Multi-UAV Cooperative Hunting Using PSO in 3D Cluttered Environment

Shichen Fan[1], Ge Song[2], Chih-Chun Chen[1], Peng Shi[3], and Hugh H-.T. Liu[1]

[1] University of Toronto Institute for Aerospace Studies, Canada,
shichen.fan@mail.utoronto.ca kccchen@gmail.com liu@utias.utoronto.ca
[2] Harbin Engineering University, China,
gesong@hrbeu.edu.cn
[3] School of Electrical and Electronic Engineering, The University of Adelaide, Australia, and College of Engineering and Science, Victoria University, Australia
peng.shi@adelaide.edu.au

Abstract. Cooperative hunting has been an active research field in unmanned aerial vehicles (UAVs) studies. In a three-dimensional (3D) world, the underneath UAV could be influenced by the downwash impact created from the UAV above when a trajectory crossing exists. This paper introduced a solution to realize the cooperative hunting of multi-UAV in a cluttered environment with obstacles. The comparison with a flocking navigated solution with an adaptive height downwash model considered in an obstacle-free environment and the distributed sliding mode position controller solution in a cluttered environment where the target is moving are presented and discussed at the end of the paper. The proposed solution is observed to succeed in cooperative hunting of a moving/static target in a cluttered/obstacle-free environment with faster convergences and shorter trajectories than the other two solutions.

Keywords: cooperative hunting, collision avoidance, pso path planning

1 Introduction

Multi-agent cooperative hunting problem has been an active research field in the past decades. Various solutions have been conducted to solve this problem in different scenarios ([1], [2], [3], [4], [5]). Liao [3] and Furtado [5] found solutions to this problem in two-dimensional (2D) dynamic world. Recently, Chen ([1], [2]) developed a flocking navigated solution to navigate multiple UAVs to corresponding destinations when considering the downwash impact from the UAV above on the underneath UAV when there is a trajectory crossing. This author developed a collision avoidance solution of adaptive height by considering the downwash impact as a cylindrical shape underneath each UAV. Her solution includes a path planning algorithm, agent collision avoidance, and a controller so that each UAV can follow the generated trajectory to its respective target. However, no moving/static obstacles and no moving targets were considered in her solution. Hence, this paper proposes extending a solution to realize the cooperative hunting of multiple UAVs in a dynamic environment by considering

encircling a moving target when there are moving/static obstacles. In the meantime, Song [4] developed a distributed sliding mode position controller to solve the cooperative hunting of multi-UAV in a cluttered environment and hunting a moving/static target in a 3-dimensional world. However, this author did not address time-optimal hunting since she applied leader-follower strategy and was based on predictions of the other agents. Optimal trajectory or shortest time to hunt is essential hence it is necessary to develop a solution that can achieve these goals. The developed solution will be compared with this sliding mode position controller as well.

A solution solving cooperative hunting in a 3D environment where the downwash effect exists has yet to be developed. Therefore, this paper proposes a solution to solve this problem by generating optimal trajectories for the hunter agents to hunt and encircle a moving target while avoiding the downwash impact and pathing around the moving or static obstacles in the environment. This paper first discusses the two existing solutions which the proposed solution is about to be compared with in this section and presents the proposed solution in detail in the Methodology section. Then Section 3 includes an implementation of the solution and comparisons with those two existing solutions mentioned above in different scenarios and followed by a conclusion.

2 Methodology

This solution consists of four main components, including vertices assignment, path planning algorithm, collision-avoidance algorithm, and a controller so that hunter agents can follow the computed trajectories. This section introduces these four components separately in detail.

2.1 Vertices Assignment

First, the Hungarian algorithm is added to assign vertices to the agents to minimize trajectory crossing. This algorithm is a combinatorial optimization algorithm that can solve the assignment problem in a polynomial time. The idea of platonic solids is adopted to determine the vertices for hunter agents so that the agents evenly form a platonic solid around the target. In 4-agents case, a tetrahedron centred at the target position (x_t, y_t, z_t) is selected. Figure 1 presents an illustration of the relationship between selected vertices (labelled by circles in orange) and the target position (labelled by the circle in green).

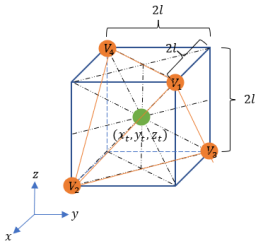


Fig. 1: Vertices Selection in 3-dimensional

The Cartesian coordinates of the vertices in inertial space are:

$$\begin{aligned} V_1 &= (x_t + l, y_t + l, z_t + l), \\ V_2 &= (x_t + l, y_t - l, z_t - l), \\ V_3 &= (x_t - l, y_t + l, z_t - l), \\ V_4 &= (x_t - l, y_t - l, z_t + l). \end{aligned} \tag{1}$$

Hungarian assignment is used here to determine hunter agents' corresponding target locations to minimize the trajectory crossing phenomenon. The cost matrix is described in the equation below. a_i and v_j are the unit position vector of agent i and vertex j in platonic solids that agents can potentially form. The optimal assignment is to minimize the cost of matching agent i and vertex j. θ_{ij} stands for the angle between a_i and v_j.

$$\mathbf{C} = \begin{bmatrix} a_1 \cdot v_1 & a_1 \cdot v_2 & a_1 \cdot v_3 & a_1 \cdot v_4 \\ a_2 \cdot v_1 & a_2 \cdot v_2 & a_2 \cdot v_3 & a_2 \cdot v_4 \\ a_3 \cdot v_1 & a_3 \cdot v_2 & a_3 \cdot v_3 & a_3 \cdot v_4 \\ a_4 \cdot v_1 & a_4 \cdot v_2 & a_4 \cdot v_3 & a_4 \cdot v_4 \end{bmatrix} = \begin{bmatrix} cos(\theta_{11}) & cos(\theta_{12}) & cos(\theta_{13}) & cos(\theta_{14}) \\ cos(\theta_{21}) & cos(\theta_{22}) & cos(\theta_{23}) & cos(\theta_{24}) \\ cos(\theta_{31}) & cos(\theta_{32}) & cos(\theta_{33}) & cos(\theta_{34}) \\ cos(\theta_{41}) & cos(\theta_{42}) & cos(\theta_{43}) & cos(\theta_{44}) \end{bmatrix} \tag{2}$$

2.2 Design of Path Planning Algorithm

Particle Swarm Optimization (PSO) path planning algorithm is adopted to optimize trajectory and be of the shortest. This path planning algorithm is a population-based stochastic optimization technique that solves problems in an optimal way based on the movement and involution of swarms.

$$\mathbf{V_i}(t+1) = \omega \mathbf{V_i}(t) + c_1 r_1(\mathbf{pbest(i,t)} - \mathbf{P_i}(t)) + c_2 r_2(\mathbf{gbest(t)} - \mathbf{P_i}(t)) \tag{3}$$

$$\mathbf{P_i}(t+1) = \mathbf{P_i}(t) + \mathbf{V_i}(t+1) \tag{4}$$

$\mathbf{V_i}$ and $\mathbf{P_i}$ are the velocity and position of agent i. r_1,r_2, used to avoid premature convergences, are random numbers selected from [0,1]. c_1 weighs the importance of the particle's own previous experiences. c_2 weighs the importance of the global learning of the swarm. ω is a value chosen from [0,1].A lower ω means an increased chance of finding a global optimum but more time-consuming.

Several intermediate points are selected between the goal locations and the agent's current position. Within a range from each intermediate point, a number of random agents are generated to simulate swarms. Initialize a list of trajectories by connecting the start location, one of the randomly generated points at each intermediate point, and the goal location. Then by updating each agent's position and velocity to realize the generation towards the optimal trajectory for each UAV.

A cost function is created to find the optimal trajectory of a short length and the least obstacle collision possibility. The cost of violation is calculated based on the internal distance between agent i and every obstacle k located within agent i's sensing range d_{ith_agent} in Eq.6 . $\mathbf{L_{trajectory}}$ stands for the trajectory lengths of hunter agents. β is a constant number used to exaggerate the influence of $\mathbf{Cost_{violation}}$.

$$\mathbf{TotalCost} = \mathbf{L_{trajectory}} + \beta * \mathbf{Cost_{violation}} \tag{5}$$

$$Cost_{violation}(i) = \sum_{k=1}^{n_o} mean(max\{1 - \frac{d_i}{r_k}, 0\}), i = 1, 2, ...n_a \tag{6}$$

A pseudo-code describing how this path planning algorithm works is here:

Algorithm 1. **Path Generation**

Input: agent i's current **position**(**i**) and its goal position **T**(**i**), and obstacles' information

Output: agent i's trajectory

Initialization

for $nPop \leftarrow 1$ **to** Population size **do**

1: Draw straight lines between the agent and its goal location and pick equally spaced intermediate waypoints on the straight lines

2: Generate a random swarm around each intermediate waypoint and initialize its velocity to be zero

3: Based on **position**(**i**), swarm around every intermediate waypoint and **T**(**i**), interpolate a 1-D cubic function

4: Calculate $TotalCost(i)$ using Equation 5. Marked down $Violation(i)$

5: Less total cost means better solution. Update LocalBest(i) and GlobalBest

Main Loop

for $it \leftarrow 1$ **to** Max Iteration **do**

1: Update swarms' positions and velocities using equation 4 and equation 3

2: Update position bounds and velocity bounds from velocity limits and position limits

3: Calculate $TotalCost(i)$ using Equation 5. Marked down $Violation(i)$. Update $LocalBest(i)$ and $GlobalBest$

4: if $Cost_{Violation}(i) = 0$, Break and output the GlobalBest trajectory

2.3 Collision-avoidance algorithm

A collision avoidance controller is developed to solve the downwash impact existing in 3D cooperative hunting. This developed controller is graphically presented

in Fig.2, with the cylinders in grey and blue representing the regions where this collision avoidance controller comes into effect, R_{c_i}, and the yellow cylinders representing downwash impacts. When the grey and blue cylinder are invaded by each other, this collision-avoidance algorithm detects xy-conflicts when agents within r_{sens} on a xy-horizontal plane and z-conflicts when agents within h_{sens} in altitude, then repulsive forces are added to agents themselves to repel from the other agent.

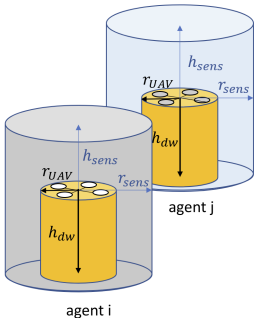


Fig. 2: Reserved cylinder to detect xy-conflict or z-conflict

The developed controller is as follows: Each α-agent applies a control input that consists of two terms as presented in Eq.7. $\mathbf{q}, \mathbf{p}$ are agent's position and velocity. c_1 and c_2 are scaling factors for the position and the velocity differences. $\|.\|_\sigma$ stands for saturated distance norm.

$$\mathbf{u_i} = \mathbf{f_i^g} + \mathbf{f_i^d} = c_1 \sum_{j \in N_i} (\phi_\alpha(\|\mathbf{q_j} - \mathbf{q_i}\|_\sigma) \times \mathbf{n_{i,j}}) + c_2 \sum_{j \in N_i} (b_{i,j}(\mathbf{q_j}, \mathbf{q_i}) \times (\mathbf{p_j} - \mathbf{p_i})) \quad (7)$$

where equation of $\phi_\alpha(\Delta p)$ is included below.

$$\phi_\alpha(\mathbf{\Delta p}) = \rho(\frac{\mathbf{\Delta p}}{d_\beta}) \times (\sigma(\mathbf{\Delta p} - \mathbf{d}_\beta) - 1) \quad (8)$$

$$\sigma(\mathbf{z}) = \frac{\mathbf{z}}{\sqrt{1 + \mathbf{z}^2}} \quad (9)$$

$\mathbf{f_i^g}$ is a gradient-based term that enables agent i to avoid its neighbouring agents j. This is done by creating a repulsive force determined by position differences, pointing from the neighbouring agents j to agent i.

$\mathbf{f_i^d}$ is a velocity consensus term that acts as a damping force to maintain agent i in the same direction as agents j.

$\mathbf{d}_\beta$ is the distance at which both the gradient-based term and the velocity consensus term become to 0.

$\mathbf{n_{i,j}}$ is the unit vector along the line connecting $\mathbf{q_i}$ to $\mathbf{q_j}$. $b_{i,j}(\mathbf{q_j}, \mathbf{q_i})$ is the heterogeneous adjacency term.

$$b_{i,j}(\mathbf{q_j}, \mathbf{q_i}) = \rho(\frac{\|\mathbf{q_j} - \mathbf{q_i}\|_\sigma}{d_\beta}) \quad (10)$$

$$\|\mathbf{z}\|_\sigma = \frac{\sqrt{1+\epsilon\times\|\mathbf{z}\|^2}-1}{\epsilon} \tag{11}$$

In order to create a smooth potential function with finite cut-offs and smooth adjacency matrices, bump function $\rho(\mathbf{\Delta p})$ is developed as follow:

$$\rho_{(\Delta p_{hor})} = \begin{cases} 1 & \Delta p_{hor} \in [0, r_{bet}), \\ \frac{1}{2}\times[1+\cos(\pi\times\frac{\Delta p_{hor}-r_{bet}}{1-r_{bet}})] & \Delta p_{hor} \in [r_{bet}, 1], \\ 0 & \text{otherwise.} \end{cases} \tag{12}$$

$$\rho_{(\Delta p_{ver})} = \begin{cases} 1 & \Delta p_{ver} \in [0, h_{bet}), \\ \frac{1}{2}\times[1+\cos(\pi\times\frac{\Delta p_{ver}-h_{bet}}{1-h_{bet}})] & \Delta p_{ver} \in [h_{bet}, 1], \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

r_{bet} and h_{bet} are real numbers between 0 and 1.

A pseudo-code for the modified collision avoidance is as follows:

Algorithm 2. **Collision Avoidance**

1: Monitor positions and velocities of agent i and its surrounding agents j.
2: Apply repulsive force to agent i when an agent j is within the range of cylinder $\mathbf{R_{c_i}}$ of agent i.
3: If xy-conflict is detected,
then agent i moves away from agent j horizontally
4: If z-conflict is detected,
then agent i moves away from agent j vertically

A flow chart summarizes the path planning algorithm with collision-avoidance algorithm and Hungarian vertices assignment algorithm.

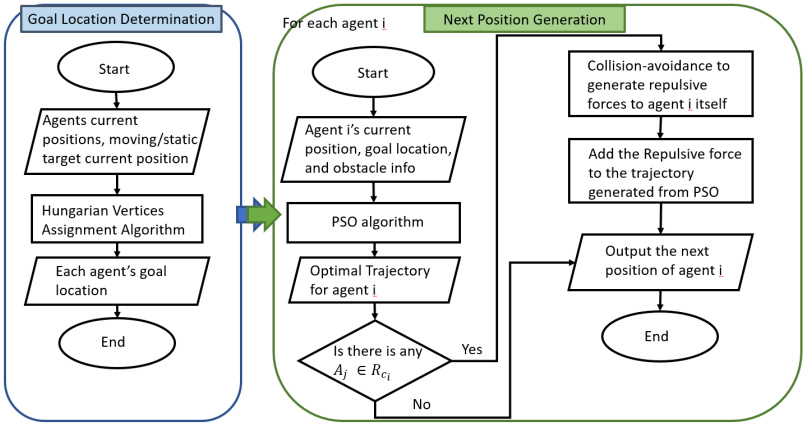


Fig. 3: Flow Chart for the Trajectory Generation with Collision-avoidance Algorithm and Hungarian Vertices Assignment

2.4 Controller

A cascade PI controller combined with the PX4 native onboard controller is used here. The cascade PI controller acts as an outer loop controller to yield desired angles and thrust by sending its outputs to the inner loop attitude control. The inputs for the position controller are the differences between hunter agents' positions and its desired positions ($\mathbf{x}$) and velocities ($\mathbf{v}$), and the controller outputs $\mathbf{u}$, consists of pitch angles (θ), roll angles (ϕ), and thrust force (T).

$$\mathbf{e} = \mathbf{K_{p_x}}\mathbf{x} + \mathbf{v} \tag{14}$$

$$\mathbf{u} = \begin{bmatrix} \theta \\ \phi \\ T \end{bmatrix} = -\mathbf{K_{p_v}}\mathbf{e} - \mathbf{K_{I_v}} \int \mathbf{e}\, dt \tag{15}$$

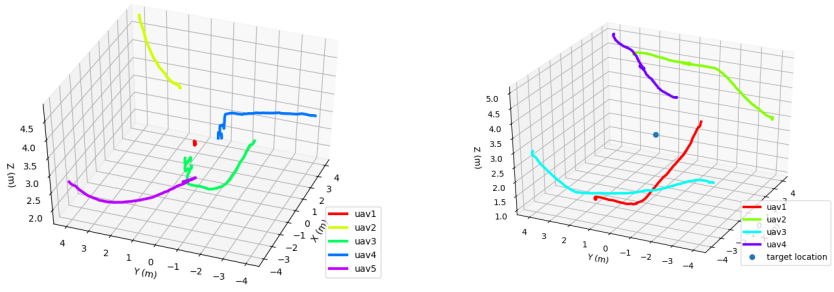
$$\mathbf{u} = -\mathbf{K_{p_x}}\mathbf{K_{I_v}} \int \mathbf{x}\, dt - (\mathbf{K_{p_x}}\mathbf{K_{p_v}} + \mathbf{K_{I_v}})\mathbf{x} - \mathbf{K_{p_v}}\mathbf{v} \tag{16}$$

$\mathbf{K_{P_x}}$, $\mathbf{K_{P_v}}$, and $\mathbf{K_{I_v}}$ are gains matrices and their values are tuned to gain a good performance. Their values are set to be the same as the gain values used in Chen's thesis [2] to better compare the proposed path planner with Chen's flocking-based path planner.

3 Implementation and Comparisons

This section has four hunter agents who cooperatively hunt a target in different scenarios in a Gazebo simulation environment to compare the performance of the proposed solution with the other solutions as mentioned in the section of Introduction. In particular, the flocking navigated solution is applied to cooperative hunting in an obstacle-free environment with a static target. Also, the distributed sliding mode position controller is applied to cooperative hunting in a cluttered environment with a moving/static target. In particular, these simulation results are compared in three cases, including cooperative hunting of a static target in the obstacle-free environment, cooperative hunting of a static target when there is a static obstacle in the environment, cooperative hunting of a moving target in an obstacle-free environment.

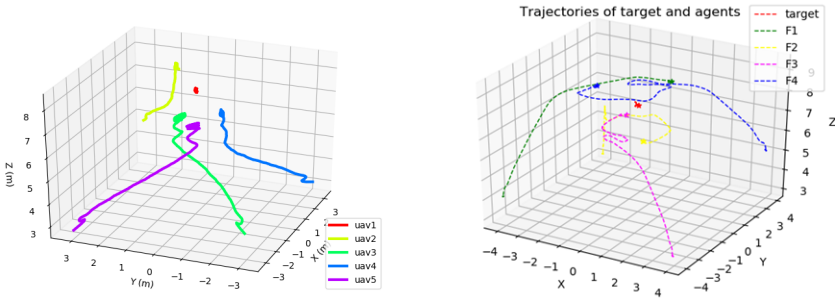
To make the simulations comparable, for every scenario, we set the same velocity limits ($V_{max} = \pm 5.0 m/s$), the same acceleration limits ($a_{max} = \pm 5.0 m/s^2$), and the same time interval ($dt = 0.2 sec$).



(a) Proposed Solution, Total Trajectory Length = 26.30

(b) Flocking Navigated Solution, Total Trajectory Length = 36.00

Fig. 4: Comparison when starting locations are (4,4,5), (4,-4,3), (-4,4,3), (-4,-4,5)m, target hovering at (0,0,3)m, no obstacle

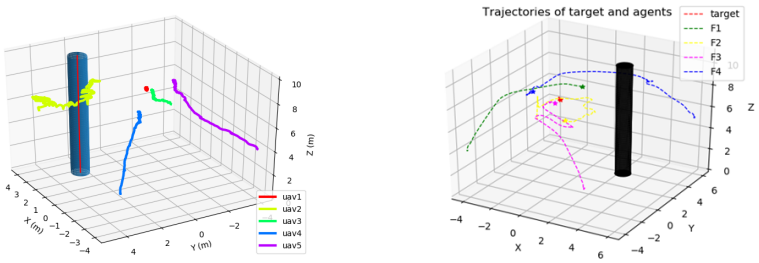


(a) Proposed Solution, Total Trajectory Length = 36.07

(b) Distributed Sliding Mode Position Controller, Total Trajectory Length = 57.02

Fig. 5: Comparison when starting locations are (4,4,5), (4,-4,3), (-4,4,3), (-4,-4,4)m, target hovering at (0,0,8)m, no obstacle

All three solutions succeeded in cooperative hunting when there is no obstacle and target is not moving in Fig.4 and Fig.5. The proposed solution is observed to have shorter total trajectory lengths than the other two existing solutions. Also, this solution has a faster convergence than the distributed sliding mode solution.

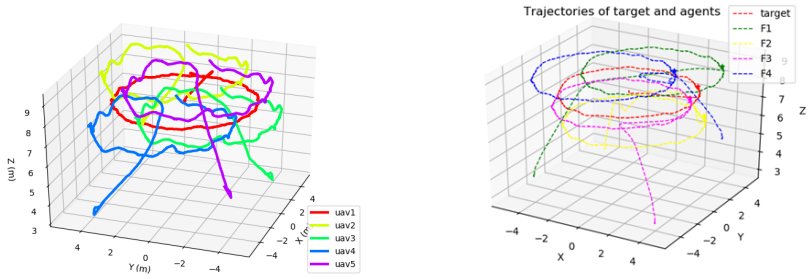


(a) Proposed Solution, Total Trajectory Length = 53.94

(b) Distributed Sliding Mode Position Controller, Total Trajectory Length = 65.51

Fig. 6: Comparison when starting locations are (4,4,5), (4,-4,3), (-4,4,3), (-4,-4,4)m, target hovering at (0,0,8)m, a static cylindrical obstacle of radius=0.5m and height=10m, at location x=2.5m, y=2.5m

Hunter agents path around the obstacle and succeeded in cooperative hunting with both solutions. The same observation as in an obstacle-free environment, the proposed solution has a faster convergence and shorter total trajectory length than the distributed sliding mode solution. Zipzaggings are observed in the proposed solution's trajectory in Fig.6, which is the result of repulsive force from the collision-avoidance algorithm and the attractive force from the PSO path planning algorithm. This phenomenon leads to a future task - developing a solution to smooth trajectories.



(a) Proposed Solution

(b) Distributed Sliding Mode Position Controller

Fig. 7: Comparison when starting locations are (4,4,5), (4,-4,3), (-4,4,3), (-4,-4,4)m, target moving in a 2-dimensional circular path, center at (0,0,8)m, r=4m, v=0.1m/s, no obstacle

When the target begins to move, both solutions are observed to be able to make hunters agents encircle and follow this moving target in Fig.7. Note that the proposed solution generates waver trajectories than the other solution. This phenomenon is because the proposed solution is running vertices assignment

online during the whole process. When the target moves, the relative positions between target and agent keep changing, leading to a potential change of vertices assignment. In the other solution, vertices assignment was done offline before the simulation, leading to the waver trajectories generated by the proposed solution.

4 Conclusion

This paper presents a solution to realize the cooperative hunting of a static/moving target in a cluttered environment with obstacles. The solution helps to assign corresponding target locations to the hunter agents to reduce the downwash impact possibility being caused by a trajectory crossing; meanwhile, this solution navigates the hunter agents to path around obstacles to hunt and encircle target by addressing time-optimal hunting. In addition, this solution includes a cascade PI controller to control the hunter agents to follow the generated trajectories. A comparative study between this proposed solution and Chen's flocking navigated solution is conducted in an obstacle-free environment when the target is static. Another comparative study between this proposed solution and Ge's distributed sliding mode controller is also conducted where the target is static/moving and an obstacle in the environment. Simulation results show that the proposed solution can have hunter agents succeed in cooperative hunting in a cluttered environment and when the target is static or even moving. Compared with the other existing solutions performances, the proposed solution is observed with a faster convergence than the distributed sliding mode solution.

As discussed in the Section 3, one of the future considerations could be developing a solution to smooth trajectories.

References

1. Chen*, C.-C. & Liu, H. H. T. Adaptive Modelling for Downwash Effects in Multi-UAV Path Planning. *Guidance, Navigation and Control,* to appear (Jan. 1, 2022). published.
2. Chen, C.-C. *Practical Implementation of Multi-UAV Flocking Path Planning* (University of Toronto (Canada), 2021).
3. Liao*, J., Liu*, C. & Liu, H. H. T. *Model Predictive Control for Cooperative Hunting in Obstacle Rich and Dynamic Environments* in *IEEE International Conference on Robotics and Automation* (June 1, 2021), 1–8. https://www.flight.utias.utoronto.ca/fsc/wp-content/uploads/2021/05/icra2021jl.mp4%20https://www.flight.utias.utoronto.ca/fsc/wp-content/uploads/2021/05/icra2021jl.pdf. published.
4. Song, G., Peng, S. & W, X. *Distributed target-encirclement control of multiple quadrotors with a leader-follower framework* in *15th International Conference on Innovative Computing, Information and Control* (2021).
5. Furtado, J. & Liu, H. H. T. Multi-UAV Path Planning and Guidance for Cooperative Hunting in a Cluttered Environment. *Journal of Intelligent and Robotic Systems* (Jan. 1, 2019). published.