



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:

Claudio POSE
cldpose@fi.uba.ar

Co-Director:

Leonardo GARBEROGLIO
lgarberoglio@frsn.utn.edu.ar

COMPLETAR FECHA

Índice

1. Objetivo	3
2. Introducción	4
3. Análisis de Sistemas Críticos y Tolerancia a Fallas	5
3.1. Caso de Ejemplo: Sistema de Control de Vuelo en Aviones	5
3.1.1. Bus de Comunicaciones	7
3.1.2. Comparación de Resultados y Tolerancia a Fallas	7
3.2. Comparación con Sistemas de Control de Vuelo en UAVs	8
3.2.1. Computadoras de Vuelo Comerciales	8
3.2.2. Casos de Trabajos con Componentes COTS	9
4. Diseño y Construcción de la Computadora de Vuelo	13
4.1. Funcionalidades y Diagrama en Bloques	13
4.2. Criterios Generales Para la Selección de Componentes	14
4.2.1. Uso de Componentes de Grado Automotriz	14
4.2.2. Requerimientos de Conectores	14
4.3. Circuitos Implementados	14
4.3.1. Microcontrolador	14
4.3.2. Sensor IMU	14
4.3.3. Barómetro	19
4.3.4. Magnetómetro	22
4.3.5. Interfaz de Comunicación CAN	22
4.3.6. Circuito de Alimentación	25
4.3.7. Micro SD	28
4.3.8. Interfaz USB	28
4.3.9. Micro Switch	28
4.3.10. LEDs Indicadores	28
4.4. Desarrollo del PCB	28
4.4.1. Requerimientos de Manufacturabilidad	28
4.4.2. Requerimientos de layout del sensor IMU	28
4.4.3. Layout del Regulador Lineal	29
5. Demostración del Funcionamiento de la Computadora de Vuelo	30
6. Diseño Tolerante a Fallas de Hardware RE ACOMODAR EN OTRA SECCIÓN	31
6.1. Introducción al Análisis de Tolerancia a Fallas	31
6.2. Causales de Fallas de Hardware y Modelo de Fallas Arbitrarias	32
6.3. Tolerancia a Fallas a Partir de Redundancias	32
6.3.1. Redundancia Doble	33
6.3.2. Redundancia Triple	34
6.4. Algunos Requerimientos de un Sistema Redundante	35
6.4.1. Sincronismo de los Nodos	35
6.4.2. Consenso	36
6.5. Redundancia Cuádruple: <i>The Byzantine Generals Problem</i>	38
6.5.1. Presentación del Problema	38
6.5.2. Solución al Problema	39
6.5.3. Relación del Problema con la Tolerancia a Fallas	41
7. Arquitectura de Redundancia Propuesta RE ACOMODAR EN OTRA SECCIÓN	43
7.1. Simplificación del Problema de Tolerancia a Fallas Arbitrarias de Hardware	43
7.1.1. Consenso	44
7.1.2. Sincronismo de los nodos	45
7.2. Arquitectura Del Sistema: <i>The Time-Triggered Architecture</i>	46
7.2.1. Bus de Comunicaciones	48
7.2.2. Nodos	49
7.2.3. Scheduler	49

7.2.4. Global Time y Sincronización	49
8. Conclusiones	52
9. Agradecimientos	53
Apéndices	54
Apéndice A: Circuito Esquemático	54
Referencias	55

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 ciertas fallas de hardware que puedan ocurrir en pleno vuelo. Lo que se busca, es que estas fallas no impacten en la misión del vehículo y que puedan ser detectadas lo antes posible para tomar una acción.

En primera medida, se hace un análisis e investigación acerca del estado del arte, para vehículos aéreos no tripulados de carácter comercial, principalmente drones. El objetivo es conocer los mecanismos de seguridad que se implementan 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 vehículos aéreos no tripulados, principalmente drones.

Lo siguiente es el desarrollo de una computadora de vuelo. Esto comprende la definición de los requerimientos de la misma, principalmente de hardware en cuanto a sensores, conectores y funcionalidades deseadas. A partir de estos, se hace una investigación de la variedad de componentes disponibles. Luego, se pasa 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 abordar la tolerancia a fallas de hardware de la computadora de vuelo, se plantea utilizar técnicas que involucren la redundancia, tanto de hardware como de software. Para ello, se lleva a cabo una investigación de las técnicas comúnmente utilizadas en el sector aeronáutico para tolerancia de fallas. Finalmente, se define un esquema y una arquitectura a utilizar como mecanismo de tolerancia a fallas.

Para demostrar los resultados, se presentan resultados de pruebas de control de un motor en una arquitectura redundante, sobre la cual se simula la manifestación de distintos tipos de fallas. Se presentan los resultados y la respuesta del sistema.

2. Introducción

COMPLETAR

3. Análisis de Sistemas Críticos y Tolerancia a Fallas

Un sistema crítico (en inglés *safety-critical system*) es aquel en el que un fracaso puede traer consigo consecuencias catastróficas, como daños graves a personas, pérdida de vidas o daños importantes al medio ambiente [1, p. 271]. Por su definición, un sistema puede caracterizarse como crítico o no crítico antes de su implementación, ya que lo que determina esto es simplemente el uso que se le dará. Algunas áreas típicas donde se pueden encontrar estos sistemas son aviónica, equipamiento médico y energía nuclear.

3.1. Caso de Ejemplo: Sistema 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 [2], con el objetivo de aliviar la carga del avión y mejorar su rendimiento en cuanto al consumo de combustible. Su nombre deriva del hecho de que la acción de control del piloto no es directa sobre el avión, sino que es una computadora la que la ejecuta. Todas las señales de sensores y de comandos son transmitidas eléctricamente desde y hacia una computadora de vuelo.

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

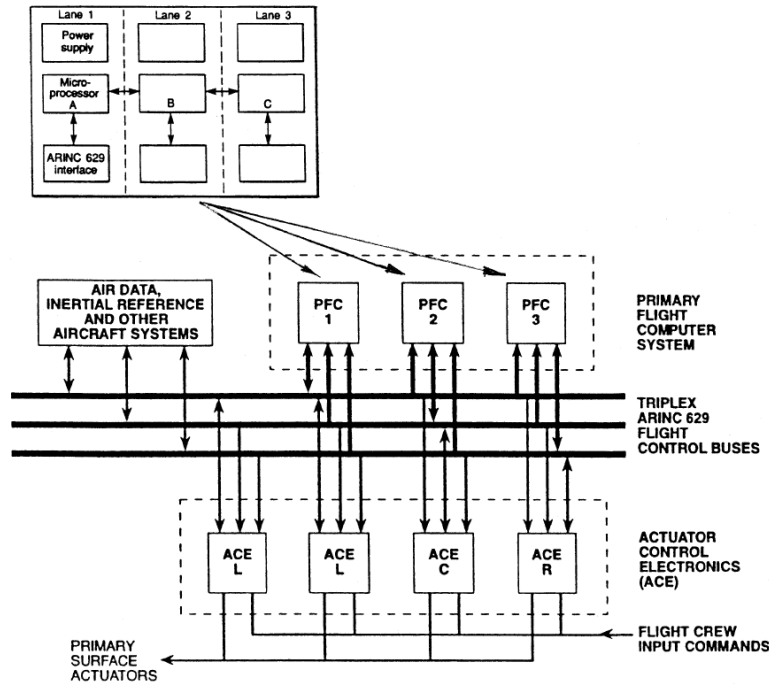


Figura 1: Arquitectura del sistema principal de vuelo, conformado por varias computadoras en una configuración redundante. La imagen se extrajo de [3].

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 [4] 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 $10^{-9}/h$ [3, p. 217]. Este valor es tan bajo que incluso es imposible de verificar de forma experimental, ya que habría que ejecutar el sistema durante 10^9 horas aproximadamente. La probabilidad de falla de los semiconductores no alcanza este valor [1, p. 272]. Este es el motivo por el cual se incluyen bloques redundantes en los sistemas de control de vuelo. Por ejemplo, el hecho de que cada PFC tenga 3 procesadores de distintos fabricantes permite eliminar problemas que sean propios del componente. A su vez, el hecho de que cada procesador tenga un software distinto desarrollado por un grupo de personas distinto permite que las fallas que estos puedan tener sean eliminadas a tiempo.

ACÁ AGREGAR UNA CUENTA SÚPER FÁCIL CON UN SISTEMA CON REDUDANCIAS EN PARALELO Y CON LA PROBABILIDAD DE FALLA EXPONENCIAL.

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

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

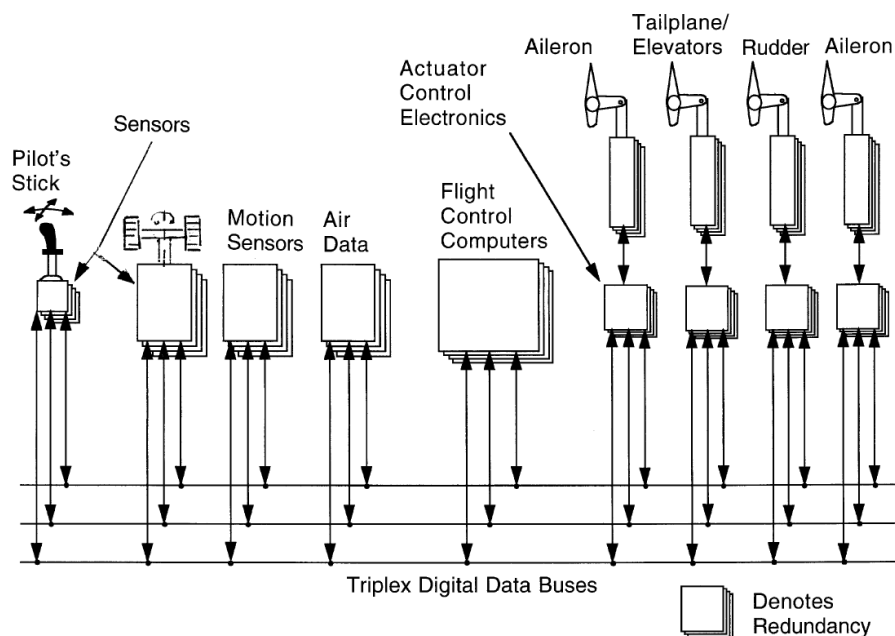


Figura 2: Se muestra de manera general las conexiones entre los distintos elementos del sistema de control de vuelo típico en aviones. La imagen se extrajo de [3].

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.

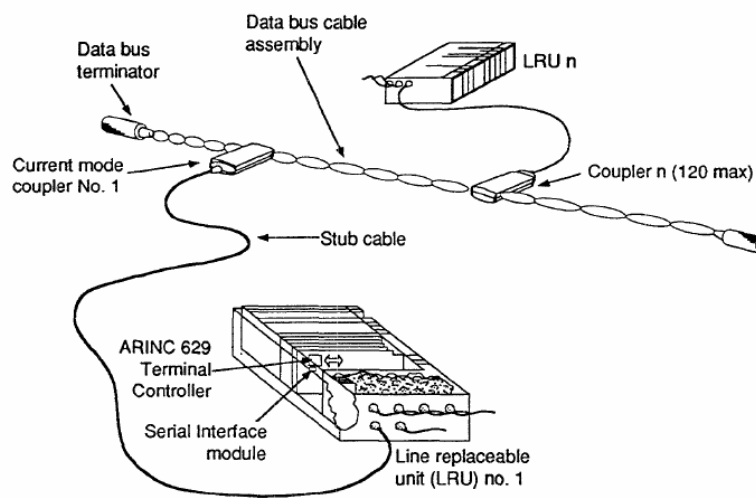


Figura 3: Se muestra un bus ARINC 629 con 2 nodos conectados. Este consiste en un par trenzado, con terminaciones en los extremos para evitar reflexiones. La conexión de los nodos al bus no es directa si no que se utilizan acopladores. La imagen se extrajo de [4].

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 acutación.

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 [3, 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 .

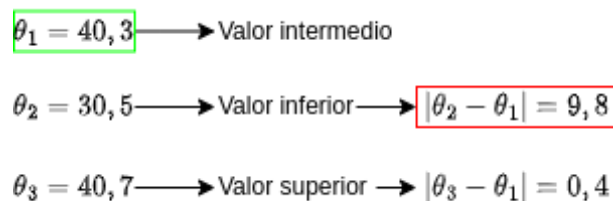


Figura 4: La comparación entre 3 sensores da como resultado que el sensor 2 presentó una falla. En consecuencia deberá tomarse una acción, por ejemplo ignorar al sensor en próximas mediciones o informar al piloto.

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

3.2. Comparación con 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 pueden trasladar cientos de personas y realizar vuelos de muchas horas. En un vehículo aéreo no tripulado, no hay tripulación ni piloto a bordo del vehículo, por lo que suelen estar contruidos con otros requerimientos de seguridad más laxos. Para UAVs de uso militar, la probabilidad de fracaso se encuentra en el orden de $10^{-5}/h$ [5][3, 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 [6]. 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.

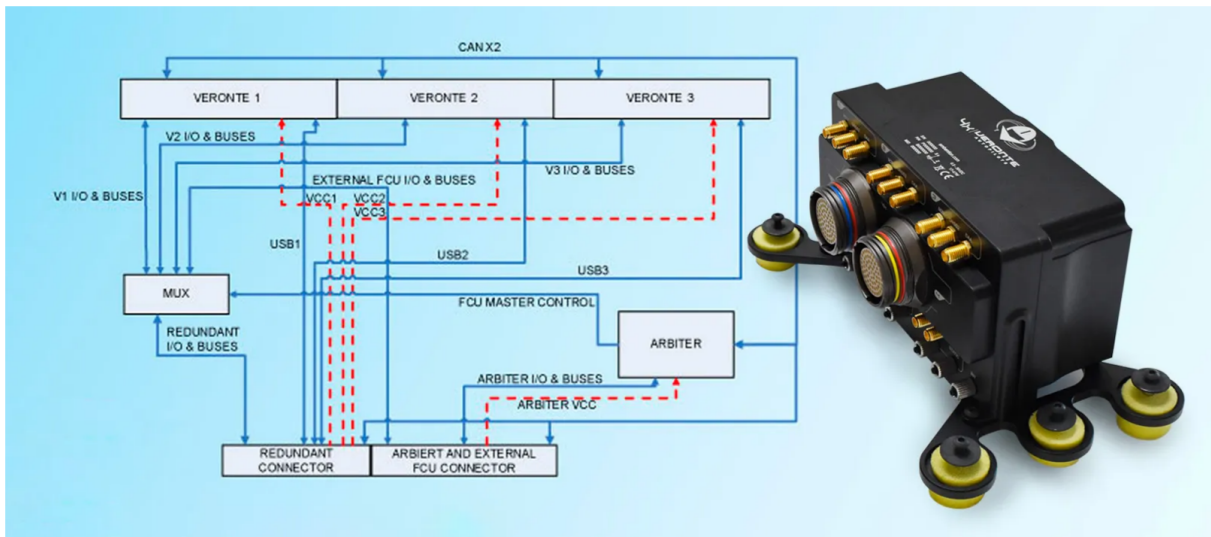


Figura 5: Diagrama en bloques del autopiloto Veronte de la empresa Embention. La imagen se extrajo de [6].

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 [7]. 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 [8]. 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 [9] y [10] 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 [9] se muestran los resultados, en [10] 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.

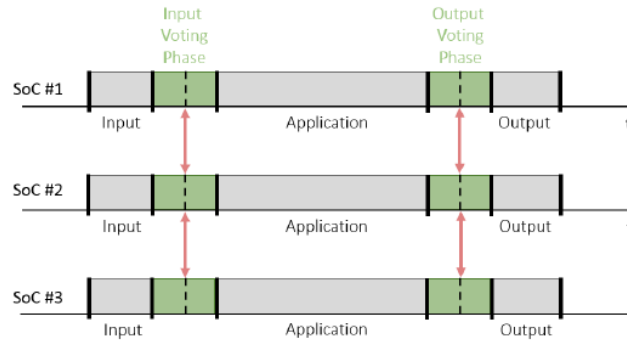


Figura 6: Imagen que demuestra la sincronización entre 3 microcontroladores que realizan las mismas tareas en paralelo. La imagen se extrajo de [10].

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 [9] 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 probabilidad 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.

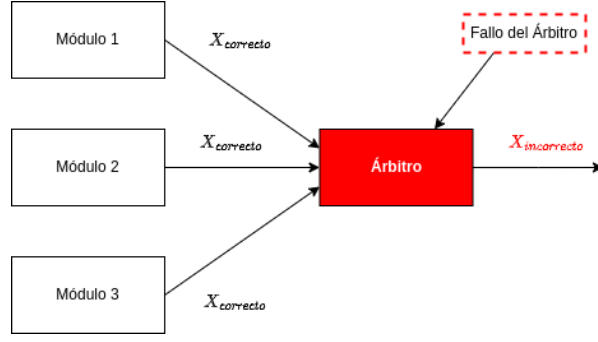


Figura 7: A pesar de que los 3 módulos redundantes funcionen correctamente, una falla del árbitro se traduce directamente en un error en el sistema. Esto lo vuelve un punto singular de falla.

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 [5] 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 [11]. 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 [12] 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.

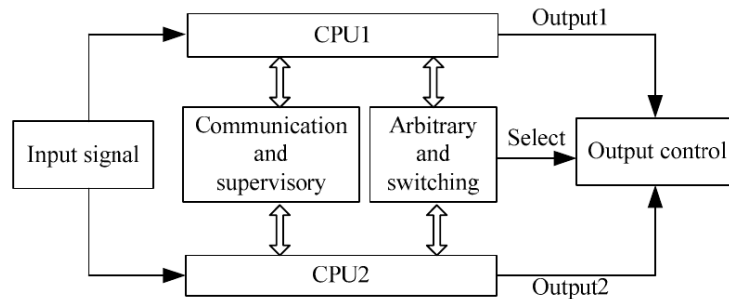


Figura 8: Diagrama en bloques del sistema de redundancia doble. Las CPU1 y CPU2 comparan sus resultados y envían sus salidas al bloque *Output control*. A través de un bloque árbitro se selecciona a cuál de las dos CPU será la que controle los actuadores. La imagen se extrajo de [12].

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 [13] y [14] pueden encontrarse otros 2 trabajos más que utilizan la sincronización entre los nodos redundantes. El primero de ellos trabaja con redundancia triple y un árbitro que selecciona cuál de los nodos controla los actuadores. El segundo también trabaja con redundancia triple, pero no utiliza un árbitro sino que las tres computadoras realizan la votación y cada una de ellas envía un mensaje a un nodo que se encuentra en el mismo bus y se encarga de controlar el actuador. En la figura 9 se muestra el diagrama en bloques.

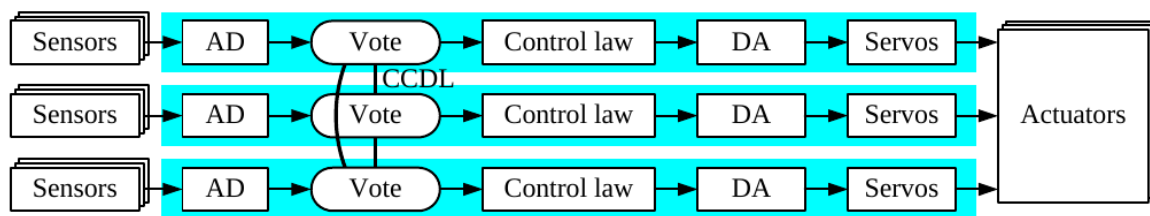


Figura 9: Diagrama en bloques del sistema redundante de [14]. Cada uno de los nodos tiene sus propios sensores, la única comunicación que se realiza a través del bus CCDL (*Cross-Communication Data Link*) es para realizar la comparación de resultados y la votación.

En [15] 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 [16], conectadas a una tercera computadora de vuelo central implementada con una Raspberry Pi que funciona como árbitro. En la figura 10 se muestra un diagrama en bloques de esta arquitectura. La técnica utilizada consiste en que el árbitro continuamente le pide a ambas computadoras información acerca de su “estado de salud”. A partir de la información recibida de ambas, el árbitro controla unas llaves implementadas como relés de estado sólido que seleccionan cuál de las 2 computadoras será la que controle los motores.

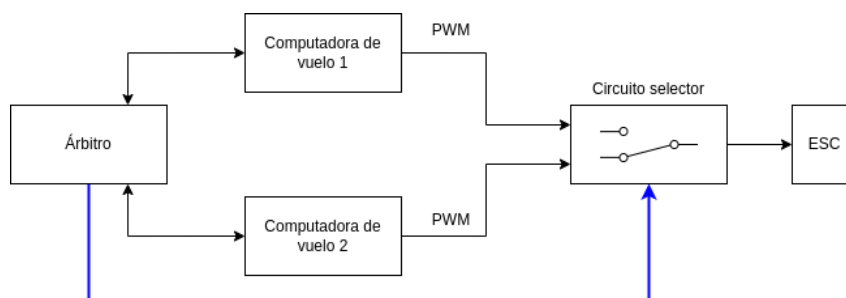


Figura 10: Arquitectura de la computadora de vuelo utilizada en [15]. El árbitro selecciona cuál de las dos señales PWM se utiliza para controlar el *Electronic Speed Controller* (ESC).

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.

COMPLETAR

4. Diseño y Construcción de la Computadora de Vuelo

En esta sección se presentan los criterios tenidos en cuenta para el diseño y la construcción de la computadora de vuelo. Se presentan cuáles son las funcionalidades de la placa y los circuitos que se implementaron para cumplir con estas. Además, se muestra el análisis de la selección de distintos componentes como sensores y circuitos integrados.

4.1. 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.

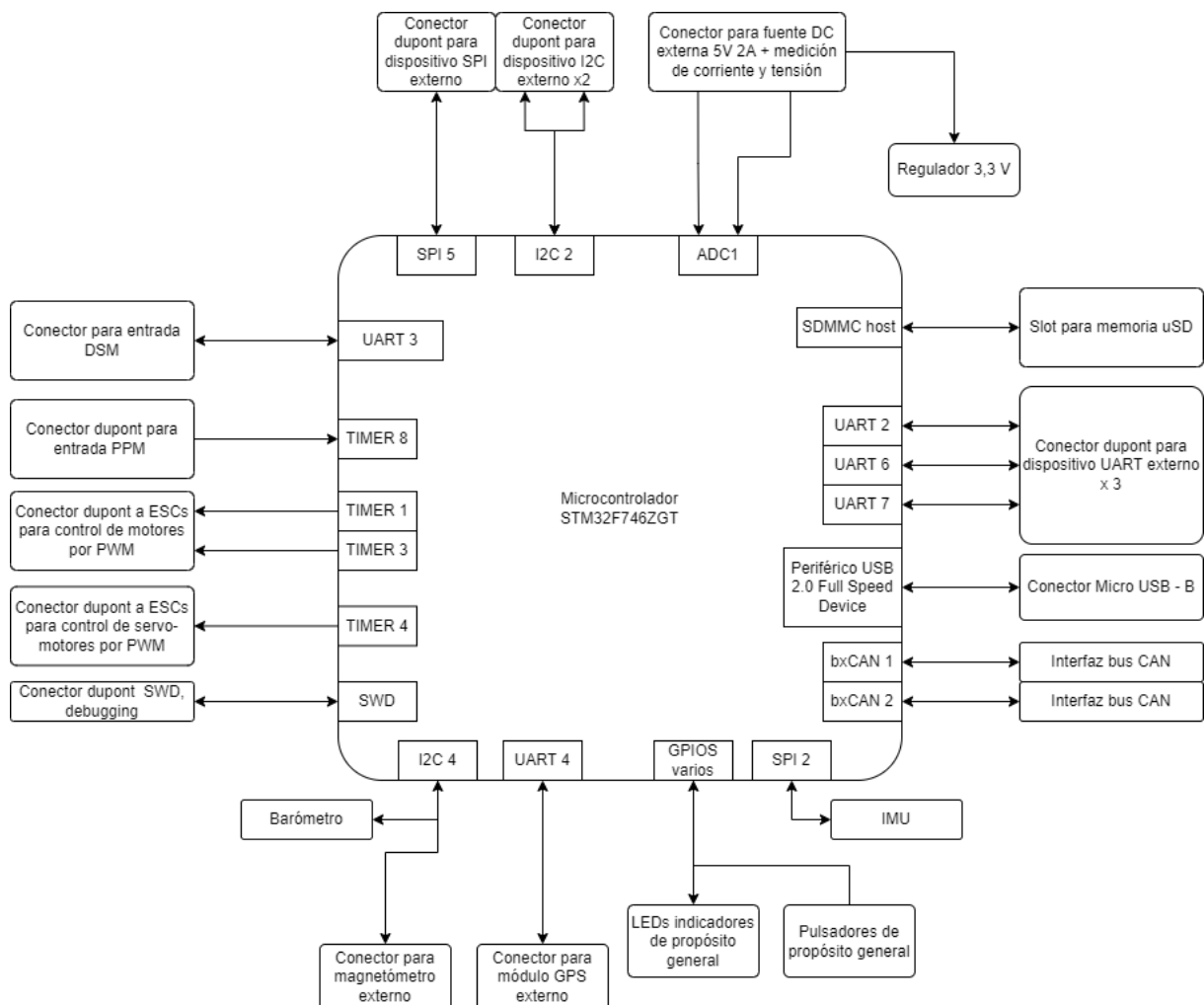


Figura 11: Diagrama en bloques de la computadora de vuelo.

COMPLETAR

4.2. Criterios Generales Para la Selección de Componentes

Para el diseño y la implementación de cada circuito, se tuvieron en cuenta distintas necesidades particulares para cada uno de ellos. A su vez, hay ciertos criterios y que son comunes a todos los circuitos. Estos se mencionan a continuación.

4.2.1. 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

4.2.2. 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 del LAR, con el objetivo de tener compatibilidad con distintos módulos que son comúnmente utilizados con otras computadoras de vuelo que fueron desarrolladas en el laboratorio.

4.3. Circuitos Implementados

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 criterios particulares tenidos en cuenta para cada circuito.

4.3.1. Microcontrolador

COMPLETAR

4.3.2. 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. 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 12.

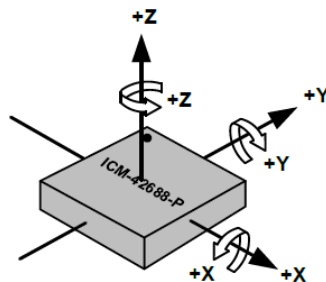


Figura 12: Todas las mediciones que entrega el sensor son reactivas a una terna solidaria a este. La imagen se extrajo de [17].

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 13, 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.

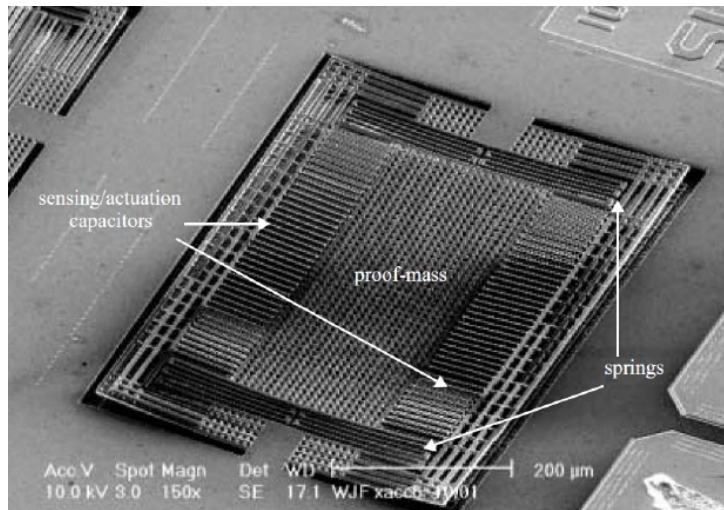


Figura 13: Fotografía tomada de un acelerómetro construido con tecnología MEMS. La imagen se extrajo de [18].

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 [18], como se muestra en la figura 14.

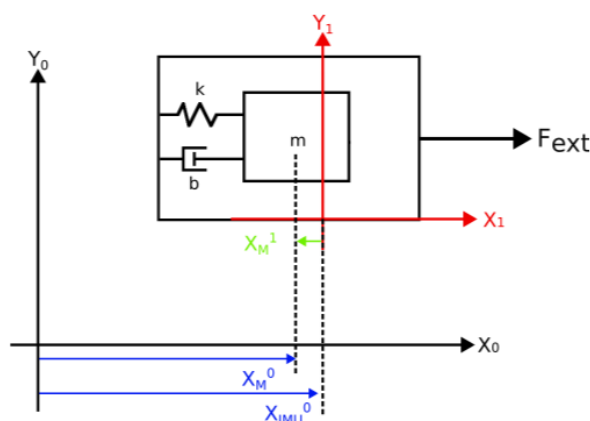


Figura 14: Sistema mecánico simplificado del acelerómetro.

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 $q_1 = X_{IMU}^0$ y $q_2 = X_M^1$. Sin considerar efectos de la gravedad, se llega a la ecuación (1). 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.

$$X_M'' + \frac{b}{m} X_M' + \frac{k}{m} X_M^1 = -X_{IMU}'' \quad (1)$$

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 (2).

$$X_M'' \approx 0 \quad (2a)$$

$$X_M' \approx 0 \quad (2b)$$

$$\frac{k}{m} X_M^1 \approx -X_{IMU}'' \quad (2c)$$

El desplazamiento de la masa produce una variación de la capacidad entre esta y el sustrato. Esta capacidad es utilizada para medir una variación de tensión [18]. El circuito medido puede modelarse como en la figura 15.

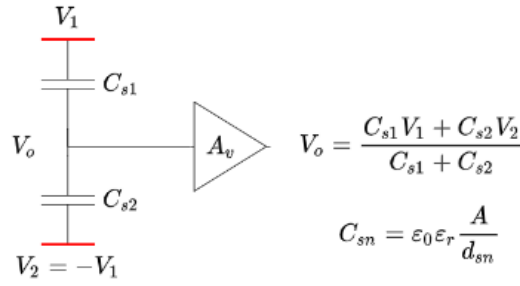


Figura 15: Circuito equivalente que mide el desplazamiento de la masa. Las tensiones V_1 y V_2 representan señales de tensión aplicadas por un circuito externo. El movimiento de la masa modifica la capacidad y por ende la tensión medida.

Se puede resolver este circuito y llegar a que existe una relación lineal entre la tensión V_o y el desplazamiento de la masa Δd , ecuación (3). En [18] 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.

$$V_o = \frac{\Delta d}{d_0} V_1 \quad (3)$$

COMPLETAR UN ANÁLISIS SIMILAR PARA EL GIRÓSCOPO

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 16 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		<i>g</i>	2
	ACCEL_FS_SEL=1		±8		<i>g</i>	2
	ACCEL_FS_SEL=2		±4		<i>g</i>	2
	ACCEL_FS_SEL=3		±2		<i>g</i>	2
ADC Word Length	Output in two's complement format		16		bits	2, 5
Sensitivity Scale Factor	ACCEL_FS_SEL=0		2,048		LSB/ <i>g</i>	2
	ACCEL_FS_SEL=1		4,096		LSB/ <i>g</i>	2
	ACCEL_FS_SEL=2		8,192		LSB/ <i>g</i>	2
	ACCEL_FS_SEL=3		16,384		LSB/ <i>g</i>	2

Figura 16: Extracto de la hoja de datos del sensor ICM42688p. Se muestra parte de las especificaciones para los acelerómetros.

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

Para entender los parámetros, *scale factor error*, *offset* y *noise* se plantea un modelo sencillo de medición, tanto para acelerómetros como para giróscopos [19]. Este se presenta en la ecuación (4), donde S es el *scale factor error*, $\omega_b(t)$ es el *offset* el cual es variable con el tiempo, $\eta \sim \mathcal{N}(0, \sigma^2)$, ω_m es el valor medido y ω sería la velocidad angular verdadera para el giróscopo.

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

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

Para tener un criterio de selección del sensor IMU, se siguió el análisis planteado en [19]. Este paper presenta un análisis de los parámetros de los acelerómetros y gróscopos 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 1 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
$b_{accel}a(t)$	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 1: 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 estos parámetros. Solamente una 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 selecciona 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 2 se muestra una comparación entre el sensor anterior ICM20602 y el sensor seleccionado ICM42688.

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

Tabla 2: 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. La interfaz SPI permite obtener velocidades de transferencia más altas que utilizando otras interfaces como I2C. 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, el maestro, cuando este último le solicite la información. La IMU genera lecturas de sus acelerómetros y giróscopos de manera periódica. De manera de que la IMU pueda informar al microcontrolador el momento en el que generó una nueva lectura, el sensor dispone de una salida digital que puede conectarse a una entrada digital del microcontrolador. De esta forma, cuando el microcontrolador detecta un cambio de nivel en esa entrada digital, procede a pedirle el dato generado a través de la comunicación SPI. En la figura 17 se muestra un esquema de la conexión entre el controlador y el sensor IMU.

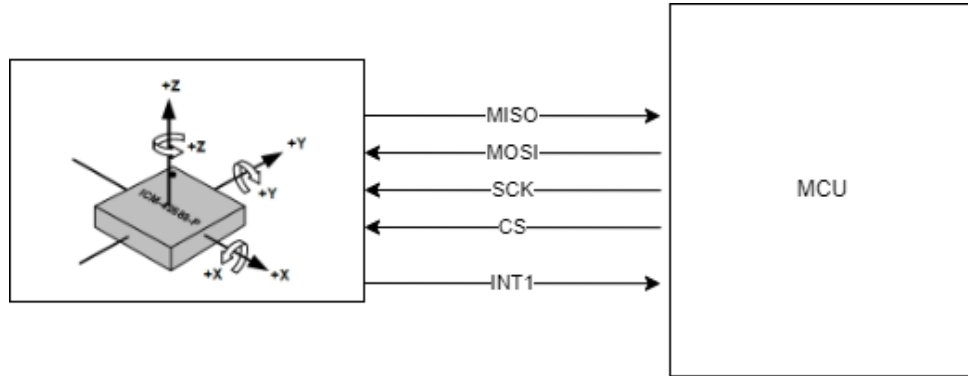


Figura 17: Líneas de comunicación entre la IMU y el microcontrolador.

4.3.3. Barómetro

Al igual que la IMU, el barómetro que se utiliza corresponde a un sistema MEMS. Este sensor se utiliza para estimar la altura del vehículo respecto del suelo, a partir de mediciones de presión. 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. A través de un proceso llamado *anodic bonding* [20][21] se construye esta cavidad dentro del chip, la cual se encuentra sellada a una presión muy baja, en comparación con las presiones que se esperan medir utilizando el componente. Para medir la presión, se coloca una membrana sobre la cavidad. En la figura 18 se puede apreciar el efecto de la presión externa sobre la membrana, la presión atmosférica. Sobre las zonas de color violeta, se colocan resistores de efecto piezoresistivo. En consecuencia, la deformación de la membrana genera tensión sobre estos, modificando su resistencia.

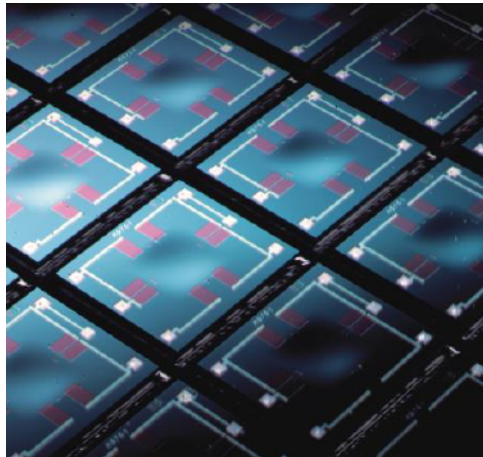


Figura 18: Sensores de presión sobre una oblea de silicio [20].

Los resistores se conectan en configuración puente de Wheatstone, de manera tal de que la presión comprime a dos de los resistores y estira a los otros dos [22]. En (5) se despeja la relación entre la variación de tensión y la variación de la resistencia. Como se observa, la relación es proporcional.

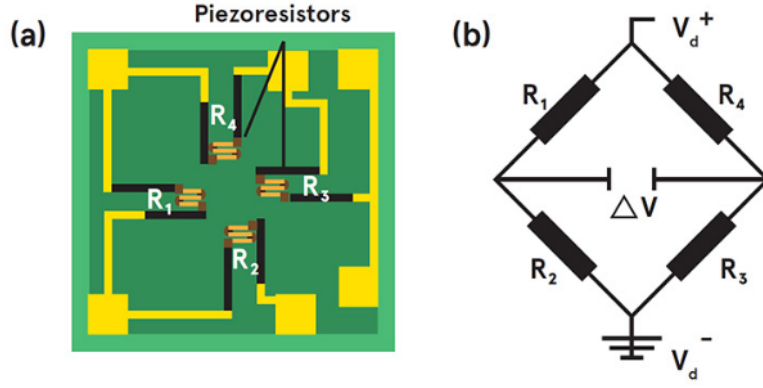


Figura 19: Puente de Wheatstone conformado por los resistores del sensor de presión. La imagen se extrajo de [22].

$$\Delta V = (V_d^+ - V_d^-) \left[\frac{R_2}{R_2 + R_1} - \frac{R_3}{R_3 + R_4} \right] \quad (5a)$$

$$R_1 = R_3 = R - \Delta R \quad (5b)$$

$$R_2 = R_4 = R + \Delta R \quad (5c)$$

$$\Delta V = (V_d^+ - V_d^-) \frac{\Delta R}{R} \quad (5d)$$

En la aplicación del vehículo aéreo, el barómetro se utiliza con el fin de medir la altura del vehículo, respecto de una altura inicial. La forma de medir la altitud a través de la presión es utilizando la ecuación de los gases nobles para el aire [23]. En las ecuaciones (6) se obtiene una expresión para la presión, en función de la densidad del aire, la constantes de los gases y la masa molar del aire.

$$PV = nRT \quad (6a)$$

$$n = \frac{m}{M} \quad (6b)$$

$$PV = \frac{m}{M} RT \quad (6c)$$

$$P = \frac{m}{V} \frac{RT}{M} \quad (6d)$$

$$P = \rho \frac{RT}{M} \quad (6e)$$

Luego, utilizando la condición hidrostática, la presión es la ejercida por la columna de aire [23]. En la condición hidrostática de la ecuación 7 se puede despejar la densidad del aire y reemplazarla en (6e). Finalmente, se obtiene la ecuación diferencial de (8).

$$dp = -\rho g dh \quad (7)$$

$$\frac{dP}{P} = -\frac{gM}{RT} dh \quad (8)$$

Esta ecuación puede integrarse a ambos lados para hallar la relación entre la presión y la altura. Todos los términos de la ecuación (8) son constantes, a excepción de la temperatura. Según el modelo *International Standard Atmosphere* (ISA), se modela la relación entre la temperatura y la altitud, según la ecuación (9). Esta relación es válida solamente hasta la estratósfera.

$$T(h) = T_0 + L h \quad (9a)$$

$$T(h) = 288,15K - h \cdot 6,5 \cdot 10^{-3} m/K \quad (9b)$$

Finalmente, se obtiene una expresión de la presión en función de la altura, ecuación (10).

$$P(h) = P_0 \left[1 + \frac{Lh}{T_0} \right]^{-\frac{Mg}{RL}} \quad (10a)$$

$$P(h) = 1013,25 \text{ hPa} \left[1 - 0,0065 \frac{h}{288,15K} \right]^{5,2561} \quad (10b)$$

Se puede obtener un modelo más simplificado si se asume una temperatura constante, independiente de la altitud. Se reemplaza en (8) y resolviendo la ecuación diferencial se obtiene la ecuación (11).

$$P(h) = P_0 e^{-\frac{gMh}{RT}} \quad (11a)$$

$$P(h) = 1013,25 \text{ hPa} e^{-\frac{h}{8840,2m}} \quad (11b)$$

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

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

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

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

$$P_m = (1 + S)P + P_b(t) + \eta \quad (12)$$

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* [24]. 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 [20]. Por otro lado, el término $S P$ se lo denomina *relative accuracy*. Este hace referencia a mediciones diferenciales de presión. Algunos barómetros MEMS traen consigo una funcionalidad para realizar una compensación de offset. Esto dejaría como parámetro de interés para mediciones de presión a la *relative accuracy*, la cual hace referencia al error introducido para mediciones de variaciones de presión.

El parámetro *solder drift* se refiere al offset que se introduce por el propio proceso de soldadura [24]. 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 3 se muestra una comparación de los distintos barómetros considerados. Sumado a esto, se tuvo en consideración otro aspecto que fue mencionado anteriormente, la longevidad del componente.

	BMP390	BMP581	ICP-20100	LPS22HH	ILPS22QSTR	DPS368
Full scale range [hPa]	300 - 1250	300 - 1250	260 - 1260	260 - 1260	260 - 1260 ; 260 - 4060	300 - 1200
absolute acc [hPa]	$\pm 0,5$	$\pm 0,3$	$\pm 0,2$	$\pm 0,5$	$\pm 0,5$	± 1
realtive acc [hPa]	$\pm 0,03$	$\pm 0,06$	$\pm 0,01$	$\pm 0,025$	$\pm 0,015$	$\pm 0,06$
longevidad	N/A	N/A	N/A	N/A	10 años, enero 2023	N/A
AEC-Q100	No	No	No	No	No	No

Tabla 3: 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 [25]. Finalmente el sensor seleccionado fue este último.

En cuanto al circuito, 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 es más lento que SPI, las mediciones del barómetro no son tan críticas como las de la IMU. Este sensor, a diferencia de la IMU, no cuenta con una línea de interrupción, por lo que los datos deben obtenerse por *polling* de forma periódica. El circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático.

4.3.4. Magnetómetro

COMPLETAR

4.3.5. Interfaz de Comunicación CAN

Como fue mencionado, la computadora de vuelo cuenta con la capacidad de conexión a un bus CAN. La especificación original del protocolo [26] incluye dentro de sus definiciones, la capa física. Cada nodo de un bus CAN se conecta a este a través de 2 cables, los cuales llevan la señal diferencial. Esto se muestra en la figura 20. Todo el bus CAN se compone de 2 cables que llevan los mensajes a todos los nodos de la red. Esto es así debido a que el bus CAN originalmente fue diseñado para utilizarse en automóviles y reemplazar la enorme cantidad de conexiones entre módulos. El hecho de que se trate de una señal diferencial hace que la comunicación sea robusta, 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.

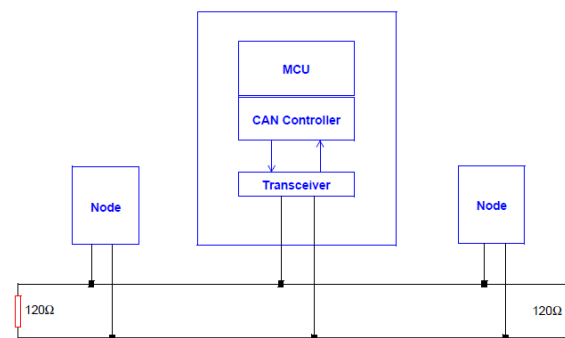


Figura 20: La conexión de un nodo al bus es a través de 2 cables que llevan dos señales, CAN-H y CAN-L. La imagen se extrajo de [27].

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 al mismo bus. Se recomienda que la conexión entre cada nodo y el bus no sea de más de 30 cm. El hecho de poder tener hasta 30 nodos expande las posibilidades de uso del bus, 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.**

La impedancia característica del bus debe ser de $120\ \Omega$. 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 este se desconecta de forma accidental del bus, todas las comunicaciones entre los demás nodos se verán perjudicadas. Este es el motivo por el cual la computadora de vuelo no incluye este resistor en el circuito.

En la figura 20 se muestran 2 elementos que forman parte del nodo, el *transceiver* y el *controller*. El primero de ellos forma parte de la capa física y es un circuito que convierte las señales diferenciales del bus en señales de modo común, que luego son transferidas al elemento *controller*. Este componente sirve como interfaz física con el bus.

El microcontrolador seleccionado para la computadora de vuelo, cuenta con un *controller* embebido, el periférico bxCAN [28]. Este cuenta con dos líneas de comunicación con el transceiver, CAN TX y CAN RX. En la figura 21 se muestra la comunicación entre transceiver, controller y el bus. Cuando se quiere transferir un mensaje a través del bus CAN, el periférico envía un mensaje a través del terminal CAN TX. El transceiver lo recibe y lo convierte en una señal diferencial para inyectarlo en el bus. Cuando el nodo recibe un mensaje del bus CAN, el transceiver es el que interactúa con el bus y genera una señal de modo común, la cual es enviada a través del terminal CAN RX al microcontrolador.

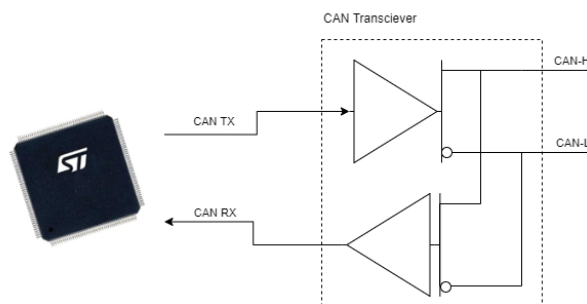


Figura 21: El periférico embebido en el microcontrolador, a través del transceiver, puede interactuar con el bus CAN.

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 transceiver 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 22.

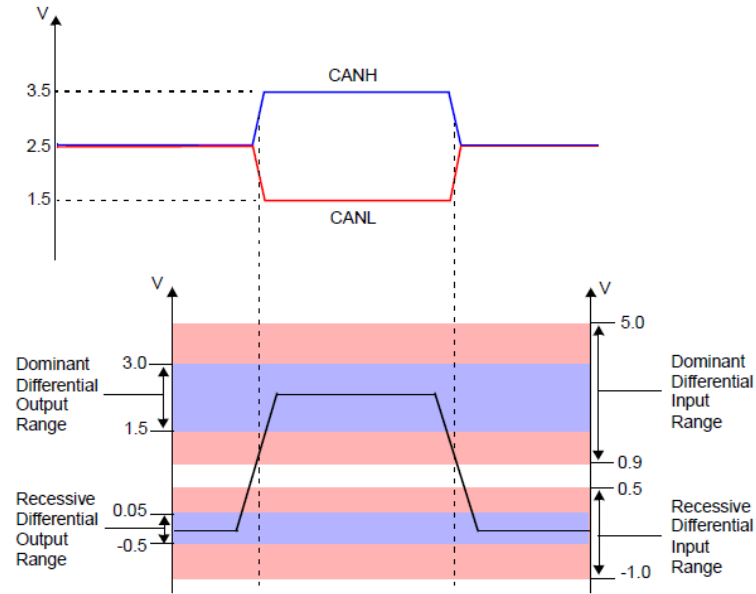


Figura 22: Se muestran los estados recessive y dominant del bus CAN y sus equivalentes lógicos.

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 lo que permite es 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*. Esto es lo que permite implementar el mecanismo de acceso al bus por prioridades.

La interfaz CAN se compone del transceiver, su comunicación con el microcontrolador y el conector. En cuanto al transceiver, se trata de un componente que es ampliamente utilizado en la electrónica automotriz, por lo que hay mucha disponibilidad. Existen transceivers 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 transceiver para esta tensión. El componente seleccionado es el SN65HVD230 de Texas Instruments [29], el cual es compatible con la especificación de capa física de CAN, ISO 11898-2. Este cuenta con una protección por exceso de temperatura, donde el componente pone sus salidas CAN-H y CAN-L en alta impedancia, de manera de no perturbar al resto de los nodos. Por otro lado cuenta con una funcionalidad que permite detectar si el transceiver fue desconectado del bus, fijando un estado alto constante en su salida RX hacia el *controller*, informándole de la situación.

El transceiver seleccionado además cuenta con un terminal 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. Se coloca un resistor de 10 k Ω en el terminal 8 denominado *Rs*. Según la hoja de datos, esto corresponde a un slew-rate de 15 V/ μ s.

La especificación de la capa física de CAN no define un conector. Se buscó seleccionar alguno que no ocupe demasiado espacio al ser montado en el PCB. En [30] 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 [31], 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, los DF-13 de Hirose. Estos conectores son pequeños, por lo que no ocupan demasiado espacio en la placa.

En cuanto al circuito implementado, se tomó como punto de partida las recomendaciones de la hoja

de datos del transceiver SN65HVD230. Este recomienda el agregado de capacitores de desacople en las líneas de alimentación. Además, se agregaron resistores en serie con las líneas de CAN RX y CAN TX, es decir, en las pistas que llevan las señales que van desde el controller al transceiver y viceversa. Esto se hizo como medida preventiva para amortiguar las señales, en caso de que estas presenten sobrepicos en las conmutaciones. A priori se fijan en $0\ \Omega$ y se modificarán de ser necesario. El circuito completo de la interfaz CAN puede encontrarse en el Apéndice A: Circuito Esquemático.

COMPLETAR EL ANÁLISIS DEL MODELO DE PARÁMETROS CONCENTRADOS, PARA LAS PISTAS CAN-TX Y CAN-RX, PARA CALCULAR EL EVENTUAL RESISTOR EN SERIE.

4.3.6. Circuito de Alimentación

Para alimentar el microcontrolador y los demás componentes, se incluye una fuente de alimentación de $3,3\ V$. En las primeras dos versiones de la computadora de vuelo, desarrolladas por el LAR en trabajos anteriores, se integró una fuente conmutada de salida $5\ V$ junto con un regulador lineal de $3,3\ V$. A partir de la tercera versión, esta fuente conmutada se eliminó dejando solamente el regulador lineal de $3,3\ V$ [32]. El motivo principal fue reducir las dimensiones de la placa y simplificar el diseño. Siguiendo esta misma línea, la fuente conmutada tampoco se incluyó en este trabajo. Se incorporó un conector, de manera de poder alimentar la placa utilizando una fuente externa de $5\ V$.

El circuito implementado se comprende del regulador lineal, junto con capacitores de entrada y de salida. Para la selección del componente se buscó un regulador que sea de grado automotriz, por cuestiones de confiabilidad ya mencionadas, pero además se buscó un componente con un encapsulado SOT-223-3, figura 23. En el mercado local puede encontrarse una gran variedad de reguladores con este encapsulado, pensando en la necesidad de modificar el circuito, utilizando otro regulador o en caso de que haya que reemplazarlo.

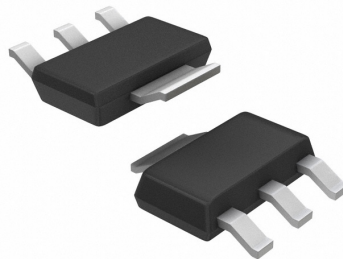


Figura 23: Encapsulado SOT-223-3 seleccionado para el regulador lineal. El terminal de mayor tamaño se encuentra conectado al terminal de tensión de salida, $3,3\ V$.

El chip seleccionado es el ZLDO1117QG33TA de DIODES Incorporated [33], un regulador low-dropout de grado automotriz. 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.

La principal función del capacitor de entrada es la de proveer corriente al regulador, en caso de que ocurra un escalón en la corriente consumida por la carga del regulador [34]. 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. Se seleccionó un capacitor cerámico multicapa de valor $4,7\ \mu F$. Este tipo de capacitores tienen una ESR muy baja, lo que permite que se descarguen rápido, en caso de que tengan que entregar carga para mantener la tensión de entrada en el regulador.

En cuanto al capacitor de salida, este cumple una doble función. Por un lado, aportar a la regulación de línea. Este parámetro es importante teniendo en cuenta que el regulador se utiliza para alimentar un circuito digital, donde aparecen muchas variaciones de corriente. En el caso en el que ocurra un cambio brusco de la corriente de salida del regulador, el capacitor de salida se descargará en parte, para intentar

mantener la tensión de salida en un valor constante. Por otro lado, este capacitor también es utilizado para la compensación del regulador [34]. En la figura 24 se muestra un diagrama en bloques simplificado para un regulador lineal, donde puede verse que se trata de un sistema a lazo cerrado.

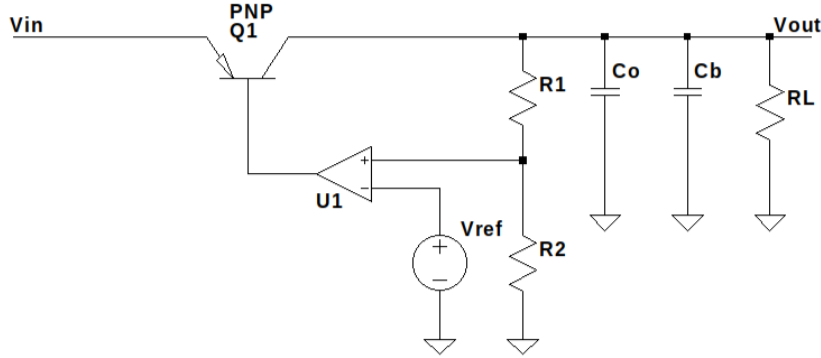


Figura 24: Diagrama en bloques de un regulador lineal. Este se compone de un transistor de paso, un amplificador de error, una tensión de referencia y un bloque realimentador, formado por R_1 y R_2 . En el diagrama además se muestra R_L que simboliza una carga resistiva y dos capacitores C_o y C_b . Estos elementos forman un circuito a lazo cerrado que estabiliza la tensión de salida V_{out} .

Para analizar la ganancia de lazo, se plantea el esquema de realimentación en la figura 25. Para el transistor de paso, se utiliza su modelo de pequeña señal.

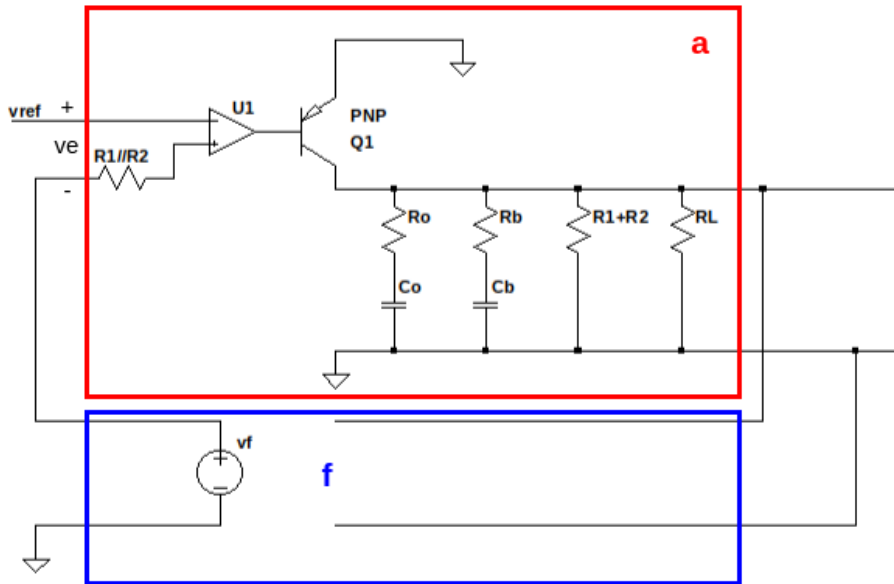


Figura 25: Circuito equivalente para analizar la realimentación del regulador lineal. La tensión v_{in} se pasiva ya que se analiza el circuito en el punto de operación.

Se trata de un esquema que muestrea tensión y suma tensión. El bloque f es tal que cumple con la relación de la ecuación (13b). En cuanto al bloque a , la transferencia corresponde a la ecuación (14b), donde se ha asumido que $R_b \rightarrow 0$ por ser la ESR de un capacitor cerámico multicapa, $R_L \ll r_o$ y $R_L \ll (R_1 + R_2)$. Además, se asume que $R_o \ll R_L$.

$$v_f = f v_o \quad (13a)$$

$$f = \frac{R_2}{R_1 + R_2} \quad (13b)$$

$$v_o = a v_{ref} \quad (14a)$$

$$a = A_v(s) g_m R_L \frac{1 + sC_o R_o}{(1 + sC_o R_L)(1 + sC_b R_o)} \quad (14b)$$

La transferencia $A_v(s)$ corresponde al amplificador de error y se modela como de un polo simple. Juntando los bloques a y f puede obtenerse la ganancia de lazo para analizar la estabilidad, ecuación (15). Esta se compone de 3 polos y 1 cero. Este último junto con uno de los polos dependen directamente del valor R_o , es decir, la ESR del capacitor de salida C_o . En caso de que la ESR sea muy pequeña, se obtiene un sistema con dos polos. Si bien este es estable, su margen de fase es muy pequeño. Esto se traduce en respuestas a transitorios con oscilaciones, por ejemplo causados por variaciones en la corriente de salida del LDO. En la figura 26 se muestra la comparación entre el sistema compensado y sin compensación.

$$af = \frac{R_2}{R_1 + R_2} A_v(s) g_m R_L \frac{1 + sC_o R_o}{(1 + sC_o R_L)(1 + sC_b R_o)} \quad (15)$$

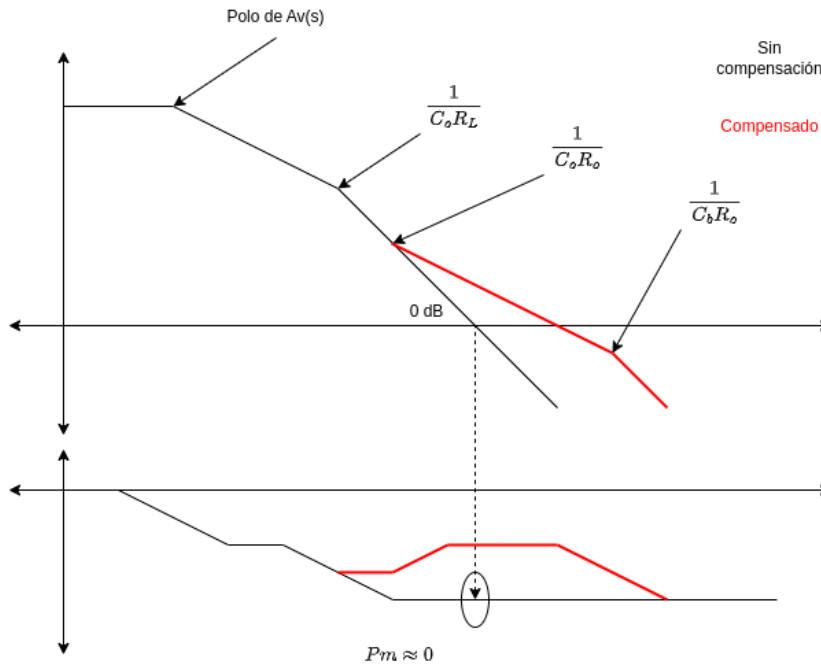


Figura 26: Se muestra un gráfico representativo de la ganancia de lazo de un LDO. En rojo se destaca cómo varía la ganancia de lazo con el agregado de un polo y un cero.

Al agregar los capacitores de salida C_o y C_b , se forma un compensador en adelante que incrementa el margen de fase. De este gráfico se desprende que si la resistencia ESR del capacitor de salida C_o es muy pequeña, luego tanto el polo en $1/(C_b R_o)$ como el cero $1/(C_o R_o)$ se desplazan hacia la derecha en frecuencia y no funcionan como compensador en adelante. Por otro lado, si la ESR es muy grande, el polo y el cero se desplazan a la izquierda y tampoco funcionarán como compensador [35]. De aquí se desprende el hecho de que la elección del valor de ESR debe ser adecuada.

El fabricante recomienda un valor de ESR entre $0,05 \Omega$ y $0,5 \Omega$, de un valor de por lo menos $4,7 \mu F$. Los capacitores MLCC típicamente tienen ESR muy bajas, del orden de unos pocos $m\Omega$. Debido a esto, se selecciona un capacitor de tantalio para el capacitor de salida [36], que suelen exhibir una ESR mayor a la de los MLCC. El componente seleccionado es el T491A106K016ATAUTO de KEMET. Al igual que Murata, este proveedor ofrece curvas de las características del componente.

Como fue ya fue mencionado, se incorpora un conector que permite adosar una fuente externa. El conector se seleccionó pensando en utilizar algunos módulos típicos para drones y vehículos aéreos [37].

Estos módulos, además de las líneas de tensión y de retorno, tienen dos líneas más que entregan información del estado de la batería. Estas son 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.

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.

4.3.7. Micro SD

COMPLETAR

4.3.8. Interfaz USB

COMPLETAR

4.3.9. Micro Switch

COMPLETAR

4.3.10. LEDs Indicadores

COMPLETAR

4.4. Desarrollo del PCB

COMPLETAR

4.4.1. Requerimientos de Manufacturabilidad

COMPLETAR

4.4.2. 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 de las lecturas.

Se siguieron una serie de guías y recomendaciones que tienen como objetivo minimizar el estrés sobre el componente [38] [39] [40]. 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 27 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.

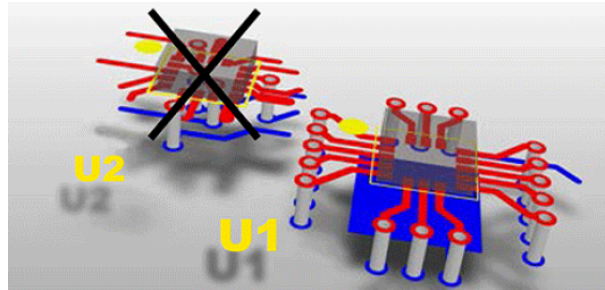


Figura 27: Se muestran dos ejemplos de layout de una IMU. El layout U2 no sigue las recomendaciones mencionadas, mientras que U1 sí. La imagen se extrajo de [40].

4.4.3. Layout del Regulador Lineal

COMPLETAR

5. Demostración del Funcionamiento de la Computadora de Vuelo

COMPLETAR

6. Diseño Tolerante a Fallas de Hardware RE ACOMODAR EN OTRA SECCIÓN

6.1. 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 [41]. 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 [41], 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 (16), y se define como la probabilidad de que el sistema pueda cumplir su función de manera correcta en un intervalo de tiempo $[t_0; t]$, dado que en el instante inicial t_0 el sistema podía hacerlo.

$$R(t) = P(\text{funcionamiento correcto en } t | \text{funcionamiento correcto en } t_0) \quad (16)$$

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 (17). 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 .

$$R(t) = P(\text{no ocurrió una falla en } [t_0; t]) + P(\text{funcionamiento correcto en } t | \text{ocurrió una falla en } [t_0; t]) P(\text{ocurrió una falla en } [t_0; t]) \quad (17)$$

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

$$R(t) = P(\text{no ocurrió una falla en } [t_0; t]) \quad (18)$$

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 [41]. 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 (17), 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.

6.2. Causales de Fallas de Hardware y Modelo de Fallas Arbitrarias

Uno de los métodos para aplicar mecanismos de tolerancia a fallas consiste en hacer un análisis de los posibles modos de falla. Un ejemplo es el del análisis *Failure Modes and Effects Analysis* (FMEA). Este consiste en realizar un análisis exhaustivo de los posibles modos de falla más probables y sus posibles efectos en el sistema. En función de este análisis, se toman medidas para tolerar las fallas más críticas. El objetivo de este tipo de análisis suele ser demostrar ante alguna autoridad certificante, que la confianza del sistema se mantiene por encima de cierto valor. Este tipo de análisis suele consumir mucho tiempo y esfuerzo, lo que se traduce en un mayor costo del desarrollo [42].

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 [43][10][42]. Se define un sistema tolerante a este tipo de fallas.

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

Dado que no se asume un modo de falla del hardware, no se requiere un análisis tan exhaustivo como el mencionado FMEA. Considerando el costo y esfuerzo que lleva realizar un análisis de modos de fallas, el hecho de poder contar con un sistema con las características que aquí se mencionan resulta atractivo para alivianar el trabajo relacionado a la validación del sistema tolerante a fallas en cuestión.

A priori, puede parecer que desarrollar un sistema tolerante a fallas arbitrarias representa un trabajo sumamente complejo. La manera de implementar un sistema tolerante a fallas bizantinas es a través del uso de redundancias. Este resultado se toma a partir de un problema teórico denominado *The Byzantine Generals Problem* [44], el cual se presentará más adelante.

6.3. Tolerancia a Fallas a Partir de Redundancias

La principal técnica de tolerancia a fallas es el uso de redundancias [41][45][42][46]. 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 6.1, 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.

6.3.1. 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.

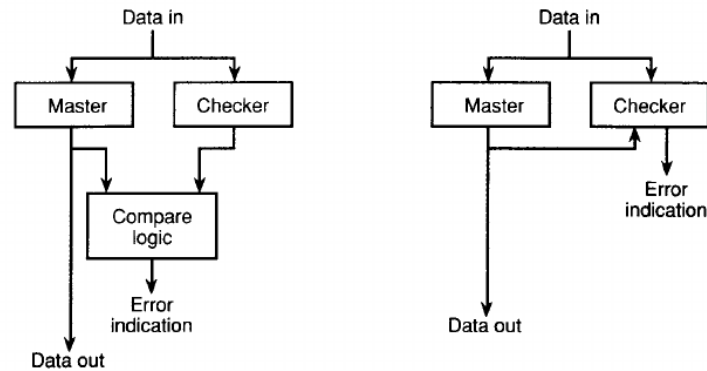
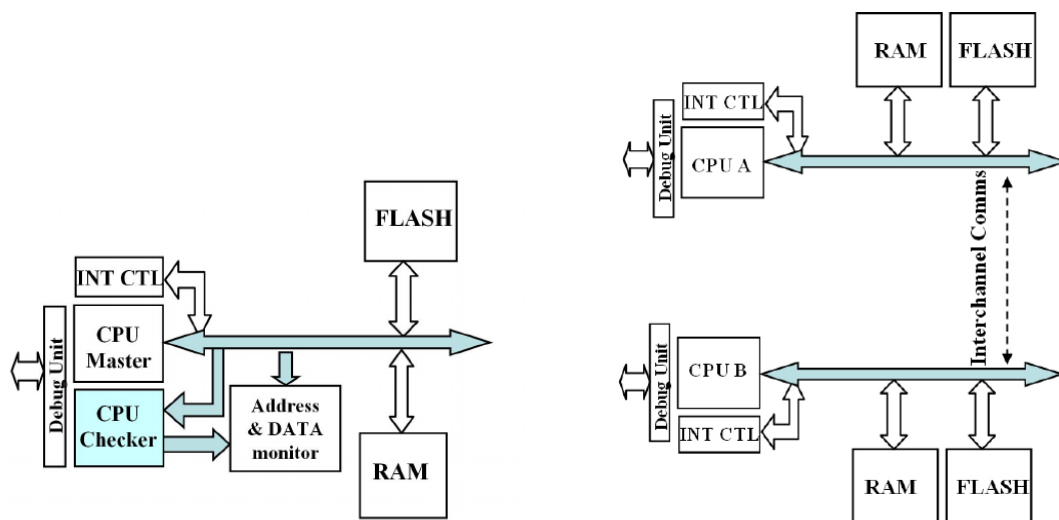


Figura 28: En la figura de la izquierda, dos sistemas ejecutan las mismas operaciones, mientras que otro sistema externo se encarga de comparar las salidas de ambos para detectar errores. En la figura de la derecha, el bloque comparador se encuentra integrado en el sistema *checker*. La imagen fue extraída de [41].

Este tipo de arquitectura permite detectar si ocurrió un error, pero no permite identificar de qué nodo proviene el error. En la figura 28 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*[47]. En esta última, ambos se encuentran sincronizados por estar en el mismo chip y compartir fuente de clock. En la figura 29 se muestra un esquema de ambos casos.



(a) Lockstep dual processor architecture.

(b) Loosely synchronized dual processor architecture.

Figura 29: Se muestran dos casos para un sistema con redundancia doble. La imagen fue extraída de [47].

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 [12] y en [48] se pueden encontrar proyectos de redundancia doble para UAVs.

6.3.2. Redundancia Triple

Esta arquitectura puede encontrarse en la literatura con el nombre *Triple Modular Redundant (TMR) Architecture* [47][41][45][49]. 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 30 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*.

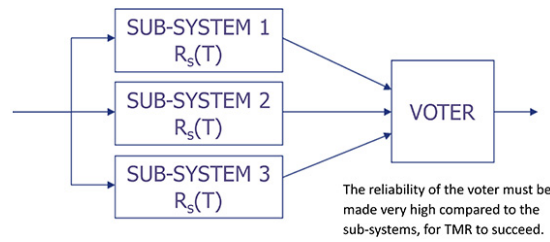


Figura 30: Arquitectura TMR. La imagen fue extraída de [50].

Como indica el texto de la imagen, una cuestión clave de esta arquitectura es el bloque denominado *VOTER*. Debido a que este bloque es el que determina cuál es el resultado correcto, se requiere que la fiabilidad, $R(t)$, de este sea mucho mayor que la de cada computadora de vuelo. Esto se logra a través del uso de hardware más robusto, lo que resulta en que el bloque *VOTER* sea más costoso que cada computadora de vuelo. Por ejemplo, cada computadora de vuelo puede comprender un microcontrolador COTS, mientras que el bloque voter puede estar implementado con un ASIC específico para esa aplicación [10]. Si bien este bloque tiene una fiabilidad mucho mayor, siempre existe la probabilidad de que ocurra un error en este. En cuyo caso, el error puede decantar en un fracaso, por ejemplo si el *VOTER* elige como resultado correcto, aquel que realmente no lo era.

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

Una forma de combatir esto es replicar los bloques que realizan la votación [41][49]. De esta manera, también pueden enmascarse errores de los bloques que realizan la votación. La arquitectura sería como la que se muestra en la figura 31.

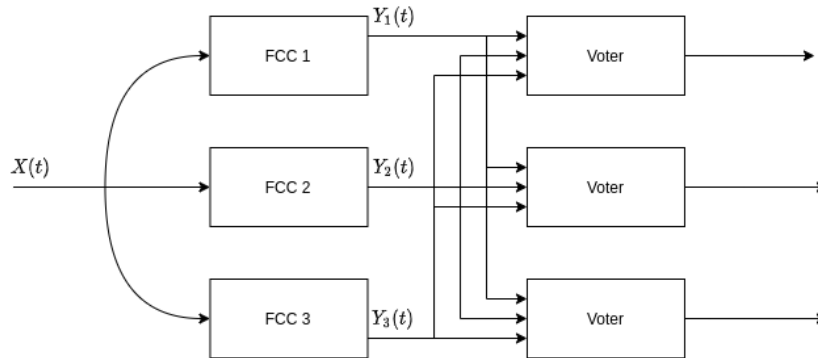


Figura 31: Arquitectura TMR con redundancia en los elementos votantes.

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 [10] se propone que los microcontroladores automotivos ofrecen las interfaces necesarias para implementar una red redundante para tolerar fallas. En el artículo [9], 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 32.

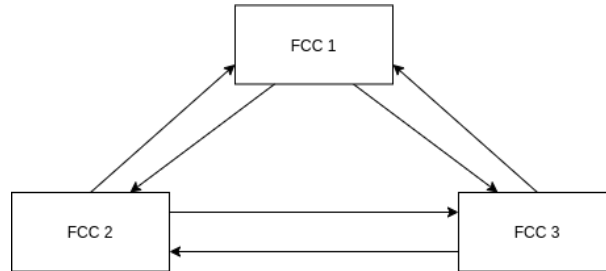


Figura 32: Arquitectura de redundancia triple, donde los bloques votantes son las mismas FCCs. Los votantes se encuentran integrados dentro de cada FCC.

6.4. Algunos Requerimientos de un Sistema Redundante

Si bien el uso de redundancias apunta a incrementar la fiabilidad del sistema y tolerar fallas, es un error pensar que el simple hecho de tener un sistema redundante equivale a un incremento de la fiabilidad [42]. Esto es principalmente por el hecho de que un sistema redundante incluye además las comunicaciones y rutinas necesarias para ejecutar las votaciones. Si estas funcionalidades no son administradas de manera correcta, un sistema redundante solamente traerá más fallas.

6.4.1. Sincronismo de los Nodos

En las arquitecturas antes presentadas, se menciona que se realiza una comparación de los resultados calculados por cada nodo, para detectar/enmascarar errores. Para que el funcionamiento de esta comparación sea adecuado, los nodos deben estar sincronizados. Esto es un requerimiento para sistemas de tiempo real, como el caso de la computadora de vuelo de un UAV.

En la figura 33 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 33b, 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 *Voter* 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 corresponde 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 *Voter* simplemente asume algún valor por defecto.

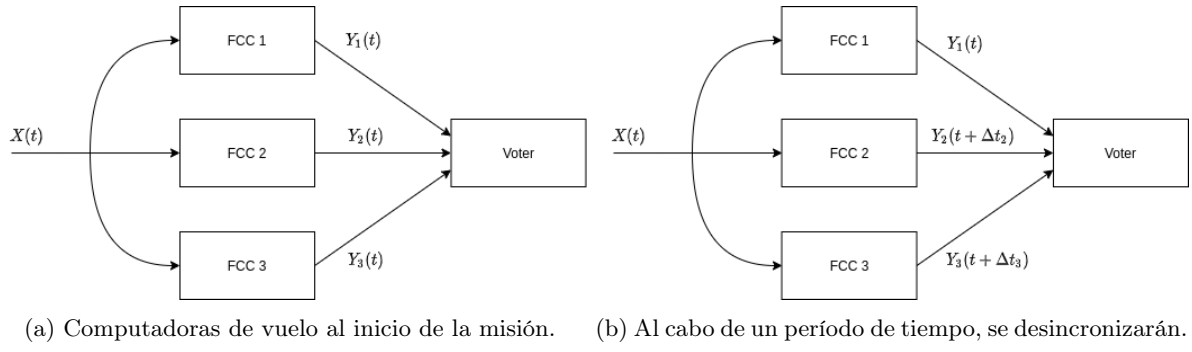


Figura 33: A medida que transcurra el tiempo, la desincronización entre FCCs impactará en el sistema redundante.

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 los *Voters* simplemente realizarán la votación asumiendo que el dato es válido.

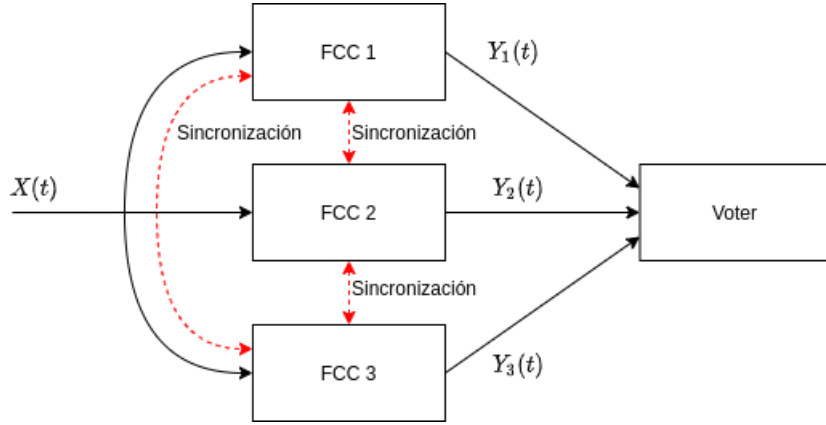


Figura 34: La sincronización entre nodos es necesaria para un correcto funcionamiento de las redundancias.

Se concluye que es mandatorio utilizar alguna técnica de sincronización entre los nodos. Como detalle de la figura 34, se muestra que la sincronización entre nodos presupone otro canal de comunicación más. Otra forma podría ser relegar la tarea de la sincronización al bloque *Voter*, 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.

6.4.2. Consenso

Lo que se plantea en esta sección, es que existe la posibilidad de que una de las FCC entregue distintos valores a cada *Voter*. En la figura 35 se muestra una situación en la que la FCC1 entrega dos valores distintos a los demás *Voters*, siendo los valores posibles *True* y *False*. Esto puede deberse por ejemplo a que la FCC1 así lo quiso, debido a una falla muy compleja de analizar y que se manifiesta como un error de esta manera. Otra posibilidad más realista puede ser el hecho de que, debido a que el dato enviado por la FCC1 llegó más tarde al tercer *Voter* que a los demás, este interpretó mal el valor recibido por la línea de comunicación, generando la situación de la figura 35.

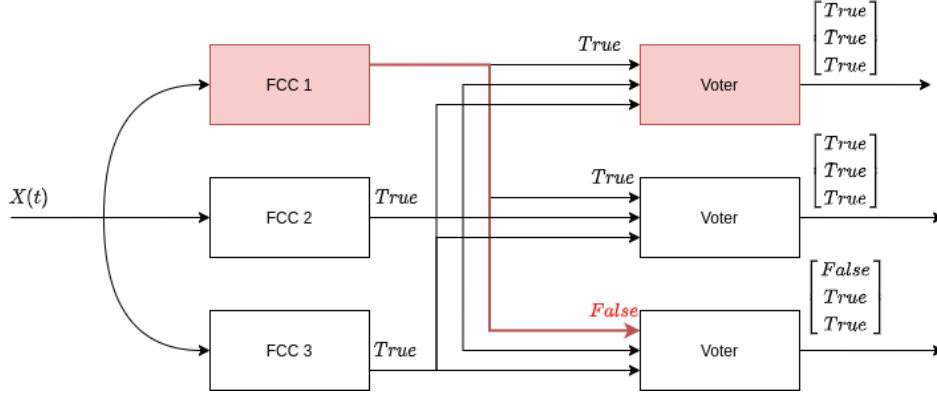


Figura 35: La FCC1 entrega el valor de *True* a un *Voter* y *False* a otro. Los vectores representan los valores sobre los que cada *Voter* debe decidir y votar por un único valor.

Este caso a priori parecería no presentar un problema que la arquitectura TMR no pueda resolver. El primer y segundo *Voter* decidirán por el valor *True*, ya que todas sus entradas son iguales a este valor. El tercer *Voter* también decidirá por el valor *True* ya que 2 de 3 de sus valores son *True*. En definitiva, la arquitectura TMR resuelve el problema del consenso para este caso.

A continuación se plantea un caso diferente. Se analiza la situación en la que cada FCC propone un valor distinto, que fue calculado por su propia observación del escenario en el que se encuentra. Esto podría ser por ejemplo, el valor de algún sensor interno a esta. Las FCC 1, 2 y 3 se encuentran dentro del mismo UAV, por lo que si poseen sensores redundantes, uno esperaría que se obtengan las mismas lecturas (si es que todos los sensores funcionan adecuadamente). Esto puede no ser así, ya que por ejemplo estas pueden presentar pequeñas variaciones por tratarse de lecturas analógicas. De manera de que el algoritmo de control se ejecute de manera consistente en las tres FCCs, ellas deben ponerse de acuerdo en un valor del sensor.

Como se mencionó en la sección anterior, se requiere lograr una sincronización entre computadoras de vuelo redundantes. De manera de ejecutar un algoritmo de sincronización adecuado, las computadoras de vuelo deben compartirle a las demás, un valor asociado a su propio clock interno.

Estos dos últimos escenarios difieren del caso en el que la FCC comparte un valor que puede ser *True* o *False*. Lo que se plantea aquí es un caso en el que cada FCC comparte un valor que corresponde a la propia perspectiva de cada una de ellas. Debido a esto, no existe un valor correcto a transferirle a las demás. Se muestra un ejemplo para la sincronización de FCCs en la figura 36.

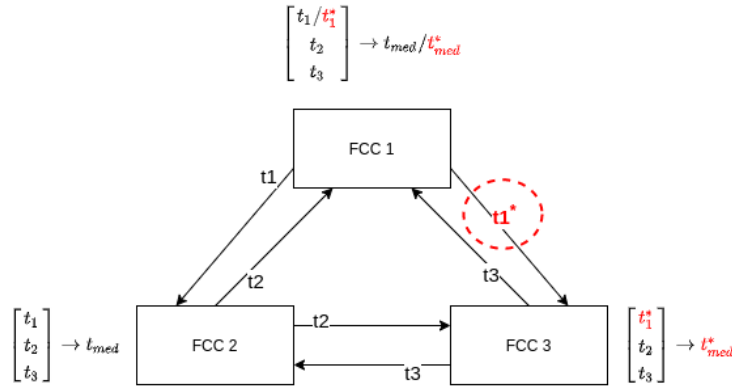


Figura 36: La FCC1 entrega un valor distinto de timing a las demás FCCs

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 37.

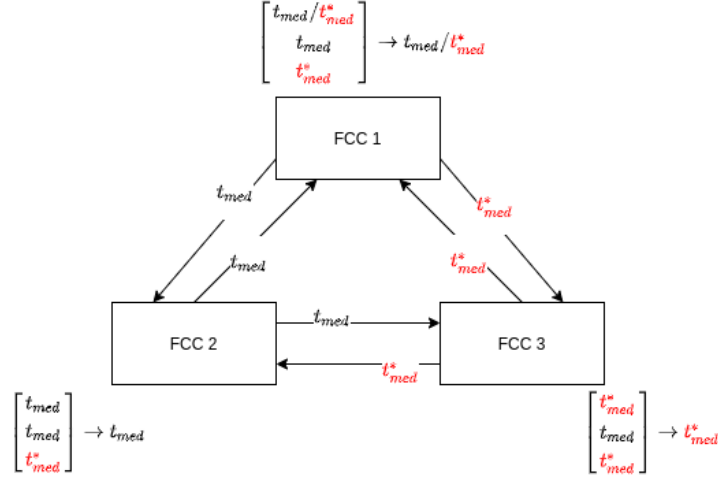


Figura 37: Luego de calcular los promedios, las FCCs intercambian sus resultados. Nuevamente, la FCC1 comete una falla en el envío del dato.

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 antes mencionado, la falla bizantina, ya que contempla fallas de hardware que se manifiestan como comportamientos arbitrarios. El ejemplo presentado en esta sección, se corresponde con un comportamiento arbitrario.

6.5. 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 [44]. 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 [44]. Otros trabajos que tratan el mismo problema son [51] y [52]. 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.

6.5.1. 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.

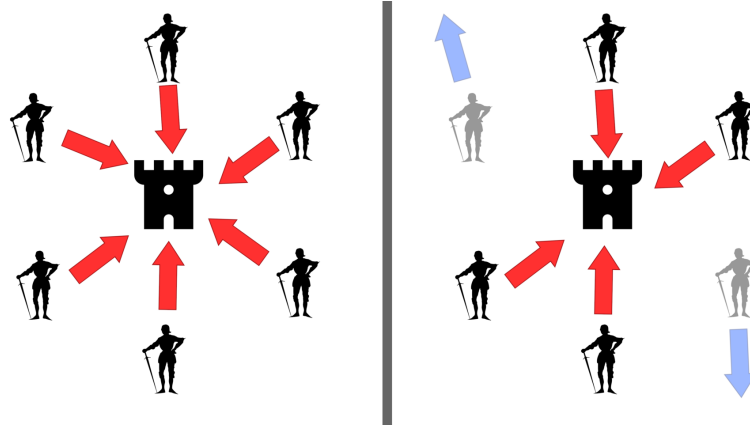


Figura 38: La situación que se presenta, donde los generales deben tomar una decisión común. La figura de la derecha muestra la situación donde algunos generales atacan mientras que los generales traidores no lo hacen. La imagen se extrajo de [53].

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 informarse que su voto es por atacar la ciudad.

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

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

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

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

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

6.5.2. 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 39 se muestra un caso para 4 generales y 1 traidor. El General 1 es el traidor y le envía información diferente a cada general.

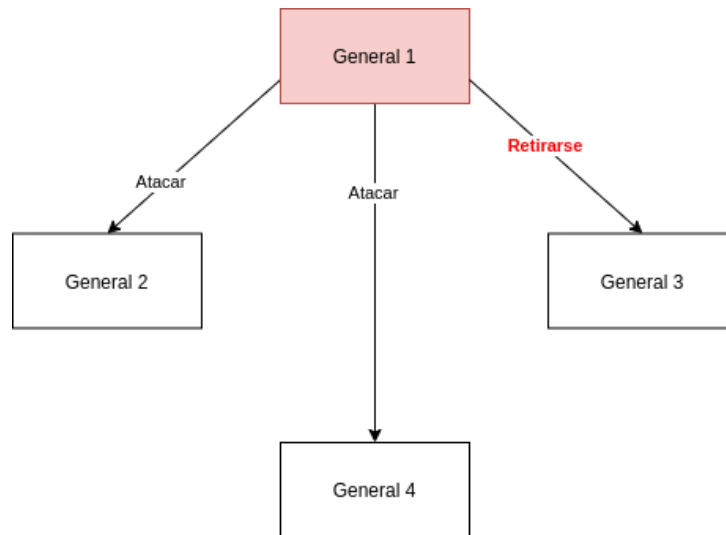


Figura 39: El general 1 es un traidor y le envía información conflictiva a los demás generales.

Como fue mencionado, para llegar a una decisión común, todos los generales deben conocer la opinión de los demás. El problema en este caso es que el General 1 envió una información diferente a sus pares. Para resolver esto, el algoritmo plantea realizar un segundo intercambio de mensajes como el de la figura 40.

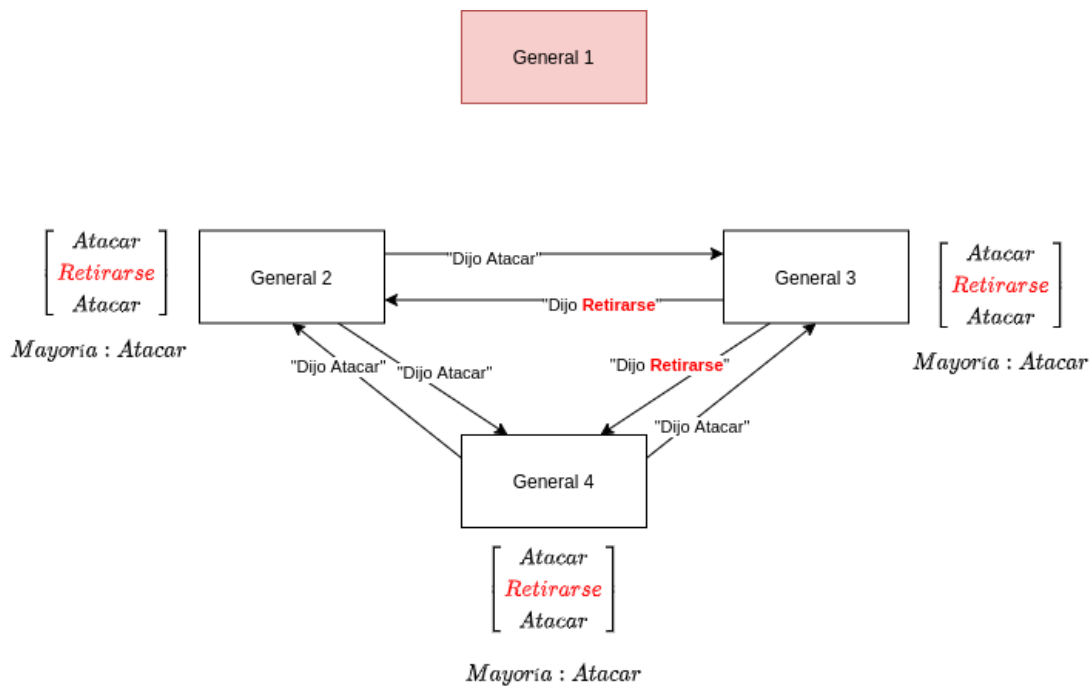


Figura 40: Se produce un intercambio entre los demás generales, para ponerse de acuerdo respecto de si el General 1 dijo “Atacar” o “Retirarse”.

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.

6.5.3. 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 [52], 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.

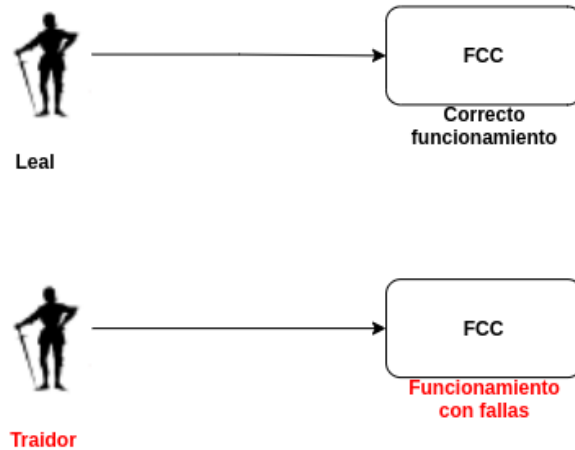


Figura 41: En el problema, un general leal representa un nodo, en este caso una computadora de vuelo, que funciona correctamente. Un General traidor es equivalente a una computadora de vuelo que presenta fallas.

Los generales traidores representan a las computadoras de vuelo que presentan fallas. En [52] 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.

Se plantea una situación como la de la figura 39, pero se reemplazan a los generales por computadoras de vuelo. 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 42 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 las computadoras de vuelo.

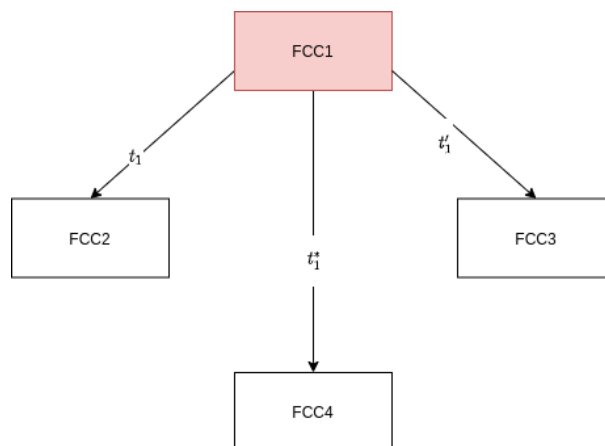


Figura 42: Debido a una falla, la computadora de vuelo 1 le entrega valores distintos de timestamp a las demás.

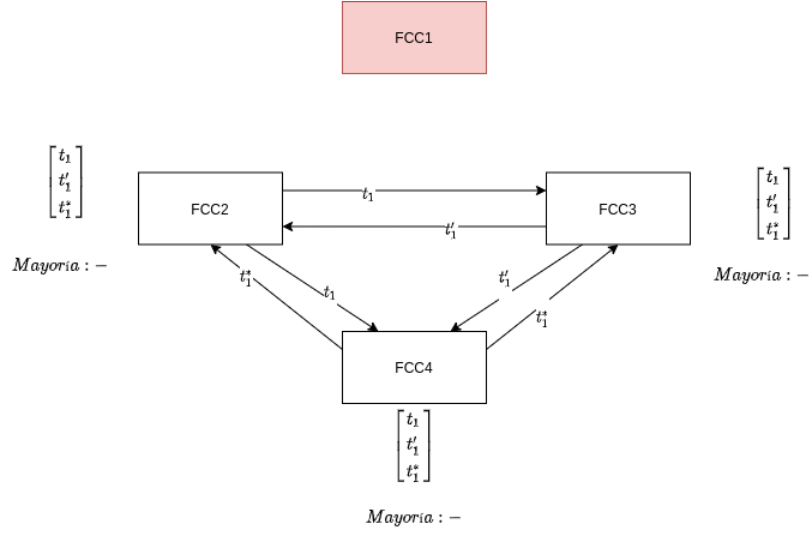


Figura 43: Debido a una falla, la computadora de vuelo 1 le entrega valores distintos de timestamp a las demás.

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 compara a un general traidor con una computadora con fallas. Como se mencionó, los generales traidores pueden tener cualquier comportamiento. 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 [42]. 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.

7. Arquitectura de Redundancia Propuesta RE ACOMODAR EN OTRA SECCIÓN

En esta sección, se presenta la arquitectura implementada para la tolerancia a fallas de hardware. Como se mostró en la sección anterior, *The Byzantine Generals Problem* sienta las bases para la tolerancia a fallas arbitrarias de hardware. A través de una serie de intercambios de mensajes entre los nodos de la red, estos pueden llegar a un consenso, para tomar la misma decisión. Además, se mostró que para poder tolerar una falla arbitraria, se requiere por lo menos de 4 nodos interconectados.

Luego de presentar el problema, se hizo una comparación entre los generales leales y las computadoras de vuelo sin fallas y entre los generales traidores y las computadoras con fallas. Una de las cuestiones que no se mencionó, es el hecho de que las computadoras de vuelo constituyen sistemas de tiempo real. Esto es debido a que 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 generales 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 generales es 1 a 1. Debido a esto, los generales traidores pueden entregar información confusa a sus pares para tratar de romper el consenso. Esto es lo que vuelve complejo al problema [44] y costosa a su solución [43].

En esta sección se muestra que, si el sistema tiene ciertas características, en particular ser un sistema de tiempo real y contar con un bus común a los nodos para las comunicaciones, luego el problema del consenso se simplifica mucho.

7.1. Simplificación del Problema de Tolerancia a Fallas Arbitrarias de Hardware

En sistemas de tiempo real para aplicaciones *safety-critical*, es común encontrar sistemas distribuidos con comunicación a través de un bus. Por ejemplo en los automóviles, los nodos que se encuentran repartidos por todo el vehículo se comunican a través de redes como CAN[26] o FlexRay[11]. 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.

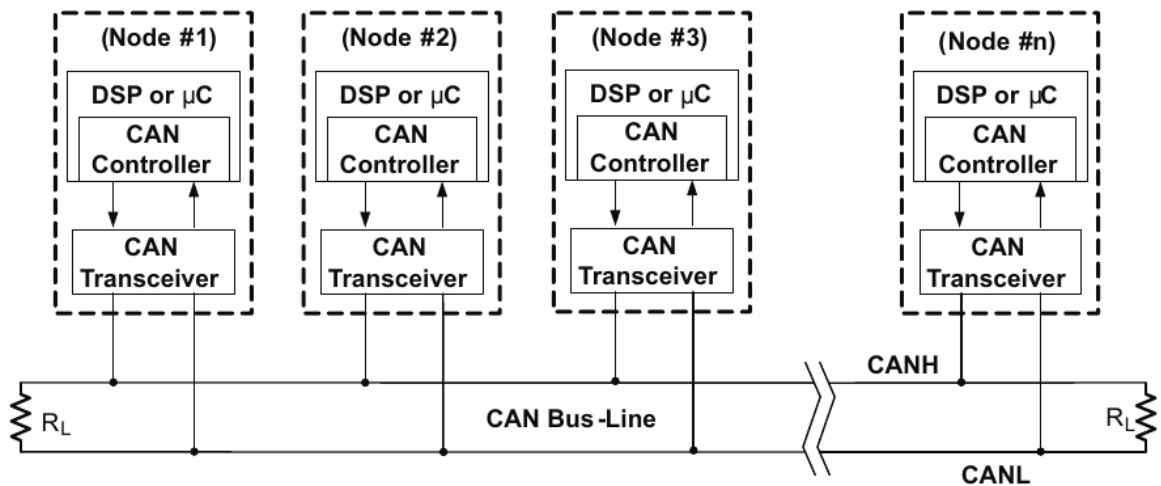


Figura 44: Todos los nodos se encuentran conectados al mismo bus de comunicaciones. En el caso del bus CAN, se compone de dos cables, CAN-H y CAN-L, terminados en sus extremos por resistencias de adaptación. La imagen se extrajo de [54].

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 45 con la figura 42.

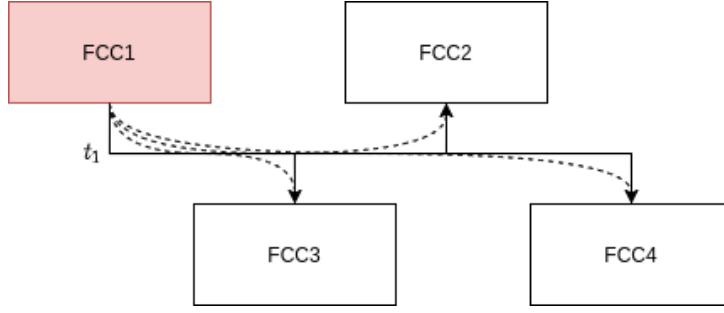


Figura 45: En este caso, la conexión tipo bus no permite el envío de información diferente a los demás miembros. La FCC1 envía el valor t_1 y todos sus pares reciben el mismo valor.

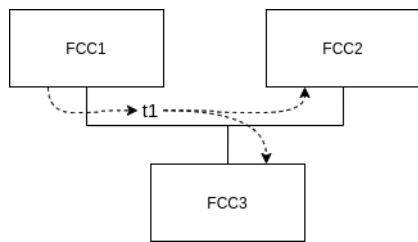
Como contrapartida, debido a que los nodos comparten canal de comunicación, estos deben tomar turnos para enviar la información a sus pares. De otra forma, habría una colisión en el bus y la información nunca llegaría a su destino. Sumado a esto, el bus se convierte en un punto singular de falla, ya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

ACÁ AGREGAR EJEMPLOS DE USO DE DOBLE BUS. POR EJEMPLO LOS AUTOS CON DOBLE CAN O DOBLE FLEX RAY, EL PAPER QUE USA DOBLE TIME TRIGGERED BUS, ETC

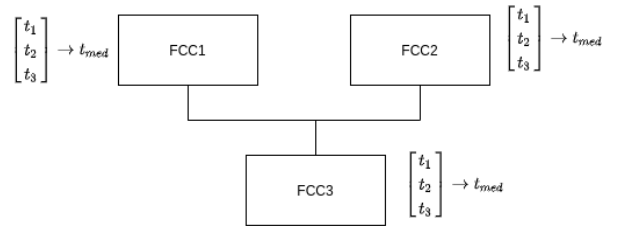
7.1.1. Consenso

Al igual que como se hizo en la sección 6.4.2, se analiza el problema del consenso para la arquitectura propuesta en esta sección. 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.

Para el caso en el que se utiliza un bus de comunicación, como 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 46a, 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 46b 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 46: 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.

7.1.2. Sincronismo de los nodos

En la sección 6.4.1 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 [9]:

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 47 se muestra un gráfico con la secuencia de ejecución periódica de las tareas.

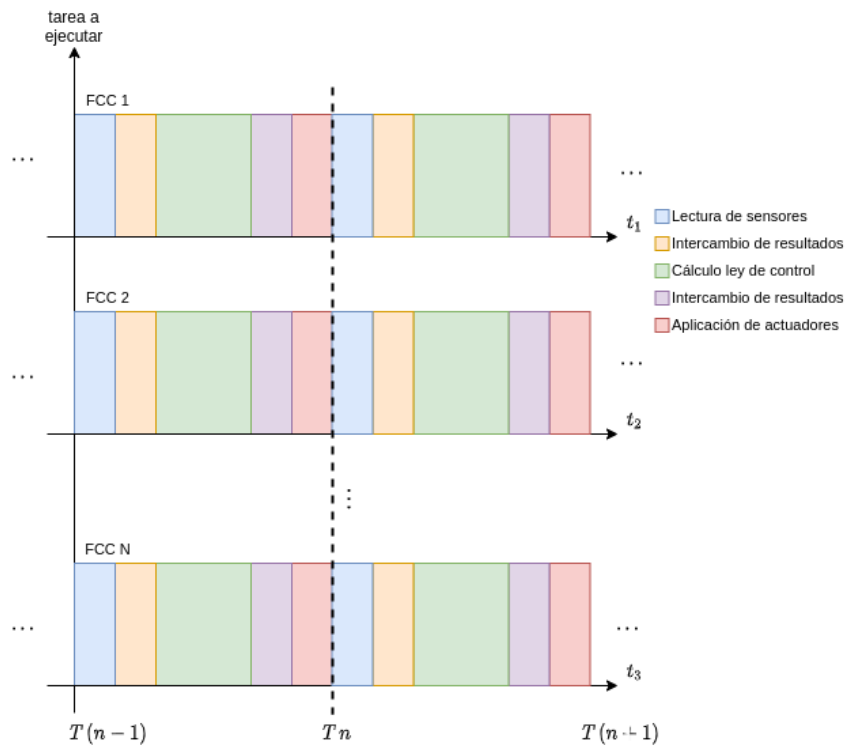


Figura 47: El eje horizontal representa el paso del tiempo. Las barras de colores representan el tiempo dedicado a ejecutar cada tarea, como la lectura de sensores, cálculo de la ley de control, etc., de forma periódica. En la imagen se puede ver que las FCC 1, FCC 2, ..., FCC N se encuentran sincronizadas 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 48 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.

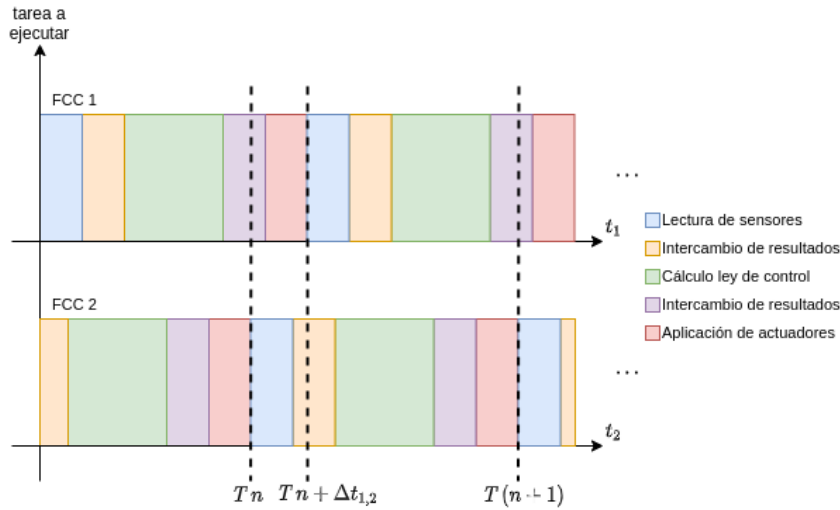


Figura 48: Se muestra un ejemplo para dos FCCs. A diferencia de la figura 47, las FCC 1 y la FCC 2 se encuentran desincronizadas.

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.

7.2. Arquitectura Del Sistema: *The Time-Triggered Architecture*

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*[46], 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 6.1, 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[46][55] y libros[1] 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.

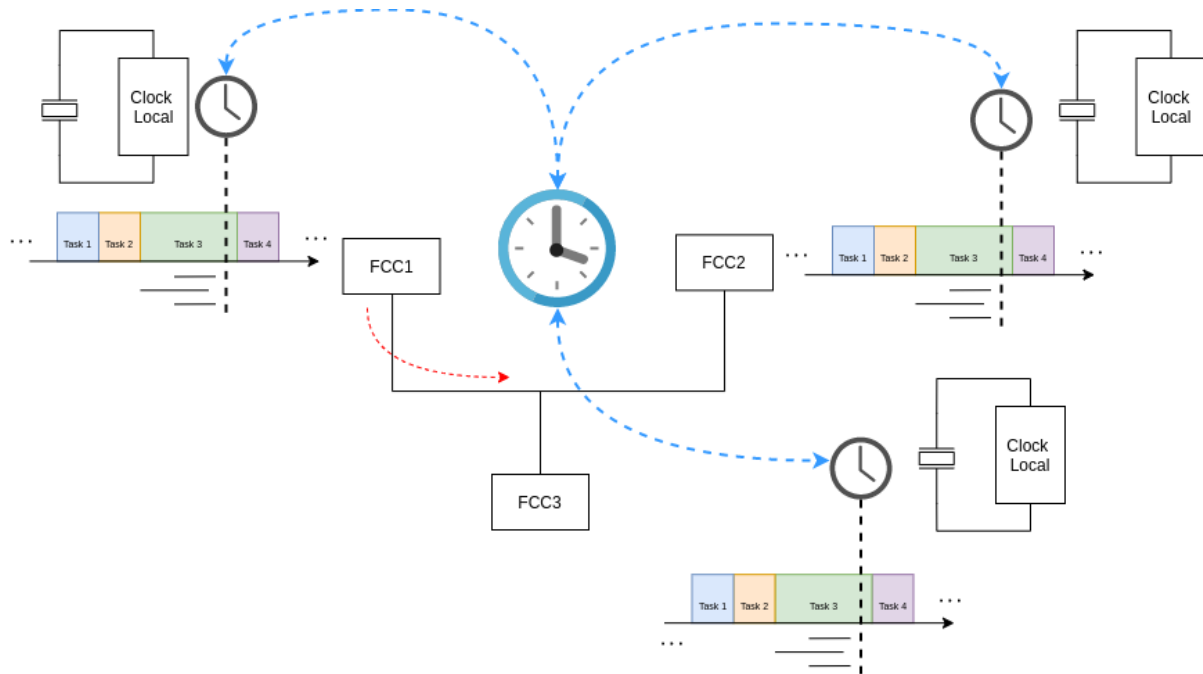


Figura 49: Se muestra un esquema que representa la arquitectura. Cada nodo tiene un *clock* local, funcionando a partir de su propio cristal. Todos estos a su vez se sincronizan periódicamente con el *global time*, representado por el reloj azul en el centro. En la imagen, el nodo 1 está enviando un 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**[1, 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 49 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 49 el nodo FCC1 presenta una falla y no envía el mensaje 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 [1] y [46].

7.2.1. Bus de Comunicaciones

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 50 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.

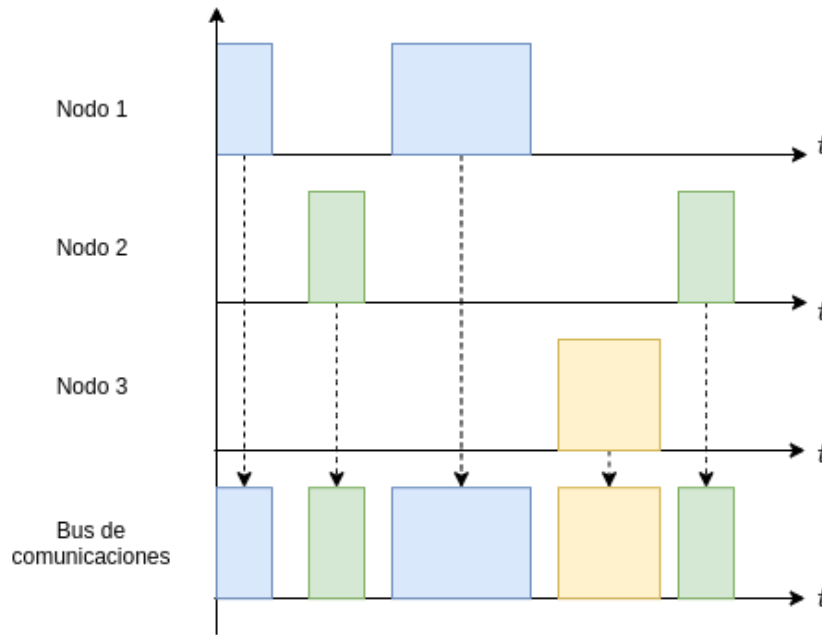


Figura 50: El ejemplo muestra como los 3 nodos pueden compartir el bus de comunicaciones. Cada uno de ellos sabe en qué instante de tiempo enviar un mensaje y en qué instantes de tiempo no deben hacerlo y solamente pueden escuchar el bus. Para que esto funcione adecuadamente, los nodos deben estar sincronizados.

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

En [46] 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 *pusha* 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.

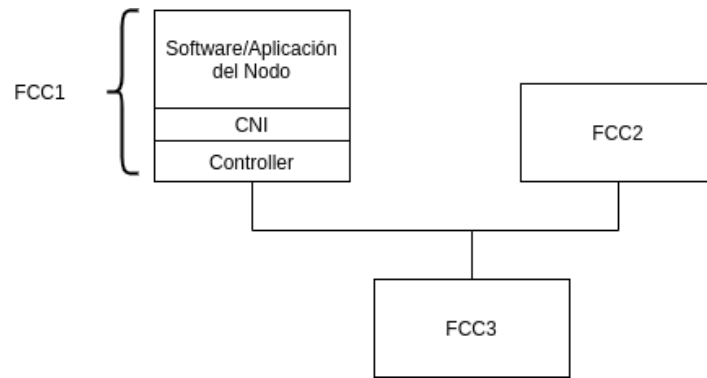


Figura 51: Misma imagen que 46a. Se muestra el detalle de los nodos, en el nodo FCC1.

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 llegue un mensaje nuevo, se dispare una interrupción en el software. El problema con esto es que se le quita determinismo al sistema.

7.2.2. Nodos

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 describe en la próxima sección.

7.2.3. Scheduler

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 teléfono 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 [56, p. 247].

7.2.4. Global Time y Sincronización

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) [57] 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*[1, 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 Π . 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 52.

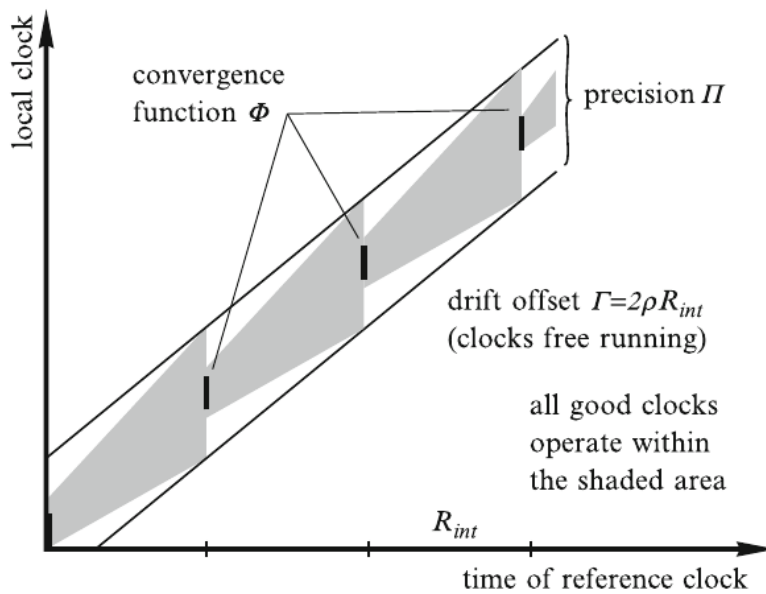


Figura 52: El eje horizontal representa el paso del tiempo físico y el eje vertical el avance de cada clock local a cada nodo. El valor R_{int} corresponde al período de resincronización. La imagen se extrajo de [1, p. 67].

Existen muchísimos algoritmos de resincronización. En [58] 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 53, donde lo que varía es la implementación de cada bloque.

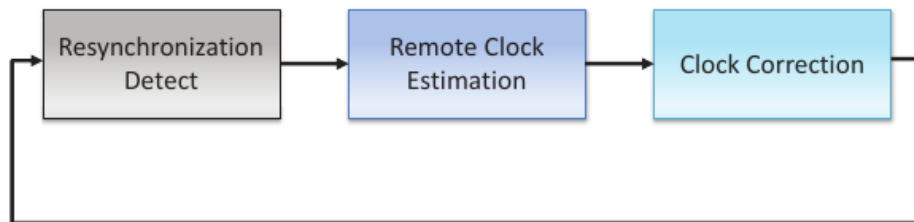


Figura 53: Tres bloques que comprenden un algoritmo de resincronización. La imagen se extrajo de [59].

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 [1, 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.

8. Conclusiones

9. Agradecimientos

COMPLETAR

Apéndices

Apéndice A: Circuito Esquemático

COMPLETAR

Referencias

- [1] 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>.
- [2] *Fly-by-Wire Systems Enable Safer, More Efficient Flight | NASA Spinoff*. URL: https://spinoff.nasa.gov/Spinoff2011/t_5.html.
- [3] Richard PG Collinson. *Introduction to avionics systems*. Springer Nature, 2023.
- [4] Ying C Yeh. «Triple-triple redundant 777 primary flight computer». En: *1996 IEEE Aerospace Applications Conference. Proceedings*. Vol. 1. IEEE. 1996, págs. 293-307.
- [5] 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.
- [6] Embention. *Veronte Autopilot 4x - Products Veronte Embention*. Oct. de 2023. URL: <https://www.embention.com/product/veronte-autopilot-4x/>.
- [7] *VECTOR-600 -Autopilot for UAV | UAV Navigation*. URL: <https://www.uavnavigation.com/products/autopilots/vector-600>.
- [8] 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>.
- [9] 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.
- [10] 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.
- [11] *MPC5744P FlexRay Interface in Pictures*. AN12233. Rev. 0. NXP Semiconductors. Mayo de 2021.
- [12] 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.
- [13] 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.
- [14] 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.
- [15] M.E. Fernando Fidencio Solano Pérez. «Development of a Redundancy System for Autopilots». Tesis de mtría. Santiago de Querétaro, México, 2019.
- [16] Dronecode Foundation. *Homepage - PIXHawk*. Jun. de 2023. URL: <https://pixhawk.org/>.
- [17] *ICM-42688-P | TDK InvenSense*. Oct. de 2023. URL: <https://invensense.tdk.com/products/motion-tracking/6-axis/icm-42688-p/>.
- [18] Kai Zhang. «Sensing and control of mems accelerometers using Kalman filter». En: (2010).
- [19] 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.
- [20] *How to measure absolute pressure using piezoresistive sensing elements*. AMSYS. Jul. de 2009.
- [21] A. C. Lapadatu y H. Jakobsen. «Anodic Bonding». En: *Handbook of Silicon Based MEMS Materials and Technologies (pp. 599-610)* (2015), págs. 599-610.
- [22] Avnet. *MEMS pressure sensors*. URL: <https://www.avnet.com/wps/portal/abacus/solutions/technologies/sensors/pressure-sensors/core-technologies/mems/> (visitado 31-10-2023).
- [23] Mustafa Cavcar. «The international standard atmosphere (ISA)». En: *Anadolu University, Turkey* 30.9 (2000), págs. 1-6.
- [24] *Choosing the Right Pressure Sensor*. AN-201610-PL38-01. Infineon. 2016.

- [25] 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).
- [26] CAN Specification. «Bosch». En: *Robert Bosch GmbH, Postfach 50* (1991), pág. 75.
- [27] *A CAN Physical Layer Discussion*. AN 228. Microchip. 2002.
- [28] *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>.
- [29] *SN65HVD23x 3.3-V CAN Bus Transceivers*. Abr. de 2018. URL: <https://www.ti.com/product/es-mx/SN65HVD230>.
- [30] *Connector Pin Assignment Recommendations*. CiA 106. CAN in Automation. Jun. de 2022.
- [31] *DroneCAN*. URL: <https://dronecan.github.io/> (visitado 11-09-2023).
- [32] Leonardo Garberoglio y col. «Diseño de un autopiloto para pequeños vehículos no tripulados». En: *Elektron 3.1* (2019), págs. 29-38.
- [33] *Automotive Compliant 1A Low Dropout Positive Regulator with Fixed and Adjustable Outputs*. Oct. de 2016. URL: <https://www.diodes.com/assets/Datasheets/ZLD01117Q.pdf>.
- [34] *Basics of Low-Dropout (LDO) Regulator ICs*. TOSHIBA. Mar. de 2021.
- [35] *Technical Review of Low Dropout Voltage Regulator Operation and Performance*. Texas Instruments. Ago. de 1999.
- [36] *AN-1482 LDO Regulator Stability Using Ceramic Output Capacitors*. Texas Instruments. Abr. de 2013.
- [37] *Holybro PM02 V3 Power Module*. URL: https://docs.px4.io/main/en/power_module/holybro_pm02.html (visitado 13-09-2023).
- [38] *PCB Design Guidelines For ICM-40607x, ICM-40608, ICM-42xxx, ICM-43xxx and ICM-45xxx Products*. AN-000262. TDK Invensense. Ene. de 2021.
- [39] *Surface Mounting Guidelines for MEMS Sensors in a QFPN Package*. TN0019. ST. Mar. de 2020.
- [40] *Soldering Guidelines for MEMS Inertial Sensors*. APP 5604. Maxim Integrated. Mar. de 2013.
- [41] Victor P. Nelson. «Fault-tolerant computing: Fundamental concepts». En: *Computer* 23.7 (1990), págs. 19-25.
- [42] 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.
- [43] 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.
- [44] Leslie Lamport, Robert Shostak y Marshall Pease. «The Byzantine generals problem». En: *Concurrency: the works of leslie lamport*. 2019, págs. 203-226.
- [45] Vinod B Prasad. «Fault tolerant digital systems». En: *IEEE Potentials* 8.1 (1989), págs. 17-21.
- [46] Hermann Kopetz y Günther Bauer. «The time-triggered architecture». En: *Proceedings of the IEEE* 91.1 (2003), págs. 112-126.
- [47] 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.
- [48] Federico Fidencio Solano Pérez. «Development of a Redundancy System for Autopilots». 2019.
- [49] 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.
- [50] *Triple Modular Redundancy*. URL: <https://www.layerzero.com/Innovations/Industry-Firsts/Triple-Modular-Redundancy.html>.
- [51] 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.

- [52] 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.
- [53] Wikipedia contributors. «Byzantine fault». En: *Wikipedia* (jul. de 2023). URL: https://en.wikipedia.org/wiki/Byzantine_fault#.
- [54] *Introduction to the Controller Area Network (CAN)*. SLOA101B. Rev. B. Texas Instruments. Mayo de 2016.
- [55] Hermann Kopetz. «The time-triggered model of computation». En: *Proceedings 19th IEEE Real-Time Systems Symposium (Cat. No. 98CB36279)*. IEEE. 1998, págs. 168-177.
- [56] Michael J Pont. *Patterns for time-triggered embedded systems*. TTE System, Ltd, 2008.
- [57] Gabriel Leen y Donal Heffernan. «TTCAN: a new time-triggered controller area network». En: *Microprocessors and Microsystems* 26.2 (2002), págs. 77-94.
- [58] Emmanuelle Anceaume e Isabelle Puaut. «Performance evaluation of clock synchronization algorithms». Tesis doct. INRIA, 1998.
- [59] 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).