Decoupled Multiagent Path Planning via Incremental Sequential Convex Programming

Yufan Chen, Mark Cutler, and Jonathan P. How

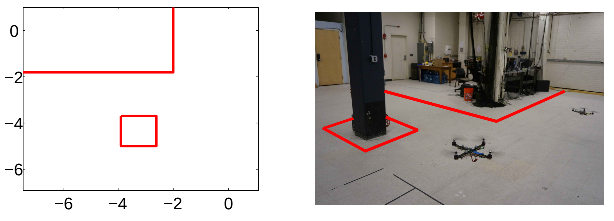
Abstract— **This paper presents a multiagent path planning algorithm based on sequential convex programming (SCP) that finds locally optimal trajectories. Previous work using SCP efficiently computes motion plans in convex spaces with no static obstacles. In many scenarios where the spaces are non-convex, previous SCP-based algorithms failed to find feasible solutions because the convex approximation of collision constraints leads to forming a sequence of infeasible optimization problems. This paper addresses this problem by tightening collision constraints incrementally, thus forming a sequence of more relaxed, feasible intermediate optimization problems. We show that the proposed algorithm increases the probability of finding feasible trajectories by 33% for teams of more than three vehicles in non-convex environments. Further, we show that decoupling the multiagent optimization problem to a number of single-agent optimization problems leads to significant improvement in computational tractability. We develop a decoupled implementation of the proposed algorithm, abbreviated dec-iSCP. We show that dec-iSCP runs 14% faster and finds feasible trajectories with higher probability than a decoupled implementation of previous SCP-based algorithms. The proposed algorithm is real-time implementable and is validated through hardware experiments on a team of quadrotors.**

I. INTRODUCTION

Multiagent robotic systems are attracting significant research interest due to their inherent flexibility, robustness to single-point failures, and potential to provide more diverse functionality than single-agent systems. Many applications of multiagent systems require coordination of mobile agents navigating in complex environments. For instance, researchers have investigated using fleets of unmanned aerial and ground vehicles for forest fire monitoring [1], persistent surveillance [2], warehouse inventory management [3], and expressive artistic performance [4]. These scenarios require computing collision free trajectories connecting initial and final positions for every agent in the fleet.

For multiagent systems, it is challenging to efficiently compute optimal collision free trajectories, where trajectory quality is determined by total path length or total required thrust. For instance, researchers have framed multiagent path planning problems as Mixed Integer Linear Programs (MILPs) [5] and Mixed Integer Quadratic Programs (MIQPs) [6]. These centralized methods often have good theoretical properties such as optimality guarantees, but the computational complexity of MILPs and MIQPs limit their applications to small teams with few obstacles [6]. Also, researchers have investigated extending well-established single agent path planning algorithms to multiagent systems.

Laboratory of Information and Decision Systems, Massachusetts Institute of Technology, 77 Massachusetts Ave., Cambridge, MA, USA {chenyuf2, cutlerm, jhow}@mit.edu



(a) Layout of ACL lab space (b) Quadrotors navigating in ACL

Fig. 1. A quadrotor fleet navigating in MIT's Aerospace Controls Laboratory (ACL). Note how the pole and the corner (marked in red) in the room make the flight space a non-convex region. The proposed algorithm quickly finds feasible multiagent trajectories in this non-convex flight space.

For instance, multiagent motion planning algorithms MA-RRT* [7] and DMA-RRT [8] are based on a combination of sampling-based RRT [9], RRT* [10], and grid-based forward search A* [11], [12]. Unfortunately, these algorithms are also computational intensive and require additional simplifying assumptions, such as sparsity in the environment. Although recent advances in grid-based methods [13], [14] have shown improved computational tractability, applications of these methods are typically restricted to discrete domains since they do not account for vehicle dynamics (e.g., control effort). Alternatively, algorithms such as potential fields [15], [16] and ClearPath [17] are computationally efficient but do not have optimality guarantees as they do not explicitly compute feasible trajectories to the goal.

Sequential convex programming (SCP) [18] has been shown to achieve a good balance between solution quality and computational tractability [19]. SCP-based methods have been applied to multiagent motion planning in convex domains, such as the coordination of a quadrotor team [20] and formation control of spacecraft [21]. However, in non-convex environments, previous SCP algorithms often fail to find feasible solutions because the convex approximation of collision constraints used in those approaches leads to forming a sequence of over-constrained optimization problems, for which it is very difficult to find a feasible solution.

This work addresses this problem by tightening constraints incrementally, thus forming a sequence of feasible intermediate optimization problems. The proposed algorithm is named incremental sequential convex programming (iSCP) to highlight the way in which constraints are added when forming intermediate optimization problems. Further, we develop a decoupled implementation of iSCP, abbreviated dec-iSCP, that achieves significant improvement in computational tractability. This decoupled algorithm converges faster and finds feasible trajectories with higher probability than a decoupled implementation of previous SCP-based algorithms. The main contributions of this work are (1) demonstrating that single-agent iSCP can navigate around static obstacles in non-convex domains, (2) presenting dec-iSCP for sequential path planning for multiagent systems, (3) presenting a final time scaling method that avoids having to solve the optimization problem multiple times with different discretization parameter settings, and (4) showing dec-iSCP is real-time implementable with a hardware demonstration on a team of quadrotors (Fig. 1(b)).

II. Problem Formulation

This section presents the multiagent path planning optimization problem. The objective is to generate collision free trajectories for a team of robotic agents, subject to inter-vehicle and static obstacle avoidance constraints. We assume that there is a low level controller for tracking trajectories specified by the planner. The problem formulation is similar to [20].

a) Objective Function: For a system of N vehicles in D dimensions with K discretization times steps, the objective is find the minimizer

$$\underset{\mathbf{a}_i[k]}{\operatorname{argmin}} \sum_{i=1}^{N}\sum_{j=1}^{N}\sum_{k=1}^{K} w_{ij}\mathbf{a}_i^T[k]\mathbf{a}_j[k], \tag{1}$$

where w_{ij} are the scalar weights, $\mathbf{a}_i[k] \in \mathbb{R}^D$ is the acceleration vector of agent $i \in \{1,\dots,N\}$ at discretized time step $k \in \{1,\dots,K\}$. Since position, velocity, and jerk can be expressed as linear functions of acceleration (Eqn. 2), we can choose w_{ij}'s to achieve different trade-offs between minimizing path length (i.e., $||\mathbf{p}_i[k+1]-\mathbf{p}_i[k]||^2$) or minimizing control effort (i.e., $\sum_{i,k}||\mathbf{a}_i[k]||^2$). In this paper, to minimize total thrust squared, we set $w_{ij}=1$ when $i=j$ and $w_{ij}=0$ otherwise, which tends to induce smooth trajectories with low curvatures.

b) Kinematic Constraints: Given acceleration decision variables $\mathbf{a}_i[k]$ and discretization step size h, position $\mathbf{p}_i[k]$, velocity $\mathbf{v}_i[k]$, and jerk $\mathbf{j}_i[k]$, can be derived via kinematic relations,

$$\begin{aligned}
\mathbf{p}_i[k+1] &= \mathbf{p}_i[k] + h\mathbf{v}_i[k] + \frac{h^2}{2}\mathbf{a}_\mathbf{i}[k]\\
\mathbf{v}_i[k+1] &= \mathbf{v}_i[k] + h\mathbf{a}_i[k]\\
\mathbf{j}_i[k] &= \frac{\mathbf{a}_i[k+1]-\mathbf{a}_i[k]}{h} \qquad \forall i,k\,.
\end{aligned} \tag{2}$$

The tuples $(\mathbf{p}_i,\mathbf{v}_i,\mathbf{a}_i)[k]$ specify a trajectory for each agent. Further, a feasible trajectory must reach the desired final waypoint,

$$\mathbf{p}_i[K]=\mathbf{p}_{fi}, \quad \mathbf{v}_i[K]=\mathbf{v}_{fi}, \quad \mathbf{a}_i[K]=\mathbf{a}_{fi} \quad \forall i\,. \tag{3}$$

Given physical limits (e.g., actuator limits), each vehicle must respect position and acceleration constraints,

$$\mathbf{p}_i[k]\in[\mathbf{p}_{\min},\mathbf{p}_{\max}], \quad \mathbf{a}_i[k]\in[\mathbf{a}_{\min},\mathbf{a}_{\max}], \quad \forall i,k\,. \tag{4}$$

c) Collision Avoidance Constraints: To prevent inter-vehicle collision, every pair of vehicles must be separated by at least distance R at all time steps,

$$||\mathbf{p}_i[k]-\mathbf{p}_j[k]||_2 \ge R \qquad \forall i,j \quad i\ne j, \qquad \forall k\,. \tag{5}$$

Further checking may be needed to ensure that there would be no inter-vehicle collision in between any two time steps [20].

d) Static Obstacle Constraints: Eqn. 4 assumes a convex rectangular environment. Static obstacles can be modeled using various functional forms to impose inequality constraints on position vectors, $\mathbf{p}_i[k]$'s. For instance, adaption of Eqn. 5 can be used to model circular static obstacles. Motivated by the need to navigate in our laboratory, which has a L-shaped flight space with a square pole in the center (Fig. 1(b)), we use an exponential loss function to model polytopic obstacles. For example, keeping trajectories out of a rectangular corner region $O_c = \{(x,y)|x<x_{cor}, y<y_{cor}\}$ requires

$$\begin{aligned}
&\exp\left(c\left(p_i^x[k]-x_{cor}\right)\right)+\\
&\qquad \exp\left(c\left(p_i^y[k]-y_{cor}\right)\right)\ge 2 \quad \forall i,k,
\end{aligned} \tag{6}$$

where $p_i^x[k]$, $p_i^y[k]$ are the x, y coordinates of $p_i[k]$. Increasing values of constant c lead to better approximations of the boundary of O_c.

e) Final Time Scaling: The optimization problem in Eqn. 1 requires specifying time step size h and end time $T=hK$ beforehand. However, these values are often difficult to estimate, particularly for multiagent scenarios. Small values of T may lead to an infeasible optimization problem, whereas large values of T may lead to inefficiencies as vehicles would travel slowly. Previous works [20], [21] suggest initially solving the optimization problem with small values of T, gradually increasing T until a feasible solution is found. This procedure leads to inefficiencies since it requires solving the optimization problem multiple times. We propose solving the optimization problem with relaxed T (i.e. 5 times the expected final time), then computing a scaling factor r,

$$r=\min\left\{\min_{i,k}\frac{||\mathbf{v}_{\max}||}{||\mathbf{v}_i[k]||},\ \min_{i,k}\frac{||\mathbf{a}_{\max}||}{||\mathbf{a}_i[k]||},\ \min_{i,k}\frac{||\mathbf{j}_{\max}||}{||\mathbf{j}_i[k]||}\right\}, \tag{7}$$

where $\mathbf{v}_{\max}$, $\mathbf{a}_{\max}$, and $\mathbf{j}_{\max}$ are maximum allowed velocity, acceleration and jerk, respectively. The solution to Eqn. 1 is then scaled by the factor r,

$$h_{scaled}=h/\sqrt{r}, \qquad \mathbf{a}_i[k]_{scaled}=\mathbf{a}_i[k]r, \qquad \forall i,k\,. \tag{8}$$

Substituting Eqn. 8 into Eqn. 2, we observe that although acceleration, velocity, and jerk commands are changed, the position commands $\mathbf{p}_i[k]$ remain unchanged for every agent at every time step. Thus, this scaling procedure only changes the rate at which vehicles travel along the trajectories, not the shape of the trajectories.

III. Incremental Sequential Convex Programming

This section presents various adaptations of SCP for solving the optimization problem described in Sec. II. SCP methods form successive convex approximations of an optimization problem about a current iterate. In this paper, each SCP-based algorithm is initialized with a straight line connecting initial and final positions for each vehicle. The stopping criterion is to require (1) all constraints to be satisfied, and (2) the maximum deviation in any vehicle's trajectory between successive iterates to be less than a tolerance value, ϵ, such that $\max_{i,k} ||\mathbf{p}_i^q[k] - \mathbf{p}_i^{q-1}[k]||_2 < \epsilon$. The superscript q denotes the iteration number. More complex stopping criteria can be found in [20], [21]. [1]

A. Categorizing SCP Methods

This work introduces incremental SCP for path planning, which can be adapted for both coupled and decoupled formulations. The distinctions between coupled and decoupled formulations, and between the two ways of adding constraints, are explained in following subsections. This section establishes that (1) decoupled formulations have significantly better computational tractability than the coupled formulations, (2) decoupled formulations of existing SCP approaches often fail to find feasible solutions in convex and non-convex environments, and (3) decoupled formulations of iSCP address the underlying problem leading to (2). To clarify further discussion, we classify various approaches of SCP in Table I.

TABLE I
CLASSIFICATION OF SCP-BASED APPROACHES.

	Add all constr. at once	Add constr. incrementally
Form coupled QP	cup-SCP	cup-iSCP
Form decoupled QP	dec-SCP	dec-iSCP

B. Coupled SCP

Coupled SCP (cup-SCP) [20] forms a convex approximation of the optimization problem described in Sec. II by linearizing Eqn. 5 (similarly for Eqn. 7) about the previous iterate q, which can be described as follows

$$||\mathbf{p}_i^q[k] - \mathbf{p}_j^q[k]|| + \frac{\left(\mathbf{p}_i^q[k] - \mathbf{p}_j^q[k]\right)^T}{||\mathbf{p}_i^q[k] - \mathbf{p}_j^q[k]||}(\mathbf{p}_i[k] - \mathbf{p}_j[k]) \geq R. \tag{9}$$

Since the objective function is quadratic and all constraints are linear, the convex approximation of the original problem is a quadratic program (QP). We call this approach "coupled" SCP because the algorithm forms a QP whose decision variables are acceleration vectors $\mathbf{a}_i[k]$'s of all vehicles.

[1]In general, SCP methods use a β parameter to control a trust region around the current iterate [18]. This scheme reduces the circumstances in which the algorithm oscillates between local minima. We found empirically that this is not necessary for solving the optimization problem described in Sec. II.

C. Decoupled SCP

Decoupled SCP (dec-SCP)[2], shown in Alg. 1, presents improved computational tractability and enables decentralized, asynchronous implementation (explained in Sec. IV). In comparison to cup-SCP, dec-SCP sequentially forms N smaller optimization problems. In line 1, dec-SCP determines the number of discretization time steps and initializes an empty obstacle list. Then, the algorithm computes trajectories for each vehicle sequentially. In particular, a single-agent optimization problem is formed in line 3, and subsequently solved using an SCP-based method outlined in Alg. 2. Each solution to the single-agent optimization problem defines a trajectory, which is appended to the obstacle list in line 4. Hence, inter-vehicle collision avoidance constraints are reinforced by casting trajectories of earlier vehicles as constraints for later vehicles. Lastly, in line 5, the algorithm invokes the final time scaling procedure as defined in Eqn. 7 and 8.

Single-agent SCP, shown in Alg. 2, is an adaption of generic SCP to solve single-agent path planning problems. In particular, the algorithm linearizes collision constraints about position $\mathbf{p}$ at the current iteration (line 3), and then forms a convex approximation with a QP (line 4). The minimizer of this QP, which is acceleration vector $\mathbf{a}$, determines position $\mathbf{p}$ at the next iteration (line 6) using kinematic relations (Eqn. 2). This process repeats until convergence or until exceeding the maximum allowed number of iterations.

Decoupled iSCP shows better computational tractability than coupled SPC. In particular, at each iteration, cup-SCP forms one QP with KND decision variables and $O(KN^2)$ inequality constraints. In comparison, dec-iSCP forms N smaller QPs, each with KD decision variables and $O(KN)$ inequality constraints. Recall that K is the number of discretization steps, N is the number of agents, and D is the number of dimensions. Since the computation time for solving QPs increases quadratically with the number of inequality constraints [20], the runtime of cup-SCP and dec-iSCP are $O(K^2N^4)$ and $O(K^2N^3)$, respectively. Since cup-SCP adds all collision constraints at every iteration, it typically runs at its worst case run time. In contrast, dec-iSCP only adds collision constraints as necessary (line 9 of Alg. 3), which is often a small subset of all collision constraints. Thus, dec-iSCP typically runs faster than its worst case run time. Fig. 4 (Sec. IV) shows an empirical evaluation of average run time, which suggests that dec-iSCP converges faster than cup-SCP.

D. Incremental SCP

The main contribution of this work is introducing incremental SCP (iSCP). Before presenting the mechanics of iSCP, we first identify key differences between cup-SCP and

[2]A similar decoupling procedure was developed in [21], which focused on application of SCP to formation reconfiguration of spacecraft swarms. The decoupling procedure in [21] is well-suited for application to spacecraft swarm reconfiguration, where the planning space is sparsely populated. In contrast, this work considers path planning in more general settings. For instance, in congested environments, the decoupling procedure in [21] would lead to convergence problems.

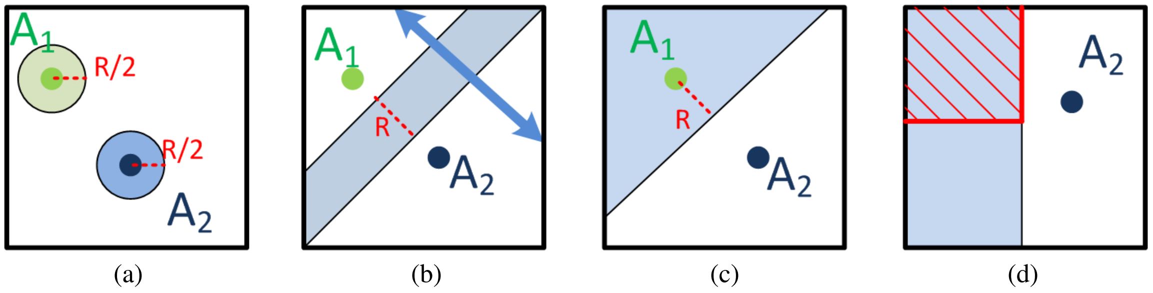


Fig. 2. Collision avoidance constraints for a pair of vehicles. A_1 and A_2 represent two different vehicles. Fig. (a) illustrates Eqn. 5, where the two circular disks of radius $R/2$, centered around each vehicle, are not allowed to overlap. Fig. (b) illustrates linearization of Eqn. 5 in cup-SCP, where the two vehicles cannot reside inside the blue band simultaneously. Note the blue band can slide along the blue arrow. Fig. (c) illustrates linearization of Eqn. 5 in dec-SCP, where A_2 is not allowed inside the blue region. Fig. (d) illustrates linearization of the corner constraint (hatched pattern), where A_2 is not allowed inside the blue region.

```
Algorithm 1: dec-SCP
Input: initial waypoint $p_{0i}$, final waypoint $p_{fi}$ for
    agents $i \in \{1, \ldots, N\}$, time step size $h$, final
    time $T$, static obstacles $O$, physical bounds $B$
Output: trajectory specified by series of waypoints
    $(p, v, a)$ for each agent
1 $K \leftarrow T/h + 1$,    $obs\_list \leftarrow \emptyset$
2 foreach Agent $i \in 1, \ldots, N$ do
3     $(p_i, v_i, a_i) \leftarrow$ singleSCP( $p_{0i}, p_{fi}, h, K, B, obs\_list$)
4     $obs\_list \leftarrow obs\_list \cup \{(p_i, v_i, a_i)\}$
5 $(p, v, a) \leftarrow$ timeScale $((p_1, v_1, a_1), \ldots, (p_N, v_N, a_N))$
6 return $(p, v, a)$
```

```
Algorithm 2: single-SCP
Input: initial waypoint $p_0$, final waypoint $p_f$, time step
    size $h$, number of steps $K$, static obstacles $O$,
    physical bounds $B$, obstacle list $l$
Output: trajectory specified by series of waypoints
    $(p, v, a)$
1 $(p, v, a) \leftarrow$ initializeStraightLine($p_0, p_f$)
2 while not converged do
3     $R \leftarrow$ linearizeAllCollConstr($p, O, B, l$)
4     $QP \leftarrow$ formQP($R, p_0, p_f, h, K$)
5     $a \leftarrow$ solve(QP)
6     $v, p \leftarrow$ propagateStates($p_0, a$)
7 return $(p, v, a)$
```

dec-SCP, which provide insights into the circumstances under which previous SCP-based algorithms might fail to find a feasible trajectory, and motivate the development of iSCP.

Note that the linearization of Eqn. 5 in dec-SCP (line 3 of Alg. 2) is not the same as Eqn. 9 in cup-SCP. In cup-SCP, solving the coupled QP updates the value of both $\mathbf{a}_i[k]$ and $\mathbf{a}_j[k]$, which subsequently updates the value of both $\mathbf{p}_i[k]$ and $\mathbf{p}_j[k]$. Thus, Eqn. 9 requires $\mathbf{p}_i[k]$ and $\mathbf{p}_j[k]$ to satisfy a *relative* position constraint, as illustrated in Fig. 2(b). In contrast, dec-SCP casts the trajectory of an earlier agent j as a dynamic obstacle for a later agent i. When forming a QP for agent i, $\mathbf{p}_i[k]$ must satisfy an *absolute* position constraint about the fixed position $\mathbf{p}_j[k]$, as illustrated in Fig. 2(c), where $\mathbf{p}_i[k]$ cannot reside in a significant portion of the domain determined by $\mathbf{p}_j[k]$. Solving each agent sequentially (as in dec-SCP) is therefore similar to having static obstacles (i.e. corner constraints as in Fig. 2(d)), where a significant portion of the domain is marked infeasible by an absolute position constraint. This important distinction shows that the dec-SCP approximation is more constrained than that of cup-SCP, as shown in Fig. 2(b) and Fig. 2(c). The following discussion shows that this distinction leads to certain scenarios in which dec-iSCP could find a feasible trajectory, while dec-SCP fails to find a feasible trajectory.

In previous SCP-based approaches [20], [21], linearization of collision avoidance constraints might lead to forming infeasible intermediate QPs, which permit no feasible solution. For an infeasible problem, QP solvers return a solution that violates at least one constraint. We call this solution an infeasible solution. Since dec-SCP forms more constrained intermediate QPs than cup-SCP, dec-SCP forms infeasible intermediate QPs more frequently than cup-SCP. SCP-based approaches can sometimes recover from an intermediate infeasible QP if successive convex approximations (linearizations) about the infeasible solution form a new feasible QP, from which a feasible solution can be found. Otherwise, SCP-based approaches can be trapped in an infeasible configuration (when the solution to an infeasible QP forms the same infeasible QP) and fail to find a feasible solution. We find that SCP-based approaches are more likely to recover if all constraints are *relative* position constraints, such as applying cup-SCP in a convex domain; SCP-based approaches are also more likely to be trapped in an infeasible configuration in the presence of *absolute* position constraints, such as applying dec-SCP or applying cup-SCP in non-convex domains (Fig. 2(b) and Fig. 2(c)). In Sec. IV, we provide examples where existing SCP approaches fail to find feasible trajectories for this reason.

The key idea of iSCP is to make the intermediate QPs less constrained by tightening collision constraints incrementally, thereby trying to ensure that all intermediate QPs are fea-

```
Algorithm 3: single-iSCP
  Input: initial waypoint $p_0$, final waypoint $p_f$, time step
         size $h$, number of steps $K$, static obstacles $O$,
         physical bounds $B$, obstacle list $l$
  Output: trajectory specified by series of waypoints
          $(p, v, a)$
1  $(p, v, a) \leftarrow$ initializeStraightLine($p_0, p_f$)
2  $addedConstr \leftarrow \emptyset$
3  while not converged do
4      $newConstrCount \leftarrow 0, \quad R \leftarrow \emptyset$
5      foreach time step $k \in 1, \ldots, K$ do
6          if $k \in addedConstr$ then
7              $r \leftarrow$ linearizeCollConstr($p[k], O, B, l$)
8              $R \leftarrow R \cup r$
9          else if $newConstrCount$ = 0 and Collision
           Constraint at time $k$ violated then
10             $r \leftarrow$ linearizeCollConstr($p[k-1], O, B, l$)
11             $R \leftarrow R \cup r$
12             $addedConstr \leftarrow addedConstr \cup \{k\}$
13             $newConstrCount$ ++
14     QP $\leftarrow$ formQP($R, p_0, p_f, h, K$)
15     $a \leftarrow$ solve(QP)
16     $v, p \leftarrow$ propagateStates($p_0, a$)
17 return $(p, v, a)$
```

sible. Alg. 3 outlines iSCP adapted to a single-agent path planning problem, which differs from SCP (Alg. 3) in the way in which collision constraints are added when forming intermediate QPs (lines 4-14). In line 2, the algorithm initializes a list to a empty set, which maintains added collision constraints at time steps $k \in \{1, \ldots, K\}$. When forming each intermediate QP, only a single new collision constraint (that is not already in *addedConstr*) will be added (line 4, 13). In particular, if the collision constraint at time step k is violated and no collision constraints at other time steps have been added (line 9), collision constraint at time step k will be included in the constraint matrix R when forming the next QP (line 10-11). Also, if the collision constraint at time step k was already added in the previous QP, this collision constraint will again be included in constraint matrix R when forming the next QP (line 6-8). Dec-iSCP is formed by replacing line 3 in dec-SCP (Alg. 1) with a procedure call of single-iSCP (Alg. 3). If *addedConstr* is initialized to $\{1, \ldots, K\}$ in line 2 of Alg. 3, which is to add all collision constraints at the beginning, Alg. 2 and 3 would be equivalent.

iSCP forms more relaxed intermediate QPs than SCP because iSCP (1) only adds a linearized collision constraint if the corresponding nonlinear collision constraint was violated (line 9), (2) adds new collision constraints one at a time (line 4), and (3) linearizes about position $\mathbf{p}[k-1]$ when the collision constraint at time step k is first added (line 10). Yet, iSCP will add collision constraints at all time steps in at most K iterations. Thus, after K iterations, iSCP will be identical

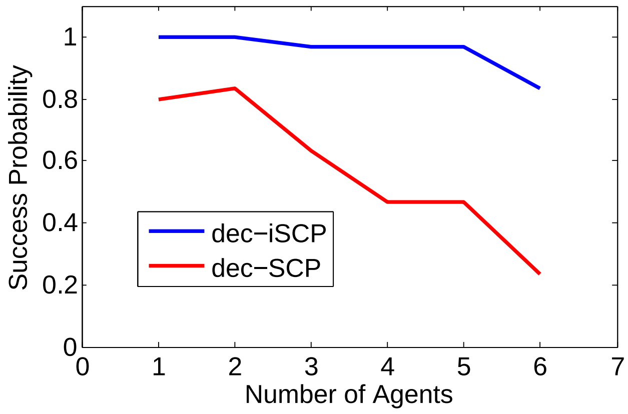


Fig. 3. The probability of converging to a feasible solution in a non-convex domain. Fig. 5 illustrates a scenario where SCP fails to converge and iSCP converges when planning in a non-convex domain. This figure shows the probability that similar scenarios occur. The algorithms dec-SCP and dec-iSCP were run on 30 randomly generated test cases with initial and final positions uniformly sampled from the convex domain.

to SCP. As shown in Sec. IV, iSCP typically converges in fewer iterations than the number of discretization steps K.

To provide some intuition for this approach, we note that collision constraints at discretized time steps respect a temporal ordering, which is an important property of the path planning problem (not true for generic optimization problems). Adding constraints incrementally corresponds to gradually growing a path from the initial position towards the final position. As collision constraints of later time steps are introduced, path segments at previous time steps are modified accordingly.

IV. Empirical Analysis

This section presents simulation and hardware results of applying SCP approaches to solve the path planning problem described in Sec. II. In particular, we compare and contrast the performance of various SCP approaches in different path planning scenarios. Algorithms were implemented in Matlab and ran on a computer with Intel i7 Q740 Processor and 6GB of RAM. Inter-vehicle collision avoidance constraints (Eqn. 5) required at least 0.8m of separation between every pair of vehicles at every time step.

A. Navigating Around Static Obstacles

This work is motivated by the need to develop a multiagent path planner for ACL's non-convex flight volume (Fig. 1(a)), which has a corner constraint and a pole constraint. Fig. 5 illustrates a scenario in which a single quadrotor attempts to navigate around the upper left corner. In both SCP and iSCP[3], the trajectory is initialized to a straight line connecting the initial and final positions, shown as a dotted line. Linearization of Eqn. 7 leads to the formulation of an infeasible QP, so SCP finds no feasible trajectory, as shown in Fig. 5(a). As explained in Sec. III, SCP in this case forms a sequence of infeasible QPs and gets trapped in an infeasible configuration

[3]The coupled vs. decoupled distinction is irrelevant for single agent problems.

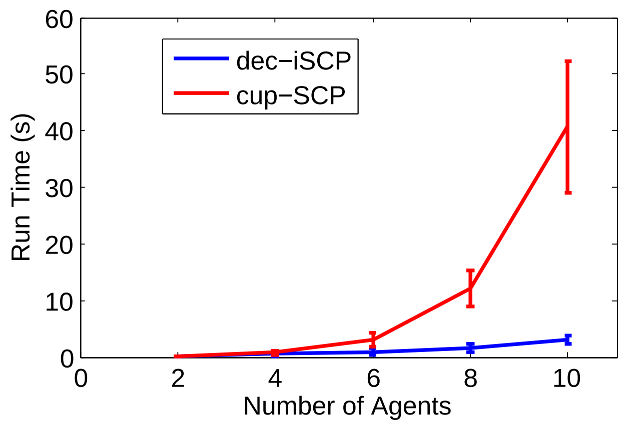


Fig. 4. Average run time of dec-iSCP and cup-SCP in a 8m × 8m convex domain. Each algorithm was run on 30 randomly generated test cases for different numbers of agents. Error bars denote one standard deviation.

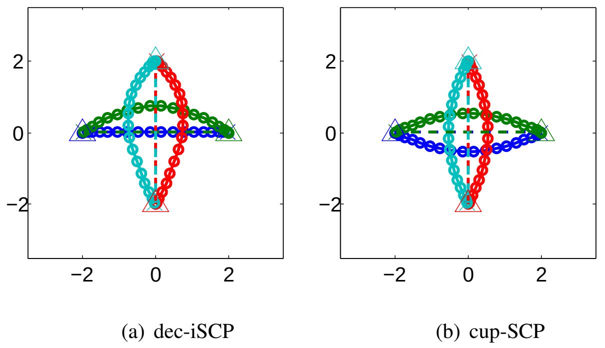


Fig. 6. Comparing the solution quality of dec-iSCP and cup-SCP in a convex domain. Two pairs of vehicles attempt to swap positions, with initial positions marked with triangles. Fig. 6(a) and Fig. 6(b) illustrate trajectories found by dec-iSCP and cup-SCP, respectively. The trajectory found by dec-iSCP has 15.5% higher objective function value than that of cup-SCP.

(Fig 5(a)). In contrast, iSCP adds collision constraints one at a time, thereby forming a sequence of feasible QPs, where each intermediate trajectory shows a small improvement over the past iterate, shown in Fig. 5(b). In this case, SCP fails to find a feasible trajectory whereas iSCP has converged to a feasible solution in 9 iterations. We evaluated the frequency that similar scenarios occur by generating a number of random test cases. Fig. 3 shows that dec-iSCP finds feasible solutions with 33% higher probability than dec-SCP for teams of more than three agents.

B. *cup-SCP and dec-iSCP*

The simulation result illustrated in Fig. 4 empirically verifies that dec-iSCP presents significantly lower computational complexity than cup-SCP as discussed in Sec. III. More importantly, decoupled iSCP allows for potentially decentralized and asynchronous implementation, whereas cup-SCP explicitly requires all vehicles to be synchronized by forming a coupled optimization problem, in which the entire team must wait for the most constrained vehicle (i.e., the vehicle traveling the longest trajectory). This can be inefficient in situations where interactions between vehicles are sparse. In contrast, dec-iSCP solves for each agent sequentially. This procedure allows for a decentralized implementation such that each vehicle can plan its own trajectory using iSCP by treating other vehicles as dynamic obstacles.

We acknowledge that dec-iSCP is a sequential greedy solution approach to the optimization problem described in Sec. II. Thus, dec-iSCP typically finds inferior solutions when compared to cup-SCP in terms of objective function value. Simulations of 30 test cases with random initial and final positions show that trajectories found by dec-iSCP are on average 7% higher in objective function value than that of cup-SCP. An example is illustrated in Fig. 6.

C. *dec-iSCP in Convex Domains*

Fig. 7 illustrates an application of dec-SCP and dec-iSCP to solve a multiagent path planning problem in a convex domain, where two pairs of vehicles attempt to swap positions. In both algorithms, the blue, green and red agents are solved before the teal agent. The teal agent views the other three agents as dynamic obstacles with known trajectories. This causes dec-SCP to form an infeasible QP and get trapped in an infeasible configuration, as shown in Fig. 5(a). In contrast, dec-iSCP finds a feasible solution in 7 iterations. As explained in Sec. III, this example shows that solving for each agent sequentially is similar to having static obstacles (Fig. 5). We found the frequency of similar scenarios occurring and the average computation time by generating a number test cases with random initial and final positions. The performance of dec-SCP and dec-iSCP is illustrated in Fig. 8 by dotted and solid lines, respectively. The figures show that dec-iSCP finds feasible solutions with higher probability than dec-SCP. Also, dec-iSCP converges to feasible solution sooner than dec-SCP. The error bars are mainly due to variability in randomly generated test cases[4]. A case by case comparison shows that dec-iSCP is on average 14% faster than dec-SCP.

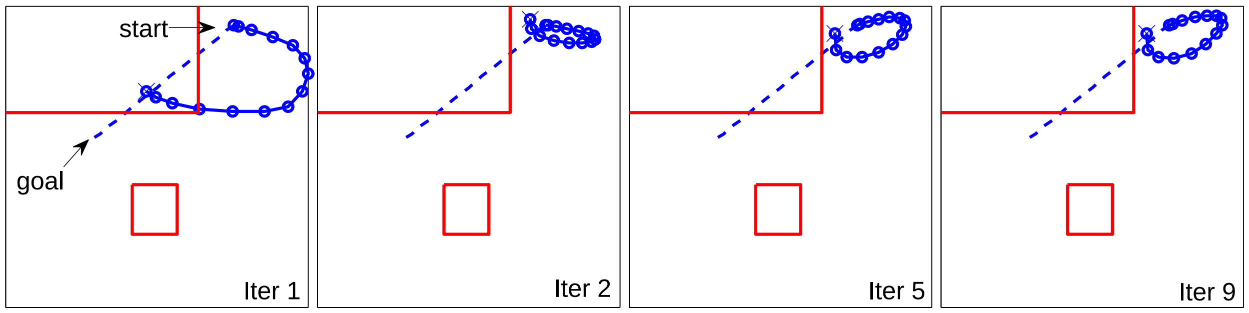
D. *Hardware Experiment*

For hardware demonstration, dec-iSCP is implemented in C++ and utilizes the MOSEK library for solving QPs at each iteration. The dec-iSCP algorithm consistently produced smooth, feasible trajectories for a team of 4 quadrotors navigating in ACL's non-convex flight domain (Fig. 1(b)). A hardware demonstration video can be found at `http://youtu.be/Zkxc4PRGvC4`.

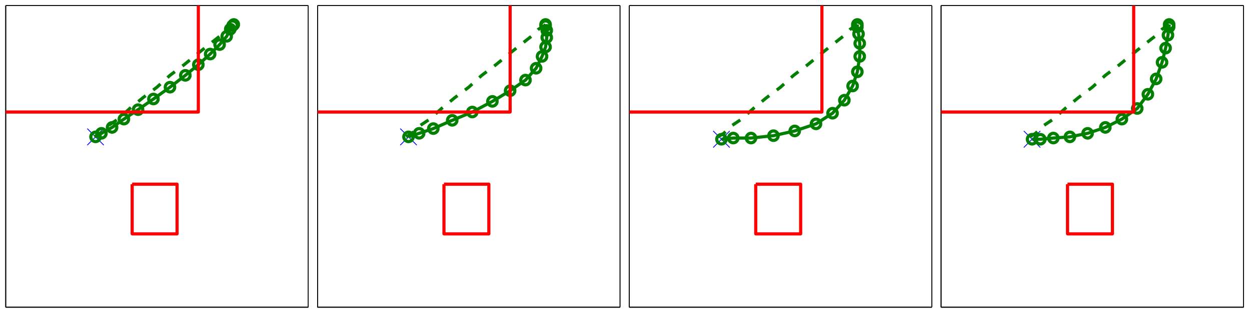
V. Conclusion

This paper has developed the incremental sequential convex programming (iSCP) algorithm for multiagent path planning. We demonstrated that decoupling the multiagent optimization problem yields better computational tractability. We also examined why previous SCP-based methods often fail to find feasible trajectories in non-convex domains, as well as why decoupled implementations of SCP-based

[4]For example, test cases in which vehicles do not cross paths are easier than those where many vehicles cross paths. Easy test cases require many times fewer iterations to converge than difficult test cases.

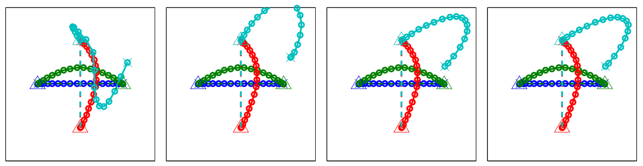


(a) SCP did not find a feasible solution.

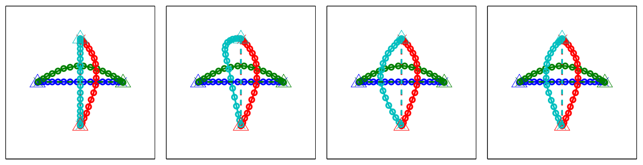


(b) iSCP found a feasible solution.

Fig. 5. Convergence pattern around a corner. A vehicle attempts to negotiate a corner in ACL (see Fig. 1(a) for axis label). Fig. 5(a) and Fig. 5(b) illustrate the vehicle's trajectory as computed by SCP and iSCP, respectively, at iterations 1, 2, 5, and 9 from left to right. SCP fails to find a feasible solution while iSCP has found a feasible solution.



(a) dec-SCP did not find a feasible solution.



(b) dec-iSCP found a feasible solution.

Fig. 7. Convergence patterns of multiple vehicles in a 6m $\times$ 6m convex domain. In the decoupled solution approach, agents are solved sequentially and the trajectory of an earlier agent is used as constraints for later agents. Two pairs of vehicles attempt swap positions, with initial positions marked with triangles. The red, blue and green agents are solved for before the teal agent. Fig. 7(a) and Fig. 7(b) illustrate the teal agent's trajectory found by dec-SCP and dec-iSCP, respectively, at iterations 1, 3, 5, and 7 from left to right. dec-SCP failed to find a feasible solution while dec-iSCP found a feasible solution.

methods often fail to find feasible trajectories even in convex domains. The iSCP algorithm was developed to address these problems, and was shown to successfully find a feasible trajectory in many scenarios where previous algorithms failed. A decoupled implementation of the proposed algorithm, dec-iSCP, finds feasible solutions with higher probability and in less computation time than dec-SCP. The proposed algorithm is real-time implementable and is validated through hardware experiments. In future studies, we will investigate ways for parallel implementation of iSCP to enable decentralized and asynchronous path planning for multiagent systems.

References

[1] L. Merino, F. Caballero, J. R. Martínez-de Dios, I. Maza, and A. Ollero, "An unmanned aircraft system for automatic forest fire monitoring and measurement," *Journal of Intelligent & Robotic Systems*, vol. 65, no. 1-4, pp. 533–548, 2012.

[2] Y. F. Chen, N. K. Ure, G. Chowdhary, J. P. How, and J. Vian, "Planning for large-scale multiagent problems via hierarchical decomposition with applications to uav health management," in *American Control Conference (ACC)*, Portland, OR, June 2014.

[3] E. Guizzo, "Three engineers, hundreds of robots, one warehouse," *Spectrum, IEEE*, vol. 45, no. 7, pp. 26–34, 2008.

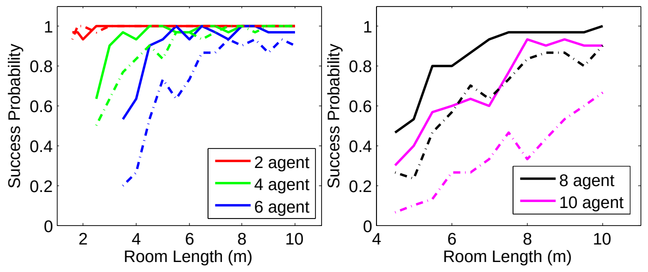
[4] A. Schöllig, F. Augugliaro, and R. DAndrea, "A platform for dance performances with multiple quadrocopters," *Improving Tracking Performance by Learning from Past Data*, p. 147, 2012.

[5] T. Schouwenaars, B. De Moor, E. Feron, and J. How, "Mixed integer programming for multi-vehicle path planning," in *European control conference*, vol. 1. Citeseer, 2001, pp. 2603–2608.

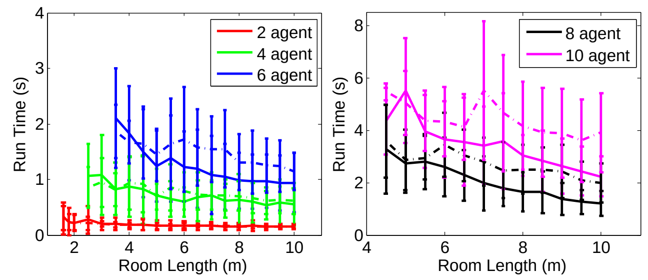
[6] D. Mellinger, A. Kushleyev, and V. Kumar, "Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 477–483.

[7] M. Čáp, P. Novák, J. Vokrínek, and M. Pěchouček, "Multi-agent rrt: sampling-based cooperative pathfinding," in *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems, 2013, pp. 1263–1264.

[8] V. R. Desaraju and J. P. How, "Decentralized path planning for multi-agent teams with complex constraints," *Autonomous Robots*, vol. 32, no. 4, pp. 385–403, 2012.



(a) Probability of finding feasible solutions



(b) Average run time

Fig. 8. Comparing dec-SCP to dec-iSCP. The performance of dec-SCP and dec-iSCP is illustrated in dotted and solid lines, respectively. The x-axis shows the side length of a square region, in which all agents reside. Dec-iSCP has greater probability of finding a feasible trajectory and also converges sooner than dec-SCP.

[9] S. M. LaValle, "Rapidly-exploring random trees a new tool for path planning," Computer Science Department, Iowa State University, Tech. Rep. TR 98-11, 1998. [Online]. Available: http://janowiec.cs.iastate.edu/papers/rrt.ps

[10] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt*," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1478–1483.

[11] A. Stentz, "The focussed dˆ* algorithm for real-time replanning," in *IJCAI*, vol. 95, 1995, pp. 1652–1659.

[12] M. Likhachev, D. I. Ferguson, G. J. Gordon, A. Stentz, and S. Thrun, "Anytime dynamic a*: An anytime, replanning algorithm." in *ICAPS*, 2005, pp. 262–271.

[13] J. Yu and S. M. LaValle, "Multi-agent path planning and network flow," in *Algorithmic Foundations of Robotics X*. Springer, 2013, pp. 157–173.

[14] K.-H. C. Wang and A. Botea, "Tractable multi-agent path planning on grid maps." in *IJCAI*, vol. 9, 2009, pp. 1870–1875.

[15] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *Robotics and Automation, IEEE Transactions on*, vol. 8, no. 1, pp. 23–32, 1992.

[16] C. W. Warren, "Multiple robot path coordination using artificial potential fields," in *Robotics and Automation, 1990. Proceedings., 1990 IEEE International Conference on*. IEEE, 1990, pp. 500–505.

[17] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. Lin, D. Manocha, and P. Dubey, "Clearpath: highly parallel collision avoidance for multi-agent simulation," in *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. ACM, 2009, pp. 177–187.

[18] S. Boyd, "Sequential convex programming," *Lecture Notes, Stanford University*, 2008.

[19] J. Schulman, J. Ho, A. Lee, I. Awwal, H. Bradlow, and P. Abbeel, "Finding locally optimal, collision-free trajectories with sequential convex optimization." in *Robotics: Science and Systems*, vol. 9, no. 1. Citeseer, 2013, pp. 1–10.

[20] F. Augugliaro, A. P. Schoellig, and R. D'Andrea, "Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 1917–1922.

[21] D. Morgan, S.-J. Chung, and F. Y. Hadaegh, "Model predictive control of swarms of spacecraft using sequential convex programming," *Journal of Guidance, Control, and Dynamics*, pp. 1–16, 2014.