Flight Control System (FCS) Architecture & Design Report

Introduction

This report details the design and architecture of a highly deterministic, safety-critical Flight Control System (FCS) for a 15 kg fixed-wing UAV. The architecture was developed using Model-Based Design (MBD) principles in Simulink/Stateflow, strictly adhering to embedded Real-Time Operating Systems (RTOS) constraints. The design successfully integrates multi-rate sensor fusion, hierarchical mode management, and robust Fault Detection, Isolation, and Recovery (FDIR) mechanisms.

Hardware Specifications

Sensor Specifications

Sensor_Name	Output	Dimension	Rate
6-axis IMU	[ax,ay,az,p,q,r]	6x1	800Hz
3-axis Magnetometer	[mx,my,mz]	3x1	50Hz
Barometric Altitude	[pressure]	1x1	50Hz
GPS	[lat,lon,height,spd,heading]	3x1	10Hz
Pitot Tube	[pressure]	1x1	50Hz

ax,ay,az are linear acceleration rates in body axis

p,q,r are angular acceleration rates in body axis

mx,my,mz is orientation of earth's magnetic field with magnetometer

MCU Specifications

MCU model: STM32F765
Max Operating Frequency: 216MHz
RAM Size: 512 Kbytes
ISR: > 100

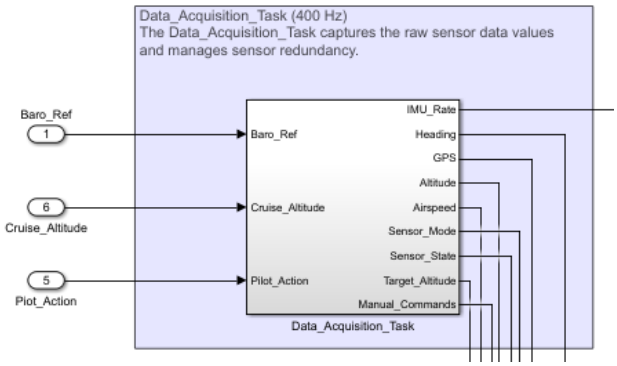
Data Acquisition

The hardware ports using ISRs to write into the MCU memory. Data acquisition is responsible to retrieve all data, filter noise, manage redundancy and provide the valid sensor

values to be used for control law computation. Data acquisition also flags the sensor status for FDIR.

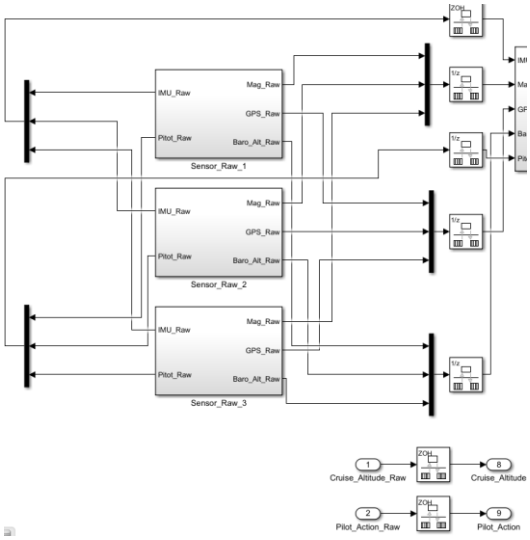
There are two types of data to be prepared:

- Sensor: Data from the sensors
- Pilot: Data from the telemetry about pilot actions (control stick, target parameters, arm/disarm)

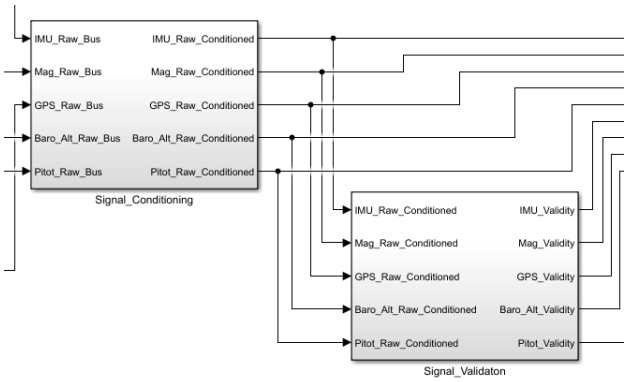


Sensor Redundancy

All sensors have triple redundancy for fault tolerance

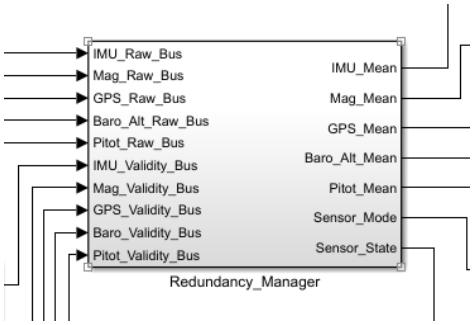


Signal Conditioning and Validation



Data from all the sensors are passed through a low-pass filter to reduce noise. The conditioned signals are validated in terms of data-type and min-max ranges. Signals from sensors that fail validation are flagged as invalid.

Redundancy Management



Conditioned and validated signals are passed to the Redundancy Management. The responsibility of the redundancy manager is to:

- Ignore invalid signal
- Compute relative difference between sensor values
- Compute mean sensor value from the healthy sensors
- Flag unhealthy sensors and determine the overall mode of sensor operation

Sensor Mode

Sensor_Mode.m

```
classdef Sensor_Mode < Simulink.IntEnumType
    enumeration
        OK (0)
        D1 (1)
        D2 (2)
    end
end
```

The sensor modes are defined as:

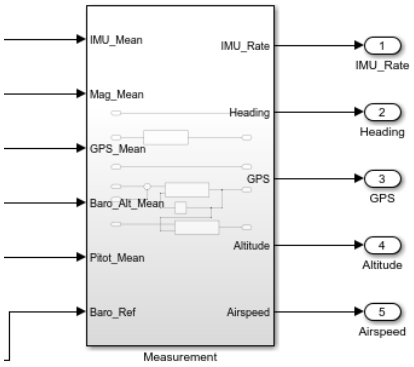
- OK : All three sensors are valid and within tolerable relative errors

- D1: One sensor is invalid or not within tolerable relative error
- D2: Two sensors are invalid or neither of the three sensors agree in tolerable relative error

Measurement Data

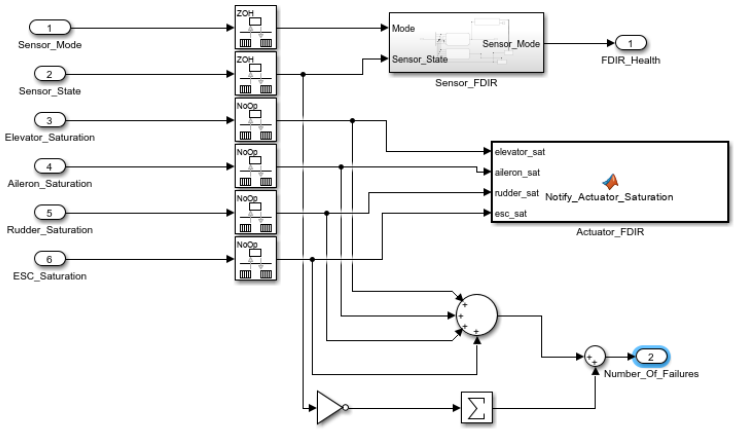
The final output of data acquisition is measurement data from the raw values of the valid sensor readings. The measurement data is defined as :

[IMU_rate, heading, GPS_data, altitude, airspeed]

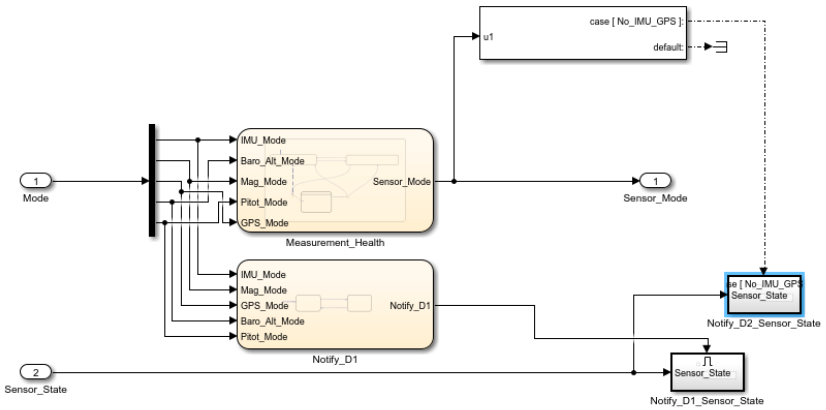


Fault Detection, Isolation, and Recovery

The FDIR responsibility is to identify failure modes, number of failures and provide this data to the control loops to execute fail-safe control laws. It also notifies through telemetry about the failures.



Sensor FDIR



The Sensor FDIR mode are defined as:

Sensor_FDIR_Mode.m

```
classdef Sensor_FDIR_Mode < Simulink.IntEnumType
    enumeration
        % Healthy Modes
        Healthy (0)
        % Compensated Modes
        GPS_Alt (1)
        GPS_Spd (2)
        GPS_Hdg (3)
        GPS_Alt_Spd (4)
        GPS_Spd_Hdg (5)
        GPS_Alt_Hdg (6)
        GPS_Compensated (7)
        No_GPS (8)
        % Failure Modes
        No_IMU (9)
        No_IMU_GPS (10)
    end
end
```

Compensated Modes are used when Heading, Speed, or Altitude sensor values are not valid. In this case, GPS data is used to compensaate for the lost measurement values.

The decision logic for the sensor FDIR mode is defined in the **`Measurement_Health`** state chart.

Loss of IMU or loss of IMU and GPS is critical to the system and requires fail-safe control mechanism.

Actuator FDIR

Actuator FDIR notifies telemetry when elevator, aileron, rudder or ESC control commands are saturated.

Number of Failures

Each sensor with D1 or D2 state is considered as a failure count.

Each actuator in saturation is considered as a failure count.

Mission Management

The mission management responsibility is to:

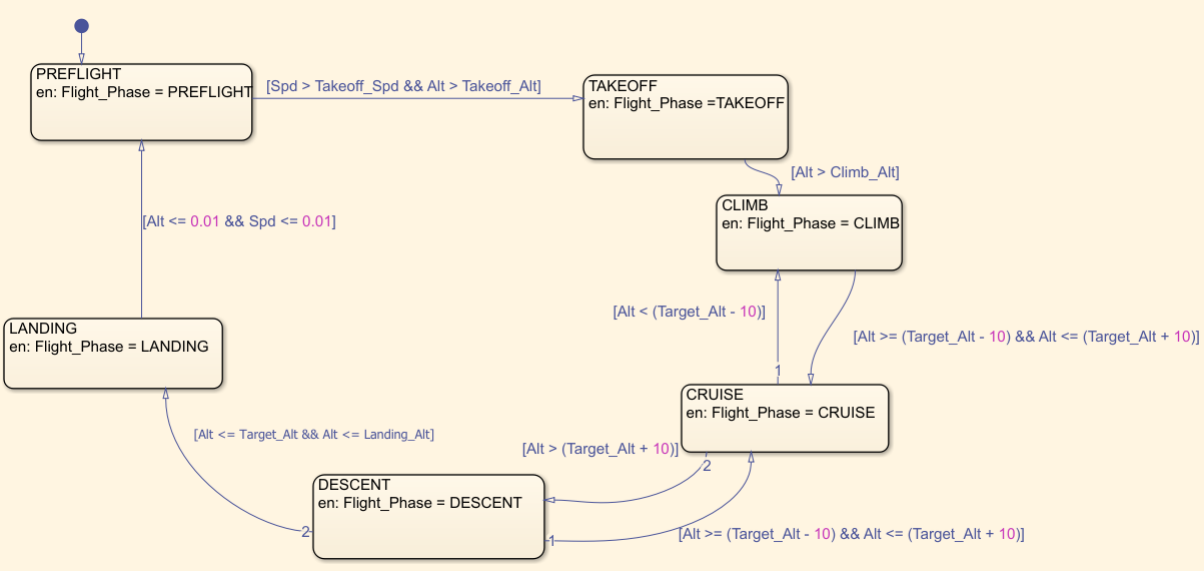
- Compute the flight phase of the mission
- Compute the autopilot mode based on the FDIR health and pilot actions
- Manage transition between the autopilot modes

Flight Phase

The flight phases are defined as:

```
Flight_Phase.m
classdef Flight_Phase < Simulink.IntEnumType
    enumeration
        PREFLIGHT (0)
        TAKEOFF (1)
        CLIMB (2)
        CRUISE (3)
        DESCENT (4)
        LANDING (5)
    end
end
```

The transition logic is as follows:



Autopilot Mode

The autopilot modes are defined as:

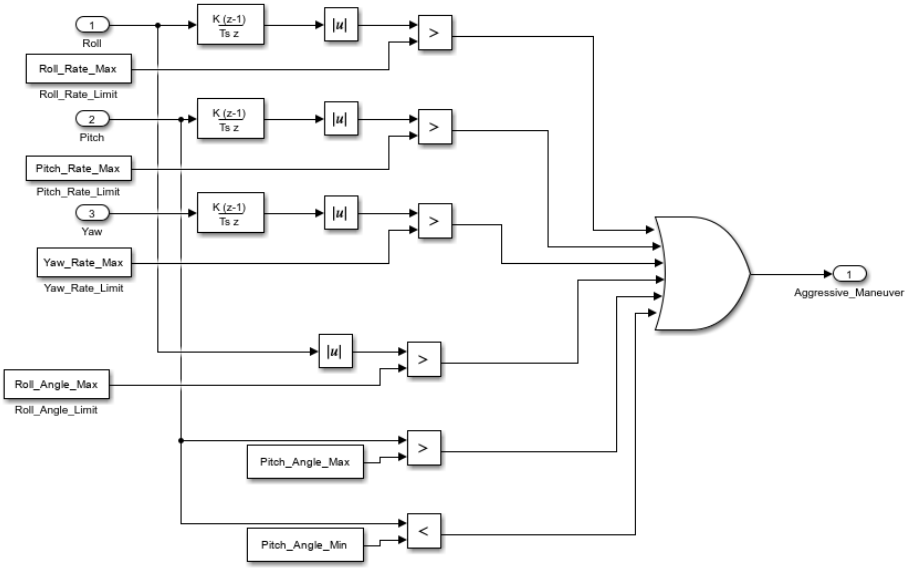
- **Autonomous**: The UAV is able to fly within constraints on the lateral and vertical profile reference without any intervention.
- **Manual:** Pilot uses control stick to control the roll, pitch, yaw and throttle.
- **Stabilized:** The UAV stabilizes its roll, pitch, yaw axis to maintain zero.
- **Return To Home**: The UAV flies back to the origin point of the mission.
- **Arcade**: The UAV is able to fly pilot defined altitude, speed and heading targets autonomously.

- **Degraded**: Pilot uses control stick to control the control surface deflections and throttle directly.
- **Disable**: No commands to the actuators.

Autopilot Mode	Inner Loop Input	Outer Loop Input	Outer Loop Output
Disable	X	X	X
Degraded	X	X	X
Manual	Rates from control stick	X	X
Stabilized	Internal dynamics of UAV	X	X
Arcade	Alt/Spd/Hdg target from pilot	X	Alt/Spd/Hdg target from pilot
Autonomous	Alt/Spd/Hdg target from Outer Loop	GPS/Alt/Spd/Hdg measurements	Alt/Spd/Hdg target from Outer Loop

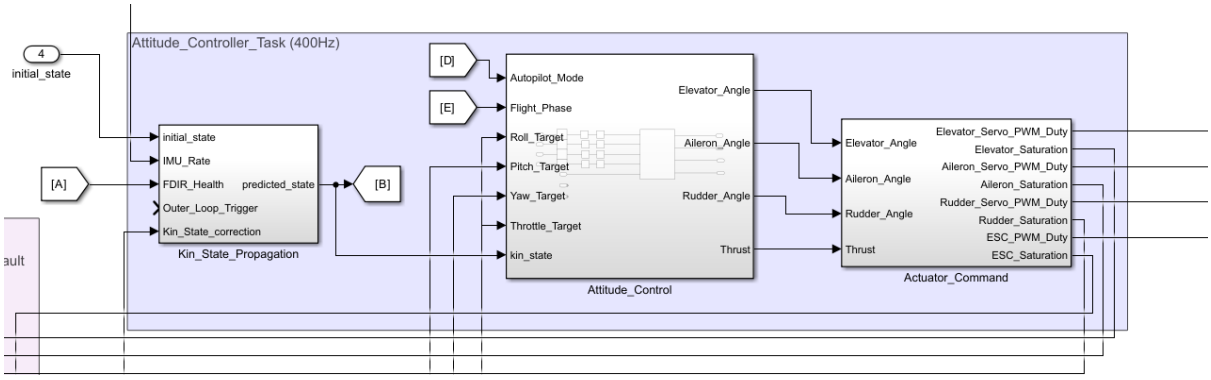
The transition logic between the autopilot modes is defined in **`Autopilot_Mode_Manager`** state chart.

Aggressive Manoeuvre Detection



If the rate of change of the roll, pitch, or yaw is more than acceptable values, it is considered as “aggressive manoeuvre”. This will trigger the Autopilot to move back to Manual or Stabilized mode. (Refer **`Autopilot_Mode_Manager`** state chart).

Attitude Control

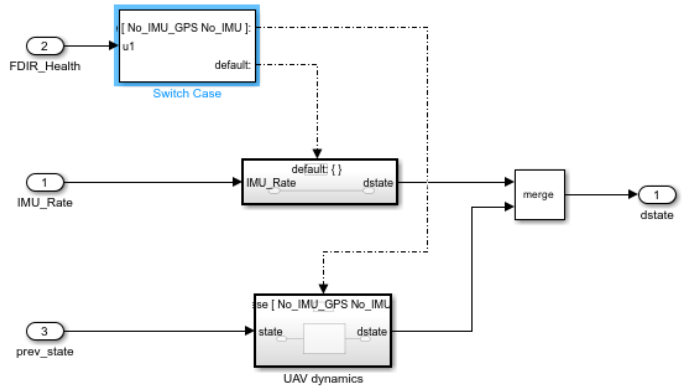


This is the inner control loop. Attitude control handles high-frequency state estimation, control law execution, command output validation and physical actuator commanding.

The execution flow is structured sequentially across three main subsystems:

State Prediction

Kin_State_Propagation block handles the high-rate attitude prediction. It integrates raw IMU_Rate data to calculate the continuous predicted_state of the aircraft. It also includes input ports for Outer_Loop_Trigger and Kin_State_correction, which allows it to periodically overwrite its own drifting IMU integration with the highly accurate 50 Hz Kalman filter estimates.

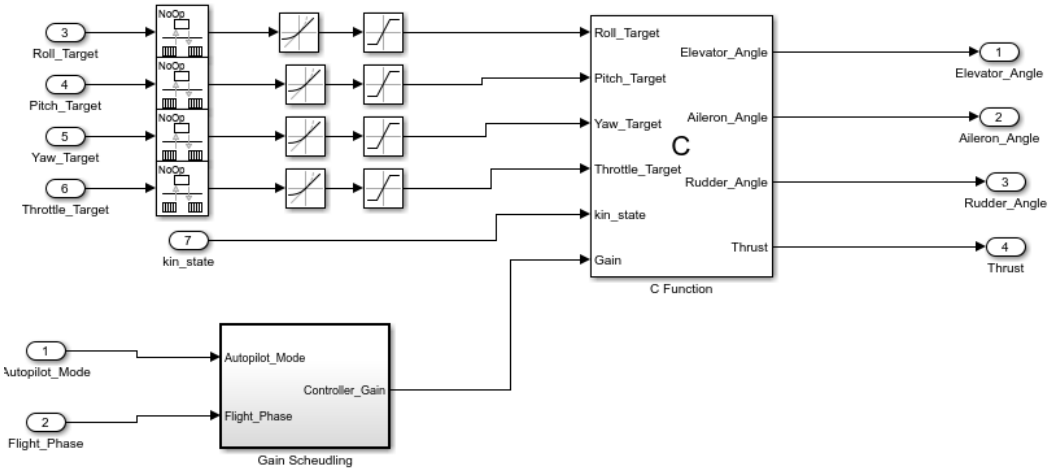


For failure modes when IMU is not available, an internal UAV dynamics is used to propagate the kinematic state of the UAV from the last predicted state.

Attitude Control

Attitude_Control is the core flight dynamics block. It ingests the newly calculated predicted_state alongside the target setpoints (Roll_Target, Pitch_Target, Yaw_Target, Throttle_Target) provided by the slower navigation loop. Using the current Autopilot_Mode and Flight_Phase context, it calculates the required aerodynamic surface deflections (Elevator_Angle, Aileron_Angle, Rudder_Angle) and Thrust needed to achieve those targets.

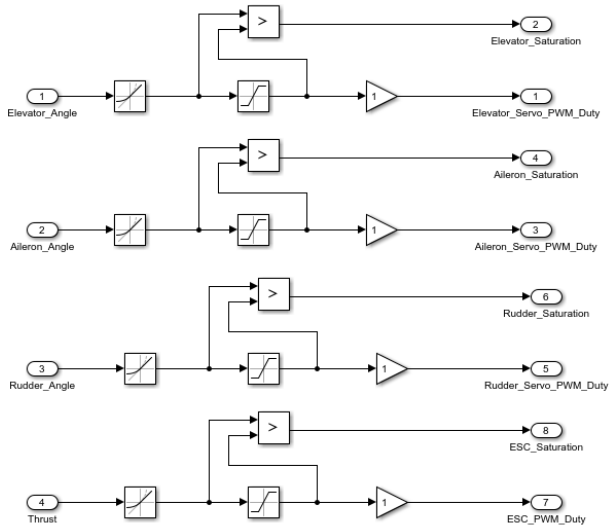
The Autopilot_Mode and Flight_Phase are used correctly schedule gain and limits of the controller. The black box control law in C is placed inside the Attitude_Control block.



Actuator Command

This final hardware-abstraction layer translates the aerodynamic math into physical electrical signals. It maps the requested angles and thrust into specific PWM duty cycles (_Servo_PWM_Duty and ESC_PWM_Duty) to drive the servos and electronic speed controller.

The command signals are rate limited and saturated for safe operation.



Crucially, it also calculates saturation flags for each actuator, which are used to notify the system if a control surface has reached its maximum physical deflection limit.

Guidance and Navigation

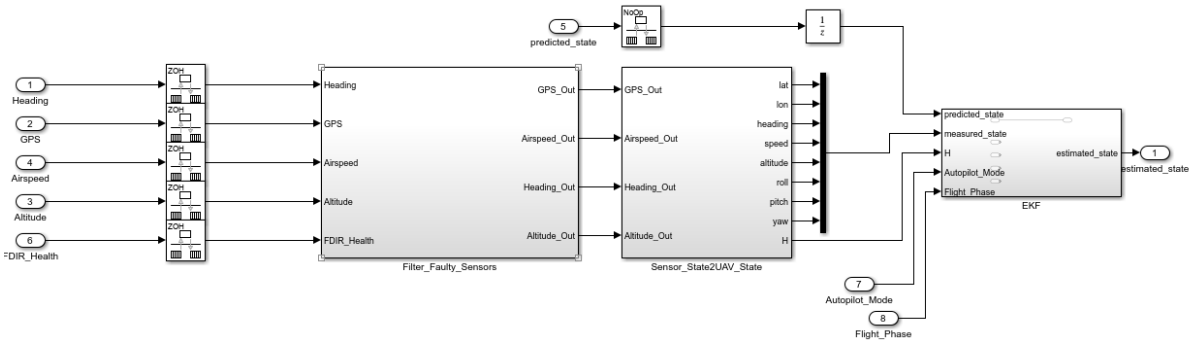
This is the outer control loop. The primary responsibility is navigation: translating high-level mission flight plans into immediate, actionable attitude setpoints for the inner loop. It performs the correction step of state estimation using Extended Kalman Filter (EKF) for better state estimate.

Measurement Update

Faulty sensors are removed from the measurement update step by updating the corresponding rows of measurement matrix to zero.

The sensor quantities are converted to measurement vector quantities.

The predicted state from Attitude Control is corrected using propagated process error covariance, Kalman Gain, and propagated measurement error covariance.

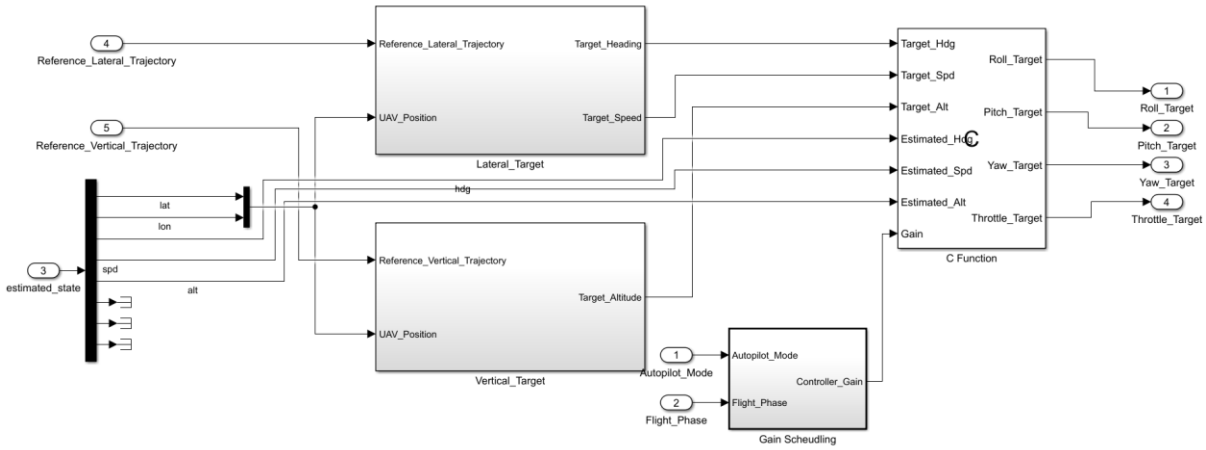


Autopilot_Mode and Flight_Phase are used to compute the correct linearization of the measurement matrix and state matrix.

Guidance (Position and Altitude Controller)

These blocks act as the trajectory managers. They ingest the overall mission waypoints (Reference_Lateral_Trajectory, Reference_Vertical_Trajectory) and compare them against the UAV's current 3D position extracted from the estimated_state (Latitude, Longitude, Altitude). They output the immediate kinematic goals needed to stay on the path: Target_Heading, Target_Speed, and Target_Altitude.

The Autopilot_Mode and Flight_Phase are used correctly schedule gain and limits of the controller. The black box control law in C is placed inside the Guidance block.



RTOS Architecture

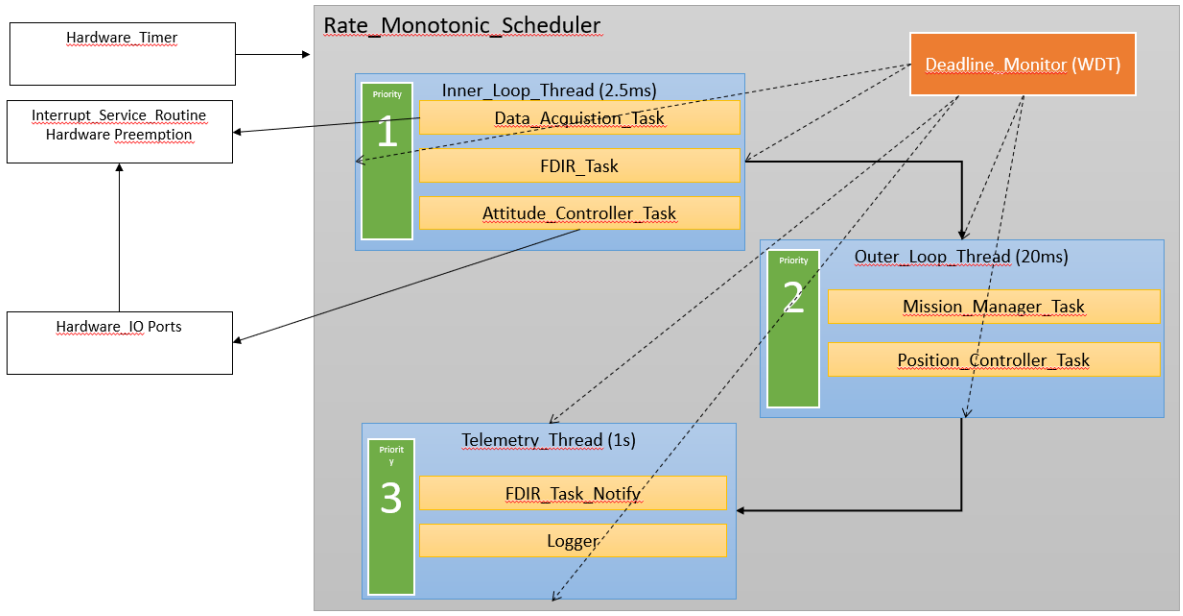
Scheduler

The flight software employs a pre-emptive Rate Monotonic Scheduler (RMS) managed by a hard Real-Time Operating System (RTOS).

Tasks

Task Name	Type	Period	Simulink Reference
Data_Acquisition_Task	Periodic	2.5ms	Data_Acquisition_Task
FDIR_Task	Periodic	2.5ms	FDIR_Task (without notification)
Attitude_Controller_Task	Periodic	2.5ms	Kin_State_Propagation, Attitude_Control, Actuator_Command
Mission_Manager_Task	Periodic	20ms	Mission_Manager_Task
Position_Controller_Task	Periodic	20ms	Measure_State_Propagation, Guidance
Telemetry_Task	Periodic	1s	FDIR_Task (notification) + Logger (not in Simulink architecture)

Threads and Priority



The tasks are assigned into three threads based on the function, data flow and execution time.

Thread	Description	Priority
Inner_Loop_Thread	Sensor ports reading FDIR computation State prediction Attitude Control	1 (highest)
Outer_Loop_Thread	Mission mode computation State measurement update Guidance and Navigation	2
Telemetry_Thread	Notification from FDIR and logging	3 (lowest)

Deadline Monitor

A deadline monitor keeps track of the time elapsed during each thread's execution (time before thread execution starts and time when thread execution ends).

Detection

Rather than relying solely on software timers which can drift or freeze, the Deadline Monitor is driven by an independent Hardware Watchdog Timer (WDT) or a highest-priority RTOS trace hook. At the end of every successful 2.5 ms cycle, the inner loop must "kick" (signal) the monitor. If the thread takes too long to compute—due to a software freeze or excessive matrix math in the black-box control laws—the monitor does not receive the signal in time and officially registers an execution overrun.

Response

The monitor dictates a two-tiered fault response based on severity:

Single Overrun (Transient Fault): If an overrun occurs once, the monitor logs the anomaly in the FDIR system, aborts the delayed cycle, and immediately forces the start of the next 2.5 ms frame to maintain the physical actuator update cadence.

Consecutive Overruns (Critical Fault): If multiple overruns occur in a row (e.g., 5 consecutive missed deadlines), it indicates a catastrophic software lock-up or infinite loop. The Deadline Monitor possesses the highest system authority: it completely bypasses the RTOS scheduler and triggers a hardcoded hardware failsafe, which immediately drives all control surfaces to a neutral position and cuts throttle to the ESC.

Inter-Process Communication

Task	**Data**	**Read/Write**	**Mechanism**
Data_Acquisition_Task	IMU_Rate Heading GPS Altitude Airspeed Sensor_Mode Sensor_State Target_Altitude Manual_Commands	Read	Priority-Inheritance Mutex
FDIR_Task	FDIR_Health Number_Of_Failures	Write	Priority-Inheritance Mutex
Attitude_Controller_Task	IMU_Rate Kin_State_Correction Roll_Target Pitch_Target Yaw_Target Throttle_Target Autopilot_Mode Flight_Phase	Read	Priority-Inheritance Mutex
Attitude_Controller_Task	Predicted_State Elevator_Saturation Aileron_Saturation Rudder_Saturation ESC_Saturation Predicted_State	Write	Priority-Inheritance Mutex/Double-Buffering for Predicted_State
Mission_Manager_Task	Pilot_Action Target_Alt FDIR_Health Predicted_State Number_Of_Failures	Read	Priority-Inheritance Mutex/Double-Buffering for Predicted_State
Mission_Manager_Task	Autopilot_Mode Flight_Phase	Write	Priority-Inheritance Mutex

Position_Controller_Task	Heading GPS Altitude Airspeed Predicted_State FDIR_Health	Read	Priority-Inheritance Mutex/Double-Buffering for Predicted_State
Position_Controller_Task	Roll_Target Pitch_Target Yaw_Target Throttle_Target	Write	Priority-Inheritance Mutex