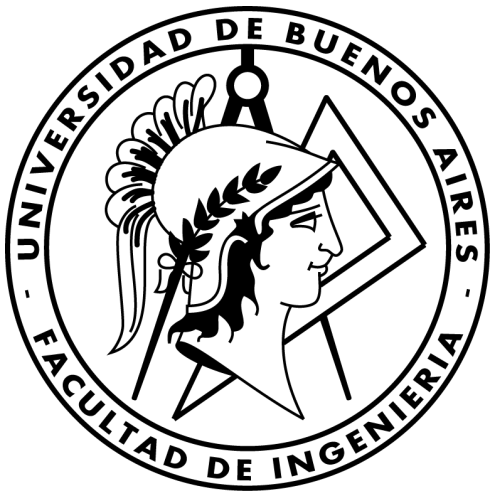


Facultad de Ingeniería

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

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 caracter 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 aéres 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 involucran 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

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.

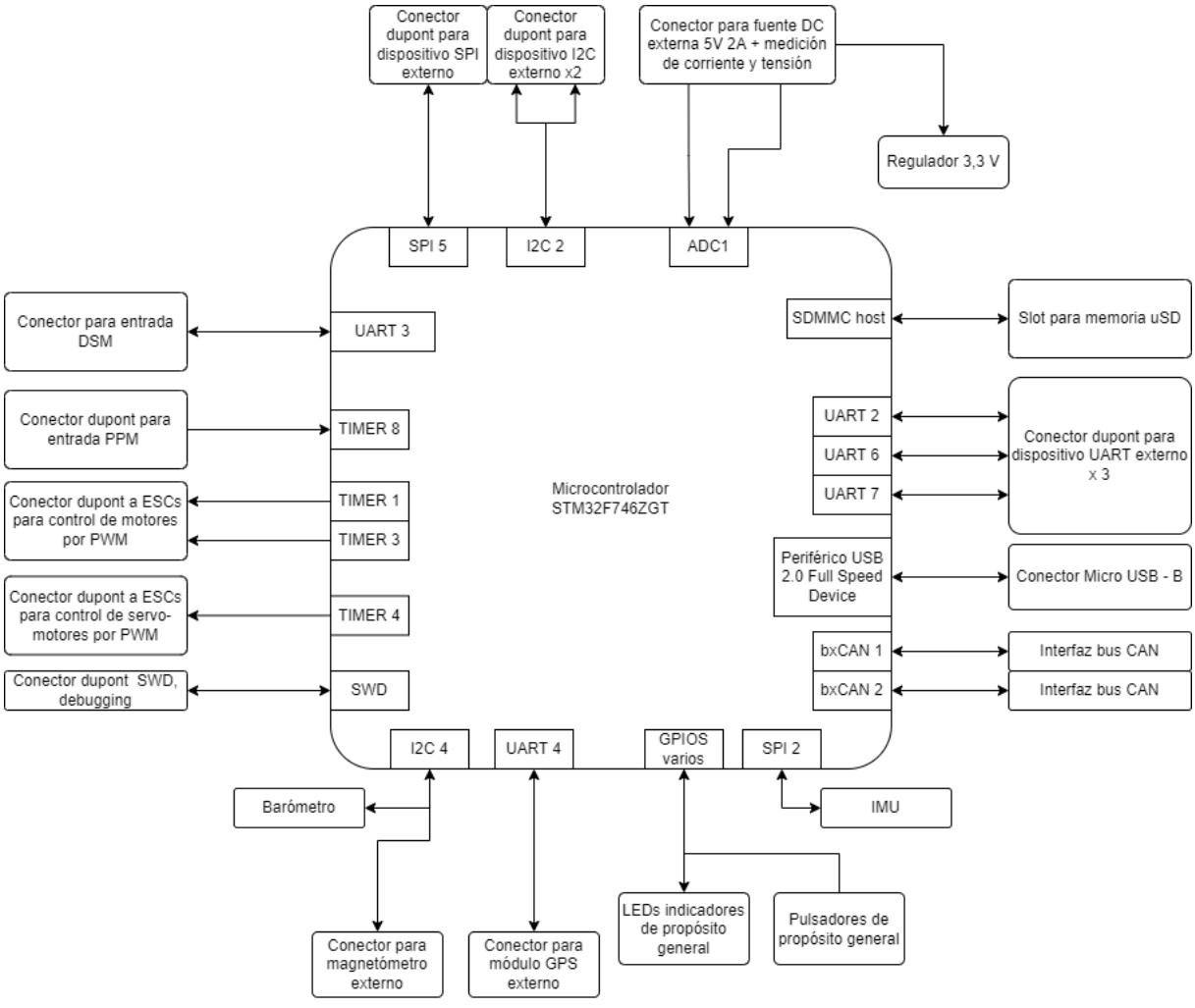


Figura 1: Diagrama en bloques de la computadora de vuelo.

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 2.

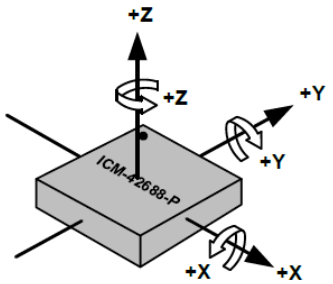


Figura 2: Todas las mediciones que entrega el sensor son realtivas a una terna solidaria a este. La imagen se extrajo de [1].

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 3, 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.

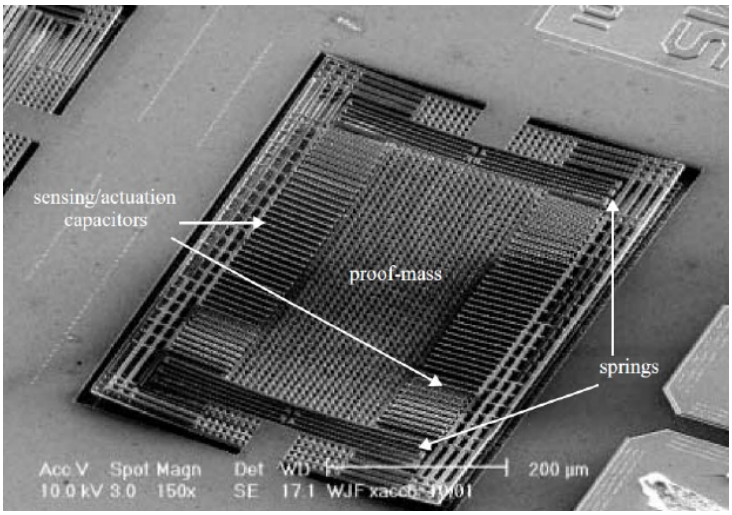


Figura 3: Fotografía tomada de un acelerómetro construido con tecnología MEMS. La imagen se extrajo de [2].

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 [2], como se muestra en la figura 4.

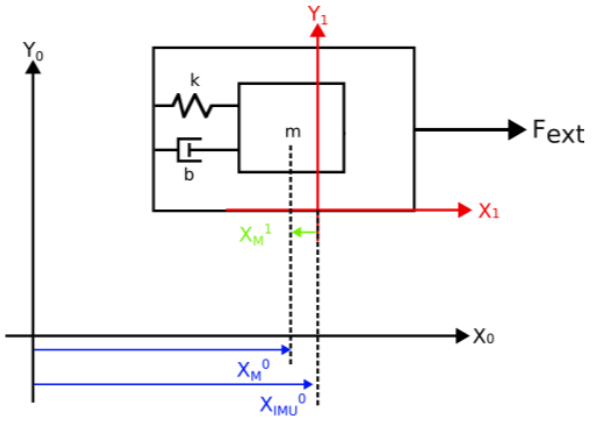


Figura 4: Sistema mecánico simplificado del acelerómetro.

La terna 0 corresponde a una terna inercial, mientras que la terna 1 es no inercial, solidaria al movimiento del acelerómetro. Cabe aclarar que tal como se mencionó, la masa se encuentra sujeta al sustrato solamente a través del resorte. El elemento amortiguador representa pérdidas de energía, causadas por rozamiento con el aire o de la propia estructura electromecánica. Se puede resolver el sistema mecánico, tomando como coordenadas generalizadas $q_1 = X^0_{IMU}$ y $q_2 = X^1_M$. Sin considerar efectos de la gravedad, se llega a la ecuación (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.

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

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).

$$\ddot{X}_M^1 \approx 0 \tag{2a}$$

$$\dot{X}_M^1 \approx 0 \tag{2b}$$

$$\frac{k}{m}X_M^1 \approx -\ddot{X}_{IMU}^0 \tag{2c}$$

El desplazamiento de la masa produce una variación de la capacidad entre esta y el sustrato. Esta capacidad es utilizada para medir una varaición de tensión [2]. El circuito medido puede modelarse como en la figura 5.

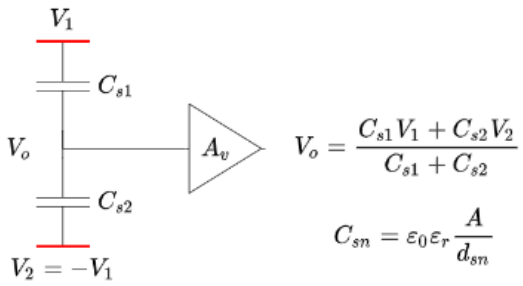


Figura 5: Circuito equivalente que mide el desplazamiento de la masa. Las tensiones $V1$ y $V2$ representan señales de tensión aplicadas por un circuito externo. El movimiento de la masa modifica la capcidad y por ende la tensión medida.

Se puede resolver este circuito y llegar a que existe una relación lineal entre la tensión Vo y el desplazamiento de la masa Δd, ecuación (3). En [2] puede encontrarse la demostración completa. En la ecuación, d_0 representa la separación de reposo entre el sensor y el sustrato y Δd la variación de la separación.

$$V_o = \frac{\Delta d}{d_0}V_1 \tag{3}$$

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 6 se muestra una captura de la hoja de datos del sensor seleccionado, el ICM42688p.

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

Figura 6: Extracto de la hoja de datos del sensor ICM42688p. Se muestra parte de las especificaciones para los acelerómetros.

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

Para entender los parámetros, *scale factor error*, *offset* y *noise* se plantea un modelo sencillo de medición, tanto para acelerómetros como para giróscopos [3]. 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.

$$\omega_m = (1 + S)\omega + \omega_b(t) + \eta \tag{4}$$

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 [3]. Este paper presenta un análisis de los parámetros de los acelerómetros y grióscopos y su impacto en las estimaciones de posición en sistemas de navegación inercial (INS). En este se concluye que los parámetros más importantes para la selección del sensor son:

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

Se buscaron modelos de IMUs de distintos fabricantes, para comparar características. Existe una gran cantidad de fabricantes y de componentes para seleccionar. Se buscaron componentes que sean accesibles y que no tengan un costo muy elevado. Existen IMUs de una excelente calidad, pero que tienen

precios que no están al alcance (cientos o miles de dólares). Con este criterio, se realizó una comparación entre distintos modelos de sensores. En la tabla 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]	±250/500/1000/2000	±15/31/62/125/250/500/1000/2000
Scale Factor Error[%]	1,0 @ 25°C	0,5 @ 25°C
Scale factor error vs temp[%/°C]	0,016 @ -40°C - 85 °C	0,005 @ 0°C - 70 °C
Offset[dps]	±1	±0,5
Offset vs temp[dps/°C]	0,01	0,005
Offset vs time[°/h]	N/A	N/A
Noise[$mdps/\sqrt{Hz}$]	4	2,8
Acelerómetros		
Full Scale Range[g]	±2/4/8/16	±2/4/8/16
Scale Factor Error[%]	1,0 @ 25°C	0,5 @ 25°C
Scale factor error vs temp[%/°C]	0,012 @ -40°C - 85 °C	0,005
Offset[mg]	±25	±20
Offset vs temp[mg/°C]	X,Y: ±0,5, Z: ±1	±0,15
Offset vs time[μg/h]	N/A	N/A
Noise[$\mu g/\sqrt{Hz}$]	100	X,Y: 65, Z: 70

Tabla 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 aceleróemtros 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 microcontroldor. 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 7 se muestra un esquema de la conexión entre el controlador y el sensor IMU.

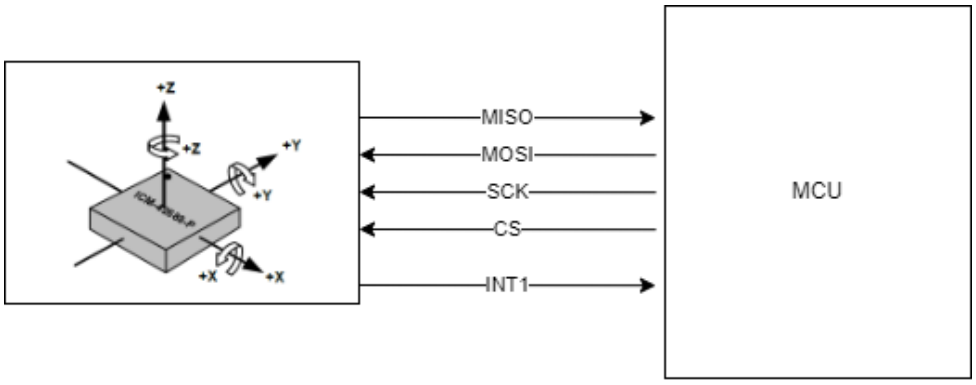


Figura 7: Líneas de comunicación entre la IMU y el microcontrolador.

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* [4][5] 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 8 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.

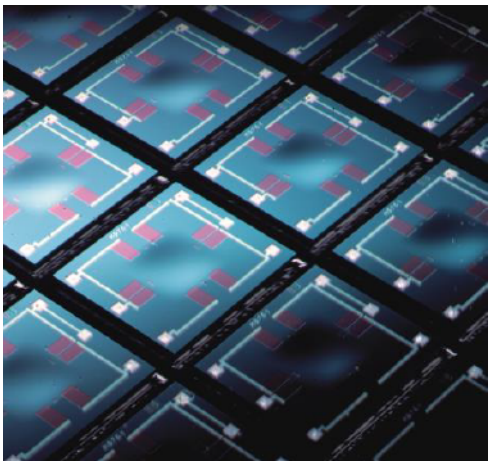


Figura 8: Sensores de presión sobre una oblea de silicio [4].

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 [6]. En (5) se despeja la relación entre la variación de tensión y la variación de la resistencia. Como se observa, la realación es proporcional.

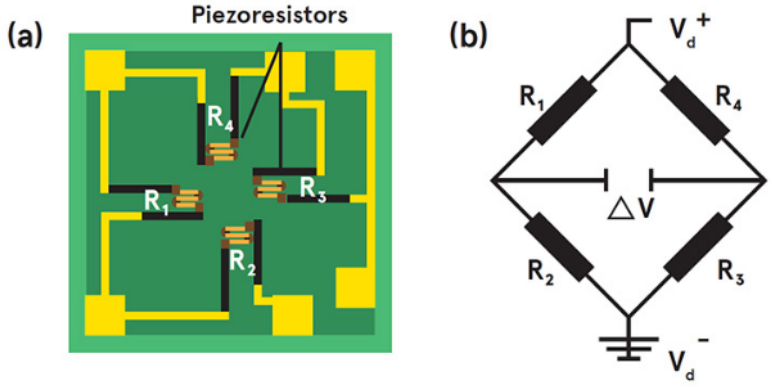


Figura 9: Puente de Wheatstone conformado por los resistores del sensor de presión. La imagen se extrajo de [6].

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

$$R_1 = R_3 = R - \Delta R \tag{5b}$$

$$R_2 = R_4 = R + \Delta R \tag{5c}$$

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

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 [7]. 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.

$$PV = nRT \tag{6a}$$

$$n = \frac{m}{M} \tag{6b}$$

$$PV = \frac{m}{M}RT \tag{6c}$$

$$P = \frac{m}{V}\frac{RT}{M} \tag{6d}$$

$$P = \rho\frac{RT}{M} \tag{6e}$$

Luego, utilizando la condición hidrostática, la presión es la ejercida por la columna de aire [7]. 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).

$$dp = -\rho g dh \tag{7}$$

$$\frac{dP}{P} = -\frac{gM}{RT}dh \tag{8}$$

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.

$$T(h) = T_0 + L\ h \tag{9a}$$

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

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

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

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

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).

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

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

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.

$$P_m = (1 + S)P + P_b(t) + \eta \tag{12}$$

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* [8]. 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 [4]. Por otro lado, el término $S\ P$ se lo denomina *relative accuracy*. Este hace referencia a mediciones diferenciales de presión. Algunos barómetros MEMS traen consigo una funcionalidad para realizar una compensación de offset. Esto dejaría como parámetro de interés para mediciones de presión a la *relative accuracy*, la cual hace referencia al error introducido para mediciones de variaciones de presión.

El parámetro *solder drfit* se refiere al offset que se introduce por el propio proceso de soldadura [8]. 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]	±0,5	±0,3	±0,2	±0,5	±0,5	±1
realtive acc [hPa]	±0,03	±0,06	±0,01	±0,025	±0,015	±0,06
longevidad	N/A	N/A	N/A	N/A	10 años, enero 2023	N/A
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 [9]. 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 a IMU, no cuenta con una línea de interrupción, por lo que los datos deben obtenerse por *polling* de forma peródica. El circuito completo puede encontrarse en el Apéndice A: Circuito Esquemático.

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 [10] 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 10. 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.

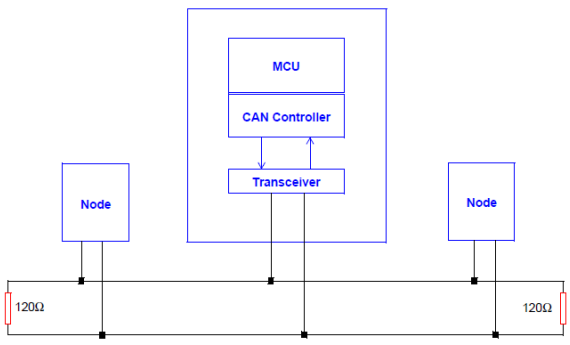


Figura 10: La conexión de un nodo al bus es a través de 2 cables que llevan dos señales, CAN-H y CAN-L. La imagen se extrajo de [11].

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. Se

recomienda que la conexión entre cada nodo y el bus no sea de más de 30 cm. La impedancia característica del bus debe ser de 120Ω. Es común agregar resistores de terminación en ambos extremos, para evitar reflexiones.

En la figura 10 se muestran 2 elementos que forman parte del nodo, el *trasnciever* y el *controller*. El primero de ellos forma parte de la capa física y es un circuito que convierte las señales diferenciales del bus en señales de modo común, que luego son transferidas al elemento *controller*. Este componente sirve como interfaz física con el bus.

El micronotrolador seleccionado para la computadora de vuelo, cuenta con un *controller* embebido, el periférico bxCAN [12]. Este cuenta con dos líneas de comunicación con el transciever, CAN TX y CAN RX. En la figura 11 se muestra la comunicación entre transciever, 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 transciever lo recibe y lo convierte en una señal diferencial para inyectarlo en el bus. Cuando el nodo recibe un mensaje del bus CAN, el transciever 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.

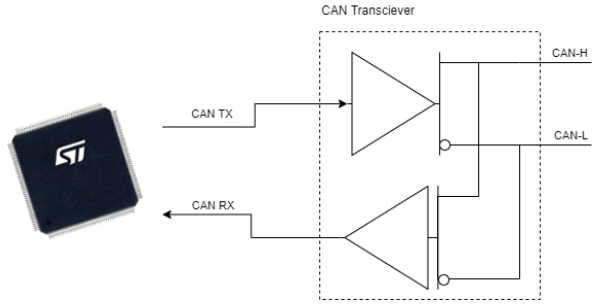


Figura 11: El periférico embebido en el microcontrolador, a través del transciever, puede interactuar con el bus CAN.

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 transciever del nodo transmisor fija la tensión de las líneas CAN-H y CAN-L de tal forma de generar una tensión diferencial $V_{CAN-H} - V_{CAN-L} \geq 1,5V$. Esto se muestra en la figura 12.

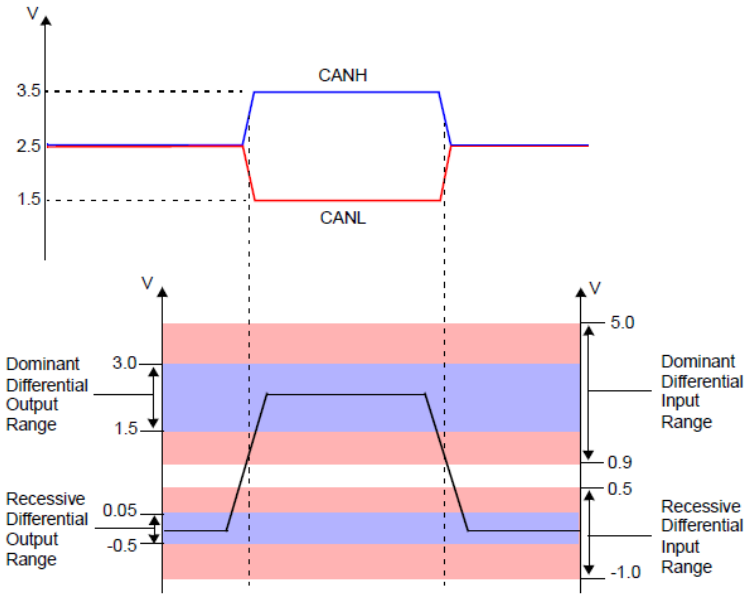


Figura 12: Se muestran los estados recessive y dominant del bus CAN y sus equivalentes lógicos.

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 por defecto, el estado del bus es *recessive*. 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 transciever, su comunicación con el microcontrolador y el conector. En cuanto al transciever, puede encontrarse mucha disponibilidad de estos. Se trata de un componente que es ampliamente utilizado en la electrónica automotriz, por lo que hay mucha disponibilidad. Existen transcievers que utilizan distintos niveles de tensión en sus salidas. La gran mayoría de los componentes de la computadora de vuelo utilizan tensiones de 3,3 V para su funcionamiento, por lo que se buscó algún transciever para esta tensión. El componente seleccionado es el SN65HVD230 de Texas Instruments [13], el cual es compatible con la especificación de capa física de CAN, ISO 11898-2. Este cuenta con una protección por exceso de tempertura, donde el componente pone sus salidas CAN-H y CAN-L en alta impdeancia, de manera de no perturbar al resto de nodos. Por otro lado cuenta con una funcionalidad que permite detectar si el transciever fue desconectado del bus, fijando un estado alto constante en su salida RX hacia el *controller*.

El transciever 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 incrementar 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 $k\Omega$ en el terminal 8 denominado Rs. Según la hoja de datos, esto corresponde a un slew-rate de $15V/\mu s$.

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 [14] 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 eligió el connector que corresponde a la especificación de un protocolo CAN desarrollado para ser usado en drones, DroneCAN [15], 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.

El circuito completo de la interfaz CAN puede encontrarse en el Apéndice A: Circuito Esquemático.

Justificar el conector. Mencionar que CAN no define un conector estándar. Luego, se siguieron las recomendaciones de CANinAutomation, del conector JST. Pero como no hay disponibilidad, elegimos otro similar. Lo importante es que tenga trabita para que no se salga.

Justificar la forma de conexión, que no es daisy chain para evitar que si se desconecta uno, se rompa todo el bus. Explicar que va a ser necserio agregar los resistores de terminación por fuera.

Justificar los resistores que setean el rise tiem del transciever y los resistores en serie con las líneas TX y RX.

COMPLETAR

4.3.6. Circuito de Alimentación

COMPLETAR

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 [16] [17] [18]. 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 13 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.

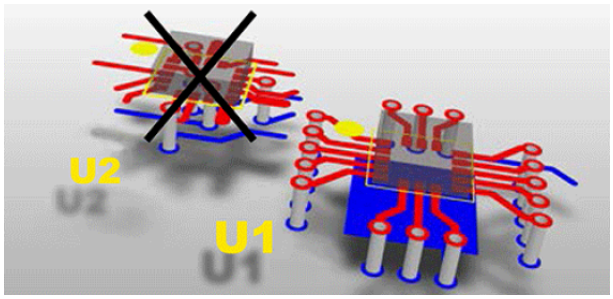


Figura 13: Se muestran dos ejemplos de layout de una IMU. El layout U2 no sigue las recomendaciones mencionadas, mientras que U1 sí. La imagen se extrajo de [18].

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 [19]. 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 [19], 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 (13), y se define como la probabilidad de que el sistema pueda cumplir su función de manera correcta en un intervalo de tiempo $[t_0; t]$, dado que en el instante inicial t_0 el sistema podía hacerlo.

$$R(t) = \mathtt{P}\,(\text{funcionamiento correcto en } t | \text{funcionamiento correcto en } t_0) \tag{13}$$

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 (14). Si no ocurre ninguna falla, luego el sistema podrá seguir cumpliendo su función en t. Además, si llegase a ocurrir una falla, pero el sistema tiene la capacidad de tolerarla, luego el sistema de igual manera podrá seguir cumpliendo su función en el instante t.

$$\begin{aligned} R(t) = {} & \mathtt{P}\,(\text{no ocurrio una falla en } [t_0; t]) \\ & + \mathtt{P}\,(\text{funcionamiento correcto en } t | \text{ocurrió una falla en } [t_0; t])\ \ \mathtt{P}\,(\text{ocurrió una falla en } [t_0; t]) \end{aligned} \tag{14}$$

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

$$R(t) = \mathtt{P}\,(\text{no ocurrio una falla en } [t_0; t]) \tag{15}$$

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 [19]. 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 (14), 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 ceritifcante, que la confianza del sistema se mantiene por encima de cierto valor. Este tipo de análisis suele consumir mucho tiempo y esfuerzo, lo que se traduce en un mayor costo del desarrollo [20].

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 [21][22][20]. 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* [23], 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 [19][24][20][25]. 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.

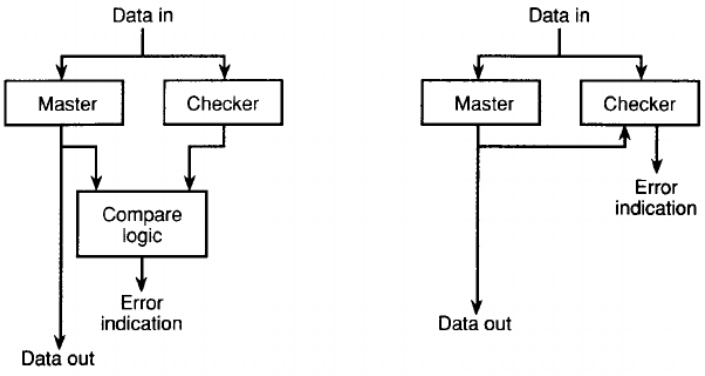
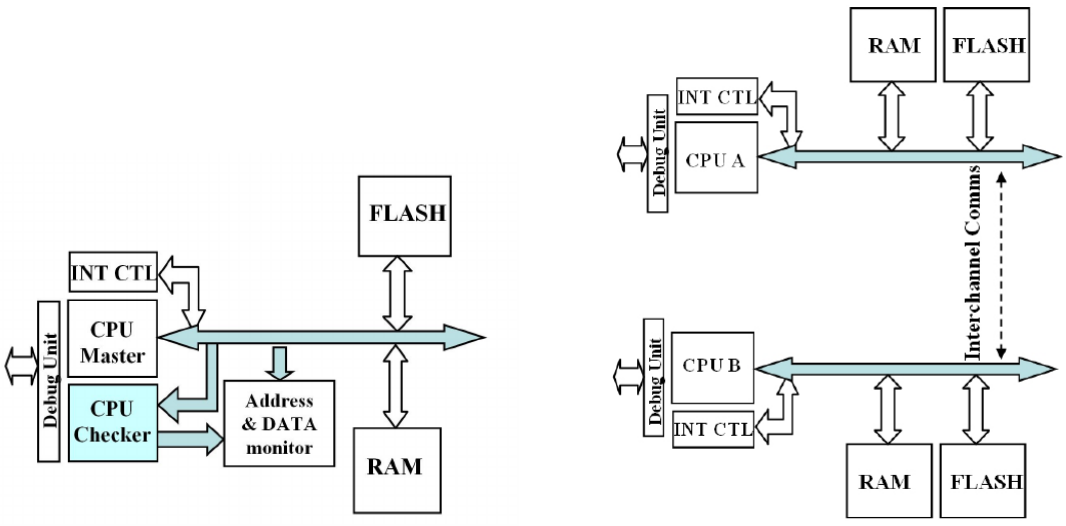


Figura 14: En la figura de la izquierda, dos sistemas ejecutan las mismas operaciones, mientras que otro sistema externo se encarga de comparar las salidas de ambos para detectar errores. En la figura de la derecha, el bloque comparador se encuentra integrado en el sistema *checker*. La imagen fue extraida de [19].

Este tipo de arquitectura permite detectar si ocurrió un error, pero no permite identificar de qué nodo proviene el error. En la figura 14 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*[26]. En esta última, ambos se encuentran sincronizados por estar en el mismo chip y compartir fuente de clock. En la figura 15 se muestra un esquema de ambos casos.



(a) Lockstep dual processor architecture.

(b) Loosely synchronized dual processor architecture.

Figura 15: Se muestran dos casos para un sistema con redundancia doble. La imagen fue extraida de [26].

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 [27] y en [28] 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* [26][19][24][29]. 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 16 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*.

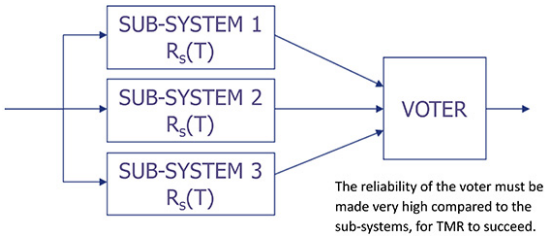


Figura 16: Arquitectura TMR. La imagen fue extraida de [30].

Como indica el texto de la imagen, una cuestión clave de esta arquitectura es el bloque denominado *VOTER*. Debido a que este bloque es el que determina cuál es el resultado correcto, se requiere que la fiabilidad, $R(t)$, de este sea mucho mayor que la de cada computadora de vuelo. Esto se logra a través del uso de hardware más robusto, lo que resulta en que el bloque *VOTER* sea más costos que cada computadora de vuelo. Por ejemplo, cada computadora de vuelo puede comprender un microcontrolador COTS, mientras que el bloque voter puede estar implementado con un ASIC específico para esa aplicación [22]. 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 [19][29]. De esta manera, también pueden enmascararse errores de los bloques que realizan la votación. La arquitectura sería como la que se muestra en la figura 17.

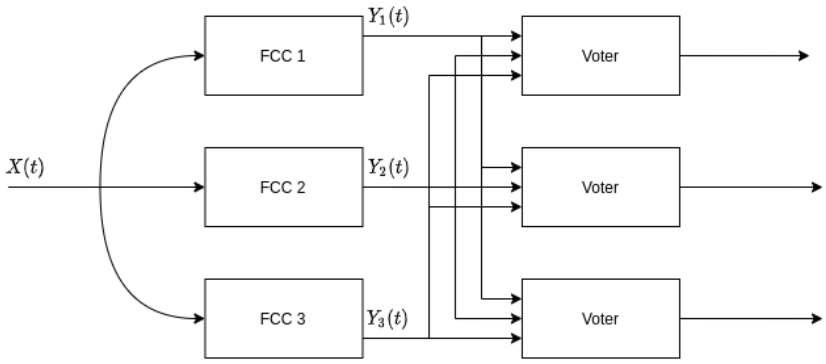


Figura 17: Arquitectura TMR con redundancia en los elementos votantes.

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 [22] se propone que los microcontroladores automotivos ofrecen las interfaces necesarias para implementar una red redundante para tolerar fallas. En el artículo [31], 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 18.

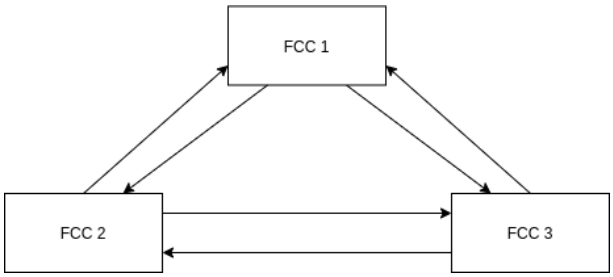


Figura 18: Arquitectura de redundancia trple, donde los bloques votantes son las mismas FCCs. Los votantes se encuentran integrados dentro de cada FCC.

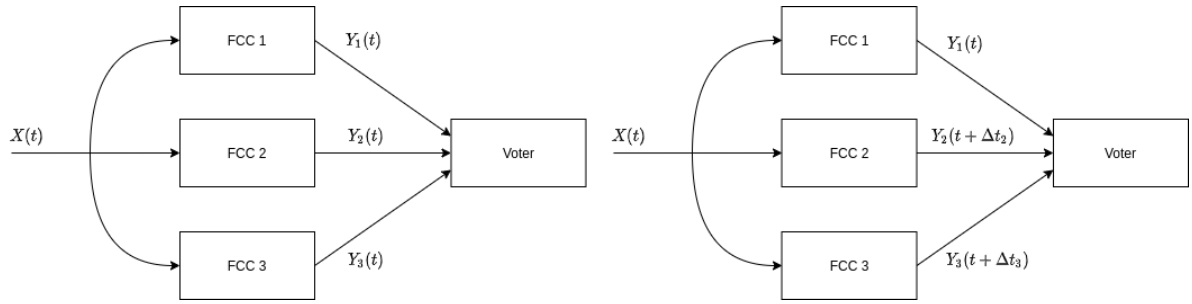
6.4. Algunos Requerimientos de un Sistema Redundante

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 [20]. 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 19 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 19b, 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 correpsonde al caso en el que una computadora de vuelo presentó un error y debido a ello no respondió con ningún valor (por ejemplo, se reinició su procesador debido a un *watchdog*). En esos casos el *Voter* simplemente asume algún valor por defecto.



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

Figura 19: 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.

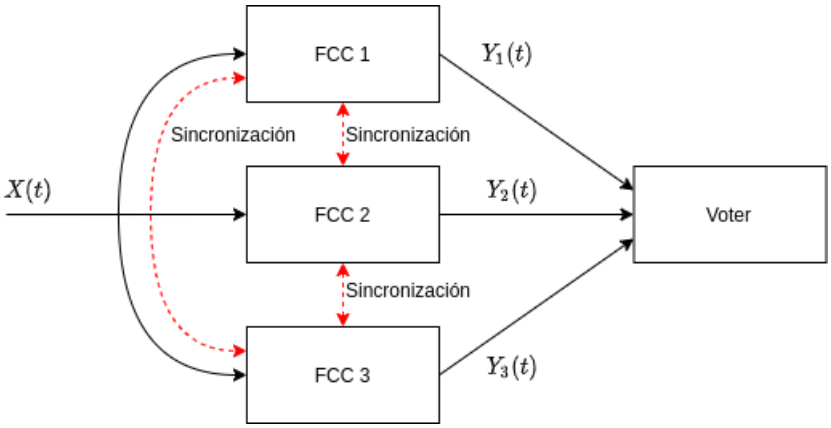


Figura 20: La sincronización entre nodos es necesaria para un correcto funcionamiento de las redundancias.

Se concluye que es mandatorio utilizar alguna técnica de sincronización entre los nodos. Como detalle de la figura 20, se muestra que la sincronización entre nodos presupone otro canal de comuniación 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 21 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 comuniación, generando la situación de la figura 21.

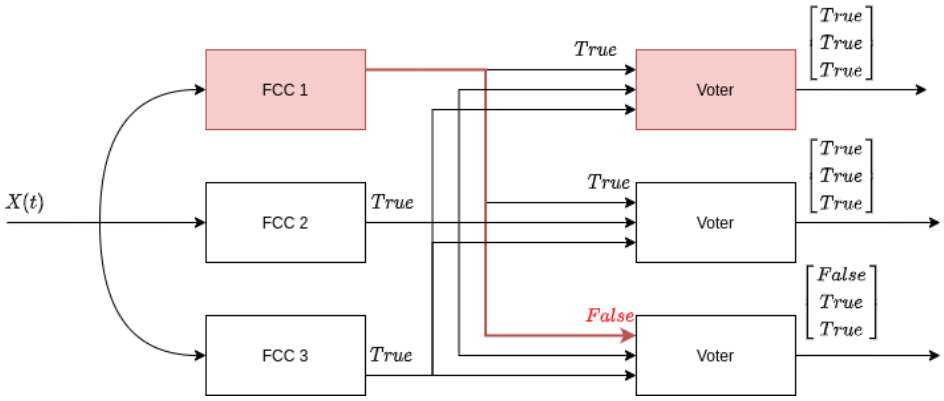


Figura 21: La FCC1 entrega el valor de *True* a un *Voter* y *False* a otro. Los vectores representan los valores sobre los que cada *Voter* debe decidir y votar por un único valor.

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 últmos 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 22.

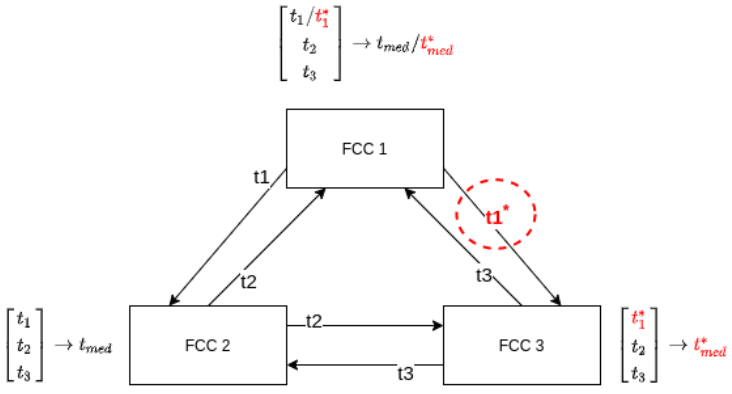


Figura 22: La FCC1 entrega un valor distinto de timing a las demás FCCs

.

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 23.

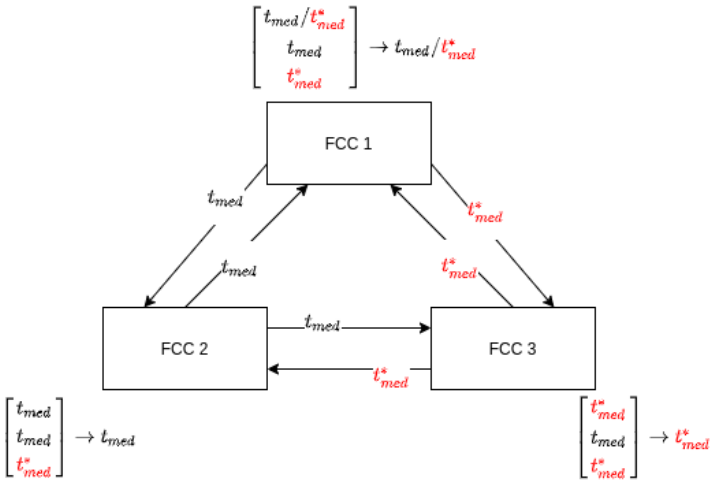


Figura 23: Luego de calcular los promedios, las FCCs intercambian sus resultados. Nuevamente, la FCC1 comete una falla en el envío del dato.

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 [23]. 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 [23]. Otros trabajos que tratan el mismo problema son [32] y [33]. 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.

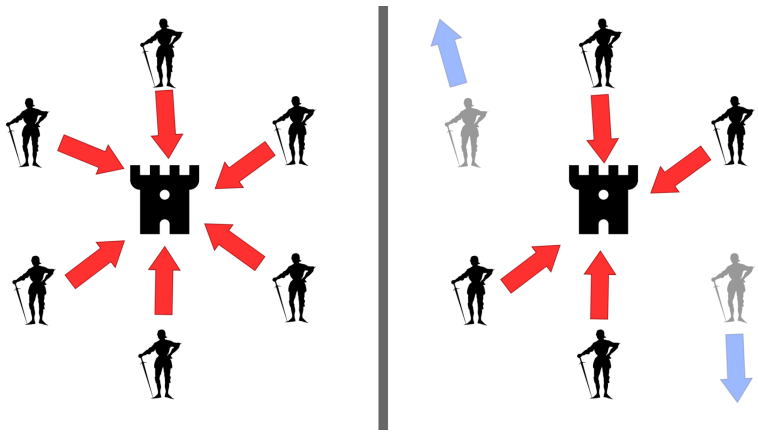


Figura 24: La situación que se presenta, donde los generales deben tomar una decisión común. La figura de la derecha muestra la situación donde algunos generales atacan mientras que los generales traidores no lo hacen. La imagen se extrajo de [34].

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

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

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

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

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

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

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

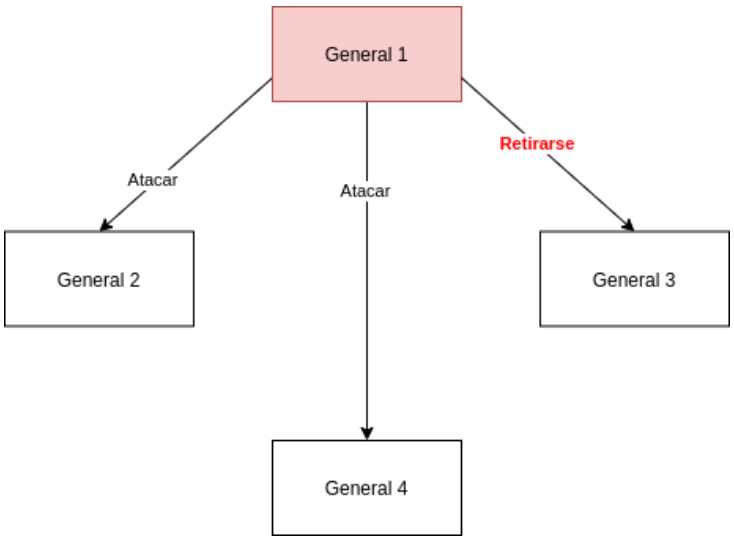


Figura 25: El general 1 es un traidor y le envía información conflictiva a los demás generales.

Como fue mencionado, para llegar a una decisión común, todos los generales deben conocer la opinión de los demás. El problema en este caso es que el General 1 envió una información difernte a sus pares. Para resolver esto, el algoritmo plantea realizar un segundo intercambio de mensajes como el de la figura 26.

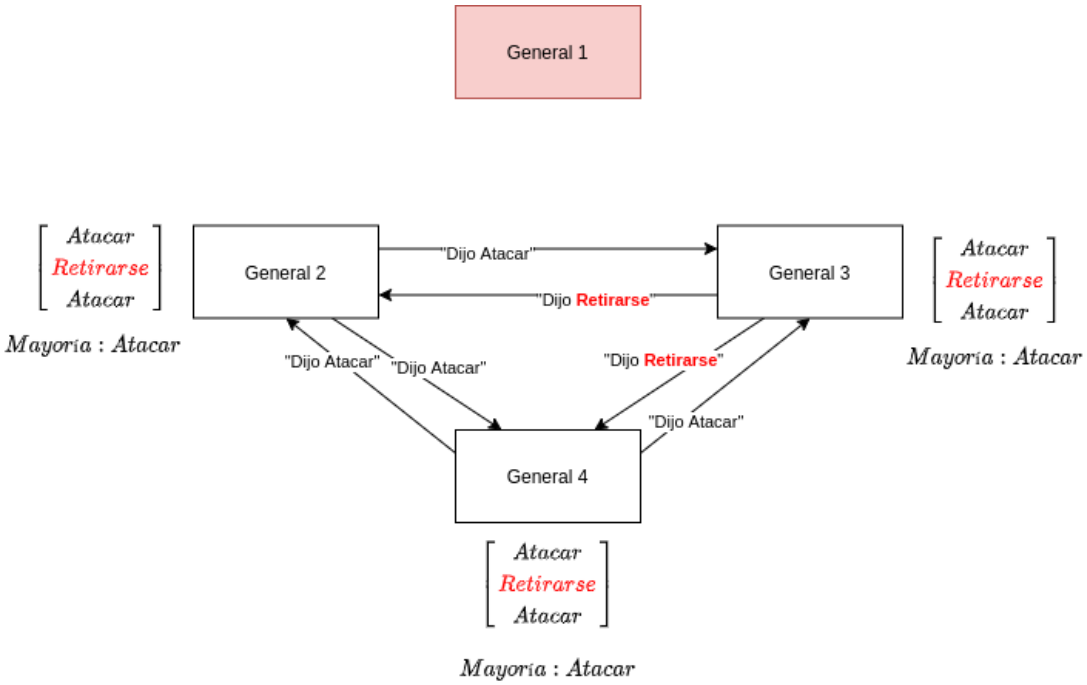


Figura 26: Se produce un intercambio entre los demás generales, para ponerse de acuerdo respecto de si el General 1 dijo “Atacar” o “Retirarse”.

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 [33], 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.

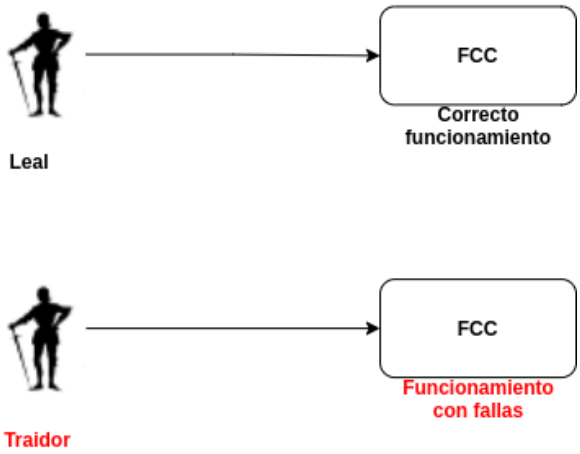


Figura 27: En el problema, un general leal representa un nodo, en este caso una computadora de vuelo, que funciona correctamente. Un General traidor es equivalente a una computadora de vuelo que presenta fallas.

Los generales traidores representan a las computadoras de vuelo que presentan fallas. En [33] 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 25, 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 28 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.

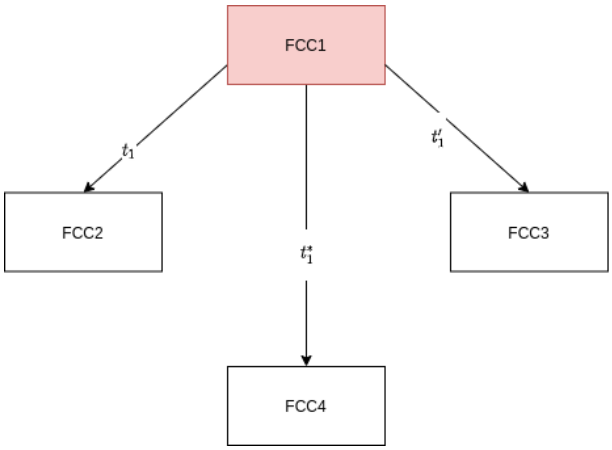


Figura 28: Debido a una falla, la computadora de vuelo 1 le entrega valores distintos de timestamp a las demás.

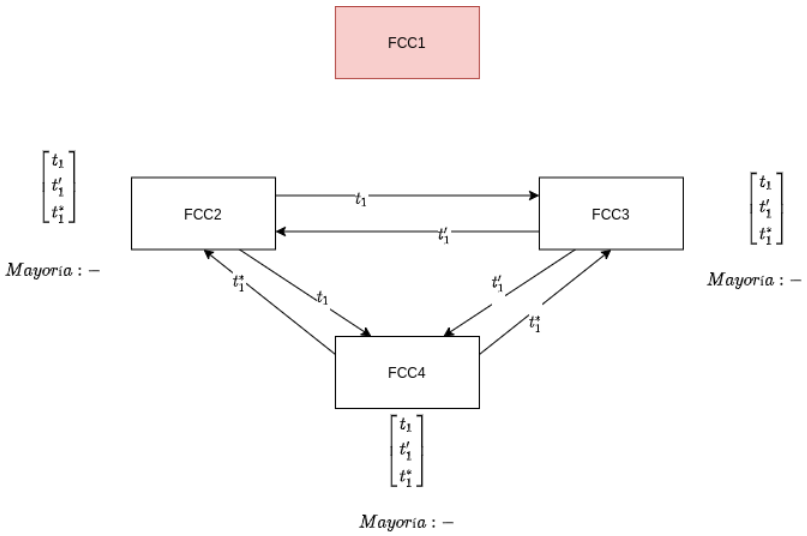


Figura 29: Debido a una falla, la computadora de vuelo 1 le entrega valores distintos de timestamp a las demás.

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 [20]. 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 [23] y costosa a su solución [21].

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[10] o FlexRay[35]. 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.

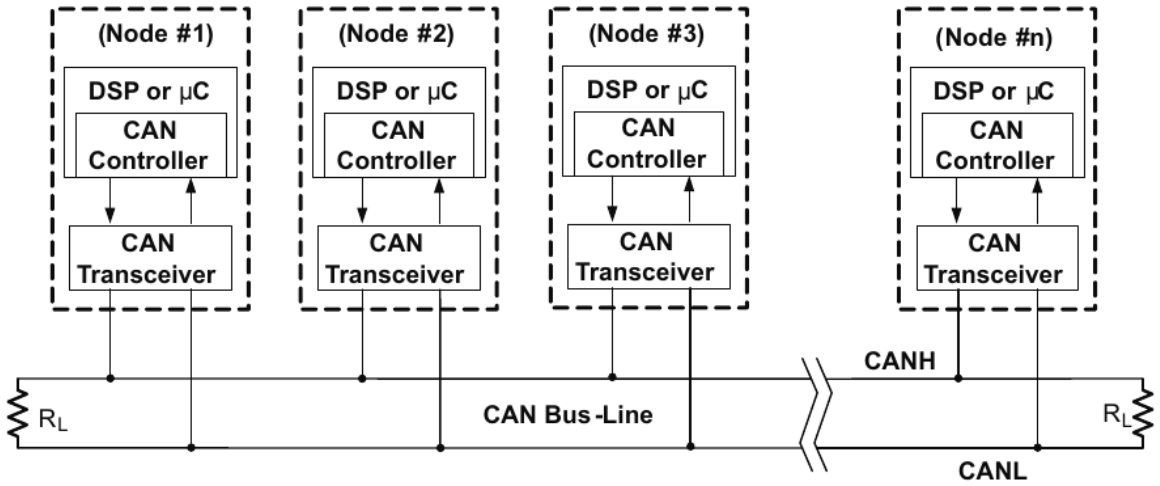


Figura 30: Todos los nodos se encuentran conectados al mismo bus de comunicaciones. En el caso del bus CAN, se compone de dos cables, CAN-H y CAN-L, terminados en sus extremos por resistencias de adaptación. La imagen se extrajo de [36].

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 31 con la figura 28.

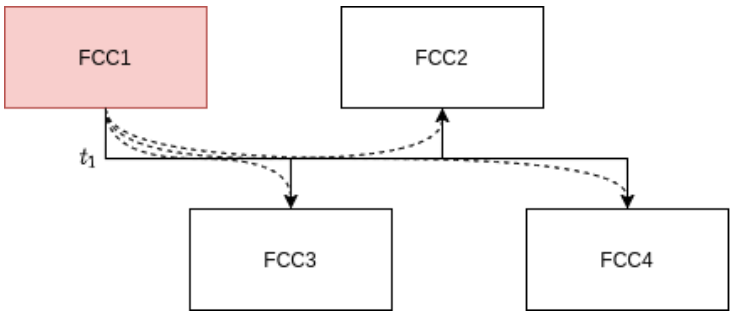


Figura 31: En este caso, la conexión tipo bus no permite el envío de información diferente a los demás miembros. La FCC1 envía el valor t_1 y todos sus pares reciben el mismo valor.

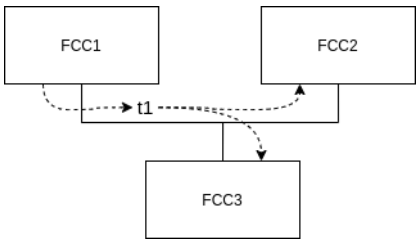
Como contrapartida, debido a que los nodos comparten canal de comunicación, estos deben tomar turnos para enviar la información a sus pares. De otra forma, habría una colisión en el bus y la información nunca llegaría a su destino. Sumado a esto, el bus se convierte en un punto singular de falla, ya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

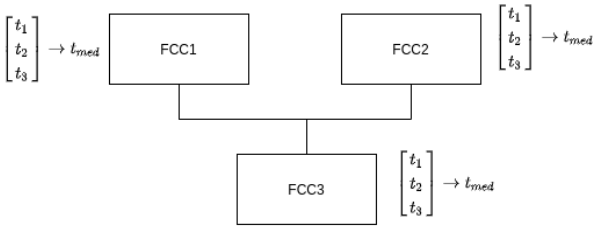
7.1.1. Consenso

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 32a, 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 32b 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 32: 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 [31]:

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

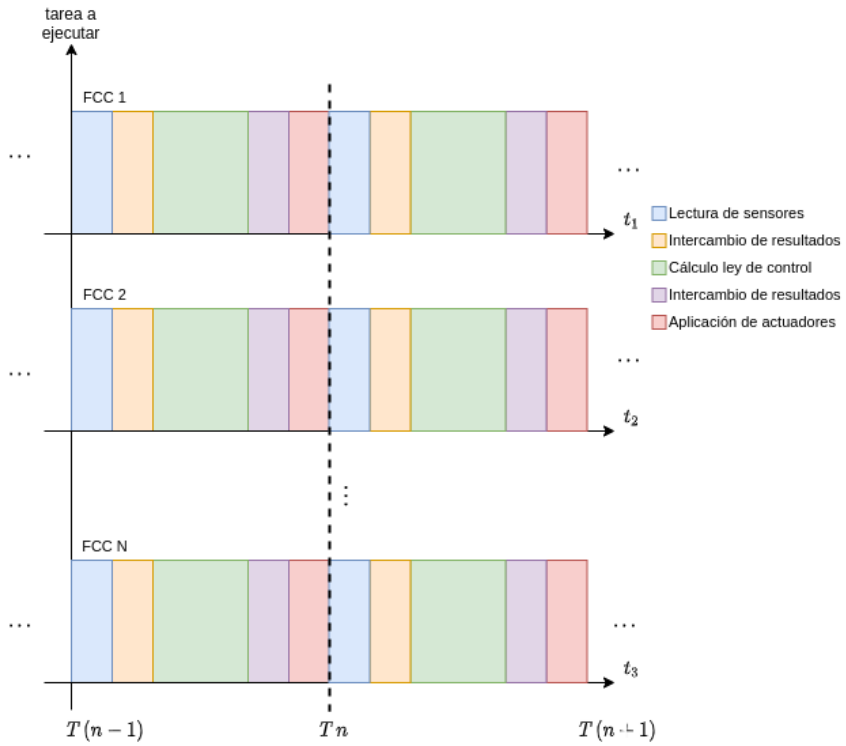


Figura 33: El eje horizontal representa el paso del tiempo. Las barras de colores representan el tiempo dedicado a ejecutar cada tarea, como la lectura de sensores, cálculo de la ley de control, etc., de forma periódica. En la imagen se puede ver que las FCC 1, FCC 2, ..., FCC N se encuentran sincronizadas 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 34 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.

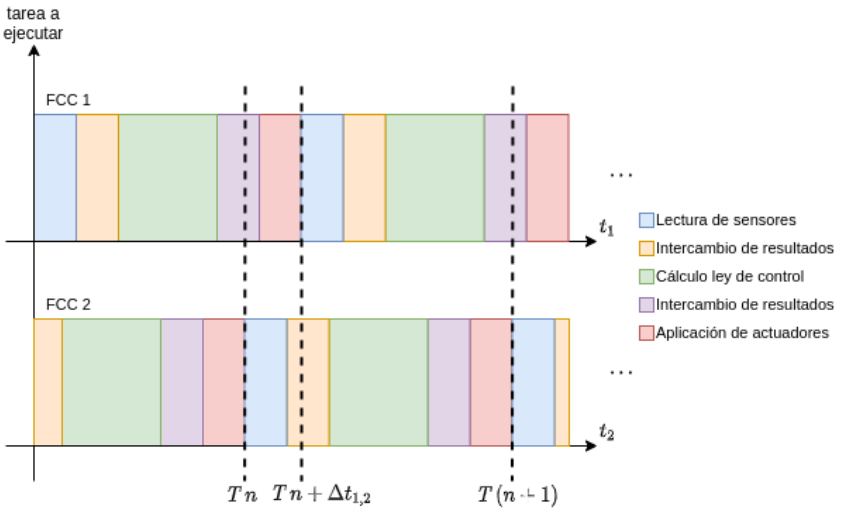


Figura 34: Se muestra un ejemplo para dos FCCs. A diferencia de la figura 33, las FCC 1 y la FCC 2 se encuentran desincronizadas.

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*[25], 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[25][37] y libros[38] 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.

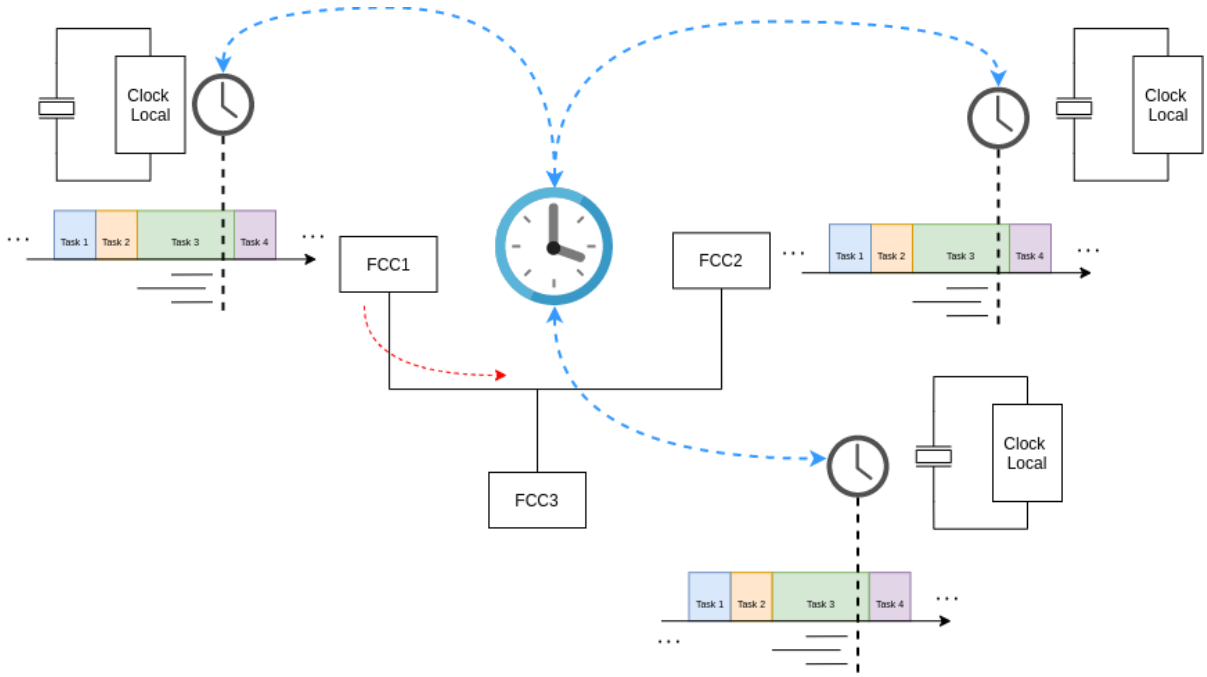


Figura 35: Se muestra un esquema que representa la arquitectura. Cada nodo tiene un *clock* local, funcionando a partir de su propio cristal. Todos estos a su vez se sincronizan periódicamente con el *global time*, representado por el reloj azul en el centro. En la imagen, el nodo 1 está enviando un 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**[38, 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 35 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 35 el nodo FCC1 presenta una falla y no envía el mesanje en el tiempo correspondiente, luego los nodos FCC2 y FCC3 no recibirán nada. Es común encontrar protocolos de comunicación donde este tipo de fallas se resuelven solicitando el reenvío del mensaje. Sin embargo, debido a que el sistema ya tiene un scheduling predefinido, esto no se permite ya que uno a priori no puede saber si habrá que hacer un pedido de reenvío de mensaje o no, lo que puede corromper el scheduling del sistema. Justamente, la *Time-Triggered Architecture* busca que el sistema sea predecible.

A continuación, se describe brevemente cada uno de los componentes de este tipo de sistema. Para una explicación más detallada, referirse a [38] y [25].

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 36 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.

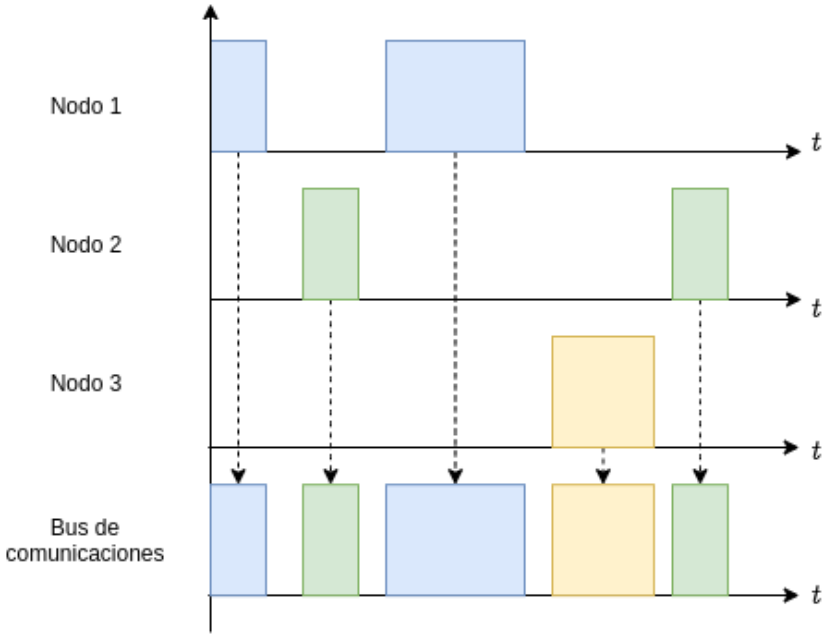


Figura 36: El ejemplo muestra como los 3 nodos pueden compartir el bus de comuncaciones. Cada uno de ellos sabe en qué instante de tiempo enviar un mensaje y en qué instantes de tiempo no deben hacerlo y solamente pueden escuchar el bus. Para que esto funcione adecuadamente, los nodos deben estar sincronizados.

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

En [25] se define un elemento del nodo denominado *Communication Network Interface*(CNI). Esta es una interfaz entre el software del nodo y el acceso al medio físico. Utilizando el protocolo de acceso al medio TDMA, el software del nodo *pushea* un mensaje a la CNI. Es esta última la que se encarga de administrar los tiempos de envío y recepción. Es decir, mientras el nodo continua con sus tareas, la CNI se encarga de cumplir con el timing del envío del mensaje.

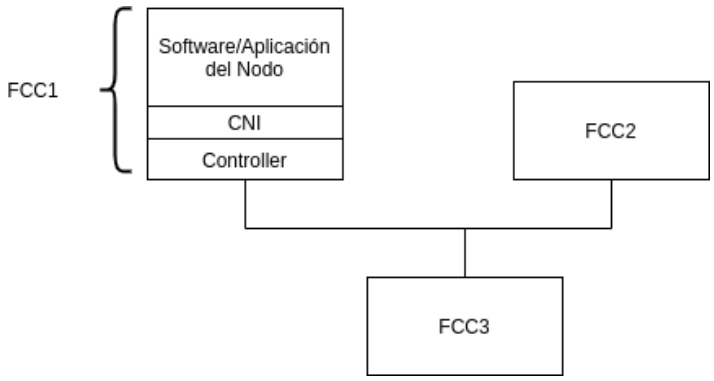


Figura 37: Misma imagen que 32a. Se muestra el detalle de los nodos, en el nodo FCC1.

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

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 describie 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 telófono celular, el cual tiene una pantalla que el usuario puede presionar en distintos lugares para abrir una aplicación, enviar un mensaje, etc. Cuando esto ocurre, el celular debe dar una respuesta casi inmediata. Este tipo de sistemas se llaman *Event-Triggered* y son controlados por los eventos que pueden ocurrir. A priori, se asumen eventos asincrónicos, es decir, que pueden ocurrir en cualquier momento y en cualquier orden. A diferencia de esto, en un sistema *Time-Triggered* los eventos no pueden ocurrir en cualquier momento. Mejor dicho, los eventos pueden ocurrir en cualquier momento, pero el sistema solamente prestará atención a esos eventos en un lapso de tiempo predefinido.

En los sistemas operativos de tiempo real se definen una serie de tareas, las cuales utiliza un scheduler que determina cuál es la próxima tarea que corresponde ejecutar. Un sistema *Time-Triggered* también define su comportamiento a través de tareas. La diferencia está en la estrategia utilizada por el scheduler. En el caso de un RTOS es común encontrar schedulers *preemptive* con un sistema de prioridad. En un sistema *Time-Triggered*, los schedulers pueden ser *preemptive* o bien cooperativos. La ventaja de utilizar un scheduler cooperativo es nuevamente el determinismo en la ejecución de las tareas [39, 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) [40] 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*[38, p. 51]. Esta se define como un acuerdo entre los nodos de la red respecto a una base de tiempo a utilizar como referencia. Como ya fue mencionado, cada nodo tiene un reloj interno propio, implementado con un cristal oscilador. Esta base de tiempos global puede implementarse por ejemplo utilizando un contador local. Cada nodo incrementa en uno el valor de esta variable, de forma periódica. Para que los nodos puedan utilizar esta variable como base de tiempos, todos los nodos deben tener el mismo número almacenado en su instancia local de dicha variable, al mismo tiempo.

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

Luego de ejecutar el algoritmo de sincronización, los clocks de cada nodo quedarán sincronizados con cierta precisión Φ. A medida que pase el tiempo, inevitablemente los clocks de cada nodo comenzarán a desfasarse uno del otro, lo cual incrementará el error. Por consiguiente, la sincronización debe ejecutarse de forma periódica. Esto se grafica en la figura 38.

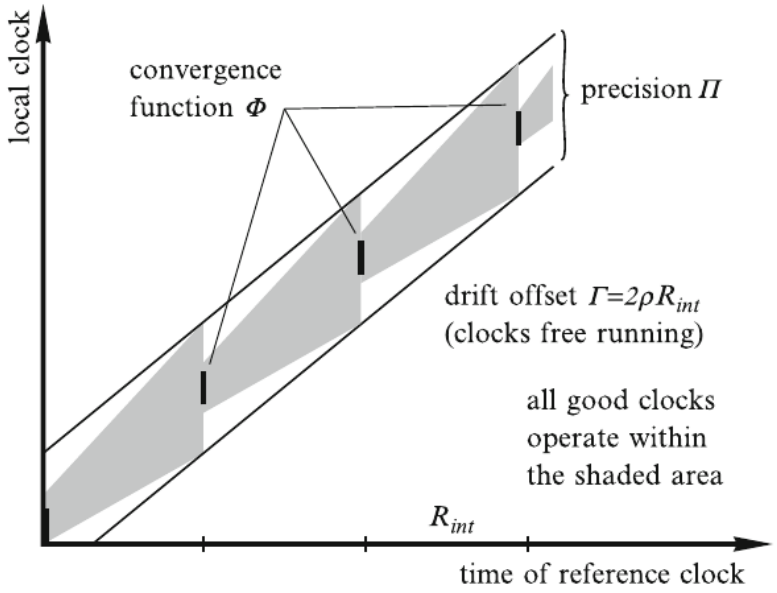


Figura 38: El eje horizontal representa el paso del tiempo físico y el eje vertical el avance de cada clock local a cada nodo. El valor R_{int} corresponde al período de resincronización. La imagen se extrajo de [38, p. 67].

Existen muchísimos algoritmos de resincronización. En [41] 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 39, donde lo que varía es la implementación de cada bloque.

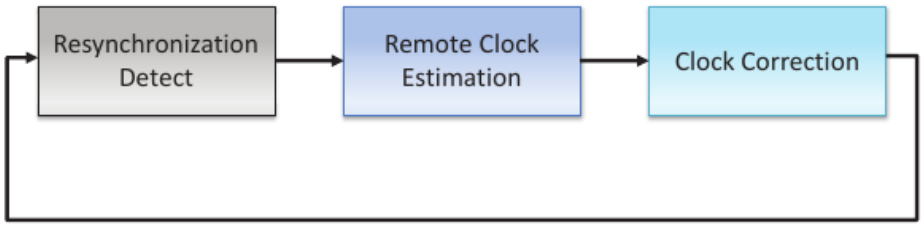


Figura 39: Tres bloques que comprenden un algoritmo de resincronización. La imagen se extrajo de [42].

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

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

[3] 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.

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

[5] A. C. Lapadatu y H. Jakobsen. «Anodic Bonding». En: *Handbook of Silicon Based MEMS Materials and Technologies (pp. 599-610)* (2015), págs. 599-610.

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

[7] Mustafa Cavcar. «The international standard atmosphere (ISA)». En: *Anadolu University, Turkey* 30.9 (2000), págs. 1-6.

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

[9] 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).

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

[11] *A CAN Physical Layer Discussion*. AN 228. Microchip. 2002.

[12] *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.

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

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

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

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

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

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

[19] Victor P. Nelson. «Fault-tolerant computing: Fundamental concepts». En: *Computer* 23.7 (1990), págs. 19-25.

[20] 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.

[21] 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.

[22] 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.

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

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

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

[26] 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.

[27] 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.

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

[29] 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.

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

[31] 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.

[32] 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.

[33] 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.

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

[35] *MPC5744P FlexRay Interface in Pictures*. AN12233. Rev. 0. NXP Semiconductors. Mayo de 2021.

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

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

[38] 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`.

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

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

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

[42] 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).