Index

Suparna Drone Detailed Architecture

• Flight Controller

1. Overview

Suparna control stack runs on autopilot hardware to control drones, UAVs, and other unmanned vehicles. It offers robust capabilities for controlling a wide range of vehicles like multi-copters, fixed-wing, VTOLs, and ground vehicles. Here a quadcopter with autonomous capabilities is presented

2. System Components

The system is composed of several key components that work together to ensure smooth flight operations and management of the vehicle. These components include the Flight Stack, Middleware, Hardware Abstraction, and the Operating System.

3. Flight Stack

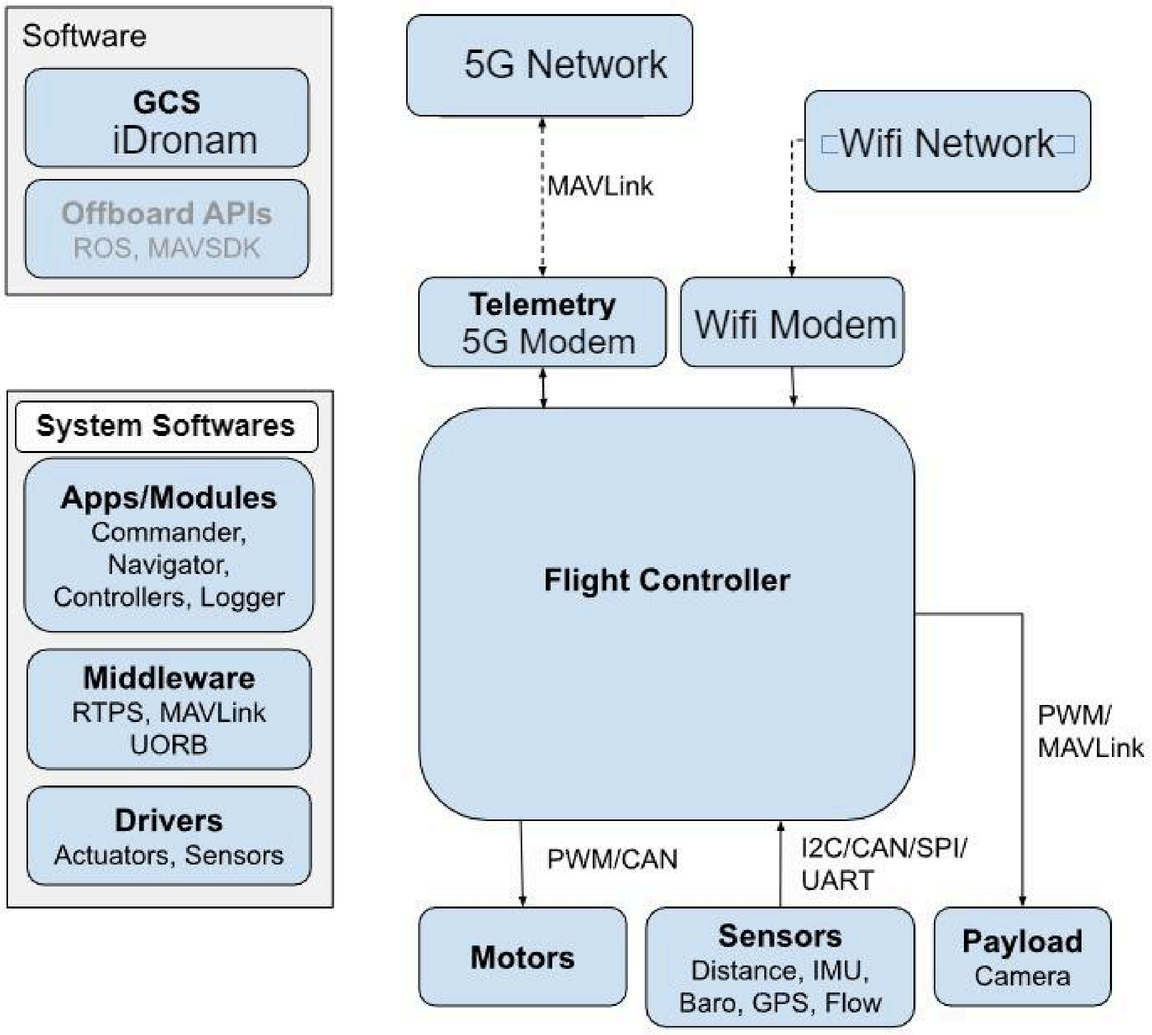
The Flight Stack includes the navigation, position estimation, and attitude controllers. It is responsible for ensuring the vehicle follows flight paths, maintains stability, and reaches its intended destination.

4. Middleware

PX4's Middleware facilitates communication between different parts of the system, such as the flight control and sensors. It provides standard interfaces for vehicle components, making the system more modular and extensible.

5. Hardware Abstraction Layer (HAL)

The Hardware Abstraction Layer separates the operating system and hardware-specific implementations from the higher-level flight logic. This ensures that PX4 can run on various



autopilot hardware with minimal changes to the software.

6. Operating System

The controller is designed to run on top of a real-time operating system, providing low-latency, predictable scheduling required for flight control applications.

• Companion Computer

Here’s how the Companion (Mission) Computer works with the Suparna drone and its Flight Management System (FMS):

1. Mission Computer (Companion Computer)

Role: It serves as the brain of the drone, handling onboard decision-making. It integrates artificial intelligence (AI) for complex tasks such as collision avoidance, object detection, and autonomous navigation.

Inputs:

It receives data from multiple Sensors (like the Depth Camera, IMU, GPS, Barometer) and Payloads (MAVLink Camera) to assess the drone's surroundings and flight conditions.

It communicates with the Flight Controller via the MAVLink protocol to issue flight commands based on the AI-driven decisions.

Outputs:

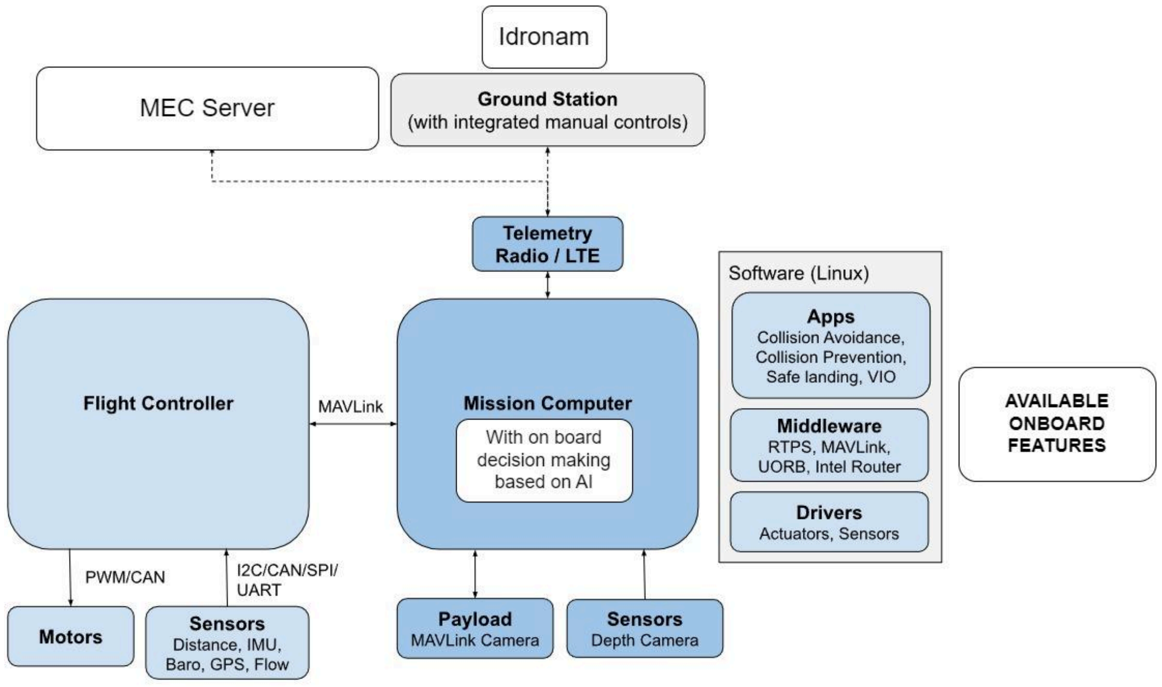
Provides instructions back to the Flight Controller for managing the Motors and executing commands related to the drone's movement, stability, and flight path.

2. Flight Controller

Role: The Flight Controller (e.g., Pixhawk) is responsible for real-time, low-level flight operations, controlling the drone's motors, ensuring balance, and managing input from the companion computer.

Inputs: It collects real-time data from various onboard Sensors (like IMU, GPS, Flow sensors) and receives commands from the Mission Computer through the MAVLink communication interface.

Outputs: Executes commands for controlling Motors and regulating flight through PWM and


Idronam
MEC Server
Ground Station (with integrated manual controls)
Telemetry Radio / LTE
Software (Linux)
Apps
Collision Avoidance, Collision Prevention, Safe landing, VIO
Middleware
RTPS, MAVLink, UORB, Intel Router
Drivers
Actuators, Sensors
AVAILABLE ONBOARD FEATURES
Flight Controller
MAVLink
Mission Computer
With on board decision making based on AI
PWM/CAN
I2C/CAN/SPI/ UART
Motors
Sensors
Distance, IMU, Baro, GPS, Flow
Payload
MAVLink Camera
Sensors
Depth Camera

CAN interfaces.

3. Communication Channels

MAVLink Protocol: The Mission Computer and Flight Controller communicate using the MAVLink protocol, enabling exchange of real-time flight data and control commands.

Telemetry 5G/WiFi: Provides communication between the Ground Station and the onboard systems (Mission Computer, Flight Controller) for sending and receiving telemetry data, manual control signals, or updates.

4. Ground Station and MEC Server

Ground Station: Acts as a remote control hub, integrating manual controls when necessary. It communicates with the Mission Computer via the telemetry radio.

MEC Server: A cloud-based system that connects to the Ground Station, offering advanced computing resources for data analysis or offloading AI tasks.

5. Onboard Software Stack (Linux)

Apps: Includes collision avoidance, prevention, safe landing, and visual-inertial odometry (VIO), all operating on the Mission Computer.

Middleware: Handles communication protocols such as RTPS, MAVLink, and UORB, facilitating data transfer between the Mission Computer, sensors, and actuators.

Drivers: Connect sensors and actuators, enabling the Mission Computer to control hardware components effectively.

In summary, the Mission Computer is critical for high-level decision-making, while the Flight Controller focuses on stabilizing and executing flight commands. The two communicate through MAVLink, integrating data from sensors and issuing commands to the motors for autonomous operations.

• Data and Control Flow architecture

Figure below describes the proposed multilayered architecture of the system. It consists of four layers for the integration of diverse technologies and providing abstraction.

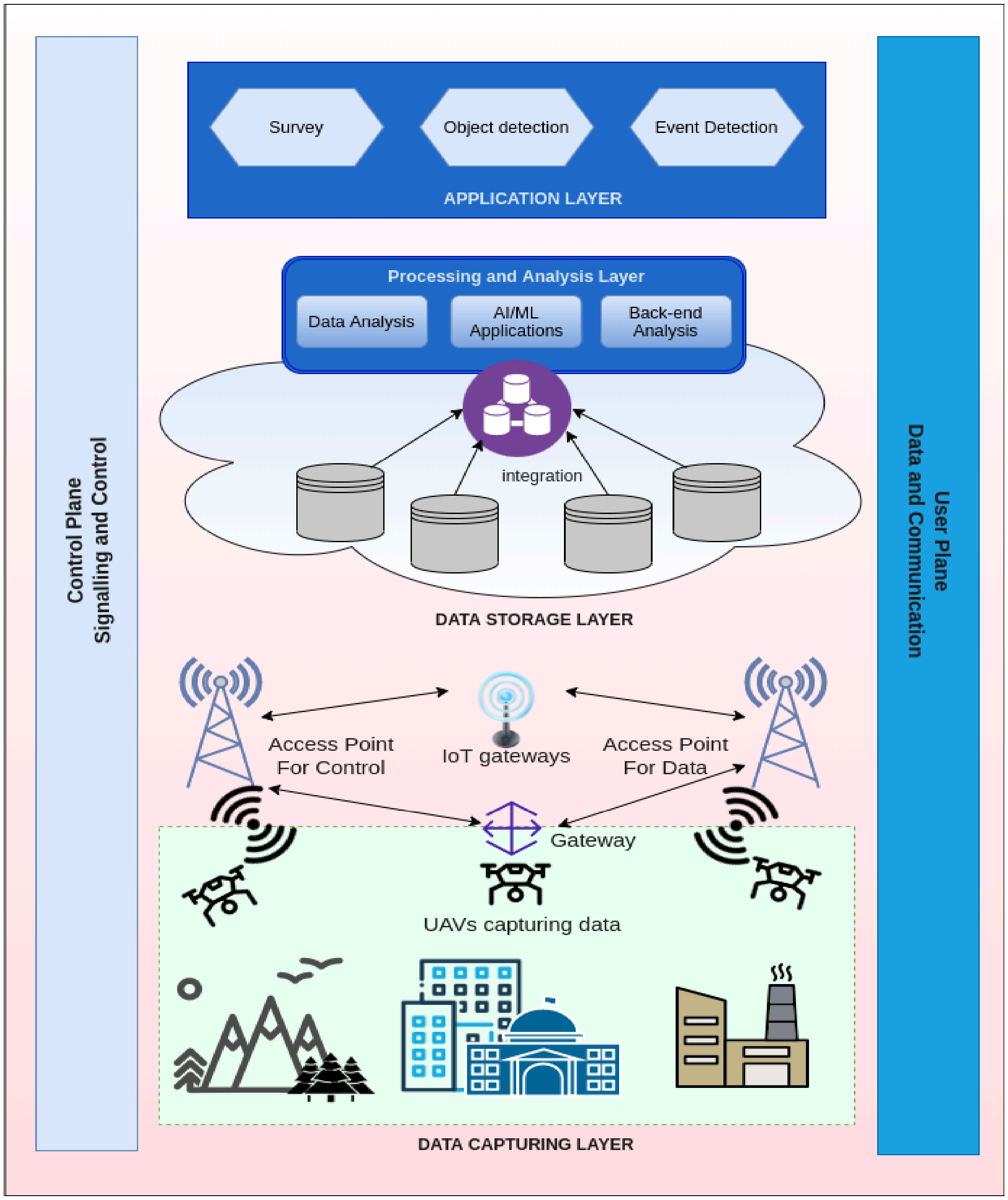
Data Capturing Layer: This is the most important step in the whole design. The proposed system tends to offer a variety of methods like UAVs, standalone cameras, and cameras on UAVs, microphones, sensors, IoT sensors, etc. to capture data. The data capturing layer will facilitate data capturing across multiple devices with minimum complexities to end-users.

Interaction with Data Storage Layer: The data capturing layer will provide seamless connectivity to the data storage layer.

Data Storage Layer: This layer will accept raw data from the data storage layer and store it in the agreed format. The layer may provide facilities for data annotations and similar tasks. The major challenge here will be designing schema for the multi-domain data. The layer shall be responsible for enforcement of security policies, framing data access guidelines, and shall offer access to the data in real-time. Provisions will be made to offer data in the format as desired by the Processing and Data Analytics layer.

Processing and Data Analytics Layer:

The aim of this layer is to offer a variety of data analytics services to different applications. The layer will offer facilities for designing and developing machine learning models and data analysis models for monitoring and surveillance-centric applications.



Application Layer: This layer is concerned with user applications. Different end-users using the system will be citizens, industry, government agencies, etc.

User interface: Different interfaces may be offered to different users and the type of interface will be governed by the applications being offered. The data in the processed form will be accepted by this layer from the data storage layer and will be offered to end-users.

Communication pillars: The pillars consist of the Control Plane and User Plane for providing seamless end-to-end communication between devices and layers of the system.

On-Board processing:

The UAVs will be equipped with hardware for executing lightweight deep learning models for onboard processing of the data that is captured to provide real-time feedback to the navigation and data aggregation system. These will be loaded with algorithms and models developed under the project and will then be used for onboard decision making.

• Collision avoidance Mapping and autonomy Architecture

The system utilizes multiple sensor data fusion for mapping and collision avoidance in the Suparna drone; various sensors (such as depth cameras, LiDAR, IMUs, GPS, etc.) provide complementary data to enhance environmental perception and navigation capabilities. Here's a revised explanation of the architecture:

1. Input from Multiple Sensors:

The system receives data from a combination of sensors, including:

Camera: Provides depth information of the surroundings.

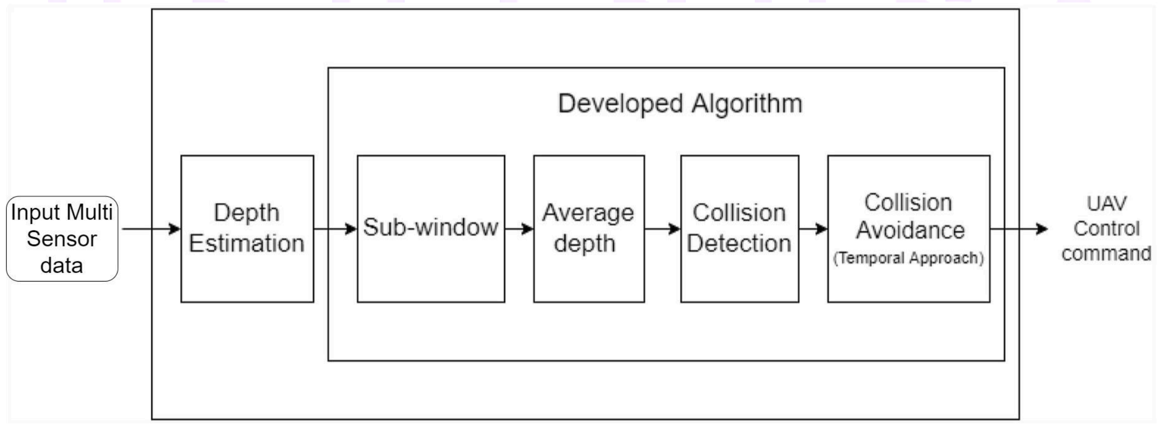
LiDAR: Measures precise distances to objects using laser-based detection.

IMU (Inertial Measurement Unit): Provides information about the drone's orientation, acceleration, and angular velocity.

GPS: Offers precise location and altitude data for global positioning.

Range finding Sensors: Provide additional proximity data for close-range object detection.

These sensors work together to give a comprehensive understanding of the environment.



2. Data Fusion:

The data from all sensors is fused in a centralized module to generate a multi-dimensional environmental map. This process combines the depth information from the depth camera and LiDAR, positional data from GPS, and motion/orientation information from the IMU.

This fusion allows the system to overcome the limitations of individual sensors. For example, LiDAR provides high-accuracy distance measurements, while depth cameras offer a rich visual understanding, and IMU/GPS ensures stable positioning.

3. Sub-windowing:

Similar to the original design, the combined sensor data is divided into sub-windows for localized analysis. Each sub-window contains fused data (e.g., depth, distance, motion) for a specific region around the drone.

4. Average Depth Calculation:

For each sub-window, the system calculates an average depth based on the combined sensor inputs, ensuring the most accurate representation of object distances. This step smooths out discrepancies between sensors and reduces noise from individual data sources.

5. Collision Detection:

The fused sensor data in each sub-window is then analyzed for potential obstacles. The collision detection algorithm compares real-time fused data against pre-set thresholds for safe distances and detects whether any object is on a collision path.

6. Collision Avoidance:

Once a potential collision is detected, the system engages in temporal collision avoidance. Using data from the IMU and GPS, it predicts the drone's future movement trajectory and adjusts its flight path in real-time to avoid obstacles.

The system considers the drone’s speed, direction, and the position of obstacles to execute precise avoidance maneuvers.

7. UAV Control Command:

Finally, the system sends control commands to the flight controller based on the combined sensor data. This ensures that the drone adjusts its velocity, direction, or altitude to avoid obstacles while maintaining a safe and optimal flight path.

Summary:

By using multiple sensor data fusion, the Suparna drone benefits from increased accuracy, redundancy, and robustness in its perception and navigation capabilities. Fusing inputs from depth cameras, LiDAR, IMU, GPS, and other sensors ensures that the drone has a comprehensive understanding of its environment. The system can detect obstacles with greater reliability and execute more efficient collision avoidance maneuvers, resulting in safer and more autonomous flight

Protocol Details for Communication Between the GCS and Drone

Mission Protocol

The mission sub-protocol allows a GCS or developer API to exchange mission (flight plan), geofence and safe point information with a drone/component.

The protocol covers:

Operations to upload, download and clear missions, set/get the current mission item number, and get notification when the current mission item has changed.

Message type(s) and enumerations for exchanging mission items.

Mission Items ("MAVLink commands") that are common to most systems.

The protocol supports re-request of messages that have not arrived, which allows missions to be reliably transferred over a lossy link.

Mission Types

MAVLink 2 supports three types of "missions": flight plans, geofences and rally/safe points. The protocol uses the same sequence of operations for all types (albeit with different types of Mission Items). The mission types must be stored and handled separately/independently.

Mission protocol messages include the type of associated mission in the mission_type field (a MAVLink 2 message extension). The field takes one of the MAV_MISSION_TYPE enum values: MAV_MISSION_TYPE_MISSION, MAV_MISSION_TYPE_FENCE, MAV_MISSION_TYPE_RALLY.

Mission Items (MAVLink Commands)

Mission items for all the mission types are defined in the MAV_CMD enum.

MAV_CMD is used to define commands that can be used in missions ("mission items") and commands that can be sent outside of a mission context (using the Command Protocol). Some MAV_CMD can be used with both mission and command protocols. Not all commands/mission items are supported on all systems (or for all flight modes).

The items for the different types of mission are identified using a simple name prefix convention:

Flight plans:

NAV commands (MAV_CMD_NAV_*) for navigation/movement (e.g. MAV_CMD_NAV_WAYPOINT, MAV_CMD_NAV_LAND)

DO commands (MAV_CMD_DO_*) for immediate actions like changing speed or activating a servo (e.g. MAV_CMD_DO_CHANGE_SPEED).

CONDITION commands (MAV_CMD_CONDITION_*) for changing the execution of the mission based on a condition - e.g. pausing the mission for a time before executing next command (MAV_CMD_CONDITION_DELAY).

Geofence mission items:

Prefixed with MAV_CMD_NAV_FENCE_ (e.g. MAV_CMD_NAV_FENCE_RETURN_POINT).

Rally point mission items:

There is just one rally point MAV_CMD: MAV_CMD_NAV_RALLY_POINT.

The commands are transmitted/encoded in MISSION_ITEM or MISSION_ITEM_INT messages. These messages include fields to identify the particular mission item (command id) and up to 7 command-specific optional parameters.

Field Name	Type	Values	Description
command	uint16_t	MAV_CMD	Command id, as defined in MAV_CMD.
param1	float		Param #1.
param2	float		Param #2.
param3	float		Param #3.
param4	float		Param #4.
param5 (x)	float / int32_t		X coordinate (local frame) or latitude (global frame) for navigation commands (otherwise Param #5).
param6 (y)	float / int32_t		Y coordinate (local frame) or longitude (global frame) for navigation commands (otherwise Param #6).
param7 (z)	float		Z coordinate (local frame) or altitude (global - relative or absolute, depending on frame) (otherwise Param #7).

The first four parameters (shown above) can be used for any purpose - this depends on the particular command. The last three parameters (x, y, z) are used for positional information in MAV_CMD_NAV_* commands, but can be used for any purpose in other commands.

The remaining message fields are used for addressing, defining the mission type, specifying the reference frame used for x, y, z in MAV_CMD_NAV_* messages, etc.:

Field Name	Type	Values	Description
target_system	uint8_t		System ID
target_component	uint8_t		Component ID
seq	uint16_t		Sequence number for item within mission (indexed from 0).
frame	uint8_t	MAV_FRAME	The coordinate system of the waypoint. PX4 support global frames in MAVLink commands (local frames may be supported if the same command is sent via the command protocol).
mission_type	uint8_t	MAV_MISSION_TYPE	Mission type.
current	uint8_t	false:0, true:1	When downloading, whether the item is the current mission item.
autocontinue	uint8_t		Autocontinue to next waypoint when the command completes.

MISSION_ITEM_INT vs MISSION_ITEM

MISSION_ITEM and MISSION_ITEM_INT are used to exchange individual mission items between systems. MISSION_ITEM messages encode all mission item parameters into float parameters fields (single precision IEEE754) for transmission. MISSION_ITEM_INT is exactly the same except that param5 and param6 are Int32 fields.

Protocol implementations must allow both message types in supported operations (along with the corresponding MISSION_REQUEST and MISSION_REQUEST_INT message types).

MAVLink users should always prefer MISSION_ITEM_INT because it allows latitude/longitude to be encoded without the loss of precision that can come from using MISSION_ITEM.

Message/Enum Summary

The following messages and enums are used by the service.

Message	Description
MISSION_REQUEST_LIST	Initiate mission download from a system by requesting the list of mission items.
MISSION_COUNT	Send the number of items in a mission. This is used to initiate mission upload or as a response to MISSION_REQUEST_LIST when downloading a mission.

MISSION_REQUEST_INT	Request mission item data for a specific sequence number be sent by the recipient using a MISSION_ITEM_INT message. Used for mission upload and download.
MISSION_REQUEST	Request mission item data for a specific sequence number be sent by the recipient using a MISSION_ITEM message. Used for mission upload and download.
MISSION_ITEM_INT	Message encoding a mission item/command (defined in a MAV_CMD). The message encodes positional information in integer parameters for greater precision than MISSION_ITEM. Used for mission upload and download.
MISSION_ITEM	Message encoding a mission item/command (defined in a MAV_CMD). The message encodes positional information in float parameters. Used for mission upload and download.
MISSION_ACK	Acknowledgment message when a system completes a mission operation (e.g. sent by autopilot after it has uploaded all mission items). The message includes a MAV_MISSION_RESULT indicating either success or the type of failure.
MISSION_CURRENT	Message containing the current mission item sequence number. This is emitted when the current mission item is set/changed.
MISSION_SET_CURRENT	Set the current mission item by sequence number (continue to this item on the shortest path).
STATUSTEXT	Sent to notify systems when a request to set the current mission item fails.
MISSION_CLEAR_ALL	Message sent to clear/delete all mission items stored on a system.
MISSION_ITEM_REACHED	Message emitted by system whenever it reaches a new waypoint. Used to monitor progress.

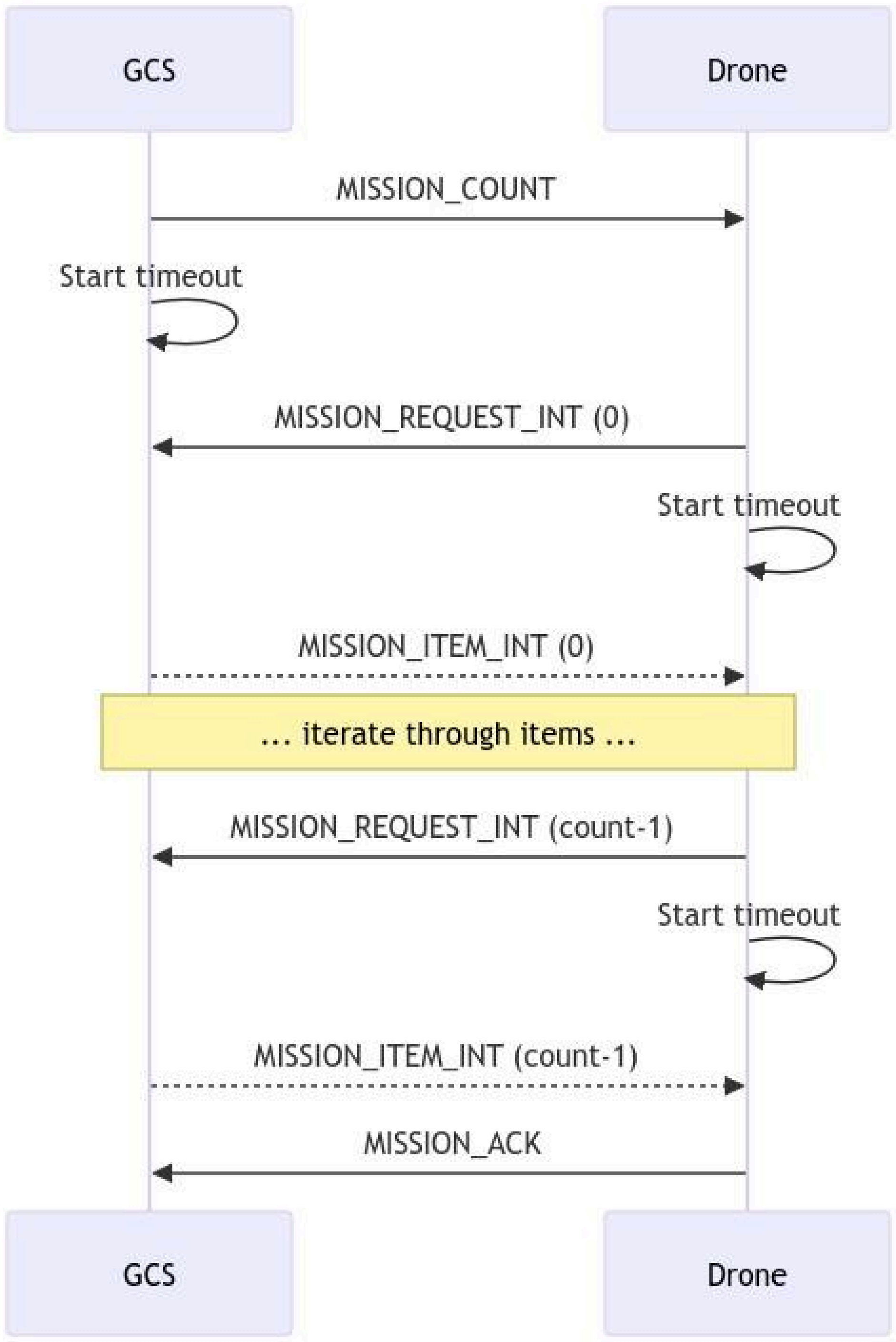
Enum	Description
MAV_MISSION_TYPE	Mission type for message (mission, geofence, rallypoints).
MAV_MISSION_RESULT	Used to indicate the success or failure reason for an operation (e.g. to upload or download a mission). This is carried in a MISSION_ACK.
MAV_FRAME	Co-ordinate frame for position/velocity/acceleration data in the message.
MAV_CMD	Mission Items (and MAVLink commands). These can be sent in MISSION_ITEM or MISSION_ITEM_INT.

Operations

This section defines all the protocol operations.

Upload a Mission to the Vehicle

The diagram below shows the communication sequence to upload a mission to a drone (assuming all operations succeed).



Mission update must be robust! A new mission should be fully uploaded and accepted before the old mission is replaced/removed.

Mission Upload Sequence

In more detail, the sequence of operations is:

GCS sends MISSION_COUNT including the number of mission items to be uploaded (count).

A timeout must be started for the GCS to wait on the response from Drone (MISSION_REQUEST_INT).

Drone receives message and responds with MISSION_REQUEST_INT requesting the first mission item (seq==0).

A timeout must be started for the Drone to wait on the MISSION_ITEM_INT response from GCS.

GCS receives MISSION_REQUEST_INT and responds with the requested mission item in a MISSION_ITEM_INT message.

Drone and GCS repeat the MISSION_REQUEST_INT/MISSION_ITEM_INT cycle, iterating seq until all items are uploaded (seq==count-1).

After receiving the last mission item the drone responds with MISSION_ACK with the type of MAV_MISSION_ACCEPTED indicating mission upload completion/success.

The drone should set the new mission to be the current mission, discarding the original data.

The drone considers the upload complete.

GCS receives MISSION_ACK containing MAV_MISSION_ACCEPTED to indicate the operation is complete.

Note:

A timeout is set for every message that requires a response (e.g. MISSION_REQUEST_INT). If the timeout expires without a response being received then the request must be resent.

Mission items must be received in order. If an item is received out-of-sequence the expected item should be re-requested by the vehicle (the out-of-sequence item is dropped).

An error can be signaled in response to any request using a MISSION_ACK message containing an error code. This must cancel the operation and restore the mission to its previous state. For example, the drone might respond to the MISSION_COUNT request with a MAV_MISSION_NO_SPACE if there isn't enough space to upload the mission.

The sequence above shows the mission items packaged in MISSION_ITEM_INT messages. Protocol implementations must also support MISSION_ITEM and MISSION_REQUEST in the same way.

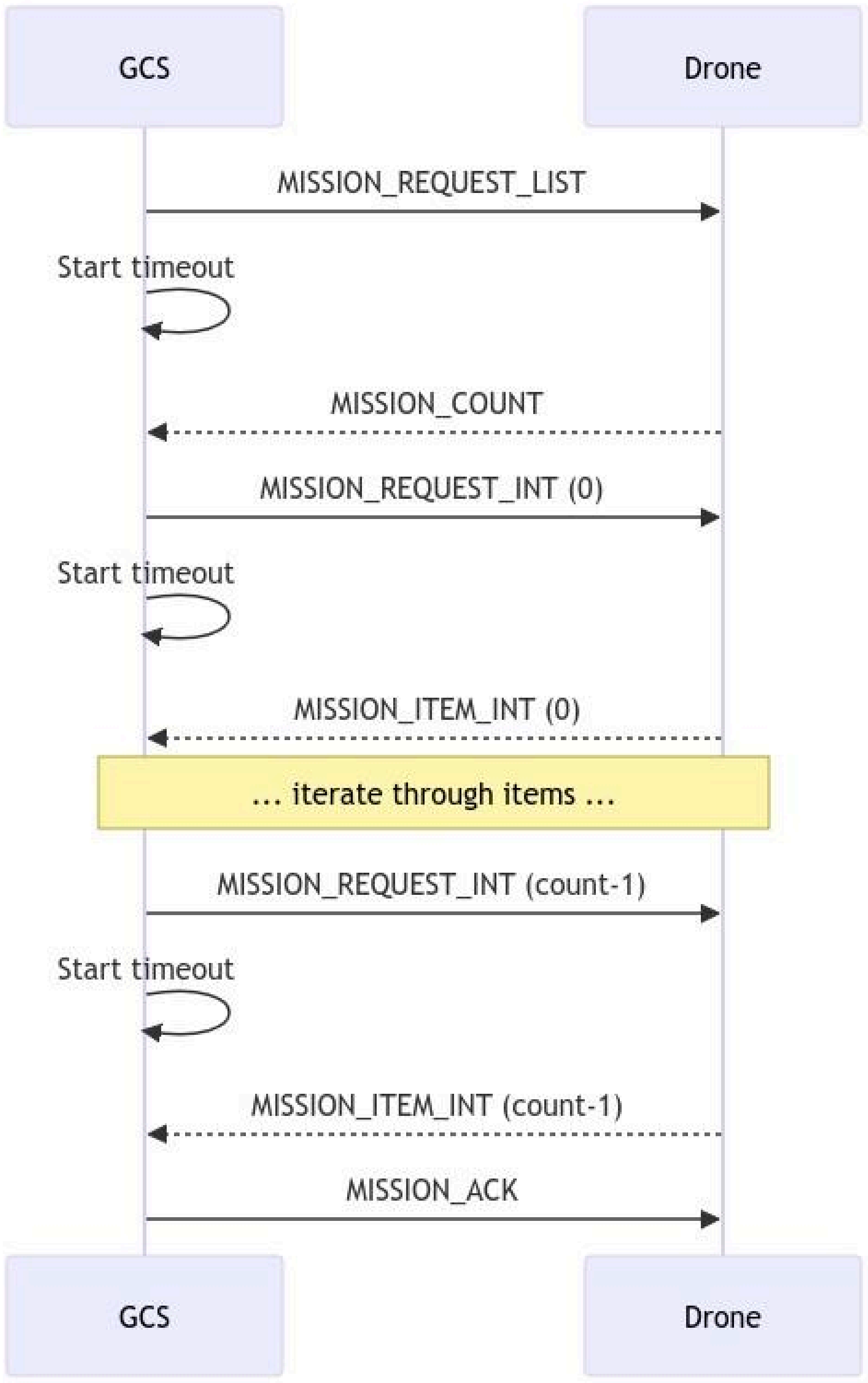
Uploading an empty mission (MISSION_COUNT is 0) has the same effect as clearing the mission.

Download a Mission from the Vehicle

The diagram below shows the communication sequence to download a mission from a drone (assuming all operations succeed).

Sequence: Download mission

The sequence is similar to that for uploading a mission. The main difference is that the client (e.g. GCS) sends MISSION_REQUEST_LIST, which triggers the autopilot to respond with the current count of items. This starts a cycle where the GCS requests mission items, and the drone supplies them.


GCS
Drone
MISSION_REQUEST_LIST
Start timeout
MISSION_COUNT
MISSION_REQUEST_INT (0)
Start timeout
MISSION_ITEM_INT (0)
... iterate through items ...
MISSION_REQUEST_INT (count-1)
Start timeout
MISSION_ITEM_INT (count-1)
MISSION_ACK
GCS
Drone

Note:

A timeout is set for every message that requires a response (e.g. MISSION_REQUEST_INT). If the timeout expires without a response being received then the request must be resent.

Mission items must be received in order. If an item is received out-of-sequence the expected item should be re-requested by the GCS (the out-of-sequence item is dropped).

An error can be signaled in response to any request using a MISSION_ACK message containing an error code. This must cancel the operation.

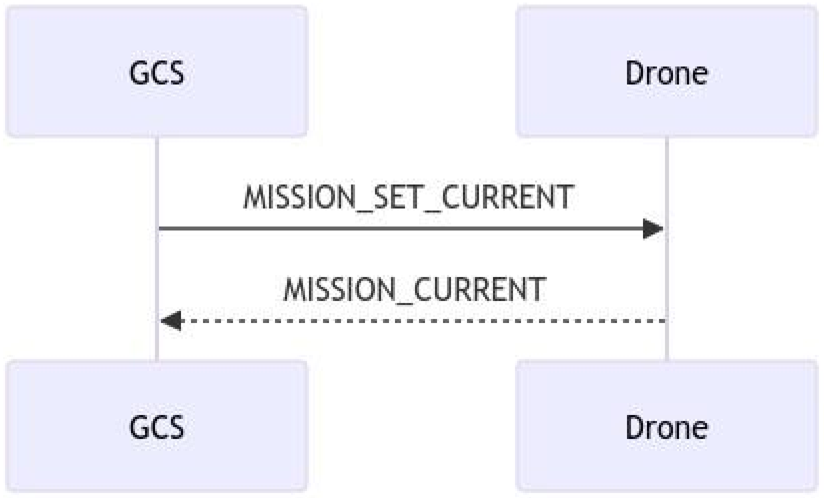
The sequence above shows the mission items packaged in MISSION_ITEM_INT messages. Protocol implementations must also support MISSION_ITEM and MISSION_REQUEST in the same way.

Set Current Mission Item

The diagram below shows the communication sequence to set the current mission item.

Set mission item

In more detail, the sequence of operations is:



GCS/App sends MISSION_SET_CURRENT, specifying the new sequence number (seq).

Drone receives message and attempts to update the current mission sequence number.

On success, the Drone must broadcast a MISSION_CURRENT message containing the current sequence number (seq).

On failure, the Drone must broadcast a STATUSTEXT with a MAV_SEVERITY and a string stating the problem. This may be displayed in the UI of receiving systems.

Notes:

There is no specific timeout on the MISSION_SET_CURRENT message.

The acknowledgment of the message is via broadcast of mission/system status, which is not associated with the original message. This differs from error handling in other operations. This approach is used because the success/failure is relevant to all mission-handling clients.

Monitor Mission Progress

GCS/developer API can monitor progress by handling the appropriate messages sent by the drone:

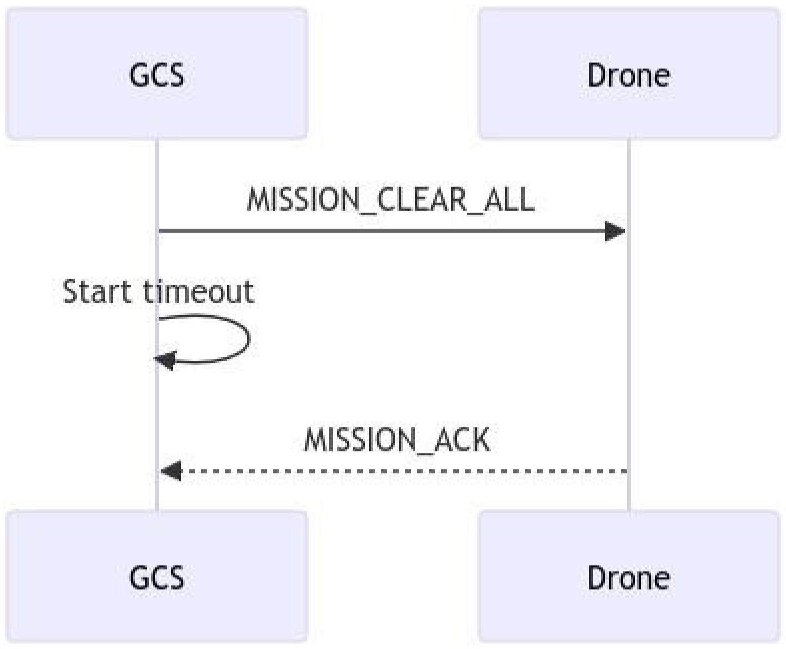
The vehicle must broadcast a MISSION_ITEM_REACHED message whenever a new mission item is reached. The message contains the seq number of the current mission item.

The vehicle must also broadcast a MISSION_CURRENT message if the current mission item is changed.

Clear Missions

The diagram below shows the communication sequence to clear the mission from a drone (assuming all operations succeed).

Clear Missions



In more detail, the sequence of operations is:

GCS/API sends MISSION_CLEAR_ALL

A timeout is started for the GCS to wait on MISSION_ACK from Drone.

Drone receives the message, and clears the mission from storage.

Drone responds with MISSION_ACK with result type of MAV_MISSION_ACCEPTEDMAV_MISSION_RESULT.

GCS receives MISSION_ACK and clears its own stored information about the mission. The operation is now complete.

Note:

A timeout is set for every message that requires a response (e.g. MISSION_CLEAR_ALL). If the timeout expires without a response being received then the request must be resent.

An error can be signaled in response to any request (in this case, just MISSION_CLEAR_ALL) using a MISSION_ACK message containing an error code. This must cancel the operation. The GCS record of the mission (if any) should be retained.

Canceling Operations

The above mission operations may be canceled by responding to any request (e.g. MISSION_REQUEST_INT) with a MISSION_ACK message containing the MAV_MISSION_OPERATION_CANCELLED error.

Both systems should then return themselves to the idle state (if the system does not receive the cancellation message it will resend the request; the recipient will then be in the idle state and may respond with an appropriate error for that state).

Operation Exceptions

Timeouts and Retries

A timeout should be set for all messages that require a response. If the expected response is not received before the timeout then the message must be resent. If no response is received after a number of retries then the client must cancel the operation and return to an idle state.

The recommended timeout values before resending, and the number of retries are:

Timeout (default): 1500 ms

Timeout (mission items): 250 ms.

Retries (max): 5

Errors/Completion

All operations complete with a MISSION_ACK message containing the result of the operation (MAV_MISSION_RESULT) in the type field.

On successful completion, the message must contain type of MAV_MISSION_ACCEPTED; this is sent by the system that is receiving the command/data (e.g. the drone for mission upload or the GCS for mission download).

An operation may also complete with an error - MISSION_ACK.type set to MAV_MISSION_ERROR or some other error code in MAV_MISSION_RESULT. This can occur in response to any message/anywhere in the sequence.

Errors are considered unrecoverable. In an error is sent, both ends of the system should reset themselves to the idle state and the current state of the mission on the vehicle should be unaltered.

Note:

timeouts are not considered errors.

Out-of-sequence messages in mission upload/download are recoverable, and are not treated as errors.

Mission File Formats

The defacto standard file format for exchanging missions/plans is discussed in: File Formats > Mission Plain-Text File Format.

Implementations

PX4

The protocol has been implemented in C.

Source code:

src/modules/mavlink/mavlink_mission.cpp

The implementation status is :

Flight plan missions:

upload, download, clearing missions, and monitoring progress are supported as defined in this specification.

Geofence missions" are supported as defined in this specification.

Rally point "missions" are not supported on PX4.

Mission operation cancellation works for mission download (sets system to idle). Mission operation cancellation does not work for mission uploading; PX4 resends MISSION_REQUEST_INT until the operation times out.

Source code:

src/modules/mavlink/mavlink_mission.cpp

Overall same messages and message flow described in this specification. There are (anecdotally) some implementation differences that affect compatibility. These are documented below.

Source:

/libraries/GCS_MAVLink/GCS_Common.cpp

Flight Plan Missions

Mission upload, download, clearing missions, and monitoring progress are supported.

We implements also partial mission upload using MISSION_WRITE_PARTIAL_LIST, but not partial mission download (MISSION_REQUEST_PARTIAL_LIST). Partial mission upload/download is not an official/standardised part of the mission service.

The implementation differs from this specification (non-exhaustively):

The first mission sequence number (seq==0) is populated with the home position of the vehicle instead of the first mission item.

Mission uploads are not "atomic". An upload that fails (or is canceled) part-way through will not match the pre-update state. Instead it may be a mix of the original and new mission.

Even if upload is successful, the vehicle mission may not match the version on the uploading system (and if the mission is then downloaded it will differ from the original).

There is rounding on some fields (and in some cases internal maximum possible values due to available storage space). Failures can occur if you do a straight comparison of the float params before/after upload.

A MISSION_ACK returning an error value (NACK) does not terminate the upload (i.e. it is not considered an unrecoverable error). As long as GCSt has not yet timed-out a system can retry the current mission item upload.

A mission cannot be cleared while it is being executed (i.e. while in Auto mode). Note that a new mission can be uploaded (even a zero-size mission - which is equivalent to clearing).

Explicit cancellation of operations is not supported. If one end stops communicating the other end will eventually timeout and reset itself to an idle/ready state.

The following behaviour is not defined by the specification (but is still of interest):

System performs some validation of fields when mission items are submitted. The validation code is common to all vehicles; mission items that are not understood by the vehicle type are accepted on upload but skipped during mission execution.

System preforms some vehicle-specific validation at mission runtime (e.g. of jump targets).

A new mission can be uploaded while a mission is being executed. In this case the current waypoint will be executed to completion even if the waypoint sequence is different in the new mission (to get the new item you would need to reset the sequence or switch in/out of auto mode).

System missions are not stored in an SD card and therefore have a vehicle/board-specific maximum mission size (as a benefit, on System , missions can survive SD card failure in flight).

Geofence & Rally Point Plans

System supports Geofence and Rally points on Copter Rover and Sub using this protocol (for MAVLink 2 connections).