Heterogeneous Multi-Agent Coordination

Kai Zheng, Zishuo Li

***Abstract*—A significant issue that arises in the automation of many surveillance tasks is that of monitoring the movements of targets navigating in a bounded field of interest. This project is concerned with the effective coordination between autonomous agents to perform foraging tasks with a UAV to constantly reposition. Furthermore, this paper provides optimal trajectories for single UAV to maintain maximal USVs in view or minimal data average time.**

I. INTRODUCTION

Multiagent robotics affords the opportunity to solve problems more efficiently and effectively than any single agent could achieve. An important issue that arises in the automation of many surveillance, security, and reconnaissance tasks is that of monitoring (or observing) the movements of targets navigating in a bounded area of interest. A key research issue in these problems is that of sensor placement determining where sensors should be located to maintain the targets in view [1].

In a number of future civilian and military applications, it is expected that heterogeneous, unmanned vehicles have to be coordinated despite their highly varying dynamics and sensing capabilities [7]. In particular, UAVs are expected to solve such tasks as surveillance, clearing of hostile terrains, transportation in convoys, and the establishment of logistic support chains. At the same time, the UAVs will need to interact with more dynamically capable UAVs and ground vehicles to give them access to large-scale area surveillance, convoy protection, and advanced scouting capabilities. Furthermore, there is a growing need to enable the UAVs to collect static information autonomously without the need for direct human control, which is costly. For example, in urban environments, the field of view of UAV's on-board sensors can get occluded in the presence of tall buildings and/or narrow streets. This can result in areas left uncovered, which might be exploited by the adversary [5]. So, it is necessary to implement a control mechanism that realistically models and explicitly eliminates the effect of occlusions.

This paper is concerned with the effective coordination between USVs to perform foraging tasks with single UAV to constantly reposition. We focus primarily on developing the tracking strategies in order to let the UAV act as a surveillance robot keeping up with the movements of USVs.

We will present an optimal algorithm to maximize the number of monitoring agents in view and minimize the total time in which chases escape observation in response to test results.

II. RELATED WORK

In this section, we provide a survey of relevant research works on heterogeneous multi-agent coordination

[1] developing the distributed control strategies that allow the robot team to attempt to minimize the total time in which targets escape observation by some robot team member in the area of interest. The approach was simulated on a team of four Nomadic Technologies robots. [3] proposed a novel occlusion surveillance algorithm to control the flight of UAVs with on-board cameras so that the coverage and recency of the information about the designated area is maximized. It was conducted within the AGENTFLY framework for flight and air traffic simulation. However, the goal of the algorithm is to find the shortest trajectory travelling all the static buildings and landmarks. Due to limited battery capacity and bounded wireless communication, small-scale UAVs pose fundamental challenges for achieving persistence. [4] proposed an offline path planning algorithm that ensures that the UAVs can always reach the base station to replace their batteries and that each UAV is always connected with the base station via a single or multi-hop link. [5] presents time-optimal trajectories for Unmanned Aerial Vehicles (UAVs) to provide convoy protection to a group of stationary ground vehicles. The UAVs are modelled as Dubins vehicles flying at a constant altitude. Due to kinematic constraints of the UAVs, it is not possible for a single UAV to provide convoy protection indefinitely. The UAS (Unmanned Aircraft System) may effectively extend communication capability to disaster-affected people (who have lost cellular and Internet communication infrastructures on the ground) by quickly constructing a communication relay system among a number of UAVs. [6] proposed a simple but effective dynamic trajectory control algorithm for UAVs to alleviate the congestion at UAVs.

III. APPROACH

A. Assumptions

In order to reduce the complexity of the problem, we make assumptions as follow:

- The UAV sensory system is of limited range, but is available for the entire 360 around the UAV.
- The movements of USVs, i.e., their waypoints, are generated randomly.
- Identical UAV travels at constant velocity at constant altitude along its trajectories.

- The simulating water surface consists only of the base (z = 0) plane.
- The UAV and USVs are always within the range of control.
- The UAV and USVs are treated as holonomic robots.

B. Algorithm

Crowded Condition

The more common situation (shown in Figure 1) is when the distance between USVs is relatively short. The circle in Figure 1 is the coverage of the UAV. The USVs are relatively close to each other and almost all of them are in the UAV' s sensor coverage. In order to keep track of most USVs, we need to find a circle to cover maximum number of USVs. Then, the UAV need to reach the center of the circle whose coverage is optimal at that moment. In addition, we need to repeatedly update that circle using the location and waypoints sent back from the USVs.

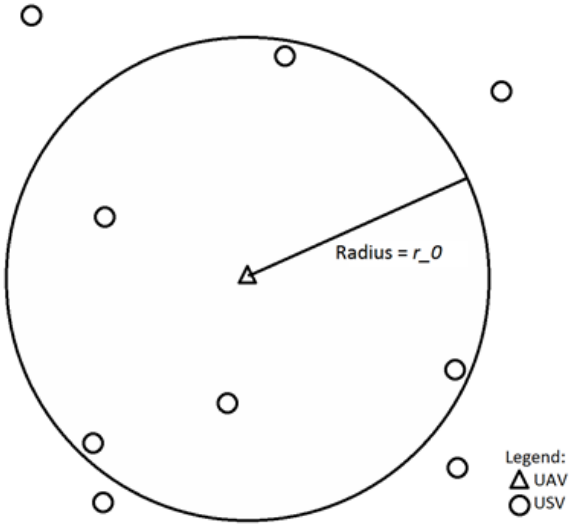

Figure 1 Crowded condition

Since the USVs and UAV are holonomic as assumed, we can consider them as 2D points. The steps of the algorithm are:

1. Construct a circle with radius R that touches each pair of points(USVs) in the given set. There are at most two such circles for each points pair.
2. Count the number of points inside each constructed circle.
3. Find the circle that encloses the maximum number of points.

The pseudocode for this condition is shown below:

```
function move_UAV_crowded(USV_positions)
    UAV_radius= RADIUS
    max_USV_num ← 0
    i ← 0
    while i < USV_numbers - 1 do
        j ← i + 1
        while j < USV_numbers do
            circle_center ← get_circle_center(USV_Positions[i],
USV_Positions[j], UAV_radius)
            if get_USV_in_circle(circle_center) > max_USV_num
                max_USV_num ← get_USV_in_circle
                best_position ← circle_center
            end
        end
    end
    move_UAV_to(best_position);
end
```

The UAV will always move towards the center of the circle, which is constantly updated using the latest positions of USVs. Since The velocity of UAV is much faster than the velocity of USVs, UAV can always reach the ideal position in time. The runtime of this algorithm is O(n^3), because there are at most O(n^2) such circles in step 1, and the linear scan at step 2 takes O(n) time.

Scattered Condition

When it degenerates to the condition (shown in Figure 2) that there is at most only one USV within the surveillance area no matter where the UAV is, the previous algorithm will be invalid. Note that if this condition happens frequently, then it means we probably need more UAVs to execute the surveillance task.

However, we still need to handle this situation just in case. The core concept is to find a convex hull that consist of USVs that haven't been covered for a period of time, and then visited them. Since the UVs are always moving and some USVs can possibly be covered in the meanwhile, the UAV needs to dynamically update the convex hull.

The pseudocode for this condition is shown below:

```
function move_UAV_scattered(USV_positions)
    spiral_points ← empty list
    while USV_positions not empty do
        hull_points ← ConstructConvexHull(USV_positions);
        spiral_points.append(hull_points);
        while spiral_points not empty do
            move_UAV_to(spiral_points.top);
            spiral_points.pop;
        end
        USV_positions.Remove(hull_points);
    end
end
```

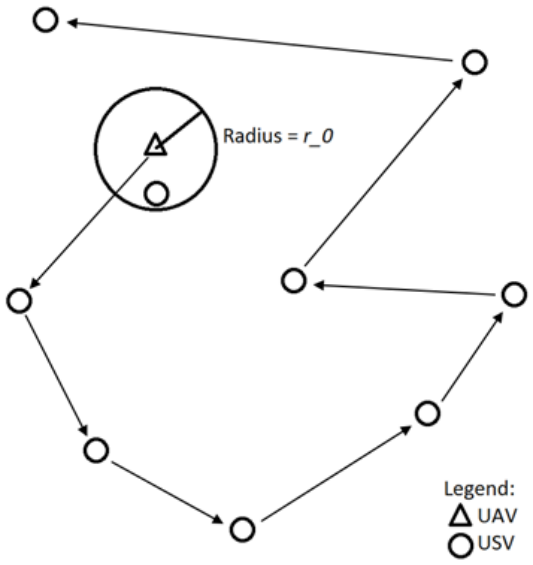

Figure 2 Scattered condition

Additionally, we assumed a boundary between crowded condition and scattered condition. When the maximum USVs under monitor is less than 2, the UAV will call the function "move_UAV_crowded(USV_positions)"; otherwise, the condition will be defined as scattered condition and the function "move_UAV_scattered(USV_positions)" will be executed.

C. Naive Method

There is not any paper presenting an algorithm which can be used in our project. Thus, we set up a naive method for comparison purpose. The naive method is quite straightforward: the UAV will always try to reach the USV that the UAV hasn't covered for longest time.

IV. IMPLEMENTATION

A. Simulator

We used MASON, a fast, easily extensible, discrete-event multi-agent simulation toolkit in Java as our simulation platform. MASON was designed to serve as the basis for a wide range of multiagent simulation tasks ranging from swarm robotics to machine learning to social complexity environments. MASON carefully delineates between model and visualization, allowing models to be dynamically detached from or attached to visualizers, and to change platforms mid-run [2].

MASON is written in three layers: the utility layer, the model layer, and the visualization layer. The utility layer consists of classes which may be used for any purpose. These include a random number generator; data structures more efficient than those provided in the Java distribution; various GUI widgets; and movie and snapshot-generating facilities. Next comes the model layer, a small collection of classes consisting of a discrete event schedule, schedule utilities, and a variety of fields which hold objects and associate them with locations. This code alone is sufficient to write basic simulations running on the command line. The visualization layer permits GUI-based visualization and manipulation of the model [2].

B. Software Architecture

We developed the test program using the MASON framework. The class diagram is shown below.

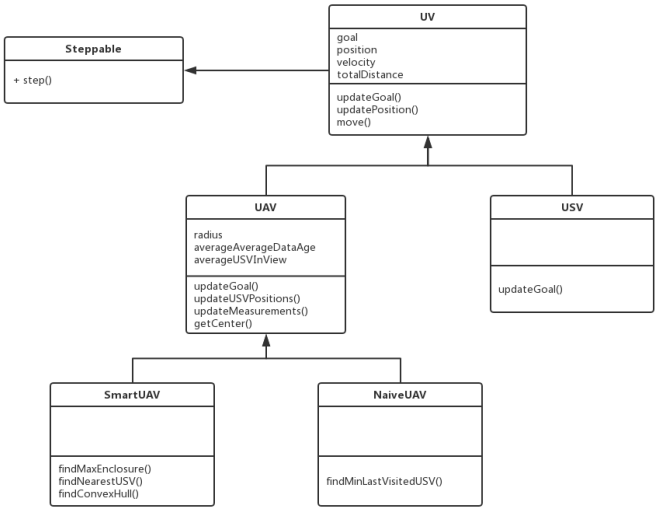

Figure 3 Software Architecture

Class UV is the base class for all unmanned vehicles. It implements the "sim.engine.Steppable" interface by implementing the method "step()". By being the "Steppable" interface, the UV can be placed on the Schedule to have its "step()" method called at various times in the future. This graduate the USV and UAV from being a mere object in the simulation to being something potentially approximating a real agent.

Class UV also implements a "move()" method. All unmanned vehicles are supposed to move in a same pattern given their velocities, as we assume each UV is an ideal vehicle that can be represented as a material point.

Then, class UAV and class USV extend class UV, where the action of each individual UAV and USV is defined. The main difference is how they constantly update their current goal, i.e. the implementation of "updateGoal()" method.

For experimental purpose, we define two UAV classes, class SmartUAV and class NaiveUAV, that both extend class UAV. In class SmartUAV, we implement the intended surveillance algorithms. On the other hand, class NaiveUAV is where we implement the naive surveillance algorithm for comparison use in the experiments.

V. EXPERIMENTS

A. Methodology

To test our algorithm, we simultaneously send out two UAVs, one smart UAV that implements our algorithm and one naive UAV that implements a naive algorithm, to compare their performance in the same surveillance area.

The procedure is: first, we let USVs move randomly in the surveillance area; second, we simultaneously send out the two UAVs and let them keep track of USVs independently; finally, after a fixed period of time, we can compare the two

UAVs' performance through the measurements we will define next.

B. Measurements

Average Number of USVs Under Surveillance

As we say that a UAV is monitoring a USV when a USV is within that UAV's observation sensory field of view. Then, our main goal is to maximize the number of USVs under surveillance. Since we also need to consider time factor, the metric we use is the average number of USVs under surveillance over time.

Average Data Age

The other metric we use is the average data age, i.e. the time since a given USV of the surveillance area has been last covered by the UAV's sensor, averaged over all points in the surveillance area, has been used as the primary measure of performance. Since this metric describes the performance at a specific moment, we also need to consider time factor, i.e. observe how it changes over time.

In the experiment, we record the value of average data age at each time point. When the simulation ends, we calculate their average for measurement.

Total Distance Traveled by UAV

We also want to minimize the movement of UAV to save its energy. So, another metric is the total distance traveled by the UAV.

C. Configuration

The inputs are the environmental setting for the experiments. We define the environment such as its size, simulating time, number of USVs, etc., to make the experiment close to real conditions.

Table 1 Configuration of the simulation

Surveillance Area (km×km)	1×1
Simulating Time (s)	3600
Number of USVs	10

D. Results

We repeat our experiment in the same environment setting for 10 times and collect data in the meanwhile.

Average Number of USVs Under Surveillance

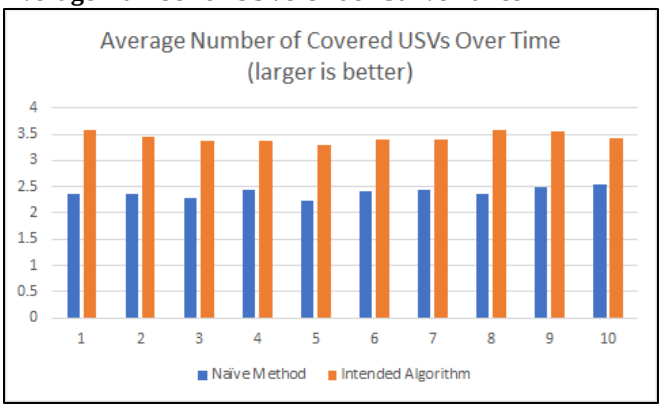

Figure 4 Average Number of Covered USVs Over Time

As we can see, this data indicates our algorithm can averagely track approximate 1 more USV in the 3600 seconds' period. So, our algorithm performs significantly better than naive method in the meaning of "Average Number of covered USVs Over Time".

Average Data Age

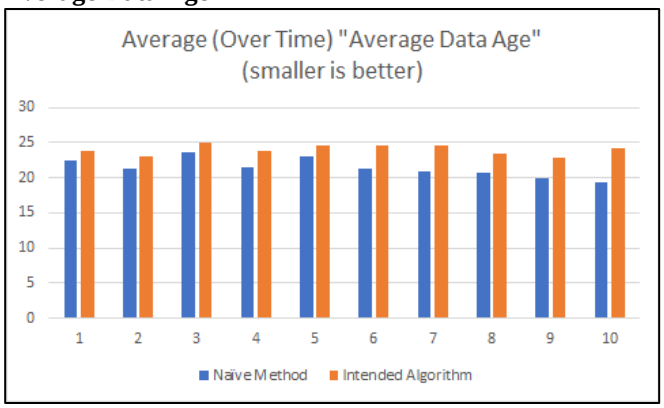

Figure 5 Average (Over Time) "Average Data Age"

However, when it comes to "Average Data Age", it seems that our algorithm is even worse than naive method.

After re-checking the program and analyzing this situation carefully, we think this result is realistic. At most of time, our algorithm tends to keep track of USVs as many as possible (crowded condition). However, there might be some USVs dissociating the major group of USVs. Thus, while the data ages of the USVs inside the surveillance area are all zeroes, the data ages of those dissociating USVs just keep growing. Consequently, they let the "Average Data Age" to be larger and larger.

The root cause is how we measure the goodness of "keep track of multiple USVs". Consider the situation that almost all USVs are in the surveillance area while three USVs are relatively far from the UAV. If the UAV wants to keep USVs in the surveillance area as many as it can, it should stay with the major group; if the UAV wants to cover every USV as frequently as possible, it should leave the major group to reach the dissociating USVs. So, different metrics will lead to different decisions.

Total Distance Traveled by UAV

We also find the "total distance traveled" has very little difference between intended algorithm and naive approach. The most likely reason is, the UAV is always trying to reposition itself to an optimal position, so it's always moving.

VI. Conclusion

In this paper, we have obtained optimal paths for a single UAV so that it provides USVs with efficient surveillance. In addition, our experiment has demonstrated that our algorithm is statistically better than the naive method, if the UAV is intended to cover USVs as many as it can. For the further study, we may improve our algorithm so that it can provide multi-UAV with optimal trajectories.

VII. REFERENCES

[1] L. E. Parker, "Behavior-based cooperative robotics applied to multi-target observation," SERIES IN MACHINE PERCEPTION AND ARTIFICIAL INTELLIGENCE, vol. 27, pp. 356-376, 1997.

[2] A. R. Girard, A. S. Howell, and J. K. Hedrick, "Border patrol and surveillance missions using multiple unmanned air vehicles," in 2004 43rd IEEE Conference on Decision and Control (CDC) (IEEE Cat. No.04CH37601), 2004, vol. 1, pp. 620-625 Vol.1.

[3] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," Simulation, vol. 81, no. 7, pp. 517-527, 2005.

[4] X. C. Ding, A. Rahmani, and M. Egerstedt, "Optimal multi-UAV convoy protection," in 2009 Second International Conference on Robot Communication and Coordination, 2009, pp. 1-6.

[5] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek, "Autonomous UAV Surveillance in Complex Urban Environments," in 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, 2009, vol. 2, pp. 82-85.

[6] Z. M. Fadlullah, D. Takaishi, H. Nishiyama, N. Kato, and R. Miura, "A dynamic trajectory control algorithm for improving the communication throughput and delay in UAV-aided networks," IEEE Network, vol. 30, no. 1, pp. 100-105, 2016.

[7] A. L. Sumalan, D. Popescu, and L. Ichim, "Flood evaluation in critical areas by UAV surveillance," in 2016 8th International Conference on Electronics, Computers and Artificial Intelligence (ECAI), 2016, pp. 1-6.