conclusiones del capítulo

El software Diagnosticador de Fallos muestra los resultados de los fallos detectados en el vuelo, mostrando la evolución de la Red de Petri del modelo del Quadcopter, así como los valores de las variables cuando ocurren estos eventos en los diferentes registros generados por el programa.

Este diagnóstico permite mejorar la eficiencia de las misiones del vehículo previniendo fallos graves y averías en futuros vuelos.

CONCLUSIONES Y RECOMENDACIONES

Conclusiones

Como resultado de esta investigación se propone un software Diagnosticador de Fallos basado en un modelo en Redes de Petri del Quadcopter X4 GARP, el cual es capaz de analizar los datos obtenidos del controlador de vuelo del vehículo y determinar la ocurrencia de fallos que pudieran afectar el correcto funcionamiento del mismo, lo cual queda demostrado mediante las diferentes simulaciones. Teniendo en cuenta estos resultados se plantean las siguientes conclusiones generales:

1. Después de un análisis profundo de los fundamentos teóricos conceptuales de los temas abordados se llega a la conclusión de que el diagnóstico de fallos empleando Redes de Petri es una de las formas más óptimas dadas las facilidades de interpretación que brinda este método a los sistemas en general.
2. Con el estudio del funcionamiento del UAV Quadcopter X4 GARP se identifican los parámetros de funcionamiento y principales fallos sobre todo, los relacionados con los actuadores y sus rangos de operación, lo que posibilita un mejor desarrollo del Diagnosticador de Fallos.
3. Haciendo uso del lenguaje de programación Python se desarrolla un software Diagnosticador de Fallos de manera sencilla dadas las facilidades que brinda este lenguaje en comparación con otros clásicos como son C++, JAVA entre otros, que no presentan ventajas como el tipado dinámico ni las reglas de identación.
4. Con el empleo de software Diagnosticador de Fallos desarrollado en esta investigación, después del procesamiento de los archivos de vuelos obtenidos del Quadcopter, se pueden detectar los fallos que se manifiestan en los diferentes

módulos, así como tener un dominio de la evolución de la Red de Petri y el correcto funcionamiento del vehículo X4 GARP. Con las simulaciones realizadas queda demostrada la efectividad del mismo.

Recomendaciones

Con el fin de lograr mejoras en el Diagnosticador de Fallos propuesto en el presente trabajo se sugieren las siguientes recomendaciones:

1. Implementar sensores de temperatura, velocidad y voltage en el módulo correspondiente a los motores, así como sensores de voltage en el módulo ESC del Quadcopter X4 GARP posiblilitando realizar un diagnostico más acertado sobre los diferentes fallos que ocurren relacionados a estos partes tan importantes del vehículo.
2. Desarrollar la programación del Diagnosticador de Fallos fuera de línea optimizando los diferentes algoritmos del software, buscando un más rápido procesamiento de los archivos de los vuelos.
3. Continuar con la investigación sobre el Diagnosticador de Fallos para lograr una futura implementación en línea con el Quadcopter.

REFERENCIAS BIBLIOGRÁFICAS

ALAIN, R. C. & NICOLAS, L. 2010. *Safety and reliability in cooperating unmanned aerial systems*, World Scientific.

ALFIE, E., GUOZDEN, T., MAESTRI, M., MARTNEZ, J., SOSA, S. & ZIELLA, D. 2010. Monitoreo inteligente de procesos.

ARÉCHAGA, L. A. P. 2017. *Caracterización de fallos en Vehículo Autónomo Aéreo Quadcopter X4 GARP*. TRABAJO DE DIPLOMA, Universidad Central "Marta Abreu" de Las Villas.

ARRABITO, G. R., HO, G., LAMBERT, A., RUTLEY, M., KEILLOR, J., CHIU, A., AU, H. & HOU, M. 2010. Human Factors Issues for Controlling Uninhabited Aerial Vehicles: Preliminary Findings in Support of the Canadian Forces Joint Unmanned Aerial Vehicle Surveillance Target Acquisition System Project (Incidence des Facteurs Humains sur le Pilotage des Vehicules Aeriens Telepilotes: Constatations Preliminaires a L'Appui du Projet de Systeme Interarmees D'Acquisition D'Objectif au Moyen de Vehicules Aeriens Telepilotes de Surveillance des Forces Canadiennes). DEFENCE RESEARCH AND DEVELOPMENT TORONTO (CANADA).

BRAUER, W. & REISIG, W. 2009. Carl adam Petri and "Petri nets". *Fundamental Concepts in Computer Science.* World Scientific.

CANALES, I. J. C. 2005. *Representación y aprendizaje de conocimiento con Redes de Petri difusas.* CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL INSTITUTO POLITÉCNICO NACIONAL.

CASWELL, G. & DODD, E. 2014. Improving UAV Reliability. *no*, 301**,** 7.

CORTÉS, J. C. B. 2017. *Modelado de Fallos en un Quadcopter utilizando*

Redes de Petr. Universidad Central "Marta Abreu" de Las Villas.

DUCARD, G. J. J. 2007. *Fault-tolerant flight control and guidance systems for a small unmanned aerial vehicle.* ETH Zurich.

DYBSJORD, K. A. 2013. *Fault-Tolerant UAV Flight Control System.* Institutt for teknisk kybernetikk.

EL-FAKIH, K., PROKOPENKO, S., YEVTUSHENKO, N. & BOCHMANN, G. V. Fault diagnosis in extended finite state machines. IFIP International Conference on Testing of Software and Communicating Systems, 2003. Springer, 197-210.

HOBBS, A. & HERWITZ, S. R. 2006. Human Challenges in the Maintenance of Unmanned Aircraft Systems. *FAA and NASA Report*.

HULBERT, I. A. & FRENCH, J. 2001. The accuracy of GPS for wildlife telemetry and habitat mapping. *Journal of Applied Ecology*, 38**,** 869-878.

JOHRY, A. & KAPOOR, M. 2016. Unmanned Aerial Vehicle (UAV): Fault Tolerant Design. *International Journal of Engineering Technology Science and Research,* 3**,** 1-7.

MARTÍNEZ, M. A. T. 2016. *Las Redes de Petri interpretadas en el Diagnostico de Fallos de Sistemas Híbridos. Aplicación a un Helicóptero no tripulado* Tesis de Doctorado, Universidad Politecnica de Madrid.

MURATA, T. 1989. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE,* 77**,** 541-580.

NAHUM, J. 2016. *Simulador de aeronaves no tripuladas: estudio de integración y ensayo en vuelo con el sistema de piloto automático" Ardupilot".* Universidad Nacional de Córdoba. Facultad de Ciencias Exactas, Físicas y Naturales.

OLONDO, J. M., MACHO, J. L. Z. & GALÁN, J. P. 2015. Control de un cuadricóptero para vuelos autónomos en interiores. *Universidad pontificia de Comillas*.

ONCU, M. & YILDIZ, S. 2014. An analysis of human causal factors in Unmanned Aerial Vehicle (UAV) accidents. NAVAL POSTGRADUATE SCHOOL MONTEREY CA GRADUATE SCHOOL OF BUSINESS AND PUBLIC POLICY.

RODRÍGUEZ, C. P. F. 2006. *Análisis de un sistema de eventos discretos mediante RdP*. Escuela Politécnica Nacional.

RUIZ-BELTRÁN, E., JIMENEZ-OCHOA, I., RAMÍREZ-TREVIÑO, A., LOPEZ-MELLADO, E. & MEDA-CAMPANA, M. Fault detection and location in DES using Petri nets. Systems, Man and Cybernetics, 2005 IEEE International Conference on, 2005. IEEE, 1645-1650.

TRIGOS, M., BARRIENTOS, A., DEL CERRO, J. & CANCAR, L. Diagnosticador de Fallos Basado en Redes de Petri y LabVIEW para Evaluar el Estado de un UAV.

VIZCAYA, M. T., BARRIOS, T. R., MORENO, A. P. & SANTIAGO, O. L. 2011. Diagnóstico de fallos en el generador de vapor BKZ-340-140-29M. *Revista Ingeniería Electrónica, Automática y Comunicaciones ISSN: 1815-5928,* 32**,** 31-41.

WHITLOCK, C. 2014. Part One: War Zones When Drones Fall From The Sky. *Washington Post*.

RICAÑO, R. R. 2016. El Control Fotográfico orientado a la creación de Mapas con Vehículos Aéreos No Tripulados.

ALLEN DOWNEY, J. E. C. M., 2002. *Aprenda a Pensar Como un Programador.* Massachusetts: s.n.

Anexo I Estados y Transiciones del Modelo en RdP

ESTADOS	TRANSICIONES
P1 Quadcopter apagado	T1 Mando encender Quadcopter (manual)
P2 Quadcopter encendido	T2 Chequeo Motor correct. armado
P3 Motor apagado	T3 Chequeo ESC correct. conectado
P4 Motor encendido (correct. armado)	T4 Chequeo Batería correct. conectada
P5 ESC apagado	T5 Chequeo IMU correct. funcionando
P6 ESC encendido (correct. conectado)	T6 Chequeo GPS correct. conectado
P7 Batería apagada	T7 Mando encender motor (OnMotor)
P8 Batería encendida (correct. conectada)	T8 Mando apagar motor (OffMotor)
P9 IMU encendida (correct. funcionando)	T9 Mando encender ESC (OnESC)
P10 IMU apagada	T10 Mando apagar ESC (OffESC)
P11 GPS encendido (correct. funcionado)	T11 Mando encender batería (OnBat)
P12 GPS apagado	T12 Mando apagar batería (OffBat)
P13 Quadcopter listo para volar (RTF)	T13 Mando encender IMU (OnIMU)
P14 Quadcopter en vuelo normal	T14 Mando apagar IMU (OffIMU)
P15 Quadcopter en vuelo con fallos	T15 Mando encender GPS
P16 Fallo Motor (Alarma)	T16 Mando apagar GPS

P17 Fallo Calentamiento Motor (sin volar)	T17 Condición de fallo motor (alarma)

P18 Fallo Atascamiento del Motor (sin volar)	T18 Condición de Fallo ESC ($V_{out} < 7V$)
P19 Fallo Nivel de Voltaje ESC (si volar)	T19 Condición de fallo batería ($V_{bat} < 12V$)
P20 Fallo Suministro de Energía (sin volar)	T20 Motor listo para vuelo
P21 Fallo Sensores IMU (sin volar)	T21 ESC listo para vuelo
P22 Fallo Medición Altura (sin volar)	T22 Batería lista para vuelo
P23 Fallo Motor (en vuelo)	T23 IMU lista para vuelo
P24 Fallo Calentamiento Motor (vuelo)	T24 GPS listo para vuelo
P25 Fallo Atasco Motor (vuelo)	T25 Condición fallo IMU ($A * G * M$)
P26 Fallo Nivel de Voltaje ESC (vuelo)	T26 Condición de fallo GPS (GPS)
P27 Fallo Suministro de Energía (vuelo)	T27 Condición de fallo calent. Motor (T)
P28 Fallo Sensores IMU (vuelo)	T28 Condición de atascamiento del motor (RPM)
P29 Fallo Medición de Altura (vuelo)	T29 Mando de vuelo del Quadcopter

P30 Retorno al Punto de Lanzamiento (RTL)	T30 Condición Vuelo con Fallos (T30=T20*T21*T22*T23*T24)
	T31 Condición de Fallo en el Motor (vuelo)
	T32 Condición de Fallo Nivel de Voltaje ESC (vuelo)
	T33 Condición de Fallo Batería (vuelo)
	T34 Condición de Fallo IMU (vuelo)
	T35 Condición de Fallo GPS (vuelo)
	T36 Condición Fallo Calent. Motor (vuelo)
	T37 Condición de Fallo Atascamiento Motor (vuelo)
	T38 Condición RTL desde el fallo calent. Motor (persistencia del fallo durante 30 seg)
	T39 Condición RTL desde el fallo atascamiento motor (persistencia del fallo durante 30 seg)
	T40 Condición RTL desde el fallo nivel voltaje ESC (persistencia del fallo durante 30 seg)
	T41 Condición RTL desde el fallo suministro de energía (persistencia del fallo durante 30 seg)

	T42 Condición RTL desde el fallo en los sensores de la IMU (persistencia del fallo durante 30 seg)
	T43 Condición RTL desde el fallo en la medición de altura (persistencia del fallo durante 30 seg)
	T44 Condición de apagado del Quadcopter
	T45 Relación entre fallo suministro de energía y fallo nivel de voltaje ESC (sin volar)
	T46 Relación entre fallo nivel de voltaje ESC y fallo por atascamiento del motor (sin volar)
	T47 Relación entre fallo suministro de energía y fallo calent. Motor (sin volar)
	T48 Relación entre el fallo por atascamiento del motor y fallo por suministro de energía (sin volar)
	T49 Relación entre el fallo por suministro de energía (sin volar)
	T50 Relación entre el fallo por suministro de energía y fallo medición de altura (sin volar)

	T51 Relación entre el fallo de los sensores de la IMU y fallo en el nivel de voltaje ESC (sin volar)
	T52 Relación entre fallo de los sensores de la IMU y fallo por calentamiento del motor (sin volar)
	T53 Relación ente el fallo en la medición de altura y fallo nivel de voltaje ESC (sin volar)
	T54 Relación entre el fallo en la medición de altura y fallo por calentamiento del motor (sin volar)
	T55 Condición de persistencia del fallo por calentamiento del motor (sin volar) durante 30 seg
	T56 Persistencia del fallo por atasco del motor (sin volar) durante 30 seg
	T57 Persistencia del fallo de nivel de voltaje del ESC (sin volar) durante 30 seg
	T58 Persistencia del fallo suministro de energía (sin volar) durante 30 seg

	T59 Persistencia del fallo e los sensores de la IMU (sin volar) durante 30 seg
	T60 Persistencia del fallo en la medición de altura (sin volar) durante 30 seg
	T61 Relación entre el fallo de los sensores de la IMU y el fallo nivel de voltaje ESC (en vuelo)
	T62 Relación entre el fallo de nivel de voltaje ESC y el fallo por atascamiento del motor (en vuelo)
	T63 Relación entre el fallo suministro de energía y fallo por calentamiento del motor (en vuelo)
	T64 Relación entre el fallo por atascamiento del motor y fallo en el suministro de energía (en vuelo)
	T65 Relación entre el fallo en el suministro de energía y el fallo en los sensores de la IMU (en vuelo)
	T66 Relación entre el fallo en el suministro de energía y el fallo en la medición de altura (en vuelo)

	T67 Relación entre el fallo en los sensores de la IMU y el fallo en el nivel de voltaje ESC (en vuelo)
	T68 Relación entre el fallo en los sensores de la IMU y el fallo por calentamiento del motor (en vuelo)
	T69 Relación entre el fallo en la medición de altura y el fallo en el nivel de voltaje ESC (en vuelo)
	T70 Relación entre el fallo en la medición de altura y el fallo por calentamiento del motor (en vuelo)

Anexo II Modelo General Refinado Redes de Petri