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

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 caracter 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 giróscopos 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 incoporar 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 sismteas analizados utilizan distintas técnicas para implenetar 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 aviónica, 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 alivianar 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).

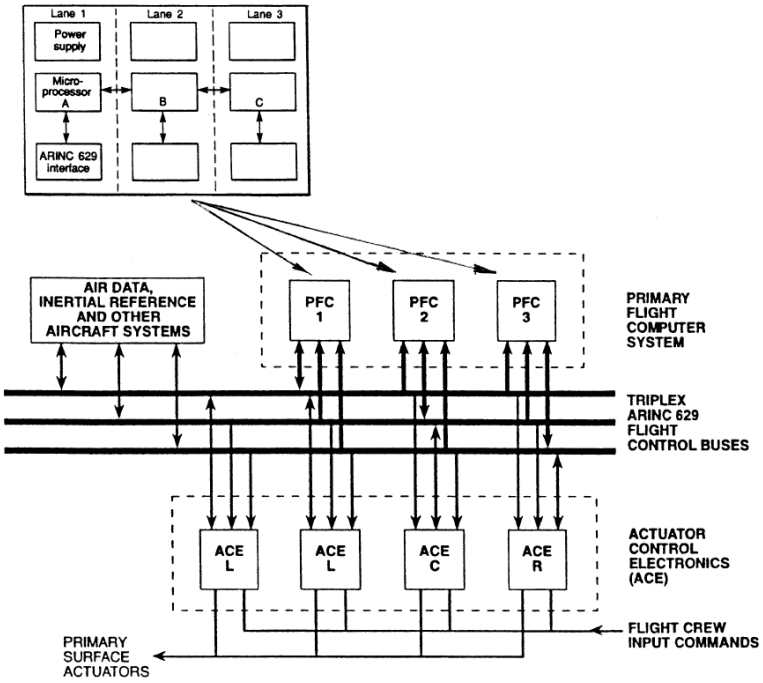


Figura 1: Arquitectura del sistema principal de vuelo, conformado por varias computadoras en una configuración redundante. La imagen se extrajo de [1].

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 expermiental, 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, actudaores y computadoras de vuelo.

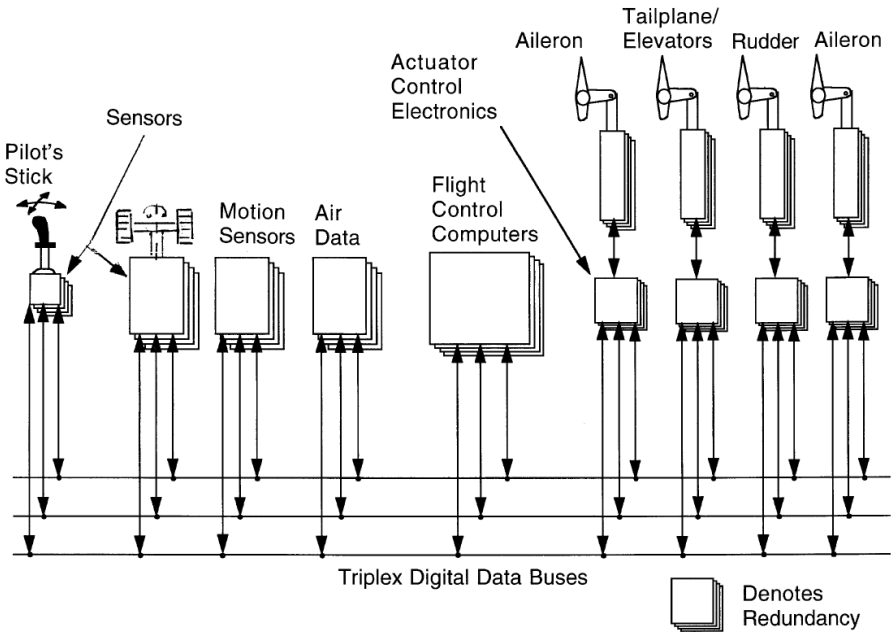


Figura 2: Se muestra de manera general las conexiones entre los distintos elementos del sistema de control de vuelo típico en aviones. La imagen se extrajo de [1].

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.

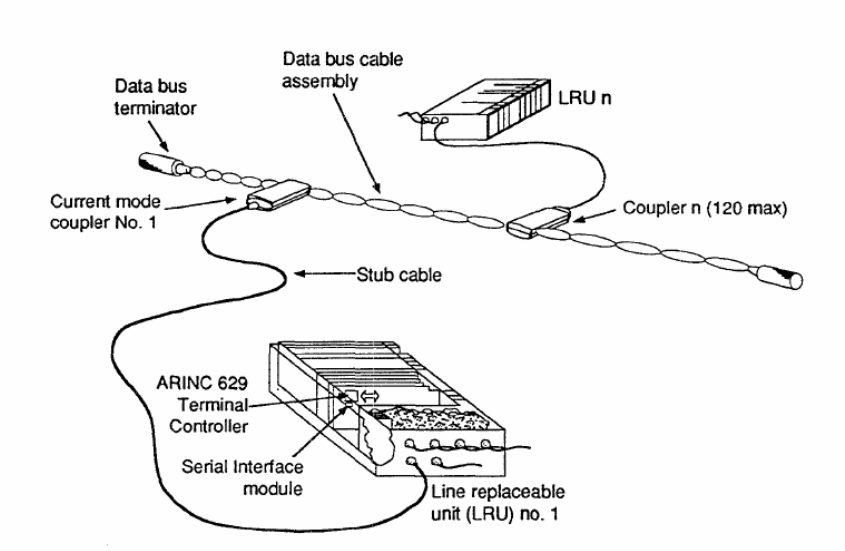


Figura 3: Se muestra un bus ARINC 629 con 2 nodos conectados. Este consiste en un par trenzado, con terminaciones en los extremos para evitar reflexiones. La conexión de los nodos al bus no es directa si no que se utilizan acopladores. La imagen se extrajo de [8].

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 respresentará 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.

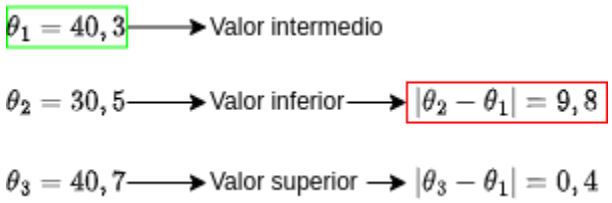


Figura 4: La comparación entre 3 sensores da como resultado que el sensor 2 presentó una falla. En consecuencia deberá tomarse una acción, por ejemplo ignorar al sensor en próximas mediciones o informar al piloto.

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.

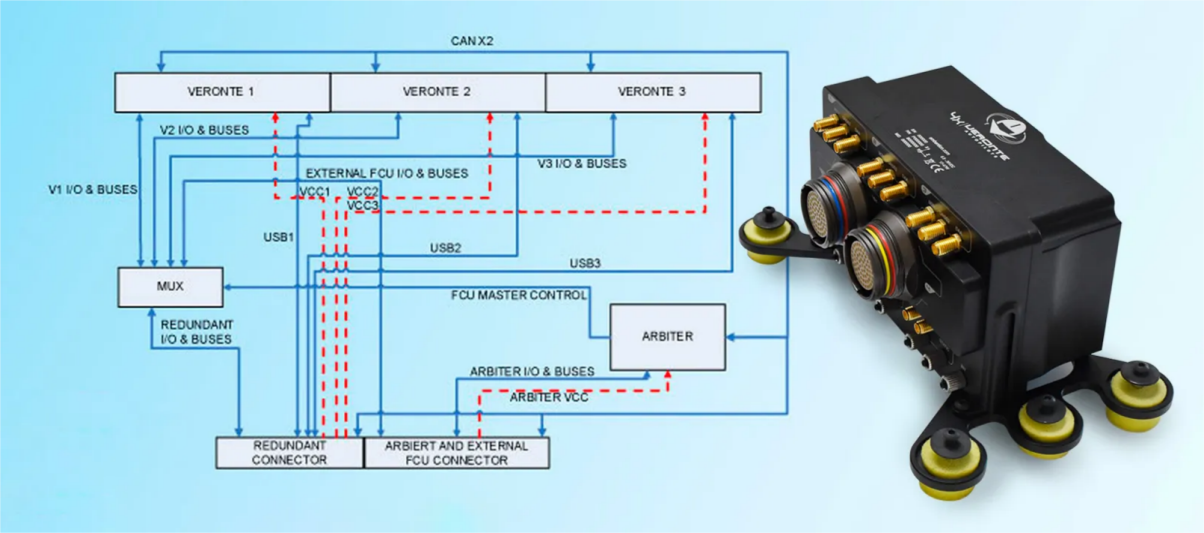


Figura 5: Diagrama en bloques del autopiloto Veronte de la empresa Embention. La imagen se extrajo de [11].

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.

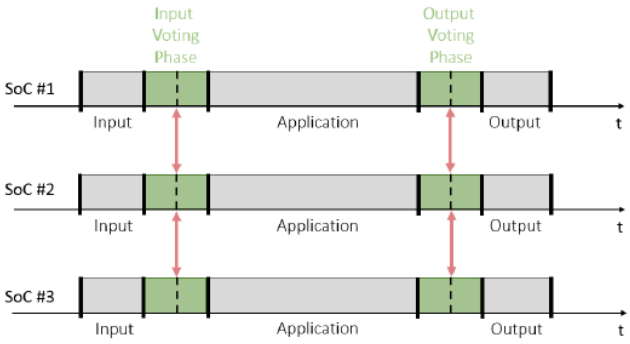


Figura 6: Imagen que demuestra la sincronización entre 3 microcontroladores que realizan las mismas tareas en paralelo. La imagen se extrajo de [15].

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.

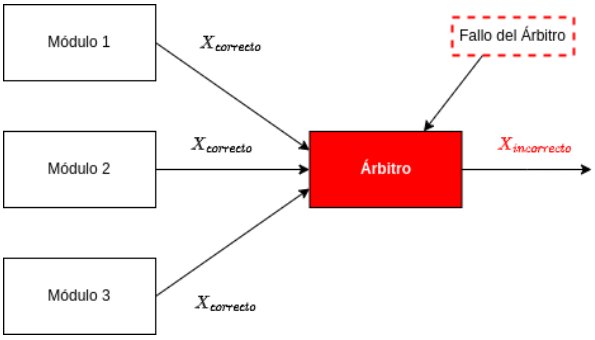


Figura 7: A pesar de que los 3 módulos redundantes funcionen correctamente, una falla del árbitro se traduce directamente en un error en el sistema. Esto lo vuelve un punto singular de falla.

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 alivianar 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.

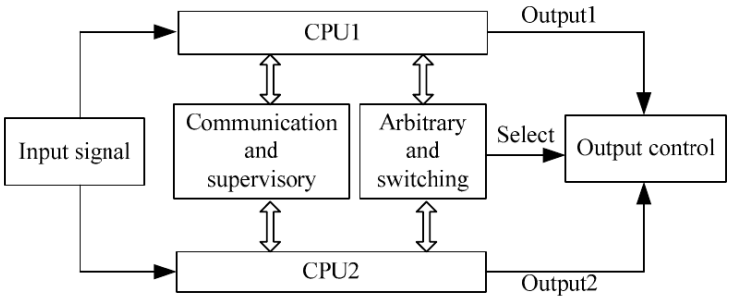


Figura 8: Diagrama en bloques del sistema de redundancia doble. Las CPU1 y CPU2 comparan sus resultados y envían sus salidas al bloque *Output control*. A través de un bloque árbitro se selecciona a cuál de las dos CPU será la que controle los actuadores. La imagen se extrajo de [17].

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 árbritro 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.

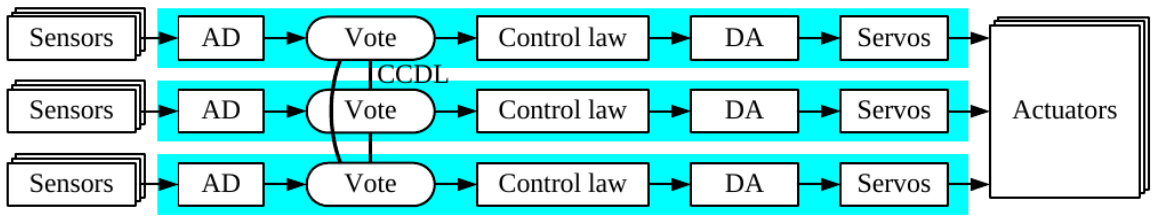


Figura 9: Diagrama en bloques del sistema redundante de [19]. Cada uno de los nodos tiene sus propios sensores, la única comunicación que se realiza a través del bus CCDL (*Cross-Communication Data Link*) es para realizar la comparación de resultados y la votación.

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 comptuadoras será la que controle los motores.

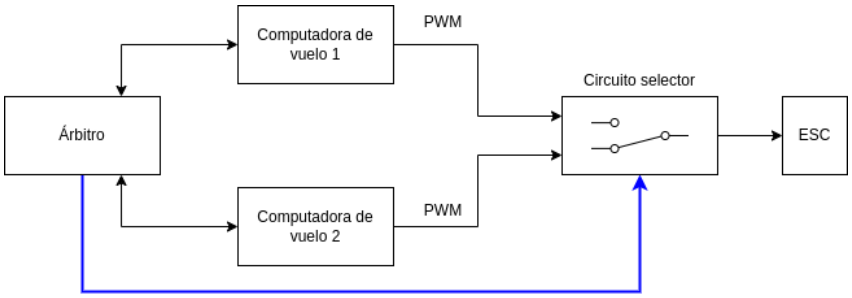


Figura 10: Arquitectura de la computadora de vuelo utilizada en [20]. El árbitro selecciona cuál de las dos señales PWM se utiliza para controlar el *Electronic Speed Controller* (ESC).

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 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 (9), 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.

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

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 (10). 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.

$$\begin{aligned} R(t) = {} & \mathtt{P}\,(\text{no ocurrio una falla en } [t_0; t]) \\ & + \mathtt{P}\,(\text{funcionamiento correcto en } t | \text{ocurrió una falla en } [t_0; t])\ \mathtt{P}\,(\text{ocurrió una falla en } [t_0; t]) \end{aligned} \tag{2}$$

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 (11) y la única manera de mejorarla sería incrementando la probabilidad de que no ocurra ninguna falla en el intervalo $[t_0; t]$.

$$R(t) = \mathtt{P}\,(\text{no ocurrio una falla en } [t_0; t]) \tag{3}$$

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 (10), 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.

Teniendo en consideración las consecuencias que puede traer el fracaso del sistema en cuestión, resulta adecuado tomar una actitud conservadora y adoptar técnicas de tolerancia a fallas, aceptando que estas pueden ocurrir.

4.2. Uso de Redundancias

Todos los trabajos y ejemplos presentados en la sección 3 comparten la característica de implementar la tolerancia a fallas utilizando varias copias del mismo elemento de hardware. Estas copias trabajan en paralelo y se comparan los resultados obtenidos por cada una de ellas. Las fallas se detectan cuando ocurre una diferencia en los resultados de las copias. Esta es la principal técnica de tolerancia a fallas [6][22][23][24] y es la que se utilizará en este trabajo. 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 6.3.5, 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.

4.2.1. 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.

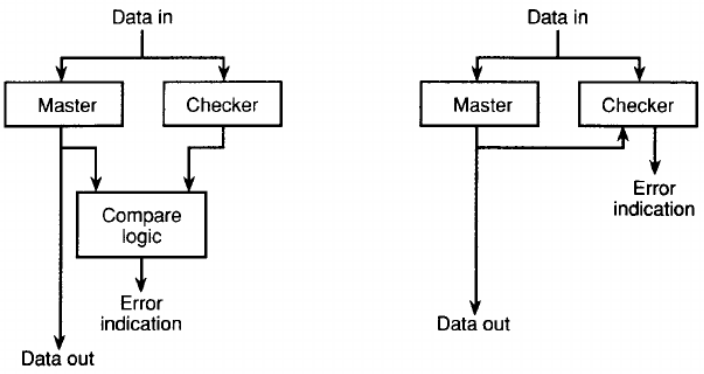
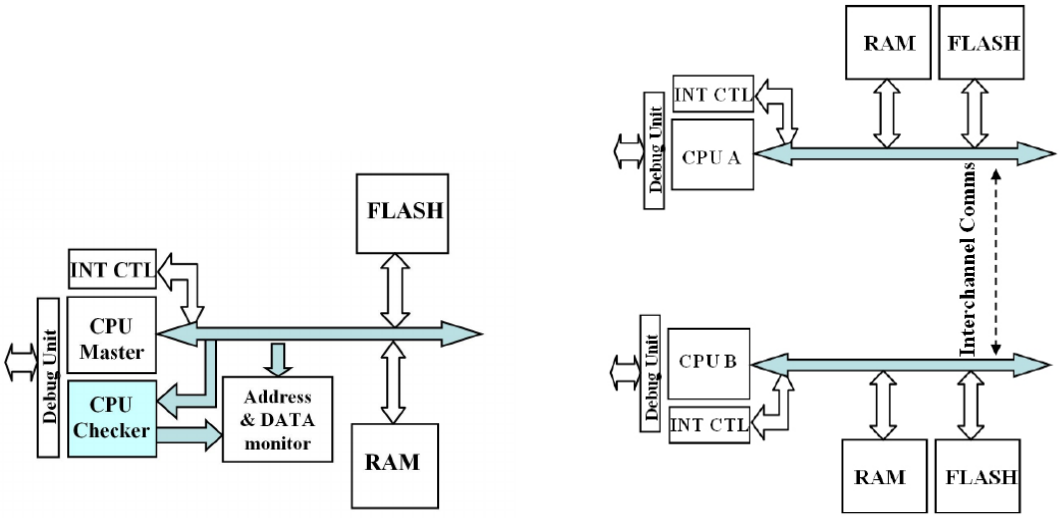


Figura 11: En la figura de la izquierda, dos sistemas ejecutan las mismas operaciones, mientras que otro sistema externo se encarga de comparar las salidas de ambos para detectar errores. En la figura de la derecha, el bloque comparador se encuentra integrado en el sistema *checker*. La imagen fue extraida 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 51 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*[25]. En esta última, ambos se encuentran sincronizados por estar en el mismo chip y compartir fuente de clock. En la figura 52 se muestra un esquema de ambos casos.



(a) Lockstep dual processor architecture.

(b) Loosely synchronized dual processor architecture.

Figura 12: Se muestran dos casos para un sistema con redundancia doble. La imagen fue extraida de [25].

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 [26] se pueden encontrar proyectos de redundancia doble para UAVs.

4.2.2. Redundancia Triple

Esta arquitectura puede encontrarse en la literatura con el nombre *Triple Modular Redundant (TMR) Architecture* [25][6][22][27]. 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 53 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*.

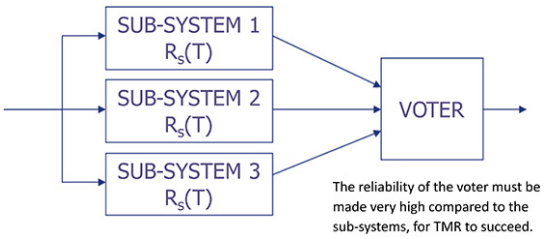


Figura 13: Arquitectura TMR. La imagen fue extraida de [28].

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 costos 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 2. ***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][27]. 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 54.

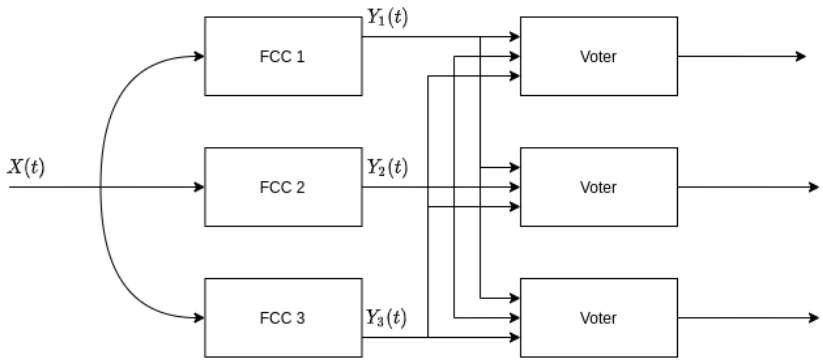


Figura 14: Arquitectura TMR con redundancia en los elementos votantes.

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 55.

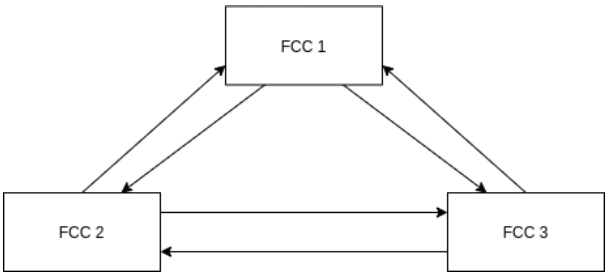


Figura 15: Arquitectura de redundancia trple, donde los bloques votantes son las mismas FCCs. Los votantes se encuentran integrados dentro de cada FCC.

4.2.3. Necesidad del Consenso entre Nodos

Como se mostró en la figura 21a, las computadoras de vuelo pueden comunicarse entre ellas para lograr una sincronización, por ejemplo compartiendo a sus pares un valor asociado a su propio clock interno. Cada FCC propone un valor distinto y estas buscarán ponerse de acuerdo en un valor único. Para que cada una de ellas llegue a la misma conclusión acerca del valor de clock correcto, si todas ellas ejecutan el mismo algoritmo y poseen los mismos valores de entrada, luego llegarán a la misma conclusión. En la figura 16 se muestra un caso en el que una de las computadoras de vuelo comparte valores de clock distintos a sus pares.

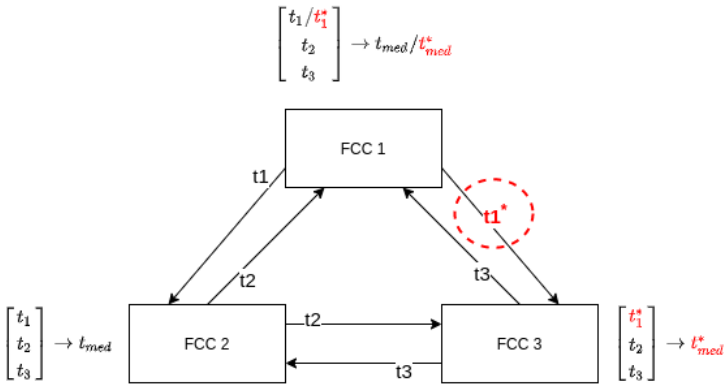


Figura 16: La FCC1 entrega un valor distinto de timing a las demás FCCs

.

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 17.

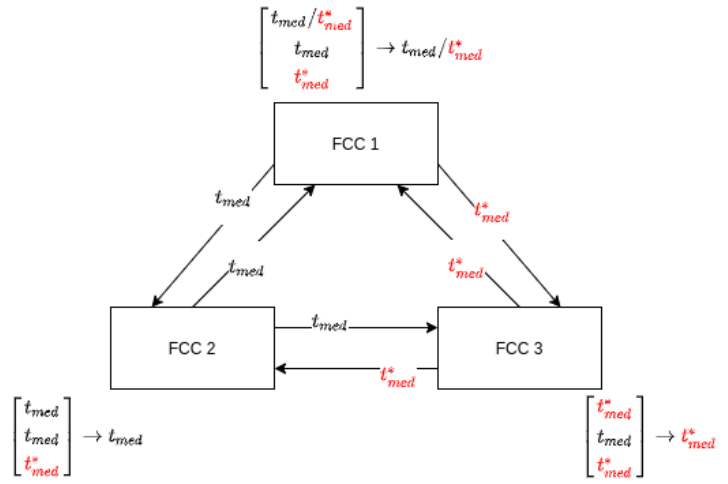


Figura 17: Luego de calcular los promedios, las FCCs intercambian sus resultados. Nuevamente, la FCC1 comete una falla en el envío del dato.

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 16, 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 [29][15][23]. El nombre proviene de un problema denominado *The Byzantine Generals Problem*, formalizado en [30]. Otros trabajos que tratan el mismo problema son [31] y [32]. 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 16, 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 18 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.

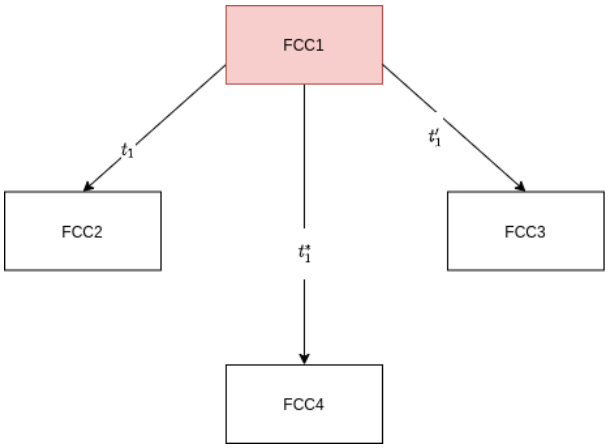


Figura 18: Debido a una falla, la computadora de vuelo 1 le entrega valores distintos de timestamp a las demás.

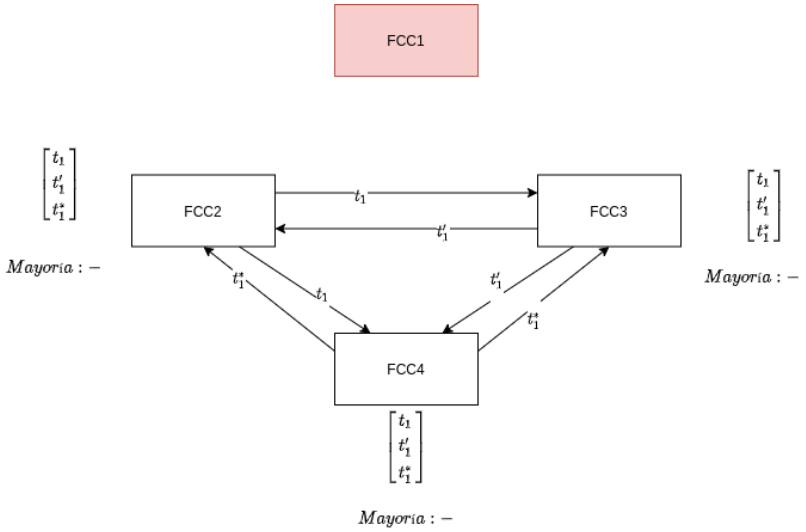


Figura 19: Las FCC2, 3 y 4 comparten entre sí lo que les dijo la FCC1 a cada una y llegan a la conclusión de que la información es inconsistente.

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**.

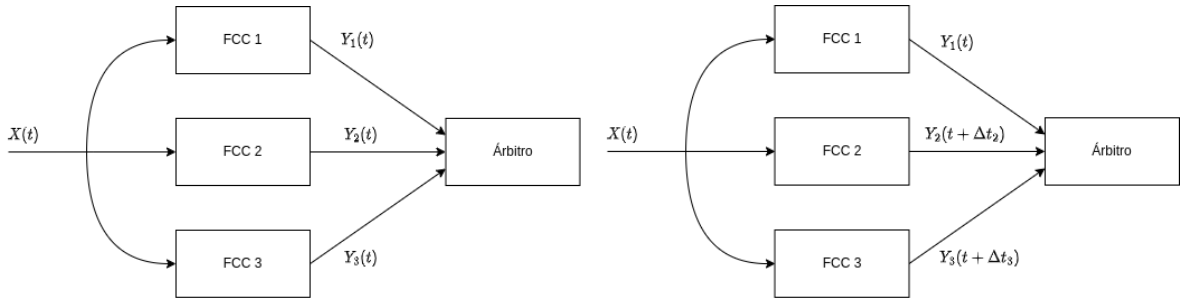
Un aspecto interesante es el hecho de que en el paper original, se hace una analogía entre un nodo redundante con fallas y un nodo traidor, es decir, que busca corromper el consenso de los demás nodos. Esto lo que quiere decir es que las fallas presentadas por las computadoras de vuelo pueden ser justamente de cualquier característica, incluso al extremo de presentar un comportamiento malicioso, con el objetivo de perjudicar al sistema [23]. 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.

4.2.4. Necesidad del Sincronismo entre Nodos

Prácticamente en todos los trabajos presentados se menciona que los nodos reundantes trabajan de manera sincronizada. Esta necesidad surge debido a que las comparaciones se realizan sobre variables que cambian en el tiempo y que tienen validez solamente durante un período de tiempo. Esto es algo característico de sistemas de tiempo real, ya que un dato de un sensor $X(t)$ solamente tendrá validez durante un breve período de tiempo. En la figura 20 se muestra un ejemplo. En el instante t, se presenta una nueva medición de un sensor a las tres computadoras de vuelo. Al comienzo de la misión, todas ellas estarán sincronizadas y generarán un resultado del cálculo de la ley de control que corresponde al mismo intervalo de tiempo. Luego, se realiza la votación para elegir el valor correcto. La figura 20b, muestra lo que sucede al cabo de un período de tiempo. Se presenta una nueva medición de un sensor en el instante t. Debido a la desincronización, es posible que las computadoras de vuelo no presenten sus resultados al árbitro a tiempo, por lo que este asumirá que una de las FCCs no presentó ninguna respuesta. Este caso suele estar contemplado dentro de las posibilidades y correpsonde al caso en el que una computadora

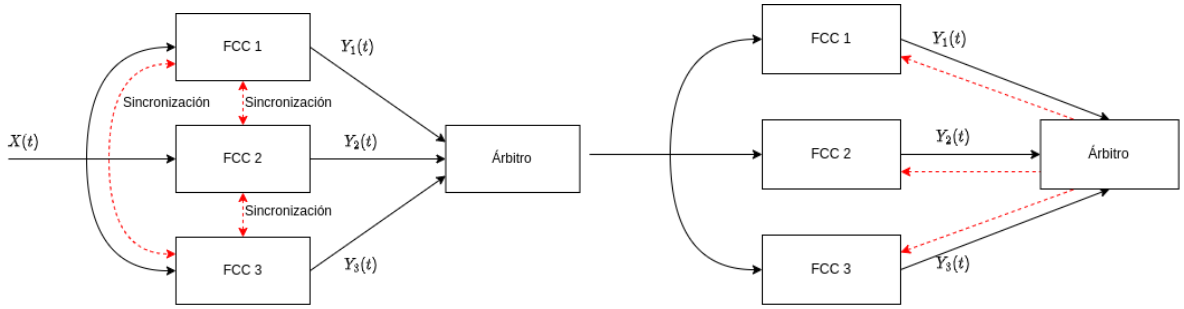
de vuelo presentó un error y debido a ello no respondió con ningún valor (por ejemplo, se reinició su procesador debido a un *watchdog*). En esos casos el árbitro simplemente asume algún valor por defecto.



(a) Computadoras de vuelo al inicio de la misión. (b) Al cabo de un período de tiempo, se desincronizarán.

Figura 20: A medida que transcurra el tiempo, la desincronización entre FCCs impactará en el sistema redundante.

Otra situación que puede presentarse, es que los resultados propuestos por las computadoras de vuelo Y_1, Y_2 e Y_3 correspondan a intervalos de tiempo distintos. Este caso es todavía peor que el anterior, ya que no se encuentra contemplado y el árbitro simplemente realizará la votación asumiendo que el dato es válido.



(a) Las mismas computadoras de vuelo se encargan de la sincronización. (b) En este caso, la sincronización es administrada por el árbitro.

Figura 21: La sincronización entre nodos es necesaria para un correcto funcionamiento de las redundancias.

Se concluye que es mandatorio utilizar alguna técnica de sincronización entre los nodos. Como detalle de la figura 21a, se muestra que la sincronización entre nodos presupone otro canal de comuniación más. Otra forma podría ser relegar la tarea de la sincronización al árbitro, aunque esto nuevamente presenta un punto singular de falla. Como se demostró en esta sección, el sincronismo es un aspecto crítico en el sistema redundante, por lo que se prefiere evitar esto último. Aunque de todas formas quisiera relegarse la sincronización al árbitro, este no solo recibirá mensajes de cada una de las FCCs, sino que además les enviará mensajes. Esto se muestra en la figura 21b. Puede ocurrir una situación en la que el árbitro entregue valores distintos a cada una de las FCCs, evitando que estas se sincronicen.

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 [30] y costosa a su solución [29]. 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 tambień ocurre por ejemplo en los automóviles, los nodos que se encuentran repartidos por todo el vehículo se comunican a través de redes como CAN[33] 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.

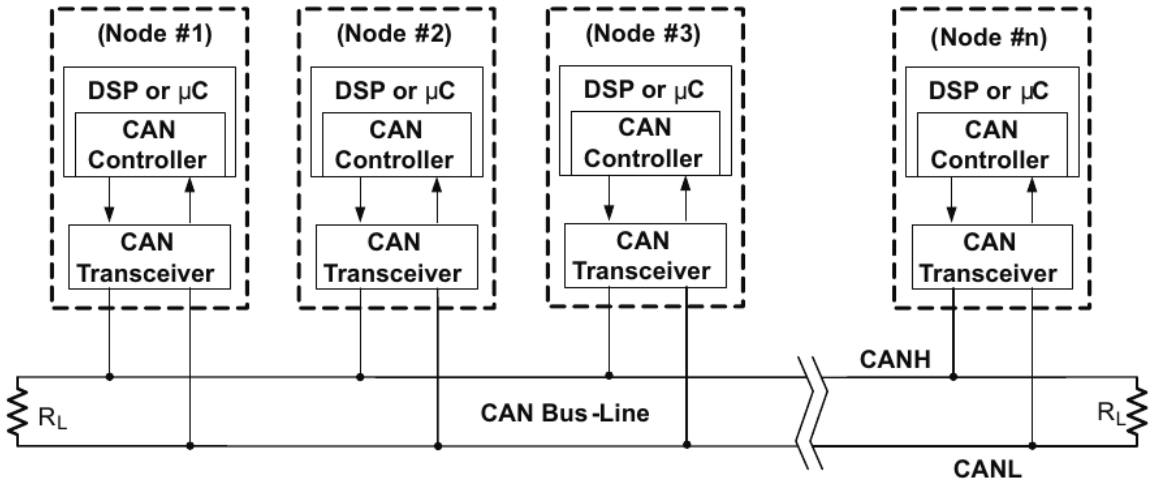


Figura 22: Todos los nodos se encuentran conectados al mismo bus de comunicaciones. En el caso del bus CAN, se compone de dos cables, CAN-H y CAN-L, terminados en sus extremos por resistencias de adaptación. La imagen se extrajo de [34].

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 23 con la figura 18.

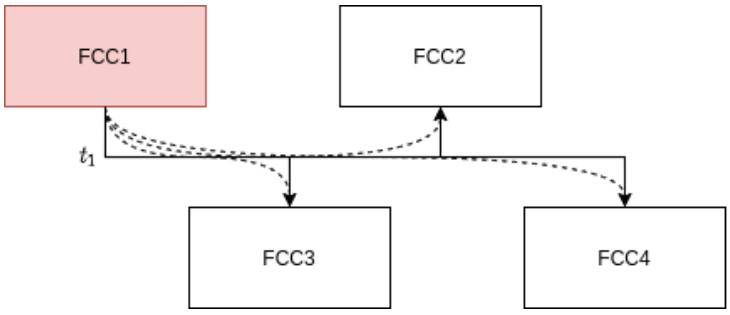


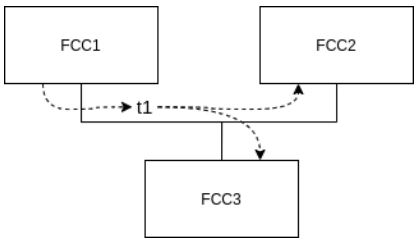
Figura 23: En este caso, la conexión tipo bus no permite el envío de información diferente a los demás miembros. La FCC1 envía el valor t_1 y todos sus pares reciben el mismo valor.

Como contrapartida, debido a que los nodos comparten canal de comunicación, estos deben tomar

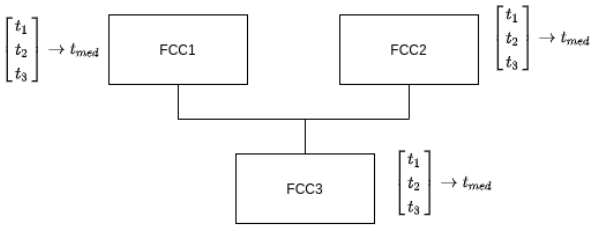
turnos para enviar la información a sus pares. De otra forma, habría una colisión en el bus y la información nunca llegaría a su destino. Sumado a esto, el bus se convierte en un punto singular de falla, ya que es posible que un problema en el bus deje a los nodos incomunicados.

Al igual que como se hizo en la sección 4.2.3, 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 24a, 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 24b 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 24: 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 25 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.

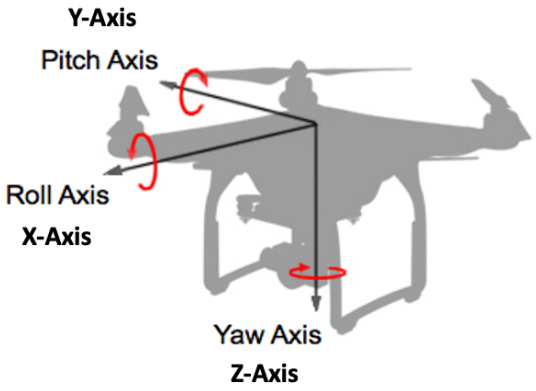


Figura 25: Se utiliza una terna solidaria al vehículo para conocer su orientación en el espacio. Los ángulos *yaw*, *pitch* y *roll* indican rotaciones respecto de una terna inercial.

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 25 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 inerciales. 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 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. Además, de manera de ampliar las capacidades, se incorpora una gran variedad de conectores que facilitan la comunicación con dispositivos y sensores externos. Esto le da una gran versatilidad, en el sentido de que permite 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

Una de las premisas de cualquier trabajo de desarrollo de electrónica, consiste en que este sea de un bajo costo. Gracias al avance de la tecnología, en los últimos años se han ido abaratando los costos de fabricación de chips y componentes electrónicos. Haciendo una búsqueda rápida en sitios web de distintos proveedores de componentes puede encontrarse que existe una gran variedad de estos, como sensores y microcontroladores, a precios razonables.

En el caso particular de sistemas críticos, el aspecto más importante y fundamental es el de la confiabilidad. Generalmente este requerimiento impacta en el costo del desarrollo, ya que la confiabilidad suele traer consigo altos costos de fabricación. Por ejemplo,

COMPLETAR

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 módulos externos. Algunos de esos conectores fueron seleccionados por una necesidad de tener compatibilidad con distintos módulos que son comúnmente utilizados con otras computadoras de vuelo. Estos serán mencionados en las secciones correspondientes.

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 Cotex-M7, y pueden llegar a velocidades entre 400 MHz y 550 MHz [35], 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 mircocontrolador. 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* [36]. 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 periódo 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 ELEIGIÓ 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.

SE PUEDE SIMPLEMENTE MENCIONAR LA CRISIS DE CHIPS COMO CAUSA DE LA FALTA DE DISPONIBILIDAD.

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.

Suamdo 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.

SE PUEDE EXPLICAR LA SELECCIÓN DE LOS CRISTALES Y EL USO DE LOS CAPACIOTRES DE DESACOPLE, SIGUIENDO LA APP NOTE DE ST.

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 inerciales, 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 26.

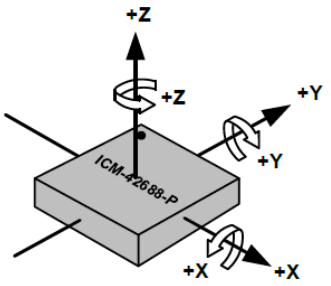


Figura 26: Todas las mediciones que entrega el sensor son realtivas a una terna solidaria a este. La imagen se extrajo de [37].

REEMPLAZAR LO QUE SIGUE POR UNA SIMPLE MENCIÓN DE SENSORES MEMS, DESDE ACÁ:

Los acelerómetros y giróscopos de la IMU utilizada en este trabajo, se construyen utilizando la tecnología MEMS: *Microelectromechanical Systems*. Utilizando técnicas de fabricación de circuitos integrados, se construyen los acelerómetros y giróscopos, integrando en el silicio partes que son móviles. En la figura 27, se muestra una imagen tomada con un microscopio electrónico de un acelerómetro MEMS. Lo que se observa en este caso, es que en el mismo silicio se integra una masa llamada *proof-mass*, la cual se encuentra sujeta al sustrato a través de dos resortes.

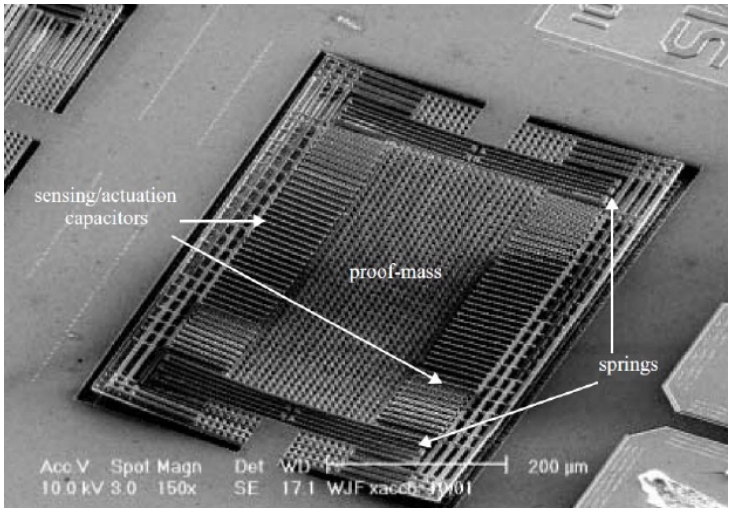


Figura 27: Fotografía tomada de un acelerómetro construido con tecnología MEMS. La imagen se extrajo de [38].

Una aceleración sobre el componente genera que la masa del acelerómetro se mueva. Estos desplazamientos producen una variación en la capacidad existente entre el sustrato y la masa, lo que lleva a una variación de la tensión entre ambos. Esta diferencia de potencial variable es medida por un circuito dentro del chip, y que luego se utiliza para generar las mediciones de aceleración.

El acelerómetro puede modelarse de manera simple, como un sistema mecánico con una masa, un resorte y un amortiguador [38], como se muestra en la figura 28.

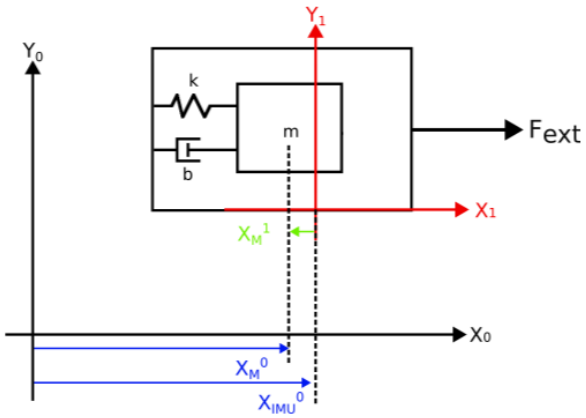


Figura 28: Sistema mecánico simplificado del acelerómetro.

La terna 0 corresponde a una terna inercial, mientras que la terna 1 es no inercial, solidaria al movimiento del acelerómetro. Cabe aclarar que tal como se mencionó, la masa se encuentra sujeta

al sustrato solamente a través del resorte. El elemento amortiguador representa pérdidas de energía, causadas por rozamiento con el aire o de la propia estructura electromecánica. Se puede resolver el sistema mecánico, tomando como coordenadas generalizadas $q_1 = X^0_{IMU}$ y $q_2 = X^1_M$. Sin considerar efectos de la gravedad, se llega a la ecuación (4). Este es un sistema de segundo orden, donde la entrada es la aceleración del acelerómetro y la salida es el desplazamiento de la masa respecto de la terna solidaria al acelerómetro.

$$\ddot{X^1_M} + \frac{b}{m}\dot{X^1_M} + \frac{k}{m}X^1_M = -\ddot{X^0_{IMU}} \tag{4}$$

Como resultado interesante, se observa que el sistema es de segundo orden, típicamente con respuesta sub-amortiguada. Algunos fabricantes de estos sensores indican en sus hojas de datos, un valor de frecuencia de resonancia. Este valor resulta de interés ya que si se excita al sensor con frecuencias cercanas a la resonancia, dejará de funcionar como dispositivo para medir la aceleración lineal. Por otro lado, a muy bajas frecuencias se puede despreciar la velocidad de la masa y luego se obtiene una relación directa entre la aceleración del acelerómetro y el desplazamiento de la masa, ecuación (5).

$$\ddot{X^1_M} \approx 0 \tag{5a}$$

$$\dot{X^1_M} \approx 0 \tag{5b}$$

$$\frac{k}{m}X^1_M \approx -\ddot{X^0_{IMU}} \tag{5c}$$

El desplazamiento de la masa produce una variación de la capacidad entre esta y el sustrato. Esta capacidad es utilizada para medir una varaición de tensión [38]. El circuito medido puede modelarse como en la figura 29.

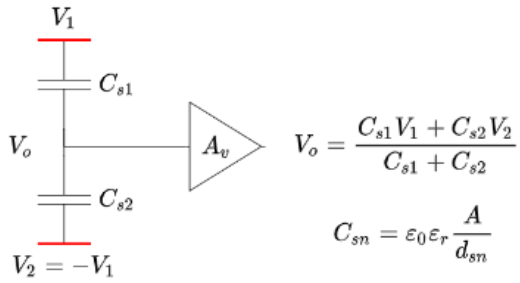


Figura 29: Circuito equivalente que mide el desplazamiento de la masa. Las tensiones $V1$ y $V2$ representan señales de tensión aplicadas por un circuito externo. El movimiento de la masa modifica la capcidad y por ende la tensión medida.

Se puede resolver este circuito y llegar a que existe una relación lineal entre la tensión Vo y el desplazamiento de la masa Δd, ecuación (6). En [38] puede encontrarse la demostración completa. En la ecuación, d_0 representa la separación de reposo entre el sensor y el sustrato y Δd la variación de la separación.

$$V_o = \frac{\Delta d}{d_0}V_1 \tag{6}$$

HASTA ACÁ

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 30 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 30: 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 10^{-6} g. Luego para un rango de $\pm 4g$ la resolución es de 8192 LSB/g, es decir, 122,07 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 [39]. Este se presenta en la ecuación (7), 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.

$$\omega_m = (1 + S)\omega + \omega_b(t) + \eta \tag{7}$$

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 [39]. Este paper presenta un análisis de los parámetros de los acelerómetros y grióscopos 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]	±250/500/1000/2000	±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	±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]	±2/4/8/16	±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: ±0,5, Z: ±1	±0,15
Offset vs time[μg/h]	N/A	N/A
Noise[$\mu g/\sqrt{Hz}$]	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 microcontroldor. 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 31 se muestra un esquema de la conexión entre el controlador y el sensor IMU.

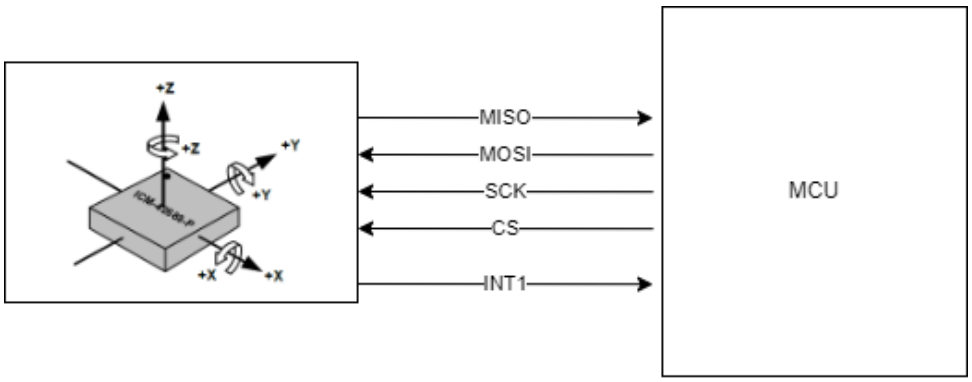


Figura 31: Líneas de comunicación entre la IMU y el microcontrolador.

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 32 se puede apreciar el efecto de la presión externa.

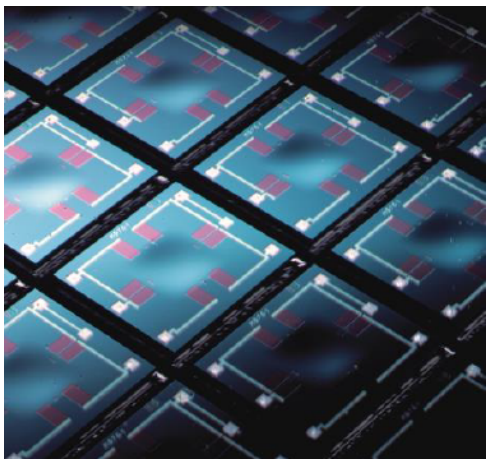


Figura 32: Sensores de presión sobre una oblea de silicio [40].

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 [41], 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 7 pero para la presión.

$$P_m = (1 + S)P + P_b(t) + \eta \tag{8}$$

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* [42]. 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 [40]. 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 drfit* se refiere al offset que se introduce por el propio proceso de soldadura [42]. 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]	±0,5	±0,3	±0,2	±0,5	±0,5	±1
realtive acc [hPa]	±0,03	±0,06	±0,01	±0,025	±0,015	±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 [43]. 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 peródica. 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, a partir de mediciones del campo magnético terrestre. Estas mediciones son relativas a una terna solidaria al sensor. Esto se muestra en la figura . Luego el ángulo de *yaw* puede obtenerse a partir del ángulo entre las componentes en H_y y H_x.

AGREGAR UNA IMAGEN DEL CAMPO MAGNÉTICO MEDIDO POR EL MAGNETÓMETRO CON SUS COMPONENTES

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	±0,1 % @ ±200 uT	1 %	±0,5 % @ ±200 uT	0,1 % @ ±400uT
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	± 40 uT	N/A	±50 uT
Offset Over Temp	N/A	N/A	N/A	±2nT/°C @ -40°C - 105°C
Noise	200 nT @ FSR = ±88 uT	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 proyceto 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 conectará como un sensor externo, utilizando alguno de los concetores 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)[33]. 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 similiar 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 33.

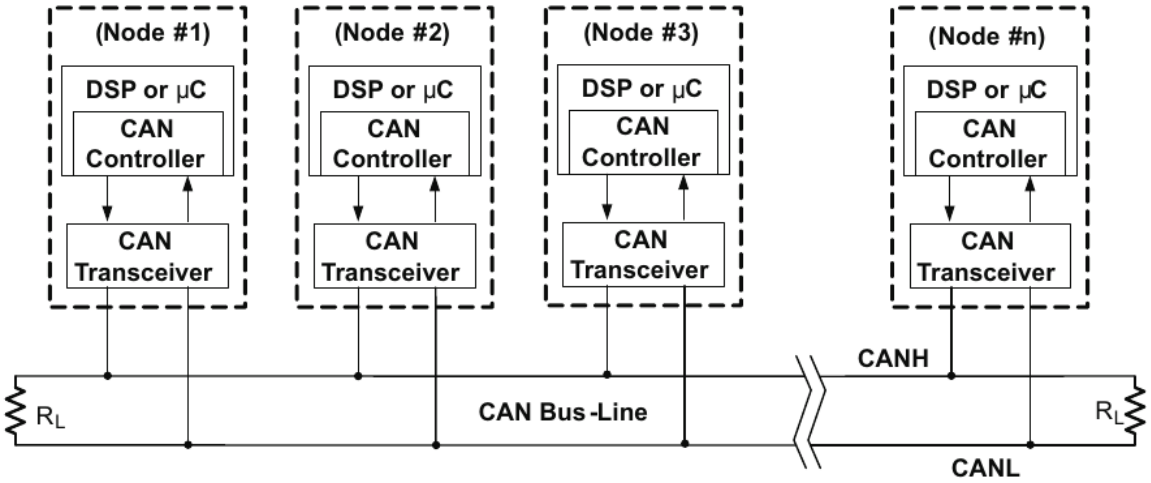


Figura 33: Todos los nodos se encuentran conectados al mismo bus de comunicaciones. El bus CAN se compone de dos cables, CAN-H y CAN-L, terminados en sus extremos por resistencias de adaptación. La imagen se extrajo de [34].

La impedancia característica del bus debe ser de 120 Ω. 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 OCSILOCSOPIO 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 34.

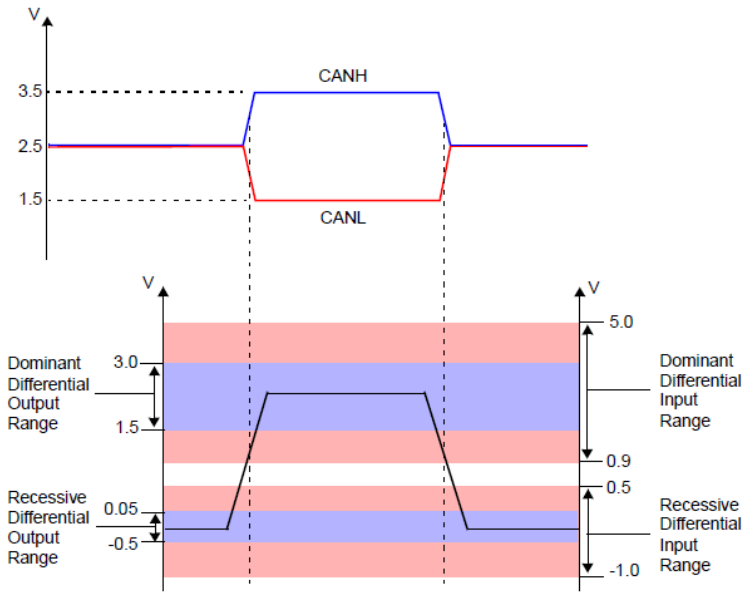


Figura 34: Se muestran los estados recessive y dominant del bus CAN y sus equivalentes lógicos.

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 33 se muestra que cada nodo se compone de 2 elementos, el *trasnciever* 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 comopone de varios campos. Además, se definen 2 tipos de tramas, estándar, figura 35 y extended, figura 36.

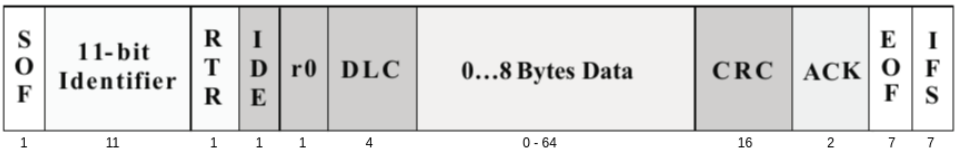


Figura 35: Se muestran los campos de una trama CAN estándar. Debajo de cada campo se indica la cantidad de bits. La imagen se extrajo de [34].

- 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 36: Se muestran los campos de una trama CAN extendida. Debajo de cada campo se indica la cantidad de bits. La imagen se extrajo de [34].

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 37 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.

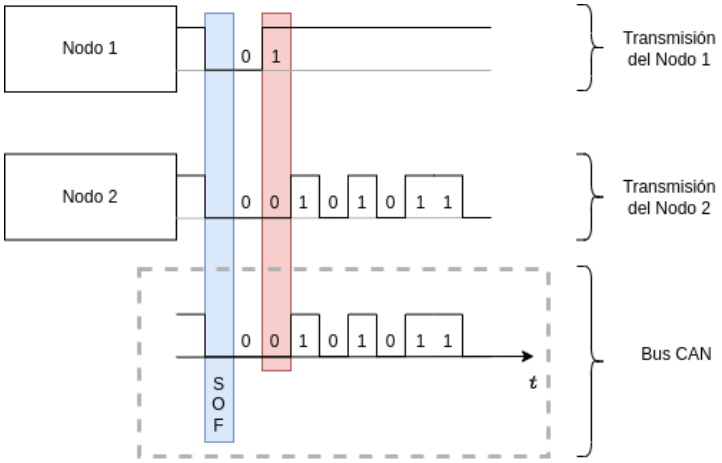


Figura 37: Mecanismo para la detección de colisiones. El nodo 2 gana por prioridad y completa su transmisión, mientras que el nodo 1 deja de usar el bus, aguardando a que el nodo 2 finalice.

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[44] 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 [45]. 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 *trasnciever*. En cuanto a este útlimo, 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 [46], el cual es compatible con la especificación de capa física de CAN, ISO 11898-2. En la figura 38 se muestra la comunicación del microcontrolador con el bus CAN a través del *transciever*. Este cuenta con una protección por exceso de tempertura, donde el componente pone sus salidas CAN-H y CAN-L en alta impdeancia, 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.

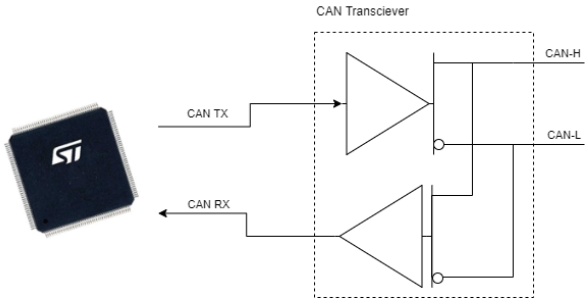


Figura 38: Se muestran las conexiones entre el *controller* embebido en el microcontroldaor y el *transciever*, a través de las líneas CAN TX y CAN RX. CAMBIAR ESTA IMAGEN POR UNA MÁS CLARA Y PROLIJA

Como se muestra en la figura 38, el *transciever* cuenta con un terminal *Rs* 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 $k\Omega$ en el terminal 8 denominado *Rs*, la hoja de datos indica un slew-rate de 15 $V/\mu s$.

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 [47] 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 [48], 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 39. 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.

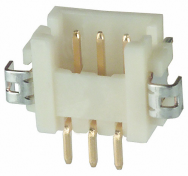


Figura 39: Se muestra el conector utilizado para las interfaces CAN. El mismo cuenta con 3 terminales, de los cuales 2 de se utilizaron para la señal CAN diferencia. El terminal restante corresponde al GND de la placa.

En cuanto al circuito implementado, se tomaron como punto de partida las recomendaciones de la hoja de datos del trasnciever 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 40. Estos componentes son de fácil acceso en el mercado local, lo que facilita el armado final de la placa.

Figura 40: 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 [49], 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 μF. 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 [50]. 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 [51]. Para ello, el fabricante recomienda un valor de ESR entre 0,05 Ω y 0,5 Ω, de un valor de por lo menos 4,7 μF. 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 [52]. Estos suelen excibir una ESR mayor a la de los MLCC.

Como fue ya fue 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 [53]. 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 41 se muestra un esquema con las conexiones entre el master y la memoria.

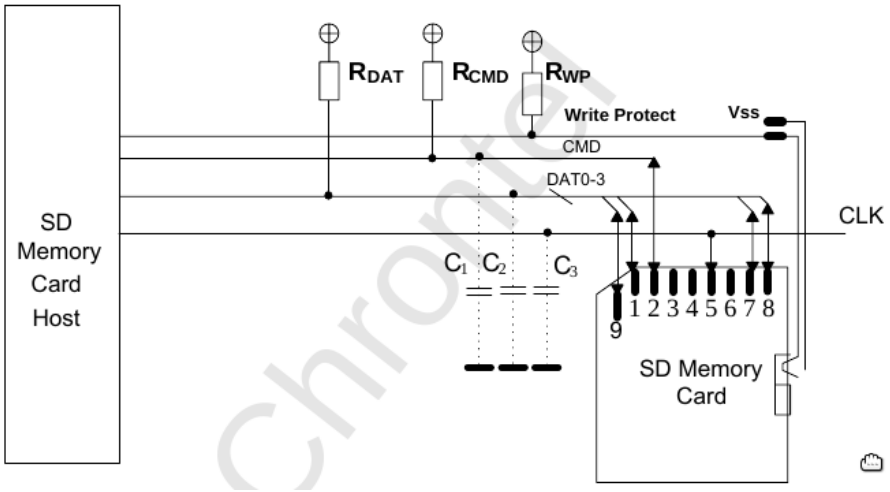


Figura 41: Esquema circuital de la conexión entre el SD host y una memoria SD, tal y como lo indica la especificación de la SD Association. ESTA IMAGEN NO PUEDE QUEDAR, TENGO QUE DIBUJAR UNA SIMILAR YO.

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

La placa además cuenta con un puerto micro USB B a través del cual también puede alimentarse la placa. Tanto la línea de 5 V del conector USB, como la del conector de alimentación principal, tienen en serie un diodo schottky. Estos diodos fuerzan a que las corrientes solamente fluyan desde las entradas de 5 V hacia el regulador. El circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático.

COMPLETAR

5.3.9. Micro Switch

COMPLETAR

5.3.10. LEDs Indicadores

COMPLETAR

5.3.11. Conector para Módulo GPS

COMPLETAR

5.3.12. Conectores para Salidas de PWM

COMPLETAR

5.3.13. Conector para Programación por SWD

COMPLETAR

5.3.14. Conector para Control por Radio

COMPLETAR

5.4. Desarrollo del PCB

COMPLETAR

5.4.1. Requerimientos de Manufacturabilidad

COMPLETAR
Acá poner el motivo por el que está todo en una sola cara.
También justificar el espesor del cobre y las vías pasantes en ves de vías ciegas.
Justificar tamaño de las pistas, dimensiones de la placa por costos de fabricación.

5.4.2. Requerimientos de layout del sensor IMU

El sensor IMU tiene ciertos requerimientos particulares que deben cumplirse. En primera instancia se buscó ubicar al componente lo más centrado posible en el PCB. De esta forma, la terna solidaria al sensor coincidirá con la terna solidaria a todo el vehículo en el que se utilice la placa. Esto ahorra cuestiones de cómputo que modifiquen la terna de referencia respsecto de la cual se obtengan las lecturas. **Completar qué tan corrida quedó la IMU del centro de la placa**

Se siguieron una serie de guías y recomendaciones que tienen como objetivo minimizar el estrés sobre el componente [54] [55] [56]. Esto debido a que, como la IMU es un sistema electromecánico, el estrés puede alterar las mediciones que esta realice, o incluso un alto nivel de estrés puede llegar a dañar el componente. Se enumeran algunas de esas recomendaciones:

- Montar la IMU 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. Los orificios deben estar a una distancia de por lo menos 2 mm del sensor.
- 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.

La imagen de la figura 42 resume algunas de estas recomendaciones. Lo que se observa es que las pistas del sensor son simétricas. Por más de que algunos terminales de la IMU no se utilicen, se recomienda que el routeo sea simétrico. Durante el proceso de soldadura, el estaño presente en los distintos pads del componente generará una tensión que tratará de atraer al componente hacia este. Si el routeo no es simétrico, es posible que el sensor no quede centrado, lo que resultaría en un grave problema durante el uso de la computadora de vuelo.

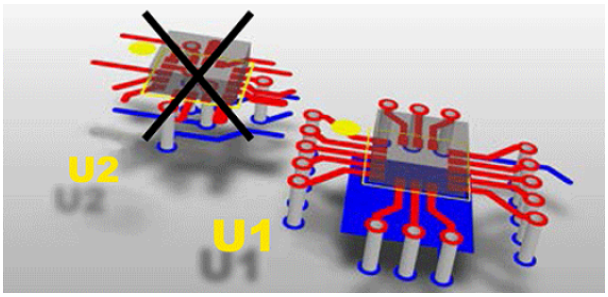


Figura 42: Se muestran dos ejemplos de layout de una IMU. El layout U2 no sigue las recomendaciones mencionadas, mientras que U1 sí. La imagen se extrajo de [56].

5.4.3. Layout del Regulador Lineal

COMPLETAR

5.4.4. Comunicación con el Slot para Tarjeta Micro SD

COMPLETAR

5.4.5. Comunicación USB

COMPLETAR

5.5. Funcionalidades y Diagrama en Bloques

La computadora de vuelo tiene la tarea esencial de adquirir las mediciones de los sensores, procesar dichos datos, y realizar el control de los diversos aspectos del vehículo, fundamentalmente de los motores. Las funcionalidades de la computadora de vuelo son las siguientes:

1. Adquirir datos de sensores.
2. Cálculo de la estimación de la pose y de la ley de control.
3. Actuación sobre los motores.
4. **FALTA ALGUNA FUNCIONALIDAD DE TOLERANCIA A FALLAS**
5. Control de LEDs indicadores de propósito general.
6. Comunicación a través de distintas interfaces, con módulos y sensores externos a la placa.
7. Loggeo de datos.

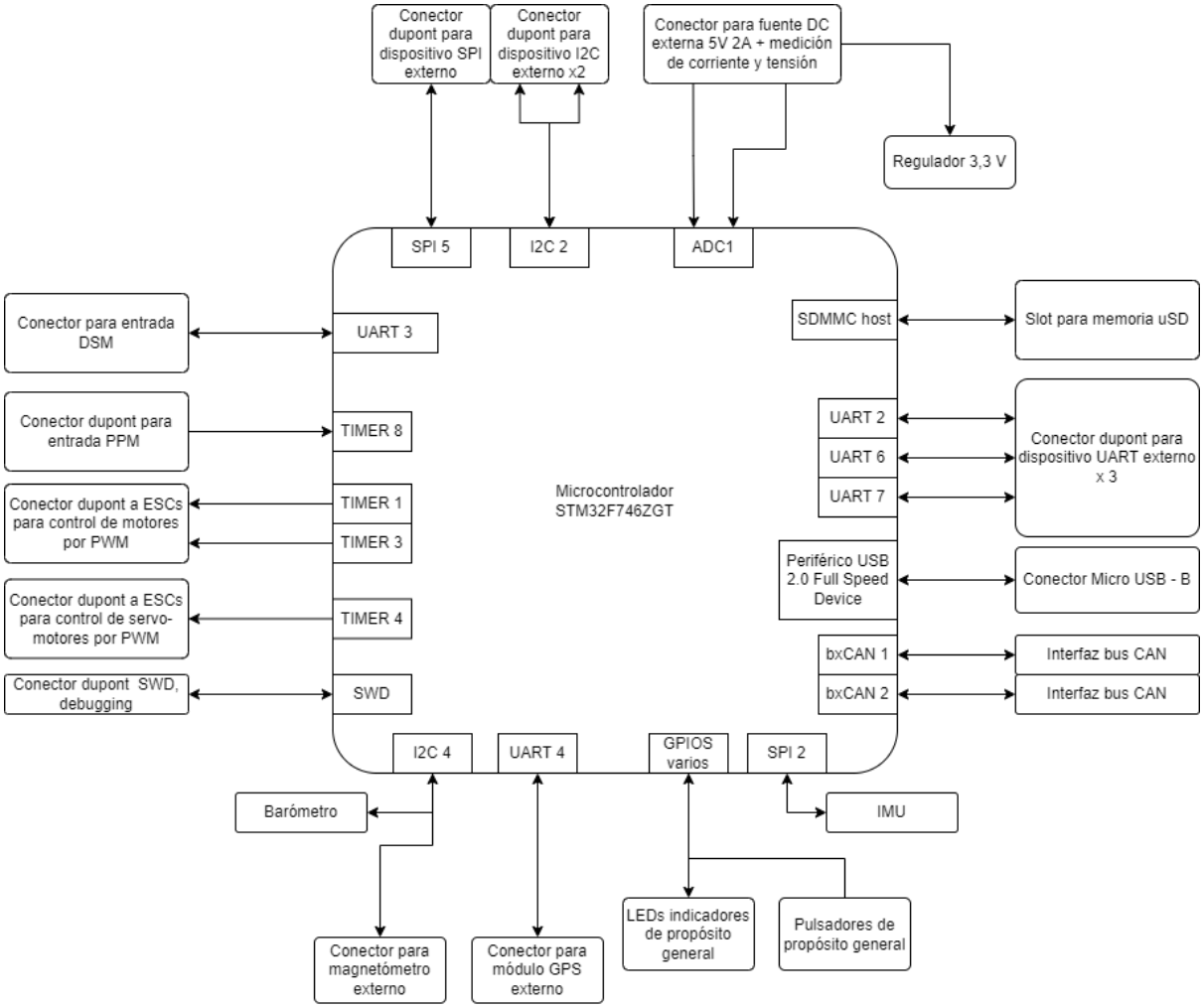


Figura 43: Diagrama en bloques de la computadora de vuelo.

COMPLETAR

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. Para demostrar las capacidades de toleranca a fallas, se hicieron una serie de pruebas simulando fallas en un sensor. Además, se describen las características del firmware desarrollado para implementar el sistema propuesto. Finalmente, se muestran los resultados obtenidos sobre las pruebas realizadas.

6.1. Arquitectura Propuesta: *The Time-Triggered Architecture*

En la sección **??** se mencionó la necesidad del sincronismo entre los nodos y que esta se logra a partir de un intercambio de mensajes. Para la arquitectura propuesta, ese intercambio de mensajes se hace a través del mismo bus. Debido a que el medio es compartido, los nodos de la red deben tomar turnos para acceder al medio, de manera de que todos puedan enviar sus respectivos mensajes.

Típicamente, una FCC ejecuta las mismas tareas relacionadas al control del vehículo, de manera periódica [14]:

1. Lectura de los sensores.
2. Cálculo de la ley de control.
3. Aplicación del resultado a los actuadores.

Debido a que se trata de un sistema de tiempo real, cada una de las FCCs debe realizar estas tareas en un período de tiempo dado. En la figura 44 se muestra un gráfico con la secuencia de ejecución periódica de las tareas.

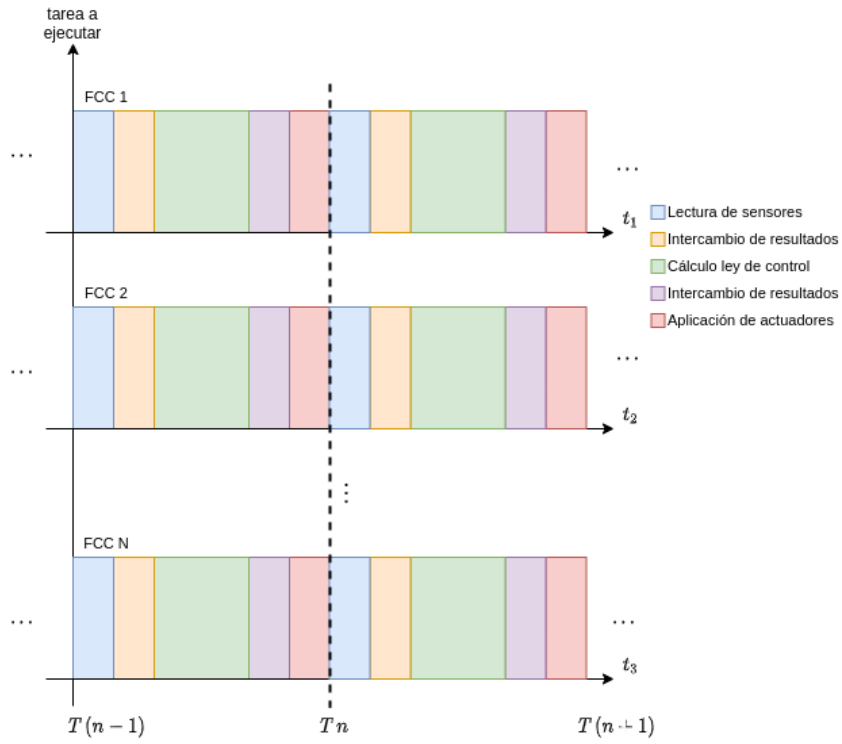


Figura 44: El eje horizontal representa el paso del tiempo. Las barras de colores representan el tiempo dedicado a ejecutar cada tarea, como la lectura de sensores, cálculo de la ley de control, etc., de forma periódica. En la imagen se puede ver que las FCC 1, FCC 2, ..., FCC N se encuentran sincronizadas ya que realizan las tareas al mismo tiempo.

En un sistema con redundancias, como ya se mencionó, cada una de las FCCs realiza las mismas tareas. Además, estas intercambiarán resultados relacionados a mediciones de sensores y a valores de actuación para los motores con sus pares, justamente para enmascarar y tolerar las fallas. A partir de esto, se desprende que el intercambio de mensajes también corresponde a tareas que deben ejecutarse periódicamente y en un determinado período de tiempo acotado.

Cada una de las computadoras de vuelo incluye un clock interno, el cual utiliza como base para cumplir con los tiempos de ejecución. Cuando se habla del sincronismo entre los nodos de la red, lo que se busca es que las ejecuciones de las N computadoras se encuentren coordinadas. Por ejemplo, en la figura 45 se muestra un caso para dos computadoras de vuelo cuya ejecución se encuentra desfasada. Es fácil ver que este sistema nunca podrá cumplir con el objetivo de controlar el vuelo del UAV.

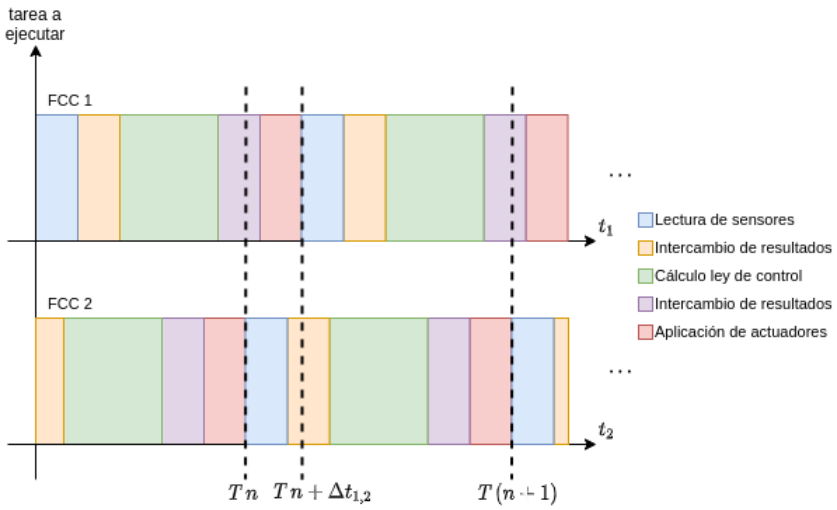


Figura 45: Se muestra un ejemplo para dos FCCs. A diferencia de la figura 44, las FCC 1 y la FCC 2 se encuentran desincronizadas.

En la figura, se muestra que cuando la FCC 2 termina de enviar la señal de control a los actuadores (instante T_n), la FCC 1 se encuentra intercambiando resultados con la FCC 2. Debido a que la FCC 2 ya pasó dicha tarea, la FCC 1 no recibirá ningún valor de su par y asumirá erróneamente, que la FCC 2 se encuentra en un estado con alguna falla, ya que no responde. Si bien este ejemplo es muy simple, muestra la necesidad de la sincronización entre nodos de la red, siendo el motivo principal, el hecho de que el sistema es de tiempo real.

A partir de lo analizado hasta aquí, se enumeran los requerimientos más elementales del sistema:

1. El sistema es de tiempo real, es decir, se requiere determinismo temporal en la ejecución de las tareas.
2. El sistema es redundante, por lo que cada nodo ejecuta las mismas tareas en paralelo y al mismo tiempo.
3. El uso del bus de comunicación obliga al uso de un protocolo de acceso al medio físico compartido (el bus) por turnos, que permita que todos los nodos tengan acceso a este.

A continuación, se presentan distintas arquitecturas posibles para la implementación y se selecciona una de ellas, la *Time-Triggered Architecture*[24], cuyas características se ajustan a los requerimientos.

ACÁ COMENTAR LAS ALTERNATIVAS DE LAS DISTINTAS ARQUITECTURAS, COMO SUPER LOOP, EVEN TRIGGERED CON RTOS, SIN RTOS Y TIME TRIGGERED. COMENTAR POR QUÉ ELIJO TIME TRIGGERED Y DESCARTO LAS DEMÁS. PONER CASOS DE TRABAJOS CON TTA. EN EL PAPER QUE USA TIME TRIGGERED BUSES HAY VARIOS EJEMPLOS

En este tipo de arquitectura, el sistema en cuestión ejecuta sus tareas en instantes de tiempo predefinidos. Estas tareas pueden ser tales como tomar datos de un sensor, enviar un dato a otra parte del sistema o realizar algún cálculo. Este tipo de arquitectura es típicamente utilizada en aplicaciones de tiempo real críticas. **PONER EJEMPLOS DE SISTEMAS CRÍTICOS CON TIME TRIGGERED ARCHITECTURE**. Esto es porque esta arquitectura vuelve al sistema predecible. Teniendo en cuenta lo mencionado en la sección 6.3.5, que el comportamiento del sistema sea predecible lo vuelve más confiable y por ende más seguro.

Existe otro criterio por el cual típicamente se prefiere este tipo de sistemas para aplicaciones de este estilo y tiene que ver con los procesos de validación que deben realizarse frente a entes reguladores. El hecho de tener un sistema cuyo comportamiento es altamente predecible, simplifica mucho su proceso de validación. **DESARROLLAR ESTA PARTE CON EJEMPLOS Y CITANDO DO-254, ETC**.

A continuación, se presenta brevemente los componentes y el funcionamiento de la arquitectura para el sistema de este trabajo. Pueden encontrarse trabajos[24][57] y libros[9] que explican formalmente esta arquitectura y sus distintos componentes con más detalle.

Como ya se mencionó en las secciones anteriores, el sistema se compone de una serie de **nodos** interconectados a través de un **bus de comunicación**. Cada uno de los nodos ejecuta una serie de tareas con un **scheduling** predefinido por el paso del tiempo.

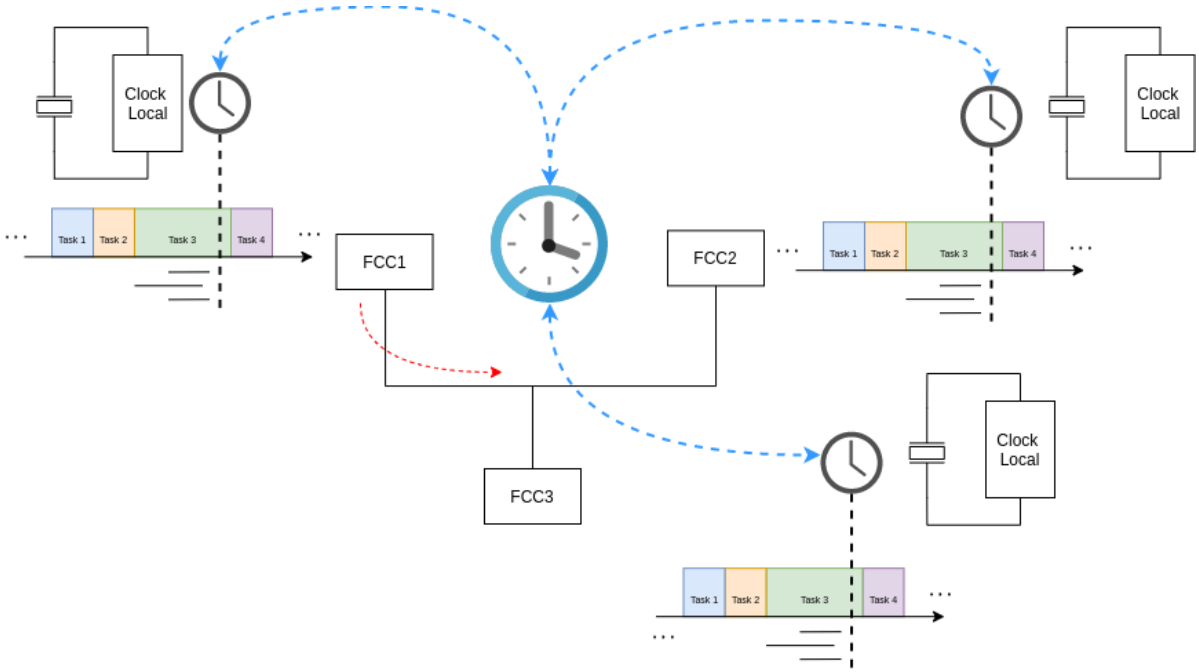


Figura 46: Se muestra un esquema que representa la arquitectura. Cada nodo tiene un *clock* local, funcionando a partir de su propio cristal. Todos estos a su vez se sincronizan periódicamente con el *global time*, representado por el reloj azul en el centro. En la imagen, el nodo 1 está enviando un mensaje por el bus. Los demás nodos saben previamente que deben esperar este mensaje.

Para que el comportamiento de los nodos sea consistente, estos deben estar **sincronizados**. Se define entonces una base de tiempo global a todos los nodos, denominada en la bibliografía **global time base**[9, p. 51]. Esta representa a un reloj que no existe físicamente, sino que es un acuerdo entre los nodos del sistema respecto a un reloj al que todos los nodos deben seguirle el ritmo. Para esto último, cada nodo tiene su propio **reloj local**.

En la figura 46 se muestra un esquema de la arquitectura *Time Triggered*. Los tres nodos de la imagen, FCC1, FCC2 y FCC3 se encuentran sincronizados ejecutando la tarea *Task 3*. Esta tarea implica que el nodo FCC1 envíe un mensaje por el bus, representado en la imagen por la flecha roja. En cuanto a los nodos FCC2 y FCC3, la *Task 3* les dice que ellos deben escuchar el bus y esperar el mensaje proveniente del FCC1. Esto quiere decir que el comportamiento predefinido por el scheduler también define en qué instantes de tiempo cada nodo puede enviar un mensaje y en qué instantes de tiempo debe recibirlo. Con respecto a esto último, si en el ejemplo de la figura 46 el nodo FCC1 presenta una falla y no envía el mesanje en el tiempo correspondiente, luego los nodos FCC2 y FCC3 no recibirán nada. Es común encontrar protocolos de comunicación donde este tipo de fallas se resuelven solicitando el reenvío del

mensaje. Sin embargo, debido a que el sistema ya tiene un scheduling predefinido, esto no se permite ya que uno a priori no puede saber si habrá que hacer un pedido de reenvío de mensaje o no, lo que puede corromper el scheduling del sistema. Justamente, la *Time-Triggered Architecture* busca que el sistema sea predecible.

A continuación, se describe brevemente cada uno de los componentes de este tipo de sistema. Para una explicación más detallada, referirse a [9] y [24].

El bus de comunicaciones es el medio a través del cual los nodos intercambian información. Como se describió anteriormente, la arquitectura requiere la sincronización de los nodos para una ejecución consistente. Esto quiere decir que como mínimo, los nodos intercambian mensajes utilizados para la sincronización entre estos. Más allá de esto, en un sistema distribuido, lo más común es que haya un intercambio de información constante entre nodos.

De manera de minimizar las colisiones y favorecer el cumplimiento en el timing del sistema de tiempo real, el envío y recepción de mensajes se implementa por turnos. Esto corresponde a un protocolo de acceso al medio denominado *Time Division Multiple Access* (TDMA). En la figura 47 se grafica esto para 3 nodos. 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.

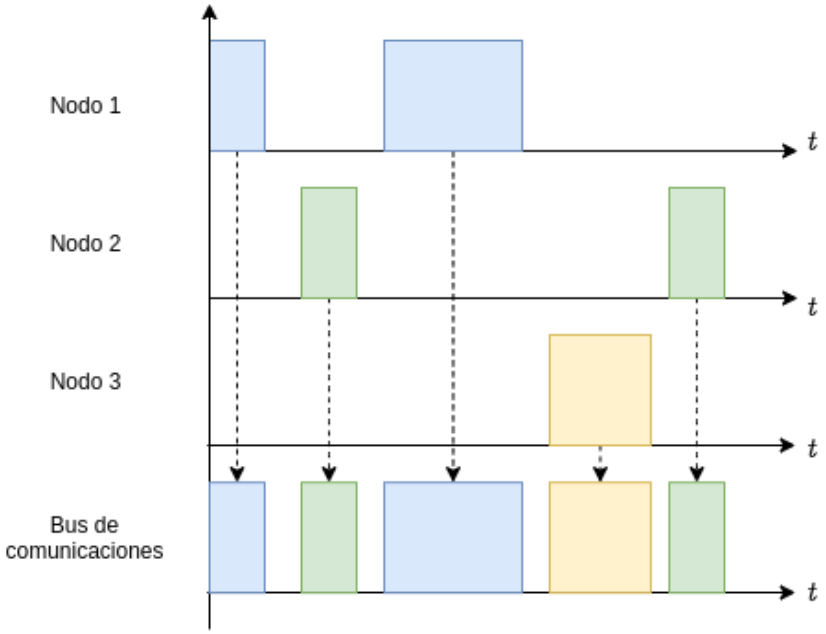


Figura 47: El ejemplo muestra como los 3 nodos pueden compartir el bus de comuncaciones. Cada uno de ellos sabe en qué instante de tiempo enviar un mensaje y en qué instantes de tiempo no deben hacerlo y solamente pueden escuchar el bus. Para que esto funcione adecuadamente, los nodos deben estar sincronizados.

ACÁ MENCIONAR ALGUNOS PROTOCOLOS COMO FLEXRAY O TTP/C QUE DE POR SÍ YA EJECUTAN UNA SINCRONIZACIÓN.

En [24] se define un elemento del nodo denominado *Communication Network Interface*(CNI). Esta es una interfaz entre el software del nodo y el acceso al medio físico. Utilizando el protocolo de acceso al medio TDMA, el software del nodo *pushea* un mensaje a la CNI. Es esta última la que se encarga de administrar los tiempos de envío y recepción. Es decir, mientras el nodo continua con sus tareas, la CNI se encarga de cumplir con el timing del envío del mensaje.

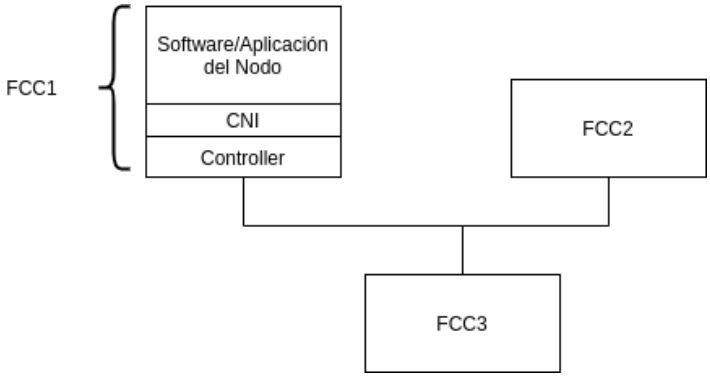


Figura 48: Misma imagen que 24a. Se muestra el detalle de los nodos, en el nodo FCC1.

Al recibir un mensaje, ocurre algo similar. El software del nodo *pollea* a la CNI en determinado instante de tiempo para obtener el mensaje recibido. Otra forma de implementar esto podría ser a través de una interrupción en el software, es decir, que cuando llege un mensaje nuevo, se dispare una interrupción en el software. El problema con esto es que se le quita determinismo al sistema.

Los nodos son la unidad elemental de los sistemas distribuidos, y también de la arquitectura *Time-Triggered*. Estos son los que ejecutan las tareas y le dan vida al sistema. Los nodos se componen generalmente de un procesador (microcontrolador en el caso de este trabajo), un clock local (en este trabajo se implementa con un circuito oscilador con un cristal) una unidad de control de acceso al bus de comunicaciones y el software local al nodo. En la sección anterior además, se mencionó la existencia de un elemento llamado CNI, que comunica al software con el bus. Sumado a esto, el nodo a su vez contiene un scheduler, el cual se describie en la próxima sección.

El sistema conformado por el conjunto de nodos y las comunicaciones entre ellos, se basa en un esquema de tareas que se encuentra predefinido. Esto se diferencia de sistemas de otras características como puede ser alguno con una interfaz con una persona. El caso más cercano puede ser el de un telófono celular, el cual tiene una pantalla que el usuario puede presionar en distintos lugares para abrir una aplicación, enviar un mensaje, etc. Cuando esto ocurre, el celular debe dar una respuesta casi inmediata. Este tipo de sistemas se llaman *Event-Triggered* y son controlados por los eventos que pueden ocurrir. A priori, se asumen eventos asincrónicos, es decir, que pueden ocurrir en cualquier momento y en cualquier orden. A diferencia de esto, en un sistema *Time-Triggered* los eventos no pueden ocurrir en cualquier momento. Mejor dicho, los eventos pueden ocurrir en cualquier momento, pero el sistema solamente prestará atención a esos eventos en un lapso de tiempo predefinido.

En los sistemas operativos de tiempo real se definen una serie de tareas, las cuales utiliza un scheduler que determina cuál es la próxima tarea que corresponde ejecutar. Un sistema *Time-Triggered* también define su comportamiento a través de tareas. La diferencia está en la estrategia utilizada por el scheduler. En el caso de un RTOS es común encontrar schedulers *preemptive* con un sistema de prioridad. En un sistema *Time-Triggered*, los schedulers pueden ser *preemptive* o bien cooperativos. La ventaja de utilizar un scheduler cooperativo es nuevamente el determinismo en la ejecución de las tareas [58, p. 247].

De forma de que el funcionamiento del sistema de tiempo real distribuido sea correcto, todos los nodos deben ejecutar sus tareas de forma consistente. Para ello, sus schedulers deben estar sincronizados. Para el caso de este trabajo, cada una de las FCCs debe encontrarse ejecutando la misma tarea al mismo tiempo. De esta forma pueden ejecutarse los algoritmos de votación para realizar tolerancia a fallas de hardware, como ya se describió anteriormente.

Algunos sistemas distribuidos de tiempo real utilizan un clock maestro, implementado como un nodo de la red al que todos los demás nodos utilizan como referencia. Por ejemplo, la extensión del protocolo automotivo CAN, denominada Time-Triggered CAN (TTCAN) [59] 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.

Otra forma es utilizar una *global time base*[9, p. 51]. Esta se define como un acuerdo entre los nodos

de la red respecto a una base de tiempo a utilizar como referencia. Como ya fue mencionado, cada nodo tiene un reloj interno propio, implementado con un cristal oscilador. Esta base de tiempos global puede implementarse por ejemplo utilizando un contador local. Cada nodo incrementa en uno el valor de esta variable, de forma periódica. Para que los nodos puedan utilizar esta variable como base de tiempos, todos los nodos deben tener el mismo número almacenado en su instancia local de dicha variable, al mismo tiempo.

A priori, puede parecer que el hecho de que el valor del contador sea igual en cada nodo al mismo tiempo sea muy exigente. En la implementación, esto no es posible pero tampoco es necesario. Puede existir cierta precisión en la sincronización que dependiendo del hardware y del algoritmo de sincronización, se obtendrá un mejor o peor resultado.

Luego de ejecutar el algoritmo de sincronización, los clocks de cada nodo quedarán sincronizados con cierta precisión Φ. A medida que pase el tiempo, inevitablemente los clocks de cada nodo comenzarán a desfasarse uno del otro, lo cual incrementará el error. Por consiguiente, la sincronización debe ejecutarse de forma periódica. Esto se grafica en la figura 49.

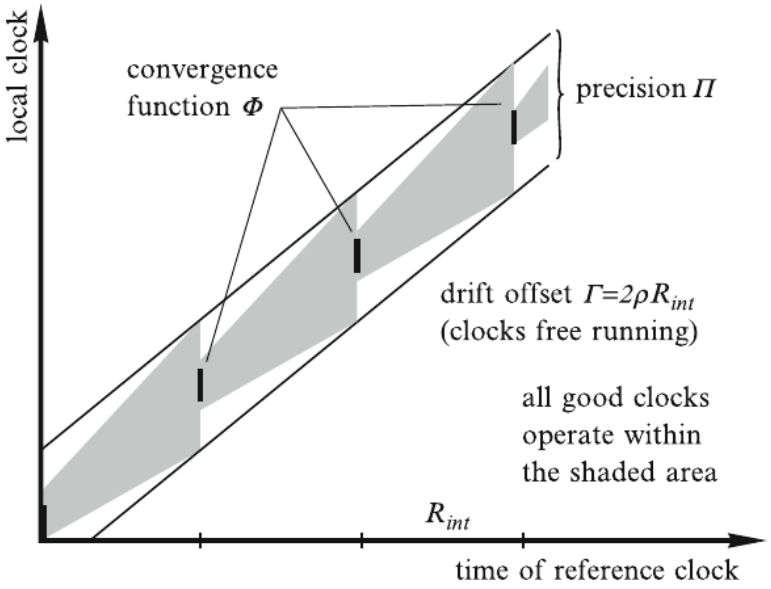


Figura 49: El eje horizontal representa el paso del tiempo físico y el eje vertical el avance de cada clock local a cada nodo. El valor R_{int} corresponde al período de resincronización. La imagen se extrajo de [9, p. 67].

Existen muchísimos algoritmos de resincronización. En [60] se puede encontrar un estudio que compara distintos tipos de algoritmos. Un planteo interesante de este trabajo es que los algoritmos de resincronización pueden dividirse en tres bloques básicos, figura 50, donde lo que varía es la implementación de cada bloque.

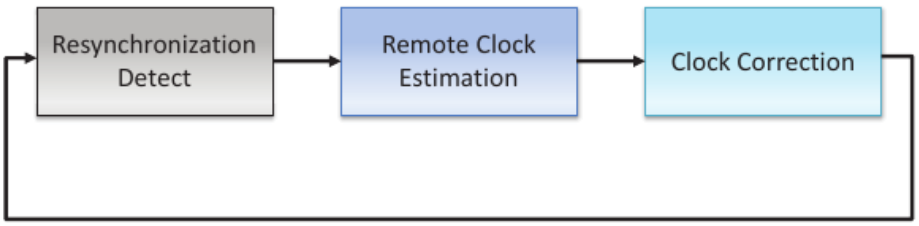


Figura 50: Tres bloques que comprenden un algoritmo de resincronización. La imagen se extrajo de [61].

1. *Resynchronization Detect*: Este bloque está dedicado a detectar e informar al nodo de que se va a ejecutar la resincronización.
2. *Remote Clock Estimation*: Este es el bloque que realiza la cuenta del error entre el clock del nodo y la corrección a aplicar.
3. *Clock Correction*: Este bloque corresponde a la aplicación de la corrección al clock local.

Para la arquitectura de este trabajo, el primer bloque, *Resynchronization Detect* simplemente consiste en ejecutar una tarea que estará incluida en el scheduling. El segundo bloque es el que realiza el cálculo y puede variar dependiendo de la implementación. Más adelante, se describirá el algoritmo utilizado. Por último, el bloque *Clock Correction*, corresponde a aplicar la corrección al clock local. En general existen dos formas de realizar esto. La primera es pisando el contador existente con el nuevo valor calculado. Este método puede generar inconsistencias en la ejecución de las tareas del nodo [9, p. 72]. La forma que se prefiere es ir aplicando correcciones sucesivas conforme se van reajustando los clocks. Esto es algo similar a un algoritmo de control, donde se calcula un error y en función de dicho error, se aplica una corrección a la acción de control. En este último caso, la corrección puede implementarse por ejemplo dejando pasar más o menos tiempo para incrementar el contador del clock local.

6.2. Implementación en Firmware

El objetivo de esta sección es explicar cómo se implementaron las características del sistema Time-Triggered Architecture. Por ejemplo, cómo se implementó el scheduler, la sincronización y la comunicación entre nodos.

6.2.1. Scheduler

6.2.2. Comunicación entre Nodos

6.2.3. Sincronización

6.2.4. Tarea de Ejemplo: Adquisición de Datos del Sensor IMU

El objetivo de esta sección es dar una primera muestra de que lo que se implementó funciona como se dijo que debe funcionar. Para eso, quiero poner capturas de analizador lógico, mostrando que los mensajes se envían en los tiempos configurados y que el intervalo de tiempo entre mensajes se mantiene estable gracias al efecto de la sincronización periódica.

6.3. Pruebas Realizadas

Se describe el setting común de las pruebas: Que las 3 placas calculan pitch y roll, luego lo comparten, comparan y se obtienen los residuos. Las fallas se inyectan articialmente. En cada sección se explica la característica de la falla en particular.

6.3.1. Funcionamiento Sin Fallas

6.3.2. Bias en Valores de Giróscopo

6.3.3. Saltos Aleatorios en Valores de Giróscopo

6.3.4. Medición Constante e Invariante de Acelerómetros y Giróscopos

6.3.5. Mediciones Inconsistentes de Acelerómetro

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 3. ***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 (9), 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.

$$R(t) = \mathtt{P}\,(\text{funcionamiento correcto en } t | \text{funcionamiento correcto en } t_0) \tag{9}$$

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 (10). 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.

$$\begin{aligned} R(t) = {} & \mathtt{P}\,(\text{no ocurrio una falla en } [t_0; t]) \\ & + \mathtt{P}\,(\text{funcionamiento correcto en } t | \text{ocurrió una falla en } [t_0; t])\ \mathtt{P}\,(\text{ocurrió una falla en } [t_0; t]) \end{aligned} \tag{10}$$

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 (11) y la única manera de mejorarla sería incrementando la probabilidad de que no ocurra ninguna falla en el intervalo $[t_0; t]$.

$$R(t) = \mathtt{P}\,(\text{no ocurrio una falla en } [t_0; t]) \tag{11}$$

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 (10), 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 ceritifcante, 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 [23].

Una forma de alivianar 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 [29][15][23]. Se define un sistema tolerante a este tipo de fallas.

Definición 4. ***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* [30], 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][22][23][24]. 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 6.3.5, 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.

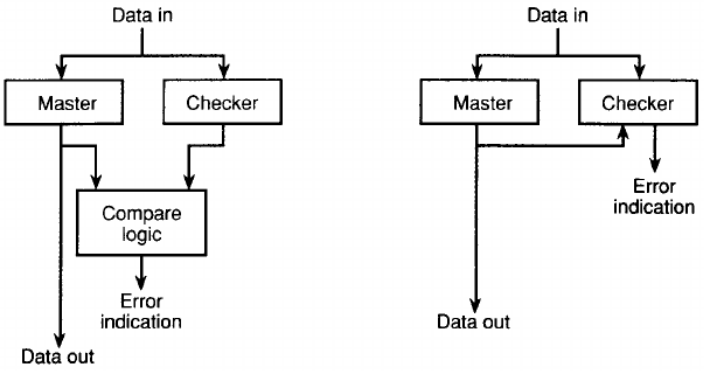
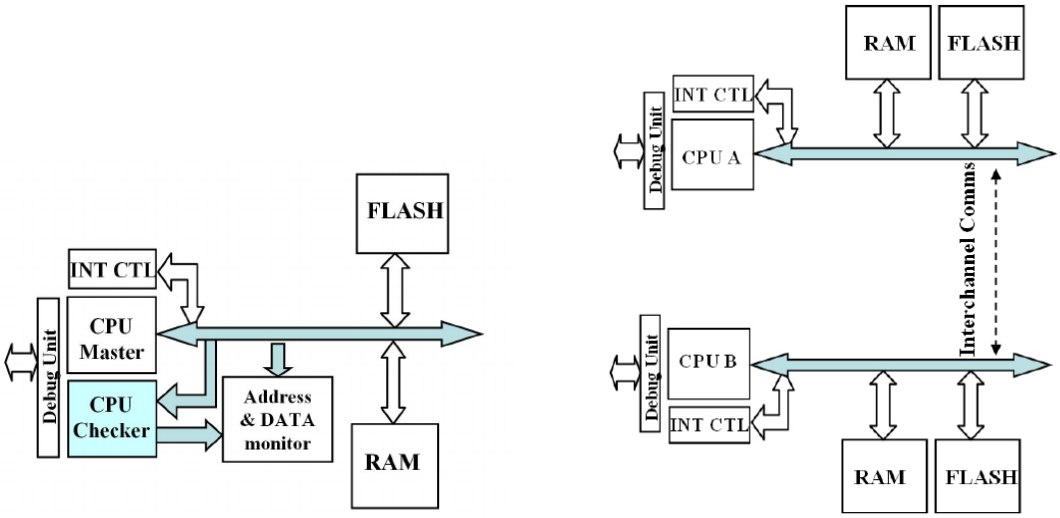


Figura 51: En la figura de la izquierda, dos sistemas ejecutan las mismas operaciones, mientras que otro sistema externo se encarga de comparar las salidas de ambos para detectar errores. En la figura de la derecha, el bloque comparador se encuentra integrado en el sistema *checker*. La imagen fue extraida 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 51 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*[25]. En esta última, ambos se encuentran sincronizados por estar en el mismo chip y compartir fuente de clock. En la figura 52 se muestra un esquema de ambos casos.



(a) Lockstep dual processor architecture. (b) Loosely synchronized dual processor architecture.

Figura 52: Se muestran dos casos para un sistema con redundancia doble. La imagen fue extraida de [25].

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 [26] 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* [25][6][22][27]. 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 53 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*.

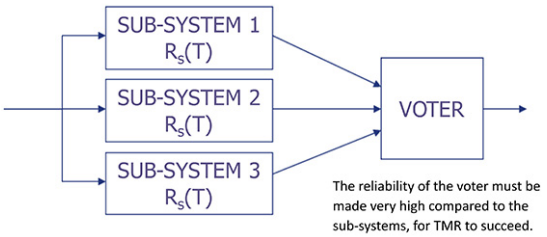


Figura 53: Arquitectura TMR. La imagen fue extraida de [28].

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 costos 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 5. ***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][27]. 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 54.

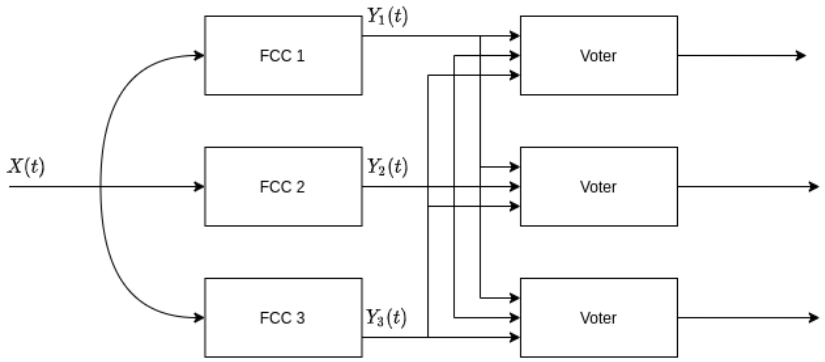


Figura 54: Arquitectura TMR con redundancia en los elementos votantes.

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 55.

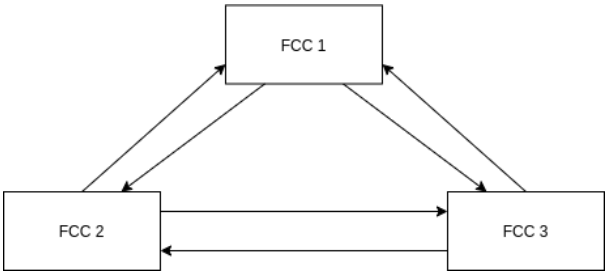


Figura 55: Arquitectura de redundancia trple, donde los bloques votantes son las mismas FCCs. Los votantes se encuentran integrados dentro de cada FCC.

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 [30]. 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 [30]. Otros trabajos que tratan el mismo problema son [31] y [32]. 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 secenario 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.

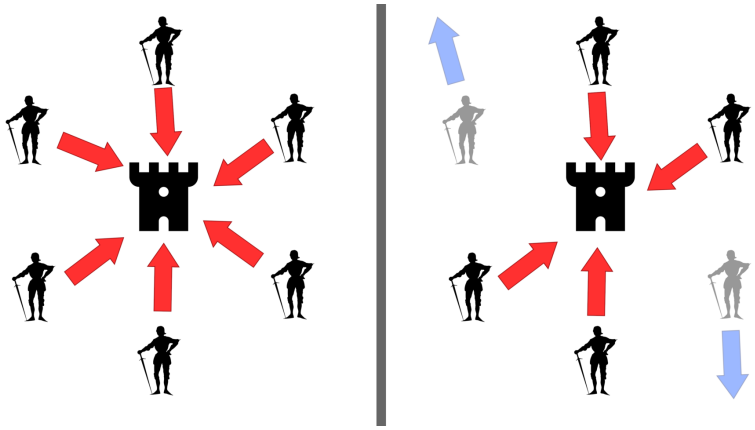


Figura 56: La situación que se presenta, donde los generales deben tomar una decisión común. La figura de la derecha muestra la situación donde algunos generales atacan mientras que los generales traidores no lo hacen. La imagen se extrajo de [62].

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 informarlse 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 opinón 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:

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

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

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 57 se muestra un caso para 4 generales y 1 traidor. El General 1 es el traidor y le envía información diferente a cada general.

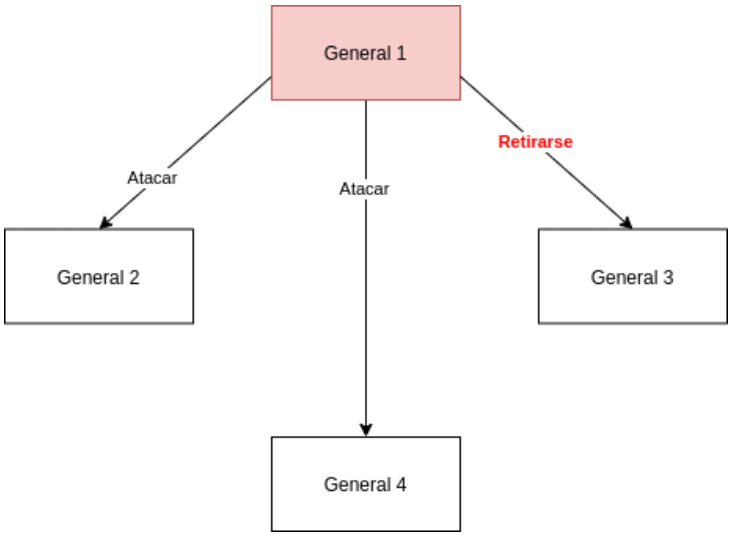


Figura 57: El general 1 es un traidor y le envía información conflictiva a los demás generales.

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 difernte a sus pares. Para resolver esto, el algoritmo plantea realizar un segundo intercambio de mensajes como el de la figura 58.

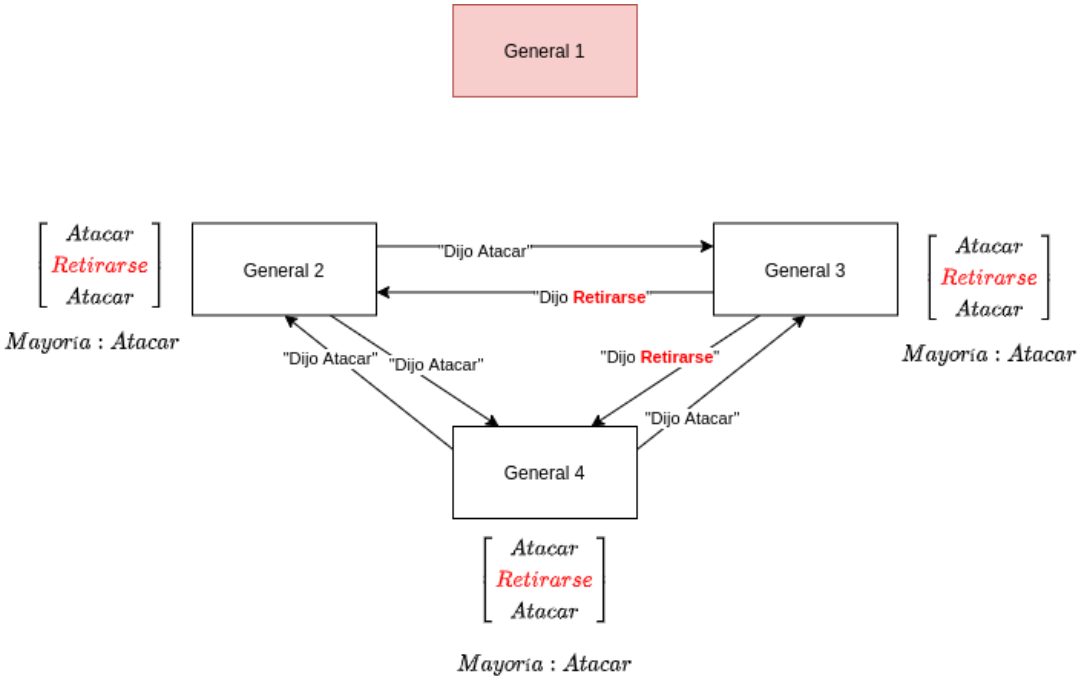


Figura 58: Se produce un intercambio entre los demás generales, para ponerse de acuerdo respecto de si el General 1 dijo “Atacar” o “Retirarse”.

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 [32], 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.

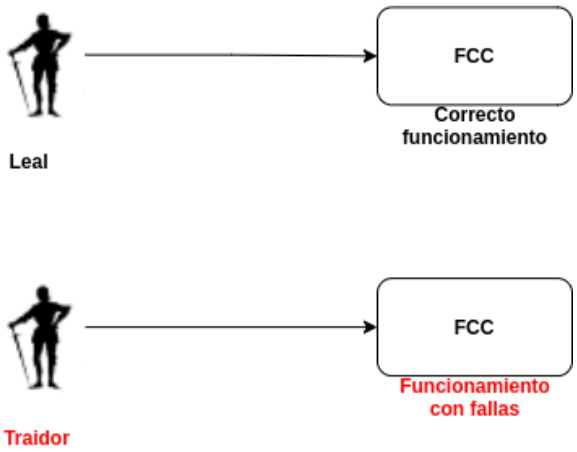


Figura 59: En el problema, un general leal representa un nodo, en este caso una computadora de vuelo, que funciona correctamente. Un General traidor es equivalente a una computadora de vuelo que presenta fallas.

Los generales traidores representan a las computadoras de vuelo que presentan fallas. En [32] 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

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] Vinod B Prasad. «Fault tolerant digital systems». En: *IEEE Potentials* 8.1 (1989), págs. 17-21.

[23] 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.

[24] Hermann Kopetz y Günther Bauer. «The time-triggered architecture». En: *Proceedings of the IEEE* 91.1 (2003), págs. 112-126.

[25] 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.

[26] Federico Fidencio Solano Pérez. «Development of a Redundancy System for Autopilots». 2019.

[27] 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.

[28] *Triple Modular Redundancy*. URL: https://www.layerzero.com/Innovations/Industry-Firsts/Triple-Modular-Redundancy.html.

[29] 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.

[30] Leslie Lamport, Robert Shostak y Marshall Pease. «The Byzantine generals problem». En: *Concurrency: the works of leslie lamport*. 2019, págs. 203-226.

[31] 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.

[32] 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.

[33] CAN Specification. «Bosch». En: *Robert Bosch GmbH, Postfach* 50 (1991), pág. 75.

[34] *Introduction to the Controller Area Network (CAN)*. SLOA101B. Rev. B. Texas Instruments. Mayo de 2016.

[35] *Migration guide from STM32F7 Series to STMH74x/75x, STM32H72x/73x and STMH7A3/7Bx devices*. STMicroelectronics, ago. de 2022.

[36] *Product Longevity - STMicroelectronics*. URL: https://www.st.com/content/st_com/en/about/quality-and-reliability/product-longevity.html#10-year-longevity.

[37] *ICM-42688-P | TDK InvenSense*. Oct. de 2023. URL: https://invensense.tdk.com/products/motion-tracking/6-axis/icm-42688-p/.

[38] Kai Zhang. «Sensing and control of mems accelerometers using Kalman filter». En: (2010).

[39] 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.

[40] *How to measure absolute pressure using piezoresistive sensing elements*. AMSYS. Jul. de 2009.

[41] Avnet. *MEMS pressure sensors*. URL: https://www.avnet.com/wps/portal/abacus/solutions/technologies/sensors/pressure-sensors/core-technologies/mems/ (visitado 31-10-2023).

[42] *Choosing the Right Pressure Sensor*. AN-201610-PL38-01. Infineon. 2016.

[43] 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).

[44] *Road vehicles — Controller area network (CAN) — Part 4: Time-triggeredcommunication*. Standard. International Organization for Standardization, ago. de 2004.

[45] *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 itf, cam and LCD.* STMicroelectronics, feb. de 2016. URL: https://www.st.com/en/microcontrollers-microprocessors/stm32f746zg.html#documentation.

[46] *SN65HVD23x 3.3-V CAN Bus Transceivers.* Abr. de 2018. URL: https://www.ti.com/product/es-mx/SN65HVD230.

[47] *Connector Pin Assignment Recommendations.* CiA 106. CAN in Automation. Jun. de 2022.

[48] *DroneCAN.* URL: https://dronecan.github.io/ (visitado 11-09-2023).

[49] *Automotive Compliant 1A Low Dropout Positive Regulator with Fixed and Ajdustable Outputs.* Oct. de 2016. URL: https://www.diodes.com/assets/Datasheets/ZLDO1117Q.pdf.

[50] *Basics of Low-Dropout (LDO) Regulator ICs.* TOSHIBA. Mar. de 2021.

[51] *Technical Review of Low Dropout Voltage Regulator Operation and Performance.* Texas Instruments. Ago. de 1999.

[52] *AN-1482 LDO Regulator Stability Using Ceramic Output Capacitors.* Texas Instruments. Abr. de 2013.

[53] *Holybro PM02 V3 Power Module.* URL: https://docs.px4.io/main/en/power_module/holybro_pm02.html (visitado 13-09-2023).

[54] *PCB Design Guidelines For ICM-40607x, ICM-40608, ICM-42xxx, ICM-43xxx and ICM-45xxx Products.* AN-000262. TDK Invensense. Ene. de 2021.

[55] *Surface Mounting Guidelines for MEMS Sensors in a QFPN Package.* TN0019. ST. Mar. de 2020.

[56] *Soldering Guidelines for MEMS Inertial Sensors.* APP 5604. Maxim Integrated. Mar. de 2013.

[57] Hermann Kopetz. «The time-triggered model of computation». En: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE. 1998, págs. 168-177.

[58] Michael J Pont. *Patterns for time-triggered embedded systems.* TTE System, Ltd, 2008.

[59] Gabriel Leen y Donal Heffernan. «TTCAN: a new time-triggered controller area network». En: *Microprocessors and Microsystems* 26.2 (2002), págs. 77-94.

[60] Emmanuelle Anceaume e Isabelle Puaut. «Performance evaluation of clock synchronization algorithms». Tesis doct. INRIA, 1998.

[61] Eloy Martins de Oliveira Junior y Marcelo Lopes de Oliveira e Souza. «An Overview of Clock Synchronization Algorithms and their Uses in Aerospace and Automotive Systems». En: (2013).

[62] Wikipedia contributors. «Byzantine fault». En: *Wikipedia* (jul. de 2023). URL: https://en.wikipedia.org/wiki/Byzantine_fault#.