Module Interface Specification for Mechatronics Engineering

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

February 24, 2023

1 Revision History

Date	Version	Notes
January 18, 2023	1.0	Initial Revision

2 Symbols, 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

3 Introduction

The following document details the Module Interface Specifications for the Parking Lot Hawk project. Parking Lot Hawk allows a Parking Lot Manager (Operator) to gather information about their parking lot. The user instructs a drone that offers various autonomous modes to help the Operator understand their parking lot. The drone gives two primary pieces of information, live camera images of what the drone currently sees, as well as an occupancy map based on what the drone has observed in the past.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at Parking Lot Hawk

4 Notation

Data Type	**Notation**	**Description**
character	char	a single symbol or digit
integer	$\mathbb{Z}$	a number without a fractional component in (-∞, ∞)
natural number	$\mathbb{N}$	a number without a fractional component in [1, ∞)
real	$\mathbb{R}$	any number in (-∞, ∞)
Image Array	Image Array	2D Array to hold the pixels of images.
Sockets	Sockets	Software structure within a network that serves as an endpoint for sending and receiving messages across the network.
Occupancy Map	Occupancy Map	2D Map that describes parking lot area, non-parking lot area, and unoccupied parking lots.
User Error	User Error	Enum: 0 - None, 1 - Desired_Location_Out_Of_Bounds, 2 - No_Lot_Detected
Health Status	Health Status	Enum: 0 - Healthy, 1 - Unhealthy
Dictionary	dict	-
Local Pose	Local Pose	3D position and orientation of an object, XY coordinates are relative to the arm location.
Global Pose	Global Pose	3D position and orientation of an object, coordinates are latitude and longitude.

Data Type	Notation	Description
ROS Node	ROS Node	Synonym for a process with ROS functionality (registered with the ROS master node).
ROS Master Node	ROS Master Node	ROS Node that runs in the background. All ROS nodes must register with this node.
ROS Topic	ROS Topic	Used to stream continuous data between two ROS Nodes.
ROS Service	ROS Service	Type of inter-process communication where "server nodes" can advertise certain services and "client nodes" can send synchronous service requests.
Ardupilot Mode	Ardupilot Mode	Ardupilot defined enum. Ardupilot interfaces with the hardware (actuators and sensors), and offer several flight modes such as: RTL (Return to Launch), LAND, and GUIDED mode which is used to fly to a specific location.
State	State	A MavROS defined struct. Contains several key diagnostic signals from the flight controller, including whether the drone is armed, the Ardupilot Mode, etc.
BatteryInfo	BatteryInfo	A MavROS defined struct. Contains several key diagnostic signals about the battery, such as percentage of capacity left, voltage, etc.
Diagnostics	BatteryInfo	A MavROS defined struct. Contains several summary diagnostic signals about the hardware.

The specification of Mechatronics Engineering uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, Mechatronics Engineering uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

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	Algorithm Topic Interface
		ROS
		DDC Topic Interface
		DDC Service Interface
Interface Hiding	User Interface	
	Main Interface Module	
Algorithm Hiding	Vision App	
	Mapper 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 MIS of Message Socket

6.1 Module

The secret of this module is the socket used for communicating string data between two different platforms over a LAN network, in order to present a user-friendly interface for other communication modules.

6.2 Uses

Module	Imported Types, Constants, Access Programs	Description
Base Socket	BaseSocket	BaseSocket is the underlying low-level module used for communication, Message Socket wraps this class to be more user-friendly.

6.3 Syntax

6.3.1 Exported Types

MessageSocket = ?

6.3.2 Exported Access Programs

Name	In	Out	Exceptions
new MessageSocket	inType:string, inHOST: string, inPort: int ($\mathbb{N}$)	-	ConnectionTimeout
init	-	-	ConnectionTimeout
sendMessageSync	x : string	-	FailedToSend
sendMessageAsync	x : string	-	-
getMessage	-	string	NoMessagesError
isConnected	-	bool	-
close	-	-	

6.4 Semantics

6.4.1 State Variables

Name	Type	Description
conn	BaseSocket	This variable is the secret of the module, it is not accessible to users of this module.
type	string ($\mathbb{N}$)	Type of socket (Server/Drone or Client/Operator).
PORT	int ($\mathbb{N}$)	Port number used for communication.
HOST	string	IP address. This is empty for a server connection, while for a client connection, it contains the server's IP address.
connected	bool	Indicates if a connection is established
receivedMessages	Sequence of string	Serves as a queue for received messages.
messagesToSend	Sequence of string	Serves as a queue for messages that need to be sent to the opposite socket.

6.4.2 Environment Variables

6.4.3 Assumptions

The init() routine is called before any other access programs.

6.4.4 Access Routine Semantics

new MessageSocket(inType, inHOST, inPORT):

- transition: type, HOST, PORT := inType, inHOST, inPORT
- out: out := self
- description: Constructor

init():

- transition: conn := new BaseSocket.BaseSocket();
  connected := conn.connect(HOST, PORT);
  connected = TRUE $\Rightarrow$ { receivedMessages := <> ;
  messagesToSend := <> ;
  startReadThread();
  startWriteThread(); }
- exception: {$\neg$ connected $\Rightarrow$ ConnectionTimeout}

- description: Initializes the connection with the Operator's PC Application. Returns an exception if a timeout occurred while connecting.

sendMessageSync(x):

- transition: {connected = FALSE ⇒ reconnect() }; temp := conn.send(x)}
- exception: {connected = FALSE | temp = FALSE ⇒ FailedToSend}
- description: Sends a synchronous string message to the socket, i.e. program will block until data is sent. This function will try to reconnect if a connection is not established. If the reconnect failed, then an exception is returned.

sendMessageAsync(x):

- transition: messagesToSend := messagesToSend || <x>
- description: Sends an asynchronous string of data to the Operator Application's Socket. Non-blocking function call, so data will be sent in the background.

getMessage():

- transition: receivedMessages := receivedMessages[1..|receivedMessages| −1]
- output: out := receivedMessages[0]
- exception: exc = {|receivedMessages| = 0 ⇒ NoMessagesError}
- description: Returns a string of data from the Operator Application. Returns empty string if there are no messages. If multiple messages have been sent since the last getMessage call, it returns the earliest message (through the use of the queue). The routine is quite similar to pop() call on a queue.

isConnected():

- output: out := connected
- description: Returns if a connection is established.

close():

- transition: conn.close()
- description: Closes the socket connection.

6.4.5 Local Functions

reconnect():

- transition: conn.connect(HOST, PORT);
  connected = true;
- description: Reconnects a previously established socket connection. Also utilized by the background threads.

startReadThread():

- output: -
- transition: This starts a background thread that constantly reads messages in the messagesToSend queue and sends them to the receiving module. It is up to the developer to design and implement this internal routine.

startWriteThread():

- output: -
- transition: This starts a background thread that constantly sends messages in the messagesReceived queue. Its operation is quite complicated, as strings received from the socket may contain multiple messages or may contain only a partial message. It is up to the developer to design and implement this internal routine.

7 MIS of Drone Camera

7.1 Module

The secret of this module is accessing the Drone's camera images for modules running on the drone.

7.2 Uses

Module	Imported Types, Constants, Routines	Description
Video Streamer	VideoStreamer	A VideoStreamer application is run on the Drone, where it reads images from the camera and shares the images over an IP/port for other clients to read. Through the OpenCV library, images from VideoStreamer can be accessed through Python.

7.3 Syntax

7.3.1 Exported Constants

7.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getImage	-	Image Array	NoImage

7.4 Semantics

7.4.1 State Variables

Name	Type	Description
image	Image Array	Contains the latest image captured from the camera.
IP	string	Contains the IP address used to create the video stream.
port	$\mathbb{N}$	Contains the port used to create the video streams.

7.4.2 Environment Variables

7.4.3 Assumptions

The init() routine is called before any other access programs.

7.4.4 Access Routine Semantics

init():

- transition: startReadThread();
- output: -
- description: Starts the background thread to read images from the camera.

getImage():

- transition: -
- output: out := image
- exception: $\{$image = NULL $\Rightarrow$ NoImage)$\}$
- description: Returns the latest image captured from the camera. If for some failure the images cannot be read from the hardware, a NoImage error is returned.

7.4.5 Local Functions

startReadThread():

- description: This routine starts a background thread for updating the "image" array with the latest camera image. As this routine is internal and specific to the camera hardware/interface, it is left to the developers to design and implement.

8 MIS of Operator Camera

8.1 Module

The secret of this module is the communication means used to obtain images from the drone.

8.2 Uses

Module	Imported Types, Constants, Routines	Description
Video Streamer	VideoStreamClient	This datatype creates a video stream client object that is used to receive images from the video stream created by the drone. The datatype has a receiveImage method, which performs the appropriate decompression before returning an image from the video streamer.

8.3 Syntax

8.3.1 Exported Constants

8.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getImage	-	Image Array	NoImage

8.4 Semantics

8.4.1 State Variables

Name	Type	Description
image	Image Array	Contains the latest image captured from the camera.
IP	strings	Contains the IP address to the video stream.
port	int ($\mathbb{N}$)	Contains the port to the video streamer.
videoStreamClient	VideoStreamClient	This variable is used to read images from a video stream created externally by the Drone Camera.

8.4.2 Environment Variables

8.4.3 Assumptions

The init() routine is called before any other access programs.

8.4.4 Access Routine Semantics

init():

- transition:
  startReadThread();
- output: -
- description: Starts the background thread to read images from the VideoStreamer.

getImage():

- output: out := image
- exception: $\{\text{image} = \text{NULL} \Rightarrow \text{NoImage}\}$
- description: Returns the latest image captured from the stream.

8.4.5 Local Functions

startReadThread():

- description: This routine starts a background thread for updating the "image" array with the latest image from the video stream created by the drone. As this routine is internal and specific to the GStreamer API, it is left to the developers to design and implement.

9 MIS of Algorithm Topic Interface

9.1 Module

The secret of this module is reading data from various topics created by other processes, and presenting the data in easy-to-use "get" routines. Furthermore, it contains ROS topics for sending information to other processes running on the drone. Unlike the other Topic Interface Modules, this module is designed specifically for usage by the Algorithms Modules (Vision App, Mapper App,Path Plan App).

9.2 Uses

Module	Imported Types, Constants, Routines	Description
ROS	Topic Subscriber, Topic Publisher	-

9.3 Syntax

9.3.1 Exported Constants

Name	Type	Purpose
parkLotDetectedPub	ROS Topic Publisher for publishing booleans.	Indicates if a parking lot is currently visible at the drone's current position. Meant for usage by the Operations Manager.
desLocInboundPub	ROS Topic Publisher for publishing booleans.	Indicates if the location that the Operations Manager App is intends the drone to move to is within the parking lot. Meant for usage by the Operations Manager.
occupancyMapPub	ROS Topic Publisher for publishing occupancy maps.	Contains the latest occupancy map, created from observations ever since the drone launched. Meant for usage by the Operations Manager.
localPosPub	ROS topic Publisher for publishing local poses.	Contains the next position the drone should move to. This topic is consumed by the MavROS process, which in turn relays this instruction to the flight controller.
visionAppHealthPub	Health Status	Health status of the Vision App module.
mapperAppHealthPub	Health Status	Health status of the Mapper App module.
pathPlanAppHealthPub	Health Status	Health status of the Path Plan App module.

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
getDroneState	-	string	-
getDesiredPose	-	Local Pose	-
getLocalPose	-	Local Pose	-

9.4 Semantics

9.4.1 State Variables

Name	Type	Description
droneState	string	The latest estimate of the current Operation State (Operation States) the drone is currently operating in. Received from the Operations Manager.
desiredPose	Local Pose	The latest estimate of the location the user desires the drone to move to. Received from the Operations Manager.
localPose	Local Pose	The current pose of the drone, with respect to the location the drone was armed.

9.4.2 Environment Variables

9.4.3 Assumptions

The init() routine is called before any other access programs.

9.4.4 Access Routine Semantics

init():

- transition: initSubscribers()
- description: Initializes the Topic Subscribers used to read ROS Topic data from other ROS Nodes running on the drone.

getDroneState():

- output: out := droneState

getDesiredPose():

- output: out := desiredPose

getLocalPose():

- output: out := localPose

9.4.5 Local Functions

initSubscribers():

- description: Initializes the mechanisms to subscribe to the relevant ROS Topics. Freedom is given to the developer to implement the subscription mechanism, but ultimately the state variables (droneState, desiredPose, and localPose) should be regularly updated to contain the latest estimates of each of their respective signals.

10 MIS of Vision App

10.1 Module

The secret of this module is the image processing and analysis algorithms that are used to yield insights.

10.2 Uses

Module	Imported Types, Constants, Routines	Description
Drone Camera	Drone Camera	This module runs on the Drone, thus it receives and sends annotated images using the Drone Camera.
Algorithm Topic Interface	Algorithm Topic Interface	-

10.3 Syntax

10.3.1 Exported Constants

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	inDC: Drone Camera, inTopicInterface: Algorithm Topic Interface	-	-
process	-	-	NoImage, AlgoError
publish	-	-	-
getParkLotDetected	-	bool	-
getHealth	-	Health Status	-

10.4 Semantics

10.4.1 State Variables

Name	Type	Description
droneCamera	Drone Camera	Contains a reference to the Drone Camera object.
topicInterface	Algorithm Topic Interface	Contains a reference to the Algorithm Topic Interface object.
parkLotDetected	bool	Indicates if the drone currently sees a parking lot, used as a transition in the state machine as per SRS (TRANS_007,TRANS_008).
annotatedImage	Image Array	Contains the image for analysis and segmentation.
health	Health Status	Health of module, whether results can be trusted.

10.4.2 Environment Variables

10.4.3 Assumptions

The Drone Camera and Algorithm Topic Interface objects are already created. The init() routine is called before any other access programs.

10.4.4 Access Routine Semantics

init(inDC, inTopicInterface):

- transition: droneCamera := inDC;
  topicInterface := topicInterface;
  health := Healthy;
- description: Initializes all Vision App member fields, such as droneCamera. Algorithm specific data structures are a secret of this module and are not shown in the MIS.

process():

- transition: annotatedImage := droneCamera.getImage();
  res = runImageAlgorithms(topicInterface);
  sendUpdate(topicInterface);

- exception: $\{\text{annotatedImage} = \text{NoImage} \rightarrow \text{NoImage}, \text{res} = \text{AlgoError} \Rightarrow \text{AlgoError}\}$

- description: Called every frame, this function runs the vision algorithm on the latest raw images from the Drone Camera module and information from the Topic Interface. Then it updates the relevent topics and modules.

getParkLotDetected():

- output: out := parkLotDetected

getHealth():

- output: out := health

publish(topicInterface):

- transition: topicInterface.parkLotDetectedPub.publish(getParkLotDetected()); topicInterface.visionAppHealthPub.publish(getHealth())

- description: Shares the algorithm results on the relevant topics.

10.4.5 Local Functions

runImageAlgorithms(topicInterface):

- exception: If the algorithm fails to run for any issues, it returns an AlgoError exception.

- description: Runs the image processing algorithm and stores the resulting image in annotatedImage. Updates the parkLotDetected variable.

11 MIS of Mapper App

11.1 Module

The secret of this module is the algorithm creating an occupancy map of the parking lot.

11.2 Uses

Module	Imported Types, Constants, Routines	Description
Algorithm Topic Interface	Algorithm Topic Interface	-
Vision App	getParkLotDetected	-

11.3 Syntax

11.3.1 Exported Constants

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	inVisionApp: Vision App, inTopicInterface: Algorithm Topic Interface	-	-
process	-	-	AlgoError
publish	-	-	-
getOccupancyMap	-	Occupancy Map	-
getHealth	-	Health Status	-

11.4 Semantics

11.4.1 State Variables

Name	Type	Description
visionApp	Vision App	Contains a reference to the Vision App object.
topicInterface	Algorithm Topic Interface	Contains a reference to the Algorithm Topic Interface object.
occupancyMap	occupancyMap	Contains the latest estimate of the parking lot's occupancy.
health	Health Status	Health of module, whether results can be trusted.

11.4.2 Environment Variables

11.4.3 Assumptions

The init() routine is called before any other access programs.

11.4.4 Access Routine Semantics

init(inVisionApp, inTopicInterface):

- transition: health := Healthy;
  visionApp := inVisionApp;
  topicInterface:= inTopicInterface;
  occupancyMap := OccupancyMap();
  Empty Occupancy Map
- description: Initializes the constructs needed by the Mapping Algorithm.

process():

- transition: res := runMappingAlgorithms(visionApp, topicInterface);
  sendUpdate(topicInterface);
- exception: $\{$res = AlgoError $\Rightarrow$ AlgoError $\}$
- description: Called every frame, this function runs the mapping algorithm on the latest raw information from the Vision App and Topic Interface. Then it updates the topics with the latest mapping results.

getOccupancyMap():

- output: out := occupancyMap

publish(topicInterface):

- transition: topicInterface.parkOccupancyMapPub.publish(getOccupancyMap()); topicInterface.mapperAppHealthPub.publish(getHealth())
- description: Updates the relevant topics and modules with the algorithm results.

11.4.5 Local Functions

getHealth():

- output: out := health

runMappingAlgorithms(visionApp, topicInterface):

- exception: If the algorithm fails to run for any issues, it returns an AlgoError exception.
- description: Runs the mapping algorithm, storing the results in occupancyMap. The algorithm will use the visionApp.getParkLotDetected() routine.

12 MIS of Path Plan App

12.1 Module

The secret of this module is the path-planning decisions of the drone.

12.2 Uses

Module	Imported Types, Constants, Routines	Description
Algorithm Topic Interface	Algorithm Topic Interface	-
Mapper App	MappingApp	-

12.3 Syntax

12.3.1 Exported Constants

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	inMappingApp: Mapper App, inTopicInterface: Algorithm Topic Interface	-	-
process	-	-	AlgoError
publish	-	-	-
getLocalPose	-	Local Pose	-
getDesPoseInbound	-	bool	-
getHealth	-	Health Status	-

12.4 Semantics

12.4.1 State Variables

Name	Type	Description
mappingApp	Mapper App	Contains a reference to the Mapper App object.
topicInterface	Algorithm Topic Interface	Contains a reference to the Algorithm Topic Interface object.
desPoseInbound	bool	This boolean indicates if the pose that the user requests the drone to move toward is valid (i.e within the parking lot).
localPose	Local Pose	This is the pose that the drone is actually instructed to move toward.
health	Health Status	Health of module, whether results can be trusted.

12.4.2 Environment Variables

12.4.3 Assumptions

The init() routine is called before any other access programs.

12.4.4 Access Routine Semantics

init(inMappingApp, inTopicInterface):

- transition: health:= Healthy;
  mappingApp:= inMappingApp;
  topicInterface := inTopicInterface;
  desPoseInbound := TRUE;// localPose = new Pose();
- description: Initializes the constructs needed by the Path Plan App. Initialize desPoseInbound to TRUE because all poses are assumed to be within the bounds of the parking lot until proven otherwise.

process():

- transition: res := runPathPlanAlgorithms(mappingApp, topicInterface);
  sendUpdate(topicInterface);
- exception: $\{res = AlgoError \Rightarrow AlgoError\}$
- description: Called every frame, this function runs the path planning algorithm on the latest raw information from the Mapper App and Topic Interface. Then it updates the topics with the latest algorithm results.

getOccupancyMap():

- output: out := occupancyMap

publish(topicInterface):

- transition: topicInterface.desLocInboundPub.publish(getDesLocInbound());
  topicInterface.localPosePub.publish(getLocalPose());
  topicInterface.pathPlanHealthPub.publish(getHealth());

- description: Updates the relevant topics and modules with the algorithm results.

12.4.5 Local Functions

runPathPlanAlgorithms(visionApp, topicInterface):

- exception: If the algorithm fails to run for any issues, it returns a AlgoError exception.
- description: Runs the path plan algorithm, storing the results in localPose and desPoseInbound. The algorithm will use the Occupancy Map (from mapping.getOccupancyMap()) to make suggest a path.

  Note that localPose is different from desPose. The former is the actual pose that the drone is instructed to move toward. The latter (available in the Algorithm Topic Interface) is the pose that the user requests the drone to move toward. They are the same in all flight states, except for the Autonomous Explore state, under which the drone moves toward the location that will yield the most amount of new parking lot information. Responsibility is given to the algorithm engineer to implement an exploration algorithm, as well as to obey the state requirements indicated in the SRS.

13 MIS of Algorithm Manager App

13.1 Module

The secret of this module is the execution order and data exchange between the modules Drone Camera, Vision App, Mapper App, and Path Plan App.

13.2 Uses

Module	Imported Types, Constants, Routines	Description
Algorithm Topic Interface	Algorithm Topic Interface	-
Vision App	Vision App	-
Mapper App	Mapper App	-
Path Plan App	Path Plan App	-
Drone Camera	Drone Camera	-

13.3 Syntax

13.3.1 Exported Constants

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	inVisionApp: Vision App, mappingApp: Vision App, pathPlanApp: Path Plan App, algoTopicInterface: Algorithm Topic Interface, inDroneCamera: Drone Camera	-	-
process	-	-	AlgoError

13.4 Semantics

13.4.1 State Variables

Name	Type	Description
algoTopicInterface	Algorithm Topic Interface	-
visionApp	Reference to the Vision App object.	-
mapperApp	Reference to the Mapper App.	-
pathPlanApp	Reference to Path Plan App object.	-

13.4.2 Environment Variables

13.4.3 Assumptions

The init() routine is called before any other access programs.

13.4.4 Access Routine Semantics

init(inAlgoTopicInterface, inVisionApp, inMapperApp, inPathPlanApp, inDroneCamera):

- transition: algoTopicInterface = inAlgoTopicInterface;
  visionApp = inVisionApp;
  mappingApp = inMapperApp;
  pathPlanApp = inPathPlanApp;
  droneCamera = inDroneCamera;
  algoTopicInterface.init();
  droneCamera.init()
  visionApp.init(inDroneCamera, algoTopicInterface);
  mappingApp.init(visionApp, algoTopicInterface);
  pathPlanApp.init(mappingApp, algoTopicInterface);

- description: Initializes all the Algorithms Applications.

process(visionApp, topicInterface):

- transition: res1 := visionApp.process();
  res2 := mappingApp.process();
  red3 := pathPlanApp.process();
  visionApp.publish();
  mappingApp.publish();
  pathPlanApp.publish();

- exception: { res1 = AlgoError | res2 = AlgoError | res3 = AlgoError ⇒ AlgoError }

- description: Called every frame, this function runs all of the algorithms.

13.4.5 Local Functions

14 MIS of Main Algorithm Module

14.1 Module

This module is one of four modules run as processes on the drone's hardware. This module runs the process responsible for running the algorithms. Most of the management of sub-algorithms are available by the routines in the Algorithm Manager App. Because this module is run as a standalone process, it creates abstract objects/modules as well.

The secret of this module is the execution and operation of the process running the algorithm.

14.2 Uses

Module	Imported Types, Constants, Routines	Description
Algorithm Manager App	Algorithm Manager App	-
Vision App	Vision App	-
Mapper App	Mapper App	-
Path Plan App	Path Plan App	-
Drone Camera	Drone Camera	-
Algorithm Topic Interface	Algorithm Topic Interface	-

14.3 Syntax

14.3.1 Exported Constants

14.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	Starting point for program execution.

14.4 Semantics

14.4.1 State Variables

Name	Type	Description
algorithmApp	Algorithm Manager App	-
visionApp	Vision App	-
mapperApp	Mapper App	-
pathPlanApp	Path Plan App	-
droneCamera	Drone Camera	-
topicInterface	Algorithm Topic Interface	-

14.4.2 Environment Variables

14.4.3 Assumptions

14.4.4 Access Routine Semantics

main():

- transition: droneCamera = new DroneCamera();
  visionApp = new VisionApp();
  mapperApp = new MapperApp();
  pathPlanApp = new PathPlanApp();
  topicInterface = new TopicInterface();
  algorithmApp = new AlgorithmApp(topicInterface, visionApp, mapperApp, pathPlanApp, droneCamera);
  algorithmApp.init()
  while (True) {algorithmApp.process();};

14.4.5 Local Functions

15 MIS of DDC Topic Interface

15.1 Module

The secret of this module is reading data from various topics published by other processes, and presenting the data in easy-to-use "get" routines for DDC Modules. Furthermore, it contains ROS Topics for sending DDC information to other processes running on the drone. Unlike the other Topic Interface Nodes, this module is designed specifically for usage by the Drone Decision and Control (DDC) Hiding (Operations Manager and Operation States).

15.2 Uses

Module	Imported Types, Constants, Routines	Description
ROS	Topic Subscriber, Topic Publisher	-

15.3 Syntax

15.3.1 Exported Constants

Name	Type	Purpose
desPosePub	ROS Topic Publisher for publishing Local Poses.	Contains the pose that the user desires the drone to move to. Meant for usage by the Algorithm Manager App.
currStatePub	ROS Topic Publisher for publishing strings.	Contains a unique string to indicate the current Operation State. Meant for usage by the Algorithm Manager App.

15.3.2 Exported Access Programs

Name	In	Out	Exceptions
getState	-	State	-
getGlobalPose	-	Global Pose	-
getDiagnostics	-	Diagnostics	-
getParkLotDetected	-	bool	-
getDesLocInbound	-	bool	-
getHealthStatus	-	Health	-
getLocalPose	-	Local Pose	-
getBatteryInfo	-	BatteryState	-
getRelAlt	-	float ($\mathbb{R}$)	-
getOccupancyMap	-	Occupancy Map	-
getVisionHealth	-	Health Status	-
getMapperHealth	-	Health Status	-
getPathPlanHealth	-	Health Status	-

15.4 Semantics

15.4.1 State Variables

Name	Type	Description
state	State	Contains several key diagnostics signals from the flight controller, such as if the drone is armed, if the flight controller is connected to MavROS, if the drone waiting for manual input, the current Ardupilot state of the drone, etc.
globalPose	Global Pose	Contains the current pose of the drone, in global format (i.e. contains latitude, longitude, etc.).
diagnostics	Diagnostics	Contains diagnostics about the hardware on the drone (sensors, motors, etc.).
parkLotDetected	bool	The latest estimate of whether or not the parking lot is currently detected, received from the Vision App.
desLocInbound	bool	The latest estimate of whether or not the user request location is valid (within the parking lot boundaries), received from the Path Plan App.
healthStatus	Health Status	Another type of diagnostic. However, this contains information about the overall health status of the drone (including firmware and hardware).
localPose	Local Pose	Contains the current pose of the drone (i.e. position relative to takeoff location, etc.).
batteryInfo	BatteryInfo	Contains the information about the current battery, such as capacity left, voltage, etc.
relAlt	float ($\mathbb{R}$)	Contains the relative altitude the drone is currently flying at, relative to the launch height.
occupancyMap	Occupancy Map	The latest estimate of the parking lot occupancy map received from the Mapper App.
visionHealth	Health status	The latest estimate of the Vision App's health received from the Vision App.
mapperHealth	Health status	The latest estimate of the Mapper App's health received from the Mapper App.
pathPlanHealth	Health status	The latest estimate of the Path Plan App's health received from the Path Plan App.

15.4.2 Environment Variables

15.4.3 Assumptions

The init() routine is called before any other access programs.

15.4.4 Access Routine Semantics

init():

- transition: initSubscribers()
- description: Initializes the Topic Subscribers used to receive data from other processes running on the drone.

getState():

- output: out := state

getGlobalPose():

- output: out := globalPose

getDiagnostics():

- output: out := diagnostics

getParkLotDetected():

- output: out := parkLotDetected

getDesLocInbound():

- output: out := desLocInbound

getHealthStatus():

- output: out := healthStatus

getBatteryInfo():

- output: out := batteryInfo

getRelAlt():

- output: out := relAlt

getOccupancyMap():

- output: out := occupancyMap

getVisionHealth():

- output: out := visionHealth

getMapperHealth():

- output: out := mapperHealth

getPathPlanHealth():

- output: out := pathPlanHealth

15.4.5 Local Functions

initSubscribers():

- description: Initializes the mechanisms to subscribe to the relevant ROS topics. Freedom is given to the developer to implement the subscription mechanism, but ultimately the state variables (e.g. state, globalPose, diagnostics, parkLotDetected) should be regularly updated to contain the latest estimates of each of their respective signals.

16 MIS of DDC Service Interface

16.1 Module

The secret of this module is the choice of ROS services used to send instructions to the flight controller, as well as simplifying the usage of the services.

16.2 Uses

Module	Imported Types, Constants, Routines	Description
ROS	Service Client, Service Response	-

16.3 Syntax

16.3.1 Exported Constants

16.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	-	-	-
setArm	bool	-	ReqFailed
setRtlAlt	int ($\mathbb{N}$)	-	ReqFailed
sendTakeoffCmd	Global Pose ($\mathbb{R}$)	-	ReqFailed
setMode	Ardupilot Mode	-	ReqFailed

16.4 Semantics

16.4.1 State Variables

Name	Type	Description
armingClient	ROS Service Client	Used for sending arm/disarm commands to the flight controller.
setModeClient	ROS Service Client	Used for sending flight mode commands in the flight controller's firmware. This Service Client is used to set the Ardupilot Mode used by the flight controller's firmware, which is different from the Operation States used by the Operations Manager.
takeoffClient	ROS Service Client	Used for sending takeoff commands.
paramSetClient	ROS Service Client	Used for setting the values of flight controller parameters.

16.4.2 Environment Variables

16.4.3 Assumptions

The init() routine is called before any other access programs.

16.4.4 Access Routine Semantics

init():

- transition: createServiceClients();
- description: This routine initializes the module.

setArm(armReq):

- transition: temp := armingClient.call(armReq)
- output: out := temp.success

setRtlAlt(altitude):

- transition: res := paramSetClient.call(altitude)
- exception: { res.success = FAIL ⇒ ReqFailed }
- description: This method sets the return to launch altitude parameter, which is the altitude the drone hovers at while in the "RTL" flight controller state.

sendTakeoffCmd(pose):

- transition: res := takeoffClient.call(pose)
- exception: { res.success = FAIL ⇒ ReqFailed }

- description: If given 'True' as input, this method arms the drone, while if given 'False' this method disarms the drone.

setMode(modeReq):

- transition: res := setModeClient.call(modeReq)
- exception: { res.success = FAIL ⇒ ReqFailed }
- description: Used for sending flight mode commands in the flight controller's firmware. This Service Client is used to set the Operation States used by the flight controller's firmware, which is different from the Operation States used by the Operations Manager.

16.4.5 Local Functions

createServiceClients():

- description: This routine creates and initializes Service Clients (armingClient, paramSetClient, takeoffClient setModeClient).

17 MIS of Operation States

17.1 Module

The secret of this module is the implementation of each of the nearly dozen drone operation states specified in the SRS (3.2.2 and 3.2.3).

The information here shows the state interface that each of the nearly dozen states/types inherit from. In the Operations Manager, the abstract state interface described below is used to manipulate the concrete state class.

17.2 Uses

Module	Imported Types, Constants, Routines	Description
Operations Manager	Operations Manager	Operations Manager contains all of the data the various Operation States may need, such as a reference to the Message Socket, DDC Topic Interface, DDC Service Interface etc.

17.3 Syntax

17.3.1 Exported Types

State = ?

17.3.2 Exported Access Programs

Name	In	Out	Exceptions
new State	opMan: Operations Manager, initCommand: dict, name: string	-	-
process	-	-	-
transitionNextState	-	OperationState	-
getIdentity	-	string	-

17.4 Semantics

17.4.1 State Variables

Name	Type	Description
identity	string	A unique string identifier for the Operation States, created during object construction.
context	Operations Manager	This variable is a reference to the Operations Manager. Operation States manipulate the Operations Manager to implement their behaviors specified in the SRS. The Operations Manager stores any topics, services, and functionalities a state could need to implement its responsibilities as per SRS (3.2.2 and 3.2.3).
command	dict	Contains the initial command information that was used to transition into this state. Its fields contain state-specific information, e.g. for the Configure state the command contains the value of the height parameters that the user wanted to configure.

17.4.2 Environment Variables

17.4.3 Assumptions

17.4.4 Access Routine Semantics

new State(opMan, initCommand, name):

- transition: context, command, identity := opMan, initCommand, name
- output: out := self

getIdentity():

- output: out := identity

process():

- description: This method performs any state-specific behavior by utilizing the services, topics, and modules found in the context variable. Its behavior is unique to the Operation States, and is specified in the SRS. This function will be called every frame while this state is still active and operational (i.e. until a new state is transitioned to).

transitionNextState():

- transition: // determine the new state
- output: out := newState
- description: Using the information within the context and within the current Operation State, this method determines the next Operation State. If the next state is the same as the current state, this function returns itself. If the next state is a different state, this function returns a new state object. It is called every frame. Its behavior is unique to the Operation State and is specified in the SRS.

17.4.5 Local Functions

18 MIS of Operations Manager

18.1 Module

The secret of this module is the storage and execution of modules running on the drone, that is it collects relevant topics, data, services, etc., and then hands over execution to the Operation States Modules. The Operation States routines (that implement SRS requirements) will manipulate the fields of the Operations Manager.

The Operations Manager and Operation States implement the "State" UML design pattern.

The MIS of this module is formalized to increase clarity.

18.2 Uses

Module	**Imported Types, Constants, Routines**	**Description**
Operation States	Operation State	-
DDC Topic Interface	DDC Topic Interface	-
Message Socket	Message Socket	-
DDC Service Interface	DDC Service Interface	-

18.3 Syntax

18.3.1 Exported Constants

18.3.2 Exported Access Programs

Name	In	Out	Exceptions
init	messageSocket: Message Socket, ddcTopicInterface: DDC Topic Interface, ddcServiceInterface: DDC Service Interface	-	-
process	-	-	-

18.4 Semantics

18.4.1 State Variables

Name	Type	Description
droneSocket	Message Socket	This is the Communication Module to send updates and receive commands from the user. This should be configured to be a server socket.
paramFile	string	Filename that permanently stores user parameters for the drone, so that the parameters can be saved and reset during boot.
params	dict	Contains user parameters for the drone.
FSMState	OperationState	Contains the current Operation States Module.
userError	User Error Enum	Contains the user error.
healthStatus	Health Enum	contains the health status.
topicInterface	DDCTopicInterface	Contains the topic interface module.
serviceInterface	DDCServiceInterface	Contains the service interface module.

18.4.2 Environment Variables

18.4.3 Assumptions

The init() routine is called before any other access programs.

18.4.4 Access Routine Semantics

init(messageSocket, ddcTopicInterface, ddcServiceInterface):

- transition: droneSocket = messageSocket;
  topicInterface = ddcTopicInterface;
  serviceInterface = ddcServiceInterface;
  droneSocket.init(); topicInterface.init(); serviceInterface.init(); userError = None;
  healthStatus = Healthy;
  params = readParam(paramFile);
  FSMState = Idle(this, , 'Idle'); // Creates an 'Idle' Operation State

- description: Called once at the very beginning, this routine initializes the member fields. Initially there are no errors, and everything is assumed healthy. Parameters are read from the path specified in paramFile. The operation state is initialized to the Idle states, as per SRS (TRANS_002).

process():

- transition: FSMState.process();
  FSMState = FSMState.transitionNextState();
  sendHeartbeat();

- description: Called every frame, this routine first executes the process routine of the active OperationState. Then it activates the new FSMState. Finally it sends a heartbeat message to the Operator Application.

18.4.5 Local Functions

sendHeartbeat():

- transition: if (FSMState.getIdentity() != 'CommunicationLost):
  droneSocket.sendAsyncMessage({
  'Type': 'Heartbeat',
  'State': FSMState.getIdentity(),
  'Ardupilot State': topicInterface.getState().mode,
  'Occupancy Map': topicInterface.getOccupancyMap(),
  'Relative Altitude': topicInterface.getRelAlt(),
  'Armed': topicInterface.getState().armed,

'Latitude': topicInterface.getPose().latitude,
'Longitude': topicInterface.getPose().longitude,
'Local Position X': topicInterface.getLocalPose().pose.position.x,
'Local Position Y': topicInterface.getLocalPose().pose.position.x,
'Battery Percentage': topicInterface.getBatteryInfo().percentage,
'User Error': userError.value,
'Health Status': healthStatus.value
})

- description: A heartbeat message contains all the information the User Interface needs from the drone, such as information about the parking lot and drone status. This message is only sent if the drone is in a connected state

readParam(filename):

- description: Opens and reads the parameter file, stores the parameters in a dict, and returns it. The choice of format used to store the parameters and to read the parameters is left to the developer.

19 MIS of Main DDC Module

19.1 Module

The secret of this module is the execution and operation of the process running the finite state machine.

This module is one of four modules that are run as processes on the drone's hardware. This module is will run the central decision-making process, that instructs the drone's controllers based on information from various sources (e.g. user's commands, battery capacity, height of the drone, and results of algorithms). These responsibilities are handled by routines in the Operations Manager module. Because this module is run as a standalone process, it also needs to create the Abstract Objects.

19.2 Uses

Module	Imported Types, Constants, Routines	Description
Operations Manager	Operations Manager	-
DDC Topic Interface	DDC Topic Interface	-
Message Socket	Message Socket	-
DDC Service Interface	DDC Service Interface	-

19.3 Syntax

19.3.1 Exported Constants

19.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	This is the starting point for program execution.

19.4 Semantics

19.4.1 State Variables

Name	Type	Description
operationManager	Operations Manager	-
topicInterface	DDC Topic Interface	-
droneSocket	Message Socket	-
serviceInterface	DDC Service Interface	-
serverIP	string	IP address of the server socket.
port	int	Port used for communication with the client socket.

19.4.2 Environment Variables

19.4.3 Assumptions

19.4.4 Access Routine Semantics

main():

- transition: serverIP = ""; // Server sockets have no IP, as per Base Socket implementation
  serverPort = 3000; // The port number can be changed
  droneSocket = new MessageSocket("SERVER", serverIP ,serverPort)
  topicInterface = new TopicInterface();
  serviceInterface = new ServerInterface();
  operationManager = new OperationManager(droneSocket, topicInterface, serviceInterface);
  opeartionManager.init()
  while (True) {operationManager.process();};

19.4.5 Local Functions

20 MIS of User Interface

20.1 Module

The secret of this module is the interaction with the user to display outputs and gather inputs.

20.2 Uses

Module	Imported Types, Constants, Routines	Description
Operator Camera	Operator Camera	-
Message Socket	Message Socket	-

20.3 Syntax

20.3.1 Exported Constants

20.3.2 Exported Access Programs

Name	In	Out	Exceptions
UserInterface	droneInterf: MessageSocket	-	-
StartDroneCameraDisplay	opCam: OperatorCamera	-	-
exec_	-	-	

20.4 Semantics

20.4.1 State Variables

Name	Type	Description
operatorCamera	Operator Camera	-
droneInterface	Message Socket	-
commandForDrone	dict	-

20.4.2 Environment Variables

The variables listed in the table below are in the environment, and it is the secret of the GUI to detect them, and notify the drone of there values.

For the boolean environment variables, at a given moment in time only one will of the variables will be true.

Name	Type	Description
kill	bool	
connect	bool	
configure	bool	
arm	bool	
takeoff	bool	
autonomousExplore	bool	
compulsiveMove	bool	
autonomousMove	bool	
desiredLocationX	float ($\mathbb{R}$)	Relative movement in the latitudinal direction.
desiredLocationY	float ($\mathbb{R}$)	Relative movement in the longitudinal direction.
minHoverHeight	float ($\mathbb{R}$)	
maxHoverHeight	float ($\mathbb{R}$)	
desiredHoverHeight	float ($\mathbb{R}$)	

20.4.3 Assumptions

20.4.4 Access Routine Semantics

new UserInterface(droneInterf):

- transition: droneInterface = droneInterf;
  setupUI();
- output: -
- description: Launches and starts the GUI thread.

StartDroneCameraDisplay(opCam):

- transition:
  operatorCamera = opCam;
  operatorCamera.init();
  while True: ShowImage(Camera.getImage());
- output: -
- description: Launches and starts the GUI thread.

exec_():

- transition: while (¬ kill) {
  processInputs();
  updateDisplay(operatorSocket.getMessage()); }

- output: -

- description: This routine does not exit until the user closes the app.

20.4.5 Local Functions

processInputs():

- transition: if (connect) droneSocket.init();
  elif (configure) operatorSocket.send({"Type":"Configure", "Min":minHoverHeight, "Des": desiredHoverHeight, "Max": maxHoverHeight});
  elif (arm) operatorSocket.send({"Type": "Arm"});
  elif (takeoff) operatorSocket.send({"Type": "Takeoff"});
  elif (autonomousExplore) operatorSocket.send({"Type": "Autonomous Explore"});
  elif (compulsiveMove) operatorSocket.send({"Type": "Compulsive Move", "X": desiredLocationX, "Y": desiredLocationY});
  elif (autonomousMove) operatorSocket.send({"Type": "Autonomous Move", "X": desiredLocationX, "Y": desiredLocationY});

- description: This routine checks all of the environment variables. If an environment variable is triggered, send the appropriate command to the drone. For the boolean environment variables, at a given moment in time, only one of the variables will be true.

updateDisplay(heartBeat: dict):

- description: This routine updates the display using the heartbeat message received from the drone. Widgets displaying the occupancy map and the live drone images are updated. Furthermore widgets showing the status of the drone (such as drone altitude, drone location, etc.) are updated. As this module is specific to the widgets, its design is left to the developer.

setupUI():

- description: Launches user interface.

ShowImage(image: Image Array):

- description: Shows the image in a resizeable popup window.

21 MIS of Main Interface Module

21.1 Module

This module is run as a stand-alone process on the Operator's PC. This module runs the user interface and the communication of the operator with the drone. Because this module is run as a standalone process, it creates the abstract objects/modules as well.

21.2 Uses

Module	Imported Types, Constants, Routines	Description
Operator Camera	Operator Camera	-
Message Socket	Message Socket	-
User Interface	User Interface, StartDroneCameraDisplay	-

21.3 Syntax

21.3.1 Exported Constants

21.3.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

21.4 Semantics

21.4.1 State Variables

Name	Type	Description
operatorCamera	Operator Camera	-
operatorSocket	Message Socket	-
userInterface	User Interface	-
IP	string	Private IP address of the drone
port	int ($\mathbb{N}$)	Private IP address of the drone

21.4.2 Environment Variables

21.4.3 Assumptions

21.4.4 Access Routine Semantics

main():

- transition: operatorCamera = new OperatorCamera();
  IP = '192.168.1.100';
  This is the static IP address of the drone. It may be something else.
  port = 3000;
  It may be something else.
  droneInterface = new MessageSocket("OPERATOR", IP, port);
  StartDroneCameraDisplay(droneInterface);
  userInterface = new UserInterface(droneInterface);
  userInterface.exec_();
- description: This is the starting point of execution for the process running on the Operator's PC.

21.4.5 Local Functions

22 Appendix

[Extra information if required —SS]