Comparative Analysis of Agent-Oriented Task Assignment and Path Planning Algorithms Applied to Drone Swarms

Rohith Gandhi Ganesan*, Samantha Kappagoda*, Giuseppe Loianno†, David K. A. Mordecai*

*RiskEcon® Lab for Decision Metrics @ Courant Institute of Mathematical Sciences
†Agile Robotics and Perception Lab (ARPL) @ Tandon School of Engineering
New York University, New York, NY 10011
{rgg296, loiannog} @nyu.edu, {kappagoda, mordecai} @cims.nyu.edu

Abstract—Autonomous drone swarms are a burgeoning technology with significant applications in the field of mapping, inspection, transportation and monitoring. To complete a task, each drone has to accomplish a sub-goal within the context of the overall task at hand and navigate through the environment by avoiding collision with obstacles and with other agents in the environment. In this work, we choose the task of optimal coverage of an environment with drone swarms where the global knowledge of the goal states and its positions are known but not of the obstacles. The drones have to choose the Points of Interest (PoI) present in the environment to visit, along with the order to be visited to ensure fast coverage. We model this task in a simulation and use an agent-oriented approach to solve the problem. We evaluate different policy networks trained with reinforcement learning algorithms based on their effectiveness, i.e. time taken to map the area and efficiency, i.e. computational requirements. We couple the task assignment with path planning in an unique way for performing collision avoidance during navigation and compare a grid-based global planning algorithm, i.e. Wavefront and a gradient-based local planning algorithm, i.e. Potential Field. We also evaluate the Potential Field planning algorithm with different cost functions, propose a method to adaptively modify the velocity of the drone when using the Huber loss function to perform collision avoidance and observe its effect on the trajectory of the drones. We demonstrate our experiments in 2D and 3D simulations.

I. INTRODUCTION

Swarm robotics has been an active area of research and recent improvements in computational capabilities, deep learning and reinforcement learning methods have led to increased interest in autonomous swarm systems. Using a swarm of robots over a single robot to accomplish a task would reduce the time taken to accomplish it, increase the robustness, flexibility and scalability of the system [1]. Drone swarms, in particular, have significant applications as they can be used to map an unknown area [2], [3], [4], inspect objects which are dangerous or inaccessible to humans [5], [6], transport object between locations [7], [8], [9] and perform aerial monitoring [10], [11].

In this work, we consider the task of mapping an unknown area with Points of Interest (PoI) using drone swarms. Points of Interest are locations present in the environment that we would want our drones to reach in order to map them. The drone swarms have to choose the PoI it would visit along with the order in which it would visit them based on the priority level of the points. The drones should also navigate the environment by avoiding collision with obstacles and with each other to reach the PoI. We characterize the first part of the problem, i.e. choosing the order of PoI to visit as a task assignment problem where each drone would choose the order in which it would visit the points to map them and minimize coverage time. The prioritized order is then executed by the path planning algorithm which performs collision avoidance with obstacles and with other drones. Note that the computation of both task assignment and path planning is designed to take place in a decentralized manner, i.e. on the edge device and does not require perfect global knowledge of the environment.

We use Reinforcement Learning (RL) algorithms instead of heuristic based search algorithms (e.g.: Dijkstra, A* Search etc.) to solve the task assignment problem as the search algorithms require exponential time and space complexity. The heuristic based search algorithms do not perform any exploration to gain additional information about the environment. Instead, they only exploit using hard-coded search strategies. They also do not perform an adaptive search strategy for a dynamic environment. Conversely, the Bellman optimization method through dynamic programming also suffers from exploding searching state spaces. Therefore, using neural networks as function approximators with RL algorithms could enable a method to adaptively and scalably search large state spaces and produce approximately optimal solutions, conditioned upon the relative stationarity of the state space and sufficient computing resources. We evaluate the effectiveness of RL based algorithms to exploration problems that involve Unmanned Aerial Vehicles (UAVs). We perform a comparative analysis on several neural network architectures and identify the benefits and drawback of each solution based on its efficiency and effectiveness. We use the policy gradient algorithm with different policy networks including Sequence-to-Sequence model, and Sequence-to-Sequence model with attention, Pointer Net, Transformer networks [12] and compare their computational requirements and time taken to map the PoI. Furthermore, we couple the aforementioned RL approach to two path planning strategies that does not require perfect knowledge of the environment for obstacle avoidance - Wavefront planner [13], which is a grid-based global planner and Potential Field planner [13], which is a gradient-based local planner. We also study the effect of different cost functions in the Potential Field planner on the trajectory of the drone to
allow safety. In order to avoid a completely conservative or radical strategy based on the distance, we propose a method to adaptively change the velocity of the drone by scaling the value of $\delta$ in the Huber loss function.

II. RELATED WORK

A. Task assignment using RL

The problem of predicting the order of points to be visited is analogous to the Travelling Salesman Problem (TSP) where an agent visits all locations with minimal tour length. Given that the TSP is an NP-Hard problem, it is intractable to find optimal solutions for large state spaces. Therefore, using neural networks allows the problem to be more tractable by acting as a function approximator and produce an approximately optimal solution. In [16], authors propose a new model architecture called Pointer Net and use it to solve the Travelling Salesman Problem through supervised learning. The labels for each input are generated using the Held-Karp algorithm [17] for a smaller number of points and for a larger number of points, approximate TSP solvers are used. However, the trained model at best, would attempt to imitate the approximate solvers and produce sub-optimal solutions. [18] uses the Pointer Net architecture and optimizes the network using the Policy Gradient [19] method and negative tour length as a reward signal. [20] uses the transformer architecture [21] to solve routing problems including TSP and train the model architecture using Policy Gradient method. [22] proposes a combination of graph neural network and Q-learning to solve the TSP. The above mentioned TSP solution methods are proposed for an environment with a single agent and the prediction order is conditioned upon the tour length and not its priority level. Therefore, in our work, we evaluate the performance of different policy architectures in a multi-agent setting for mapping PoI in a simulation.

B. Path Planning for Drones

The classical algorithms for performing path planning in robots are Wavefront, Dynamic Window [23], Potential Field [24] and Velocity Obstacle method. [25] proposes a variation of the velocity obstacle method used for performing obstacle avoidance in a multi-agent setting. [26] applies the Potential Field algorithm to multi-agent Unmanned Aerial Vehicles (UAVs) to avoid static and dynamic obstacles in simulation. In our work, we evaluate the Wavefront and Potential Field algorithm in a multi-agent setting and also study the effects of different cost functions used for the Potential Field algorithm in simulation in 2D and 3D environments.

III. PROBLEM FORMULATION

We split the problem of performing optimal coverage by drone swarms into two parts - Task Assignment for predicting the PoI to be visited and Path Planning to perform obstacle avoidance. The task assignment problem is described as follows. We are interested in the task of mapping an environment using drone swarms with minimum coverage time. The environment contains certain PoI which must be mapped. Only when all the PoI are mapped is the task considered completed. Some PoI have a higher priority level than others. Therefore, we prefer that the drones visit these prior to visiting PoI with a relatively lower priority. Each drone has to predict the PoI to be visited along with the order in which it would visit them to minimize total steps taken and map all PoI. The path planning algorithm utilizes the predicted order to visit the PoI and creates a trajectory to avoid collision with obstacles and reach the goal points.

IV. TASK ASSIGNMENT

In the task assignment problem, the drones predict the PoI to be visited along with the order to be visited such that the total number of time steps to completely map the environment is minimized. The drones use the global information about the location of PoI and independently choose actions that enable them to map the environment as fast as possible. The actions predicted at each time step is obtained from a policy network which is trained using an policy gradient algorithm.

A. Simulation - 2D

We model the problem of task assignment in a 2D simulation where the environment is represented as a grid and the drones are spawned at random locations within the environment. The PoI are uniformly distributed within the grid and a certain number of points are randomly sampled and assigned a higher priority level than others. The drones can choose one out of eight possible actions, i.e. 8 possible directions it can move which are North, North-East, East, South-East, South, South-West, West, North-West. A rendered image of the 2D simulation is displayed in Figure 2 and all possible actions that can be taken by a drone at time step during the task assignment.

B. Reward Function

The incentive structure of the simulation plays an important role as it models the task we are trying to achieve and it directly affects the performance of the reinforcement learning algorithm. Equation (3) shows the total reward calculation.
Fig. 2: A rendered image from the 2D simulation of 5 × 5 grid. The green triangles represent the drones, the stars represent the PoI. The red stars have a higher priority level compared to black stars. The white stars show that these points are mapped by the drone.

Equation (1) represents the positive reward each drone gets for mapping a point. It obtains a reward of $\alpha$ for mapping a low priority point and a reward of $2\alpha$ for mapping a high priority point, where $\alpha$ is a positive real number. In our experiments we set $\alpha = 5$.

$$R_1 = \begin{cases} \alpha & \text{if low priority point mapped} \\ 2\alpha & \text{if high priority point mapped} \end{cases}$$  

Equation (2) represents the penalty the drone receives for delaying the mapping of high priority points. If all high priority points are mapped, the drone does not receive any penalty, else, the drone is penalized by the $L_2$ norm between the drone’s location and the closest unmapped high priority point. Equation (3) represents the final reward signal that drones receive. The agent receives a negative reward of value one at each time step which encourages the policy to map the points faster as it would maximize the cumulative sum of rewards.

$$R_2 = \begin{cases} 0 & \text{high priority points mapped} \\ \min_i \|d - x_i\|_2 & \text{otherwise} \end{cases}$$

$$\text{Reward} = R_1 - R_2 - 1$$

C. Policy Networks

We evaluate Sequence-to-Sequence (Seq2Seq) model, Sequence-to-Sequence model with attention, Pointer Net and Transformer networks as our policy network and train them using REINFORCE algorithm.

1) Seq2Seq: The architecture of the Seq2Seq model used in our experiments is shown in Figure 3. The input to the encoder part of the model consists of the drone location, location of all of the PoI and its priority level. The encoder block of the model compresses the input information into a fixed dimensional vector called context vector. The decoder block uses the context vector and predicts the location the drone has to visit at each time step. The output at the previous time step is provided as input to the current time step for the decoder block. At the first time step, the input to the decoder block is a learnable vector $\langle LW \rangle$.

2) Seq2Seq with Attention: The architecture of the Seq2Seq model with attention is shown in Figure 4. The decoder at each step receives additional context from the attention module which is a weighted combination of the hidden states of the encoder model. The input to the encoder and decoder blocks of the model are identical to that of the Seq2Seq model.

3) Transformers: The architecture of the transformer model is shown in Figure 5. The input of the model contains drone location, location of PoI and its priority level. This information is projected into a high dimensional space through an embedding layer which is a non-linear projection of the input. The high dimensional vectors are then passed through the two Transformer blocks. Each transformer block consists of a multi-head attention module, layer norm and a feed forward layer with skip connections. The output of the transformer model is projected back to a lower-dimensional space.
The input contains location of PoI and its priority level which is then projected onto a higher dimensional space using an embedding layer. The embeddings are then provided as input to the transformer encoder blocks.

The PointerNet has an encoder-decoder architecture with LSTM cells. The decoder at each step points to the location in the encoder states that the drones have to visit. \( \text{LW} \) represents a learnable parameter.

空间 to predict the point to visit at each time step.

4) **Pointer Net**: The architecture of the Pointer Net [10] is shown in Figure 6. The input to the encoder and decoder blocks of the model are identical to that of the Seq2Seq model. At each step, the decoder points to location in the encoder block that it should visit.

## D. Results

The results of the different policy networks are shown in Table I. The time steps reported in the table depicts the number of time steps a single agent takes to map a 5 × 5 grid. We were able to observe that the Seq2Seq model takes the most number of steps compared to other policy networks. This is reasonable as all other models use additional context in the form of attention modules to make the prediction. Therefore, we can infer that the attention modules help in improving the prediction. However, the Seq2Seq model has the least number of parameters among all of the mentioned policy networks. Therefore, there is a trade-off to be made between performance (number of time steps) and computation.

## E. Discussion

1) **Plug and Play transfer learning**: An advantage of using these models is the ability to scale-up to multiple agents/drones without additional training, i.e. we can train a single agent using the REINFORCE algorithm and replicate the parameters of the model to multiple agents and perform inference. This allows us to save computational resources during the training phase.

2) **Disadvantages**: The disadvantage of using the above-mentioned policy networks is that the order in which the PoI are visited (policy prediction) is done at the beginning. In real life, however, the drones might have drifted off the desirable path due to sensory measurement errors or to navigate around an obstacle. To adjust the predicted action for the correction, we must perform another forward pass through the policy network which would increase the computational requirements.

## V. PATH PLANNING

The policy network predicts the order in which the drones have to visit the PoI to ensure fast coverage of all points. In real life, however, moving between one point of interest to another is not straightforward as the drones might have to avoid obstacles on their paths to reach the goals. We take the prediction of our policy network and perform path planning to avoid collision with obstacles. We compare two path planning algorithms for obstacle avoidance: Wavefront planner and Potential Field planner, which are tested in 2D and 3D simulations with obstacles present in the environment.

A rendered image of the 2D simulation with obstacles is shown in Figure 7. Note that the Pointer Net described in the section IV-C4 is used to generate the order in which the PoI have to be visited by the drones and the path planning algorithms only performs obstacle avoidance to reach the goal state, i.e. the assigned point at that time step for each drone.

### A. Wavefront Planner

The Wavefront planning algorithm [31] is a grid-based global planning algorithm, i.e. it assumes that the entire environment is visible to drones and the obstacles present in the environment are static. Any changes to the location of the obstacle have to be updated to the drones and the planning algorithm has to be re-run to make adjustments to the
The environment is split into discrete grids and the goal cell is initialized with value 2, obstacle cells with value 1 and all other cells with value 0.

- At iteration 1, start from the goal state and choose all reachable cells based on connectivity and initialize them with value $2 + 1 = 3$.
- At iteration 2, choose all reachable cells from cells with value 3 and initialize them with value $3 + 1 = 4$.
- Perform the above step iteratively until you reach the drone’s location cell. Do not consider obstacle cells with value 1 while performing the above steps.
- To obtain the shortest path from the drone’s location to the goal state, choose the cell with the lowest value at each time step.

The algorithm performs a breadth-first search among the grids to predict the shortest path between the drone and the goal state. The disadvantage with the Wavefront planning algorithm is the assumption that the obstacles have to be static and the entire environment space is always known to the drones. Another disadvantage is the discrete nature of the algorithm. In real life, the drones can only move in continuous velocities and distances and cannot make discrete jumps. Increasing the granularity of the algorithm, i.e. increasing the number of grid cells can mitigate this issue to a certain extent, however, this increase comes at a higher computational cost since we would have to perform a breadth-first search through a larger number of cells.

### B. Potential Field Planner - 2D Simulation

The Potential Field planning algorithm is a gradient-based local planning algorithm, i.e. it does not require global information about the environment and the objects only within the Field of View (FoV) of the drone have an impact on its trajectory. The Potential Field planning algorithm minimizes a cost function characterized by attractive potential between the goal state and drone and a repulsive potential between the drone and the obstacles within its FoV. Note that other drones within the FoV of a particular drone are also considered as obstacles. We minimize the cost function with respect to the position of the drone and update the position by calculating the gradients to the cost function. This would provide us with a trajectory to the goal state.

\[
C = U_{\text{att}} + U_{\text{rep}} \tag{4}
\]

\[
L_i(x, y) = \begin{cases} 
0.5(x_i - y_i)^2 & \text{if } |x_i - y_i| < \delta \\
\delta(|x_i - y_i| - 0.5\delta) & \text{otherwise}
\end{cases} \tag{5}
\]

\[
U_{\text{att}} = L_i(x, g) \tag{6}
\]

\[
U_{\text{rep}} = \begin{cases} 
-L_i(x, o) & \text{if dist} \leq D \\
0 & \text{otherwise}
\end{cases} \tag{7}
\]

The cost function used to model the Potential Field algorithm that we aim to minimize is shown in Equation (4). Huber loss, $L_1$ loss and Mean-Squared-Error (MSE) are among the most commonly used loss functions for regression tasks. In our task of path planning, these loss functions help minimize the distance between the drone and the goal state while maximizing the distance between the drone and the obstacles. They also have an implication in the trajectory of the drone. The different loss functions explored in our work are visualized in Figure 8.

Fig. 8: Visualization of $L_1$ loss, MSE, Huber loss and Log-Cosh functions
1) L_1 loss: The L_1 loss is represented in Equation (8). This loss is less sensitive to outlier values compared to MSE as we are only calculating the absolute difference between the values. Therefore, it is a linear equation which provides only constant gradients to update the position of the drone irrespective of its distance to the goal or obstacle. The drawback of this function is that the derivative does not exist at value 0. To overcome this issue, we consider a point to be mapped when the Euclidean distance between the drone and the point of interest is below a threshold value \( \gamma \). In our experiments, we set \( \gamma \) to 0.5.

\[
L_i(x, y) = |x_i - y_i |
\]  
\( (8) \)

2) MSE loss: The Mean-Squared-Error loss is represented in Equation (9). Relative to L_1 loss, MSE is sensitive to outliers in the data, as it is a quadratic representation of the error value. The loss function provides linear gradients, however, the gradients exhibit high values compared to L_1 loss when the goal point is located at a significant distance from the drone.

\[
L_i(x, y) = (x_i - y_i)^2
\]
\( (9) \)

3) Log-Cosh loss: The log-cosh function is the logarithm of the hyperbolic cosine function. Equation (10) describes the loss function. For very high values of error, the \( e^x \) term dominates in Equation (11) and for very low values of error, \( e^{-x} \) term dominates. Therefore the loss function behaves as a linear function at extreme values. At error values close to zero, the function can be approximated to a quadratic function using the Taylor series expansion (Refer to Appendix for proof).

\[
L_i(x, y) = \log(\cosh(x_i - y_i))
\]
\( (10) \)

\[
\cosh(x) = \frac{e^x + e^{-x}}{2}
\]
\( (11) \)

4) Huber Loss: Huber loss function is represented in Equation (12). The loss function uses a thresholding value \( \delta \), where if the L_1 loss is below that value, a scaled value of the Mean-Squared-Error is returned as the loss value, else, the L_1 loss scaled by \( \delta \) is the loss value where \( \delta \) governs the slope of the loss function. A high value of \( \delta \) will provide relatively higher gradients compared to a low value of \( \delta \).

\[
L_i(x, y) = \begin{cases} 
0.5(x_i - y_i)^2 & \text{if } |x_i - y_i| < \delta \\
\delta(|x_i - y_i|) - 0.5\delta & \text{otherwise}
\end{cases}
\]
\( (12) \)

5) Adaptively Choosing \( \delta \) - 2D: We explore different methods to adaptively choose \( \delta \) which governs the magnitude in which the drone moves at each time step. A higher value of \( \delta \) would result in larger steps in the environment or a larger velocity (in terms of step size per time step) which causes the agent to collide with an obstacle which is not in its FoV. Choosing a low value of \( \delta \) would result in the agent being more cautious in terms of navigation, i.e. making smaller updates. This, however, might not be optimal because the agent might end up taking smaller steps even though there are no obstacles in its FoV. Using this intuition, we model the equation for choosing \( \delta \) based on the Equations (13); an illustration is provided in Figure 9. The Equation (13) with parameters \( \alpha = 1 \) and \( \beta = 1 \), \( \alpha = 2 \) and \( \beta = 1 \), \( \alpha = 4 \) and \( \beta = 1 \) can be grouped together as the perceived size of the obstacle by the drone is linearly scaled and with parameters \( \alpha = 1 \) and \( \beta = 2 \), the perceived size of the obstacle is quadratically scaled, \( d \) represents the length of the edge of the FoV, shown in Figure 9.

\[
\delta = \frac{\text{Area of FoV}}{\text{Area of FoV} + \alpha(\text{Area of FoV} \cap \text{Area of Obstacle})^{\beta} \times d / 2}
\]
\( (13) \)

The area of FoV is the area that is visible to the agent and Area of an obstacle is the perceived size of the obstacle within the agent’s FoV. Note that the calculation of \( \delta \) is independent of the modality of the perception, however, dependent on its accuracy in estimating the perceived size of the obstacle. Incorrect or noisy estimates might have negative consequences as \( \delta \) could be set to a high value which could cause a collision with an obstacle.

\[\text{Fig. 9: Area of FoV is represented by the violet square, the blue colored square represents the actual size of the obstacle and the grey square represents perceived size of the obstacle by the agent, i.e. (Area of FoV} \cap \text{Area of Obstacle)}.\]
the drones to take larger steps, we add small noise to the gradients, when the gradient update is below a threshold value.

Fig. 10: Curves representing Equation (13) with parameters $\alpha = 1$ and $\beta = 1$, $\alpha = 2$ and $\beta = 1$, $\alpha = 4$ and $\beta = 1$, $\alpha = 1$ and $\beta = 2$. The x-axis represents the perceived size of the obstacle, i.e. Area of FoV $\cap$ Area of Obstacle and y-axis represents the value of $\delta$. K represents the area of the FoV which is 4 units in our example.

6) Results: Table II shows the number of time steps 2 drones take to map a $5 \times 5$ environment using the Potential Field planner. The number of time steps a single agent takes to map 25 points represented as $5 \times 5$ grid using the Wavefront planner is 113. We do not report this score in Table II as it is not an appropriate comparison.

| Loss function | $\alpha$ | $\beta$ | Time steps |
|---------------|---------|---------|------------|
| $L_1$ loss    | -       | -       | 137        |
| MSE loss      | -       | -       | 228        |
| Log-cosh loss | -       | -       | 221        |
| Huber loss - $\delta = 0.5$ | - | - | 94 |
| Huber loss - $\delta = 1.0$ | - | - | 121 |
| Adaptively choose $\delta$ - 2D | 1 | 1 | 138 |
| Adaptively choose $\delta$ - 2D | 2 | 1 | 185 |
| Adaptively choose $\delta$ - 2D | 4 | 1 | 173 |

Table II: Comparison of number of time steps each loss function takes to map 25 points. Time steps is measure of number of steps 2 drones take to map all points using the Potential Field planner in a 2D simulation. The cells are represented as a $5 \times 5$ grid. We assume the time taken for performing each action is equal, therefore, total time steps and total steps are equivalent. The time steps reported are the best score across three runs. Note that the drones can move in continuous space, i.e. it could move to any real number coordinate within the environment.

7) Discussion: We observe that Mean-Squared-Error loss takes the most number of time steps to map an environment. This is because the gradient of the loss function is linearly scaled. This causes the drone to make large jumps when the distance between goal and drone is high and aggressively reduce distance covered when the drone approaches the goal. Huber loss function with $\delta$ set to 0.5 also takes a long time because, the low value of $\delta$ makes the drone more cautious in terms of navigation. Therefore, only small updates are made to the position of the drone, which leads to increased mapping time. Adaptively choosing $\delta$ through Equation (13) with parameters $\alpha = 4$ and $\beta = 1$, $\alpha = 1$ and $\beta = 2$ also increases the mapping time for a similar reason, i.e. the presence of a small obstacle within the FoV aggressively reduces $\delta$ compared to using parameters $\alpha = 1$ and $\beta = 1$, $\alpha = 2$ and $\beta = 1$. This causes small updates to the position of the drone which leads to an increase in time steps.

C. Potential Field Planner - 3D simulation

We use the Potential Field planner in a 3D simulation and discuss the effects of different loss functions on the trajectory of the drone. In the 3D simulation, we dynamically vary the altitude of the obstacles and the Points of the Interest to be mapped. Figure 11 represents a screenshot from the simulation with obstacles of random altitude and PoI present at different heights.

A point is considered to be mapped when the $L_2$ loss between the drone’s center and point’s center is below a threshold value. We observe that in the case of varying both the obstacle height and PoI altitude, the drones oscillate when there is an obstacle within its FoV. To nudge the drone out of this oscillation, we reuse the earlier trick of adding small noise to the gradient. An ideal behaviour would be that, when a drone is close to an obstacle, it would move tangentially to the obstacle allowing it to move closer to the goal while avoiding collision. Adding small noise to the gradient allows us to achieve this but modifying gradients could translate into small jitters in the motion of the drone which is not desirable and also a waste of power resource. Therefore, we scaled the repulsive potential in the cost function by a value from range $(0, 1)$. Scaling the contribution of repulsive potential to the cost function reduces its effect but does not eliminate it.
Instead of a direct step back, the attractive potential contributes relatively more to the gradient of the cost function, therefore, causing it to move tangentially to an obstacle and breaking the oscillation.

1) Adaptively Choosing $\delta$ - 3D: We also evaluate the Huber loss function in 3D for path planning of the drones. We reuse and adapt the equations provided in Section V-B5 for the 3D simulation. The Equations are shown below. We consider the volume represented by the FoV instead of the area and we use intersection of the volumes between the perceived size of the obstacle and the FoV of the drone to scale the value of $\delta$. The value $d$ in the equation represents the length of the edge of the cube representing the FoV of the drone. This is illustrated in Figure 12

$$
\delta = \frac{\text{Vol. of FoV}}{\text{Vol. of FoV} + \alpha(\text{Vol. of FoV} \cap \text{Vol. of Obstacle})^\beta} \times d/2
$$

(14)

![Diagram](image_1)

Fig. 12: Volume of FoV is represented by the violet cube, blue colored cube represents the actual size of the obstacle and the grey cube represents perceived size of the obstacle by the agent, i.e. (Vol. of FoV $\cap$ Vol. of Obstacle).

Figure 13 shows how the value of $\delta$ scales based on the intersection of volumes. We were able to observe that Equation (14) with parameters $\alpha = 4$ and $\beta = 1$ reduces $\delta$ aggressively than other parameter choices. This translates into very smaller steps taken by the drone in the simulation when there is an obstacle within its FoV. Therefore, the drones take a longer time to map all the PoI.

2) Results: The $L_1$ loss function is used as the baseline for comparing performance in the 3D simulation. Time steps are a measure of number of total steps required to map all points in the environment. The results reported in Table III are for 2 drones and for 25 points present in the environment at random altitudes. The obstacles are also of varying height.

![Diagram](image_2)

Table III: Comparison of number of time steps for $L_1$ loss and Huber loss function with adaptively choosing $\delta$. Time steps is measure of number of steps 2 drones take to map 25 points using the Potential Field planner in a 3D simulation. We assume the time taken for performing each action is equal, therefore, total time steps and total steps are equivalent. The cells are represented as a 5 x 5 grid. The time steps reported are the best score across three runs. Note that the drones can move in continuous space, i.e. it could move to any real number coordinate within the environment.

| Loss function | $\alpha$ | $\beta$ | Time steps |
|---------------|---------|---------|------------|
| $L_1$ loss    | -       | -       | 561        |
| Adaptively choose $\delta$ - 3D | 1 | 1 | 309        |
| Adaptively choose $\delta$ - 3D | 2 | 1 | 371        |
| Adaptively choose $\delta$ - 3D | 4 | 1 | 469        |
| Adaptively choose $\delta$ - 3D | 1 | 2 | 524        |

3) Discussion: The number of time steps taken when using the $L_1$ loss function for the Potential Field algorithm is relatively higher compared to methods of adaptively choosing $\delta$. We could observe that adaptively changing $\delta$ based on parameters $\alpha = 4$ and $\beta = 1$, $\alpha = 1$ and $\beta = 2$ increases the time steps. This is because they aggressively decrease $\delta$ compared to parameters $\alpha = 1$ and $\beta = 1$, $\alpha = 2$ and $\beta = 2$. Decreasing $\delta$ causes the drone to make smaller updates to its position. We could also observe that parameter $\alpha = 1$ and $\beta = 1$ maps the environment faster than other functions.

VI. CONCLUSION & FUTURE WORK

We have performed a comparative analysis on several neural network architectures using RL for performing agent-oriented task assignment based on their effectiveness and efficiency. We couple the task assignment with path planning algorithms and perform obstacle avoidance without global knowledge of the obstacles in the environment and plan locally, i.e. plan actions
and Agile Robotics and Perception Lab (ARPL) @ NYU training could allow us to achieve this. We seek to explore 
avoid obstacles in tandem. We hypothesize that using Graph 
assignment and the path planning task under a single/universal 
solution. This could be alleviated if we could conflate the task 
of obstacles) this leads to poor approximation and hence, a bad 
in certain cases (e.g.: an environment with a very large number 
of obstacles at the edge. We also proposed a method to adaptively change 
their path planning. This is important because it can help 
Ivory, H., Kim, H., and Kim, H. J., “Planning and control for collision-
free cooperative aerial transportation,” IEEE Transactions on 
Automation Science and Engineering, vol. 15, no. 1, pp. 189–201, 2018. 
Mellinger, D., Shomin, M., Michael, N., and Kumar, V., “Cooperative 
Grasping and Transport Using Multiple Quadrupods,” in Distributed 
Autonomous Robotic Systems: The 10th International Symposium, 
Martinoi, A., Mondada, F., Correll, N., Mermoud, G., Egerstedt, M., 
Hsieh, M. A., Parker, L. E., and Stay, K., Eds. Berlin, Heidelberg: 
Springer Berlin Heidelberg, 2013, pp. 545–558. [Online]. Available: 
https://doi.org/10.1007/978-3-642-32723-0_39

10 Achtelik, M., Achtelik, M., Brunet, Y., Chli, M., Chatzichristofis, S., 
Decotignie, J., Doth, K., Frandorfer, F., Kneip, L., Gurdan, D., Heng, L., 
Kosmatopoulos, E., Doitsidis, L., Lee, G. H., Lynen, S., Martinelli, A., 
Meier, L., Pollefeys, M., Piguet, D., Renzaglia, A., Scaramuzza, D., 
Siegwart, R., Stumpf, J., Tanskanen, P., Troiani, C., and Weiss, S., “Sfly: Swarm of micro flying robots,” in 2012 IEEE/RSJ International 
Conference on Intelligent Robots and Systems, 2012, pp. 2649–2650.

11 Saska, M., Vondrak, V., Claudo, J., Thomas, J., Loiano, G., and 
Kumar, V., “Swarm Distribution and Deployment for Cooperative 
Surveillance by Micro-Aerial Vehicles,” Journal of Intelligent & Robotic 
Systems, vol. 84, no. 1, pp. 469–492, Dec. 2016. [Online]. Available: 
https://doi.org/10.1007/s10846-016-0338-z

12 Yang, S., Wang, Y., and Chu, X., “A Survey of Deep Learning Techniques 
for Neural Machine Translation,” arXiv:2002.07526 [cs], Feb. 2020. 
[Online]. Available: http://arxiv.org/abs/2002.07526

13 Galceran, E. and Carreras, M., “A survey on coverage path planning for 
robotics,” Robotics and Autonomous Systems, vol. 61, no. 12, pp. 1258 –
1276, 2013. [Online]. Available: http://www.sciencedirect.com/science/ article/pii/S092188901300167X

14 Zhang, H.-y., Lin, W.-m., and Chen, A.-x., “Path planning for the mobile 
robot: A review,” Symmetry, vol. 10, p. 450, 10 2018.

15 Zhou, A., Kang, L., and Yan, Z., “Solving dynamic tsp with evolutionary 
approach in real time,” in The 2003 Congress on Evolutionary Computation, 
2003. CEC’03., vol. 2. IEEE, 2003, pp. 951–957.

16 Vinyals, O., Fortunato, M., and Jaitly, N. “Pointer Networks,” Jan. 2017. 
[Online]. Available: http://arxiv.org/abs/1506.03134

17 Beimel, R., “Dynamic planning treatment of the travelling salesman 
problem,” J. ACM, vol. 9, no. 1, p. 61–63, Jan. 1962. [Online]. Available: 
https://doi.org/10.1145/321105.321111

18 Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S., “Neural 
Combinatorial Optimization with Reinforcement Learning,” arXiv:1611.09940. 
[Online]. Available: http://arxiv.org/abs/1611.09940

19 Sutton, R. S., McAllester, D. A., Singh, S. P., and Mansour, Y., “Policy 
Gradient Methods for Reinforcement Learning with Function Approxima-
tion,” p. 7.

20 Kool, W., van Hoof, H., and Welling, M., “Attention, Learn to Solve 
Routing Problems!” arXiv:1803.08475 [cs, stat], Feb. 2019, arXiv:1803.08475. 
[Online]. Available: http://arxiv.org/abs/1803.08475

21 Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., 
Kaiser, L., and Polosukhin, I., “Attention Is All You Need,” arXiv:1706.03762 [cs], Dec. 2017, arXiv:1706.03762. [Online]. Available: 
http://arxiv.org/abs/1706.03762

22 Dai, H., Khalil, E. B., Zhang, Y., Dilkina, B., and Song, L., “Learning 
Combinatorial Optimization Algorithms over Graphs,” arXiv:1704.01665 
[cs, stat], Feb. 2018, arXiv:1704.01665. [Online]. Available: 
http://arxiv.org/abs/1704.01665

23 Fox, D., Burgard, W., and Thrun, S., “The dynamic window 
approach to collision avoidance,” IEEE Robotics & Automation 
Magazine, vol. 4, no. 1, pp. 23–33, Mar. 1997. [Online]. Available: 
http://ieeexplore.ieee.org/document/580977/

24 Khamb, O., “Real-time obstacle avoidance for manipulators and mobile 
robots,” in Proceedings. 1985 IEEE International Conference on Robotics 
and Automation, vol. 2, 1985, pp. 500–505.

25 van den Berg, J., Ming Lin, and Manocha, D., “Reciprocal 
Velocity Obstacles for real-time multi-agent navigation,” in 2008 IEEE 
International Conference on Robotics and Automation. Pasadena, 
CA, USA, IEEE, May 2008, pp. 1928–1935. [Online]. Available: 
http://ieeexplore.ieee.org/document/4534389/

26 Budiyanto, A., Cahyadi, A., Adji, T. B., and Wahyunggoro, O., “Uav 
obstacle avoidance using potential field under dynamic environment,” in 
2015 International Conference on Control, Electronics, Renewable Energy 
and Communications (ICCEREC), 2015, pp. 187–192.

27 Williams, R. J., “Simple statistical gradient-following algorithms for 
connectionist reinforcement learning,” Machine Learning, vol. 3, no. 4, 
pp. 229–256, May 1992. [Online]. Available: https://doi.org/10.1007/ 
BF00992596

28 Sutskever, I., Vinyals, O., and Le, Q. V., “Sequence to Sequence 
Learning with Neural Networks,” arXiv:1409.3215 [cs], Dec. 2014, arXiv:1409.3215. [Online]. Available: http://arxiv.org/abs/1409.3215

29 Bahdanau, D., Cho, K., and Bengio, Y., “Neural Machine Translation 
by Jointly Learning to Align and Translate,” arXiv:1409.0473 [cs, stat], 
May 2016, arXiv:1409.0473. [Online]. Available: http://arxiv.org/abs/ 
1409.0473
The Taylor series expansion of $e^x$ and $e^{-x}$ are given in the two equations below.

\[
e^x = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \ldots
\]  

\[
e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \ldots
\]  

\[
\cosh(x) = \frac{e^x + e^{-x}}{2}
\]

\[
\Rightarrow \frac{e^x + e^{-x}}{2} = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots
\]

\[
\therefore \log \cosh(x) = \log(1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots)
\]

Let $\alpha = \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots$

\[
\Rightarrow \log \cosh(x) = \log(1 + \alpha)
\]

Using the Taylor series expansion of $\log(1 + x)$ we approximate the above equation.

\[
\log(1 + \alpha) = \alpha - \frac{\alpha^2}{2} + \frac{\alpha^3}{3} + \ldots
\]

By substituting for $\alpha$

\[
\therefore \log \cosh(x) = \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + \left(\frac{x^2}{2!} + \frac{x^4}{4!} + \ldots\right)^2 + \ldots
\]

The above equation can be approximated to $x^2$ when $-1 < x < 1$.

\[
\therefore \log \cosh(x) \approx x^2 \text{ when } -1 < x < 1
\]