Uncrewed Aerial Vehicle Flight Control Module Specifications Document

Tanner Chase

Christian Hall

Anthony Wood

November 3, 2025

Signature Page

Signature	*Signature*	*Signature*
Date and email	*Date and email*	*Date and email*

Revision History

Revision	Description	Author	Date	Approval
1				
2				
3				
4				
5				
6				
7				
8				
9				
10				

Contents

Specifications

1 SCOPE

(a) **General:** This document defines the design and verification requirements for the Uncrewed Aerial Vehicle (UAV) Flight Control Module (FCM). The system provides the core functionality required for stable, autonomous, and remote operation of a fixed-wing aircraft. It manages flight stabilization, navigation, control, and mission execution through integrated sensing, actuation, and computation. The FCM shall support interfacing with an external companion system for higher-level mission planning and enable autonomous operation in communication-limited environments, including navigation to a designated area and precision landing.

(b) **Acronyms:**

(i) UAV: Uncrewed Aerial Vehicle

(ii) FCM: Flight Control Module

(iii) HAL: Hardware Abstraction Layer

(iv) CAN: Controller Area Network

(v) GPS: Global Positioning System

(vi) IMU: Inertial Measurement Unit

(vii) PVA: Position/Velocity/Acceleration

(viii) ATL: Attitude/Throttle-Level

(ix) SD: Secure Digital

(x) ICD: Interface Control Document

(xi) API: Application Programming Interface

(xii) GPIO: General Purpose Input/Output

(xiii) I2C: Inter-Integrated Circuit

(xiv) SPI: Serial Peripheral Interface

(xv) UART: Universal Asynchronous Receiver-Transmitter

(xvi) USB: Universal Serial Bus

(xvii) CAN: Controller Area Network

(xviii) ADC: Analog-to-Digital Converter

(xix) CPU: Central Processing Unit

2 APPLICABLE DOCUMENTS

The following documents shown shall form part of the specifications for this project. In the event of a conflict between requirements, priority shall first go to the contract, second to this document, and lastly to these reference documents.

(a) **Government Documents**

(i) **FAA 14 CFR Part 107 - Small Unmanned Aircraft Systems**
Regulatory Body: FAA (eCFR.gov)
Revision/Date: Current as of 2025
Link: `https://www.ecfr.gov/current/title-14/chapter-I/subchapter-F/part-107`

(ii) **FAA 14 CFR Part 89 - Remote Identification of Unmanned Aircraft**
Regulatory Body: FAA (eCFR.gov)
Revision/Date: Current as of 2025
Link: `https://www.astm.org/f2910-21.html`

(iii) **FCC 47 CFR Part 15 - Radio Frequency Devices**
Regulatory Body: FCC (ecfr.gov)
Revision/Date: 2025
Link: `https://www.ecfr.gov/current/title-47/chapter-I/subchapter-A/part-15`

(iv) **FCC 47 CFR Part 97 - Amateur Radio Service**
Regulatory Body: FCC (ecfr.gov)
Revision/Date: 2025
Link: `https://www.ecfr.gov/current/title-47/chapter-I/subchapter-D/part-97`

(v) **MIL-STD-810 - Environmental Engineering Considerations and Laboratory Tests**
Regulatory Body: U.S. DoD (quicksearch.dla.mil)
Revision/Date: Revision H, January 2019
Link:
`https://quicksearch.dla.mil/qsDocDetails.aspx?ident_number=36060`

(b) **Industry Documents**

(i) **RFC 9293 - Transmission Control Protocol (TCP)**
Regulatory Body: IETF (ietf.org)

Revision/Date: August 2022
Link: https://www.rfc-editor.org/rfc/rfc9293.html

(ii) **RFC 768 - User Datagram Protocol (UDP)**
Regulatory Body: IETF (ietf.org)
Revision/Date: August 1980
Link: https://www.rfc-editor.org/rfc/rfc768.html

(iii) **RFC 791 - Internet Protocol (IPv4)**
Regulatory Body: IETF (ietf.org)
Revision/Date: September 1981
Link: https://www.rfc-editor.org/rfc/rfc791.html

(iv) **TIA/EIA-232-F - Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment (RS-232)**
Regulatory Body: TIA/EIA (tiaonline.org)
Revision/Date: 1997
Link: https://global.ihs.com/doc_detail.cfm?item_s_key=00012803

(v) **NXP UM10204 - I^2C-bus Specification and User Manual**
Regulatory Body: NXP Semiconductors (nxp.com)
Revision/Date: Rev. 6, April 2014
Link: https://www.nxp.com/docs/en/user-guide/UM10204.pdf

(vi) **USB 2.0 Specification**
Regulatory Body: USB-IF (usb.org)
Revision/Date: April 2000
Link: https://www.usb.org/document-library/usb-20-specification

(vii) **ISO 11898-1:2015 - CAN Data Link Layer and Physical Signaling**
Regulatory Body: ISO (iso.org)
Revision/Date: 2015
Link: https://www.iso.org/standard/63648.html

(viii) **SD Physical Layer Simplified Specification v6.00**
Regulatory Body: SD Association (sdcard.org)
Revision/Date: 2017
Link: https://www.sdcard.org/downloads/pls/simplified_specs/

(ix) **MISRA C:2012 Guidelines for the Use of the C Language in Critical Systems**
Regulatory Body: MISRA Consortium (misra.org.uk)
Revision/Date: 2012 (Amendment 3 - 2023)
Link: https://www.misra.org.uk/Activities/MISRA-C

(x) **CERT C Secure Coding Standard**
Regulatory Body: SEI / CERT (sei.cmu.edu)
Revision/Date: Continuously updated
Link: `https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard`

(xi) **POSIX 1003.1 - Portable Operating System Interface (RT Extensions)**
Regulatory Body: IEEE (ieee.org)
Revision/Date: 2017
Link: `https://pubs.opengroup.org/onlinepubs/9699919799/`

(xii) **RTCA DO-178C - Software Considerations in Airborne Systems and Equipment Certification**
Regulatory Body: RTCA, Inc. (rtca.org)
Revision/Date: 2011
Link: `https://www.rtca.org/content/rtca-do-178c`

(xiii) **ARINC 653 - Avionics Application Software Standard Interface**
Regulatory Body: ARINC / SAE International (sae.org)
Revision/Date: Part 1: Supplement 4, 2015
Link: `https://standards.sae.org/arinc_653/`

(xiv) **IPC-2221 - Generic Standard on Printed Board Design**
Regulatory Body: IPC International (ipc.org)
Revision/Date: Rev B, September 2012
Link: `https://www.ipc.org/TOC/IPC-2221B.pdf`

3 STAKEHOLDER REQUIREMENTS

3.1 Physical

3.1.1 The device must fit inside of a typical RC airplane.

3.1.2 The flight controller must be battery powered.

3.1.3 The device must have ports commonly used for connecting parts used in flight.

3.2 Behaviorial

3.2.1 The flight controller must keep stable drone flight during operation.

3.2.2 The flight controller must handle autonomous flight as well as teleoperated flight.

3.2.3 Regardless of control method, the flight controller must avoid the plane losing stability or crashing.

3.2.4 The flight controller must be able to be configured with the parameters of a given plane.

3.3 Functional

3.3.1 The flight controller must provide proper inputs to the control surfaces of the drone.

3.3.2 The flight controller must accept a programmed mission plan for autonomous flight.

3.3.3 The flight controller must read from on board sensors in order to provide stable controls to the plane.

3.3.4 The flight controller must allow for additional sensors to be connected.

3.4 Environmental

3.4.1 The flight controller must be able to operate during potentially rough flight of a small airplane.

3.4.2 The flight controller must work the same even when the tempurature or air pressure changes.

3.4.3 The flight controller must be able to function regardless of vibrations from the plane.

3.5 Ethical

3.5.1 The flight controller must have safety checks to ensure sufficient power at least 5 minutes of flight.

3.5.2 In the event of a failure the flight controller must attempt to land safely instead of simply crashing.

3.5.3 The system must report and log failures of flight components.

4 ENGINEERING REQUIREMENTS

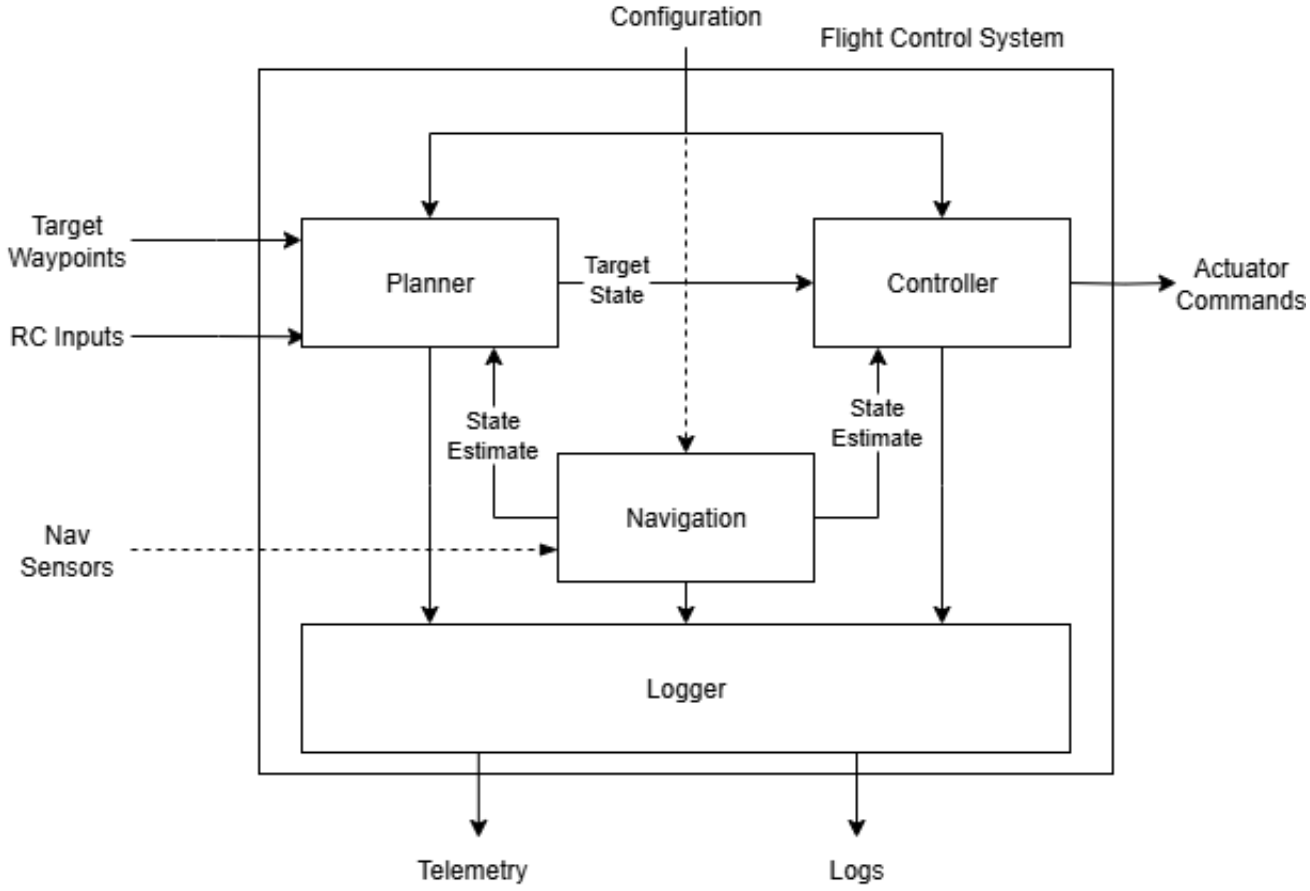

Figure 1. System functional diagram

4.1 System Requirements

The System provides the hardware and software foundation for all onboard modules. It includes processing, hardware capabilities, interfaces, hardware abstraction, and power management. The System is comprised of five main parts: Processing, Built in HW Capabilities, HW Interfaces, HAL, and Power Management.

4.1.1 Interfaces

4.1.1.1 Inputs

4.1.1.1.1 All submodules require configuration parameters that dictate their operation. These parameters will all be contained in a YAML file required for operation start. These configuration parameters will be different dependant on the individual submodule, and that submodule will define the contents of its own configuration. In software, the

parameters will be passed as a predefined object, referred to as a struct, though there is no requirement to utilize the C programming language and it's associated structs.

4.1.1.1.2 Waypoints are defined as a time index series of LLA points. Waypoints do not require an orientation or velocity component to avoid overconstraining the Planner submodule.

4.1.1.1.3 RC inputs are defined as the literal signals received from the RC receiver. This includes the raw stick inputs, as well as any auxiliary channels. The layout of these channels will be defined in the configuration parameters for a specific transmitter.

4.1.1.1.4 External sensor data is defined as raw data received from any sensors not directly connected to the onboard PCB. This includes sensors connected via serial communication.

4.1.1.2 Outputs

4.1.1.2.1 Peripheral commands are defined as commands to control onboard motors and servos. These commands are generated by the Controller submodule.

4.1.1.3 Internal

4.1.1.3.1 UAV state estimates are defined as the best estimate of the UAV's current position, orientation, velocity, and angular velocity. This estimate is generated by the Navigation submodule.

4.1.1.3.2 Desired states are defined as a time index series of the positions, orientations, velocities, and angular velocities. These desired states are generated by the Planner submodule.

4.1.1.3.3 State mask is defined as a bitmask indicating which states are to be controlled by the Controller submodule, and is generated by the Planner submodule.

4.1.1.3.4 RC adjustments are defined as trims and scaling factors for each control input. Based on the transmitter mapping, the adjusments will be applied to the raw RC inputs and passed to the Logger for logging. These adjustments are generated by the Planner submodule.

4.1.1.3.5 Raw control outputs are defined as the initial values calculated by the Controller for percent thrust, aileron deflection, elevator deflection, and rudder deflection. These values aren't directly used, but are passed to the Logger for logging purposes.

4.1.1.3.6 Error reports are defined as any fault or error detected by the HAL or Planner, Navigation, or Controller submodules during operation. These reports are passed to the Logger for logging and potential transmission. Error reports shall include a timestamp, error code, and description of the error.

4.1.1.3.7 Onboard sensor data is defined as raw data received from any sensors directly connected to the onboard PCB.

4.1.2 Processing

The Processing part provides the computational resources and pipeline for all onboard software modules.

4.1.2.1 The FCM shall provide a central processor to execute software instructions.

4.1.2.2 The central processor of the FCM shall support a processing speed such that the hyperperiod of all of the tasks is 2.5 ms(400 Hz).

4.1.2.3 The central processor of the FCM shall employ a real-time operating system to handle task scheduling.

4.1.2.4 The central processor of the FCM shall support task isolation, such that a crash on one task does not cause failure of another task.

4.1.2.5 The central processor of the FCM shall monitor the software program state in order to catch illegal operations.

4.1.2.6 The central processor of the FCM shall provide a failsafe state to provide graceful failure upon catching such faults.

4.1.2.7 All code running on the FCM shall adhere to the MISRA C:2012 C coding standard and CERT C Secure Coding Standard or equivalent standards for other programming languages used. All code shall adhere to the POSIX 1003.1 standard, RTCA DO-178C, and ARINC 653 where applicable. Any deviations should be documented and justified in the Software Design Document.

4.1.3 Built in HW Capabilities

Built-in hardware capabilities include sensors, and onboard peripherals required for mission execution.

4.1.3.1 The FCM shall provide an on-board IMU.

4.1.3.2 The FCM shall provide an on-board RC receiver for teleoperation.

4.1.3.3 The FCM shall provide an on-board micro-SD card reader for data storage.

4.1.3.4 The FCM shall provide on-board power monitoring.

4.1.3.5 The FCM shall provide on-board power regulation.

4.1.4 HW Interfaces

Hardware interfaces provide connectivity between onboard modules and external devices.

4.1.4.1 The FCM shall expose pins for the following hardware interfaces:

- I^2C
- SPI
- UART
- USB A
- 2 conductor CAN bus
- Micro SD

4.1.4.2 The ports for each of the specified interfaces shall be soldered onto the FCM.

4.1.4.3 The ports for each of the specified interfaces shall support a locking mechanism to secure interface connections.

4.1.5 HAL

The Hardware Abstraction Layer (HAL) provides a uniform interface to hardware resources, abstracting device specifics from higher-level modules. All configuration of the system will be done through the HAL. All necessary configuration information will be included in the YAML file which will be read in at program initialization.

4.1.5.1 Requirements

4.1.5.1.1 The HAL shall provide a scope and deliverables document detailing the capabilities and resources offered.

4.1.5.1.2 The HAL scope and deliverables document shall include support to at least the following functionalities:

- GPIO pin control

- I2C/SPI bus management
- UART serial communication
- USB 2.0 serial communication
- CAN communication
- Micro SD card read and write support
- ADC support
- Timer control
- Interrupt control
- Generic device interfacing (Requires associated driver)

4.1.5.1.3 The HAL shall provide an ICD outlining the APIs utilized for all functionality described in the scope and deliverables document.

4.1.5.1.4 The HAL ICD shall describe the ownership of all memory for all functions receiving or returning memory pointers.

4.1.5.1.5 The HAL shall provide common status codes which are returned from all HAL functions to avoid silent failures.

4.1.5.1.6 The HAL shall provide a method of gracefully handling faults and reporting those faults.

4.1.5.1.7 The HAL shall adhere to all coding standards described in §4.1.2.7

4.1.5.1.8 The HAL shall use no more than 25% of the total CPU time allocated to the onboard software system. This will be verified through profiling tools as such as `perf` or `htop`.

4.1.6 Power Management

Power Management ensures efficient and reliable power distribution to all system components.

4.1.6.1 The FCM shall accept power via an external source at a voltage of $5V \pm 0.25V$.

4.1.6.2 The FCM shall provide a method to protect hardware in case of current higher than 5 amps.

4.1.6.3 The FCM shall provide protection in the case of initial voltage lower than 4.75 such that the device will not initialize.

4.1.6.4 The FCM shall monitor the power source voltage throughout operation.

4.1.6.5 The FCM shall signal to the planner in the case of a critically low voltage ($4.25V$). This way the planner can attempt to ground the UAV before catastrophic crash.

4.1.7 Sub-Parts

The System is comprised of four functional sub-parts:

4.1.7.1 Planner

4.1.7.2 Controller

4.1.7.3 Navigation

4.1.7.4 Logger

4.2 Planner

The purpose of the Planner is to generate a series of realizable states for the UAV. The planner allows for both autonomous and pilot-directed operation. Inputs from both autonomous and pilot-directed modes are evaluated and potentially adjusted to fit safety and feasibility constraints. The inputs are evaluated based on the current UAV state and the current tolerances from configuration parameters. The Planner is comprised of three submodules: Waypoint Planner, RC Mixer, and State Select. Each has aids in the overall functionality of the Planner.

4.2.1 Interfaces

4.2.1.1 Inputs

4.2.1.1.1 The planner requires the mission parameters and UAS tolerances when generating waypoints. These tolerances shall include maximum and minimum airspeeds, maximum g-load, maximum climb and descent rates, maximum altitude, maximum bank angle, and minimum turn radius. Additionally, the planner's configuration parameters shall include an operating mode to determine whether RC inputs or generated waypoints are used to generate the desired states.

4.2.1.1.2 Waypoints as defined in 4.1.1.1.2.

4.2.1.1.3 RC inputs as defined in 4.1.1.1.3.

4.2.1.1.4 UAV state estimates as defined in 4.1.1.3.1.

4.2.1.2 Outputs

4.2.1.2.1 Desired States as defined in 4.1.1.3.2.

4.2.1.2.2 State mask as defined in 4.1.1.3.3.

4.2.1.2.3 Logging data as defined in 4.1.1.3.6.

4.2.1.3 Internal

4.2.1.3.1 Planned waypoint states. These inputs are generated by the Waypoint Planner submodule and consumed by the State Select submodule where they'll be multiplexed based on operating mode.

4.2.2 General Requirements

4.2.2.1 The Planner shall accept the inputs defined in §4.2.1.1 and produce a time-ordered sequence of desired states while consuming no more than 20% of the total CPU time allocated to the onboard software system.

4.2.2.2 For all output states:

$$
\begin{aligned}
V_{\text{min}} \leq v \leq V_{\text{max}}, \text{ (airspeed)} \\
|\phi| \leq \phi_{\text{max}}, \text{ (bank angle)} \\
|\dot{h}| \leq \dot{h}_{\text{max}}, \text{ (vertical speed)} \\
\kappa \leq 1/R_{\text{min}}, \text{ (turn rate)} \\
n \leq n_{\text{max}}, \text{ (load factor)}
\end{aligned}
$$

Where V_{min}, V_{max}, ϕ_{max}, $\dot{h}_{\text{max}}$, R_{min}, and n_{max} are the UAV tolerances input to the Planner defined in §4.2.1.1.1.

4.2.2.3 The Planner shall validate input presence, ranges, units, and timestamps. Invalid inputs shall result in rejection of the current input and preservation of last valid output for ≤ 1 cycle.

4.2.2.4 The Planner shall log all input and output data during operation. Additionally, all detected errors will be logged with reason and timestamp.

4.2.2.5 The Planner shall adhere to all coding standards defined in §4.1.2.7.

4.2.3 Waypoint Planner

The Waypoint Planner submodule is responsible for generating feasible waypoint trajectories during autonomous flight. It consumes high level waypoints, and creates a series of states that are determined to be safe and feasible based on the current UAV state and configuration parameters.

4.2.3.1 The Waypoint Planner shall include each commanded waypoint as a goal in the output sequence when inclusion does not violate any constraint in §4.2.2.2, otherwise, it shall mark the waypoint as unreachable and produce the state that best approximates the commanded waypoint while still satisfying all constraints.

4.2.3.2 The Waypoint Planner shall use no more than 10% of the total CPU time allocated to the onboard software system.

4.2.4 RC Mixer

The RC Mixer submodule is responsible for generating desired states based on pilot RC inputs. These inputs will be adjusted to the level defined in the configuration parameters to aid in preserving the safety of the UAV. The adjusted inputs will then be converted to desired states for the UAV to achieve.

4.2.4.1 The RC Mixer shall always produce a valid output regardless of input validity.

4.2.4.2 The RC Mixer shall converge to zero climb rate and zero turn rate in the absence of valid RC inputs within 2 seconds.

4.2.4.3 The RC Mixer shall log all RC inputs and generated outputs with timestamps, any errors detected, and any adjustments made to the inputs to preserve safety.

4.2.4.4 The RC Mixer shall use no more than 7.5% of the total CPU time allocated to the onboard software system.

4.2.5 State Select

The State Select submodule is responsible for multiplexing between the Waypoint Planner and RC Mixer outputs based on the current operating mode. Additionally, the State Select submodule is responsible for determining the state mask passed to the Controller based on the current operating mode.

4.2.5.1 The State Select shall always produce a valid output regardless of input validity.

4.2.5.2 The State Select shall use no more than 2.5% of the total CPU time allocated to the onboard software system.

4.3 Controller

The Controller computes actuator setpoints from planned trajectories and state estimates. It implements required control algorithms and scheduling, ensuring safe and efficient operation of the UAV actuators. The Controller is comprised of four submodules: Preprocessor, PVA Controller, ATL Controller, and Control Distributor. Each submodule contributes to translating planned trajectories into actuator commands.

4.3.1 Interfaces

4.3.1.1 Inputs

4.3.1.1.1 Planned setpoints defined in 4.1.1.3.2.

4.3.1.1.2 State mask as defined in 4.1.1.3.3.

4.3.1.1.3 State estimates defined in 4.4.1.2.1.

4.3.1.1.4 Controller configuration parameters.

4.3.1.2 Outputs

4.3.1.2.1 Actuator setpoints for ailerons, elevator, rudder, and throttle(s).

4.3.1.2.2 Logging data for control events and diagnostics.

4.3.1.3 Internal

4.3.1.3.1 Position, velocity, and acceleration state and setpoints.

4.3.1.3.2 Attitude, angular rate, and throttle state and setpoints.

4.3.1.3.3 Attitude, angular rate, and throttle targets.

4.3.1.3.4 Intermediary setpoints for ailerons, elevator, rudder, and throttle(s).

4.3.2 Preprocessor

The Preprocessor submodule is responsible for setting up inputs and setpoints for the controller chain given the desired operating mode.

4.3.2.1 The preprocessor shall consume planned setpoints as defined in 4.3.1.1.1.

4.3.2.2 The preprocessor shall consume state estimates as defined in 4.3.1.1.3.

4.3.2.3 The preprocessor shall output processed state and setpoints for the PVA controller as defined in 4.3.1.3.1.

4.3.2.4 The preprocessor shall output processed attitude and throttle state and setpoints for the ATL controller as defined in 4.3.1.3.2.

4.3.2.5 The preprocessor shall consume controller configuration parameters as defined in 4.3.1.1.4.

4.3.2.6 The preprocessor shall not consume more than 5% of the selected processing unit's bandwidth.

4.3.2.7 The preprocessor shall be functional agnostic to the UAV environment.

4.3.2.8 The preprocessor shall apply saturation and rate limits to setpoints determined by the controller configuration and UAV state.

4.3.2.9 The preprocessor shall log invalid or out-of-bounds inputs for diagnostics.

4.3.2.10 The preprocessor shall ensure safe defaults for missing or invalid inputs.

4.3.2.11 The preprocessor shall ensure safe operation under non-flight conditions.

4.3.2.12 The preprocessor shall enable and disable states and setpoints based on the current operating mode.

4.3.2.13 The preprocessor shall output the exact state it receives unless zeroing for disabled setpoints.

4.3.2.14 The preprocessor shall output the exact setpoints it receives modified only by slewing unless zeroing for disabled setpoints.

4.3.3 PVA Controller

The PVA (Position/Velocity/Acceleration) Controller submodule computes control setpoints to track planned trajectories and maintain the UAV along the desired path.

4.3.3.1 The PVA controller shall consume setpoints and state estimates as defined in 4.3.1.3.1.

4.3.3.2 The PVA controller shall output desired attitude and torque targets as defined in 4.3.1.3.3.

4.3.3.3 The PVA controller shall consume controller configuration parameters as defined in 4.3.1.1.4.

4.3.3.4 The PVA controller shall not consume more than 10% of the selected processing unit's bandwidth.

4.3.3.5 The PVA controller shall maintain stability with up to 20% sensor noise.

4.3.3.6 The PVA controller shall expose gains and tuning parameters via configuration.

4.3.3.7 The PVA controller shall reject external disturbances, such that tuning the gains allows for tracking a parobilic position input with maximum steady state error 2x the platform size.

4.3.3.8 The PVA controller shall ensure bounded errors and proper saturation handling.

4.3.3.9 The PVA controller shall log control errors and saturation events for diagnostics.

4.3.3.10 The PVA controller shall be stall aware and prevent commands that would lead to stalling.

4.3.3.11 The PVA controller shall expose setpoints for position, velocity, and acceleration in three dimensions.

4.3.4 ATL Controller

The ATL (Attitude/throttle-Level) Controller submodule translates higher-level commands into low-level actuator commands, respecting actuator limits and constraints.

4.3.4.1 The ATL controller shall consume setpoints and state estimates as defined in 4.3.1.3.2.

4.3.4.2 The ATL controller shall consume setpoints as defined in 4.3.1.3.3.

4.3.4.3 The ATL controller shall output low-level intermediary actuator commands as defined in 4.3.1.3.4.

4.3.4.4 The PVA controller shall consume controller configuration parameters as defined in 4.3.1.1.4.

4.3.4.5 The ATL controller shall not consume more than 10% of the selected processing unit's bandwidth.

4.3.4.6 The ATL controller shall maintain stability with up to 20% sensor noise.

4.3.4.7 The ATL controller shall expose gains and tuning parameters via configuration.

4.3.4.8 The ATL controller shall reject external disturbances, such that tuning the gains allows for tracking a ramp attitude input with maximum steady state error of 1 degree.

4.3.4.9 The ATL controller shall ensure bounded errors and proper saturation handling.

4.3.4.10 The ATL controller shall log control errors and saturation events for diagnostics.

4.3.4.11 The ATL controller shall be stall aware and prevent commands that would lead to stalling.

4.3.4.12 The ATL controller shall expose setpoints for attitude and angular velocity in three dimensions.

4.3.4.13 The ATL controller shall expose throttle setpoints independent of attitude and angular velocity.

4.3.5 Control Distributor

The Control Distributor submodule routes control outputs to the correct actuator channels, mapping and distributing logical control channels to physical hardware outputs.

4.3.5.1 The control distributor shall consume low-level intermediary actuator commands as defined in 4.3.1.3.4.

4.3.5.2 The control distributor shall output hardware actuator signals as defined in 4.3.1.2.1.

4.3.5.3 The control distributor shall consume controller configuration parameters as defined in 4.3.1.1.4.

4.3.5.4 The control distributor shall not consume more than 5% of the selected processing unit's bandwidth.

4.3.5.5 The control distributor shall be functional agnostic to the UAV environment.

4.3.5.6 The control distributor shall detect and handle hardware failures gracefully.

4.3.5.7 The control distributor shall support testing the configured output map while not in flight.

4.3.5.8 The control distributor shall be robust to configuration errors.

4.3.5.9 The control distributor shall log routing errors and hardware failures for diagnostics.

4.3.5.10 The control distributor shall attempt to continue operation in the presence of hardware output failures.

4.3.5.11 The control distributor shall disable outputs to all throttle channels when not in an armed flight mode.

4.3.5.12 The control distributor shall map logical control channels to physical hardware outputs based on configuration using the HAL defined in 4.1.5.

4.4 Navigation

The Navigation module provides accurate state estimation and sensor selection to support planning and control by running estimators, selecting appropriate sensors, and publishing state estimates. The Navigation module is comprised of two submodules: Sensor Selector and Estimator. Each submodule contributes to robust and accurate state estimation from available sensor inputs.

4.4.1 Interfaces

4.4.1.1 Inputs

4.4.1.1.1 IMU, GPS, barometer, and magnetometer data from all available units

4.4.1.1.2 Navigation configuration parameters

4.4.1.2 Outputs

4.4.1.2.1 Global position, velocity, attitude, and angular velocity state estimate and uncertainty with timestamp

4.4.1.2.2 Logging data for sensor selection and estimation events.

4.4.1.3 Internal

4.4.1.3.1 Selected IMU, GPS, barometer, and magnetometer sensor feeds

4.4.2 Sensor Selector

The Sensor Selector submodule evaluates available sensor inputs and using configuration selects the best sensors for the current conditions, ensuring robust operation and allowing user input.

4.4.2.1 The sensor selector unit shall consume all available sensor feeds defined in 4.4.1.1.1

4.4.2.2 The sensor selector unit shall consume sensor selection configuration as defined in 4.4.1.1.2

4.4.2.3 The sensor selector unit shall output selected sensor feeds for the estimator as defined in 4.4.1.3.1

4.4.2.4 The sensor selector shall not consume more than 5% of the selected processing unit's bandwidth.

4.4.2.5 The sensor selector shall be functional agnostic to the UAV environment.

4.4.2.6 The sensor selector shall expose safe fallbacks sensors if available when preferred sensors are unavailable.

4.4.2.7 The sensor selector shall use health indicators to avoid selecting faulty sensors.

4.4.2.8 The sensor selector shall log sensor selection decisions for diagnostics.

4.4.2.9 The sensor selector shall expose priority and fallback choices for sensors based on configuration.

4.4.2.10 The sensor selector shall only execute upon reception of new sensor data.

4.4.3 Estimator

The Estimator submodule fuses selected sensor inputs using sensor fusion algorithms to produce the best estimate of the system state.

4.4.3.1 The estimator shall provide state estimate outputs and uncertainties as defined in 4.4.1.2.1.

4.4.3.2 The estimator shall consume selected sensor streams defined in 4.4.1.3.1.

4.4.3.3 The estimator shall comsume navigation configuration parameters as defined in 4.4.1.1.2.

4.4.3.4 The estimator shall log raw sensor inputs and state estimates for diagnostics.

4.4.3.5 The estimator shall not consume more than 10% of the selected processing unit's bandwidth.

4.4.3.6 The estimator shall estimate and compensate for signals with up to 20% noise.

4.4.3.7 The estimator shall estimate and correct biases in sensor measurements within 20%.

4.4.3.8 The estimator shall expose uncertainties that are accurate to 3 standad deviations.

4.4.3.9 The estimator shall use the configuration to be congnisant to sensor mounting orientations.

4.4.3.10 The estimator shall log inconsistant inputs and anomalies.

4.4.3.11 The estimator shall use timestamps to identify and drop stale data.

4.4.3.12 The estimator shall ensure proper timestamps accompany each output estimate.

4.4.3.13 The estimator shall provide a common interface for inputs of various measurement types.

4.4.3.14 The estimator shall expose interfaces with data that adheres to at most one body frame coordinate system and one inertial frame coordinate system.

4.5 Logger

The Logger records mission data and provides mechanisms for transmission or storage by collecting telemetry, writing to storage, and/or sending data via communications links. The Logger is comprised of two submodules: Storage and Transmitt. Each submodule handles data persistence and transmission for mission telemetry and diagnostic information.

4.5.1 Interfaces

4.5.1.1 Inputs

4.5.1.1.1 The Logger requires configuration parameters to determine where data is logged and sent, transmission rates, and data routing policies. These parameters will inclue file storage locations and transmission rates and protocols.

4.5.1.1.2 The Logger receives input waypoints from the Companion Computer as defined in 4.1.1.1.2.

4.5.1.1.3 The Logger receives desired states from the Planner as defined in 4.1.1.3.2.

4.5.1.1.4 The Logger receives the state mask from the Planner as defined in 4.1.1.3.3.

4.5.1.1.5 The Logger receives RC inputs from the Pilot as defined in 4.1.1.1.3.

4.5.1.1.6 The Logger receives adjustments to RC inputs from the Planner. These adjustments include trims and scaling factors of each control input.

4.5.1.1.7 The Logger receives raw sensor data from any onboard sensors as defined in 4.1.1.3.7 or external sensors as defined in 4.1.1.1.4.

4.5.1.1.8 The Logger receives state estimates from the Navigation as defined in 4.1.1.3.1.

4.5.1.1.9 The Logger receives raw control outputs from the Controller as defined in 4.1.1.3.5.

4.5.1.1.10 The Logger receives peripheral commands from the Controller as defined in 4.1.1.2.1.

4.5.1.1.11 The Logger receives any error reports from the Planner, Navigation, and Controller as defined in 4.1.1.3.6.

4.5.1.2 Outputs

4.5.1.2.1 The Logger outputs stored log files to the location defined in the configuration parameters.

4.5.1.2.2 The Logger outputs telemetry packets transmitted via communications link as defined in the configuration parameters.

4.5.2 General Requirements

4.5.2.1 The Logger shall support logging of all generated data from onboard modules.

4.5.2.2 The Logger shall support rate limiting for all logged data.

4.5.2.3 The Logger shall adhere to all coding standards defined in §4.1.2.7.

4.5.2.4 The Logger shall use no more than 10% of the total CPU time allocated to the onboard software system.

4.5.3 Storage

The Storage submodule is responsible for persisting telemetry and logs to onboard non-volatile memory, ensuring data is retained for later retrieval and analysis. The location and format of stored data can be defined via configuration parameters. However, if no configuration is provided, the Storage submodule will default to storing data in a predefined location in a standard format as defined in the proceeding requirements.

4.5.3.1 The Storage submodule shall have default storage location of `/logs/` on onboard non-volatile memory.

4.5.3.2 The Storage submodule shall default to storing data from each data stream in a `.log` file with timestamps for each entry.

4.5.3.3 The Storage submodule shall support configuration parameters to define custom storage locations and file formats for all logged data.

4.5.3.4 The Storage submodule shall support a sliding window of log files, based on either time duration or storage size limits defined in configuration parameters. The default shall be 4 hours and 10MB respectively.

4.5.3.5 The Storage submodule shall implement a verification mechanism such as checksums or hashes for all stored data.

4.5.3.6 The Storage submodule shall use no more than 5% of the total CPU time allocated to the onboard software system.

4.5.4 Transmitt

The Transmitt submodule handles transmission of selected telemetry data to ground stations or other external systems via available communications links.

4.5.4.1 The Transmitt submodule shall support transmission to up to three external systems simultaneously.

4.5.4.2 The Transmitt submodule shall be support both UDP and TCP transmission protocols.

4.5.4.3 The Transmitt submodule shall use no more than 5% of the total CPU time allocated to the onboard software system.

5 VERIFICATION OF REQUIREMENTS

5.1 System Verification

5.1.1 Processing

5.1.1.1 Requirement shall be verified by inspecting the FCM hardware and its corresponding datasheet to confirm the presence of a central processor capable of executing software instructions.

5.1.1.2 Requirement shall be verified by executing a representative software load on the processor. Profiling tools and an oscilloscope will be used to measure task execution times and ensure all deadlines within the hyperperiod are met.

5.1.1.3 Requirement shall be verified by inspecting the software design documents and source code to confirm that a Real-Time Operating System (RTOS) is implemented for task scheduling.

5.1.1.4 Requirement shall be verified by intentionally injecting faults (e.g., division by zero, null pointer access) into a non-critical task and observing that other tasks continue to operate as expected.

5.1.1.5 Requirement shall be verified by executing a test suite that includes illegal operations (e.g., memory access violations) and confirming that the system's monitoring mechanisms detect and log these faults.

5.1.1.6 Requirement shall be verified by triggering a fault and demonstrating that the system enters a predefined failsafe state (e.g., motor shutdown, error signal) as specified in the system design.

5.1.2 Built in HW Capabilities

5.1.2.1 Requirement shall be verified by visual inspection of the FCM to confirm the presence of an on-board Inertial Measurement Unit (IMU).

5.1.2.2 Requirement shall be verified by visual inspection of the FCM to confirm the presence of an on-board RC receiver.

5.1.2.3 Requirement shall be verified by visual inspection of the FCM to confirm the presence of an on-board micro-SD card reader.

5.1.2.4 Requirement shall be verified by using a multimeter to measure the output of the power monitoring circuit while the input voltage is varied and comparing the measurements to the values reported by the software.

5.1.2.5 Requirement shall be verified by applying a range of input voltages and measuring the regulated output voltage to ensure it remains within its specified tolerance.

5.1.3 HW Interfaces

5.1.3.1 Requirement shall be verified by inspecting the FCM datasheet and the physical PCB to confirm that pins for all specified hardware interfaces are exposed.

5.1.3.2 Requirement shall be verified by visual inspection of the FCM to confirm that physical ports are soldered onto the board for each interface.

5.1.3.3 Requirement shall be verified by visual inspection and physical manipulation of the connectors to confirm they feature a locking mechanism.

5.1.4 HAL

5.1.4.1 Requirement 4.1.4.1 shall be verified by reviewing the scope and deliverables document and ensuring it details all required capabilities and resources offered by the HAL. The review will confirm the following:

- Document completeness and clarity.
- Coverage of all hardware abstraction functionalities.
- Consistency with ICD and implementation.

5.1.4.2 Requirement 4.1.4.2 shall be verified by reviewing the scope and deliverables document and confirming that it includes support for all specified functionalities. The review will include verification of the following capabilities:

- GPIO pin control.
- I2C/SPI bus management.
- UART serial communication.
- USB 2.0 serial communication.
- CAN communication.
- Micro SD card read and write support.
- ADC support.

- Timer control.
- Interrupt control.
- Generic device interfacing (with associated driver).

5.1.4.3 Requirement 4.1.4.3 shall be verified by reviewing the ICD and ensuring it outlines all APIs utilized for functionality described in the scope and deliverables document. The review will confirm:

- Completeness of API documentation for all functionalities.
- Clear function signatures and parameter descriptions.
- Consistency between ICD and implementation.

5.1.4.4 Requirement 4.1.4.4 shall be verified by reviewing the ICD and confirming that memory ownership is clearly described for all functions receiving or returning memory pointers. The review will validate:

- Explicit ownership documentation for each pointer parameter.
- Caller vs. callee responsibility specifications.
- Deallocation and lifecycle management guidelines.

5.1.4.5 Requirement 4.1.4.5 shall be verified using unit tests that call HAL functions and confirm that common status codes are returned to avoid silent failures. The tests will include the following cases:

- Valid operations returning success status codes.
- Invalid parameters returning appropriate error codes.
- Resource unavailability returning specific error codes.
- Verification that no function exits silently without returning a status code.

5.1.4.6 Requirement 4.1.4.6 shall be verified using unit tests that simulate fault conditions and verify that the HAL gracefully handles and reports faults. The tests will include the following cases:

- Hardware communication failures with proper error reporting.
- Invalid hardware states with graceful degradation.
- Timeout conditions with appropriate fault handling.
- Recovery mechanisms after transient faults.

5.1.4.7 Requirement 4.1.4.7 shall be verified using static analysis tools such as linters and formatters to ensure compliance with the coding standards outlined in section §4.1.2.7.

5.1.4.8 Requirement 4.1.4.8 shall be verified using a unit test that exercises HAL functions under typical load and confirms CPU usage does not exceed 25% of the total CPU time allocated to the onboard software system. This will be verified through profiling tools such as `perf` or `htop`. The test will include the following scenarios:

- Baseline HAL operations with minimal concurrent activity.
- HAL operations during peak module execution.
- Sustained HAL usage over extended operation periods.
- Profiling of individual HAL function CPU overhead.

5.1.4.9 The accuracy of the HAL will be verified by unit tests that measure the setup, and response of all associated pins and interfaces. The HAL shall have at least 90% code coverage as measured by a code coverage tool such as `gcov` or `lcov`.

5.1.5 Power Management

5.1.5.1 Requirement shall be verified by connecting a variable DC power supply to the FCM's power input. The voltage shall be varied from 4.75V to 5.25V, and the system's operational status will be monitored to confirm it functions correctly throughout this range.

5.1.5.2 Requirement shall be verified by connecting an adjustable electronic load to the FCM. The current draw shall be slowly increased above 5 amps to confirm that the onboard protection mechanism (e.g., fuse or circuit breaker) activates and interrupts power to protect the hardware.

5.1.5.3 Requirement shall be verified by setting a power supply to a voltage below 4.75V and attempting to power on the device. It will be confirmed that the FCM does not initialize. The voltage will then be raised above 4.75V to confirm the device initializes correctly.

5.1.5.4 Requirement shall be verified by connecting a variable power supply and a multimeter to the FCM's power input. The input voltage will be varied, and the voltage reported by the FCM's software will be compared against the multimeter reading to ensure accuracy.

5.1.5.5 Requirement shall be verified by lowering the input voltage to 4.25V. It will be demonstrated through software logs or a debug interface that a signal is sent to the planner, and the planner initiates its low-voltage response protocol.

5.2 Planner Verification

5.2.1 Requirement 4.2.2.1 shall be verified using the integration test outlined in §5.2.1.1 verifying that the CPU load does not exceed to allocated 20% budget via a profiling tool like `perf` or `htop`.

5.2.2 Requirement 4.2.2.2 shall be verified using the integration tests outlined in §5.2.1.1 and §5.2.2.1 and verifying that the output desired states do not exceed the tolerances as described in §4.2.2.2.

5.2.3 Requirement 4.2.2.3 shall be verified by using an integration test that inputs a set of waypoints, and ensures that valid inputs are accurately read, and invalid inputs are rejected and consistent outputs are maintained. The tests will include the following cases:

- Valid waypoints and RC commands.
- Malformed waypoints, but valid RC commands.
- Valid waypoints, but malformed RC commands.
- Malformed waypoints and malformed RC commands.

5.2.4 Requirement 4.2.2.4 shall be verified using an integration test that inputs a series of waypoints and RC commands and verifies all faults or adjustments are logged correctly. The test will include the following cases:

- Valid, achievable waypoints in autonomous mode.
- Valid, unachievable waypoints in autonomous mode.
- Malformed waypoints in autonomous mode.
- Valid, achievable RC commands in manual mode.
- Valid, unachievable RC commands in manual mode.

5.2.5 Requirement 4.2.2.5 shall be verified using static analysis tools such as linters and formatters to ensure compliance with the coding standards outlined in section §4.1.2.7.

5.2.1 Waypoint Planner

5.2.1.1 Requirement 4.2.3.1 shall be verified using an integration test that inputs a series of waypoints, and verifies the existance of all achievable goals in the output desired states. The test will include the following cases:

- Straight line waypoints achievable within all tolerances.

- Waypoints requiring the maximum of each tolerance to achieve.
- Waypoints that cannot be achieved due to airspeed tolerance limits.
- Waypoints that cannot be achieved due to climb rate tolerance limits.
- Waypoints that cannot be achieved due to turn rate tolerance limits.
- Waypoints that cannot be achieved due to acceleration tolerance limits.
- Waypoints violating climb and descent rate tolerances, but at least 60% of waypoints are achievable by adjusting airspeed and turn rate within tolerances.
- Waypoints violating turn rate tolerances, but at least 60% of waypoints are achievable by adjusting airspeed and climb rate within tolerances.
- Waypoints violating enough tolerances such that no waypoints are achievable.

5.2.1.2 Requirement 4.2.3.2 shall be verified using the integration test from §5.2.1.1 and monitoring the CPU load to ensure it does not exceed the allocated 10% budget using profiling tools outlined in §5.2.1.

5.2.2 RC Mixer

5.2.2.1 Requirement 4.2.4.1 shall be verified using an integration test that inputs a series of RC commands, and verifies a valid st of desird output states regardeless of input. The test will include the following cases:

- RC commands that require no adjustments.
- RC commands that require maximum adjustments.
- RC commands that request more airspeed than the tolerance allows.
- RC commands that request more climb rate than the tolerance allows.
- RC commands that request more turn rate than the tolerance allows.
- RC commands that request more acceleration than the tolerance allows.
- RC commands that exceed all tolerances to verify graceful degradation.
- Malformed RC commands to verify graceful degradation.
- No RC commands to verify graceful degradation.

5.2.2.2 Requirement 4.2.4.2 shall be verifed using the integration test from §5.2.2.1 and verifying that on the final two cases with either malformed or no RC commands, that the output desired states remain valid, and converge to horizontal flight after 2 seconds.

5.2.2.3 Requirement 4.2.4.3 shall be verified using the integration test from §5.2.2.1 and ensuring that all inputs, outputs, adjustments, and faults are logged correctly.

5.2.2.4 Requirement 4.2.4.4 shall be verified using the integration test from §5.2.2.1 and monitoring the CPU load to ensure it does not exceed the allocated 7.5% budget using profiling tools outlined in §5.2.1.

5.2.3 State Select

5.2.3.1 Requirement 4.2.5.1 shall be verfied using an integration test that inputs a series of waypoints and RC commands, and verifies that there is at least some valid output desired states regardless of input. The test will include the following cases:

- Valid waypoints and RC commands, but UAV in autonomous mode.
- Valid waypoints and RC commands, but UAV in manual mode.
- Malformed waypoints, in autonomous mode.
- Malformed waypoints, in manual mode.

5.2.3.2 Requirement 4.2.5.2 shall be verified using the integration test from §5.2.3.1 and ensuring that CPU load does not exceed the allocated 2.5% budget using profiling tools outlined in §5.2.1.

5.3 Controller Verification

5.3.1 Preprocessor

5.3.1.1 The Preprocessor shall be verified by supplying planned setpoints and state estimates under all supported operating modes and confirming that processed outputs match expected values for each mode.

5.3.1.2 The Preprocessor shall be verified by introducing missing, invalid, or out-of-bounds inputs and confirming that safe defaults are substituted, last valid outputs are preserved for up to one cycle, and all such events are logged with reason and timestamp.

5.3.1.3 The Preprocessor shall be verified by changing controller configuration parameters and confirming that enabling/disabling of states and setpoints occurs as specified, and that zeroing or slewing is applied only to disabled setpoints.

5.3.1.4 The Preprocessor shall be verified by applying rapid mode transitions (i.e. idle to flight, flight to disarmed) and confirming that only expected states are zeroed or slewed, and that outputs remain stable.

5.3.1.5 The Preprocessor shall be verified by measuring execution time per control cycle on the target platform and confirming it remains below 5% of the total loop period.

5.3.1.6 The Preprocessor shall be verified by repeating functional tests under simulated environmental conditions (temperature, vibration, sensor delay) and confirming consistent results.

5.3.1.7 The Preprocessor shall be verified by injecting out-of-range setpoints and confirming that saturation and rate limiting are properly applied according to configuration and UAV state.

5.3.1.8 The Preprocessor shall be verified by simulating non-flight conditions and confirming that outputs are safe and no actuator commands are issued.

5.3.1.9 The Preprocessor shall be verified by reviewing diagnostic logs to confirm that all detected input faults and mode transitions are accurately recorded.

5.3.2 PVA Controller

5.3.2.1 The PVA Controller shall be verified by applying step, ramp, and parabolic position commands and confirming that output attitude and torque targets follow the desired trajectory with steady-state error less than twice the defined platform size.

5.3.2.2 The PVA Controller shall be verified by injecting known disturbances (i.e. wind bias, external acceleration) and verifying disturbance rejection through bounded tracking errors.

5.3.2.3 The PVA Controller shall be verified by sweeping gain parameters and confirming the effect on closed-loop stability and tracking performance.

5.3.2.4 The PVA Controller shall be verified by providing invalid or missing setpoints and confirming that bounded errors and safe fallback behavior are observed.

5.3.2.5 The PVA Controller shall be verified by measuring computation duration per cycle and confirming processing load remains within 10% of available control cycle time.

5.3.2.6 The PVA Controller shall be verified by introducing up to 20% sensor noise and confirming maintained stability and bounded output.

5.3.2.7 The PVA Controller shall be verified by running tests with different gain and tuning parameter configurations and confirming that changes are reflected in controller response.

5.3.2.8 The PVA Controller shall be verified by driving control inputs to cause near-saturation conditions and confirming proper limit enforcement and diagnostic logging.

5.3.2.9 The PVA Controller shall be verified by simulating stall-prone conditions and confirming that attitude commands are limited to avoid stall regions.

5.3.2.10 The PVA Controller shall be verified by reviewing logs to confirm that all control errors and saturation events are recorded with timestamps.

5.3.2.11 The PVA Controller shall be verified by confirming that position, velocity, and acceleration setpoints are generated in all three axes and correctly propagated to dependent modules.

5.3.3 ATL Controller

5.3.3.1 The ATL Controller shall be verified by applying known attitude and throttle setpoints and confirming that output actuator commands achieve the target within allowable error.

5.3.3.2 The ATL Controller shall be verified by performing ramp attitude input tests and verifying maximum steady-state tracking error does not exceed 1 degree.

5.3.3.3 The ATL Controller shall be verified by varying configuration parameters and confirming correct mapping of gain settings to response dynamics.

5.3.3.4 The ATL Controller shall be verified by providing invalid or missing setpoints and confirming that bounded errors and safe fallback behavior are observed.

5.3.3.5 The ATL Controller shall be verified by measuring computational latency per cycle and verifying utilization under 10% of total control loop time.

5.3.3.6 The ATL Controller shall be verified by applying synthetic sensor noise (up to 20%) and external disturbances to confirm maintained control stability.

5.3.3.7 The ATL Controller shall be verified by running tests with different gain and tuning parameter configurations and confirming that changes are reflected in controller response.

5.3.3.8 The ATL Controller shall be verified by forcing actuator saturation conditions and confirming bounded error behavior and saturation logging.

5.3.3.9 The ATL Controller shall be verified by simulating aerodynamic stall conditions and confirming throttle and attitude limits are enforced.

5.3.3.10 The ATL Controller shall be verified by reviewing logs to confirm that all control errors and saturation events are recorded with timestamps.

5.3.3.11 The ATL Controller shall be verified by confirming that attitude and angular velocity setpoints are generated in all three axes, and throttle setpoints are independent of attitude and angular velocity.

5.3.4 Control Distributor

5.3.4.1 The Control Distributor shall be verified by applying known intermediary actuator commands and verifying correct mapping to physical actuator outputs based on configuration.

5.3.4.2 The Control Distributor shall be verified by conducting signal continuity checks from logical control channels to each actuator interface pin.

5.3.4.3 The Control Distributor shall be verified by executing non-flight output mapping tests and confirming correct routing of signals to simulated actuator hardware.

5.3.4.4 The Control Distributor shall be verified by changing configuration parameters and confirming that logical-to-physical channel mapping updates as expected.

5.3.4.5 The Control Distributor shall be verified by measuring task timing and verifying utilization below 5% of total processing bandwidth.

5.3.4.6 The Control Distributor shall be verified by simulating actuator line failure or hardware disconnect and confirming that fallback or degraded mode operation occurs.

5.3.4.7 The Control Distributor shall be verified by running tests in various environmental conditions and confirming robust operation.

5.3.4.8 The Control Distributor shall be verified by confirming that throttle outputs are disabled when in disarmed mode.

5.3.4.9 The Control Distributor shall be verified by inducing configuration mismatches and verifying the system logs routing errors and maintains control on remaining channels.

5.3.4.10 The Control Distributor shall be verified by injecting single-output hardware faults and confirming operation continuity and logging of the event.

5.3.4.11 The Control Distributor shall be verified by cross-checking logical-to-physical channel mapping with configuration tables for correctness.

5.4 Navigation Verification

5.4.1 Sensor Selector

5.4.1.1 The Sensor Selector shall be verified by connecting multiple functioning IMUs, GPS units, barometers, and magnetometers to the system and observing that data from each is received and processed during operation.

5.4.1.2 The Sensor Selector shall be verified by loading different sensor configuration files and observing that the module updates its active sensor preferences and priorities according to each configuration.

5.4.1.3 The Sensor Selector shall be verified by running the system with several sensors of each type and confirming that the Estimator receives data only from the sensors selected by the Sensor Selector.

5.4.1.4 The Sensor Selector shall be verified by operating the system under representative load conditions and timing selection cycles to confirm that selection operations complete within 5% of total available processing time.

5.4.1.5 The Sensor Selector shall be verified by running flight-equivalent tests in varied conditions, such as static, moving, and magnetically disturbed environments, and confirming consistent sensor selection behavior and output availability.

5.4.1.6 The Sensor Selector shall be verified by disabling the highest-priority sensor during operation and confirming that the module automatically switches to a valid fallback sensor without interrupting output availability.

5.4.1.7 The Sensor Selector shall be verified by injecting simulated sensor faults (i.e. unrealistic values or loss of updates) and confirming that the module excludes those sensors from selection.

5.4.1.8 The Sensor Selector shall be verified by executing a sequence of sensor connection and disconnection events and confirming that each selection change and its rationale are recorded in the system log.

5.4.1.9 The Sensor Selector shall be verified by configuring sensors with distinct priority levels and observing that the selected sensors match the expected order of precedence and fallback defined in configuration.

5.4.1.10 The Sensor Selector shall be verified by providing new data at controlled intervals and confirming that selection processing only occurs immediately after new sensor data are received.

5.4.2 Estimator

5.4.2.1 The Estimator shall be verified by supplying known-position and -orientation sensor data and confirming that the estimated position, velocity, attitude, and angular rate outputs match ground truth within expected uncertainty bounds as defined in 4.4.1.2.1.

5.4.2.2 The Estimator shall be verified by providing the Estimator with only the selected sensor outputs from the Sensor Selector and confirming that no data from unselected sensors influence the results.

5.4.2.3 The Estimator shall be verified by loading different sets of navigation parameters and confirming that the estimator behavior and output statistics change according to those configurations.

5.4.2.4 The Estimator shall be verified by recording all sensor inputs and state outputs during a test run and confirming that both are available in the diagnostic log files for review.

5.4.2.5 The Estimator shall be verified by operating under representative mission conditions and measuring the total processing duration per update cycle to confirm that it remains within 10% of total available computational time.

5.4.2.6 The Estimator shall be verified by applying test sensor data containing up to 20% random noise and comparing estimated states against the known true trajectory to confirm that deviations remain within accuracy limits.

5.4.2.7 The Estimator shall be verified by adding a known constant bias to one or more sensor inputs and confirming that the estimator's bias correction reduces the resulting error to within 20% of the nominal value.

5.4.2.8 The Estimator shall be verified by running multiple trials with randomized noise and comparing the spread of estimation errors to the reported uncertainty to confirm correspondence within three standard deviations.

5.4.2.9 The Estimator shall be verified by mounting sensors in different known orientations and confirming that the estimated attitude and derived quantities correctly reflect the new orientations.

5.4.2.10 The Estimator shall be verified by providing inconsistent sensor data (i.e. conflicting altitude or attitude readings) and confirming that these events are detected and logged as anomalies.

5.4.2.11 The Estimator shall be verified by delaying sensor data packets to simulate stale information and confirming that outdated data are excluded from use in state estimation.

5.4.2.12 The Estimator shall be verified by examining output data during operation to confirm that every estimate includes a timestamp that matches the time of input data used.

5.4.2.13 The Estimator shall be verified by providing various combinations of sensor measurements (i.e. IMU only, IMU + GPS, IMU + barometer) and confirming that all are processed through a common data pathway to produce consistent state outputs.

5.4.2.14 The Estimator shall be verified by collecting state outputs and confirming through comparison against reference transformations that all data conform to a single body frame and a single inertial frame definition.

5.5 Logger Verification

5.5.1 General Requirements

5.5.1.1 Requirement 4.5.3.1 shall be verified using an integration test that configures the Logger to receive all data streams from the Planner, Controller, and Navigation modules, and verifies that all data is successfully logged to storage and/or transmitted. The test will include the following cases:

- Logging all data streams simultaneously with default configuration under nominal load.
- Logging selected data streams with custom configuration under nominal load.
- Logging all data streams under high CPU and memory load to verify no data loss or corruption.

5.5.1.2 Requirement 4.5.3.2 shall be verified using an integration test that configures different rate limits for various data streams and verifies that the actual logging rates do not exceed the configured limits. The test will include the following cases:

- Rate limiting high-frequency sensor data to 10 Hz.
- Rate limiting control outputs to 50 Hz.
- Rate limiting state estimates to 100 Hz.
- Verifying that data is properly decimated or dropped when rates exceed limits.

5.5.1.3 Requirement 4.5.3.3 shall be verified using static analysis tools such as linters and formatters to ensure compliance with the coding standards outlined in section §4.1.2.7.

5.5.1.4 Requirement 4.5.3.4 shall be verified using the integration test outlined in §5.5.1.1 and monitors CPU usage via profiling tools like `perf` or `htop` to ensure the Logger does not exceed 10% of total CPU time.

5.5.2 Storage

5.5.2.1 Requirement 4.5.4.1 shall be verified using the integration test outlined in §5.5.1.1 and verifies that log files are created in the `/logs/` directory on onboard non-volatile memory.

5.5.2.2 Requirement 4.5.4.2 shall be verified using the integration test outlined in §5.5.1.2 and verifies that each data stream creates a `.log` file with timestamped entries.

5.5.2.3 Requirement 4.5.4.3 shall be verified using an integration test that configures custom storage locations and file formats for all logged data and verifies that the Storage submodule correctly writes data to the specified locations in the specified formats. The test will include the following cases:

- Custom storage location under `/custom_logs/`.
- Custom binary file format instead of text.
- Custom JSON format for structured data.
- Verifying graceful fallback to default location if custom location is unavailable.

5.5.2.4 Requirement 4.5.4.4 shall be verified using an integration test that runs the Storage submodule with configured time duration and storage size limits and verifies that older log files are automatically deleted when limits are exceeded. The test will include the following cases:

- Time-based sliding window: logging data for a simulated 5 hours.
- Size-based sliding window: logging data until storage exceeds 15MB and verifying oldest files are deleted to stay under 10MB. Input data should be artificially inflated to avoid long test durations.
- Combined time and size limits: verifying the more restrictive limit is enforced.
- Verifying default limits (4 hours and 10MB) when no configuration is provided.

5.5.2.5 Requirement 4.5.4.5 shall be verified using the integration test outlined in §5.5.1.1 and then reads it back, computing based on the designed data verification method to verify data integrity.

5.5.2.6 Requirement 4.5.4.6 shall be verified using the integration test from §5.5.1.1 and monitoring CPU load to ensure the Storage submodule does not exceed the allocated 5% budget using profiling tools outlined in §5.5.1.

5.5.3 Transmitt

5.5.3.1 Requirement 4.5.5.1 shall be verified using an integration test that configures the Transmitt submodule to transmit telemetry to three separate external systems simultaneously and verifies that all systems receive the expected data. The test will include the following cases:

- Transmitting to three ground stations with different IP addresses.
- Verifying each system receives independent, configurable data streams.
- Verifying transmission continues to remaining systems if one system becomes unreachable.

5.5.3.2 Requirement 4.5.5.2 shall be verified using an integration test that configures the Transmitt submodule to use both UDP and TCP protocols and verifies correct transmission and reception of telemetry data. The test will include the following cases:

- UDP transmission to a ground station and verifying packet reception (allowing for some packet loss).
- TCP transmission to a ground station and verifying reliable, ordered packet reception.
- Verifying protocol-specific error handling (e.g., connection drops for TCP, packet loss for UDP).

5.5.3.3 Requirement 4.5.5.3 shall be verified using the integration test from §5.5.3.1 and monitoring CPU load to ensure the Transmitt submodule does not exceed the allocated 5% budget using profiling tools outlined in §5.5.1.

5.6 Verify Coverage of Stakeholder Requirements

Paragraph Number	Test Type	Tester's Name	Pass/Fail	Date