Decentralized Path-Planning in Communication Restricted Environments

I. Introduction

In this article we will consider the problem of controlling a multi drone system in a decentralized fashion. In addition, we want to so this even when the drones are not allowed to communicate with each other. The problem can arise in many real life situations, for example, a team of search robots. We would like a team such as this to be able to navigate difficult environments where communication between themselves and to a central server are unreliable. We would like each drone to be able to manage independently, all the while avoiding collisions. We make two major contributions. We present a novel algorithm for *variable responsibility* ORCA, and provide some theoretical guarantees on time complexity. Second, we design a decentralized algorithm to perform this task, which is the main contribution of this article. Further we derive some theoretical guarantees on the safety of this algorithm. We also discuss the computational load that such an algorithm will impose. Then we discuss the limitations of our work and propose some directions for future work into the problem.

II. Problem Definition

There are n drones and some obstacles in the environment. The drones should always maintain a distance of at least r_{min} between themselves and between a drone and an obstacle. The drones are assumed to have perfect sensor knowledge about the relative positions of the other drones. We also assume that a drone knows its own position. So a drone knows the correct locations of other drones but has no information about the velocity or any higher derivatives.

There is no specified formation, and the drones are free to move with respect to each other. All the drones are told the same final goal region. The control input will be the *velocity*. Our objective is to design a decentralized algorithm which will run on every drone and compute the required velocity for each drone. The algorithm must also be *efficient.* The algorithm should not be *simulating* the motion of every drone.

- D denotes a drone
- d_i is the global position of drone i
- S denotes a point on the trajectory computed by a drone that is not the start or end point
- s denotes the coordinates of S
- G denotes the goal point
- v_i denotes the velocity command given to each drone

III. Single Drone Trajectory Generation

For a single drone, we use the algorithm described in [1]. The algorithm is probabilistically correct and aims to find the optimal one bend trajectory. The one bend trajectory is a special kind of shortest path, such that its length is exactly two edges. We will provide the pseudocode of the algorithm although understanding the same is not a prerequisite for the rest of the article.

The idea is to simplify the RRT alorithm.The Rapidly-Exploring Random Tree (RRT) algorithm is a widely used motion planning technique in robotics and computer science. It efficiently constructs a tree-like structure by iteratively expanding nodes from an initial configuration toward a goal configuration. At each iteration, a random configuration is sampled in the configuration space, and the nearest existing node in the tree is connected to it. This connection is made by extending the tree along a trajectory that respects the system's dynamics and avoids obstacles. The algorithm continues to expand the tree until it connects to the goal configuration.

Algorithm 1 Path Generation by Closed-Loop RRT

```
for t do
    Generate a sample z_s.
    Sort the nodes in the graph by heuristics in ascending order.
    for each node q in the graph, in the sorted order do
        Simulate a trajectory X(t) from q to z_s using the closed-loop system and check the trajectory against constraints in section II.
        if Satisfy the constraints then
            Add the end of X(t) to the graph with q as their predecessor. Break.
        end if
    end for
    for each newly added node q_n do
        Calculate the cost of q_n by adding the length of the trajectory from its predecessor and to the cost of its predecessor.
        Simulate a trajectory from q_n to the goal using the closed-loop system and check it against constraints.
        if Satisfy the constraints then
            Mark q_n as goal reachable.
            Calculate cost-to-go of q_n as the length of the trajectory to the goal.
        end if
    end for
end for
Select the node sequence from the graph with the smallest sum of cost and cost-to-go as the generated path.
```

This algorithm can output paths of arbitrary length. However, we would like to only consider paths which have a length

of 2 edges. Hence we modify this algorithm a bit. When we are checking a new point, we only consider one path, the *root-point-destination*. The algorithm after this modification looks like this.

Algorithm 2 Path Generation Using the Simplified Node Connection Strategy

for t **in** time steps **do**
  Generate a sample waypoint wps $= (x_s, y_s, z_s)$ that satisfies constraints.
  Predict a trajectory from the start to wps using the closed-loop system and check against the constraints in each simulation step.
  if Satisfy the constraints **then**
    Predict a trajectory from wps to the goal using the closed-loop system and check against the constraints.
    if Satisfy the constraints **then**
      Add wps to the graph as a successor of the start node.
    end if
  end if
end for
Select the sample waypoint that results in the shortest total trajectory from the graph.

IV. The Algorithm

We first try to understand why the lack of communication poses such a major problem. The reason is that in a such a scheme where communication is banned, drones cannot inform each other about their future *intentions*, loosely speaking. essentially, a drone *i* has no idea what the other drone *j* is going to do in the future. So it cannot take an evasive action right now.

To deal with the issue, we would like to select a path planning algorithm such that executing it for one drone *also reveals information about the other drones only using their relative positions.*

Consider any one drone. This drone will compute the shortest path that has a length of two edges. Let *D* be the current position of this drone and *G* be the destination. Note that the destination is the same for all drones.This drone is supposed to run the modified RRT algorithm we discussed above. Let us say the drone computes the path *D-S-G*. The central idea is as follows. This drone will now aim for point *S*. It will *also* assume that every other drone is also aiming for the point *S*. This may not be actually true, but we will show that given some conditions on the obstacle field, this idea ensures collision free flight. Now this drone *knows* the *intentions* of the other drones, thus solving the problem that the lack of communication posed.

Formally, let d_i be the position of the i^{th} drone and let r_i be the unit vector pointing to the point S from the i^{th} drone, where S is the optimal *bend* as we described earlier. The control inputs are the velocity. So we say that the velocity of the i^{th} is equal to

$$v_i = \alpha_i r_i$$

for some α_i. Also we say that the i^{th} drone would *prefer* to have a velocity of $\alpha_0 r^i$

$$v_i^{pref} = \alpha_0(d_i - s)$$

where α_0 is a constant that we select.

Our task reduces to computing α_i such that there is no collision. We can do this in a manner similar to the well known ORCA algorithm. ORCA reduces this problem down to a linear program which can be solved efficiently. The *ORCA* constraints can be looked up in the original paper, but to summarize, the constraints ensure that the drones maintain a specified minimum distance between each other. These constraints will be linear in α_i, where α_i is as defined earlier. We will denote the set of ORCA constraints as C_{ORCA}.

There can be other constraints as well. For example

$$|\alpha_i| \leq v_{max}$$

where v_{max} is the maximum allowed speed for a drone.

The objective function for this linear program is

$$J = min|\alpha_0 - \alpha_i|$$

The idea here is to compute a velocity as close to the *preferred* velocity as possible. The entire linear program becomes

$$min|\alpha_0 - \alpha_i|$$

such that

$$|\alpha_i| \leq v_{max} \ \forall 1 \leq i \leq n$$

$$C_{ORCA}$$

Each drone will thus create its own linear program. Let L be the linear program created by drone i. The drone i will compute the values of all α_j to such that the objective function is optimized. The velocity command given to drone i will be

$$v_i = \alpha_i r_i$$

where r_i is as defined as earlier and the value of α_i comes from the linear program.

Algorithm 3 The algorithm for the k^{th} drone

for t **in** time steps **do**
  Compute the optimal path $D - S - G$ of length two.
  for i = **1** to i = **n do**
    s is the coordinate of point S and d_i is the global position of the i^{th} drone
    $r_i = \frac{d_i - s}{\|d_i - s\|}$
    $v_i = \alpha_i r_i$
    $v_i^{pref} = \alpha_0(d_i - s)$
  end for
  Generate the linear program
  Publish velocity command $\alpha_k r_k$
end for

V. Variable Responsibility ORCA

A. Introduction and related work

We have shown how *ORCA* can be used here. However it has one key limitation, The ORCA distributes the responsibility of collision avoidance equally between two drones. Each drone must do at least half of the required avoidance measures. The *responsibility* to avoid the collision is equally divided between two drones. This makes the available solution set much smaller. There might be solutions with an unequal distribution of responsibility that are completely overlooked. We introduce an extension of *ORCA* in an attempt to tackle this problem.

This existing work attempts to compute the *variable responsibility factors* using another optimization problem as in [3]. This second optimization problem can be be computationally expensive to create and solve. We will try design an alternative which requires only one quadratic optimization.

B. Notation

The problem setting is the same as that for *ORCA*. We will still clarify the notation:

- v_i^{pref} denotes the preferred velocity of the i_th drone.
- v_i denotes the velocity of the i_{th} drone.
- u_{ij} will be the same as in $ORCA$. It will denote the *minimum* change in $v_i^{pref} - v_j^{pref}$ required to avoid collision.

C. The optimization problem

The constraints for the $ORCA$ look like

$$\left(v_i - (v_i^{pref} + \frac{1}{2}u_{ij})\right).\mathbf{n} \geq 0$$

We create a constraint corresponding to this

$$\boxed{\left(v_i - v_j - (v_i^{pref} - v_j^{pref} + \frac{1}{2}u_{ij})\right).\mathbf{n} \geq 0}$$

The objective function becomes

$$\boxed{J = min \sum_i ||v_i - v_i^{pref}||^2}$$

D. Correctness

To ensure correctness, it is essential that the solutions computed by each drone are the same or *atleast close* to each other. We will show that it is the case even with *approximate* solving algorithms. Let us assume we have an algorithm A which gives us an approximate solution x such that $J(x)$ is at most ϵ away from the true optimal objective $J(x_{opt}$.

$$|J(x) - J(x_{opt})| \leq \epsilon$$

We observe the the objective function is quadratic, with a hessian

$$H = 2I$$

The Hessian is positive definite, hence we can say that x_{opt} is *unique*. We now consider the solution x_i computed by some drone i.Using the Taylor Series expansion we can write

$$J(x) = J(x_{opt}) + (\nabla_x J)^T(x - x_{opt}) + \frac{1}{2}(x - x_{opt})^T H(x - x_{opt})$$

x_{opt} is the optima, hence

$$(\nabla_x J)^T(x - x_{opt}) \geq 0 \; \forall x \in feasible$$

$$\implies J(x) - J(x_{opt}) \geq 2||x - x_{opt}||^2$$

$$\implies \boxed{||x - x_{opt}|| \leq \sqrt{\frac{\epsilon}{2}}}$$

Let x_i and x_j be the solutions computed by drones x_i and x_j.

$$||x_i - x_j|| = ||x_i - x_{opt} + x_{opt} - x_j||$$

$$\implies ||x_i - x_j|| = ||x_i - x_{opt} - (x_j - x_{opt}||$$

$$\implies ||x_i - x_j|| \leq ||x_i - x_{opt}|| + ||(x_j - x_{opt}||$$

$$\implies ||x_i - x_j|| \leq \sqrt{\frac{\epsilon}{2}} + \sqrt{\frac{\epsilon}{2}}$$

$$implies \boxed{||x_i - x_j|| \leq \sqrt{2\epsilon}}$$

If we have an algorithm which can compute an approximate solution to the optimization problem, then we can ensure that the solutions computed on each drone by this algorithm will be *close* to each other. The solving algorithm can be any nice algorithm, for example, the interior point method.

We observe that the problem setup is the same as that for $ORCA$. So we can directly use the method we have described in place of $ORCA$ in the original swarm control. We only need to replace

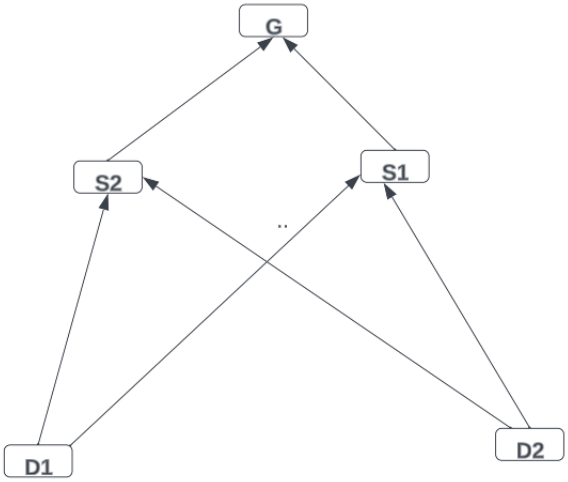
$$v_i = \alpha_i r_i$$

$$v_i^{pref} = \alpha_0(d_i - s)$$

VI. Correctness

We will now attempt to get some conditions on the safety of the algorithm. Consider two drones D_1 and D_2. D_1 computes a path $D_1 - S_1 - G$ and D_2 computes a path $D_2 - S_2 - G$. We assume that these paths are the *optimal* one bend paths. Here optimality is decided using simple the total path length. there are two possible cases. The first case is that the lines $D_1 - S_1$ and $D_2 - S_2$ do not intersect. In this case there is no risk of collision. The more interesting case is when $D_1 - S_1$ and $D_2 - S_2$ intersect. This creates the possibility of a collision.

We will now consider the situations in which a collision can occur. The two drones can only collide if the lines $D_1 - S_1$ and $D_2 - S_2$ intersect. Consider the following diagram :



Since we have assumed S_1 and S_2 to be the optimal of length two edges, we can write

$$D_1S_1 + GS_1 < D_1S_2 + GS_2$$

$$D_2S_2 + GS_2 < D_2S_1 + GS_1$$

Adding the two gives

$$D_1S_1 + D_2S_2 < D_1S_2 + D_2S_1$$

This is a contradiction. Observe that the statement implies that the sum of the diagonals is less than the sum of two opposite sides.
Hence the situation for a collision cannot arise. However the proof rests on two crucial assumptions:

- D_1 can *see* S_2
- D_2 can *see* S_1

These are the assumptions that enable us to write the inequalities we rely on. The inequalities are essentially

$$length(P_1) < length(P_2)$$

However, to write such an inequality *both* the paths P_1 and P_2 must be possible. In effect, these conditions put a limit on how dense the obstacle field can be. If either of two conditions does not hold, then it is possible that there may be a collision, the counterexamples are quite simple to generate.

VII. Why one bend paths?

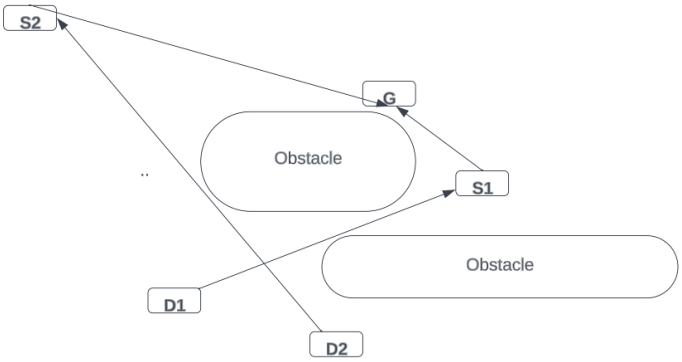
We do not have any direct estimation of the velocities of the other drones. Assuming the conditions specified in the previous section hold, the one bend paths give a way to estimate the worst case action by other drones.

If every drone is suing the same algorithm then it ensured that that the drones can only collide at the *bends*. Since every drone already assumes that all others are also coming at the same bend, such a collision is avoided in the velocity assignment phase. One bend paths reduce the possible collision location to just the bends. Since the bend can be computed by every drone without taking into account the other drones, it is computationally light.

So it is possible to narrow down the possible collision locations in to an easily calculated set even without any communication, if the trajectory generation algorithm is specified.

VIII. Limitations

The safety guarantees depend on the two conditions specified above. Either of the conditions breaking may result in a collision. For example



D_1 can reach S_2 but D_2 can't reach S_1. The violation of the second conditions results in a possible collision.

IX. Future Work

The two conditions we have laid out are sufficient to ensure safety. But there is a possibility that the conditions are too strong. Thus finding better guarantees is a direction for work. Investigation of collision situations reveals that that two-bend paths and in general k-bend paths may be useful for collision avoidance. They may require more computation but may offer safety guarantees. The possibility of using k-bed paths to tighten the conditions is another avenue for research.

References

[1] Yucong Lin and Srikanth Saripalli, *Sampling-Based Path Planning for UAV Collision Avoidance*
[2] Jur van den Berg, Stephen J. Guy, Ming Lin and Dinesh Manocha *Reciprocal n-Body Collision Avoidance*
[3] Ke Guo , Dawei Wang , Tingxiang Fan, and Jia Pan *VR-ORCA: Variable Responsibility Optimal Reciprocal Collision Avoidance*