README

Solution to MATLAB and Simulink Challenge project <'247'> <'Multi-UAV-Path-Planning-for-Urban-Air-Mobility'>

This repo involves approach based on bio-inspired algorithms to solve the MATLAB and Simulink Challenge Project - Multi-UAV Path Planning for Urban Air Mobility.

There is more information in this program link.

Introduction

Nowadays the transportation industry has witnessed rising requirement for path planning for Urban Air Mobility (UAM). An efficient path planning system will promisingly make a great contribution to the urban transportation, which it is generally accepted that the algorithm should minimize the sum of time and cost of multiple unmanned aerial vehicles (UAV) and create collision-free trajectories in an efficient way.

To balance these constraints, this project introduces manifold cutting-edge bio-inspired algorithms into the *Multi-UAV Path Planning for Urban Air Mobility* system, which offers significant advantages for UAV path finding by providing robust and adaptive solutions to complex, real-world challenges. A key strength is their ability to handle multi-objective optimization, effectively balancing competing goals such as energy efficiency, flight time, and safety. In consequence, leveraging these algorithms enables the system to solve the path planning for each UAV within 3~7 seconds (depending on which bio-inspired algorithm is selected). Additionally, this project visualizes the urban environment and the result of planned trajectories at the end of the code to indicate effectiveness of this system based on bio-inspired algorithms.

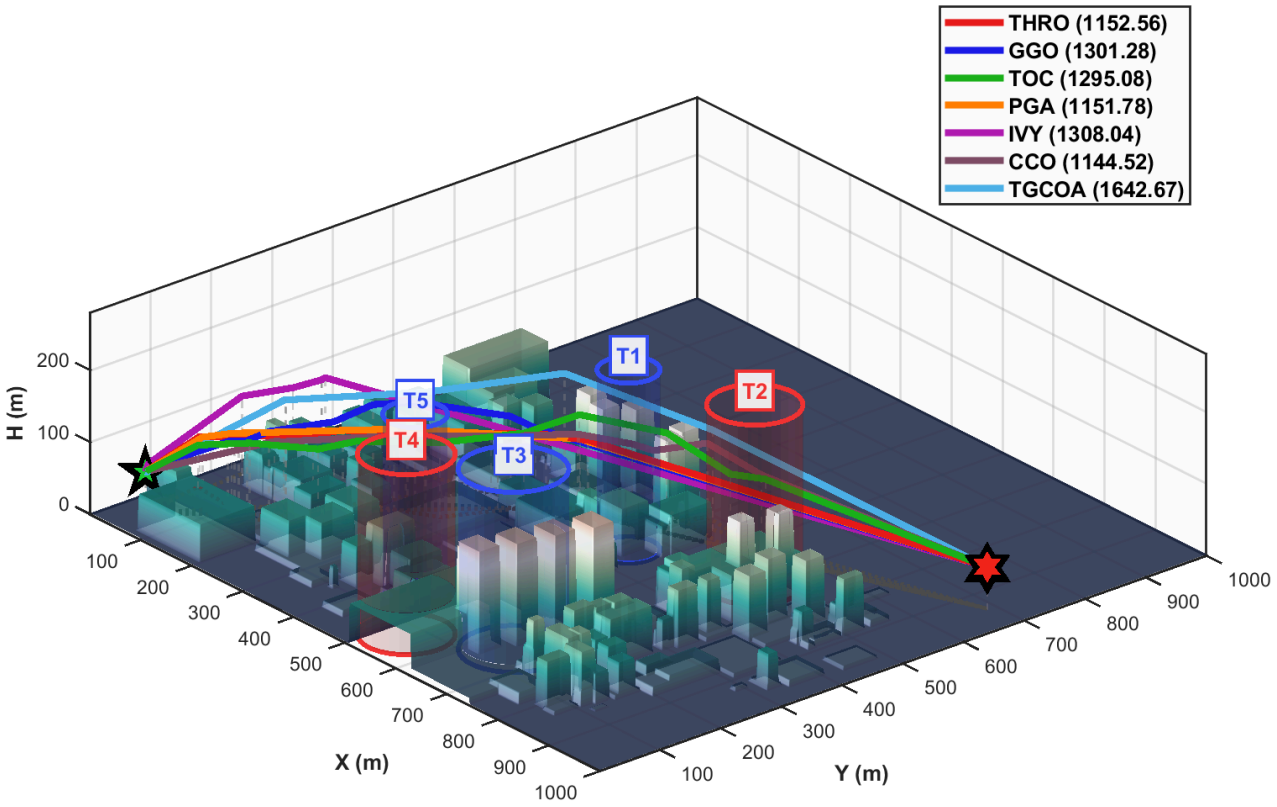
Project Details

More details are contained in this section. In the first part **System Model and Problem Formulation**, we are going to construct a system model for the urban environment and formulate a problem for optimization, where math formulas and programming implementation will be introduced. As for the second part **Utilized Algotithms**, principles of the bio-inspired algorithms we have mainly leveraged could be found there, including Cuckoo Catfish Optimizer (CCO), Greylag Goose Optimization (GGO), and Tianji's Horse Racing Optimization (THRO).

System Model and Problem Formulation

System Model

In order to address this challenge, it is necessary to construct a model for the problem. Urban environment comprises a quantity of buildings, no-fly zones and threat areas. Buildings and no-fly zones can be simplified as rectangular cuboids, while threat areas are regarded as cylinders, as shown beneath.



We implement this model by the codes below:

1. Threat Areas

   Given that threat areas are considered as cylinders, each row of the matrix `Threats` consists of the x-coordinate of the center of a cylinder, the y-coordinate of its center, its height, its radius and the type of it (The last parameter as an optional one is not used in the program).

```
% Threat Area Definition
Threats = [
    250, 300, 120, 60, 1;
    400, 500, 130, 55, 2;
    300, 700, 140, 50, 1;
    550, 250, 125, 65, 2;
    650, 550, 135, 60, 1;
    500, 800, 145, 55, 2;
    750, 350, 130, 70, 1;
    800, 700, 140, 60, 2;
];
```

2. Buildings and No-fly Zones

Buildings and no-fly zones, as cuboids, can be generated by its height and the length and width of the bottom side.

```
% Generate buildings
for cx = grid_centers_x
    for cy = grid_centers_y
        height = randi([b_height_min, b_height_max]);
        width  = randi([b_size_min, b_size_max]);
        depth  = randi([b_size_min, b_size_max]);

        if rand() > 0.5
            continue;
        end

        x_start = max(1, round(cx - width/2));
        x_end   = min(MAPSIZE_X, round(cx + width/2));
        y_start = max(1, round(cy - depth/2));
        y_end   = min(MAPSIZE_Y, round(cy + depth/2));

        H(y_start:y_end, x_start:x_end) = height;
    end
end
```

Problem Formulation

In an urban environment with buildings, no-fly zones and threat areas, suppose that the starting point of a UAV is $P_s = P_0 = [x_0, y_0, z_0]^T$ with N waypoints $P_i = [x_i, y_i, z_i]^T (i \in 1, 2, \cdots, N)$ and the end point is $P_t = P_{N+1} = [x_{N+1}, y_{N+1}, z_{N+1}]^T$. Then we can construct a matrix $\mathbf{P} \in \mathbb{R}^{3\times(N+2)}$ as:

$$\mathbf{P} = [P_0 \quad P_1 \quad P_2 \quad \cdots \quad P_N \quad P_{N+1}] = \begin{bmatrix} x_0 & x_1 & x_2 & \cdots & x_N & x_{N+1} \\ y_0 & y_1 & y_2 & \cdots & y_N & y_{N+1} \\ z_0 & z_1 & z_2 & \cdots & z_N & z_{N+1} \end{bmatrix}$$

where matrix $\mathbf{P}$ records all the coordinates of waypoints and $3 \times N$ variables need optimizing (except the starting point and the end point).

According to the matrix $\mathbf{P}$, we are able to get the expression of the path length cost, threat cost, height cost and terrain collision cost of a trajectory, and construct the cost function.

1. Path Length Cost
   Path length $L(\mathbf{P})$ is one of the primary factors in the cost function, because of its tight connection with flying time and energy consumption. Since we have already set waypoints, $L(\mathbf{P})$ can be written as

$$L(\mathbf{P}) = \sum_{n=0}^{N} \|\overrightarrow{P_n P_{n+1}}\|_2 = \sum_{n=0}^{N} \sqrt{(x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2 + (z_{n+1} - z_n)^2}$$

```
J1 = 0; % Path Length
for i = 1:N-1
    diff = [x_all(i+1)-x_all(i); y_all(i+1)-y_all(i); z_abs(i+1)-
z_abs(i)];
    J1 = J1 + norm(diff);
end
```

2. Threat Cost
   In terms of threat areas, the threat cost $T(\mathbf{P})$ relies on the distance between the UAV and the center of cylinders.

```
J2 = 0; % Threat Cost
threats = model.threats;
for i = 1:size(threats, 1)
    t = threats(i, :);
    for j = 1:N-1
        dist = DistP2S([t(1), t(2)], [x_all(j), y_all(j)], [x_all(j+1),
y_all(j+1)]);
        if dist < t(4) + 10  % Safe Distance
            J2 = J2 + J_inf;
        end
    end
end
```

3. Height Cost
   Contemporary laws usually demand that a UAV should not rise over its limited height, and UAVs are always under-performing as a consequence of heavy wind in high altitudes. Therefore, height cost $H(\mathbf{P})$ is worth considering.

```
J3 = 0; % Height Cost
for i = 1:n
    if z(i) < 150
    J3 = J3 + J_inf;
    end
end
```

4. Terrain Collision Cost
   So as to avoid collision against the ground, the cost function should also contains terrain collision cost $C(\mathbf{P})$.

```
J4 = 0; % Terrain Collision
for i = 1:N
    yid = max(1, min(round(y_all(i)), Hrmax));
    xid = max(1, min(round(x_all(i)), Hcmax));
    if z_abs(i) < H(yid, xid) + 5  % Min Height
         J4 = J4 + J_inf;
    end
end
```

And then the cost function $J(\mathbf{P})$ could be expressed as

$$J(\mathbf{P}) = [L(\mathbf{P}),\, T(\mathbf{P}),\, H(\mathbf{P}),\, C(\mathbf{P})]\mathbf{w}^T = w_1 L(\mathbf{P}) + w_2 T(\mathbf{P}) + w_3 H(\mathbf{P}) + w_4 C(\mathbf{P}),$$

where the vector $\mathbf{w} = [w_1,\, w_2,\, w_3,\, w_4]$ represents the weight of each cost respectively, and the value of it is based on experience.

```
cost_weights = [1 5 1 10];
cost = [J1 J2 J3 J4] * cost_weights.';
```

Eventually, the problem can be formulated as a mathematical programming problem and can be solved by bio-inspired algorithms:

$$\begin{aligned} \min \quad & J(\mathbf{P}) = [L(\mathbf{P}),\, T(\mathbf{P}),\, H(\mathbf{P}),\, C(\mathbf{P})]\mathbf{w}^T \\ \text{s.t.} \quad & g(\mathbf{P}) \geq 0 \end{aligned}$$

where $g(\mathbf{P}) \geq 0$ corresponds to the constraints caused by the "boundaries" of the city in the model.

```
switch ALGORITHM_OPTION
    case 'PGA'
        [fMin, bestX, conv] = PGA(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    case 'IVY'
        [fMin, bestX, conv] = IVY(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    case 'THRO'
        [fMin, bestX, conv] = THRO(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    case 'GGO'
        [fMin, bestX, conv] = GGO(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    case 'CCO'
        [fMin, bestX, conv] = CCO(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    case 'TOC'
        [fMin, bestX, conv] = TOC(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    case 'TGCOA'
        [fMin, bestX, conv] = TGCOA(pop_size, max_iter, Xmin, Xmax, dim,
fobj);
    otherwise
        error('Unknown Algorithms: %s!', ALGORITHM_OPTION);
end
```

Utilized Algorithms

We make full use of state-of-the-art bio-inspired algorithms introduced below and modify them in order that they can be applied to solve trajectories for UAVs. Actually there are more options in the `.\algorithms` directory, and all of them are executable and worth trying, including Ivy algorithm (IVY), *Tetragonula carbonaria* Optimization Algorithm (TGCOA), Tornado optimizer with Coriolis force (TOC), etc. Principles of CCO, GGO, and THRO are briefly listed as follows, and the executing results can be seen in Section Results.

CCO

The source code can be downloaded at File Exchange in MATLAB Central.

1. Core Inspiration and Fundamental Concept

   The **Cuckoo Catfish Optimizer (CCO)** is a novel metaheuristic optimization algorithm inspired by the predatory and parasitic behaviors of the cuckoo catfish (*Synodontis multipunctatus*). It simulates the fish's hunting strategies—including surrounding prey, compressing space, chaotic predation, and brood parasitism—to solve numerical optimization problems efficiently.

   CCO divides the optimization process into three adaptive stages:
   - **Exploration Phase**: Uses surround search and compressed space strategies to explore the solution space broadly.
   - **Transition Phase**: Balances exploration and exploitation for a smooth shift between phases.
   - **Exploitation Phase**: Employs chaotic predation and death/parasitism mechanisms for local refinement.
2. Mathematical Formulation
   1. Initialization

      $$x_i^d = \text{rand} \times (Ub^d - Lb^d) + Lb^d, \quad i = 1, \ldots, N, \quad d = 1, \ldots, D$$

   2. Compressed Space Strategy
      - **Equation (3)**:

        $$X_i^{new} = X_i + Z_1 \cdot |rd| \cdot \left(\frac{X_{best} + X_{r_1}}{2} - X_{r_2} \right) + \frac{r_3}{2} \cdot (X_{r_3} - X_{r_4})$$

      - **Equation (4)**:

        $$X_i^{new} = Z_2 \cdot (X_{i_1} + |rd| \cdot (X_{i_1} - X_{i_2})) + (1 - Z_2) \cdot X_i$$

      - **Equation (5)**:

        $$X_i^{new} = X_{r_1} + |rd| \cdot (X_{best} - X_i + X_{r_2} - X_{r_3})$$

      - **Self-preservation**: Triggered when $t > 0.8N$ or $\text{rand} > C$, with protection probability:

$$p = 0.2 \cdot C + 0.2$$

3. Surround Search Strategy
   - **Spiral Search (Eq. 7)**:

$$X_i^{new} = \begin{cases} X_e + F \cdot R_1 \cdot \frac{\text{step}}{2} + T^n \cdot s \cdot (1 - R_1) \cdot |\text{step}| + V \cdot \frac{J_i}{It} & \text{if } \text{mod}(i, 2) = 0 \\ X_e + F \cdot R_1 \cdot \frac{\text{step}}{2} + T^n \cdot c \cdot (1 - R_1) \cdot |\text{step}| + V \cdot \frac{J_i}{It} & \text{otherwise} \end{cases}$$

   - **Spherical Search (Eq. 16)**:

$$X_i^{new} = \begin{cases} RotX_r + 2wF\cos(Rt_1)\sin(Rt_2)(RotX_r - X_{new}) & X_q = 1 \\ RotX_r + 2wF\sin(Rt_1)\cos(Rt_2)(RotX_r - X_{new}) & X_q = 2 \\ RotX_r + 2wF\cos(Rt_2)(RotX_r - X_{new}) & X_q = 3 \end{cases}$$

   - Where:
     - $T = \left(1 - \sin\left(\frac{\pi}{2} \cdot \frac{It}{MaxIt}\right)\right)^{(It/MaxIt)}$
     - $w = 1 - \frac{e^{It/MaxIt} - 1}{e - 1}$

4. Transition Strategy (Eq. 21)

$$X_i^{new} = \begin{cases} \frac{C}{2} \cdot (r_1 X_{best} - r_3 X_i) + T^2 \cdot \text{lev}(D) \cdot |\text{Step}_2| & \text{if } \text{mod}(i, 2) = 0 \\ \frac{X_{best} + X_i}{2} + De \cdot \left(2R_1 \cdot \text{Step}_2 - \frac{R_3}{2} \cdot (De \cdot R_3 - 1)\right) & \text{otherwise} \end{cases}$$

   - **Lévy Flight**:

$$\text{lev} = 0.05 \cdot \frac{u}{|v|^{1/\beta}}, \quad \beta = 1.5$$

5. Chaotic Predation (Eq. 23)

$$X_i^{new} = \begin{cases} X_{best} + F \cdot S \cdot (X_{best} - X_i) & \text{if } J_i > J_{in} \\ X_{best}(1 + T^5 \cdot Cy \cdot E) + F \cdot S \cdot (X_{best} - X_i) & \text{else if } J_i > J_{in} \cdot Lx \\ X_{best}(1 + T^5 \cdot Gs) + F \cdot S \cdot (X_{best} - X_i) & \text{otherwise} \end{cases}$$

   where $Lx = |rd| \cdot r_1$, $Cy = \frac{1}{\pi(1 + C^2)}$ and $Gs \sim \mathcal{N}(0, C^2)$.

6. Death and Parasitism
   - **Egg distribution (Eq. 29)**:

$$X_i^{new} = r_1 \cdot (Up_c - Low_c) + Low_c$$

   - **Rebirth (Eq. 30)**:

$$X_i^{new} = r_1 \cdot (Ub - Lb) + Lb$$

   - **Switch (Eq. 31)**: Uses (29) if $\text{rand} > C$, else (30).

7. Position Update (Eq. 32)

$$X_i = \begin{cases} X_i^{new} & \text{if } F_i^{new} < F_i \\ X_i & \text{else} \end{cases}$$

GGO

The source code is now accessible at File Exchange in MATLAB Central.

1. Core Inspiration and Fundamental Concept

   GGO is a novel **nature-inspired metaheuristic algorithm** based on the social and dynamic behaviors of greylag geese, particularly their **V-shaped flight formation** during migration. It belongs to the **swarm-based optimization** family and is designed to efficiently balance **exploration** and **exploitation**.

   - Geese fly in a **V-formation** to reduce air resistance, allowing the flock to fly **~70% farther** collectively.
   - The algorithm mimics **group dynamics**: explorers search for new areas, while exploiters refine existing solutions.
   - **Dynamic group adjustment**: If the best solution stagnates for 3 iterations, the algorithm increases explorers to avoid local optima.

2. Mathematical Formulation

   1. Population Initialization
      The population X_i (where $i = 1, 2, \ldots, n$) is initialized randomly.
   2. Exploration Phase
      - **Moving toward the best solution**:

        $$\mathbf{X}(t+1) = \mathbf{X}^*(t) - \mathbf{A} \cdot |\mathbf{C} \cdot \mathbf{X}^*(t) - \mathbf{X}(t)|$$

        where $\mathbf{A} = 2\mathbf{a} \cdot r_1 - \mathbf{a}$, $\mathbf{C} = 2 \cdot r_2$, and $\mathbf{a}$ decreases from 2 to 0.
      - **Random exploration** (when $|\mathbf{A}| \geq 1$):

        $$\mathbf{X}(t+1) = w_1 \cdot \mathbf{X}_{paddle1} + \mathbf{z} \cdot w_2 \cdot (\mathbf{X}_{paddle2} - \mathbf{X}_{paddle3}) + (1 - \mathbf{z}) \cdot w_3 \cdot (\mathbf{X} - \mathbf{X}_{paddle1})$$

        where $\mathbf{z} = 1 - \left(\frac{t}{t_{max}}\right)^2$.
      - **Alternate update** (when $r_3 \geq 0.5$):

        $$\mathbf{X}(t+1) = w_4 \cdot [\mathbf{X}^*(t) - \mathbf{X}(t)] \cdot e^{bl} \cdot \cos(2\pi l) + [2w_1(r_4 + r_5)] \cdot \mathbf{X}^*(t)$$

   3. Exploitation Phase
      - **Moving toward sentries**:

        $$\mathbf{X}_1 = \mathbf{X}_{Sentry1} - \mathbf{A}_1 \cdot |\mathbf{C}_1 \cdot \mathbf{X}_{Sentry1} - \mathbf{X}|$$

        (Similarly for $\mathbf{X}_2$, $\mathbf{X}_3$)

        $$\mathbf{X}(t+1) = \frac{\mathbf{X}_1 + \mathbf{X}_2 + \mathbf{X}_3}{3}$$

      - **Searching near the best solution**:

        $$\mathbf{X}(t+1) = \mathbf{X}(t) + \mathbf{D} \cdot (1 + \mathbf{z}) \cdot w \cdot (\mathbf{X} - \mathbf{X}_{Flock1})$$

   4. Binary GGO for Feature Selection

   - Continuous values are binarized using a **sigmoid function**:

$$\chi_d^{t+1} = \begin{cases} 1 & \text{if } \mathrm{Sigmoid}(m) \geq 0.5 \\ 0 & \text{otherwise} \end{cases}$$

where $\mathrm{Sigmoid}(m) = \frac{1}{1+e^{-10(m-0.5)}}$.

THRO

The source code is publicly available at GitHub and File Exchange in MATLAB Central.

1. Core Inspiration and Fundamental Concept

   Tianji's Horse Racing Optimization (THRO) is a novel metaheuristic algorithm inspired by the ancient Chinese historical story of "Tianji's Horse Racing." The core strategic principle from the story is: **leveraging one's strengths to counteract an opponent's weaknesses through intelligent matching strategies.**

   In algorithmic terms, this translates to:
   - Maintaining two competing populations: Tianji's horses (X_T) and King's horses (X_K)
   - Each horse represents a candidate solution, with "speed" determined by fitness function values (lower fitness = faster horse in minimization problems)
   - Implementing five dynamic competition scenarios to guide search direction
2. Mathematical Formulation
   1. Population Representation

      Tianji's Horses:

      $$X_T = \begin{bmatrix} x_{T1} \\ \vdots \\ x_{T_i} \\ \vdots \\ x_{T_n} \end{bmatrix} = \begin{bmatrix} x_{T1}^1 & x_{T1}^j & x_{T1}^d \\ \vdots & \vdots & \vdots \\ x_{T_i}^1 & x_{T_i}^j & x_{T_i}^d \\ \vdots & \vdots & \vdots \\ x_{T_n}^1 & x_{T_n}^j & x_{T_n}^d \end{bmatrix}$$

      King's Horses:

      $$X_K = \begin{bmatrix} x_{K1} \\ \vdots \\ x_{K_i} \\ \vdots \\ x_{K_n} \end{bmatrix} = \begin{bmatrix} x_{K1}^1 & x_{K1}^j & x_{K1}^d \\ \vdots & \vdots & \vdots \\ x_{K_i}^1 & x_{K_i}^j & x_{K_i}^d \\ \vdots & \vdots & \vdots \\ x_{K_n}^1 & x_{K_n}^j & x_{K_n}^d \end{bmatrix}$$

   2. Core Competition Strategies
      - Scenario 1: Tianji's Slowest Horse Faster Than King's Slowest Horse

        Tianji's Slowest Horse Update:

        $$\begin{cases} x_{Ts}(t+1) = (p \times x_{Ts}(t) + (1-p) \times x_{Tf}(t) + R \times (x_{Tf}(t) - x_{Ts}(t)) + p \times (\bar{x}_{Tf}(t) - \bar{x} \\ Tsi = Tsi - 1 \end{cases}$$

King's Slowest Horse Update:

$$\begin{cases} v_{Ksi}(t+1) = (p \times x_{Ksi}(t) + (1-p) \times x_{Tsi}(t) + R \times (x_{Tsi}(t) - x_{Ksi}(t) + p \times (\bar{x}_{Tf}(t) - \\ Ksi = Ksi - 1 \end{cases}$$

- Scenario 2: Tianji's Slowest Horse Slower Than King's Slowest Horse
  Tianji's Slowest Horse Update:

$$\begin{cases} x_{Ts}(t+1) = (p \times x_{Ts}(t) + (1-p) \times x_{Tr1}(t) + R \times (x_{Tr1}(t) - x_{Ts}(t) + p \times (\bar{x}_{T}(t) - \bar{x} \\ Tsi = Tsi - 1 \end{cases}$$

  King's Fastest Horse Update:

$$\begin{cases} x_{Kf}(t+1) = (p \times x_{Kf}(t) + (1-p) \times x_{Kf}(t) + R \times (x_{Kf}(t) - x_{Kf}(t) + p \times (\bar{x}_{T}(t) - \bar{x} \\ Kfi = Kfi + 1 \end{cases}$$

- Scenarios 3-5: Other Relative Speed Conditions
  Each scenario implements different matching strategies and update equations to ensure effective balance between exploration and exploitation under various conditions.

3. Training Strategy
   Tianji's Horses Training:

$$v_{Ti}^{j}(t+1) = \begin{cases} x_{Ti}^{j}(t) + L_T \times (x_{Tr4}^{j} - x_{Tr5}^{j}) & \text{if } rand < 0.5 \\ x_{Tf}^{j}(t) + M_T \times (x_{Tf}^{j}(t) - x_{Ti}^{j}(t)) & \text{else} \end{cases}$$

   King's Horses Training:

$$v_{Ki}^{j}(t+1) = \begin{cases} x_{Ki}^{j}(t) + L_K \times (x_{Kr1}^{j} - x_{Kr2}^{j}) & \text{if } rand < 0.5 \\ x_{Kf}^{j}(t) + M_K \times (x_{Kf}^{j}(t) - x_{Ki}^{j}(t)) & \text{else} \end{cases}$$

Results

The results after selecting the algorithm and executing the program are depicted below. There is more information in Section Demo.

1. CCO
   The figure demonstrates that the average consumed time for each UAV is $(3.89 + 3.25 + 3.76)/3 = 3.63$s.

```
>>> Constructing the urban environment ...
Succeed in creating the environment!
  - Map size: 1000 x 1000 x 180 m
>>> Start solving...
    Algoithm: CCO

--- Planning UAV-1 (Medical) ---
    ✓ Done! Time consumption: 3.89 s; Final score: 1135.15
--- Planning UAV-2 (Delivery) ---
    ✓ Done! Time consumption: 3.25 s; Final score: 6279.18
--- Planning UAV-3 (Patrol) ---
    ✓ Done! Time consumption: 3.76 s; Final score: 1083.49
```

2. GGO

   The figure demonstrates that the average consumed time for each UAV is $(3.06 + 3.03 + 3.09)/3 = 3.06$s.

```
>>> Constructing the urban environment ...
Succeed in creating the environment!
  - Map size: 1000 x 1000 x 180 m
>>> Start solving...
    Algoithm: GGO

--- Planning UAV-1 (Medical) ---
GGO Algorithm completed. Best Score = 1.176537e+03
    ✓ Done! Time consumption: 3.06 s; Final score: 1176.54
--- Planning UAV-2 (Delivery) ---
GGO Algorithm completed. Best Score = 6.279234e+03
    ✓ Done! Time consumption: 3.03 s; Final score: 6279.23
--- Planning UAV-3 (Patrol) ---
GGO Algorithm completed. Best Score = 1.093927e+03
    ✓ Done! Time consumption: 3.09 s; Final score: 1093.93
```

3. THRO

   The figure demonstrates that the average consumed time for each UAV is $(8.32 + 8.25 + 8.69)/3 = 8.42$s.

```
>>> Constructing the urban environment ...
Succeed in creating the environment!
  - Map size: 1000 x 1000 x 180 m
>>> Start solving...
    Algoithm: THRO

--- Planning UAV-1 (Medical) ---
    ✓ Done! Time consumption: 8.32 s; Final score: 1170.37
--- Planning UAV-2 (Delivery) ---
    ✓ Done! Time consumption: 8.25 s; Final score: 6281.97
--- Planning UAV-3 (Patrol) ---
    ✓ Done! Time consumption: 8.69 s; Final score: 1075.88
```

Limitations and Challenges

Although we have completed all of the *Suggested Steps* in official notification, i.e.,

> 1. Become familiar with MATLAB®, Simulink®, UAV Toolbox, Sensor Fusion and Tracking Toolbox™ and Optimization Toolbox™ using resources listed in the background material section below.
> 2. Set up a cuboid scenario simulation that includes multiple static obstacles, like an urban environment, using UAV Toolbox.
> 3. evelop 3D path planning algorithm using UAV Toolbox for collision-free drone flight. Take advantage of path planning resources for single drone.
> 4. Extend the path-planning algorithm to multiple drones. It will require the centralized tracking of all the drones with information about their positions and velocities, to continuously provide collision free guidance, you can use the ground truth data from the simulated drones to start with. Learn about centralized tracking from examples mentioned in the Background Material Section.
> 5. Test the algorithm in a cuboid scenario environment with multiple drone flights.

yet we haven't combined our algorithms with *Advance Work*. However, we have already considered the second advice in *Advanced Work*, i.e.:

> Develop a task planning algorithm that considers planning pickups, and delivery tasks, and allotting them to appropriate drones. Further, optimize this process using the Optimization toolbox.

We have sought considerable reviews and essays about Multi-Agent Reinforcement Learning (MARL) without only focusing on bio-inspired algorithms, and we intend to make progress on this field to tackle *Multi-UAV Path Planning for Urban Air Mobility* problem.

Moreover, we have attempted to integrate MATLAB® App Designer into our codes and develop an interactive app for users, which enables them to modify parameters and select algorithms with UI components. It is a pity that we haven't finished it yet, so we committed the version without any app design as a result of the deadline.

Model Setup

Required Environment: `MATLAB 2023b`
Required Toolbox: None
How to run: Simply run `main.m` and the MATLAB command prompt will output running process and final results, and afterwards there will be two plots created. When it comes to changing the bio-inspired algorithm, it is suggested that you choose an optional algorithm written in `main.m` comments (`main.m` Line 22-24) and modify the string variable `ALGORITHM_OPTION` (`main.m` Line 25), which is also shown in the following figure.

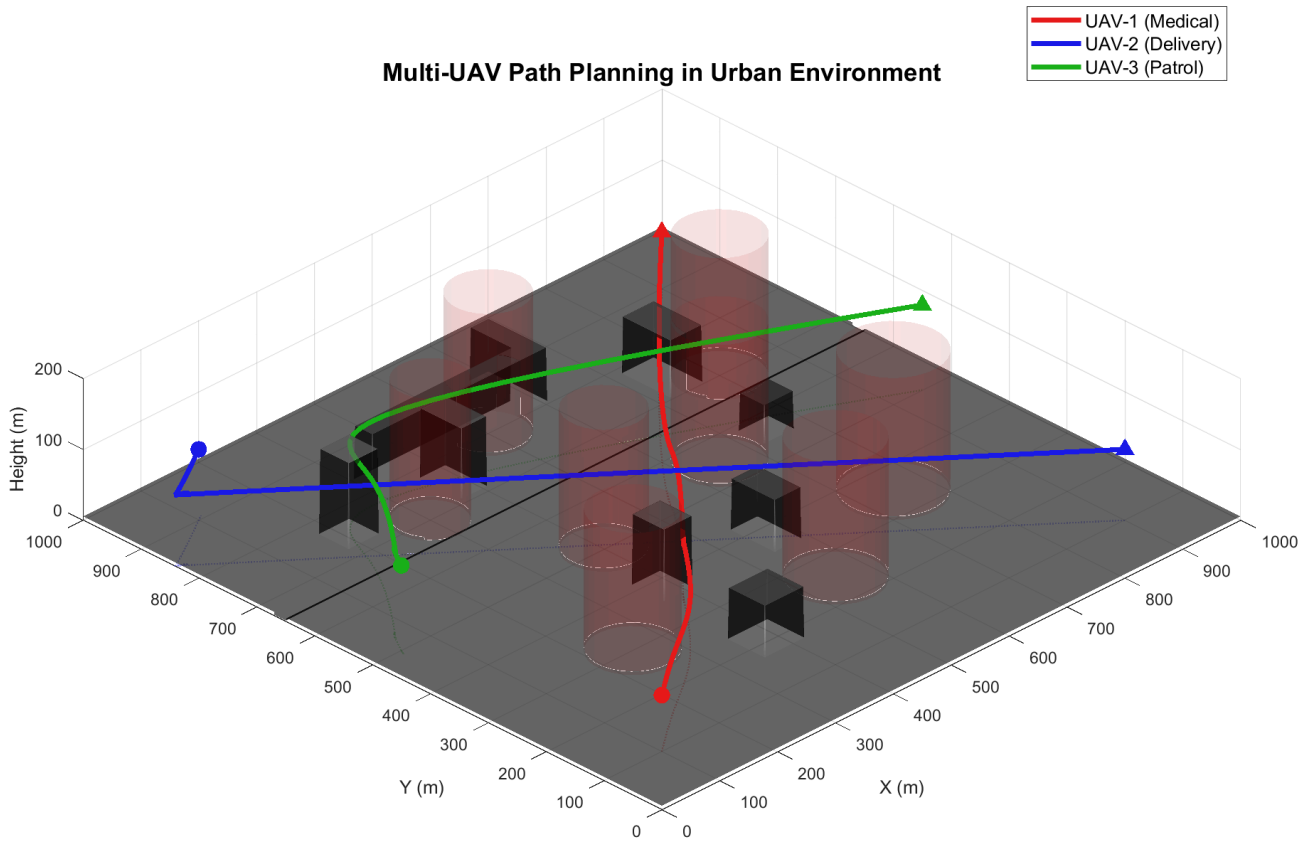
```
% Select one of the algorithms as the solver
% Algorithm options: 'THRO', 'GGO', 'TOC', 'PGA', 'IVY', 'CCO', 'TGCOA'
% More optional algorithms in '.\algorithms' directory
ALGORITHM_OPTION = 'GGO';  % <- Modify this variable to change the algorithm
```

```
22  % Select one of the algorithms as the solver
23  % Algorithm options: 'THRO', 'GGO', 'TOC', 'PGA', 'IVY', 'CCO', 'TGCOA'
24  % More optional algorithms in '.\algorithms' directory
25  ALGORITHM_OPTION = 'GGO';
26
```

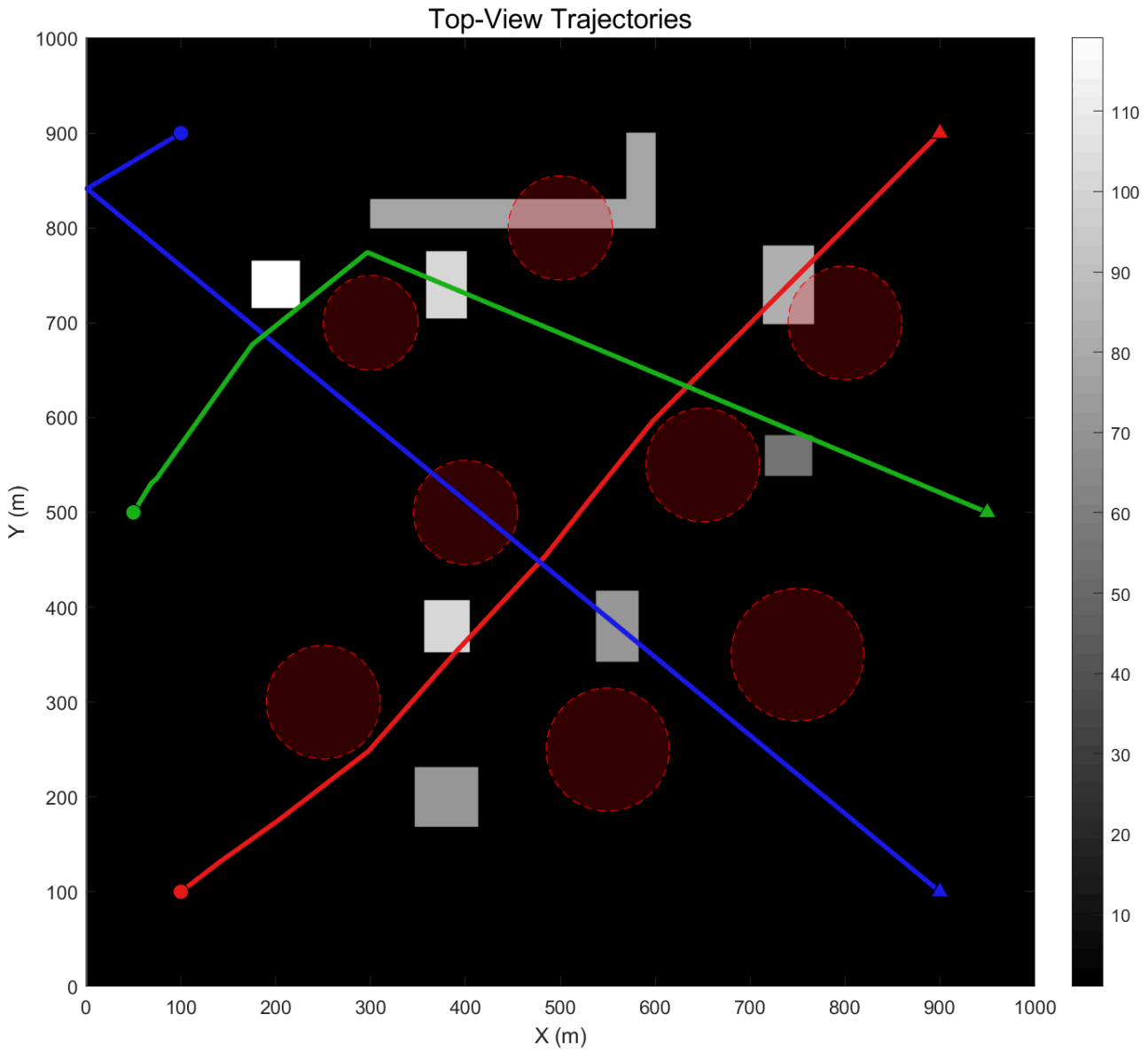
Demo

Running `main.m` in the way mentioned above will create two plots, one of which is the 3D-view trajectories in the urban environment and the other is the top view of the former figure. Here we choose three bio-inspired algorithm (CCO, GGO, THRO) as examples and the executing results are demonstrated below.

CCO:
3D-view:

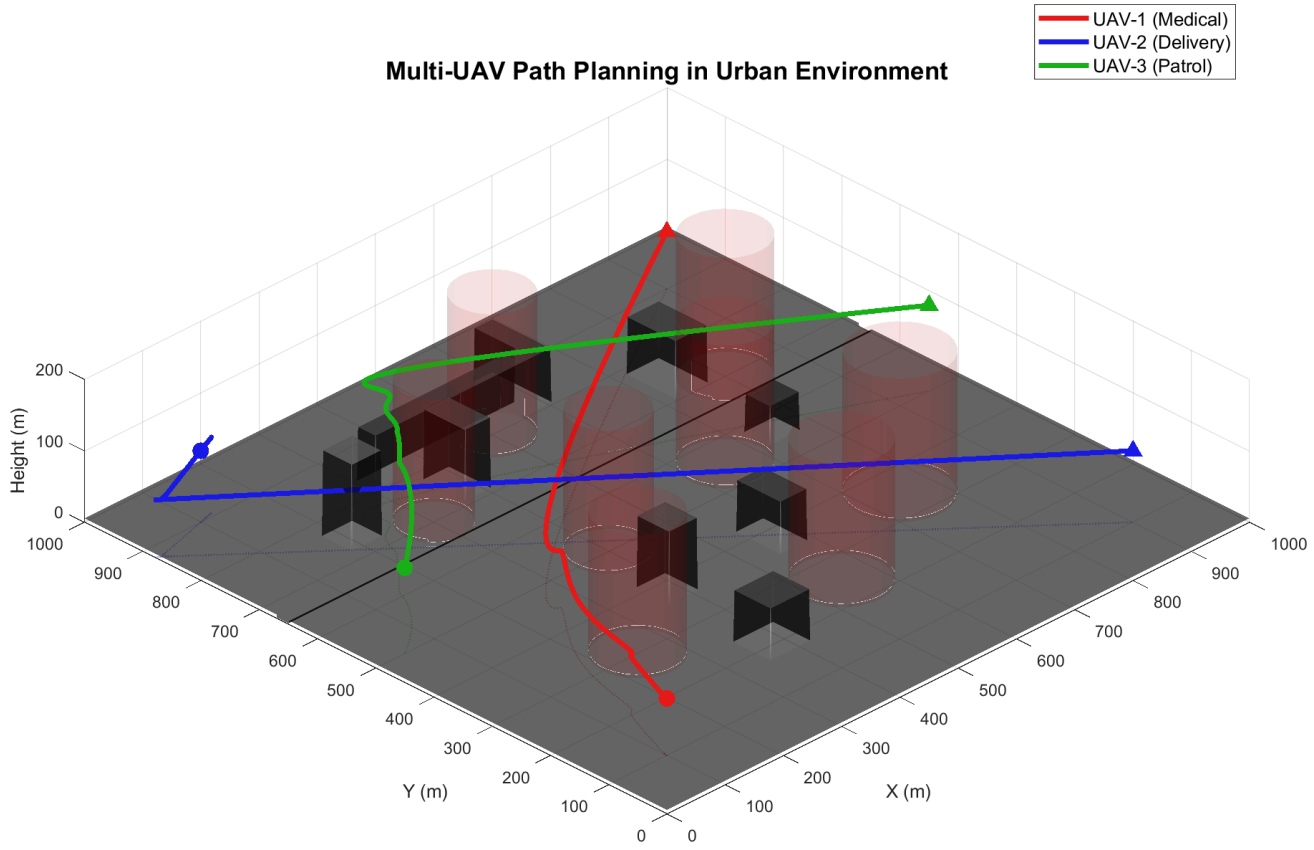

Multi-UAV Path Planning in Urban Environment
UAV-1 (Medical)
UAV-2 (Delivery)
UAV-3 (Patrol)
Height (m)
200
100
0
1000
900
800
700
600
500
400
300
200
100
0
Y (m)
X (m)

Top view:

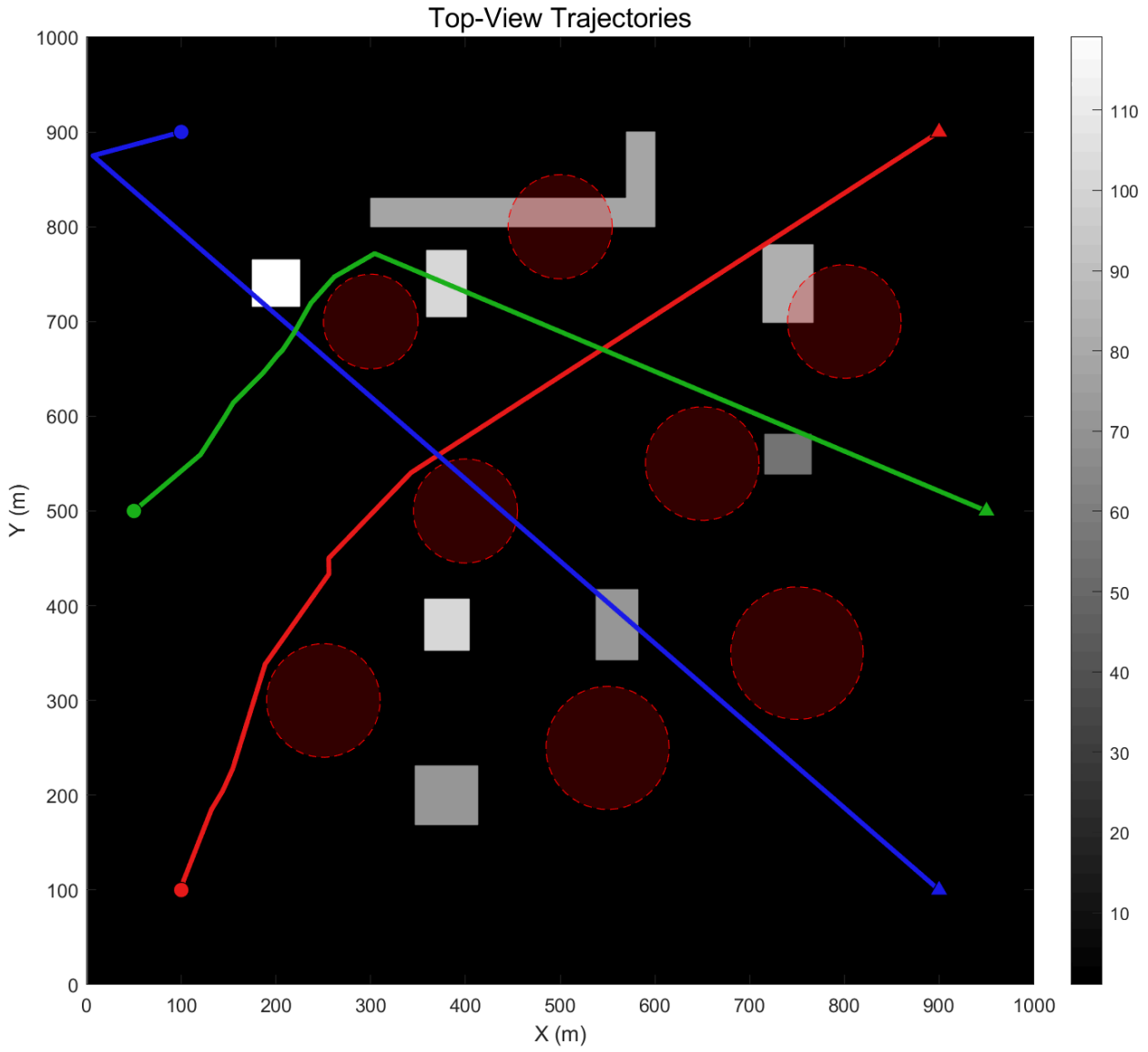

Top-View Trajectories
Y (m)
X (m)
0
100
200
300
400
500
600
700
800
900
1000
10
20
30
40
50
60
70
80
90
100
110

GGO:

3D-view:

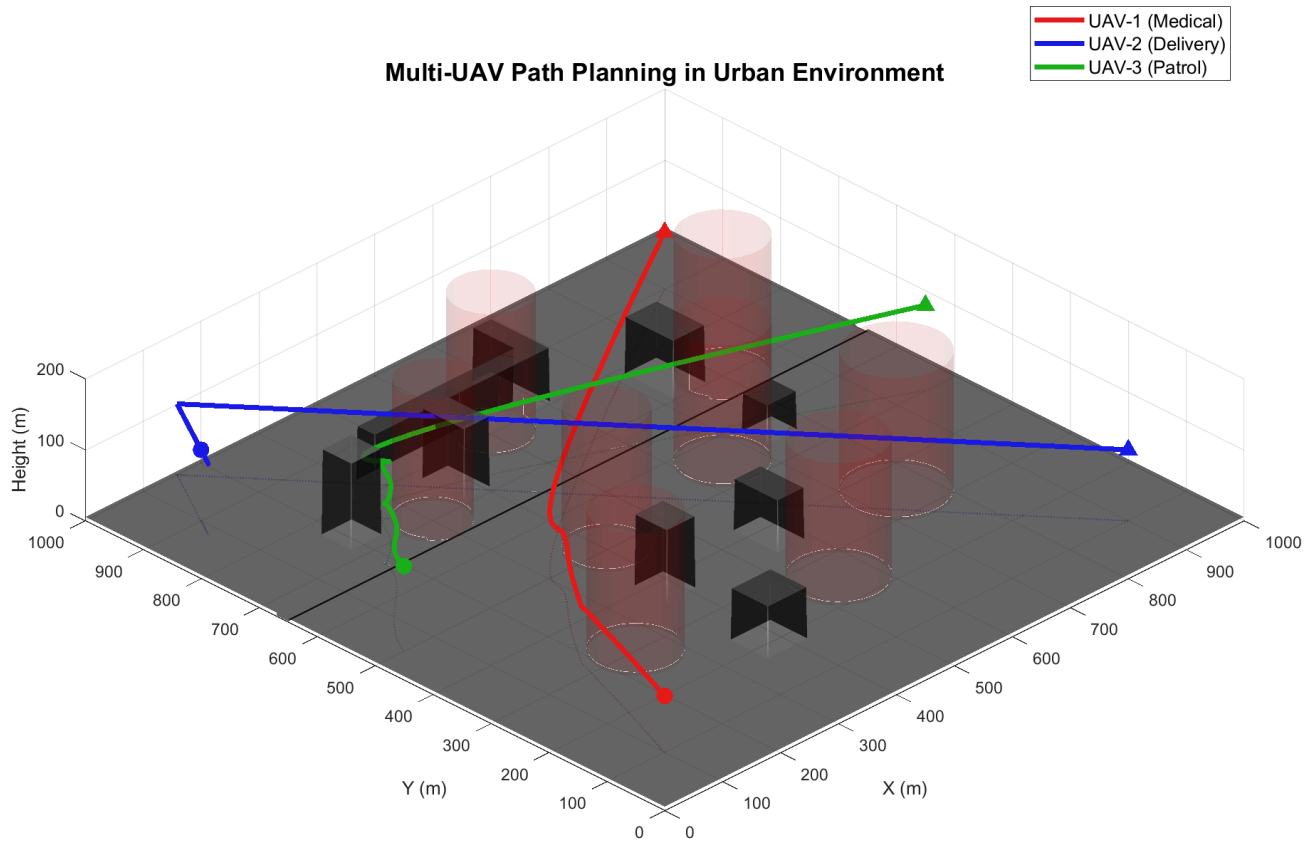

Multi-UAV Path Planning in Urban Environment
UAV-1 (Medical)
UAV-2 (Delivery)
UAV-3 (Patrol)
Height (m)
200
100
0
1000
900
800
700
600
500
400
300
200
100
0
Y (m)
X (m)

Top view:

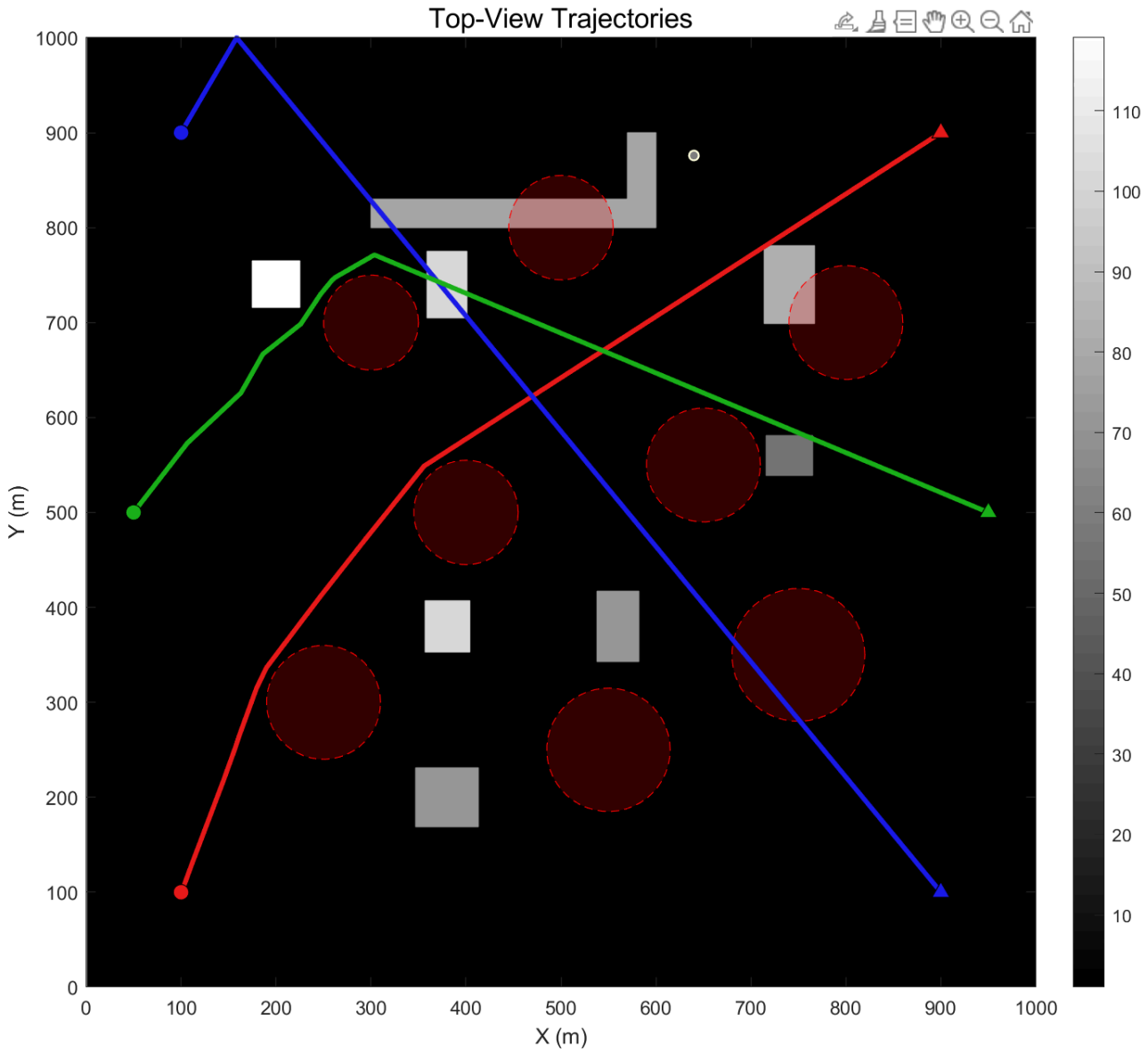

Top-View Trajectories
Y (m)
X (m)
0
100
200
300
400
500
600
700
800
900
1000
10
20
30
40
50
60
70
80
90
100
110

THRO:

3D-view:


Multi-UAV Path Planning in Urban Environment
UAV-1 (Medical)
UAV-2 (Delivery)
UAV-3 (Patrol)
Height (m)
200
100
0
1000
900
800
700
600
500
400
300
200
100
0
Y (m)
0
100
200
300
400
500
600
700
800
900
1000
X (m)

Top view:



Reference

[1] Meng W, Zhang X, Zhou L,et al. Advances in UAV Path Planning: A Comprehensive Review of Methods, Challenges, and Future Directions[J]. DRONES, 2025, 9(5):376.
[2] Wang L, Du H, Zhang Z, et al. Tianji's horse racing optimization (THRO): a new metaheuristic inspired by ancient wisdom and its engineering optimization applications[J]. Artificial Intelligence Review, 2025, 58(9).
[3] El-Kenawy E S M, Khodadadi N, Mirjalili S, et al. Greylag Goose Optimization: Nature-inspired optimization algorithm[J]. Expert Systems with Applications, 238.
[4] Braik M, Al-Hiary H, Alzoubi H, et al. Tornado optimizer with Coriolis force: a novel bio-inspired meta-heuristic algorithm for solving engineering problems[J]. Artificial Intelligence Review, 2025, 58(4).
[5] Bohat V K, Hashim F A, Batra H, et al. Phototropic growth algorithm: A novel metaheuristic inspired from phototropic growth of plants[J]. Knowledge-Based Systems, 2025, 322.
[6] Ghasemi M, Zare M, Trojovsk P, et al. Optimization based on the smart behavior of plants

with its engineering applications: Ivy algorithm[J]. Knowledge-Based Systems, 2024, 295(000):36.
[7] Wang T L, Gu S W, Liu R J, et al. Cuckoo catfish optimizer: a new meta-heuristic optimization algorithm[J]. Artificial Intelligence Review, 2025, 58(10).
[8] Gámez M G M, Vázquez H P. A Novel Swarm Optimization Algorithm Based on Hive Construction by *Tetragonula carbonaria* Builder Bees[J]. Mathematics 2025, 13, 2721.
[9] Ekechi C C, Elfouly T, Alouani A, et al. A Survey on UAV Control with Multi-Agent Reinforcement Learning[J]. DRONES, 2025, 9(7):484.
[10] Christianos F, Schfer L, Albrecht S V. Shared Experience Actor-Critic for Multi-Agent Reinforcement Learning[J]. 2020.