



FACULTAD DE INGENIERÍA
UNIVERSIDAD DE BUENOS AIRES

**Tesis de Grado de Ingeniería Electrónica
Diseño y Construcción de una Computadora de Vuelo para
Vehículos Autónomos con Tolerancia a Fallas**

Alumno:

Federico NUÑEZ FRAU (98.211)
fnunezf@fi.uba.ar

Director:

Dr. Ing. Claudio POSE

FIUBA, legajo: 203004
cldpose@fi.uba.ar

Co-Director:

Ing. Leonardo GARBEROGLIO

UTN-FRSN
lgarberoglio@frsn.utn.edu.ar

COMPLETAR FECHA

Índice

1. Objetivo	3
2. Introducción	4
2.1. Computadora de Vuelo	5
2.2. Tolerancia a Fallas y Redundancias	5
3. Estado del Arte	7
3.1. Redundancias en Sistemas de Control de Vuelo en Aviones	7
3.1.1. Bus de Comunicaciones	9
3.1.2. Comparación de Resultados y Tolerancia a Fallas	10
3.2. Redundancias en Sistemas de Control de Vuelo en UAVs	11
3.2.1. Computadoras de Vuelo Comerciales	11
3.2.2. Casos de Trabajos con Componentes COTS	12
4. Análisis de Sistemas Tolerantes a Fallas en General	17
4.1. Características de Sistemas Tolerantes a Fallas	17
4.2. Uso de Redundancias	18
4.2.1. Redundancia Doble	18
4.2.2. Redundancia Triple	19
4.2.3. Necesidad del Sincronismo entre Réplicas	20
4.2.4. Necesidad del Consenso entre Réplicas	21
4.3. Simplificación del Problema	24
5. Diseño y Construcción de la Computadora de Vuelo	27
5.1. Funcionalidades de la Computadora de Vuelo	27
5.2. Criterios Generales Para la Selección de Componentes	28
5.2.1. Uso de Componentes de Grado Automotriz	28
5.2.2. Longevidad	28
5.2.3. Requerimientos de Conectores	29
5.3. Circuitos y Componentes Seleccionados	29
5.3.1. Microcontrolador	29
5.3.2. Sensor IMU	30
5.3.3. Barómetro	34
5.3.4. Magnetómetro	36
5.3.5. Interfaz de Comunicación CAN	37
5.3.6. Circuito de Alimentación	43
5.3.7. Micro SD	44
5.3.8. Interfaz USB	45
5.3.9. Conector para Módulo GPS	46
5.3.10. Conectores para Salidas de PWM	47
5.3.11. Conectores para Control por Radio	47
5.3.12. Conector para Programación y Debugging por SWD	48
5.3.13. Otras Funcionalidades	49
5.4. Desarrollo del PCB	49
5.4.1. Requerimientos de Manufacturabilidad	50
5.4.2. Requerimientos de Posicionamiento del Sensor IMU	50
5.4.3. Características del PCB	51
5.4.4. Comunicación con el Slot para Tarjeta Micro SD	54
5.4.5. Comunicación USB	54
5.4.6. Layout del Regulador Lineal	55
5.5. Diagrama en Bloques y PCB Final	56

6. Implementación del Sistema Tolerante a Fallas	59
6.1. Descripción de la Arquitectura Propuesta	59
6.2. Implementación en Firmware	64
6.2.1. Scheduler	64
6.2.2. Comunicación a Través del Bus	66
6.2.3. Sincronización	67
6.3. Pruebas Realizadas	68
6.3.1. Funcionamiento Sin Fallas	72
6.3.2. Sesgo en Valores de Giróscopo	77
6.3.3. Saltos Aleatorios en Valores de Giróscopo	79
6.3.4. Medición Invariante de Acelerómetros y Giróscopos	81
7. Conclusiones	92
8. Agradecimientos	93
Apéndices	94
Apéndice A: Circuito Esquemático	94
Apéndice B: PCB Final	94
Referencias	95

1. Objetivo

El presente trabajo de tesis tiene por objetivo el diseño y construcción de una computadora de vuelo de bajo nivel, a ser utilizada en un vehículo aéreo hexarotor, no tripulado. Como aspecto particular, esta debe contar con la capacidad de tolerar fallas de hardware y de software que puedan ocurrir en pleno vuelo. Lo que se busca es que estas fallas no impacten en la misión del vehículo y que puedan ser detectadas lo antes posible para tomar una acción.

En primera medida, se hace un análisis e investigación acerca del estado del arte, para vehículos aéreos no tripulados de carácter comercial. El objetivo es conocer los mecanismos de seguridad que comúnmente se aplican en este tipo de vehículos, tanto de hardware como de software. Por otro lado, se busca conocer cuáles son las normas actuales pertinentes al uso y comercialización de este tipo de vehículos.

En cuanto al desarrollo de la computadora de vuelo, esto comprende la definición de los requerimientos de la misma, principalmente de hardware en cuanto a sensores, conectores y funcionalidades deseadas. Se realiza una investigación de la variedad de componentes disponibles, para luego pasar a una etapa de selección de los componentes a utilizar. Por último, se define un circuito esquemático y se diseña un PCB, el cual será enviado a fabricación.

Para demostrar el funcionamiento de la computadora de vuelo, se presentan resultados donde se emulan distintas fallas sobre uno de los sensores, la unidad de medición inercial (IMU). Se presentan los resultados y la respuesta del sistema frente a estas fallas.

2. Introducción

Los vehículos aéreos no tripulados (UAVs) originalmente fueron desarrollados para uso en aplicaciones militares. Al no contar con un piloto ni con una tripulación a bordo, estos permiten llevar adelante tareas peligrosas, que pueden llegar a poner en riesgo la vida de las personas. Sumado a esto, el desarrollo y mantenimiento de este tipo de vehículos resulta menos costoso frente al de un avión tripulado [1, p. 490]. Estos factores fueron claves como motivación para el desarrollo de este tipo de vehículos.

Algunos de los principales usos en el ámbito militar son reconocimiento, vigilancia y monitoreo. Al contar con distintos tipos de sensores, como cámaras, sensores infrarrojos, entre otros, estos pueden recolectar información en zonas hostiles, de manera económica y segura.

A partir de los avances en la tecnología y la reducción en los costos de fabricación, tanto para sensores como componentes en general, este tipo de vehículos han comenzado a ser utilizados para fines civiles y comerciales. De esta manera, las mismas ventajas por las cuales comenzaron a ser utilizados en el ámbito militar, despertaron el interés de distintos sectores por fuera de este. Hoy en día estos vehículos son utilizados para distintas aplicaciones civiles y comerciales. Algunas de ellas son:

- **Búsqueda y Rescate:** En escenarios de desastres naturales, los UAVs son utilizados para tomar imágenes y videos de las zonas afectadas o búsqueda de personas. El uso de un UAV frente a un avión o helicóptero, elimina potenciales riesgos para el piloto y la tripulación, además de reducir el costo de la operación. Un ejemplo relevante de este uso de UAVs fue durante el accidente nuclear de Fukushima del año 2011 ocurrido en Japón. Debido a la radiación de la zona, el uso de UAVs fue necesario para la recolección de imágenes y video. Además, se utilizó un UAV de la Fuerza Aérea de Estados Unidos (FAA) equipado con un sensor infrarrojo para conocer la temperatura de los reactores nucleares [2].
- **Teledetección:** El bajo coste de los UAVs permite obtener gran cantidad de datos para distintas investigaciones del suelo y del medio ambiente. En [3] se puede encontrar un trabajo en donde se utilizaron vehículos aéreos no tripulados para realizar un muestreo ambiental en el Ártico, además de estudios acerca de la temperatura de la superficie del océano. En [4] se presenta un trabajo donde se utiliza un UAV para realizar mediciones sobre gases volcánicos. Utilizando distintos tipos de espectrómetros y sensores electroquímicos, se analizan las concentraciones de dióxido de carbono y dióxido de azufre.
- **Inspección en Infraestructura y Construcciones:** Utilizando UAVs es posible realizar tareas de inspección para encontrar problemas en la red de distribución de energía eléctrica [5]. Existen algunas empresas que se dedican a estas actividades de inspección, como por ejemplo Cyberhawk. Esta además ofrece otros servicios destinados a distintos sectores industriales, como monitoreo en construcciones y de generadores eólicos.
- **Agricultura de Precisión:** El uso de UAVs en este sector está destinado a mejorar el rendimiento del cultivo, a través de distintas actividades como el riego programado, la detección temprana de pestes y el sensado de la textura del suelo. Este último puede usarse como indicador de la calidad del suelo para cultivo.
- **Vigilancia:** Los UAVs se utilizan para patrullar y controlar las fronteras, por ejemplo para detectar situaciones de tráfico ilegal.

El factor común a todas estas aplicaciones es que la incorporación de los UAVs en el ámbito civil y comercial ha abierto oportunidades para realizar tareas y actividades que de otra forma serían muy costosas y/o riesgosas para las personas.

Teniendo en cuenta la importancia que han tomado, además del hecho de que en muchas de estas aplicaciones estos sobrevuelan zonas donde circulan personas, resulta mandatorio garantizar cierto grado de confiabilidad en su funcionamiento. Este es un aspecto que caracteriza la capacidad de un sistema para funcionar correctamente durante un período de tiempo, donde “correctamente” se refiere a cumplir con la tarea para el cual fue diseñado. Para el caso de un UAV, el hecho de volar en espacio aéreo civil

puede llegar a causar daño físico a personas, si es que un vehículo presenta una falla y por ejemplo pierde el control. Otra de las posibles consecuencias tiene que ver con los costos que puede ocasionar una falla en una misión relacionada a una actividad laboral. El hecho de tener que repetir la misión puede traer mayores costos para la actividad en cuestión.

Un UAV típicamente se compone de distintos elementos. Cada uno de ellos es susceptible de manifestar distintos tipos de fallas que pueden afectar al vehículo de distintas maneras. Algunos de los elementos más importantes son la estructura mecánica, el sistema de batería y alimentación, los motores y actuadores, un sistema de comunicación remota con un piloto y la computadora de vuelo. En este trabajo se abordarán aspectos relacionados a fallas relacionadas con este último.

2.1. Computadora de Vuelo

La computadora de vuelo se compone de una unidad de procesamiento que realiza la adquisición de datos de sensores y ejecuta los algoritmos necesarios para estabilizar y controlar el vehículo. Suelen incorporar una variedad de sensores a bordo, siendo el más común de ellos la Unidad de Medición Inercial (IMU), compuesta por acelerómetros y giroscopos triaxiales. A su vez, suelen disponer de magnetómetros triaxiales, barómetros, GPS, LiDARs, sensores ultrasónicos, sensores de flujo óptico, entre otros diferentes tipos de sensores de velocidad y distancia. Sumado a esto, suelen contar con distintas interfaces para comunicación con otros módulos externos, como pueden ser sensores, actuadores u otros que sean de uso específico para cumplir con la misión del vehículo.

Los datos de los sensores son utilizados para ejecutar los distintos algoritmos de navegación y control. Periódicamente se adquieren mediciones de los distintos sensores del vehículo, se procesan dichos datos para luego aplicar acciones de control a los distintos actuadores, es decir actuar sobre los motores. De esta manera se logra que el vehículo recorra una trayectoria previamente configurada, o bien que responda adecuadamente a los comandos enviados por un piloto a distancia.

2.2. Tolerancia a Fallas y Redundancias

Resulta evidente que la computadora de vuelo es el elemento central en un vehículo aéreo no tripulado, por lo que una falla puede traer consecuencias graves. Por ejemplo, una falla en una lectura de un sensor, puede resultar en una mala estimación de la posición del vehículo o incluso decantar en la pérdida de control de este. En vehículos aéreos tripulados como aviones comerciales y militares, es común que se utilicen distintas técnicas de tolerancia a fallas para mejorar la confiabilidad. Esto mismo ocurre con vehículos aéreos no tripulados de uso militar, aunque no es tan común en aquellos de uso civil y comercial.

La computadora de vuelo de este trabajo se desarrollará teniendo como objetivo la necesidad de implementar técnicas de tolerancia a fallas a partir del uso de redundancias. Esta es la principal técnica utilizada en sistemas de tiempo real [6] e implica que las tareas realizadas se ejecuten utilizando réplicas de hardware. En el eventual caso de que una de estas réplicas presente una falla y el sistema la detecte, luego se puede tomar una acción, como por ejemplo descartar la réplica fallada y utilizar alguna de las réplicas sin fallas.

El solo hecho de incorporar redundancias en un sistema no equivale a incrementar la confiabilidad. Es necesario incorporar mecanismos que administren correctamente los aspectos relacionados al manejo de las redundancias. La forma más común de detectar fallas es realizando comparaciones entre los resultados calculados por cada réplica. Por ejemplo, si se contara con un sistema con doble redundancia, podría concluirse que alguna de las dos réplicas presentó una falla a partir de la comparación de los resultados obtenidos por cada réplica. Sin embargo, a priori no podría decirse cuál de estas fue la que falló. Ambas réplicas deberían ejecutar una rutina auxiliar que realice una verificación interna. Teniendo en cuenta que el sistema de control de vuelo del UAV es un sistema de tiempo real, la ejecución de esta rutina podría perjudicar su determinismo temporal y poner en riesgo la estabilidad del vehículo. Este ejemplo ilustra la necesidad de analizar y administrar correctamente los aspectos relacionados al manejo de las redundancias.

En este trabajo se diseñará el circuito correspondiente a la computadora de vuelo y se fabricarán tres réplicas, cada una en su propio PCB. Luego de verificar el correcto funcionamiento de cada una de estas

por separado, se procederá a proponer una arquitectura distribuida, para administrar las redundancias. Finalmente, se utilizarán las tres placas para demostrar el funcionamiento del sistema redundante.

3. Estado del Arte

En esta sección se analizan distintos casos de implementación de sistemas de control de vuelo redundantes. Si bien en este trabajo se desarrolla una computadora de vuelo para UAVs, se toma como referencia características generales del funcionamiento del sistema utilizado en aviones comerciales, por ser el sistema críticos por excelencia. Estos funcionan correctamente durante muchas horas, trasladando a muchas personas. Seguido a esto, se analizan trabajos y computadoras de vuelo utilizadas en UAVs que implementen redundancias. Los sistemas analizados utilizan distintas técnicas para implementar las redundancias. Lo que se busca es detectar similitudes y posibles requerimientos que caractericen a un sistema con redundancias.

3.1. Redundancias en Sistemas de Control de Vuelo en Aviones

En aviática, el sistema de control de vuelo es sin dudas un sistema crítico. Originalmente constituidos por sistemas mecánicos complejos, estos fueron reemplazados por sistemas denominados *fly-by-wire*, introducidos en vuelos comerciales en el Airbus 320 en 1987 y el Boeing 777 en 1994 [7], con el objetivo de aliviar la carga del avión y mejorar su rendimiento en cuanto al consumo de combustible. Su nombre deriva del hecho de que la acción de control del piloto no es directa sobre el avión, sino que es una computadora la que la ejecuta. Todas las señales de sensores y de comandos son transmitidas eléctricamente desde y hacia una computadora de vuelo.

A modo de ejemplo en la figura 1, se muestra un diagrama simplificado de la arquitectura del sistema de control de vuelo principal del avión Boeing 777. En esta imagen se muestran distintos bloques los cuales se encuentran comunicados a través de un bus. Cuando el piloto del avión quiere ejecutar una acción, este lo hace a través de métodos convencionales como palancas o pedales que se encuentran en la cabina. Estas acciones generan señales analógicas las cuales son entregadas a los bloques denominados *Actuator Control Electronics* (ACEs). Estos bloques convierten la señal analógica en una señal digital que puede enviarse a través del bus de comunicaciones a los bloques denominados *Primary Flight Control System* (PFCs).

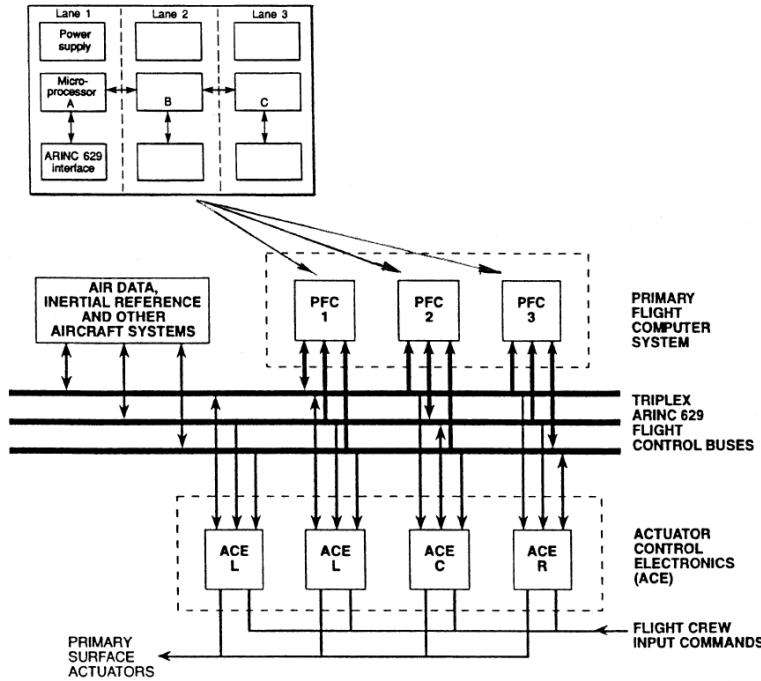


Figura 1: Arquitectura del sistema principal de vuelo, conformado por varias computadoras en una configuración redundante. La imagen se extrajo de [1].

Además de la acción del piloto, las PFCs toman datos de los distintos sensores para poder calcular todas las acciones de control que se aplicarán a los distintos actuadores del avión. Los resultados de los cálculos son enviados a los actuadores nuevamente a través del bus de comunicación a cada bloque ACE asignado para cada actuador el cual interpreta el comando recibido por el bus y lo convierte en una señal analógica que aplicará su actuador asignado. En [8] puede encontrarse una explicación más detallada del funcionamiento de este sistema en el avión Boeing 777.

En la figura 1 lo que se observa es que hay redundancias en los bloques PFCs y ACEs. No solo eso, sino que además hay redundancias en el canal de comunicación, es decir en el bus. Por si fuera poco, en el Boeing 777, cada una de las PFCs se compone a su vez de 3 microprocesadores, cada uno con su fuente de alimentación propia e interfaz de comunicación con el bus. Cada uno de esos 3 procesadores son de distintos fabricantes y sus respectivos softwares son desarrollados por grupos de trabajo distintos. Generalmente solo un procesador de cada PFC se encuentra en funcionamiento normal y los otros actúan como monitores, verificando que lo que estas calculan es correcto.

Sin dudas todo el sistema de control de vuelo presenta una complejidad muy grande. El hecho de incorporar redundancias en el sistema incrementa notablemente la seguridad. La forma en la que esta se cuantifica es a través de la probabilidad de que el sistema fracase de forma catastrófica. Para aviones comerciales típicamente debe ser $10^{-9}/h$ [1, p. 217]. Este valor es tan bajo que incluso es imposible de verificar de forma experimental, ya que habría que ejecutar el sistema durante 10^9 horas aproximadamente. La probabilidad de falla de los semiconductores no alcanza este valor [9, p. 272]. Este es el motivo por el cual se incluyen bloques redundantes en los sistemas de control de vuelo. Por ejemplo, el hecho de que cada PFC tenga 3 procesadores de distintos fabricantes permite eliminar problemas que sean propios del componente. A su vez, el hecho de que cada procesador tenga un software distinto desarrollado por un grupo de personas distinto permite que las fallas que estos puedan tener sean eliminadas a tiempo.

ACÁ AGREGAR UNA CUENTA SÚPER FÁCIL CON UN SISTEMA CON REDUDANCIAS EN

PARALELO Y CON LA PROBABILIDAD DE FALLA EXPONENCIAL.

El uso de redundancias trae consigo la necesidad de un sistema que administre todas las tareas de manera correcta con el objetivo de cumplir con el nivel de seguridad requerido. Por ejemplo, en el caso del Boeing 777 antes mencionado, detectar cuándo una de las PFCs llegó a un cálculo de la ley de control incorrecto, determinar si un sensor del avión dejó de funcionar y qué acción tomar en cada caso, entre otras.

En la figura 2 se muestra un diagrama que representa de forma general la comunicación entre los distintos elementos del sistema de control de vuelo. En este se puede ver la redundancia de buses, sensores, actuadores y computadoras de vuelo.



Figura 2: Se muestra de manera general las conexiones entre los distintos elementos del sistema de control de vuelo típico en aviones. La imagen se extrajo de [1].

Cada uno de estos elementos conforman el sistema de control de vuelo tolerante a fallas en aviones. A continuación se analizan algunas de sus características.

3.1.1. Bus de Comunicaciones

Hasta principios de los años 70s, la comunicación entre los distintos módulos dentro de los aviones se realizaba a través de arneses de muchos cables que transmitían información en paralelo. Estos eran tan grandes que podían llegar a pesar cientos de kilogramos. Sumado a esto, la enorme cantidad de cables venía acompañada de muchas conexiones, puntos que son típicos causales de fallas intermitentes. A partir de mediados de los años 70s, se comenzó a implementar el uso de buses de comunicación serial, comunes a todos los módulos del avión. Esto simplificó muchísimo el cableado, además de facilitar el desarrollo de módulos de aviónica, ya que se simplificó la forma de comunicación con el resto del avión.

La comunicación serial a través del bus utiliza un acceso al medio compartido dominado por el tiempo, *time-division multiple acces* (TDMA). Siguiendo con el caso del avión Boeing 777, el protocolo utilizado es el ARINC 629. Este funciona sin un nodo master y permite una conexión de hasta 120 nodos. Solamente uno de ellos puede acceder al medio físico a la vez, lo cual se define por el acceso al medio dominado por el tiempo. El medio de transmisión es un par trenzado, con una velocidad de 2 Mbit/s. A

continuación, en la figura 3 se muestra el bus ARINC 629.

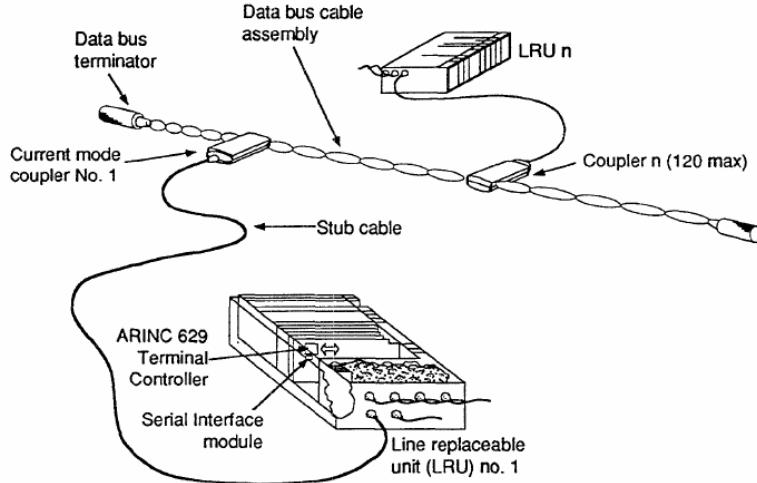


Figura 3: Se muestra un bus ARINC 629 con 2 nodos conectados. Este consiste en un par trenzado, con terminaciones en los extremos para evitar reflexiones. La conexión de los nodos al bus no es directa si no que se utilizan acopladores. La imagen se extrajo de [8].

El hecho de que todas las comunicaciones pasen por el mismo bus lo vuelve un punto clave en cuanto a la seguridad, ya que una falla en uno de sus cables dejaría a todos los módulos sin comunicación. Es por esto que se incluyen varios buses de estos en paralelo, como se mostró en la figura 2.

Un aspecto a destacar en el bus de comunicaciones es el método de acceso al medio utilizado, TDMA. El envío y recepción de mensajes se implementa por turnos. Este protocolo define en qué instantes de tiempo cada uno de los nodos puede utilizar el medio físico y en cuáles no. Para que no haya colisiones, todos los nodos deben respetar ese timing, el cual se encuentra predefinido. Esto le da determinismo y claridad al comportamiento del bus y del sistema, ya que a priori puede saberse qué mensaje se estará enviando en cada instante de tiempo. Cualquier otro tipo de comportamiento representaría una falla. Además, al tratarse de un sistema de tiempo real, no pueden permitirse las retransmisiones, ya que es evidente que se rompería el requerimiento intrínseco de este tipo de sistemas, que es cumplir con la tarea asignada antes de cierto tiempo.

3.1.2. Comparación de Resultados y Tolerancia a Fallas

El mecanismo de tolerancia a fallas es a través de la comparación entre mediciones de sensores y resultados de cálculo de la ley de control. Si todos los sensores redundantes funcionan adecuadamente, es esperable que estos entreguen mediciones muy similares. Por otro lado, si uno de ellos entrega una medición diferente a la de los otros dos, se asume que este presentó una medición incorrecta. Como resultado de la comparación, se obtiene un único valor el cual es utilizado por el sistema de control. De la misma forma, se realiza una comparación de los resultados del cálculo de la ley de control obtenido por cada una de las computadoras. Una vez que se decide por un único valor, se envía la señal de actuación.

Existen muchos criterios utilizados para seleccionar los valores de sensores. Un aspecto importante a tener en cuenta es que a pesar de que todos los sensores redundantes funcionen adecuadamente, estos presentarán ciertas diferencias en las mediciones, algo que es esperable teniendo en cuenta cuestiones propias de la construcción de cada sensor, ruido, etc. Esto debe ser tenido en cuenta al momento de

realizar las comparaciones.

En [1, p. 221] se menciona un algoritmo muy simple. Este consiste en tomar una de las mediciones como referencia y comparar las demás contra esta. En la figura 4 se toma el ángulo θ_1 como referencia, ya que $\theta_3 > \theta_1 > \theta_2$. En caso de que la diferencia $|\theta_1 - \theta_2|$ ó $|\theta_1 - \theta_3|$ supere un cierto límite, se asume que el sensor presentó una falla. En el caso de la imagen, la diferencia con el sensor 2 es mucho mayor que con el 3 y se asume que este presentó una falla. El valor que se toma como válido es el valor intermedio, θ_1 .

$$\begin{aligned} \theta_1 = 40,3 &\longrightarrow \text{Valor intermedio} \\ \theta_2 = 30,5 &\longrightarrow \text{Valor inferior} \longrightarrow |\theta_2 - \theta_1| = 9,8 \\ \theta_3 = 40,7 &\longrightarrow \text{Valor superior} \longrightarrow |\theta_3 - \theta_1| = 0,4 \end{aligned}$$

Figura 4: La comparación entre 3 sensores da como resultado que el sensor 2 presentó una falla. En consecuencia deberá tomarse una acción, por ejemplo ignorar al sensor en próximas mediciones o informar al piloto.

Este mismo esquema se repite luego del cálculo de la ley de control y de los valores a aplicar sobre cada actuador.

3.2. Redundancias en Sistemas de Control de Vuelo en UAVs

Es evidente que las consecuencias del fracaso del sistema de control de vuelo en un vehículo aéreo no tripulado, no son las mismas que en un avión comercial. Estos últimos pueden trasladar cientos de personas y realizar vuelos de muchas horas, mientras que en los primeros no hay tripulación ni piloto a bordo del vehículo. Debido a esto, suelen estar construidos con otros requerimientos de seguridad más laxos. Para UAVs de uso militar, la probabilidad de fracaso se encuentra en el orden de $10^{-5}/h$ [10][1, p. 491], una diferencia de varios órdenes de magnitud respecto de los aviones comerciales.

Al igual que en aviones de uso comercial y militar, es común el uso de redundancias en UAVs de uso militar. **CITAR VARIOS EJEMPLOS.**

En el caso de UAVs de uso civil y comercial, el uso de redundancias no es tan común. Sin embargo, existen algunas empresas que comercializan computadoras de vuelo con capacidad de utilizar redundancias. A continuación se mencionan algunas de ellas.

3.2.1. Computadoras de Vuelo Comerciales

La empresa Embention comercializa una computadora de vuelo con redundancia triple, con la posibilidad de incorporar una cuarta computadora extra [11]. Su funcionamiento se basa en que todas las computadoras de vuelo redundantes se comunican con un elemento denominado árbitro. Este ejecuta un algoritmo de votación y selecciona cuál de las tres computadoras de vuelo es la correcta. En la figura 5 se muestra un diagrama en bloques.

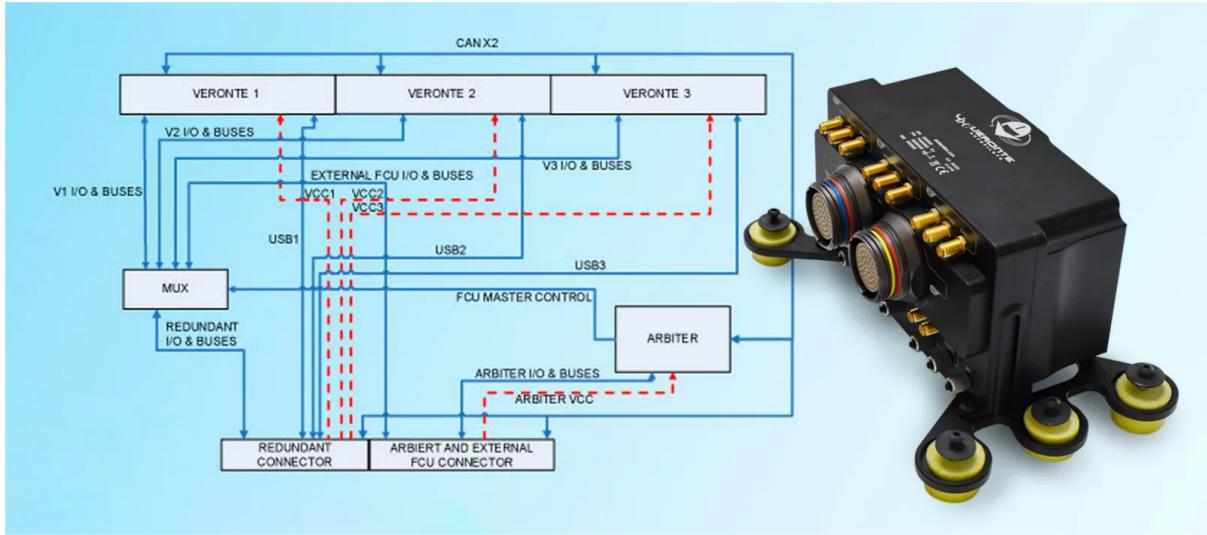


Figura 5: Diagrama en bloques del autopiloto Veronte de la empresa Embention. La imagen se extrajo de [11].

Un detalle que puede verse en este diagrama en bloques es que las computadoras de vuelo se comunican con el árbitro a través de un bus de comunicación. En el sitio web de este autopiloto se menciona que una de las interfaces de comunicación es un bus CAN doble, el cual además puede usarse para la comunicación con motores y sensores. Esto es similar al caso presentado anteriormente en aviones, donde los módulos se comunican a través de un bus de comunicaciones.

Vector-600 es una computadora de vuelo con doble redundancia, comercializada por la empresa UAV Navigation [12]. Esta ofrece redundancia doble en la CPU que realiza los cálculos de actuación y procesamiento de sensores, redundancia doble en la fuente de alimentación y en algunos de los sensores.

La empresa MicroPilot ofrece un autopiloto con redundancia triple, MP21283X [13]. Este se compone de 3 computadoras de vuelo iguales en hardware y software. Durante su uso, la primera de las computadoras de vuelo se encarga de controlar al vehículo. En caso de que esta presente una falla, el autopiloto cambia y utiliza la segunda computadora. Si esta falla, se pasa a la tercera.

Estas computadoras de vuelo tienen la particularidad de tener precios muy altos, por ejemplo la primera de ellas de Embention tiene un precio entre 23500 € y 27000 €. El presente trabajo busca desarrollar una computadora de vuelo con componentes COTS, por lo que este presupuesto excede la capacidad de este trabajo. Pueden encontrarse una gran cantidad de trabajos que abordan el desarrollo de computadoras de vuelo con redundancias y que utilizan componentes COTS. A continuación se mencionan algunos de ellos.

3.2.2. Casos de Trabajos con Componentes COTS

En los trabajos [14] y [15] los autores presentan una computadora de vuelo redundante, desarrollada con componentes COTS. Esta comprende una redundancia cuádruple utilizando cuatro microcontroladores iguales. Al igual que en el caso del avión comercial y en los autopilotos presentados, la tolerancia a fallas se aborda a partir del intercambio de información. Se utilizan cuatro interfaces SPI, donde en cada una de estas un microcontrolador diferente actúa como master. Los microcontroladores recolectan datos de sensores y realizan un intercambio para ponerse de acuerdo y llegar a un consenso acerca de cuál es el valor correcto. Una vez que esto se decide, se realiza el cálculo de la ley de control. Antes

de aplicar el resultado a los motores, se vuelven a comparar resultados para detectar y filtrar fallas. Mientras que en [14] se muestran los resultados, en [15] se abordan cuestiones relacionadas al diseño e implementación utilizando componentes COTS. Un aspecto importante de este trabajo es que los cuatro microcontroladores trabajan de manera sincronizada. Los autores mencionan que esto es algo que no puede obviarse, ya que el sistema de control del UAV es un sistema de tiempo real. Para que la tolerancia a fallas funcione adecuadamente, todos los microcontroladores deben procesar datos de sensores que correspondan al mismo ciclo de control. En otras palabras, los 4 nodos de la red redundante realizan la comparación de los datos de sensores al mismo tiempo, realizan el cálculo de la ley de control al mismo tiempo y finalmente vuelven a comparar los resultados al mismo tiempo. En la figura 6 se muestra un esquema que ejemplifica esto.

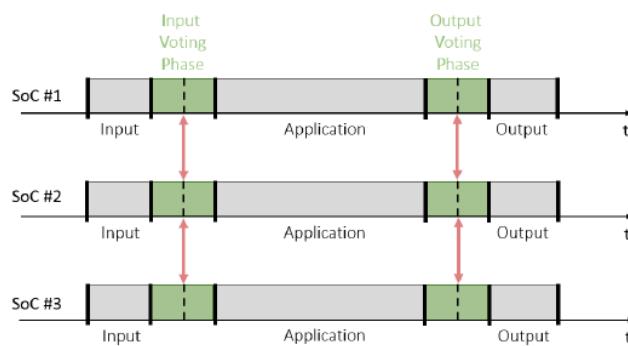


Figura 6: Imagen que demuestra la sincronización entre 3 microcontroladores que realizan las mismas tareas en paralelo. La imagen se extrajo de [15].

Cabe aclarar que la sincronización que se menciona no tiene nada que ver con los osciladores que utiliza cada microcontrolador para su propio funcionamiento. Lo que se sincroniza es el scheduling de las tareas ejecutadas por cada microcontrolador. Por otro lado, esta sincronización no es perfecta ya que sería algo prácticamente imposible. Se acepta que haya cierto desfasaje que no perjudique demasiado el control del vehículo. En [14] se explica la técnica de sincronización utilizada.

A diferencia del caso del avión, la comunicación en este trabajo no se realiza por medio de un bus, sino que es a través de 4 interfaces SPI. La justificación de los autores es porque pueden alcanzarse tasas de transferencia de hasta 50 MBit/s, muchísimo mayor que en el bus ARINC 629 que era de 2 MBit/s. Como contrapartida, una conexión SPI requiere de las líneas MOSI, MISO, CS y SCK, además del retorno GND ya que la señal eléctrica de SPI es de modo común. La cantidad de conexiones es mucho mayor que en el caso del uso de un bus. Por ejemplo el autopiloto de Embention utiliza el bus CAN, donde solamente se requieren dos cables, CAN-H y CAN-L. Además, SPI no implementa ningún mecanismo para verificar la integridad del mensaje recibido. Otro aspecto negativo es que el uso de SPI no permite integrar más módulos, como sí sucede en el caso del autopiloto de Embention, donde el mismo bus CAN se utiliza para adosar sensores y actuadores diferentes.

En el caso del avión, se había mencionado que el acceso al bus de comunicación era gobernado por el tiempo. En este trabajo además la ejecución del lazo de control y la comparación de resultados también es gobernada por el tiempo.

Otro aspecto interesante de este trabajo es que no hay un único elemento que compare los resultados de cada computadora, sino que todas ellas lo hacen. Esto es algo que realiza el autopilot de la empresa Embention mencionado anteriormente. Los autores argumentan que generalmente cuando se utiliza este tipo de árbitro que decide cuál es la computadora de vuelo correcta, esta debe tener una probabili-

dad de fracaso muy inferior a cada uno de los nodos redundantes, ya que si este árbitro fracasa, todo el sistema fracasará. Esto se muestra en la figura 7. El árbitro suele ser de un costo muy elevado, algo que se contradice con el requerimiento de que todo el sistema sea desarrollado con componentes COTS.



Figura 7: A pesar de que los 3 módulos redundantes funcionen correctamente, una falla del árbitro se traduce directamente en un error en el sistema. Esto lo vuelve un punto singular de falla.

Una cuestión que no se aclara en este trabajo es cómo se aplican las señales de control a los motores del UAV, ni cómo se recolectan los datos de sensores.

En [10] se presenta otro trabajo desarrollado con componentes COTS. Este también utiliza una arquitectura gobernada por el tiempo. En este trabajo los autores la presentan formalmente con el nombre de *Time-Triggered Architecture*. Esta consiste en que las tareas ejecutadas por el procesador se encuentran predefinidas de forma estática en tiempo de compilación. En este trabajo, al igual que en el caso del avión, se utiliza un bus de comunicación con acceso TDMA, FlexRay [16]. El bus utilizado es doble, para evitar que este sea un punto singular de falla. Al igual que en el trabajo antes mencionado, en este también se implementa una sincronización entre las distintas computadoras de vuelo.

La tolerancia a fallas se realiza a través de la comparación de resultados, como en todos los casos presentados hasta el momento. Un aspecto particular de este trabajo es que además de los nodos que realizan los cálculos de ley de control, se incorporan otros microcontroladores extra que se dedican a procesar datos de sensores y de actuadores. Los autores mencionan que esto se hace para aliviar la cantidad de datos enviados a través del bus de comunicaciones y el procesamiento que deben realizar los nodos que calculan la ley de control. Como aspecto negativo, esto encarece a la computadora de vuelo, ya que se requiere una mayor cantidad de procesadores.

La sincronización de los nodos redundantes es algo que se repite en varios trabajos encontrados. En [17] se presenta un desarrollo de una computadora de vuelo para pequeños UAVs, con redundancia doble y sincronización en la ejecución de las tareas. La redundancia también se administra a través de la comparación de entradas de sensores y resultados de cálculos de la ley de control. En la figura 8 se muestra un diagrama en bloques. Si bien ambas computadoras trabajan en paralelo, solo una de ellas es la que controla los actuadores.



Figura 8: Diagrama en bloques del sistema de redundancia doble. Las CPU1 y CPU2 comparan sus resultados y envían sus salidas al bloque *Output control*. A través de un bloque árbitro se selecciona a cuál de las dos CPU será la que controle los actuadores. La imagen se extrajo de [17].

En caso de que ocurra una discrepancia entre los resultados de ambas, eso indicará que alguna de las dos computadoras cometió un error, pero no se sabrá cuál fue. Luego de ejecutar una serie de rutinas se verifica cuál de las 2 cometió el error y en caso de que sea necesario, se le pasa el control de los actuadores a la computadora de back-up.

Un aspecto negativo de esta configuración es el hecho de que la comparación de resultados no permite identificar cuál de las dos CPUs cometió el error, solamente se puede saber que ocurrió un error. Pensando en que la ejecución del lazo de control es un sistema de tiempo real, sería deseable que a pesar de la falla, el sistema de control pueda seguir ejecutándose. El hecho de tener que ejecutar rutinas para verificar a la computadora fallada presenta un trabajo que perjudica el control del vehículo. Esto es algo que no sucede por ejemplo, si se utilizan 3 computadoras de vuelo en paralelo, ya que si una presenta un dato incorrecto, simplemente puede ignorarse el dato y utilizar los datos de las otras 2 computadoras correctas. Esto se denomina *fault masking* o enmascaramiento de la falla.

En [18] y [19] pueden encontrarse otros 2 trabajos más que utilizan la sincronización entre los nodos redundantes. El primero de ellos trabaja con redundancia triple y un árbitro que selecciona cuál de los nodos controla los actuadores. El segundo también trabaja con redundancia triple, pero no utiliza un árbitro sino que las tres computadoras realizan la votación y cada una de ellas envía un mensaje a un nodo que se encuentra en el mismo bus y se encarga de controlar el actuador. En la figura 9 se muestra el diagrama en bloques.



Figura 9: Diagrama en bloques del sistema redundante de [19]. Cada uno de los nodos tiene sus propios sensores, la única comunicación que se realiza a través del bus CCDL (*Cross-Communication Data Link*) es para realizar la comparación de resultados y la votación.

En [20] se presenta un trabajo de tesis en el que no se utiliza una sincronización entre los nodos. Este consiste en la utilización de 2 computadoras de vuelo de fácil acceso comercial, PixHawk [21], conectadas

a una tercera computadora de vuelo central implementada con una Raspberry Pi que funciona como árbitro. En la figura 10 se muestra un diagrama en bloques de esta arquitectura. La técnica utilizada consiste en que el árbitro continuamente le pide a ambas computadoras información acerca de su “estado de salud”. A partir de la información recibida de ambas, el árbitro controla unas llaves implementadas como relés de estado sólido que seleccionan cuál de las 2 computadoras será la que controle los motores.



Figura 10: Arquitectura de la computadora de vuelo utilizada en [20]. El árbitro selecciona cuál de las dos señales PWM se utiliza para controlar el *Electronic Speed Controller* (ESC).

Un aspecto que no se menciona en este trabajo es cómo se administra el switcheo de los relés. Teniendo en cuenta que este switcheo puede traer consigo un cambio brusco en la señal PWM que ven los actuadores de los motores, habrá un período de reestabilización del lazo de control. Este requiere un análisis detallado que asegure que no se pierda el control del vehículo.

4. Análisis de Sistemas Tolerantes a Fallas en General

El objetivo de esta sección es introducir el uso de redundancias como técnica para incrementar el nivel de seguridad en un sistema. Esta consiste en el uso de varias réplicas de distintos componentes, las cuales realizan las mismas tareas de manera independiente. En caso de que ocurra una falla, el sistema podrá seguir ejecutando su función de manera correcta, utilizando alguna de las demás réplicas. A partir de esto, se presentan las características de sistemas redundantes y los requerimientos que estos deben cumplir para funcionar adecuadamente. Finalmente, se muestra que si el sistema cumple con ciertas características, el funcionamiento del mecanismo para tolerar fallas puede simplificarse.

4.1. Características de Sistemas Tolerantes a Fallas

El objetivo del diseño tolerante a fallas consiste en mejorar la confiabilidad de un sistema, apuntando a que este pueda ejecutar su función correctamente, a pesar de la presencia de una cierta cantidad de fallas [6]. Tal como lo indica su nombre, un sistema tolerante a fallas es aquel donde una falla no implica necesariamente un fracaso en el funcionamiento. Tampoco quiere decir que no ocurrén fallas, sino que por el contrario, se acepta que las fallas pueden ocurrir. Lo que se busca es que el sistema igualmente pueda cumplir con su función de manera correcta.

De manera de introducir la nomenclatura que se encuentra en la bibliografía, se definen los siguientes términos:

- Falla (*Fault*): Es alguna condición anómala, no esperada.
- Error: Ocurre cuando una falla se manifiesta y produce un comportamiento fuera de lo esperado en alguna parte del sistema.
- Fracaso (*Failure*): Quiere decir que el sistema no puede cumplir con su función de manera adecuada.

Por ejemplo en un UAV, la información proveniente de los distintos sensores se utiliza para estimar la pose del vehículo. Si un sensor entrega información que no se corresponde con la realidad, luego esto implicaría a una **falla**. Si esta no es detectada y corregida, luego la computadora de vuelo utilizará esta información para realizar los cálculos de estimación, obteniendo una estimación incorrecta de la pose. Esto representa un **error**. Finalmente, esto causará que tampoco pueda seguir su trayectoria correctamente, lo que implicaría un **fracaso** del sistema.

Se define a la confiabilidad de un sistema como la probabilidad de que este pueda cumplir con su función de manera correcta en un intervalo de tiempo $[t_0; t]$, dado que en el instante inicial t_0 podía hacerlo [9, p. 10]:

$$R(t) = P(\text{funcionamiento correcto en } t | \text{funcionamiento correcto en } t_0) \quad (1)$$

En el caso en el que se tuviera un sistema sin ningún mecanismo de tolerancia, la ocurrencia de una falla implicaría un funcionamiento incorrecto, es decir un fracaso. Esto quiere decir que la confiabilidad sería exactamente igual a la probabilidad de que no ocurra ninguna falla en el intervalo $[t_0; t]$ [6]. En este caso, la confiabilidad podría expresarse como en la ecuación (10) y la única manera de mejorarlala sería incrementando la probabilidad de que no ocurra ninguna falla en el intervalo $[t_0; t]$.

$$R(t) = P(\text{no ocurrió ninguna falla en } [t_0; t]) \quad (2)$$

Si es posible demostrar que el sistema cumple con un determinado requerimiento de confiabilidad, luego no sería necesario el uso de técnicas de tolerancia a fallas. La manera de hacer esto puede ser por ejemplo, utilizando componentes o módulos de muy buena calidad lo suficientemente confiables [6]. Sin embargo, esto puede ser muy costoso, pensando en que un sistema puede estar compuesto de muchos componentes y cada uno de ellos puede manifestar una gran cantidad de posibles fallas. Sumado a esto, se dificulta la etapa de diseño de un sistema, ya que cualquier error de diseño que no se haya tenido en cuenta puede llegar a causar una falla y por ende un fracaso del sistema.

El uso de técnicas para tolerar fallas representa una actitud más conservadora, ya que se acepta que las fallas pueden ocurrir. Dado que en el intervalo $[t_0; t]$ puede o no ocurrir una falla, la probabilidad de que el sistema pueda cumplir su función en t puede expresarse como en la ecuación (9). Si no ocurre ninguna falla, luego el sistema podrá seguir cumpliendo su función en t . Además, si llegase a ocurrir una falla, pero el sistema tiene la capacidad de tolerarla, luego el sistema de igual manera podrá seguir cumpliendo su función en el instante t .

$$\begin{aligned} R(t) &= P(\text{no ocurrió una falla en } [t_0; t]) \\ &\quad + P(\text{funcionamiento correcto en } t | \text{ocurrió una falla en } [t_0; t]) P(\text{ocurrió una falla en } [t_0; t]) \end{aligned} \quad (3)$$

La probabilidad de que el sistema funcione correctamente a pesar de la falla, está pesada por la probabilidad de ocurrencia de dicha falla. A partir de esto se desprende que aplicar técnicas de tolerancia a fallas para cada una de las posibles fallas puede resultar exhaustivo, principalmente porque deberían conocerse todas las fallas posibles. Lo que se propone es considerar solo aquellas fallas cuya criticidad es alta. Una de estas corresponde al ejemplo antes mencionado, donde una falla de un sensor puede causar el fracaso de la misión del vehículo.

4.2. Uso de Redundancias

La detección de una falla en el sistema puede llevarse a cabo de distintas formas. Volviendo sobre el ejemplo antes presentado de la estimación de la pose del vehículo, una forma de detectar la falla podría ser analizando si el valor que entrega el sensor “tiene sentido”. Para esto último sería necesario definir un algoritmo que pueda discriminar entre un valor entregado por el sensor que “tiene sentido” y uno que no. Una forma muy simple podría ser monitoreando que los valores entregados por el sensor se encuentren dentro de cierto rango. Esto sería para el caso de un sensor, aunque se requeriría un mecanismo similar para los demás sensores u otros componentes del sistema.

En un sistema de tiempo real, como lo es el control de vuelo de un vehículo, la computadora de vuelo debe aplicar una nueva acción sobre sus actuadores de manera periódica. Esto impone una restricción en el tiempo de ejecución sobre los algoritmos que detecten las posibles fallas, ya que no deben perjudicar la periodicidad del sistema de control del vehículo.

A su vez, podría ocurrir que la información de los sensores sea correcta, pero que el cálculo de la estimación de la pose o de la ley de control presente un resultado incorrecto. Para tolerar esto, se requerirían otros algoritmos para la detección de estos errores, los cuales también deben tener un tiempo de ejecución que no perjudique la periodicidad del sistema de control del vehículo.

Otra forma de detectar fallas y errores puede ser a partir del uso de redundancias. Esta es la principal técnica de tolerancia a fallas [22]. Tanto las fallas como los errores pueden detectarse a partir de la comparación de resultados entre réplicas. Esto implica una carga computacional mucho menor, ya que basta con comparar los resultados entre réplicas para detectar las fallas. Por ejemplo, podría utilizarse más de un sensor del mismo tipo para tolerar fallas de sensores, o más de una computadora de vuelo, para tolerar errores en los cálculos de la ley de control y en los sistemas de navegación.

Todos los trabajos presentados en la sección 3 comparten la característica de implementar la tolerancia a fallas utilizando varias réplicas del mismo elemento de hardware. A partir de estos se pudo ver que existen muchas configuraciones típicas, algunos utilizan 2 réplicas, otros 3 o incluso 4. Con el objetivo de entender las capacidades de cada una de estas, a continuación se analizan distintas configuraciones redundantes.

4.2.1. Redundancia Doble

Una configuración muy simple es la redundancia doble. En este tipo de sistemas se utilizan 2 réplicas las cuales trabajan en paralelo ejecutando las mismas tareas y comparando sus resultados entre sí. La ocurrencia de una falla se detecta cuando ocurre una discrepancia entre los resultados obtenidos por cada una de las réplicas.

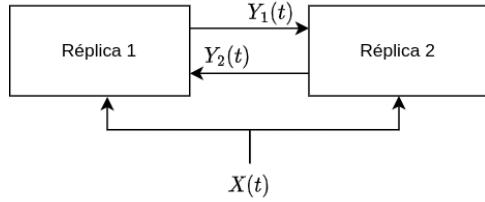


Figura 11: Esquema de redundancia doble. La entrada $X(t)$ es utilizada por ambas réplicas y cada una de ellas obtiene una salida $Y_1(t)$ e $Y_2(t)$. Si ocurre que $Y_1(t) \neq Y_2(t)$, luego esto implicará una falla de alguna de las réplicas.

Si bien este tipo de arquitectura permite detectar fallas a partir de la comparación de resultados, no es posible identificar cuál de las réplicas fue la que falló. Para lograr esto último, cada una de estas debería ejecutar una rutina para detectar si ellas fueron las que cometieron el error o no [23]. En un sistema de tiempo real, donde existe un requerimiento temporal para obtener un nuevo resultado de la estimación de la pose y de la acción de control sobre los motores, no resulta aceptable que el sistema deba detenerse o retrasar su ejecución para realizar este análisis en pleno vuelo. Debido a esto, esta configuración no se presta para uso en sistemas de tiempo real como el de la computadora de vuelo.

4.2.2. Redundancia Triple

Esta arquitectura puede encontrarse en la literatura con el nombre *Triple Modular Redundancy (TMR)* [24] y consiste en el uso de 3 réplicas, las cuales computan los mismos resultados en paralelo. En esta configuración se asume que solamente 1 de las 3 réplicas presentará una falla a la vez. A diferencia de la redundancia doble, en esta configuración sí es posible detectar cuál de las réplicas fue la que presentó la falla a partir de la comparación de resultados, ya que solamente 1 de los resultados será distinto a los otros 2.

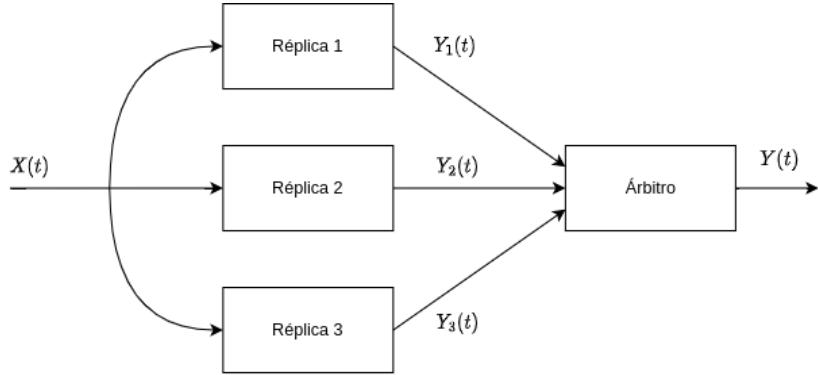


Figura 12: Esquema de redundancia triple. La entrada $X(t)$ es utilizada por las 3 réplicas para obtener sus respectivos resultados $Y_1(t)$, $Y_2(t)$ e $Y_3(t)$. Un bloque denominado árbitro se encarga de seleccionar el valor correcto para la salida $Y(t)$.

Como se muestra en la figura 12, además de las 3 réplicas se utiliza un elemento adicional denominado árbitro. Este tiene la tarea de comparar los resultados de las 3 réplicas y determinar cuál es el resultado correcto. De esta manera no solo es posible detectar cuál de las réplicas fue la que falló, sino que además es posible obtener una salida correcta $Y(t)$, sin la necesidad de que el sistema deba detenerse a hacer un análisis, algo que sí ocurría en la redundancia doble. Esto resulta especialmente útil en sistemas de tiempo real, donde continuamente debe proveerse una nueva salida del sistema de navegación y de la actuación de los motores. Esto se denomina enmascaramiento de la falla [6], ya que la falla de una de las

réplicas queda cubierta por el valor correcto entregado por las otras 2. Debido a que la selección de la salida correcta $Y(t)$ se ejecuta a través de una votación entre los valores $Y_1(t)$, $Y_2(t)$ e $Y_3(t)$, este bloque también es llamado voter en inglés [24].

El hecho de que el árbitro sea el que determine cuál es el resultado correcto, necesariamente implica que debe ofrecer una confiabilidad, $R(t)$, mucho mayor que la de cada una de las 3 réplicas. Esto se denomina punto singular de falla, ya que una falla en el árbitro inevitablemente genera un fracaso de todo el sistema. Típicamente el árbitro se constituye por hardware más robusto, volviéndolo más costoso. Por ejemplo, cada réplica puede contener un procesador como un microcontrolador COTS, mientras que el árbitro puede estar implementado con un ASIC específico para esa aplicación [15].

Para eliminar este punto singular de falla, sería necesario utilizar varias réplicas del árbitro [6][24], lo que permitiría enmascarar errores que estos puedan cometer. A priori esto podría parecer inviable, ya que se requeriría una gran cantidad de componentes, 3 computadoras de vuelo + 3 bloques árbitro, solo para poder tolerar la falla de 1 de los componentes. Además, teniendo en cuenta el argumento de que los árbitros generalmente son más costosos que las réplicas individuales, se encarecería mucho el sistema completo.

Una alternativa podría ser que las mismas réplicas sean las que ejecuten la votación, comparando sus salidas con las de sus pares. Esto es algo que se lleva a cabo en uno de los trabajos presentados en la sección 3, donde los autores presentan resultados para una arquitectura con redundancia cuádruple, donde los mismos microcontroladores de cada réplica son los encargados de realizar la votación [14]. Para el caso de una arquitectura de redundancia triple, puede diagramarse como en la figura 13.

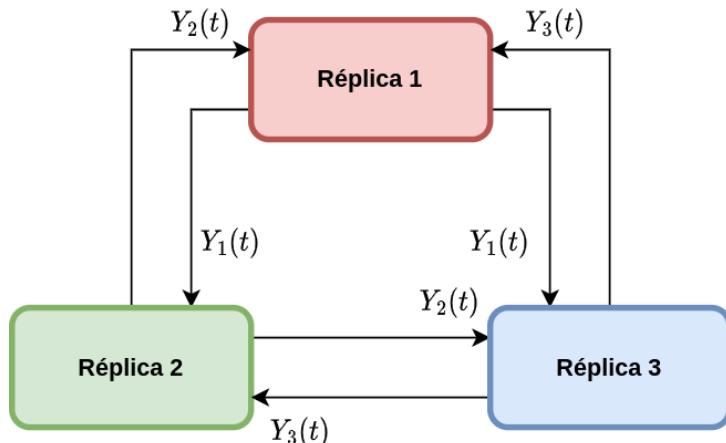


Figura 13: Cada una de las réplicas comparte los resultados con sus pares. Luego, cada una de ellas realiza una votación sobre los valores para obtener el resultado correcto y detectar fallas.

Puede encontrarse bibliografía en la que esta configuración, donde cada réplica contiene su propio elemento voter, se define como una característica fundamental de la *Triple Modular Redundancy (TMR)* [9, p. 156]. Sin embargo, hay otros requerimientos que deben cumplirse para que el sistema funcione de manera exitosa. Uno de ellos es la necesidad de la sincronización entre réplicas, la cual se explica a continuación.

4.2.3. Necesidad del Sincronismo entre Réplicas

Las tareas de estimación de la pose del vehículo, y de cálculo de la acción de control a aplicar sobre los motores comprenden tareas con requerimientos temporales, ya que periódicamente debe obtenerse un nuevo valor, acorde al estado en el que se encuentra el vehículo. En la figura 14 se muestra un ejemplo donde $Y(t)$ corresponde a la estimación de la pose del vehículo. El valor de este resultado solo tiene

validez durante el intervalo de tiempo $[t; t + 1]$. Lo mismo ocurre con todos los siguientes resultados $Y(t + N)$, los cuales solamente serán válidos durante los intervalos $[t + N; t + N + 1]$.

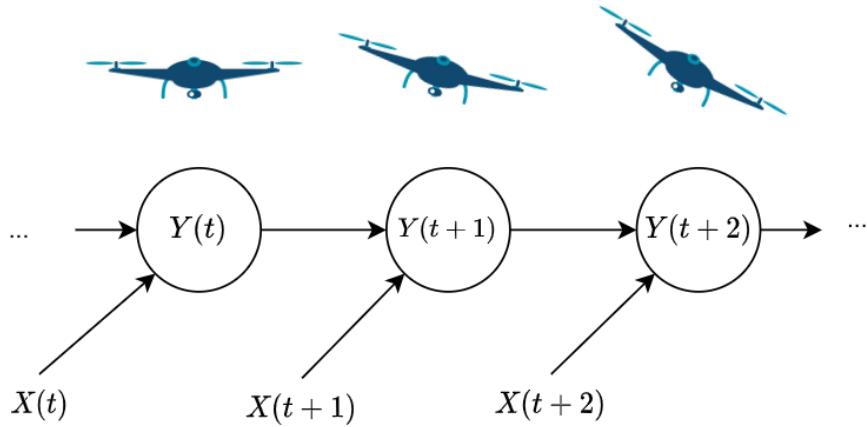


Figura 14: En cada instante de tiempo t , $t + 1$ y $t + 2$ el vehículo se encuentra en un estado distinto, por lo que el algoritmo de estimación de la pose generará 3 resultados diferentes $Y(t)$, $Y(t + 1)$ e $Y(t + 2)$. Las entradas $X(t)$, $X(t + 1)$ y $X(t + 2)$ corresponden a mediciones de los distintos sensores.

Al utilizar varias réplicas de la computadora de vuelo, cada una de estas continuamente calcula estos resultados $Y(t)$, $Y(t + 1)$, etc, de manera periódica. Luego, a partir de las comparaciones es que se detectan y se encamscaran las fallas. Para que estas comparaciones tengan sentido, deben llevarse a cabo sobre resultados que se correspondan temporalmente. Esto implica que todas las réplicas deben calcular sus correspondientes resultados $Y_i(t)$ al mismo tiempo.

Cada una de las réplicas ejecuta una serie de tareas para realizar cálculos sobre la estimación de la pose y sobre la acción de control. La manera de asegurar que estas se ejecuten al mismo tiempo es a través de un mecanismo de sincronización. Esto es lo que sucede prácticamente en todos los trabajos que se presentaron en la sección 3 para UAVs.

En la práctica una sincronización perfecta sería algo excesivo. Sin embargo esto no es necesario y puede tolerarse cierta precisión, la cual será más o menos rigurosa dependiendo de las necesidades del sistema.

4.2.4. Necesidad del Consenso entre Réplicas

La configuración de redundancia triple permite la tolerancia a fallas de 1 de las 3 réplicas. Si todas las réplicas sin fallas ejecutan el mismo algoritmo de votación y estas poseen los mismos valores de entrada $Y_1(t)$, $Y_2(t)$ e $Y_3(t)$, luego llegarán a la misma conclusión acerca del valor correcto $Y(t)$. Lo que se plantea aquí es un escenario como el de la figura , donde una de las réplica entrega valores distintos a sus pares.



Figura 15: La FCC1 entrega un valor distinto de timing a las demás FCCs

En este escenario, la FCC1 entrega dos valores distintos de su clock a las demás FCCs. Cada una de ellas luego realiza un promedio para llegar a un único valor. Lo que se observa es que las FCC2 y FCC3 calcularán un valor promedio distinto, es decir, no se sincronizarán. Una posible solución podría ser que las FCCs hagan un nuevo intercambio, con los valores promedio calculados y realicen una votación interna. Esto se muestra en la figura 16.

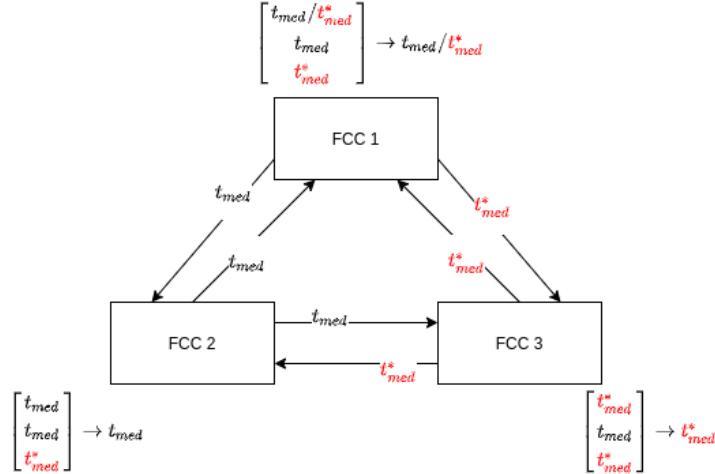


Figura 16: Luego de calcular los promedios, las FCCs intercambian sus resultados. Nuevamente, la FCC1 comete una falla en el envío del dato.

Esta última situación, donde la FCC1 nuevamente comparte dos valores distintos a las demás, puede llevar a que las computadoras de vuelo no se sincronicen, algo que como ya se mencionó, es crítico para la correcta ejecución del algoritmo de tolerancia a fallas. Podría argumentarse que es demasiado pesimista pensar que la FCC1 puede producir la misma falla 2 veces de manera consecutiva, ya que existe una baja probabilidad de que ello suceda. Sin embargo, la situación planteada en esta sección puede tratarse como un tipo de falla de hardware que se manifiesta como comportamientos arbitrarios. Que exista una sincronización entre nodos redundantes implica que estos llegan a un consenso del paso del tiempo y el ritmo al que deben ejecutar sus tareas asignadas. Este consenso resulta crítico para que el sistema pueda detectar fallas correctamente.

Algunos de los trabajos presentados anteriormente además realizan el algoritmo de votación sin la inclusión de un árbitro. Este caso es idéntico al de la figura 15, es decir que el mismo problema del consenso también está presente para las votaciones acerca de resultados de cálculo de ley de control.

El modelo de falla que se está considerando representa un comportamiento anómalo arbitrario, es decir, que a priori no se asume nada acerca de la falla. A este tipo de comportamiento se lo denomina falla bizantina o *Byzantine Fault* en inglés y básicamente consiste en asumir que el elemento que manifiesta la falla presenta un comportamiento arbitrario. Por ejemplo, un sensor puede dejar de funcionar repentinamente y no dar más respuesta, puede dejar de enviar respuesta por un período de tiempo y luego volver a funcionar, podría también enviar datos a un microcontrolador pero que esos datos sean incoherentes, etc. El modelo de falla bizantina no asume modos de falla, sino que el comportamiento es arbitrario [25][15][22]. El nombre proviene de un problema denominado *The Byzantine Generals Problem*, formalizado en [26]. Otros trabajos que tratan el mismo problema son [27] y [28]. Este último, presenta el diseño de una computadora de vuelo tolerante a fallas que utiliza los resultados del *Byzantine Generals Problem* para realizar distintas tareas de redundancia.

Se plantea una situación como la de la figura 15, pero en este caso se utilizan 4 computadoras de vuelo en lugar de 3. En este caso, las computadoras de vuelo deben sincronizarse. Para lograrlo, ellas comparten un valor de timestamp, que pueden utilizar para ajustar sus clocks. En la figura 17 se muestra un escenario en el que una de las computadoras de vuelo presenta una falla tal que le informa un valor distinto a cada una de sus pares.



Figura 17: Debido a una falla, la computadora de vuelo 1 le entrega valores distintos de timestamp a las demás.

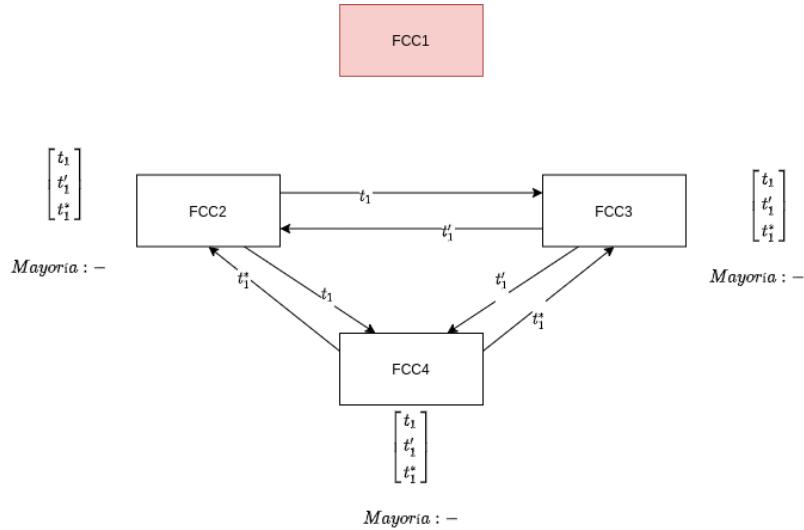


Figura 18: Las FCC2, 3 y 4 comparten entre sí lo que les dijo la FCC1 a cada una y llegan a la conclusión de que la información es inconsistente.

A través de un segundo intercambio, las FCC 2, 3 y 4 llegan a la conclusión de que el timestamp de la FCC1 no es claro. En este caso, descartan el valor. Luego de hacer todos los intercambios de timestamp, las FCCs podrán aplicar internamente la sincronización, por ejemplo, calculando un promedio de todos los timestamp. **Dado que todas las FCCs tendrán la misma información de timestamp entregado por las demás FCCs, luego todas llegarán al mismo promedio y se sincronizarán.**

Un aspecto interesante es el hecho de que en el paper original, se hace una analogía entre un nodo redundante con fallas y un nodo traidor, es decir, que busca corromper el consenso de los demás nodos. Esto lo que quiere decir es que las fallas presentadas por las computadoras de vuelo pueden ser justamente de cualquier característica, incluso al extremo de presentar un comportamiento malicioso, con el objetivo de perjudicar al sistema [22]. Esto sienta las bases para la tolerancia a fallas de hardware arbitrarias.

La implementación del algoritmo tolerante a fallas arbitrarias resulta costoso. Para poder tolerar fallas provenientes de 1 FCC se requiere un total de 4 computadoras de vuelo. Además, debe haber una interconexión entre las 4 computadoras y ellas deben intercambiar información continuamente para poder detectar y enmascarar la falla. A todo esto se le debe sumar, la necesidad de la sincronización.

4.3. Simplificación del Problema

Una de las cuestiones que no se menciona en el problema original, es el caso en el que los nodos constituyen sistemas de tiempo real. Las computadoras de vuelo deben realizar tareas que requieren determinismo temporal, por ejemplo cálculo de la ley de control, estimación de la pose, etc... En el problema original, los nodos pueden enviar sus mensajes a sus pares en cualquier momento y en cualquier orden. Otro de los puntos que caracterizan al problema original, es el hecho de que la comunicación entre los nodos es 1 a 1. Debido a esto, los traidores pueden entregar información confusa a sus pares para tratar de romper el consenso. Esto es lo que vuelve complejo al problema [26] y costosa a su solución [25]. Si el sistema en cuestión presenta la características de ser de tiempo real e implementar una comunicación a través de un bus, en conjunto, luego el problema del consenso puede simplificarse mucho.

En sistemas de tiempo real para aplicaciones *safety-critical*, es común encontrar sistemas distribuidos con comunicación a través de un bus. Esto se mencionó en la sección 3 tanto para el caso del avión como

para varios de los ejemplos de UAVs presentados. Esto también ocurre por ejemplo en los automóviles, los nodos que se encuentran repartidos por todo el vehículo se comunican a través de redes como CAN[29] o FlexRay[16]. Todos los nodos de la red se encuentran conectados al mismo bus de comunicación, por lo que cuando un nodo envía un mensaje a través del bus, todos los demás nodos reciben el mismo mensaje.

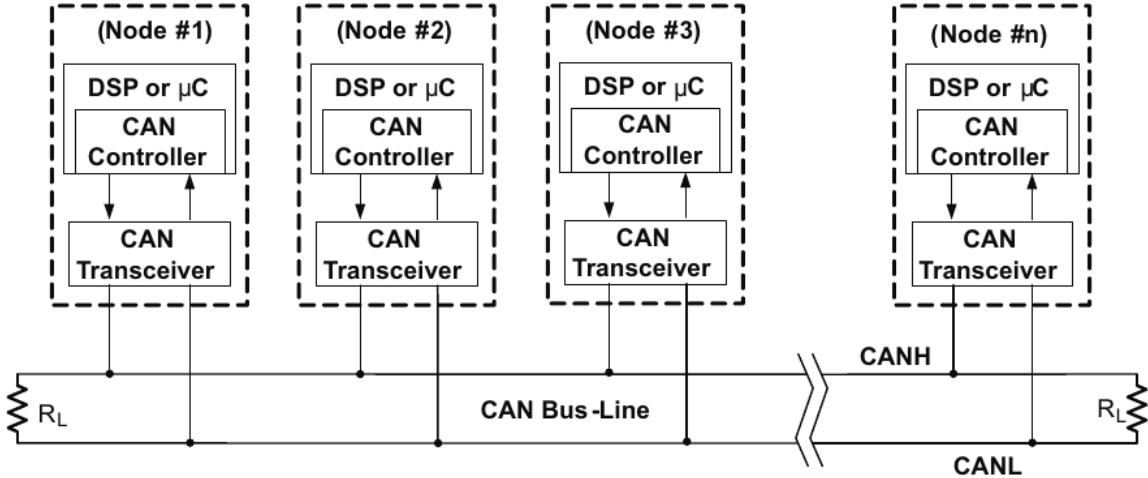


Figura 19: Todos los nodos se encuentran conectados al mismo bus de comunicaciones. En el caso del bus CAN, se compone de dos cables, CAN-H y CAN-L, terminados en sus extremos por resistencias de adaptación. La imagen se extrajo de [30].

Esto presenta una diferencia respecto de lo planteado en *The Byzantine Generals Problem*, ya que la existencia de un bus común a todos los nodos automáticamente elimina la posibilidad de que uno de los miembros de la red pueda enviar información diferente a sus pares. Puede compararse la figura 20 con la figura 17.

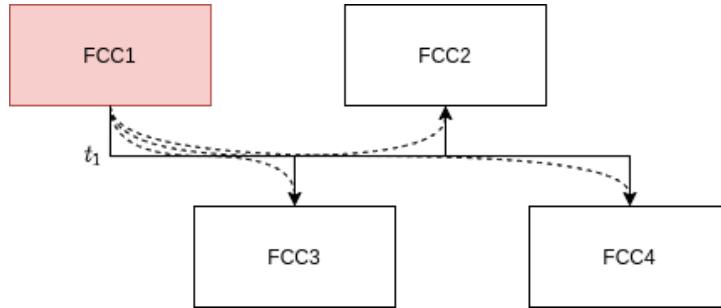


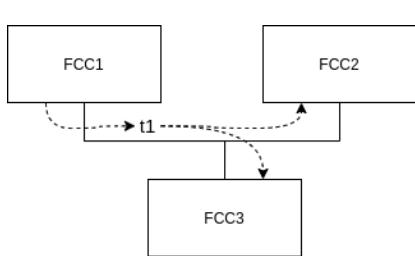
Figura 20: En este caso, la conexión tipo bus no permite el envío de información diferente a los demás miembros. La FCC1 envía el valor t_1 y todos sus pares reciben el mismo valor.

Como contrapartida, debido a que los nodos comparten canal de comunicación, estos deben tomar turnos para enviar la información a sus pares. De otra forma, habría una colisión en el bus y la información nunca llegaría a su destino. Sumado a esto, el bus se convierte en un punto singular de falla, ya que es posible que un problema en el bus deje a los nodos incomunicados.

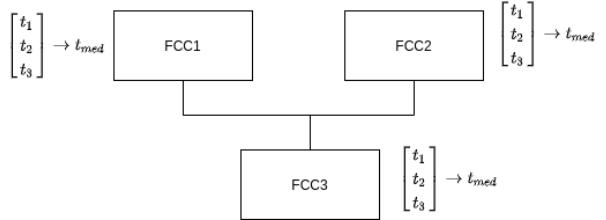
Al igual que como se hizo en la sección 4.2.4, se analiza el problema del consenso utilizando un bus. El ejemplo que se presentó anteriormente fue el necesario para lograr una sincronización entre las FCCs y se mostró que el enviar información distinta a cada computadora de vuelo puede romper el sincronismo

muy fácilmente.

Como ya se mencionó, las FCCs deben tomar turnos para utilizar el medio físico. En las próximas secciones se explicará cómo se puede lograr esto, aquí se asume que las FCCs respetan sus turnos para utilizar el medio físico compartido. En la figura 21a, la FCC1 accede al medio y envía su valor de timestamp. Las demás FCCs reciben el mismo valor, por estar conectadas al mismo bus de comunicación. Luego, las FCC2 y 3 repiten esto mismo. En la figura 21b se muestra que todas tienen la misma información respecto de sus pares. Luego por ejemplo, si calculan un promedio, llegarán al mismo resultado y se sincronizarán correctamente.



(a) La FCC1 envía su *timestamp* hacia las demás.



(b) Luego de finalizar los intercambios, todas las FCCs llegan al mismo resultado de *timestamp* para sincronizarse.

Figura 21: Debido a la existencia del bus, las FCCs no pueden mentir acerca de su *timestamp*. Luego, todas llegan a un consenso de manera casi trivial.

A partir de este análisis, se puede ver que para el caso de un sistema de tiempo real con un único bus de comunicaciones, el problema del consenso es mucho más sencillo que lo que se muestra en *The Byzantine Generals Problem*. De todas maneras, lo que se presenta aquí es un primer análisis, ya que se ha asumido que no hay colisiones en el bus y que los nodos se encuentran sincronizados. Se concluye que, para que la computadora de vuelo pueda implementar distintos mecanismos de tolerancia a fallas, esta debe contar con una interfaz que le permita la comunicación a través de un bus de comunicaciones.

5. Diseño y Construcción de la Computadora de Vuelo

Como se mencionó en la sección 2, la computadora de vuelo es el elemento central de un vehículo aéreo no tripulado. Su tarea principal y la más importante es la de ejecutar los algoritmos de guiado, navegación y control para mantener estable al vehículo y guiarlo en su trayectoria.

En esta sección se presentan los criterios tenidos en cuenta para el diseño y la construcción de la computadora de vuelo. Se presentan las distintas funcionalidades y el análisis de la selección de distintos componentes como sensores y circuitos integrados. Finalmente, se describe el circuito implementado y el diseño del PCB.

5.1. Funcionalidades de la Computadora de Vuelo

Un sistema de navegación permite realizar estimaciones de la pose del vehículo, es decir de la posición y orientación. En la figura 22 se muestra un UAV con una terna solidaria a este. La forma de indicar la orientación del vehículo es a través de los ángulos denominados *yaw*, *pitch* y *roll*. Estos expresan las rotaciones entre la terna solidaria al vehículo en su posición actual, y una terna inercial, fija en el espacio.

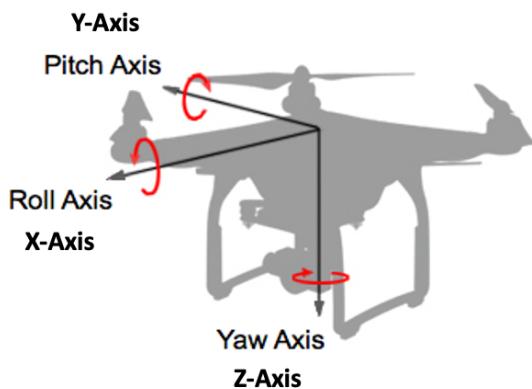


Figura 22: Se utiliza una terna solidaria al vehículo para conocer su orientación en el espacio. Los ángulos *yaw*, *pitch* y *roll* indican rotaciones respecto de una terna inercial.

El principal sistema utilizado en UAVs es el sistema de navegación inercial (INS). Este consiste en realizar estimaciones de posición y orientación a partir de integrar en el tiempo mediciones de aceleración y velocidad de rotación del vehículo. Por ejemplo, podrían conocerse los ángulos de la figura 22 a partir de mediciones de velocidad angular. Estos sistemas no solo se utilizan en UAV sino que también en vehículos tripulados y aviones comerciales.

El hecho de integrar las mediciones de velocidad y aceleración en el tiempo, trae consigo que cualquier error de los acelerómetros y los giróscopos decante en errores de posición y de orientación que crecerán con el paso del tiempo a un ritmo acelerado. Típicamente esto se corrige utilizando otro sistema de navegación auxiliar (como por ejemplo a través de GPS) junto con un filtro de Kalman o un sistema equivalente.

Para obtener las mediciones de aceleración y velocidad angular en la computadora de vuelo, se utilizará una Unidad de Medición Inercial (IMU). Esta comprende acelerómetros y giróscopos triaxiales, los cuales se denominan sensores iniciales. La IMU ofrece una alta tasa de adquisición de datos, en el orden de las decenas de kHz. Esto es de gran importancia para mantener la estabilidad del vehículo, ya que utilizando acelerómetros y giróscopos es posible estimar los ángulos de *pitch* y *roll* del vehículo.

Para realizar todos los cálculos involucrados en el INS, se utiliza una unidad de procesamiento. Todas las tareas involucradas en navegación y control de vuelo son ejecutadas de forma periódica por la computadora de vuelo. Se requiere asegurar un determinismo temporal en la ejecución de las tareas,

del orden de los milisegundos o decenas de milisegundos. La unidad de procesamiento que mejor se ajusta a las necesidades de este trabajo es un microcontrolador de 32 bits, siendo el principal motivo la gran variedad que pueden encontrarse en el mercado, los cuales ofrecen una muy buena performance por un bajo costo. La gran capacidad de cómputo que ofrecen permite que este además puede realizar otras tareas más, como encargarse de la tolerancia a fallas, el almacenamiento de datos y otras tareas relacionadas a la misión del vehículo.

Otro factor importante es el gran nivel de integración que ofrecen, incorporando gran variedad y cantidad de periféricos e interfaces de comunicación. Esto favorece la posibilidad de obtener un diseño para la computadora de vuelo de dimensiones pequeñas.

Algunas de las funcionalidades secundarias que se implementan son el control de LEDs indicadores de propósito general y la capacidad de almacenamiento de datos en una memoria externa, tanto de sensores como de datos pertinentes a la misión del vehículo. De manera de ampliar las capacidades de uso, se incorpora una gran variedad de conectores que facilitan la comunicación con dispositivos y sensores externos. Esto le da una gran versatilidad, permitiendo su utilización en distintas aplicaciones de UAVs y vehículos no tripulados en general.

5.2. Criterios Generales Para la Selección de Componentes

Para el diseño y construcción de la computadora de vuelo, se tuvieron en cuenta algunos criterios comunes a todos los componentes. Estos se mencionan a continuación.

5.2.1. Uso de Componentes de Grado Automotriz

Un sistema de alta confiabilidad trae consigo un costo mayor en su desarrollo y fabricación, respecto de un sistema donde este aspecto no es tan importante. Esto es principalmente debido a la necesidad de utilizar componentes de alta calidad y confiabilidad. Para el desarrollo de este trabajo, se cuenta con un presupuesto limitado, por lo que no puede accederse a componentes con estas características.

En la búsqueda de componentes y distintas alternativas, se encontró que existen algunos que cuentan con una calificación denominada AEC-Q100. Esta calificación creada por el *Automotive Electronics Council* (AEC) define una serie de requerimientos que debe cumplir un componente para garantizar que este se encuentra apto para uso en electrónica automotriz. Los componentes dentro del vehículo se ven sometidos a distintas condiciones de temperatura, humedad, etc., por lo que para considerarse aptos, estos deben garantizar un correcto funcionamiento bajo distintas condiciones. Los requerimientos definidos por el AEC en la especificación AEC-Q100: *Failure Mechanism Based Stress Test Qualification for Integrated Circuits In Automotive Applications* definen una serie de pruebas de estrés de componentes para demostrar la confiabilidad de los mismos.

Teniendo en cuenta que esta calificación otorga un grado de confiabilidad superior, además del hecho de que estos no representan una gran diferencia económica respecto de aquellos de uso comercial, en aquellos casos en los que fue posible se optó por la selección de componentes con esta calificación.

5.2.2. Longevidad

Para el desarrollo de la placa se buscaron componentes para los cuales el fabricante garantice la longevidad antes de entrar en fin de vida útil. En el eventual caso de que un componente deje de ser fabricado, a futuro esto impactará en el diseño, ya que será necesario buscar un reemplazo del mismo, para poder fabricar nuevas unidades. Algunas veces esto es directo, ya que algunos fabricantes ofrecen compatibilidad en los terminales de componentes de mismas funcionalidades. Sin embargo en otros casos, esto puede traer consecuencias como la necesidad de modificar el diseño original, además de volver a realizar pruebas del circuito, utilizando el nuevo componente. En el caso de que se trate de un sensor, incluso puede requerir la modificación del driver utilizado para interactuar con este.

Para la computadora de vuelo de este trabajo, se buscó tener una longevidad de 10 años.

5.2.3. Requerimientos de Conectores

La computadora de vuelo cuenta con una serie de conectores que permiten el agregado de dispositivos externos a la placa. Por una necesidad de compatibilidad con distintos sensores y módulos que son comúnmente utilizados con otras computadoras de vuelo, se utilizaron 2 tipos de conectores distintos. Estos se muestran en la figura 23.

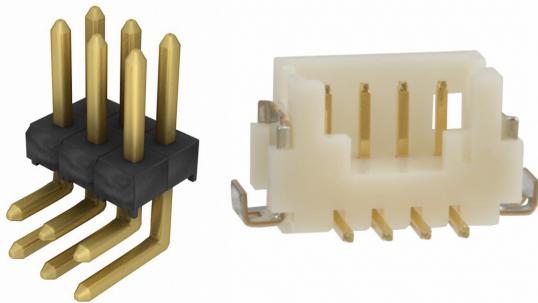


Figura 23: Formato de los conectores utilizados para sensores y módulos externos en la placa. A la izquierda se muestra un conector con formato tira de pines de 0.1" a 90° y a la derecha se muestra un conector de 4 terminales, correspondiente a la serie DF-13 del fabricante Hirose.

5.3. Circuitos y Componentes Seleccionados

A continuación se describe cada una de las partes del circuito que conforman a la computadora de vuelo. Además de los criterios generales ya mencionados, se mencionan los criterios particulares tenidos en cuenta para cada parte del circuito.

5.3.1. Microcontrolador

En la versión anterior de la computadora de vuelo, se utilizó un microcontrolador del fabricante ST, en particular el modelo STM32F722. Este cuenta con un procesador ARM Cortex-M7, que puede utilizarse con una frecuencia máxima de 216 MHz. Cuenta con unidad de punto flotante integrada, además de una memoria flash con 512 KB y una memoria RAM de 256 KB.

A todas las funcionalidades de la computadora de vuelo, en este trabajo se le suman los aspectos relacionados a la tolerancia a fallas. A su vez, se tiene la necesidad de integrar otras funcionalidades que llevan consigo una gran carga computacional. Estas pueden ser propias de la aplicación del vehículo o bien relacionadas a mejorar los algoritmos de navegación y control. Para la versión desarrollada en este trabajo, se buscó actualizar el microcontrolador a uno con mayor rendimiento, sin perder de vista la necesidad de mantener un costo reducido.

Se buscó un microcontrolador del mismo fabricante ST, de manera de tener retrocompatibilidad en el desarrollo del firmware con la versión anterior. De esta forma, muchos de los módulos de firmware que ya se encuentran desarrollados pueden reutilizarse en esta nueva versión. Con estos requerimientos, se analizaron las distintas ofertas del mercado. Además de los aspectos mencionados, se tuvieron en cuenta los periféricos presentes en cada modelo, junto con las capacidades de memorias flash y RAM. Se buscó mantener las capacidades de estas memorias en valores similares a las de la versión anterior de la computadora de vuelo.

Como primera opción surgió la posibilidad de seleccionar alguno de los microcontroladores de la serie STM32H7. Estos cuentan con procesadores ARM Cortex-M7, y pueden llegar a velocidades entre 400 MHz y 550 MHz [31], es decir, pueden llegar hasta a duplicar la performance respecto de la versión anterior de la computadora de vuelo. En la tabla 1 se muestran distintos modelos que fueron considerados durante la selección del microcontrolador.

	STM32F722	STM32H723ZG	STM32H743	STM32H753	STM32H735ZG
Flash [kB]	512	1024	1024/2048	2048	1024
SRAM [kB]	256	564	1024	1024	564
Freq [MHz]	216	550	480	480	550
UART	4	6	4	6	6
USART	4	5	4	5	5
SPI	5	6	5/6	6	6
I2C	3	5	4	4	5
CAN	1	3	2	3	3
ADC	3x12b, 24ch	2x16b, 22 ch; 1x12b, 12 ch	3x16b, 16/28/32ch	1x12b, 12ch 2x16b, 18ch	1x12b, 12ch 2x16b, 16ch
Timers	18: 16b x 16, 32b x 2	21: 16b x 17, 32b x 4	14: 16b x 12, 32b x 2	21: 16b x 17, 32b x 4	21: 16b x 17, 32b x 4
SDMMC	2	2	2	2	2
longevidad	01/2033	01/2033	01/2033	01/2033	01/2033
AEC-Q100	No	No	No	No	No

Tabla 1: Se muestra la comparación de las distintas alternativas que fueron tenidas en cuenta para la selección del microcontrolador. En verde se destaca el componente que tiene las mejores características para cada fila. El modelo STM32F722 corresponde al modelo utilizado en la versión anterior de la computadora de vuelo.

Todos los microcontroladores de la serie STM32H7 presentan mejoras en cuanto a frecuencia de operación, memoria flash y RAM. Sumado a esto, la gran mayoría de estos microcontroladores se encuentran dentro del programa *Longevity Commitment* [32]. El fabricante ST se compromete a mantener la producción de los componentes que se encuentren dentro de este programa durante un período determinado. Todos los microcontroladores de la tabla 1 tienen un período de fabricación de 10 años asegurado por ST, finalizando en 01/2033. Este aspecto es de especial interés teniendo en cuenta la posible necesidad futura de fabricar nuevas placas de la computadora de vuelo.

COMPLETAR POR QUÉ NO SE ELEGIÓ UNO AEC-Q100, QUE ES PORQUE NO HABÍA CORTEX M7 DE ST QUE SEA AEC-Q100

A pesar del análisis y de las comparaciones entre microcontroladores de la serie STM32H7, la selección del microcontrolador se vio limitada por la disponibilidad de componentes encontrada durante el período de desarrollo del circuito.

El microcontrolador seleccionado fue el STM32F746ZG. Este cuenta con un procesador ARM-Cortex M7, 1024 kB de memoria flash y 320 kB de memoria SRAM. Un aspecto relevante de este microcontrolador es que cuenta con 2 interfaces para uso de un bus denominado Controller Area Network (CAN), el cual se utilizará para establecer las comunicaciones relevantes a los algoritmos de tolerancia a fallas. La posibilidad de contar con dos de estas interfaces permite implementar un sistema donde este bus no sea un punto singular de falla.

Suando a las prestaciones, este micocontrolador cuenta con un encapsulado LQFP de 144 terminales, lo que permite realizar muchas conexiones con sensores y componentes a través de distintas interfaces comunes, como SPI, I2C, además de terminales GPIO comunes para interrupciones y manejo de otros módulos.

5.3.2. Sensor IMU

La unidad de medición inercial, IMU por sus siglas en inglés, es el sensor principal utilizado por la computadora de vuelo. Este consiste en un circuito integrado que contiene una serie de sensores iniciales, en particular acelerómetros y giróscopos triaxiales. Los acelerómetros se utilizan para realizar mediciones de aceleración lineal y los giróscopos para medir velocidad angular. A partir de estas mediciones, se pueden aplicar distintos algoritmos de procesamiento para obtener una estimación de la posición y orientación

del vehículo. Las mediciones de aceleración lineal y de velocidad angular que entrega la IMU son referidas a una terna solidaria al componente, como se muestra en la figura 24.

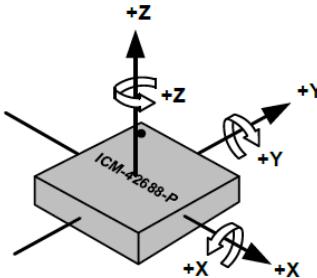


Figura 24: Todas las mediciones que entrega el sensor son reales a una terna solidaria a este. La imagen se extrajo de [33].

Los acelerómetros y giróscopos de la IMU utilizada en este trabajo, se construyen utilizando la tecnología MEMS: *Micro Electro-Mechanical Systems*. Esta consiste en utilizar técnicas de fabricación de circuitos integrados para integrar en el silicio partes que móviles que constituyen los sistemas mecánicos de los acelerómetros y giróscopos. Este tipo de sensores tienen la ventaja de que pueden conseguirse por un bajo costo, además de tener encapsulados muy pequeños, permitiendo obtener un diseño de dimensiones reducidas.

Se hizo una búsqueda de las distintas alternativas existentes para este tipo de sensores. A partir de leer las hojas de datos de distintos fabricantes, se encontró que los parámetros típicamente especificados, tanto para los acelerómetros como para los giróscopos, son los siguientes:

- *Full-scale range*
- *Sensitivity*
- *Scale factor error*
- *Scale factor error vs temp*
- *Offset*
- *Offset vs temp*
- *Offset vs time*
- *Noise*

El primero de ellos, el *Full-scale range* es el rango de medición del sensor. Para los acelerómetros se suele especificar en un rango de $\pm n \times g$, donde n es algún entero y g representa la aceleración de la gravedad. Para los giróscopos, se especifica como $\pm n \times dps$, donde dps significa *degrees-per-second*.

El parámetro *Sensitivity* hace referencia a la resolución. En algunas hojas de datos este parámetro puede encontrarse con unidades de LSB/g para los acelerómetros y en LSB/dps para los giróscopos. Este valor puede resultar confuso de entender, ya que lo que informa es la cantidad de bits por g o la cantidad de bits por dps . En la figura 25 se muestra una captura de la hoja de datos del sensor seleccionado, el ICM42688p.

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	ACCEL_FS_SEL =0	± 16			g	2
	ACCEL_FS_SEL =1	± 8			g	2
	ACCEL_FS_SEL =2	± 4			g	2
	ACCEL_FS_SEL =3	± 2			g	2
ADC Word Length	Output in two's complement format	16			bits	2, 5
Sensitivity Scale Factor	ACCEL_FS_SEL =0	2,048			LSB/g	2
	ACCEL_FS_SEL =1	4,096			LSB/g	2
	ACCEL_FS_SEL =2	8,192			LSB/g	2
	ACCEL_FS_SEL =3	16,384			LSB/g	2

Figura 25: Extracto de la hoja de datos del sensor ICM42688p. Se muestra parte de las especificaciones para los acelerómetros.

La imagen muestra que el sensor permite seleccionar distintos rangos de escala para las mediciones del acelerómetro. Por ejemplo, si se selecciona el rango $\pm 2g$, la hoja de datos especifica una resolución de 16384 LSB/g . Una mejor forma de entender este parámetro sería si se considera la inversa, es decir, la resolución del ADC. En este caso sería de $61,04 \cdot 10^{-6} g$. Luego para un rango de $\pm 4g$ la resolución es de 8192 LSB/g , es decir, $122,07 \cdot 10^{-6} g$. Este valor es el doble del anterior y tiene sentido dado que se está midiendo un rango mayor de aceleraciones utilizando la misma cantidad de bits, en este caso 16 según lo especificado en la hoja de datos.

Para entender los parámetros, *scale factor error*, *offset* y *noise* se plantea un modelo sencillo de medición, tanto para acelerómetros como para giróscopos [34]. Este se presenta en la ecuación (4), donde S es el *scale factor error*, $\omega_b(t)$ es el *offset* el cual es variable con el tiempo, $\eta \sim \mathcal{N}(0, \sigma^2)$, ω_m es el valor medido y ω sería la velocidad angular verdadera para el giróscopo.

$$\omega_m = (1 + S)\omega + \omega_b(t) + \eta \quad (4)$$

A su vez, en las hojas de datos se especifica la dependencia de estos parámetros con el tiempo y con la temperatura, como es el caso del *scale factor error*.

Para tener un criterio de selección del sensor IMU, se siguió el análisis planteado en [34]. Este paper presenta un análisis de los parámetros de los acelerómetros y grioscopos y su impacto en las estimaciones de posición en sistemas de navegación inercial (INS). En este se concluye que los parámetros más importantes para la selección del sensor son:

- Estabilidad del offset de los acelerómetros (Offset vs time).
- Estabilidad del offset de los giróscopos (Offset vs time).
- Ruido de los giróscopos (Noise).
- Error de escala del giróscopo (Scale factor error).

Se buscaron modelos de IMUs de distintos fabricantes, para comparar características. Existe una gran cantidad de fabricantes y de componentes para seleccionar. Se buscaron componentes que sean accesibles y que no tengan un costo muy elevado. Existen IMUs de una excelente calidad, pero que tienen precios que no están al alcance (cientos o miles de dólares). Con este criterio, se realizó una comparación entre distintos modelos de sensores. En la tabla 2 se muestra una comparación de los distintos sensores considerados. Sumado a esto, se tuvo en consideración otro aspecto que fue mencionado anteriormente, la longevidad del componente.

	ICM42688	LSM6DSR	IIM-42652	BMI088	ASM330LHHX
$b_{accel}(t)$	N/A	N/A	N/A	N/A	40 μ g
$b_{gyro}(t)$	N/A	N/A	N/A	2°/h	3°/h
η_{gyro}	2,8mdps/ \sqrt{Hz}	5mdps/ \sqrt{Hz}	3,8mdps/ \sqrt{Hz}	14mdps/ \sqrt{Hz}	5 – 12mdps/ \sqrt{Hz}
S_{gyro}	0,5 %	1 %	0,5 %	1 %	2 %
longevidad	N/A	N/A	10 años, dic. 2020	N/A	15 años, mayo 2022
AEC-Q100	No	No	No	No	Sí

Tabla 2: Se muestra la comparación de las distintas alternativas que fueron tenidas en cuenta para la selección del sensor. En verde se destaca el componente que tiene las mejores características para cada parámetro.

Lo primero que llama la atención es el hecho de que muchos de los sensores no especifican todos sus parámetros. Una sola de las alternativas consideradas tiene disponible toda la información en su hoja de datos. Esto dificulta mucho la selección de un componente. A priori, se seleccionó el sensor ASM330LHHX por el hecho de ser el único que ofrece toda la información en su hoja de datos, además de ser de grado automotriz y tener una longevidad garantizada de 15 años. Teniendo en cuenta aspectos de confiabilidad, resulta esencial el hecho de conocer los parámetros del sensor.

Durante la selección del sensor hubo otro aspecto importante que se tuvo en cuenta y es el hecho de la compatibilidad con el software desarrollado por el laboratorio, para computadoras de vuelo anteriores a la de este trabajo. La versión anterior contaba con un sensor IMU ICM20602, del fabricante TDK. El laboratorio cuenta con bibliotecas de código ya desarrolladas para este sensor. Este presentó excelentes resultados, lo que sienta un antecedente importante en la selección de componentes del mismo fabricante. En la tabla 3 se muestra una comparación entre el sensor anterior ICM20602 y el sensor seleccionado ICM42688.

	ICM20602	ICM42688
Año	2016	2021
Giróscopos		
Full Scale Range[dps]	$\pm 250/500/1000/2000$	$\pm 15/31/62/125/250/500/1000/2000$
Scale Factor Error[%]	1,0 @ 25°C	0,5 @ 25°C
Scale factor error vs temp[%/°C]	0,016 @ -40°C - 85 °C	0,005 @ 0°C - 70 °C
Offset[dps]	± 1	$\pm 0,5$
Offset vs temp[dps/°C]	0,01	0,005
Offset vs time[°/h]	N/A	N/A
Noise[mdps/ \sqrt{Hz}]	4	2,8
Acelerómetros		
Full Scale Range[g]	$\pm 2/4/8/16$	$\pm 2/4/8/16$
Scale Factor Error[%]	1,0 @ 25°C	0,5 @ 25°C
Scale factor error vs temp[%/°C]	0,012 @ -40°C - 85 °C	0,005
Offset[mg]	± 25	± 20
Offset vs temp[mg/°C]	X,Y: $\pm 0,5$, Z: ± 1	$\pm 0,15$
Offset vs time[μ g/h]	N/A	N/A
Noise[μ g/ \sqrt{Hz}]	100	X,Y: 65, Z: 70

Tabla 3: Se muestra la comparación del sensor ICM20602 y el sensor seleccionado ICM42688.

Para el diseño del circuito se siguieron las recomendaciones en la hoja de datos del componente. Este sugiere incluir una serie de capacitores de desacople en los terminales de alimentación del componente. Se elige utilizar una comunicación SPI en modo esclavo, donde el maestro es el microcontrolador, ya que pueden obtenerse mayores velocidades de comunicación, respecto de otros protocolos como I2C. De esta forma se logra una alta tasa de adquisición de datos de acelerómetros y giróscopos. Como ya fue

mencionado, esto resulta clave para mantener la estabilidad del vehículo. El circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático.

Dado que el sensor IMU es un esclavo en la comunicación SPI, este solo puede comunicarse con el microcontrolador cuando este último le permita hacerlo. Para que la IMU pueda informar el momento en el que se obtuvo una nueva lectura, el sensor dispone de una salida digital la cual puede conectarse a una entrada digital del microcontrolador. Cuando el microcontrolador detecta un cambio de nivel en esta entrada, el mismo procede a solicitar el dato a través de la comunicación SPI. En la figura 26 se muestra un esquema de la conexión entre el controlador y el sensor IMU.

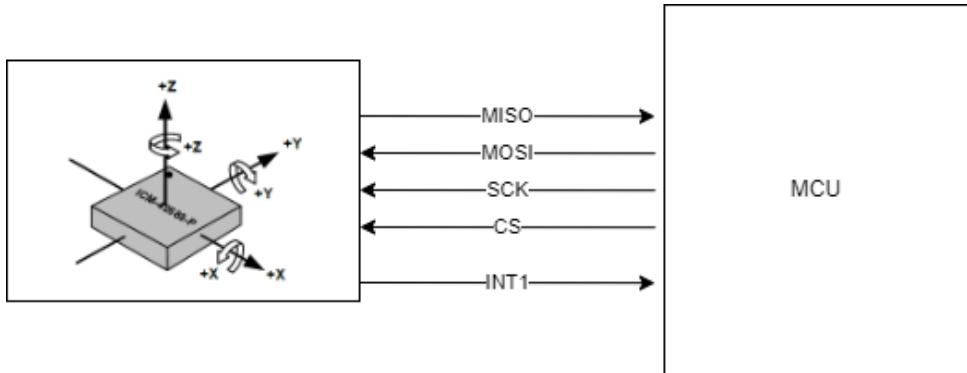


Figura 26: Líneas de comunicación entre la IMU y el microcontrolador.

5.3.3. Barómetro

El barómetro se utiliza para obtener una estimación precisa de la altitud a la que está operando el UAV, a partir de mediciones de presión. Este es uno de los sensores complementarios al INS que se incorporan en la computadora de vuelo. Al igual que la IMU, el barómetro que se utiliza corresponde a un sensor de tecnología MEMS. En particular, los barómetros MEMS cuentan con un sistema capaz de medir la presión absoluta, es decir respecto al 0 de presión. Estos cuentan con una cavidad integrada dentro del chip que se encuentra (idealmente) a presión 0. Para medir la presión, se coloca una membrana sobre la cavidad. En la figura 27 se puede apreciar el efecto de la presión externa.

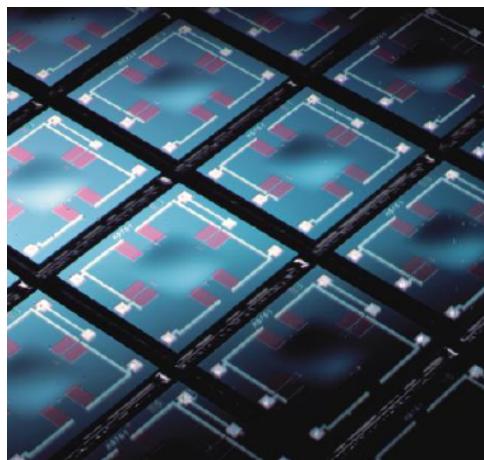


Figura 27: Sensores de presión sobre una oblea de silicio [35].

Sobre esta membrana se colocan resistores de efecto piezoresistivo en configuración puente de Wheatstone. El efecto de la presión genera compresiones y deformaciones en los resistores [36], lo que se traduce

en un desbalance del puente. A partir del sensado de la diferencia de potencial, se mide la presión aplicada sobre la membrana.

Al igual que con el sensor IMU, se hizo una búsqueda de las distintas alternativas. Los parámetros típicamente especificados son los siguientes:

- *Full-scale range*
- *Absolute Accuracy*
- *Relative Accuracy*
- *Solder Drift*
- *Offset vs temp*
- *Offset vs time*
- *Noise*

Como se puede ver, estos son similares a los de la IMU. Las diferencias se encuentran en los parámetros *Absolute Accuracy*, *Relative Accuracy* y *Solder Drift*.

Se puede plantear un mismo modelo de medición según la ecuación 4 pero para la presión.

$$P_m = (1 + S)P + P_b(t) + \eta \quad (5)$$

En el caso de la IMU, el parámetro *scale factor error* se refiere al término S y el *offset* al término P_b . En el caso del barómetro, estos valores se encuentran especificados de otra manera. Si se quiere medir una presión P , el error de medición será $\Delta P = S P + P_b(t) + \eta$. El término $S P + P_b(t)$ corresponde al parámetro *absolute accuracy* [37]. Este error es introducido debido a que la cavidad dentro del sensor no se encuentra a presión 0 perfecta, sino que a un pequeño valor [35]. Por otro lado, el término $S P$ se lo denomina *relative accuracy*. Este hace referencia a mediciones diferenciales de presión. Algunos barómetros MEMS traen consigo una funcionalidad para realizar una compensación de offset. Esto dejaría como parámetro de interés para mediciones de presión a la *relative accuracy*, la cual hace referencia al error introducido para mediciones de variaciones de presión.

El parámetro *solder drift* se refiere al offset que se introduce por el propio proceso de soldadura [37]. Este offset también puede ser compensado a través de la calibración del barómetro.

Se buscaron modelos de barómetros de distintos fabricantes, para comparar características, teniendo en cuenta la accesibilidad y el bajo costo. En la tabla 4 se muestra una comparación de los distintos barómetros considerados. Sumado a esto, se tuvo en consideración otro aspecto que fue mencionado anteriormente, la longevidad del componente.

	BMP390	BMP581	ICP-20100	LPS22HH	ILPS22QSTR	DPS368
Full scale range [hPa]	300 - 1250	300 - 1250	260 - 1260	260 - 1260	260 - 1260 260 - 4060	300 - 1200
absolute acc [hPa]	$\pm 0,5$	$\pm 0,3$	$\pm 0,2$	$\pm 0,5$	$\pm 0,5$	± 1
realitive acc [hPa]	$\pm 0,03$	$\pm 0,06$	$\pm 0,01$	$\pm 0,025$	$\pm 0,015$	$\pm 0,06$
longevidad	N/A	N/A	N/A	N/A	10 años enero 2023	N/A

Tabla 4: Se muestra la comparación de las distintas alternativas que fueron tenidas en cuenta para la selección del sensor.

Las dos alternativas que se evaluaron son los sensores ICP-20100 y el ILPS22QSTR. El primero de ellos presenta las *absolute accuracy* y *relative accuracy* más bajas de entre todas las opciones evaluadas. El sensor ILPS22QSTR presenta características similares y además tiene la particularidad de que el fabricante garantiza su fabricación por 10 años, hasta enero de 2033 [38]. Finalmente el sensor seleccionado fue este último.

En este caso también se tomó como guía el circuito de la hoja de datos del componente. La interfaz de comunicación seleccionada es I2C. Se prefiere utilizar I2C en lugar de SPI ya que puede aprovecharse el uso del mismo bus al que se conecta el barómetro, para conectar otros sensores y dispositivos. De esta manera, se ahorra la cantidad de pistas y conexiones en el diseño del PCB. Si bien I2C tiene un funcionamiento más lento que SPI, las mediciones del barómetro no son tan críticas como las de la IMU. Este sensor, a diferencia de la IMU, no cuenta con una línea de interrupción, por lo que los datos deben obtenerse por *polling* de forma periódica. El circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático.

5.3.4. Magnetómetro

El magnetómetro es otro de los sensores complementarios al INS. Este se utiliza para obtener estimaciones del ángulo de *yaw* del vehículo. En la figura 28 se muestran las componentes del campo magnético terrestre medidas por el sensor. A partir del ángulo entre las componentes en H_y y H_x , puede obtenerse el ángulo de *yaw* del vehículo. Si se rota al sensor alrededor de su eje z, la dirección del campo magnético terrestre permanecerá constante. Esto es lo que permite su uso como referencia para obtener el ángulo de *yaw*.

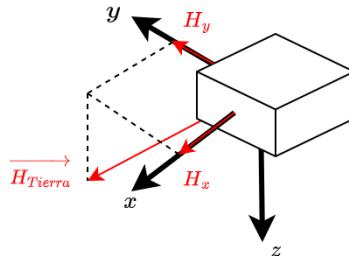


Figura 28: Se muestra un ejemplo con la terna solidaria al magnetómetro. Se muestra la dirección del campo magnético de la Tierra y las componentes expresadas en la terna solidaria al sensor, para los ejes x e y.

Para medir las componentes del campo magnético, el sensor utiliza una serie de materiales resistivos que son sensibles al campo magnético aplicado sobre estos. Se colocan 4 resistores en configuración puente de Wheatstone y se estiman las componentes del campo magnético a partir del desbalance de este.

Para la selección del componente se hicieron comparaciones de las características de varias alternativas. Los parámetros típicamente especificados para el magnetómetro son los siguientes:

- *Full-scale range*
- *Scale Factor Error*
- *Scale Factor Error vs Temp*
- *Offset*
- *Offset vs temp*
- *Noise*

Se buscaron modelos de magnetómetros disponibles. Al igual que con el resto de componentes, se redujo la búsqueda a aquellos que sean de bajo costo. En la tabla se muestra una comparativa de los modelos considerados para la selección final.

	HMC5883L	BMM150	RM3100	MMC5983MA
Scale Factor Error	$\pm 0,1\% @ \pm 200 \mu T$	1 %	$\pm 0,5\% @ \pm 200 \mu T$	0,1 % @ $\pm 400 \mu T$
Scale Factor Error Over Temp	0.3 %/C @ -40°C - 125 °C	N/A	N/A	0.07 %/°C @ -40°C - 105°C
Offset @25°C	N/A	$\pm 40 \mu T$	N/A	$\pm 50 \mu T$
Offset Over Temp	N/A	N/A	N/A	$\pm 2 nT/°C$ @ -40°C - 105°C
Noise	200 nT @ FSR = $\pm 88 \mu T$	300 nT @ 25°C, ODR = 20 Hz	30 nT	40nT @ ODR = 50 Hz 60 nT @ ODR = 100 Hz 80 nT @ ODR = 200 Hz 120 nT @ ODR = 400 Hz
AEC-Q100	No	No	No	Sí

Tabla 5: Se muestra la comparación de las distintas alternativas que fueron tenidas en cuenta para la selección del magnetómetro. En verde se destaca el modelo con las mejores prestaciones para cada especificación.

El modelo RM3100 es el que posee el nivel más bajo de ruido. Este componente se utiliza en computadoras de vuelo comerciales, pero además fue utilizado con éxito en el proyecto Artemis de la NASA. A pesar de las bondades que ofrece, el costo de este componente en el momento de la selección era entre 5 a 10 veces superior con respecto a los otros. Finalmente el sensor elegido fue el modelo MMC5983MA. Este es el único de todas las alternativas consideradas que contiene información de todos los parámetros. Además de tener unas muy buenas características, el mismo es de grado automotriz, lo que le da todavía más robustez.

El efecto del campo magnético producido por los motores del UAV afectan las mediciones del sensor, volviéndolo prácticamente inutilizable para la navegación del vehículo. Para evitar este problema, el magnetómetro se montará en una posición elevada respecto de la computadora de vuelo. Esto quiere decir que este sensor no se montará sobre la placa, sino que se conectarán como un sensor externo, utilizando alguno de los conectores de la placa.

5.3.5. Interfaz de Comunicación CAN

Como ya fue mencionado, la computadora de vuelo requiere que la comunicación entre sus réplicas sea través de un bus. Para ello, la placa debe contar con un circuito con una interfaz de comunicación con el mismo. A partir de lo presentado hasta aquí, el único requerimiento es el método de acceso al medio, el cual debe ser controlado por el tiempo. Teniendo en cuenta que se trata de un trabajo realizado con componentes COTS, el hardware a utilizar debe ser de fácil acceso y con costos razonables. Para el desarrollo de este trabajo, se seleccionó el bus *Controller Area Network* (CAN)[29]. Si bien su método de acceso al medio no es TDMA, existe una extensión del protocolo que justamente busca incorporar esta funcionalidad en otra capa superior.

El protocolo CAN fue desarrollado para ser usado en la industria automotriz, como bus de comunicación que conecta distintos módulos dentro de un automóvil. El objetivo de su desarrollo fue similar al motivo por el cual se desarrolló el bus ARINC 629 en aviones, reemplazar la gran cantidad de cables dentro del vehículo por un bus simple. El protocolo se corresponde con el modelo OSI y la especificación original define las capas física y de enlace.

A diferencia de otros protocolos típicos en sistemas embebidos como I2C o SPI, el protocolo CAN no tiene el formato maestro-esclavo. La comunicación se da entre miembros del bus denominados nodos, los cuales se encuentran conectados a los mismos 2 cables. Esto se muestra en la figura 29.

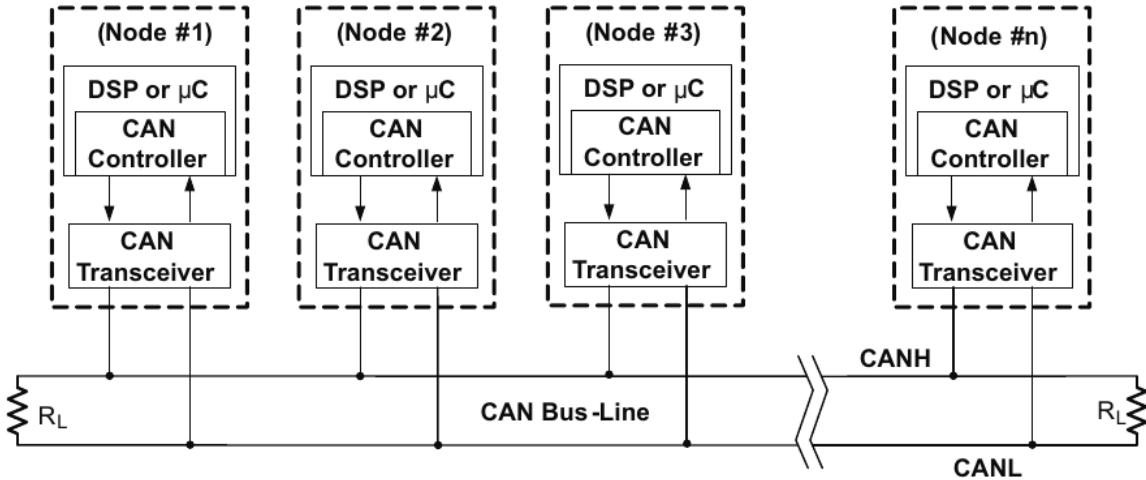


Figura 29: Todos los nodos se encuentran conectados al mismo bus de comunicaciones. El bus CAN se compone de dos cables, CAN-H y CAN-L, terminados en sus extremos por resistencias de adaptación. La imagen se extrajo de [30].

La impedancia característica del bus debe ser de 120Ω . Es común agregar resistores de terminación en ambos extremos, para evitar reflexiones. En algunos casos pueden llegar a encontrarse aplicaciones donde los resistores de terminación se incluyen dentro de alguno de los nodos del bus. Esto no es recomendable, ya que si el mismo se desconecta de forma accidental del bus, todas las comunicaciones entre los demás nodos se verán perjudicadas debido a la pérdida del resistor de adaptación.

La información enviada por los nodos a través de estos 2 cables es en formato de señal diferencial, lo que vuelve robusta a la comunicación, reduciendo las emisiones electromagnéticas generadas por este. A su vez, es común que el bus sea cableado como un par trenzado, lo que atenúa señales de modo común, producto de cualquier acoplamiento.

AGREGAR UNA IMAGEN DE OCSILOSCOPIO DONDE SE VEAN LAS SEÑALES DIFERENCIALES DE CAN.

El protocolo CAN define dos estados para el bus, *recessive* y *dominant*. Cuando no hay actividad en el bus, tanto la línea de CAN-H como la de CAN-L se encuentran a la misma tensión constante. Esto corresponde al estado *recessive* y equivale a un 1 lógico. Cuando se quiere enviar un 0 lógico, el transciever del nodo transmisor fija la tensión de las líneas CAN-H y CAN-L de tal forma de generar una tensión diferencial $V_{CAN-H} - V_{CAN-L} \geq 1,5V$. Esto se muestra en la figura 30.

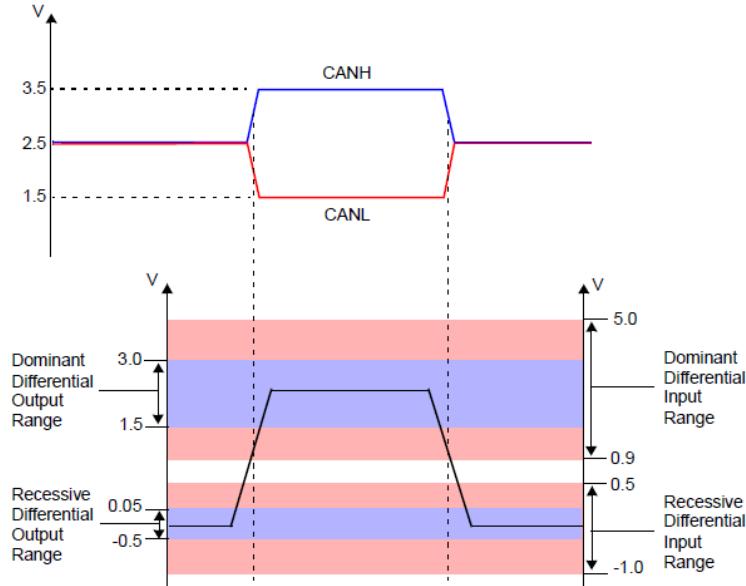


Figura 30: Se muestran los estados recessive y dominant del bus CAN y sus equivalentes lógicos.

Los nodos solamente actúan sobre el bus cuando quieren fijar un estado *dominant*. Cuando se quiere fijar un estado *recessive*, no se actúa sobre el bus ya que este es su estado por defecto. Esto permite que varios nodos puedan actuar al mismo tiempo. En caso de que esto suceda, el estado *dominant* (de allí su nombre) predominará sobre el estado *recessive*.

Existen muchas versiones del protocolo CAN, en este trabajo se utiliza la versión CAN High Speed. Esta define una velocidad máxima de transferencia de 1 Mbps, para un bus de hasta 40 m de longitud y 30 nodos conectados. Se recomienda que la conexión entre cada nodo y el bus no sea de más de 30 cm. El hecho de poder contar con hasta 30 nodos expande las posibilidades de uso, más allá de ser el medio principal de comunicación utilizado para el sistema redundante. Por ejemplo, distintos sensores o incluso actuadores como los motores del vehículo podrían conectarse al bus. **Acá tendría que agregar alguna referencia a este uso del bus CAN.**

En la figura 29 se muestra que cada nodo se compone de 2 elementos, el *transceiver* y el *controller*. El primero de ellos forma parte de la capa física y es un circuito que convierte las señales diferenciales del bus en señales de modo común. Luego las señales de modo común son transferidas al elemento *controller*. Este típicamente se encuentra implementado en hardware dentro de una unidad de procesamiento como un microcontrolador, como es el caso de este trabajo.

Una trama del protocolo CAN se compone de varios campos. Además, se definen 2 tipos de tramas, estándar, figura 31 y extended, figura 32.

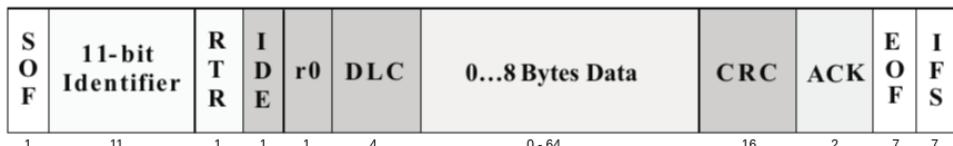


Figura 31: Se muestran los campos de una trama CAN estándar. Debajo de cada campo se indica la cantidad de bits. La imagen se extrajo de [30].

- SOF: Inicio de una nueva trama en el bus.
- Identifier: Indica el contenido del campo de datos.

- RTR: Dominante para *data frames*, recessive para *remote frames*. Estas últimas se utilizan para solicitar a otro nodo que envíe determinado mensaje. Su formato es el mismo, solo que no pueden contener bytes en su campo de datos.
- IDE: Si es dominante, se trata de una trama estándar. Si no, es una trama extended.
- r0: Reservado.
- DLC: Cantidad de bytes de datos enviados.
- Data: Datos útiles, entre 0 y 8 bytes.
- CRC: Chequeo de la integridad del mensaje de la trama.
- ACK: Se compone de 2 bits denominados ACK SLOT Y ACK DELIMITER. El emisor deja estos campos en recessive y cuando algún nodo recibe el mensaje, fuerza un nivel dominant en el bus, indicando que se recibió el mensaje correctamente.
- EOF: Fin de trama.
- IFS: Espacio antes de enviar la próxima trama, de 7 bits.



Figura 32: Se muestran los campos de una trama CAN extendida. Debajo de cada campo se indica la cantidad de bits. La imagen se extrajo de [30].

La trama extended mantiene los mismos campos pero incorpora otros 2 más:

- SRR: Debe ser recessive para extended frames.
- r1: Reservado.

Debido a que no existe un nodo que sea maestro, todos los miembros del bus pueden iniciar la transmisión de un mensaje. El método de acceso al medio se denomina *Carrer Sense Multiple Access with Collision Detection and Arbitration on Message Priority* (CSMA/CD+AMP). Antes de transmitir un mensaje, el nodo sensa el bus y en caso de que esté libre, intenta utilizarlo para transmitir un mensaje.

En el eventual caso en que más de un nodo detecte el medio sin uso, estos intentarán transmitir a través del bus al mismo tiempo. Esta situación se encuentra contemplada por el protocolo a través de un mecanismo que resuelve la colisión mientras que asegura la transmisión del mensaje con la prioridad más alta. El mensaje con la prioridad más alta será aquel que tenga en su campo Identifier el valor más bajo. De aquí se desprende la necesidad de que en un mismo bus no haya mensajes distintos con el mismo campo ID.

En la figura 33 se muestra un ejemplo donde se resuelve una colisión. Tanto el nodo 1 como el nodo 2 quieren utilizar el bus CAN. Ambos comienzan a inyectar su ID correspondiente. En algún momento, ocurre una discrepancia entre el ID inyectado por ambos nodos. Aquel con el campo dominant, es decir 0, gana y completa su transmisión, en este caso el nodo 2. El nodo 1 que quiso enviar un 1 por su lado, detecta un 0. Esto genera que detenga su transmisión, dejando que los demás nodos utilicen el medio.

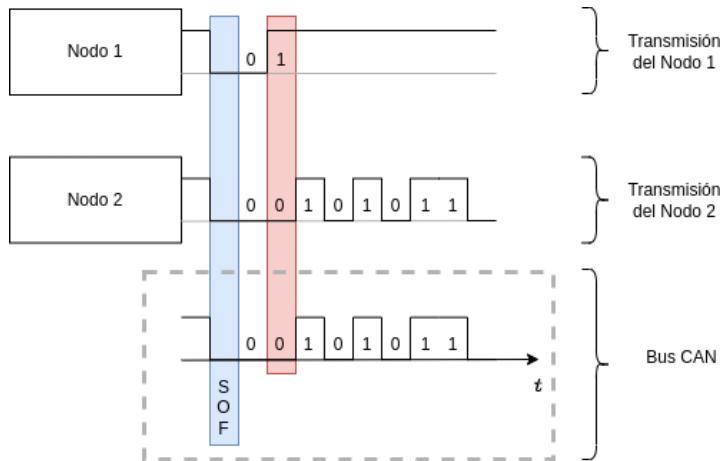


Figura 33: Mecanismo para la detección de colisiones. El nodo 2 gana por prioridad y completa su transmisión, mientras que el nodo 1 deja de usar el bus, aguardando a que el nodo 2 finalice.

El protocolo CAN de por sí, no cuenta con un acceso al medio controlado por el tiempo, sino que es dominado por eventos (*event-triggered*), ya que varios nodos pueden iniciar la transmisión de un mensaje en cualquier momento. En el estándar ISO 11898-4[39] se define una extensión del protocolo CAN denominada *Time-Triggered CAN* (TTCAN). Esta justamente fue desarrollada con el objetivo de utilizar el protocolo en aplicaciones de alta confiabilidad. Para ello, TTCAN incorpora un mecanismo de comunicación entre nodos a través de un scheduling estático, el cual es respetado por todos. A cada mensaje del scheduling se le asigna una ventana de tiempo y el mismo se repite de forma periódica.

El micronotrolador seleccionado para la computadora de vuelo, cuenta con 2 *controller* embebido, el periférico bxCAN [40]. Cada uno de ellos cuenta con 2 líneas de comunicación con el transciever, CAN TX y CAN RX, las cuales se utilizan para enviar y recibir las señales de modo común al *transciever*. En cuanto a este último, se trata de un componente que es ampliamente utilizado en la electrónica automotriz, por lo que hay mucha disponibilidad. Existen transcievers que utilizan distintos niveles de tensión en sus salidas. La gran mayoría de los componentes de la computadora de vuelo utilizan tensiones de 3,3 V para su funcionamiento, por lo que se buscó algún componente que pueda trabajar con este nivel de tensión. El componente seleccionado es el SN65HVD230 de Texas Instruments [41], el cual es compatible con la especificación de capa física de CAN, ISO 11898-2. En la figura 34 se muestra la comunicación del microcontrolador con el bus CAN a través del *transciever*. Este cuenta con una protección por exceso de temperatura, donde el componente pone sus salidas CAN-H y CAN-L en alta impedancia, de manera de no perturbar al resto de los nodos. Además, cuenta con una funcionalidad que permite detectar si el transciever fue desconectado del bus, fijando un estado alto constante en su salida RX hacia el *controller*, informándole de la situación.

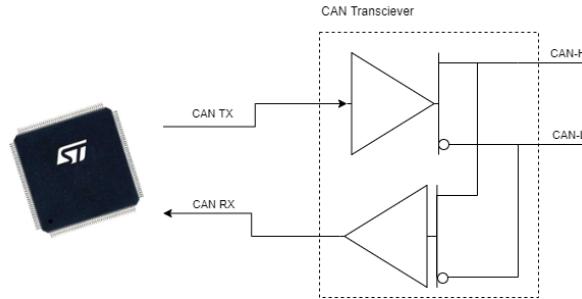


Figura 34: Se muestran las conexiones entre el *controller* embebido en el microcontrolador y el *transciever*, a través de las líneas CAN TX y CAN RX. **CAMBIAR ESTA IMAGEN POR UNA MÁS CLARA Y PROLIJA**

Como se muestra en la figura 34, el *transciever* cuenta con un terminal R_s que permite controlar el tiempo de crecimiento y de decrecimiento de las líneas CAN-H y CAN-L. Al realentizar el tiempo de crecimiento en las líneas CAN-H y CAN-L, se atenúa el contenido armónico de las más altas frecuencias, disminuyendo emisiones. Por ejemplo, si se coloca un resistor de $10\text{ k}\Omega$ en el terminal 8 denominado R_s , la hoja de datos indica un slew-rate de $15\text{ V}/\mu\text{s}$.

La especificación de la capa física de CAN no indica un conector a utilizar. Se buscó seleccionar alguno que no ocupe demasiado espacio al ser montado en el PCB. En [42] se mencionan algunas recomendaciones de conectores. Este es un documento publicado por la organización internacional sin fines de lucro, *CAN in Automation*, que se dedica a publicar recomendaciones y especificaciones relacionadas al uso del bus CAN. Dentro de las opciones que da este documento, se buscó alguno que tenga dimensiones razonables a lo requerido para la placa. De entre todas las opciones se seleccionó el conector que corresponde a la especificación de un protocolo CAN desarrollado para ser usado en drones, DroneCAN [43], el conector JST GH 4. Por cuestiones de disponibilidad, se seleccionó otro componente similar a este y que es del mismo fabricante de otros de los conectores utilizados para la computadora de vuelo, correspondiente a la serie DF-13 del fabricante Hirose. Este puede ver en la figura 35. Se utilizaron dos de estos conectores, uno por cada *transciever*. Al ser de pequeñas dimensiones, la inclusión de ambos conectores no ocupa demasiado espacio en la placa.



Figura 35: Se muestra el conector utilizado para las interfaces CAN. El mismo cuenta con 3 terminales, de los cuales 2 de se utilizaron para la señal CAN diferencia. El terminal restante corresponde al GND de la placa.

En cuanto al circuito implementado, se tomaron como punto de partida las recomendaciones de la hoja de datos del transciever SN65HVD230. Este recomienda el agregado de capacitores de desacople en las líneas de alimentación. El circuito completo de la interfaz CAN puede encontrarse en el Apéndice A: Circuito Esquemático.

5.3.6. Circuito de Alimentación

Para alimentar el microcontrolador y los demás componentes, se incluye una fuente de alimentación de 3,3 V. Si bien el uso de una fuente conmutada ofrecería mejores características de eficiencia, se opta por utilizar un circuito con un regulador lineal. El motivo principal es porque se prioriza reducir las dimensiones de la placa y simplificar el diseño. Se incorporó un conector, de manera de poder alimentar la placa utilizando una fuente externa de 5 V, a partir de los cuales se generan los 3,3 V.

El circuito implementado se comprende del regulador lineal, junto con capacitores de entrada y de salida. Por cuestiones de confiabilidad, se buscó un componente que sea de grado automotriz. Además, se buscó utilizar un encapsulado SOT-223-3, figura 36. Estos componentes son de fácil acceso en el mercado local, lo que facilita el armado final de la placa.

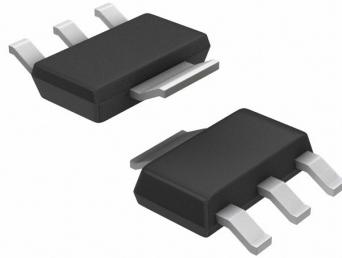


Figura 36: Encapsulado SOT-223-3 seleccionado para el regulador lineal. El terminal de mayor tamaño se encuentra conectado al terminal de tensión de salida, 3,3 V.

El modelo seleccionado es el ZLDO1117QG33TA de DIODES Incorporated [44], un regulador low-dropout de grado automotriz con capacidad para entregar hasta 1 A de corriente. Este valor puede resultar excesivo, pero puede ser de utilidad en caso de que sea necesario utilizar muchos módulos y sensores externos a la placa.

Se siguió el circuito recomendado por la hoja de datos del fabricante, donde se coloca un capacitor a la entrada del regulador y otro en su salida. Para el capacitor de entrada se seleccionó un capacitor cerámico multicapa de valor $4,7 \mu F$. Estos tienen una ESR muy baja, lo que permite que se descarguen rápido. Esto es importante ya que este capacitor se encargará de proveer corriente al regulador, en caso de que ocurra un escalón en la corriente consumida por la carga. Este capacitor ayuda a mantener la tensión de entrada del regulador, en caso de que la fuente conectada a la entrada del regulador presente un drop-out.

En cuanto al capacitor de salida, se tuvo en cuenta el hecho de que este es utilizado para la compensación del regulador [45]. Al tratarse de un circuito a lazo cerrado, debe asegurar su estabilidad para tener un correcto funcionamiento. El mecanismo de compensación es a través de un compensador en adelanto, formados por dos capacitores de salida C_o y C_b , junto con R_o , la ESR del capacitor C_o . Este compensador incrementa el margen de fase, evitando las respuestas oscilatorias por parte del regulador.

Para compensar el circuito adecuadamente, es importante seleccionar un capacitor C_o con una ESR que no sea ni muy baja ni muy alta [46]. Para ello, el fabricante recomienda un valor de ESR entre $0,05 \Omega$ y $0,5 \Omega$, de un valor de por lo menos $4,7 \mu F$. Los capacitores MLCC típicamente tienen ESR muy bajas, del orden de unos pocos $m\Omega$. Debido a esto, se selecciona un capacitor de tantalio para el capacitor de salida C_o [47]. Estos suelen exhibir una ESR mayor a la de los MLCC.

Como fue ya mencionado, se incorpora un conector que permite adosar una fuente externa de 5 V. Este se seleccionó pensando en utilizar algunos módulos típicos para drones y vehículos aéreos [48]. Además de las líneas de tensión y de retorno, cuentan con dos señales analógicas que informan acerca de la tensión y la corriente de la batería. La computadora de vuelo sensa dicha información utilizando

entradas analógicas. Además, se incluyen dos filtros pasa bajos, uno por cada señal analógica. Estos pueden encontrarse en el Apéndice A: Circuito Esquemático.

5.3.7. Micro SD

Con el objetivo de registrar datos, tanto del estado como de la misión del vehículo, se incorpora la posibilidad de uso de una memoria Micro SD. Esta permite almacenar una gran cantidad de datos sin procesar, como por ejemplo datos crudos de los sensores del vehículo. Existe una gran variedad de memorias micro SD, las cuales se pueden clasificar según su capacidad:

- Standard Capacity SD Memory Card (SDSC): hasta 2 GB.
- High Capacity SD Memory Card (SDHC): más de 2 GB, hasta 32 GB.
- Extended Capacity SD Memory Card (SDXC): más de 32 GB, hasta 2 TB.
- Ultra Capacity SD Memory Card (SDUC): más de 2 TB, hasta 128 TB.

La memoria se compone de una interfaz eléctrica, un controlador interno, una serie de registros y la memoria en sí. Tanto la capa física como el protocolo de comunicación se encuentran definidos en el estándar de la SD Association, "SD Specifications Part 1 Physical Layer Specifications". Se definen 2 formas diferentes de comunicación con la memoria, SD Bus y SPI, siendo el primero el que se utiliza en este trabajo.

El protocolo que se define en este estándar tiene el formato master-slave, en este caso el master será el microcontrolador de la computadora de vuelo. En la figura 37 se muestra un esquema con las conexiones entre el master y la memoria.

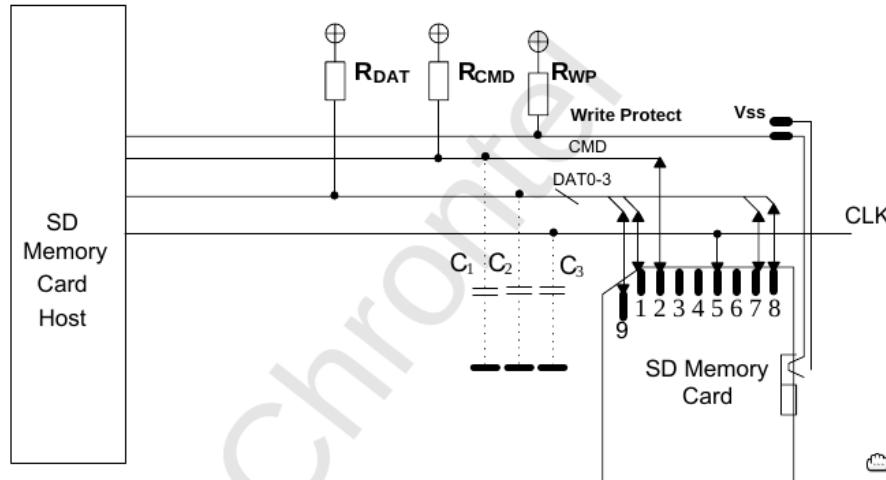


Figura 37: Esquema circuital de la conexión entre el SD host y una memoria SD, tal y como lo indica la especificación de la SD Association. **ESTA IMAGEN NO PUEDE QUEDAR, TENGO QUE DIBUJAR UNA SIMILAR YO.**

El estándar define que la alimentación V_{dd} debe estar entre 2,7 V y 3,6 V, por lo que puede alimentarse a través del mismo regulador de la placa. A través de la línea CLK, el host envía la señal de clock a la memoria SD para sincronizar la transferencia de datos. La línea CMD es una línea de envío y recepción de comandos, para la lectura de la memoria, escritura, etc. Finalmente, las líneas D0, D1, D2 y D3 se utilizan para envío y recepción de los datos desde y hacia la memoria.

El modo de comunicación SPI permite el uso con microcontroladores de propósito general, los cuales suelen incluir varios periféricos SPI. El uso de este protocolo simplifica la comunicación, aunque como

contrapartida, debido a que el bus SPI tiene una sola línea de datos, las velocidades de transferencia alcanzadas son inferiores. El bus SPI del microcontrolador que se utiliza en este trabajo, tiene una velocidad de transferencia máxima de 6,25 MB/s (50 MHz), mientras que la interfaz SD incluida alcanza velocidades de 25 MB/s (50 MHz). El motivo principal es que el protocolo SD tiene 4 líneas de datos en paralelo, mientras que con el uso de SPI solo se utiliza 1 línea.

El periférico del microcontrolador es compatible con la versión 2.0 de la SD Association, la cual solamente define las clasificaciones SDSC y SDHC. Además, se aclara que un host que puede comunicarse con una SDHC, también puede comunicarse con una SDSC, pero no al revés. En consecuencia, la computadora de vuelo podrá utilizar tarjetas de hasta 32 GB, correspondientes a las SDHC.

Se coloca un slot para memoria micro SD en la placa, modelo DM3AT-SF-PEJM5 del fabricante Hirose. Además de las líneas de comunicación ya mencionadas, este posee un terminal extra que permite detectar si se insertó una memoria SD. Este terminal funciona como una llave mecánica que se conecta a GND cuando se inserta la memoria en el slot. A través de la conexión con un GPIO del microcontrolador, se puede detectar la inserción de la memoria micro SD, para luego inicializar la comunicación. Al igual que el resto de componentes, el circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático.

5.3.8. Interfaz USB

Como parte de las funcionalidades básicas para utilización de la computadora de vuelo, se incorpora una interfaz para comunicación USB. Este puerto se incorpora con el objetivo de facilitar el uso de la placa durante las etapas de desarrollo de firmware, donde puede ser necesario hacer muchas pruebas y modificaciones. De esta forma, es posible programar la placa y proveer alimentación a través de un solo cable. La interfaz también puede aprovecharse como un puerto de comunicación más con la computadora de vuelo.

El microcontrolador utilizado integra un periférico compatible con la especificación USB 2.0 y con la extensión On-The-Go (OTG), el cual permite a pequeños dispositivos funcionar como hosts en el bus USB. Para las aplicaciones que se le quieren dar a la computadora de vuelo, se opta por un uso de modo device.

En cuanto a la velocidad de comunicación, el periférico brinda la posibilidad de utilizar tanto una configuración Full-Speed (12 Mb/s) como High-Speed (480 Mb/s). A priori, esta última sería de preferencia, dada la altísima tasa de transferencia, en comparación con la primera. Sin embargo, la utilización de esta última requiere el agregado de un componente adicional para utilizar como interfaz física con el bus USB, denominado ULPI. No sucede lo mismo para el uso de la configuración Full-Speed, para la cual la interfaz física se encuentra completamente integrada. Por una necesidad de mantener un tamaño de placa reducido y teniendo en cuenta que la interfaz USB no se usará para funciones críticas de la computadora de vuelo, se prefirió optar por el uso de una configuración Full-Speed. Cabe aclarar además, que la interfaz física integrada para Full-Speed trae consigo integrado el resistor de pull-up de $1,5\ k\Omega$ en la línea D+, necesario para el proceso de enumeración del dispositivo, por lo que tampoco fue necesario su incorporación en la placa.

Se incorpora en la placa un conector micro USB-B. El terminal 4 llamado ID se utiliza para indicar si el dispositivo funcionará como host o device. La computadora de vuelo funcionará siempre en modo device, por lo que este se deja en circuito abierto, tal como indica la especificación USB 2.0.

Como fue mencionado, el conector USB se utiliza como otra opción para alimentar la placa. La tensión nominal entregada por un host USB es de 5 V. Para alimentar el resto de la placa, se conecta esta entrada al regulador de 3,3 V que se encuentra en la placa. De forma de evitar que haya conflictos con el conector de alimentación principal mencionado en la sección 5.3.6, se incorporó un circuito simple formado por dos diodos, como se muestra en la figura 38.

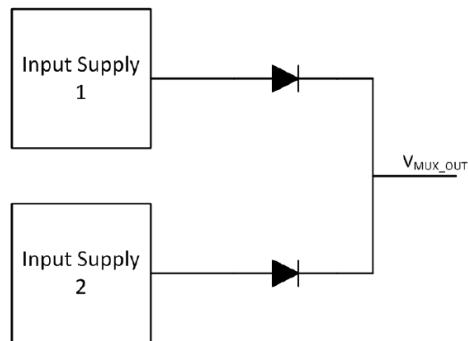


Figura 38: Se muestra un esquema como el incluido en la computadora de vuelo, el cual permite la alimentación de la misma a través de 2 fuentes distintas. **REEMPLAZAR ESTA IMAGEN POR UNA DIBUJADA POR MI.**

Estos diodos fuerzan a que las corrientes solamente fluyan desde las entradas de 5 V hacia el regulador. Debido a que estos se encuentran en serie con las corrientes de entrada a la placa, se seleccionaron diodos Schottky para tener bajas caídas de tensión, además de reducir la potencia disipada. El circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático. Si bien existen componentes especiales que permiten cumplir con esta misma funcionalidad con menor caída de tensión e incluso con la capacidad de seleccionar la fuente de alimentación a utilizar, nuevamente se opta por una solución que permita mantener unas dimensiones reducidas de la placa.

5.3.9. Conector para Módulo GPS

Se provee la posibilidad de incorporar de forma externa un módulo receptor de GPS. Estos entregan mediciones de posición y velocidad a la computadora de vuelo, los cuales se utilizan como mediciones complementarias al INS. Estos módulos además suelen proveer de una funcionalidad extra, llamada señal *Pulse-per-second*(PPS). Esta es una señal con un período de 1 segundo y que se encuentra sincronizada con los relojes atómicos del sistema GPS, los cuales tienen una exactitud de decenas de ns. Esta señal típicamente es utilizada como referencia de tiempo en el sistema de navegación de la computadora de vuelo.



Figura 39: Módulo receptor de GPS. Dentro de la carcasa de plástico se encuentra un chip que procesa las señales de GPS para extraer la información, además de una antena utilizada para recibir la señal proveniente de distintos satélites, típicamente una antena parche de cerámica.

Se proveen de 2 conectores diferentes para uso de este tipo de módulos. Por un lado, se utiliza un conector de la serie DF-13 del fabricante Hirose y por el otro, conectores de tira de pines estándar de 0.1" a 90 grados. En la figura 39 se muestra un módulo típico de uso en vehículos aéreos de uso comercial, el cual trabaja con una interfaz de comunicación serie. Los conectores incorporados están enfocados al uso de módulos de este tipo, por lo que en el conector pueden verse dos terminales con las etiquetas TX y RX. Estos además están acompañados por un terminal para la señal PPS.

5.3.10. Conectores para Salidas de PWM

Como parte del sistema de control, la computadora de vuelo debe comandar a los actuadores. En vehículos aéreos no tripulados comerciales, es común el uso de motores brushless de corriente continua. Estos se alimentan con una fuente de tensión continua, a través de un driver que genera una salida trifásica. Estos drivers se denominan *Electronic Speed Controller*(ESC).

Haciendo una búsqueda de los distintos tipos de ESC, puede verse que estos ofrecen interfaces de comunicación variadas, siendo la más común el uso de una señal PWM. Es por esto que en la computadora de vuelo se incorpora un conector el cual ofrece 8 salidas para este tipo de señal. Con motivo de tener compatibilidad con drivers comerciales, los conectores incorporados son conectores de tira de pines estándar de 0.1" a 90 grados. Las señales PWM serán generadas a través de los periféricos asociados a los timers del microcontrolador.

PONER UNA IMAGEN DONDE SE VEA QUE LA PLACA ENTREGA PWM PERO QUE LA CORRIENTE LA ENTREGA LA BATERÍA

Además de las 8 salidas para control de los actuadores del vehículo, se incorporaron otras 4 salidas extra en otro conector separado. Este segundo conector está pensado para utilizarse con otros fines, como controlar una serie de servomotores que puedan incorporarse al vehículo.

5.3.11. Conectores para Control por Radio

Como fue mencionado, el vehículo en algunas ocasiones puede ser controlado manualmente por un piloto remoto. Una de las formas de control para vehículos comerciales es a través de un control radio

transmisor RC. Estos cuentan con 2 *sticks* a través de los cuales puede guiarse al vehículo manualmente. El movimiento de los *sticks* es interpretado por un circuito que emite las señales moduladas a través de una antena. En el vehículo se monta un receptor, el cual recibe y demodula las señales de radiofrecuencia, para luego entregar los comandos del piloto a la computadora de vuelo.

La forma más común de comunicación entre el receptor y la computadora de vuelo es a través de distintas señales del tipo PWM denominadas canales. Cada canal puede indicar un set point distinto del vehículo, por ejemplo pueden tenerse 4 canales para los ángulos *pitch*, *roll*, *yaw* y otro para el empuje vertical del vehículo [49]. Debido a que esto requiere el uso de 4 líneas de comunicación entre el receptor y la computadora de vuelo, se opta por una alternativa similar. Estos utilizan una señal denominada *Pulse Position Modulation* (PPM) y tiene la ventaja de contener la información de todos los canales en una sola línea de comunicación. En la figura 40 pueden verse las características de la señal PPM.

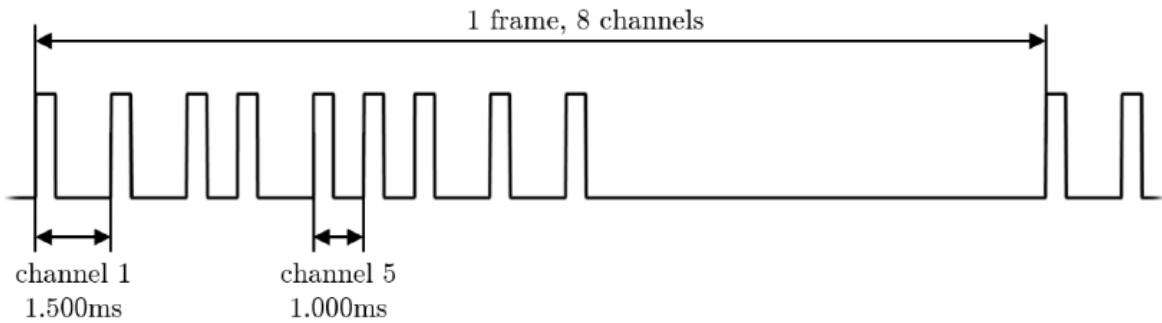


Figura 40: Ejemplo de una señal PPM para 8 canales **REEMPLAZAR ESTA IMAGEN POR UNA DIBUJADA POR MI.**

Esta se compone de *frames* de duración 20 ms. En cada *frame* puede haber una cierta cantidad de pulsos de ancho fijo. La información de cada canal se encuentra codificada en el espaciado entre pulsos, el cual puede ser un valor entre 1 y 2 ms. Un espaciado de 1 ms representa un valor de 0 %, mientras que 2 ms, un valor de 100 %. Estos porcentajes se asocian a los movimientos hechos por el piloto en los *sticks* del radio transmisor. Por ejemplo, en la figura 40 el canal 1 tiene una separación entre pulsos de 1,5 ms (50 %), mientras que el canal 5, de 1 ms (0 %).

En la computadora de vuelo se incluye un conector de tira de pines estándar de 0.1" a 90 grados de 3 terminales, 2 de ellos para alimentar el módulo desde la computadora de vuelo y el tercero para la comunicación. Para interpretar la señal se utiliza un periférico timer del microcontrolador y se aprovecha la funcionalidad *Input capture mode*. Cada vez que ocurre un flanco, el periférico captura el estado del timer, almacenándolo en un registro auxiliar. Estos valores pueden recuperarse con una interrupción o utilizando DMA. A partir de la diferencia entre valores consecutivos, puede conocerse el espaciado entre pulsos y por ende recuperar el comando enviado por el piloto.

Otro de los receptores comúnmente utilizados son los del fabricante Spektrum, denominados *Direct Signal Modulation* (DSM). En lugar de generar una señal modulada, estos utilizan una interfaz UART de comunicación serie. Existe una especificación donde se explica el protocolo, el contenido de los mensajes, velocidades, etc. [50]. De acuerdo a lo indicado en la especificación, en la computadora de vuelo se incluye un conector modelo B3B-ZR-SM4-TF(LF)(SN) del fabricante JST.

5.3.12. Conector para Programación y Debugging por SWD

Con el objetivo de facilitar el desarrollo de firmware para distintas aplicaciones, se incorpora un conector con acceso a los terminales de programación y debugging del microcontrolador. Estos corresponden a una interfaz *Serial Wire Debug* (SWD). A través de esta es posible detener el núcleo del microcontrolador utilizando *breakpoints* y examinar el estado de las variables del programa.

El protocolo necesita el agregado de unos resistores de pull-up y pull-down en las líneas SWDIO y SWCLK respectivamente. En la hoja de datos se indica que estos se encuentran integrados en el

microcontrolador, por lo que se hicieron las conexiones desde el microcontrolador directamente hacia un conector utilizado para comunicación con el dispositivo de debugging. No existe un conector estándar para SWD, aunque sí existe una recomendación por parte de ARM, Samtec FTSW 0,050”, tanto de 10 como de 20 terminales [51]. Por motivos de compatibilidad con el debugger disponible en el momento del desarrollo, se optó por un conector más simple, conformado por tiras de pines estándar de 0.1” verticales. En este mismo conector se incorporan líneas de alimentación y GND, permitiendo la alimentación de la placa, además de una línea de reset, la cual puede ser manejada por el debugger.

5.3.13. Otras Funcionalidades

Además del pulsador para la línea de RESET, se incluyen dos pulsadores extra. Estos a priori no tienen una funcionalidad definida, por lo que se incluyen para ser usados según lo requiera la aplicación de la computadora de vuelo.

Una de los posibles usos que se le puede dar a estos pulsadores es el de ejecutar una rutina de calibración de alguno de los sensores, como el magnetómetro externo, el cual se requiere una calibración que involucra un procedimiento manual.

Teniendo presente la necesidad de obtener un diseño de tamaño reducido, se incorporaron pulsadores de 2 terminales, como los que se muestran en la figura 41. Cada pulsador se encuentra acompañado de un resistor de pull-up, acompañado de un capacitor de filtrado y un resistor en serie con motivo de filtrar el efecto rebote.

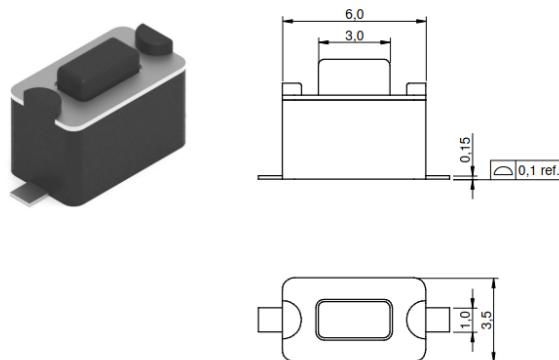


Figura 41: Pulsadores utilizados para la computadora de vuelo. Las imágenes se extrajeron de la hoja de datos del componente. Se muestran las dimensiones en mm.

Otra de las funcionalidades que se incorporan son una serie de LEDs de distintos colores. En total se incorporaron 8 LEDs, 2 de color azul, 2 de color rojo, 2 de color verde y 2 de color amarillo. Al igual que los pulsadores, estos son de propósito general y su uso dependerá de la aplicación involucrada.

Como ya fue mencionado, la computadora de vuelo debe contar con varias interfaces para comunicación con sensores y módulos externos. Para ello, se incorporaron 3 conectores para comunicación UART (además de las ya mencionadas para GPS y para control por radio), 1 para SPI y 2 buses I2C externos.

5.4. Desarrollo del PCB

En esta sección se explican las consideraciones tenidas en cuenta para el diseño de la placa. Se mencionan criterios generales, además de otros específicos para cada parte del circuito. Finalmente, se presenta el diseño final de la placa.

5.4.1. Requerimientos de Manufacturabilidad

Como ya fue mencionado en varias ocasiones, uno de los objetivos es lograr un diseño de placa de pequeñas dimensiones. Durante el desarrollo se tuvo presente una limitación para este requerimiento y es el hecho de que solamente se cuenta con la posibilidad de montar todos los componentes sobre una sola cara de la placa. Esto se tuvo en cuenta durante el proceso de routeo, ubicando todos los *footprints* en la cara top.

5.4.2. Requerimientos de Posicionamiento del Sensor IMU

Este sensor tiene ciertos requerimientos particulares que deben cumplirse, siendo el más importante la necesidad de ser ubicado en el centro de la placa. Como fue mencionado en la sección 5.3.2, todas las mediciones de aceleración y velocidad angular son relativas a una terna solidaria a este. Al centrar el sensor en la placa, luego todas las mediciones también estarán referidas a una terna solidaria a la placa.

Durante el desarrollo del layout del PCB, se encontró un impedimento por el que no se logró centrar la IMU en la placa. Esto se debió a la necesidad de que todos los componentes se encuentren ubicados de un solo lado de la placa, tal como se mencionó en la sección 5.4.1, sumado al gran tamaño del microcontrolador. El ubicar el sensor en el centro de la placa, fuerza a que el microcontrolador quede en uno de los extremos de la placa, lo que dificulta mucho las conexiones de este con el resto de los componentes. Esto se muestra en la figura .

PONER UNA IMAGEN DONDE SE VEA QUE LA IMU FUERZA A QUE EL MICRO QUEDA DE UN LADO O DEL OTRO DEL CENTRO DE LA PLACA.

A pesar de esto, luego de hacer varias iteraciones en el routeo final, el sensor pudo montarse apenas 8 mm desfasado respecto del centro. Este dato debe ser tenido en cuenta a la hora de montar la placa en un vehículo.

Como fue mencionado en la sección 5.3.2, este sensor es un sistema electromecánico, por lo que el estrés que este sufra puede alterar sus mediciones, o incluso llegar a dañar el componente. Para minimizar estos efectos, se siguieron una serie de guías y recomendaciones de distintas notas de aplicación para montaje de estos sensores sobre PCBs [52] [53] [54]. Se enumeran algunas de esas recomendaciones:

- Montar el sensor lejos de orificios de montaje para el PCB, lejos de orificios para colocar tornillos y lejos de componentes como pulsadores y conectores. Para el caso de un pulsador, por ejemplo, al presionarlo esto genera una presión sobre el PCB. Si la IMU se encuentra muy cerca del pulsador, dicha presión puede llegar a afectar la zona donde se encuentra la IMU, alterando las mediciones. El uso de pulsadores suele utilizarse para activar rutinas de calibración, por ejemplo de la IMU, del magnetómetro o de otro tipo de sensores.
- Montar la IMU lejos de los bordes del PCB.
- No ubicar test points ni conectores debajo de la IMU, es decir, en la otra cara del PCB. El conectar y desconectar continuamente puede dañar el componente.
- El layout del circuito debe ser lo más simétrico posible. No es necesario utilizar pistas de un tamaño diferente para las líneas de alimentación, ya que su consumo es muy bajo.
- No pasar pistas debajo de la IMU.
- No ubicar vías debajo del componente. El área debajo de este debe definirse como keepout area.

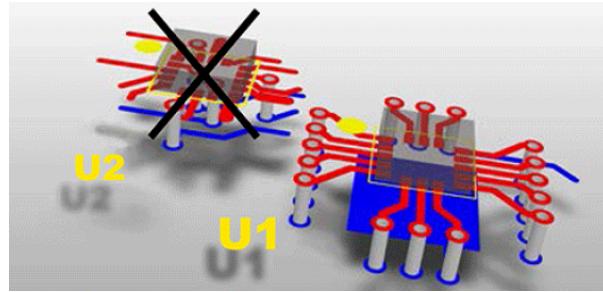


Figura 42: Se muestran dos ejemplos de layout de una IMU. El layout U2 no sigue las recomendaciones mencionadas, mientras que U1 sí. La imagen se extrajo de [54].

La imagen de la figura 42 resume algunas de estas recomendaciones. Lo que se observa es que las pistas del sensor son simétricas. A pesar de que algunos terminales de la IMU no se utilicen, se recomienda que igualmente sean incluidas en el routeo, solo para mantener la simetría. Durante el proceso de soldadura, el estaño presente en los distintos pads del componente generan una tensión que atrae el componente hacia este. En la figura 43 se muestra un ejemplo de una IMU rotada respecto de su placa. Si el routeo no es simétrico, es posible que el sensor no quede centrado, lo que resultaría en un problema durante el uso de la computadora de vuelo.

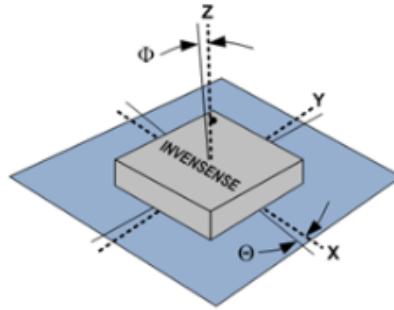


Figura 43: Se muestra un ejemplo donde la IMU se encuentra rotada respecto de la placa. En línea punteada los ejes de la terna solidaria al sensor y en línea continua la terna solidaria a la placa. La imagen se extrajo de [55].

5.4.3. Características del PCB

Dada la gran cantidad de pistas que deben trazarse en toda la placa, se opta por un diseño de PCB de 4 capas. Esta elección simplifica el routeo, ya que se cuenta con una mayor cantidad de capas para pasar pistas, en relación a una configuración típica de 2 capas. En la figura 44 se muestra el *stackup* utilizado para el PCB, donde se puede ver que se cuenta con 2 capas de señal, 1 capa de alimentación y 1 capa de retorno GND.

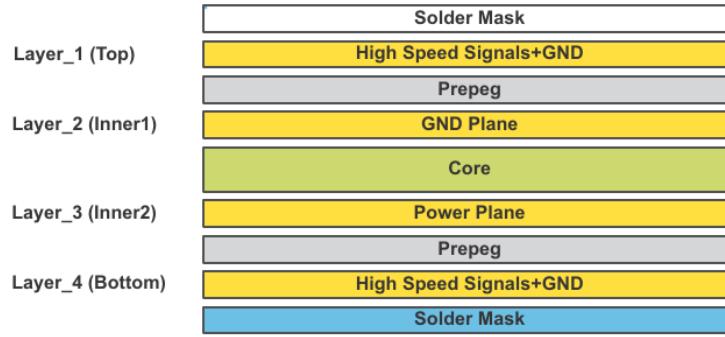


Figura 44: Configuración de capas utilizada para el PCB. REEMPLAZAR ESTA IMAGEN POR UNA DIBUJADA POR MI. APROVECHAR Y AGREGAR LAS DIMENSIONES DEL ESPACIADO ENTRE CAPAS, EL ESPESOR DE COBRE Y LOS DIELECTRICOS.

La configuración de 4 capas tiene otras ventajas respecto del uso de 2. El circuito de la computadora de vuelo es prácticamente un circuito con señales digitales, por lo que debe darse un cuidado especial, minimizando la impedancia del retorno de corriente. En el caso de un PCB de 4 capas, al tener una capa entera dedicada al retorno, esta ofrecerá una muy baja impedancia para las corrientes de alta frecuencia. En la figura 45 se muestra cómo el efecto de la inductancia presente en el retorno de señal, puede generar ruido a través de la conexión a GND en otros componentes. El ejemplo de la figura es con compuertas lógicas, pero aplica igualmente a cualquier circuito digital, por ejemplo a la comunicación entre el microcontrolador y alguno de los sensores.

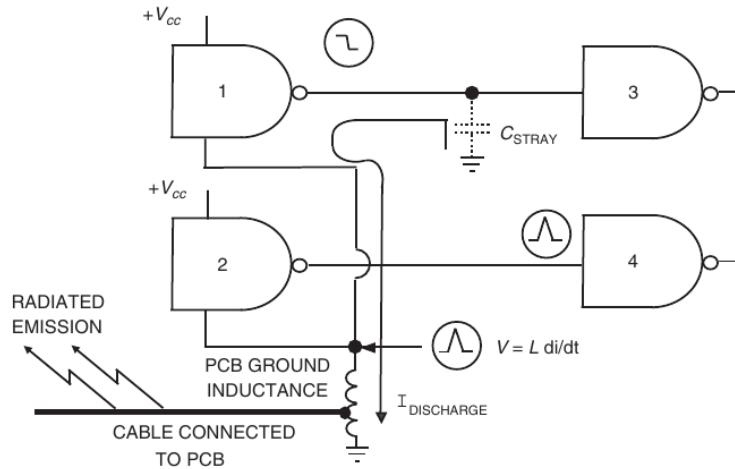


Figura 45: La compuerta lógica 2 mantiene su salida siempre en 0 V, mientras que la compuerta lógica 1 cambia su salida de estado alto a 0 V. En este momento, se produce una descarga del capacitor C_{STRAY} el cual representa capacidades que pueden ser por ejemplo de la pista que une su salida con la compuerta 3. En ese momento, la corriente circula por la inductancia del retorno, generando una caída de tensión que afecta a la compuerta 2 y a su salida. La imagen se extrajo de [56, p. 381].

En el caso de un PCB de 2 capas, es difícil reducir este efecto, ya que no se cuenta con el espacio suficiente. Otra de las ventajas es la disminución de las emisiones radiadas generadas por las caídas de potencial en la inductancia del retorno de señal [56, p. 477].

La selección del espaciado entre placas y el espesor de cobre se vieron afectadas por las posibilidades ofrecidas por el fabricante *PCBWay*. Se optó por la opción más económica, correspondiente a las dimensiones de la figura 44.

Otra de las técnicas utilizadas para reducir el ruido producto de las corrientes de alta frecuencia en la placa es el agregado de capacitores de desacople, principalmente en el microcontrolador. Estos ofrecen un camino de baja impedancia descargándose y entregando corrientes de alta frecuencia.

Algunos de los conectores de la computadora de vuelo tienen salidas con tensiones de 5 V utilizados para alimentar módulos y sensores, como es el caso del receptor GPS. Esta tensión de 5 V se obtiene directamente desde la tensión de entrada del conector de alimentación de la computadora de vuelo. Para ello, se trazó una pista en la misma cara top, con un ancho mayor al resto de las demás. Esta se muestra en la figura 46. Podría haberse incluido un plano con esta tensión en la misma capa que el plano de 3,3 V. Sin embargo, esto hubiera generado que el plano de 3,3 V se viera afectado, haciendo que este se viera partido, lo que incrementaría la impedancia vista por las corrientes sobre este.

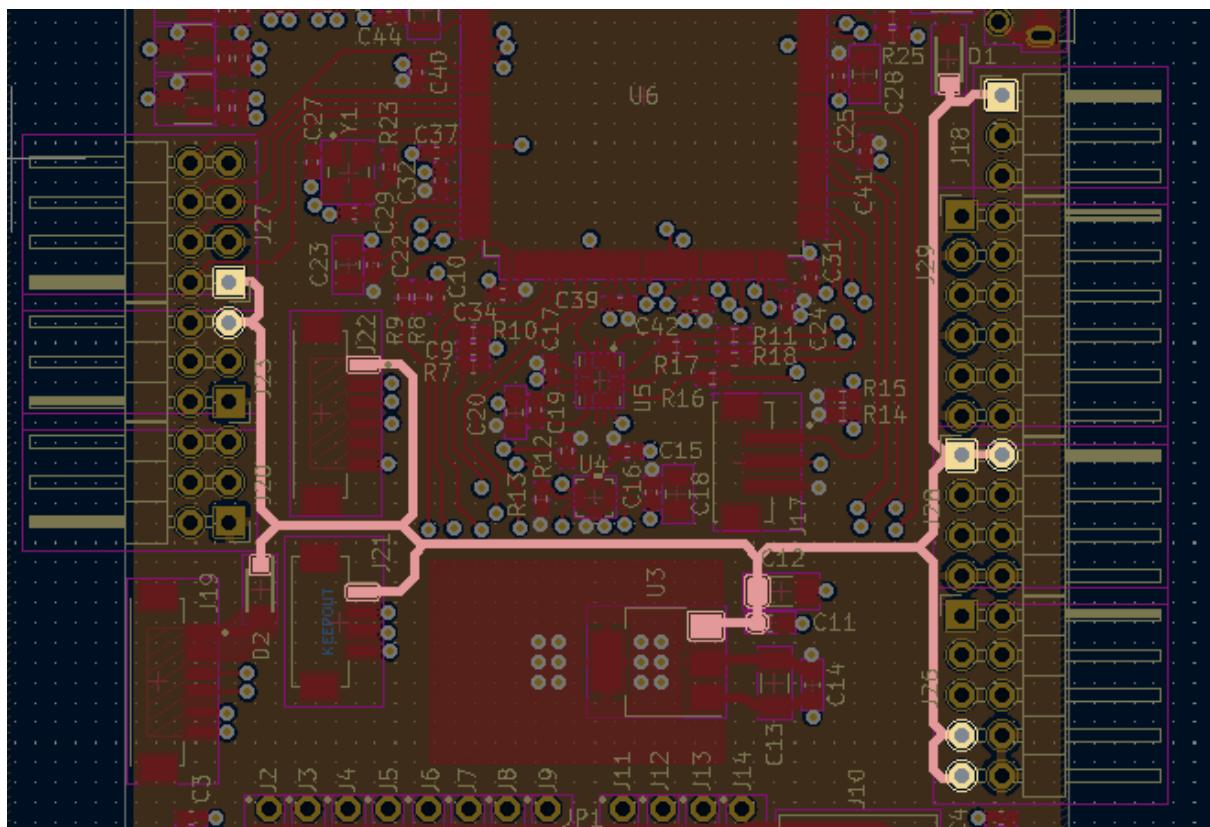


Figura 46: Se muestra una captura del software KiCad, utilizado para el desarrollo de la placa de la computadora de vuelo. Se resalta la pista de 5 V en la capa top. A su vez se puede ver la zona naranja la cual corresponde a la capa 3 y que contiene en su totalidad al plano de 3,3 V. El plano de GND se ubica en la capa 2.

Otro de los aspectos a definir para la fabricación del PCB fueron las vías. Al tener varias capas, el routeo de las pistas trae consigo una gran cantidad de vías en la placa, debido a los cambios de capa y a las conexiones con los planos internos de alimentación y GND. Existen distintos tipos de vías que ofrece el fabricante, entre ellas las vías ciegas, vías enterradas y vías pasantes. Se prefiere evitar el uso de otro tipo de vía que no sea una vía pasante, debido a que de otra forma se incrementa mucho el costo de fabricación de la placa, ya que el proceso de manufactura de las vías ciegas y las vías enterradas, es más complejo que el de las vías pasantes.

5.4.4. Comunicación con el Slot para Tarjeta Micro SD

Para trazar las pistas correspondientes a la comunicación con la memoria Micro SD, se tuvieron en cuenta los requerimientos planteados en la especificación de la SD Association. Esta indica valores máximos de capacidad y de inductancia de cada trazo, para mantener la integridad de la señal. Además de esto, se tuvieron en cuenta una serie de recomendaciones indicadas por el fabricante del microcontrolador, para uso del periférico [57].

Se buscó obtener una impedancia característica de 50Ω tanto para las líneas de datos como para la de comandos y clock. Esto puede lograrse gracias a la selección del *stack-up* del PCB, con planos de GND y de $3,3 \text{ V}$ continuos. De esta manera, las pistas en las capas exteriores son del tipo microstrip, cuya impedancia característica es conocida. En la figura 47 se muestra una imagen con las dimensiones finales de las pistas. La permitividad del dieléctrico se tomó a partir de los datos provistos por el fabricante *PCBWay*.

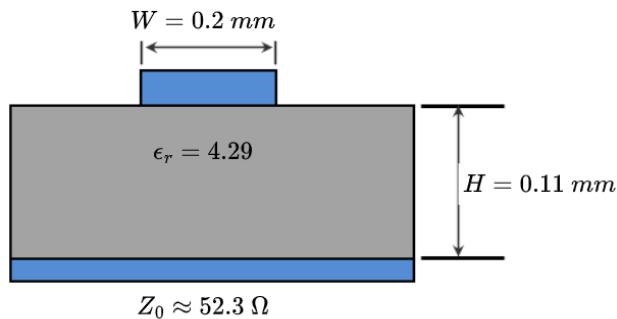


Figura 47: Se muestran las dimensiones de las pistas utilizadas para la memoria MicroSD. El alto del dieléctrico y su permitividad, están definidos por la configuración del PCB.

Con el motivo de reducir la cantidad de pistas con disintintas dimensiones en el PCB, se utilizan estas mismas dimensiones en la mayor parte de las pistas, por ejemplo para la comunicación con los sensores.

El protocolo de comunicación corresponde a un bus de datos paralelo, donde se utiliza el flanco ascendente del clock como referencia para muestrear el bus. Para que la comunicación funcione adecuadamente, el fabricante del microcontrolador recomienda que el desfasaje entre pistas sea menor a 60 ps . Para lograr esto, se buscó que todas las pistas de datos y de clock tuvieran la misma longitud. Además, se utilizó la misma cantidad de vías en cada pista.

5.4.5. Comunicación USB

Otro de los casos donde se tuvieron cuidados especiales en el trazado de las pistas, es para el caso de la comunicación USB. Como se trata de una señal ||diferencial, las pistas que conectan el periférico con el puerto USB de la placa fueron trazadas utilizando una herramienta del software *KiCad* especial para el trazado de este tipo de pistas.

En la especificación Universal Serial Bus Specification Revision 2.0 se indican las impedancias características de modo común y diferencial, de 90Ω y 30Ω respectivamente. No obstante, se permiten unos rangos de tolerancia bastante amplios, del 15 % y del 30 % respectivamente. A partir de las características del dielectrico del PCB, se definieron las dimensiones del par diferencial utilizado para la comunicación USB:

- Ancho de pista = $0,254 \text{ mm}$
- Separación entre pistas = $0,254 \text{ mm}$.
- $Z_{0,diff} = 84,5 \Omega$

- $Z_{0,comm} = 23,6 \Omega$

5.4.6. Layout del Regulador Lineal

Previo al trazado de pistas de la placa, se hizo un análisis térmico del regulador lineal. A partir de la hoja de datos [44], se detectó que para una temperatura ambiente de 25 °C y para corrientes de 500mA, puede suceder que el regulador supere su potencia máxima disipable. Si bien este valor de corriente se encuentra sobredimensionado para la placa de este trabajo, debe recordarse que la corriente dependerá en gran medida de los módulos y sensores externos conectados a la computadora de vuelo.

Durante esta etapa del desarrollo del PCB, se encontraron una serie de notas de aplicación las cuales explican técnicas y muestran resultados de la reducción de la resistencia térmica θ_{JA} . Se prefirió dedicarle un esfuerzo al diseño teniendo en cuenta la disipación de calor del regulador, con el objetivo de evitar posibles problemas luego de la fabricación del PCB.

En [58] se muestran resultados del efecto de aplicar distintas modificaciones al layout en el PCB de un integrado. Esto se pudo lograr por ejemplo incrementando la zona de cobre que hace contacto directo con el integrado, a través del uso de otras capas del PCB para decrementar todavía más la resistencia térmica y el uso de vías térmicas.

En primera medida, se incluyó un plano de cobre en la capa top, de $211 mm^2$. Este se puede ver en la figura 48. En la nota de aplicación [59] se muestran resultados de distintas pruebas utilizando un encapsulado como el de este trabajo, donde se fue incrementando la superficie del plano de cobre en la misma capa del integrado. Esta concluye que para una superficie de $211 mm^2$ se obtiene una $\theta_{JA} = 60 ^\circ\text{C}/\text{W}$. Este valor representa una reducción del 43,9 % respecto del valor reportado en la hoja de datos y en el gráfico de la figura 49 se puede ver la comparación entre ambos casos de la potencia máxima disipable.

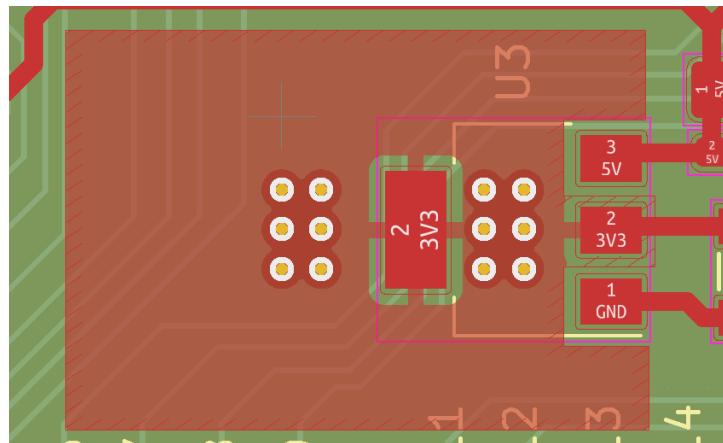


Figura 48: Se muestra el layout final del regulador en la cara top. Se agrega un plano de $211 mm^2$ en la misma capa para reducir la resistencia térmica. Además, se utilizan una serie de vías para conectar con el plano interno de 3,3 V, permitiendo disipar calor a través de este.

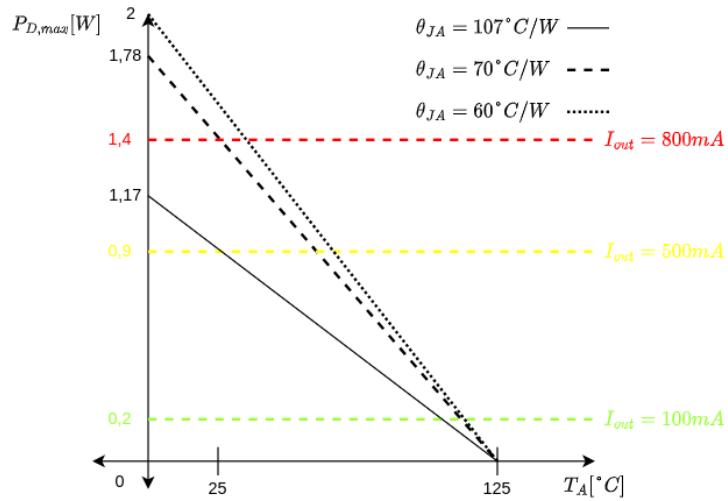


Figura 49: Se muestra el efecto de la reducción de la θ_{JA} en la potencia máxima disipable por el regulador. Esto además permite que este pueda entregar mayores valores de corriente **HAY QUE HACER ESTE GRÁFICO USANDO OCTAVE/PYTHON/MATLAB.**

Existen algunas diferencias respecto de las condiciones planteadas en [59], ya que allí se utiliza un espesor de cobre de $70 \mu\text{m}$, mientras que en este trabajo se utilizó un espesor de $35 \mu\text{m}$. A pesar de esto, los resultados que se muestran en [58] indican que el cambiar el espesor de $70\mu\text{m}$ a $35\mu\text{m}$ en un PCB 4 capas presenta un incremento menor al 10 %.

Por otro lado, en la placa de este trabajo se incorporan una serie de vías en el plano, las cuales conectan con la capa de $3,3 \text{ V}$. En otra nota de aplicación de *Texas Instruments* [60] se muestra otro estudio junto con resultados donde se aprovechan los planos internos en un PCB de 4 capas. Allí se demuestra que solamente con tener capas de cobre interiores sin la necesidad de estar conectadas al regulador, pueden reducir la θ_{JA} hasta un 50 %. Si estos se encuentran conectadas al regulador, la reducción es aún mayor. Para la ubicación de las vías, estas se colocaron inmediatamente debajo del regulador.

5.5. Diagrama en Bloques y PCB Final

Teniendo en cuenta todas las consideraciones mencionadas a lo largo de esta sección, se pudo plantear un circuito esquemático y obtener un diseño de PCB. Este último puede encontrarse en el Apéndice B: PCB Final. A modo de resumen, en la figura 50 se muestra un diagrama en bloques de la computadora de vuelo y en la figura 51 la implementación en PCB. Por último, en la figura 52, se muestran las 3 placas que se enviaron a fabricar para este trabajo.

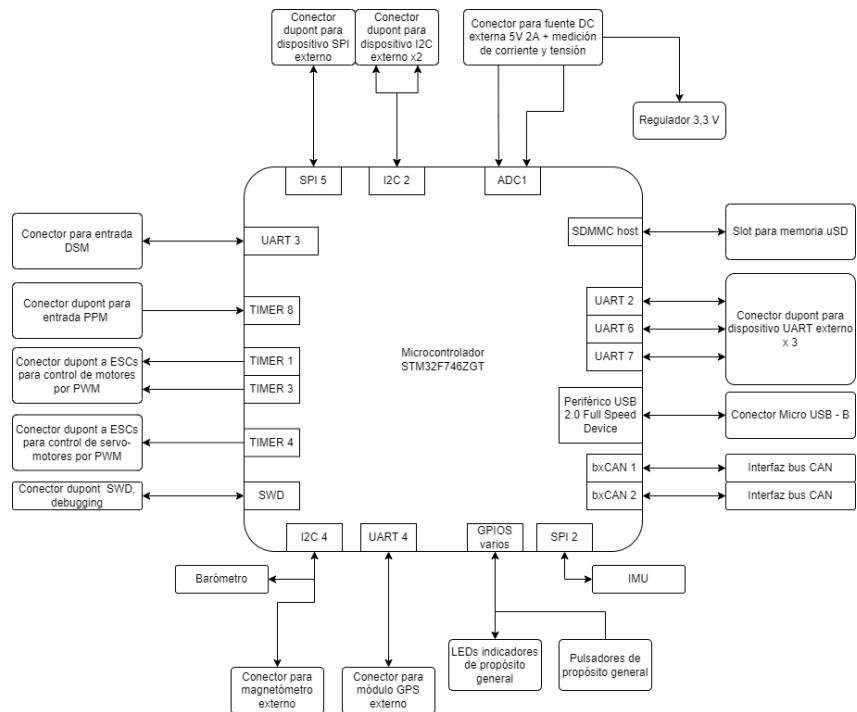


Figura 50: Diagrama en bloques de la computadora de vuelo.

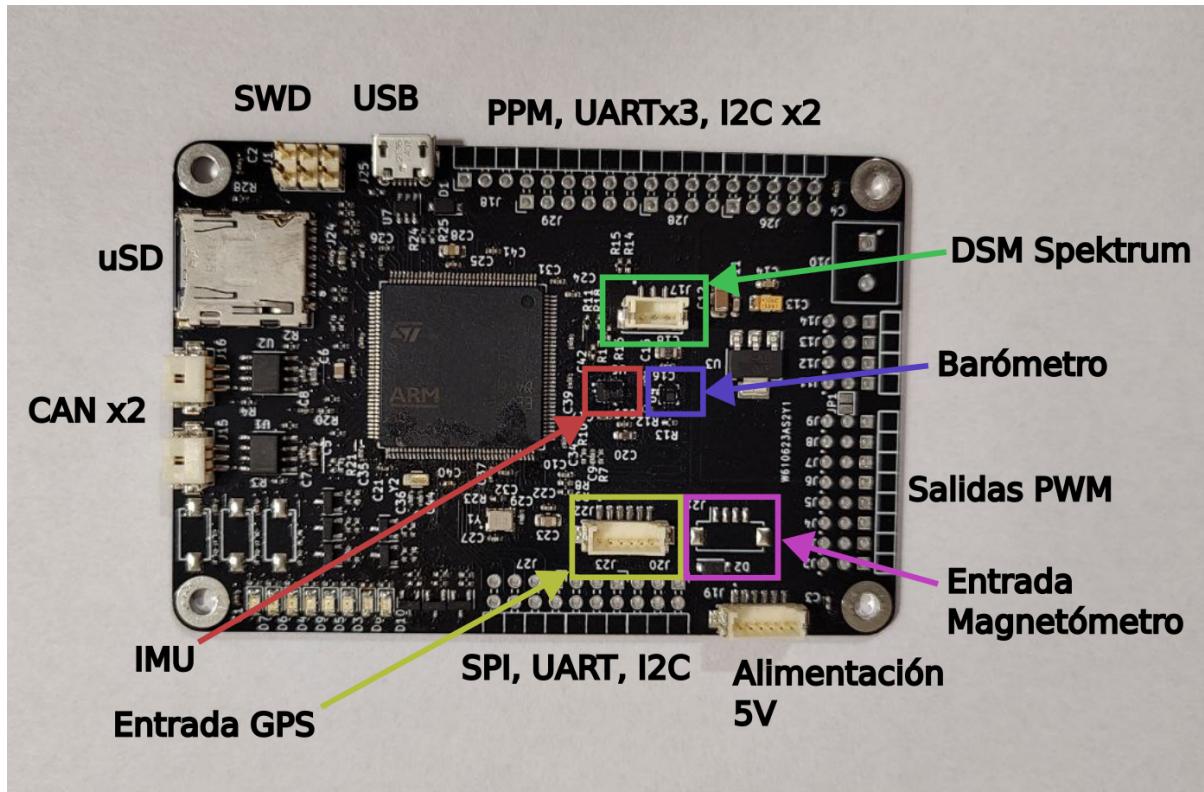


Figura 51: Se muestra el detalle del PCB de la computadora de vuelo.

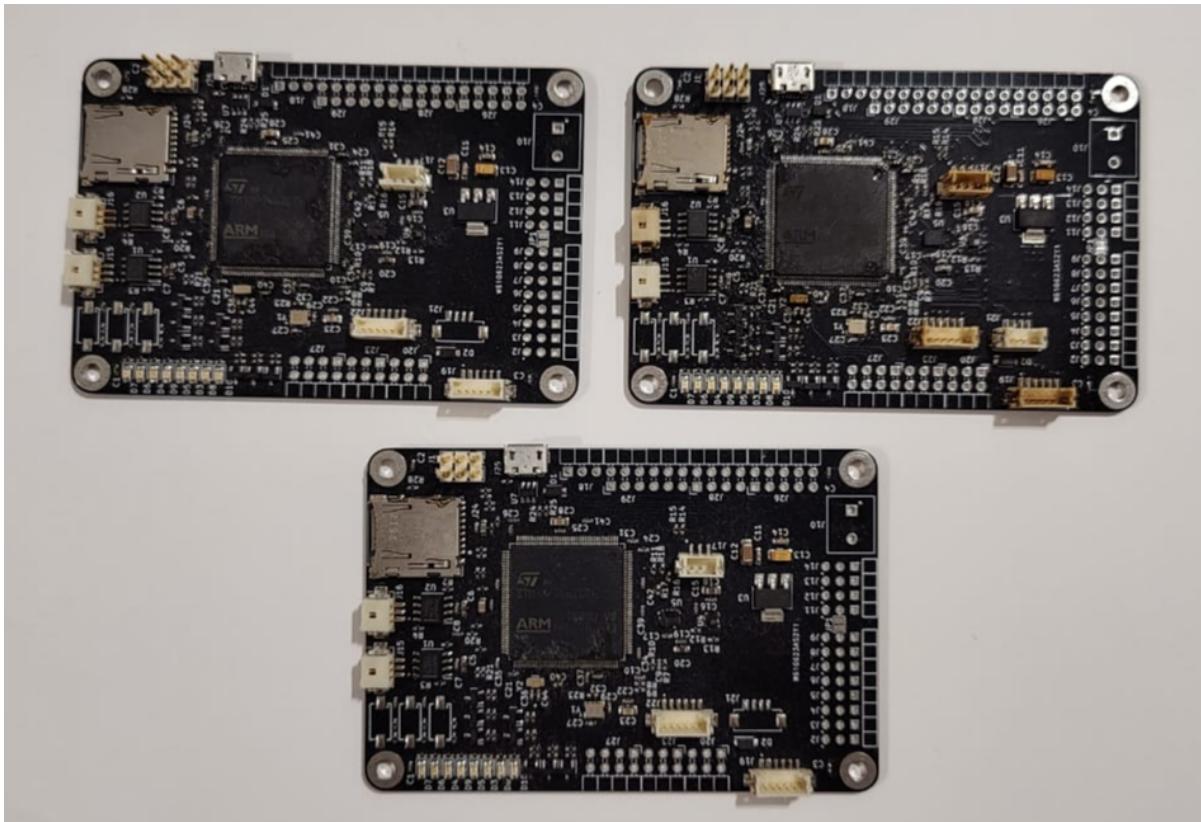


Figura 52: Se muestran las 3 placas enviadas a fabricar para este trabajo.

6. Implementación del Sistema Tolerante a Fallas

En esta sección se presenta la arquitectura de sistema propuesta para la tarea de tolerancia a fallas. Esta consiste en un sistema distribuido, conformado por las tres computadoras de vuelo construidas para este trabajo. Se describen las características del firmware desarrollado para implementar el sistema propuesto. Para demostrar las capacidades de tolerancia a fallas, se hicieron una serie de pruebas simulando fallas en un sensor. Finalmente, se muestran los resultados obtenidos sobre las pruebas realizadas.

6.1. Descripción de la Arquitectura Propuesta

Como fue explicado en la sección 5, la computadora de vuelo se encarga de ejecutar el sistema de navegación para guiar el vehículo en su trayectoria. Este sistema cumple con las características de ser de tiempo real, ya que tanto los valores de los resultados de cómputo, como el instante de tiempo en el que se obtienen, son de vital importancia para la correcta ejecución de las distintas tareas de control del vehículo. Por otro lado, en la sección 4 se hizo un análisis de las características que deben tener este tipo de sistemas para poder implementar un mecanismo de tolerancia a fallas utilizando redundancias. A partir de lo presentado hasta aquí, las características elementales de un sistema de tiempo real con tolerancia a fallas se pueden resumir en los siguientes items:

1. Se requiere determinismo temporal en la ejecución de las tareas.
2. La tolerancia a fallas se implementa a través de la comparación de resultados de las distintas réplicas.
3. Para que el mecanismo de tolerancia a fallas sea efectivo, es necesario comparar resultados que se correspondan temporalmente. Esto implica que cada réplica debe ejecutar las mismas tareas en paralelo y de manera sincronizada.
4. Es necesario utilizar un bus para el intercambio de resultados entre réplicas. A su vez, se requiere un acceso al medio por turnos, respetado por todas las réplicas.

Una arquitectura que se ajusta a estas características corresponde a sistemas que ejecutan sus tareas en instantes de tiempo predefinidos, típicamente denominados *time-triggered systems* o *time-triggered architecture*. La característica principal es que presentan un fuerte determinismo temporal, ya que el instante de tiempo en el que se ejecuta cada tarea se vuelve parte del diseño del sistema. Esta característica facilita su uso en sistemas críticos de tiempo real, debido a que su comportamiento es predecible [61, p. 12]. Pueden encontrarse trabajos [62] [63] y libros [9] [61] que explican formalmente esta arquitectura y sus distintos componentes con más detalle. En esta sección se pretende describir brevemente sus características más relevantes.

En la figura 53 se muestra un esquema que se corresponde con del funcionamiento de un sistema *time-triggered*, donde se ilustra una simplificación del funcionamiento de la computadora de vuelo. En el instante de tiempo t_1 se obtienen mediciones de acelerómetros y giróscopos, en el instante t_2 se ejecuta el algoritmo de navegación y en t_3 se aplica un nuevo comando sobre los actuadores del vehículo.

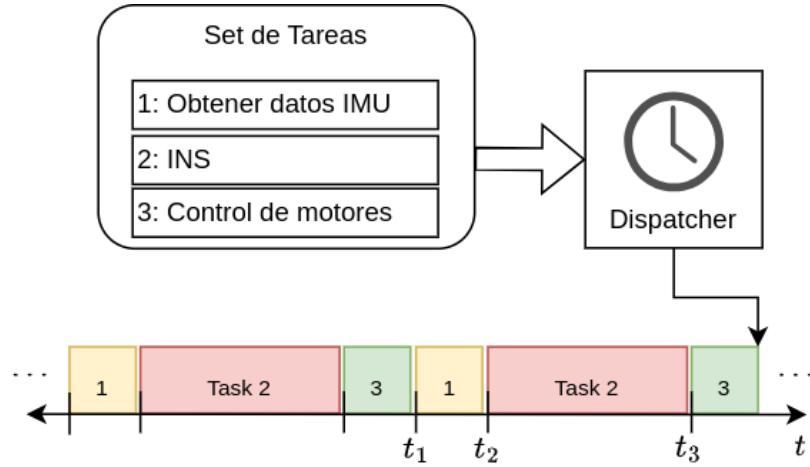


Figura 53: Esquema del sistema time-triggered. Dentro de las especificaciones de cada tarea se encuentran sus características temporales. Estas son utilizadas por un dispatcher, para saber en qué momento deben ser ejecutadas. Para el ejemplo de esta imagen, la secuencia de ejecución es periódica. Primero se obtienen datos del sensor IMU. Estos datos se utilizan en la tarea 2 para ejecutar algoritmos de estimación y control del vehículo. Finalmente, la nueva señal de control calculada en la tarea 2, se aplica a los motores en la tarea 3. El ciclo se repite periódicamente.

La gran mayoría de sistemas embebidos emplean mecanismos que trabajan a partir de ejecutar una respuesta frente a eventos, como por ejemplo a través del uso de interrupciones. Los periféricos de un microcontrolador cuentan con mecanismos para informar al procesador de la ocurrencia de determinado evento, como por ejemplo la recepción de un nuevo byte en una interfaz UART, o el cambio de nivel en una entrada de propósito general GPIO. El hecho de que estos eventos puedan ocurrir en cualquier momento y en cualquier orden, puede comprometer el determinismo temporal.

Un caso común es el de los sensores MEMS, los cuales suelen integrar un mecanismo para avisar al microcontrolador que hay nuevos datos de acelerómetros y giróscopos disponibles, a través de uno de sus terminales. Este se puede conectar a un GPIO del microcontrolador y configurar una interrupción por flanco ascendente. El set de tareas de la figura 53 puede estructurarse para incluir esta interrupción, como se muestra en la figura 54. En este ejemplo puede verse que la tarea de control de motores se retrasa debido a esta interrupción. En el caso time-triggered, el orden e instantes de tiempo de ejecución se vuelven parte del diseño del sistema, evitando que las tareas se retrase y puedan cumplir con su timing.

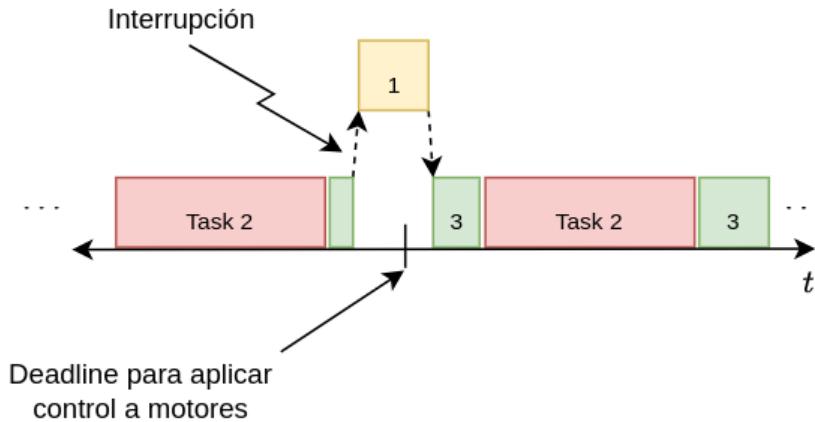


Figura 54: Mismo set de tareas que en la figura 53, pero con una arquitectura event-triggered. La interrupción genera que se ejecute la tarea 1. Debido a que las interrupciones pueden ocurrir en cualquier momento, se puede ver afectado el timing del sistema. Teniendo en cuenta que se trata de un sistema de tiempo real, esto no es aceptable y debe tratarse de una forma controlada, por ejemplo a través de una arquitectura time-triggered.

Muchos sistemas controlados por eventos funcionan con sistemas operativos de tiempo real, los cuales disponen de un scheduler que elige cuál es la tarea que corresponde ejecutar. Este aspecto también es una característica de los sistemas *time-triggered*, aunque la diferencia se encuentra en la estrategia utilizada. En el caso de un RTOS es común el uso de *schedulers* expropiativos con un sistema de prioridades. En un sistema *time-triggered*, el *scheduler* utiliza un reloj, típicamente implementado con un periférico timer, el cual determina cuál es la tarea a ejecutar. Además, a diferencia de algunos RTOS, no se utiliza el cambio de contexto, sino que cada tarea que se ejecuta, retorna y devuelve el control al *scheduler* para ejecutar la siguiente tarea. Estos aspectos refuerzan el determinismo del sistema, lo que se traduce en un comportamiento predecible y por ende seguro [61, p. 247].

En cuanto a la comparación de resultados entre distintas réplicas, estas realizarán la comparación de resultados a través de un bus de comunicaciones. De manera de minimizar las colisiones y favorecer el cumplimiento en el timing del sistema de tiempo real, el envío y recepción de los mensajes se implementa por turnos. La arquitectura que se utiliza facilita la implementación de este mecanismo, por ejemplo a partir del uso de tareas que estén dedicadas a recibir o enviar un mensaje a través del bus. En la figura 55 se muestra un caso para 3 réplicas. Este gráfico tiene una similitud con el gráfico de la figura 53. Mientras la réplica 1 ejecuta su tarea para enviar un mensaje, las réplicas 2 y 3 ejecutan una tarea para recibir ese mensaje.

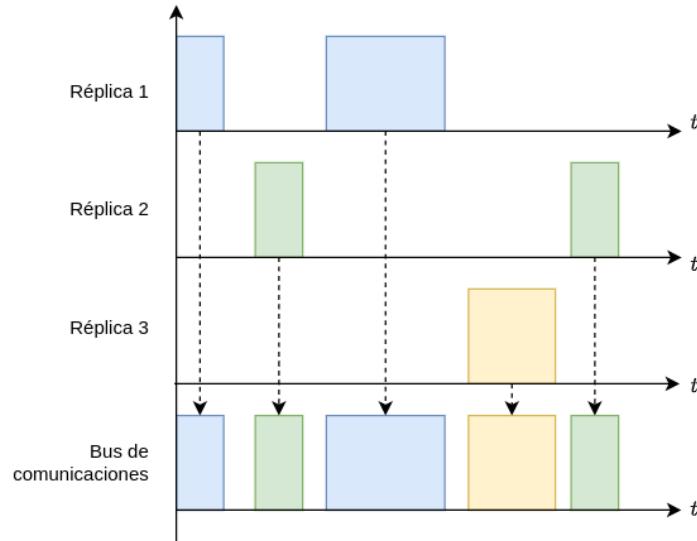


Figura 55: El ejemplo muestra como las 3 réplicas pueden compartir el bus de comunicaciones. Cada una de ellas sabe en qué instante de tiempo enviar un mensaje y en qué instantes de tiempo recibirán un mensaje. Para que esto funcione adecuadamente, las réplicas deben estar sincronizados.

El hecho de que el comportamiento del sistema sea predecible facilita la tarea de detección del fracaso del cumplimiento del scheduling de las tareas. Esto puede hacerse en tiempo de ejecución. En la figura 56 se muestra un ejemplo donde una tarea excede su tiempo de ejecución normal. Si esto no es monitoreado adecuadamente, podría bloquearse la ejecución de la siguiente tarea. Debido a que se conoce qué tarea se está ejecutando y cuál debería ejecutarse, fácilmente puede detectarse el fracaso en la ejecución y tomar una acción en consecuencia.

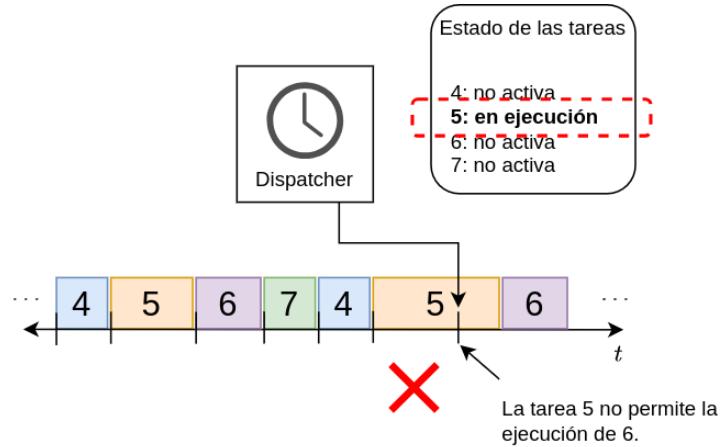


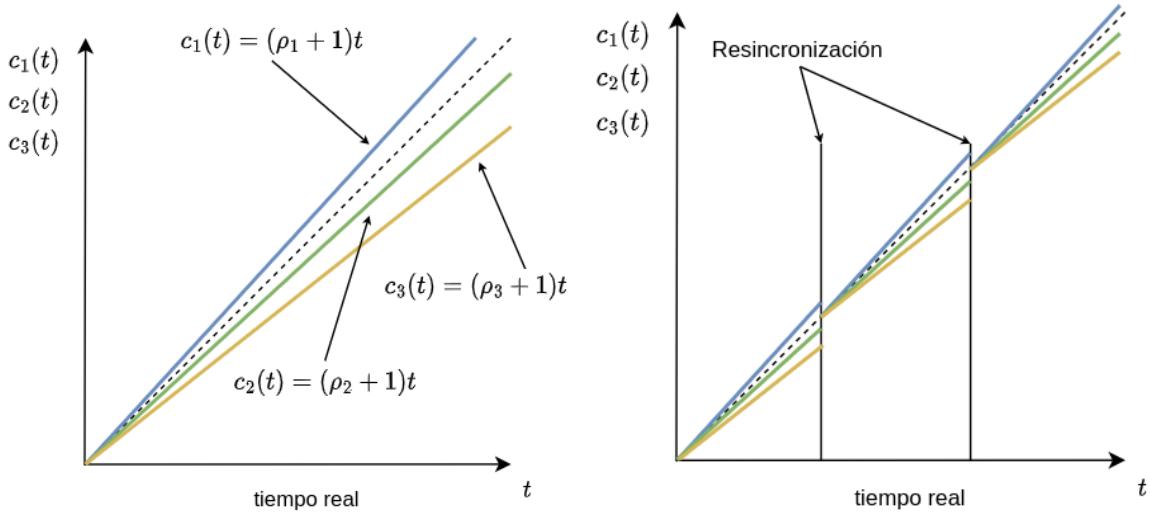
Figura 56: La tarea 5 bloquea la ejecución de la 6, evitando que se cumpla el scheduling. Mientras se ejecuta la tarea 5, se llama al dispatcher para ejecutar la siguiente, tarea 6. Gracias a que el dispatcher conoce el estado de las tareas, este detecta que la tarea 5 no ha finalizado su ejecución y por ende que no se ha respetado el scheduling.

Para que no haya colisiones en el bus de comunicaciones, todas las réplicas deben respetar el timing, el cual se encuentra predefinido gracias al *scheduler*. Sumado a esto, para que las comparaciones de resultados permitan la detección de fallas, los valores a comparar deben corresponderse temporalmente.

Ambas cuestiones se resuelven a partir de la sincronización de los *schedulers* de cada una de las réplicas, la cuál debe ejecutarse de forma periódica.

El comportamiento de cada una de las réplicas está controlado en última instancia por el *timer* del *scheduler*. Por más que todas las réplicas presenten circuitos osciladores exactamente iguales y de la misma frecuencia, inevitablemente existirá un efecto de *drift*, ya sea por cuestiones mismas de fabricación o por efecto de la temperatura. Esto traerá como consecuencia que los *timers* y por ende los *schedulers* se desfasean a medida que pasa el tiempo.

Para solucionar este problema, se requiere la ejecución de una resincronización periódica. El desfasaje entre los clocks de cada réplica irá creciendo conforme pase el tiempo. En la figura 57a se muestra un ejemplo donde cada réplica tiene un drift ρ distinto. Una forma de corrección puede ser como la que se muestra en la figura 57b. Al corregir el offset entre clocks periódicamente puede controlarse el desfasaje, manteniéndolo en niveles aceptables. Las réplicas quedarán sincronizados con cierta precisión Π . Existen muchísimos algoritmos de resincronización, en [64] se puede encontrar un estudio que compara distintos tipos de algoritmos. En [65] puede encontrarse un trabajo donde se implementa una sincronización entre distintos miembros de un bus CAN, donde no solo se hace una corrección del desfasaje, sino que además se corrige la velocidad de cada clock, obteniendo una precisión inferior a $1 \mu\text{s}$.



(a) Caso sin sincronización, las réplicas se desfasan, impidiendo la correcta ejecución del sistema. (b) La resincronización periódica mantiene el desfasaje en valores controlados.

Figura 57: Se muestra como el efecto de la resincronización periódica evita el desfasaje entre réplicas. En el eje horizontal se ubica el paso del tiempo real, mientras que el eje vertical corresponde a los clocks de las 3 réplicas, $c_1(t)$, $c_2(t)$ y $c_3(t)$. En línea punteada se muestra el caso de un clock perfecto sin drift.

Algunos sistemas distribuidos de tiempo real utilizan un clock maestro, implementado como un miembro del sistema, al que todas las demás réplicas utilizan como referencia. Por ejemplo, la extensión del protocolo automotivo CAN, denominada Time-Triggered CAN (TTCAN) [66] utiliza esta estrategia. La desventaja de este método es que dicho clock maestro se convierte en un punto singular de falla: si este presenta una falla, habrá un error en la sincronización, lo que decantará en el fracaso de todo el sistema.

Otra forma es utilizar una base de tiempos global [9, p. 51]. Esta se define como un acuerdo entre las réplicas respecto a una base de tiempo común que será utilizada como referencia. Por ejemplo, las 3 réplicas pueden intercambiar valores de sus clocks, obtener un promedio y corregir sus desfasajes contra ese valor, evitando el uso de un clock maestro.

Tomando todas los aspectos mencionados hasta aquí, el funcionamiento del sistema distribuido puede ilustrarse como en la figura 58. Cada una de las réplicas ejecuta una serie de tareas de forma periódica con cierto *timing* definido. Además, existe una base de tiempos común a todas las réplicas, permitiendo

la ejecución coordinada y sincronizada. Debido a que la sincronización debe ejecutarse de forma periódica y aprovechando las características del *scheduler*, la implementación de la tarea que corrija el desfasaje de cada clock puede incluirse como una tarea dentro del *scheduling*.

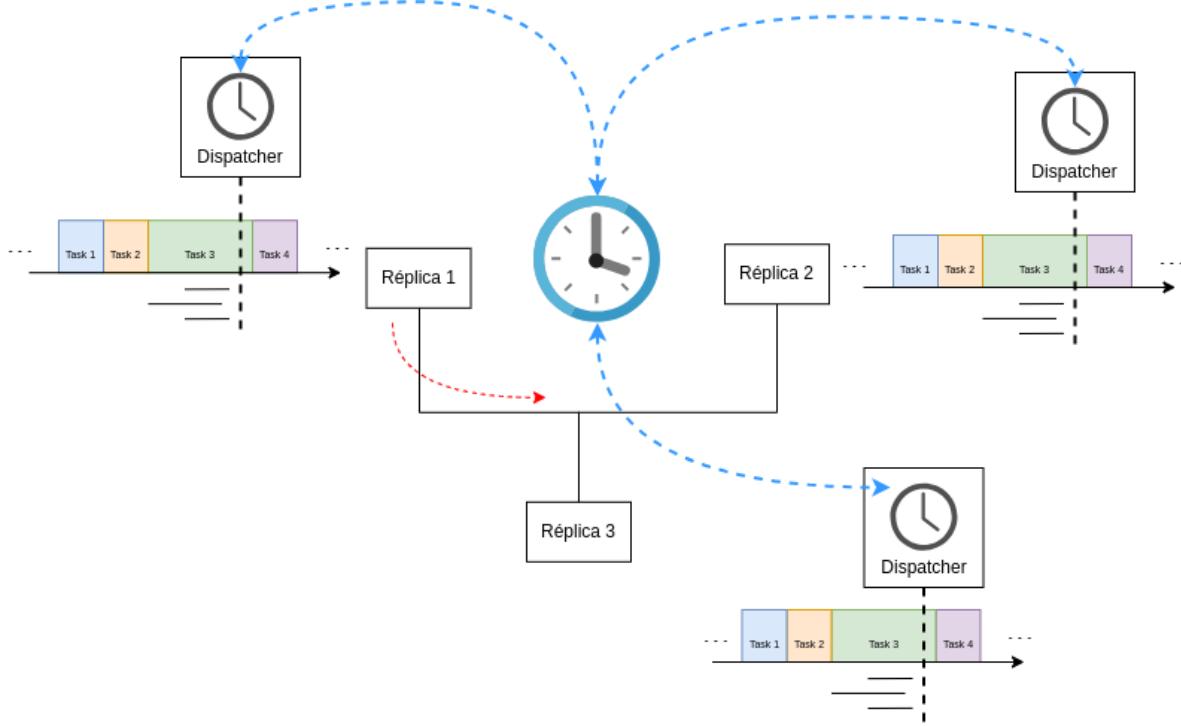


Figura 58: Se muestra un esquema que representa la arquitectura. Cada réplica tiene un *clock* local, funcionando a partir de su propio cristal. Todos estos a su vez se sincronizan periódicamente, representado por el reloj azul en el centro. En la imagen, la réplica 1 se encuentra enviando un mensaje por el bus. Al mismo tiempo, las réplicas 2 y 3 ejecutan una tarea que corresponde a recibir un mensaje y almacenarlo en memoria.

6.2. Implementación en Firmware

Para realizar pruebas con las computadoras de vuelo, se hizo una implementación en firmware de la arquitectura con las características mencionadas. Se utilizó el lenguaje de programación *C*, junto con algunas bibliotecas de terceros desarrolladas en *C++*, por ejemplo para uso del sensor IMU.

Se comienza con una inicialización de los periféricos y sensores conectados, para luego pasar a un estado en el que se espera una orden para comenzar la ejecución del *scheduler*. Esto puede ser a través de un comando externo enviado por el piloto remoto, algún mensaje por el bus CAN, o cualquier otro mecanismo. Es importante destacar que esto ocurre previo a la ejecución del *scheduler*, por lo que el sistema no tiene un comportamiento controlado por el tiempo. Previo a su ejecución, este se comporta como un sistema del tipo *event-triggered*. Si bien se mencionó que esto tiene algunas características que perjudican la seguridad, el estado previo a la ejecución del *scheduler* corresponde a una inicialización, por lo que no se compromete la seguridad del vehículo. En este estado, el UAV todavía se encuentra en tierra con sus motores apagados esperando una orden para comenzar su misión.

6.2.1. Scheduler

Para la implementación del *scheduler* se tomó como referencia lo que se propone en [61]. El autor utiliza una lista de tareas, un timer y una dispatcher que determina cuál es la siguiente tarea a ejecu-

tar. Inicialmente la lista se encuentra vacía y deben agregarse las tareas pertinentes a la aplicación a ser desarrollada. El scheduler requiere que se reserve un periférico timer configurado para generar una interrupción por overflow. Cada vez que esto ocurre, se llama al dispatcher, el cual selecciona y ejecuta la próxima tarea.

Con el objetivo de darle mayor visibilidad al código, es decir que pueda entenderse el flujo de ejecución fácilmente, tanto el scheduler como las tareas se implementaron a partir de estructuras que emulan el comportamiento de una clase del lenguaje de programación *C++*. Además, esto permite la reutilización de código, lo cual se aprovecha sobre todo para la creación de cada una de las tareas. En la figura 59 se muestra un diagrama de clases que relaciona al scheduler con las tareas a ejecutar.

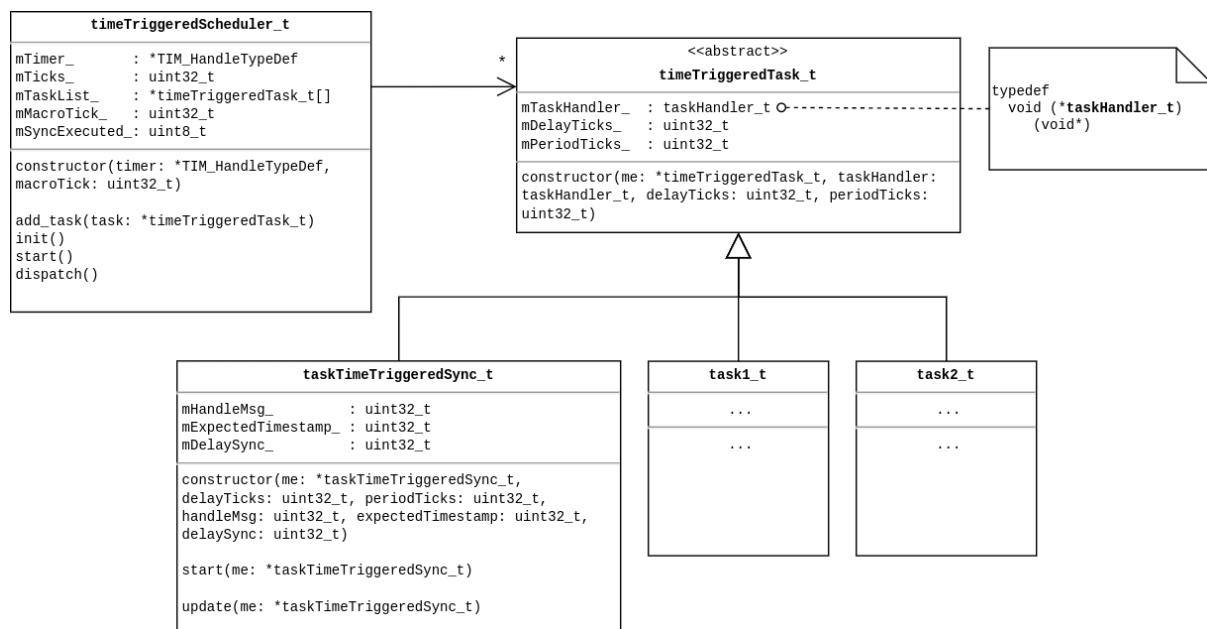


Figura 59: Diagrama de clases. La lista de tareas del scheduler se implementa como un arreglo de punteros. Cada una de las tareas del scheduler se crea a partir de una tarea base, *timeTriggeredTask_t*. Se muestra a modo de ejemplo la tarea encargada de ejecutar la resincronización.

Todas las tareas del sistema comparten una serie de atributos básicos que son aprovechados por el scheduler para determinar sus características temporales, en particular los atributos delay (*mDelayTicks_*) y período (*mPeriodTicks_*). A su vez, cada tarea tiene otras características que la diferencian de las demás. Esto se resuelve a partir de la definición de la clase *timeTriggeredTask_t*. A partir del uso de herencia, se definen todas las demás tareas. Si bien este aspecto se asocia más con el lenguaje *C++*, es posible emularlo en el lenguaje *C* [67, p. 32].

El tiempo de ejecución se divide en intervalos discretos de duración fija. El comienzo de cada intervalo se denomina tick y es allí donde se llama al dispatcher. En la figura 60 se muestra un ejemplo de cómo el scheduler realiza la selección de la tarea a ejecutar. Como se mostró anteriormente, cada tarea cuenta con un atributo que indica su delay y su período. Estos atributos se miden en cantidad de ticks y son observados por el scheduler al momento de seleccionar la siguiente tarea a ejecutar.

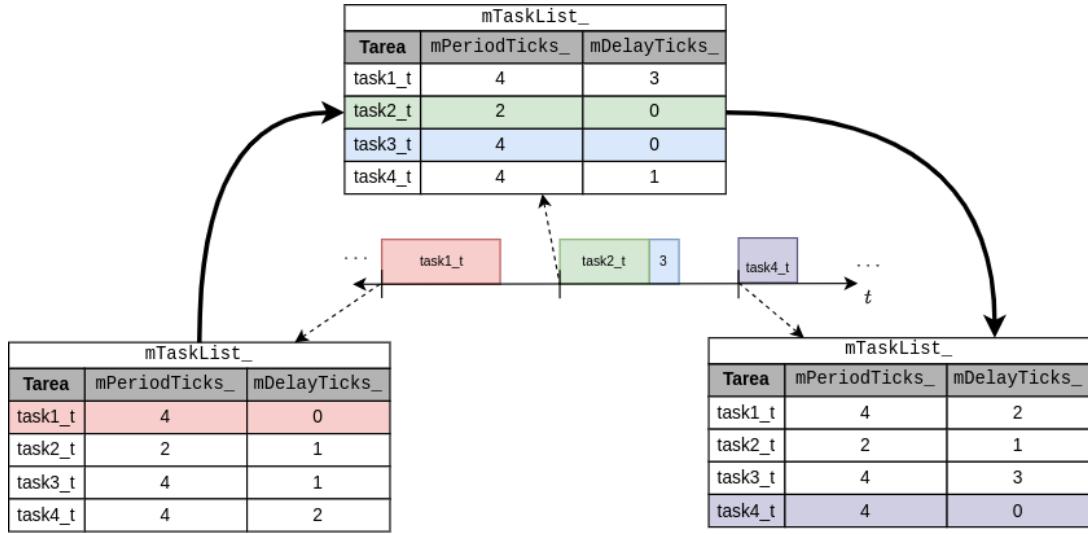


Figura 60: En cada tick se llama al dispatcher el cual selecciona la tarea a ejecutar. Todas las tareas se almacenan en la lista del atributo `mTaskList_` del scheduler. En la imagen se muestra la evolución de la lista a medida que ocurren los ticks. En cada tick, el atributo `mDelayTicks_` de todas las tareas se decrementa en 1. En caso de que se detecte alguna tarea que alcanza un `mDelayTicks_ = 0`, esta es ejecutada. Para finalizar, se vuelve a setear `mDelayTicks_ = mPeriodTicks_`, permitiendo que las tareas puedan volver a ser ejecutadas.

En el ejemplo de la figura 60, en el primer tick se ejecuta la tarea `task1_t`, lo cual corresponde con el llamado a su función `update()`. Luego en el siguiente tick se ejecutan 2 tareas, `task2_t` y `task3_t`. En dicho caso, las tareas se ejecutarán una a continuación de la otra. Finalmente en el tercer tick, se ejecuta la función `update()` de `task4_t`.

El tick generado por el scheduler es la única interrupción que se permite en todo el programa. Ninguna de las tareas puede hacer uso de las interrupciones, ya que como fue mencionado anteriormente, todas las acciones deben tener un instante de ejecución predefinido con motivo de preservar el determinismo del sistema.

6.2.2. Comunicación a Través del Bus

Dado que el comportamiento del sistema se encuentra predefinido, luego los mensajes, su información y sus correspondientes instantes de tiempo en los que se recibirán y en los que se enviarán, también lo están. Para la implementación de la comunicación a través del bus CAN, se utiliza una lista estática de mensajes definida en tiempo de compilación. Cada entrada de la lista contiene información asociada a un mensaje distinto. Para esto se creó un tipo de dato `CANmsg_t` con los atributos que se muestran en la figura 61. El primer elemento de cada entrada de la lista, `mServiceID_`, es un indicador del contenido del mensaje, mientras que `mNodeID_` indica quién es el responsable de enviar ese mensaje.

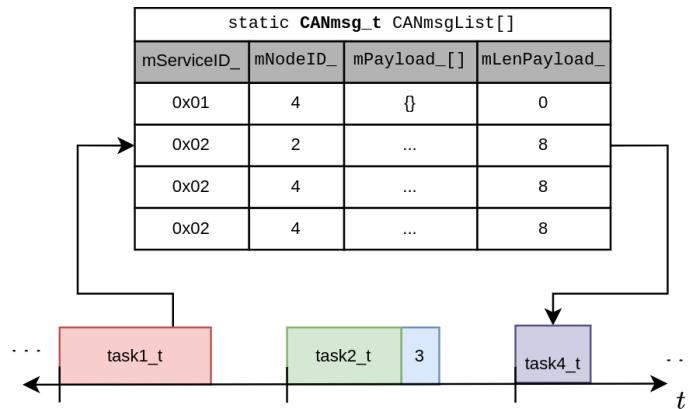


Figura 61: La tarea `task1_t` almacena información relevante en formato de mensaje, en la lista `CANmsgList[]`, en una posición conocida por la misma. Luego, la tarea `task4_t` lee dicha información y envía el mensaje a través del bus CAN.

En la lista no se almacena ningún aspecto relacionado a la temporalidad de los mensajes. Cada tarea debe acceder a la lista para obtener el mensaje y enviarlo cuando sea necesario, o bien recibir un mensaje proveniente de otra réplica y almacenarlo en su lista. Cabe aclarar que todas las réplicas deben tener definido en firmware una copia exacta de la misma tabla para que el sistema funcione adecuadamente.

En cuanto al uso del periférico CAN, este se configuró para trabajar en su velocidad más alta, 1 MBit/s. En su configuración por defecto, en caso de no poder injectar un mensaje en el bus de comunicaciones, el periférico no descarta el mensaje, sino que lo retiene hasta que pueda ser enviado. Este comportamiento no es aceptable, ya que perjudica gravemente el determinismo de las comunicaciones y no puede tolerarse. Afortunadamente, dentro de las tantas configuraciones, pueden deshabilitarse las retransmisiones automáticas. De esta forma, en caso de que la transmisión falle, el mensaje será descartado y se dará aviso al programa principal.

En cuanto a la recepción de mensajes, el funcionamiento es similar. En este caso además existe un mecanismo de filtrado de mensajes indeseados. Como fue mencionado en la sección 5.3.5, el campo identifier indica el contenido del mensaje CAN. La configuración de filtros permite que solamente se acepten mensajes con determinado valor en su campo identifier. Cada vez que se recibe un mensaje, este queda almacenado en otras mailboxes, las cuales también se configuran para trabajar como FIFO.

Una aclaración importante es que en la hoja de datos del microcontrolador utilizado, se menciona que este contiene el hardware necesario para utilizar el bus CAN con un funcionamiento *time-triggered* [68, p. 1295]. Sin embargo, el fabricante ST ofrece una errata [69] donde se aclara que esto no es así. Es por esto que tuvo que utilizarse el timer del scheduler para darle determinismo a la transmisión y recepción de mensajes.

6.2.3. Sincronización

Como fue mencionado anteriormente, algunos sistemas time-triggered utilizan un clock maestro. Por ejemplo, el protocolo TTCAN mencionado en la sección 5.3.5 utiliza este mecanismo para sincronizar a todos los miembros del bus. Si bien esto tiene como desventaja el hecho de que el clock maestro se convierte en un punto singular de fallas de todo el sistema, con el motivo de simplificar las pruebas realizadas, se adopta una sincronización con estas características. Dentro del trabajo futuro a realizar debe implementarse un algoritmo distribuido, que sea más seguro y confiable para el sistema. En este trabajo, el uso de un clock maestro es suficiente para demostrar el funcionamiento de la arquitectura que se propone.

Al igual que el resto de las funcionalidades, la sincronización se ejecuta como una tarea más del scheduler. Dependiendo de si se trata del maestro o del esclavo, la tarea realiza una acción distinta. En

la figura 62 se resume el proceso de sincronización entre las réplicas.

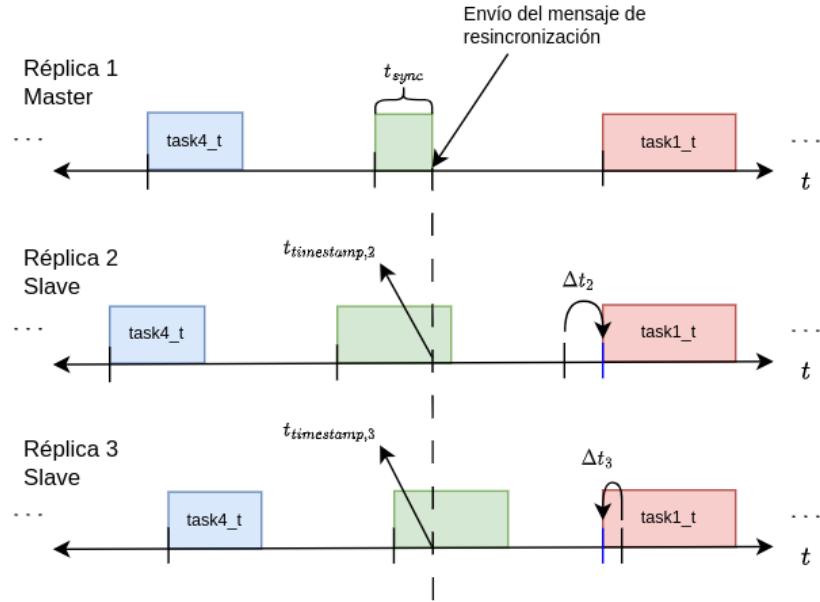


Figura 62: Se muestra el funcionamiento del mecanismo de resincronización periódica, implementado en firmware. La línea punteada indica el instante de tiempo en el que la réplica 1 envía el mensaje de resincronización a las demás réplicas. Estas toman un timestamp cuando reciben el mensaje y calculan qué tan desfasadas se encuentran respecto del master. Finalmente, aplican la corrección a sus timers.

El proceso comienza cuando la réplica 1 ejecuta su tarea de resincronización, en verde en la figura. En esta tarea, primero se deja pasar un breve período de tiempo t_{sync} y luego simplemente se envía un mensaje con un determinado mServiceID_— = 0x01. Este período de tiempo es necesario para asegurar que aquellas réplicas que sean más lentas que el maestro, tengan tiempo de entrar en la tarea de resincronización. Este es el caso de la réplica 3 de la figura.

En cuanto a las demás réplicas, cuando estas ejecutan su tarea de sincronización, estarán esperando a recibir el mensaje proveniente del maestro. En ese momento, consultarán el valor del contador del timer utilizado para el scheduler y obtendrán un valor $t_{timestamp}$. A partir de la comparación de este valor contra el valor esperado t_{sync} , se obtendrá un error de sincronización, el cual deberán utilizar para corregir sus propios schedulers. En este caso, $\Delta t_2 = t_{timestamp,2} - t_{sync}$ y $\Delta t_3 = t_{timestamp,3} - t_{sync}$, donde $\Delta t_2 > 0$ y $\Delta t_3 < 0$.

El paso de corrección se aplica alargando o achicando la duración del tick actual. El periférico del timer utilizado en el microcontrolador de este trabajo permite modificar el valor del registro de overflow, TIMx_ARR, de forma directa. Esto se realiza a partir de setear el bit ARPE = 0 (auto-reload preload bit) en el registro TIMx_CR1 [68, p. 745]. El valor a setear en el registro TIMx_ARR debe ser $t_{tick} + \Delta t_{sync}$, donde t_{tick} es la duración de un tick durante el funcionamiento normal y $\Delta t_{sync} = t_{timestamp} - t_{sync}$.

6.3. Pruebas Realizadas

Para demostrar las capacidades de todo el sistema, se presentan una serie de pruebas realizadas para la detección de fallas en el sensor IMU. Durante la ejecución del scheduler, las réplicas intercambian resultados de estimaciones de los ángulos de pitch y roll, obtenidas a partir de un filtro complementario [70]. Estas comparan sus resultados con los de sus pares y luego aquella que difiera con respecto a las demás se considera en falla. Utilizando este método y a partir de la arquitectura propuesta, es posible detectar la falla simultánea de 1 de las réplicas.

En la tabla 6 se muestran las tareas con sus características temporales y en la figura 63 se muestra un gráfico temporal. A partir de esta secuencia, cada réplica obtiene una nueva estimación de ángulos de pitch y roll cada 10 ms.

Duración Tick = 1 ms		
Tarea	Período [ticks]	Delay [ticks]
Heartbeat	1000	0
Watchdog	1	0
Obtener dato IMU	10	0
Filtro Complementario	10	0
Enviar pitch y roll	10	1
Recibir pitch y roll de 2	10	2
Recibir pitch y roll de 3	10	3
Calcular residuos	10	4
Enviar residuos	10	5
Sincronización	1000	8

Tabla 6: Set de tareas correspondiente a la réplica 1, para las pruebas realizadas.

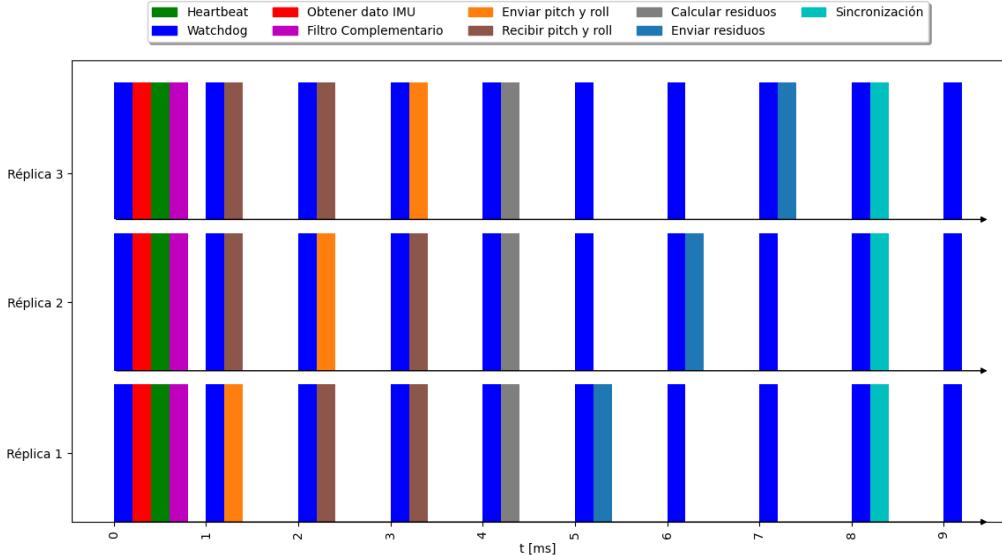


Figura 63: Se muestran los primeros 9 ms de ejecución del scheduler, donde se muestran las tareas de las 3 réplicas.

Mientras que la tarea “Obtener dato IMU” se encarga de consultar al sensor y obtener las mediciones de acelerómetros y giróscopos, la tarea “Filtro Complementario” realiza el procesamiento y obtiene una nueva estimación de los ángulos de pitch y roll. Este filtro realiza una estimación combinada, fusionando los datos de los acelerómetros y giróscopos. Se aplica un filtro pasa bajos a la estimación utilizando las mediciones de acelerómetro, debido a que sus mediciones son muy ruidosas. En cuanto a las mediciones de giróscopos, estas poseen un sesgo el cual se agrava por efecto de la integración. Es por esto que se aplica un filtro pasa altos a la estimación por giróscopos. Finalmente, se suman ambos resultados para obtener una mejor estimación, representados por las siguientes ecuaciones, donde $\theta[n]$ es el ángulo de pitch y $\phi[n]$ es el ángulo de roll:

$$\theta[n] = \alpha \operatorname{atan}_2(-a_x[n], a_z[n]) + (1 - \alpha) (\Delta T g_y[n] + \theta[n - 1]) \quad (6a)$$

$$\phi[n] = \alpha \operatorname{atan}_2(a_y[n], a_z[n]) + (1 - \alpha) (\Delta T g_x[n] + \phi[n - 1]) \quad (6b)$$

El valor α es un parámetro de ajuste del filtro, con un valor entre 0 y 1. Para las pruebas realizadas fue configurado con un valor de 0,025.

Seguido de esto, hay un período de intercambio de resultados de las estimaciones, a través del bus CAN. Esto corresponde a las tareas “Enviar pitch y roll” y “Recibir pitch y roll”, las cuales permiten que todas las réplicas tengan una copia de las estimaciones de sus pares. Estos resultados se comparan en el tick 6, cuando se ejecuta la tarea “Calcular residuos”. Se llama residuo a la diferencia entre cualquier par de resultados obtenido por las réplicas. Así, se obtienen 6 residuos, 3 para el ángulo de pitch y 3 para el ángulo de roll:

$$r_{1,2}^\theta[n] = | \theta_1[n] - \theta_2[n] | \quad (7a)$$

$$r_{1,2}^\phi[n] = | \phi_1[n] - \phi_2[n] | \quad (7b)$$

$$r_{1,3}^\theta[n] = | \theta_1[n] - \theta_3[n] | \quad (7c)$$

$$r_{1,3}^\phi[n] = | \phi_1[n] - \phi_3[n] | \quad (7d)$$

$$r_{2,3}^\theta[n] = | \theta_2[n] - \theta_3[n] | \quad (7e)$$

$$r_{2,3}^\phi[n] = | \phi_2[n] - \phi_3[n] | \quad (7f)$$

En caso de que todas las réplicas obtengan resultados consistentes, los residuos tendrán valores cercanos a 0. Estos nunca alcanzarán un valor exactamente 0, debido a variaciones entre unidades del mismo sensor, propias del proceso de fabricación, además de errores de cuantización. En caso de que ocurra una falla en una de las IMUs, la estimación del pitch y/o roll de esa réplica divergirá con respecto a las demás. Esto generará que los valores de 2 de los residuos se alejen de cero, indicando cuál es el sensor en falla.

Con motivo de observar el valor de los residuos, se agrega una tarea “Enviar residuos” donde cada réplica envía por el bus CAN los residuos que calcularon. Utilizando un analizador lógico, se lograran todos los mensajes intercambiados en el bus CAN.

El set de tareas se ejecuta durante 60 segundos, orientando a las réplicas siguiendo el esquema de la figura 64. Para que la orientación de las réplicas sea la misma en todo momento, estas se apilan con una serie de soportes, como se muestra en la figura 65.

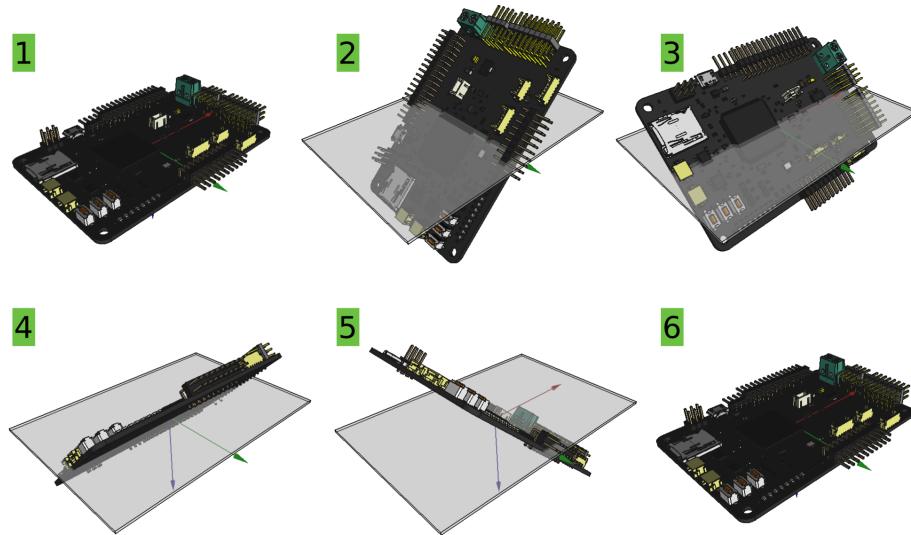


Figura 64: Durante los primeros 10 segundos, las réplicas se orientan como en el caso 1. Luego se continua con el caso 2 durante otros 10 segundos, y así sucesivamente hasta finalizar en el caso 6, donde la placa vuelve a su condición inicial. Cada una de las orientaciones que se muestran corresponden a una rotación de aproximadamente 45° .

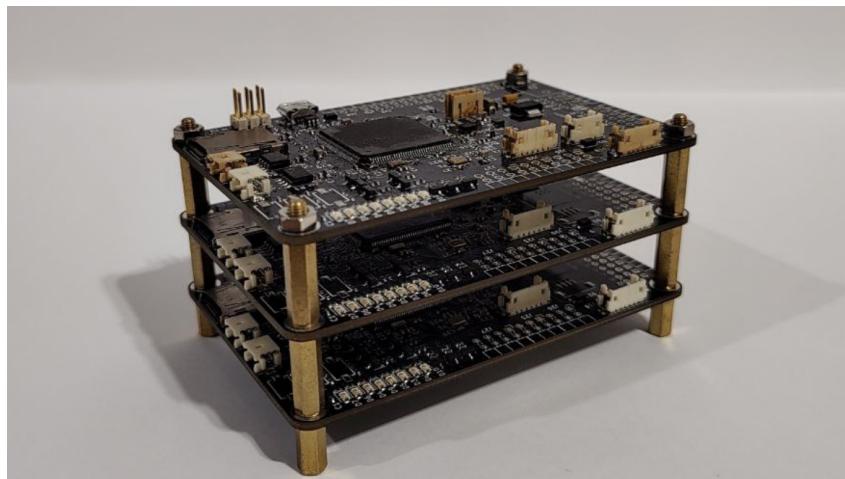


Figura 65: Se muestran las 3 réplicas apiladas. Esta foto estaría bueno que muestre a las 3 réplicas, pero conectadas al bus CAN y con la Núcleo junto con el analizador lógico. La idea es mostrar algo más "de laboratorio", que se vea que esto está vivo. En esta foto parece una maqueta, algo que se mira y no se toca, cuando no es así.

Para verificar que se está cumpliendo con los tiempos durante el intercambio de mensajes en el bus CAN, se tomaron capturas de todos los mensajes. En los resultados que se presentan a continuación, todas las fallas fueron inyectadas de manera artificial, con el objetivo de emular un determinado comportamiento del sensor IMU. La inyección de la falla siempre se aplica dentro de la tarea “Obtener dato IMU”, antes de guardar las mediciones en la tabla de cada réplica.

6.3.1. Funcionamiento Sin Fallas

En este primer caso no se inyectó ninguna falla en las mediciones de la IMU. Aquí se presentan los primeros resultados y se muestra el funcionamiento del sistema conformado por las 3 réplicas.

Para confirmar que el sistema se encuentra funcionando correctamente, lo más importante es verificar que las 3 réplicas ejecutan sus tareas en los tiempos definidos, además de que se encuentran trabajando de manera sincronizada. Para ello, se hicieron distintas mediciones sobre el bus CAN. En primera medida, en la figura se muestra una captura de la actividad en el bus CAN, utilizando un osciloscopio.

CAPTURA DE OSCILOSCOPIO DEL BUS CAN. TAMBIÉN MOSTRAR CÓMO SE CONECTÓ, QUE SE VEA LA PUNTA DEL OSCILOSCOPIO COLGADA DEL BUS.

Para verificar que las réplicas se encuentran sincronizadas es necesario observar la actividad del bus durante un mayor período de tiempo. Para esto, se hicieron capturas utilizando el analizador lógico. En la figura 66 se muestra un fragmento de 100 ms, donde se pueden ver una serie de mensajes, los cuales se repiten de forma periódica. Estos corresponden a la secuencia que fue definida en el schedule de la tabla 6.

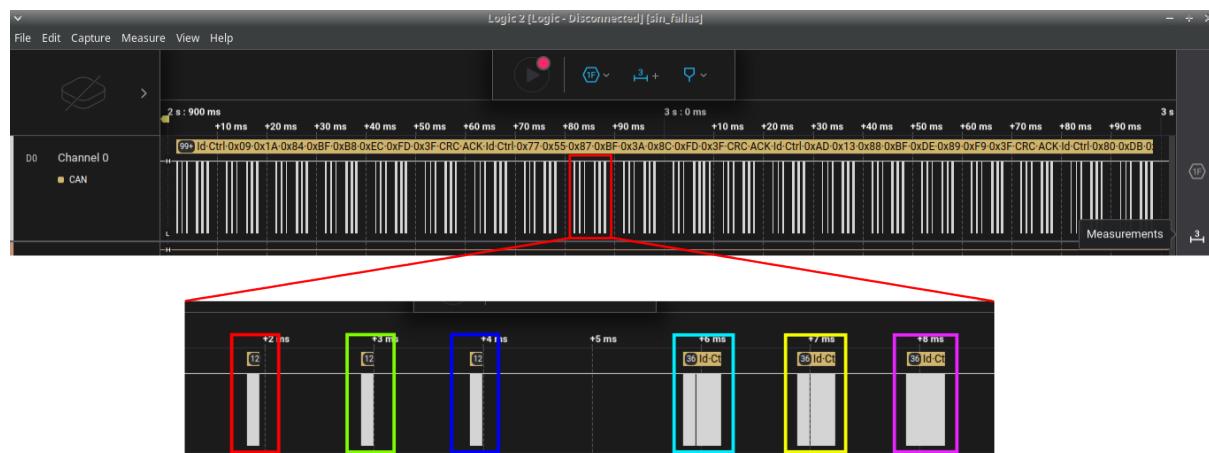


Figura 66: Captura tomada del software del analizador lógico. Se muestra un fragmento de 100 ms de actividad en el bus CAN. Se muestra un acercamiento de uno de los períodos, donde se destacan todos los mensajes involucrados en el schedule. En rojo, verde y azul se muestran los mensajes correspondientes a la tarea “Enviar pitch y roll” de las placas 1, 2 y 3 respectivamente. En celeste, amarillo y rosado, los mensajes correspondientes a la tarea “Enviar residuos” para las placas 1, 2 y 3.

Utilizando las herramientas que provee el software del analizador lógico, se midieron los intervalos de tiempo entre mensajes. Estos se muestran en la figura 67. Comparando los valores medidos contra los valores de la tabla 6, se verifica que las placas respetan los tiempos de uso del bus

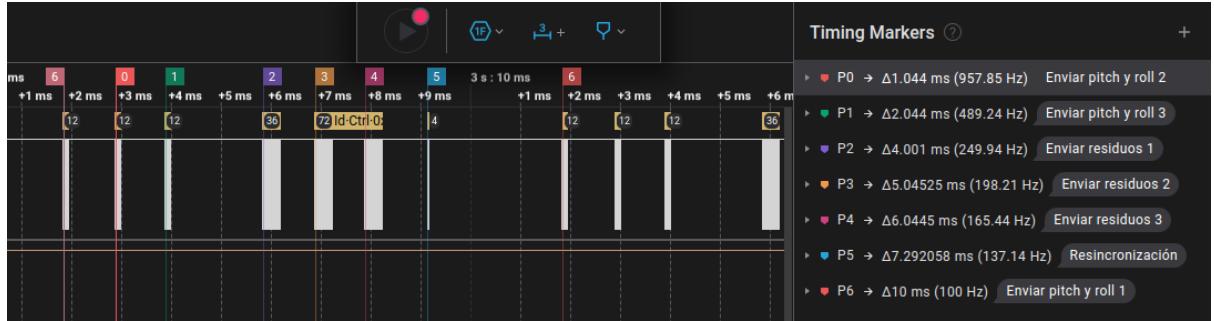


Figura 67: Utilizando la funcionalidad de markers se midieron los intervalos de tiempo entre los mensajes presentes en el bus CAN. La imagen muestra la misma secuencia de mensajes de la figura 66. En el apartado Timing Markers se muestran los valores medidos. Se tomó como referencia el primer mensaje, correspondiente a la tarea “Enviar pitch y roll” de la placa 1. El último valor muestra que la tarea vuelve a repetirse al cabo de 10 ms.

Para determinar que las 3 réplicas efectivamente se encuentran funcionando de manera sincronizada, se hizo un análisis similar al de la figura 67 para los 60 segundos de ejecución del scheduler. En la secuencia de mensajes, siempre que la réplica 2 envía un mensaje, esta lo hace 1 ms después que la placa 1, tanto para la tarea “Enviar pitch y roll” como “Enviar residuos”. En cuanto a la réplica 3, esta lo hace 2 ms después que la placa 1. A partir de los datos tomados con el analizador lógico se hizo el gráfico de la figura 67, donde se verifica que esto se cumple durante los 60 segundos de ejecución.

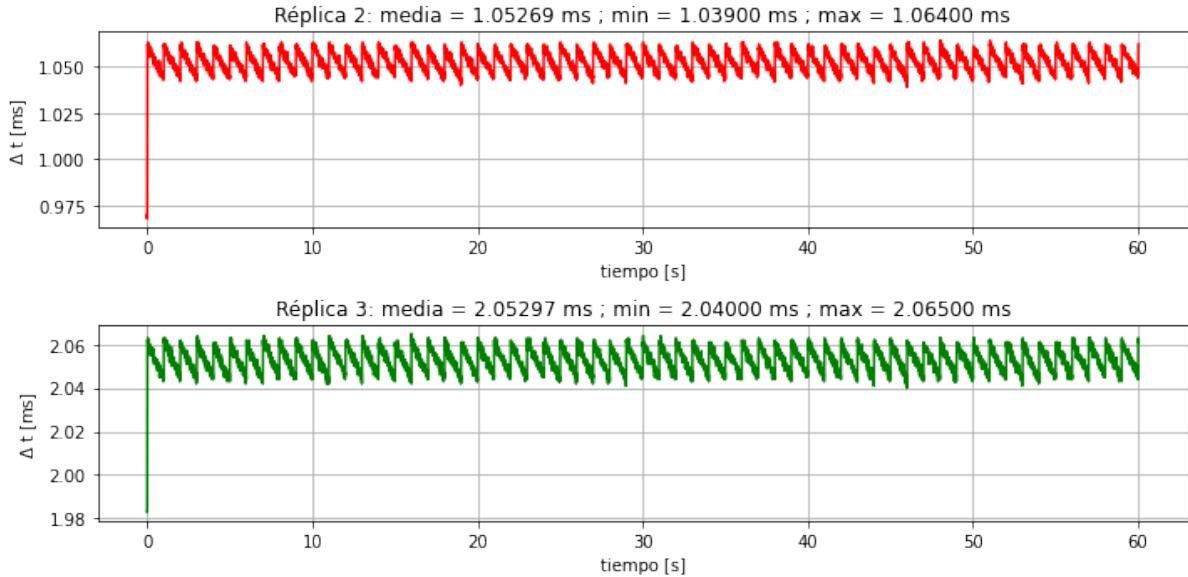


Figura 68: En cada gráfico se muestran los intervalos de tiempo, medidos por el analizador lógico, entre mensajes enviados por las réplicas 2 y 3 con respecto a los mensajes enviados por la réplica 1. Los intervalos de tiempo se miden entre mensajes homónimos.

El salto que se muestra en el instante inicial corresponde a la primera sincronización. La forma característica del gráfico tipo “diente de sierra” muestra el efecto de la sincronización periódica, donde cada salto abrupto indica una nueva resincronización. Esto ocurre cada 1 segundo, de acuerdo al período de tiempo de la tarea “Sincronización”, definida en la tabla 6.

Con motivo de mostrar la necesidad de la sincronización periódica, se repitió la ejecución del scheduler

pero removiendo la tarea de sincronización en la réplica 3. Es esperable que en los primeros instantes de ejecución, el sistema funcione normalmente. Sin embargo, debido a que la placa 3 no ajusta su timer, esta se irá desincronizando lentamente. Esto último es lo que se muestra en la figura 69. El hecho de que el gráfico sea una recta es debido al efecto del drift, mencionado previamente en la figura 57.

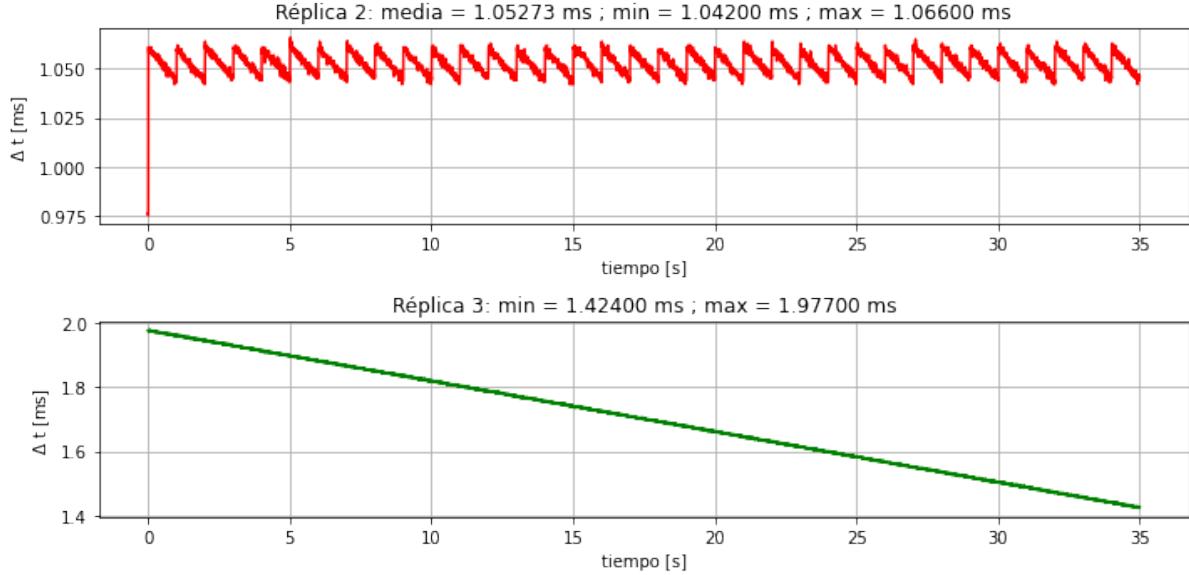


Figura 69: El hecho de que la réplica 3 no ejecute una sincronización periódica genera que esta no respete los tiempos de uso del bus. Al cabo de 35 segundos el desincronismo de la réplica 3 es tal que comienza a perjudicar la comunicación entre las demás réplicas.

La desincronización también se puede ver a partir de la actividad registrada en el bus CAN. En la figura 70 se muestran 3 fragmentos capturados con el analizador lógico. A medida que pasa el tiempo, los mensajes enviados por la réplica 3 se acercan lentamente a los instantes de tiempo donde se envían los mensajes de la réplica 2. Comparando los valores de los intervalos de tiempo con las etiquetas P1 y P4 de las 3 imágenes, se puede ver que al cabo de 10 s hubo un corrimiento de aproximadamente 150 μ s. Luego, a los 30 s de ejecución el corrimiento alcanza los 470 μ s. En este último caso, el mensaje de la réplica 3 se encuentra próximo a causar una colisión con el de la réplica 2.

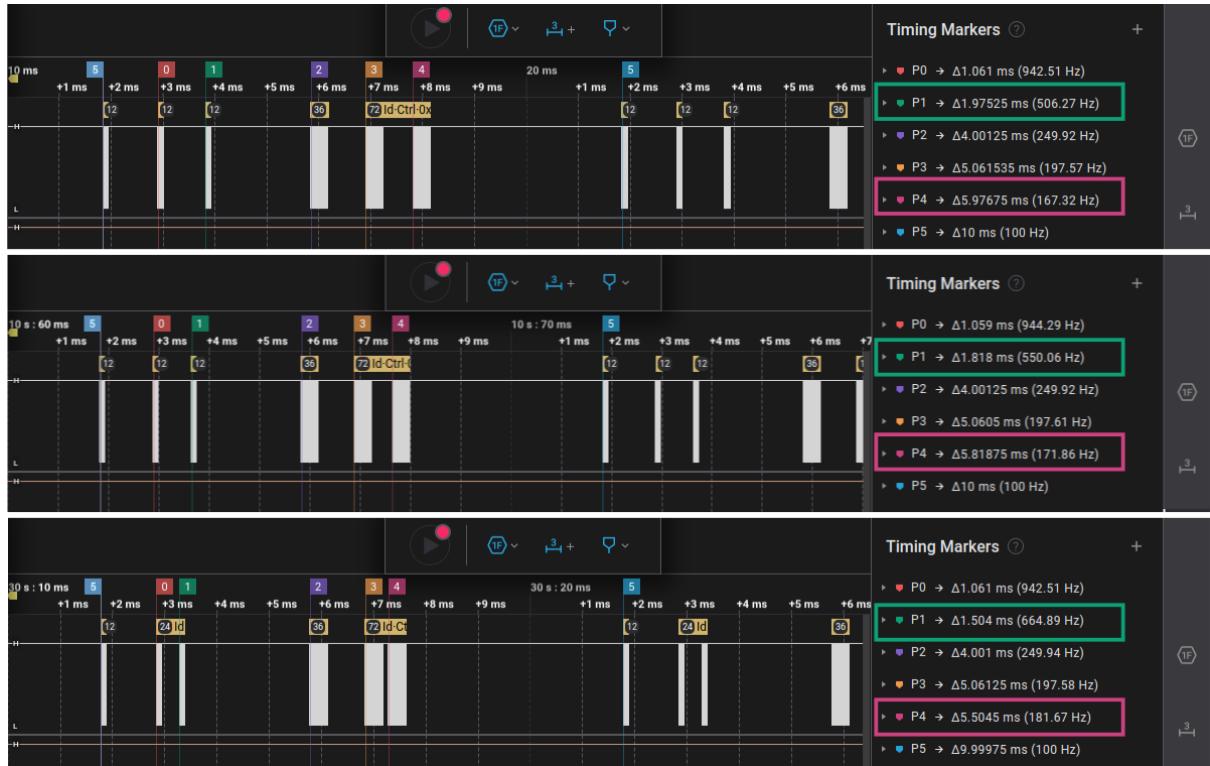


Figura 70: Se muestran 3 fragmentos capturados con el analizador lógico. La primera imagen corresponde a los primeros 10 ms de ejecución, la segunda se tomó luego de 10 s y la tercera a los 30 s. En el apartado Timing Markers se muestran las mismas mediciones de la figura 67.

Utilizando una serie de scripts desarrollados en el lenguaje de programación Python, se pudo extraer la información de los mensajes capturados con el analizador lógico, correspondientes a las tareas “Enviar pitch y roll” y “Enviar residuos”. En la figura 71 se grafican los valores de pitch y roll de las 3 placas superpuestas. En la figura 72 se muestran los residuos de pitch y en 73 los residuos de roll.

Debido a que no se inyectaron fallas en esta ejecución, las curvas de pitch y roll se encuentran prácticamente superpuestas, indicando que las 3 réplicas obtuvieron resultados consistentes durante la ejecución. Sin embargo en los gráficos de residuos se puede ver que sí existen ciertas diferencias entre las 3 réplicas, aunque estas se encuentran por debajo de los 0,5°.

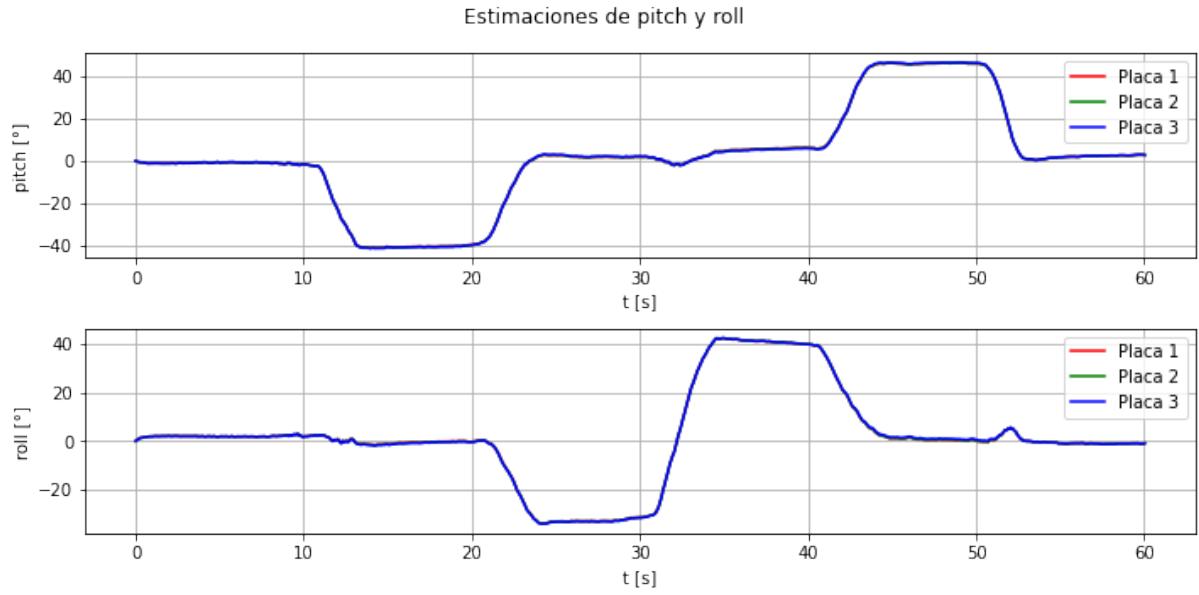


Figura 71: Se muestran los resultados de las estimaciones de pitch y roll para las 3 placas. Los gráficos se encuentran prácticamente superpuestos todo el tiempo, ya que no hubo fallas en la ejecución.

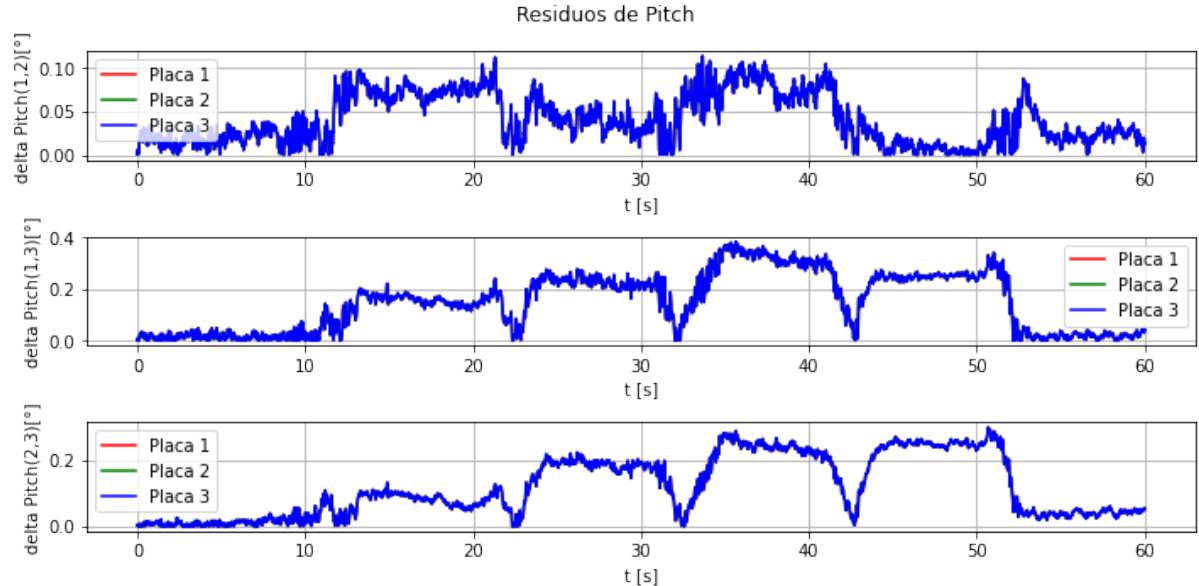


Figura 72: Se muestran los residuos de pitch calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\theta[n]$, el segundo a $r_{1,3}^\theta[n]$ y el tercero a $r_{2,3}^\theta[n]$.

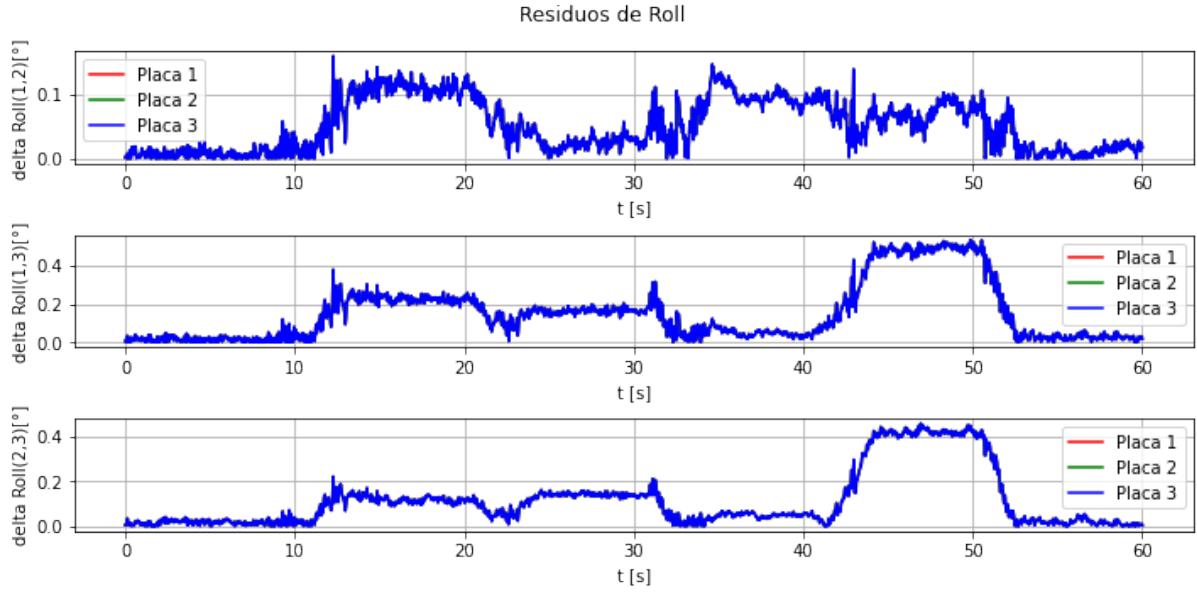


Figura 73: Se muestran los residuos de roll calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\phi[n]$, el segundo a $r_{1,3}^\phi[n]$ y el tercero a $r_{2,3}^\phi[n]$.

6.3.2. Sesgo en Valores de Giróscopo

En esta sección se repitió la ejecución del scheduling de la tabla 6, pero inyectando de manera artificial un sesgo de $+10^\circ/\text{s}$ en las mediciones de giróscopo en el eje x, $g_x[n]$, reportadas por la IMU de la réplica 3. Al igual que en la sección anterior, se presentan los gráficos con las estimaciones de pitch y roll, además de los residuos calculados por las 3 réplicas.

La falla se inyectó pasados los 10 segundos de ejecución, por lo que hasta ese instante, los residuos se encuentran en valores similares a los del caso sin fallas. En el momento en el que se inyecta la falla, los residuos $r_{1,3}^\phi$ y $r_{2,3}^\phi$ rápidamente crecen. Debido a que el residuo $r_{1,2}^\phi$ no se ve afectado, es evidente que la réplica 3 se encuentra en falla.

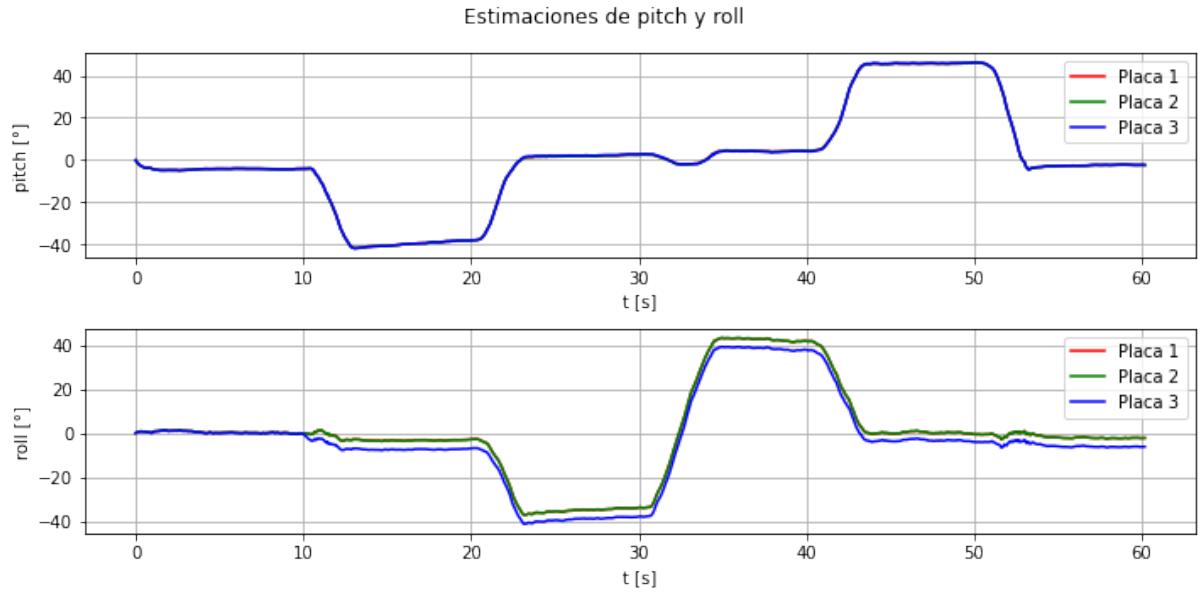


Figura 74: Se muestran los resultados de las estimaciones de pitch y roll para las 3 placas. Los gráficos se encuentran prácticamente superpuestos todo el tiempo, ya que no hubo fallas en la ejecución.

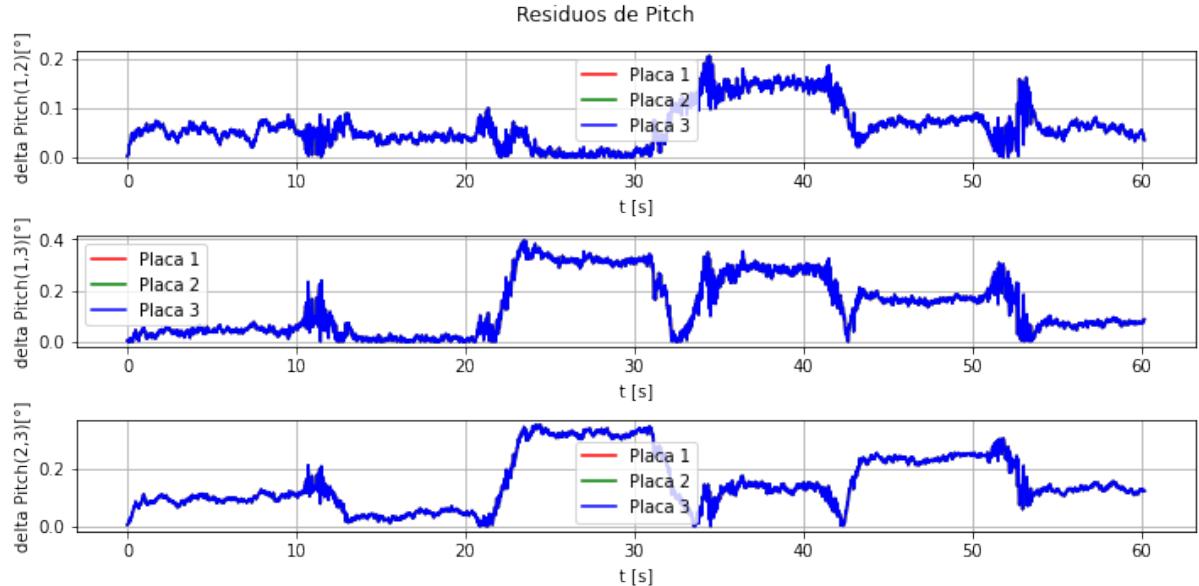


Figura 75: Se muestran los residuos de pitch calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\theta[n]$, el segundo a $r_{1,3}^\theta[n]$ y el tercero a $r_{2,3}^\theta[n]$.

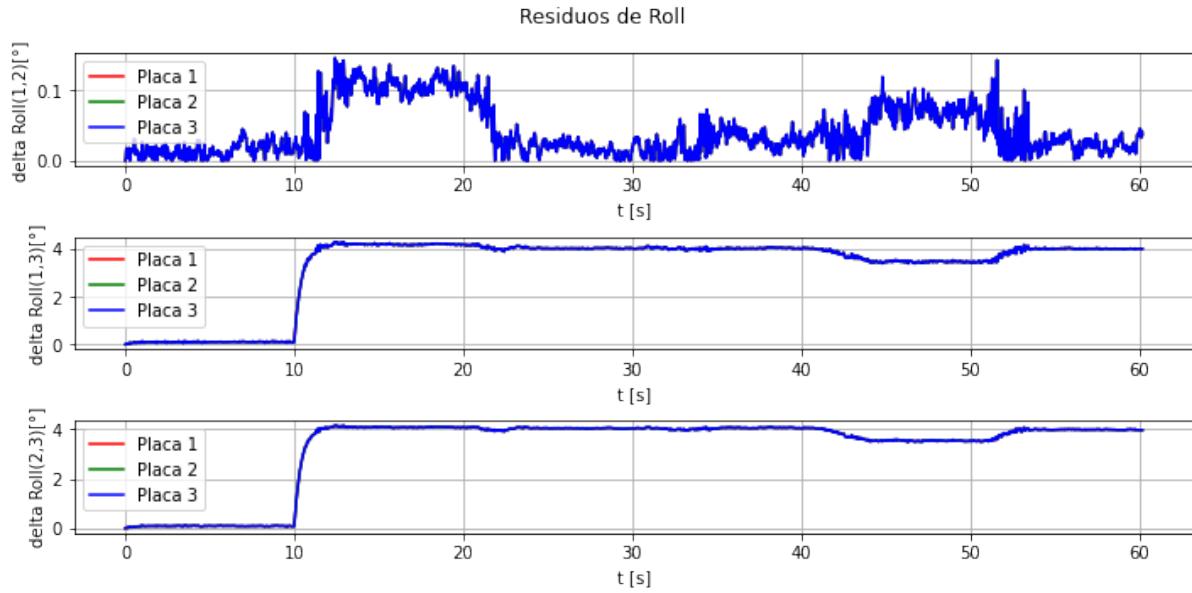


Figura 76: Se muestran los residuos de roll calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\phi[n]$, el segundo a $r_{1,3}^\phi[n]$ y el tercero a $r_{2,3}^\phi[n]$.

6.3.3. Saltos Aleatorios en Valores de Giróscopo

En esta sección se presentan los resultados de aplicar una falla que se manifiesta como saltos aleatorios sobre los valores de giróscopo en el eje x, reportados por la IMU de la réplica 3. Cada cierto período de tiempo, se le aplica un sesgo a la medición de giróscopo en el eje x de la réplica 3. Este período de tiempo, al igual que el valor del sesgo aplicado, se definen de manera aleatoria. Al igual que en el caso anterior, la falla comienza a manifestarse pasados los 10 segundos de ejecución.

La falla de esta sección tiene la particularidad de que se manifiesta durante un período de tiempo muy breve, y luego desaparece. Lo que se destaca de los resultados de esta sección, es el hecho de que la falla igualmente puede detectarse de manera inmediata. Esto es gracias a la arquitectura utilizada, donde todas las tareas tienen sus instantes de ejecución predefinidos.

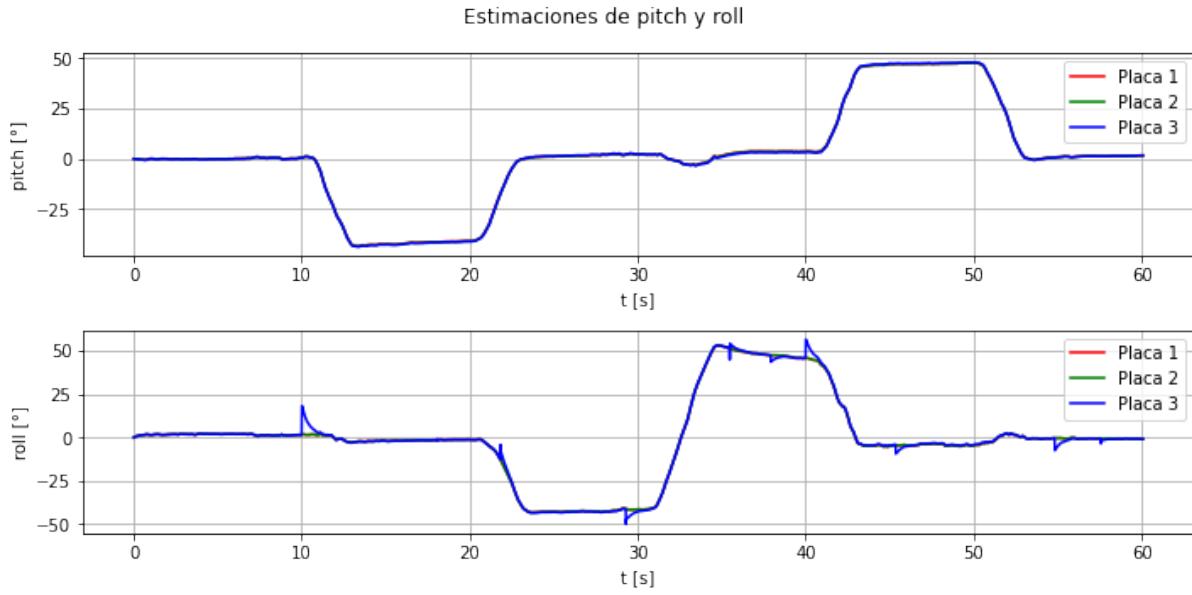


Figura 77: Se muestran los resultados de las estimaciones de pitch y roll para las 3 placas. Los gráficos se encuentran prácticamente superpuestos todo el tiempo, ya que no hubo fallas en la ejecución.

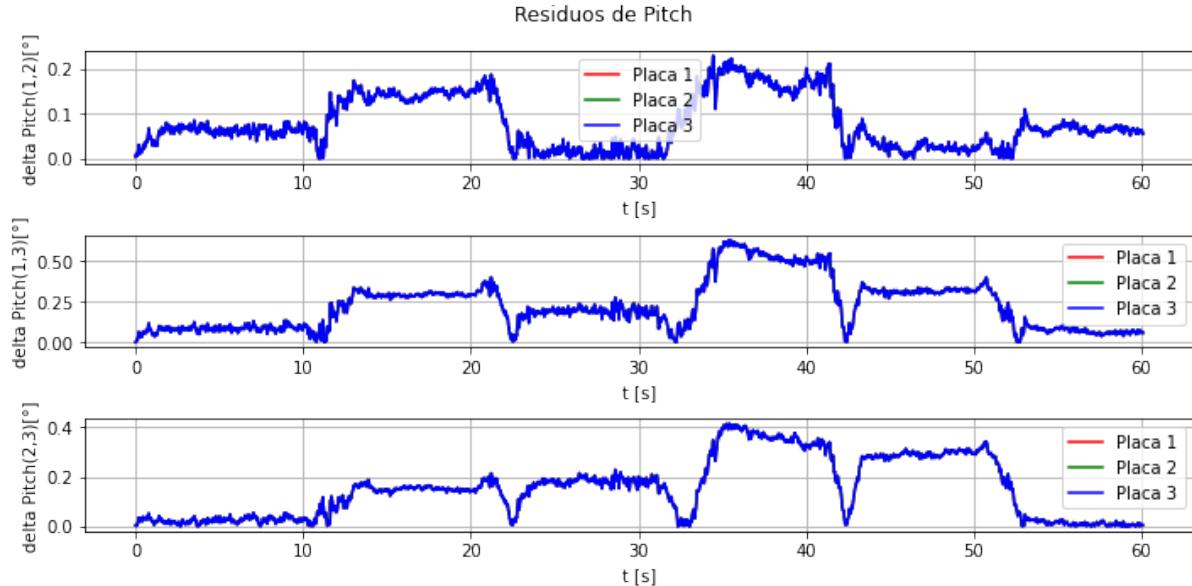


Figura 78: Se muestran los residuos de pitch calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\theta[n]$, el segundo a $r_{1,3}^\theta[n]$ y el tercero a $r_{2,3}^\theta[n]$.

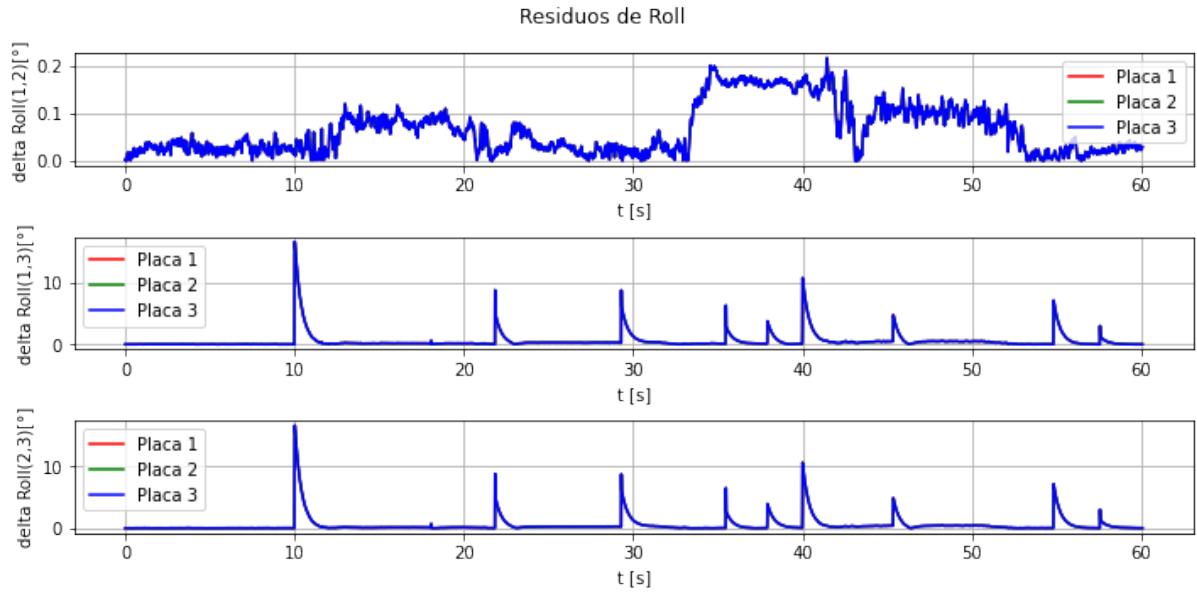


Figura 79: Se muestran los residuos de roll calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\phi[n]$, el segundo a $r_{1,3}^\phi[n]$ y el tercero a $r_{2,3}^\phi[n]$.

6.3.4. Medición Invariante de Acelerómetros y Giróscopos

El modo de falla evaluado aquí es uno donde el sensor IMU de la réplica 3 entrega siempre los mismos valores de mediciones, para los acelerómetros y giróscopos. Al igual que en los casos anteriores, la falla se inyecta a partir de los 10 segundos de ejecución. Es fácil ver en los gráficos de residuos, que la réplica 3 es la que manifiesta la falla.

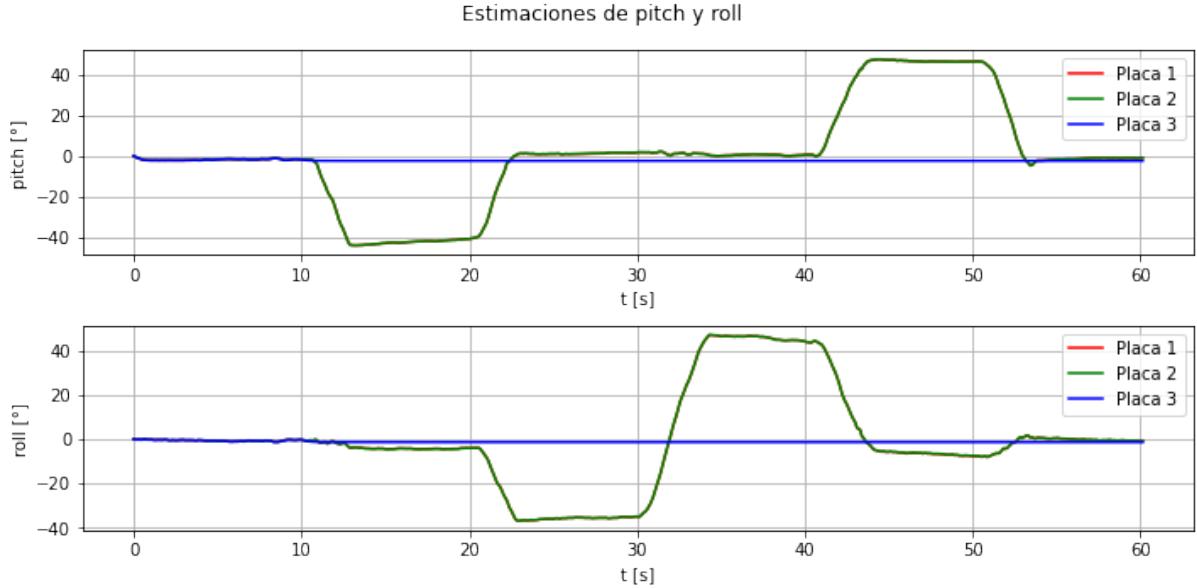


Figura 80: Se muestran los resultados de las estimaciones de pitch y roll para las 3 placas. Los gráficos se encuentran prácticamente superpuestos todo el tiempo, ya que no hubo fallas en la ejecución.

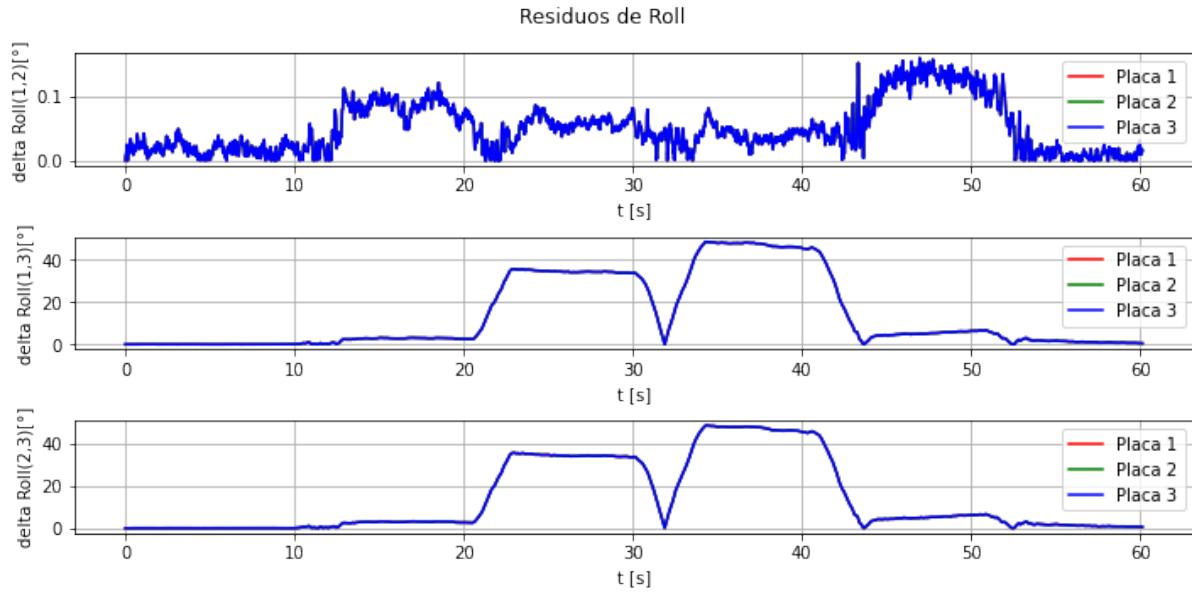


Figura 82: Se muestran los residuos de roll calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\phi[n]$, el segundo a $r_{1,3}^\phi[n]$ y el tercero a $r_{2,3}^\phi[n]$.

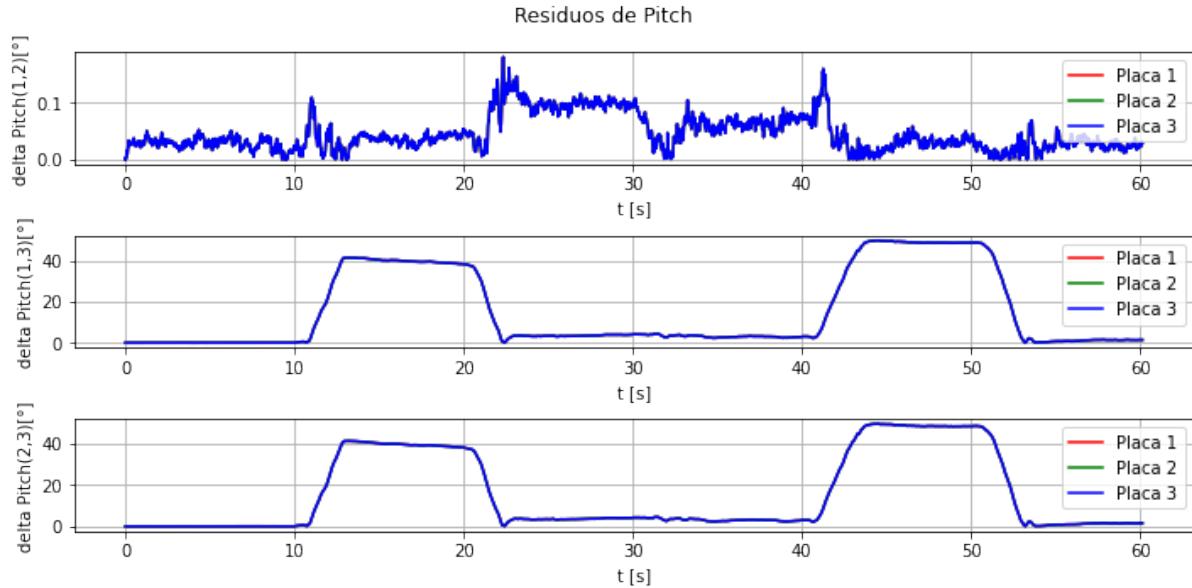


Figura 81: Se muestran los residuos de pitch calculados por cada placa, superpuestos. El primer gráfico corresponde a $r_{1,2}^\theta[n]$, el segundo a $r_{1,3}^\theta[n]$ y el tercero a $r_{2,3}^\theta[n]$.

Diseño Tolerante a Fallas de Hardware RE ACOMODAR EN OTRA SECCIÓN

Introducción al Análisis de Tolerancia a Fallas

En los últimos años se ha incrementado mucho la presencia de UAVs en espacio aéreo civil. Debido a esto, se plantea que los UAVs deberían presentar características que permitan un funcionamiento correcto, tolerante a fallas. Como consecuencias posibles, el hecho de volar en espacio aéreo civil puede llegar a causar daño físico a personas, si es que un vehículo presenta una falla y por ejemplo pierde el control. Otra de las posibles consecuencias tiene que ver con los costos que puede ocasionar una falla en una misión relacionada a una actividad laboral. El hecho de tener que repetir la misión puede traer mayores costos para la actividad en cuestión.

El objetivo del diseño tolerante a fallas consiste en mejorar la confianza (*Dependability*) del sistema, apuntando a que este pueda seguir ejecutando su función de manera correcta a pesar de la presencia de una cierta cantidad de fallas [6]. De esta última expresión se puede tomar una definición de lo que es un sistema tolerante a fallas.

Definición 1. *Sistema Tolerante a Fallas:* es aquel donde una falla no implica necesariamente un fracaso en el funcionamiento. Un sistema tolerante a fallas no es aquel donde no ocurren fallas, sino que más bien, se acepta que las fallas pueden ocurrir en el sistema, pero lo que se pretende es que el sistema pueda cumplir con su función de igual manera.

De manera de introducir la nomenclatura que se encuentra en la bibliografía [6], se definen los siguientes términos:

- Falla (*Fault*): es alguna condición anómala, no esperada.
- Error: ocurre cuando una falla se manifiesta y produce un comportamiento fuera de lo esperado en alguna parte del sistema.
- Fracaso (*Failure*): quiere decir que el sistema no puede cumplir con su función de manera adecuada.

Una de las formas de cuantificar la confianza es a través de la fiabilidad del sistema (*Reliability*). Esta se expresa en la ecuación (8), y se define como la probabilidad de que el sistema pueda cumplir su función de manera correcta en un intervalo de tiempo $[t_0; t]$, dado que en el instante inicial t_0 el sistema podía hacerlo.

$$R(t) = P(\text{funcionamiento correcto en } t | \text{funcionamiento correcto en } t_0) \quad (8)$$

Dado que en el intervalo $[t_0; t]$ puede o no ocurrir una falla, la probabilidad de que el sistema pueda cumplir su función en t puede expresarse como en la ecuación (9). Si no ocurre ninguna falla, luego el sistema podrá seguir cumpliendo su función en t . Además, si llegase a ocurrir una falla, pero el sistema tiene la capacidad de tolerarla, luego el sistema de igual manera podrá seguir cumpliendo su función en el instante t .

$$\begin{aligned} R(t) &= P(\text{no ocurrió una falla en } [t_0; t]) \\ &\quad + P(\text{funcionamiento correcto en } t | \text{ocurrió una falla en } [t_0; t]) P(\text{ocurrió una falla en } [t_0; t]) \end{aligned} \quad (9)$$

En el caso en el que se tuviera un sistema que no comprende ningún mecanismo de tolerancia a fallas, luego la fiabilidad sería exactamente igual a la probabilidad de que no ocurra una falla, ya que la ocurrencia de una falla causaría un funcionamiento incorrecto. Esto no necesariamente representa un problema. Si el sistema en cuestión es tal que puede demostrarse que la probabilidad de que no ocurra una falla es lo suficientemente alta, luego no se requeriría el uso de técnicas de tolerancia a fallas.

En un sistema donde no hay tolerancia a fallas, la fiabilidad quedaría definida como en la ecuación (10) y la única manera de mejorarla sería incrementando la probabilidad de que no ocurra ninguna falla en el intervalo $[t_0; t]$.

$$R(t) = P(\text{no ocurrió una falla en } [t_0; t]) \quad (10)$$

La manera de hacer esto puede ser por ejemplo, utilizando componentes o módulos de muy buena calidad, lo suficientemente confiables como para cumplir con los requerimientos de fiabilidad [6]. Sin embargo, esto puede ser muy costoso, pensando en que un sistema puede tener una enorme cantidad de posibles fallas. No solo eso, sino que esto dificulta la etapa de diseño de un sistema, ya que cualquier error de diseño que no se haya tenido en cuenta puede llegar a causar una falla y por ende un fracaso del sistema. Por el contrario, la tolerancia a fallas plantea permitir que las fallas existan, pero aplicando técnicas para tolerarlas.

Volviendo a la ecuación (9), la probabilidad de que el sistema funcione correctamente a pesar de la falla, está pesada por la probabilidad de ocurrencia de dicha falla. A partir de esto se desprende que aplicar técnicas de tolerancia a fallas para cada una de las posibles fallas puede resultar exhaustivo, principalmente porque deberían conocerse todas las fallas posibles, además de ser algo costoso. Lo que se propone es considerar solo aquellas fallas cuya criticidad es alta.

A modo de ejemplo, una **falla en un sensor de la computadora de vuelo puede generar una lectura incorrecta**. En consecuencia, esto decantará en un **error, es decir, en un cálculo de la ley de control incorrecto**. Finalmente, este error puede llevar al **fracaso de la misión, por ejemplo si el vehículo no es capaz de seguir una trayectoria dada en tiempo y forma**. Esto da a entender que una falla en un sensor es crítica y que por ende requiere la aplicación de técnicas de tolerancia a fallas.

Aquí se habla de falla en un sensor como algo general. Un sensor podría fallar de muchas maneras y debido a muchas razones. Por ejemplo, puede dejar de funcionar por un defecto propio del componente, puede entregar lecturas erróneas debido a interferencias electromagnéticas, por efectos de la temperatura, falta de calibración, etc. Cada uno de estos requeriría la aplicación de un mecanismo tolerante a fallas.

Causales de Fallas de Hardware y Modelo de Fallas Arbitrarias

Uno de los métodos para aplicar mecanismos de tolerancia a fallas consiste en hacer un análisis de los posibles modos de falla. Un ejemplo es el del análisis *Failure Modes and Effects Analysis* (FMEA). Este consiste en realizar un análisis exhaustivo de los posibles modos de falla más probables y sus posibles efectos en el sistema. En función de este análisis, se toman medidas para tolerar las fallas más críticas. El objetivo de este tipo de análisis suele ser demostrar ante alguna autoridad certificante, que la confianza del sistema se mantiene por encima de cierto valor. Este tipo de análisis suele consumir mucho tiempo y esfuerzo, lo que se traduce en un mayor costo del desarrollo [22].

Una forma de aliviar esta tarea es la de considerar un modelo de falla de hardware más conservador, donde se asume que una falla de hardware consiste en que esta presente un comportamiento anómalo arbitrario, es decir, de cualquier tipo. A este tipo de comportamiento se lo denomina falla bizantina o *Byzantine Fault* en inglés y básicamente consiste en asumir que el elemento que manifiesta la falla presenta un comportamiento arbitrario. Por ejemplo, un sensor puede dejar de funcionar repentinamente y no dar más respuesta, puede dejar de enviar respuesta por un período de tiempo y luego volver a funcionar, podría también enviar datos a un microcontrolador pero que esos datos sean incoherentes, etc. El modelo de falla bizantina no asume modos de falla, sino que el comportamiento es arbitrario [25][15][22]. Se define un sistema tolerante a este tipo de fallas.

Definición 2. Sistema Byzantine Resilient: es aquel capaz de tolerar una cierta cantidad de fallas arbitrarias a la vez.

Dado que no se asume un modo de falla del hardware, no se requiere un análisis tan exhaustivo como el mencionado FMEA. Considerando el costo y esfuerzo que lleva realizar un análisis de modos de fallas,

el hecho de poder contar con un sistema con las características que aquí se mencionan resulta atractivo para alivianar el trabajo relacionado a la validación del sistema tolerante a fallas en cuestión.

A priori, puede parecer que desarrollar un sistema tolerante a fallas arbitrarias representa un trabajo sumamente complejo. La manera de implementar un sistema tolerante a fallas bizantinas es a través del uso de redundancias. Este resultado se toma a partir de un problema teórico denominado *The Byzantine Generals Problem* [26], el cual se presentará más adelante.

Tolerancia a Fallas a Partir de Redundancias

La principal técnica de tolerancia a fallas es el uso de redundancias [6][71][22][62]. Esto quiere decir, que se replica el hardware en el sistema y cada réplica realiza la misma tarea en paralelo. De esta forma, si una de las réplicas presenta una falla (arbitraria por ejemplo), esta puede detectarse a partir de la comparación con las demás réplicas, o incluso pasar desapercibida. Utilizando la nomenclatura definida en la sección 6.3.4, que una falla pase desapercibida quiere decir que no se manifiesta como un error, sino que esta es contenida. A continuación se presentan algunas arquitecturas redundantes para la tolerancia a fallas.

Redundancia Doble

Una arquitectura simple es la redundancia doble. En este tipo de sistemas, dos nodos de un sistema funcionan en paralelo y comparan sus resultados. La comparación permite detectar si los resultados difieren entre sí, lo que se traduce en que ocurrió un error.

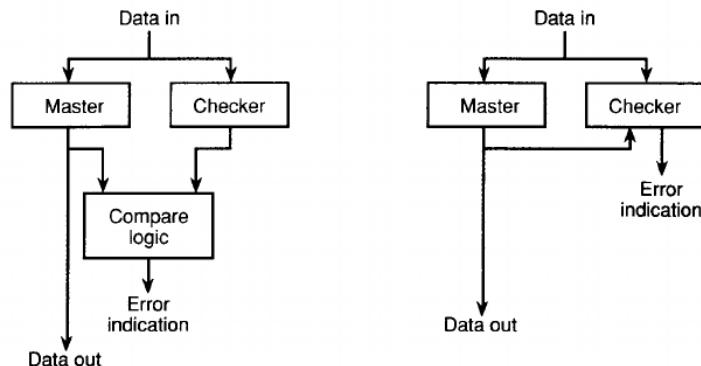


Figura 83: En la figura de la izquierda, dos sistemas ejecutan las mismas operaciones, mientras que otro sistema externo se encarga de comparar las salidas de ambos para detectar errores. En la figura de la derecha, el bloque comparador se encuentra integrado en el sistema *checker*. La imagen fue extraída de [6].

Este tipo de arquitectura permite detectar si ocurrió un error, pero no permite identificar de qué nodo proviene el error. En la figura 83 se muestran dos configuraciones. La configuración de la derecha puede ser implementada a través de dos CPUs totalmente independientes (a veces denominada *Loosely-Synchronized Dual Processor Architecture*) o a través del uso de un procesador de dos núcleos, donde uno sería el *Master* y otro el *checker*[23]. En esta última, ambos se encuentran sincronizados por estar en el mismo chip y compartir fuente de clock. En la figura 84 se muestra un esquema de ambos casos.

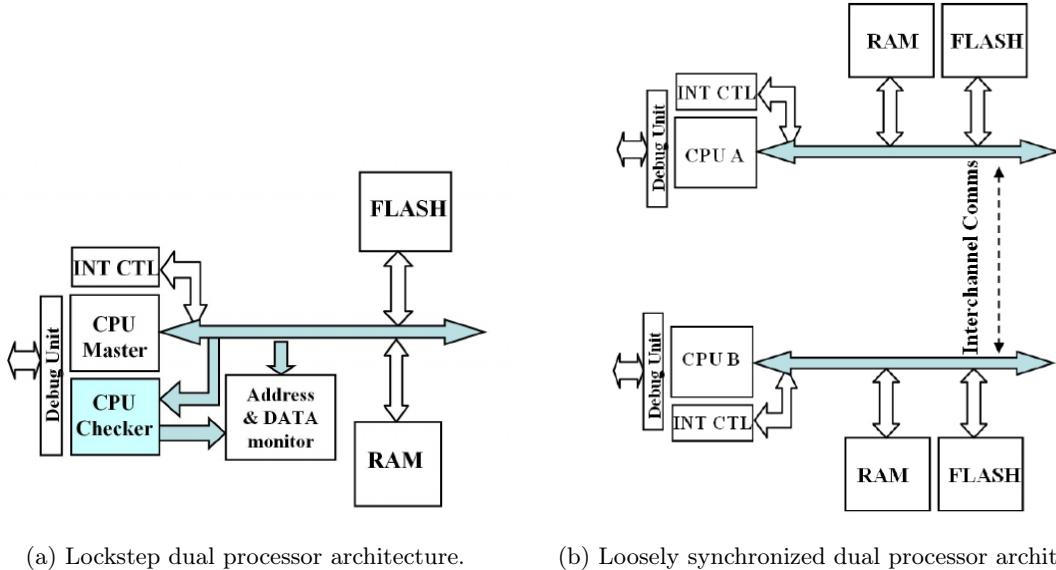


Figura 84: Se muestran dos casos para un sistema con redundancia doble. La imagen fue extraida de [23].

Debido a que no se puede saber cuál de las dos CPUs cometió el error, esta arquitectura plantea que en el caso en el que la comparación entre ambas CPUs genere una discrepancia en los resultados, cada una de ellas deben ejecutar un algoritmo interno, para detectar si ellas fueron las que cometieron el error o no. En [17] y en [72] se pueden encontrar proyectos de redundancia doble para UAVs.

Redundancia Triple

Esta arquitectura puede encontrarse en la literatura con el nombre *Triple Modular Redundant (TMR) Architecture* [23][6][71][24]. Esta arquitectura consiste en utilizar tres computadoras en paralelo, las cuales computan los mismos resultados. Luego, se comparan los resultados. Se asume que solamente 1 de las 3 presentará una falla a la vez. En dicho caso, los resultados de dos computadoras serán iguales y la de la tercera será distinto, por lo que solamente se descarta el resultado erróneo. En la figura 85 se muestra un diagrama con la arquitectura TMR. Una diferencia de esta arquitectura respecto de la doble redundancia, es el hecho de que puede detectarse cuál de las computadoras falló y además, no es necesario que todas las computadoras ejecuten una rutina para verificar si cometieron el error o no. Esto resulta especialmente útil en sistemas de tiempo real, donde no puede detenerse el sistema para realizar una verificación interna. Esto se denomina *Fault Masking*.

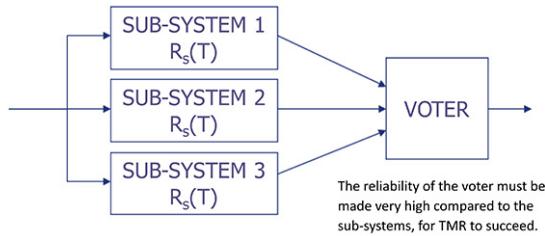


Figura 85: Arquitectura TMR. La imagen fue extraida de [73].

Como indica el texto de la imagen, una cuestión clave de esta arquitectura es el bloque denominado *VOTER*. Debido a que este bloque es el que determina cuál es el resultado correcto, se requiere que la

fiabilidad, $R(t)$, de este sea mucho mayor que la de cada computadora de vuelo. Esto se logra a través del uso de hardware más robusto, lo que resulta en que el bloque *VOTER* sea más costoso que cada computadora de vuelo. Por ejemplo, cada computadora de vuelo puede comprender un microcontrolador COTS, mientras que el bloque voter puede estar implementado con un ASIC específico para esa aplicación [15]. Si bien este bloque tiene una fiabilidad mucho mayor, siempre existe la probabilidad de que ocurra un error en este. En cuyo caso, el error puede decantar en un fracaso, por ejemplo si el *VOTER* elige como resultado correcto, aquel que realmente no lo era.

Definición 3. Single-Point Failure: si la arquitectura del sistema es tal que una parte del sistema X fracasa en cumplir su trabajo dentro del sistema, luego el sistema completo fracasará en cumplir su función. En dicho caso, X es un punto único de falla.

Una forma de combatir esto es replicar los bloques que realizan la votación [6][24]. De esta manera, también pueden enmascararse errores de los bloques que realizan la votación. La arquitectura sería como la que se muestra en la figura 86.

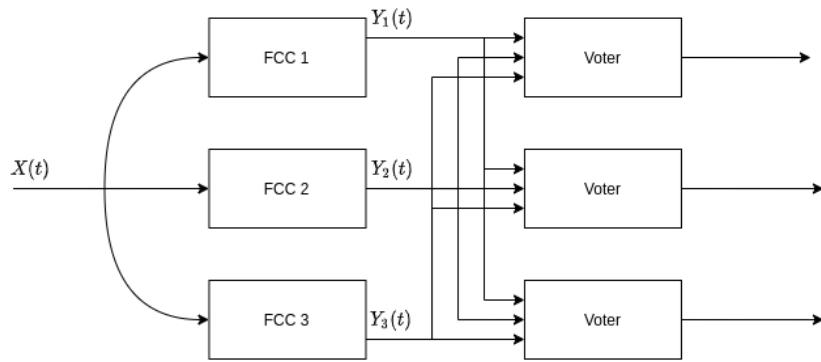


Figura 86: Arquitectura TMR con redundancia en los elementos votantes.

Los tres elementos *Voter* reciben las mismas entradas y en el caso de que ninguno de los *voters* cometa un error, dado que las entradas de los *Voters* son exactamente iguales, luego los tres decidirán por el mismo resultado como el valor correcto.

Esta arquitectura es más compleja que las anteriores, ya que requiere una gran cantidad de nodos, 3 FCCs + 3 bloques votantes, dando un total de 6. Además, pensando en que se argumentó que los votantes generalmente son más confiables que las FCCs, la triplicación del bloque *Voter* encarece mucho al UAV.

Como medida para evitar esto último, los bloques votantes pueden integrarse dentro de cada una de las FCC. Esto quiere decir, que en lugar de tener 3 bloques votantes, las mismas FCC sean las encargadas de realizar la votación. En el artículo [15] se propone que los microcontroladores automotivos ofrecen las interfaces necesarias para implementar una red redundante para tolerar fallas. En el artículo [14], los mismos autores presentan resultados para una arquitectura con redundancia cuádruple, donde los mismos microcontroladores de cada FCC son los encargados de realizar la votación. Para el caso de una arquitectura de redundancia triple, puede diagramarse como en la figura 87.

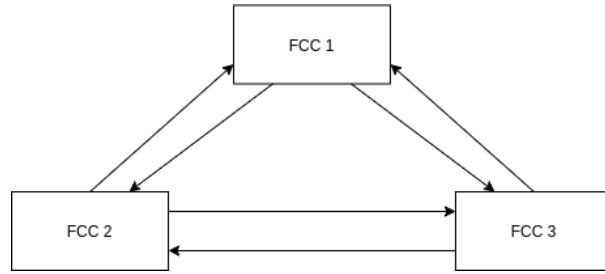


Figura 87: Arquitectura de redundancia triple, donde los bloques votantes son las mismas FCCs. Los votantes se encuentran integrados dentro de cada FCC.

Redundancia Cuádruple: *The Byzantine Generals Problem*

En las secciones anteriores se habla de un modelo de falla de hardware arbitraria, denominada falla bizantina. El nombre proviene de un problema denominado *The Byzantine Generals Problem*, formalizado en [26]. Este paper plantea un escenario que sirve como base para el análisis de fallas arbitrarias. En esta sección, se presenta brevemente el problema y su relación con la tolerancia a fallas. El análisis completo puede encontrarse en el trabajo original [26]. Otros trabajos que tratan el mismo problema son [27] y [28]. Este último, presenta el diseño de una computadora de vuelo tolerante a fallas que utiliza los resultados del *Byzantine Generals Problem* para realizar distintas tareas de redundancia.

Presentación del Problema

El escenario que se plantea es el siguiente: un grupo de generales, cada uno liderando su respectivo ejército, se encuentran rodeando una ciudad enemiga. Todos los generales deben ponerse de acuerdo, respecto de si la mejor decisión es atacar la ciudad o retirarse. Independientemente de cuál sea la decisión, todos deben tomar la misma decisión.

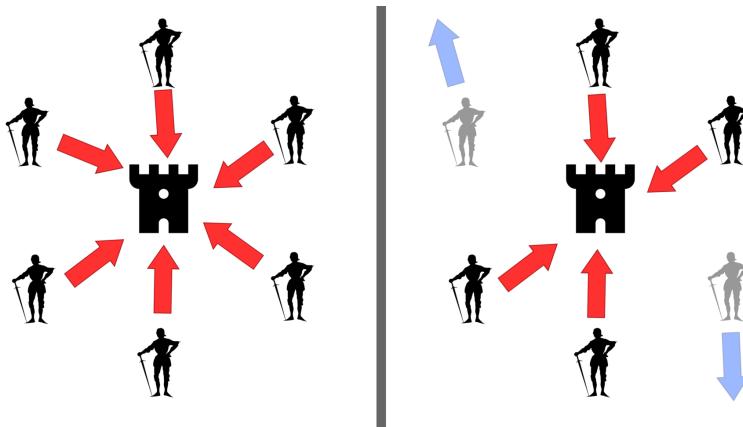


Figura 88: La situación que se presenta, donde los generales deben tomar una decisión común. La figura de la derecha muestra la situación donde algunos generales atacan mientras que los generales traidores no lo hacen. La imagen se extrajo de [74].

Debido a que los generales se encuentran alejados unos de otros, estos solo pueden comunicarse con mensajes uno a uno, por ejemplo con un soldado que lleve un mensaje a caballo, desde un ejército a otro ejército. Por ejemplo, si el general 1 decide que lo mejor es atacar, este enviará un mensaje a cada uno de los otros generales para informarles que su voto es por atacar la ciudad.

Además, el problema plantea la posibilidad de que algunos de los generales sean traidores. Esto quiere decir que ellos pueden actuar de manera independiente a la decisión común.

Cada general vota por atacar o por retirarse, y la decisión final será la que tenga más votos. **Esto quiere decir que cada general debe conocer la opinión de los demás generales, para así poder coincidir en el resultado final, es decir, atacar o retirarse.** El problema, es que los generales traidores pueden mentir o enviar información diferente a cada general. Esto último se refiere a que un traidor puede decirle a un general que su opinión es “atacar” y a otro general decirle que su opinión es “retirarse”. **Esto último implica que todos los generales deben disponer de la misma información para así poder tomar la misma decisión y que los traidores no perjudiquen el consenso al que deben llegar los generales.** Por ejemplo, si se tienen 3 generales y los generales 1 y 2 reciben los votos:

$$\text{General 1} = \begin{bmatrix} \text{Atacar} \\ \text{Retirarse} \\ \text{Atacar} \end{bmatrix}$$

$$\text{General 2} = \begin{bmatrix} \text{Atacar} \\ \text{Retirarse} \\ \text{Retirarse} \end{bmatrix}$$

Esto llevará a que el General 1 ataque mientras que el General 2 se retire. El error fue causado por la presencia del traidor, el General 3.

Solución al Problema

El paper plantea una solución para este problema, pero que solamente es válida en el caso en el que se tienen m traidores y al menos $3m + 1$ generales en total. En la figura 89 se muestra un caso para 4 generales y 1 traidor. El General 1 es el traidor y le envía información diferente a cada general.

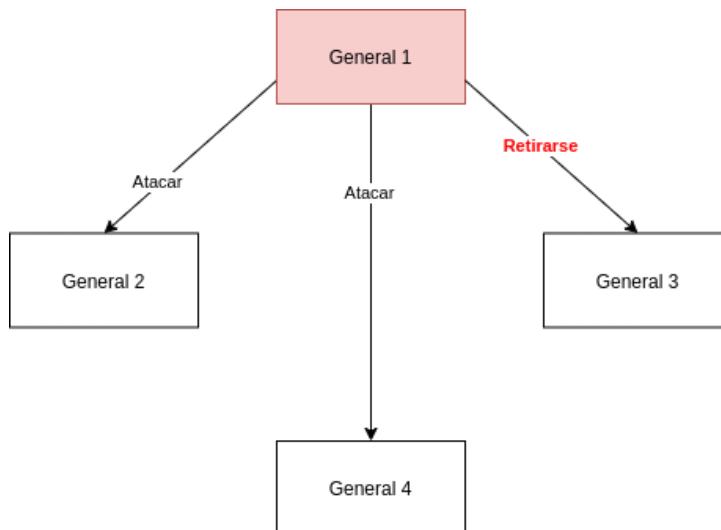


Figura 89: El general 1 es un traidor y le envía información conflictiva a los demás generales.

Como fue mencionado, para llegar a una decisión común, todos los generales deben conocer la opinión de los demás. El problema en este caso es que el General 1 envió una información diferente a sus pares. Para resolver esto, el algoritmo plantea realizar un segundo intercambio de mensajes como el de la figura 90.

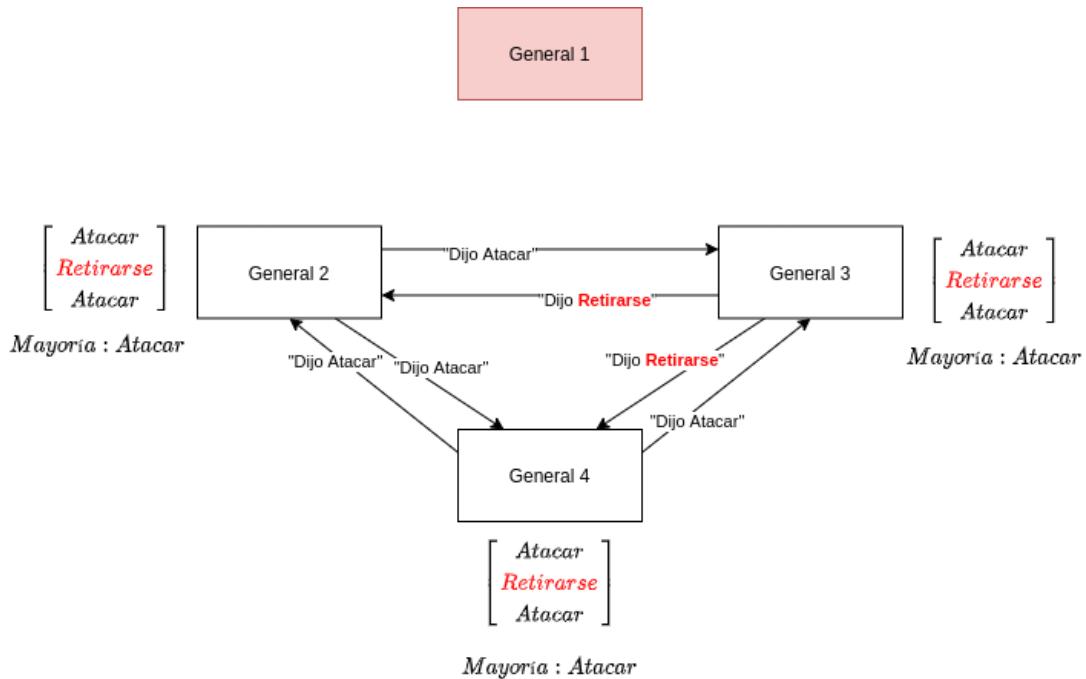


Figura 90: Se produce un intercambio entre los demás generales, para ponerse de acuerdo respecto de si el General 1 dijo “Atacar” o “Retirarse”.

Al lado de cada General, se muestra un vector que contiene los mensajes informados por los otros Generales, respecto del voto del General 1. Lo que se muestra es que en este caso, los Generales leales logran ponerse de acuerdo en que el General 1 dijo “Atacar”, es decir, llegan a un consenso. Para continuar con el algoritmo, se debe repetir el mismo procedimiento de intercambio de mensajes para los otros tres generales. Al finalizar todos los intercambios de mensajes, los Generales leales tendrán la misma información respecto a los votos de sus pares y llegarán a la misma decisión final.

Relación del Problema con la Tolerancia a Fallas

Si bien el análisis del problema se plantea como un juego, la motivación surge de realizar un análisis de tolerancia a fallas a partir de redundancias. En [28], los mismos autores de *The Byzantine Generals Problem* presentan un trabajo de diseño y análisis de una computadora de vuelo tolerante a fallas. Este es anterior a la formalización del problema, pero menciona que la necesidad del consenso entre cada nodo de la red redundante, es un requerimiento para aplicar los mecanismos de tolerancia a fallas correctamente.

Se traza un paralelismo entre los generales que deben llegar a un consenso con una serie de computadoras interconectadas, cuyo objetivo es también generar consenso respecto de alguna variable.

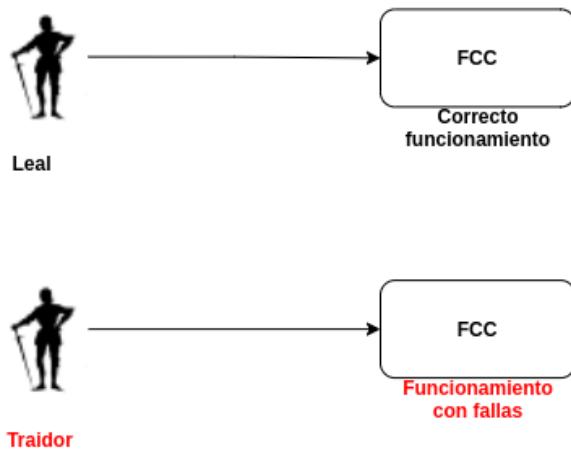


Figura 91: En el problema, un general leal representa un nodo, en este caso una computadora de vuelo, que funciona correctamente. Un General traidor es equivalente a una computadora de vuelo que presenta fallas.

Los generales traidores representan a las computadoras de vuelo que presentan fallas. En [28] se presenta un ejemplo de la aplicación del algoritmo de *The Byzantine Generals Problem* para lograr sincronizar a los nodos. A continuación, se analiza brevemente este problema, con motivo de demostrar su importancia en los sistemas redundantes tolerantes a fallas.

7. Conclusiones

8. Agradecimientos

COMPLETAR

Apéndices

Apéndice A: Circuito Esquemático

COMPLETAR

Apéndice B: PCB Final

COMPLETAR

Referencias

- [1] Richard PG Collinson. *Introduction to avionics systems*. Springer Nature, 2023.
- [2] Stuart M Adams y Carol J Friedland. «A survey of unmanned aerial vehicle (UAV) usage for imagery collection in disaster research and management». En: *9th international workshop on remote sensing for disaster response*. Vol. 8. 2011, págs. 1-8.
- [3] Tommaso Francesco Villa y col. «An overview of small unmanned aerial vehicles for air quality measurements: Present applications and future prospectives». En: *Sensors* 16.7 (2016), pág. 1072.
- [4] AJS McGonigle y col. «Unmanned aerial vehicle measurements of volcanic carbon dioxide fluxes». En: *Geophysical research letters* 35.6 (2008).
- [5] Luis F Luque-Vega y col. «Power line inspection via an unmanned aerial system based on the quadrotor helicopter». En: *MELECON 2014-2014 17th IEEE Mediterranean electrotechnical conference*. IEEE. 2014, págs. 393-397.
- [6] Victor P. Nelson. «Fault-tolerant computing: Fundamental concepts». En: *Computer* 23.7 (1990), págs. 19-25.
- [7] *Fly-by-Wire Systems Enable Safer, More Efficient Flight / NASA Spinoff*. URL: https://spinoff.nasa.gov/Spinoff2011/t_5.html.
- [8] Ying C Yeh. «Triple-triple redundant 777 primary flight computer». En: *1996 IEEE Aerospace Applications Conference. Proceedings*. Vol. 1. IEEE. 1996, págs. 293-307.
- [9] Hermann Kopetz. *Real-Time systems*. Springer Science+Business Media, ene. de 2011. DOI: 10.1007/978-1-4419-8237-7. URL: <https://doi.org/10.1007/978-1-4419-8237-7>.
- [10] Xunying Zhang y Xiaodong Zhao. «Architecture design of distributed redundant flight control computer based on time-triggered buses for UAVs». En: *IEEE Sensors Journal* 21.3 (2020), págs. 3944-3954.
- [11] Embention. *Veronte Autopilot 4x - Products Veronte Embention*. Oct. de 2023. URL: <https://www.embention.com/product/veronte-autopilot-4x/>.
- [12] *VECTOR-600 -Autopilot for UAV / UAV Navigation*. URL: <https://www.uavnavigation.com/products/autopilots/vector-600>.
- [13] www.micropilot.com. *MicroPilot - World leader in professional UAV autopilots / Product Page - MP2128 3x Triple Redundant Autopilots*. URL: <https://www.micropilot.com/products-mp21283x.htm>.
- [14] Sebastian Hiergeist y Georg Seifert. «Implementation of a SPI based redundancy network for SoC based UAV FCCs and achieving synchronization». En: *2018 IEEE/AIAA 37th Digital Avionics Systems Conference (DASC)*. IEEE. 2018, págs. 1-10.
- [15] Sebastian Hiergeist y Georg Seifert. «Internal redundancy in future UAV FCCs and the challenge of synchronization». En: *2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)*. IEEE. 2017, págs. 1-9.
- [16] *MPC5744P FlexRay Interface in Pictures*. AN12233. Rev. 0. NXP Semiconductors. Mayo de 2021.
- [17] Xiaolin Zhang, Haisheng Li y Dandan Yuan. «Dual redundant flight control system design for microminiature UAV». En: *2015 2nd International Conference on Electrical, Computer Engineering and Electronics*. Atlantis Press. 2015, págs. 785-791.
- [18] Junhua Chen, Dong Cao y Xunhong Lv. «Design of FlexRay-based communication on triplex redundancy flight control computer». En: *2015 Chinese Automation Congress (CAC)*. IEEE. 2015, págs. 1969-1974.
- [19] Jun An Wang y Zhen Shui Li. «Development of flight control system Using embedded computer PC-104». En: *26th International Congress of the Aeronautical Sciences*. 2008, págs. 1-5.
- [20] M.E. Fernando Fidencio Solano Pérez. «Development of a Redundancy System for Autopilots». Tesis de mtría. Santiago de Querétaro, México, 2019.

- [21] Dronecode Foundation. *Homepage - PIXHawk*. Jun. de 2023. URL: <https://pixhawk.org/>.
- [22] Jaynarayan H Lala y Richard E Harper. «Architectural principles for safety-critical real-time applications». En: *Proceedings of the IEEE* 82.1 (1994), págs. 25-40.
- [23] Massimo Baleani y col. «Fault-tolerant platforms for automotive safety-critical applications». En: *Proceedings of the 2003 international conference on Compilers, architecture and synthesis for embedded systems*. 2003, págs. 170-177.
- [24] Robert E Lyons y Wouter Vanderkulk. «The use of triple-modular redundancy to improve computer reliability». En: *IBM journal of research and development* 6.2 (1962), págs. 200-209.
- [25] Edo Roth y Andreas Haeberlen. «Do Not Overpay for Fault Tolerance!» En: *2021 IEEE 27th Real-Time and Embedded Technology and Applications Symposium (RTAS)*. IEEE. 2021, págs. 374-386.
- [26] Leslie Lamport, Robert Shostak y Marshall Pease. «The Byzantine generals problem». En: *Concurrency: the works of leslie lamport*. 2019, págs. 203-226.
- [27] Marshall Pease, Robert Shostak y Leslie Lamport. «Reaching agreement in the presence of faults». En: *Journal of the ACM (JACM)* 27.2 (1980), págs. 228-234.
- [28] John H Wensley y col. «SIFT: Design and analysis of a fault-tolerant computer for aircraft control». En: *Proceedings of the IEEE* 66.10 (1978), págs. 1240-1255.
- [29] CAN Specification. «Bosch». En: *Robert Bosch GmbH, Postfach* 50 (1991), pág. 75.
- [30] *Introduction to the Controller Area Network (CAN)*. SLOA101B. Rev. B. Texas Instruments. Mayo de 2016.
- [31] *Migration guide from STM32F7 Series to STMH74x/75x, STM32H72x/73x and STMH7A3/7Bx devices*. STMicroelectronics, ago. de 2022.
- [32] *Product Longevity - STMicroelectronics*. URL: https://www.st.com/content/st_com/en/about/quality-and-reliability/product-longevity.html#10-year-longevity.
- [33] *ICM-42688-P / TDK InvenSense*. Oct. de 2023. URL: <https://invensense.tdk.com/products/motion-tracking/6-axis/icm-42688-p/>.
- [34] Krystian Borodacz, Cezary Szczepański y Stanisław Popowski. «Review and selection of commercially available IMU for a short time inertial navigation». En: *Aircraft Engineering and Aerospace Technology* 94.1 (2022), págs. 45-59.
- [35] *How to measure absolute pressure using piezoresistive sensing elements*. AMSYS. Jul. de 2009.
- [36] Avnet. *MEMS pressure sensors*. URL: <https://www.avnet.com/wps/portal/abacus/solutions/technologies/sensors/pressure-sensors/core-technologies/mems/> (visitado 31-10-2023).
- [37] *Choosing the Right Pressure Sensor*. AN-201610-PL38-01. Infineon. 2016.
- [38] ST Microelectronics. *Product Longevity*. URL: https://www.st.com/content/st_com/en/about/quality-and-reliability/product-longevity.html#10-year-longevity (visitado 11-05-2023).
- [39] *Road vehicles — Controller area network (CAN) — Part 4: Time-triggered communication*. Standard. International Organization for Standardization, ago. de 2004.
- [40] *ARM based Cortex M7 32b MCU+FPU, 462DMIPS, up to 1MB Flash/320+16+ 4KB RAM, USB OTG HS/FS, ethernet, 18 TIMs, 3 ADCs, 25 com ift, cam and LCD*. STMicroelectronics, feb. de 2016. URL: <https://www.st.com/en/microcontrollers-microprocessors/stm32f746zg.html#documentation>.
- [41] *SN65HVD23x 3.3-V CAN Bus Transceivers*. Abr. de 2018. URL: <https://www.ti.com/product/es-mx/SN65HVD230>.
- [42] *Connector Pin Assignment Recommendations*. CiA 106. CAN in Automation. Jun. de 2022.
- [43] *DroneCAN*. URL: <https://dronecan.github.io/> (visitado 11-09-2023).

- [44] *Automotive Compliant 1A Low Dropout Positive Regulator with Fixed and Adjustable Outputs.* Oct. de 2016. URL: <https://www.diodes.com/assets/Datasheets/ZLD01117Q.pdf>.
- [45] *Basics of Low-Dropout (LDO) Regulator ICs.* TOSHIBA. Mar. de 2021.
- [46] *Technical Review of Low Dropout Voltage Regulator Operation and Performance.* Texas Instruments. Ago. de 1999.
- [47] *AN-1482 LDO Regulator Stability Using Ceramic Output Capacitors.* Texas Instruments. Abr. de 2013.
- [48] *Holybro PM02 V3 Power Module.* URL: https://docs.px4.io/main/en/power_module/holybro_pm02.html (visitado 13-09-2023).
- [49] Leonardo Garberoglio y col. «Diseño de un autopiloto para pequeños vehículos no tripulados». En: *Elektron* 3.1 (2019), págs. 29-38.
- [50] *Specification for Spektrum Remote Receiver Interfacing.* URL: <https://www.spektrumrc.com/ProdInfo/Files/Remote%20Receiver%20Interfacing%20Rev%20A.pdf>.
- [51] *Documentation – Arm Developer.* URL: <https://developer.arm.com/documentation/ddi0314/h/Serial-Wire-Debug-and-JTAG-Trace-Connector/About-the-SWD-and-JTAG-trace-connector?lang=en>.
- [52] *PCB Design Guidelines For ICM-40607x, ICM-40608, ICM-42xxx, ICM-43xxx and ICM-45xxx Products.* AN-000262. TDK Invensense. Ene. de 2021.
- [53] *Surface Mounting Guidelines for MEMS Sensors in a QFPN Package.* TN0019. ST. Mar. de 2020.
- [54] *Soldering Guidelines for MEMS Inertial Sensors.* APP 5604. Maxim Integrated. Mar. de 2013.
- [55] *MEMS Motion Handling and Assembly Guide.* AN-IVS-0002A-00. TDK. Oct. de 2013.
- [56] Henry W Ott. *Electromagnetic compatibility engineering.* John Wiley & Sons, 2011.
- [57] *Getting Started with STM32F7 Series MCU Hardware Development.* STMicroelectronics, feb. de 2017.
- [58] *PCB Layout Thermal Design Guide.* No .65AN002E Rev.001. ROHM Semiconductor. Jun. de 2022.
- [59] *AN-1028 Maximum Power Enhancement Techniques for Power Packages (Rev. B).* SNVA036B. Texas Instruments. Mayo de 2013.
- [60] *An empirical analysis of the impact of board layout on LDO thermal performance.* SLVAE85. Texas Instruments. Feb. de 2019.
- [61] Michael J Pont. *Patterns for time-triggered embedded systems.* TTE System, Ltd, 2008.
- [62] Hermann Kopetz y Günther Bauer. «The time-triggered architecture». En: *Proceedings of the IEEE* 91.1 (2003), págs. 112-126.
- [63] Hermann Kopetz. «The time-triggered model of computation». En: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE. 1998, págs. 168-177.
- [64] Emmanuelle Anceaume e Isabelle Puaut. «Performance evaluation of clock synchronization algorithms». Tesis doct. INRIA, 1998.
- [65] Sascha Einspieler y col. «High Accuracy Software-Based Clock Synchronization Over CAN». En: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* 52.7 (2022), págs. 4438-4446. DOI: 10.1109/TSMC.2021.3096597.
- [66] Gabriel Leen y Donal Heffernan. «TTCAN: a new time-triggered controller area network». En: *Microprocessors and Microsystems* 26.2 (2002), págs. 77-94.
- [67] Miro Samek. *Practical UML statecharts in C/C++: event-driven programming for embedded systems.* CRC Press, 2008.
- [68] *STM32F75xxx and STM32F74xxx advanced Arm®-based 32-bit MCUs.* STMicroelectronics, jun. de 2018. URL: https://www.st.com/resource/en/reference_manual/rm0385-stm32f75xxx-and-stm32f74xxx-advanced-armbased-32bit-mcus-stmicroelectronics.pdf.

- [69] *STM32F74xxx and STM32F75xxx device limitations*. STMicroelectronics, jun. de 2019. URL: https://www.st.com/resource/en/errata_sheet/es0290-stm32f74xxx-and-stm32f75xxx-device-limitations-stmicroelectronics.pdf.
- [70] Claudio Pose. *Desarrollo de algoritmos de navegación y control para un vehículo aéreo autónomo*. Tesis de Grado, Facultad de Ingeniería de la Universidad de Buenos Aires, 2014.
- [71] Vinod B Prasad. «Fault tolerant digital systems». En: *IEEE Potentials* 8.1 (1989), págs. 17-21.
- [72] Federico Fidencio Solano Pérez. «Development of a Redundancy System for Autopilots». 2019.
- [73] *Triple Modular Redundancy*. URL: <https://www.layerzero.com/Innovations/Industry-Firsts/Triple-Modular-Redundancy.html>.
- [74] Wikipedia contributors. «Byzantine fault». En: *Wikipedia* (jul. de 2023). URL: https://en.wikipedia.org/wiki/Byzantine_fault#.