Universidad de Sevilla

Escuela Politécnica Superior

TRABAJO FIN DE GRADO EN INGENIERÍA ELÉCTRICA E INGENIERÍA ELECTRÓNICA INDUSTRIAL

DISEÑO, TESTEO Y VALIDACIÓN DEL SUBSISTEMA OBC EN UN CUBESAT

TUTOR:

AUTOR: ÁLVARO PÉREZ MÁRQUEZ

Sevilla, mes de año

Trabajo Fin de Grado: Diseño, testeo y validación del subsistema OBC en CubeSat.

Tutor:

Autor: Álvaro Pérez Márquez

Tribunal nombrado:

Presidente:

Vocales:

Secretario:

Acuerdan otorgar la calificación de:

El secretario del Tribunal:

Fecha:

Contenido

LISTA DE ACRÓNIMOS 5
INDICE DE FIGURAS 6
INDICE DE TABLAS 7
1. INTRODUCCIÓN 8
1.1. CONSIDERACIONES 9
1.2. SUBSISTEMAS 10
2. DISEÑO 10
2.1. FUNCIONES: 10
2.2. FIRMWARE: 11
2.3. PINOUT: 19
3. TESTEO 20
4. VALIDACIÓN 20
BIBLIOGRAFÍA. 20

RESUMEN

El espacio es la última barrera que le falta al ser humano por conquistar, y por ello las distintas agencias espaciales de nuestro planeta crean programas para estudiar distintos aspectos del espacio, y por ello desde nuestra asociación aportamos nuestro grano de arena a este gran ecosistema formado por otras asociaciones y empresas.

Este trabajo que tiene ante usted, pretende servir como documento base e introductorio para los futuros diseñadores de Fycus, otras asociaciones o personas que de forma privada deseen crear un subsistema como este. Se describen los razonamientos, criterios y métodos usados en los años 2025 y 2026 para el diseño del software de control del subsistema OBC, pasando por conocimientos generales. Además, a lo largo de este trabajo se dan pinceladas generales de otros subsistemas del cubesat de los que se compone, intentando lograr un mayor entendimiento del lector y sobre todo para que sirva como documento introductorio a las futuras generaciones.

LISTA DE ACRÓNIMOS

OBC:	Computador de a bordo.
CubeSat:	Satélite con medidas estandarizadas.
FSW:	Software de vuelo.
GPIO:	Entradas/Salidas de propósito general.
I^2C:	Circuito inter-integrado.
SPI:	Interfaz periférica serie.
UART:	Transceptor asíncrono universal.
ADC:	Convertidor analógico-digital.
CAN:	Controlador de red de área.

INDICE DE FIGURAS

Ilustración 1 ..... 1**¡Error! Marcador no definido.**
Ilustración 2 ..... 1**¡Error! Marcador no definido.**
Ilustración 3 20

INDICE DE TABLAS

Tabla 1 9
Tabla 2 11
Tabla 3 11
Tabla 4 12
Tabla 5 12

1. INTRODUCCIÓN

Los avances tecnológicos nos han llevado a la creación de satélites más pequeños que pueden ser fácilmente sostenidos y manejados por una sola persona. Una categoría popular de estos satélites más pequeños es la categoría de nanosatélites (que pesa entre 1 y 10 KG). Un tipo de nanosatélite aún más popular es el Cube Satellite (CubeSat), un satélite miniaturizado construido a partir de cubos estandarizados, el nombre proviene de las unidades cúbicas utilizadas para construir el satélite.

Cualquier CubeSat puede tener el tamaño de 1 Unidad (1U), o múltiplos de dichas unidades donde cada unidad mide 10 x 10 x 10 cm. Estas unidades suelen tener un poco más de 1 kilogramo de masa por unidad. Los CubeSats fueron desarrollados por primera vez en 1999 por la Universidad Politécnica Estatal de California y la Universidad de Stanford y desde entonces han crecido en popularidad debido a su bajo costo y flexibilidad. Un CubeSat se puede construir rápidamente con componentes y componentes electrónicos desarrollados internamente o comerciales disponibles en el mercado.

Los CubeSats se utilizan normalmente para investigación espacial y uso comercial. Esto significa que los desarrolladores con experiencia limitada o nula en tecnología satelital lejos de las grandes agencias espaciales, como estudiantes de universidades y centros de investigación o incluso entusiastas, ahora pueden explorar los reinos del espacio con CubeSats. La estandarización y adopción del hardware CubeSat en el ámbito académico y la industria ha llevado a un aumento de los volúmenes de producción de componentes disponibles comercialmente. Si bien esto ha resultado en una reducción significativa en el tiempo y el costo de desarrollo de satélites, todavía se dedica una cantidad considerable de tiempo y esfuerzo de desarrollo de misiones al desarrollo de software de vuelo (FSW), a menos que el código sea reutilizable. Al explorar el dominio del software de vuelo CubeSat (FSW), este trabajo realiza primero una revisión exhaustiva de la literatura para examinar las prácticas de desarrollo de FSW existentes, las metodologías de diseño y los desafíos de integración inherentes a las misiones CubeSat.

A través de este examen detallado, identificamos deficiencias críticas en los enfoques actuales, particularmente en términos de modularidad, reutilización y eficiencia. Estas deficiencias ponen de relieve la necesidad de un marco más racionalizado que pueda adaptarse a los rápidos avances en la tecnología espacial y a los requisitos de las misiones. El análisis de los desarrollos de software de vuelo (FSW) existentes revela una notable ausencia de varias características clave, consideradas críticas para la eficacia y el éxito de las misiones CubeSat. Estos elementos faltantes y sus impactos tanto en las FSW como en los objetivos generales de la misión se identifican y resumen como se muestra en la Tabla 1.

Características esenciales que a menudo faltan en las soluciones FSW actuales.	Impacto en las FWS y la misión.
Modularidad y arquitectura orientada a servicios.	Limita el potencial de reutilización de software en misiones posteriores, lo que genera mayores esfuerzos y costos de desarrollo.
Arquitectura multicapa.	Disminuye la abstracción y portabilidad del software, lo que afecta negativamente en la modularidad y la capacidad de adaptar el software en diferentes contextos de misión.
Utilización de FreeRTOS.	La preferencia por sistemas operativos más sofisticados requiere la implementación de hardware más robusto, lo que potencialmente aumenta los costos y la complejidad de la misión.
Implementación de una interfaz de línea de comandos.	Complica los procedimientos de prueba y el desarrollo de motores de script eficientes, lo que potencialmente dificulta la agilidad operativa.
Incorporación de un motor de scripts.	Restringe la versatilidad de los sistemas de control de misiones para su uso en misiones futuras, lo que limita la longevidad y adaptabilidad del software.
Integración de un gestor de arranque.	La ausencia de capacidades de actualización o parcheo de software dentro de la misión hace que cualquier problema de software posterior al lanzamiento sea intratable, con el riesgo de que la misión falle

Tabla 1. Falta de características esenciales en las soluciones existentes y su impacto en el éxito de la misión.

1.1. CONSIDERACIONES

La finalidad de este documento es la de servir como base para futuras actualizaciones de este sistema, en la asociación Fycus.

Además de servir de punto de inicio para otras personas que se quieran diseñar este sistema de control y adaptarlo a sus necesidades.

Teniendo todo esto en cuenta, este trabajo y su software estará disponible en mi cuenta de github, que indico a continuación:

github.com/Varo-0/OBC

1.2. SUBSISTEMAS

En este apartado daré una breve explicación de los distintos subsistemas de los que se compone el cubesat que se desarrolla en la asociación:

- OBC – On Board Computer

  Este subsistema se encarga de ejecutar firmware, gestionar la telemetría/telecomandos, tareas de la misión y de la comunicación de los datos entre los otros subsistemas. En nuestro caso también realizará las tareas de lectura de sensores, determinación de altitud y actitud para el guiado del dispositivo y almacenamiento de datos en una memoria FRAM.

- EPS – Electrical Power Subsistem

  Se encarga de la generación, regulación, almacenamiento y distribución de la energía eléctrica, informando al OBC de los distintos niveles de voltaje y corriente.

- ADCS – Attitude Determination & Control System

  Determina la actitud, por medio de sensores como IMU y magnetómetro, controlando un volante de inercia o thrusters. En nuestro caso solo controlará el volante de inercia, gracias a las consignas dadas por el OBC.

- COMMS/TT&C – Comunications & Telemetry

  Realiza el enlace con tierra para el envío de telemetría y la recepción de los telecomandos dados por el OBC, empaquetados mediante el protocolo CCSDS, que explicaré más adelante.

- Payload

  Se trata de la carga útil del dispositivo, cámaras, sensores científicos, payload radio. Todo supervisado por el OBC.

2. DISEÑO

Esquema general del sistema.

2.1. FUNCIONES

Las funciones que desempeñará el OBC serán:

- Control de actitud y orientación, mediante algoritmos para navegación en modo normal y emergencias.
- Recepción de telecomandos para modos de funcionamiento.
- Recopilación y envío de telemetría del Cubesat a tierra.
- Guardado de telemetría en memoria.

2.2. FIRMWARE

La idea tras el firmware del OBC es la de crear una arquitectura modular de capas software, formada por:

- Capa de bajo nivel:

  Esta capa se encarga de abstraer la comunicación de sensores (GPIO, I²C, SPI, UART, ADC), por medio de las librerías HAL y de los sensores para obtener funciones más eficientes y comprensibles para futuras modificaciones o actualizaciones.

- Capa protocolo:

  Esta capa será la encargada de la gestión de la comunicación con otras placas del CubeSat y el empaquetamiento de dicha información.

  Se utilizará el protocolo CAN para la comunicación interna, mediante la librería MCP_CAN suministrada por el departamento de Arquitectura y Tecnología de Computadoras de la Universidad de Sevilla. Y el protocolo Space Packet, para el envío de telemetría desde el CubeSat y la recepción de telecomandos desde tierra.

  A continuación, se muestran en la Tabla 2 las tramas desde el CubeSat a tierra y en la Tabla 3 las tramas desde tierra al CubeSat, mediante el protocolo Space Packet (CCSDS).

Emisor	Tipo	ID	Datos							
			D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
GPS	0	0x001	Tipo	ID	lat_1	lat_2	ing_1	ing_2	UTC	0xFF
IMU	0	0x002	Tipo	ID	roll_1	roll_2	pitch_1	pitch_2	yaw_1	yaw_2
BMP	0	0x003	Tipo	ID	temp_1	temp_2	press_1	press_2	0xFF	0xFF
AIR	0	0x004	Tipo	ID	tvoc_1	tvoc_2	temp_1	temp_2	humd_1	humd_2

Tabla 2.Tramas de datos para envío de telemetría.

Emisor	Tipo	ID	Datos							
			D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]

Tierra	1	0x010	Tipo	ID	deploy_1	deploy _2	0xFF	0xFF	0xFF	0xFF
Tierra	1	0x020	Tipo	ID	roll_1	roll_2	pitch_1	pitch_2	yaw_1	yaw_2
Tierra	1	0x030	Tipo	ID	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

Tabla 3.Tramas de recepción de datos de telecomandos.

Nota: Se ha de aclarar que Tipo = 0 es para designar la telemetría y Tipo = 1 para designar los telecomandos.

En la Tabla 4 se muestra las tramas CAN desde el OBC a los distintos y en la Tabla 5 las tramas CAN desde los distintos subsistemas al OBC.

Subsistema emisor	Subsistema receptor	Datos								
		ID	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
OBC	ADCS	0x010	PWM_Torque	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
OBC	COMMS	0x020	Tipo	ID	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
OBC	EPS	0x030	Switch_off	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
OBC	Payload	0x040	Start	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
OBC	Propulsión	0x050	On	PWM_Válvula	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF

Tabla 4. Tramas CAN desde el OBC.

Subsistema emisor	Subsistema receptor	Datos								
		ID	D[0]	D[1]	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
COMMS	OBC	0x010	Tipo	ID	D[2]	D[3]	D[4]	D[5]	D[6]	D[7]
EPS	OBC	0x020	Voltaje_bat	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF	0xFF
Propulsión	OBC	0x030	On	Temperatura	Presión	0xFF	0xFF	0xFF	0xFF	0xFF

Tabla 5. Tramas CAN desde los subsistemas.

- Capa de misión:

En esta capa se programarán las tareas, colas de mensajes, semáforos y candados necesarios que se utilizarán en la programación del OBC.

Todas las capas estarán supervisadas por un RTOS, en concreto FreeRTOS. Y gracias al CMSIS V2 (Cortex Microcontroler Software Interface Standard), podemos obtener una multitarea en tiempo real con planificador apropiativo y gestión del tiempo de ejecución de las tareas.

Las tareas se separarán en dos grupos; las tareas de modo normal, en las que el funcionamiento del CubeSat será normal. Y la tarea de modo de emergencia, en la que dependiendo de una condición de error, designada más adelante, se ejecutarán unos comandos de seguridad para que el CubeSat y los componentes sufran los menores daños posibles.

Las tareas se explicarán a continuación, mencionando su propósito y pasos a seguir por cada una de ellas.

- TC_listener:
    - Propósito: Recibir, validar y encolar telecomandos que se reciben por CAN, para ADCS_controler.

- Pasos:
  - Activar candado (Mutex_1).
  - Esperar datos de entrada.
  - Verificar datos.
  - Encolar datos de salida en la cola (Cola_1).
  - Desactivar candado (Mutex_1).
  - Generar fallo en caso de error.

- SC_generator:
  - Propósito: Lectura de sensores por I^2C, SPI, UART y empaquetar en una cola.
  - Pasos:
    - Activar candado (Mutex_1).
    - Lectura de sensores.
    - Encolar datos de salida en la cola (Cola_1).
    - Desactivar candado (Mutex_1).
    - Generar fallo en caso de error.

- ADCS_controller:
  - Propósito: Obtención de valores para mantener orientación deseada, de los datos suministrados por la cola.
  - Pasos:
    - Recepción de datos de la cola (Cola_1).
    - Obtención de valores para control de actitud y orientación, mediante filtro de Kalman.
    - Encolar datos de salida en la cola (Cola_2).
    - Generar fallo en caso de error.

- ADCS_CAN_Tx:
  - Propósito: Lectura de los datos de la cola, para ser transmitidos por CAN al módulo del adcs.
  - Pasos:
    - Recepción de datos de la cola (Cola_2).

        - Activar candado (Mutex_2).
        - Transmisión de datos por CAN.
        - Desactivar candado (Mutex_2)
        - Generar fallo en caso de error.

- Memory:
    - Propósito: Almacenar los datos en memoria del vuelo, para su tratamiento en tierra.
    - Pasos:
        - Recepción de datos de la cola (Cola_2).
        - Activar candado (Mutex_2).
        - Escritura de datos.
        - Desactivar candado (Mutex_2).
- Ground_CAN_Tx:
    - Propósito: Extraer datos para telemetría de la cola y enviarla por CAN al módulo del emisor.
    - Pasos:
        - Recepción de datos de la cola (Cola_2).
        - Activar candado (Mutex_2).
        - Transmisión de datos por CAN.
        - Desactivar candado (Mutex_2)
        - Generar fallo en caso de error.
- safe_mode:
    - Propósito: En caso de fallo, suspende tareas no esenciales y entra en modo seguro.
    - Condición de activación del modo: Detección de fallo en dos o más tareas.
    - Pasos:
        - Suspensión de todas las tareas.
        - Desactivar sensores excepto IMU.
        - Bucle:

- Rutina preestablecida.
- Actuación sobre actuadores.
- Verificar condición de recuperación.

A continuación, podrá ver en la siguiente imagen el flujo de trabajo de las tareas

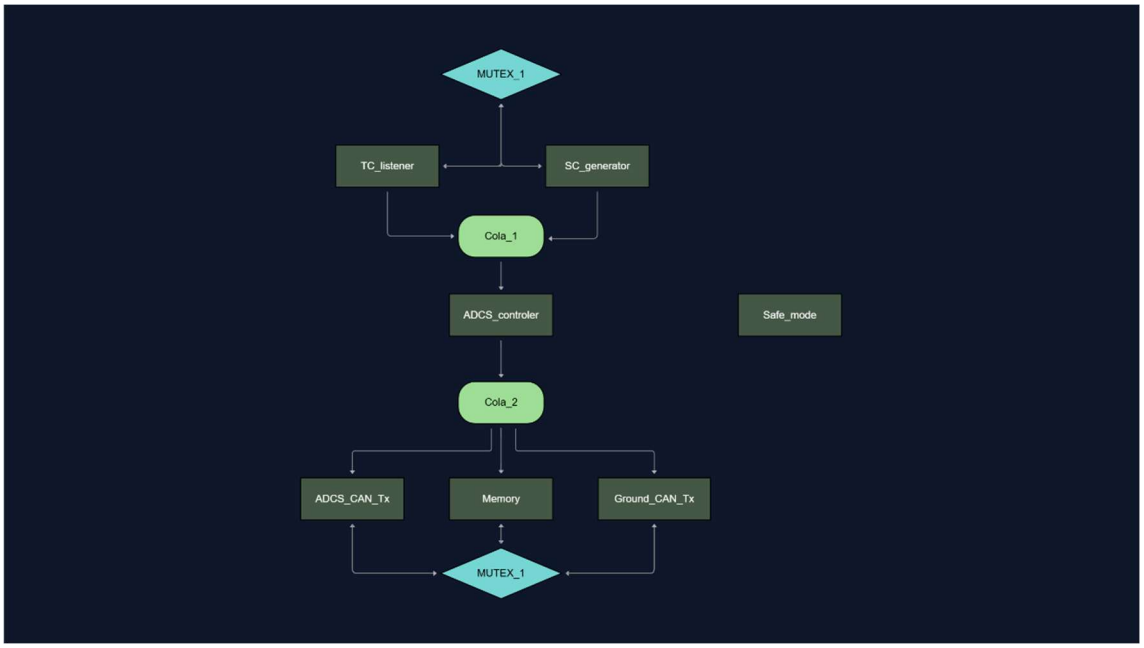

Ilustración 1. Flujo de trabajo de las tareas del OBC.

En la taréa ADCS_controller, he mencionado la utilización del filtro de Kalman y a continuación realizaré una breve explicación de su funcionamiento y programación para mi aplicación.

El filtro de Kalman es un filtro que permite la fusión en la medición de varios sensores, junto al ruido que producen dicho sensores o interferencias externas, para poder obtener un valor minimizando dichas componentes de ruido. Este filtro permite obtener una predicción de un valor, basado en una medición en el tiempo actual y el valor que le demos a la covarianza de los ruidos asiciados; ruido del proceso y ruido de medición, siguiendo un algoritmo de predicción-actualización. Como se verá por la nomenglatura, este filtro de Kalman será de tiempo discreto, debido a la naturaleza de la medición de los sensores.

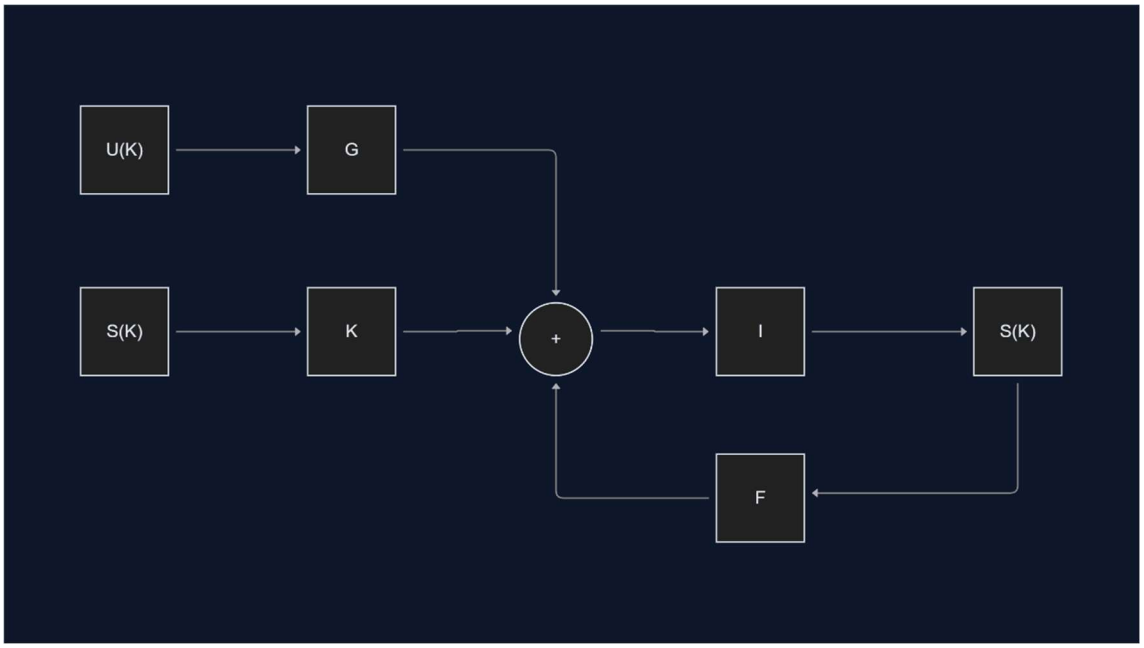

Ilustración 2. Diagrama de bloques filtro de Kalman.

1.- Predicción del estado actual.

En este proceso vemos que estado es más probable que tenga nuestro sistema:

$$S(K) = F * S(K-1) + G * U(K)$$

Siendo;

S(K) ≡ Vector de estado = $\begin{bmatrix} Altitud_K \\ Velocidad_K \end{bmatrix}$

F ≡ Matriz de transición de estados = $\begin{bmatrix} 1 & Ts \\ 0 & 1 \end{bmatrix}$

S(K-1) ≡ Vector de estado anterior = $\begin{bmatrix} Altitud_{K-1} \\ Velocidad_{K-1} \end{bmatrix}$

G ≡ Matriz de control = $\begin{bmatrix} 0{,}5 * Ts^2 \\ Ts \end{bmatrix}$

U(K) ≡ Variable de entrada = Acc_z,inertial(K)

El valor de Ts inicialmente será de 0,004 s, pero podría ser distinto si el testeo y validación no son óptimas. El valor de Acc_z,inertial(K), se determinará a partir de las siguientes ecuaciones:

Acc_z,inertial(K) = ($Acc_{z,inertial}$ - 1) * 9,81 * 100

$Acc_{z,inertial}$ = -A_{CCX} * Sen(Θ_{pitch}) + A_{CCY} * Sen(Ψ_{roll}) * Cos(Θ_{pitch}) + A_{CCZ} * Cos(Ψ_{roll}) * Cos(Θ_{pitch})

Θ_{pitch} = -arctg ($\frac{ACCX}{\sqrt[2]{ACCY^2 + ACCZ^2}}$)

Ψ_{roll} = arctg ($\frac{ACCY}{\sqrt[2]{ACCX^2 + ACCZ^2}}$)

2.- Covarianza de la predicción.

Se establece cuanto de cerca estamos del valor objetivo:

$$P(K) = F * P(K-1) * F^T + Q$$

Siendo;

P(K) ≡ Vector de covarianza = P(K=0) = $\begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$

Q ≡ Covarianza del proceso = G * G^T *10^2

Nota: El factor de 10^2 en la covarianza del proceso, dependerá de la desviación estandar de la medición. Inicialmente tomaremos este valor, pero podrá variar en caso de ser necesario tras los pasos de testeo y validación.

3.- Ganancia de Kalman.

Otro factor a tener en cuenta para determinar la exactitud de las mediciones.

$$L(K) = H * P(K) * H^T + R$$

$$K = P(K) * \frac{H^T}{L(K)} = P(K) * H^T * L(K)^{-1}$$

Siendo;

L(K) ≡ Matriz intermedia

K ≡ Ganancia de Kalman

H ≡ Matriz de observación = $\begin{bmatrix} 1 & 0 \end{bmatrix}$

R ≡ Covarianza medida = 30^2

Nota: El factor de 30^2 en la covarianza del proceso, dependerá de la desviación estandar de la medición. Inicialmente tomaremos este valor, pero podrá variar en caso de ser necesario tras los pasos de testeo y validación.

4.- Actualizar la predicción del estado actual.

Debemos calcular el nuevo estado, para determinar la nueva posición.

$$S(K) = S(K) + K * (M(K) - H * S(K))$$

Siendo;

M(K) ≡ Vector de medida

5.- Actualizar covarianza.

Colculamos la nueva covarianza del sistema para actualizar la medida de cuanto de exacta es la medida.

$$P(K) = (I - K * F) * P(K)$$

I ≡ Matriz identidad = $\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$

Una vez entendidas las ecuaciones y pasos para realizar en el filtro de Kalman, solo faltaría implementarlas en su tarea, junto a las ecuaciones que calibran los sensores.

Todo este firmware seguirá la siguiente estructura:

Core/
- Inc/
  - neo-m8m.h
  - ubx.h
  - nav_pvt.h
  - lsm6dso.h
  - lis2mdltr.h
  - mcp9804.h
  - tcan332g.h
  - thvd2450.h
  - fm25v10-gtr.h
  - rf317990687.h
- Src/
  - neo-m8m.c
  - ubx.c
  - cfg_payload.c
  - nav_pvt.c
  - lsm6dso.c
  - lis2mdltr.c
  - mcp9804.c
  - tcan332g.c
  - thvd2450.c
  - fm25v10-gtr.c
  - rf317990687.c
  - main.c

2.3. PINOUT

En la organización de pines, se ha tenido en cuenta principalmente la organización dada por el prototipo hardware, en el que primaba la optimización del espacio dado el poco espacio que disponemos en el cubesat

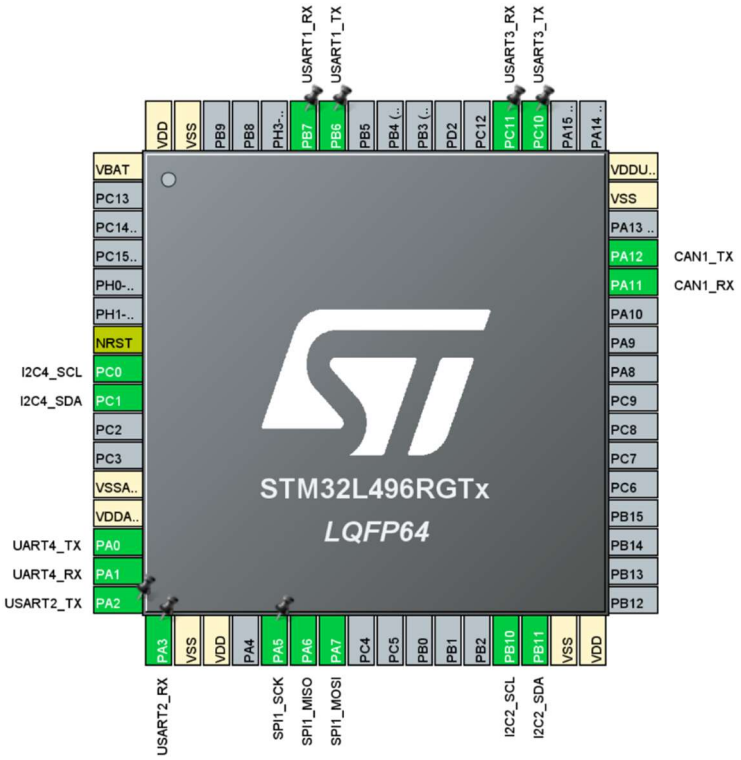

Ilustración 3. Esquema organización de pines

3. TESTEO

Test en laboratorio

Datos del test

Mediciones del test

4. VALIDACIÓN

Validación de vuelo

Datos de vuelo

BIBLIOGRAFÍA.

- https://github.com/spel-uchile/SUCHAI (SUCHAI Cubesat Flight Software).
- https://www.masat.space/en/projektek/masat-1/muszaki-bemutatas/ (Masat-1 FSW).
- https://github.com/SFUSatClub/obc-firmware?utm (Firmware OBC).
- Documentación interna de la asociación Fycus.

- CubeSat Flight Software: Insights and a Case Study written by Mohammed Eshaq*, M. Sami Zitouni†, and Shadi Atalla University of Dubai, Dubai, P. O. Box: 14143, United Arab Emirates, Saeed Al-Mansoori Mohammed Bin Rashid Space Center (MBRSC), Dubai, P. O. Box: 211833, United Arab Emirates, Malcolm Macdonald University of Strathclyde, Glasgow, G1 1XQ, United Kingdom.

- SmallSat Avionics written by NASA.

- State-of-the-Art Small Spacecraft Technology written by Sasha V Weston, Millennium Engineering and Integration Services, an Astrion Subsidiary, Craig D Burkhard, Jan M Stupl, Rachel L Ticknor, Bruce D Yost, Ames Research Center, Moffett Field, California, Rebekah A Austin, Pavel Galchenko, Lauri K Newman, Luis Santos Soto, Goddard Space Flight Center, Greenbelt, Maryland.