Module Guide for Mechatronics Engineering

Team # 34, ParkingLotHawk
Fady Zekry Hanna
Winnie Trandinh
Muhammad Ali
Muhammad Khan

January 19, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	Initial Revision

2 Reference Material

There are no external references used within this document.

2.1 Abbreviations and Acronyms

symbol	description
AC	Anticipated Change
DAG	Directed Acyclic Graph
M	Module
MG	Module Guide
MIS	Module Interface Specifications
OS	Operating System
R	Requirement
Algo	Algorithms
SRS	Software Requirements Specification
Mechatronics Engineering	Mechatronics Engineering is an engineering stream that mixes electrical, mechanical, and software engineering.
UC	Unlikely Change
DDC	Drone Decision and Control
Ardupilot	Open-source Autopilot Software Suite for unmanned vehicles.
ROS Node	Process using ROS functionality.
MavROS	Process presents a ROS interface to interact with Ardupilot and MAVLINK.
High-level Motion Commands	3-dimensional move to commands, e.g. hover at 5m, move 5m left, etc.
MAVLINK	Typical communication network used by hardware peripherals, low-level interface.
ROS	Robot Operating System, open-source robotics middleware suite.
main	Starting point of execution for a process.
PC	Personal Computer.

Contents

List of Tables

List of Figures

3 Introduction

The practice of breaking down a system into smaller, manageable parts, called modules, is a common method used in software development. Each module is assigned to a programmer or programming team as a task. We suggest using a method of decomposition that follows the principle of information hiding. This approach allows for flexibility in future changes as the "hidden information" within each module represents potential changes. This is particularly useful in scientific computing, where modifications and adjustments are often made during the early stages of development as the problem is being explored.

Our design follows the principles of information hiding. Information hiding is a concept in computer science and software engineering that refers to the technique of hiding internal information and state of a software module from other modules to reduce the dependencies between them. The rules we follow are as follows:

- System details that are likely to change independently should be the secrets of separate modules.
- Each data structure is implemented in only one module.
- Any other program that requires information stored in a module's data structures must obtain it by calling access programs belonging to that module.

The Module Guide (MG) is developed after the first stage of design, the Software Requirements Specification (SRS) is completed. The MG outlines the modular structure of the system and serves as a guide for both designers and maintainers to easily identify the different parts of the software. This document can be used by several groups of readers: new project members can use it as a guide to quickly understand the overall structure and find the relevant modules they need, maintainers can use it to improve their understanding of the system when making changes and update the relevant sections after changes have been made, and designers can use it to check for consistency, feasibility, and flexibility in the design, such as consistency among modules, the feasibility of the decomposition, and flexibility of the design.

The remaining portion of the document is organized as follows: Section 4 lists the expected and unexpected changes to the software requirements. Section 5 provides a summary of how the system was divided into modules based on the likely changes. Section 6 outlines the connections between the software requirements and each individual module. Section 7 offers a detailed explanation of each module. Section 8 includes two traceability matrices, one to confirm that the design meets the requirements outlined in the SRS and another that shows the relationship between anticipated changes and the modules. Finally, Section 9 explains the relationship between different modules.

4 Anticipated and Unlikely Changes

This section lists possible changes to the system. According to the likeliness of the change, the possible changes are classified into two categories. Anticipated changes are listed in Section 4.1, and unlikely changes are listed in Section 4.2.

4.1 Anticipated Changes

Anticipated changes are the source of the information that is to be hidden inside the modules. Ideally, changing one of the anticipated changes will only require changing the one module that hides the associated decision. The approach adapted here is called design for change.

AC1 : Interfacing with sensors to read data and share it on the MAVLINK Network.

AC2 : Interfacing with actuators to read and follow commands on the MAVLINK Network.

AC3 : Control algorithms that make the drone follow high level motion commands.

AC4 : Presentation of all sensor and firmware status information into ROS Topics.

AC5 : Presentation of the API to control the drone in the form of ROS Services and ROS Topics.

AC6 : Rudimentary LAN Socket used by higher level modules for LAN communication. There are various types of different sockets, and multiple standards regarding the interface they implement.

AC7 : High-level communication interface used to send and receive string data between Drone and Operator.

AC8 : Framework used for creating video streams to send video from the drone to the Operator's PC.

AC9 : Interfacing with camera hardware to get the latest live image.

AC10 : Framework for the creation of the video streaming pipeline, and routines to send images over it. Components of the pipeline may include compression, communication, and decompression.

AC11 : Receiving images from the video stream created by the drone and presenting them in a simple-to-use API for modules running on the Operator's PC.

AC12 : Choice and layout of interface widgets, e.g. buttons, dropdown boxes, maps, images, etc.

AC13 : Display of parking lot information to the Operator (occupancy map, live video stream).

AC14 : Display of drone information, e.g. current location, current altitude, health status, etc.

AC15 : The starting point of execution (main function) for the process running the Operator's PC.

AC16 : Message passing system used for inter-process communication on the drone.

AC17 : Subscription and extraction of data from ROS Topics to read data needed by the various Algorithm's e.g. if an algorithm module needs a different sensor reading then one module should be modified to provide the sensor reading.

AC18 : The ROS Topic Publisher's needed by Algorithm's process to publish their results to other processes.

AC19 : Subscription and extraction of data from ROS Topics to read data needed by the various Operations processes e.g. if an Operation State specified in the SRS needs a different sensor reading then one module should be modified to provide the sensor reading.

AC20 : The ROS Topic Publisher's needed by Operation's modules to command/instruct the drone.

AC21 : Calling of ROS Services by the Operations modules, as well as processing the responses returned from the services i.e. there are a plethora of ROS Services that do different things and that have different conventions with respect to calling them.

AC22 : Visual perception algorithm used to yield insights from the raw parking lot images.

AC23 : Environment mapping algorithm used to create an occupancy map of the parking lot.

AC24 : Choosing the next location for the drone to move to.

AC25 : Path planning algorithm that suggests the optimal path for further exploration of the parking lot.

AC26 : Routines for managing the execution and data exchange between the sub-algorithms.

AC27 : The starting point of execution (main function) for the process running the algorithm modules.

AC28 : Routines that implement the state machine specified in the SRS, i.e. the states and transitions in the SRS may change.

AC29 : Presentation of data/information for the operation state routines to make decisions.

AC30 : Presentation of services needed by the operation state routines to command/instruct the drone, e.g. a new operation state may require a different type of command/instruction to implement its required behavior.

AC31 : Management of operation state, i.e. storing the current operation state and calling its routines appropriately.

AC32 : The starting point of execution (main function) for the process running the operations.

4.2 Unlikely Changes

UC1 : Flight controller hardware its associated sensors: The decision to use a flight controller was quite straight forward, as it saves a lot of development time. It offers a multitude of hardware interfacing and control algorithm services, meaning the team will not have to develop such low-level moduls. It makes the project much more feasible. Flight controller is an expensive purchase, thus once this decision is made it becomes even more expensive to change the hardware.

UC2 : Firmware (Ardupilot): Although there are other firmware options with the chosen flight controller, Ardupilot comes associated with the purchased flight controller. This is also the most popular and dominant firmware used in most drone projects.

UC3 : Edge Compute device: The Raspbery PI is the officially supported compute device for the chosen Flight Controller.

UC4 : Edge Compute Power: The drone's frame and size limitations make adding another compute device unfeasible. Thus the drone's compute power is limited to that of a Raspberry Pi PI.

UC5 : Canadian Laws regarding Drones: Laws regarding the drone's maximum frame size limit the choices of frame size and mechanical design.

UC6 : Operating System (OS) of the User Interface: The OS was specified to be Windows in the SRS. Windows is also the most common OS used in a professional setting.

5 Module Hierarchy

This section provides an overview of the module design. Modules are summarized in a hierarchy decomposed by secrets in Table 1. The modules listed below, which are leaves in the hierarchy tree, are the modules that will actually be implemented.

5.1 Launched Modules

Among the many modules, there are five modules that are run as standalone processes. Modules run on the drone's hardware include Main DDC Module, Main Algorithm Module, ROS-Mavlink Communication Driver, and ROS (also known as the master node). On the Operator's PC, the Main Interface Module is run.

Level 1	**Level 2**	**Level 3**
Hardware Hiding	Ardupilot	
	MavROS	
Communication Hiding	Drone-PC Communication Hiding	Base Socket
		Message Socket
		GStreamer
		Drone Camera
		Operator Camera
	Drone Inter-Process Communication Hiding	ROS
		Algorithm Topic Interface
		DDC Topic Interface
		DDC Service Interface
Interface Hiding	User Interface	
	Main Interface Module	
Algorithm Hiding	Vision App	
	Mapping App	
	Path Planning App	
	Algorithm Manager App	
	Main Algorithm Module	
Drone Decision and Control (DDC) Hiding	Operation States	
	Operations Manager	
	Main DDC Module	

Table 1: Module Hierarchy

6 Connection Between Requirements and Design

The design of the system is intended to satisfy the requirements developed in the SRS. In this stage, the system is decomposed into modules. The connection between requirements and modules is listed in Table 2.

7 Module Decomposition

Modules are decomposed according to the principle of "information hiding" proposed by **?**. The *Secrets* field in a module decomposition is a brief statement of the design decision hidden by the module. The *Services* field specifies *what* the module will do without documenting *how* to do it. For each module, a suggestion for the implementing software is given under the *Implemented By* title. If the entry is *OS*, this means that the module is provided by the operating system or by standard programming language libraries. *Mechatronics Engineering* means the module will be implemented by the Mechatronics Engineering software.

Only the leaf modules in the hierarchy have to be implemented. If a dash (–) is shown, this means that the module is not a leaf and will not have to be implemented.

7.1 Hardware Hiding Modules

Secrets: The secret of this module is the interfacing with hardware in a manner to present the software modules with an easy to use interface for reading sensor data, processing sensor data, and sending motion commands.

Services: Presents sensor data as ROS Topics. Presents a high-level motion command interface through ROS Topics and ROS Services.

Implemented By: -

7.1.1 Firmware Module

Secrets: The secret of this module is interfacing the drone's hardware with the MAVLINK Interface.

Services: Shares sensor data with the MAVLINK interface. Makes the drone follow high-level motion commands given by the MAVLINK interface, by utilizing control algorithms to instruct the actuator hardware. Examples of high-level commands include hovering at a given height, moving to a certain location, landing at a location, etc.

Implemented By: Ardupilot, an open source unmanned vehicle Autopilot Software Suite, capable of controlling autonomous

Type of Module: Library, simply interfaces between two layers and has no state information

7.1.2 ROS-Mavlink Communication Driver

Secrets: The secret of this module is the presentation of the MAVLINK interface into a ROS interface, such that other software modules can use ROS constructs to interface with MAVLINK, and thus the hardware.

Services: Creates ROS Topics that can be used to access sensor data. Offers ROS Services that can be used to make the drone obey high-level commands (such as hovering at a given height, moving to a certain location, etc.). Lastly, other modules can publish certain topics for sending certain types of motion commands for the drone to follow (for example if a module publishes location targets to a specific movement topic, then the hardware module will move the drone to the location specified in the topic).

Implemented By: MavROS, an open-source communication driver for various autopilots with MAVLink communication protocol. It is itself a ROS Node that can be run as a process.

Type of Module: Library, simply interfaces between two layers and has no state information

7.2 Communication Hiding

Secrets: The secret of this module is the sharing of data between modules.

Services: Mechanisms for communication between processes running on the drone, as well as between processes running on different platforms (Drone hardware and Operator's PC).

Implemented By: –

7.2.1 Drone-PC Communication Hiding Module

Secrets: The secret of this module is the communication mechanism to send and receive data between modules running on the drone and the Operator's PC.

Services: APIs for sending and receiving string data between the Drone and the Operator's PC. APIs for sending images from the Drone to the Operator's PC.

Implemented By: -

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.2.1.1 Base Socket

Secrets: The secret of this module is the rudimentary LAN Socket used by higher-level modules for LAN communication.

Services: A class that can be used to create server and client socket objects. Objects of this class contain the fundamental procedures for communication:
connect(string, int): Connects to a socket given an IP address and a port.
send(string): Sends a string of data to all connected sockets. receive(): Blocks until a string is received from the opposite socket. reconnect(): Reconnects to a previously established connection.

Implemented By: OS

Type of Module: Abstract Data Type

7.2.1.2 Message Socket

Secrets: The secret of this module is the socket used for communicating string data between two different platforms over a LAN network.

Services: A class to create objects that can send and receive string data. The objects should have an abstract and usable interface so that modules can use the procedures without knowing the communication mechanisms.

Implemented By: MessageSocket.py

Type of Module: Abstract Data Type

7.2.1.3 Video Streamer

Secrets: The secret of this module is the creation and maintenance of complex video streaming pipelines.

Services: Offers a class that can be used to create complex video streaming pipelines. Users of this module can set up pipelines with complex operations, like compression and decompression. Users can also specify the protocol, IP, and port over which the images will be published.

Implemented By: GStreamer, an open-source framework for streaming media.

Type of Module: Abstract Data Type

7.2.1.4 Drone Camera

Secrets: The secret of this module is the reading of images from the camera, and the communication pipeline for sending images from the drone to the Operator Camera module.

Services: Offers methods to read images from the camera and send images to the Operator Camera module.

Implemented By: DroneCamera.py

Type of Module: Abstract Object

7.2.1.5 Operator Camera

Secrets: The secret of this module is the receiving of images from the Drone Camera.

Services: Presents the modules on the Operator's PC to receive the latest image from the drone.

Implemented By: OperatorCamera.py

Type of Module: [Record, Library, Abstract Object, or Abstract Data Type]

7.2.2 Drone Inter-Process Communication Hiding Module

Secrets: The secret of this module is the communication mechanism between processes running on the drone.

Services: Various ROS Topics and ROS Services for inter-process communication.

Implemented By: –

7.2.2.1 ROS

Secrets: The secret of this module is the message-passing package used for inter-process communication on the drone.

Services: Offers two main components that can be used for communication:
Topics: named buses over which nodes exchange streams of data. Topic Publishers (information producers) are decoupled and unaware of "Topic Subscribers" (information consumers). "Topic Publishers" publish using a "publish" procedure.
ROS Service allows inter-node communication via a client/server system. Clients can make synchronous calls to servers by creating a "Service Client Object". When a Service Client sends a request, using the "call" method, a "Service Response" is returned. A field in the Service Response, called "success", it indicates if the service was successfully returned.

Implemented By: ROS, an open-source robotics middleware suite. It is run in a background process on the Raspberry PI.

Type of Module: Abstract Object

7.2.2.2 Algorithm Topic Interface

Secrets: The secret of this module is the sending and receiving of data to other processes, specifically for usage by the algorithm modules.

Services: Presents simple getters so that algorithm modules can access the data they need, as well as stores Topic Publishers that the publish algorithms use to share their results.

Implemented By: AlgorithmTopicInterface.py

Type of Module: Abstract Object

7.2.2.3 DDC Topic Interface

Secrets: The secret of this module is the sending and receiving of data to other processes, specifically for usage by the Operations modules.

Services: Presents simple getters so that algorithm modules can access the data they need, as well as stores Topic Publishers that the publish Operation States can use to share their results.

Implemented By: DDCTopicInterface.py

Type of Module: Abstract Object

7.2.2.4 DDC Service Interface

Secrets: The secret of this module is the ROS Services used by the operations modules to send instructions.

Services: Easy to use APIs for Operation Modules to call services, without having to know the service name, the data structure for making requests, or the data structure returned from the service call.

Implemented By: DDCServiceInterface.py

Type of Module: Abstract Object

7.3 Interface Hiding Module

Secrets: The secret of this module is the interfaces used to interact with the user.

Services: Interacting with the user to gather instructions and convey parking lot information, live images, and Drone status.

Implemented By: –

7.3.1 User Interface

Secrets: The secret of this module is the widgets and their layout used to interact with the user.

Services: Interacting with the user to gather instructions into a format that can be interpreted by the other modules. Also conveys feedback to the user, regarding parking lot information, live images, and Drone status.

Implemented By: UserInterface.py

Type of Module: Abstract Object

7.3.2 Main Interface Module

Secrets: The secret of this module is the execution of the process used to interact with the user.

Services: Offers a starting point of execution (main function) for the process running on the Operator's PC Application.

Implemented By: MainInterface.py

Type of Module: Abstract Object

7.4 Algorithm Hiding Module

Secrets: The secret of this module is the algorithms used on the drone to implement its autonomous features.

Services: Executes and shares the results of the vision algorithm, the mapping algorithm, and the path planning algorithm.

Implemented By: –

7.4.1 Vision App

Secrets: The secret of this module is the visual perception algorithms used to yield insights from drone images.

Services: Sends annotated and processed images to the Drone Camera module. Insights and analysis of from the images are shared and used by other modules, such as the Mapper App.

Implemented By: VisionApp.py

Type of Module: Abstract Object

7.4.2 Mapper App

Secrets: The secret of this module is the algorithm for creating the occupancy map, using all images of the parking lot seen at present and in the past.

Services: Creates an estimate of the occupancy map of the parking lot, shared and used by other modules such as the Path Plan App.

Implemented By: MapperApp.py

Type of Module: Abstract Object

7.4.3 Path Plan App

Secrets: The secret of this module is the algorithm for determining a suggested path for future exploration. Another secret is the logic for choosing the next location the drone should move toward.

Services: Publishes the next location the drone should move to, based on the current state of the drone and the optimal path for future exploration.

Implemented By: PathPlanApp.py

Type of Module: Abstract Object

7.4.4 Algorithm Manager App

Secrets: The secret of this module is the coordination of execution between the sub-algorithms (vision, mapping, and path planning).

Services: Offers a simple to use interface to execute all of the sub-algorithms correctly (Vision App, Mapper App, Path Plan App).

Implemented By: AlgorithmManager.py

Type of Module: Abstract Object

7.4.5 Main Algorithm Module

Secrets: The secret of this module is the execution of the process used to run the algorithms.

Services: Offers a starting point of execution (main function) for running the various types of algorithms.

Implemented By: MainAlgorithm.py

Type of Module: Abstract Object

7.5 Drone Decision and Control (DDC) Hiding

Secrets: The secret of this module is the decision making of the drone. Given access to data/information modules and control modules, the DDC Modules acts as the central command center that links the two components. It decides tells the drone what to do based on what the drone has observed thus far.

Services: Calls services in the control modules to make the drone obey SRS specified behaviour based on data it received from the data it received from topics and the user.

Implemented By: –

7.5.1 Operation States

Secrets: The secret of this module is the implementation of the operation states specified in the SRS.

Services: Offers routines that implement each of the operation states specified in the SRS.

Implemented By: OperationStates.py

Type of Module:

7.5.2 Operations Manager

Secrets: The secret of this module is the management of the Operation States.

Services: Storage of all modules, data, and services needed by the operation states to implement their required behavior. Furthermore, this module stores the current operation state, shares its fields with the operation state, and executes its routines.

Implemented By: OperationManager.py

Type of Module: Abstract Object

7.5.3 Main DDC Module

Secrets:

Services: Offers a starting point of execution (main function) for the process responsible for drone decision and control.

Implemented By: MainFSM.py

Type of Module: Abstract Object

8 Traceability Matrix

This section shows two traceability matrices: between the modules and the requirements and between the modules and the anticipated changes.

Req.	Modules
GEN_001	Vision App
GEN_002	Vision App, Mapper App, Operations Manager, Message Socket, User Interface
GEN_003	Operation States
GEN_004	Operation States
GEN_005	Vision App
GEN_006	Vision App
STA_000	Operation States
STA_001	Operation States, DDC Service Interface
STA_002	Operation States, Vision App, Path Plan App, Path Plan App
STA_003	Operation States, Vision App, , Path Plan App, Mapper App
STA_004	Operation States
STA_005	Operation States
STA_006	Operation States, DDC Service Interface
STA_007	Operation States, Path Plan App, User Interface
STA_008	Operation States, Path Plan App, User Interface
STA_009, User Interface	Operation States, DDC Service Interface
STA_010	Operation States, User Interface, Message Socket, Path Plan App
STA_011	Operation States, Path Plan App
STA_012	Operation States, DDC Service Interface
STA_013	Operation States, DDC Service Interface

Table 2: Trace Between Requirements and Modules

Req.	Modules
TRANS_001	Operation States
TRANS_002	Operation States
TRANS_003	Operation States, User Interface
TRANS_004	Operation States, Vision App
TRANS_005	Operation States, User Interface
TRANS_006	Operation States, Vision App, Mapper App, Path Plan App
TRANS_007	Operation States, User Interface, Vision App
TRANS_008	Operation States, User Interface, Vision App
TRANS_009	Operation States, User Interface
TRANS_010	Operation States, Message Socket
TRANS_011	Operation States, Message Socket
TRANS_012	Operation States, User Interface
TRANS_013	Operation States, User Interface
TRANS_014	Operation States, DDC Topic Interface
TRANS_015	Operation States
PERF_001	Vision App, Mapper App, Path Plan App
PERF_002	Firmware Module
PERF_003	Firmware Module
PERF_004	Message Socket, Operations Manager, User Interface
PERF_005	Firmware Module
PERF_006	Firmware Module
PERF_007	
PERF_008	Firmware Module
DES_001	
STD_001	
STD_002	Drone Camera, Operator Camera, Message Socket
SEC_001	User Interface
SEC_002	User Interface

Table 3: Trace Between Requirements and Modules

Req.	Modules
MTNC_001	
MTNC_002	Operation States, Firmware Module
MTNC_003	
SAFE_001	Operation States, Firmware Module
SAFE_002	User Interface
SAFE_003	Operation States
SAFE_004	Operation States
SAFE_005	
USE_001	User Interface
USE_002	User Interface
USE_003	
USE_004	
USE_005	User Interface
SR_002	User Interface
SR_003	Operations Manager, User Interface
SR_004	
SR_005	
SR_006	
SR_007	User Interface, Operation States
SR_008	
SR_009	User Interface, Vision App
SR_010	
SR_011	Operation States
SR_012	Operation States, User Interface
SR_013	User Interface

Table 4: Trace Between Requirements and Modules

AC	Modules
AC1 :	Firmware Module
AC2 :	Firmware Module
AC3 :	Firmware Module
AC4 :	ROS-Mavlink Communication Driver
AC5 :	ROS-Mavlink Communication Driver
AC6 :	Base Socket
AC7 :	Message Socket
AC8 :	Video Streamer
AC9 :	Drone Camera
AC10 :	Drone Camera
AC11 :	Operator Camera
AC12 :	User Interface
AC13 :	User Interface
AC14 :	User Interface
AC15 :	Main Interface Module
AC16 :	ROS
AC17 :	Algorithm Topic Interface
AC18 :	Algorithm Topic Interface
AC19 :	DDC Topic Interface
AC20 :	DDC Topic Interface
AC21 :	DDC Service Interface
AC22 :	Vision App
AC23 :	Mapper App
AC24 :	Path Plan App
AC25 :	Path Plan App
AC26 :	Algorithm Manager App
AC27 :	Main Algorithm Module
AC28 :	Operation States
AC29 :	Operations Manager
AC30 :	Operations Manager
AC31 :	Operations Manager
AC32 :	Main DDC Module

Table 5: Trace Between Anticipated Changes and Modules

9 Use Hierarchy Between Modules

In this section, the uses hierarchy between modules is provided. **?** said of two programs A and B that A *uses* B if correct execution of B may be necessary for A to complete the task described in its specification. That is, A *uses* B if there exist situations in which the correct functioning of A depends upon the availability of a correct implementation of B. Figure 1 illustrates the use relation between the modules. It can be seen that the graph is a directed acyclic graph (DAG). Each level of the hierarchy offers a testable and usable subset of the system, and modules in the higher level of the hierarchy are essentially simpler because they use modules from the lower levels.

The modules with a white background are leaf modules (which are implemented). Arrows in red indicate use relationships between leaf modules.

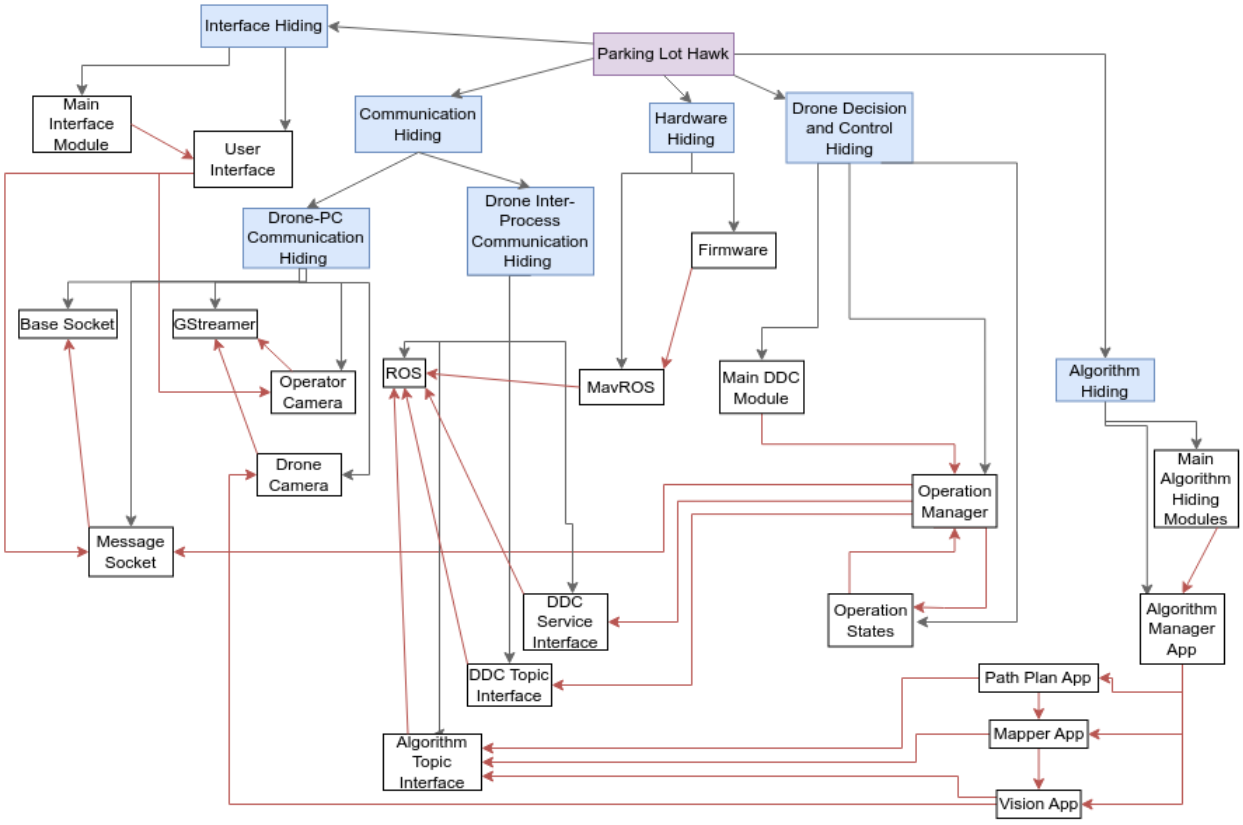

Figure 1: Use hierarchy among modules