End-to-End Learning for Fighting Forest Fires (EELFFF)

Ravi Haksar and Adam Caccavale
Department of Mechanical Engineering, Stanford University
{rhaksar,awc11}@stanford.edu

Abstract

Fighting fires is a dangerous task for firefighters, and therefore provides an opportunity for automation. Wildfires happen over large areas, which suggests using multiple firefighting agents is a good approach. A challenge for systems like these is coordination, since as the number of agents increase the computational and communication requirements increases and can overwhelm a single centralized controller. Decentralized systems, where agents have little to no communication and only local information, resolve this issue, though at a cost of sub-optimality. This paper proposes training a convolutional neural network architecture using simple local data to provide control for any number of agents in an environment of any size. Information is limited to what can be collected with no communication (desirable for a scalable system). The data includes an overhead image of a small subregion of the environment, the location of the nearest agent, and the ignition point of the fire. Results show promise for the single agent case, with the agent moving balancing the objectives of fire coverage and limited path lengths. The case with multiple agents does not exhibit the desired level of cooperation suggesting that the gathered information is insufficient for coordination. Future work will focus on determining what minimum level of information is required to achieve cooperation (e.g. information about several nearest agents and potentially their planned paths) as well as different problem formulations.

1. Introduction

Fighting forest fires is an appealing application for robotics as terrain, weather, and other factors can create significant challenges for firefighters – including the potential for loss of life. An attractive solution is to use multi-agent systems, such as a team of quadrotors, to monitor and contain a wildfire. Autonomous aerial vehicles can be more agile, can be disposable, can be deployed in much larger numbers, and can also be distributed – there is no centralized agent gathering information and making decisions, eliminating a single point of failure. Information can also be provided to firefighters to assist their planning, monitoring, and control efforts.

The goal of this work is to use a convolutional neural network architecture to plan the actions for autonomous quadrotor agents. The input data consists of images of the forest as well as a distance measurement to the nearest agent and to the initial ignition point. The output of this network is a series of waypoints which defines the agent's motion trajectory. A single trained network will be used for all agents so that the system scales linearly with the number of agents and so that there is no influence from the size of the forest.

1.1. Related Work

Prior work for this type of application has introduced methods for various aspects of fighting fores fires.

A number of models have been introduced to describe the spread of a wildfire, including elliptical PDE models [10], vector propagation models using spatial data [11] (used by US agencies), and a probabilistic lattice-based model [12].

Several methods have been published on using autonomous agents to surveil and monitor a forest fire. Methods include using computer vision techniques (but not using neural networks) to detect a fire [4][13] or to monitor a fire and relevant data of the fire (e.g. temperature, flame height) [14][7][8][2]. Neural networks have however been used successfully for other multi-robot tasks like collision avoidance with no communication [3]. Most notably, some of the monitoring work is based on systems of autonomous agents, typically centralized. Types of agents used include fixed-wing aircraft, quadrotors, and ground vehicles. Some works, like [5], use both ground and aerial vehicles.

Surprisingly, there are relatively few published methods for control. One method [6] constructs a fully decentralized method using potential field methods, which is common in robotics applications. The agents are able to surround a wildfire and suppress it. Another method [9] uses a centralized formulation, with one agent collect observations, which are then sent to a base station. The station then commands other agents to take actions.

The fire models in literature are roughly comparable and are able to capture the bulk behavior of the fire. A strength of the probabilistic model is that it can also describe other spreading processes (e.g. epidemics). The monitoring methods are typically based on a single agent, which may not be sufficient for large fires and can be prone to failure. These methods are also usually concerned with systems deployed in the field (e.g. fixed sensor stations) that would not be suitable for the control aspect. Finally, it is clear the published control methods rely on classical control methods, which can suffer from modeling or computational issues.

A focus of this work is to augment the classical sensing and control literature for this application with a neural network approach. These control techniques can be combined with hardware that is being developed, like that in [15], to combat wildfires.

2. Methods

2.1. Problem Formulation

A probabilistic discrete-time model from literature [12] is used to represent the spreading of a wildfire. The forest representation is therefore a NxN sized grid of trees, with three possible states for each tree: healthy, on fire, and burned. It is only possible to "treat" trees on fire, which has the effect of reducing the probability that the fire continues to burn. Parameters for this model are chosen so that the wildfire is predicted to burn down at least 90% of the total forest area. Figure 1 shows the fire at various iterations demonstrating how an unconstrained fire will spread. The ignition point is the center of the forest and the wildfire spreads to burn almost the entire forest.

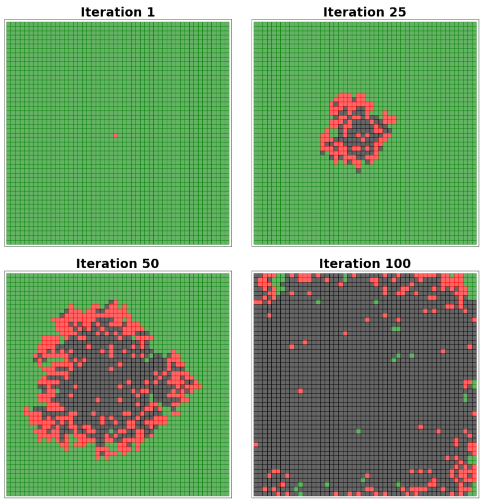

Figure 1: Evolution of fire over time (no control)

There are r quadrotor agents attempting to prevent the forest from burning down. Each agent is equipped with a downward-facing camera that produces an image containing a region of hxw trees. The agents are also equipped with a simple sonar sensor that provides the Euclidean distance to other agents. The position of each agent p_i is continuous, in contrast to the grid description of the forest. Figure 2 shows an example quadrotor with a downward facing camera.

Figure 2: Example quadrotor agent[1]

At the beginning of the simulation, a fire is started at the center of forest and it is assumed that each agent knows this position p_f.

At each time step, the agents take an image of the forest. The image is passed through a convolutional neural network (CNN) and produces a list of k waypoints that define a path in $\mathbf{R}^2$ that the agent will move along. The path is created by linearly interpolating between each pair of waypoints (i.e. connecting a straight line between each pair of waypoints). All trees that are on fire and intersect this path are considered to be "treated."

Table 1 shows the values used for the parameters defined by the problem formulation.

Table 1: Problem parameters

Parameter	Value
Grid size N	simulation dependent
Number of agents r	simulation dependent
Image pixel size	16x16x3
Image tree size hxw	16x16
Number of waypoints k	6

2.2. CNN Architecture

Figure 3 shows the CNN architecture[2]. The network consists of a subsection that generates image features and another section that takes the image features and the other data to generate waypoints.

Note that the first fully-connected layer combines the output of the image processing filters with the input distance measurements (the distances are appended to the end of the vector of image features).

[1]Source: `http://bit.ly/2rHb7Ud`

[2]The network architecture was inspired by code that the authors wrote for CS231N Assignment 2

Table 2: Network parameters for CNN

	Layer			
Parameter	Convolution 1	Convolution 2	Max Pool 1	Max Pool 2
Number of filters	128	128	–	–
Filter size	5x5	3x3	4x4	2x2
Filter stride	1	1	2	1
Padding	0	0	0	0

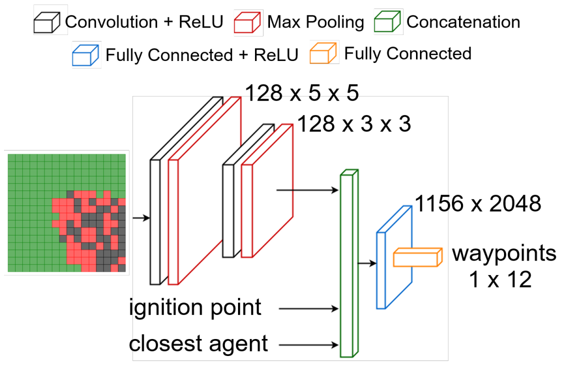

Figure 3: CNN architecture used to generate waypoints. Numbers next to each set of layers represents the output size.

The output is a vector with $2k$ elements, representing the x and y coordinates for all k waypoints. Table 2 shows the parameter values the convolutional network section of the architecture. The first fully connected layer has a size of 1156x2048 and the second fully connected layer has a size of 2048x12.

2.3. Loss Function

The loss function is used to evaluate how well a set of waypoints, generated by agent i, will perform. Equation 1 describes the loss function at a high level; a more rigorous definition is given in Appendix A. In Equation 1, the λ values are hyperparameters that control the relative influence of each term. The choice of hyperparamter values is discussed in section 4.2. The term λ_1 weights the Euclidean distance between the agent's location and the ignition point. The next term with λ_2 discourages the agent from being too close (measured by Euclidean distance) to the nearest agent. The λ_3 term can be used to encourage or discourage the agent to cover a large area of the forest. The λ_4 minimizes the path length to discourage physically unrealistic paths. The path length is measured by the straight line distance between each pair of waypoints. Finally, the λ_5 term encourages using the waypoints to cover trees that are on fire.

$$\begin{aligned}\text{Loss}_i &= \lambda_1 \|q_{end} - p_f\|_2 + \lambda_2 \sum_{i=1}^{k} \frac{i/k}{\|q_i - p_j\|_2 + \epsilon} + \dots \\ &+ \lambda_3 \left(\max q_x - \min q_x\right)\left(\max q_y - \min q_y\right) + \dots \\ &+ \lambda_4 \sum_{i=1}^{k} \|q_{i+1} - q_i\|_2 + \lambda_5 \sum_{i=1}^{k} \|q_i - p_{\text{fire}}\|_1 \qquad (1)\end{aligned}$$

3. Dataset and Features

The data required to train the CNN consists of a local image observation of the forest and the relative positions of the ignition point and the closest agent. For this problem, all data can be generated offline and the network can be trained offline before being implemented online. This is desireable so behavior can be verified before deployment.

First, the image data is generated by creating a forest size equal to the size of the images, which is 16x16 trees. Figure 4 shows what each image represents: a small subsection of the entire forest. Then, the fire ignition point is set to one of nine different locations spread across the forest, and the simulator is run until the fire self extinguishes (since no control is used). At each time step, the state of the forest is saved as an image to be used for training. Figure 5 shows the different ignition points used. Different random number generator seeds were used to deterministically generate different wildfire spreading patterns. Roughly 5,000 images were generated for training.

Next, positions of the ignition point and the nearest agent were generated by first randomly sampling one of eight direction vectors, denoted v. Then, a distance t was randomly sampled, and a position p was calculated as $p = tv$. Roughly 5,000 of these positions were generated for both the ignition point and the nearest agent point (the same number as the number of images generated). The blue vectors in Figure 5 denote the direction vectors used to generate the position data.

Finally, Figure 6 shows a complete set of data that would be generated by an individual agent and provided to the CNN online to produce a set of waypoints.

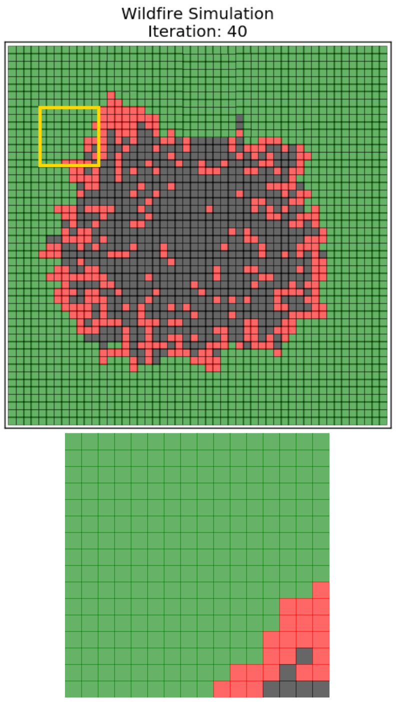

Figure 4: Top: snapshot of forest wildfire simulation, orange box denotes example location of agent observation. Bottom: example observation given to an agent.

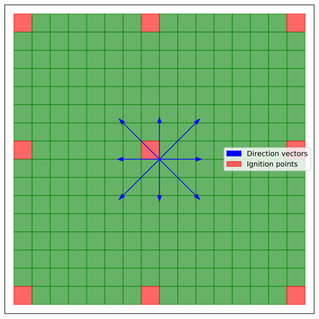

Figure 5: Illustration of method that generates the dataset. The red squares denote different points where the simulated fire is started to create different spreading patterns for the CNN to generalize from. The blue vectors denote directions used to generate the sensor data (e.g. nearest agent position).

3.1. Training

The major benefit of the problem formulation previously described is that the training is very similar to a supervised learning problem. The loss function is used in place of labels to quantitatively judge the performance of the CNN output. The loss function therefore must be differentiable. As a result, simple gradient descent can be used to train the CNN. This is in contrast to an unsupervised learning problem formulation, which would involve trying to learn a good approximation of the value function. Of course, the requirement of a strictly differentiable loss function may be restrictive, whereas the reinforcement learning task simply requires a reward value to be specified.

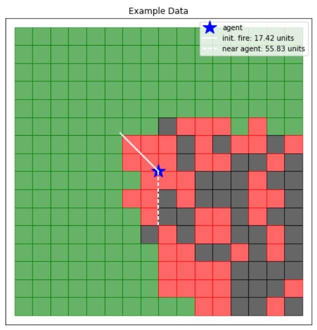

Figure 6: Example of a complete dataset that would be created by an agent. White lines denote the direction of the ignition point (solid) and the nearest agent (dashed) – the lines are scaled to be the same length, and the true distance values are given in the legend. Blue star denotes the agent position, which is assumed to be the center of the sampled image.

4. Results

4.1. Baselines

Four methods have been implemented as a baseline on a grid size of 25x25 and a varying number of agents, and the results are shown in Figure 7. The performance metric is given in Equation 2 and is calculated over many different simulation runs and then averaged.

$$\text{Fraction Burned} = \frac{\text{number of burned trees}}{\text{total number of trees}} \qquad (2)$$

First, without any treatment, on average 95% of the forest burns down which provides an upper bound for the methods.

Two methods use a centralized agent that gathers information and then computes an action. The “random fires” method simply randomly chooses fires to treat; the number of treated trees is limited by the number of agents. The “priority” method chooses to treat fires in order of how many healthy trees they neighbor. The “priority” method performs better than the “random” method for a fewer number of agents, but the results equalize once there are an overwhelming number of agents.

The last two methods use a decentralized scheme. The “decentral+weight” method has each agent compute a Voronoi partition of the space (see [1] for details regarding

Voronoi partitions), and then treat the area with the highest density of fires. The "decentral+priority" method again computes a Voronoi partition, and then has each agent prioritize fires in each cell by the number of healthy neighbors. As expected, the decentralized methods do not perform as well initially compared to the centralized methods, as each agent has less information available to compute a solution. However, all solutions asymptotically tend to the same performance given enough agents to overwhelm the wildfire.

The results of the convolutional neural network approach will be compared to these baseline methods. Note that a relatively small grid size of 25x25 was used as these methods scale poorly with both domain size and number of agents. This is something that the network approach is expected to address.

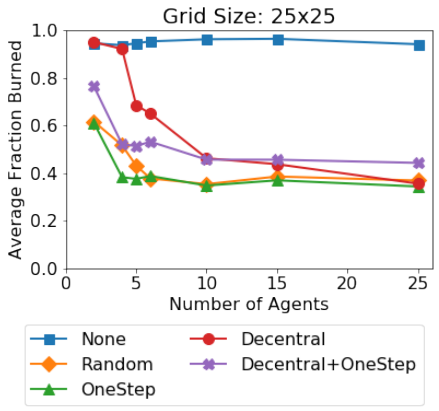

Figure 7: Performance of several baseline methods

4.2. Tuning the Loss Function

The loss function (Equation 1) was created by mathematically describing features of the waypoints and their relation to the environment. However, the relative importance of these features is unclear, so each is weighted by a hyperparameter which was then tuned to achieve good performance. An example of how the total loss decreased for a given set of weights is shown in Figure 8. In order to gain insight into which features in the loss function were contributing the most to the loss, an additional visualization was used which shows the contribution from each of the terms in the loss function as a function of training iteration (see Figure 9). Many combinations of weights were evaluated, and throughout this process several trends became apparent.

The first step in tuning the loss function was to verify that each term correctly led to the desired behavior. To test this, all weights except one were set to zero. It was expected that the first term should draw the robot towards the ignition location, the second term should repel the agent from its neighbors, and so on. This process revealed that the third

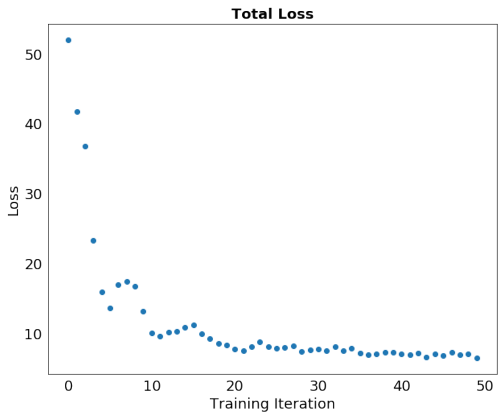

Figure 8: Plot of evolution of loss

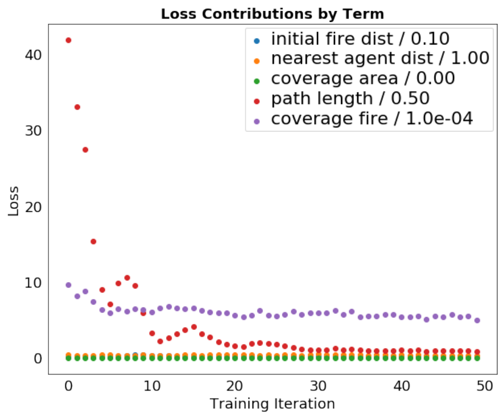

Figure 9: Contribution of each term in the loss function versus training iteration. The values in the legend correspond to the hyperparameter value used for each term.

term in the loss function, the one corresponding to the area covered by the path, performed as it should but would be a poor indicator of path value. In some scenarios the agent is far from the fire and the best path is for it to move in a straight line, resulting in a very small area (e.g. if the agent moves solely in the x direction, then Δy is zero). However, any path where the agent should move in both x and y results in a non-zero area. Due to this realization, the corresponding weight term λ_3 was set to zero. All other terms in the loss function performed as expected in isolation.

Another lesson learned while tuning these hyper parameters was that the "attraction" of the initial fire had to be carefully balanced with the penalty on path length. This term was important because if agents are initialized over all healthy trees they should fly towards the only location that is known to have been on fire – the ignition point. However, the path regularization term needed to be significant enough to prevent the agent from making unrealistically large jumps. When these terms were not balanced well, the agent would not move towards the ignition point, even if it was only observing healthy trees. Logic could have been added so that if the agent doesn't see any burning trees than it should move in the direction of the ignition point, but this additional layer of control was deemed to violate the goal of training a single CNN as a controller. A hybrid controller approach is being considered for for the future work on this project.

4.3. Hyperparameter Values

Training was performed by sampling mini-batches of image data and ignition point and nearest agent data and then using backpropagation to update the weights. The batch size used was 500. The Adam update rule was used with a learning rate of 10^{-4} and a weight decay (l_2 regularization strength of weights) of 10^{-2}. Stochastic gradient decent is the naive approach, and Adam improves on this by incorporating momentum along with a smoothed version of gradient (via a running average). The default coefficients for computing the running averages of the gradient and its square were used (0.9 and 0.999, respectively). The PyTorch library was used to perform the training on a GPU which greatly reduced training time.

A large batch size was used to expose the CNN to many different data combinations. Considering the low dimensionality of the input data, not many training iterations were required to find a local minima. The regularization was included to avoid producing waypoints with large values.

The final hyperparameter values for the loss function are given below. The values were chosen after many iterations of choosing weights, training the CNN, and then qualitatively judging the performance of the waypoints on various test data cases.

$$\lambda_1 = 0.1,\ \lambda_2 = 1,\ \lambda_3 = 0,\ \lambda_4 = 0.5,\ \lambda_5 = 10^{-4}$$

4.4. Single Agent Results

Figure 10 shows four examples of inputs given to the CNN and the corresponding waypoints chosen. The examples qualitatively show that the agent is balancing all of the objectives: the agent moves toward the fires, does not make long paths, and tries to move closer to the ignition point and further from the nearest agent (these positions are not plotted). When the CNN is provided with an image of all healthy trees, the agent moves to ward the ignition point. It is also clear that the agent will struggle with areas that have a lot of fires. However, if the agents are able to treat separate boundary areas of the overall wildfire, then this cooperation should overcome the deficits of a single agent.

4.5. Multi-agent Results

While the behavior of the individual robots was appropriate, the collective behavior of the system was disappointing. Figure 11 shows a sequence of states of a simulation

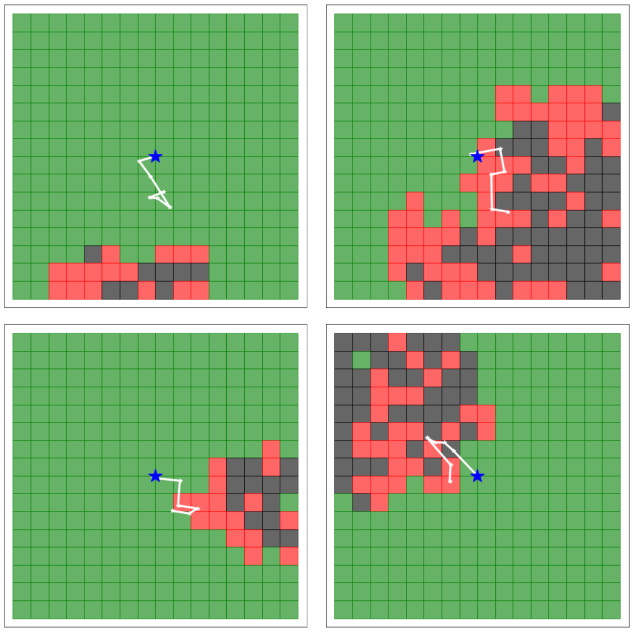
Figure 10: CNN output in various configurations for single agent scenarios

with 6 robots that initially surround a fire. Iterations 1 - 7 show the robots converging on the fire, which is the desired behavior. Unfortunately the agents are unable to realize that the best approach to putting out the whole fire is to divide and conquer, instead clumping together in a small area. As the fire spreads, this clump moves along with it and continues to extinguish fires, but the lack of cooperation dooms the effort as shown by iteration 50. Qualitatively, the agents merge and start to act as a single agent.

Figure 12 shows the quantitative results of the CNN control compared to the baseline approaches. These poor results are likely due to our clump of agents "chasing the fire," i.e. putting out unimportant fires that would eventually burn themselves out, instead of treating the wildfire front (the burning trees that are most likely to spread to healthy trees). It is clear that the agents are not able to cooperate effectively, as they probably required more information from other agents in order to create better cooperative paths.

After being unable to achieve cooperation between the agents, visualizations of the weights were generated to try to gain insight into the behavior of the CNN. However, the visualizations (which can be found in Appendix B) did not have any clear interpretation.

5. Conclusions

In this project several variations of loss functions were explore with the aim of training a convolutional neural network to enable agents to cooperatively fight fires. The re-

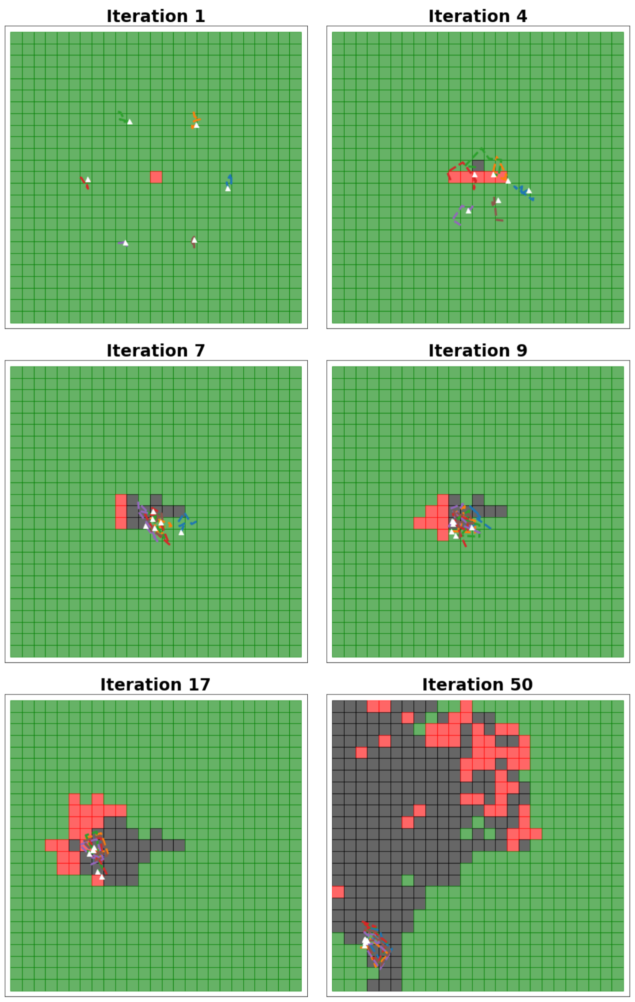

Figure 11: Evolution of multi-agent simulation over time. The dashed lines denote the path created by each agent after interpolating between the waypoints. The white triangles denote the new position of each agent after executing the complete path.

sults of the simulation show successful training of agents to move to and treat fires. The ability to cooperate with nearby agents was less successful.

In testing of various loss function weights and formulations, it became clear that certain features had a tendency to dominate the agent behavior. As might be expected, the agents were strongly compelled to travel to the trees that were burning. In order for the other objectives to be met, the weights on penalizing the path length and closeness of the nearest neighbor had to be set larger by two orders of magnitude.

Despite tuning these hyperparameters to encourage meeting multiple objectives, the data was not rich enough for the agents to make nuanced decisions – if the agent

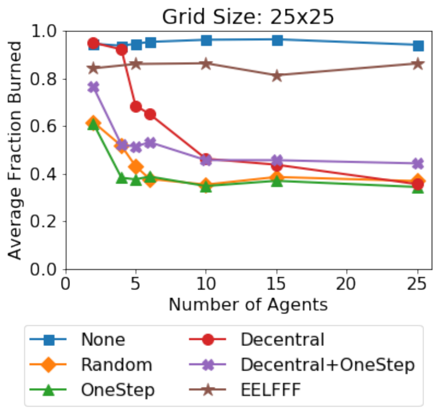

Figure 12: Results of EELFFF algorithm compared to baseline methods

learned to go toward the fire it would do so regardless of the other agents position or actions. Each component of the loss function was tested individually to ensure proper behavior, but when combined one or two objectives always dominated (usually the fire coverage objective and the path length objective).

6. Future Work

Future work will focus on optimizing the balance between the cooperation, scalability, and complexity of the system. It is strongly desirable for agents to only rely on information they can directly sense, but if needed this requirement can be relaxed to include information transmitted by their nearest neighbors. More sources of data that are expected to assist the CNN in learning to cooperate include the locations of more agents, these agents planned trajectories, and potentially information about the closest burning tree.

It may also be necessary to modify the network architecture if more information is added. More nonlinear layers and a deeper network overall could help in capturing some of the subtleties that are missed by the current network.

Another possibility for future direction is to add a higher level of logic that can direct the robot in simple cases and save the neural network for when there are more complex choices to be made. For example, when there are no burning trees in view, the agent could head directly to the ignition point.

Finally, it is possible to cast this problem in a Reinforcement Learning framework, which may alleviate some issues with evaluating the performance of an agent's actions. Instead, the agent could make discrete actions in the domain and have a reward provided to it. Multi-agent deep Q-learning is being considered as another possible problem formulation.

A. Loss Function

A more rigorous definition of the loss function follows. The output of the CNN is denoted as $q \in \mathbf{R}^{2k}$ where k is the number of waypoints generated. Decompose q as $q = \begin{bmatrix} q_x & q_y \end{bmatrix}^T$ with $q_x, q_y \in \mathbf{R}^k$. The sub-vectors q_x and q_y describe the x and y coordinates of each waypoint. The loss function can be written as the sum of five functions:

$$\begin{aligned}\text{Loss}_i =& \lambda_1 f_{\text{fire}}(q, p_f) + \lambda_2 f_{\text{agent}}(q, p_j) + \lambda_3 f_{\text{area}}(q) + \\ &+ \lambda_4 f_{\text{path length}}(q) + \lambda_5 f_{\text{cover fire}}(q, p_{\text{fires}})\end{aligned}$$

The first function f_{fire} encourages the agent to approach the ignition point $p_f \in \mathbf{R}^2$ based on the last waypoint on the path.

$$f_{\text{fire}}(q, p_f) = \| \begin{bmatrix} q_{x,k} & q_{y,k} \end{bmatrix}^T - p_f \|_2$$

The second function f_{agent} penalizes a path that approaches the nearest agent position $p_j \in \mathbf{R}^2$. A small positive constant is used to prevent numerical issues. If the waypoints were to lie exactly at the position p_j then the loss for this term would be very high. This term affects all waypoints, but with a increasing cost for each waypoint. This emphasizes the desire that the agents should not end their path close to another agent.

$$f_{\text{agent}}(q, p_j) = \sum_{i=1}^{k} \frac{i/k}{\| \begin{bmatrix} q_{x,i} & q_{y,i} \end{bmatrix}^T - p_j \|_2 + 10^{-4}}$$

The third function f_{area} encourages a path that covers a large area by constructing a rectangular bounding box around the waypoints. Larger bounding box areas can be penalized or encouraged depending on the sign of the hyperparameter λ_3.

$$f_{\text{area}}(q) = (\max q_x - \min q_x)(\max q_y - \min q_y)$$

The fourth function $f_{\text{path length}}$ penalizes long paths. A path is constructed from the waypoints by a simple linear interpolation. The local origin for each agent is assumed to be the position $(x, y) = (0, 0)$.

$$\begin{aligned}f_{\text{path length}}(q) =& \| \begin{bmatrix} q_{x,1} & q_{y,1} \end{bmatrix}^T \|_2 + \dots \\ &+ \sum_{i=2}^{k} \| \begin{bmatrix} q_{x,i} & q_{y,i} \end{bmatrix}^T - \begin{bmatrix} q_{x,i-1} & q_{y,i-1} \end{bmatrix}^T \|_2\end{aligned}$$

Finally, the fifth function $f_{\text{cover fire}}(q, p_{\text{fires}})$ encourages an agent to use the waypoints to move close to trees on fire. The positions of these fires in an image frame are given as $p_{\text{fires}} \in \mathbf{R}^{m \times 2}$ where m denotes the number of fires present. For a given input image, there are three possible scenarios. First, if there are no fires ($m = 0$), then the loss is zero.

$$f_{\text{cover fire}}(q, p_{\text{fires}}) = 0 \text{ if no fires}$$

Second, if there are less fires than waypoints ($m \leqslant k$), then the loss is based on the first m waypoints.

$$f_{\text{cover fire}}(q, p_{\text{fires}}) = \sum_{i=1}^{m} \| \begin{bmatrix} q_{x,i} & q_{y,i} \end{bmatrix}^T - \begin{bmatrix} p_{\text{fires},x,i} & p_{\text{fires},y,i} \end{bmatrix}^T \|_1$$

Third, if there are more fires than waypoints $m > k$, then intermediate points are created along the path, using linear interpolation, to match to the fire positions. Specifically, $m/(k-1)$ points are added per pair of waypoints. The vector $\tilde{q} \in \mathbf{R}^m$ contains these new interpolated points as well as the original waypoints.

$$f_{\text{cover fire}}(q, p_{\text{fires}}) = \sum_{i=1}^{m} \| \begin{bmatrix} \tilde{q}_{x,i} & \tilde{q}_{y,i} \end{bmatrix}^T - \begin{bmatrix} p_{\text{fires},x,i} & p_{\text{fires},y,i} \end{bmatrix}^T \|_1$$

B. Network Visualizations

Figures 13 and 14 depict visualizations of the first and second layer weights[3]. Unfortunately, there are no clear patterns to suggest what could be done to improve this part of the network.

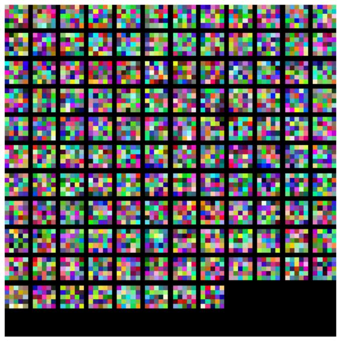

Figure 13: Visualization of weights of first convolution filter layer

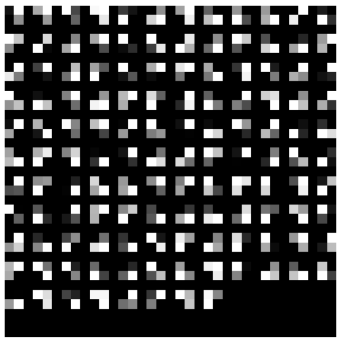

Figure 14: Visualization of weights of second convolution filter layer

[3]The code to visualize the network weights was adapted from code provided for CS231N Assignment 1

References

[1] F. Aurenhammer. Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, Sept. 1991.

[2] D. W. Casbeer, R. W. Beard, T. W. McLain, S.-M. Li, and R. K. Mehra. Forest fire monitoring with multiple small uavs. In *Proceedings of the 2005, American Control Conference, 2005.*, pages 3530–3535 vol. 5, June 2005.

[3] Y. F. Chen, M. Liu, and H. J. P. Everett, Michael. Decentralized non-communicating multiagent collision avoidance with deep reinforcement learning. IEEE, 2017.

[4] J. M. de Dios, B. Arrue, A. Ollero, L. Merino, and F. Gmez-Rodrguez. Computer vision techniques for forest fire perception. *Image and Vision Computing*, 26(4):550 – 562, 2008.

[5] K. A. Ghamry, M. A. Kamel, and Y. Zhang. Cooperative forest monitoring and fire detection using a team of uavs-ugvs. In *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 1206–1211, June 2016.

[6] M. Kumar, K. Cohen, and B. HomChaudhuri. Cooperative control of multiple uninhabited aerial vehicles for monitoring and fighting wildfires. *Journal of Aerospace Computing, Information, and Communication*, 8(1):1–16, 2011.

[7] L. Merino, F. Caballero, J. R. Martínez-de Dios, J. Ferruz, and A. Ollero. A cooperative perception system for multiple uavs: Application to automatic detection of forest fires. *Journal of Field Robotics*, 23(3-4):165–184, 2006.

[8] L. Merino, F. Caballero, J. R. Martínez-de Dios, I. Maza, and A. Ollero. An unmanned aircraft system for automatic forest fire monitoring and measurement. *Journal of Intelligent & Robotic Systems*, 65(1):533–548, 2012.

[9] C. Phan and H. H. Liu. A cooperative uav/ugv platform for wildfire detection and fighting. In *System Simulation and Scientific Computing, 2008. ICSC 2008. Asia Simulation Conference-7th International Conference on*, pages 494–498. IEEE, 2008.

[10] G. D. Richards. An elliptical growth model of forest fire fronts and its numerical solution. *International Journal for Numerical Methods in Engineering*, 30(6):1163–1179, 1990.

[11] R. C. Rothermel. A mathematical model for predicting fire spread in wildland fuels. 1972.

[12] A. Somanath, S. Karaman, and K. Youcef-Toumi. Controlling stochastic growth processes on lattices: Wildfire management with robotic fire extinguishers. In *53rd IEEE Conference on Decision and Control*, pages 1432–1437, Dec 2014.

[13] D. Stipaničev, M. Štula, D. Krstinić, L. Šerić, T. Jakovčević, and M. Bugarić. Advanced automatic wildfire surveillance and monitoring network. In *6th International Conference on Forest Fire Research, Coimbra, Portugal.(Ed. D. Viegas)*, 2010.

[14] P. Sujit, D. Kingston, and R. Beard. Cooperative forest fire monitoring using multiple uavs. In *Decision and Control, 2007 46th IEEE Conference on*, pages 4875–4880. IEEE, 2007.

[15] D. Twidwell, C. R. Allen, C. Detweiler, J. Higgins, C. Laney, and S. Elbaum. Smokey comes of age: unmanned aerial systems for fire management. *Frontiers in Ecology and the Environment*, 14(6):333–339, 2016.