BENEMÉRITA UNIVERSIDAD AUTÓNOMA DE PUEBLA

FACULTAD DE CIENCIAS DE LA COMPUTACIÓN.

PROGRAMACIÓN DE LA VELOCIDAD ANGULAR EN YAW DE UN HELICÓPTERO MINIATURA PARA LA NAVEGACIÓN SEGURA.

T E S I S

Que para obtener el título de:

LICENCIATURA EN INGENIERÍA EN CIENCIAS DE LA COMPUTACIÓN.

Presenta:

Mario Alberto Mateos Rodriguez.

Director de tesis:

M.C. Mariano Larios Gómez.

Puebla, Pue. Julio, 2021

Dedicatoria.

Como todos los actos del universo, la dedicatoria de un libro es un acto mágico. También cabria definirla como el modo más grato y más sensible de pronunciar un nombre. Yo pronuncio ahora sus nombres.

A mis padres, siempre presentes en mis momentos de necesidad y de angustia, brindándome todo el apoyo y cariño que un hijo puede pedir, a ellos por enseñarme a siempre levantarme y seguir peleando, incluso en los momentos más difíciles de la vida, porque la vida es eso, una incesante batalla. Por ser el faro que me guío cuando más a la deriva me encontraba, por ser las bases que cimentaron este logro, en el cual dejo atrás una etapa importante de mi vida, logrando alcanzar una meta más, iniciando así el camino hacia un destino desconocido e incierto, pero seguro de que con sus enseñanzas lograre afrontarlo sin miedo y con la cara en alto.

Este trabajo está dedicado a toda mi familia, padres, hermanos, profesores, amigos y a todas y cada una de las personas que se vieron involucradas en este camino, gracias por permitirme aprender más de la vida a su lado, convirtiéndose así en los protagonistas de este logro.

Agradecimientos:

No alcanzarían las paginas para agradecer a todas las personas que de alguna forma han estado involucradas en la realización de este trabajo, sin embargo, merecen todo mi reconocimiento.

Primeramente, a Mario y Carolina, mis amados padres por apoyarme en mis estudios y brindarme lo necesario, sé que no fue fácil y es por eso que las palabras no bastarían para agradecer por todo.

A mis hermanos, Rosa Isela, Sergio y Mercedes, por todo el apoyo y ánimos para llegar al final de este camino, los amo.

A mis tíos, Cesar, Aranza, Francisco, Claudia y Raúl, por ser grandes ejemplos y consejeros para mí.

Al M.C. Mariano Larios Gómez por todo el apoyo brindado, por la paciencia, dedicación, motivación, pero, sobre todo, gracias por la amistad de quien es el principal coordinador de todo este proceso, no ha sido fácil el camino, ha sido un privilegio colaborar y trabajar a su lado. Gracias por confiar en mi para la realización de este trabajo.

A la Benemérita Universidad Autónoma de Puebla, institución que me formo en todos estos años, la Facultad de Ciencias de la Computación, lugar donde conocí gente extraordinaria y viví momentos inolvidables, mi más profundo agradecimiento a todo el personal académico y administrativo de la FCC.

¡Gracias!, todo esto es posible gracias a ustedes.

Índice general.

Índice de figuras.

Índice de tablas.

1.0 Introducción.

1.1 Resumen.

En la actualidad existen diversos entornos o ambientes donde surge la necesidad de patrullar o explorar con la ayuda de un dron; en especial lugares donde la presencia de árboles o una vasta vegetación sea común, como en bosques o selvas. Es por eso que el presente trabajo de investigación se enfocara en desarrollar algoritmos novedosos que le permitan a un dron navegar de manera segura a través de un ambiente boscoso, de tal manera que evite colisiones con árboles u otros obstáculos existentes en la zona. Es por eso que se requieren tecnologías, como tarjetas de desarrollo con mayor potencia de cálculo, concretamente la tarjeta de desarrollo PIXHAWK, que es un proyecto de independiente de hardware libre realizado con el objetivo de proveer un autopiloto, el cual contiene algoritmos de navegación. El PIXHAWK, tiene su software llamado DroneCode que lo controla, cuenta con telemetría para su comunicación con computadoras o tablets que permite un completo, avanzado y de bajo costo control de vuelo con el software QGroundControl. Este software también pertenece a la plataforma DroneCode y corre en cualquier plataforma Android, IOS, Mac, Windows y Linux.

En este proyecto de investigación de tesis, utilizaremos librerías de C.

Se tendrá como resultado la estabilización de un dron-helicóptero, en su velocidad angular llamada YAW por medio de un algoritmo de control PD.

1.2 Antecedentes del Proyecto.

Esta tesis surgió de un proyecto admitido en el laboratorio de supe cómputo LNS-BUAP, con número de registro 201901014C, gracias a esto se podrá realizar las pruebas necesarias para obtener los resultados deseados.

A través de los tiempos se han propuesto varios algoritmos novedosos para la solución de reconocimiento de caminos, así también se han combinado diferentes metodologías para alcanzar mejores resultados. De los trabajos más destacados y en los que basaremos parte de este trabajo de investigación de tesis, proponen algoritmos para detección de caminos en áreas boscosas, caminos con diferentes texturas como veredas o pavimento.

Los caminos para que un robot pueda tomar una trayectoria son ambiguos, se tiene la dificultad de generar un movimiento controlado en sus ángulos PITCH, ROLL y YAW, además de que un robot terrestre tiene la dificultad de recorrer caminos con tierra suelta y piedras de gran tamaño, es por eso que proponemos un robot aéreo, es decir un dron, que por sus características aerodinámicas nos permitirá el fácil acceso a zonas cerradas, con obstáculos difíciles de esquivar, o de difícil acceso para las personas. Se tomó en cuenta los inconvenientes de energía y las complejidades que esto conlleva, es por eso que se determina implementar un algoritmo sencillo de rápida ejecución, eficiente para el consumo de energía, ya que se tiene poco tiempo de vuelo y poco espacio de memoria en la electrónica embebida.

1.3 Objetivos.

General: El motivo se centra en el análisis e implementación de un algoritmo de control PID (Proporcional Integral Derivativo) para la obtención de resultados satisfactorios en el control de la velocidad angula YAW en un dron tipo helicóptero.

Específicos:

- Obtener la trayectoria y velocidad angular YAW de un dron.
- Poseer la mayor calidad en su desplazamiento.
- Programar el movimiento YAW de un helicóptero clásico miniatura no tripulado
- Permitir la solución de problemas de navegación mediante la implemetacion de un algortimo eficaz.

1.4 Metodología.

En primera instancia se proponen algoritmos de planificación de trayectorias basados en la búsqueda y obtención de estrategias de control seguras, para obtener la trayectoria adecuada con un dron y que posean la mayor calidad en su desplazamiento angular en Yaw. Así como el modelado y la programación en un helicóptero clásico no tripulado miniatura.

La planificación de trayectorias tiene como fundamento la planificación de movimientos y el seguimiento de rutas. Es por eso que se propone, en su caso aplicar o simular algoritmos como Roadmaps probabilístico, otras variaciones con Lyaponov o Voronoi [6, 7, 8, 9] para su respectivo análisis. Así como los algoritmos básicos de planificación RRT (Rapidly-Exploring Random Trees) y RRT-Connect propuestos por LaValle en [8], sin dejar pasar algoritmos de planificación como los propuestos por Dubins [9,10] que consisten en planificar exactamente con 3 segmentos de rutas, creando arboles con los métodos LSL, LSR, RSR y RSL aplicado a un dron como se puede ver en la figura 1.

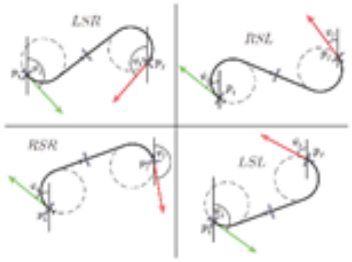

Figura 1. Dubins Paths

Como se comentó anteriormente, los algoritmos y las tecnologías para el reconocimiento de caminos son actualmente un área difícil, más aún que el reconocimiento en tiempo real. En este proyecto de tesis contemplamos tales algoritmos para la comprobación del funcionamiento de nuestra programación en la velocidad angular en Yaw.

Es por esta razón que se propone trabajar en estos aspectos para proponer algoritmos que nos permitan resolver el problema planteado anteriormente. Con esta propuesta nosotros podemos abarcar en un futuro el reconocimiento de caminos en un bosque utilizando una cámara digital de tamaño mínimo y peso posible para poder agregarlo al dron y no se tengan dificultades de peso y/o espacio al momento del vuelo. Utilizando este tipo de herramientas tecnológicas, nosotros podemos adecuar con una cámara a más de 30 imágenes por segundo.

1.5 Cronograma de actividades.

Los resultados reportados se entregaron de manera mensual, revisados por el asesor, la entrega de avances se muestra en la siguiente tabla.

Tabla 1. Cronograma de actividades.

#	Descripción del Hito	Agosto	Septiembre	Octubre	Noviembre	Diciembre	Enero	Febrero
1	Investigación y documentación del funcionamiento de Drones							
2	Modelado y desarrollo de una plataforma para un Dron y su planificación a utilizar							
3	Desarrollo de una aplicación para reconocimiento de caminos en un bosque							
4	Configuración de la localización de un Dron							
5	Mapeo de trayectorias en zonas especificas							
6	Aplicación de una máquina de aprendizaje a un Dron							
7	Pruebas y resultados del sistema en un Dron							

1.6 Infraestructura.

Las características a cumplir del helicóptero son las siguientes:

- Longitud: 634mm, Diámetro
- Motor principal: 715mm
- Peso: 584 g.
- Peso extra que pueda levantar: 700 g.

1.6.1 Hardware necesario.

Con respecto a la a la tarjeta de desarrollo, se recomienda se consideraron la tarjeta Odroid y la tarjeta Jetson TX1 developer kit de NVIDIA [11], por sus multiprocesamiento y facilidad de uso.

El Procesamiento de para el vuelo del dron se programará en la tarjeta de drones Pixhawk Autopilot.

1.6.2 Software necesario.

- SDFormatTool: herramienta para el formateo de unidades de disco.
- 7-zip: herramienta para extraer imágenes con extensión .zip, .rar, etc.
- WinMD5: usado para verificar la integridad de la imagen del sistema operativo.
- Win32DiskImager: herramienta para la carga de un sistema operativo en una tarjeta SD.
- Editor Java: NetBeans, el editor de código Java de mayor uso.
- Librerías Java: Java Development Kit que son librerías diseñadas para el uso de lenguajes orientados a objetos.

Proponemos un sistema de control basado en PID para controlar el movimiento en YAW con posibles perturbaciones arbitrarias para un dron, así también se propone trabajar con varios métodos para obtener una configuración y calibración de la velocidad angular YAW y la posible experimentación en áreas peligrosas o zonas forestales con pequeños caminos.

Este trabajo se enfocará en aplicar algoritmos que le permitan a un dron navegar de manera segura a través de un ambiente boscoso, de tal manera que evite las colisiones contra los árboles, volando sobre caminos o veredas. Este es un impacto socioeconómico, pues nos permite ayudar a la gente o un ser vivo extraviado, navegando de forma segura en zonas de difícil acceso, por ejemplo, poder ayudar a una persona extraviada en algún cerro o el poder detectar elementos peligrosos que perjudiquen el bienestar de un entorno.

Las aportaciones principales consisten en tener una plataforma que manipule y programe un dron, gracias a esta se pretende aplicar otros algoritmos y realizar experimentos, en futuros trabajos, otro aporte es la documentación completa del modelado, programación y control de arquitectura de un dron tipo helicóptero clásico miniatura que se aportara a la FCC-BUAP.

2.0 Estado del arte.

Los robots se han posicionado en todas las áreas, desde las productivas industriales hasta las tecnológicas, se han incorporado de manera acelerada en la vida cotidiana de las personas, todo esto ha representado uno de los avances más espectaculares de la era moderna.

Ningún autor define cuántos y cuáles son los tipos de robots, menos de sus características esenciales. El uso creciente de robots en procesos tanto de producción como de la vida cotidiana, da lugar a que se desarrollen controladores rápidos y potentes, que se basan en microprocesadores, permitiendo establecer con exactitud posiciones de los componentes del robot, así como su desviación o error. Toda esta evolución ha generado toda una serie de tipos de robot, pero en es esta sección solo daremos detalles de los robots aéreos. En este tipo de robots existen dos aproximaciones, en función de vuelo y estructura; helicópteros, normalmente helicópteros RC (Remote Control) convencionales a los que se les añade la electrónica necesaria para tener una visión artificial y la capacidad de toma de decisiones autónoma; drones o aviones no tripulados, actualmente en uso por el ejército de EEUU para tareas de logística militar, apoyo cartográfico y tareas de espionaje [15].

Uno de los problemas difíciles de resolver en el área de la navegación por visión es el reconocimiento de caminos, sobre todo, cuando no se tiene el límite de anchura, así como muy poca información sobre la localización, además de la mínima visibilidad por el pasto o la hierba. Por otro lado, la exploración en zonas boscosas es de gran utilidad pues se pueden adaptar los sistemas de visión para el reconocimiento o localización de animales que habitan en áreas de difícil acceso o también el reconocimiento de objetos que puedan dañar a alguien o detección de sustancias que afecten al medio ambiente en general.

A través de los tiempos se han propuesto varios algoritmos novedosos para la solución de del reconocimiento de caminos. También se han combinado metodologías para lograr mejores resultados. De los trabajos más destacados y en los que se basa este proyecto de tesis son [1,2], que proponen algoritmos para detección de caminos en áreas boscosas. El primero aplica una extracción de 30 iteraciones por frame y el segundo a 50 iteraciones por frame, este último reconoce otro tipo de caminos con diferentes texturas como caminos de madera o pavimento, basándose en algoritmos biológicamente inspirados.

Hoy en día se utilizan algoritmos y paradigmas de aprendizaje para el reconocimiento y evasión de obstáculos que impiden el paso de robots. En este trabajo de tesis se sugiere el uso de algoritmos de aprendizaje basándonos en redes neuronales profundas DNN (Deep Neuronal Networks), porque nos proporcionan más capas para el aprendizaje, así como mejores resultados al examinar una imagen, reconociendo y clasificando obstáculos [1]. Los caminos para que el robot pueda tomar una trayectoria son ambiguos y tienen muchas texturas difíciles de reconocer y clasificar, además de que un robot terrestre tiene dificultades para recorrer caminos con tierra suelta o con piedras de gran tamaño, es por eso que se propone usar un robot aéreo, conocidos como drones, que por sus características aerodinámicas nos permiten el fácil acceso a zonas cerradas, elevadas y con caminos difíciles de transitar, pero cuentan con inconvenientes en cuanto al consumo de energía y las complejidades aeronáuticas que esto conlleva. En primer lugar, necesitamos implementar

algoritmos de rápida ejecución, que sean eficientes y con un óptimo consumo de energía, ya que contamos con un tiempo de vuelo reducido y poco espacio en memoria en la electrónica embebida.

En las siguientes secciones se proponen algoritmos de planificación de trayectorias evitando obstáculos, principalmente árboles, también tomaremos en cuenta algún otro obstáculo que no se reconozca, propondremos también la posible arquitectura a utilizar del dron y sus principales algoritmos de control. De misma manera en otra sección se plantea el problema en forma general sobre el reconocimiento de caminos en zonas boscosas que son de difícil acceso para un robot. Se presenta en otra sección, metodologías y algoritmos a utilizar para establecer las regiones a explorar por medio de localización y mapeo proporcionado mediante un GPS.

2.1 Caso de uso: Aplicación de la Investigación.

En la última década, los desastres en el mundo han dejado sin vida a miles de personas en diferentes lugares de la tierra. Los efectos de un desastre pueden amplificarse debido a na mala planificación de los asentamientos humanos, falta de medidas de seguridad, nulos planes de emergencia o falta de sistema de alertas. Por otra parte, algunos desastres son causados únicamente por las actividades humanas, algunos de estos son: la contaminación del medio ambiente, la explotación irracional de los recursos naturales renovables como lo son bosques y el subsuelo, y los no renovables como minerales. La pérdida de vidas es cada vez más alta y el daño físico, psicológico y material es bastante considerable. Por ejemplo, el terremoto del 2010 en Chile, la investigación al respecto se ha enfocado principalmente en la evaluación del incremento de trastornos psicopatológicos post-desastres y en particular, en trastornos de estrés postraumáticos y depresivos. Los modelos habituales de conceptualización psicopatológica, sin embargo, pueden ser limitados para una apropiada comprensión de los efectos psicológicos de los desastres.

Los desastres han cobrado miles de vidas a la humanidad como se puede ver en la siguiente tabla.

Tabla 2. Desastres naturales recientes.

Lugar	Fecha	Personas fallecidas	Personas desaparecidas
Chile	2010	562	56
Japón	2011	13,135	230
Indonesia	2012	125,000	90
Ecuador	2016	663	9
Nepal	2015	2245	1075

México	2017	98	69

El acceso de estas pequeñas aeronaves a zonas devastadas por deslizamientos, huaycos (corrimientos de tierra en el Perú), inundaciones, tsunamis, terremotos, incendios, entre otros, permite una mayor rapidez en el acceso a zonas inaccesibles, así como la búsqueda de posibles víctimas. Gracias a ello puede evaluarse el territorio, previamente a su acceso, y planificar las formas de acceso y medidas preventivas a tener en cuenta. En desastres como los ocurridos en Nepal se ha empleado esta tecnología para evaluar desde el cielo las repercusiones y el impacto de los accidentes sufridos. Un video que habla por sí solo, a vista de pájaro con drones.

Los drones también pueden ser utilizados para la evaluación de daños posteriores a los eventos catastróficos con el fin de tomar decisiones preventivas u operativas como el proceso de reconstrucción que contribuyan a fortalecer la gestión del riesgo de desastres.

3.0 Diseño y desarrollo del proyecto.

En Park (2016), se plantea el problema de la planificación de tareas en tiempo real con drones o UAV (Unmnned Aerial Vehicle) y propone una solución basada en problemas de satisfacción con una reducción de tiempo de computo factible, en estos ambientes se propone la creación de herramientas precisas para visualizar los procesos en tiempo real como las presentadas en Larios (2017, 2018). Estas herramientas están ligadas a librerías nativas que obtienen los tiempos exactos en sistemas para los procesos y monitorear la planificación de tareas.

3.1 Algoritmos distribuidos de planificación en tiempo real.

En Kshemkalyan (2011), se describe el planteamiento del problema sobre el consenso y acuerdo, "el acuerdo entre los procesos de un sistema distribuido es un requerimiento fundamental para una amplia gama de aplicaciones. Muchas formas de coordinación requieren que los procesos intercambien información, lo cual sirve para negociar procesos y llegar a un entendimiento común, antes de tomar acciones específicas en la aplicación".

Nos enfocaremos en los temas de consenso y acuerdo, Cortés (2016), discute la aplicación de los resultados para el problema de consenso basados en sistemas multiagentes, aquí se proponen dos algoritmos basados en la matriz laplaciana del grafo G de redes que logra el consenso en un tiempo finito usando funciones Lyapunov, de esta manera se propone un mapa distribuido especial para la clase de grafos no dirigidos, estos mapas se utilizan en Cortes (2008), donde se propone un análisis y diseño de estrategias cooperativas para el consenso, como se propone en este trabajo de investigación de tesis, se plantea el uso de redes AD HOC, es por esta razón que se contemplaron los trabajos antes mencionados, por validar sus resultados en redes con topologías de intercomunicación y cambios dinámicos. Otro trabajo con propuestas novedosas sobre los algoritmos de consenso es Hui (2008), en este artículo se re-direcciono el problema de consenso para sistemas distribuidos multiagentes no-lineales y se aplica un controlador con grafos dirigidos y no dirigidos, además este control permite que se pueda trabajar en topologías fijas y cambiables. La aplicación de consensos y acuerdos en multiagentes bajo ambientes distribuidos la podemos encontrar en Hong (2008), este trabajo se basa en las reglas de vecindarios L, para la coordinación de multiagentes. En la búsqueda de un líder activo, se describe la dinámica del agente a seguir con entradas interactivas de control.

Un ejemplo de las aplicaciones de los sistemas distribuidos móviles (SDM), lo podemos encontrar en Esquivel (2013), y en Gómez (2015), este trabajo se enfoca en el problema de determinar una ruta optima, a través de un algoritmo de descubrimiento de rutas, se aplica un algoritmo de consenso en un ambiente distribuido empleando una matriz de adyacencia, después se utiliza la distancia numérica para aplicar el consenso para el caso del estudio de la topología móvil, expresada en la imagen 2.

$$y_i^k = \frac{\sum_{k=1}^{m} w(i,J_i^h) f^k_{\left(i,J_i^h\right)}(\rho)}{\sum_{k=1}^{m} w_{(i,J_i^h)}}$$

Figura 2. Ecuación de la Topología móvil

Donde:

y_i^k = ganador de consenso con la interacción i.

$w(i, j_i^h)$ = nivel de acuerdo con respecto a los pares de entrada.

$f^k_{(i,j_i^h)(p)}$ = función de disponibilidad de los vecinos de i.

En Méndez (2012), podemos observar la aplicación del control difuso basándose en la teoría de Takagi-Sugeno. En este trabajo de tesis nos basamos en plataformas comerciales, aplicando la metodología Takagi-Sugeno para el control difuso, tanto en cuadricóptero y helicóptero se establecen tres puntos de regulación:

- $\theta = -45°$
- $\theta = 0°$
- $\theta = 45°$

Basándose en el modelo matemático de ambas configuraciones de helicóptero se propone reglas del modelo difuso:

- $if\ x(t) = -45°$ $then$ $x(t) = A_1x(t) + B_1u(t)$, $y_1(t) = C_1\,x(t)$
- $if\ x(t) = 0°$ $then$ $x(t) = A_2x(t) + B_2u(t)$, $y_2(t) = C_2x(t)$
- $if\ x(t) = 45°$ $then$ $x(t) = A_3x(t) + B_3u(t)$, $y_3(t) = C_3x(t)$

Estas condiciones son importantes para la obtención de las cuantificaciones deseadas, aunque se limitan a solo tres conjuntos de ellas. Este modelo no lineal independiente de las posiciones angulares φ y ψ (sobre el eje del dron).

Otro trabajo de investigación donde se destaca el análisis y diseño de los sistemas de control en redes (NCS), se encuentra en Esquivel (2015), aquí se experimenta un control LQR (Linear-Quadratic Regulator), aplicado a los retardos en un sistema de tiempo real que se encuentra en un NCS.

A comparación del trabajo Esquivel (2013) y Méndez (2012), se presenta un modelado usando Euler-LaGrange para el helicóptero. Basados en este modelo se aplica un control FF-LQR para controlar un helicóptero regulando el eje en ángulo Pitch don FF. El control FF+LQR usa integrador en el lazo retroalimentado para reducir el error en estado estacionario, usando la FF y la velocidad integral proporcional (PIV) se regula el ángulo Pitch y solo con la PIV se controla el ángulo Yaw.

El control LQR ajusta de manera afectiva el conjunto de frecuencias sin que el cálculo de la ley de control y la reconfiguración impacte en la transición, es de resaltar que esta última reconfiguración disminuye el índice de desempeño provocada por una utilización de la red fuera de su ancho de banda o debido a bajo muestreo de la transmisión fuera de la región de planificación en los ángulos y velocidades en Pitch y en Yaw, es decir φ y ψ respectivamente. Otro trabajo de investigación seria Oriol (2015), donde se presenta una propuesta para el control del helicóptero, el objetivo es mantener el ángulo deseado de Pitch y Yaw, a través de dos hélices con dos motores como actuadores, además de otros casos de uso como la simulación de un sistema de levitación magnética, con un electro magneto como actuador y dos sensores que miden la posición de una bola de acero y la corriente electromagnética. Para mostrar la aplicación del diseño de un NCS se presenta la perdida de paquetes en una comunicación remota entre computadores o dispositivos móviles, basada en el protocolo UDP, el control difuso diseñado es aplicado a ambos casos de estudio, utilizando el modelo difuso diseñado y el modelo de las imperfecciones. Se diseñan las matrices de retroalimentación para cada submodelo discreto a través de un diseño LQR.

3.2 Principios de un dron helicóptero clásico.

En primera instancia se proponen algoritmos de planificación de trayectorias basados en la búsqueda y obtención de estrategias de control seguras, para obtener la trayectoria adecuada con un dron y que posean la mayor calidad en su desplazamiento, así como el modelado y la programación en un helicóptero no tripulado no miniatura.

Proponemos modelar y programar un vehículo no identificado, por ejemplo, un helicóptero clásico no miniatura, con un controlador PID. Se seleccionó la configuración tipo helicóptero por sus características, que otras configuraciones no tienen, como, por ejemplo, el avión. El helicóptero tiene mejor velocidad angular en yaw y en roll, así como la característica del aterrizaje vertical. No tienen funciones complejas porque solo tiene dos motores. Existen configuraciones en helicópteros que dependen solamente de un rotor principal, y este es utilizado para mover las hélices de la cola por medio de una banda atada al motor principal. Otra característica importante que tiene la fuerza de levantar mayor peso. La propuesta de modelo se basa en el modelo VTOL (Vertical Take-Off and Landing) presentado en Zavala (2003), para luego simular la funcionalidad de un helicóptero en miniatura que se pretende desarrollar mediante el concepto de programación multi-threading en un SDM.

En primer lugar, consideramos desarrollar la ley de control y su programación en un microordenador, teniendo en cuenta que este microordenador ya debe contar con una unidad inercial IMU (Inertial Measurement Unit), luego se calibran las ganancias de control en altitud y el control del ángulo yaw, en Larios (2019, pendiente) las pruebas se desarrollan en una plataforma que permite el desarrollo de estos grados de libertad sin tomar demasiados riesgos como usuarios, de esta forma se tendrá montado en una base de seguridad al helicóptero.

En esta sección hablamos del helicóptero y como controlarlo, del control PID y como influye en los canales de la emisora.

Figura 3. Simulación de un helicóptero miniatura como un Dron.

3.2.1 Ángulos de navegación.

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

Se basan en describir la forma de alcanzar la posición final desde la inicial con tres rotaciones, llamadas YAW (guiñada), PITCH (cabeceo) y ROLL (alabeo) y el resultado final dependerá del orden en que se apliquen, primero el yaw, luego el picth y por último el roll. A continuación, se presenta las tres rotaciones sobre un helicóptero.

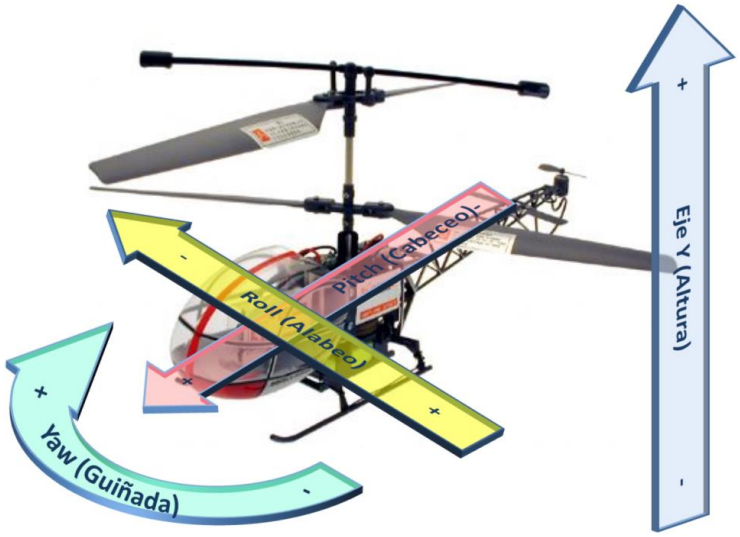

Figura 4. Ángulos de navegación.

A continuación, se detalla cada ángulo.

- **PITCH:** Inclinación del morro del avión, o rotación respecto al eje ala-ala.
- **ROLL:** Rotación intrínseca alrededor del eje longitudinal del helicóptero.
- **YAW:** Movimiento respecto del eje imaginario vertical que pasa por el centro de gravedad del helicóptero. Este eje es perpendicular al Pitch y al Roll, pasa por el morro y la cola del aparato y que normalmente divide a este en dos partes simétricas.

3.2.2 Calibración del helicóptero.

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

En primer lugar, nos aseguramos que las palas del helicóptero estén equilibradas, para esto verificamos que los pesos de cada par estén compensados.

A continuación, aplicamos la maniobra de Hover.

- Revolucionamos el motor hasta que la base se separe ligeramente del suelo, para esto debemos considerar una distancia de mínimo 4 metros del helicóptero.
- Cuando las palas vayan al compás parecerá como si las puntas se solaparan visto desde un lado del rotor, si las palas no van al compás, se ha de ajustar el varillaje que conecta con el brazo del rotor principal.
- Como el comportamiento del helicóptero aún puede ser inestable, para evitar que se balancee y caiga fácilmente, habrá que estabilizarlo, esto consiste en modificar ligeramente los valores que la emisora envía al helicóptero como punto de equilibrio del canal.
- En caso de no ser un valor adecuado para el helicóptero, la razón puede deberse a múltiples causas: en nivel de batería del helicóptero, la batería de la emisora o las interferencias que puede haber en el entorno, si esto no se corrige puede provocar balanceos no deseados del helicóptero impidiendo un control regular.

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

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

3.3 PID.

Un PID es un mecanismo de control por retroalimentación. Un controlador PID corrige el error entre un valor medio y el valor que se quiere obtener calculándolo y sacando una acción correctora que puede ajustar al proceso acorde. El algoritmo de cálculo del control PID se da en tres parámetros:

1. **Proporcional:** Determina la reacción del error actual.
2. **Integral:** Genera una corrección proporcional a la integral del error, asegurándonos que, aplicando un esfuerzo de control suficiente, el error de seguimiento se reduce a cero.
3. **Derivativo:** Determina la reacción del tiempo en el que el error se produce.

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

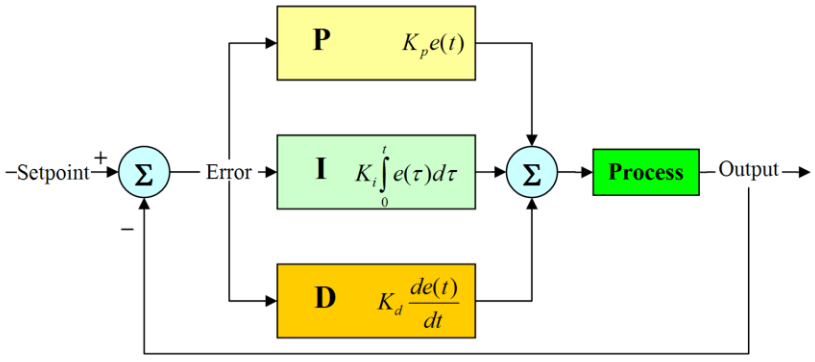

Figura 5. Diseño de un controlador PID.

La representación de la estructura del sistema del controlador bajo punto de vista del controlador PID, desde la entrada hasta la salida enviada a la emisora.

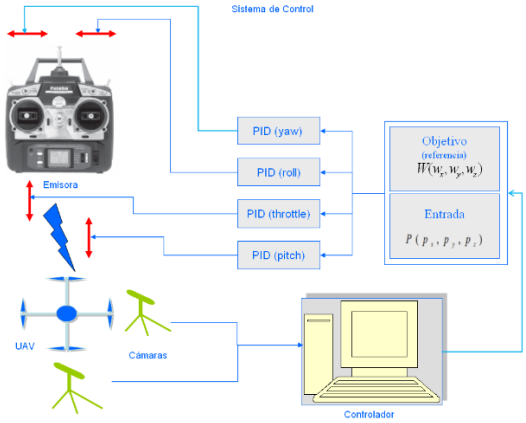

Figura 6. Sistema de conexión PID.

4.0 Modelado y desarrollo de FrameWorks.

En este capítulo describimos las herramientas o frameworks de trabajo que se desarrollaron para el análisis, diseño e implementación de algoritmos de planificación de tareas en tiempo real y la obtención de resultados que estos nos proporcionan. Cabe mencionar que estas herramientas fueron ejecutadas por el tiempo requerido sobre el nodo llamado Fénix que se encuentra en la supercomputadora del LNS.

4.1 FrameWorks JScheduling y JPeer.

A lo largo de los años, el modelo de comunicación centralizada se ha utilizado ampliamente en todo tipo de aplicaciones relacionadas con las ciencias de la computación, informática y tecnologías, sin embargo, los clientes no comparten ninguno de sus recursos, si el servidor falla, todos los solicitantes pueden perder la comunicación. JPeer es una aplicación que monitorea y simula un sistema distribuido móvil, distribuyendo tareas o cargas de trabajo entre el proveedor de servicios y peticiones de servicios. En una red de igual a igual, las cargas de trabajo son particiones entre iguales que tienen el mismo privilegio y hacen parte de sus recursos disponibles para el resto de la red P2P, eliminando la necesidad de una coordinación central. Con las mejoras en las comunicaciones de hardware y red en los últimos años, esta arquitectura puede superar a las redes centralizadas convencionales. Los pares en este proyecto representan dispositivos móviles y con el uso de JNI (Interfaz Java Native), es posible comunicar los Peers creados en Java con los Peers creados en C++, por lo tanto, el paso de mensajes es posible entre diferentes lenguajes de programación y sistemas operativos. Este framework se ejecutó con múltiples pares de software en un nodo del LNS.

Entre las tareas que realizan los algoritmos de planificación están la provisión de tiempo para cada nodo y la gestión de procesos, ya sea en una cola de tareas o en una lista. Es posible, a través de un terminal de línea de comando, rastrear el rendimiento de estos algoritmos. Sin embargo, un entorno visual que facilite este seguimiento sería muy útil. En el Frameworks JScheduling se describe el diseño y la implementación de una interfaz gráfica didáctica para un software integrado distribuido, que permite la representación visual de un algoritmo de planificación en tiempo real en un sistema distribuido móvil virtual.

4.2 FrameWork JScheduling.

El Frameworks JScheduling fue presentado en Larios (2017). Entre las tareas que realizan los algoritmos de planificación se encuentra la resolución del tiempo para cada nodo y la gestión de procesos, ya sea en una cola de tareas o en una lista. Es posible, a través de un terminal de línea de comando, rastrear el rendimiento de estos algoritmos. Sin embargo, un entorno visual que facilite este seguimiento sería muy útil. Este trabajo describe el diseño y la implementación de una interfaz gráfica didáctica para un software integrado distribuido, que permite la representación visual de un algoritmo de programación de procesos en un sistema distribuido móvil virtual. Donde destaca la ayuda que se tiene para testear algoritmos

de planificación. Se desarrolló en Java por la interface gráfica, pero con ayuda de código nativo, esta herramienta logra obtener el tiempo real del sistema.

Este Framework describe la implementación de una interfaz gráfica, llamada JScheduling, para un software integrado; este software muestra el uso de un algoritmo de programación de procesos que permite a una supercomputadora la asignación de sus recursos de manera óptima entre los diferentes nodos conectados a él.

Un conocido algoritmo de planificación de procesos es la utilidad simple de Linux para la gestión de recursos (SLURM) (Breitbart, Weidendorfer y Trinitis, 2015), que es un código abierto y se implementa en muchas supercomputadoras basadas en el sistema operativo por capas llamado Linux. Este programador les da a los nodos un tiempo para ejecutar sus tareas, administra los procesos iniciados por cada nodo y considera una cola de tareas pendientes para acceder a los recursos. Además, SLURM optimiza la asignación de los nodos con un algoritmo centrado en la curva de Hilbert (Costa y Oliveira, 2003).

Otro algoritmo de programación de procesos, que se llamará Fan, otorga a los nodos la misma cantidad de tiempo para ejecutar sus tareas y, después de alcanzar un consenso, decide el acceso a los recursos.

A veces, la comprensión de estos algoritmos puede representar una dificultad debido a la abstracción asociada. Por lo tanto, la implementación de JScheduling para representar un sistema distribuido móvil, donde el usuario puede administrar los nodos de una manera simple, fácil y transparente, así como visualizar cómo cada nodo ejecuta sus procesos, se convierte en una herramienta muy útil y didáctica.

Las siguientes secciones describen la implementación de JScheduling para poder representar, configurar y comunicar los nodos, así como la forma en que se utiliza el algoritmo de planificación de procesos Fan para dar acceso a los recursos a cada nodo.

4.2.1 Desarrollo del algoritmo utlilzando el Framework JScheduling.

Durante la fase de análisis de requisitos y especificación, las funcionalidades identificadas fueron:

a) la creación de la red de nodos (siendo los nodos los dispositivos móviles).
b) la obtención de la información requerida para implementar un programador de procesos en un entorno virtual.

Las siguientes subsecciones describen como se lograron estas funcionalidades.

4.2.2 Representación del nodo.

Para representar un dispositivo móvil virtual, se tuvieron en cuenta algunas consideraciones:

a) Diseño simple.
b) Insertar tantos dispositivos móviles como sea posible.

El primero se logró mediante el uso de un rectángulo simple para representar un nodo, líneas rectas para las conexiones (consulte la imagen 3) y el uso del botón derecho del ratón para editar las opciones. Es importante mencionar que en el back-end del software, se usó una única lista de adyacencia con cuatro elementos (nodo transmisor, nodo receptor, nodo puente y nodo enrutador) para almacenar las conexiones. Esto último se logró combinando la fecha y la hora de la computadora, generando un identificador único para cada nodo.

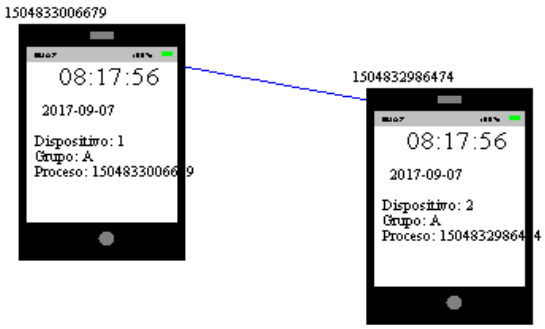

Figura 7. Representación de dos nodos con una sola conexión.

Además, para representar los dispositivos móviles y sus respectivas solicitudes de recursos, se utilizaron las redes de Petri [3], debido a su diseño gráfico. Una gráfica se define como un triplete de conjuntos G = <P, R, A>, donde P es el conjunto de procesos utilizados por el algoritmo de planificación, R es el conjunto de recursos utilizados en un entorno distribuido y A es el conjunto de bordes que se relacionan procesos unos con otros [4].

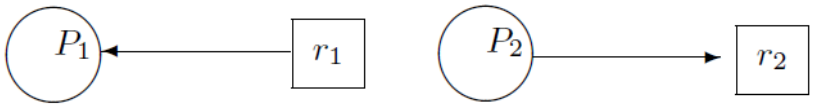

Figura 8. Ejemplo de la aplicación de Petri en un SDM.

4.2.3 Configuración del Nodo.

Para la creación y manipulación de cada nodo en el área de trabajo, se puede usar un menú de control contextual (Clic derecho) para mostrar un conjunto de acciones posibles:

a) Nuevo Smartphone.
b) Arrastrando.
c) Conexión.
d) Eliminación individual.
e) Eliminación general.
f) Intercomunicación – JNI.
g) Cancelación.

La figura 8 muestra el menú que resalta la función Java Native Interface (JNI), que permite las llamadas nativas para la comunicación entre el software incorporado y el sistema operativo (SO) inferior.

Figura 9. Menú de control contextual.

4.2.4 Comunicación entre nodos.

La funcionalidad principal de JScheduling es la comunicación entre los dispositivos móviles virtuales a través de un flujo de información entre dos capas, la capa gráfica y la capa de bajo nivel. Por este motivo, se implementó una multiplataforma con JNI y el sistema operativo del supercomputador (MINIX).

MINIX es un sistema operativo modular en el que los servicios se consideran procesos, en comparación con otros sistemas operativos, donde los servicios se tratan como simples llamadas al sistema [5]. En MINIX, cualquier tipo de proceso puede comunicarse entre ellos a través de primitivas para intercambiar información por medio de mensajes.

En JScheduling la información intercambiada entre los nodos incluye el identificador del nodo, el tiempo de ejecución (inicio, final) y la cantidad asignada por el sistema al proceso. Las líneas de código que se muestran en la figura 9 corresponden a la función que solicita el tiempo en el algoritmo de planificación del proceso y asignan la ID correspondiente al nodo.

```
#include <jni.h>
#include <stdio.h>
#include <string.h>
#include <time.h>
JNIEXPORT void JNICALL Java_DelayTime
  (JNIEnv * env, jobject jobj,jstring i){
 int ID;
 time_t rawtime;
  struct tm * timeinfo;
  time ( &rawtime );
  timeinfo = localtime ( &rawtime );
  const char *str = (*env)->GetStringUTFChars(env, i, 0);
  printf("Dispositivo: %d  Grupo: A  Proceso: %s  Date: %s  \n",
  ID++, str,asctime (timeinfo));
}
```

Figura 10. Generación de ID para cada nodo en la capa de bajo nivel.

Para representar la programación de procesos, un controlador capaz de usar recursos reales se implementó como un archivo DLL (Dinamic Link Library). Cuando se selecciona la opción JNI en el menú contextual, el controlador solicita los procesos de ID de los nodos conectados al supercomputador que trabaja en tiempo real. Luego se usa un terminal de línea de comando para verificar que el intercambio de mensajes entre los nodos conectados fue correcto (figura 10).

```
C: \Users\MarioMateos\Documents\AI>java InsJava
Dispositivo: 2 Grupo: A Proceso: 1504832986474 Date: thu May 07 20:18:45 2020

Dispositivo: 3 Grupo: A Proceso: 1504833002490 Date: thu May 07 20:18:57 2020
```

Figura 11. Intercambio de mensajes entre nodos emisor y receptor.

Se logró la implementación de una interfaz que permite representar cualquier sistema móvil distribuido. Los dispositivos móviles se representan como rectángulos simples, las líneas rectas indican la conexión entre ellos y la información dentro de cada nodo corresponde a los recursos solicitados al sistema. La interfaz se centra en gráficos, bordes y relaciones entre los dispositivos móviles.

Además de la representación gráfica de la red distribuida móvil, es posible visualizar el funcionamiento de un algoritmo de planificación de procesos (fan) por medio de la

información mostrada en cada representación de dispositivo móvil. La figura 11 muestra como la interfaz gráfica y el algoritmo de planificación de procesos están trabajando juntos.

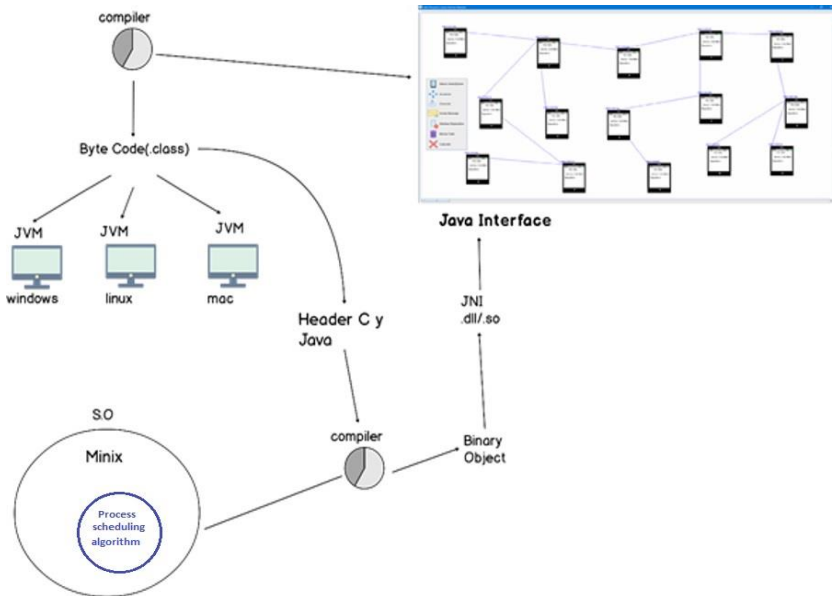

Figura 12. La interfaz gráfica de planificación de tareas con JNI.

La interfaz gráfica implementada permite la representación visual de un sistema distribuido móvil de una manera simple, fácil y transparente. Al mismo tiempo, es posible visualizar como se ejecuta un algoritmo de programación de procesos de abanico a través del intercambio de información entre los nodos por medio de etiquetas insertadas en cada nodo.

Hasta ahora, cada nodo puede ejecutar un solo proceso, pero se planea modificar las funciones para permitir la generación de más procesos. Además, se considera el uso de otros algoritmos de programación de procesos.

4.3 Framework JPeer.

En esta parte de describe la implementación de una interfaz gráfica, llamada JPeer, que es software integrado; esta herramienta o Framework muestra el uso de una red P2P que permite la asignación de recursos de manera óptima entre los diferentes nodos conectados a una red ad-hoc dentro de un nodo de una supercomputadora (LNS, 2019), este trabajo fue presentado en (Larios, Carmona y Placido, 2019).

A veces, la comprensión de los algoritmos de cómputo distribuidos y de sistemas de tiempo real puede representar una dificultad debido a la abstracción asociada. Por lo tanto, la aportación de este Framework es el lograr la representar un entorno de sistema distribuido móvil, donde el usuario puede administrar los nodos (peers) de una manera simple, fácil y

transparente, así como visualizar el cómo cada nodo ejecuta sus tareas y realiza sus servicios, por tal razón esta se convierte en una herramienta útil y didáctica.

El paradigma de igual a igual (P2P) es una red clásica, mientras que todas las computadoras participantes tienen capacidades y responsabilidades equivalentes. Los nodos pueden intercambiar directamente recursos y servicios entre sí sin la necesidad de servidores centralizados. Pueden colaborar para realizar tareas agregando el conjunto de recursos, por ejemplo, almacenamiento, ciclos de CPU disponibles en la red P2P.

La naturaleza distribuida del Framework JPeer proporciona oportunidades emocionantes para el diseño y desarrollo de nuevas aplicaciones con el fin de mejorar y facilitar el análisis de redes P2P. El paradigma P2P se distingue de la computación distribuida tradicional en tres aspectos principales. Primero, la escalabilidad de los sistemas P2P va mucho más allá de los sistemas distribuidos tradicionales. En particular, dado que los sistemas P2P pueden escalar a miles de nodos, pueden aprovechar el poder del cómputo a través de Internet. Segundo, el P2P en su definición más intransigente, requiere que todo esté completamente descentralizado. Idealmente, no deberían existir estructuras centralizadas en los sistemas P2P. Finalmente, y más importante, las aplicaciones P2P a menudo funcionan en entornos altamente dinámicos. Específicamente, en términos de topología de red, ya que los nodos P2P pueden unirse y salir del sistema en cualquier momento, los sistemas P2P no tienen una topología fija. En cambio, su topología cambia según los nodos del sistema. Además, el contenido y la carga del sistema se distribuyen en tiempo real de acuerdo con la demanda y la capacidad de recursos de los nodos.

El paradigma computacional P2P es la base de la programación distribuida, comparte el conjunto de problemas que los investigadores del cómputo distribuido han estado abordando a lo largo de los años (seguridad, confianza, anonimato, tolerancia a fallas, escalabilidad, procesamiento distribuido de consultas y coordinación.). Sin embargo, la computación P2P se distingue de la computación distribuida tradicional en varios aspectos, algunos de los más importantes los podemos apreciar en “Architecture of Peer-to-Peer Systems. In: Peer-to-Peer Computing”

1. Papel simétrico: Cada nodo participante en un sistema P2P generalmente actúa como un servidor y como un cliente a la vez. De hecho, cada nodo instala un solo paquete que abarca tanto código del cliente como del servidor. Como tal, un nodo puede emitir consultas (cliente) y atender solicitudes (servidor).
2. Escalabilidad: A diferencia de los sistemas distribuidos tradicionales, los sistemas P2P pueden escalar a mies de nodos. Como resultado, pueden aprovechar el poder de las computadoras a través de internet. Para lograr esta propiedad, los protocolos P2P no requieren comunicación o coordinación, es un “todos a todos”.
3. Heterogeneidad: Un sistema P2P puede ser heterogéneo en términos de la capacidad de nodos: un nodo puede ser una maquina muy lenta y otro puede ser una computadora de alto rendimiento (supercomputadora).
4. Control distribuido: En su definición más estricta, P2P requiere que todo esté completamente descentralizado. Idealmente, no deberían existir estructuras centralizadas en los sistemas P2P.

5. Dinamismo: Las aplicaciones P2P a menudo funcionan en entornos altamente dinámicos. La topología de los sistemas P2P pueden cambiar muy rápidamente debido a la unión de nuevos nodos o al abandonar los nodos existentes. El contenido y la carga de los sistemas P2P por lo general cambian de acuerdo con la demanda real y la capacidad de recursos de los nodos.

Un framework similar es PeerSim (Montresor y Jelasity, 2009). Este es un entorno de simulación extremadamente escalable que admite escenarios dinámicos, como la rotación y otros modelos de fallas. Los protocolos deben implementarse específicamente para la API de Java de PeerSim, pero con un esfuerzo razonable pueden convertirse en una implementación real. También se admite la prueba en espacios de parámetros especificados. Los iniciadores se ejecutan antes de la simulación, mientras que los controles se ejecutan durante la simulación. Pueden modificar o controlar cada componente. Por ejemplo, pueden agregar nuevos nodos o destruir los existentes; o pueden actuar a nivel de protocolos proporcionándoles información externa o modificando sus parámetros. Los controles también se pueden usar para monitorear pasivamente la simulación. PeerSim, lee el archivo de configuración y carga las clases especificadas en tiempo de ejecución. La función del archivo de configuración consiste en cargar los motores de simulación controlados por ciclos o eventos. El primero, que permite la escalabilidad, utiliza algunas suposiciones simplificadoras, como ignorar los detalles de la capa de transporte en la pila de protocolos. Este último es menos eficiente pero más realista. Entre otras cosas, también admite la simulación de la capa de transporte: los protocolos basados en ciclos también pueden ser ejecutados por el motor basado en eventos, pero no al revés.

Otro framework lo podemos observar en “Cluster computing with Java”. Aquí los autores desarrollaron bibliotecas de software integradas en un grupo de comunidades para el uso de MPI (Message Passing Interface) con una interfaz especifica bien definida disponible en muchas plataformas y en varios lenguajes de programación. Con la necesidad de utilizar lenguajes orientados a objetos, como la realización de múltiples idiomas. Internamente, realizaron la traducción de objetos JVM Java a través de la interfaz JNI nativa. Una vez en el espacio de memoria nativa MPI Java usó una implementación nativa de MPI. Desafortunadamente, el JNI tiene un bajo rendimiento por que la mayoría de los datos transferidos entre la máquina virtual y el espacio nativo deben copiarse. Esencialmente, la implementación MPI Java es un contenedor Java para una implementación nativa de MPI. Por último, en este framework propuesto se evitó la sobrecarga de JNI al integrar estrechamente MPI Java con una versión de Oracle Hyperion. Esta integración también aseguró que las implementaciones de los dos enfoques de clúster de Java, es decir, la técnica de Hyperion de traducir el código de bytes a C y luego la de C al código nativo.

En “Benchmarking Java application using JNI and native C application on Android” los autores muestran la diferencia en el rendimiento y también descubren la causa. Esto podría proporcionar ayuda a los desarrolladores de aplicaciones de Android para la creación de aplicaciones con eficiencia. La interfaz nativa de Java (JNI) se utiliza para para llamar a las aplicaciones y bibliotecas nativas que se escriben utilizando C/C++ en el código Java. Hicieron una biblioteca compartida nativa de código Mibench usando NDK-Build. El

resultado de este experimento muestra que las aplicaciones de Android que utilizan la biblioteca compartida nativa a través del JNI del NDK de Android son más rápidas en comparación con aquellas aplicaciones que se compilan mediante el compilador cruzado nativo en cinco de los seis casos. Parece que la diferencia en las bibliotecas utilizadas también hace una diferencia en el rendimiento.

Android NDK usa la biblioteca Bionic C/C++, pero el compilador cruzado nativo usa la biblioteca GNU C. la biblioteca Bionic C/C++ está desarrollada por Google para sistemas integrados de Android. En general, es pequeño y rápido, ya que esta optimizado para entornos integrados.

El testeo en "Evaluating performance of Android platform using native C for embedded systems" es más detallado, probaron cinco métodos diferentes:

1. Retraso de comunicación de JNI.
2. Cálculo de enteros.
3. Cálculo de punto flotante.
4. Algoritmo de acceso a la memoria.
5. Algoritmo de asignación de memoria del montón.

Como resultado, en "Benchmark dalvik and native code for Android system" también presento que las aplicaciones de Android que usan la biblioteca compartida nativa son más rápidas en comparación con las aplicaciones que usan solo el lenguaje Java en la máquina virtual de Android.

4.3.1 Análisis para los recursos y servicios.

Los sistemas embebidos (Mulchandani y Deepak, 1998) tradicionalmente se han diferenciado de los sistemas de escritorio con respecto a la funcionalidad. Otra consideración es el tamaño de la plataforma Java completa. Muchas personas creen que todo lo que necesita para ejecutar aplicaciones Java es la compatibilidad con una máquina virtual JVM. Sin embargo, para calcular correctamente el tamaño total de la plataforma Java, debe agregar el tamaño de la máquina virtual, el paquete API de Java, la aplicación Java y las bibliotecas de código nativo asociadas. Los paquetes actuales de API de Java tienden a ser grandes, por lo que la API especifica seleccionada para su dispositivo tendrán un impacto significativo en su tamaño. Los componentes agregados también pueden afectar el tamaño de la plataforma.

Durante la fase de análisis de requisitos y especificación, las funcionalidades identificadas fueron:

a) La creación de la red de nodos (siendo los nodos dispositivos móviles).

b) La obtención de la información requerida para implementar un planificador de procesos en un entorno virtual.

Las siguientes subsecciones describen como se lograron estas funcionalidades. En “PeerSim: A scalable P2P simulator” y en “Cluster computing with Java” el orden para representar un dispositivo móvil virtual se tuvieron en cuenta algunas consideraciones, como la realización de un diseño simple y la facilidad para insertar tantos dispositivos móviles sean posibles. El primero se logró mediante el uso de un rectángulo simple para representar un nodo, líneas rectas para sus conexiones y el uso del botón derecho del ratón para las opciones de edición. Es importante mencionar que en el back-end del framework se usó una única lista adyacente con cuatro elementos (nodo transmisor, nodo receptor, nodo puente y/o nodo enrutador) para almacenar las conexiones. Este último se logró combinando la fecha y hora, generando un identificador único para cada nodo.

En una SDM se tienen tres configuraciones:

1. Nodo de envió.
2. Nodo de recepción.
3. Nodo enrutador y/o puente.

Este último puede ser visto como un estado local, según los requisitos y restricciones en un sistema de comunicación de P2P, como se explica en “Peer-to-Peer Query Processing over Multidimensional Data” en un SDM, los nodos pueden comunicarse con los vecinos por medio de una configuración adecuada, que nos permite una multidifusión, en un rango de transmisión desde el transmisor al receptor dentro de una vecindad L, establecida en “Architecture of Peer-to-Peer Systems. In: Peer-to-Peer Computing”.

La comunicación en este entorno de configuración SDM describe una topología dinámica, lo que dificulta el establecimiento y mantenimiento de una ruta, lo que dificulta el establecimiento y mantenimiento de una ruta de comunicación entre los nodos $vi \rightarrow vj$, teniendo en cuenta que:

$$i \neq j \text{ y } vi, vj \in V$$

Donde:

V es el conjunto de nodos o pares en una red móvil, probablemente remota debido a la falta de voluntad para comunicarse en un momento dado. Suponiendo que la solución de usar un grafo cíclico relacionado, reduce estos errores en la planificación y la comunicación entre pares, como se propones en “A scheduling algorithm for a platform in real time”.

Las pruebas y ejecuciones del framework JPeer se llevaron a cabo en un nodo asignado por el laboratorio de supercomputo LNS, como proyecto de investigación en 2018-2019, utilizando instancias del sistema operativo reducido Minix. El núcleo tiene recursos básicos, siendo ideal varias pruebas con al menos 1,000 instancias de pares y un manejo de 1,000 procesos.

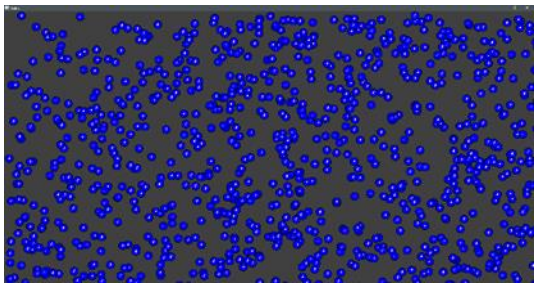

Figura 13. Peers en Java.

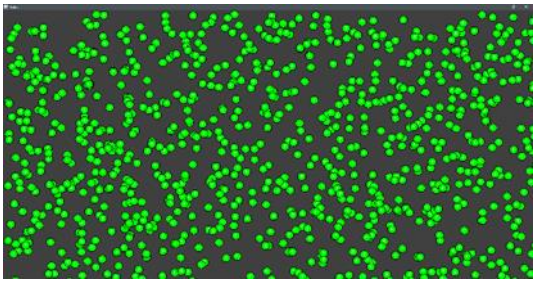

Figura 14. Peers en C++.

En las figuras 12 y 13 mostramos la red P2P, en la figura 9 hay más de 100 pares en Java bajo la JVM. En la figura 10 hay pares en C++ con entorno grafico de Java, lo que implica el uso de JNI, de modo que la interfaz entre el lenguaje de bajo nivel y la interfaz del entorno gráfico.

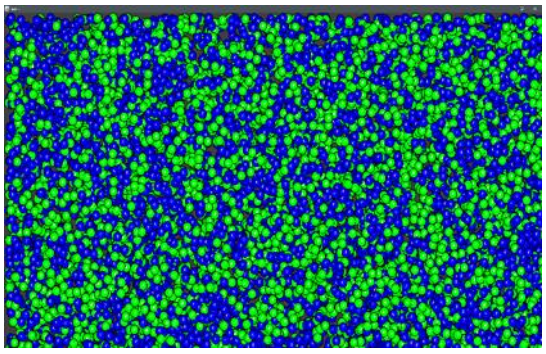

Figura 15. Interfaz de la comunicación entre varios pares con C ++ y Java.

En la figura 14 puede ver la saturación de Peers en diferentes lenguajes de programación para la evaluación del framework con respecto a los límites de recursos y servicios de comunicación a través de mensajes. Los Peers en verde se comunican con los Peers en azul con la ayuda de JNI. Esta interfaz nativa fue muy útil para obtener los datos reales que los

Peers lanzan en C++ y poder comunicarse con los Peers en Java, además de ser visible en un entorno gráfico con la JVM.

4.3.2 Resultados del Framework JPeer.

Para la creación y manipulación de cada Peer en el framework JPeer, se puede usar un menú de control contextual (clic derecho) para mostrar un conjunto de acciones posibles:

a) New Peer C++.
b) New Peer Java.
c) Eliminación individual.
d) Eliminación general.
e) Inter-comunicación JNI.

```
#include <jni.h>;
#include <stdio.h>;
#include "MainWindow.h";
#include <string.h>;
JNIEXPORT jstring JNICALL java_MainWondow_peer(JNIEnv *obj, jobject thisObj)
{
    char msg[60] = "Cmessage";
    jstring result;
    result = (*env)->NewStringUTF(env,msg);
}
```

Figura 16. Método nativo para la comunicación entre Peers.

Para representar el framework JPeer, se implementó un controlador capaz de usar recursos reales como un archivo DLL (Dinamic Link Library). Cuando se selecciona la opción JNI en el menú contextual, el controlador solicita los procesos de ID de los nodos conectados al sistema que trabaja en tiempo real. Luego, se utiliza la línea de comando para verificar el intercambio de mensajes entre los nodos conectados sea correcto (figura 7). Debido a que una computadora personal no cuenta con recursos suficientes para probar la creación de más de un numero de dispositivos móviles, surgió la necesidad de usar una supercomputadora, cuyas características se describen a continuación. La supercomputadora de Cuetlaxcoapan (LNS, 2018), está compuesta de un clúster de cálculo estándar con procesadores Intel Xeon y un clúster con procesadores Intel Xeon Knights Landing con 228 nodos de cálculo Thin (5472 núcleos en total). Cada nodo contiene 2 procesadores Intel Xeon E5-2680 v3 (Haswell) a 2.5 GHz, 12 núcleos por procesador (24 núcleos totales), 128 GB de memoria DDR4 a 2133 MHz, 2 interfaces de red Gigabit Ethernet para administrar el hardware de la supercomputadora y el suministro de software a los nodos.

Por último, se logró la implementación de una interfaz que permite representar el paradigma del sistema distribuido. Los Peers entre ellos y la información dentro de cada nodo

corresponden a los recursos solicitados al sistema. La interfaz se centra en los bordes de los gráficos y las relaciones entre las redes P2P.

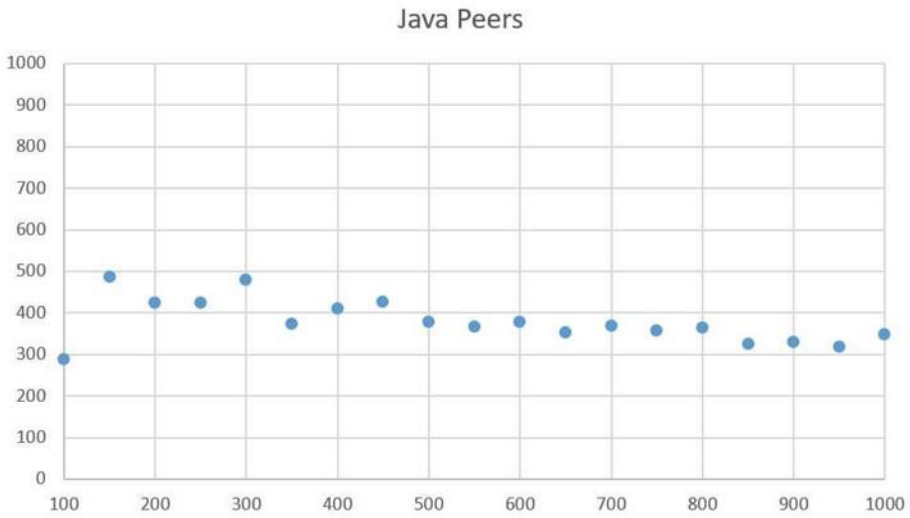

Figura 17. Balance de Carga del Clúster.

En la figura 16, mostramos el saldo de la carga del clúster en el momento en que aumenta el uso del mismo tiempo mientras crece el número de acompañantes, es decir, con 100 Peers, el uso de la computadora fue de 288 por segundo, mientras que con 10000 es de 6 a 7 segundos. El tiempo de inicio y lanzamiento es de aproximadamente 13.4 segundos.

El framework con interfaz gráfica propuesto, permite la representación visual de las redes P2P en un sistema distribuido, para mostrar de una manera fácil y transparente. Al mismo tiempo es posible visualizar como se ejecuta una red P2P en múltiples lenguajes de programación a través del intercambio de información entre los nodos, mediante etiquetas insertadas en cada nodo. Hasta ahora, cada nodo puede ejecutar un solo proceso, pero se propone en un trabajo futuro modificar las funciones para permitir la generación de más Peers.

5.0 Desarrollo y programación de algoritmos de control en el Dron.

5.1 Configuración y programación de los controles.

En esta parte del codigo declaramos la configuracion con respecto al control que usaremos para la manipulacion del dron.

```
// add throttle to more control
//-------------------------------------------------------
                        actuators.control[3] = manual.throttle;
                        actuators.control[4] = manual.aux1;
                        //actuators.control[5] = manual.assisted_switch;
                        //actuators.control[6] = manual.mission_switch;
//---------------------------------------------------------------------
                        actuators.control[7] = manual.aux3;


                        actuators.timestamp = hrt_absolute_time();
                        orb_publish(ORB_ID_VEHICLE_ATTITUDE_CONTROLS,
actuator_pub, &actuators);


                        perf_end(mc_loop_perf);
```

5.2 **Programación de la altitud de del dron.**

El peso, el modelo y la fuente de energía de un dron son factores que importantes que influyen en la altitud máxima, duración de vuelo, rango de vuelo y carga útil máxima.

Es por eso que debemos tener un algoritmo eficiente donde todos los parámetros estén bien definos. A continuación presentamos el algoritmo que se planteo.

```
if (initialized == false) {
		parameters_init(&h);
		parameters_update(&h, &p);

		pid_init(&pitch_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f, PID_MODE_DERIVATIV_SET, 0.0f);
		pid_init(&roll_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f, PID_MODE_DERIVATIV_SET, 0.0f);
//Prueba.-------------------------------------------------------------------------
		pid_init(&yaw_controller, p.yaw_p, p.yaw_i, p.yaw_d, 1000.0f, 1000.0f, PID_MODE_DERIVATIV_SET, 0.0f);

		initialized = true;
	}

	/* load new parameters with lower rate */
	if (motor_skip_counter % 500 == 0) {
		/* update parameters from storage */
		parameters_update(&h, &p);

		/* apply parameters */
		pid_set_parameters(&pitch_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f);
		pid_set_parameters(&roll_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f);
//Prueba.-------------------------------------------------------------------------
		pid_set_parameters(&yaw_controller, p.yaw_p, p.yaw_i, p.yaw_d, 1000.0f, 1000.0f);
	}

	/* reset integrals if needed */
	if (reset_integral) {
		pid_reset_integral(&pitch_controller);
		pid_reset_integral(&roll_controller);
//Prueba.-------------------------------------------------------------------------
```

```
			pid_reset_integral(&yaw_controller);
		}

		/* calculate current control outputs */

		/* control pitch (forward) output */
/*
		Modificacion de pitch con respecto al mixer. resultado por -1
		MARINANO LARIOS G ----------------------------------------------------
		pid_calculate(&pitch_controller, att_sp->pitch_body ,att->pitch, att->pitchspeed, deltaT);
*/
		rates_sp->pitch = -pid_calculate(&pitch_controller, att_sp->pitch_body ,
						att->pitch, att->pitchspeed, deltaT);

		/* control roll (left/right) output */
		rates_sp->roll = pid_calculate(&roll_controller, att_sp->roll_body ,
					att->roll, att->rollspeed, deltaT);

		if (control_yaw_position) {
			/* control yaw rate */// TODO use pid lib
			/* positive error: rotate to right, negative error, rotate to left (NED frame) */
			// yaw_error = _wrap_pi(att_sp->yaw_body - att->yaw);

			yaw_error = att_sp->yaw_body - att->yaw;

			if (yaw_error > M_PI_F) {
				yaw_error -= M_TWOPI_F;

			} else if (yaw_error < -M_PI_F) {
				yaw_error += M_TWOPI_F;
			}
// Cambio de direccion en control de attidtude para yaw
//			MARINANO LARIOS G ----------------------------------------------------
//			rates_sp->yaw = p.yaw_p * (yaw_error) - (p.yaw_d * att->yawspeed);
//			rates_sp->yaw = -(p.yaw_p * (yaw_error) - (p.yaw_d * att->yawspeed));
			rates_sp->yaw = -pid_calculate(&yaw_controller, att_sp->yaw_body , att->yaw, att->yawspeed, deltaT);
//
//			if((float)fabs(rates_sp->yaw)>yaw_lim){
```

```
//                  rates_sp->yaw=constrain(rates_sp->yaw,-
yaw_rate_max,yaw_rate_max);
//                  }
```

5.3 El tunning, programación del control del dron.

Realizamos el ajuste de valores con los cuales controlaremos el vuelo del dron, realizamos la comunicacion con el controlador de vuelo.

```
int tunning_main(int argc, char *argv[])
{
	if (argc < 1)
		usage("missing command");

	if (!strcmp(argv[1], "start")) {

		if (thread_running) {
			warnx("daemon already running\n");
			/* this is not an error */
			exit(0);
		}

		thread_should_exit = false;
		tunning_task = task_spawn_cmd("tunning",
					SCHED_DEFAULT,
					SCHED_PRIORITY_DEFAULT,
					2048,
					tunning_thread_main,
					(argv) ? (const char **)&argv[2] : (const char **)NULL);
		if(tunning_task<0)
		{
			warnx("Error running task\n");
		}

		exit(0);
```

```
        }

        if (!strcmp(argv[1], "stop")) {
                thread_should_exit = true;
                exit(0);
        }

        if (!strcmp(argv[1], "status")) {
                if (thread_running) {
                        warnx("\trunning\n");
                } else {
                        warnx("\tnot started\n");
                }
                exit(0);
        }

        usage("unrecognized command");
        exit(1);
}
```

5.4 Librería del tunning.

Seleccionamos y declaramos los parametros que con los que realizaremos el ajuste al momento de inicializar la comunicacion con el control del vuelo.

```
/**
 * Mainloop of mav3dsim.
 */
int tunning_thread_main(int argc, char *argv[]);


/**
 * Print the correct usage.
 */
static void usage(const char *reason);


typedef union
{
	int32_t l;
	unsigned char b[4];
}convDouble;


char* select_params[] = {"MC_ROLLRATE_P",// 0
                                        "MC_ROLLRATE_I",        // 1
                                        "MC_ROLLRATE_D",        // 2
                                        "MC_YAWPOS_P",          // 3
                                        "MC_YAWPOS_I",          // 4
                                        "MC_YAWPOS_D",          // 5
                                        "MC_PITCHRATE_P",       // 6
                                        "MC_PITCHRATE_I",       // 7
                                        "MC_PITCHRATE_D",       // 8
                                        "MC_YAWRATE_P",         // 9
                                        "MC_YAWRATE_I",         // 10
                                        "MC_YAWRATE_D",         // 11
```

```
"MC_ROLLPOS_P",     // 12
"MC_ROLLPOS_I",  // 13
"MC_ROLLPOS_D",          // 14
"MC_PITCHPOS_P",         // 15
"MC_PITCHPOS_I", // 16
"MC_PITCHPOS_D",    // 17
"MPC_XY_VEL_I",          // 18
"MPC_XY_VEL_D",          // 19
"MPC_XY_VEL_MAX",     // 20
"MPC_Z_P",               // 21
"MPC_Z_I",               // 22
"MPC_Z_D",               // 23
"MPC_Z_MAX",             // 24

};
```

6.0 Pruebas y Resultados.

Las pruebas que se realizaron se ulizaron dos tipos de drones, el Trex-450, es un helicóptero miniatura y el parrot versión 2, que es un cuatricopero.

Figura 18. Modelo del helicóptero miniatura.

Un helicóptero es un VTOL (Vertical Take-Off and Landing). Como se muestra en la figura 19, se tiene la forma de funcionar en un plano 2D. Un VTOL despegue y aterrizaje verticales, es una capacidad de ciertos aviones para efectuar las maniobras de despegue y aterrizaje de forma vertical utilizando pods mientras que en vuelo recto y nivelado se utiliza el método de propulsión horizontal.

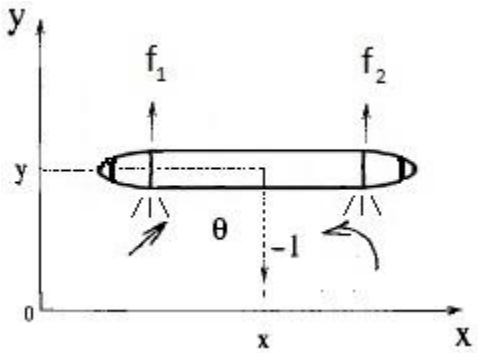

Figura 19. VTOL.

Donde se tiene f1 y f2 dos fuerzas aplicadas al VTOL y esto genera una velocidad angular en ROLL asignado a theta. Este es el principio de funcionamiento de un Dron y es muy importante.

Las capacidades VTOL en los aviones era enorme, y mientras resultaba fácil mover los aviones a otros sitios, no así su mantenimiento completo (armamento, combustible, repuestos, mecánicos). Hacia mediados de los 60 el interés en las capacidades VTOL decreció, sobre todo por la masiva introducción de los misiles balísticos intercontinentales como principal arma nuclear de los sistemas de defensa y ataque.

Aunque también se optó por soluciones intermedias adaptando cohetes a cazas y bombarderos con el programa JATO (Jet Assisted Take Off, Despegue Asistido por Cohetes) que fue probado con éxito en varios aparatos y no tanto en otros.

Pasando el disñeo a una simulación en MatLab con SimuLink, nosotros obtenemos el diseño como enla figura 20.

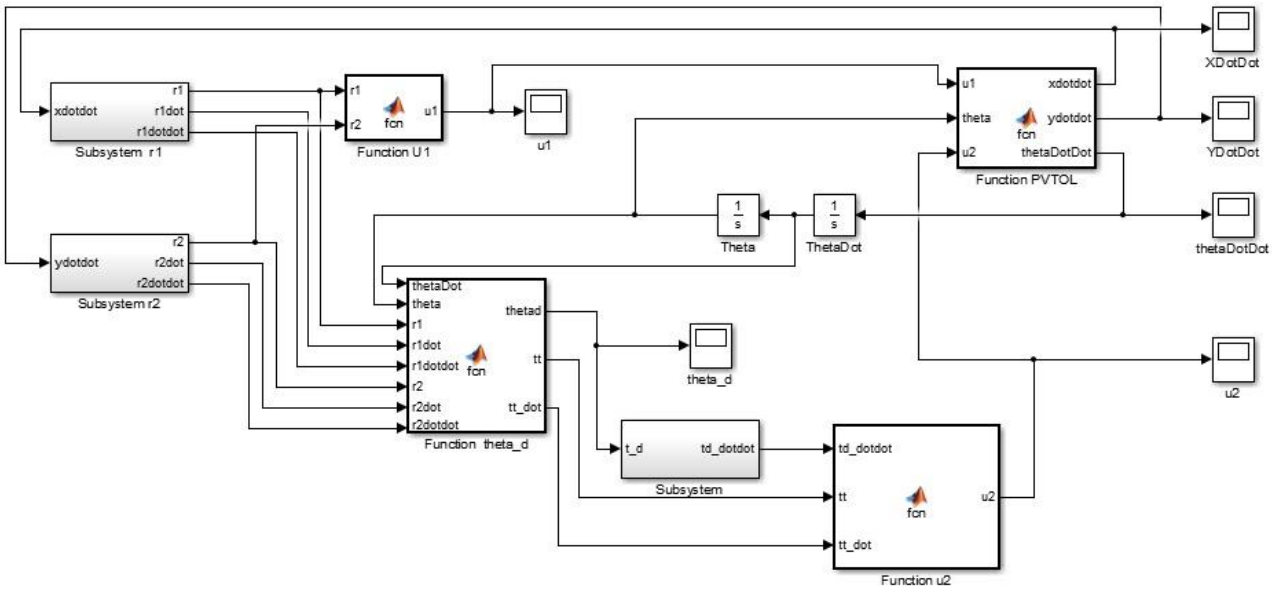

Figura 20. Simulación del VTOL.

Nos da los resultados en Theta comos e muestra en la figura 21. Se tiene una perturbación y se controla en un momento después.

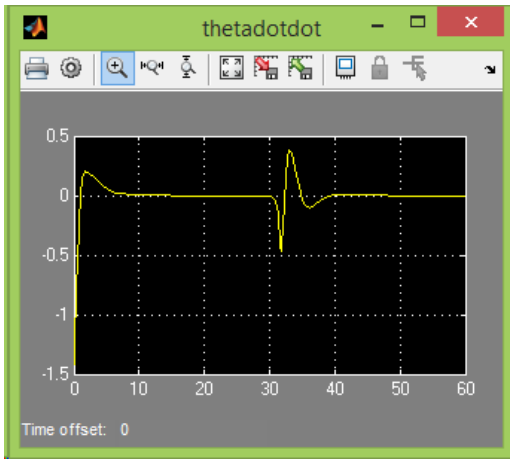

Figura 21. Desplazamiento angular en $\ddot{\theta}$

En la Figura 22 se tiene el resultado de la simulación aplicando la fuerza en X y nos da un desplazamiento como se muestra en la gráfica. Así también se puede observa la respuesta al impulso del lazointerno y externo, se puede observar que en la primera respuesta es más rápida que la respuesta del lazo externo.

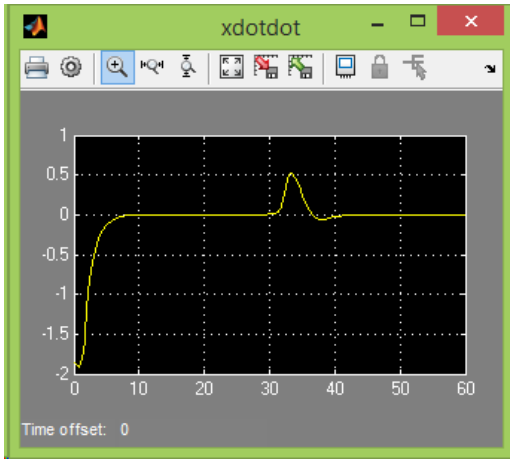

Figura 22. Desplazamiento en $\ddot{X}$

En la figura 23 se muestra ahiora resultado de la simulación aplicando la fuerza en Y y nos da un desplazamiento como se muestra en la gráfica.

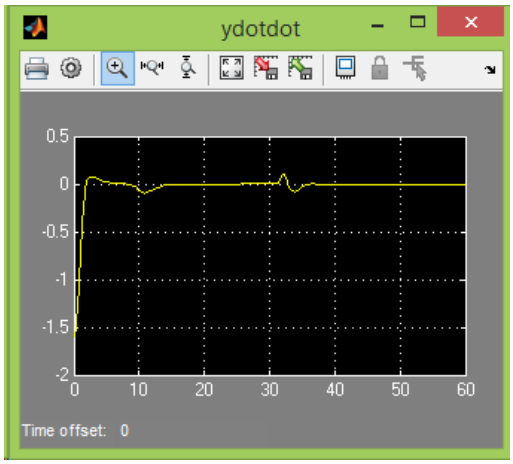

Figura 23. Desplazamiento en Y¨

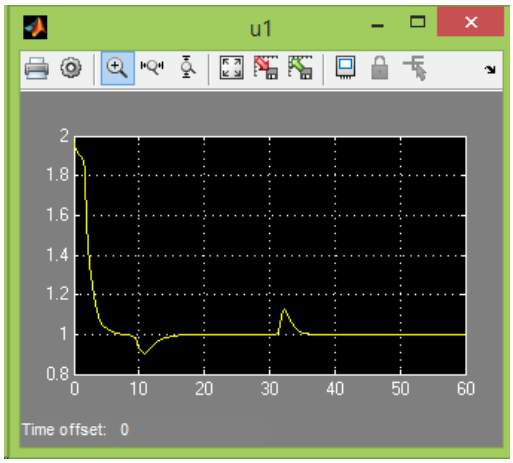

Figura 24. Resultado en U1.

.

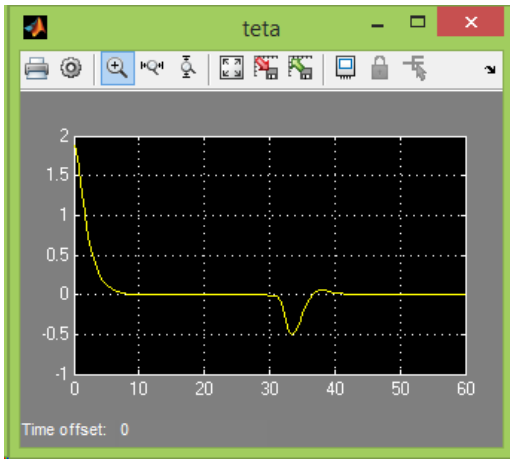

Figura 25. Resultado en U2.

En la figura 26 se muestra la forma de configurar el control remoto para el dron. Se asigan los canales para una acción especifica. En el canal uno se configuro para controlar el servo que hace girar al dron en roll. El canal 2 esta destinado para controlar el movimiento en Pitch, el canal 3 y 4 se junto para controlar al motor y poderle dar potencia o disminuirla. Yel canal 7 se configuo para controlar el movimiento del Yaw. Este ultimo es el que se enfoca esta tesis.

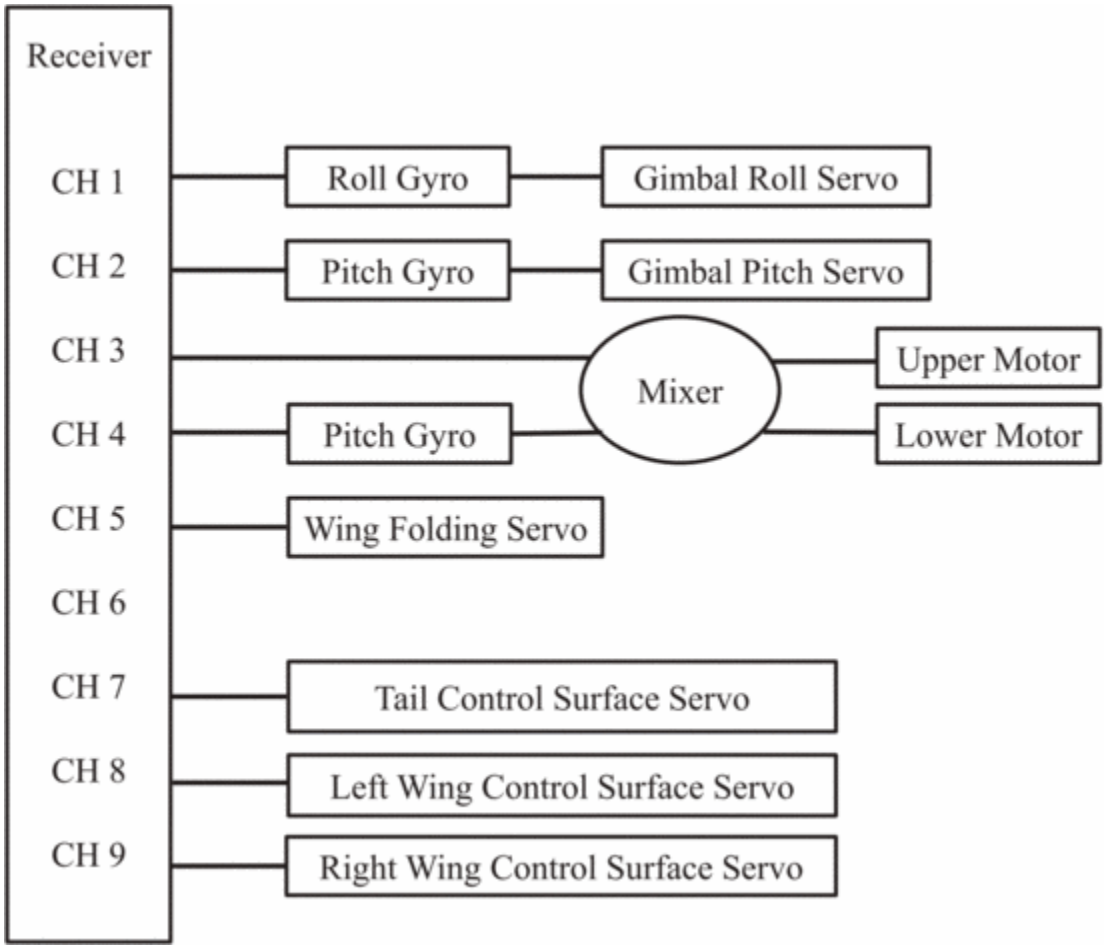

Figura 26. Mixer para el control remoto del Dron.

El Mixer transforma los pares solicitados por el controlador y el empuje total en velocidad de giro los rotores al cuadrado, aplicando las saturaciones y parámetros correspondientes, configurados por el autor [3].

En la figura 27 se muestra el dron realizando pruebas en un espacio controlado por parte del profesor Larios de la FCC-BUAP.

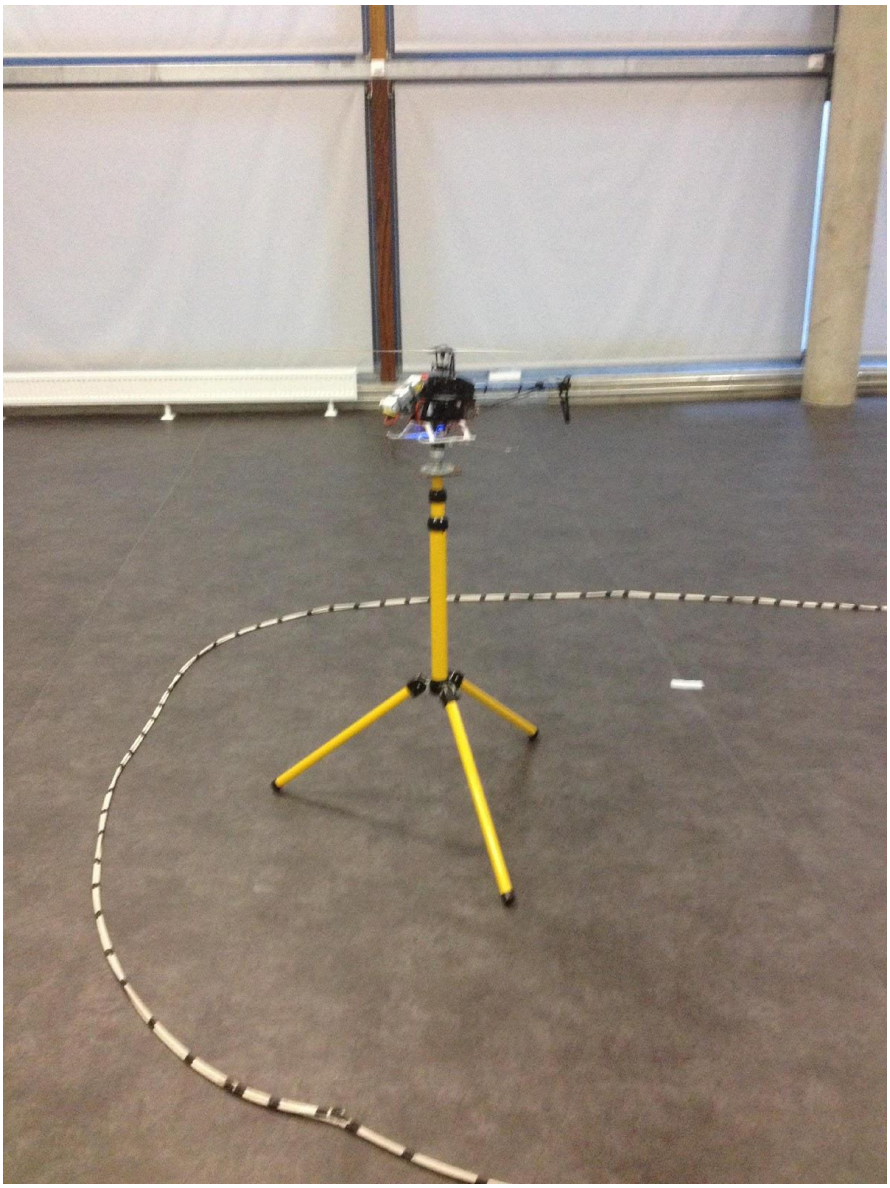

Figura 27. Base de Seguridad para las pruebas.

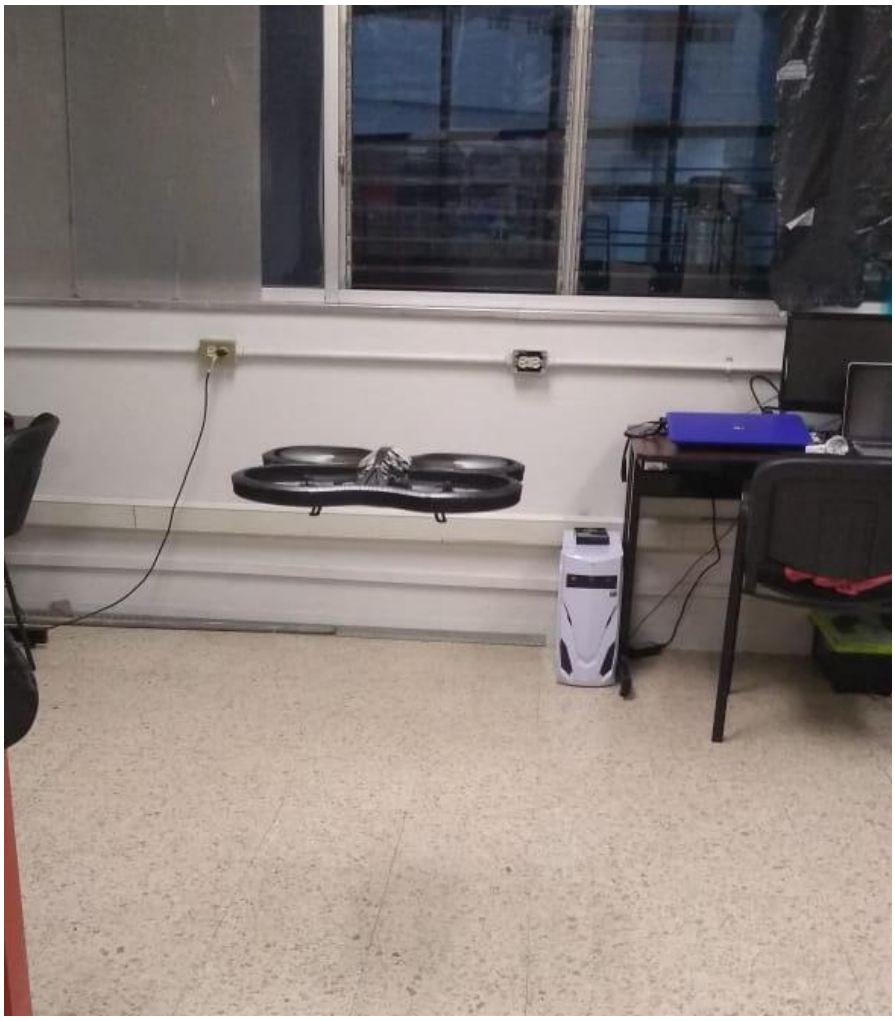

Figura 28. Resultado del control en Yaw para un cudrimotor.

En la figura 28 se muestra como se aplica al control en Yaw a un dron tipo cuatrirotor llamado Parrot. Se pude mostrar su estabilidad en Yaw.

Figura 29. Resultado del control en Yaw para un helicóptero clásico miniatura.

En la figura 28 se aplica el control a un helicóptero miniatura, este se sigue haciendo en un espacio controlado por los riesgos que esto conlleva.

7.0 Conclusiones.

Este proyecto de tesis es la continuación de un proyecto realizado en el 2016 por el profesor Mariano Larios. Aquí se aplicó un control PD básico en una arquitectura tipo dron, que es un helicoperto miniatura clásico, también se realizaron prubas en otro tipo de arquitectura, llamada Cuadrirotor Parrot. Se obtuvieron buenos resultados en ambos drones. La programación se baso en la arquitectura de PIXHAW realizada en C.++. Se aporta una investigación de drones para el acervo de la FCC-BUAP, puesto que es un tema que hoy en día es muy importante para el desarrollo e innovación tecnológico. La programación de arquitecturas como drones es importante, ya que aplicamos nuestro conocimiento de programación en un aspecto de la vida cotidiana y que ayuda a la humanidad en tareas difíciles de realizar y que son de peligro. Partiendo de este trabajo de investigación debemos resaltar los aportes que estamos realizándolo, como lo pueden ser: mayor capacidad de computo en un dispositivo móvil, la interconexión de sistemas, acceso a hardware especializado, se le están dando solución a problemas conocidos partiendo de la combinación de conocimientos, tanto adiquiridos durante el desarrollo asi como los ya establecidos.

Bibliografía.

1. Alessandro Giusti et all. A Machine Learning Approach to Visual Perception of Forest Trails for Mobile Robots. Browse Journals & Magazines IEEE Robotics and Automation. Volume: 1 Issue: 2. December 2015.
2. P. Santana, L. Correia, R. Mendonça, N. Alves and J. Barata, "Tracking natural trails with swarm-based visual saliency", J. Field Rob., vol. 30, no. 1, pp. 64-86, 2013.
3. Aufrere, R., Chapuis, R., & Chausse, F. (2001). A model-driven approach for real-time road recognition. Machine Vision and Applications, 13(2), 95-107.
4. Zavala-Río, A., Fantoni, I., & Lozano, R. (2003). Global stabilization of a PVTOL aircraft model with bounded inputs. International Journal of Control, 76(18), 1833-1844.
5. Duda, R. O., & Hart, P. E. (1972). Use of the Hough transformation to detect lines and curves in pictures. Communications of the ACM, 15(1), 11-15.
6. Latombe, Jean-Claude. Robot Motion Planning. Kluwer Academic Publishers 1991.
7. Carrillo, L. R. G., Flores Colunga, G. R., Sanahuja, G., & Lozano, R. (2014). Quad rotorcraft switching control: An application for the task of path following. Control System s Technology, IEEE Transactions on, 22(4), 1255-1267.
8. Frazzoli, E., Dahleh, M. A., & Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. Journal of Guidance, Control, and Dynamics, 25(1), 116-129.
9. Choset, H. M. (2005). Principles of robot motion: theory, algorithms, and implementation. MIT press. L.E.Dubins, “On curves of minimal length with a constraint on average curvature and with prescribed initial and terminal positions and tangents,” American Journal of Mathematics, vol. 79, no. 3, pp. 497–516, jul 1957.
10. Lugo-Cárdenas, I., Flores, G., Salazar, S., & Lozano, R. (2014, May). Dubins path generation for a fixed wing UAV. In Unmanned Aircraft Systems (ICUAS), 2014 International Conference on (pp. 339-346). IEEE.
11. NVIDIA Inc. Embedded Systems. Jetson TX1 developer kit Full-featured development platform for visual computing.http://www.nvidia.com/object/jetson-tx1-dev-it.html#sthash.xM9kPgjh.dpuf
12. Arellano Vázquez Magali (2015). Estudio de Sistemas Adaptables de Encaminamiento para Sistemas Distribuidos Móviles, Tesis del Posgrado en Ciencias e Ingeniería de la Computación, UNAM.
13. Arellano-Vázquez, M., Benítez-Pérez, H., & Ortega-Arjona, J. (2015). A consensus routing algorithm for mobile distributed systems. International Journal of Distributed Sensor Networks, 11(10), 510707.
14. Audsley, N., & Burns, A. (1990). Real-time system scheduling.
15. Bertuccelli, L., Beckers, W., & Cummings, M. (2010, August). Developing operator models for UAV search scheduling. In AIAA Guidance, Navigation, and Control Conference (p. 7863).
16. Breitbart, J., Weidendorfer, J., & Trinitis, C. (2015, September). Case study on co-scheduling for HPC applications. In 2015 44th International Conference on Parallel Processing Workshops (pp. 277-285). IEEE.

17. Castillo Gutiérrez Octavio Oriol (2015). Diseño de Sistemas de Control Difuso para un Grupo de Sistemas Desacoplados y Análisis de su Estabilidad ante Retardos de Tiempo, Director de Tesis, Posgrado en Ingeniería Eléctrica-Control, UNAM.
18. Cheng Albert M. K. (2003). Real-time systems: scheduling, analysis, and verification. John Wiley & Sons.
19. Corson, M. S., Macker, J., & Batsell, S. G. (1996). Architectural considerations for mobile mesh networking. In Military Communications Conference, 1996. MILCOM'96, Conference Proceedings, IEEE (Vol. 1, pp. 225-229). IEEE.
20. Cortés, J. (2006). Finite-time convergent gradient flows with applications to network consensus. vol. 42, no 1. pp. 1993-2000. Automatica.
21. Cortés, J. (2008). Distributed algorithms for reaching consensus on general functions. 44(3), pp. 726-737. Automatica.
22. Costa, L., & Oliveira, P. (2003). An elitist genetic algorithm for multiobjective optimization. In Metaheuristics: computer decision-making (pp. 217-236). Springer, Boston, MA.
23. Cova, F., & Rincón, P. (2010). El terremoto y tsunami del 27-F y sus efectos en la salud mental. Terapia Psicológica, 28(2), 179-185.
24. DINASED /FGE (2016). Informe de situación Nº71 -Terremoto 7.8 Pedernales» (pdf). Secretaría de Gestión de Riesgos de Ecuador (Quito).
25. Esquivel-Flores, O., Benítez-Pérez, H., & Ortega-Arjona, J. (2012). Issues on Communication Network Control System Based Upon Scheduling Strategy Using Numerical Simulations. INTECH Open Access Publisher.
26. Esquivel Flores Oscar (2013). Estudio de Sistemas Multi-agentes Reconfigurables. Tesis Posgrado en Ciencias e Ingeniería de la Computación, UNAM.
27. Gómez, J. A. H., & Pérez, H. B. (2015). Global Scheduling of Confined Tasks Based on Consensus. IEEE Latin America Transactions, 13(3), 825-834.
28. Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning (adaptive computation and machine learning series).
29. Hatcher, P., Reno, M., Antoniu, G., & Bouge, L. (2005). Cluster computing with Java. Computing in science & engineering, 7(2), 34-39.
30. Hermosillo Gómez José Ángel (2013). Consenso en ambientes no homogéneos en base a una plataforma de tiempo real; Tesis del Posgrado en Ciencia e Ingeniería de la Computación, UNAM.
31. Hong, Y., Chen, G., & Bushnell, L. (2008). Distributed observers design for leader-following control of multi-agent networks. Automatica, 44(3), pp. 846-850.
32. Hui, Q., & Haddad, W. M. (2008). Distributed nonlinear control algorithms for network consensus. 44(9), pp. 2375-2381. Automatica.

33. Khamis Marion, Osorio Claudio. UNESCO (2010). Análisis de riesgos de desastres en chile. Recuperado en: http://www.unesco.org/new/fileadmin/MULTIMEDIA/FIELD/Santiago/pdf/Analisis-de-riesgos-de-desastres-en-Chile.pdf

34. Jakarta (2018). "Central Sulawesi quake, tsunami inflicted US$911 million in losses: Govt". The Jakarta Post.
35. Jeong, S., Simeone, O., & Kang, J. (2018). Mobile edge computing via a UAV-mounted cloudlet: Optimization of bit allocation and path planning. IEEE Transactions on Vehicular Technology, 67(3), 2049-2063.
36. Khazai, B., Daniell, J.E., Wenzel, F. [2011] "The March 2011 Japan Earthquake – Analysis of losses, impacts, and implications for the understanding of risks posed by extreme events", Technikfolgenabschätzung – Theorie und Praxis 20. Jg., Heft 3, November 2011.
37. Khosiawan, Y., Park, Y., Moon, I. et al. (2018) Task scheduling system for UAV operations in indoor environment. Neural Comput & Applic.. https://doi.org/10.1007/s00521-018-3373-9
38. Kim, Y. J., Cho, S. J., Kim, K. J., Hwang, E. H., Yoon, S. H., & Jeon, J. W. (2012, October). Benchmarking Java application using JNI and native C application on Android. In 2012 12th International Conference on Control, Automation and Systems (pp. 284-288). IEEE.
39. Kreemer, C.; Holt W.E. & Haines A.J. (2003). «An integrated global model of present-day plate motions and plate boundary deformation». Geophysical Journal International (Royal Astronomical Society) 154: 8–34.
40. Kshemkalyani, A. D., & Singhal, M. (2011). Distributed computing: principles, algorithms, and systems. Capítulo 14. Cambridge University Press.
41. Lim, G. J., Kim, S., Cho, J., Gong, Y., & Khodaei, A. (2018). Multi-UAV pre-positioning and routing for power network damage assessment. IEEE Transactions on Smart Grid, 9(4), 3643-3651.
42. Larios Gómez M., Hernández Beristain A., Martínez Mirón E. (2017). Scheduling: A graphical interface for applying a process scheduling algorithm. Research in Computing Science. Vol. 145, pp. 119-125.
43. Larios Gómez M., Migliolo Carrera J., Anzures García M., Aldama Díaz A. (2018). A scheduling algorithm for a platform in real time. CCIS 948 9th international supercomputing conference in Mexico ISUM 2018. Springer International Publishing AG. DOI: https://doi.org/10.1007/978-3-030-10448-1_1
44. Larios-Gómez M., Carmona-Flores M. E., Placido-Velazco C.E. (2019). JPeer: Framework for real-time planning on distributed mobile environments. 10th international supercomputing conference in Mexico ISUM 20119. Springer International.
45. Lee, S., & Jeon, J. W. (2010, October). Evaluating performance of Android platform using native C for embedded systems. In ICCAS 2010 (pp. 1160-1163). IEEE.
46. Lin, C. M., Lin, J. H., Dow, C. R., & Wen, C. M. (2011, December). Benchmark dalvik and native code for android system. In 2011 Second International Conference on Innovations in Bio-inspired Computing and Applications (pp. 320-323). IEEE.
47. LNS (2017). Laboratorio Nacional de Supercómputo del Sureste de México Boulevard Valsequillo y Av. las Torres, Universitaria, C.P. 72570, Teléfono

+52(222) 2295500 ext. 5119. http://lns.org.mx/?q=content/proyectos-aceptados-primavera-2017
48. LNS (2018). Laboratorio Nacional de Supercómputo del Sureste de México Boulevard Valsequillo y Av. las Torres, Universitaria, C.P. 72570,Teléfono +52(222) 2295500 ext. 5119. http://lns.org.mx/?q=content/proyectos-aceptados
49. LNS (2019). Laboratorio Nacional de Supercómputo del Sureste de México Boulevard Valsequillo y Av. las Torres, Ciudad Universitaria, C.P. 72570, Teléfono +52(222) 2295500 ext. 5119. http://lns.org.mx/?q=content/proyectos-aceptados-primavera-2019
50. Méndez Monroy Erick (2012). Estudios de Sistemas de Control por Red Con Retardos, Tesis del Posgrado en Ingeniería, Campo de Conocimiento de Eléctrica, UNAM.
51. Montresor, A., & Jelasity, M. (2009, September). PeerSim: A scalable P2P simulator. In 2009 IEEE Ninth International Conference on Peer-to-Peer Computing (pp. 99-100). IEEE.
52. Mulchandani, Deepak (1998). "Java for embedded systems." IEEE Internet Computing 3. pp 30-39.
53. Nouiri, M., Bekrar, A., Jemai, A., Niar, S., & Ammari, A. C. (2018). An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. Journal of Intelligent Manufacturing, 29(3), 603-615
54. Park, V. D., & Corson, M. S. (1997). A highly adaptive distributed routing algorithm for mobile wireless networks. In INFOCOM'97. Sixteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Driving the Information Revolution. Proceedings IEEE (Vol. 3, pp. 1405-1413). IEEE.
55. Park Y., Khosiawan Y., Moon I., Janardhanan M.N., Nielsen I. (2016) Scheduling System for Multiple Unmanned Aerial Vehicles in Indoor Environments Using the CSP Approach. In: Czarnowski I., Caballero A., Howlett R., Jain L. (eds) Intelligent Decision Technologies 2016. IDT 2016. Smart Innovation, Systems and Technologies, vol 56. Springer, Cham
56. Pearl, J. (2014). Probabilistic reasoning in intelligent systems: networks of plausible inference. Morgan Kaufmann.
57. Ramasubramanian, V., Haas, Z. J., & Sirer, E. G. (2003). SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks. In Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing (pp. 303-314). ACM.
58. Sengul, C., & Kravets, R. (2006). Bypass routing: An on-demand local recovery protocol for ad hoc networks. Ad Hoc Networks, 4(3), 380-397.
59. Schmidt, G. (2000). Scheduling with limited machine availability. European Journal of Operational Research, 121(1), 1-15.
60. Stankovic, J. A. (1985). An Application of Bayesian Decision Theory to. IEEE Transactions on Computers, 100(34).
61. Stankovic, J. A., Spuri, M., Di Natale, M., & Buttazzo, G. C. (1995). Implications of classical scheduling results for real-time systems. Computer, 28(6), 16-25.

62. SNU (2017). Este reporte es elaborado por la Oficina del Coordinador Residente en México, en colaboración con el SNU y socios humanitarios con presencia en el país y en base a la información disponible. https://www.paho.org/disasters/index.php?option=com_docman&view=download&category_slug=mexico-earthquake-september-2017-1232&alias=2531-terremoto-mexico-informe-situacion-1-13-setiembre-2017-coordinador-residente-onu-531&Itemid=1179&lang=en
63. USGS. Astonishing Wave Heights. Among the Findings of an International Tsunami Survey Team on Sumatra". U.S. Geological Survey. Retrieved 16 June 2016.
64. Tanenbaum, A. S. (2016). Lessons learned from 30 years of MINIX. Communications of the ACM, 59(3), 70-78.
65. Vlachou A. et al. (2012). Peer-to-Peer Query Processing over Multidimensional Data. In Computer Science, DOI 10.1007/978-1-4614, SpringerBriefs.
66. Vu Q.H., Lupu M., Ooi B.C. (2010) Architecture of Peer-to-Peer Systems. In: Peer-to-Peer Computing. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-03514-2_2. ISBN978-3-642-03513-5.
67. Wu, Q., & Zhang, R. (2018). Common throughput maximization in UAV-enabled OFDMA systems with delay consideration. Available online: arxiv. org/abs/1801.00444.
68. Zavala-Río, A., Fantoni, I., & Lozano, R. (2003). Global stabilization of a PVTOL aircraft model with bounded inputs. International Journal of Control, 76(18), 1833-1844.

Anexo A. Código Attitud manual.

```
#include <nuttx/config.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>
#include <unistd.h>
#include <fcntl.h>
#include <errno.h>
#include <debug.h>
#include <getopt.h>
#include <time.h>
#include <math.h>
#include <poll.h>
#include <sys/prctl.h>
#include <drivers/drv_hrt.h>
#include <uORB/uORB.h>
#include <drivers/drv_gyro.h>
#include <uORB/topics/vehicle_control_mode.h>
#include <uORB/topics/vehicle_attitude.h>
#include <uORB/topics/vehicle_attitude_setpoint.h>
#include <uORB/topics/manual_control_setpoint.h>
#include <uORB/topics/offboard_control_setpoint.h>
#include <uORB/topics/vehicle_rates_setpoint.h>
#include <uORB/topics/vehicle_status.h>
#include <uORB/topics/sensor_combined.h>
#include <uORB/topics/actuator_controls.h>
#include <uORB/topics/parameter_update.h>

#include <systemlib/perf_counter.h>
#include <systemlib/systemlib.h>
```

```
#include <systemlib/param/param.h>

#include "multirotor_attitude_control.h"
#include "multirotor_rate_control.h"

__EXPORT int multirotor_att_control_main(int argc, char *argv[]);

static bool thread_should_exit;
static int mc_task;
static bool motor_test_mode = false;
static const float min_takeoff_throttle = 0.2f;
static const float yaw_deadzone = 0.01f;

static int mc_thread_main(int argc, char *argv[])
{
        /* declare and safely initialize all structs */
        struct vehicle_attitude_s att;
        memset(&att, 0, sizeof(att));
        struct vehicle_attitude_setpoint_s att_sp;
        memset(&att_sp, 0, sizeof(att_sp));
        struct offboard_control_setpoint_s offboard_sp;
        memset(&offboard_sp, 0, sizeof(offboard_sp));
        struct vehicle_control_mode_s control_mode;
        memset(&control_mode, 0, sizeof(control_mode));
        struct manual_control_setpoint_s manual;
        memset(&manual, 0, sizeof(manual));
        struct sensor_combined_s sensor;
        memset(&sensor, 0, sizeof(sensor));
        struct vehicle_rates_setpoint_s rates_sp;
        memset(&rates_sp, 0, sizeof(rates_sp));
        struct vehicle_status_s status;
```

```
	memset(&status, 0, sizeof(status));
	struct actuator_controls_s actuators;
	memset(&actuators, 0, sizeof(actuators));

	/* subscribe */
	int vehicle_attitude_sub = orb_subscribe(ORB_ID(vehicle_attitude));
	int parameter_update_sub = orb_subscribe(ORB_ID(parameter_update));
	int vehicle_attitude_setpoint_sub = orb_subscribe(ORB_ID(vehicle_attitude_setpoint));
	int offboard_control_setpoint_sub = orb_subscribe(ORB_ID(offboard_control_setpoint));
	int vehicle_control_mode_sub = orb_subscribe(ORB_ID(vehicle_control_mode));
	int manual_control_setpoint_sub = orb_subscribe(ORB_ID(manual_control_setpoint));
	int sensor_combined_sub = orb_subscribe(ORB_ID(sensor_combined));
	int vehicle_rates_setpoint_sub = orb_subscribe(ORB_ID(vehicle_rates_setpoint));
	int vehicle_status_sub = orb_subscribe(ORB_ID(vehicle_status));

	/* publish actuator controls */
	for (unsigned i = 0; i < NUM_ACTUATOR_CONTROLS; i++) {
		actuators.control[i] = 0.0f;
	}

	orb_advert_t actuator_pub = orb_advertise(ORB_ID_VEHICLE_ATTITUDE_CONTROLS, &actuators);
	orb_advert_t att_sp_pub = orb_advertise(ORB_ID(vehicle_attitude_setpoint), &att_sp);
	orb_advert_t rates_sp_pub = orb_advertise(ORB_ID(vehicle_rates_setpoint), &rates_sp);

	/* register the perf counter */
	perf_counter_t mc_loop_perf = perf_alloc(PC_ELAPSED, "multirotor_att_control_runtime");
```

```
	perf_counter_t mc_interval_perf = perf_alloc(PC_INTERVAL, "multirotor_att_control_interval");
	perf_counter_t mc_err_perf = perf_alloc(PC_COUNT, "multirotor_att_control_err");

	warnx("starting");

	/* store last control mode to detect mode switches */
	bool control_yaw_position = true;
	bool reset_yaw_sp = true;

	struct pollfd fds[1] = {
		{ .fd = vehicle_attitude_sub, .events = POLLIN },
	};

	while (!thread_should_exit) {

		/* wait for a sensor update, check for exit condition every 500 ms */
		int ret = poll(fds, 1, 500);

		if (ret < 0) {
			/* poll error, count it in perf */
			perf_count(mc_err_perf);

		} else if (ret > 0) {
			/* only run controller if attitude changed */
			perf_begin(mc_loop_perf);

			/* attitude */
			orb_copy(ORB_ID(vehicle_attitude), vehicle_attitude_sub, &att);

			bool updated;
```

```
                /* parameters */
                orb_check(parameter_update_sub, &updated);

                if (updated) {
                        struct parameter_update_s update;
                        orb_copy(ORB_ID(parameter_update),
parameter_update_sub, &update);
                        /* update parameters */
                }

                /* control mode */
                orb_check(vehicle_control_mode_sub, &updated);

                if (updated) {
                        orb_copy(ORB_ID(vehicle_control_mode),
vehicle_control_mode_sub, &control_mode);
                }

                /* manual control setpoint */
                orb_check(manual_control_setpoint_sub, &updated);

                if (updated) {
                        orb_copy(ORB_ID(manual_control_setpoint),
manual_control_setpoint_sub, &manual);
                }

                /* attitude setpoint */
                orb_check(vehicle_attitude_setpoint_sub, &updated);

                if (updated) {
```

```
                    orb_copy(ORB_ID(vehicle_attitude_setpoint),
vehicle_attitude_setpoint_sub, &att_sp);
                }

                /* offboard control setpoint */
                orb_check(offboard_control_setpoint_sub, &updated);

                if (updated) {
                    orb_copy(ORB_ID(offboard_control_setpoint),
offboard_control_setpoint_sub, &offboard_sp);
                }

                /* vehicle status */
                orb_check(vehicle_status_sub, &updated);

                if (updated) {
                    orb_copy(ORB_ID(vehicle_status), vehicle_status_sub,
&status);
                }

                /* sensors */
                orb_check(sensor_combined_sub, &updated);

                if (updated) {
                    orb_copy(ORB_ID(sensor_combined), sensor_combined_sub,
&sensor);
                }

                /* set flag to safe value */
                control_yaw_position = true;

                /* reset yaw setpoint if not armed */
```

```
                if (!control_mode.flag_armed) {
                        reset_yaw_sp = true;
                }

                /* define which input is the dominating control input */
                if (control_mode.flag_control_offboard_enabled) {
                        /* offboard inputs */
                        if (offboard_sp.mode == OFFBOARD_CONTROL_MODE_DIRECT_RATES) {
                                rates_sp.roll = offboard_sp.p1;
                                rates_sp.pitch = offboard_sp.p2;
                                rates_sp.yaw = offboard_sp.p3;
                                rates_sp.thrust = offboard_sp.p4;
                                rates_sp.timestamp = hrt_absolute_time();
                                orb_publish(ORB_ID(vehicle_rates_setpoint), rates_sp_pub, &rates_sp);

                        } else if (offboard_sp.mode == OFFBOARD_CONTROL_MODE_DIRECT_ATTITUDE) {
                                att_sp.roll_body = offboard_sp.p1;
                                att_sp.pitch_body = offboard_sp.p2;
                                att_sp.yaw_body = offboard_sp.p3;
                                att_sp.thrust = offboard_sp.p4;
                                att_sp.timestamp = hrt_absolute_time();
                                /* publish the result to the vehicle actuators */
                                orb_publish(ORB_ID(vehicle_attitude_setpoint), att_sp_pub, &att_sp);
                        }

                        /* reset yaw setpoint after offboard control */
                        reset_yaw_sp = true;
```

```
                } else if (control_mode.flag_control_manual_enabled) {
                        /* manual input */
                        if (control_mode.flag_control_attitude_enabled) {
                                /* control attitude, update attitude setpoint depending on mode */
                                if (att_sp.thrust < 0.1f) {
                                        /* no thrust, don't try to control yaw */
                                        rates_sp.yaw = 0.0f;
                                        control_yaw_position = false;

                                        if (status.condition_landed) {
                                                /* reset yaw setpoint if on ground */
                                                reset_yaw_sp = true;
                                        }

                                } else {
                                        /* only move yaw setpoint if manual input is != 0 */
                                        if (manual.yaw < -yaw_deadzone || yaw_deadzone < manual.yaw) {
                                                /* control yaw rate */
                                                control_yaw_position = false;
                                                rates_sp.yaw = manual.yaw;
                                                reset_yaw_sp = true;  // has no effect on control, just for beautiful log

                                        } else {
                                                control_yaw_position = true;
                                        }
                                }

                                if (!control_mode.flag_control_velocity_enabled) {
```

```
                                        /* update attitude setpoint if not in position
control mode */
                                        att_sp.roll_body = manual.roll;
                                        att_sp.pitch_body = manual.pitch;

                                        if
(!control_mode.flag_control_climb_rate_enabled) {
                                                /* pass throttle directly if not in altitude
control mode */
                                                att_sp.thrust = manual.throttle;
                                        }
                                }

                                /* reset yaw setpoint to current position if needed */
                                if (reset_yaw_sp) {
                                        att_sp.yaw_body = att.yaw;
                                        reset_yaw_sp = false;
                                }

                                if (motor_test_mode) {
                                        printf("testmode");
                                        att_sp.roll_body = 0.0f;
                                        att_sp.pitch_body = 0.0f;
                                        att_sp.yaw_body = 0.0f;
                                        att_sp.thrust = 0.1f;
                                }

                                att_sp.timestamp = hrt_absolute_time();

                                /* publish the attitude setpoint */
                                orb_publish(ORB_ID(vehicle_attitude_setpoint),
att_sp_pub, &att_sp);
```

```
                } else {
                        /* manual rate inputs (ACRO), from RC control or joystick */
                        if (control_mode.flag_control_rates_enabled) {
                                rates_sp.roll = manual.roll;
                                rates_sp.pitch = manual.pitch;
                                rates_sp.yaw = manual.yaw;
                                rates_sp.thrust = manual.throttle;
                                rates_sp.timestamp = hrt_absolute_time();
                        }

                        /* reset yaw setpoint after ACRO */
                        reset_yaw_sp = true;
                }

        } else {
                if (!control_mode.flag_control_auto_enabled) {
                        /* no control, try to stay on place */
                        if (!control_mode.flag_control_velocity_enabled) {
                                /* no velocity control, reset attitude setpoint */
                                att_sp.roll_body = 0.0f;
                                att_sp.pitch_body = 0.0f;
                                att_sp.timestamp = hrt_absolute_time();

orb_publish(ORB_ID(vehicle_attitude_setpoint), att_sp_pub, &att_sp);
                        }
                }

                /* reset yaw setpoint after non-manual control */
                reset_yaw_sp = true;
        }
```

```
                /* check if we should we reset integrals */
                bool reset_integral = !control_mode.flag_armed || att_sp.thrust < 0.1f;
        // TODO use landed status instead of throttle

                /* run attitude controller if needed */
                if (control_mode.flag_control_attitude_enabled) {
                        multirotor_control_attitude(&att_sp, &att, &rates_sp,
control_yaw_position, reset_integral);
                        orb_publish(ORB_ID(vehicle_rates_setpoint), rates_sp_pub,
&rates_sp);
                }

                /* measure in what intervals the controller runs */
                perf_count(mc_interval_perf);

                /* run rates controller if needed */
                if (control_mode.flag_control_rates_enabled) {
                        /* get current rate setpoint */
                        bool rates_sp_updated = false;
                        orb_check(vehicle_rates_setpoint_sub, &rates_sp_updated);

                        if (rates_sp_updated) {
                                orb_copy(ORB_ID(vehicle_rates_setpoint),
vehicle_rates_setpoint_sub, &rates_sp);
                        }

                        /* apply controller */
                        float rates[3];
                        rates[0] = att.rollspeed;
                        rates[1] = att.pitchspeed;
                        rates[2] = att.yawspeed;
```

```
                                multirotor_control_rates(&rates_sp,      rates,      &actuators,
reset_integral);
//          rates[3] = manual.throttle;

                        } else {
                                /* rates controller disabled, set actuators to zero for safety */
                                actuators.control[0] = 0.0f;
                                actuators.control[1] = 0.0f;
                                actuators.control[2] = 0.0f;
                                actuators.control[3] = 0.0f;
                        }

                        /* fill in manual control values */
// add throttle to more control
//                      MARINANO LARIOS G ----------------------------------------------------
                        actuators.control[3] = manual.throttle;
                        actuators.control[4] = manual.aux1;
                        //actuators.control[5] = manual.assisted_switch;
                        //actuators.control[6] = manual.mission_switch;
//---------------------------------------------------------------------
                        actuators.control[7] = manual.aux3;

                        actuators.timestamp = hrt_absolute_time();
                        orb_publish(ORB_ID_VEHICLE_ATTITUDE_CONTROLS,
actuator_pub, &actuators);

                        perf_end(mc_loop_perf);
                }
        }

        warnx("stopping, disarming motors");
```

```
	/* kill all outputs */
	for (unsigned i = 0; i < NUM_ACTUATOR_CONTROLS; i++)
		actuators.control[i] = 0.0f;

	orb_publish(ORB_ID_VEHICLE_ATTITUDE_CONTROLS, actuator_pub, &actuators);

	close(vehicle_attitude_sub);
	close(vehicle_control_mode_sub);
	close(manual_control_setpoint_sub);
	close(actuator_pub);
	close(att_sp_pub);

	perf_print_counter(mc_loop_perf);
	perf_free(mc_loop_perf);

	fflush(stdout);
	exit(0);
}

static void
usage(const char *reason)
{
	if (reason)
		fprintf(stderr, "%s\n", reason);

	fprintf(stderr, "usage: multirotor_att_control [-m <mode>] [-t] {start|status|stop}\n");
	fprintf(stderr, "    <mode> is 'rates' or 'attitude'\n");
	fprintf(stderr, "    -t enables motor test mode with 10%% thrust\n");
	exit(1);
}
```

```
int multirotor_att_control_main(int argc, char *argv[])
{
	int	ch;
	unsigned int optioncount = 0;

	while ((ch = getopt(argc, argv, "tm:")) != EOF) {
		switch (ch) {
		case 't':
			motor_test_mode = true;
			optioncount += 1;
			break;

		case ':':
			usage("missing parameter");
			break;

		default:
			fprintf(stderr, "option: -%c\n", ch);
			usage("unrecognized option");
			break;
		}
	}

	argc -= optioncount;
	//argv += optioncount;

	if (argc < 1)
		usage("missing command");

	if (!strcmp(argv[1 + optioncount], "start")) {
```

```
                thread_should_exit = false;
                mc_task = task_spawn_cmd("multirotor_att_control",
                                         SCHED_DEFAULT,
                                         SCHED_PRIORITY_MAX - 15,
                                         2048,
                                         mc_thread_main,
                                         NULL);
                exit(0);
        }

        if (!strcmp(argv[1 + optioncount], "stop")) {
                thread_should_exit = true;
                exit(0);
        }

        usage("unrecognized command");
        exit(1);
}
```

Anexo B. Código Altitud Dron.

```
#include "multirotor_attitude_control.h"
#include <stdio.h>
#include <stdlib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
#include <float.h>
#include <math.h>
#include <systemlib/pid/pid.h>
#include <systemlib/param/param.h>
#include <drivers/drv_hrt.h>

PARAM_DEFINE_FLOAT(MC_YAWPOS_P, 2.0f);
PARAM_DEFINE_FLOAT(MC_YAWPOS_I, 0.15f);
PARAM_DEFINE_FLOAT(MC_YAWPOS_D, 0.0f);
//PARAM_DEFINE_FLOAT(MC_YAWPOS_AWU, 1.0f);
PARAM_DEFINE_FLOAT(MC_YAWPOS_LIM, 3.0f);
//PARAM_DEFINE_FLOAT(MC_ATT_P, 6.8f);
//PARAM_DEFINE_FLOAT(MC_ATT_I, 0.0f);
//PARAM_DEFINE_FLOAT(MC_ATT_D, 0.0f);

PARAM_DEFINE_FLOAT(MC_ROLLPOS_P, 6.8f);
PARAM_DEFINE_FLOAT(MC_ROLLPOS_I, 0.0f);
PARAM_DEFINE_FLOAT(MC_ROLLPOS_D, 0.0f);

PARAM_DEFINE_FLOAT(MC_PITCHPOS_P, 6.8f);
PARAM_DEFINE_FLOAT(MC_PITCHPOS_I, 0.0f);
PARAM_DEFINE_FLOAT(MC_PITCHPOS_D, 0.0f);

//PARAM_DEFINE_FLOAT(MC_ATT_AWU, 0.05f);
```

```
//PARAM_DEFINE_FLOAT(MC_ATT_LIM, 0.4f);

//PARAM_DEFINE_FLOAT(MC_ATT_XOFF, 0.0f);
//PARAM_DEFINE_FLOAT(MC_ATT_YOFF, 0.0f);

struct mc_att_control_params {
        float yaw_p;
        float yaw_i;
        float yaw_d;
        //float yaw_awu;
        //float yaw_lim;

//      float att_p;
//      float att_i;
//      float att_d;

        float roll_p;
        float roll_i;
        float roll_d;

        float pitch_p;
        float pitch_i;
        float pitch_d;
        //float att_awu;
        //float att_lim;

        //float att_xoff;
        //float att_yoff;
// Prueba
        float yaw_rate_max;
};
```

```
struct mc_att_control_param_handles {
		param_t yaw_p;
		param_t yaw_i;
		param_t yaw_d;
		//param_t yaw_awu;
		param_t yaw_lim;

//		param_t att_p;
//		param_t att_i;
//		param_t att_d;

		param_t roll_p;
		param_t roll_i;
		param_t roll_d;

		param_t pitch_p;
		param_t pitch_i;
		param_t pitch_d;

		//param_t att_awu;
		//param_t att_lim;

		//param_t att_xoff;
		//param_t att_yoff;
//Prueba
//		param_t yaw_rate_max;
};

/**
 * Initialize all parameter handles and values
```

```
 *
 */
static int parameters_init(struct mc_att_control_param_handles *h);

/**
 * Update all parameters
 *
 */
static int parameters_update(const struct mc_att_control_param_handles *h, struct mc_att_control_params *p);


static int parameters_init(struct mc_att_control_param_handles *h)
{
	/* PID parameters */
	h->yaw_p	=	param_find("MC_YAWPOS_P");
	h->yaw_i	=	param_find("MC_YAWPOS_I");
	h->yaw_d	=	param_find("MC_YAWPOS_D");
	//h->yaw_awu		=	param_find("MC_YAWPOS_AWU");
	h->yaw_lim	=	param_find("MC_YAWPOS_LIM");

//	h->att_p	=	param_find("MC_ATT_P");
//	h->att_i	=	param_find("MC_ATT_I");
//	h->att_d	=	param_find("MC_ATT_D");

	h->roll_p	=	param_find("MC_ROLLPOS_P");
	h->roll_i	=	param_find("MC_ROLLPOS_I");
	h->roll_d	=	param_find("MC_ROLLPOS_D");

	h->pitch_p	=	param_find("MC_PITCHPOS_P");
	h->pitch_i	=	param_find("MC_PITCHPOS_I");
```

```
	h->pitch_d	=	param_find("MC_PITCHPOS_D");

	//h->att_awu	=	param_find("MC_ATT_AWU");
	//h->att_lim	=	param_find("MC_ATT_LIM");

	//h->att_xoff	=	param_find("MC_ATT_XOFF");
	//h->att_yoff	=	param_find("MC_ATT_YOFF");
//Prueba
//	h->yaw_rate_max = param_find("MC_YAW_RATE_MAX");

	return OK;
}

static int parameters_update(const struct mc_att_control_param_handles *h, struct mc_att_control_params *p)
{
	param_get(h->yaw_p, &(p->yaw_p));
	param_get(h->yaw_i, &(p->yaw_i));
	param_get(h->yaw_d, &(p->yaw_d));
	//param_get(h->yaw_awu, &(p->yaw_awu));
	//param_get(h->yaw_lim, &(p->yaw_lim));

//	param_get(h->att_p, &(p->att_p));
//	param_get(h->att_i, &(p->att_i));
//	param_get(h->att_d, &(p->att_d));

	param_get(h->roll_p, &(p->roll_p));
	param_get(h->roll_i, &(p->roll_i));
	param_get(h->roll_d, &(p->roll_d));

	param_get(h->pitch_p, &(p->pitch_p));
```

```
	param_get(h->pitch_i, &(p->pitch_i));
	param_get(h->pitch_d, &(p->pitch_d));
	//param_get(h->att_awu, &(p->att_awu));
	//param_get(h->att_lim, &(p->att_lim));

	//param_get(h->att_xoff, &(p->att_xoff));
	//param_get(h->att_yoff, &(p->att_yoff));
// prueba
//	param_get(h->yaw_rate_max, p->yaw_rate_max);
	return OK;
}

void multirotor_control_attitude(const struct vehicle_attitude_setpoint_s *att_sp,
				 const struct vehicle_attitude_s *att, struct vehicle_rates_setpoint_s *rates_sp, bool control_yaw_position, bool reset_integral)
{
	static uint64_t last_run = 0;
	static uint64_t last_input = 0;
	float deltaT = (hrt_absolute_time() - last_run) / 1000000.0f;
	last_run = hrt_absolute_time();

	if (last_input != att_sp->timestamp) {
		last_input = att_sp->timestamp;
	}

	static int motor_skip_counter = 0;

	static PID_t pitch_controller;
	static PID_t roll_controller;
//prueba.-----------------------------------------------------------------------
	static PID_t yaw_controller;
```

```
	static struct mc_att_control_params p;
	static struct mc_att_control_param_handles h;

	static bool initialized = false;

	static float yaw_error;

	/* initialize the pid controllers when the function is called for the first time */
	if (initialized == false) {
		parameters_init(&h);
		parameters_update(&h, &p);

		pid_init(&pitch_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f, PID_MODE_DERIVATIV_SET, 0.0f);
		pid_init(&roll_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f, PID_MODE_DERIVATIV_SET, 0.0f);
//Prueba.-------------------------------------------------------------------------
		pid_init(&yaw_controller, p.yaw_p, p.yaw_i, p.yaw_d, 1000.0f, 1000.0f, PID_MODE_DERIVATIV_SET, 0.0f);

		initialized = true;
	}

	/* load new parameters with lower rate */
	if (motor_skip_counter % 500 == 0) {
		/* update parameters from storage */
		parameters_update(&h, &p);

		/* apply parameters */
		pid_set_parameters(&pitch_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f, 1000.0f);
```

```
                pid_set_parameters(&roll_controller, p.pitch_p, p.pitch_i, p.pitch_d, 1000.0f,
1000.0f);
//Prueba.-------------------------------------------------------------------
                pid_set_parameters(&yaw_controller, p.yaw_p, p.yaw_i, p.yaw_d, 1000.0f,
1000.0f);
        }

        /* reset integrals if needed */
        if (reset_integral) {
                pid_reset_integral(&pitch_controller);
                pid_reset_integral(&roll_controller);
//Prueba.-------------------------------------------------------------------
                pid_reset_integral(&yaw_controller);
        }

        /* calculate current control outputs */

        /* control pitch (forward) output */
/*
        Modificacion de pitch con respecto al mixer. resultado por -1
        MARINANO LARIOS G ---------------------------------------------------
        pid_calculate(&pitch_controller,  att_sp->pitch_body  ,att->pitch,  att->pitchspeed,
deltaT);
*/
        rates_sp->pitch = -pid_calculate(&pitch_controller, att_sp->pitch_body ,
                                        att->pitch, att->pitchspeed, deltaT);

        /* control roll (left/right) output */
        rates_sp->roll = pid_calculate(&roll_controller, att_sp->roll_body ,
                                       att->roll, att->rollspeed, deltaT);

        if (control_yaw_position) {
```

```
                /* control yaw rate */// TODO use pid lib
                /* positive error: rotate to right, negative error, rotate to left (NED frame) */
                // yaw_error = _wrap_pi(att_sp->yaw_body - att->yaw);

                yaw_error = att_sp->yaw_body - att->yaw;

                if (yaw_error > M_PI_F) {
                        yaw_error -= M_TWOPI_F;

                } else if (yaw_error < -M_PI_F) {
                        yaw_error += M_TWOPI_F;
                }
// Cambio de direccion en control de attidtude para yaw
//              MARINANO LARIOS G ----------------------------------------------------
//              rates_sp->yaw = p.yaw_p * (yaw_error) - (p.yaw_d * att->yawspeed);
//              rates_sp->yaw = -(p.yaw_p * (yaw_error) - (p.yaw_d * att->yawspeed));
                rates_sp->yaw = -pid_calculate(&yaw_controller, att_sp->yaw_body , att->yaw, att->yawspeed, deltaT);
//
//              if((float)fabs(rates_sp->yaw)>yaw_lim){
//              rates_sp->yaw=constrain(rates_sp->yaw,-yaw_rate_max,yaw_rate_max);
//              }

        }
        rates_sp->thrust = att_sp->thrust;
   //need to update the timestamp now that we've touched rates_sp
   rates_sp->timestamp = hrt_absolute_time();

        motor_skip_counter++;
}
```

Anexo C. Código Manejo del tunning en control manual.

```
#include <nuttx/config.h>
#include <nuttx/sched.h>
#include <unistd.h>
#include <stdio.h>
#include <termios.h>
*/
#include <nuttx/config.h>
#include <unistd.h>
#include <pthread.h>
#include <stdio.h>
#include <math.h>
#include <stdbool.h>
#include <fcntl.h>
#include <mqueue.h>
#include <string.h>
#include <drivers/drv_hrt.h>
#include <time.h>
#include <float.h>
#include <unistd.h>
#include <nuttx/sched.h>
#include <sys/prctl.h>
#include <termios.h>
#include <errno.h>
#include <stdlib.h>
#include <poll.h>
#include <mavlink/mavlink_log.h>


#include <systemlib/systemlib.h>
#include <systemlib/err.h>
```

```
#include <uORB/uORB.h>
#include <uORB/topics/rc_channels.h>
#include <uORB/topics/parameter_update.h>
//#include <uORB/topics/debug_controller.h>

#include "tunning.h"
#include "tunning_params.h"
//#include "../mavlink/mavlink_parameters.h"

static bool thread_should_exit = false;		/**< daemon exit flag */
static bool thread_running = false;		/**< daemon status flag */
static int tunning_task;				/**< Handle of daemon task / thread */


/* protocol interface */
#define MAX_PACKET_LEN 100
static int uart;
//static int baudrate;
//static uint8_t msg_buf[MAX_PACKET_LEN];



static void usage(const char *reason)
{
	if (reason)
		warnx("%s\n", reason);
	errx(1, "usage: mav3dsim {start|stop|status} [-p <additional params>]\n\n");
}
```

```
/**
 * The daemon app only briefly exists to start
 * the background job. The stack size assigned in the
 * Makefile does only apply to this management task.
 *
 * The actual stack size should be set in the call
 * to task_create().
 */
int tunning_main(int argc, char *argv[])
{
	if (argc < 1)
		usage("missing command");

	if (!strcmp(argv[1], "start")) {

		if (thread_running) {
			warnx("daemon already running\n");
			/* this is not an error */
			exit(0);
		}

		thread_should_exit = false;
		tunning_task = task_spawn_cmd("tunning",
					      SCHED_DEFAULT,
					      SCHED_PRIORITY_DEFAULT,
					      2048,
					      tunning_thread_main,
					      (argv) ? (const char **)&argv[2] : (const char **)NULL);
		if(tunning_task<0)
		{
```

```
                warnx("Error running task\n");
        }

        exit(0);
    }

    if (!strcmp(argv[1], "stop")) {
        thread_should_exit = true;
        exit(0);
    }

    if (!strcmp(argv[1], "status")) {
        if (thread_running) {
            warnx("\trunning\n");
        } else {
            warnx("\tnot started\n");
        }
        exit(0);
    }

    usage("unrecognized command");
    exit(1);
}


/**
 * Tunning main function.
 */
int tunning_thread_main(int argc, char *argv[])
{
```

```
warnx("Tunning started\n");

static int mavlink_fd;
mavlink_fd = open(MAVLINK_LOG_DEVICE, 0);
mavlink_log_info(mavlink_fd,"[tun] started");
thread_running = true;

param_t param = param_find("TN_ENABLE");
int32_t new_param =0;
param_set(param, &(new_param));

/* subscribe to sensor_combined topic */

int parameter_update_sub = orb_subscribe(ORB_ID(parameter_update));
int rc_sub = orb_subscribe(ORB_ID(rc_channels));

int freq = 200;

orb_set_interval(rc_sub, freq);
orb_set_interval(parameter_update_sub, freq);

/* parameters init*/
struct tunning_params params;
struct tunning_param_handles param_handles;
parameters_init(&param_handles);
parameters_update(&param_handles, &params);

//struct debug_controller_s debug_controller;
//memset(&debug_controller, 0, sizeof(debug_controller));
```

```
	//orb_advert_t debug_controller_pub = orb_advertise(ORB_ID(debug_controller), &debug_controller);

	/* one could wait for multiple topics with this technique, just using one here */
	struct pollfd fds[] = {
			{ .fd = rc_sub,   .events = POLLIN },
			{ .fd = parameter_update_sub,   .events = POLLIN },
		/* there could be more file descriptors here, in the form like:
		 * { .fd = other_sub_fd,   .events = POLLIN },
		 */
	};

	int error_counter = 0;
	float old_param =0;
	while (!thread_should_exit) {

		/* obtained data for the first file descriptor */
		struct rc_channels_s rc_raw;

			/* wait for sensor update of 2 file descriptor for 1000 ms (1 second) */
				int poll_ret = poll(fds, 2, freq);

				/* handle the poll result */
				if (poll_ret == 0) {
					/* this means none of our providers is giving us data */
					warnx("Got no data within a second\n");
				} else if (poll_ret < 0) {
					/* this is seriously bad - should be an emergency */
					if (error_counter < 10 || error_counter % 50 == 0) {
```

```
                                        /* use a counter to prevent flooding (and
slowing us down) */
                                        warnx("ERROR return value from
poll(): %d\n"
                                                , poll_ret);
                                }
                                error_counter++;
                        } else {

                                if (fds[0].revents & POLLIN)
                                {

                                        /* copy sensors raw data into local buffer
*/
                                        orb_copy(ORB_ID(rc_channels),
rc_sub, &rc_raw);
                                        /*warnx("Ch1: %.2f, Ch2: %.2f, Ch3:
%.2f, Ch4: %.2f, Ch5: %.2f, Ch6: %.2f, Ch7: %.2f, Ch8: %.2f"
                                                ,
(double)rc_raw.chan[0].scaled
                                                ,
(double)rc_raw.chan[1].scaled
                                                ,
(double)rc_raw.chan[2].scaled
                                                ,
(double)rc_raw.chan[3].scaled
                                                ,
(double)rc_raw.chan[4].scaled
                                                ,
(double)rc_raw.chan[5].scaled
                                                ,
(double)rc_raw.chan[6].scaled
                                                ,
(double)rc_raw.chan[7].scaled);
```

```
                                                        */

                                                        //warnx("Selected:Value,
%i:%.2f",params.channel_number,rc_raw.chan[params.channel_number-1].scaled);




                                                        /* attempt to find parameter, set and send
it */
                                                        char *name;
                                                        name                                    =
select_params[params.selected_param];
                                                        param_t param = param_find(name);


                                                        float           new_param       =
((params.param_max-params.param_min)*(rc_raw.channels[params.channel_number-
1]+1)/2)+params.param_min;
                                                        unsigned int new_param_temp =
(new_param*(float)pow(10, params.decimal_precision)); //Note I'm using unsigned int so
that I can increase the precision of the truncate
                                                        new_param                       =
(((double)new_param_temp)/pow(10,params.decimal_precision));




                                                        char* buffer[30];
                                                        //sprintf(buffer, "new value %%s:,
%%.%df",params.decimal_precision);
                                                        //warnx(buffer,name,new_param);
```

```
                                                    if (param ==
PARAM_INVALID) {
                                                            warnx("unknown:    %s",
name);
                                                            mavlink_log_info("[tun]
unknown param: %s", name);

                                                    } else {
                                                            /* set and send parameter
*/


        if(old_param!=new_param)
                                                            {


                                                                    old_param        =
new_param;
                                                                    //sprintf(buffer, "-
-UPDATED-- %%.%df,%%.%df",params.decimal_precision,params.decimal_precision);

        //warnx(buffer,old_param,new_param);

                                                                    if(params.enable)
                                                                    {

        param_set(param, &(new_param));

        sprintf(buffer, "[tun] --UPDATED-- %%s:%%.%df",params.decimal_precision);
                                                                    }else
                                                                    {

        sprintf(buffer, "[tun] %%s:, %%.%df",params.decimal_precision);
                                                                    }
```

```
        mavlink_log_info(mavlink_fd,buffer,name,(double)new_param);

        //debug.timestamp_ms =  hrt_absolute_time();


        //debug_controller._1 = new_param;

        //orb_publish(ORB_ID(debug_controller),                debug_controller_pub,
&debug_controller);




                                                                }

        //mavlink_pm_send_param(param);
                                                        }


                                                }

                                                if (fds[1].revents & POLLIN)
                                                {

                                                        /* read from param to clear updated flag
*/
                                                        struct parameter_update_s update;
                                                        orb_copy(ORB_ID(parameter_update),
parameter_update_sub, &update);

                                                        parameters_update(&param_handles,
&params);
                                                }
```

```
                                        /* there could be more file descriptors here, in
the form like:
                                         * if (fds[1..n].revents & POLLIN) {}
                                         */




                                    }

                /* sleep 25 ms */
                //usleep(25000);
        }


        close(uart);
        thread_running = false;

        return 0;
}
```