The International Journal of Robotics Research
35(10):1–32
©The Author(s) 2016
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Swarm Assignment and Trajectory Optimization Using Variable-Swarm, Distributed Auction Assignment and Sequential Convex Programming

Daniel Morgan[1], Giri P. Subramanian[1], Soon-Jo Chung[1], and Fred Y. Hadaegh[1]

Abstract
This paper presents a distributed, guidance and control algorithm for reconfiguring swarms composed of hundreds to thousands of agents with limited communication and computation capabilities. This algorithm solves both the optimal assignment and collision-free trajectory generation for robotic swarms, in an integrated manner, when given the desired shape of the swarm (without pre-assigned terminal positions). The optimal assignment problem is solved using a distributed auction assignment that can vary the number of target positions in the assignment, and the collision-free trajectories are generated using sequential convex programming. Finally, model predictive control is used to solve the assignment and trajectory generation in real time using a receding horizon. The model predictive control formulation uses current state measurements to resolve for the optimal assignment and trajectory. The implementation of the distributed auction algorithm and sequential convex programming using model predictive control produces the Swarm Assignment and Trajectory Optimization (SATO) algorithm that transfers a swarm of robots or vehicles to a desired shape in a distributed fashion. Once the desired shape is uploaded to the swarm, the algorithm determines where each robot goes and how it should get there in a fuel-efficient, collision-free manner. Results of flight experiments using multiple quadcopters show the effectiveness of the proposed SATO algorithm.

1 Introduction

Motion planning, often called guidance in the aerospace community, and feedback control of multi-agent systems have been a major area of research over the past decades. The majority of this work focused on swarm robotics Jadbabaie et al. (2003); Earl and D'Andrea (2005); Kingston and Egerstedt (2010); Milutinović and Lima (2006); Cheah et al. (2009); Zhao et al. (2011); Hsieh et al. (2008); Alonso-Mora et al. (2012); Yu et al. (2015). However, in recent years the idea of multi-agent systems has been extended to spacecraft Scharf et al. (2003, 2004); Alfriend et al. (2009); Breger and How (2007); Vaddi et al. (2005); Campbell (2003, 2005); Zanon and Campbell (2006); Hadaegh et al. (2016). The most recent idea is to fly a swarm containing a large number (hundreds to thousands) of femtosatellites (100-gram-class spacecraft) Hadaegh et al. (2016).

For a swarm to have a cost benefit compared to a monolithic agent or a smaller formation, the individual agents need to be smaller and cheaper. Due to their small size and low cost, the agents in a swarm have limited actuation, communication, and computation capabilities, which require the guidance and control algorithms

[1]University of Illinois at Urbana-Champaign, USA
[2]Jet Propulsion Laboratory, California Institute of Technology, USA

Corresponding author:
Soon-Jo Chung, Department of Aerospace Engineering & Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, Urbana, IL, 61801, USA
Email: sjchung@alum.mit.edu

of the swarm to be both fuel and computationally efficient. Swarms of agents create interesting challenges in guidance and control due to the large number of agents, the small size of each individual agent, and the complicated dynamics. Specifically, the large number of vehicles moving in 3-D makes collision avoidance a major challenge Morgan et al. (2012, 2014). Also, the limited computation and communication capabilities of each agent require the swarm reconfiguration algorithm to be very simple so that it can be run on board each small robot or vehicle in real time.

The swarm reconfiguration problem consists of two parts: assignment and trajectory generation. The assignment problem consists of finding the optimal mapping from a set of agents to a set of targets or tasks in order to minimize the total cost of interest. This problem has been well researched and many methods exist for finding the optimal assignment, including the Hungarian algorithm Kuhn (1955) and iterative methods Bertsekas (1981); Hung (1983). The drawback to many of these algorithms is that they are *centralized* with respect to communication and computation. On the other hand, the auction algorithm Bertsekas and Castanon (1991); Bertsekas (1992) possesses a distributed nature suitable for distributed systems. More recent research on the auction algorithm Zavlanos et al. (2008); Choi et al. (2009) has shown that it can be implemented in a distributed manner. In contrast with the prior work, one important aspect of the assignment algorithm in this paper is its versatility to adapt to information, specifically collision avoidance and disconnected communication networks.

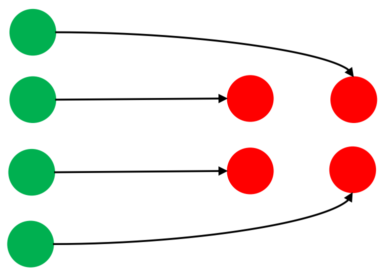
The second part of the swarm reconfiguration is optimal trajectory generation and collision avoidance. This part requires an algorithm that can solve the nonlinear optimization that minimizes the cost of the trajectory while satisfying collision avoidance and dynamic constraints. The trajectory optimization has been solved using a variety of methods, including mixed integer linear programming Richards et al. (2003), pseudospectral methods Ross and Fahroo (2003), convex programming Schulman et al. (2013), and sequential convex programming (SCP) Morgan et al. (2014).

In this paper, we simultaneously solve both the target assignment and trajectory optimization problems using a variable-swarm, distributed auction assignment (VSDAA), which allows the swarm size to change over time, to solve the assignment problem and our prior work on SCP Morgan et al. (2014) to solve the trajectory optimization. Additionally, we integrate these algorithms using model predictive control (MPC) Bemporad and Morari (1999); Mayne et al. (2000) in order to run the algorithms in real time and on swarms with distributed communication networks. The integration with the trajectory optimizer is what allows the auction algorithm the ability to adjust its assignment based on collisions that were previously undetected.

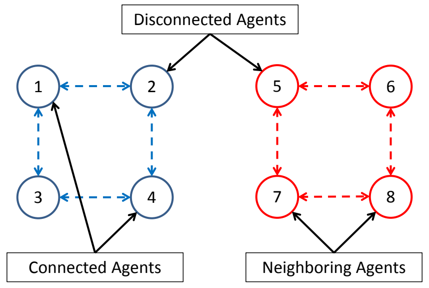
The contributions of this paper, compared with the existing literature including our own work Morgan et al. (2014), are in the development of VSDAA, its integration with trajectory optimization in the MPC-SCP formulation, and the convergence proof for SCP as well as in the experimental validation using multiple quadrotors. The variable-swarm characteristic means that VSDAA can adapt the number of targets in the assignment to match the number of agents. This is incredibly useful when the number of agents in the swarm changes. In the case of a significant loss of agents due to an external object or fuel/battery depletion, VSDAA would adjust the number of targets to match the number of remaining agents and the agents would fill in the gap left by the external object. This allows the swarm to handle the loss of a significant number of agents and still maintain the desired shape. On the other hand, VSDAA will also increase the number of targets if there are more agents than targets. This is a situation that will break a typical auction algorithm Bertsekas and Castanon (1991); Bertsekas (1992); Zavlanos et al. (2008); Choi et al. (2009) since the agents cannot all be assigned to a target so they bid indefinitely. Also, it should be noted that VSDAA can handle an arbitrary distribution shape of targets, as opposed to the use of uniform target distributions Yu et al. (2015).

Another contribution is that the SCP method is shown to converge to a Karush-Kuhn-Tucker (KKT) point Boyd and Vandenberghe (2004); Ruszczynski (2006) of the nonconvex program. This proof shows that once a feasible solution is found, the sequence of optimal solutions resulting from the convex programs will converge. Additionally, the trajectory to which they converge satisfies the KKT conditions, which are necessary conditions for an optimal trajectory.

Additionally, the implementation of VSDAA with SCP using MPC allows an assignment to be achieved even in a disconnected communication network. Since the assignment is updated throughout the reconfiguration, the distance-based, swarm communication network will be different every time an assignment is computed. Therefore, the agents do not need to be fully connected to every other agent at all times. In fact, if two agents from separate, disconnected networks are assigned to the same target, they will eventually move close enough to become connected and will be assigned to different targets.



(a) Visualization of a swarm reconfiguration



(b) Visualization of connected and disconnected networks. Dashed lines represent communication links.

Figure 1. Visualization of problem statement and communication networks

The result of the MPC implementation of VSDAA and SCP is the swarm assignment and trajectory optimization (SATO) algorithm. SATO, the main algorithm of this paper, is distributed in both communication and computation, and provides near-optimal, collision-free trajectories for swarm reconfiguration. In contrast to other methods that simultaneously solve the assignment and trajectory optimization Turpin et al. (2014), SATO uses SCP as underlying trajectory optimization, which allows it to handle complex dynamic environments. Additionally, this algorithm works with disconnected communication networks and is robust to the loss or gain of a significant number of agents. Finally, the model predictive control formulation allows SATO to run on board each agent and provides robustness to unmodeled disturbances. A preliminary version of this paper was previously presented at a conference Morgan et al. (2015), but this paper adds significant results of experimental validation along with detailed description of dynamic modeling and nonlinear control design.

The paper is organized as follows. In Sec. 2, the swarm reconfiguration problem is formulated as a constrained, nonlinear optimal control problem and converted to a nonlinear optimization. In Sec. 3, the swarm reconfiguration problem is broken into an assignment problem and a trajectory optimization problem. Then, the assignment is solved using VSDAA. In Sec. 4, the trajectory optimization problem is converted to a convex optimization and the SCP algorithm is described. Additionally, the SCP algorithm is shown to converge to a trajectory that satisfies the KKT conditions of the nonconvex problem. In Sec. 5, MPC is used to integrate VSDAA and SCP, and to implement a finite horizon so that the resulting algorithm, SATO, can be run on board each agent in real time with a disconnected communication network. In Sec. 6, SATO is run for both a 2-D, double integrator dynamics scenario and a 3-D, relative orbit dynamics scenario. The results of the two scenarios are analyzed and discussed. In Sec. 7, we elaborate on our experimental setup, the feedback control law design to track the desired trajectories obtained by SATO, and the experimental results.

2 Problem Statement

In this section, the optimal swarm reconfiguration is presented as a continuous, finite horizon optimal control problem. The swarm reconfiguration involves the transfer of hundreds to thousands of agents from their current shape to a desired shape while satisfying various constraints, such as collision avoidance, and minimizing the total fuel used during the transfer. A visualization of a swarm reconfiguration and communication networks is shown in Fig. 1.

Figure 1a shows a visualization of the swarm reconfiguration problem. The agents begin at their current positions (green) and move towards the desired formation (red). The agents are interchangeable so any agent can go to any target. In Figure 1b, the connectedness of various agents is shown. The dashed arrows represent communication links between the agents and the different colors represent different communication networks. In other words, the blue agents are connected to the other blue agents (agents 1 and 4) but not to the red agents (agents 2 and 5). Two agents that have a communication link between them are called neighboring agents (agents 7 and 8).

2.1 Nonlinear Optimal Control Problem

The objective of the optimal swarm reconfiguration is to minimize the $\mathcal{L}_1$-norm of the control input. Therefore, we can define the swarm reconfiguration as follows

Problem 1 (Constrained, Nonlinear Optimal Control).

$$\min_{\mathbf{u}_j(t), j=1,\ldots,N} \sum_{j=1}^{N} \int_0^{t_f} \|\mathbf{u}_j(t)\|_q \, dt \quad \text{subject to} \tag{1}$$

$$\dot{\mathbf{x}}_j(t) = \mathbf{f}(\mathbf{x}_j(t)) + B\mathbf{u}_j(t)) \qquad \forall t \in [0, t_f], \qquad j = 1, \ldots, N \tag{2}$$

$$\|\mathbf{u}_j(t)\|_r \leq U_{\max} \qquad \forall t \in [0, t_f], \qquad j = 1, \ldots, N \tag{3}$$

$$\|G[\mathbf{x}_j(t) - \mathbf{x}_i(t)]\|_2 \geq R_{\text{col}} \qquad \forall t \in [0, t_f], \qquad i < j, \qquad j = 1, \ldots, N-1 \tag{4}$$

$$\mathbf{x}_j(0) = \mathbf{x}_{j,0}, \qquad j = 1, \ldots, N \tag{5}$$

$$\mathbf{x}_j(t_f) \in \mathcal{X}_f, \qquad j = 1, \ldots, N \tag{6}$$

where $B = [\mathbf{0}_{3\times3} \quad \mathbf{I}_{3\times3}]^T$, $G = [\mathbf{I}_{3\times3} \quad \mathbf{0}_{3\times3}]$, $\mathbf{x}_j = (\boldsymbol{\ell}_j^T, \dot{\boldsymbol{\ell}}_j^T)^T$, $\boldsymbol{\ell}_j \in \mathbb{R}^n$ is the position vector of agent j, $\mathbf{x}_{j,0}$ is the initial state of agent j, $\mathbf{u}_j$ is the control vector of agent j, and N is the number of agents in the swarm. (2)-(5) represent the dynamics constraint, maximum control constraint, collision avoidance constraint, and initial state constraint, respectively, with $U_{\max}$ being the maximum control magnitude and R_{col} being the minimum allowable distance between two agents. (6) represents the terminal state constraint with $\mathcal{X}_f$ being a set of M discrete points in $\mathbb{R}^n$. This constraint is what introduces the need for solving for the optimal assignment and differentiates this paper from our prior work Morgan et al. (2014) where individual terminal state assignments were given.

Remark 1 (Norms). The norms used in (1) and (3), $\|\cdot\|_q$ and $\|\cdot\|_r$, respectively, are dependent on the hardware used on board the agents. In this paper, we use $q = 1$ and $r = \infty$. However, the convex optimizations are valid for $q, r \in \{1, 2, \infty\}$.

Remark 2 (Fixed Terminal Time). The optimizations used in this paper all have a fixed terminal time. This is due to the fact that the swarm is trying to reconfigure to a specific shape so the agents need to arrive at their terminal positions at the same time. If some agents arrive earlier than others, they will either drift off of their target position or require extra cost to maintain their position in the presence of dynamics. Additionally, the trajectories are generally cheaper for longer reconfiguration times so having some agents arrive earlier than others usually increases the cost for those agents arriving early.

2.2 Convexification of Differential Equations

In this section, the dynamics constraints in (2) are converted to affine equality constraints. This is done by linearizing (2) and discretizing Problem 1. This results in a finite number of linear equality constraints, which are acceptable in a convex programming problem.

In order to rewrite the dynamics in (2) as a constraint that can be used in a convex programming problem, these equations must first be linearized about the nominal trajectory $\mathbf{x}_j^0$. Linearizing (2) yields

$$\dot{\mathbf{x}}_j = A(\mathbf{x}_j^0)\mathbf{x}_j + B\mathbf{u}_j + z(\mathbf{x}_j^0) \tag{7}$$

where $A(\mathbf{x}_j^0) = \left.\dfrac{\partial \mathbf{f}}{\partial \mathbf{x}_j}\right|_{\mathbf{x}_j^0}$ and $z(\mathbf{x}_j^0) = \mathbf{f}(\mathbf{x}_j^0) - \left.\dfrac{\partial \mathbf{f}}{\partial \mathbf{x}_j}\right|_{\mathbf{x}_j^0} \mathbf{x}_j^0$.

The next step in the process of converting (2) into a constraint that can be used in convex programming is to convert the ordinary differential equation in (7) to a finite number of algebraic constraints. In order to do this, the problem is discretized using a zero-order-hold approach such that

$$\mathbf{u}_j(t) = \mathbf{u}_j[k], \qquad t \in [t_k, t_{k+1}), \qquad k = k_0, \ldots, T-1 \tag{8}$$

where $t_f = T\Delta t$, T is the number of discrete time steps, $t_0 = 0$, $t_T = t_f$, and $\Delta t = t_{k+1} - t_k$ for $k = k_0, \ldots, T-1$. This method of discretization reduces (7) to

$$\mathbf{x}_j[k+1] = A_j[k]\mathbf{x}_j[k] + B_j[k]\mathbf{u}_j[k] + z_j[k], \qquad k = k_0, \ldots, T-1, \qquad j = 1, \ldots, N \tag{9}$$

where $\mathbf{x}_j[k] = \mathbf{x}_j(t_k)$, $\mathbf{u}_j[k] = \mathbf{u}_j(t_k)$, and

$$A_j[k] = e^{A(\mathbf{x}_j^0(t_k))\Delta t}, \qquad B_j[k] = \int_0^{\Delta t} e^{A(\mathbf{x}_j^0(t_k))\tau} B \, d\tau, \qquad z_j[k] = \int_0^{\Delta t} e^{A(\mathbf{x}_j^0(t_k))\tau} z(\mathbf{x}_j^0(t_k)) d\tau \tag{10}$$

Now that the nonlinear, continuous-time equations of motion from (2) have been rewritten as linear, finite dimensional constraints in (9), they can be used in a convex programming problem. The constraints from (3)-(6) can be written in discretized form as

$$\|\mathbf{u}_j[k]\|_\infty \leq U_{\max} \qquad k = k_0, \ldots, T-1, \qquad j = 1, \ldots, N \tag{11}$$
$$\|G(\mathbf{x}_j[k] - \mathbf{x}_i[k])\|_2 \geq R_{\text{col}} \qquad k = k_0, \ldots, T, \qquad i < j, \qquad j = 1, \ldots, N-1 \tag{12}$$
$$\mathbf{x}_j[0] = \mathbf{x}_{j,0}, \qquad j = 1, \ldots, N \tag{13}$$
$$\mathbf{x}_j[T] \in \mathcal{X}_f, \qquad j = 1, \ldots, N \tag{14}$$

Note that the only constraints that do not satisfy the requirements of convex programming are (12) and (14). These constraints will be modified in the following sections so that the problem can be efficiently solved using convex programming.

3 Distributed Optimal Target Assignment

In this section, the nonlinear optimal control problem (Problem 1) is broken into two parts: an optimal assignment problem and an optimal trajectory-planning problem. This separation allows us to rewrite the terminal constraints in (14), which are nonconvex and require the problem to include integer variables. By solving an assignment problem to determine the terminal states of each agent, the remaining trajectory-planning problem can be approximated by a convex program and efficiently solved.

Claim 1 (Assignment). If the terminal set ($\mathcal{X}_f$) is a set of points with every pair of points separated by a safe distance (R_{col}), then the constraints $\mathbf{x}_j[T] \in \mathcal{X}_f$ (14) and $\|G(\mathbf{x}_j[T] - \mathbf{x}_i[T])\|_2 \geq R_{\text{col}}$ ((12) at $k = T$) can be equivalently written as

$$\mathbf{x}_j[T] \in \mathcal{X}_f, \qquad \mathbf{x}_j[T] \neq \mathbf{x}_i[T], \qquad \forall j \neq i \tag{15}$$

Now, the assignment problem can be written as shown below.

Problem 2 (Assignment Problem).

$$\min_{\mathbf{x}_{j,f},\ j=1\ldots N} \sum_{j=1}^{N} C(\mathbf{x}_{j,0}, \mathbf{x}_{j,f}) \quad \text{subject to} \tag{16}$$

$$\mathbf{x}_{j,f} \in \mathcal{X}_f, \qquad \mathbf{x}_{j,f} \neq \mathbf{x}_{i,f}, \qquad \forall j = 1 \ldots N, \qquad \forall i \neq j$$

where $C(\mathbf{x}_0, \mathbf{x}_f)$ is the cost required for an agent to go from $\mathbf{x}_0$ to $\mathbf{x}_f$.

The solution to the Assignment Problem (Problem 2) will yield the desired terminal points for each agent ($\mathbf{x}_{j,f}$), which are then used to formulate the following terminal constraint for the trajectory optimization problem.

$$\mathbf{x}_j[T] = \mathbf{x}_{j,f}, \qquad j = 1, \ldots, N \tag{17}$$

The resulting trajectory optimization can be written as follows:

Problem 3 (Trajectory Optimization).

$$\min_{\mathbf{u}_j, j=1,\ldots,N} \sum_{j=1}^{N} \sum_{k=k_0}^{T-1} \|\mathbf{u}_j[k]\|_1 \Delta t \quad \text{subject to} \quad \{(9), (11), (12), (13), (17)\} \tag{18}$$

Remark 3 (Assignment Cost Function). The assignment cost function ($C_{\text{auc}}(\mathbf{x}_0, \mathbf{x}_f)$) should approximate the optimal cost of solving Problem 5 so that the assignment and trajectory optimization are optimizing the same quantity. However, an exact solution to Problem 5 is difficult to obtain when calculating the optimal assignment due to the coupling between agents in the collision avoidance constraint. Therefore, the cost function used in Problem 2 should be the solution to Problem 5 without the collision avoidance constraint. The resulting problem is a decoupled convex program, which can be efficiently solved.

Remark 4 (Motion Primitives and Distance-based Cost for Assignment). In some cases, predetermined motion primitives Paranjape et al. (2015) can be used as the cost function in Problem 2 to further simplify the calculations. These primitives allow the assignment cost to be calculated using simple algebraic functions, which greatly reduces the computation time when compared to solving an optimization for each robot-target pair. For example, when the dynamics are double integrator, a function of the distance between the initial and terminal points can be used as the assignment cost in (16) (e.g., $C(\mathbf{x}_{j,0}, \mathbf{x}_{j,f}) \propto \|G(\mathbf{x}_{j,0} - \mathbf{x}_{j,f})\|_2^{\nu}$).

Remark 5 (Maximum Fuel Available). Since each vehicle (or robot) will have a limited amount of fuel (or energy), some targets might be unreachable. In order to ensure that a robot is not assigned to an unreachable target, every cost that exceeds the available fuel should be set to infinity or some sufficiently large number. This will ensure that every robot can reach its assigned target given its available fuel.

3.1 Distributed Auction Algorithm

To solve the Assignment Problem (Problem 2), an auction algorithm is used. This algorithm is typically used to solve a centralized assignment problem Bertsekas and Castanon (1991); Bertsekas (1992), but its structure allows it to be implemented in a distributed manner Zavlanos et al. (2008) even in some less desirable situations, including limited communication and changes in the number of agents.

In a typical auction algorithm, the computations can be centralized or decentralized, but the current bid prices, current highest bidders, and current bid for each agent must all be stored in shared memory that can be accessed by all of the agents. This requires all-to-all or all-to-one (star) communication, which is an impractical requirement of a large swarm. Additionally, the agents take turns bidding, which requires that each agent knows how many agents are in the swarm.

A distributed auction algorithm, VSDAA, is developed to solve the assignment problem for a swarm containing a varying number of agents that do not have all-to-all or all-to-one communication. VSDAA (Method 1) follows the same steps as a typical auction algorithm with a few exceptions. First, all of the computations are run in parallel and all of variables are stored locally (values for agent i are denoted by superscript i in Method 1). The bid prices of each agent ($\mathbf{p}^i$) are communicated to the neighboring agents and each agent stores the largest price received for each target (line 31). This introduces the possibility that agents are bidding based on outdated prices, but over time all of the agents that are connected on the communication graph will receive updated prices and bid appropriately. Finally, the number of target locations that agent i is bidding on (m^i) can be changed if the number of bids being received by agent i is different than the number of target locations.

In VSDAA (Method 1), each agent runs the algorithm in parallel (line 8) and all computations are done locally. First, we assume that each agent knows the cost ($\mathbf{c}^i(j) = C(\mathbf{x}_{i,0}, \mathcal{X}_f(j))$) of moving from its current location ($\mathbf{x}_{i,0}$) to every target position ($\mathcal{X}_f(j)$). Then, the agents begin the bidding process. Each agent first checks to see if it has been outbid by another agent (line 10). If it has been outbid, it checks to see if the number of bids is equal to the number of targets (m^i) in the assignment (line 12). If this is the case, the agent increases the number of targets used in the assignment by one (line 13) and increases the magnitude of the bid on each target (line 14) so that each agent has the opportunity to bid on the new target. Additionally, every bid is made negative so that every target becomes available. Then, the agent bids on one of the first m^i target positions by choosing the target with the lowest total cost (line 16), which is composed of the fixed, predetermined cost ($\mathbf{c}^i$) and the variable, bidding cost ($\mathbf{p}^i$). Once the desired target (j^i) for agent i is chosen, the bid amount (γ^i) is set as the difference between the cost of the two cheapest targets (v^i and w^i) with the addition of a small increment (ϵ), which ensures that the bid price on any target is strictly increasing. Finally, the bid price of the chosen target is increased by the bid amount (line 20). The agent then resets its counter to zero. If the agent was not outbid, but some other bid has changed, the agent sets its counter to zero but does not go through the bidding calculations. If none of the bids have changed, the agent increases its counter by one (line 26). Finally, the agent stores its current bid estimates ($\mathbf{p}^i_{\text{old}}$), communicates its bid estimates to its neighbors, and updates its current bids based on the bids received from its neighbors. The neighboring agents of agent i are defined by a distance-based communication network so the resulting neighborhood ($\mathcal{N}_{[i]}$) is closed ($i \in \mathcal{N}_{[i]}$) and the communication links are bidirectional ($j \in \mathcal{N}_{[i]}$ if and only if $i \in \mathcal{N}_{[j]}$). The bidding process continues until the agent's counter reaches twice the graph diameter Harary (1994) of the communication network (D_{net}), which guarantees that no agents are bidding and an assignment has been achieved. In Method 1, $|\{\cdot\}|$ is used to denote the cardinality of a set.

Method 1 Variable-Swarm, Distributed Auction Algorithm (VSDAA)

```
1:  $\mathcal{X}_f$ = terminal positions in desired shape
2:  $\mathbf{c}^i(s)$ = cost of agent $i$ choosing target $s$
3:  $m^i$ = # of targets available for agent $i$ to bid on
4:  $\mathbf{p}^i = \mathbf{0}_{1\times m^i}$
5:  $\mathbf{p}^i_{\text{old}} = -\mathbf{1}_{1\times m^i}$
6:  $j^i = 1$
7:  $count^i = 0$
8:  for all $i$ (run in parallel) do
9:    while $count^i < 2D_{\text{net}}$ do
10:     if $|\mathbf{p}^i(j^i)| > \mathbf{p}^i_{\text{old}}(j^i)$ ($i$ is outbid) then
11:       $m^i = \max\left(m^i, |\{s|\mathbf{p}^i(s) \neq 0\}|\right)$
12:       if $|\{s|\mathbf{p}^i(s) > 0\}| = m^i$ then
13:         $m^i = |\{s|\mathbf{p}^i(s) > 0\}| + 1$
14:         $\mathbf{p}^i(1:m^i) = -\left(|\mathbf{p}^i(1:m^i)| + \epsilon\right)$
15:       end if
16:       $v^i = \min_{s=1\dots m^i}\left(\mathbf{c}^i(s) + |\mathbf{p}^i(s)|\right)$
17:       $j^i = \arg\min_{s=1\dots m^i}\left(\mathbf{c}^i(s) + |\mathbf{p}^i(s)|\right)$
18:       $w^i = \min_{s=1\dots m^i, s\neq j^i}\left(\mathbf{c}^i(s) + |\mathbf{p}^i(s)|\right)$
19:       $\gamma^i = w^i - v^i + \epsilon$
20:       $\mathbf{p}^i(j^i) = |\mathbf{p}^i(j^i)| + \gamma^i$
21:       $count^i = 0$
22:     else if $\mathbf{p}^i \neq \mathbf{p}^i_{\text{old}}$ (another agent is outbid) then
23:       $m^i = \max\left(m^i, |\{s|\mathbf{p}^i(s) \neq 0\}|\right)$
24:       $count^i = 0$
25:     else
26:       $count^i = count^i + 1$
27:     end if
28:     $\mathbf{p}^i_{\text{old}} = \mathbf{p}^i$
29:     Communicate $\mathbf{p}^i$ to all agents in $\mathcal{N}_{[i]}$
30:     for $s = 1\dots m^i$ do
31:       $\mathbf{p}^i(s) = \min_{q\in\arg\max_{q\in\mathcal{N}_{[i]}}(|\mathbf{p}^q(s)|)}\left(\mathbf{p}^q(s)\right)$
32:     end for
33:   end while
34:   Optional: $m^i = |\{j|\mathbf{p}^i(j) \neq 0\}|$
35:   Optional: Go back to line 4 and rerun with new $m^i$
36:   $\mathbf{x}_{i,f} = \mathcal{X}_f(j^i)$
37: end for
```

3.2 Properties of Variable-Swarm Distributed Auction Assignment

In addition to distributing the computation and communication requirements of the auction algorithm, Method 1 also allows for the number of target locations to adjust based on the number of agents in the swarm. This is done by only using the bidding information so no additional consensus algorithm is needed to determine the number of agents. This is a very useful property when the number of agents in the swarm is large and the loss of some agents needs to be overcome to achieve the goal.

In Method 1, each agent keeps track of the number of targets it thinks the swarm needs (m^i), which is the same as the number of agents it thinks are in the swarm. The number of targets can be increased during the algorithm (line 13) if an agent is about to place a bid and all of the m^i targets already have been bid upon. Since an agent will only change its bid if it is outbid, every target that has been bid upon will have a bidder at all future times. Therefore, every target having a bid implies that there are m^i agents in the swarm plus the agent that is bidding, and therefore is not assigned to a target. For this reason, m^i is increased to allow enough targets for all of the agents. This can happen as many times as needed provided that m^i does not exceed M, which is the number of targets defined in $\mathcal{X}_f$.

In addition to increasing m^i when there are more agents than expected, the algorithm will also adjust to having fewer agents than expected if it notices that some of the targets have not been bid upon. This case is more complicated because it may not be desirable in certain situations and cannot be detected until the bidding is finished. An assignment with more targets than agents can be solved by any auction algorithm, but in the case of swarm reconfiguration, it may not result in the desired shape if the number of agents is significantly less than the number of targets. Therefore, it may be desirable to either rerun the algorithm with the correct number of targets or update the number of targets needed for the next reconfiguration. This idea is shown in the optional step on line 34 where the agents adjust the number of targets in the assignment to be equal to the number of bids it received. This step is labeled optional because it might not be desirable to decrease the number of targets in certain situations. Specifically, if the number of agents is not expected to change and the communication network may become disconnected. In this case, the algorithm will only see the bids of the agents that it is connected to and it will decrease the number of desired targets accordingly. In other words, the algorithm cannot distinguish between agents it cannot communicate with because they are disconnected and agents that have been lost.

Remark 6 (Desired Targets ($\mathcal{X}_f$)). VSDAA (Method 1) will access the first m^i targets in $\mathcal{X}_f$ based on how many agents it thinks are in the swarm. To achieve a desirable shape, $\mathcal{X}_f$ should have two characteristics. First, the number of targets (M) defined in $\mathcal{X}_f$ should be large enough that m^i never exceeds the number of targets defined. Second, Method 1 will access the first m^i targets of $\mathcal{X}_f$. Therefore, $\mathcal{X}_f$ should be created so that the desired shape is maintained when only the first m^i points are used. For example, if a unit circle defined by $\{x, y\} = \{\cos\theta, \sin\theta\},\ \theta \in [0, 2\pi]$ is the desired shape, $\theta = \{0 : \frac{\pi}{M} : 2\pi\}$ is not desirable since it would result in a semicircle if only half of the targets are used. Instead, $\theta = \{0 : \frac{2\pi}{M} : 2\pi, \frac{\pi}{M} : \frac{\pi}{M} : \frac{(2M-1)\pi}{M}\}$ is desirable since using half the targets would still result in a circle.

The ability of Method 1 to adjust the number of targets in the assignment to match the number of agents in the swarm allows the algorithm to handle a variety of situations that a typical auction algorithm cannot handle. The main benefit of having the algorithm vary the number of targets is in the case where the number of agents is changing. This is a likely scenario when the swarm has a very large number of agents (over 100) because in this situation, the loss of a few agents is acceptable provided that the swarm can maintain its shape. However, the desired targets will need to be reduced so that the desired shape does not have holes in it. Additionally, varying the number of targets also allows the algorithm to run when the communication network is disconnected. In this case, the agents can only communicate with other agents connected to them in the network, which means that as the network changes and more agents can communicate, there might need to be more targets added to the assignment.

In addition to the aforementioned properties, Method 1 maintains the termination and optimality properties of a typical auction algorithm. The proofs in this section are based on those for a typical auction algorithm Zavlanos et al. (2008), but are modified to account for the changing number of agents and the use of negative price values. In the following proposition, we show that Method 1 terminates in a finite number of bidding iterations and determine an upper bound on the number of iterations.

Proposition 1 (Maximum bids). *The maximum number of bidding iterations that can occur in Method 1 is upper bounded by*

$$D_{\text{net}}(N-1)\max_{i=1,\ldots,N}\left(\lceil\frac{\max_{j=1,\ldots,N}\left(\mathbf{c}^i(j)\right)-\min_{j=1,\ldots,N}\left(\mathbf{c}^i(j)\right)}{\epsilon}\rceil\right) \tag{19}$$

where $\lceil\cdot\rceil$ represents the ceiling operator (round up to next integer).

Proof. To determine the maximum number of bids, the worst case scenario is considered. In this scenario, only minimum bids of increment ϵ are placed. Without loss of generality, we order the targets so that the cost vector for each agent i satisfies $\mathbf{c}^i(j+1) >= \mathbf{c}^i(j)$ for all j.

First, we consider the case when there are at least as many targets as agents ($m^i \geq N$). Now, we will consider how many minimum bids each agent can place before N targets become equally attractive. Initially, each agent will bid on its cheapest target ($\mathbf{c}^i(1)$) until it reaches $\mathbf{c}^i(2)$. This will require at most $\lceil\frac{\mathbf{c}^i(2)-\mathbf{c}^i(1)}{\epsilon}\rceil$ bids. Now, we define a new cost ($\bar{\mathbf{c}}^i(j)$), which represents the lowest bid which exceeds the cost of target j.

$$\begin{aligned}\bar{\mathbf{c}}^i(j+1) &= \epsilon\lceil\frac{\mathbf{c}^i(j+1)-\bar{\mathbf{c}}^i(j)}{\epsilon}\rceil+\bar{\mathbf{c}}^i(j) \\ \bar{\mathbf{c}}^i(1) &= \mathbf{c}^i(1)\end{aligned} \tag{20}$$

Next, it will bid on targets 1 and 2 until they reach $\mathbf{c}^i(3)$, which will require at most $2\lceil\frac{\mathbf{c}^i(3)-\bar{\mathbf{c}}^i(2)}{\epsilon}\rceil$ more bids. By continuing with this process until all of the prices are $\mathbf{c}^i(N)$, we say that the maximum number of bids is $\sum_{j=1}^{N-1} j\lceil\frac{\mathbf{c}^i(j+1)-\bar{\mathbf{c}}^i(j)}{\epsilon}\rceil$. This quantity can be bounded as follows

$$\begin{aligned}\sum_{j=1}^{N-1} j\lceil\frac{\mathbf{c}^i(j+1)-\bar{\mathbf{c}}^i(j)}{\epsilon}\rceil &\leq (N-1)\sum_{j=1}^{N-1}\lceil\frac{\mathbf{c}^i(j+1)-\bar{\mathbf{c}}^i(j)}{\epsilon}\rceil \\ &= (N-1)\lceil\frac{\mathbf{c}^i(N)-\mathbf{c}^i(1)}{\epsilon}\rceil \\ &= (N-1)\lceil\frac{\max_{j=1,\ldots,N}\left(\mathbf{c}^i(j)\right)-\min_{j=1,\ldots,N}\left(\mathbf{c}^i(j)\right)}{\epsilon}\rceil\end{aligned} \tag{21}$$

Once an agent has N equally desirable targets it can always find one that it desires regardless of the choices of the other agents. Once a target is bid above $c^i(N)$, there will always be a more desirable option for agent i. Therefore, the bidding must be finished if all of the agents have N equally desirable choices. In the worst case, this requires $(N-1)\max_{i=1,\ldots,N}\left(\lceil\frac{\max_{j=1,\ldots,N}\left(c^i(j)\right)-\min_{j=1,\ldots,N}\left(c^i(j)\right)}{\epsilon}\rceil\right)$. Additionally, each bid can take up to D_{net} iterations to propagate through the communication network. Multiplying, the maximum number of bids by the number of iterations needed to communicate the bids results in the desired condition.

Now we consider $m^i < N$. In this case, there are more agents than targets and m^i will increase as described in line 13. When m^i is increased, the magnitude of the price of every target is increased by ϵ (line 14). However, since every price is increased, the net change in the cost of all of the targets is zero so this step does not affect the number of bidding iterations. Additionally, when an agent increases m^i, it still places a minimum bid on a single target. Therefore, each iteration still increases one of the target prices by the minimum and the bound on the maximum number of iterations still holds. □

In addition to showing that the VSDAA (Method 1) terminates in finite time, we show in the following proposition that the assignment that results from this algorithm is near optimal.

Proposition 2 (Near-Optimal Assignment). *Assuming that the communication network is connected, the assignment achieved by Method 1 (j^i) satisfies the following equation:*

$$\sum_{i=1}^{N}\left(\mathbf{c}^i(j^i)\right) \leq \min_{\tilde{\mathbf{s}}}\left(\sum_{i=1}^{N}\left(\mathbf{c}^i(s^i)\right)\right)+N\epsilon \tag{22}$$

where $\tilde{\mathbf{s}} = (s^1,\ldots,s^N)^T$ can be any assignment mapping of N agents to m targets and $\min_{\tilde{\mathbf{s}}}\left(\sum_{i=1}^{N}\left(\mathbf{c}^i(s^i)\right)\right)$ is the total cost of the optimal assignment. Each element s^i is the target assigned to agent i and $s^i \neq s^q$, $\forall i \neq q$.

Proof. Since Method 1 waits for all bids to be communicated throughout the swarm before terminating and all agents are connected, we now can define $\mathbf{p}(j) = \mathbf{p}^i(j)$ and $m = m^i$ for all i, j. These quantities represent the values for the bidding vector and number of targets, on which all agents agree. After any bid, the bidding process sets the new price to be $\mathbf{p}(j^i) = |\hat{\mathbf{p}}(j^i)| + \gamma^i$ where $\hat{\mathbf{p}}$ is the price vector before a bid is placed on line 20. This equation is equivalent to the following.

$$\begin{aligned}
\mathbf{p}(j^i) &= |\hat{\mathbf{p}}(j^i)| + w^i - v^i + \epsilon \\
\mathbf{p}(j^i) &= |\hat{\mathbf{p}}(j^i)| + \min_{s=1,\dots,m, s\neq j^i} \left(\mathbf{c}^i(s) + |\hat{\mathbf{p}}(s)|\right) - \min_{s=1\dots m} \left(\mathbf{c}^i(s) + |\hat{\mathbf{p}}(s)|\right) + \epsilon \\
\mathbf{p}(j^i) &= |\hat{\mathbf{p}}(j^i)| + \min_{s=1,\dots,m, s\neq j^i} \left(\mathbf{c}^i(s) + |\hat{\mathbf{p}}(s)|\right) - \left(\mathbf{c}^i(j^i) + |\hat{\mathbf{p}}(j^i)|\right) + \epsilon \\
\mathbf{p}(j^i) + \mathbf{c}^i(j^i) &= \min_{s=1\dots m, s\neq j^i} \left(\mathbf{c}^i(s) + |\hat{\mathbf{p}}(s)|\right) + \epsilon \\
\mathbf{p}(j^i) + \mathbf{c}^i(j^i) &\leq \min_{s=1\dots m, s\neq j^i} \left(\mathbf{c}^i(s) + |\mathbf{p}(s)|\right) + \epsilon
\end{aligned} \tag{23}$$

Since $\mathbf{p}(j^i)$ can only increase and is positive at the termination of the algorithm, we can state that after the termination of the Method 1, $\mathbf{c}^i(j^i) + |\mathbf{p}(j^i)| \leq \min_{s=1,\dots,m} \left(\mathbf{c}^i(s) + |\mathbf{p}(s)|\right) + \epsilon$. Now consider the total cost for any assignment $\tilde{\mathbf{s}} = (s^1, \dots, s^N)^T$.

$$\begin{aligned}
\sum_{i=1}^{N} \left(\mathbf{c}^i(s^i)\right) &= \sum_{i=1}^{N} \left(\mathbf{c}^i(s^i) + |\mathbf{p}(s^i)|\right) - \sum_{s=1}^{m} \left(|\mathbf{p}(s)|\right) \\
&\geq \sum_{i=1}^{N} \left(\min_{s=1,\dots,m} \left(\mathbf{c}^i(s) + |\mathbf{p}(s)|\right)\right) - \sum_{s=1}^{m} \left(|\mathbf{p}(s)|\right)
\end{aligned} \tag{24}$$

since summing over $|\mathbf{p}(s)|$ is equivalent for all permutations of s. (24) holds for any assignment so we can write the total cost of the optimal assignment as

$$\begin{aligned}
\min_{\tilde{\mathbf{s}}} \left(\sum_{i=1}^{N} \left(\mathbf{c}^i(s^i)\right)\right) &\geq \sum_{i=1}^{N} \left(\min_{s=1,\dots,m} \left(\mathbf{c}^i(s) + |\mathbf{p}(s)|\right)\right) - \sum_{s=1}^{m} \left(|\mathbf{p}(s)|\right) \\
&\geq \sum_{i=1}^{N} \left(\mathbf{c}^i(j^i) + |\mathbf{p}(j^i)| - \epsilon\right) - \sum_{s=1}^{m} \left(|\mathbf{p}(s)|\right) \\
&= \sum_{i=1}^{N} \left(\mathbf{c}^i(j^i)\right) - N\epsilon
\end{aligned} \tag{25}$$

which is equivalent to (22) and completes the proof. □

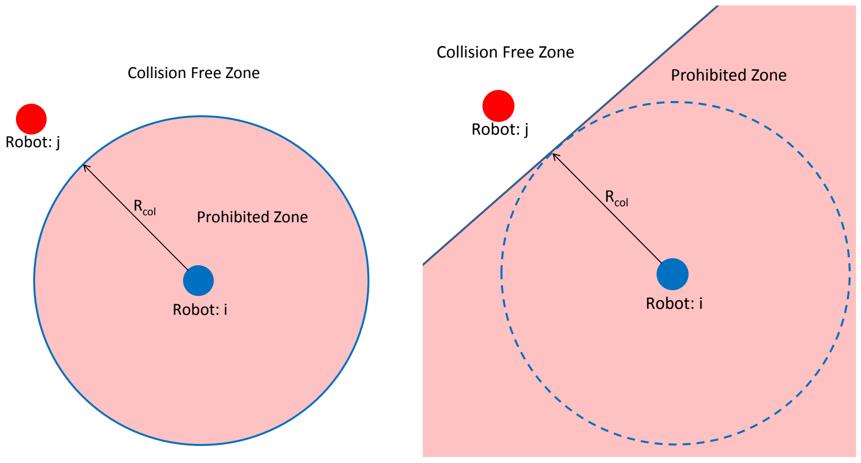
4 Optimal Trajectory Generation

In this section, the optimization problem described in Problem 3 is solved to determine the optimal, collision-free trajectories that take each agent from its current position to its desired target, which was found using VSDAA (Method 1) in the previous section. To solve Problem 3 efficiently, the collision avoidance constraints (12) are convexified and decoupled so that each agent can use SCP to determine its optimal trajectories. The SCP method is described in more detail in Morgan et al. (2014).

4.1 Decoupling and Convexification of Collision Avoidance Constraints

The collision avoidance constraints are dependent on the position of other agents. This makes the optimization coupled in the sense that every agent must know the optimal trajectory of every other agent, which requires the entire optimization to be solved at once. This is undesirable since it requires centralized computations. In order to decouple the optimizations, the nominal trajectories $(\bar{\mathbf{x}}_j)$ are used for the positions of the other agents. These nominal trajectories will be defined in Method 2. Now, each agent can solve its own optimization since the objective and constraints no longer depend on other agents' optimal trajectories.

The decoupled version of Problem 3 is written as the following optimization for each agent $j = 1, \dots, N$.



(a) Nonconvex prohibited zone (b) Convex approximation of prohibited zone

Figure 2. Convexification of the 2-D collision avoidance constraint Morgan et al. (2014)

Problem 4 (Decoupled Optimization).

$$\min_{\mathbf{u}_j} \sum_{k=k_0}^{T-1} \|\mathbf{u}_j[k]\|_1 \Delta t \quad \text{subject to} \tag{26}$$

$$\mathbf{x}_j[k+1] = A_j[k]\mathbf{x}_j[k] + B_j[k]\mathbf{u}_j[k] + z_j[k], \qquad k = k_0, \ldots, T-1 \tag{27}$$
$$\|\mathbf{u}_j[k]\|_\infty \leq U_{\max} \qquad k = k_0, \ldots, T-1 \tag{28}$$
$$\mathbf{x}_j[0] = \mathbf{x}_{j,0} \tag{29}$$
$$\mathbf{x}_j[T] = \mathbf{x}_{j,f} \tag{30}$$
$$\|G(\mathbf{x}_j[k] - \bar{\mathbf{x}}_i[k])\|_2 \geq R_{\text{col}} \qquad k = k_0, \ldots, T, \qquad i \in \{\mathcal{N}_{[j]} \cap \mathcal{P}_j\} \tag{31}$$

where $\mathcal{N}_{[j]}$ is the closed neighborhood of agent j and $\mathcal{P}_j$ is the set of agents that have a higher priority than agent j meaning that j must avoid them. The closed neighborhood ($\mathcal{N}_{[j]}$) will be further defined in Section 5 and the set of higher priority agents ($\mathcal{P}_j$) is further defined in Theorem 8 in Section 4.3. For now, they are defined as $\mathcal{N}_{[j]} = 1, \ldots, N$ and $\mathcal{P}_j = \{i|i < j\}$. It is important to note that either $i \in \mathcal{P}_j$ or $j \in \mathcal{P}_i$ must be true in order to guarantee that agents i and j do not collide.

To use convex programming to solve the trajectory optimization (Problem 4), the nonconvex collision avoidance constraint must be converted to a convex constraint. The best convex approximations of the collision avoidance constraints will be affine constraints. In other words, the circle (2-D) or sphere (3-D) which defines the prohibited region is replaced by a line (2-D) or plane (3-D) which is tangent to the circle (2-D) or sphere (3-D) and perpendicular to the line segment connecting the nominal positions ($\bar{\mathbf{x}}_j$) of the agents. The 2-D version of this idea is shown in Fig. 2.

Figure 2a shows the prohibited zone for the nonconvex collision avoidance constraint. Figure 2b demonstrates the convex approximation of the constraint. As can be seen in Fig. 2b, the new prohibited zone includes the old prohibited zone so the convex collision avoidance constraint is a sufficient condition for the original, nonconvex constraint. Therefore, collision avoidance is still guaranteed using this approximation.

The convex approximation of the nonconvex program in Problem 4 is shown below for agent j Morgan et al. (2014).

Problem 5 (Decentralized Convex Program).

$$\min_{\mathbf{u}_j} \sum_{k=k_0}^{T-1} \|\mathbf{u}_j[k]\|_1 \Delta t \quad \text{subject to} \tag{32}$$

$$\mathbf{x}_j[k+1] = A_j[k]\mathbf{x}_j[k] + B_j[k]\mathbf{u}_j[k] + z_j[k], \qquad k = k_0, \ldots, T-1 \tag{33}$$
$$\|\mathbf{u}_j[k]\|_\infty \leq U_{\max} \qquad k = k_0, \ldots, T-1 \tag{34}$$
$$\mathbf{x}_j[0] = \mathbf{x}_{j,0} \tag{35}$$
$$\mathbf{x}_j[T] = \mathbf{x}_{j,f} \tag{36}$$
$$(\bar{\mathbf{x}}_j[k] - \bar{\mathbf{x}}_i[k])^T G^T G(\mathbf{x}_j[k] - \bar{\mathbf{x}}_i[k]) \geq R_{\text{col}} \|G(\bar{\mathbf{x}}_j[k] - \bar{\mathbf{x}}_i[k])\|_2 \qquad k = k_0, \ldots, T, \qquad i \in \{\mathcal{N}_{[j]} \cap \mathcal{P}_j\} \tag{37}$$

4.2 Sequential Convex Programming

The approximations used to make the trajectory optimization into a convex program, require nominal trajectories ($\bar{\mathbf{x}}_j$) for each robot (or vehicle). Additionally, the nominal trajectories should be close to the actual state trajectories to minimize the approximation error. To ensure that the nominal vectors are good estimates of the actual state vectors, SCP is used. SCP is an iterative method, which solves a convex approximation of a nonconvex problem and uses that solution in the next iteration to convexify the problem, i.e., $\bar{\mathbf{x}}_{j,w}[k] = \mathbf{x}_{j,w-1}[k], \ \forall k, w$ where w is the SCP iteration. This process is repeated until the sequence of trajectories converges according to the following condition.

$$\|\mathbf{x}_{j,w}[k] - \mathbf{x}_{j,w-1}[k]\|_\infty \ < \epsilon_{\text{SCP}}, \ \forall j, k \tag{38}$$

To enforce the collision avoidance constraints, each agent communicates its own nominal trajectory to its neighboring agents.

The SCP method is described in Method 2. First, an initial trajectory is generated for each agent without considering collision avoidance (line 2). Then, the iterative process begins with each agent solving for its optimal trajectory (line 9). Next, each agent stores the current trajectory as the nominal trajectory for the next iteration (line 12) and communicates that trajectory to its neighboring agents (line 13). Finally, the iteration is repeated until the trajectories converge and the agents are collision free (line 14).

Method 2 Sequential Convex Programming Morgan et al. (2014)

```
1: $\bar{\mathbf{x}}_j[k] := \mathbf{0}_{6\times 1}, \ \forall j, k$
2: $\mathbf{x}_{j,0}[k]$ := the solution to Problem 5 (Decentralized Convex Program) with $\mathcal{P}_j = \emptyset, \ \forall j, k$
3: $\bar{\mathbf{x}}_j[k] := \mathbf{x}_{j,0}[k], \ \forall j, k$
4: Communicate $\bar{\mathbf{x}}_j[k]$ to all neighboring agents ($i \in \mathcal{N}_{[j]}$)
5: $\mathcal{K} := \{1, \ldots, N\}$
6: $w := 1$
7: while $\mathcal{K} \neq \emptyset$ do
8:    for all $j \in \mathcal{K}$ (run in parallel) do
9:       $\mathbf{x}_{j,w}[k]$ :=the solution to Problem 5 (Decentralized Convex Program), $\forall k$
10:   end for
11:   for all $j$ (run in parallel) do
12:      $\bar{\mathbf{x}}_j[k] := \mathbf{x}_{j,w}[k], \ \forall k$
13:      Communicate $\bar{\mathbf{x}}_j[k]$ to all neighboring agents ($i \in \mathcal{N}_{[j]}$)
14:      if $\|\mathbf{x}_{j,w}[k] - \mathbf{x}_{j,w-1}[k]\|_\infty < \epsilon_{\text{SCP}} \ \forall k$ and $\|G(\mathbf{x}_{j,w}[k] - \mathbf{x}_{i,w}[k])\|_2 > R_{\text{col}} \ \forall k, \forall i \in \mathcal{N}_{[j]} \cap \mathcal{P}_j$ then
15:         Remove $j$ from $\mathcal{K}$
16:      end if
17:   end for
18:   $w := w + 1$
19: end while
20: $\mathbf{x}_{j,w-1}[k]$ is the approximate solution to Problem 3
```

Because the convex optimizations can be solved efficiently, the run time is now on the order of a time step or two so VSDAA and SCP can be implemented using MPC by updating the future assignments and control commands based on the current state, which may have drifted off the optimal trajectories due to unmodeled disturbances or other errors. MPC can provide some robustness to disturbances and allows the communication network to be distributed and disconnected.

4.3 Convergence of SCP

In this subsection we will show that SCP (Method 2) converges to a point, which satisfies the KKT conditions of the nonconvex optimization in Problem 4. In order to do this, we will show that the sequence of convex programs for agent j converges in two different situations. First, the sequence converges when agent j does not have to avoid any other agents.

Proposition 3 (Convergence without Collision Avoidance). *If $\mathcal{N}_{[j]} \cap \mathcal{P}_j = \emptyset$, the solution to the convex program (Problem 5) for agent j is equivalent to the global minimum of the nonconvex program (Problem 4).*

Proof. Since $\mathcal{N}_{[j]} \cap \mathcal{P}_j = \emptyset$, agent j does not avoid any other agents, which means that there are no collision avoidance constraints. Without collision avoidance constraints, the nonconvex program (Problem 4) and the convex program (Problem 5) are equivalent and the solution to either problem will be a global minimum to both problems. □

Next, we will show that SCP converges for agent j when all of the agents that agent j must avoid have a fixed trajectory, i.e., their trajectories do not change from one iteration to the next. First, we establish that a solution to the convex program will always be a feasible solution to the nonconvex program. To simplify the notation in the following proofs, we introduce the following definition.

Definition 1 (CP($\bar{\mathbf{x}}_j$,$\bar{\mathbf{x}}_i$)). We define CP($\bar{\mathbf{x}}_j$,$\bar{\mathbf{x}}_i$) to be the convex program in Problem 5 where the nominal trajectories $\bar{\mathbf{x}}_j$ and $\bar{\mathbf{x}}_i$ are used for the convexification and decoupling, respectively, of the nonconvex program in Problem 4. The sets $\mathcal{N}_{[j]}$ and $\mathcal{P}_j$ are the same as in the nonconvex program (Problem 4).

Proposition 4 (Feasible Solutions). *If $(\mathbf{x}_{j,w}, \mathbf{u}_{j,w})$ is a feasible solution to* CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$)*, then $(\mathbf{x}_{j,w}, \mathbf{u}_{j,w})$ is a feasible solution to the nonconvex program (Problem 4).*

Proof. Since $(\mathbf{x}_{j,w}, \mathbf{u}_{j,w})$ is a solution to Problem 5, it satisfies all of the constraints of Problem 5. Except for the collision avoidance constraint (31), the constraints of the nonconvex program (27)-(30) are the same as those of the convex program (33)-(36). Additionally, $\mathbf{x}_{j,w}$ satisfies (37). Therefore, the following is true for all $k = k_0, \dots, T$ and $i \in \{\mathcal{N}_{[j]} \cap \mathcal{P}_j\}$:

$$\begin{aligned}(\mathbf{x}_{j,w-1}[k] - \bar{\mathbf{x}}_i[k])^T G^T G(\mathbf{x}_{j,w}[k] - \bar{\mathbf{x}}_i[k]) &\geq R_{\text{col}} \|G(\mathbf{x}_{j,w-1}[k] - \bar{\mathbf{x}}_i[k])\|_2 \\ \|G(\mathbf{x}_{j,w-1}[k] - \bar{\mathbf{x}}_i[k])\|_2 \|G(\mathbf{x}_{j,w}[k] - \bar{\mathbf{x}}_i[k])\|_2 &\geq R_{\text{col}} \|G(\mathbf{x}_{j,w-1}[k] - \bar{\mathbf{x}}_i[k])\|_2 \\ \|G(\mathbf{x}_{j,w}[k] - \bar{\mathbf{x}}_i[k])\|_2 &\geq R_{\text{col}}\end{aligned} \tag{39}$$

The last equation in (39) shows that $\mathbf{x}_{j,w}$ satisfies the collision avoidance constraint in (31). Therefore, all of the constraints in Problem 4 are satisfied and $\mathbf{x}_{j,w}$ is a feasible solution to Problem 4. □

The above proposition shows that any solution to the convex program is a feasible solution to the nonconvex program. Next, this fact will be used to establish that a sequence of optimal solutions exist.

Proposition 5 (Optimal Sequence). *Let $\mathbf{x}_i$ be fixed trajectories for all $i \in \{\mathcal{N}_{[j]} \cap \mathcal{P}_j\}$. If $(\mathbf{x}_{j,w}, \mathbf{u}_{j,w})$ is a feasible solution to* CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$) *or the nonconvex program (Problem 4), then* CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) *has an optimal solution and $(\mathbf{x}_{j,w}, \mathbf{u}_{j,w})$ is a feasible solution to* CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$).

Proof. Since $(\mathbf{x}_{j,w}, \mathbf{u}_{j,w})$ is a solution to CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$) or Problem 4, it satisfies (33)-(36), which are the same as the constraints of CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) except for the collision avoidance constraints (37). Additionally, $\mathbf{x}_{j,w}$ satisfies the collision avoidance constraints of CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$) or Problem 4. If $\mathbf{x}_{j,w}$ satisfies the constraints of Problem 4, the first line of (40) is established, otherwise, this line is established by Proposition 4.

$$\begin{aligned} \|G(\mathbf{x}_{j,w}[k]-\bar{\mathbf{x}}_i[k])\|_2 &\geq R_{\text{col}} \\ \|G(\mathbf{x}_{j,w}[k]-\bar{\mathbf{x}}_i[k])\|_2^2 &\geq R_{\text{col}}\|G(\mathbf{x}_{j,w}[k]-\bar{\mathbf{x}}_i[k])\|_2 \\ (\mathbf{x}_{j,w}[k]-\bar{\mathbf{x}}_i[k])^T G^T G(\mathbf{x}_{j,w}[k]-\bar{\mathbf{x}}_i[k]) &\geq R_{\text{col}}\|G(\mathbf{x}_{j,w}[k]-\bar{\mathbf{x}}_i[k])\|_2 \end{aligned} \tag{40}$$

The last equation in (40) shows that $\mathbf{x}_{j,w}$ satisfies the collision avoidance constraints in CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$). Therefore, ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) is a feasible solution to CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) and the set of feasible solutions to CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) is not empty. Additionally, this set is an intersection of half spaces and equality constraints, which results in a closed set, and the (33)-(35) ensure that the set is bounded. Also, the cost of CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) is continuous. By the Weierstrass theorem Jongen et al. (2004), a continuous function over a closed and bounded set achieves an optimum. Therefore, an optimal solution to CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) exists. □

Proposition 5 shows that once a feasible solution to CP($\mathbf{x}_{j,w}$,$\bar{\mathbf{x}}_i$) exists, all of the following convex problems have an optimal solution and a sequence of optimal solutions $\{\mathbf{x}_{j,w}\}$ exists. The next proposition shows that the optimality of this sequence is improving.

Proposition 6 (Decreasing Cost). *If ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) is the optimal solution to* CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$) *and ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$) is a feasible solution to* CP($\mathbf{x}_{j,w-2}$,$\bar{\mathbf{x}}_i$)*, then*

$$J(\mathbf{u}_{j,w}) \leq J(\mathbf{u}_{j,w-1}) \tag{41}$$

where $J(\mathbf{u}_j) = \sum_{k=k_0}^{T-1} \|\mathbf{u}_j[k]\|_1 \Delta t$. Additionally, if ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) is the unique optimal solution, then either ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) = ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$) or

$$J(\mathbf{u}_{j,w}) < J(\mathbf{u}_{j,w-1}) \tag{42}$$

Proof. Since ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$) is a feasible solution to CP($\mathbf{x}_{j,w-2}$,$\bar{\mathbf{x}}_i$), Proposition 5 states that ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$) is a feasible solution to CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$). Additionally, ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) is the optimal solution to CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$), which means that any ($\mathbf{x}_j, \mathbf{u}_j$) that is a feasible solution of CP($\mathbf{x}_{j,w-1}$,$\bar{\mathbf{x}}_i$) satisfies

$$J(\mathbf{u}_{j,w}) \leq J(\mathbf{u}_j) \tag{43}$$

Substituting ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$) into the right hand side of the above equation establishes (41). If ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) is a unique optimal solution, then (43) has a strict inequality provided that $\mathbf{u}_j \neq \mathbf{u}_{j,w}$. Substituting in ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$) establishes (42) unless ($\mathbf{x}_{j,w}, \mathbf{u}_{j,w}$) = ($\mathbf{x}_{j,w-1}, \mathbf{u}_{j,w-1}$). □

This proposition establishes that a sequence of optimal solutions $\{\mathbf{x}_{j,w}\}$ has a nonincreasing cost and if these solutions are unique, they have a decreasing cost.

We will now use the propositions developed in this section to show that a sequence of optimal solutions exists and converges to an optimal solution of Problem 4. The following theorem establishes these claims.

Theorem 7 (Convergence of SCP). *Let $\mathbf{x}_i$ be fixed trajectories for all $i \in \{\mathcal{N}_{[j]} \cap \mathcal{P}_j\}$. If ($\mathbf{x}_{j,1}, \mathbf{u}_{j,1}$) is a feasible solution to Problem 4, then a sequence of optimal solutions ($\{\mathbf{x}_{j,w}\}, \{\mathbf{u}_{j,w}\}$) exists. If each optimal solution is unique, the sequence converges to ($\mathbf{x}_j^\infty, \mathbf{u}_j^\infty$), which is a KKT point of Problem 4.*

Proof. Since ($\mathbf{x}_{j,1}, \mathbf{u}_{j,1}$) is a feasible solution to Problem 4, it follows from Proposition 5 that CP($\mathbf{x}_{j,1}$,$\bar{\mathbf{x}}_i$) has an optimal solution, which we call ($\mathbf{x}_{j,2}, \mathbf{u}_{j,2}$). Applying Proposition 5 again results in an optimal solution to CP($\mathbf{x}_{j,2}$,$\bar{\mathbf{x}}_i$) defined to be ($\mathbf{x}_{j,3}, \mathbf{u}_{j,3}$). Repeating this process yields the optimal solution sequence ($\{\mathbf{x}_{j,w}\}, \{\mathbf{u}_{j,w}\}$). Since the optimal solutions are unique, applying Proposition 6 to this sequence yields the following condition for all w:

$$J(\mathbf{u}_{j,w}) < J(\mathbf{u}_{j,w-1}) \tag{44}$$

Since every solution in the sequence ($\{\mathbf{x}_{j,w}\}, \{\mathbf{u}_{j,w}\}$) satisfies (33)-(36), which form a closed and bounded set, there is an infinite subsequence ($\{\mathbf{x}_{j,w_i}\}, \{\mathbf{u}_{j,w_i}\}$) that converges. Let the convergence point be called ($\mathbf{x}_j^\infty, \mathbf{u}_j^\infty$). Because the cost function ($J(\mathbf{u})$) is continuous, $J(\mathbf{u}_{j,w_i})$ converges to $J(\mathbf{u}_j^\infty)$. Additionally, the cost function is decreasing as seen in (44). Therefore, the cost function of the entire sequence ($J(\mathbf{u}_{j,w})$) converges to $J(\mathbf{u}_j^\infty)$. In the remainder of this proof, $\rightarrow$ will be used to denote that the sequence on its left converges to value on its right.

We define a mapping $M(\mathbf{x}_j)$ that represents solving CP($\mathbf{x}_j$,$\bar{\mathbf{x}}_i$). The mapping M has a fixed point at $\mathbf{x}_j^\infty$. We will show that this is true by contradiction. Assume that $(\mathbf{x}_j^{\infty+1}, \mathbf{u}_j^{\infty+1}) = M(\mathbf{x}_j^\infty)$ and $(\mathbf{x}_j^{\infty+1}, \mathbf{u}_j^{\infty+1}) \neq (\mathbf{x}_j^\infty, \mathbf{u}_j^\infty)$. In this case, $J(\mathbf{u}_{j,\infty+1}) < J(\mathbf{u}_j^\infty)$. However, $\{J(\mathbf{u}_{j,w})\} \rightarrow J(\mathbf{u}_j^\infty)$ so we have, for $w > \infty + 1$, that $J(\mathbf{u}_{j,w}) < J(\mathbf{u}_j^\infty)$ and $\{J(\mathbf{u}_{j,w})\} \rightarrow J(\mathbf{u}_j^\infty)$, which is a contradiction. Therefore, $(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) = M(\mathbf{x}_j^\infty)$.

Additionally, the mapping $M(\mathbf{x}_j)$ is equivalent to solving the KKT conditions of CP($\mathbf{x}_j$,$\bar{\mathbf{x}}_i$), which are continuous with respect to $\mathbf{x}_j$. Therefore, the mapping M is continuous. Since the subsequence $(\{\mathbf{x}_{j,w_i}\}, \{\mathbf{u}_{j,w_i}\}) \rightarrow (\mathbf{x}_j^\infty, \mathbf{u}_j^\infty)$ and M is continuous, the following is true.

$$\{M(\mathbf{x}_{j,w_i})\} \rightarrow M(\mathbf{x}_j^\infty) \tag{45}$$

Additionally, $\mathbf{x}_j^\infty$ is a fixed point and $\mathbf{x}_{j,w_i+1} = M(\mathbf{x}_{j,w_i})$. Therefore,

$$\{\mathbf{x}_{j,w_i+1}\} \rightarrow \mathbf{x}_j^\infty \tag{46}$$

This process can be repeated to show that all subsequences $\{\mathbf{x}_{j,w_i+n}\}$ converge to $\mathbf{x}_j^\infty$. Therefore, the sequence $\{\mathbf{x}_{j,w}\}$ converges to $\mathbf{x}_j^\infty$.

Finally, we will show that $(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty)$ is a KKT point of Problem 4. Since $\mathbf{x}_j^\infty$ is a fixed point of M, it is a solution to CP($\mathbf{x}_j^\infty$,$\bar{\mathbf{x}}_i$) and from Proposition 4, it is a feasible solution to Problem 4. Additionally, Problem 5 is convex so any solution to this problem is a KKT point $(\mathbf{x}_j^*)$ and satisfies stationarity (47), complementary slackness (48), and dual feasibility (49)-(50):

$$\begin{aligned}
&0 = \partial J(\mathbf{u}_j^*) + (\lambda_1^*)^T \partial g_1(\mathbf{x}_j^*, \mathbf{u}_j^*) + (\lambda_2^*)^T \partial g_2(\mathbf{x}_j^*, \mathbf{u}_j^*) + (\lambda_{\text{eq}}^*)^T \partial g_{\text{eq}}(\mathbf{x}_j^*, \mathbf{u}_j^*) &(47)\\
&0 = (\lambda_1^*)^T g_1(\mathbf{x}_j^*, \mathbf{u}_j^*) + (\lambda_2^*)^T g_2(\mathbf{x}_j^*, \mathbf{u}_j^*) &(48)\\
&\lambda_1^* \geq \mathbf{0} &(49)\\
&\lambda_2^* \geq \mathbf{0} &(50)
\end{aligned}$$

where

$$\begin{aligned}
g_1(\mathbf{x}_j, \mathbf{u}_j) &= \|\mathbf{u}_j[k]\|_\infty - U_{\max}, \quad \forall k = k_0, \ldots, T &(51)\\
g_2(\mathbf{x}_j, \mathbf{u}_j) &= R_{\text{col}} \|G(\mathbf{x}_j^*[k] - \bar{\mathbf{x}}_i)\|_2 - (\mathbf{x}_j^*[k] - \bar{\mathbf{x}}_i)^T G^T G (\mathbf{x}_j[k] - \bar{\mathbf{x}}_i), \quad \forall k = k_0, \ldots, T &(52)\\
g_{\text{eq}}(\mathbf{x}_j, \mathbf{u}_j) &= \begin{bmatrix} \mathbf{x}_j[k+1] - A_j[k]\mathbf{x}_j[k] - B_j[k]\mathbf{u}_j[k] - s_j[k], \quad \forall k = k_0, \ldots, T-1 \\ \mathbf{x}_j[0] - \mathbf{x}_{j,0} \\ \mathbf{x}_j[T] - \mathbf{x}_{j,f} \end{bmatrix} &(53)
\end{aligned}$$

We note that the cost J and the constraints in g_1 and g_{eq} are the same in both Problems 4 and 5. Therefore, we will only substitute (52) for g_2 in the KKT conditions. Since $(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty)$ is a KKT point of CP($\mathbf{x}_j^\infty$,$\bar{\mathbf{x}}_i$), the following equations hold.

$$\begin{aligned}
0 = {} & \partial J(\mathbf{u}_j^\infty) + (\lambda_1^\infty)^T \partial g_1(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) - (\lambda_2^\infty)^T (\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)^T G^T G \\
& + (\lambda_{\text{eq}}^\infty)^T \partial g_{\text{eq}}(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) &(54)\\
0 = {} & (\lambda_1^\infty)^T g_1(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) + (\lambda_2^\infty)^T (R_{\text{col}} - \|G(\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)\|_2) \|G(\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)\|_2 &(55)\\
\lambda_1^\infty & \geq \mathbf{0} &(56)\\
\lambda_2^\infty & \geq \mathbf{0} &(57)
\end{aligned}$$

Now let $\lambda_1^* = \lambda_1^\infty$, $\lambda_2^* = \lambda_2^\infty \|G(\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)\|_2$, and $\lambda_{\text{eq}}^* = \lambda_{\text{eq}}^\infty$, where $(\lambda_1^\infty, \lambda_2^\infty, \lambda_{\text{eq}}^\infty)$ are the KKT multipliers that satisfy the KKT conditions for the convex program. (54)-(57) become the KKT conditions for Problem 4.

$$0 = \partial J(\mathbf{u}_j^\infty) + (\lambda_1^*)^T \partial g_1(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) - (\lambda_2^*)^T \frac{(\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)^T G^T G}{\|G(\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)\|_2} + (\lambda_{\text{eq}}^*)^T \partial g_{\text{eq}}(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) \tag{58}$$

$$0 = (\lambda_1^*)^T g_1(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty) + (\lambda_2^*)^T (R_{\text{col}} - \|G(\mathbf{x}_j^\infty[k] - \bar{\mathbf{x}}_i)\|_2) \tag{59}$$

$$\lambda_1^* \geq \mathbf{0} \tag{60}$$

$$\lambda_2^* \geq \mathbf{0} \tag{61}$$

Therefore, $(\mathbf{x}_j^\infty, \mathbf{u}_j^\infty)$ satisfies the KKT conditions for the nonconvex program (Problem 4) with lagrange multipliers, $(\lambda_1^*, \lambda_2^*, \lambda_{\text{eq}}^*)$. □

Proposition 3 and Theorem 7 show that the SCP algorithm (Algorithm 2) applied to the convex program (Problem 5) converges to a trajectory that satisfies the KKT conditions of the nonconvex program (Problem 4). Proposition 3 applies when there are no collision avoidance constraints ($\mathcal{N}_{[j]} \cap \mathcal{P}_j = \emptyset$) and Theorem 7 applies when the agents that need to be avoided have a fixed trajectory ($\mathbf{x}_{i,w} = \mathbf{x}_i^\infty, \ \forall w, \forall i \in \mathcal{N}_{[j]} \cap \mathcal{P}_j$). In order to guarantees that the SCP algorithm for every agent converges, a priority value (ρ_j) is defined for each agent and is used to construct $\mathcal{P}_j$.

Theorem 8 (Convergence of All Agents). *Let each agent j have a priority value ρ_j such that $\rho_i \neq \rho_j$ for any $i \neq j$. Define the priority set as follows:*

$$\mathcal{P}_j = \{i | \rho_i < \rho_j\} \tag{62}$$

If there is a feasible solution to Problem 4 for each agent and $\mathcal{N}_{[j]} = 1, \ldots, N$, all of the agents will converge to a KKT solution of Problem 4 and no collisions occur.

Proof. Since $\rho_i \neq \rho_j$ for any $i \neq j$, there exists an agent j_1 such that $\rho_{j_1} < \rho_j$ for all $j \neq j_1$. This will result in $\mathcal{P}_{j_1} = \emptyset$. Using Proposition 3, agent j_1 converges to a solution that is a global minimum of Problem 4 and therefore, satisfies the KKT conditions. This solution is defined as $\mathbf{x}_{j_1}^*$.

Now there exists an agent j_2 such that $\rho_{j_2} < \rho_j$ for all $j \neq \{j_1, j_2\}$ and $\rho_{j_1} < \rho_{j_2}$. This implies that $\mathcal{P}_{j_2} = j_1$. Now, Theorem 7 can be applied, using the assumption that there is a feasible solution and the fact that $\mathbf{x}_{j_1}^*$ is fixed, to show that the SCP algorithm for agent j_2 converges to a solution ($\mathbf{x}_{j_2}^*$) that is a KKT point of Problem 4. This step can be repeated for all of the remaining agents to show that every agent's trajectory converges to a KKT point.

Additionally, since $i \in \mathcal{P}_j$ if and only if $\rho_i < \rho_j$ we can state that exactly one of the following statements is true for all pairs (i, j): $i \in \mathcal{P}_j$ or $j \in \mathcal{P}_i$. Since either i or j will avoid the other one, the collision is accounted for in the optimizations. Since this is true for all agent pairs, every possible collision is considered in the optimizations so no collisions will occur. □

Remark 7 (Priority Value (ρ_j)). The priority values (ρ_j) described in Theorem 8 should be defined by a physical parameter that reflects the ability of each agent to avoid other agents. The most obvious choice for this value is the fuel/battery remaining for each agent. Using this quantity, the agent with more fuel/battery remaining would avoid the collision. Also, the priority values should remain constant throughout a complete reconfiguration for Theorems 7 and 8 to hold. However, they can change from one reconfiguration formation shape to the next.

Remark 8 (SCP Computational Complexity). The number of robots affects the number of collision avoidance constraints. When the swarm has all-to-all communication, the number of constraints scales linearly with the number of robots so the complexity is $O(N)$. However, when robots can only communicate with neighboring robots, the number of constraints scales linearly with the density of robots so the complexity is $O(N(R_{\text{comm}}/L)^d)$ where d is the dimension of the physical space and L is the size of the physical space in any dimension.

5 Swarm Assignment and Trajectory Optimization

In this section, MPC is used to implement the VSDAA and SCP methods in real time creating the swarm assignment and trajectory optimization (SATO) algorithm. Additionally, the communication requirement in the swarm is reduced so that the network does not need to be centralized or even connected for the algorithm to work. MPC uses a receding horizon to update the optimal assignments and trajectories based on the current state information. Once an optimal control sequence or optimal assignment is calculated, those values are used until updated values are calculated.

5.1 Model Predictive Control Formulation

SATO updates the state ($\mathbf{x}_{j,k_0}$) and time (k_0) and uses this information as the initial conditions in the optimization. Additionally, a receding horizon, which is T_H time steps long, is used to reduce the size of the optimizations so that they can be run more frequently. Additionally, we will consider communication networks that are distributed and disconnected. With these ideas in mind, each agent will only consider collision avoidance with their neighboring agents and only for the length of the horizon. This allows us to reduce the size of the optimizations without hindering the safety of the agents by considering future collision avoidance in future optimizations. To incorporate these ideas into SATO we modify the collision avoidance constraint in (37) as follows.

$$\begin{gathered}(\bar{\mathbf{x}}_j[k] - \bar{\mathbf{x}}_i[k])^T G^T G(\mathbf{x}_j[k] - \bar{\mathbf{x}}_i[k]) \geq R_{\text{col}} \|G(\bar{\mathbf{x}}_j[k] - \bar{\mathbf{x}}_i[k])\|_2 \\ k = k_0, \ldots, \min\{k_0 + T_H, T\}, \qquad i \in \mathcal{N}_{[j]} \cap \mathcal{P}_j\end{gathered} \tag{63}$$

where

$$\mathcal{N}_{[j]} = \{i | \; \|\mathbf{x}_j[k_0] - \mathbf{x}_i[k_0]\|_2 \leq R_{\text{comm}}\} \tag{64}$$

and R_{comm} is the communication radius of each agent.

Since we are only considering collisions between neighboring agents, it is important to guarantee that no agents can start in positions where they cannot communicate and end up colliding before a new trajectory is generated. To do this, an artificial velocity constraint is introduced so that a bound can be placed on the distance two agents can move relative to each other in a limited amount of time. This constraint is shown below.

$$\|H\mathbf{x}_j[k]\|_2 \leq V_{\max} \qquad k = k_0, \ldots, T \tag{65}$$

where $H = [0_{3\times3} \quad I_{3\times3}]$.

Next, we define two modifications of Problem 5. The first modification is used in SATO to estimate the costs used in VSDAA. The problem is defined as follows.

Problem 6 (Auction Cost).

$$\min_{\mathbf{u}_j} \sum_{k=k_0}^{T-1} \|\mathbf{u}_j[k]\|_1 \Delta t \quad \text{subject to} \quad \{(9), (11), (13), (65)\}, \text{ and } \mathbf{x}_j[T] = \mathcal{X}_f(j) \tag{66}$$

It is important to note that this problem does not include collision avoidance. This is due to the fact that collisions are highly dependent on the assignment, which has not been determined when this problem is used. Additionally, the number of collisions will generally be small for the optimal assignment so excluding them should not have a large effect on the cost.

The other modified problem that will be introduced will be used when SATO uses SCP. This problem is defined below.

Problem 7 (Limited Horizon SCP).

$$\min_{\mathbf{u}_j} \sum_{k=k_0}^{T-1} \|\mathbf{u}_j[k]\|_1 \Delta t \quad \text{subject to} \quad \{(9), (11), (13), (17), (63), (65)\} \tag{67}$$

This problem uses the new collision avoidance constraint, which considers collisions only in the short term, with respect to both position and time.

SATO is described in Method 3. First, the auction costs are generated for each agent going to each target position (line 5). Then, VSDAA is used to compute the optimal assignment (line 9). Next, SCP is used to generate the optimal trajectory and corresponding control sequence (line 14). Finally, the optimal control sequence is applied until a new optimal control is generated (line 16). The current time and state are then updated and the process is repeated until the terminal time is reached.

Method 3 Swarm Assignment and Trajectory Optimization (SATO)

```
1:  $k_0 = 0$
2:  while $k_0 \leq T$ do
3:    for all $i = 1, \dots, N$ $(parallel)$ do
4:      for all $j = 1, \dots, M$ do
5:        Solve Problem 6 using SCP (Method 2)
6:        $\mathbf{c}^i(j)$ = cost of optimal solution to Problem 6
7:      end for
8:    end for
9:    Solve Problem 2 using VSDAA (Method 1)
10:   $\mathbf{x}_{j,f}$ = solution to Problem 2, $\forall j$
11:   if # of bids has changed then
12:     $k_0 = 0$
13:   end if
14:   Solve Problem 7 using SCP (Method 2)
15:   $\mathbf{u}_j[k]$ = control solution to Problem 7, $\forall j$, $k = k_0 \dots k_0 + T_H - 1$
16:   Apply $\mathbf{u}_j[k]$ for $k = k_0 \dots k_0 + T_H - 1$
17:   Update $k_0$ and $\mathbf{x}_{j,k_0}$ to current time
18: end while
```

After the optimal assignment is computed, the algorithm checks to see if the number of targets that have been bid upon has changed from the previous iteration. If the number of bids has changed, that means that the number of agents in the communication network has changed. This can result in significantly different assignments due to new information about other agents. To prevent the new assignment from creating an infeasible trajectory optimization, the current time or the terminal time is reset to give each agent enough time to go to its new target.

Remark 9 (Extend the Terminal Time). If the optimization is infeasible due to maximum velocity or control constraints, the final time T can be extended to $T + n_t \Delta T$, where the integer value n_t increases from 1 until the optimization becomes feasible. Then the agents can communicate the new terminal time. The ability to bound the terminal time is largely dependent on the problem formulation, i.e., dynamics, cost function, constraints. A bound on maximum terminal time could be calculated by upper bounding the distance between agents and targets, and determining the time to cover that distance given the velocity and control constraints.

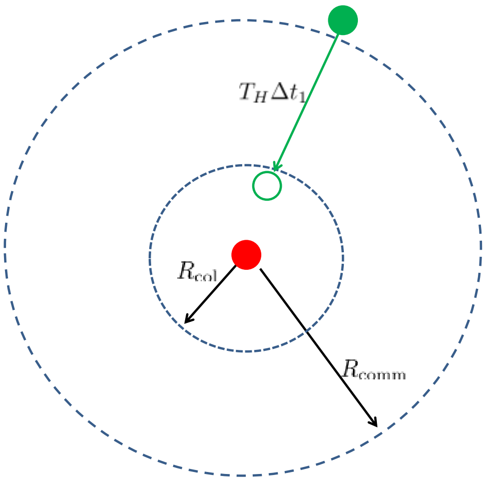
5.2 Disconnected Communication Network

A typical auction algorithm cannot achieve a proper assignment when the communication network is disconnected. Without the assumption of a connected network, the proof of Proposition 2 breaks down. In fact, we cannot even guarantee that every target has at most one agent assigned to it since the bid of one agent cannot reach all of the other agents. However, we will show that using the MPC implementation contained in SATO allows the assignment algorithm to achieve an optimal assignment. First, we develop a condition that guarantees that two robots, which cannot communicate, will not collide before the end of the MPC horizon.

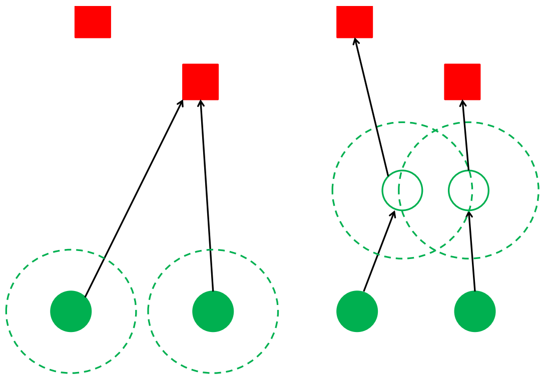
Proposition 9 (Detectable Collisions Morgan et al. (2014)). *If two agents cannot communicate, they will not collide before the end of the current horizon if*

$$R_{\text{comm}} \geq 2V_{\max} T_H \Delta t + R_{\text{col}} \quad \text{and} \quad \Delta t V_{\max} < R_{\text{col}} \tag{68}$$

Proof. The amount of time in the current horizon is the number of time steps (T_H) multiplied by the time step size (Δt). Since the relative velocity is bounded by $2V_{\max}$, the maximum change in the relative distance between



(a) Illustration of an undetectable collision



(b) Resolution of an assignment conflict

Figure 3. Visualization of an undetectable collision and an assignment conflict being resolved

two agents is $2V_{\max}T_H\Delta t$. Therefore, this distance must be less than the difference between the communication radius (R_{comm}) and the collision radius (R_{col}). This establishes the first inequality of (68). Also, the second condition in (68) ensures collision-free motion during two consecutive time steps k and $k+1$. $\square$

This condition guarantees that any agent that could potentially cause a collision before the end of the MPC horizon is detected and therefore considered in the optimization. An illustration of a pair of robots that violate this condition is shown in Fig. 3a. Furthermore, the drift of each agent off of its desired trajectory can be bounded based on the robust nonlinear tracking control law design given in Section 7 and measurement/modelling errors. This error bound can then be used to increase the collision radius (R_{col})-see Morgan et al. (2014) for details.

While Proposition 9 was created in our prior work Morgan et al. (2014) with respect to trajectory optimizations, it also applies to the assignment part of SATO. This condition guarantees that before two agents, which are in disconnected communication networks and assigned to the same target, collide, a new assignment will be run with those agents being able to communicate. At this point, VSDAA guarantees that they will be assigned to different targets and that within their connected network, the assignment will be optimal. This idea allows each connected network to perform its own assignment, which will be optimal. If there is no conflict between the networks, no two agents in different networks have bid on the same target so all of the current bids are the highest bids and the solution is optimal. On the other hand, if there is a conflict, the conflicting agents will go to the same target resulting in them being able to communicate before they collide. At this point, they are on the same communication network and will resolve the conflict. Figure 3b shows an assignment conflict being resolved once the agents can communicate. When the agents are at the solid circles, they cannot communicate and they choose the same target. However, as they move towards the target, they reach the open circles and become connected. At this point, they will be assigned to different target, which resolves the conflict.

Although each individual conflict will eventually be resolved, there is no guarantee that the new assignment does not cause a conflict with another robot that is outside of the communication radius. Once again, this conflict will be resolved, but the resolution could cause another conflict. This process will not repeat indefinitely due to the bound on the number of bids that can be placed in the auction algorithm, but an individual vehicle can run out of fuel before all of the conflicts are resolved. The most likely scenario where this can occur is when one vehicle is moving back and forth between two disconnected networks. In this case, a single vehicle essentially becomes a messenger between the two networks and must physically move between the networks to transfer the bids. This vehicle runs the risk of using all of its fuel before an optimal assignment is achieved. Additionally, if the maximum available fuel is used to set the cost of unreachable targets to infinity, the problem can become infeasible since some of the vehicles are using fuel to move between the networks.

This problem is more likely to occur when the communication network is disconnected and targets are far apart from one another. The simplest solution is to ensure that the targets are sufficiently close together so that the target swarm has a connected communication network. The other solution is to decrease the probability that the communication network can become disconnected. The probability of having a connected communication network is dependent on the number of agents and the size of the communication radius relative to the volume in which the robots are located. The following equation (Corollary 6 in Yu et al. (2015)) determines the number of agents required to achieve a certain probability of connectedness.

$$N \geq \lceil \frac{\sqrt{5}}{R_{\text{comm}}} \rceil^2 \log \left[\frac{1}{1-p} \left(\frac{1}{2} \lceil \frac{\sqrt{5}}{R_{\text{comm}}} \rceil^2 + \lceil \frac{\sqrt{5}}{R_{\text{comm}}} \rceil \right) \right] \tag{69}$$

where N is the number of randomly distributed agents with communication radius (R_{comm}) required to achieve a connected network with probability p in a 2-D unit square. While this bound applies specifically to a 2-D unit square, the general trends are the same: Having more agents and larger communication radii increases the probability of a connected network.

6 Simulation Results

In this section, we apply SATO to swarms in two different dynamic environments. First, we apply SATO to a 2-D environment with double integrator dynamics. This simulation represents the most common problem in the swarm robotics community and can be used to compare SATO to the existing algorithms in that field. In the second simulation, we apply SATO to a swarm of spacecraft in 3-D using the high-fidelity dynamic models of spacecraft in low Earth orbit. This simulation shows that SATO can be used even when the dynamics are nonlinear and state dependent.

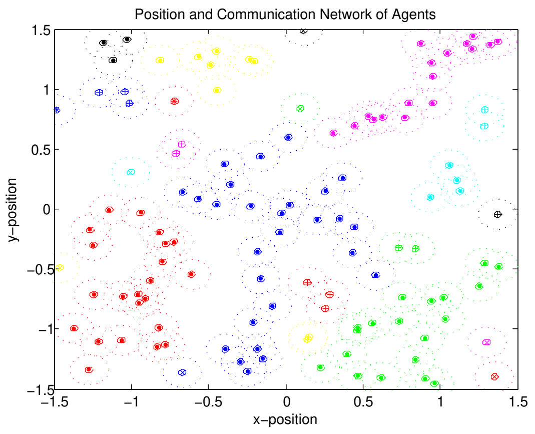
6.1 Simulation with 2-D Double Integrator Dynamics

In the 2-D double integrator simulation, a swarm of 123 agents is randomly initialized and undergoes four reconfigurations forming the letters 'U', 'I', 'U', and 'C', respectively. As discussed in Remark 4, we use the squared distance between the initial and the terminal points as the assignment cost function. During the first reconfiguration, the communication network is initially disconnected since the swarm is randomly distributed. As the agents begin to form the 'U', they move closer to each other and the communication network becomes more connected. As the agents become more connected conflicting assignments are resolved and the swarm achieves the desired shape. After the second reconfiguration ('I'-shape), a significant number of agents are lost causing the communication network to become disconnected. As the agents reconfigure from 'I' to 'U', they still believe there are 123 agents in the swarm, which results in some holes in the target shape. However, after completing the 'U', the agents adjust the number of targets in the assignment based on the number of bids they received while reconfiguring from 'I' to 'U'. This is seen when the swarm successfully achieves a 'C' without any holes. The results of the simulation are shown in Figs. 4 and 5.

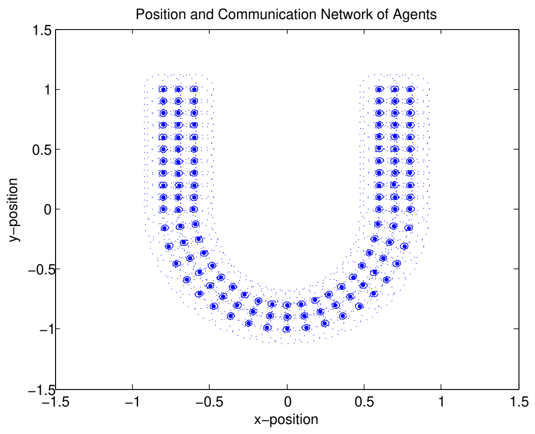
Figure 4 shows the swarm shape after each reconfiguration. Each agent is shown with a given marker shape and color, which denotes its communication network. Agents with the same shape and color are connected and agents with different shapes or colors are not connected. Additionally, the dotted circle around each agent represents half of the agents communication radius. Therefore, any agents whose communication circles touch can communicate with each other.

Figure 4a shows the initial swarm distribution and the disconnected communication networks due to the random distribution. Figure 4b shows the 'U'-shaped swarm after the first reconfiguration. In Fig. 4c, the 'I'-shaped swarm that occurs after the second reconfiguration is shown. Figure 4d shows the 'I'-shaped formation after many of the agents have been lost. This loss of agents results in two disconnected groups of agents (red and blue). In Fig. 4e, the swarm forms a 'U' shape but due to the loss of agents there are holes in the formation. Finally, Fig. 4f shows the 'C' shape swarm after the final reconfiguration. Although the 'C' is not complete due to the loss of agents, VSDAA adjusts the number of targets in the assignment so that there are no large gaps in the desired shape.

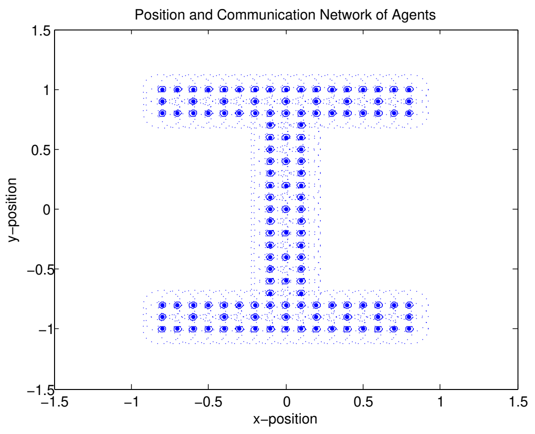
Figure 5 shows several time instances of the first reconfiguration. In this reconfiguration, a random swarm transforms into a 'U' shape. As with Fig. 4, the connected agents are shown with the same color and shape while



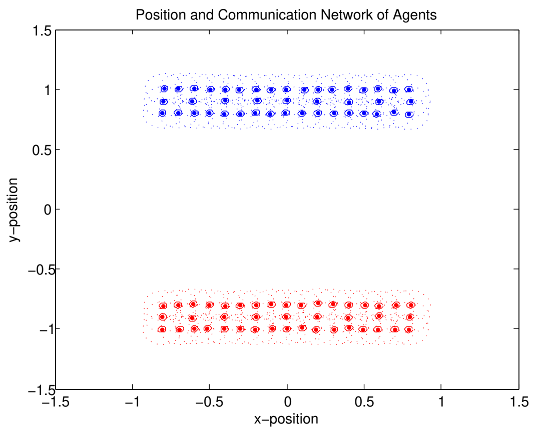
(a) Initial swarm (k = 0)



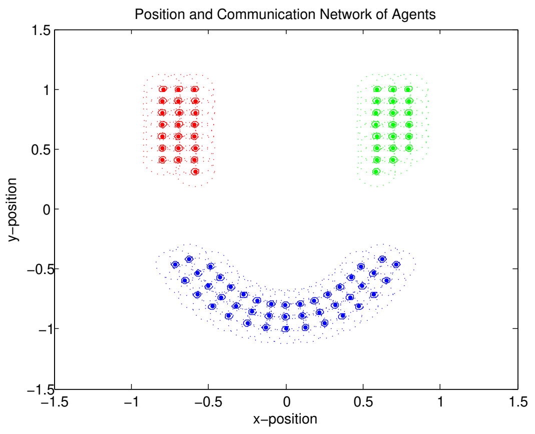
(b) 'U'-shaped swarm (k = 34)



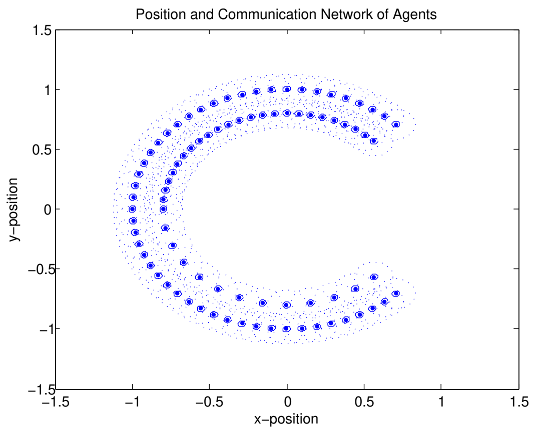
(c) 'I'-shaped swarm (k = 56)



(d) 'I'-shaped swarm after loss of agents (k = 57)

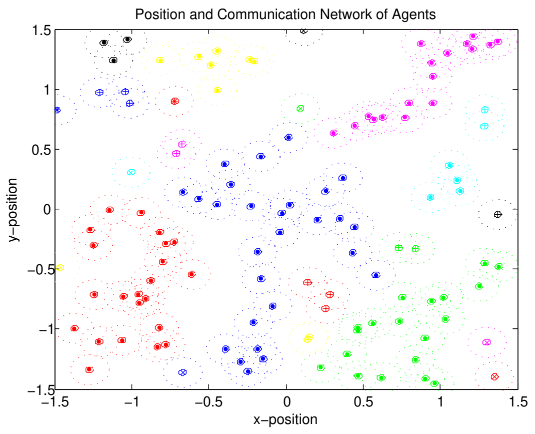


(e) 'U'-shaped swarm (k = 76)

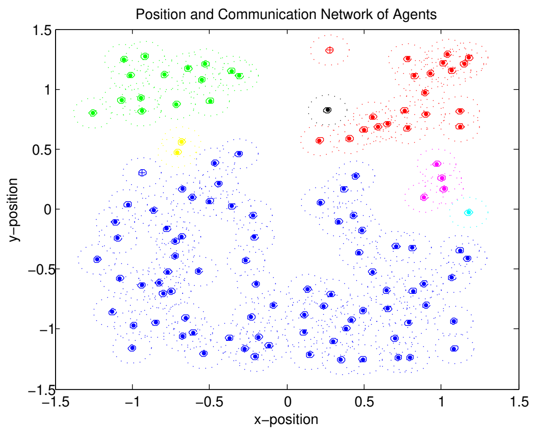


(f) 'C'-shaped swarm (k = 104)

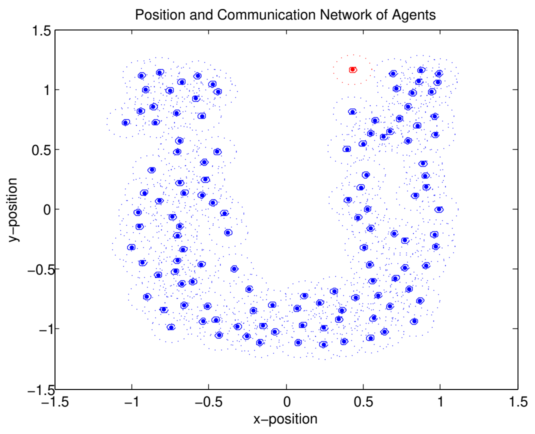
Figure 4. Various time instances in the UIUC reconfiguration simulation. See Extension 1.



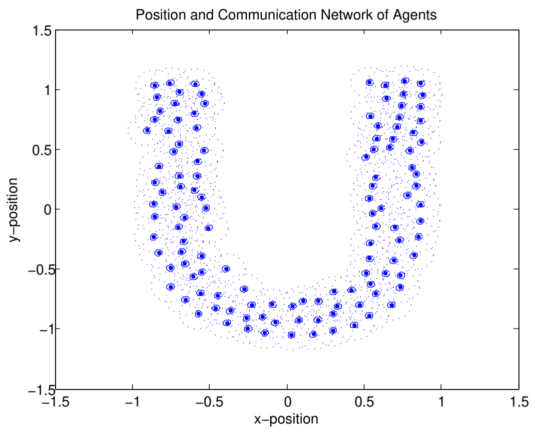
(a) k = 0



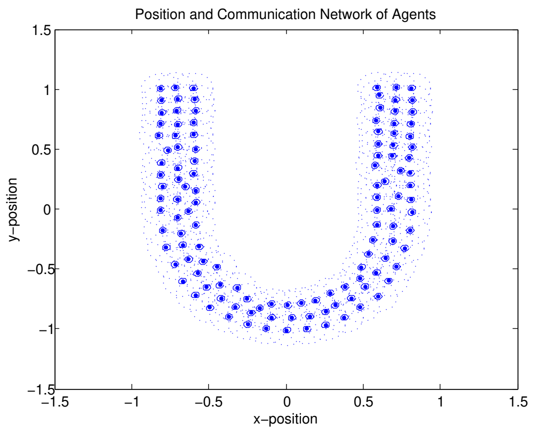
(b) k = 7



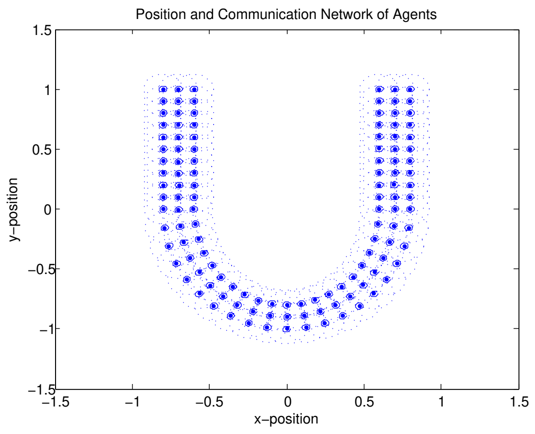
(c) k = 14



(d) k = 21



(e) k = 28



(f) k = 34

Figure 5. Various time instances in the reconfiguration simulation from a random swarm to a ‘U’ shape. See Extension 1.

the dotted circle represents half of the communication radius. Figure 5 shows the evolution of the communication network as the swarm converges to its desired shape.

Figure 5a shows the initial swarm with many, disconnected communication networks. In Fig. 5b, over half of the swarm is connected (blue) and the agents are roughly starting to form a 'U'. In Fig. 5c, all but one of the agents are connected so the assignment is nearly complete and the agents are moving towards their targets. Figures 5d-f show the movement of the swarm once the agents are all connected and the assignment is complete. These figures show the trajectory optimization part of SATO as the agents reach their desired positions in Fig. 5f.

6.2 Simulation with 3-D Relative Orbital Dynamics with J_2

The objective of this simulation study is to show that the SATO algorithm can effectively handle complex dynamic models defined for 3-D motions. In the 3-D spacecraft swarm simulation, the agents are randomly initialized and reconfigured to form a circle. In this simulation, the relative orbital dynamics with J_2 Morgan et al. (2012) are used as the dynamics constraints when SATO calls Problems 6 and 7. The initial and terminal positions are random and circular, respectively, and the velocities are generated using J_2-invariant conditions from our prior work Morgan et al. (2012). Additionally, the random initial positions and the target circle are both located in the x (radial)-y (alongtrack) plane, but the trajectories are allowed to move out of plane. We place the initial and terminal conditions with this constraint so that the plots can be more easily interpreted. However, SATO is equally effective when the initial and terminal positions also have out-of-plane components.

Figure 6 shows the reconfiguration of a swarm of 48 spacecraft from a random distribution to a circular formation. The reference orbit used for this simulation is a circular orbit with a semimajor axis of 6878 km, and an inclination of 45 degrees. The reconfiguration occurs in a quarter of an orbit, which is equal to about 24 minutes. The time step size (Δt) is one minute, the communication radius (R_{comm}) is 2 km, and the collision radius (R_{col}) is 50 m.

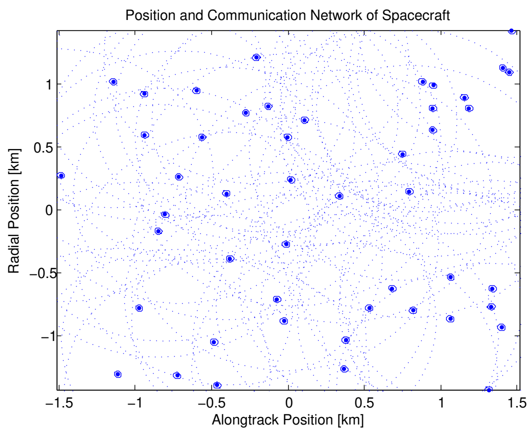
Figure 6a shows the initial swarm distribution where the spacecraft are randomly distributed. Figures 6b-d show the reconfiguration after 8, 16, and 24 minutes, respectively. In Fig. 6d, the swarm has achieved its desired formation of a circle. The results of this simulation show that SATO still achieves the optimal assignment and trajectory generation when the dynamics are as complicated as the relative dynamics of a spacecraft in low Earth orbit.

7 Hardware Experimental Results using Nonlinear Tracking Controller

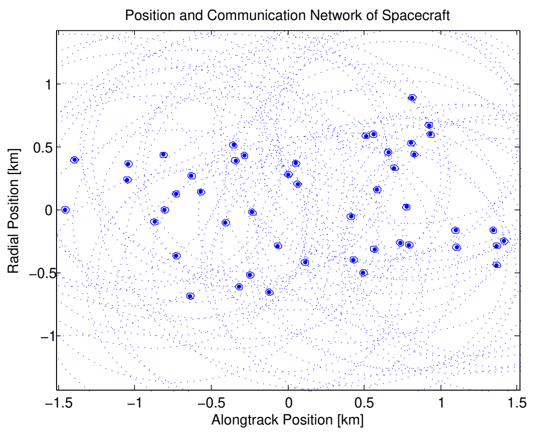
We first describe the position controller and the nonlinear attitude tracking controller that are used to track the desired position trajectories computed by SATO. The experimental results from flight tests are presented in Section 7.3. The algorithm was tested on our formation flying testbed at the University of Illinois at Urbana-Champaign. The experimental setup is composed of sixteen VICON motion capture cameras, the communication dongles for sending signals to the agents and a single computer for implementing the flight control algorithms. The computer uses a 3.4 GHz Intel Core i7 chip with 12 GB of DDR3 RAM and runs on Windows 8. The communication dongle used is a Crayradio 2.4GHz radio USB dongle which connects to the computer via a USB hub. Each dongle communicates with one quadrotor at a data rate of 250Kbps. The entire software back-end (except the motion capture system) was developed in-house (in Python 2.7) and integrates the path planner, controller, motion capture system, and radio communication into a single package. The quadrotor vehicle used for the experiments are called 'Crazyflies', sold by the company Bitcraze, and can be obtained commercially (see Fig. 7). This platform is open-source which provides us with access to the onboard attitude sensors and actuators and hence, allowing us to design our own flight control strategies for the quadrotor.

7.1 Robust Nonlinear Attitude Tracking Control

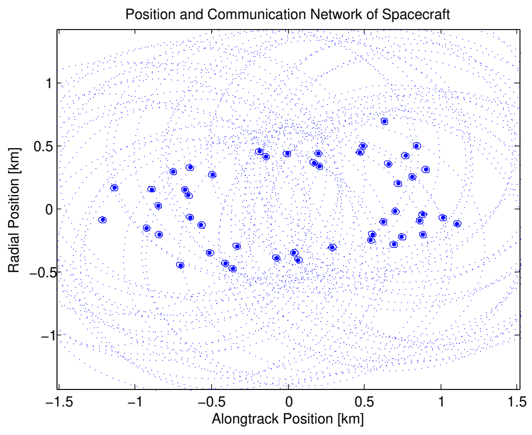
The control law is broken down into two parts, namely, the feedback attitude control loop (inner loop) and the feedback position control loop (outer loop) as shown in Fig. 7. While there has been a lot of controllers designed for quadcopters, many of them tend to linearize the dynamic model about hover or some other points and try to apply a controller on the linearized model. Even though this might be good for slow and gradual movements, it is not ideal for demanding trajectories and aggressive maneuvers. The nonlinear flight control law given in this section utilizes the full dynamic model of the quadrotor while being able to guarantee global exponential



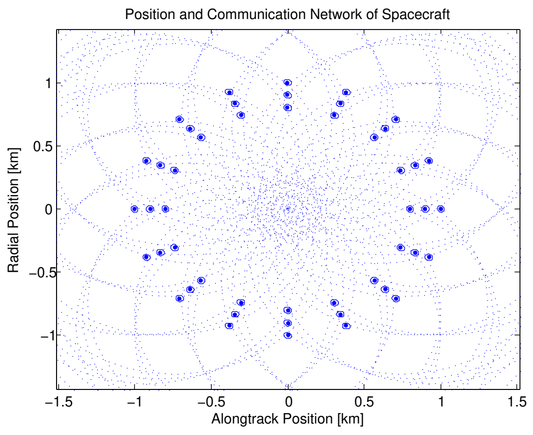
(a) k = 0



(b) k = 8



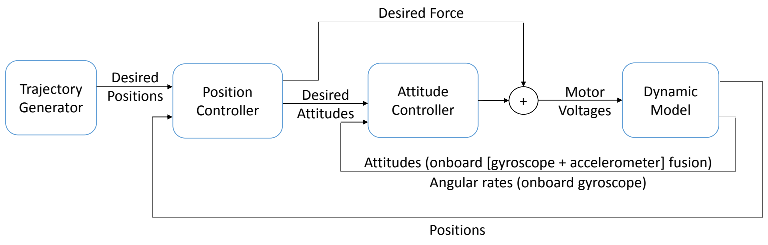
(c) k = 16



(d) k = 24

Figure 6. Various time instances in the reconfiguration simulation of a swarm of spacecraft in the LVLH frame. The out-of-plane (z) motion is not shown in this figure.

(a) Crazyflie quadrotor



(b) Block diagram of quadrotor control

Figure 7. The position controller takes the current position and desired positions to generate the total desired force and the attitude commands which are tracked by the nonlinear attitude controller (71).

stability and bounded tracking errors with respect to bounded disturbance. This stability result allows us to tightly bound and control the size of the trajectory error for collision-free motion planning.

The attitude dynamics and the corresponding kinematics equation of a rigid body are given as

$$\boldsymbol{J}_{\text{tot}}\dot{\boldsymbol{\omega}} = (\boldsymbol{J}_{\text{tot}}\boldsymbol{\omega}) \times \boldsymbol{\omega} + \boldsymbol{u} + \boldsymbol{d}_{\text{ext}}, \qquad \dot{\boldsymbol{q}} = \boldsymbol{Z}(\boldsymbol{q})\boldsymbol{\omega} \tag{70}$$

where $\boldsymbol{q}$ is a vector of three-dimensional attitude representation (e.g., Euler angles, classical Rodrigues parameters, first three elements of quaternions, or modified Rodrigues parameters) with the appropriate definition of $\boldsymbol{Z}(\boldsymbol{q}) \in \mathbb{R}^{3\times3}$. Also, $\boldsymbol{J}_{\text{tot}}$ is the inertia matrix, $\boldsymbol{\omega}$ is the angular rate of the body frame with respect to the inertial frame, the vector $\boldsymbol{u} = (u_x, u_y, u_z)^T$ is the control torque in the body axis, and $\boldsymbol{d}_{\text{ext}}$ denotes the external disturbance torque.

The following robust nonlinear tracking control law for the attitude dynamics (70) is used:

$$\begin{aligned} \boldsymbol{u} &= \boldsymbol{J}_{\text{tot}}\dot{\boldsymbol{\omega}}_r - (\boldsymbol{J}_{\text{tot}}\boldsymbol{\omega}) \times \boldsymbol{\omega}_r - \boldsymbol{K}(\boldsymbol{\omega} - \boldsymbol{\omega}_r) \\ \boldsymbol{\omega}_r &= \boldsymbol{Z}^{-1}(\boldsymbol{q})\dot{\boldsymbol{q}}_d(t) + \boldsymbol{Z}^{-1}(\boldsymbol{q})\boldsymbol{\Lambda}(\boldsymbol{q}_d(t) - \boldsymbol{q}) \end{aligned} \tag{71}$$

where the positive-definite matrix $\boldsymbol{K} \in \mathbb{R}^{3\times3}$ is the feedback gain, $\boldsymbol{\Lambda} \in \mathbb{R}^{3\times3}$ is a positive-definite matrix, and $\boldsymbol{q}_d(t)$ is the time-varying desired (reference) trajectory. Note that $(\boldsymbol{J}_{\text{tot}}\boldsymbol{\omega}) \times \boldsymbol{\omega}$ in the attitude dynamics (70) is not cancelled exactly in order to reduce the effect of $(\Delta\boldsymbol{J}_{\text{tot}}\boldsymbol{\omega}) \times \boldsymbol{\omega}$ with an error from an estimated inertia matrix, $\Delta\boldsymbol{J}_{\text{tot}}$.

The stability proof of (71), obtained by following the standard setup detailed in our prior work Chung et al. (2013); Bandyopadhyay et al. (2016), indicates that all system trajectories converge exponentially fast to a single trajectory regardless of initial conditions with a rate given by $\lambda_{\text{conv,robust}} = \frac{\lambda_{\min}(\boldsymbol{K})}{\lambda_{\max}(\boldsymbol{J}_{\text{tot}})}$, where $\lambda_{\min}(\cdot)$ and $\lambda_{\max}(\cdot)$ are the smallest and the largest eigenvalues respectively. Then the smallest path integral $R(t) = \int_{\boldsymbol{q}_d}^{\boldsymbol{q}} \|\delta\boldsymbol{y}\|_2$ exponentially converges to the following error ball Chung et al. (2013):

$$\lim_{t\to\infty} R(t) \le \sup_t \frac{\lambda_{\max}(\boldsymbol{J}_{\text{tot}})\|\boldsymbol{d}(t)_{\text{ext}}\|_2}{\lambda_{\min}(\boldsymbol{\Lambda})\lambda_{\min}(\mathbf{K})\lambda_{\min}(\boldsymbol{J}_{\text{tot}})}. \tag{72}$$

This ensures that the speed of convergence to any desired time-varying trajectory $\mathbf{q}_d(t)$ is exponentially fast and that the controller can perform agile maneuvers starting from any initial condition with a predicted uncertainty bound. Note that the system with bounded disturbance $\boldsymbol{d}_{\text{ext}}$ is also input-to-state stable (ISS) and finite-gain $\mathcal{L}_p$ stable because of the global exponential stability of the unperturbed system.

If the constant bias term in $\boldsymbol{d}(t)_{\text{ext}}$ is larger than the time-varying term, the advantage of (71) is its straightforward extension to an integral control law: $\boldsymbol{u} = \boldsymbol{J}_{\text{tot}}\dot{\boldsymbol{\omega}}_r - (\boldsymbol{J}_{\text{tot}}\boldsymbol{\omega}) \times \boldsymbol{\omega}_r - \boldsymbol{K}(\boldsymbol{\omega} - \boldsymbol{\omega}_r) - \int_0^t \boldsymbol{K}_I(\boldsymbol{\omega} - \boldsymbol{\omega}_r)dt$, whose stability analysis is detailed in Bandyopadhyay et al. (2016). Now that the attitude of the quadrotor can be commanded, the inner loop of Fig. 7 is complete. The next step of the controller design is to develop the outer control loop of Fig. 7, i.e., the position controller.

7.2 *Position Tracking Controller*

The position control law forms the outer-loop, which generates the desired attitude trajectory $\boldsymbol{q}_d(t)$ for the attitude tracking controller presented in Sec. 7.1 as a function of the position errors ($\mathbf{x}$-$\mathbf{x}_d$). The desired vehicle trajectory $\mathbf{x}_d$ and the estimated vehicle position $\mathbf{x}$ are computed in the local reference frame (e.g., local obstacle map) that is evolving in the absence of a global map and $\mathbf{x}_d$ is provided by the algorithms presented in Sec. 5. We used Euler angles for $\boldsymbol{q}$ in our experiments.

The translational motion along the three axes can be given as

$$m\ddot{x} = \overline{X}, \; m\ddot{y} = \overline{Y}, \; m\ddot{z} = -mg + \overline{Z} \tag{73}$$

where $\overline{X} = F(\cos\psi\cos\phi\sin\theta + \sin\psi\sin\phi)$, $\overline{Y} = F(\sin\psi\cos\phi\sin\theta - \cos\psi\sin\phi)$, $\overline{Z} = F\cos\phi\cos\theta$, g is the acceleration due to gravity, and m is the mass of the quadrotor. Also, (ϕ, θ, ψ) are the Euler angles of the quadrotor and F is the total force generated by the quadrotor. In order to control the position of the quadrotor, a PID controller to track the desired position trajectory along each of the three axes is sufficient (e.g., $\overline{Z} = m\left(g + \ddot{z}_d + K_{D_z}(\dot{z}_d - \dot{z}) + K_{P_z}(z_d - z) + K_{I_z}\int(z_d - z)dt\right)$). The PID controller (with weight cancellation for the z-axis) yields the closed-loop dynamics in linear form, thereby yielding an exact method of computing the gains for global exponential convergence to the desired position trajectory.

Since the yaw angle (ψ) can be decoupled from the linear motion of the quadrotor, i.e., the quadrotor can yaw while holding its position, a separate PID controller can be applied to control it to the desired yaw angle (ψ_d). Hence, the vertical force and desired attitude for (71) are given as:

$$F_d = \sqrt{\overline{X}^2 + \overline{Y}^2 + \overline{Z}^2},\ \ \phi_d = \sin^{-1}(\frac{\overline{X}\sin\psi_d - \overline{Y}\cos\psi_d}{F_d}),\ \ \theta_d = \tan^{-1}\left(\frac{\overline{X}\cos\psi_d + \overline{Y}\sin\psi_d}{\overline{Z}}\right) \tag{74}$$

These attitude and desired force commands are then tracked by the inner attitude control loop thus completing the entire control algorithm as shown in Fig. 7. As a result, the squared values of four motor RPMs $\boldsymbol{n^2} = (n_1^2, n_2^2, n_3^2, n_4^2)^T$ can be computed from

$$\boldsymbol{n^2} = \boldsymbol{B}^{-1}(F_d, \boldsymbol{u}^T)^T \tag{75}$$

where F_d is the desired force from (74) and the vector $\boldsymbol{u} = (u_x, u_y, u_z)^T$ is the control torque from the attitude control law (71). Also, the 4×4 matrix $\boldsymbol{B}$ is a function of the thrust (C_T) and the power coefficients (C_P), which were measured in the aerodynamics test facility at UIUC Deters et al. (2014) (see Subramanian (2015) for the plots of C_T and C_P).

7.3 Flight Test and Experimental Results using SATO and Position/Attitude Tracking Control

In this section, the swarm assignment and trajectory generation are simultaneously solved for 4 agents and 4 targets to generate the desired position trajectories in an MPC fashion. These position trajectories are tracked by the position control law and the nonlinear attitude controller described in Sections 7.1-7.2. The agents must determine the optimal assignment in addition to their own optimal trajectories since the agents are not preassigned to specific targets. As discussed in Remark 4, the squared distance between the current initial points and the targets is used as the assignment cost. Two configurations were used to demonstrate SATO. In the first scenario, the agents start close together so that they can all communicate with each other resulting in a connected network and a conflict-free initial assignment. In the second scenario, the agents are spread apart so that none of them can communicate, which initially results in several assignment conflicts that must be resolved in future iterations of the auction algorithm. In both scenarios, the length of the reconfiguration is 20 s with a time step (Δt) of 1 s and horizon (T_H) of 5 time steps. Additionally, the collision radius (R_{col}) is 350 mm, and the max speed ($V_{\max}$) is 5 m/s. In the figures of this section, the initial, final, and current positions of the agents are denoted by a square, 'x', and circle, respectively. Additionally, the solid lines show the trajectories of the agents and the dotted lines are circles with radii equal to half of the communication radius. If the dotted lines of two agents intersect, those two agents can communicate. Each agent's trajectory is shown in a different color so that the corresponding markers can be easily identified.

7.3.1 Real-Time Flight Test with a Connected Communication Network In the first SATO hardware experiment, the quadrotors are placed in a square formation with the targets located 3 m away in a line. The communication radius (R_{comm}) for this demo is 1000 mm. The initial locations of the quadrotors are chosen so that every agent can communicate with every other agent creating a connected communication network. The setup for this experiment is shown in Fig. 8. This allows the initial auction algorithm to find a conflict-free assignment, which allows the quadrotors to move directly towards their assigned targets. Both the actual trajectories traversed by the quadrotors and the optimal reference trajectories generated by SATO at multiple time steps are shown in Fig. 9.

The quadrotors are initially located in a square formation with agent 1 (blue) in the bottom right, agent 2 (red) in the top right, agent 3 (green) in the bottom left, and agent 4 (magenta) in the top left. Figure 9 also shows the initial positions of the agents. As mentioned before, the agents can communicate with each other initially, as shown by the intersection of the dotted lines.

Additionally, Fig. 9 shows the reference trajectories generated by SATO throughout the reconfiguration. In this figure, the targets ('x's) are shown in black to emphasize that any quadrotor can choose any target. Due to the fact that the quadrotors are all on the same communication network at the initial time, the trajectories do not change much throughout the reconfiguration and the agents follow nearly straight lines to their assigned targets.

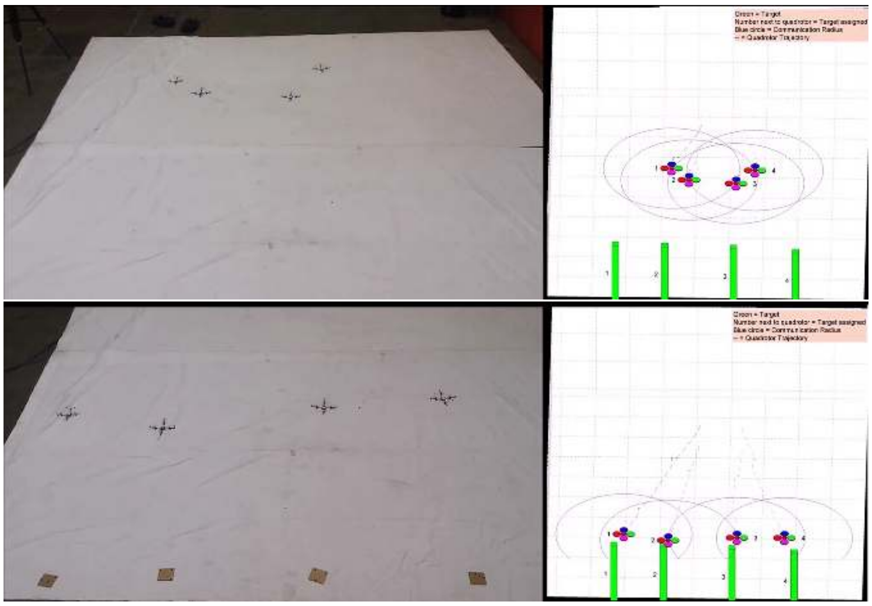
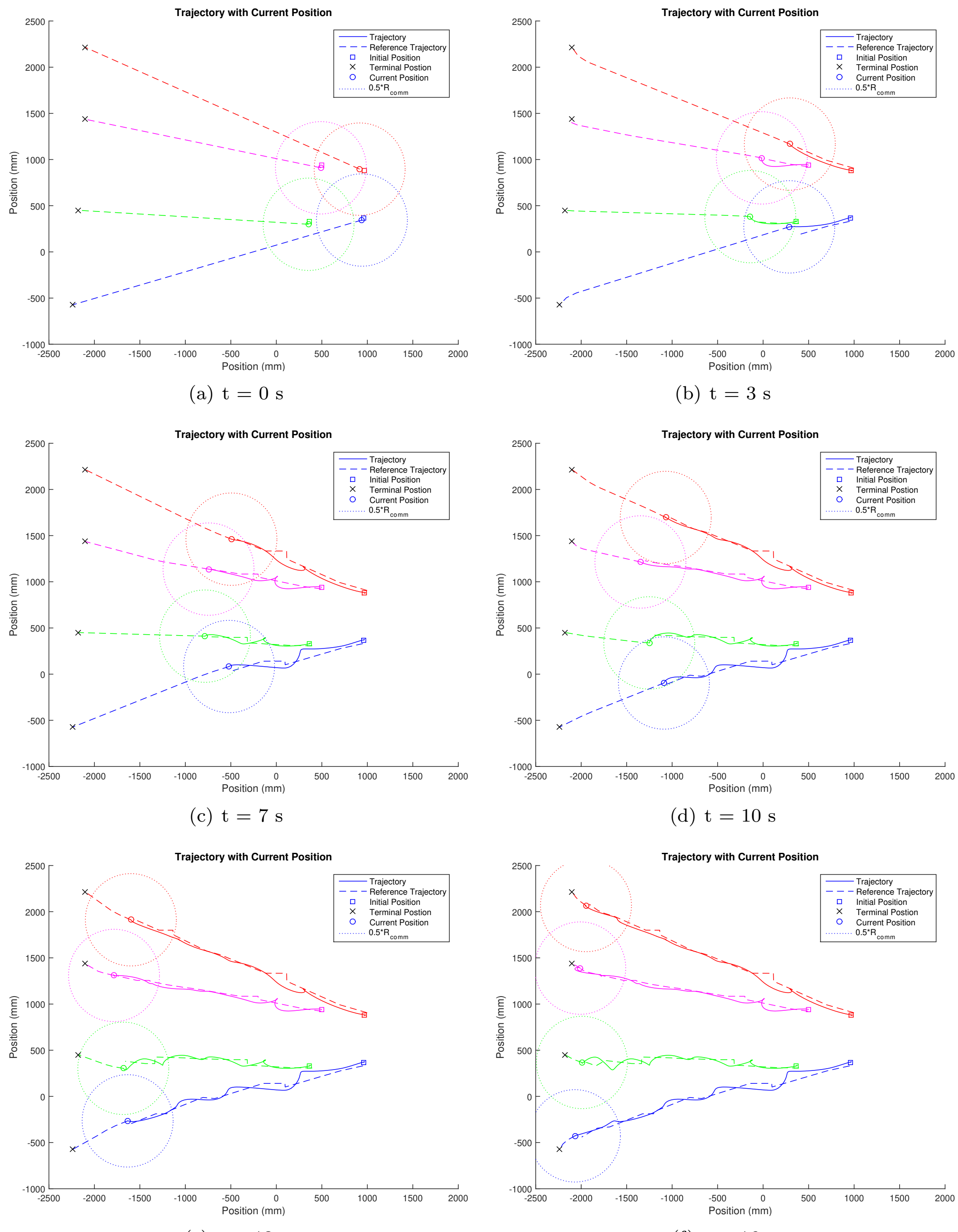

Green = Target
Number next to quadrotor = Target assigned
Blue circle = Communication Radius
-- = Quadrotor Trajectory

Figure 8. Experimental setup for four quadrotors in a connected communication network (see Extension 2).

7.3.2 Real-Time Flight Test with a Disconnected Communication Network In the second SATO hardware experiment, the communication radius (R_{comm}) is 750 mm and the quadrotors are placed in initial positions so that they cannot communicate with each other, which results in multiple agents being initially assigned to the same target. This layout, shown in Fig. 10, is intentionally chosen to create several assignment conflicts in order to show how SATO can handle disconnected communication networks. The results of this demonstration are shown in Fig. 11.

Figure 11 shows the reference trajectories generated by SATO and the actual trajectories traversed by all of the quadrotors throughout the reconfiguration. The quadrotors are initially located in a trapezoid formation with agent 1 (blue) in the top left, agent 2 (red) in the top right, agent 3 (green) in the bottom left, and agent 4 (magenta) in the bottom right. In this figure, the targets ('x's) are shown in black to emphasize that any agent can choose any target. In Fig. 11a-b, the four reference trajectories terminate at only two targets, which is the result of the two assignment conflicts between agents 1 (blue) and 2 (red) and between agents 3 (green) and 4 (magenta). When the third set of assignments and trajectories are generated as shown in Fig. 11c, agents 3 (green) and 4 (magenta) are able to communicate to resolve the assignment conflict and the third (green) agent's trajectory terminates at a different target. Similarly, Fig. 11d shows the trajectories generated by the fourth calculation, which results in agent 1 (blue) being assigned to a new target due to its ability to communicate with agent 2 (red). However, this reassignment creates a new conflict between agents 1 (blue) and 3 (green), which now share the same target. At the fifth computation, shown in Fig. 11e, agents 1 (blue) and 3 (green) still cannot communicate so they remain assigned to the same target. By the time of the sixth computation, shown in Fig. 11f, agents 1 (blue) and 3 (green) can communicate and agent 3 (green) is reassigned to the fourth target.



(a) t = 0 s (b) t = 3 s (c) t = 7 s (d) t = 10 s (e) t = 13 s (f) t = 16 s

Figure 9. Real-time, optimal trajectories generated by SATO (dashed lines) and actual trajectories (solid lines) throughout a reconfiguration of four quadrotors when the initial communication network is connected. Two quadrotors can communicate when their dotted circles intersect.

8 Conclusion

In this paper, we developed a new variable-swarm, distributed auction algorithm to solve the optimal assignment problem for a swarm of agents. The variable-swarm quality allowed the algorithm called VSDAA (Method 1) to

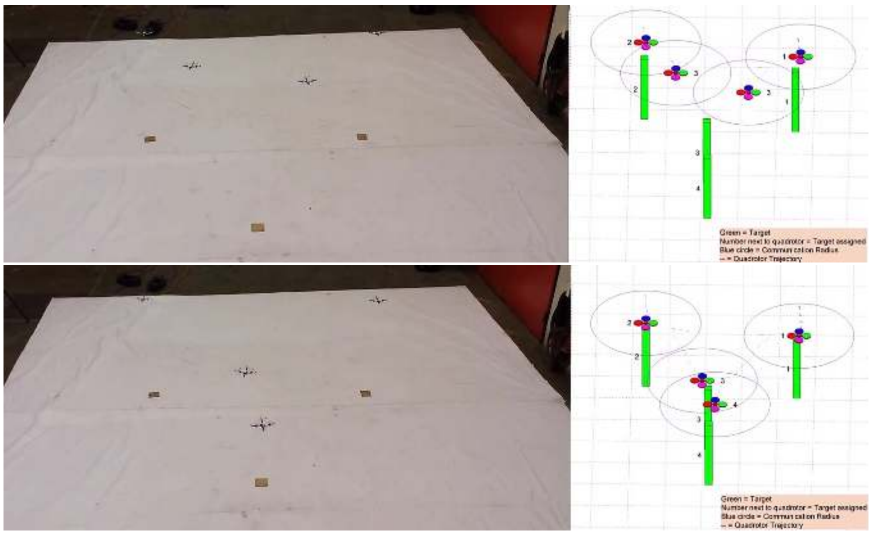

Green = Target
Number next to quadrotor = Target assigned
Blue circle = Communication Radius
-- = Quadrotor Trajectory

Figure 10. Experimental setup for four quadrotors in a disconnected communication network (see Extension 3).

adjust the number of targets in the assignment to match the number of agents in the swarm that are bidding on targets. This ultimately resulted in a distributed assignment with the same number of agents and targets. This quality was especially useful when the number of agents in the swarm changed or when there were initially more agents than targets. Additionally, VSDAA (Method 1) can be implemented on a swarm of robots or robotic vehicles with distributed communications and computations while still terminating in a finite number of iterations and achieving the optimal assignment.

We also introduced SATO (Method 3) as an algorithm that integrates the optimal assignment solver, VSDAA, and the trajectory optimizer, SCP. This integrated approach used MPC to update the optimal assignment and trajectory in real time so that changes in the state of each agent and changes in the communication network were included in the new solution. This allowed SATO to ultimately achieve the optimal assignment and trajectory even when the communication network was disconnected at the initial time or when errors prevented the agents from following their optimal trajectories.

Finally, we showed through simulation and experimentation that SATO can be implemented in a variety of situations. Results with double integrator dynamics showed that SATO can overcome an initially disconnected communication network and a loss of a significant number of agents, which caused the communication network to become disconnected. Additionally, a 3-D simulation result using relative orbital dynamics showed that SATO can be run on a swarm of spacecraft even when the dynamics are complicated. The results of these simulations and experiments showed that SATO can be used to solve for the optimal assignment and trajectory in a variety of undesirable conditions in a distributed, real-time manner.

Acknowledgements

This research was carried out in part at the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration. ©2016 California Institute of Technology. This work was supported by a NASA Office of the Chief Technologist Space Technology Research Fellowship and an Air Force Office of Scientific Research (AFOSR grant FA95501210193). Government sponsorship acknowledged. Additional thanks to Saptarshi Bandyopadhyay for stimulating discussions and constructive comments.

References

Alfriend KT, Vadali SR, Gurfil P, How JP and Breger L (2009) *Spacecraft Formation Flying: Dynamics, Control and Navigation*. Elsevier, Oxford, UK.

Alonso-Mora J, Breitenmoser A, Rufli M, Siegwart R and Beardsley P (2012) Image and animation display with multiple robots. *Int. Journal of Robotics Research* 31(6): 753–773.

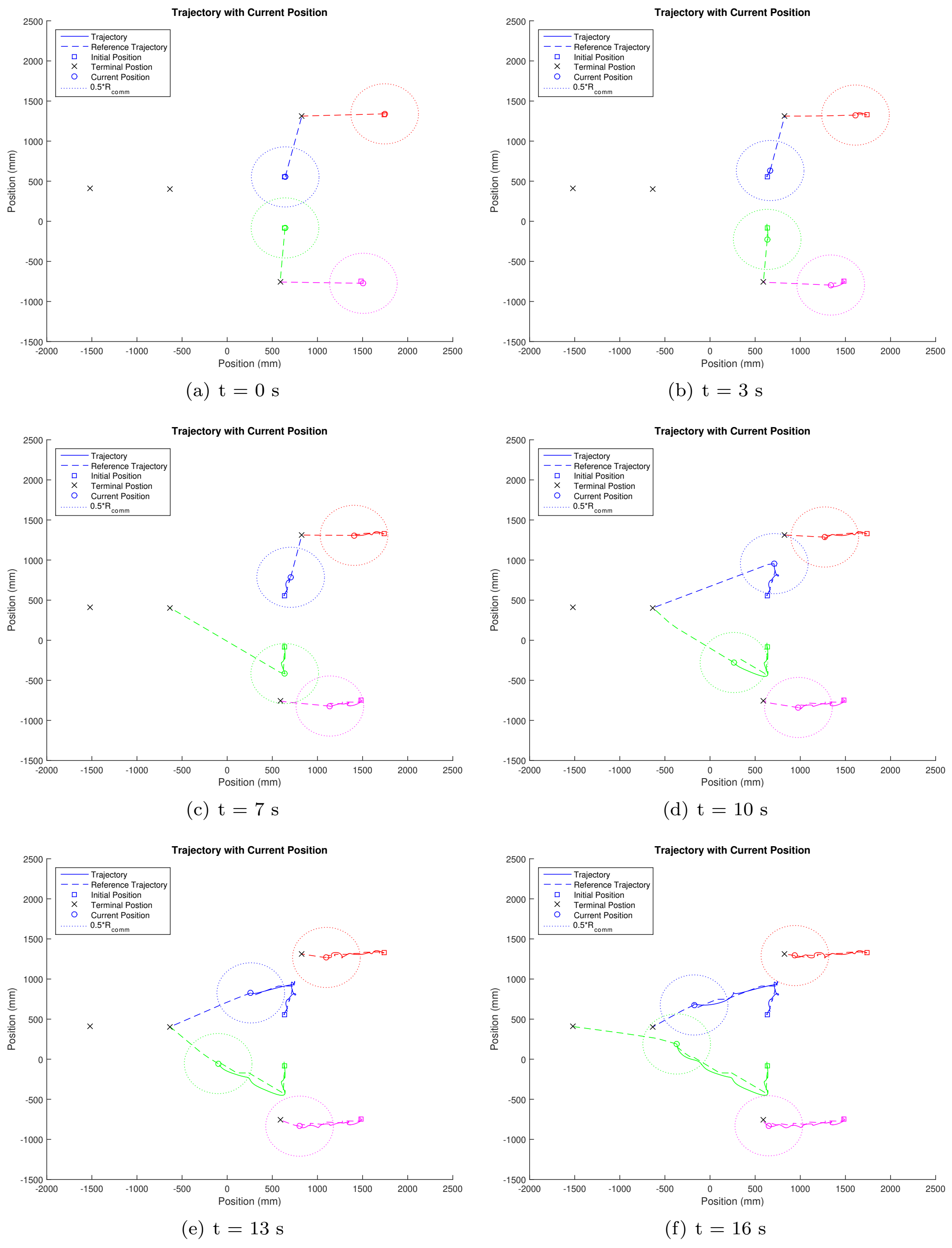


Figure 11. Real-time, optimal reference trajectories generated by SATO (dashed lines) and actual trajectories (solid lines) throughout a reconfiguration of four quadrotors when the initial communication network is disconnected. Two quadrotors can communicate when their dotted circles intersect.

Bandyopadhyay S, Chung SJ and Hadaegh FY (2016) Nonlinear attitude control of spacecraft with a large captured object. *Journal of Guidance, Control, and Dynamics* 39(4): 754–769.

Bemporad A and Morari M (1999) Robust model predictive control: A survey. In: *Robustness in Identification and Control*. pp. 207–226.

Bertsekas DP (1981) A new algorithm for the assignment problem. *Mathematical Programming* 21: 152–171.

Bertsekas DP (1992) Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications* 1: 7–66.

Bertsekas DP and Castanon DA (1991) Parallel synchronous and asynchronous implementations of the auction algorithm. *Parallel Computing* 17: 707–732.

Boyd S and Vandenberghe L (2004) *Convex Optimization*. Cambridge, U.K.: Cambridge Univ. Press.

Breger L and How JP (2007) Gauss's variational equation-based dynamics and control for formation flying spacecraft. *Journal of Guidance, Control, and Dynamics* 30(2): 437–448.

Campbell ME (2003) Planning algorithm for multiple satellite clusters. *Journal of Guidance, Control, and Dynamics* 26(5): 770–780.

Campbell ME (2005) Collision monitoring within satellite clusters. *IEEE Transactions on Control Systems Technology* 13(1): 42–55.

Cheah CC, Hou SP and Slotine JJE (2009) Region-based shape control for a swarm of robots. *Automatica* 45(10): 2406–2411.

Choi HL, Brunet L and How JP (2009) Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics* 25(4): 912–926.

Chung SJ, Bandyopadhyay S, Chang I and Hadaegh FY (2013) Phase synchronization control of complex networks of Lagrangian systems on adaptive digraphs. *Automatica* 49(5): 1148–1161.

Deters RW, Ananda GK and Selig MS (2014) Reynolds number effects on the performance of small-scale propellers. In: *AIAA Aviation*. Atlanta, GA: 32nd AIAA Applied Aerodynamics Conference, pp. 2014–2151.

Earl MG and D'Andrea R (2005) Iterative MILP methods for vehicle-control problems. *IEEE Transactions on Robotics* 21(6): 1158–1167.

Hadaegh FY, Chung SJ and Manaroha HM (2016) On development of 100-gram-class spacecraft for swarm applications. *IEEE Systems Journal* 10(2): 673–684.

Harary F (1994) *Graph Theory*. Reading, MA: Addison-Wesley.

Hsieh MA, Kumar V and Chaimowicz L (2008) Decentralized controllers for shape generation with robotic swarms. *Robotica* 26: 691–701.

Hung M (1983) A polynomial simplex method for the assignment problem. *Operations Research* 31: 595–600.

Jadbabaie A, Lin J and Morse AS (2003) Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control* 48(6): 2976–2984.

Jongen HT, Meer K and Triesch E (2004) *Optimization Theory*. Norwell, MA: Kluwer Academic.

Kingston P and Egerstedt M (2010) Index-free multiagent systems: An eulerian approach. In: *2nd IFAC Workshop on Distributed Estimation and Control in Networked Systems*. Annecy, France, pp. 215–220.

Kuhn HW (1955) The Hungarian method for the assignment problem. *Naval Research Logistics* 2(1-2): 83–97.

Mayne DQ, Rawlings JB, Rao CV and Scokaert POM (2000) Constrained model predictive control: Stability and optimality. *Automatica* 36: 789–814.

Milutinović D and Lima P (2006) Modeling and optimal centralized control of a large-size robotic population. *IEEE Trans. Robotics* 22(6): 1280–1285.

Morgan D, Chung SJ, Blackmore L, Acikmese B, Bayard D and Hadaegh FY (2012) Swarm-keeping strategies for spacecraft under J_2 and atmospheric drag perturbations. *Journal of Guidance, Control, and Dynamics* 35(5): 1492–1506.

Morgan D, Chung SJ and Hadaegh FY (2014) Model predictive control of swarms of spacecraft using sequential convex programming. *Journal of Guidance, Control, and Dynamics* 37(6): 1725–1740.

Morgan D, Chung SJ and Hadaegh FY (2015) Swarm assignment and trajectory optimization using variable-swarm, distributed auction assignment and model predictive control. In: *AIAA Guidance, Navigation, and Control Conference*. Kissimmee, FL. AIAA 2015-0599.

Paranjape AA, Meier KC, Shi X, Chung SJ and Hutchinson S (2015) Motion primitives and 3D path planning for fast flight through a forest. *The International Journal of Robotics Research* 34(3): 357–377.

Richards A, Kuwata Y and How J (2003) Experimental demonstration of real-time MILP control. In: *AIAA Guidance, Navigation, and Control Conference*. Austin, Texas. AIAA-2003-5802.

Ross IM and Fahroo F (2003) Legendre pseudospectral approximations of optimal control problems. *Lecture Notes in Control and Information Systems* 295: 327–342.

Ruszczynski A (2006) *Nonlinear Optimization*. Princeton, NJ: Princeton University Press.

Scharf DP, Hadaegh FY and Ploen SR (2003) A survey of spacecraft formation flying guidance and control (part i): Guidance. In: *American Control Conference*. Denver, CO, pp. 1733–1739.

Scharf DP, Hadaegh FY and Ploen SR (2004) A survey of spacecraft formation flying guidance and control (part ii): Control. In: *American Control Conference*. Boston, MA, pp. 2976–2985.

Schulman J, Ho J, Lee A, Awwal I, Bradlow H and Abbeel P (2013) Finding locally optimal, collision-free trajectories with sequential convex optimization. In: *Robotics: Science and Systems*. Berlin, Germany.

Subramanian GP (2015) *Nonlinear control strategies for quadrotors and CubeSats*. Master's Thesis, University of Illinois at Urbana-Champaign. Chapter 3: https://www.ideals.illinois.edu/handle/2142/88078.

Turpin M, Michael N and Kumar V (2014) CAPT: Concurrent assignment and planning of trajectories for multiple robots. *The International Journal of Robotics Research* 33(1): 98–112.

Vaddi SS, Alfriend KT, Vadali SR and Sengupta P (2005) Formation establishment and reconfiguration using impulsive control. *Journal of Guidance, Control, and Dynamics* 28: 262–268.

Yu J, Chung SJ and Voulgaris PG (2015) Target assignment in robotic networks: Distance optimality guarantees and hierarchical strategies. *IEEE Transactions on Automatic Control* 60(2): 327–341.

Zanon DJ and Campbell ME (2006) Optimal planner for spacecraft formations in elliptical orbits. *Journal of Guidance, Control, and Dynamics* 29(1): 161–171.

Zavlanos MM, Spesivtsev L and Pappas GJ (2008) A distributed auction algorithm for the assignment problem. In: *IEEE Conference on Decision and Control*. Cancun, Mexico, pp. 1212–1217.

Zhao S, Ramakrishnan S and Kumar M (2011) Density-based control of multiple robots. In: *American Control Conference*. San Francisco, USA, pp. 481–486.

Appendix A: Index to Multimedia Extensions

Archives of IJRR multimedia extensions published prior to 2014 can be found at http://www.ijrr.org. After 2014 all videos are available on the IJRR YouTube channel at http://www.youtube.com/user/ijrrmultimedia.

Extension	Type	Description
1	Video	Simulation demonstrating agents forming the shapes U, I, U and C. The simulation also shows the robustness properties of the algorithm.
2	Video	Experimental validation of the algorithm using 4 quadrotors. In this experiment, all the quadrotors initially form a connected graph.
3	Video	Experimental validation of the algorithm using 4 quadrotors. In this experiment, all the quadrotors do not form a connected graph in the beginning.