Cooperative Fire Monitoring using Multiple UAVs

David W. Casbeer
Electrical Engineering
Brigham Young University

Abstract—This paper discusses the possibility of using multiple low-altitude, short endurance (LASE) Unmanned Air Vehicles (UAVs) to cooperatively monitor a forest fire. The UAVs will individually track the fire’s perimeter by means of an on board infrared sensor. Using a decentralized cooperation scheme the UAV team will collect data and upload it to a base station. Simulation results are presented that show the monitoring is effective with dynamic fire perimeters. This approach will give fire fighters the time-critical information needed to safely and effectively fight the fire.

I. INTRODUCTION

Forest fires cause billions of dollars in damage each year. In order to effectively combat forest fires, early detection and continuous tracking are essential. Advances in image processing have made the ability to detect forest fires feasible using satellite images [1], [2]. However due to the orbital period of satellites designated for this purpose, the frequency at which these images are acquired is insufficient for tracking. Because of the difficulty monitoring forest fires, fighters must enter the region with little knowledge of how and where the fire is propagating, placing their lives at risk. For these reasons, there is an urgent need to develop more effective fire monitoring technologies.

The introduction and improvements of Unmanned Air Vehicles (UAVs) has engendered a new approach to fire monitoring. Interest has been shown in High Altitude Long Endurance (HALE) UAVs such as ALTAIR [3] because of their ability to increase image resolution and update rates over those of satellite systems. However their limited availability during peak fire season emphasizes the need for lower-cost, locally implementable systems.

Low Altitude, Short Endurance (LASE) UAVs have the potential to improve the monitoring of forest fires. However, a number of challenges have to be solved before LASE UAVs can be used for fire monitoring. First, with the fire growing and changing directions, UAVs need to be able to plan their path using limited real-time information. Second, LASE UAVs typically cannot carry enough fuel to endure a long fire fighting mission, which means that the UAV must have the intelligence to return to the home base for refueling. Furthermore, for large forest fires, the information update rate may still be too low if only a single LASE UAV is employed. Finally, it is noted that fires generate tremendous heat and turbulence directly above the burning region. Therefore crossing directly over the fire will place low-altitude UAVs at significant risk. As a consequence LASE UAVs are effectively restricted to the air space over the unburned region of the fire.

To ensure the adequate frequency of updates concerning large forest fires, multiple UAVs are necessary. Significant research has been applied to the cooperation of UAV teams (c.f. [4]–[6]). A key challenge in implementing decentralized cooperation strategies is to form consensus among members of the team concerning the mission’s implementation when communication links are intermittent or noisy and sensed information is inconsistent among team members. Recent work on consensus algorithms provide a means for convergence to consistent cooperation information among team members (c.f. [7]–[10]).

This paper will discuss a decentralized cooperation strategy developed to minimize the delay by which fire data is uploaded to a base station. As discussed in [11] the centralized solution is to spread the UAVs equally along the fire’s perimeter. The decentralized solution [12] accomplishes this without global information concerning the location of the forest fire and without full knowledge of UAV team member locations. The monitoring problem is set up in Section II along with an explanation of the metric to be minimized. The decentralized solution is presented in Section III. In Section IV simulation results are presented. To demonstrate the effectiveness of our approach in realistic scenarios, the forest fire propagation model EMBYR (c.f. [11], [13], [14]) has been implemented in Simulink to generate a realistic simulation of the propagation of a typical forest fire. This model is used to verify the cooperation algorithms in a simulated forest fire scenario.

II. PROBLEM STATEMENT

As mentioned in Section I the LASE UAVs will not be able to fly directly over the fire due to the extreme heat and turbulence. Fortunately, the information of essential importance to fighters is the propagation of the perimeter of the fire, enabling them to know if they are in a safe position. The monitoring scenario and the metric we are trying to minimize will be presenting in the sequel.

We assume the UAVs are equipped with infrared cameras and on-board processing ability to adjust their flight paths. This will enable them to track the fire’s perimeter autonomously. A means of following the edge of the fire is given in [11].

A. Monitoring Scenario

In Figure 1 two UAVs are depicted monitoring the perimeter of the forest fire. The objective of the mission is to collect data images from the forest fire perimeter. The location of the fire with the respective images are then uploaded to the base station.
Since the LASE UAV class can only support a small payload, powerful transmitters are not feasible thus decreasing communication distances. For analysis purposes, we assume that the UAVs communicate when they are co-located. Communication and data transfer with the base station will occur as a UAV passes over the line segment connecting the base station with the center of the fire. Location coordinates will be collected continuously while images will be taken frequently enough to ensure a continuous image of the fire edge.

B. Latency Metric

The delay between when the images are collected and when they are transmitted to the base station, can serve as a measure of the quality of the fire monitoring algorithm. Let \( \delta(x, t) \) denote the latency at the base station associated with information about the position \( x \) along the perimeter at time \( t \). As time passes, the information at the base station grows older (more latent) until a UAV arrives to transmit the latest information it has gathered. For a particular position \( x \) along the perimeter, \( \delta(x, t) \) will simply increase with time until it is replaced by the data downloaded from a UAV. Figure 2 gives a typical depiction of latency evolution for a particular point \( x_0 \) on the perimeter of the fire. The vertical edges of the sawtooth waveform represent the transmission of data from the UAV to the base station, while the linearly increasing portion of the waveform represents the increase in latency between UAV updates. The minimum latency \( \delta_{\text{min}} \) corresponds to the flight time between the point of interest and the base station. The maximum latency depends on the total time required to make an observation at \( x_0 \) and to deliver that data to the base station.

Figure 3(a) depicts the latency associated with some point on the perimeter of fire when a UAV flying clockwise arrives back at the base station after traversing the entire circumference. The path’s width denotes the latency of the base station’s update of that point. Since the state of the fire is transmitted only after the UAV has traversed the entire fire perimeter, the latency is largest for data gathered at the beginning of the flight near the base station. Because the UAV is traveling at constant velocity, the latency profile is a linear function of the distance traveled.

Let one UAV be assigned to survey the upper half of the perimeter while a second is assigned to the lower half. If the UAVs depart from the base station simultaneously and fly at the same speed, the update rate will be the same as the single UAV case (since both UAVs arrive back at the base station at the same time), but the latency associated with the information on both sides of the base station will be symmetric and reduced, as can be seen in Figure 3(b).

The latency profile shown in Figure 3(b) is the minimum possible latency for every \( x \) along the perimeter of the fire. To see that this is true, note that the minimum latency (\( \delta_{\text{min}} \)) associated with data gathered at \( x \) on the perimeter is the time needed to travel from \( x \) to the base station along the shortest path. Dividing the perimeter equally between two UAVs ensures that the distance traveled between any point on the perimeter and the base station is minimal. The consequence is that adding more than two UAVs will not improve the minimum latency for any particular point on the perimeter. However, the rate at which updates occur will increase linearly with the number of UAV pairs employed. By increasing frequency of updates, \( \delta_{\text{max}} \) (shown in Figure 2) is decreased. This can be seen by noting \( \delta_{\text{max}} \) is a function of \( \delta_{\text{min}} \) and the time between updates. \( \delta_{\text{min}} \) cannot be made lower, however as the number of UAV increases the time between updates decreases thus lowering \( \delta_{\text{max}} \). To maintain the minimum latency profile
of Figure 3(b) and to maximize the frequency of updates at the base station, UAVs should be distributed equally around the perimeter with each UAV assigned to monitor a segment of length \( P/N \) where \( N \) is the number of UAVs and \( P \) is the perimeter of the fire.

III. DECENTRALIZED LOAD BALANCING

For a fixed perimeter length and a fixed number of UAVs, the minimum latency configuration is when pairs of UAVs are uniformly spread along the perimeter of the fire in both directions (i.e., for every pair of UAVs, one is headed in the clockwise direction and the other in the counter-clockwise direction). Pairs of UAVs will meet, transmit gathered information, and then each UAV will reverse its direction to meet its neighbor in the other direction.

The aim of our distributed algorithm is to converge to this minimum latency configuration. The algorithm must converge for any perimeter size and must re-adjust when the perimeter length or the number of pairs of UAVs changes. By designing the algorithm so that changes in the system parameters are propagated across the team quickly, we will be able to address insertion and deletion of UAVs as well as expansion and contraction of the perimeter.

The fundamental idea is for each UAV to take the action that will allow neighboring pairs to “share” the perimeter between them. When two UAVs meet, each has knowledge of the length just traveled from its previous rendezvous. The sum of these lengths can then be divided equally between them: the UAV that has traveled the least will loiter at the mid-point of this segment to wait for its neighbor the next time the two are to meet.

To illustrate the idea, consider a simple line segment \( \overline{ab} \) with two UAVs tasked to cooperatively gather information along the segment as shown in Figure 4. Let \( \ell_1 \) be the distance traveled by the \( i^{th} \) UAV from the last endpoint visited. In Figure 3, UAV 1 has traveled the least, so after returning to endpoint \( a \), it will travel \( \ell_1 + \ell_2 \) and then begin to loiter. UAV 2 will return to endpoint \( b \) and then reverse direction until it encounters UAV 1. Since UAV 1 traveled the shorter distance it will arrive at the midpoint of \( \overline{ab} \) first, and when both arrive, each will have traveled the same distance (i.e., \( \ell_1 = \ell_2 \)) from the endpoints which implies that the UAV pair has achieved the minimum latency configuration.

Any change in the size of the segment will be tracked since the pair effectively measures the current perimeter length by summing the distances traveled from the endpoints. In other words, since the UAVs only have memory of the state of the perimeter from one previous iteration, a continuous load balancing algorithm will track finite changes in the perimeter.

To enable information on the growth of the perimeter to be accounted for more rapidly, the UAV assigned to wait for its neighbor will loiter at the point a distance \( \frac{\ell_1 + \ell_2}{2} \) from where the endpoint was previously. By measuring the discrepancy of \( \ell_1 \) and the new distance back to the endpoint, UAV 1 can update the loiter distance to negate the effect of the growth in that region.

Adding UAVs to the perimeter is equivalent to stringing perimeter segments together that have changing endpoints: the endpoints for a segment shared by one pair of UAVs are the outside neighbors of these UAVs. Monte-Carlo simulations will be used to verify that pairwise load balancing will lead to team convergence. By balancing the length shared by every pair of UAVs, the team as a whole will spread itself evenly around the perimeter and so achieve the minimum latency configuration. If the algorithm can be shown to converge for arbitrary initial conditions with an arbitrary number of team members, then insertion/deletion can be analyzed by considering the modified system (after the insertion/deletion) with initial conditions given from the state of the original system at the time of the insertion/deletion.

Each UAV implements the following algorithm.

**Load Balancing Algorithm.**

1) Maintain an estimate of the distance traveled from the last rendezvous in each direction.
2) At a rendezvous, the UAV that has traveled the smallest distance since its last rendezvous agrees to loiter at the mid-point of the shared segment the next time it is tracking the perimeter in that direction (clockwise/counter-clockwise).
3) If the endpoint of a segment has changed then the loiter distance is augmented by the change in distance of the endpoint. This keeps the loiter point at the same position relative to the segment length as communicated during the rendezvous.
4) At least one UAV in a rendezvous pair must not loiter en route to the next anticipated rendezvous of this pair. This ensures that pairs of UAVs will always meet again, independent of the change in the perimeter.

IV. RESULTS

**A. Monte Carlo Simulations**

By means of Monte-Carlo simulation the load balancing algorithm is seen to converge to the minimum latency configuration for arbitrary initial conditions. In each simulation \( N \) pairs of UAVs will be launched from the base station at random times around a fixed length circular perimeter. Each member of the team continuously balances the load shared with each of its two neighbors. The simulation continues until all agents are within \( \epsilon \) of the minimum latency configuration or the maximum time is reached.

For each \( N \in \{2, \ldots, 7\} \), 100,000 simulations were performed and the time required to reach steady state recorded. Since time to convergence is a function of the speed of the UAVs and the size of the perimeter, convergence time is normalized by the time required for information to travel around the perimeter. For example, if the convergence time is listed as \( T \), then one UAV could traverse the entire perimeter \( T \) times in the amount of time required for the team to converge (to within \( \epsilon \)) to the minimum latency configuration.
Figure 5 shows the mean and standard deviation in normalized convergence time over the 100,000 iterations for each $N$ with normalized $\epsilon = 0.0003$.

Each simulation instance converged to the minimum latency configuration for every $N \in \{2, \ldots , 7\}$. An insertion or deletion of a pair of UAVs can be represented by a set of initial conditions given by the state of the system before the change with a new value of $N$. Monte-Carlo simulations show that under any initial conditions for any $N$, the stability of the algorithm will not be affected.

Fig. 5. Monte-Carlo results for 100,000 iterations for $N = 2, \ldots , 7$ pairs of UAVs.

Using the EMBYR fire simulation, it can be seen how this load balancing algorithm works on a growing fire. Figure 6 shows four time instances of the EMBYR fire simulation with no wind. The fire is growing starting from Figure 6(a) to 6(g). In Figure 6(a) $N = 6$ UAVs are launched from a base station in the lower left. Figure 6(b) shows the 6 planes along the perimeter of the fire not yet in the minimum latency configuration. In Figures 6(c) and 6(d) the UAVs are monitoring equal length along the fire’s perimeter and the neighboring planes are meeting at the same time which puts them in the minimum latency configuration.

V. Conclusion

The load balancing algorithm tracks an expanding or shrinking perimeter. Monte-Carlo simulations have shown that insertion/deletion of UAV pairs will not affect the stability of the algorithm. The conclusion is that the load balancing algorithm will converge to the minimum latency configuration in the presence of team member insertion/deletion and finite changes in perimeter length.

References


