User Manual

Dynamic Autorouting Program for Multi-Agent Systems

v.3.4

written by

Komsun Tamanakijprasart

School of Aerospace, Transport and Manufacturing (SATM)

PhD in Aerospace

Dr. Sabyasachi Mondal, Prof. Antonios Tsourdos

Table of Contents

Acknowledgement

We would like to acknowledge that the Dynamic Autorouting Program for Multi-Agent Systems is currently in the developmental phase. The purpose of this user manual is to facilitate a comprehensive understanding of the program without delving extensively into the underlying physics of the algorithms employed.

This manual also provides a broad overview of the program's capabilities and flexibility, serving as a foundational guide for its future development. Utilizing the Object-Oriented Programming (OOP) style, the program has been engineered to optimize efficiency and establish a robust groundwork for potential migration to other programming languages or platforms.

Your feedback is invaluable to us. Should you have any inquiries or encounter errors or bugs within the program, we welcome you to reach out to Komsun. Your contributions and insights are very crucial in refining and completing this program.

Komsun T.

PhD in Aerospace, Cranfield University

k.tamanakijprasart.263@cranfield.ac.uk

11 December 2023

Nomenclature

CCA3D : Three-dimensional Carrot-Chasing Algorithm

DA : Dynamic Autorouting

IFDS : Interfered Fluid Dynamical System

IPN : Inverse Proportional Navigation

UAV : Unmanned Aircraft Vehicle

Terminology

Class : A class describes the contents of the objects that belong to it: it describes a collection of instance variables (properties), and defines the operations (methods)

Object : An object is an element (or instance) of a class; objects have the behaviours of their class

Method : An action which an object is able to perform

Public Method : A method that user can call from the object of that class

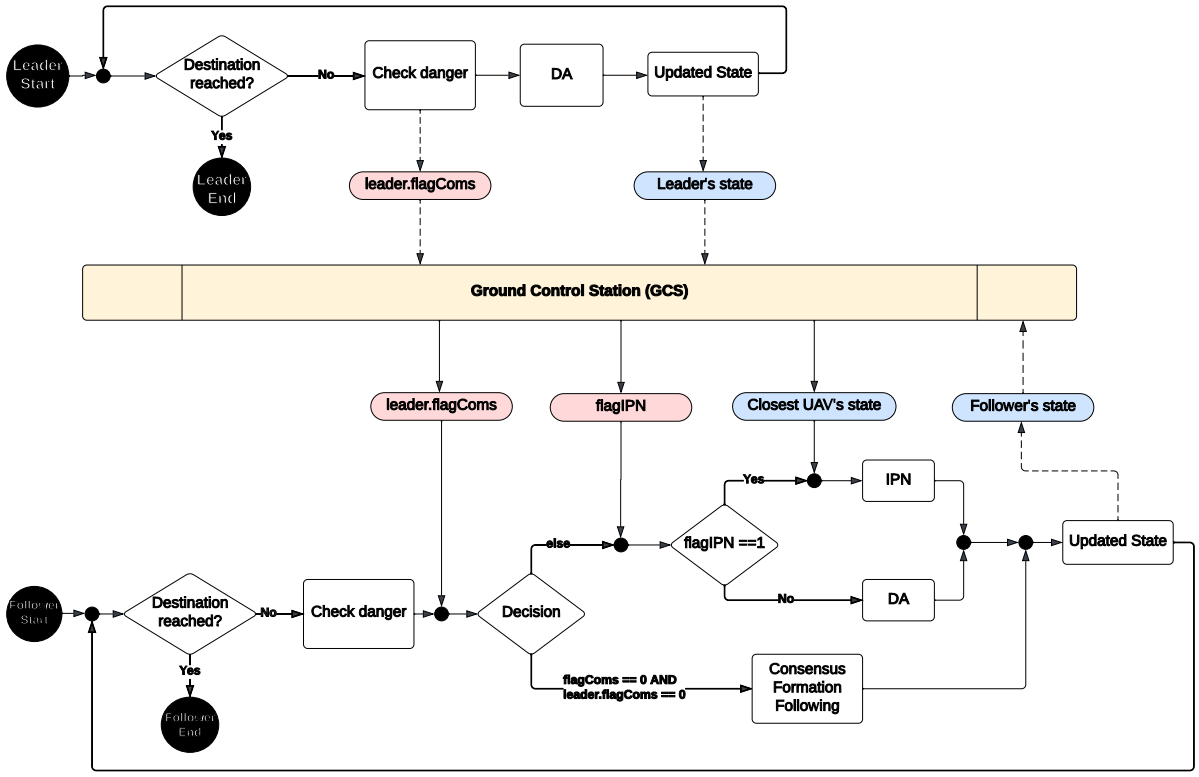
Protected Method : A hidden method that only the object of that class and its inheritance can call, but not user

Program Overview

The program is structured around three key classes: **BasicUAV**, **FollowerUAV**, and **GCS**. In general, the leader UAV will be assigned to be **BasicUAV**. The **FollowerUAV** class, inheriting from **BasicUAV**, inherits all attributes, methods, and properties of the **BasicUAV**, while introducing additional methods specifically tailored for follower functionalities.

The **FollowerUAV** introduces a dynamic element with the option to operate in either independent Dynamic Autorouting (DA) mode or Consensus Formation Following mode. This decision is autonomously determined, taking into account a variety of influencing factors.

Facilitating communication and reducing computational load on the leader UAV, the Ground Control Station (**GCS**) serves as an intermediary between the leader and its follower UAVs. This strategic architecture enhances the overall efficiency of the system. The program flow is illustrated in the diagram below:



The program is coded in MATLAB 2022b and require the installation of following toolboxes:

1) Instrument Control Toolbox (essential)
2) Symbolic Math Toolbox (for visualization)

1. Declaring the UAV

The UAV can be declared by creating an object of either **BasicUAV** class, or **FollowerUAV** class.

Example:

```
1 | leader = BasicUAV;
2 | agent1 = FollowerUAV;
3 | agent2 = FollowerUAV;
```

Console output:

```
* [Unnamed FollowerUAV] is created
* [Unnamed FollowerUAV] is created
* [Unnamed FollowerUAV] is created
```

2. UAV's Placement

2.1 Default placement

When the UAV is declared without any input argument, the UAV is placed at the default location with default orientation

Default location (x, y, z) (m) : $[20, 20, 30]$

Default orientation (ψ, γ) (rad) : $[0, 0]$

2.2 Manual placement

There are 2 ways that user can place the UAV manually:

Method 1: Input the argument of $(x_i, y_i, z_i, \psi_i, \gamma_i)$ when declaring the UAV

Method 2: Call the **SetCurrentPos(x, y, z)** and **SetCurrentAngle(psi, gamma)**

Example:

```
1 | % UAV placement (Method 1)
2 | agent1 = BasicUAV(1,2,3, pi/2, pi/4);
3 | % UAV placement (Method 2)
4 | agent2 = BasicUAV;
5 | agent2.SetCurrentPos(10, 0, 30)
6 | agent2.SetCurrentAngle(pi/6, pi/8)
```

2.3 Checking UAV's current position and orientation

To check the current position and angle of the UAV, call **GetCurrentPos()** and **GetCurrentAngle()** methods

Example:

```
1 | % Check current placement
2 | agent1.GetCurrentPos()
3 | agent1.GetCurrentAngle()
```

Console output:

```
ans =
         1×3 single row vector
            1     2     3
       ans =
         1×2 single row vector
           1.5708    0.7854
```

3. Configuration File

User can adjust and modify the configuration for the simulation, the UAV specs, the algorithms tuning, and the scenarios selection in the function **Config.m** The output of this function is a structure containing all the information.

Please note that the function **Config.m** will be called in the main script and will be the input argument of the **Initialize()** method of the **BasicUAV** and **FollowerUAV** class.

4. Initialization

4.1 Setting UAV's name

The UAV name should be set before the initialization so that the log can be easily understood. To set and get the UAV's name, call **SetName(name)** and **GetName()** methods respectively.

Example:

```
1 | someUAV = BasicUAV;
2 | someUAV.SetName("Leader");
3 | someUAV.GetName()
```

Console output:

```
* [Unnamed BasicUAV] is created
* [Unnamed BasicUAV]'s name has been changed to Leader
ans =
    "Leader"
```

4.2 Scenario creation

In this program, a scenario encompasses simplified shape obstacles such as spheres, cylinders, cones, and parallel pipes. While predefined scenarios serve as templates, users have the flexibility to craft custom scenes. To achieve this, navigate to the folder **@BasicUAV** and locate the file **IFDS.m**. Starting from line 245 within this script, users can freely modify the coordinates and shapes of existing obstacles.

Additionally, users can create entirely new scenes within the create_scene function. To position obstacles, utilize functions like **create_sphere(), create_cylinder(), create_cone(),** or **create_pipe()** by providing appropriate input arguments for coordinates and parameters.

```
function Obj = create_scene(num, Obj, X, Y, Z, rt)
    switch num
        case 0
            Obj(1) = create_ceiling(100, 0, 50, 200, 10, Obj(1));

        case 1  % Single object
            Obj(1) = create_sphere(100, 5, 0, 50, Obj(1));
            % Obj(1) = create_cone(100, 5, 0, 50, 80, Obj(1));
            % Obj(1) = create_sphere(100, 180, 0, 50, Obj(1));


        case 2 % 2 objects
            Obj(1) = create_cone(90, 5, 0, 50, 80, Obj(1));
            Obj(2) = create_cylinder(160, -20, 0, 40, 70, Obj(2));

            % Obj(1) = create_cylinder(60, 5, 0, 30, 50, Obj(1));
            % Obj(2) = create_sphere(120, -10, 0, 50, Obj(2));

        case 3 % 3 objects
            Obj(1) = create_cylinder(60, 5, 0, 30, 50, Obj(1));
            Obj(2) = create_sphere(120, -10, 0, 50, Obj(2));
            Obj(3) = create_cone(168, 0, 0, 25, 80, Obj(3));
```

However, it is essential to note that if a new scenario is introduced, users must specify the number of obstacles for this scenario in the **Config()** function, starting from line 86, as illustrated below.

```
86   switch scene
87       case 0, numObj = 1;
88       case 1, numObj = 1;
89       case 2, numObj = 2;
90       case 3, numObj = 3;
91       case 4, numObj = 3;
92       case 5, numObj = 3;
93       case 7, numObj = 7;
94       case 12, numObj = 12;
95       case 41, numObj = 3;
96       case 42, numObj = 4;
97       case 43, numObj = 4;
98   end
```

4.3 Initializing the UAV

Before simulating the UAV's flight, the UAV must be initialized to support the Dynamic Autorouting program (DA), the constraint matrix, and the declaration of variables. To initiate the UAV, call the **Initialize()** method. It is essential to provide the output of the **Config()** function as the input for this method, ensuring that all parameters are appropriately loaded for seamless operation.

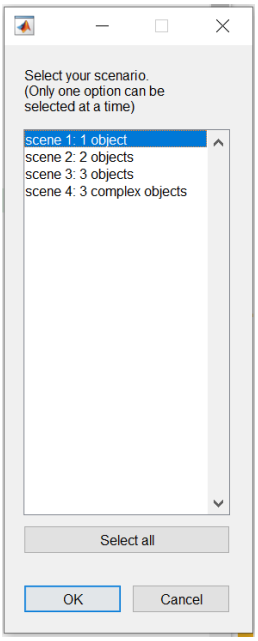
Example:

```
% Load the configuration
conf = Config();
% Declare the UAV
someUAV = BasicUAV;
someUAV.SetName("Leader");
someUAV.Initialize(conf)
```

Console output:

```
* [Unnamed BasicUAV] is created
* Rename [Unnamed BasicUAV] to Leader
* Initializing Leader . . .
_________DA configurations : __________
Number of object: 2
Shape-following: Off
Environmental constraints: None
Path optimization: Off
```

Upon invoking the **Initialize()** method, a dialog box will appear, prompting the user to select a scenario. This step enables the user to specify the particular scenario under consideration.



4.4 Initializing the UAV's path

The default path's starting point and its destination are as follows:

Default path's origin $(x_{ini}, y_{ini}, z_{ini})$ (m) : $[20, 0, 30]$

Default destination $(x_{final}, y_{final}, z_{final})$ (m) : $[250, 0, 50]$

The path's starting point may not necessarily be the same as the UAV's initial location. To set the path's starting point and the destination, call **SetItsPathOrigin(x, y, z)** and **SetItsDestination(x, y, z)** methods.

Example:

```
someUAV = BasicUAV;
someUAV.SetName("Leader");
someUAV.SetItsPathOrigin(10,20,30)
someUAV.SetItsDestination(200,10,50)
```

Console output:

```
* [Unnamed BasicUAV] is created
* [Unnamed BasicUAV]'s name has been changed to Leader
* Leader's path origin has been changed from (20,0,30) to (10,20,30)
* Leader's destination has been changed from (250,0,50) to (200,10,50)
```

5. Propagation of the UAV

5.1 BasicUAV

For the class **BasicUAV**, the Dynamic Autorouting (DA) is the only method to propagate the UAV. The Dynamic Autorouting program includes 1) the IFDS algorithm for path planning, 2) the CCA3D for path following, and 3) IPN for collision avoidance. These three algorithms will be automatically chosen based on detection range, the danger flag, the IPN flag, etc.

To propagate the **BasicUAV** using Dynamic Autorouting, call the **UpdateDA()** method. It is important to note that the UAV must be initialized before calling this method, and every time **UpdateDA()** is called, the timestep is getting incremented.

Example:

```
conf = Config();
someUAV = BasicUAV;
someUAV.SetName("Leader");
someUAV.Initialize(conf)
for i = 1:5
    someUAV.UpdateDA()
end
```

Console output:

```
* [Unnamed BasicUAV] is created
* [Unnamed BasicUAV]'s name has been changed to Leader
* Initializing Leader . . .
_________DA configurations : __________
Number of object: 2
Shape-following: Off
Environmental constraints: None
Path optimization: Off
Loading Constraint Matrix...
_____________________________________
Leader: Simulation time = 0.01, Computed time = 0.015983 s
Leader: Simulation time = 0.02, Computed time = 0.016102 s
Leader: Simulation time = 0.03, Computed time = 0.015258 s
Leader: Simulation time = 0.04, Computed time = 0.015015 s
Leader: Simulation time = 0.05, Computed time = 0.016098 s
```

5.2 FollowerUAV

The class **FollowerUAV** is derived from **BasicUAV**, thus it has all the functions as the **BasicUAV**. This means the **FollowerUAV** can also propagate its state using **UpdateDA()** method. However, the **FollowerUAV** has additional **UpdateFollower()** and **Follow()** method in which the basic decision making is implemented to select between the DA mode or the Consensus-Formation-Following mode. This decision making is based on three factors: the position to be followed, the obstacles in the field, and the safety of the leader UAV.

To propagate **FollowerUAV** with decision making, call **UpdateFollower(X, Object, leaderFlag).** This method takes in 3 inputs: the position to follow, the object structure, and the danger flag from the leader UAV.

Example:

```
1 | agent1.UpdateFollower(X_nei(1,:), leader.DA.Object, leader.GetFlagDanger);
```

6. Ground Control Station (GCS)

6.1 Formation Flight Creation

A formation flight, determined by a reference position, can be generated through the **GCS** class using the **CreateFormation(XL, no_uav, option)** method. This method requires three inputs: the reference position XL (x, y, z), the number of FollowerUAVs denoted as no_uav, and the chosen formation option.

The available formation options include: 1) diagonal line, 2) star, 3) triangle, 4) clockwise twisting star, 5) clockwise twisting star (large), and 6) horizontal line.

GCS Example:

```
1 | conf = Config()
2 | someUAV = BasicUAV;
3 | someUAV.SetName("Leader");
4 | someUAV.Initialize(conf)
5 |% Set reference position = Leader's current position
6 | XL = someUAV.GetCurrentPos();
7 |% Option for the formation
8 | no_uav = 5;
9 | formation = 2;
10 |   % Generate the formation
11 |   pos2Fol = gcs.CreateFormation(XL, no_uav, formation);
```

* In this version (v.3.4), the number of followerUAV is fixed to be 5 (no_uav = 5)

6.2 Consensus Formation Following

In this program version (v.3.4), the consensus formation following can be called directly from the main file with **consensus_func(X_nei, pos2Fol, XL_states, no_uav)** function. This function takes 4 inputs: current followers positions X_nei, the position to follow pos2Fol, the reference (leader) states XL_states, and number of followers no_uav.

The **consensus_func** function compute the Consensus-Formation-Following algorithm and output the states to be followed.

GCS Example 2:

```
conf = Config()
someUAV = BasicUAV;
someUAV.SetName("Leader");
someUAV.Initialize(conf)
% Set reference position = Leader's current position
XL = someUAV.GetCurrentPos();
% Option for the formation
no_uav = 5;
formation = 2;
   % Generate the formation
   pos2Fol = gcs.CreateFormation(XL, no_uav, formation);
   % Compute the Consensus-Formation-Following
   X_nei = consensus_func(X_nei, pos2Fol, XL_states, no_uav);
```

6.3 IPN Activator

As the **GCS** continuously receives updates on the positions of all UAVs, it capitalizes on this information to assess if any pair of UAVs has approached within an undesirable proximity, falling below a specified threshold. When such a situation arises, the **GCS** triggers the activation of the Inverse Proportional Navigation (IPN) algorithm to facilitate collision avoidance among UAVs.

This critical operation is executed through the **CheckIPN(XL, X_nei)** method within the **GCS** class. Input parameters for this method include the current position XL of the leader UAV and the positions X_nei of its followers. The output from this method comprises the index of the UAV requiring IPN activation and detailed information about the closest UAV involved in the potential collision scenario.

GCS Example 3:

```
   conf = Config()
   someUAV = BasicUAV;
   someUAV.SetName("Leader");
   someUAV.Initialize(conf)
   % Set reference position = Leader's current position
   XL = someUAV.GetCurrentPos();
   % Option for the formation
   no_uav = 5;
   formation = 2;
   % Generate the formation
   pos2Fol = gcs.CreateFormation(XL, no_uav, formation);
   % Compute the Consensus-Formation-Following
   X_nei = consensus_func(X_nei, pos2Fol, XL_states, no_uav);
   % Check the activation of IPN for followerUAV
   [ipnIdx, closestIdx] = gcs.CheckIPN([XL; X_nei(:,1:3)]);
```

7. Program Example

7.1 The Initialization

```
% .-------------------------------------------------------------.
% | Dynamic Autorouting Program for Multi-Agent Systems v.3.4   |
% | created by Komsun Tamanakijprasart and Dr.Sabyasachi Mondal |
% '-------------------------------------------------------------'

clc, clear, close all

animation = 0;
saveVid = 0;

formation = 2;  % 1) Diagonal line 2) Star 3) Triangle
                % 4) Twisting star 5) Large twisting star
                % 6) Horizontal line

% Initialize GCS
gcs = GCS;
no_uav = 5; % fixed for this version

% Pre-allocation
Xnei_states = zeros(3,no_uav);
pos2Fol = zeros(3,no_uav);

% Create UAVs
leader = BasicUAV;
leader.SetName("Leader");

XX = leader.GetCurrentPos();

% Setting Initial states for followers
for ii = 1:1:no_uav
Xnei_states(:,ii) = [XX(1)-(-1)^ii*ii*1 ;
                     XX(2)+(-1)^(ii+1)*ii*0.5;
                     XX(3)+(-1)^(ii+1)*ii*1];
end

% Initialize followers
agent1 = FollowerUAV(Xnei_states(1,1), Xnei_states(2,1), Xnei_states(3,1),
    0, 0);
agent2 = FollowerUAV(Xnei_states(1,2), Xnei_states(2,2), Xnei_states(3,2),
    0, 0);
agent3 = FollowerUAV(Xnei_states(1,3), Xnei_states(2,3), Xnei_states(3,3),
    0, 0);
agent4 = FollowerUAV(Xnei_states(1,4), Xnei_states(2,4), Xnei_states(3,4),
    0, 0);
agent5 = FollowerUAV(Xnei_states(1,5), Xnei_states(2,5), Xnei_states(3,5),
    0, 0);
agent1.SetName("Agent1");
agent2.SetName("Agent2");
agent3.SetName("Agent3");
agent4.SetName("Agent4");
agent5.SetName("Agent5");

gcs.numUAV = 5;

% Load the configuration
conf = Config();
```

```
51|
52|% Initialization
53|leader.Initialize(conf)
54|agent1.Initialize(conf)
55|agent2.Initialize(conf)
56|agent3.Initialize(conf)
57|agent4.Initialize(conf)
58|agent5.Initialize(conf)
59|
60|% Set different destination
61|followerDestin = gcs.CreateFormation(leader.GetItsDestination, no_uav,
62|formation);
63|ds1 = followerDestin(:,1)';
64|ds2 = followerDestin(:,2)';
65|ds3 = followerDestin(:,3)';
66|ds4 = followerDestin(:,4)';
67|ds5 = followerDestin(:,5)';
68|
69|agent1.SetItsDestination(ds1(1), ds1(2), ds1(3))
70|agent2.SetItsDestination(ds2(1), ds2(2), ds2(3))
71|agent3.SetItsDestination(ds3(1), ds3(2), ds3(3))
72|agent4.SetItsDestination(ds4(1), ds4(2), ds4(3))
73|agent5.SetItsDestination(ds5(1), ds5(2), ds5(3))
74|
75|X_nei = [agent1.GetCurrentState();
76|          agent2.GetCurrentState();
77|          agent3.GetCurrentState();
78|          agent4.GetCurrentState();
79|          agent5.GetCurrentState()];
80|
81|swarm = {leader, agent1, agent2, agent3, agent4, agent5};
82|
```

The program begins by defining a specific formation, denoted by the variable **formation**. The ground control station (gcs) is then initialized on line 16 using a class **GCS**. The number of UAVs in the swarm is set to 5 (**no_uav**). Two matrices, **Xnei_states** and **pos2Fol**, are pre-allocated with zeros on line 20-21.

From line 24 - 27, a leader UAV, belonging to the **BasicUAV** class and named "Leader," is created, and its initial position is initialized. The initial states for follower UAVs are calculated based on the leader's position, and follower UAVs (**FollowerUAV** class) are then initialized with their respective initial states on line 30 -34.

From line 50 - 58, a configuration structure (**conf**) is created using the **Config()** function, and both leader and follower UAVs are initialized with this configuration. From line 61 – 73, the followers are assigned specific destinations based on the previously created formation.

The current states of all UAVs, including the leader and followers, are retrieved and stored in the matrix **X_nei** on line 75. Finally, a cell array named **swarm** is created, containing references to all UAVs in the swarm (leader and followers) on line 81.

7.2 The Main Loop

```
%% Main Loop
leader.SetFlagComs(1)
i = 0;
while true
i = i+1;
% Synchronize time step
gcs.SetTimeStep(i)

% Propagate Leader with DA
leader.UpdateBasicUAV()

% Check if leader has reached the destination or not
if leader.GetFlagDestin == 1
     disp("* Leader has reached the destination!")
     break
end

      % Get Leader's current state
      XL_states = leader.GetCurrentState();
      XL = leader.GetCurrentPos();

      % Create formation from leader's current position
      pos2Fol = gcs.CreateFormation(XL, no_uav, formation);

      % Compute Consensus-Formation-Following for followers
      X_nei = consensus_func(X_nei, pos2Fol, XL_states, no_uav);

      % Check for IPN for collision avoidance among followerUAVs
      [ipnIdx, closestIdx] = gcs.CheckIPN([XL; X_nei(:,1:3)]);
      selectedUav = swarm(ipnIdx);
      closestUav = closestIdx(ipnIdx);

      % Manually feed the UAVs information to the IPN-activated UAVs
      if ~isempty(selectedUav)
          for j = 1:length(selectedUav)
              cuav = swarm{closestUav(j)};
              selectedUav{j}.flagIPN = 1;
              selectedUav{j}.gcsData = [cuav.GetCurrentPos,...
              cuav.GetCurrentAngle, cuav.itsCruisingSpeed];
          end
      End

      % ----------------------- Manual Logic ---------------------------
      % if i >= 1300
      %     leader.SetFlagDanger(0)
      % elseif i >= 600
      %     leader.SetFlagDanger(1)
      % end
      %
      % if i >= 2100
      %     leader.SetFlagDanger(0)
      % elseif i >= 1800
      %     leader.SetFlagDanger(1)
      % end

      % Update FollowerUAV states
```

```
139|      agent1.UpdateFollower(X_nei(1,:), leader.DA.Object,
   leader.GetFlagDanger);
140|      agent2.UpdateFollower(X_nei(2,:), leader.DA.Object,
   leader.GetFlagDanger);
141|      agent3.UpdateFollower(X_nei(3,:), leader.DA.Object,
   leader.GetFlagDanger);
142|      agent4.UpdateFollower(X_nei(4,:), leader.DA.Object,
   leader.GetFlagDanger);
143|      agent5.UpdateFollower(X_nei(5,:), leader.DA.Object,
   leader.GetFlagDanger);
144|
145|      X_nei = [agent1.GetCurrentState();
146|               agent2.GetCurrentState();
147|               agent3.GetCurrentState();
148|               agent4.GetCurrentState();
149|               agent5.GetCurrentState()];
150|      end
```

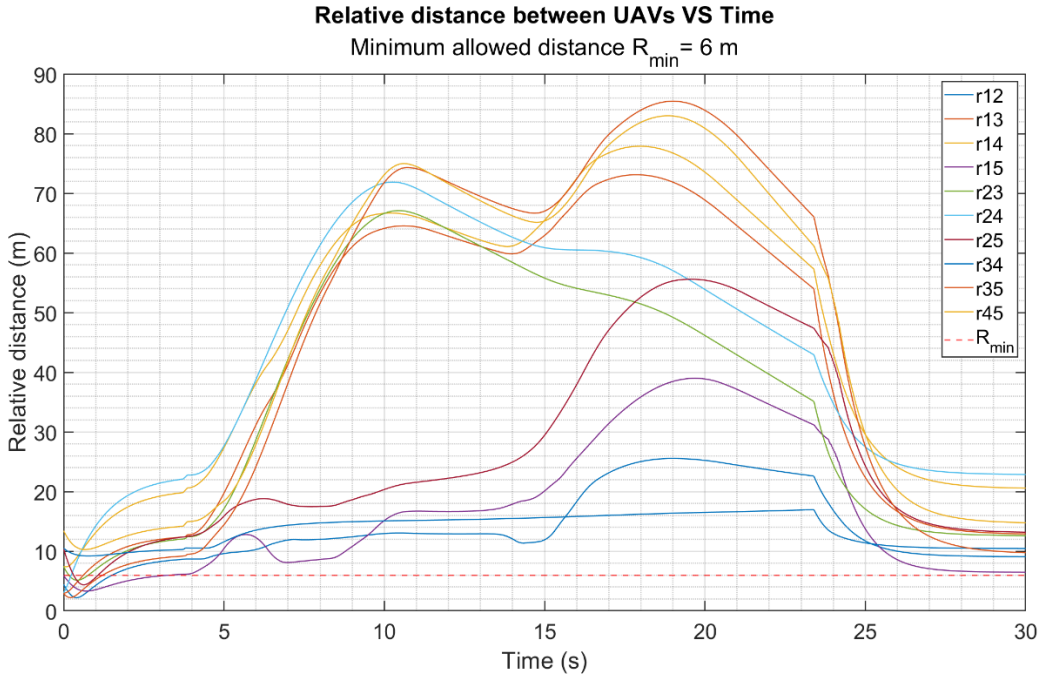
The main loop of the program commences with the activation of communication flags for the leader UAV (**leader.SetFlagComs(1)** on line 84). Then the loop iterates infinitely, incrementing a counter i with each cycle. Within this loop, the GCS's time step is synchronized with the loop counter (**gcs.SetTimeStep(i)).** The leader UAV undergoes an update through the **UpdateBasicUAV()** function on line 92. This will execute the Dynamic Autorouting program on the leader UAV.

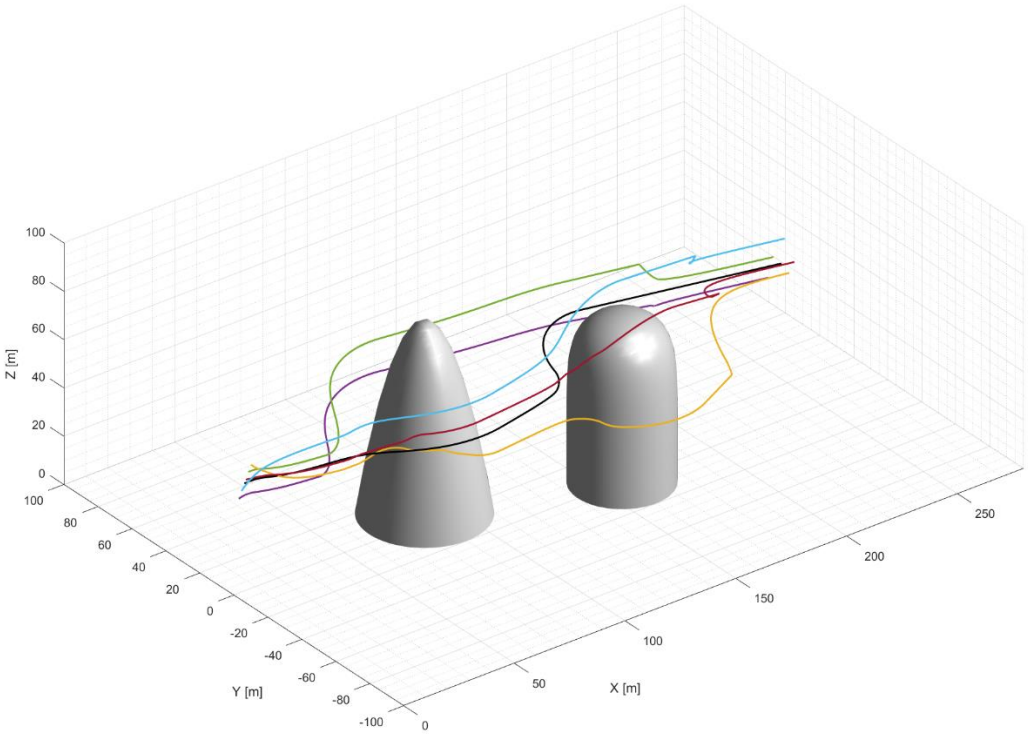
A check is performed on line 95 to determine if the leader has reached its destination (**leader.GetFlagDestin** == 1). If this condition is met, a message is displayed, and the loop terminates. After the leader is propagated, the current state and position of the leader are retrieved, and a formation is generated for the followers based on the leader's position (**gcs.CreateFormation(XL, no_uav, formation)**) on line 101 and 102. The **consensus_func** function is employed on line 108 to compute consensus-formation-following for the followers, incorporating their current states and the formation data. Collision avoidance logic is implemented through IPN among the leader and follower UAVs.

Additionally, the commented-out section from line 125 – 136 serve as an example to deploy a manual logic to set danger flags at specific time steps. From line 139 – 143, the states of follower UAVs (agent1 to agent5) are updated, considering the leader's data, its danger flag, and the current states of follower UAVs. This update encompasses the current states of all UAVs, comprising both the leader and followers. Then on line 145 – 149, the **X_nei** variable is updated with these new states. The loop persists until the leader reaches its destination, upon which it breaks out of the iterative cycle.

7.3 Results

The outcomes of this example program are illustrated in the following figures. To view the animated representation of the swarm, users can activate animation by setting the variable **animation** to 1 on line 8. Furthermore, should users desire to save this animation as a video file, they can enable the **saveVideo** option by setting it to 1 on line 9.





Appendix

Method list for class BasicUAV

class BasicUAV				
No	**Public Methods**	**Input**	**Output**	**Description**
1	Initialize(config)	Structure (config)	(UAV is initialized)	Initialize Dynamic Autorouting and load the constraint matrix
2	UpdateDA()	-	(UAV's state is updated)	Execute the Dynamic Autorouting and update the state
3	UpdateIPN()	-	(UAV's state is updated)	Execute the IPN and update the state
4	CheckDanger(Object)	structure (Object)	-	Check the range of the inputted obstacle. Change flagDanger to 1 if closer than the threshold
5	GetCurrentPos()	-	1x3 array (x, y, z)	Get UAV's current position
6	GetCurrentAngle()	-	1x2 array (psi, gamma)	Get UAV's current orientation (yaw and pitch angle)
7	GetCurrentVel()	-	1x3 array (vx, vy, vz)	Get UAV's current velocity vector
8	GetCurrentState()	-	1x6 array (x, y, z, vx, vy, vz)	Get UAV's current position and velocity
9	GetCurrentTime()	-	1x1 (t)	Get current timestep
10	GetItsDestination()	-	1x3 array (x_final, y_final, z_final)	Get the UAV's destination
11	GetItsPathOrigin()	-	1x3 array (x_ini, y_ini, z_ini)	Get the path's starting location
12	GetFlagComs()	-	1x1 uint8	Get the communication flag
13	GetFlagDestin()	-	1x1 uint8	Get the destination flag
14	GetFlagDanger()	-	1x1 uint8	Get the danger flag
15	GetName()	-	string	Get UAV's name
16	SetCurrentPos(x, y, z)	1x3 array (x, y, z)	-	Set UAV's current position
17	SetCurrentAngle(psi, gamma)	1x2 array (psi, gamma)	-	Set UAV's current orientation (yaw and pitch angle)
18	SetCurrentVel(vx, vy, vz)	1x3 array (vx, vy, vz)	-	Set UAV's current velocity vector
19	SetCurrentState(x, y, z, vx, vy, vz)	1x6 array (x, y, z, vx, vy, vz)	-	Set UAV's current position and velocity
20	SetCurrentTime(t)	1x1 (t)	-	Set current timestep
21	SetItsDestination(x, y, z)	1x3 array (x_final, y_final, z_final)	-	Set the UAV's destination
22	SetItsPathOrigin(x, y, z)	1x3 array (x_ini, y_ini, z_ini)	-	Set the path's starting location
23	SetFlagComs(x)	1x1 uint8	-	Set the communication flag (1 = coms available, 0 = no)
24	SetFlagDestin(x)	1x1 uint8	-	Set the destination flag (1 = destination reached, 0 = no)
25	SetFlagDanger(x)	1x1 uint8	-	Set the danger flag (1 = UAV is in danger, 0 = no)
26	SetName(name)	string	-	Set UAV's name
27	SetTimeStep(t)	1x1 single	-	Set current timestep
28	IncrementTimeStep()	-	1x1 (t)	Increment current timestep

No	Protected Methods	Input	Output	Description
1	DynamicAutorouting()	-	(performing DA)	Dynamic Autorouting program
2	InitializeDA()	-	(Initializing DA)	Initialize Dynamic Autorouting program
3	IPN()	-	(uh, uv)	IPN Algorithm
4	IFDS()	rho0, sigma0, loc_final, rt, Wp, Paths, Param, L, Object, weatherMat, dwdx, dwdy	(Paths, Object, totalLength, foundPath)	IFDS Algorithm
5	CCA3D_straight()	Wi, Wf, x0, y0, z0, psi0, gamma0, V, tuning	(x, y, z, psi, gamma, timeSpent)	CCA3D Algorithm

Method lists for class FollowerUAV

class FollowerUAV (derived from BasicUAV)				
No.	**Public Methods**	**Input**	**Output**	**Description**
1	Follow(state)	-	(UAV's state is updated)	FollowerUAV follows the given state
2	UpdateFollower(state, Object, leaderFlagDanger)	state, Object, leaderFlagDanger	(UAV's state is updated)	FollowerUAV chooses between following-mode and DA-mode

Method lists for class GCS

class GCS				
No.	**Public Methods**	**Input**	**Output**	**Description**
1	CreateFormation(XL, no_uav, option)	(XL, no_uav, option)	(Positions to be followed)	Generate a formation positions based on leader's current location
2	StoreTempUAVState(uavID, state)	(uavID, state)	(UAVs' states are temporary stored)	Temporary save all the UAVs' states at current timestep to GCS
3	CheckIPN(X)		(ipnIdx, closestIdx)	Determine which UAV is activating the IPN and find its closest UAV
4	GetFlagComs()	-	1x1 uint8	Get the communication flag (1 = coms available, 0 = no)
5	GetFlagDestin()	-	1x1 uint8	Get the destination flag (1 = destination reached, 0 = no)
6	SetFlagComs(val)	1x1 uint8	-	Set the communication flag (1 = coms available, 0 = no)
7	SetFlagDestin(val)	1x1 uint8	-	Set the destination flag (1 = destination reached, 0 = no)
No.	**Private Methods**	**Input**	**Output**	**Description**
1	consensus_func(X_nei, XL_pos_to_fol, XL_states, comms)	(X_nei, XL_pos_to_fol, XL_states, comms)	(Output states from Consensus-Formation-Following)	Consensus-Formation-Following calculation